Multitask Learning for Bayesian Neural Networks

by

Krunoslav Kovač

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

Abstract

Multitask Learning for Bayesian Neural Networks

Krunoslav Kovač

Master of Science

Graduate Department of Computer Science

University of Toronto

2005

This thesis introduces a new multitask learning model for Bayesian neural networks based on ideas borrowed from statistics: random regression coefficient models. The output of the model is a combination of a common hidden layer and a task specific hidden layer, one for each task. If the tasks are related, the goal is to capture as much structure as possible in the common layer, while the task specific layers reflect the fine differences between the tasks. This can be achieved by giving different priors for different model parameters.

The experiments show that the model is capable of exploiting the relatedness of the tasks to improve its generalisation accuracy. As for other multitask learning models, it is particularly effective when the training data is scarce. The feasibility of applying the introduced multitask learning model to Brain Computer Interface problems is also investigated.

Mojim roditeljima.

Acknowledgements

Thanks go first to my supervisor Radford Neal for introducing me to Bayesian neural networks. His guidance as the research progressed, many questions, interpretations and suggestions made this thesis possible.

I would also like to thank my second reader Geoffrey Hinton for his time and valuable comments on the thesis. Thanks also go to all the other members of the machine learning group at the University of Toronto who have helped me in some way.

Special thanks go to my parents. Although being far away from me, they were a constant source of help and comfort.

At the end, I would also like to thank Dane Jurčić and his family for helping me out in the first days in an unknown country, as well as later on.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The traditional approach in machine learning is to train a separate learner for each problem. There are a number of learning methods that have proven to be very successful in such a setup. However, the success of these methods largely depends on the abundance of training data, which is in contrast with the way that humans learn. Studies have shown that we are able to learn a concept with only a single datum [2, 34], which certainly implies that people must learn using something more than just the training data. It seems that the answer lies in a massive reuse of previously learned knowledge from related tasks. As a consequence, recently, there has been a growing interest in machine learning methods that can exploit knowledge from such other tasks to improve accuracy either generally or in situations when the training data is scarce. There are quite a few ways to transfer knowledge from task to task. The multitask learning approach trains multiple tasks in parallel. Chapter 2 reviews the techniques of knowledge transfer with special attention given to multitask learning.

A Brain-Computer Interface (BCI) is a communication system that transforms human brain activity into signals that can control mechanical devices. The idea of such a system isn't new. In 1967 Edmond Dewan, who worked at the Data Sciences Laboratory of the Air Force Cambridge Research Laboratories in Bedford, managed to train subjects to control the alpha rhythms of their brain waves and then used them to send Morse code of the word ‘cybernetics’ to a computer (mentioned in ACM SIGART Newsletter, No. 19, December, 1969). A few years later Jacques Vidal published a paper “Toward Direct Brain-Computer Communication” [89] in which he described the first BCI project. The research was part of a large scale project in biocybernetics and human-computer interaction and sponsored by the US government, who had a keen interest in the matter during the seventies. Today, dozens of research groups focus their efforts on BCI.

Human brain activity for the purposes of BCI systems is usually collected in the form of EEG recordings due to the low cost of the equipment involved and the noninvasive nature of recording. However, other electrophysiological measures like Functional Magnetic Resonance Imaging (fMRI), Magnetoencephalographic (MEG) measurements, and electrocorticograms (ECoG) are being studied as well. Features are extracted from the measurements, and then processed (usually by computer software) and transformed into a form suitable to operate a device. As can be seen, BCI research is a multidisciplinary field involving neuroscience, psychology, machine learning, computer science in general, mathematics, clinical rehabilitation and so on.

The main motivation for the work on BCI is to provide a means of communication and control functions (e.g. basic web browsing and word processing, neuroprosthesis control etc.) to so-called “locked-in” patients. Locked-in patients can make no voluntary movements other than those with their eyes, but are fully conscious and aware of the environment, and their intellectual capabilities are often intact. Although current BCIs cannot achieve information transfer rates of more than about 20-30 bits/min, even this limited throughput can allow rudimentary interfaces that can improve the quality of life for such people. BCI is the subject of chapter 3, with an emphasis on its machine learning aspect.

Chapter 4 introduces a novel approach to multitask learning. It is inspired by random regression coefficients models and builds upon Bayesian neural networks. For each datum, the output of the network is a linear combination of a task specific hidden layer associated with the given datum and a special common hidden layer. We want the common part to have the biggest impact on the final output, while the task specific hidden layers are to account for the variation between the tasks, which should be small, if the tasks are sufficiently related. This corresponds to the weights in the common part being significantly bigger than the weights associated with the task specific layers. This can be achieved by specifying different priors for hyperparameters of different groups of model parameters. Besides the description of the model itself, a short introduction to Bayesian neural networks and to Markov chain Monte Carlo methods of sampling the posterior distribution of the network parameters is also given in this chapter.

The new model is tested on artificial data in chapter 5. We show that the model is capable of exploiting task relatedness and gain insight into the behaviour of the sampled neural networks.

Chapter 6 investigates the application of multitask learning to BCI problems. The results and contributions of the thesis are summarised in the final chapter.

Chapter 2

Knowledge transfer in machine learning

2.1 Human learning

Most research in machine learning so far has used a "tabula rasa" approach: the goal was to improve methods that did the learning on separate, individual tasks, and the learning was started from scratch, with no previous knowledge reused from other learning sessions. For some problems, such algorithms produced very good results. However, if we compare the learning abilities of humans and animals with that of today's machines, we find dissimilarities. Humans are able to learn from a remarkably small number of training examples, despite the immense complexity of the real world where the learning occurs. Current machine learning approaches don't scale very well to such complex domains due to the huge number of training examples that would be required to have some success.

A number of psychological studies have touched this aspect of human learning. It has been suggested and empirically proved that a person's prior knowledge affects the rate or the accuracy of learning [49, 50, 65]. Pazzani in [56] further explored the influence of prior causal knowledge on the number of trials needed to learn a concept. He gave experimental evidence that the prior knowledge of the learner may be as influential as the informational structure of the environment in concept learning. Trying to explain his results, he hypothesizes that the ability to generalise quickly comes from the ability to apply relevant background knowledge to the learning task and the ability to fall back on weaker methods in the absence of background knowledge. There are other ways of knowledge transfer besides directly applying experience in some form. People often use analogies and metaphors (see e.g. [28]) to reuse previous knowledge even in the domains they are not familiar with. But although it is generally accepted that humans benefit by transferring knowledge they have already acquired, and use it to learn a new task better or faster, there are examples when transferring knowledge from other domains can have negative effects. Luchins and Luchins ([40]) performed experiments with water jug problems which showed that prior experience can sometimes hurt performance on learning related tasks.

2.2 Machine approaches to knowledge transfer

In computer science, the research field of "knowledge transfer" is trying to diminish these differences between human and machine learning abilities. Some authors use different terms for essentially the same concept, like "inductive bias shifts" ([32, 68, 86]). Several frameworks and approaches to knowledge transfer have been developed like *"lifelong learning", "metalearning", "learning to learn", "bias learning"*. All of them focus on the idea of transferring knowledge in some form in the learning process.

One of the first approaches that employs transfer of knowledge is the VBMS (Variable Bias Management System) [66] which contains a set of conventional learning methods and chooses the most appropriate one based on previous, related tasks. In [48] an approach is described where previous training data is used to tune the learning parameters in a new experiment (e.g. the learning rate in neural networks).

Knowledge based approaches that hand-code prior knowledge into the learning process have also been investigated (see e.g. [57, 91]). Neural network methods of this kind ([26, 43, 85]) use domain knowledge acquired in previous learning sessions to initialise neural network weights and/or other network parameters, and then train the network using the traditional training algorithms for the current problem.

A representative example of such an approach is the "Knowledge-Based Artificial Neural Networks" (KBANN) described in [85], a hybrid learning system built on top of conventional neural network techniques. KBANN learns in two independent phases. In the first phase, knowledge about the problem domain is stated in the form of acyclic propositional Horn clauses, and these rules are then mappped to an artificial neural network. The idea is to create a hierarchical rule structure, where some rules/clauses would operate directly on inputs, and higher level clauses would operate on both inputs and lower level clauses. The topmost clause is the quantity we are learning; the other clauses are referred to as intermediate conclusions. These intermediate conclusions would be mapped to the hidden units (or even layers), and the topmost clause to the output. The links in the neural network are added in a way that directly corresponds to the rule hierarchy graph. In the second phase, the weights are initialised to small random values and a backpropagation learning algorithm is used to refine the network. Their tests showed that these nets generalise better than plain-vanilla nets when such rules about the problem domain can be given. Finally, a third phase might be added to the KBANN system. In this phase, the resulting trained network could be used to extract new rules about this particular problem and/or domain. Despite some attempts ([69, 77, 84]), not much progress has been made in this direction.

The chief problem of KBANN, as well as other knowledge based approaches, is that an initial domain theory must be available, which usually has to be provided by a human expert, who is often not available in practice. KBANN in particular has two additional, and rather severe, limitations. First, all inputs must be binary values, and second, our knowledge about the domain must be expressed with propositional clauses.

It may be worthwhile to mention that early studies ([44, 79]) showed that literal transfer of neural network weights from previous learning sessions, i.e. copying them without modification and using them as the initial state for the target task, was usually more harmful than using random values. Even some refinements of this idea ([23, 62, 90]) produced very little or no performance boost at all.

2.3 Lifelong learning

Sebastian Thrun uses the framework of "lifelong learning" for the study of the transfer of knowledge [82]. In his framework a learner is faced with a stream of learning problems over its entire lifetime, which provides the opportunity for the transfer of knowledge. For each new problem we can use the experience gathered while learning previous problems to improve learning performance. He presented several algorithms which extend traditional approaches to lifelong learning ones, demonstrating that when learning in a lifelong context, one can learn more complex tasks with less training data, independent of the particular learning approach. Most of these algorithms are based on re-representing data and can be modified to work with a variety of learning methods. We will consider three of them, two being memory based, and one neural network based.

Memory based learning algorithms typically don't have a learning session before we use them with test data. Instead, they remember all the training data available and explicitly use them to generate the output for the test data. One of the best known algorithms of this kind is K-nearest neighbour ([25, 19]). If x is an input for which we want to know the output y, K-nearest neighbour (KNN) locates K training examples $\langle x_i, y_i \rangle$ whose input patterns x_i are nearest to x using some distance metric (most commonly the Euclidean distance if the inputs are in $\mathbf{R}^m$). KNN then uses the mean value of the y_i's as the output value y. Other statistical quantities like mode, median etc. can also be used.

A similar method was introduced by Shepard in [72]. Instead of looking at some number of the nearest inputs, we use them all, but we weight each training example according to the inverse distance to the query point x:

$$out(x) = \left(\sum_{\langle x_i, y_i \rangle \in X^n} \frac{y_i}{|x - x_i| + \epsilon} \right) \cdot \left(\sum_{\langle x_i, y_i \rangle \in X^n} \frac{1}{|x - x_i| + \epsilon} \right)^{-1} \tag{2.1}$$

X^n here denotes the set of training examples for task n, and ϵ is a small positive constant to prevent numerical overflows.

We can see that both of these methods act directly on the training set X^n. Hence, it is not obvious how to incorporate other training sets, since their examples have the wrong class labels. Thrun proposes two ideas. The first one involves learning the representation of the data. This can be done by transforming the data by a function $g : I \to I'$, which maps inputs from I to a new space I'. The memory based algorithms then work on this new input space I'. Function g is chosen in a way in which multiple examples of the same class are mapped to similar representations, whereas examples of different classes should get mapped further from each other in the new space. This property can be directly expressed as an "energy function" for g, which is determined by the training data for the $n-1$ previous problems:

$$E = \sum_{k=1}^{n-1} \sum_{\langle x,y \rangle \in X^k} \left(\sum_{\langle \hat{x},\hat{y} \rangle \in X^k, \hat{y}=y} |g(x) - g(\hat{x})| \quad - \sum_{\langle \hat{x},\hat{y} \rangle \in X^k, \hat{y}\neq y} |g(x) - g(\hat{x})| \right) \tag{2.2}$$

Minimising the energy function E will force the distance between pairs of examples of the same concept to be small (the first term inside brackets), and the distance between examples from different concepts to be large (the second term inside the brackets). To obtain g, Thrun used an artificial neural network (ANN) trained using the backpropagation algorithm ([67]) with the energy function given above as the error function. The memory based algorithm for the n^{th} problem will work with the re-represented training data set $\{\langle g(x), y \rangle\}$ given $X^n = \{\langle x, y \rangle\}$. In a parallel approach we would use the data for all n tasks to obtain g.

The second way of exploiting other problem's training data sets with memory based learning algorithms is to learn the distance function. To this end, Thrun introduces a comparator function $d : I \times I \rightarrow [0, 1]$ ([81]). It should be noted that d is not a distance metric. Its usage was shown on the concept learning problem, which is nothing but a noise-free pattern classification task with only two classes, 0 and 1. A pattern is a member of the concept if its class label is 1. Training examples for d are pairs of examples $\langle x, y\rangle$ and $\langle \hat{x}, \hat{y}\rangle$, both elements of the same X^k for all $k = 1, \ldots, n-1$ for which d is defined as:

$$d(x, \hat{x}) = \begin{cases} 1 & \text{if } y = \hat{y} = 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

The value of d is 1 for inputs that both belong to the concept, and 0 otherwise. Therefore, once d is trained, the output of $d(x, \hat{x})$ can be interpreted as the conditional probability that x belongs to the concept f^n given that $\langle \hat{x}, \hat{y}\rangle$ belongs to the concept. Since the training examples for d don't carry any concept specific information, all the training sets from all the tasks can be used to train d. We can combine the votes of multiple examples in X^n, assuming they are (conditionally) independent, using Bayes' Rule, which gives:

$$Bel(f^n(x) = 1) \;=\; 1 - \frac{1}{1 + \prod_{\langle \hat{x}, \hat{y}=1\rangle \in X^n} \frac{d(x, \hat{x})}{1 - d(x, \hat{x})}} \tag{2.4}$$

$Bel(f^n(x) = 1)$ represents the belief that x belongs to the concept f^n.

Again, d can be realized by an artificial neural network.

These two ideas for extending memory based learning algorithms to fit into the lifelong learning framework can be applied to neural network approaches as well. The idea of re-representing the learning data has been carried out in a variety of ways. They are all based on the fact that the networks to re-represent the training data and to learn on it afterward, can be realised with a single neural network. Some authors do the training sequentially ([63, 70]), while others do it in parallel ([8, 16, 75]). One of these approaches is multitask learning ([17]) which is discussed in more detail in 2.5. The advantage of sequential learning is that we don't need all the training data for all the tasks available all the time. But sequential training with backpropagation can suffer from catastrophic interference or "forgetting": the inability of an ANN to learn a second set of information without forgetting what it previously learned ([71]).

The approach with the comparator function was used for explanation-based neural network (EBNN) learning in [80]. It is based on the Tangent-Prop algorithm developed in [74], which uses training examples of the form $\langle x, f(x), \nabla_x f(x) \rangle$, that is, in addition to the target values, the slopes are also being estimated. The problem is to get the slope information, and this is where the comparator function d comes into the game. For a fixed positive example $\langle \hat{x}, \hat{y} = 1 \rangle$ and $t \in I$, $d(t, \hat{x})$ measures the probability that t belongs to the concept, according to d and given the training example $\langle \hat{x}, \hat{y} \rangle$. Hence, if $\langle \hat{x}, \hat{y} \rangle \in X^n$ is a positive example of a concept f^n (i.e. $\hat{y} = 1$), then the function $d_{\hat{x}} : I \to [0, 1]$ defined as:

$$d_{\hat{x}}(t) = d(t, \hat{x}) \tag{2.5}$$

can be thought of as an approximation of the target function f^n at t. Since EBNN estimates d by a neural network, d is differentiable. $d_{\hat{x}}$ is then also differentiable and the gradient

$$\frac{\partial d_{\hat{x}}(t)}{\partial t}$$

is used as an estimate of the slope of f^n at t. $\nabla_x f^n(x)$ is defined by substituting $t = x$. The network is then trained using the Tangent-Prop algorithm.

If there is more than one positive example of a concept, the slope is defined as the average of all of them. The comparator function d is learned from all the previous tasks and can therefore transfer knowledge to the new task.

All of the methods in this section, according to Thrun, generalise noticeably better than their traditional counterparts when the data are scarce, and when the tasks are in fact related. As the number of training examples increases, conventional techniques rapidly approach the methods involving knowledge transfer in performance.

It is crucial that the tasks be related, as blindly reusing knowledge from tasks that are irrelevant to the target task will very likely hurt the overall performance. All the approaches described so far give equal importance to all the tasks. The Task Clustering (TC) algorithm ([83]) incorporates an additional step in the learning process in which tasks are grouped into clusters. When learning a new task, the most appropriate cluster is determined, and only the tasks from that cluster are used to bias the learning of the new task. This kind of selective knowledge transfer is particularly useful when there are a number of tasks from some domain, but only a small subset of them may be related to a specific target task.

2.4 Theoretical work on knowledge transfer and task relatedness

Perhaps the most significant work on theoretical treatment of knowledge transfer, especially in multitask learning, was done by J. Baxter. His work is based on the fact that in order to generalise well, a learner must be biased in some way. That might be done by a human expert, but this has many limitations. He introduces two formal models which are capable of automatic bias learning. The underlying assumption for both of them is that the learner is embedded within an environment of related problems, and the goal is to learn all of them, and not just a single one. The first is the Empirical Process (EP) model ([11]), which is built upon the PAC model of machine learning ([87, 88]). The second model is based on Bayesian inference and information theory ([10]).

The EP approach consists of:

- an input space X and an output space Y,
- a loss function $l : Y \times Y \rightarrow \mathrm{R}$,
- an environment $(\mathcal{P}, Q)$ where $\mathcal{P}$ is the set of all probability distributions on $X \times Y$

and Q is a distribution on $\mathcal{P}$

- a hypothesis space family $\mathrm{H} = \{\mathcal{H}\}$ where each $\mathcal{H}$ is a set of functions $h : X \to Y$.

$\mathcal{P}$ is interpreted as the set of all possible learning problems, while Q tells us which learning problems are more likely to be seen. The goal is to find a hypothesis space minimizing the loss er_Q over all learning problems, which depends on Q. Since we don't generally know Q, the loss is being estimated from the training examples z to get er_z. z contains an equal number of examples from some number of learning problems. Baxter in [11] gives some bounds on the number of tasks and the number of samples from each task needed to get er_z to be within at most ϵ from er_Q, with a probability of $1 - \delta$ for arbitrarily small and positive ϵ and δ.

Baxter's bounds are formulated using the average error on all the tasks. It was shown in [12] that this can be strengthened to obtain similar bounds for a particular task. The same work also introduces a formal definition of task relatedness.

The Bayes bias learning model consists of:

- an input space X and an output space Y,
- a set of probability distributions P_θ on $X \times Y$, parameterised by $\theta \in \Theta$,
- a set of prior distributions P_π on Θ, parameterised by $\pi \in \Pi$,
- a prior distribution P_{π^*} where $\pi^* \in \Pi$.
- a learner also has a subjective hyper-prior distribution P_Π on Π.

P_θ can be viewed as a subset of $\mathcal{P}$ from the EP model with Q as the underlying distribution. The goal is to find the right prior distribution Q which is chosen from a set of possible distributions $\{P_\pi : \pi \in \Pi\}$. The idea is that we start with a hyper-prior distribution P_Π, then based on the training examples that are obtained by using the environmental prior distribution P_{π^*}, we update the hyper-prior to a hyper-posterior.

This hyper-posterior is then used as the prior distribution when learning new tasks. This model, having two levels, is an example of a hierarchical Bayesian model. Baxter in [10] gave bounds similar to those for the EP model.

2.5 Multitask learning

Multitask learning (MTL) is an approach to knowledge transfer whose main characteristic is that it learns multiple tasks in *parallel* and uses a *shared representation*. MTL is not a single learning algorithm. The MTL backpropagation implementation is perhaps the most widely known, but there are other examples, some of which are discussed later in this section. All of them are based on some conventional learning algorithm.

We are usually focused on a single task, which is called the main task. All the other tasks, called the extra tasks, have the sole purpose of improving the performance on the main task, and we do not care about the performance on them.

The easiest way to explain MTL with backpropagation networks is to use a simple example. Let's assume that we have four tasks from the same domain, i.e. they all have the same set of inputs. The single task learning (STL) approach, illustrated on figure 2.1, is to have four separate neural networks, each with one output (the case where tasks have more than a single output doesn't bring anything conceptually new into the picture). The learning is done by training each network individually on the training cases for its task only. Since there is nothing that connects the networks, nor the process of learning, what one network learns has no effect on other networks.

The MTL approach with backpropagation networks is shown on figure 2.2. It is a single network with the same inputs as the STL ones, and with four output units, one for each task. Caruana in his thesis ([18]) restricts himself to domains where all tasks are defined on a common set of input features. An example is the 1D-DOORS problem. The task was to locate doorknobs and recognise door type (single or double) in images of

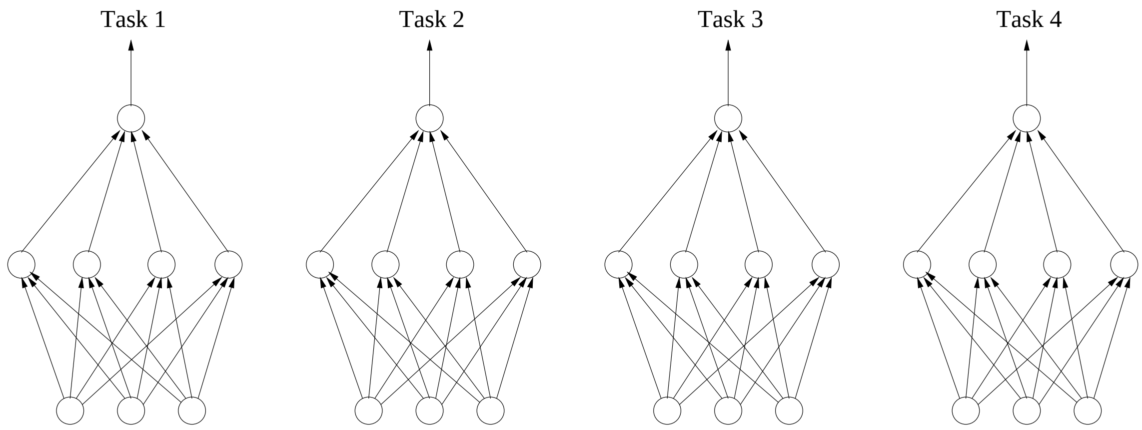


Figure 2.1: Single Task Learning approach

doors. Besides these two tasks, additional tasks were added: location of the door center, width of doorway, locations of left door jamb, location of right door jamb, width of left door jamb, width of right door jamb, location of left edge of door, location of right edge of door. For this problem, each training case has outputs defined for all tasks. In contrast, the method introduced in chapter 4 uses separate training data sets for each task.

Some particular tasks may in essence use only a subset of input features. However, if the tasks are related, there should be a significant intersection of the sets of input features for different tasks. MTL also works better when this is true.

In the architecture in figure 2.2 all the tasks share the input to hidden weights and the hidden layer units, and this is the shared representation that the MTL methodology requires, and which allows knowledge transfer between tasks.

The general conclusion is that MTL, as well as all the other knowledge transfer approaches, work better when the tasks are related. However, task relatedness is a rather vague concept. For instance, Caruana illustrates with a simple example that related tasks, although correlated at the representation level, are not necessarily correlated at the output level. Caruana lists some task relationships that the MTL approach to back-propagation networks can exploit:

- Data amplification: an effective increase in sample size through the extra informa-

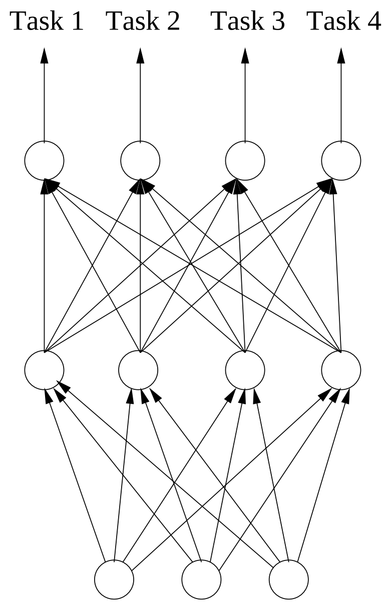


Figure 2.2: Multitask Learning approach

tion in the training data of related tasks. Effective means the extra information from the extra training examples, not the extra training examples themselves.

- Eavesdropping: a situation where some feature is useful for two tasks, and one of them can learn it easily, while the other uses it in a more complex fashion. The other task *eavesdrops* on the internal representation from the first task to learn that feature better. The simplest case when the first task is the feature itself was used by Abu-Mostafa in [1], who called such helper features "catalytic hints".

- Attribute selection: when a subfeature shared by several tasks uses only a fraction of inputs to the net. With a small amount of data and/or significant noise, MTL nets will be able to better select the attributes (inputs) relevant to the common subfeature.

- Representation bias. It is believed that backpropagation (BP) networks work better if they can better represent the domain's regularities. A particular regularity is usually learned differently even by one task in two different training sessions, due to the stochastic nature of BP search (the weights are initialised to random values). If two tasks are trained on one network, search will be biased toward the intersection of the representations each of the tasks would learn on its own. The conjecture is that representations of the regularity near this intersection better capture the true regularity, since they satisfy more than one task from the domain.

- Overfitting prevention. Caruana describes two situations where a task T' can help some other task T not to overfit some feature F, assuming that they both use it. In the first situation T is about to begin overfitting, while T' is still not at that point. T' will then provide a gradient component that will steer T from overfitting F. In the second situation, both T and T' depend on F, but on slightly different ways. The conjecture is that there are fewer changes that will lower error on both tasks and overfit F for one of the tasks at the same time.

Caruana empirically demonstrated that MTL with backprop networks is indeed capable of taking advantage of these types of task relationships.

The MTL approach also exhibits a somewhat peculiar behaviour: some inputs work better when used as extra outputs. This also allows MTL nets to be used in wider number of situations, since we don't necessarily need 'true' multiple tasks. This idea has been shown to work on real-world problems such as the DNA splice-junction problem. Unfortunately, deciding which features are better used as inputs and which as extra outputs can be difficult.

Caruana also showed that the MTL approach can be combined with other learning approaches as well, even if the choice of the shared representation is not immediately obvious. Two such examples are K-nearest neighbour and locally weighted averaging (also known as kernel regression). For these methods, the exact distance metric used is important for the optimal performance. There are ways to learn the distance metric, and this distance metric is what is shared between the tasks. Caruana also gave the sketch of an algorithm for multitask learning in decision trees. MTL can also be applied to the same algorithm in more than one way.

A new alternative algorithm for MTL with backprop networks is introduced in chapter 4.

2.6 Some refinements of MTL with backpropagation networks

Ghosn and Bengio in [29] conducted a number of experiments with backpropagation networks where different sets of shared parameters were used. The experiments included sharing only input to hidden weights, sharing only hidden to output weights, sharing all weights, and sharing no weights. They applied the methods on a stock selection problem, with stocks from 35 companies treated as different tasks. The inputs were five explanatory variables: two macro-economic variables known to influence business cycles, and three variables obtained from the previous price changes of the stock. The output was the expected future return. On that particular problem they reported that sharing all parameters displayed a marked drop in performance compared to sharing some or sometimes even no parameters at all. Although the difference among the other three models wasn't very significant, the best and the most consistent results were obtained while sharing only input to hidden weights.

In the ordinary MTL networks all tasks learn at the same learn learning rate. The ηMTL method introduced in [73] uses a separate learning rate η_k for each task. The primary task uses the base learning rate η, while all the other tasks define their learning rates η_k as a product of η and a measurement of relatedness R_k between the task k and the primary task. The range of R_k is $[0, 1]$. If all R_k's are 0, we have an STL network, and if $R_k = 1$ for all k, we have a standard MTL network. Hence, ηMTL can be thought of as a generalisation of STL and MTL.

R_k is defined as:

$$R_k = \tanh\left(\frac{a_k}{d_k^2 + \psi} \times \frac{1}{RELMIN}\right) \tag{2.6}$$

The $tanh$ function is here simply for the purpose of converting the expression to the $[0, 1]$ range. d_k is the distance in weight space between the k^{th} and the primary task. The belief is that the weights for similar tasks tend to stay close to each other. a_k measures the accuracy of the k^{th} task hypothesis, and is defined simply as inverse of the sum of squared errors over the training set for the task k. $RELMIN$ controls the decay of $tanh$ from 1. Its value was experimentally set based on the size of the training set, number of hidden nodes etc. ψ is a small positive constant to prevent division by zero and/or numerical overflows.

One additional possibility for the use of ηMTL is in problems involving multiple outputs considered as a single task. It may be beneficial to allow different learning rates for different outputs.

Section 2.4 presented Baxter's Bayes bias learning model [9]. A functional implementation of that model was the goal of Tom Heskes in his Empirical Bayes model [33]. Bakker and Heskes in [7] further augmented that model to include clustering and gating of multiple tasks. Suppose that $D_i = \{x_i^\mu, y_i^\mu\}$ is the data set for task i, $\mu = 1, \ldots, n_i$, the number of training examples for task i. The assumption of their model is that the response y_i^μ (for simplicity taken to be one dimensional) is the output of a multi-layered perceptron with additional Gaussian noise of standard deviation σ. Putting the input to hidden weights into an $n_{\text{hidden}} \times (n_{\text{input}} + 1)$ matrix W (bias included) and hidden to output weights into an $N \times (n_{\text{hidden}} + 1)$ matrix A, where N is the number of tasks, we have:

$$y_i^\mu = \sum_{j=1}^{n_{\text{hidden}}} A_{ij} h_{ij}^\mu + A_{i0} + \text{noise}, \quad h_{ij}^\mu = g\left(\sum_{k=1}^{n_{\text{input}}} W_{jk} x_{ik}^\mu + W_{j0}\right) \tag{2.7}$$

The complete data set will be denoted as $D = \{D_i\}$, with $i = 1, \ldots, N$, the number of tasks. The hidden to output weights in A are specific for each task; all the other parameters are shared. Assuming the tasks are IID given the hyperparameters, a prior distribution is defined for the task specific parameters as:

$$A_i \sim \mathcal{N}(m, \Sigma) \tag{2.8}$$

with hyperparameters $\Lambda = \{W, m, \Sigma, \sigma\}$.

The joint distribution of data and model parameters is then:

$$P(D, A|\Lambda) = \prod_{i=1}^{N} P(D_i|A_i, W, \sigma) P(A_i|m, \Sigma) \tag{2.9}$$

Integrating over A, Bakker and Heskes obtain:

$$P(D|\Lambda) = \prod_{i=1}^{N} P(D_i|\Lambda) \tag{2.10}$$

where

$$P(D_i|\Lambda) \propto \left(|\Sigma|\sigma^{2n_i}|Q_i|\right)^{-\frac{1}{2}} \exp\left[\frac{1}{2}(\mathrm{R}_i^T Q_i^{-1} \mathrm{R}_i - S_i)\right] \tag{2.11}$$

Q_i, R_i and S_i are functions of D_i and Λ defined as follows:

$$Q_i = \sigma^{-2} \sum_{\mu=1}^{n_i} h_i^\mu h_i^{\mu T} + \Sigma^{-1}, \qquad \mathrm{R}_i = \sigma^{-2} \sum_{\mu=1}^{n_i} y_i^\mu h_i^\mu + \Sigma^{-1} m, \qquad S_i = \sigma^{-2} \sum_{\mu=1}^{n_i} y_i^{\mu 2} + m^T \Sigma^{-1} m \tag{2.12}$$

The likelihood (2.10) is maximised to obtain the optimal parameters Λ^*. After that $P(A_i|D_i, \Lambda^*)$ can be easily computed.

In the above model the prior (2.8) is the same for all tasks, which is sensible if all tasks are "equally similar". By introducing "features" f_i of the task that will vary among different tasks, but not within a single task, we can have $m_i = M f_i$ with an $(n_{\text{hidden}} + 1) \times n_{\text{feature}}$ matrix M, and replace m in (2.8) with m_i to get task dependent prior means.

We could also assume that there are several clusters of similar tasks, instead of a single cluster. We could then take the prior for A_i to be:

$$A_i \sim \sum_{\alpha=1}^{n_{\text{cluster}}} q_\alpha \mathcal{N}(m_\alpha, \Sigma_\alpha) \tag{2.13}$$

where q_α represents the *a priori* probability for any task to be assigned to cluster α.

Finally, we could combine the above two ideas: let's give each task its own *a priori* probability to be assigned to some cluster α:

$$q_{i\alpha} = \frac{e^{U_\alpha^T f_i}}{\sum_{\alpha'} e^{U_{\alpha'}^T f_i}} \tag{2.14}$$

with U_α an n_{feature}-dimensional vector. f_i's are the features for the task dependent means above.

To optimise the likelihood of Λ, the EM algorithm was used for both this and the plain clustering model. The M-step for U_α is more complicated and was solved using an iterative reweighted least-squares (IRLS) algorithm.

The task gating model is similar to the mixture of experts model ([35]), but instead of using individual inputs to gate tasks, Bakker and Heskes use a separate set of higher level features f_i.

2.7 Random Coefficient Models

As was concluded earlier, the MTL approach offers an advantage over the STL approach when we have data for several tasks that are related. Data sets with observations that can be grouped into such clusters have been analysed by statisticians much prior to the work presented in previous sections. They are routinely handled by various variance component methods. The most common ones are analysis of variance (ANOVA) and analysis of covariance (ANCOVA).

At the heart of the variance component methods is the fact that variance can be partitioned into within-cluster variation and between-cluster variation. Let's assume that some city has three hospitals and we have collected data on blood pressures for heart-failure patients for all three of them. The ANOVA model for this situation is:

$$y_{ij} = M + \mu_j + \epsilon_{ij} \tag{2.15}$$

where y_{ij} is the measured blood pressure value for the i^{th} patient in the j^{th} hospital,

M is the overall mean, μ_j is the variation due to the influence of the j^{th} hospital, and ϵ_{ij} is the random variation of a particular patient. In the fixed-effects ANOVA model, μ_j are unknown constants. This is a natural interpretation if the number of clusters is fixed and we are not interested in other clusters (hospitals). But suppose that we are interested in all hospitals in some country. It is then natural to think of the selected hospitals as a random sample of possible hospitals and interpret μ_j as the cluster-level effect drawn from some population of effects. In the random-effects ANOVA (RANOVA) μ_j are not fixed, but have a normal distribution $\mathcal{N}(0, T^2)$ just as ϵ_{ij}. Which interpretation is more suitable depends on the context. A Bayesian ANOVA model will not really have a distinction, since we would have a prior in any case.

The ANCOVA/RANCOVA methods extend the above model by incorporating the covariates that may influence the observations. For instance, in the above example covariates might be age, gender, body mass etc. The model is:

$$y_{ij} = M + X_{ij}\beta + \mu_j + \epsilon_{ij} \tag{2.16}$$

where y_j is the i^{th} outcome for the j^{th} hospital, M is the overall mean, X_{ij} is the vector of the explanatory variables for the i^{th} patient from the j^{th} hospital, β is the vector of regression coefficients, μ_j is the effect of hospital j variation and ϵ_{ij} is again the variation of a particular patient. $\{\epsilon_{ij}\}$ are mutually independent samples from $\mathcal{N}(0, \sigma^2)$. As in ANOVA/RANOVA, the only difference between ANCOVA and RANCOVA is in the interpretation of the cluster effects $\{\mu_j\}$. The former treats them as a set of fixed, unknown parameters, while the latter treats them as a random sample from $\mathcal{N}(0, \tau^2)$.

The ANOVA/ANCOVA models are not without their problems; the maximum likelihood estimates of variance components they give are generally biased. If this is not desirable, one can use the Restricted Maximum Likelihood (REML) model. The method is due to Patterson and Thompson ([55]), and the REML estimates of variances and covariances are known to be unbiased.

Longford in [39] introduces random coefficient models, which extend the above model to allow for cluster to cluster variation in β as well.

The random coefficient model is given by

$$y_{ij} = M + X_{ij}\beta_j + \mu_j + \epsilon_{ij} \tag{2.17}$$

The factors are the same as in the random-effects ANCOVA, but now $\{\beta j\}$ is assumed to be a random sample from a multivariate distribution:

$$\beta_j \sim \mathcal{N}(\beta, \Sigma^*) \tag{2.18}$$

independent of $\{\epsilon_{ij}\}$ and μ_j. The model can be rewritten as

$$y_{ij} = (M + \mu_j) + X_{ij}(\beta + \gamma_j) + \epsilon_{ij} \tag{2.19}$$

where $\gamma_j = \beta_j - \beta$ is the vector of deviation of the regression coefficients from their expectation, so that $\gamma_j \sim \mathcal{N}(0, \Sigma^*)$. This form of the model can be seen as a simple, linear form of the model introduced in chapter 4.

Chapter 3

Brain Computer Interfaces

BCI research has gained a considerable momentum over the last decade. Cheap and powerful computers, low cost equipment for neurophysiological recording, and a deeper understanding of brain function are the chief reasons that BCI devices have become feasible. The research aims to benefit people with severe muscular disabilities, who may use BCIs for basic communication capabilities or even to control a neuroprosthesis ([78]).

3.1 BCI data sources

The dominant neural activity measurements used in current BCI approaches are EEG recordings. They offer very good temporal resolution, the cost of equipment is as low as a few thousand US dollars, and, finally, gathering EEG data is noninvasive.

Several other neurophysiological measurements are being used in BCI as well. Electrocorticogram (ECoG) is not really much different from EEG; it also measures the electrical activity of the brain, but the electrodes are in a direct contact with the brain cortex surface. The advantage of ECoG over EEG is a lower level of background noise, and a better spatial resolution, since in scalp-recorded EEG the signals from close sources mix up before they reach the scalp surface. Due to its invasive nature, it's much less practical for the intended use of a BCI. Some authors use the term direct brain interface

(DBI) instead of BCI when using ECoG (e.g. [30]). Some studies use even more invasive procedures involving the electrical activities of individual cortical neurons, for example [37]. This study used human subjects suffering from ALS (Amyotrophic Lateral Sclerosis). As an illustrative example, one of the patients was able to spell words at a rate of 3 letters per minute.

BCI based on functional magnetic resonance imaging (fMRI) has also been studied ([27, 31, 47, 92]). fMRI measures the blood oxygen level-dependent (BOLD) signal. It has a spatial resolution of about 1mm, which is much better than EEG (about 30mm). Although fMRI devices can take images every 100-200 ms, the intrinsic inertia of changes in blood vessels limits its time resolution to about one second. The cost is also considerably higher than EEG. During image generation, a patient is exposed to a high static magnetic field and a small alternating radio-frequency field, and although, to date, no health hazards are known, it is considered weakly invasive. However, unlike PET (Positron Emission Tomography), it doesn't use radioactive isotopes or any other chemicals in the patient's body. The feasibility of a BCI based on fMRI has been established even in the case of a single fMRI image (e.g. [47])

MEG (Magnetoencephalography) is a relatively new technology that measures the very faint magnetic fields that emanate from the head as a result of brain activity. It has a temporal resolution of 1 ms and a spatial resolution of 2 mm; so it combines the best of EEG and fMRI. Unfortunately, this literally comes with a price, as current MEG devices cost millions of dollars. Another shortcoming is that a typical current MEG device weights about 8 tons. So, despite being a potentially outstanding source of BCI data, the current BCI approaches seldom use it.

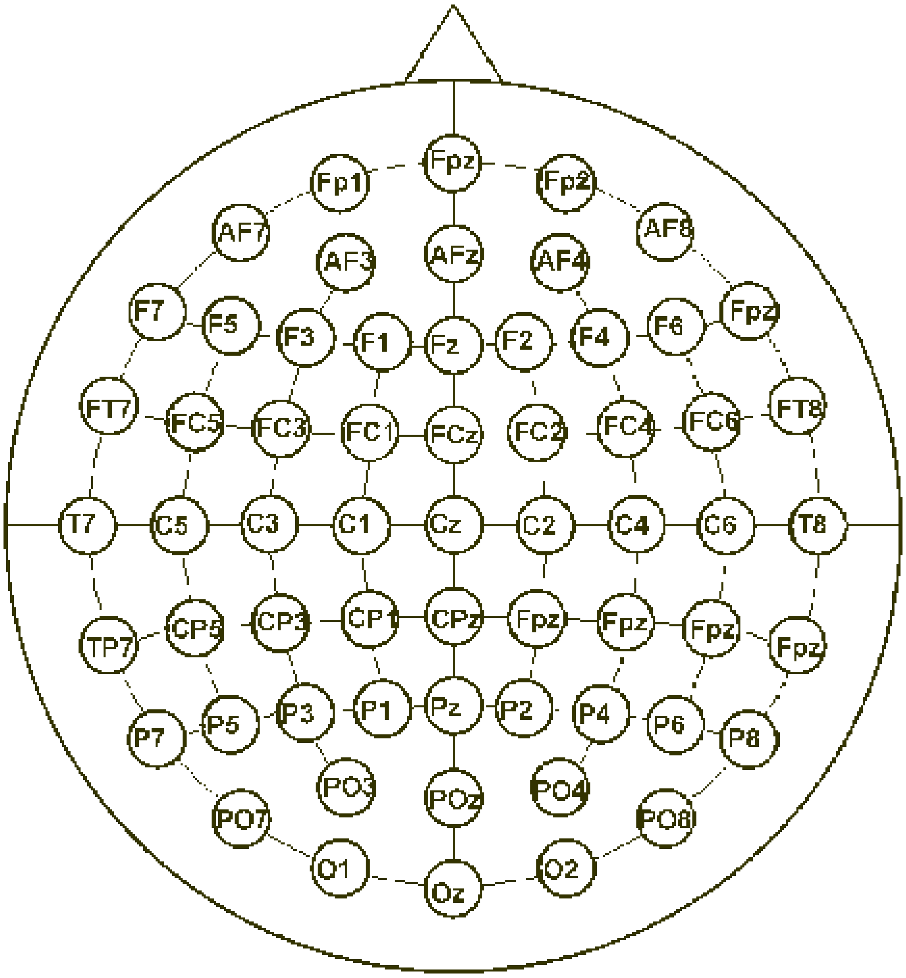


Figure 3.1: Extended International 10-20 System of Electrode Positions

3.2 EEG features

Figure 3.1 shows the extended international 10-20 system of electrode locations on the human scalp. Most EEG studies don't use all of them, but only some subset of interesting locations (also called channels).

In addition to EEG, EMG (electromyogram) of some relevant muscles and EOG (electroculogram) signals are commonly recorded. They are almost exclusively used to discard recording trials that are too contaminated by artifacts like eye blinks or limb movement. Unfortunately, such artifacts can have a substantial impact on the EEG recording, and their removal is generally hard, since EEG affects EOG and EMG. Therefore, plain removal of their impact might also remove parts of EEG not caused by artifacts. EOG mainly affects slow brain potentials, while the EMG primarily affects the oscillatory EEG components. The use of EOG and EMG themselves would be contrary to the very definition of BCI set forth in 1999 on the first international BCI workshop: *A brain-computer interface is a communication system that does not depend on the brain's normal output pathways of peripheral nerves and muscles.*

After the EEG data is collected, a preprocessing phase is usually necessary. It has profound repercussions on performance in terms of accuracy and suitability of different algorithms for classification. There is a trade-off between losing too much information and suffering from the "curse of dimensionality". E.g., a single one second EEG recording sampled at 1000 Hz from all 61 channels from figure 3.1 will have 61000 data points, and that will present a challenge for virtually all machine learning methods. Even with some downsampling, using such raw EEG signals yields noticeably lower classification accuracy; see [14] for example.

The results of preprocessing are a number of EEG "features" that are usually based on known neural characteristics of EEG and are obtained by signal processing methods. Current approaches can be divided into two categories: (i) those using evoked responses, i.e. EEG signals induced by specific external stimuli and (ii) those using spontaneous EEG activities that occur as a result of normal brain function, and are not strongly linked to a particular physical stimulus. The "evoked" approaches are more easily detected and processed, but require that the user's attention be wholly or in part devoted to a stimulus. They fall into three main classes: Evoked Potentials (EP), Event Related Potentials (ERP) and Event Related Desynchronisations (ERD). ERDs are AC changes, while EPs and ERPs are DC changes [1] which occur in response to events. The spontaneous approach uses endogenous signals and are more convenient for the user. Typical examples used in

[1] AC signals change their amplitude as a function of time; DC signals are not oscillatory

BCI are μ and central β rhythms, and Movement Related Potentials (MRPs).

Visual Evoked Potential (VEP) is a typical evoked potential: different visual stimuli will elicit changes in the EEG signal over the visual cortex. One common type of visual stimulus is a pattern reversing image. For example, in [38] two phase reversing black-and-white checkerboards are displayed on each side of the screen. The frequencies at which the patterns are reversed are different for the two sides (the authors used 17 and 20 Hz), and were preselected in an off-line analysis. The user shifts his gaze direction to one of the sides and the steady state visual evoked potentials (SSVEPs) are measured and used to control a character. SSVEP is a periodic response caused by the repetitive presentation of a visual stimulus at a rate of 6-8 Hz or more, and has already been successfully used in BCI (e.g. [46, 76]). VEPs are universally exhibited among humans and are easily detected, since the visual cortex lies close to the scalp.

Event Related Potentials (ERPs) occur in response to, or in advance of a particular event, but do not need a physical stimulus as EPs. The most commonly used ERP potential in BCI is P300. It is a late positive brain wave that occurs about one third of a second after a meaningful stimulus that a subject is instructed to detect. Examples of P300-based BCI are [20, 24]. In [20] the BCI presents a user with a 6×6 matrix of alphabet letters. Rows and columns are intensified one after another. The user focuses his attention on the cell containing the desired letter, so only intensifying the corresponding rows and columns will educe a P300 response. An accuracy of 80% in the offline version has been obtained.

Event Related Sychronisations or Desychronisations (ERS/ERD) are among the most commonly used EEG features in BCI. The frequency analysis of EEG defines five basic frequency bands: delta ($<$ 4 Hz), theta (4–8 Hz), alpha (8–13 Hz), beta (13–30 Hz), and gamma ($>$ 30 Hz). Humans at rest exhibit a strong signal component in the alpha band over the sensorimotor cortex, called the μ–rhythm, and in the central part of the beta band (18–22 Hz), called the central β–rhythm. Actual or imagined movement will cause the amplitudes of these rhythms to decrease, or desynchronise, and that is what the ERD-based BCI approaches exploit. The advantage of ERD over some other EEG features like slow cortical potentials discussed later in this section, is that they do not require training on the subject's part; they are always present. The paper by Wolpaw et al. ([93]) was among the first ERD-based approaches. The authors used the changes in the μ–rhythm during movement planning and execution, both imagined and actual, to control a cursor on a screen. Penny and Roberts in [58] performed similar experiments achieving 75% accuracy and an information transfer rate of one bit per eight seconds, basing the BCI solely on the μ–rhythm. Pfurtscheller also uses ERDs in his BCI work ([61, 59]). Besides the μ and the central β rhythms, he discovered that other frequency bands might also be useful ([60]), as well as the use of autoregressive models, which also pick up oscillatory features. The μ and the central β rhythms occur without stimulus, and belong to the endogenous signals. ERDs are put in the evoked responses group, because they detect the absence of these rhythms during motor processes.

Slow cortical potentials (SCP) are non-oscillatory EEG features, i.e. potentials having very low frequencies of 0.1–0.5 Hz. Birbaumer et al. showed in [13] that healthy subjects and locked-in patients can learn to control the cortical positivity or negativity, i.e. whether the SCP amplitude shifts in an electrically positive or negative direction. This method requires a feedback of the course of subject's SCP and training usually takes a long time before a decent level of accuracy is obtained. In [13] Birbaumer described a system he called a Thought Translation Device (TTD). TTD received a considerable public attention after it was successfully used with ALS patients, and to date remains one of the few practical applications of BCI research.

Movement Related Potential (MRP) is a variant of a slow brain potential specifically related to the preparation and the execution of a motor command. It is a well known fact that motor processes involve several highly localised brain areas, the sensorimotor cortex (EEG channels C3 and C4 in the international 10–20 system of electrode positioning).

Of particular interest for BCI is *Bereitschaftspotential* (BP) or “readiness potential”: a negative EEG potential over the sensorimotor cortex, which slowly increases and reaches a maximum just before the voluntary limb movement. For BCI it is important that this phenomenon occurs for imagined movements as well, although with smaller amplitudes. Blankertz et al. in [14] used this approach to develop a highly accurate ($> 96\%$) binary decision BCI system that didn't need trained users and had short response times. The task subjects performed was self–paced key typing, and the goal was to output whether it was the left or the right hand that pressed a key. The information transmission rate reported was 23 bits per minute, which is at the limit of what is currently possible. They used several classification algorithms, the ones that produced the best results were the Regularised Fischer Discriminant and the Support Vector Machine. However, it should be noted that they worked with healthy subjects who performed actual movements, not imagined movements, which have less expressed BP.

Babiloni et al. in [6] showed evidence that MRP and ERD of the μ–rhythm reflect different aspects of motor cortical processes, and may provide complementary information for BCI. The idea of combining these two features was studied in [21], which further supports this hypothesis. On the same problem of self-paced key typing, but here with imagined movements, they managed to reduce the average error rate for three subjects from 13.7% to 9%.

Except for the Birbaumer's TTD, most studies have worked with healthy subjects. The data sets IA and IB of the BCI 2003 competition ([64]) indicate possible problems of applying such approaches to locked-in patients. Both data sets involved experiments having two possible outcomes. Data set IA was taken from a healthy subject and the winning submission achieved an error rate of 11.3%. Data set IB was taken from a locked-in patient with Amyotrophic Lateral Sclerosis's (ALS) and the best submission achieved an error rate of 45.6%, which is close to chance level, i.e. 50%.

In the early days of BCI, classification methods were almost strictly linear. Nowadays, with more powerful computers, neural networks are commonly used, and other more advanced machine learning methods can be found as well. However, several authors have reported that neural networks and other nonlinear classifiers did not manage to outperform linear classifiers, which suggests that nonlinearities in the EEG data are removed in the feature extraction phase, before the classifiers get to do their work.

Chapter 4

A new scheme for neural network multitask learning

4.1 Bayesian neural networks

In this chapter, I will introduce a new learning architecture. Like other multitask learning algorithms, in a situation with some number of related tasks, it will try to improve the accuracy and/or speed of training of some or all of the tasks. Like the approaches considered in section 2.5, it will do that by learning them in parallel. The algorithm is in part based on the idea of the random coefficient models covered in chapter 2.

As for Caruana's MTL algorithm for a feedforward backpropagation neural network (MLP - from Multilayer perceptron), the new algorithm will also be based on a MLP, but it will use the Bayesian approach to learning for neural networks developed in [53].

Figure 4.1 shows a typical MLP with one hidden layer, having three inputs, four units in the hidden layer and a single output unit.

The output of such a network on input (x_1, x_2, x_3) is then:

$$o = a + \sum_j v_j h_j \tag{4.1}$$

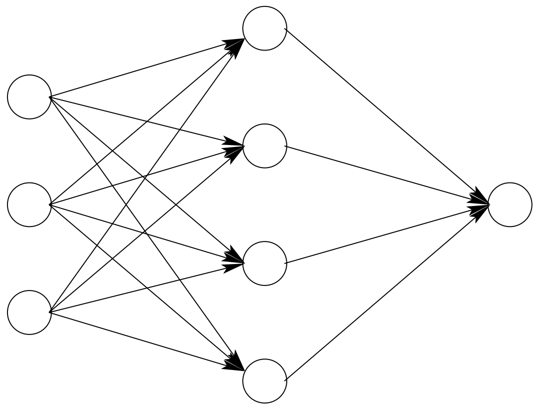


Figure 4.1: Backpropagation neural network

$$h_j = f(b_j + \sum_i w_{i,j} x_i) \tag{4.2}$$

where a and b_j's are biases for the output and the j^{th} hidden unit respectively, $w_{i,j}$ is the weight from the i^{th} input to the j^{th} unit in the hidden layer, v_j is the weight from the j^{th} unit in the hidden layer to the output and f is an activation functions.

Such MLPs can be used to define probabilistic learning models by using the network outputs to define a conditional distribution for a target y, given the input vector x.

If we are using a neural network for a regression task, we can define this conditional distribution to be Gaussian with mean $o_k(x)$ (the k^{th} network output) and a standard deviation σ_k. The different outputs o_k are taken to be independent given the input, so the distribution is given by:

$$P(y|x) = \prod_k \frac{1}{\sqrt{2\pi}\sigma_k} \exp\left(\frac{-(o_k(x) - y_k)^2}{2\sigma_k^2}\right) \tag{4.3}$$

The parameters σ_k might be fixed, or might be regarded as hyperparameters.

In a classification task, where the target y is a single value specifying one of the K possible classes that the associated input x belongs to, a common choice is to use the softmax model. In this model, the conditional probabilities of each of the K classes, given input x, and using a neural network with K outputs, is defined as:

$$P(y = k|x) = \frac{\exp(o_k(x))}{\sum_{k'} \exp(o_{k'}(x))} \tag{4.4}$$

The neural network parameters (weights and biases) are learned given a training set $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)})\}$ of inputs $x^{(i)}$ and corresponding targets $y^{(i)}$. The conventional learning methods are based on an error function defined on the network output for the training cases; usually it is the sum of the squared differences between the network outputs and the targets, or minus the log probability of the target values. Next, a training procedure, usually gradient based, is used to adjust the weights and biases, so as to minimise this error function. To reduce overfitting, a penalty term proportional to the sum of squares of the weights and biases is often included in the error function. This term, known as weight decay, makes the learning procedure favour models with smaller weights and biases.

The objective in the Bayesian approach is to find the predictive distribution for the target values for a new test case, given the input for that case, as well as the targets and the inputs for the training cases. This distribution is obtained by integrating the predictions of the model with respect to the posterior distribution of the network parameters:

$$\begin{aligned} &P(y^{(n+1)}|x^{(n+1)}, (x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})) \\ &\qquad = \int P(y^{(n+1)}|x^{(n+1)}, \theta) P(\theta|(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})) d\theta \end{aligned} \tag{4.5}$$

In the above expression, θ represents the neural network parameters: weights and biases. The posterior distribution for these parameters is proportional to the product of a prior and the likelihood function given by:

$$L(\theta|(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)}))) = \prod_{i=1}^{n} P(y^{(i)}|x^{(i)}, \theta) \tag{4.6}$$

This predictive distribution can then be used for various purposes. If we want to guess a single component of $y^{(n+1)}$ and we have a squared loss, we would use the mean of the predictive distribution. In a regression model, this is given by:

$$\hat{y}_k^{(n+1)} = \int o_k(x^{(n+1)}, \theta) P(\theta | (x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})) d\theta \tag{4.7}$$

The computation of the predictive distribution (4.5) is often analytically intractable. Even the common approximation methods will be in trouble with such high dimensional integrals. A practical method using Markov chain Monte Carlo techniques was introduced by Neal [53, 51], and the MTL model described in this chapter is based on that implementation. Instead of a direct computation, the posterior distribution is estimated using a sample of values for the network parameters taken from that same distribution. Ideally, the values would be independent, but with complicated distributions that arise with neural networks, this is often infeasible. However, dependent values still give an unbiased estimate. Such a series of dependent values can be generated using a Markov chain that has the true distribution as its stationary distribution.

One simple Markov chain Monte Carlo method is Gibbs sampling, also known as the heatbath method. It generates the next value component by component, each one being chosen from the conditional distribution given all the other components. Gibbs sampling isn't directly applicable to Bayesian neural networks, as the posterior distribution is usually complex, and sampling from these conditional distributions is infeasible.

The Metropolis algorithm [45] is another well-known Markov chain Monte Carlo method. The next value is obtained by generating a candidate from a proposal distribution that may depend on a current value. The candidate is then accepted with some probability.

Neal's flexible Bayesian modeling software [1] uses the hybrid Monte Carlo algorithm [22] that combines the stochastic dynamics method [4] with the Metropolis algorithm. The problem is defined in terms of potential energy $E(x)$, and the parameter distribution $P(x)$ is proportional to $\exp(-E(x))$. The total energy is a sum of "potential" energy

[1] available online at http://www.cs.toronto.edu/~radford/fbm.software.html

and "kinetic" energy due to the momentum, which allows the use of dynamical methods. The Markov chain is obtained by performing deterministic dynamical transitions and stochastic Gibbs sampling updates of the momentum. However, stochastic dynamics methods suffer from a systematic error. By combining it with the Metropolis algorithm, this error can be eliminated.

Hybrid Monte Carlo is used for updating the network parameters. Hyperparameters are updated separately using Gibbs sampling. It usually takes some time before the chain converges to its stationary distribution. This can be visually verified and we discard these burn-in iterations in the Monte Carlo estimate of the network parameters.

The first step in Bayesian inference is to select a prior for the model parameters, which is supposed to embody our prior knowledge about them. This may seem a problem, since the parameters of an MLP do not have a meaningful interpretation. However, past work shows that suitable priors can be found nevertheless ([15, 41, 42, 53]). Furthermore, Neal in [52] showed that priors over hidden units can be scaled according to the number of parameters, so that priors over computed functions reach reasonable limits as we increase the number of hidden units. This avoids overfitting, so the correct approach is to use as many hidden units as is computationally feasible.

4.2 Description of the new model

As we saw in Chapter 2, the classical MTL learning approach for MLPs is to add an additional output unit for each additional task, together with the corresponding hidden to output weights. The shared representation in this case is the input to hidden weights. One shortcoming of this approach is that it works better if the number of tasks we are learning is not small. This is also true for methods that also try to couple the hidden to output weights such as that of Bakker and Heskes [7]. They work best when the number of tasks is much bigger than the number of hyperparameters used to define the connection among the hidden to output weights. The more elaborate the algorithm, the bigger the number of hyperparameters required – the tests Bakker and Heskes used typically had the tasks numbering in hundreds.

The new model is also an extension of the ordinary MLP. Let's suppose that we have k tasks and that in a single task learning approach we would train an MLP with one hidden layer having N inputs and H units in the hidden layer for each task. For now, I will assume only one output as more outputs do not present a conceptual difficulty. If the tasks really are sufficiently related, the number of units in the hidden layer should be close enough so that we could agree on a common one. In the Bayesian model which I adopt, this is not an issue. If some care is taken while specifying the prior ([52]), the proper approach is to use as many units in the hidden layer as is computationally feasible since there is no danger of overfitting. The number of input units (and the meaning of the inputs themselves) does not have to be the same for all the tasks, but just as in Caruana's approach, if there are no common input features, there will be no benefits using the model I'm about to describe. The tests I did in later chapters all use the same number of inputs with the same meaning, which is probably the most typical situation if the tasks are substantially related.

Figure 4.2 shows the situation in a simple case. For each of the k tasks we have a layer with H units connected to the input layer and to the output unit. For a particular data point only the layer corresponding to the task to which the data point belongs will be used to compute the forward and the backward information. At this point there is nothing that links different layers – this is the purpose of the common layer, which will do that by "overlaying" other layers.

We will distinguish between overlaying the input to hidden and hidden to output connections. To say that a common layer C overlays the input to hidden connections of a task n specific layer will mean the following:

- For a particular training case and a given activation function f, the values of the

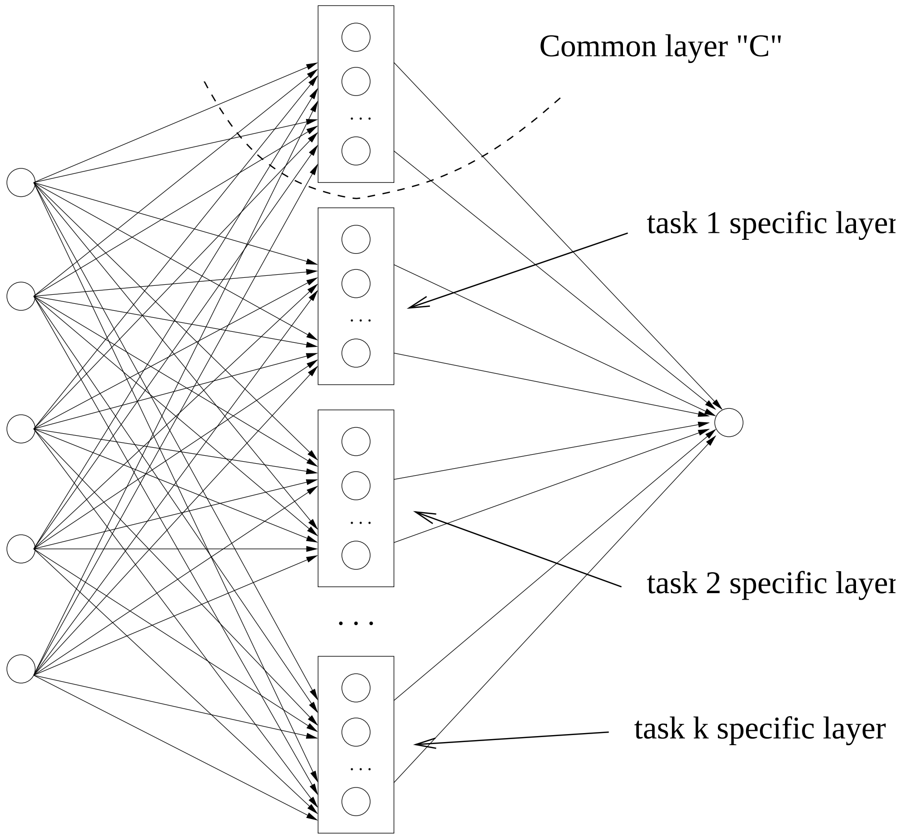


Figure 4.2: The MTL model

units in the common layer C will be computed by the usual MLP rules:

$h_j^C(x) = f(b^C + \sum_i w_{i,j}^C x_i)$

- The values of the units in the task n specific layer will be computed by:

  $h_j^n(x) = f(b^C + b^n + \sum_i (w_{i,j}^C + w_{i,j}^n) x_i)$

To say that a common layer C overlays the hidden to output connections of a task n specific layer will, in a similar fashion, mean that the contribution of the layer C to the output will be computed by the ordinary MLP rules, and the contribution of the task n specific layer will be:

$$\sum_i (w_{i,out}^C + w_{i,out}^n) h_i^n$$

This was intentionally worded as a contribution instead of giving a formula for the output value. The reason is that the values of the layer C might be discarded or not computed at all; its impact on the output value might be solely through the effects of overlaying the task specific layers. We can however choose to use the common layer directly as well.

Let's look at the case with a single unit in all the hidden layers and identity activation functions. Furthermore, let's assume we have one common hidden layer with input to hidden weights W and a bias b, and n task-specific hidden layers with input to hidden weights W_j and biases b_j, $j = 1, \ldots, n$. The output of a unit in the hidden layer corresponding to the j_{th} task with an input vector X for that task will be:

$$h_j(X) = b + b_j + X(W + W_j) \tag{4.8}$$

This value would be multiplied by the hidden to the output weight v^j and the output bias would be added to form the final result. Let's suppose that we keep the v^j's fixed at 1, and the output bias at 0. Then the above output would be the final output for the task j on the input vector X.

Let's recall the random regression coefficients model from section 2.7:

$$y_{ij} = (M + \mu_j) + X_{ij}(\beta + \gamma_j) + \epsilon_{ij} \tag{4.9}$$

We have been defining the behaviour of the model on a generic input vector X in this chapter. The index i in X_{ij} in the above formula indicates the i^{th} input vector from some set of training data for the task j, which we can omit here. By taking $M = b$, $\mu_j = b_j$, $\beta = W$ and $\gamma_j = W_j$, we can think of the simple model (4.8) as a neural network based method for finding the parameters of the random regression model. The bigger potential power and flexibility of the new neural network based model lies in the use of nonlinear activation functions.

More than one common layer can be used. For example, if we had 15 related tasks that could further be grouped into 3 categories, we might use a model with one common layer which would overlay all the task specific layers, and 3 additional layers which would overlay only the layers associated with the task of a particular category. An example of such a data set would be collecting blood pressure measurements from 3 countries, each of which was represented with 5 hospitals. Additionally, common layers can be themselves overlaid by other layers, allowing a nested design. These categories or clusters must be known beforehand.

Although a one-hidden-layer MLP can learn the same set of functions as MLPs with more hidden layers (provided it has enough units in its single hidden layer), sometimes it is advantageous to use MLPs with more than one hidden layer. In such a situation we may want to overlay the hidden to hidden connections as well. Figure 4.3 shows a situation where this could be done. It is a direct generalisation of a MLP with two hidden layers: both the common part and all the task specific parts will have two hidden layers. Let's order the layers so that for each pair of layers that are connected, the index of the source layer is smaller by one than the index of the target layer of the hidden to hidden connections. The output of the i^{th} unit in the l^{th} layer, assuming l is the index of a layer that has input connections from the other hidden layer is defined as:

$$h_i^l = f_l \left[A + \sum_{k=1}^{N_{l-1}} \left(W_{k,i}^{l-1,l} h_k^{l-1} + \sum_{j=1}^{|\Gamma^l|} W_{k,i}^{\Gamma_j^l - 1, \Gamma_j^l} h_k^{l-1} \right) \right] \tag{4.10}$$

where:

- f_l is the activation function of the l^{th} hidden layer.

- A is the contribution of the input to hidden connections. They could also be overlaid and the contribution would then be computed as in the input-to-hidden weights overlaying case. Typically, there are no input to hidden weights to the second-level hidden layer, as is the situation in the figure 4.3, in which case A is 0.

- N_{l-1} is the number of units in the hidden layer with index $l-1$

- $W_{k,i}^{t,s}$ is the weight from the k^{th} unit in the hidden layer t to the i^{th} unit in the hidden layer s. With the ordering convention above, we will always have $t = s - 1$.

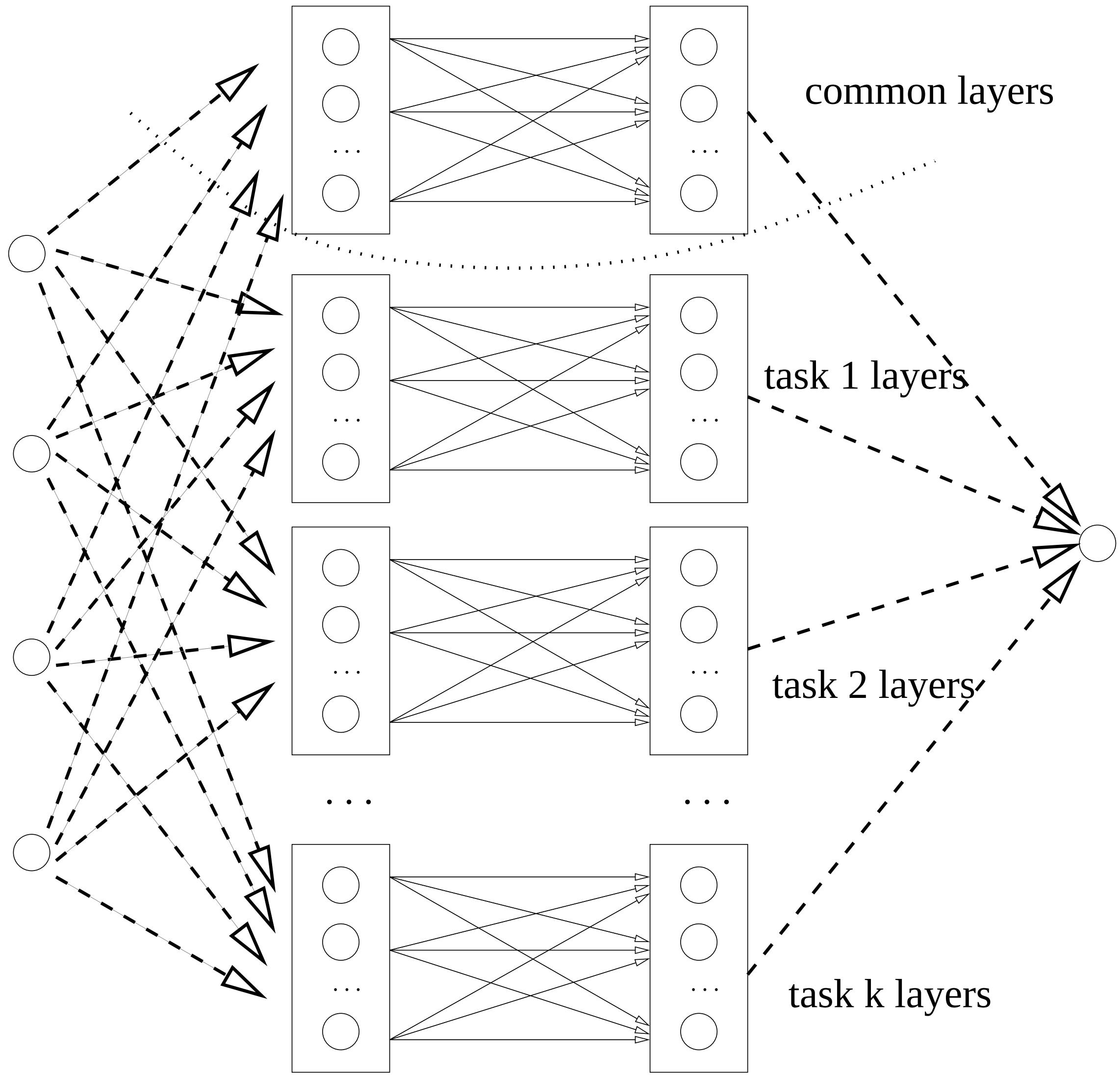


Figure 4.3: The MTL model with two hidden layers per task

- h_k^l is the output value of the k^{th} unit in the l^{th} hidden layer.

- $\Gamma^l = \{\Gamma_1^l, \ldots, \Gamma_{|\Gamma^l|}\}$ is the set of indices of layers that have hidden to hidden connections that overlay hidden to hidden connections of the layer l.

The biases are omitted from the formula. They usually exist, and are overlaid in the same way as all other weights. The formula can be generalised to any number of hidden layers per task.

If the tasks are significantly related, it would be expected that the between cluster variation is small. In our model this corresponds to the weights and biases in the task specific layers being significantly smaller than the weights and biases in the common layer(s). This can be achieved by selecting different priors for different model parameters. Priors in Neal's approach are specified with hyperparameters which are organized into a hierarchical model. This allows us to control a whole group of parameters (such as the input to hidden weights in the common layer) with a single hyperparameter. By making these hyperparameters themselves be variable, with moderately vague priors, the model can determine itself how related tasks are.

The new model was implemented as an extension of flexible Bayesian modeling software mentioned in section 4.1. It bears some resemblance with the Bakker and Heskes' Bayesian multitask learning model [7] from section 2.6. Their simplest model uses the same prior for all hidden to output weights, which is a typical situation for our model too if we restrict ourselves to the task specific weights. More complex models that include clustering of tasks and task dependent prior means are specified with explicit formulas. Since we are specifying priors for each group of weights individually, we could adjust the priors to produce a similar effect. They use normal distributions, while we use gamma distributions for specifying priors, so we can't directly use the formulas. A key difference between the models is that Bakker and Heskes share the input to hidden weights among tasks and treat them as a hyperparameter, while we have a separate hidden layer for each task that share an extra, common hidden layer. Also, we have one layer for each task and a single, shared output unit, while they have a single shared hidden layer and an output for each task.

The empirical studies show that training the model takes time comparable to training all the tasks individually.

Chapter 5

Experiments to test the multitask learning scheme

This chapter will demonstrate that the multitask learning method introduced in Chapter 4 is able to exploit task relatedness to improve its learning performance. All the data in this chapter were artificially generated.

5.1 Linear data

The first few examples are kept simple to make it is easier to gain insight into the behaviour of the method. In the first example we have two tasks, each having one hundred training data points. The inputs are real numbers drawn randomly from $[0, 10]$. The outputs for the tasks are generated from the following formulas:

$$y_1 = 1.40x + 2.30 + (1/2)\epsilon \tag{5.1}$$

$$y_2 = 1.34x + 2.80 + (1/2)\epsilon \tag{5.2}$$

where ϵ is drawn from a normal distribution with mean zero and standard deviation one. A natural single task learning network for this problem would have no hidden layers. However, we cannot extend such a network to our multitask learning model, as there

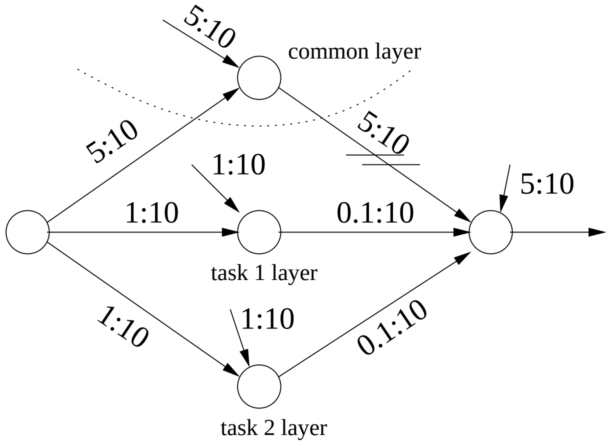


Figure 5.1: The MTL model for a simple two-task linear regression problem.

are no weights to be overlaid. This is because all the tasks share the same inputs and outputs, and no multiple input to output weights are allowed. Instead, we will be using the model shown in figure 5.1

The model has three hidden layers, each with a single unit. The topmost layer/unit is the common layer; its input to hidden and hidden to output weights overlay the corresponding weights of the two other, task specific, layers. Both hidden units and the output unit use the identity activation function. Only one task specific layer is active for a particular data point. The common layer does not contribute to the output, other than through the effect of overlaying. This is indicated by the two horizontal lines over the arrow representing its hidden to output weights. Since we are using a Bayesian approach to neural networks, we need to specify priors. The numbers above the links indicate the specification for the priors of the associated weights in the same syntax as used by R. Neal in his flexible Bayesian modeling software[1]. These models use a hierarchy of hyperparameters to control the prior distributions of individual network parameters. Full details on how the hyperparameters are handled are given in [53]; here, I will only explain what is needed to understand the neural network in figure 5.1.

Priors for the parameters in a particular group $\{u_1, u_2, \ldots, u_k\}$ are taken to be inde-

[1] available online at http://www.cs.toronto.edu/~radford/fbm.software.html

pendent, conditional on the value of the associated hyperparameter, and have a Gaussian distribution with mean zero and standard deviation σ_u. For convenience, σ_u is represented in terms of the corresponding precision $\tau_u = \sigma_u^{-2}$. The distribution of $\{u_1, u_2, \ldots, u_k\}$ is hence given by:

$$P(u_1, u_2, \ldots, u_k | \tau_u) = (2\pi)^{-\frac{k}{2}} \tau_u^{\frac{k}{2}} \exp(-\tau_u \sum_i \frac{u_i^2}{2}) \tag{5.3}$$

The prior for the common precision is a Gamma distribution with some mean ω_u and shape parameter α_u:

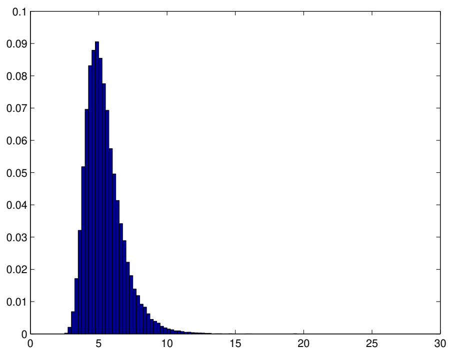
$$P(\tau_u) = \frac{(\alpha_u / 2\omega_u)^{\frac{\alpha_u}{2}}}{\Gamma(\frac{\alpha_u}{2})} \tau_u^{\frac{\alpha_u}{2}-1} \exp(-\tau_u \frac{\alpha_u}{2\omega_u}) \tag{5.4}$$

In figure 5.1, the precision for the hyperparameters for a group (in our case all the groups are singletons), τ_u, is specified in the form $\frac{1}{\sqrt{\omega_u}} : \alpha_u$. For illustration, figure 5.2 shows the probability distributions for some selections of means and shape parameters used in this chapter.

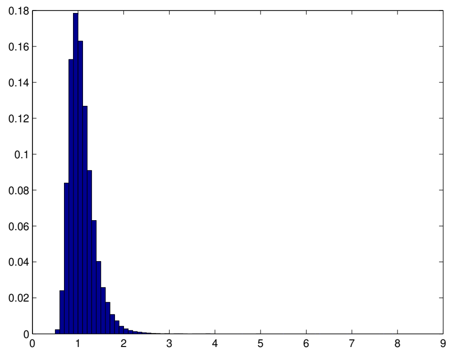
Smaller values of the shape parameter α_u will give more vague prior distributions, i.e. they will be more spread around the mean, while the bigger values will give distributions more concentrated around the mean.

To use the Bayesian approach, we need to specify the noise level σ_k of equation (4.3). The noise levels are usually regarded as hyperparameters and in the above case were specified with $0.5 : 10$. This is a natural choice since we know the data was generated by adding noise with mean zero and standard deviation 0.5, but we want to pretend we don't have precise knowledge of σ_k.

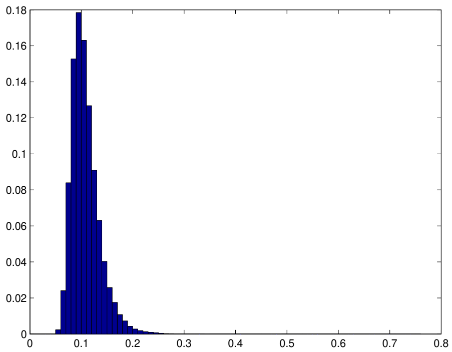
The corresponding STL neural network is shown in figure 5.3. Besides training one such network for each task, one more neural network was trained that regarded all the data as belonging to the same task, i.e. it did not distinguish between the tasks (from now on I will refer to this model as the NDT model – **N**ot **D**istinguishing **T**asks). It typically used the same architecture and prior specification as the STL network.



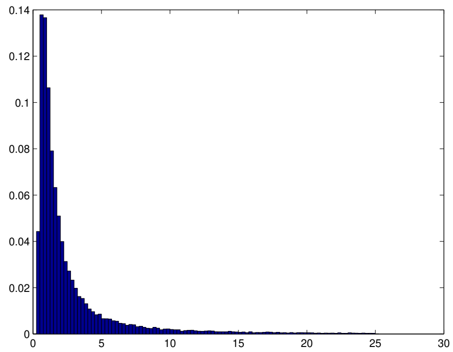
(a) 5:10



(b) 1:10



(c) 0.1:10



(d) 1:1

Figure 5.2: Histogram of the distributions of deviations σ_u for different prior specifications.

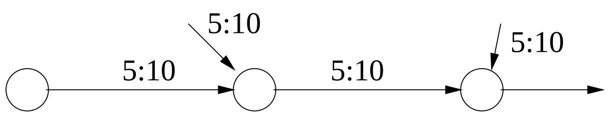


Figure 5.3: The STL model used for a simple two-task regression problem.

Ten training data sets were generated and the MTL, STL and NDT models were trained on each. Their performance was evaluated on a test data set with 10,000 points (5,000 per task). The average performance is summarised in table 5.1, which gives the average squared error of both tasks if we guess the mean of the predictive distribution.

trial	MTL	STL	NDT
1	0.256	0.257	0.262
2	0.250	0.249	0.262
3	0.261	0.262	0.265
4	0.251	0.251	0.265
5	0.256	0.255	0.262
6	0.248	0.248	0.271
7	0.249	0.249	0.262
8	0.246	0.247	0.264
9	0.247	0.247	0.262
10	0.248	0.248	0.263
average	0.251	0.251	0.264

Table 5.1: Average squared error on two-task linear regression problem.

With such a simple task and a large training data set, it is no surprise that the STL network performs almost optimally (0.25 being the average squared error with optimal prediction). The difference between the MTL and STL is negligible. The NDT version, on the other hand, performs significantly worse.

We'll look at the first trial more closely. The Markov chain converged after a couple of iterations, 900 of the 1000 iterations were used to produce table 5.1. The output of each individual network is a linear function of its input, so we can calculate the expectations for parameters of the final linear model produced by the sampled networks. The common part on its own computes:

$$y_c = 1.48x + 2.55 \tag{5.5}$$

while the two task specific parts, after accounting for overlaying, compute:

$$y_1 = 1.38 + 2.36 \tag{5.6}$$

$$y_2 = 1.35x + 2.66 \tag{5.7}$$

The difference is:

$$y_1^d = -0.10x - 0.19 \tag{5.8}$$

$$y_2^d = -0.13x + 0.11 \tag{5.9}$$

Just as a comparison, an ordinary least square estimate for all the data is given by:

$$y_c = 1.36x + 2.50 \tag{5.10}$$

and for the specific tasks:

$$y_1 = 1.38x + 2.34 \tag{5.11}$$

$$y_2 = 1.35x + 2.66 \tag{5.12}$$

We can see that the common part is doing the bulk of the job, as we would want it in this case, when the tasks are very similar.

Let's change the priors of the network weights to be as in figure 5.4.

The difference is that the priors for the task specific weights are now more concentrated on the mean (α_u parameters), and the inverse square root of the mean of the precision is now 0.1 instead of 1 for the input to hidden weights. The sampled networks defined the function for the common part to be:

$$y_c = 1.40x + 2.51 \tag{5.13}$$

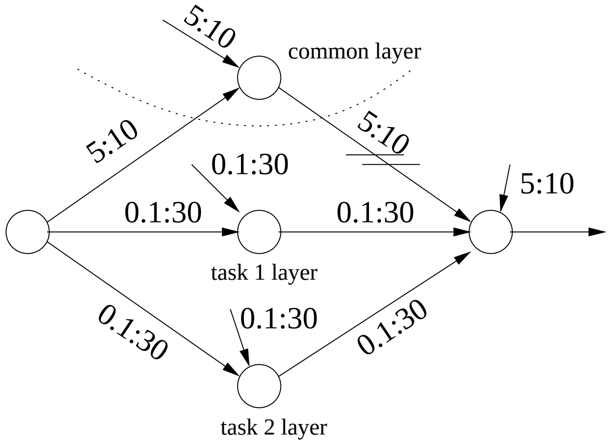


Figure 5.4: The new MTL model for a simple two-task regression problem.

and the difference of the functions generated for the task specific part from the common part is now:

$$y_1^d = -0.03x - 0.13 \tag{5.14}$$

$$y_2^d = -0.05x + 0.12 \tag{5.15}$$

The influence of the task specific layers is now much smaller, and we see that y_c is much closer to the OLS estimate (5.10). However, the performance also came closer to the NDT version; the average squared error over all trials was 0.258. We can also give vague priors for all weights and give more freedom to the model to select the weights. In this situation the performance didn't change much compared to MTL in table 5.1, which is no surprise as it would be expected that the performance would be between the MTL version with carefully chosen priors and the STL version, and these two were basically the same.

Two tasks and/or big training data sets are not the intended scenario for the use of the MTL methods. The following experiments still use the linear data, but now we will have four tasks and only six training data points per task. The data for the four tasks

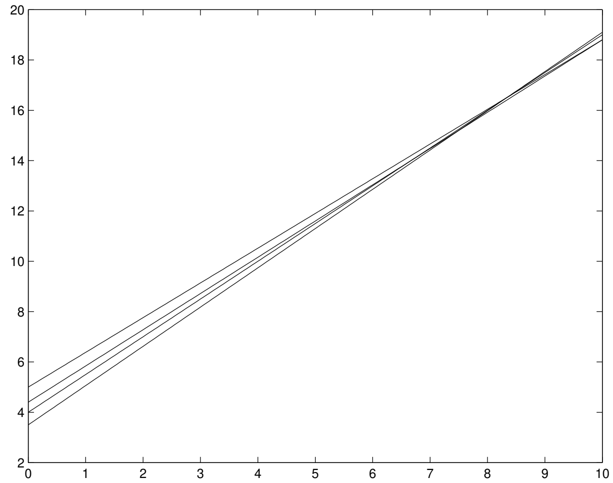


Figure 5.5: The functions used to generate the training data, dataset 1.

were generated as follows:

$$y_1 = 1.50x + 4.00 + (1/2)\epsilon \tag{5.16}$$

$$y_2 = 1.44x + 4.40 + (1/2)\epsilon \tag{5.17}$$

$$y_3 = 1.38x + 5.00 + (1/2)\epsilon \tag{5.18}$$

$$y_4 = 1.56x + 3.50 + (1/2)\epsilon \tag{5.19}$$

where x, ϵ were generated exactly as in the previous example. The lines are shown in figure 5.5.

The prior specification for the weights, as well as the network architecture, is shown in figure 5.6. The common layer, again, does not contribute to the output on its own.

Again, an MTL network, four STL networks and one NDT network that doesn't distinguish the tasks were trained on 10 training sets. Table 5.2 gives the average squared error computed by guessing the mean of the predictive distribution defined by the sampled neural networks. The test data set had 10,000 data points.

The MTL method outperformed the STL method on eight out of the ten training batches. Setting the priors for the weights to give more substance to the common weights does provide a small performance boost. The best performance achieved was an average squared error of 0.329. Giving the weights vague priors didn't influence the performance

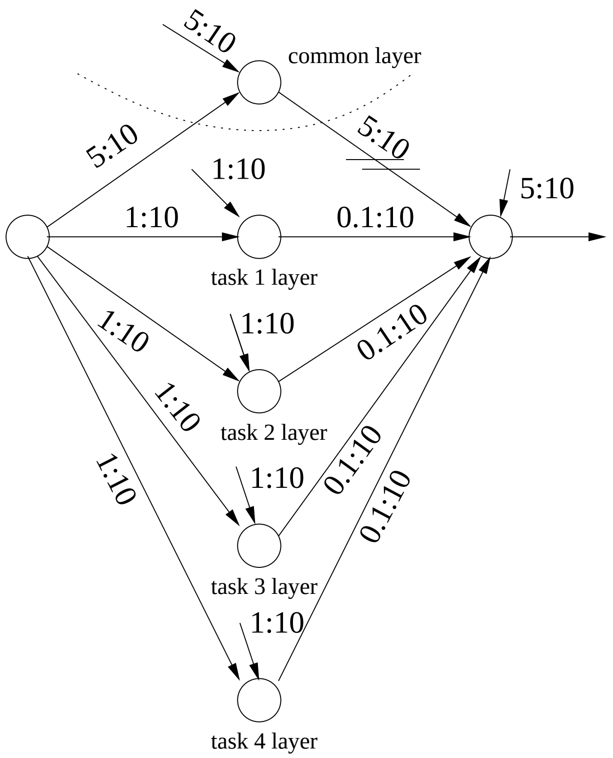


Figure 5.6: The MTL model for a simple four-task linear regression problem.

Trial	6 points per task			3 points per task		
	MTL	STL	NDT	MTL	STL	NDT
1	0.433	0.396	0.369	0.326	0.337	0.348
2	0.389	0.432	0.433	0.310	0.291	0.332
3	0.284	0.296	0.337	0.426	0.472	0.408
4	0.294	0.317	0.347	0.366	0.361	0.360
5	0.307	0.340	0.360	0.354	0.385	0.354
6	0.299	0.307	0.351	0.581	0.744	0.537
7	0.283	0.303	0.375	0.302	0.298	0.347
8	0.425	0.426	0.350	0.328	0.361	0.376
9	0.310	0.331	0.341	0.431	0.489	0.416
10	0.321	0.300	0.392	0.340	0.343	0.345
Average	0.335	0.345	0.366	0.376	0.408	0.383

Table 5.2: Average squared error on four-task linear regression, dataset 1.

much; using a 1:1 specification for all the weights gave an average squared error of 0.338 – still noticeably better than the STL version.

Results using three training data points per task are also shown in table 5.2. The MTL version did much better than the STL, but was only slightly better than the NDT version.

In the next example, the training data for the tasks were generated from:

$$y_1 = 1.52x + 5.50 + (1/2)\epsilon \tag{5.20}$$

$$y_2 = 1.47x + 3.80 + (1/2)\epsilon \tag{5.21}$$

$$y_3 = 1.51x + 1.60 + (1/2)\epsilon \tag{5.22}$$

$$y_4 = 1.55x + 4.20 + (1/2)\epsilon \tag{5.23}$$

If we draw the lines (figure 5.7), or look more carefully at the above formulas, we can see

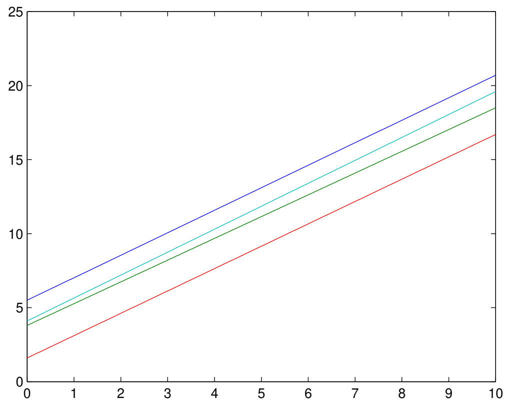


Figure 5.7: The functions used to generate the training data, dataset 2.

Trial	6 points per task				3 points per task			
	MTL	MTL2	STL	NDT	MTL	MTL2	STL	NDT
1	0.315	0.311	0.319	2.380	0.321	0.300	0.306	2.361
2	0.308	0.310	0.321	2.350	0.272	0.272	0.274	2.412
3	0.344	0.336	0.346	2.360	0.403	0.399	0.412	2.393
4	0.344	0.330	0.324	2.364	0.319	0.313	0.322	2.472
5	0.423	0.426	0.425	2.380	0.383	0.368	0.376	2.383
6	0.416	0.401	0.404	2.426	0.622	0.628	0.676	2.379
7	0.288	0.288	0.302	2.589	0.382	0.343	0.367	2.406
8	0.418	0.373	0.362	2.364	0.789	0.761	0.952	2.391
9	0.297	0.295	0.294	2.382	0.298	0.292	0.293	2.396
10	0.356	0.344	0.357	2.392	0.332	0.324	0.331	2.410
Average	0.351	0.341	0.345	2.414	0.413	0.399	0.432	2.412

Table 5.3: Average squared error on four-task linear regression with six and three data points per task, dataset 2.

that they all have similar slopes, but the intercepts are not so close. Using the same network and prior specification as in figure 5.6, and using six training data points per task, the results are given in table 5.3.

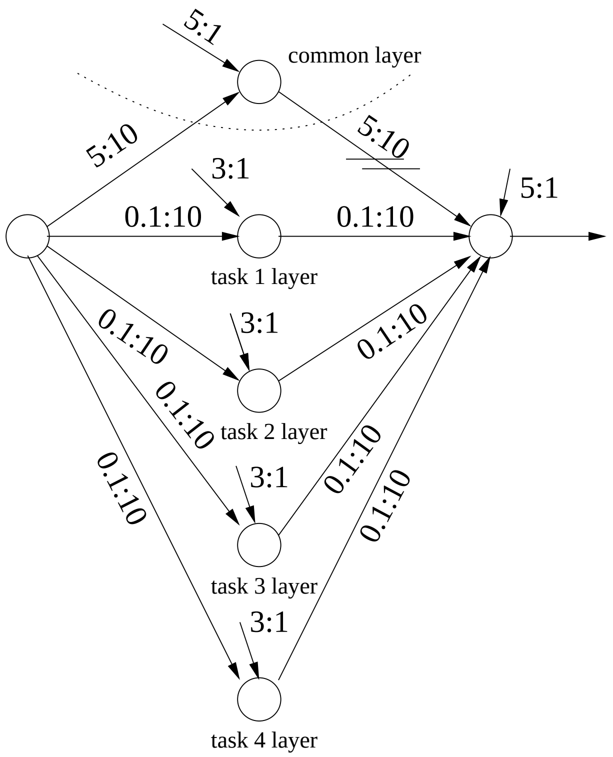


Figure 5.8: The MTL model with vague priors for the biases.

The NDT version was bad, as expected, since a single linear function cannot fit the data points from four lines whose values are this different. The MTL method, on the other hand, managed to keep up with the STL method. Six points is still quite enough for the STL version to work very well, but when the same experiment was repeated with three data points per task, the MTL version performed better. However, in this situation we know that we should give more freedom to the biases for the task specific layers, while forcing the input to hidden weights for the task specific layers to be small. Using priors as in figure 5.8, which we refer to as MTL2, performance is slightly better with six training points than STL, but not by much; it achieved an average squared error of 0.341. While we can pinpoint the slope, the intercept still remained the problem as task relatedness does not help us here. The difference is a bit more obvious with three points per task – MTL2 achieved quite a decent average squared error of 0.399 compared to 0.432 of STL. The NDT version was always above 2.4.

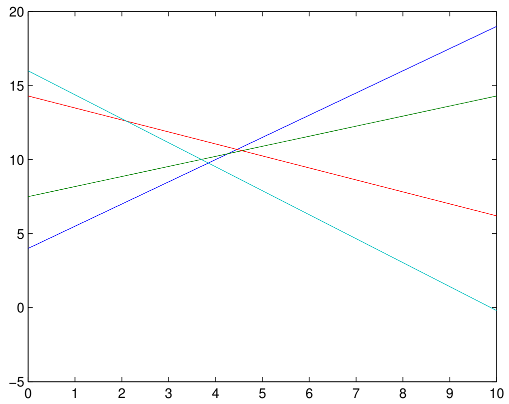


Figure 5.9: The functions used to generate the "unrelated" training data, dataset 3.

In the last example with linear data, we will have four rather unrelated tasks, whose output values were generated by:

$$y_1 = 1.50x + 4.00 + (1/2)\epsilon \tag{5.24}$$

$$y_2 = 0.68x + 7.50 + (1/2)\epsilon \tag{5.25}$$

$$y_3 = -0.81x + 14.30 + (1/2)\epsilon \tag{5.26}$$

$$y_4 = -1.62x + 16.00 + (1/2)\epsilon \tag{5.27}$$

The graph of the generating functions is shown in figure 5.9.

Using the same priors as in figure 5.6 wasn't a very good idea here. This is understandable. Forcing the weights associated with the task specific layers to be very small means that it will only be able to account for small differences between the tasks. And here, the differences are big. The performance is summarised in table 5.4

Using the prior specifications as shown in figure 5.10, however, yields much better results. The average squared error was 0.387, which is even better than the STL model. One of the training sets gave very poor results (STL:0.659, MTL:0.635), so the STL version is noticeably worse than in the previous cases, although it shouldn't care that the tasks aren't related. Without that trial, the MTL and the STL version performed almost identically.

A typical network sampled from the posterior of the MTL model is shown in figure

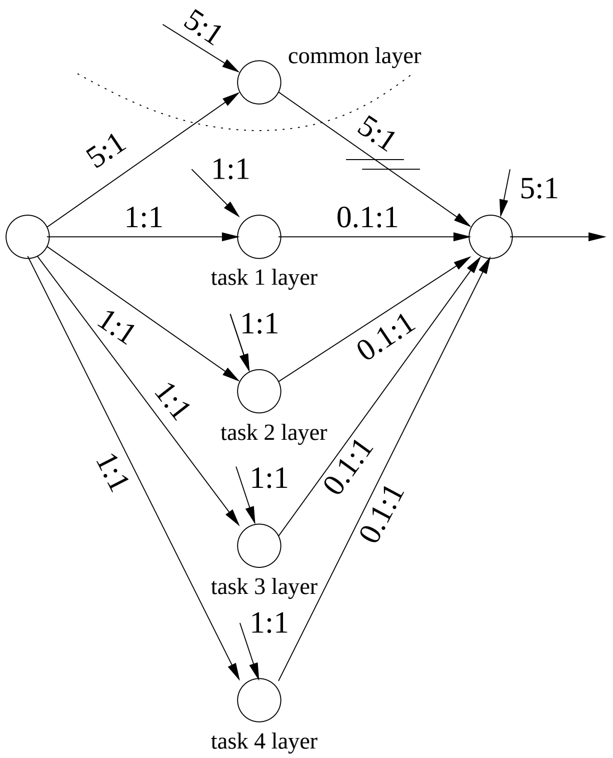


Figure 5.10: MTL model with vague priors for all the weights.

	Average squared error
MTL	0.44819
STL	0.39569
NDT	15.09064

Table 5.4: Average squared error on four-task linear regression with six data points per task, dataset 3.

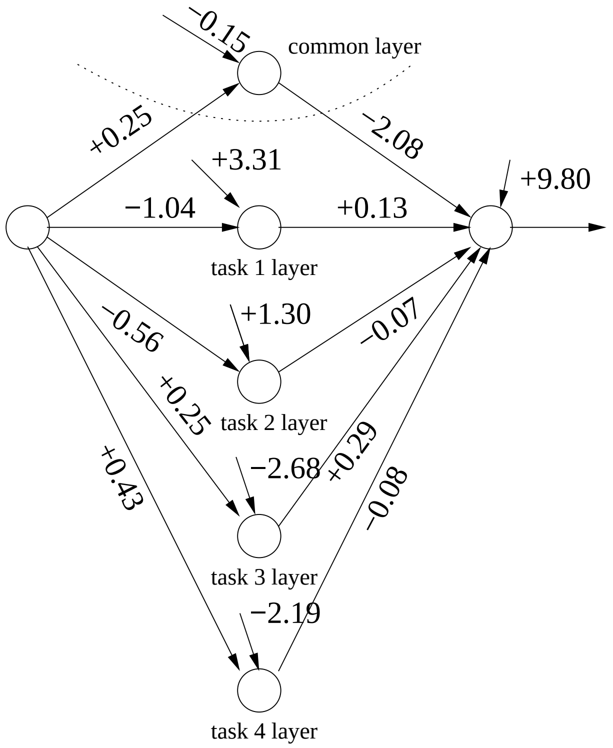


Figure 5.11: MTL network sampled from a model with vague priors.

5.11.

Even though we still favoured bigger weights for the common part, the weights in the task specific parts are fairly large in the networks used in the Monte Carlo estimation.

From the simple examples so far, we can see that if the tasks are substantially related, the MTL method will generally outperform both the STL model and the NDT model. And if we have knowledge about the way they are related, we can use that knowledge to tune the priors and further improve the performance of the MTL method. If the tasks aren't related, using more vague priors, the MTL method will generally suffer little or no performance loss compared to the STL method.

5.2 Sine waves regression

The next experiment uses ten tasks, and varies the number of training data points per task. For $i = 1, \ldots, 10$ the targets for the i^{th} task were generated with the following formula:

$$y = a_i + b_i \sin(x^{p_i} + d_i) + 0.2\epsilon \tag{5.28}$$

where:

$a_i \sim \mathcal{N}(0, 0.1^2)$, $b_i, p_i \sim \mathcal{N}(1, 0.1^2)$, $d_i \sim \mathcal{N}(0.5, 1)$, $\epsilon \sim \mathcal{N}(0, 1)$

x was chosen from a uniform distribution over $[0, 7]$

Figure 5.12 shows the ten functions that were used to generate the training and test data, with no noise added.

We will begin with ten points per task, giving a total of one hundred training points. One such training set is shown in figure 5.13, together with the corresponding function that was used to generate it.

The network architecture is shown in figure 5.14. Each hidden layer, represented as an ellipse, had six units and used the tanh activation function. The single output unit was just the weighted sum of its inputs. The prior specification is indicated above the weights, as was done before, but now it applies to a group of weights. The x sign in front of the prior specification for the hidden to output weights means that the priors are rescaled based on the number of hidden units (in the way mentioned in section 4.1). As in all examples in this chapter, the common layer does not contribute to the output.

The STL and NDT networks again are the same as the common part of the MTL network. Ten sets of training data for each of the ten tasks were generated, each having

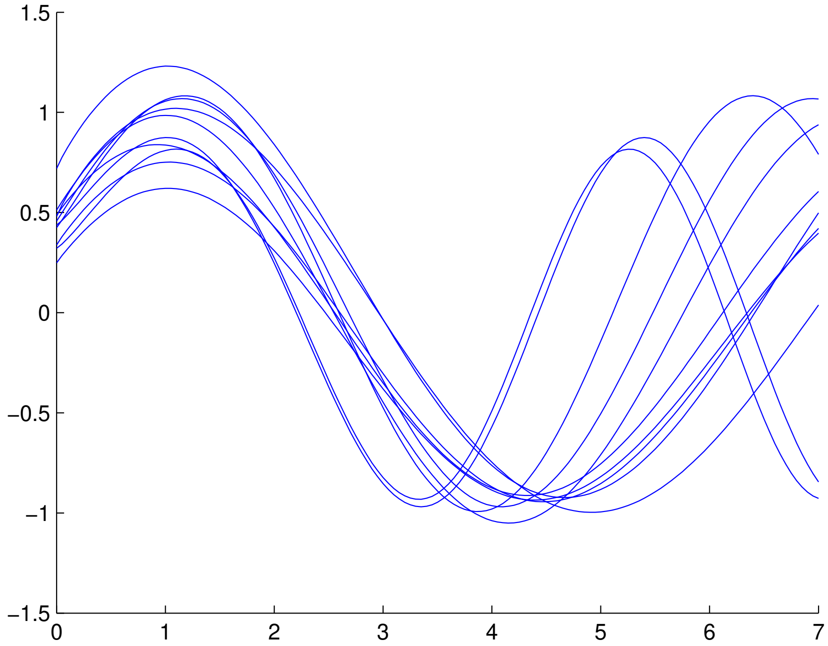


Figure 5.12: Generator functions for the sine waves learning data.

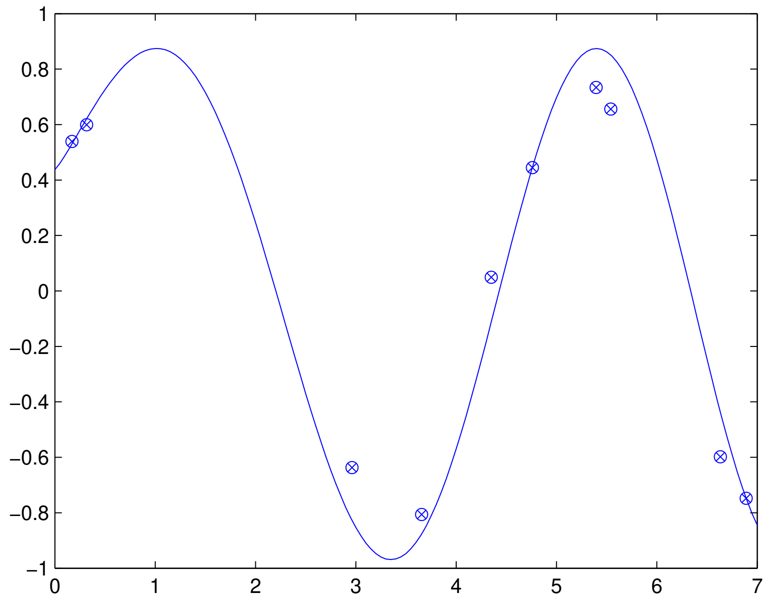


Figure 5.13: An example of a sine wave training data for one of the tasks.

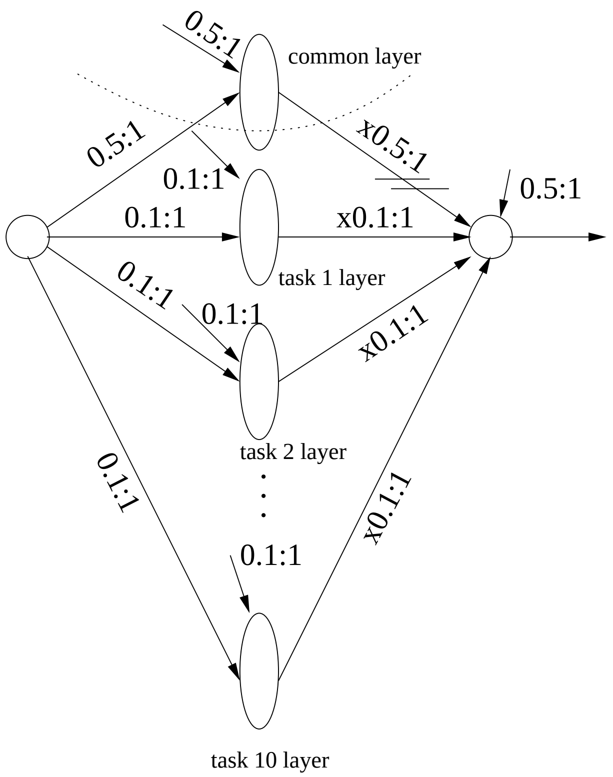


Figure 5.14: The MTL model for the sine waves learning problem

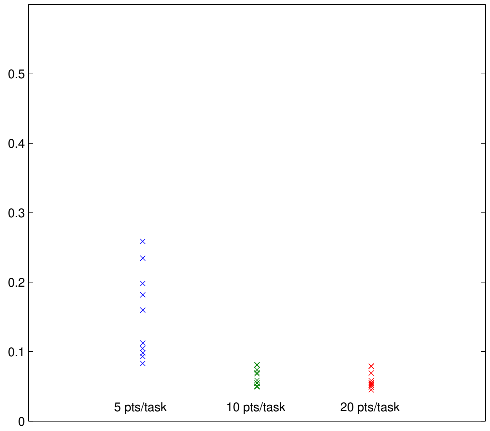
ten training data points per task. The performance of different models with different numbers of training points per task is presented in figure 5.15. Since we added Gaussian noise with mean zero and standard deviation 0.2, average squared error of 0.04 is the best we can expect. Using different priors didn't change the performance much, as long as the weights corresponding to the common layer weren't forced to be very small.

The results with ten points per task definitely show that our MTL scheme managed to use the knowledge contained in other training tasks. It consistently achieved lower errors in all trials for all tasks by a large margin.

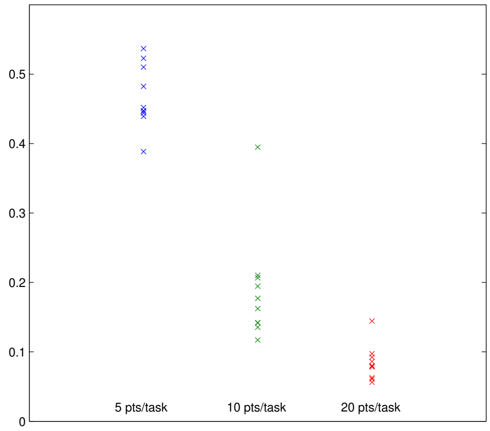
Using only five points per task produced an even bigger difference between the MTL and STL networks. The total average squared error for the NDT model of 0.242 was also worse than in the ten points per task case, but it didn't rise so steeply as the STL version. The gap between the MTL's and NDT's performance continued to reduce with further reduction of the number of points per task. With only two points per task, they were very similar, while the STL version became hopeless. Even if I used two points for a specific task, and 9×10 points for the other nine tasks, and then evaluated the network only on that specific task, the MTL and NDT networks were very similar.

On the other hand, with twenty points per task, the STL version started to catch up with the MTL version. The overall average squared errors were 0.059 for the MTL model and 0.084 for the STL model. The NDT model performed almost the same as with ten points per task. With forty points per task, the MTL version had a slight advantage over the STL version, and with even more points, they performed very similarly.

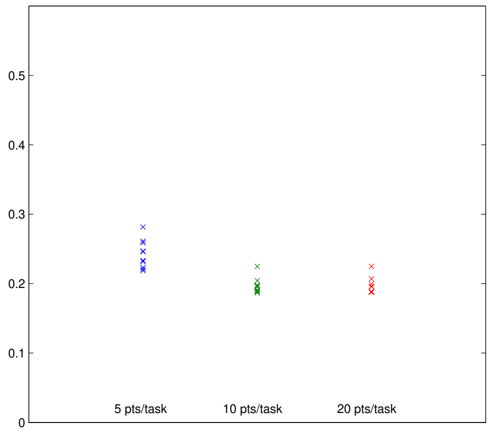
As was the case with the linear data, the biggest difference between MTL and the other two methods was when the STL and NDT methods performed at about the same level. The smaller the number of training points, the bigger the advantage of the MTL method over the STL method. However, even with abundant training data, the STL network never significantly outperformed the MTL network.



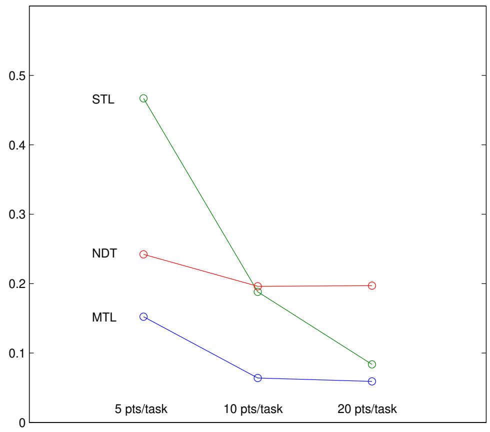
(a) MTL



(b) STL



(c) NDT



(d) The averages over all 10 trials

Figure 5.15: Average squared error on ten-task sine wave regression. The three figures show the distribution of the error averaged over 10 training trials for individual methods with different number of training points per task. The last figure shows the averages over all tasks and trials for all the methods. The test data set had a total of 10,000 data points.

5.3 Approximately concentric ellipses classification

The next example is a classification problem with two classes. A total of eight tasks were generated in the following way:

- the training data for the members of class A for the i^{th} task were chosen randomly from an elliptical ring whose center was at $(0,0)$, the outer semimajor axis was a_i, the outer semiminor axis was a'_i, the inner semimajor axis was b_i, and the inner semiminor axis was b'_i. The whole ring was then rotated by the angle ϕ_i. Gaussian noise with standard deviation 0.25 was then added. The parameters had the following distributions:

  $a_i \sim \mathcal{N}(5,1)$, $a'_i \sim \mathcal{N}(2,0.6^2)$, $b_i \sim \mathcal{N}(3,0.5^2)$, $b'_i \sim \mathcal{N}(1.5,0.3^2)$

  ϕ_i was chosen from a uniform distribution over $[0,\pi/2]$

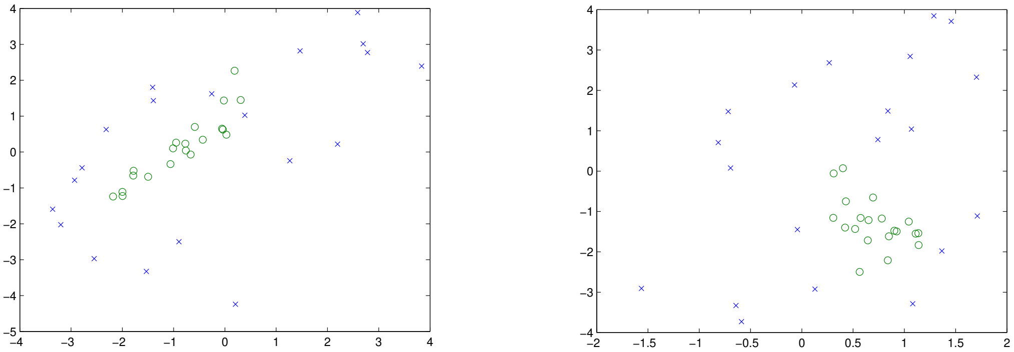
- the training data for the members of class B for the i^{th} task were chosen randomly from an ellipse with a semimajor axis c_i, semiminor axis c'_i, and rotated by the angle ψ_i equal to ϕ_i plus Gaussian noise of standard deviation 0.05. The center of the ellipse was at (m_x, m_y). Gaussian noise with the same distribution as above was added here too. The distributions for the parameters were:

  $c_i \sim \mathcal{N}(2,0.6^2)$

  $c'_i \sim \mathcal{N}(\frac{2(b_i-b'_i)}{a_i-a'_i}, 0.36^2)$

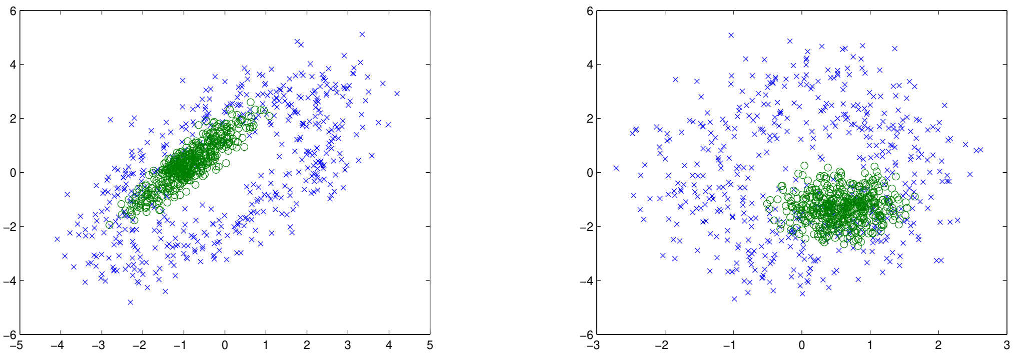
  m_x, m_y were chosen from a uniform distribution over $[-1.5,1.5]$

The number of points per tasks was varied, and, again, for each situation, ten training data sets were generated. Examples for two training data sets with twenty points per task are shown in figure 5.16, the corresponding test data sets are shown in figure 5.17. All the training data, regardless of tasks, is shown in figure 5.18 – this is the data that the NDT version works with.



(a) (b)

Figure 5.16: Two training sets for the approximately concentric ellipses classification problem.



(a) (b)

Figure 5.17: Two test data sets for the approximately concentric ellipses classification problem corresponding to the tasks shown in figure 5.16.

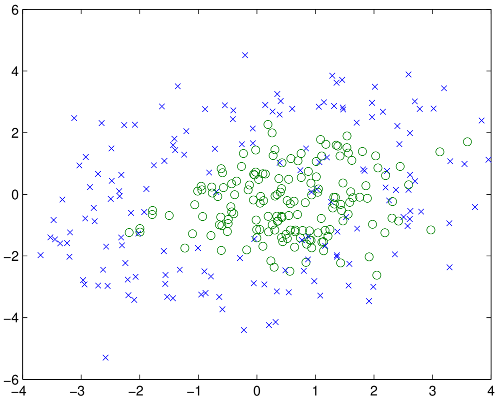


Figure 5.18: The complete training data for a single trial.

The MTL network architecture is shown in figure 5.19 and is very similar to that used for the sine wave regression problem. However, the output here is interpreted as the input to the logistic function $f(x) = 1/(1 + \exp(-x))$ which gives the probability that the input belongs to one of the classes. The hidden layers used the tanh activation function, and consisted of 16 units each. The common layer did not contribute to the output.

The error rates for varying numbers of points per task, averaged over all of the ten trials and eight tasks, are presented in figure 5.20.

I also trained a network with the same prior specification as above, and again, with only one task specific layer active for a particular data point, but without any overlaying. I.e. the common layer was shared among the tasks and it contributed to the output, but the weights for each task's layer were not overlaid. The achieved error rate was 12.0%, which is better than the NDT (15.7%) and the STL networks (14.2%), but worse than the MTL (9.3%) method. This was the generally observed behaviour; it never outperformed the MTL method, although it was never far off performance-wise.

When trained with forty or more points, the STL and the MTL approaches performed very close to each other. Even with five hundred data points per task, the error rate didn't go below 7.0%. So the error rate of 9.3% with only twenty training data points per task

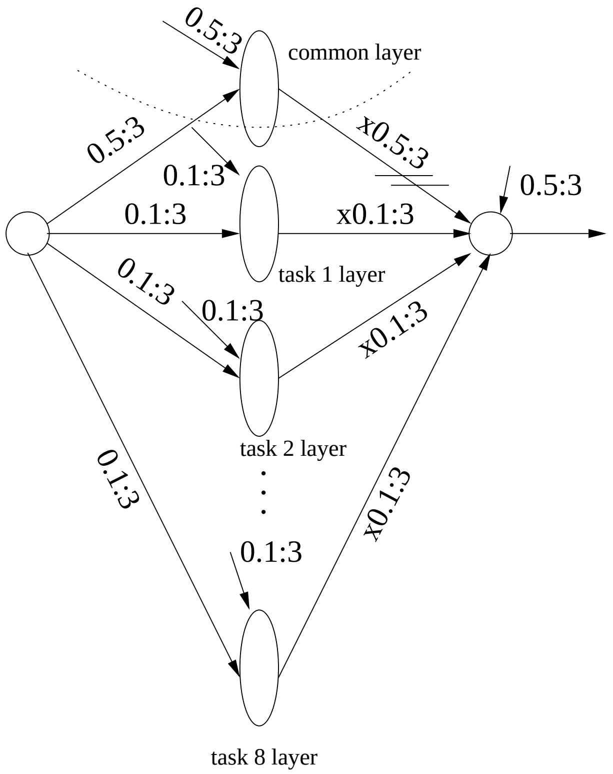


Figure 5.19: The MTL model for the approximately concentric ellipses classification problem.

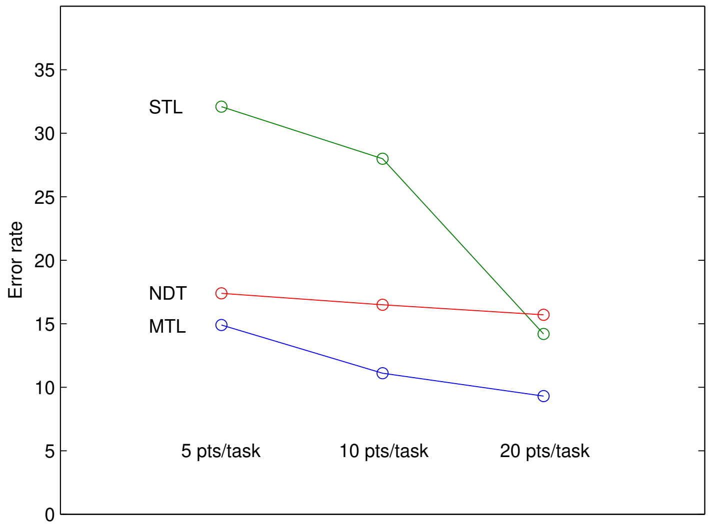


Figure 5.20: Error rates on the eight-task classification problem.

is not far from the lowest error rate that could be expected, given the inherent noise in the data.

Again, the MTL's advantage over the STL rose as I reduced the number of training data points per task as can be seen in figure 5.20. The NDT version had the smallest rise in error rate, from 15.7% to 16.5%, quite expected as 80 points per trial is still enough for a low dimensional, not overly demanding training data set. The MTL also suffered a small rise in error rate, from 9.3% to 11.1%, but it was nowhere near as big as the STL, whose error rate rose from 14.2% to 28.0%.

With only five points per task, the difference became even a bit bigger. The MTL still achieved a low error rate of 14.9%, although it came closer to the NDT version (17.4%). The STL version was much worse. Playing with the priors has rather small effects. Varying the inverse square root of the mean of the precision between 0.01 and 1, and shape parameter between 1 and 30, the error rate of the MTL with ten points per task was between 11.0% and 13.6%.

Chapter 6

Experiments with MTL for BCI

This chapter explores the feasibility of using multitask learning algorithms in BCI problems. The EEG data is often collected from several subjects, which seems like a good scenario in which to use multitask learning. On the other hand, different people have different cranial structure, the strength and the form of the electrical signals also vary between different persons, and the data is usually heavily preprocessed. Therefore, it is not clear that the data from different subjects is sufficiently related for the MTL approach to work well.

The first data set used here was recorded by Keirn and Aunon [36] and was also used by Anderson and Sijerčić [5]. Six EEG channels: C3, C4, P3, P4, O1 and O2, as defined by the International 10/20 system of electrode placement, were recorded at a sampling rate of 250 Hz. A separate channel was used for the detection of eye blinks.

The experiment consisted of four subjects performing five tasks, which were chosen to have large hemispheric asymmetry in brain activity ([54]). The five tasks are:

- the baseline task, where the subject did nothing
- the letter task, for which the subjects composed a letter without vocalising it
- the math task, which consisted of multiplying integers without vocalising or making

any other movement

- the visual counting task, for which the subjects imagined a blackboard, and were asked to visualise numbers being written sequentially

- the geometric figure rotation task, for which the subjects were to visualise a particular three-dimensional object rotating about an axis

Each trial lasted ten seconds, giving 2500 samples per channel. Each trial was divided into half-second segments that overlapped. Anderson and Sijerčić discarded the segments containing the eye blinks. I used all the segments. The eye channel was not used in classification.

Based on the results of Keirn and Aunon, as well as of Anderson and Sijerčić, I used autoregressive (AR) coefficients of order six to represent the EEG signals. Let $a_i, i = 1, \ldots, 6$ be the six AR coefficients. The prediction for the value of the EEG signal $x(t)$ at time t is given by:

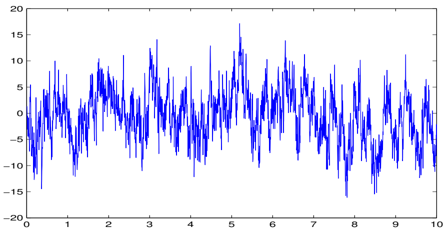
$$x(t) = \sum_{i=1}^{6} a_i x(t-i) \tag{6.1}$$

The Burg method[1] was used to estimate the coefficients that minimise the squared error of the above prediction. Since we had six channels, the complete input vector was of size 36.

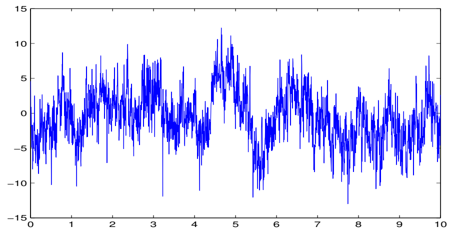
Figure 6.1 shows examples of the EEG signals for the C3 channel for all four subjects.

The MTL network used is shown in figure 6.2. Five classes are represented with five output units that use identity activation function, but which are used in a "softmax" model: the probability of the class i is $\frac{\exp(o_i)}{\sum_j \exp(o_j)}$ where o_j is the value of the j^{th} output. The common weights from the hidden layer to the j^{th} output overlay the task specific weights from its hidden layer to that same output in the same manner as before. The input to hidden weights are also being overlaid. The common layer does not contribute

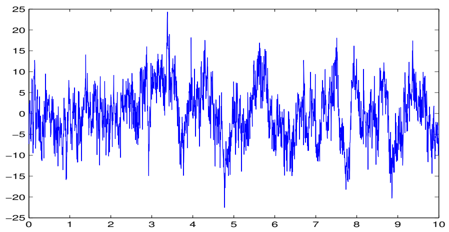
[1]as implemented by the MATLAB's function *arburg*



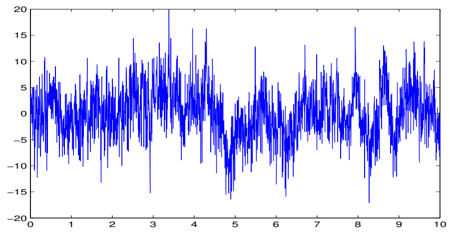
(a) Subject 1, baseline



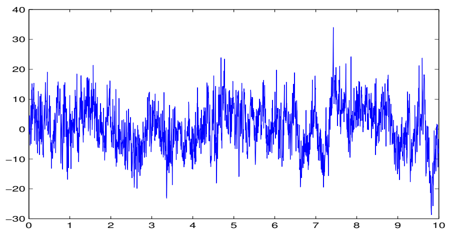
(b) Subject 1, geometric figure rotation



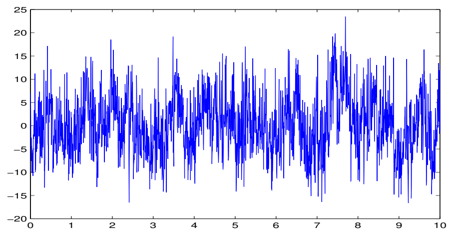
(c) Subject 2, baseline



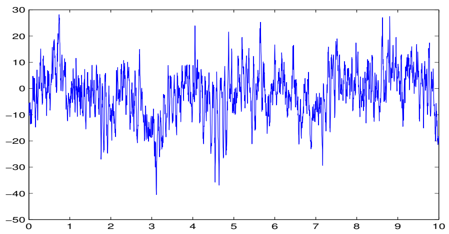
(d) Subject 2, geometric figure rotation



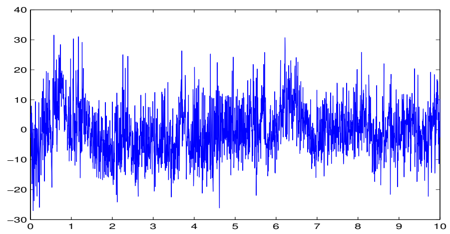
(e) Subject 3, baseline



(f) Subject 3, geometric figure rotation



(g) Subject 4, baseline



(h) Subject 4, geometric figure rotation

Figure 6.1: EEG signal for the C3 channel. For each subject on the left is the EEG for the baseline task, and on the right is the EEG for the geometric figure rotation task.

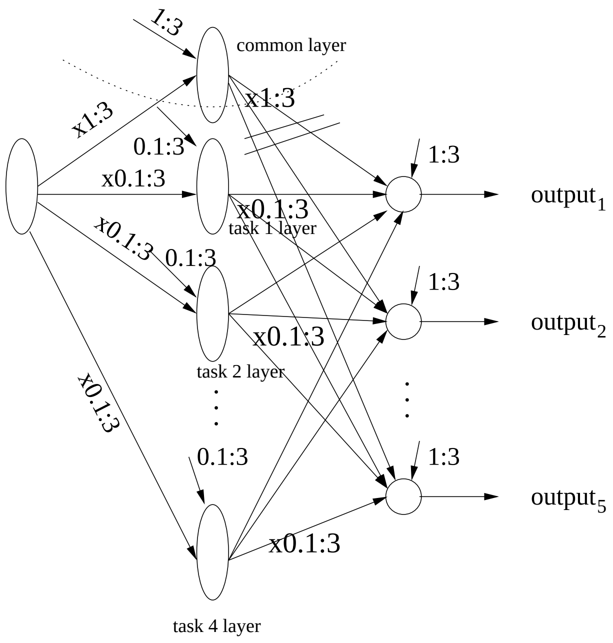


Figure 6.2: The MTL network for the four-subject EEG classification problem.

	Subject 1	Subject 2	Subject 3	Subject 4
STL	65.2%	44.0%	48.2%	55.7%
MTL	34.8%	22.7%	27.3%	25.6%
MTL2	63.1%	44.1%	49.0%	48.5%

Table 6.1: Accuracy rates for the four-subject EEG problem

to the output on its own. The single prior specification in the figure over the hidden to outputs weights means that all the groups of weights (five of them, for each of the five outputs) have the same indicated prior specification. All the hidden layers have 20 units, and use the tanh activation function. The STL network is identical to the common part of the MTL network.

The STL network for the four subjects gave the results in table 6.1.

The MTL network, as specified above, performed very poorly; it was below 40% for subject 1, and below 30% for all the other subjects. Given that with blind guessing, we would expect the accuracy rate of 20%, it wasn't doing a good job. Setting all the prior specifications for the input to hidden and hidden to output weights to be x0.1:0.5, and the priors for the bias weights to 0.1:0.5 does help tremendously. However, the MTL method still fell a bit short of the STL. The results for this setup is given in the MTL2 row of the table 6.1

The inspection of the network weights for MTL2 shows that the task specific weights are as big as the weights in the common part. Forcing the common part to have small weights would give the performance of STL, but such an MTL setup is, in essence, nothing more than training four STL networks independently, but at the same time. Still, we can notice that the difference between the MTL2 and STL networks given in table 6.1 isn't very big and, given that the test data set wasn't big (90 data points), might not even be significant. Using less training data points, the situation for which MTL is designed for, didn't put MTL over STL either, somewhat confirming that if there are any similarities

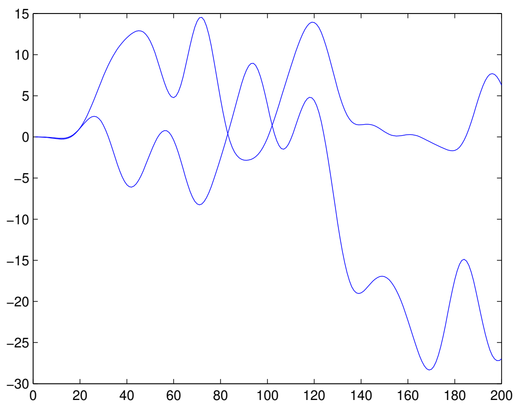


Figure 6.3: Filtered C3 channel for the left and the right hand movement

between subjects, they are not present after we extract the AR coefficients from the raw EEG signal.

The next data set was used at the NIPS 2001 BCI Workshop competition and was provided by Allen Osman. A total of nine subjects completed ninety trials. Each trial lasted for six seconds, during which a subject was instructed to imagine either a left hand or a right hand movement. The cues were given on a screen at highly predictable times. EEG was recorded from 59 electrodes positioned on the skull according to the International 10/20 system and referenced to the left mastoid. The signals were sampled at 100 Hz.

In the preprocessing phase I manually selected four channels that exhibited the highest oscillation of the CSD (current source density) power (see [3]). These were the left and the right motor cortex (sites C3 and C4 on figure 3.1), and the left and the right posterior parietal cortex (sites PO7 and PO8).

The AR coefficients again didn't seem to be the right choice, so I took a look at the traditional MRP and ERD features. The raw EEG signal for the above four channels was filtered with a Hamming-window based, linear-phase FIR (finite impulse response) low-pass filter of order 35, and a cut-off frequency of 4Hz.

Figure 6.3 shows plots of the C3 channel for the left and the right hand movement, after filtering. Only two seconds of the trial were used, so the above figure has 200 points. The start was 0.5 seconds before the instruction to move the left or the right hand appeared on the screen, and the end was 1.5 seconds after that event, and 0.25 seconds after the time the subjects were supposed to actually imagine the movement. Two time points were selected: for the C3 and C4 channels they were at 0.5 and 2 seconds in the trial (the selected interval of the original trial), and for the PO7 and PO8 channels they were at 0.6 seconds and 1.65 seconds in the trial. The differences between the values at the corresponding time points were computed for each of the four channels. The obtained value for the C4 channel was subtracted from the value for the C3 channel, and the value for the PO8 channel was subtracted from the value for the PO7 channel to obtain two input features. The third one was obtained by computing the difference between the values of the PO8 channel at time points 0.8 and 0.95 seconds in the selected 2-second trial. Additionally, the PSD (power spectral density) was estimated via Burg's method for the difference between the raw C3 and C4 channel in the last 0.25 seconds. The logarithm of the power at 14 Hz was taken as an input feature. The same procedure was carried out for the [0.75,1.75] interval for the raw PO7 channel, and the log of the power at 10.1 Hz was taken as the final feature. The total size of the input vector was five.

The MTL network used is shown in figure 6.4. Again, as in the concentric ellipses learning problem, we have two classes, and a single output is used as an argument for the logistic function, which gives the probability that the subject performed an imagined right hand movement. The hidden layers used tanh as the activation function and had 20 hidden units. The prior specification is indicated above the corresponding group of weights in the same fashion as before.

The STL network architecture is, again, identical to the common part of the MTL network. The results are shown in table 6.2. The average performance over all nine subjects was the same, although it wasn't the same for each individual subject. For some

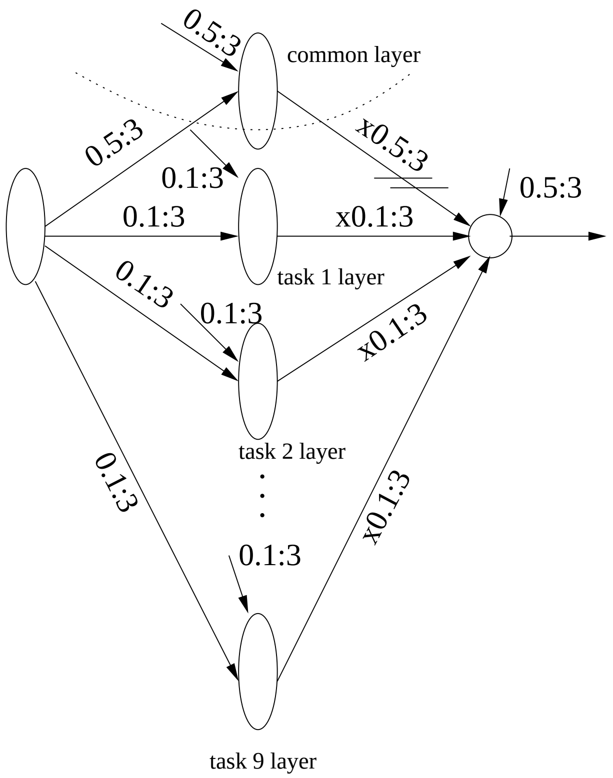


Figure 6.4: The MTL network for the nine-subject EEG classification

Subject	STL	MTL
1	70.0%	70.0%
2	56.7%	60.0%
3	65.6%	65.6%
4	56.7%	55.6%
5	44.5%	52.2%
6	68.9%	66.7%
7	64.4%	62.2%
8	63.3%	58.9%
9	71.1%	70.0%
Avg.	62.4%	62.4%

Table 6.2: Accuracy rates for the nine-subject EEG problem

subjects, the results were bad. The accuracy rate for the fifth subject was even below 50%, which is the expected accuracy rate if we guess blindly.

I selected five subjects for which the STL method perfomed the best: subjects number 1, 3, 6, 7 and 9, and retrained the MTL method. The network used the same priors as above, but now it had six hidden layers instead of ten: one common plus five task-specific layers. The results for the five selected subjects are summarised in table 6.3.

We can see that the MTL method comes out behind the STL method. Given the experience with the AR coefficients, and the fact that the priors above assume the tasks to be significantly related, the new MTL model in figure 6.5 was trained, and its results are given in the "MTL2" column of the table 6.3.

The MTL method now manages to outperform the STL, but the difference isn't very convincing. One of the reasons can be seen in figure 6.6 that shows the distribution of one of the input features for subject number 1. The x-axis is the index of the data point and the y-axis is the value of the input. There are 45 data points per class, and we

	90 points per task			10 points per task	
Subject	STL	MTL	MTL2	STL	MTL2
1	70.0%	70.0%	73.3%	67.8%	71.1%
3	65.6%	66.7%	65.6%	60.0%	60.0%
6	68.9%	66.7%	67.8%	61.1%	64.4%
7	64.4%	64.4%	67.8%	60.0%	62.2%
9	71.1%	70.0%	70.0%	62.2%	68.9%
Avg.	68.0%	67.3%	68.9%	62.2%	65.3%

Table 6.3: Accuracy rates for the five-subject EEG problem

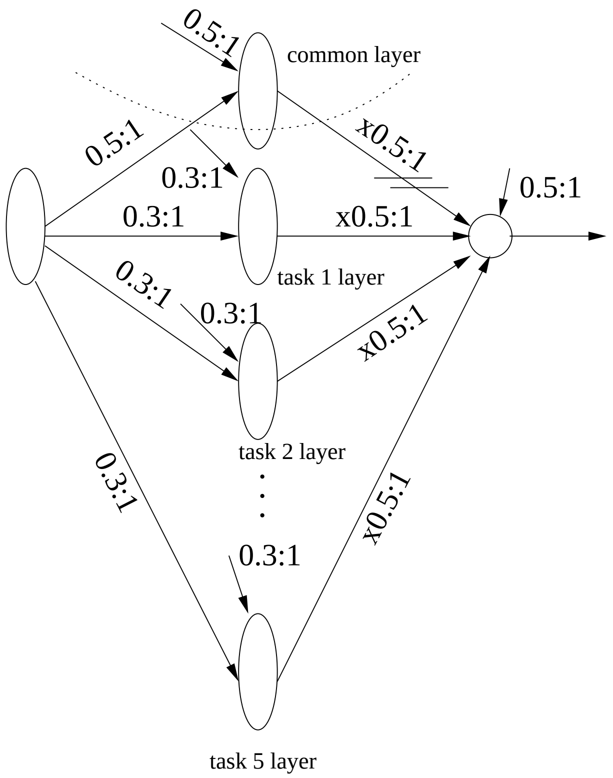


Figure 6.5: The new MTL network for the five-subject EEG classification

can see that a single linear boundary would do a decent job. In this light, 90 training data points for each task is a lot, and in situations with a lot of data, STL is generally better than or at least as good as MTL. As was reported in chapter 2, linear methods for

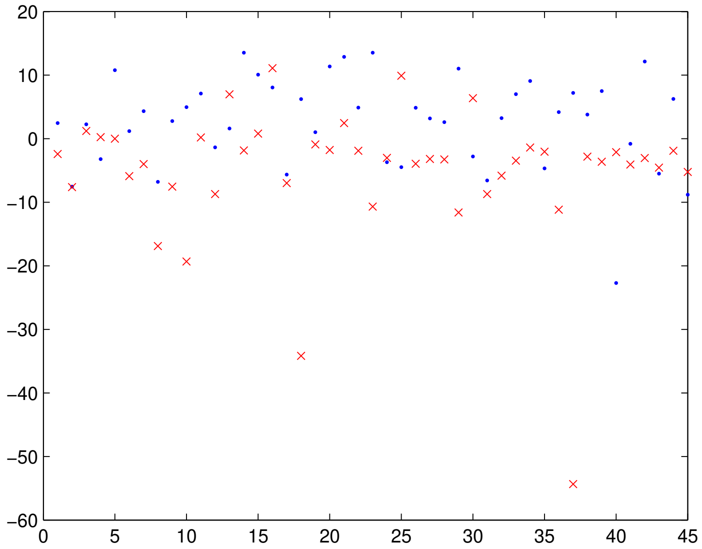


Figure 6.6: Distribution of one input feature for the two hand movements. Each class has 45 points

BCI have generally been as successful as nonlinear methods. One of the debates at the 2^{nd} International BCI Workshop was titled "Linear versus Non-linear Methods in BCI Research". This linearity also suggests that we could use few data points and still achieve acceptable performance. This was indeed true. STL and the second (more successful) version of MTL were retrained with only 10 training data points per task, five per class. The results are also given in table 6.3.

MTL does manage to achieve a meaningful advantage over STL, and does so pretty consistently. The test data set had a total of 90 points.

One additional thing to note is that the input features were selected by hand by looking at a few trials for one subject (the first subject in the table). Specific frequencies where the power was taken, as well as the time points for MRP features, were tailored to that subject. This is probably the biggest reason that the accuracy rate was the highest for him. However, the selected input features also worked well for other subjects too. Together with the limited success of the MTL with few points, this suggests that the relatedness of EEG signals gets reduced in the preprocessing phase, or even completely lost, as in the case with AR coefficients as input features.

Chapter 7

Conclusions

7.1 Summary

The human ability to generalise complex concepts with very few examples is in contrast with the most common approaches to machine learning. Recently, a few approaches that do tackle this aspect of learning have emerged. One such approach is multitask learning (MTL). The main idea of MTL is to use data from other tasks to help learn the task of interest more accurately and/or faster. It does that by training all tasks in parallel, while using a shared representation.

This thesis introduces a new MTL model that combines random regression coefficients models with powerful Bayesian neural networks. The role of the shared representation is played by a separate hidden layer which "overlays" task-specific layers. Priors for the network parameters play an important role here. They allow us to put an emphasis on the common layer, which is useful if the tasks are substantially related. In this case, the common layer does the bulk of the job, while the task specific hidden layers reflect the fine differences between the tasks.

The model was tested on artificially generated data sets, as well as on data sets from Brain Computer Interface problems. The focus was on generalisation accuracy of the learning, rather than on its speed. The experiments show that the new model succeeds in its goal; it achieves better performance than the single task learning models in situations with small training data sets. Moreover, even with abundant training data, the MTL model wasn't significantly outperformed by the STL method, provided that the priors were given some flexibility.

The application of the MTL method to the BCI data wasn't totally conclusive. Experiments with AR coefficients derived from the EEG time series didn't suggest that the EEG signals from different human subjects had much in common. However, with vague priors, the MTL method performed similarly to the STL method. With input features taken from the frequency domain, the MTL managed to acquire a limited success compared to the STL method, when the training data was scarce. This is in accord with the reports of researchers in the BCI field, who have discovered that certain actions can be detected in specific frequency bands of the corresponding EEG signal. A disadvantage of the input features taken from the frequency domain for MTL, is that linear classifiers are often sufficient. This implies that relatively small training data sets are required for decent performance, thus limiting the potential of the MTL approach. It may also suggest information loss during the preprocessing phase.

7.2 Future work

Since the majority of machine learning approaches are oriented toward single task learning, most of the available data sets are tailored to use with STL models. More data suitable for the MTL approach would allow a more accurate estimate of MTL's capabilities.

Brain computer interfaces have been a red hot field in the last few years. New preprocessing procedures may be developed that would optimise the trade-off between the input space dimensionality and the information loss, and might also help MTL. As mentioned in the previous section, current preprocessing techniques heavily reduce the number of input features, often producing features with a linear relationship to the class, which is not a very good situation for multitask learning. However, it may be that this is an inherent characteristic of the EEG signal. One could also try to find a way to have an MTL method that would select features like frequency bands a bit differently for each subject – in a way, pushing multitask learning into the preprocessing phase as well.

It would also be interesting to compare the non-linear neural network method to random regression coefficient models, and to other statistical models that work with clustered data, most of which assume that the data model is linear.

Bibliography

[1] Y.S. Abu-Mostafa. Learning from hints in neural networks. *Journal of Complexity*, 6(2):192–198, 1990.

[2] W.-K. Ahn and W.F. Brewer. Psychological studies of explanation-based learning. In G. DeJong, editor, *Investigating Explanation-Based Learning*, Boston/Dordrecht/London, 1993. Kluwer Academic Publishers.

[3] O. Allen and A. Robert. Time-course of cortical activition during overt and imagined movements. In *Cognitive Neuroscience Annual Meeting*, New York, March 2001.

[4] H. C. Andersen. Molecular dynamics simulations at constant pressure and/or temperature. *Journal of Chemical Physics*, 72:2384–2393, 1980.

[5] C. Anderson and Z. Sijercic. Classification of EEG signals from four subjects during five mental tasks, 1996.

[6] C. Babiloni, F. Carducci, F. Cincotti, P. Rossini, C. Neuper, G. Pfurtscheller, and F. Babiloni. Human movement-related potentials vs desynchronization of EEG alpha rhythm: A high-resolution EEG study. *NeuroImage*, 10(6):658–665, 1999.

[7] B. Bakker and T. Heskes. Task clustering and gating for Bayesian multitask learning. *Journal of Machine Learning Research*, 4(1):83–99, 2004.

[8] J. Baxter. Learning internal representations. In *COLT: Proceedings of the Workshop on Computational Learning Theory, Morgan Kaufmann Publishers*, 1995.

[9] J. Baxter. A Bayesian/information theoretic model of learning to learn via multiple task sampling. *Machine Learning*, 28(1):7–39, 1997.

[10] J. Baxter. Theoretical models of learning to learn. In T. Mitchell and S. Thrun, editors, *Learning to Learn*. Kluwer, Boston, 1997.

[11] J. Baxter. A model of inductive bias learning. *Journal of Artificial Intelligence Research*, 12:149–198, 2000.

[12] S. Ben-David and R. Schuller. Exploiting task relatedness for multiple task learning. *COLT 2003*, 2003.

[13] N. Birbaumer, N. Ghanayim, T. Hinterberger, I. Iversen, B. Kotchoubey, A. Kübler, J. Perelmouter, E. Taub, and H. Flor. A spelling device for the paralysed. *Nature*, 398(6725):297–298, 1999.

[14] B. Blankertz, G. Curio, and K. R. Müller. Classifying single trial EEG: Towards brain computer interfacing. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, Cambridge, MA, 2002. MIT Press.

[15] W.L. Buntine and A.S. Weigend. Bayesian back-propagation. *Complex systems*, 5:605–643, 1991.

[16] R. Caruana. Multitask learning: A knowledge-based source of inductive bias. In *International Conference on Machine Learning*, pages 41–48, 1993.

[17] R. Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997.

[18] R. Caruana. *Multitask Learning*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburg, PA, 1997.

[19] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13:21–27, 1967.

[20] E. Donchin, K. Spencer, and R. Wijesinghe. The mental prosthesis: Assessing the speed of a P300-Based Brain-Computer Interface. *IEEE Transactions on Rehabilitation Engineering*, 8(2):174–179, 2000.

[21] G. Dornhege, B. Blankertz, G. Curio, and K. Mller. Combining features for BCI, 2003.

[22] S. Duane, A. D. Kennedy, B.J. Pendleton, and D. Roweth. Hybrid Monte Carlo. *Physics Leters*, 195:216–222, 1987.

[23] S. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2. Morgan Kaufmann, San Mateo, CA, 1990.

[24] L.A. Farwell and E. Donchin. Talking off the top of your head: toward a mental prosthesis utilizing event-related brain potentials. *Electroenceph. Clin. Neurophysiol.*, pages 510–523, 1988.

[25] E. Fix and J.L. Hodges. Discriminatory analysis, non-parametric discrimination, USAF School of Aviation Medicine, Randolph Field, TX, Project 21-49-004, Report 4, Contract AF41(128)-3, 1951.

[26] L.-M. Fu. Integration of neural heuristics into knowledge-based inference. In *Connection Science*, volume 1, pages 325–339. 1992.

[27] C. Genovese. Statistical inference in functional magnetic resonance imaging, 1997.

[28] D. Gentner. The mechanisms of analogical learning. In S. Vosniadou and A. Ortony, editors, *Similarity and Analogical Reasoning*, pages 199–241. Cambridge University Press, New York, 1989.

[29] J. Ghosn and Y. Bengio. Multi-task learning for stock selection. In *Advances in Neural Information Processing Systems 9*, 1997.

[30] B. Graimann, J. Huggins, S. Levine, and G. Pfurtscheller. Toward a direct brain interface based on human subdural recordings and wavelet-packet analysis. *IEEE Transactions on Biomedical Engineering*, 51(6):954–962, 2004.

[31] N. Hartvig. Parametric modelling of functional magnetic resonance imaging data, 2000.

[32] D. Haussler. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, 36(2):177–221, 1988.

[33] T. Heskes. Emiprical bayes for learning to learn. In P. Langley, editor, *Proceedings of ICML*, pages 367–374, San Francisco, CA, 2000. Morgan Kaufmann.

[34] T. Hume and M. Pazzani. Learning sets of related concepts: A shared task model. In *Proceedings of the Eighteenth Annual Conference of the Cognitive Science Society*, 1996.

[35] M.I. Jordan and R.A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. Technical Report AIM-1440, 1993.

[36] Z.A. Keirn and J.I. Aunon. A new mode of communication between man and his surroundings. *IEEE Transactions on Biomedical Engineering*, 37(12):1209–1214, 1990.

[37] P. Kennedy, R. Bakay, M. Moore, K. Adams, and J. Goldwaithe. Direct control of a computer from the human central nervous system. *IEEE Transactions on Rehabilitation Engineering*, 8(2):198–202, 2000.

[38] E. Lalor, S.P. Kelly, C. Finucane, R. Smith, R. Burke, R. B. Reilly, and G. McDarby. Steady-state VEP-based brain computer interface control in an immersive 3-D gaming environment. *Journal of Applied Digital Signal Processing - Trends of Brain Computer Interfaces Special Issue*. In submission.

[39] N.T. Longford. *Random Coefficient Models.* Clarendon Press, Oxford, 1993.

[40] A.S. Luchins and E.H. Luchins. New experimental attempts at preventing mechanization in problem solving. *Journal of general Psychology*, 42:279–297, 1950.

[41] D. MacKay. *Bayesian Methods for Adaptive Models.* PhD thesis, Caltech, 1992.

[42] D. J. C. MacKay. A practical Bayesian framework for backprop networks. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 839–846, 1992.

[43] J. Mahoney and R. Mooney. Combining symbolic and neural learning to revise probabilistic theories. In *Proceedings of the 1992 Machine Learning Workshop on Integrated Learning in Real Domains*, Aberdeen, Scotland, 1992.

[44] G. Martin. The effects of old learning on new in Hopfield and backpropagation networks. Technical Report ACA-HI-019, Microelectronics and Computer Technology Corporation (MCC), 1988.

[45] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.

[46] M. Middendorf, G. McMillan, G. Calhoun, and K. S. Jones. Brain-computer interfaces based on the steady-state visual-evoked response. *Rehabilitation Engineering, IEEE Transactions on [see also IEEE Trans. on Neural Systems and Rehabilitation]*, 8:211–214, 2000.

[47] T. Mitchell, R. Hutchinson, M. Just, S. Newman, R. Niculescu, F. Periera, and X. Wang. Machine learning of fMRI virtual sensors of cognitive states. In *The 16th Annual Conference on Neural Information Processing Systems, Computational Neuroimaging: Foundations, Concepts and Methods Workshop*, 2002.

[48] A.W. Moore, D.J. Hill, and M.P. Johnson. An empirical investigation of brute force to choose features, smoothers and function approximators. In S. Hanson, S. Judd, and T. Petsche, editors, *Computational Learning Theory and Natural Learning Systems*, volume 3. MIT Press, 1992.

[49] G. Murphy and D. Medin. The role of theories in conceptual coherence. *Psychological Review*, 92:289–316, 1985.

[50] G.V. Nakamura. Knowledge-based classification of ill-defined categories. *Memory and Cognition*, 13:377–384, 1985.

[51] R. M. Neal. Probabilistic inference using Markov chain Monte Carlo methods. Technical Report CRG-TR-93-1, University of Toronto, 1993.

[52] R. M. Neal. Priors for infinite networks. Technical Report CRG-TR-94-1, University of Toronto, 1994.

[53] R. M. Neal. *Bayesian Learning for Neural Networks.* Springer-Verlag New York, 1996.

[54] M. Osaka. Peak alpha frequency of EEG during a mental task: Task difficulty and hemispheric differences. *Psychophysiology*, 21:101–105, 1984.

[55] H.D. Patterson and R. Thompson. Recovery of inter-block information when block sizes are unequal. *Biometrika*, 77:549–555, 1971.

[56] M. Pazzani. Influence of prior knowledge on concept acquisition: Experimental and computational results. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 17(3):416–432, 1991.

[57] M. Pazzani, C. Brunk, and G. Silverstein. A knowledge intensive approach to learning relational concepts. In *Proceedings of the Eight International Workshop on Machine Learning*, Ithaca, New York, 1991. Morgan Kaufmann.

[58] W. Penny and S. Roberts. Experiments with an EEG-based computer interface. Technical report, Department of Electrical and Electronic Engineering, Imperial College, London, 1999.

[59] G. Pfurtscheller. EEG event-related desynchronization (ERD) and synchronization (ERS). *Electroencephalography and Clinical Neurophysiology*, 103(1):26–26, 1997.

[60] G. Pfurtscheller, D. Flotzinger, and C. Neuper. Differentiation between finger, toe and tongue movement in man based on 40 hz EEG. *Electroencephalography and Clinical Neurophysiology*, 90(6):456–460, 1994.

[61] G. Pfurtscheller and C. Neuper. Event-related synchronization of mu rhythm in the EEG over the cortical hand area in man. *Neuroscience Letters*, 174(1):93–96, 1994.

[62] L. Pratt. Experiments in the transfer of knowledge between neural networks. In S. Hanson, G. Drastal, and R. Rivest, editors, *Computational Learning Theory and Natural Learning Systems, Constraints and Prospects*, chapter 4.1. MIT Press, 1993.

[63] L. Pratt. *Transferring Previously Learned Back-Propagation Neural Networks to New Learning Tasks*. PhD thesis, Rutgers University, May 1993.

[64] Progress and Perspectives. The BCI competition 2003:.

[65] G. Collins R. Schank and L. Hunter. Transcending inductive category formation in learning. *Behavioral and Brain Sciences*, 9:639–686, 1986.

[66] L. Rendell, R. Seshu, and D. Tcheng. Layered concept-learning and dynamically-variable bias management. In *Proceedings of the IJCAI-87*, pages 308–314, 1987.

[67] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[68] J. Schmidhuber, J. Zhao, and M. Wiering. Shifting inductive bias with success-story algorithm, adaptive levin search, and incremental self-improvement. *Machine Learning*, 28(1):105–130, 1997.

[69] R. Setiono. Extracting rules from neural networks by pruning and hidden-unit splitting. *Neural Comput.*, 9(1):205–225, 1997.

[70] N. Sharkey and A. Sharkey. Adaptive generalization and the transfer of knowledge. In *Proceedings of the Second Irish Neural Network Conference*, Belfast, 1992.

[71] N. Sharkey and A. Sharkey. An analysis of catastrophic interference. *Connection Science*, 7:301–329, 1995.

[72] D. Shepard. A two-dimensional interpolation function for irregularly spaced data. In 23^{rd} *National Conference ACM*, pages 517–523, 1968.

[73] D. Silver and R. Mercer. The parallel transfer of task knowledge using dynamic learning rates based on a measure of relatedness, 1996.

[74] P. Simard, B. Victorri, Y. LeCun, and J. Denken. Tangent prop - a formalism for specifying selected invariances in an adaptive network. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 895–903, San Mateo, CA, 1992. Morgan Kaufmann.

[75] S. Suddarth and A. Holden. A symbolic neural systems and the use of hints for developing complex systems. *International Journal of Man-Machine Studies*, 35:291–311, 1991.

[76] E. Sutter. The brain response interface: communication through visually-induced electrical brain responses. *J. Microcomput. Appl.*, 15(1):31–45, 1992.

[77] I. Taha and J. Ghosh. Symbolic interpretation of artificial neural networks. *Knowledge and Data Engineering*, 11(3):448–463, 1999.

[78] D. Taylor, S. Tillery, and A. Schwartz. Direct cortical control of 3d neuroprosthetic devices. *Science*, 296(5574):1829–1832, 2002.

[79] M. Tenorio and W.-T. Lee. Self organizing neural network for optimum supervised learning. Technical Report TR-EE 89-30, Purdue Univ. School of Elec. Eng., 1989.

[80] S. Thrun. Lifelong learning: A case study. Technical Report CMU-CS-95-208, Carnegie Mellon University, Computer Science Department, Pittsburgh, PA, 1995.

[81] S. Thrun. Is learning the n-th thing any easier than learning the first? In David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 640–646. The MIT Press, 1996.

[82] S. Thrun. Lifelong learning algorithms. In S. Thrun and L.Y. Pratt, editors, *Learning to Learn*, chapter 8. Kluwer Academic Publishers, Boston, MA, 1998.

[83] S. Thrun and J. O'Sullivan. Discovering structure in multiple learning tasks: The TC algorithm. In *International Conference on Machine Learning*, pages 489–497, 1996.

[84] G. Towell and J. Shavlik. Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13:71–101, 1993.

[85] G. Towell and J. Shavlik. Knowledge-based artificial neural networks. *Artificial Intelligence*, 70(1-2):119–165, 1994.

[86] P. Utgoff. Shift of bias for inductive concept learning. In R. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence approach*, volume 2. Morgan Kaufmann, 1986.

[87] L.G. Valiant. A theory of the learnable. *Communications of ACM*, 27(11):1134–1142, 1984.

[88] V.N. Vapnik. *Estimation of Dependencies Based on Empirical Data*. Springer-Verlag, New York, 1982.

[89] J. Vidal. Toward direct brain-computer communication. In L.J. Mullins, editor, *Annual Review of Biophysics and Bioengineering*, volume 2, pages 157–180, Palo Alto, 1973. Annual Reviews, Inc.

[90] A. Waibel, H. Sawai, and K. Shikano. Modularity and scaling in large phonemical neural networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(12):1888–1898, 1989.

[91] G. Webb, J. Wells, and Z. Zheng. An experimental evaluation of integrating machine learning with knowledge acquisition. *Machine Learning*, 35(1):5–23, 1999.

[92] N. Weiskopf, K. Mathiak, S. Bock, F. Scharnowski, R. Veit, W. Grodd, R. Goebel, and N. Birbaumer. Principles of a brain-computer interface (BCI) based on real-time functional magnetic resonance imaging (fMRI). *IEEE Transactions on Biomedical Engineering*, 51(6):966–970, 2004.

[93] J. Wolpaw, D. McFarland, G. Neat, and C. Forneris. An EEG-based brain-computer interface for cursor control. *Electroencephalography and Clinical Neurophysiology*, 78:252–259, 1991.