CSC2515 Fall 2007

Introduction to Machine Learning

Lecture 4: Backpropagation

All lecture slides will be available as .ppt, .ps, & .htm at
www.cs.toronto.edu/~hinton

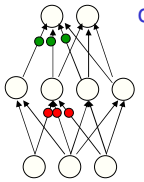
Many of the figures are provided by Chris Bishop
from his textbook: "Pattern Recognition and Machine Learning"

Why we need backpropagation

- Networks without hidden units are very limited in the input-output mappings they can model.
  - More layers of linear units do not help. Its still linear.
  - Fixed output non-linearities are not enough
- We need multiple layers of adaptive non-linear hidden units. This gives us a universal approximator. But how can we train such nets?
  - We need an efficient way of adapting all the weights, not just the last layer. This is hard. Learning the weights going into hidden units is equivalent to learning features.
  - Nobody is telling us directly what hidden units should do.

Learning by perturbing weights

- Randomly perturb one weight and see if it improves performance. If so, save the change.
  - Very inefficient. We need to do multiple forward passes on a representative set of training data just to change one weight.
  - Towards the end of learning, large weight perturbations will nearly always make things worse.
- We could randomly perturb all the weights in parallel and correlate the performance gain with the weight changes.
  - Not any better because we need lots of trials to "see" the effect of changing one weight through the noise created by all the others.



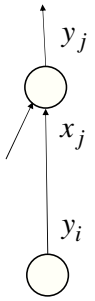
Learning the hidden to output weights is easy. Learning the input to hidden weights is hard.

The idea behind backpropagation

- We don't know what the hidden units ought to do, but we can compute how fast the error changes as we change a hidden activity.
  - Instead of using desired activities to train the hidden units, use error derivatives w.r.t. hidden activities.
  - Each hidden activity can affect many output units and can therefore have many separate effects on the error. These effects must be combined.
  - We can compute error derivatives for all the hidden units efficiently.
  - Once we have the error derivatives for the hidden activities, its easy to get the error derivatives for the weights going into a hidden unit.

A difference in notation

- For networks with multiple hidden layers Bishop uses an explicit extra index to denote the layer.
- The lecture notes use a simpler notation in which the typical index is used to denote the layer implicitly.
  - y is used for the output of a unit in any layer
  - x is the summed input to a unit in any layer
  - The index indicates which layer a unit is in.



Non-linear neurons with smooth derivatives

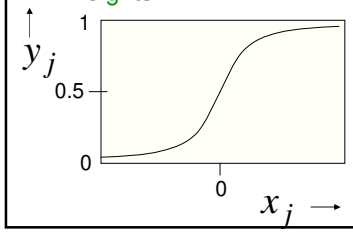
- For backpropagation, we need neurons that have well-behaved derivatives.
  - Typically they use the logistic function
  - The output is a smooth function of the inputs and the weights.

$$x_j = b_j + \sum_i y_i w_{ij}$$

$$y_j = \frac{1}{1+e^{-x_j}}$$

$$\frac{\partial x_j}{\partial w_{ij}} = y_i \qquad \frac{\partial x_j}{\partial y_i} = w_{ij}$$

$$\frac{dy_j}{dx_j} = y_j\,(1-y_j)$$



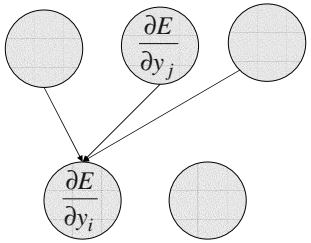
Sketch of the backpropagation algorithm on a single training case

- First convert the discrepancy between each output and its target value into an error derivative.

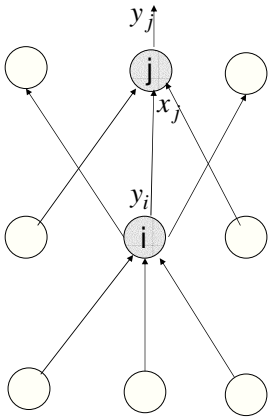
$$E = \sum_j \tfrac{1}{2}(y_j - d_j)^2$$

$$\frac{\partial E}{\partial y_j} = y_j - d_j$$

- Then compute error derivatives in each hidden layer from error derivatives in the layer above.
- Then use error derivatives w.r.t. activities to get error derivatives w.r.t. the weights.



The derivatives



$$\frac{\partial E}{\partial x_j} = \frac{dy_j}{dx_j}\frac{\partial E}{\partial y_j} = y_j\,(1-y_j)\frac{\partial E}{\partial y_j}$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial x_j}{\partial w_{ij}}\frac{\partial E}{\partial x_j} = y_i\frac{\partial E}{\partial x_j}$$

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{dx_j}{dy_i}\frac{\partial E}{\partial x_j} = \sum_j w_{ij}\frac{\partial E}{\partial x_j}$$

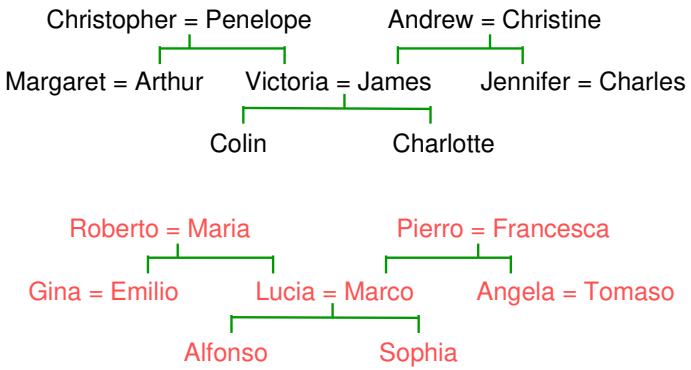
Some Success Stories

- Back-propagation has been used for a large number of practical applications.
  - Recognizing hand-written characters
  - Predicting the future price of stocks
  - Detecting credit card fraud
  - Recognize speech (wreck a nice beach)
  - Predicting the next word in a sentence from the previous words
    - This is essential for good speech recognition.

Overview of the applications in this lecture

- Modeling relational data
  - This toy application shows that the hidden units can learn to represent sensible features that are not at all obvious.
  - It also bridges the gap between relational graphs and feature vectors.
- Learning to predict the next word in a sentence
  - The toy model above can be turned into a useful model for predicting words to help a speech recognizer.
- Reading documents
  - An impressive application that is used to read checks.

An example of relational information



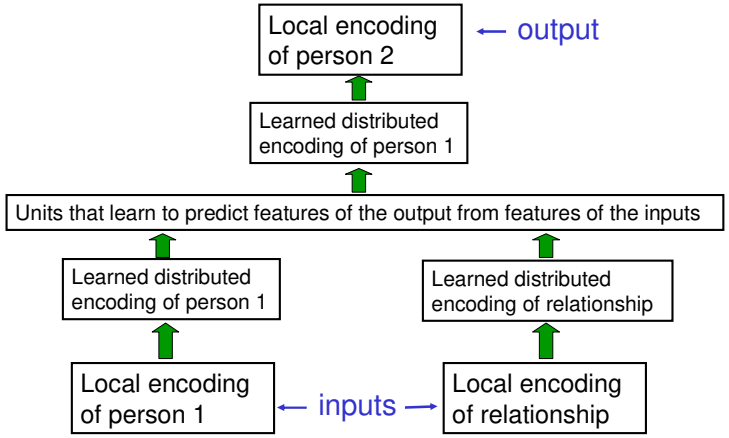
Another way to express the same information

- Make a set of propositions using the 12 relationships:
  - son, daughter, nephew, niece
  - father, mother, uncle, aunt
  - brother, sister, husband, wife
- (colin has-father james)
- (colin has-mother victoria)
- (james has-wife victoria) this follows from the two above
- (charlotte has-brother colin)
- (victoria has-brother arthur)
- (charlotte has-uncle arthur) this follows from the above

A relational learning task

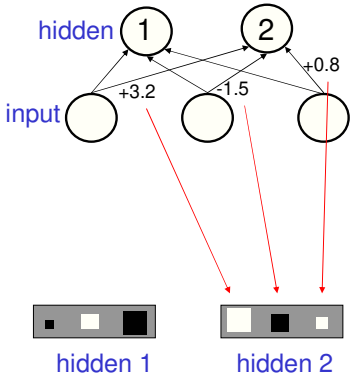
- Given a large set of triples that come from some family trees, figure out the regularities.
  - The obvious way to express the regularities is as symbolic rules
    (x has-mother y) & (y has-husband z) => (x has-father z)
- Finding the symbolic rules involves a difficult search through a very large discrete space of possibilities.
- Can a neural network capture the same knowledge by searching through a continuous space of weights?

The structure of the neural net

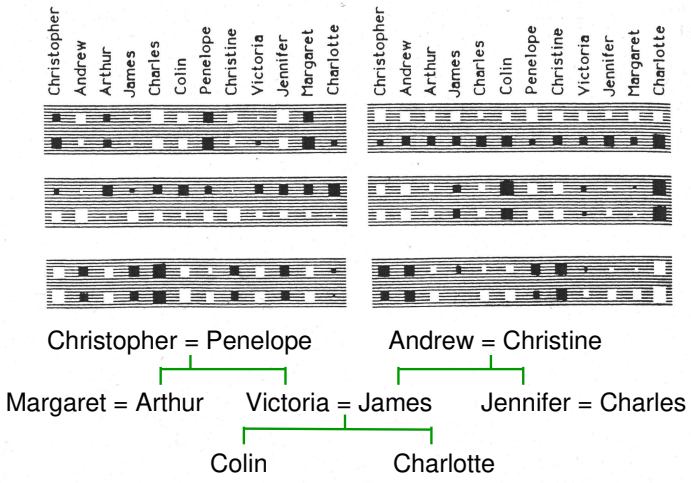


How to show the weights of hidden units

- The obvious method is to show numerical weights on the connections:
  - Try showing 25,000 weights this way!
- Its better to show the weights as black or white blobs in the locations of the neurons that they come from
  - Better use of pixels
  - Easier to see patterns



The features it learned for person 1



What the network learns

- The six hidden units in the bottleneck connected to the input representation of person 1 learn to represent features of people that are useful for predicting the answer.
  - Nationality, generation, branch of the family tree.
- These features are only useful if the other bottlenecks use similar representations and the central layer learns how features predict other features. For example:

  Input person is of generation 3 and

  relationship requires answer to be one generation up

  implies

  Output person is of generation 2

Another way to see that it works

- Train the network on all but 4 of the triples that can be made using the 12 relationships
  - It needs to sweep through the training set many times adjusting the weights slightly each time.
- Then test it on the 4 held-out cases.
  - It gets about 3/4 correct. This is good for a 24-way choice.

Why this is interesting

- There has been a big debate in cognitive science between two rival theories of what it means to know a concept:

  The feature theory: A concept is a set of semantic features.
  - This is good for explaining similarities between concepts
  - Its convenient: a concept is a vector of feature activities.

  The structuralist theory: The meaning of a concept lies in its relationships to other concepts.
  - So conceptual knowledge is best expressed as a relational graph (AI's main objection to perceptrons)
- These theories need not be rivals. A neural net can use semantic features to implement the relational graph.
  - This means that no explicit inference is required to arrive at the intuitively obvious consequences of the facts that have been explicitly learned. The net "intuits" the answer!

A basic problem in speech recognition

- We cannot identify phonemes perfectly in noisy speech
  - The acoustic input is often ambiguous: there are several different words that fit the acoustic signal equally well.
- People use their understanding of the meaning of the utterance to hear the right word.
  - We do this unconsciously
  - We are very good at it
- This means speech recognizers have to know which words are likely to come next and which are not.
  - Can this be done without full understanding?

The standard "trigram" method

- Take a huge amount of text and count the frequencies of all triples of words. Then use these frequencies to make bets on the next word in a b ? (you should try this in google!)

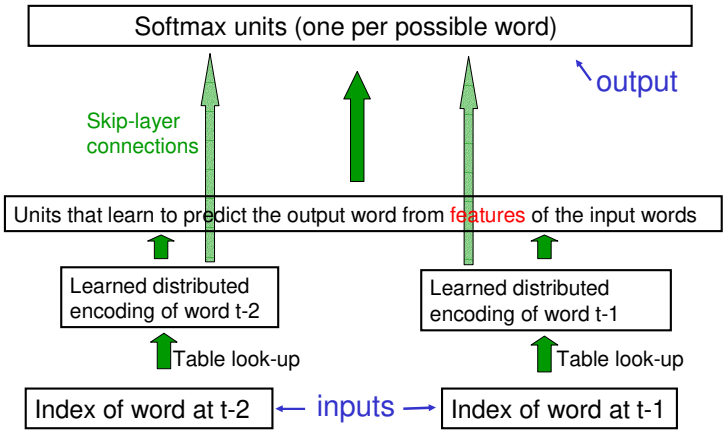
$$\frac{p(w_3 = c \mid w_2 = b, w_1 = a)}{p(w_3 = d \mid w_2 = b, w_1 = a)} = \frac{count(abc)}{count(abd)}$$

- Until very recently this was state-of-the-art.
  - We cannot use a bigger context because there are too many quadgrams
  - We have to "back-off" to digrams when the count for a trigram is zero.
    - The probability is not zero just because we didn't see one.

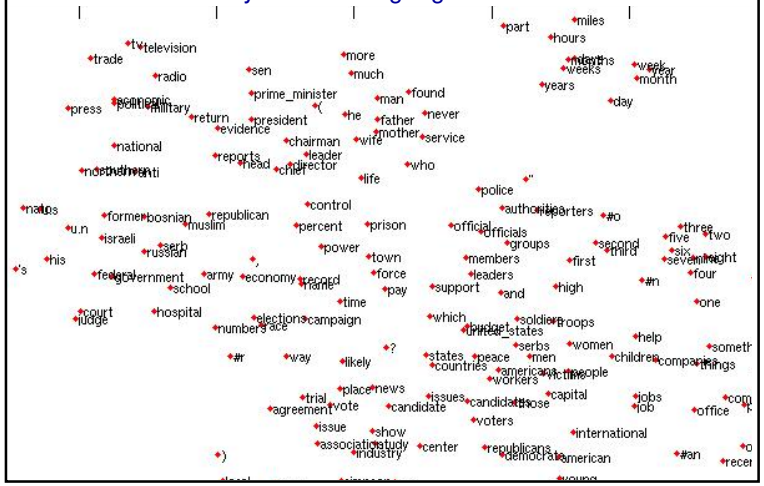
Why the trigram model is silly

- Suppose we have seen the sentence
  "the cat got squashed in the garden on friday"
- This should help us predict words in the sentence
  "the dog got flattened in the yard on monday"
- A trigram model does not understand the similarities between
  - cat/dog squashed/flattened garden/yard friday/monday
- To overcome this limitation, we need to use the features of previous words to predict the features of the next word.
  - Using a feature representation and a learned model of how past features predict future ones, we can use many more words from the past history.

Bengio's neural net for predicting the next word



2-D display of some of the 100-D feature vectors learned by another language model



Applying backpropagation to shape recognition

- People are very good at recognizing shapes
  - It's intrinsically difficult and computers are bad at it
- Some reasons why it is difficult:
  - Segmentation: Real scenes are cluttered.
  - Invariances: We are very good at ignoring all sorts of variations that do not affect the shape.
  - Deformations: Natural shape classes allow variations (faces, letters, chairs, bodies).
  - A huge amount of computation is required.

The invariance problem

- Our perceptual systems are very good at dealing with invariances
  - translation, rotation, scaling
  - deformation, contrast, lighting, rate
- We are so good at this that its hard to appreciate how difficult it is.
  - Its one of the main difficulties in making computers perceive.
  - We still don't have generally accepted solutions.

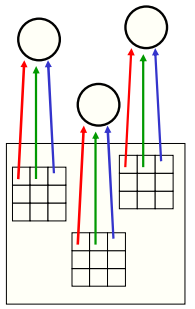
Le Net

- Yann LeCun and others developed a really good recognizer for handwritten digits by using backpropagation in a feedforward net with:
  - Many hidden layers
  - Many pools of replicated units in each layer.
  - Averaging of the outputs of nearby replicated units.
  - A wide net that can cope with several characters at once even if they overlap.
- Look at all of the demos of LENET at http://yann.lecun.com

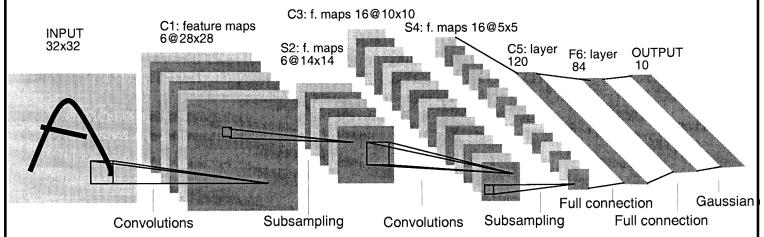
The replicated feature approach

- Use many different copies of the same feature detector.
  - The copies all have slightly different positions.
  - Could also replicate across scale and orientation.
    - Tricky and expensive
  - Replication reduces the number of free parameters to be learned.
- Use several different feature types, each with its own replicated pool of detectors.
  - Allows each patch of image to be represented in several ways.

The red connections all have the same weight.



The architecture of LeNet5



Backpropagation with weight constraints

- It is easy to modify the backpropagation algorithm to incorporate linear constraints between the weights.
- We compute the gradients as usual, and then modify the gradients so that they satisfy the constraints.
  - So if the weights started off satisfying the constraints, they will continue to satisfy them.

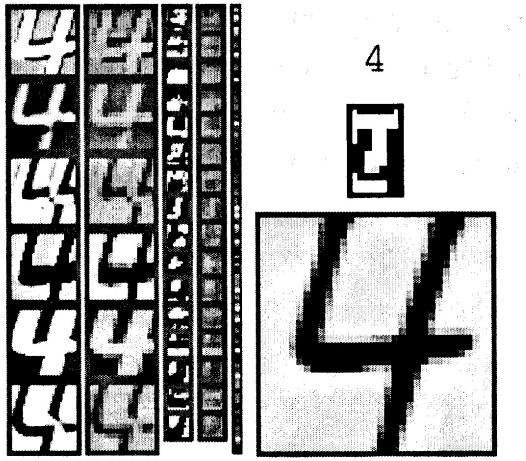
$$To\ \ constrain: \quad w_1 = w_2$$
$$we\ \ need: \quad \Delta w_1 = \Delta w_2$$

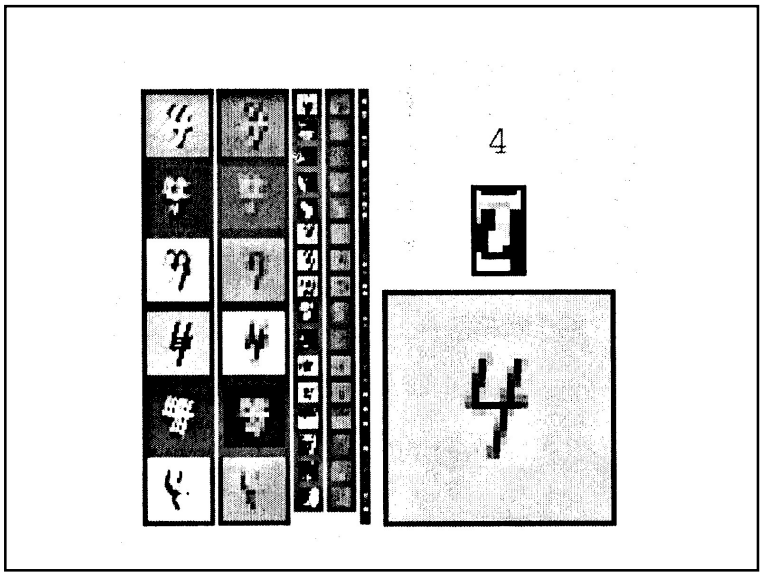
$$compute: \quad \frac{\partial E}{\partial w_1} \quad and \quad \frac{\partial E}{\partial w_2}$$

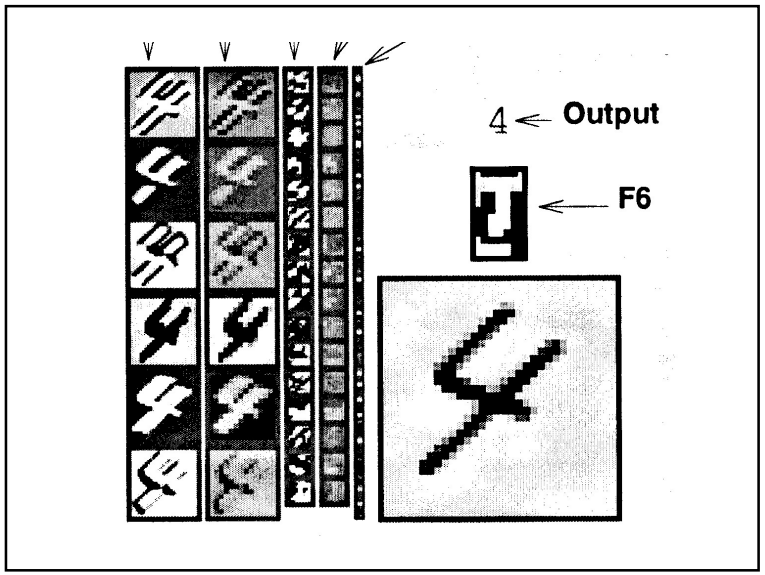
$$use \quad \frac{\partial E}{\partial w_1} + \frac{\partial E}{\partial w_2} \quad for\ w_1\ and\ w_2$$

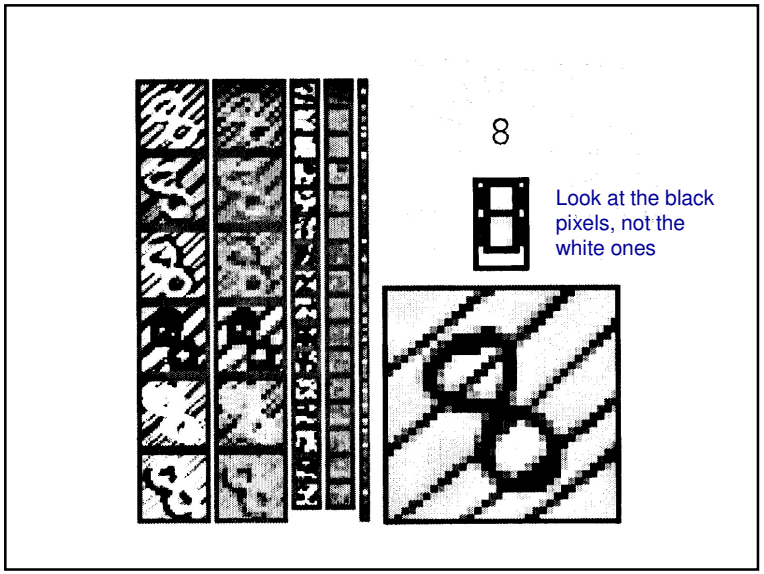
Combining the outputs of replicated features

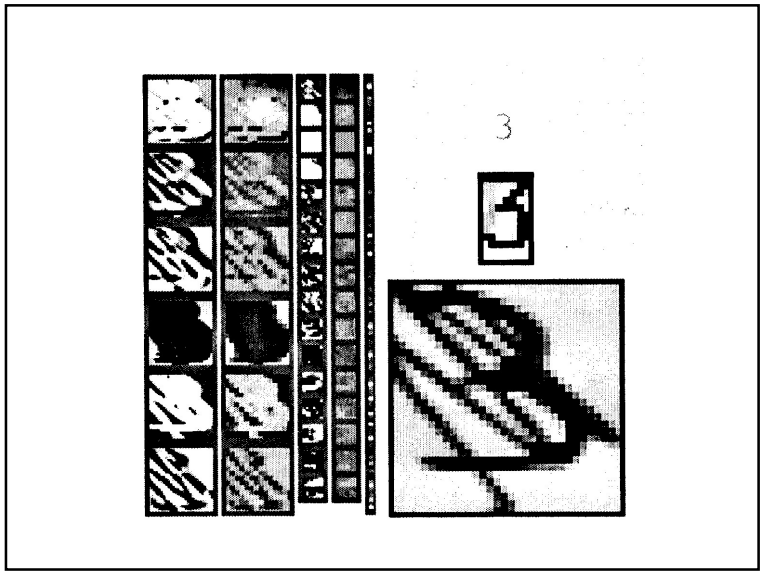
- Get a small amount of translational invariance at each level by averaging four neighboring replicated detectors to give a single output to the next level.
  - Taking the maximum of the four should work better.
- Achieving invariance in multiple stages seems to be what the monkey visual system does.




4


4 ← Output
F6


8
Look at the black pixels, not the white ones


3

4

4

3

3

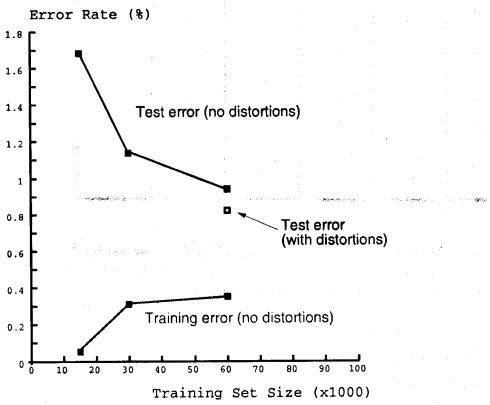


Fig. 6. Training and test errors of LeNet-5 achieved using training sets of various sizes. This graph suggests that a larger training set could improve the performance of LeNet-5. The hollow square show the test error when more training patterns are artificially generated using random distortions. The test patterns are not distorted.

The 82 errors made by LeNet5

Notice that most of the errors are cases that people find quite easy.

The human error rate is probably 20 to 30 errors

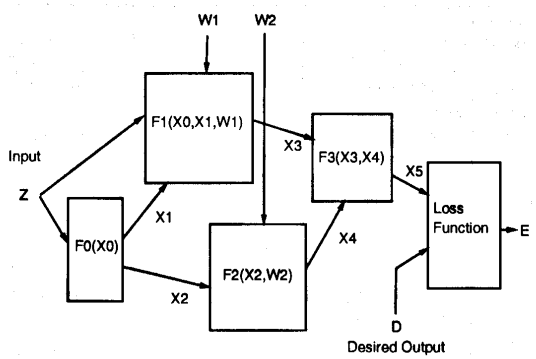


Fig. 14. A trainable system composed of heterogeneous modules.

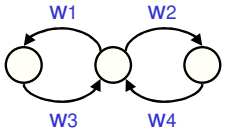
Recurrent networks

- If the connectivity has directed cycles, the network can do much more than just computing a fixed sequence of non-linear transforms:
  - It can oscillate. Good for motor control?
  - It can settle to point attractors.
    - Good for classification
  - It can behave chaotically
    - But that is usually a bad idea for information processing.
  - It can use activities to remember things for a long time
    - The network has internal state. It can decide to ignore the input for a while if it wants to.
  - It can model sequential data in a natural way.
    - No need to use delay taps to spatialize time.

An advantage of modeling sequential data

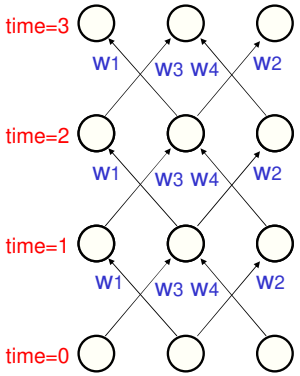
- We can get a teaching signal by trying to predict the next term in a series.
  - This seems much more natural than trying to predict each column of pixels in an image from the columns to the left of it.
    - But its actually equivalent.

The equivalence between layered, feedforward nets and recurrent nets



Assume that there is a time delay of 1 in using each connection.

The recurrent net is just a layered net that keeps reusing the same weights.

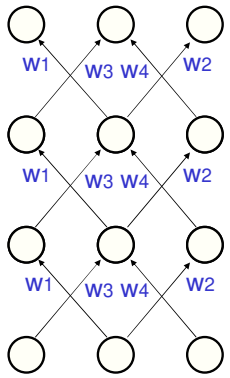


Backpropagation through time

- We could convert the recurrent net into a layered, feed-forward net and then train the feed-forward net with weight constraints.
  - This is clumsy. It is better to move the algorithm to the recurrent domain rather than moving the network to the feed-forward domain.
- So the forward pass builds up a stack of the activities of all the units at each time step. The backward pass peels activities off the stack to compute the error derivatives at each time step.
- After the backward pass we add together the derivatives at all the different times for each weight.
- There is an irritating extra problem:
  - We must specify the initial state of all the non-input units.
    - We could backpropagate all the way to these initial activities and learn the best values for them.

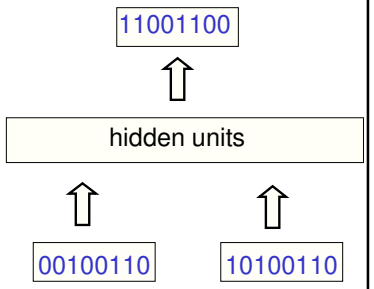
Teaching signals for recurrent networks

- We can specify targets in several ways:
  - Specify the final activities of all the units
  - Specify the activities of all units for the last few steps
    - Good for learning attractors
    - It is easy to add in extra error derivatives as we backpropagate.
  - Specify the activity of a subset of the units.
    - The other units are "hidden"

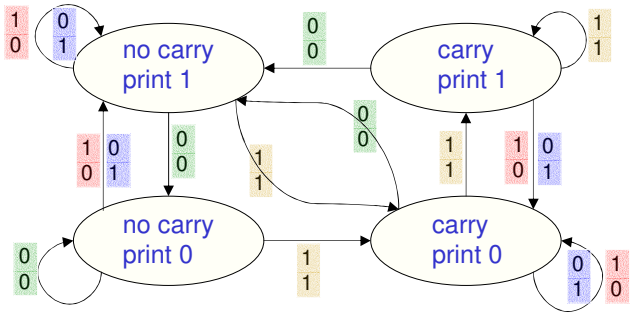


A good problem for a recurrent network

- We can train a feedforward net to do binary addition, but there are obvious regularities that it cannot capture:
  - We must decide in advance the maximum number of digits in each number.
  - The processing applied to the beginning of a long number does not generalize to the end of the long number because it uses different weights.
- As a result, feedforward nets do not generalize well on the binary addition task.



The algorithm for binary addition



This is a finite state automaton. It decides what transition to make by looking at the next column. It prints after making the transition.
It moves from right to left over the two input numbers.

A recurrent net for binary addition

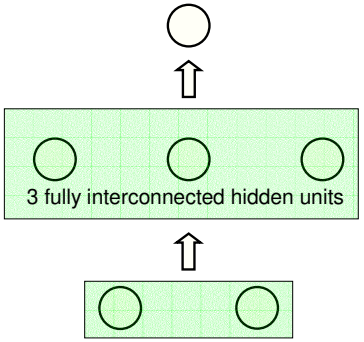
- The network has two input units and one output unit.
- The network is given two input digits at each time step.
- The desired output at each time step is the output for the column that was provided as input two time steps ago.
  - It takes one time step to update the hidden units based on the two input digits.
  - It takes another time step for the hidden units to cause the output.

0 0 1 1 0 1 0 0
0 1 0 0 1 1 0 1
1 0 0 0 0 0 0 1

time

The connectivity of the network

- The 3 hidden units have all possible interconnections in all directions.
  - This allows a hidden activity pattern at one time step to vote for the hidden activity pattern at the next time step.
- The input units have feedforward connections that allow then to vote for the next hidden activity pattern.



What the network learns

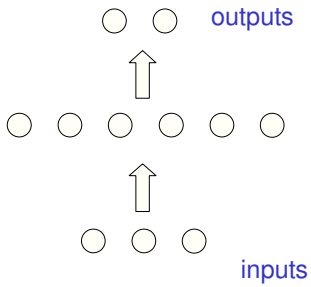
- It learns four distinct pattern of activity for the 3 hidden units. These patterns correspond to the nodes in the finite state automaton.
  - Do not confuse units in a neural network with nodes in a finite state automaton. It is patterns that are mutually exclusive, so patterns correspond to nodes.
  - The automaton is restricted to be in exactly one state at each time. The hidden units are restricted to have exactly one vector of activity at each time.
- A recurrent network can emulate a finite state automaton, but it is exponentially more powerful. With N hidden neurons it has 2^N possible binary activity vectors in the hidden units.
  - This is important when the input stream has several separate things going on at once. A finite state automaton cannot cope with this properly.

Preventing overfitting by early stopping

- If we have lots of data and a big model, its very expensive to keep re-training it with different amounts of weight decay.
- It is much cheaper to start with very small weights and let them grow until the performance on the validation set starts getting worse (but don't get fooled by noise!)
- The capacity of the model is limited because the weights have not had time to grow big.

Why early stopping works

- When the weights are very small, every hidden unit is in its linear range.
  - So a net with a large layer of hidden units is linear.
  - It has no more capacity than a linear net in which the inputs are directly connected to the outputs!
- As the weights grow, the hidden units start using their non-linear ranges so the capacity grows.



Full Bayesian Learning

- Instead of trying to find the best single setting of the parameters (as in ML or MAP) compute the full posterior distribution over parameter settings
  - This is extremely computationally intensive for all but the simplest models.
- To make predictions, let each different setting of the parameters make its own prediction and then combine all these predictions by weighting each of them by the posterior probability of that setting of the parameters.
  - This is also computationally intensive.
- The full Bayesian approach allows us to use complicated models even when we do not have much data

How to deal with the fact that the space of all possible parameters vectors is huge

- If there is enough data to make most parameter vectors very unlikely, only a tiny fraction of the parameter space makes a significant contribution to the predictions.
  - Maybe we can just sample parameter vectors in this tiny fraction of the space.

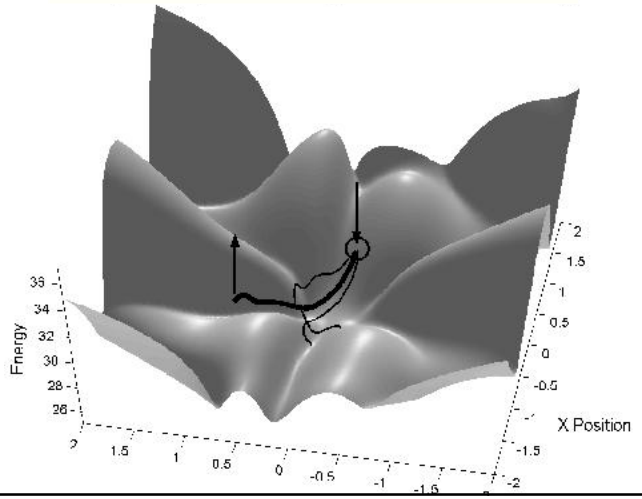
$$p(t_{test} \mid \mathbf{x}_{test}, D) = \sum_i p(\mathbf{W}_i \mid D)\, p(t_{test} \mid \mathbf{x}_{test}, \mathbf{W}_i)$$

Sample weight vectors with this probability

One method for sampling weight vectors

- In standard backpropagation we keep moving the weights in the direction that decreases the cost
  - i.e. the direction that increases the log likelihood plus the log prior, summed over all training cases.
- Suppose we add some Gaussian noise to the weight vector after each update.
  - So the weight vector never settles down.
  - It keeps wandering around, but it tends to prefer low cost regions of the weight space.

An amazing fact

- If we use just the right amount of Gaussian noise, and if we let the weight vector wander around for long enough before we take a sample, we will get a sample from the true posterior over weight vectors.
  - This is called a "Markov Chain Monte Carlo" method and it makes it feasible to use full Bayesian learning with hundreds or thousands of parameters.
  - There are related MCMC methods that are more complicated but more efficient (we don't need to let the weights wander around for so long before we get samples from the posterior).
- Radford Neal (1995) showed that this works extremely well when data is limited but the model needs to be complicated.

Trajectories with different initial momenta