CSC2535:

Lecture 3: Ways to make backpropagation generalize better, and ways to do without a supervision signal.

Geoffrey Hinton

Overfitting

- The training data contains information about the regularities in the mapping from input to output. But it also contains noise
  - The target values may be unreliable.
  - There is sampling error. There will be accidental regularities just because of the particular training cases that were chosen.
- When we fit the model, it cannot tell which regularities are real and which are caused by sampling error.
  - So it fits both kinds of regularity.
  - If the model is very flexible it can model the sampling error really well. This is a disaster.

Preventing overfitting

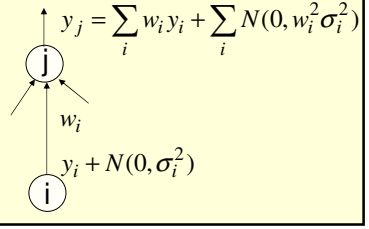
- Use a model that has the right capacity:
  - enough to model the true regularities
  - not enough to also model the spurious regularities (assuming they are weaker).
- Standard ways to limit the capacity of a neural net:
  - Limit the number of hidden units.
  - Limit the size of the weights.
  - Stop the learning before it has time to overfit.

Limiting the size of the weights

- Weight-decay involves adding an extra term to the cost function that penalizes the squared weights.
  - Keeps weights small unless they have big error derivatives.
- This reduces the effect of noise in the inputs.
  - The noise variance is amplified by the squared weight

$$C = E + \frac{\lambda}{2}\sum_i w_i^2$$

$$\frac{\partial C}{\partial w_i} = \frac{\partial E}{\partial w_i} + \lambda w_i$$

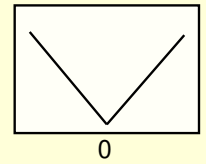


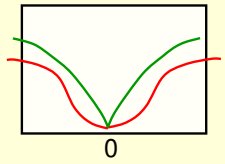
The effect of weight-decay

- It prevents the network from using weights that it does not need.
  - This helps to stop it from fitting the sampling error. It makes a smoother model in which the output changes more slowly as the input changes.
- It can often improve generalization a lot.
- If the network has two very similar inputs it prefers to put half the weight on each rather than all the weight on one.

Other kinds of weight penalty

- Sometimes it works better to penalize the absolute values of the weights.
  - This makes some weights equal to zero which helps interpretation.
- Sometimes it works better to use a weight penalty that has negligible effect on large weights.





Deciding how many hidden units or how much weight-decay

- How do we decide how to limit the capacity of the network?
  - If we use the test data we get an unfair prediction of the error rate we would get on new test data.
  - Suppose we compared a set of models that gave random results, the best one on a particular dataset would do better than chance. But it wont do better than chance on another test set.
- So use a separate validation set to do model selection.

Using a validation set

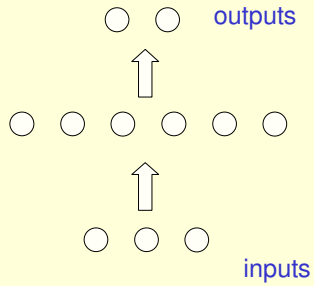
- Divide the total dataset into three subsets:
  - Training data is used for learning the parameters of the model.
  - Validation data is not used of learning but is used for deciding what type of model and what amount of regularization works best.
  - Test data is used to get a final, unbiased estimate of how well the network works. We expect this estimate to be worse than on the validation data.
- We could then re-divide the total dataset to get another unbiased estimate of the true error rate.

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.



The Bayesian framework

- The Bayesian framework assumes that we always have a prior distribution for everything.
  - The prior may be very vague.
- When we see some data, we combine our prior with a likelihood term to get a posterior distribution.
  - The likelihood term takes into account how likely the observed data is given the parameters of the model.
  - It favors parameter settings that make the data likely.
- With enough data, the likelihood term always dominates the prior.

Bayes Theorem

joint probability

conditional probability

$$p(D)\,p(W \mid D) = p(D,W) = p(W)\,p(D \mid W)$$

Prior probability of weight vector W

Probability of observed data given W

$$p(W \mid D) = \frac{p(W)\,p(D \mid W)}{p(D)}$$

Posterior probability of weight vector W given training data D

$$p(D) = \int_W p(W)\,p(D \mid W)$$

A cheap trick to avoid computing the posterior probabilities of all weight vectors

- Suppose we just try to find the most probable weight vector.
  - We can do this by starting with a random weight vector and then adjusting it in the direction that improves p(W | D).
- It is easier to work in the log domain. If we want to minimize a cost we use negative log probabilities:

$$p(W \mid D) = p(W)\ p(D \mid W) \ / \ p(D)$$

$$Cost = -\log p(W \mid D) = -\log p(W) - \log p(D \mid W) + \log p(D)$$

Why we maximize sums of log probs

- We want to maximize the product of the probabilities of the outputs on the training cases
  - Assume the output errors on different training cases, c, are independent.

$$p(D \mid W) = \prod_c p(d_c \mid W)$$

- Because the log function is monotonic, it does not change where the maxima are. So we can maximize sums of log probabilities

$$\log p(D \mid W) = \sum_c \log p(d_c \mid W)$$

A even cheaper trick

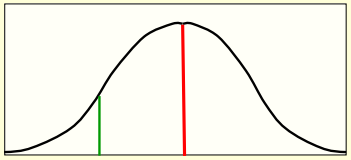
- Suppose we completely ignore the prior over weight vectors
  - This is equivalent to giving all possible weight vectors the same prior probability density.
- Then all we have to do is to maximize:

$$\log p(D \mid W) = \sum_c \log p(D_c \mid W)$$

- This is called maximum likelihood learning. It is very widely used for fitting models in statistics.

Supervised Maximum Likelihood Learning

- Minimizing the squared residuals is equivalent to maximizing the log probability of the correct answer under a Gaussian centered at the model's guess.



d = the correct answer

y = model's estimate of most probable value

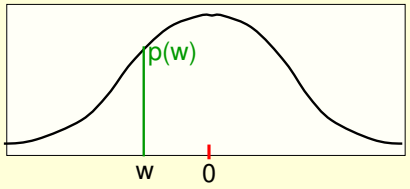
$$y_c = f(input_c, W)$$

$$p(output = d_c \mid input_c, W) = p(d_c \mid y_c) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(d_c - y_c)^2}{2\sigma^2}}$$

$$-\log p(output = d_c \mid input_c, W) = k + \frac{(d_c - y_c)^2}{2\sigma^2}$$

Maximum A Posteriori Learning

- This trades-off the prior probabilities of the parameters against the probability of the data given the parameters. It looks for the parameters that have the greatest product of the prior term and the likelihood term.
- Minimizing the squared weights is equivalent to maximizing the log probability of the weights under a zero-mean Gaussian prior.



$$p(w) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{w^2}{2\sigma_W^2}}$$

$$-\log p(w) = \frac{w^2}{2\sigma_W^2} + k$$

The Bayesian interpretation of weight decay

$$-\log p(W|D) = -\log p(D|W) - \log p(W) + \log p(D)$$

$$C^* = \frac{1}{2\sigma_D^2}\sum_c (y_c - d_c)^2 + \frac{1}{2\sigma_W^2}\sum_i w_i^2$$

constant

assuming that the model makes a Gaussian prediction

assuming a Gaussian prior for the weights

$$C = E + \frac{\sigma_D^2}{\sigma_W^2}\sum_i w_i^2$$

So the correct value of the weight decay parameter is the ratio of two variances. Its not just an arbitrary hack.

Full Bayesian Learning

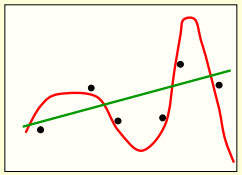
- Instead of trying to find the best single setting of the parameters (as in ML or MAP) compute the full posterior distribution over parameter settings
  - This is extremely computationally intensive for all but the simplest models.
- To make predictions, let each different setting of the parameters 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

Overfitting: A frequentist illusion?

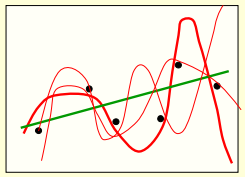
- If you do not have much data, you should use a simple model, because a complex one will overfit.
  - This is true. But only if you assume that fitting a model means choosing a single best setting of the parameters.
  - If you use the full posterior over parameter settings, overfitting disappears!
  - With little data, you get very vague predictions because many different parameters settings have significant posterior probability

A classic example of overfitting

- Which model do you believe?
  - The complicated model fits the data better.
  - But it is not economical and it makes silly predictions.



- But what if we start with a reasonable prior over all fifth-order polynomials and use the full posterior distribution.
  - Now we get vague and sensible predictions.
- There is no reason why the amount of data should influence our prior beliefs about the complexity of the model.



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.

$$p(y_{test} \mid input_{test}, D) = \sum_i p(W_i \mid D)\; p(y_{test} \mid input_{test}, W_i)$$

Sample weight vectors with this probability

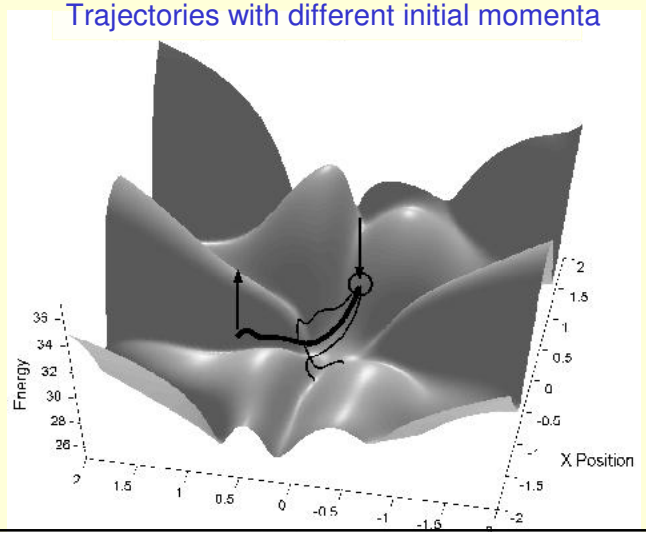
One method for sampling weight vectors

- 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



The frequentist version of the idea of using the posterior distribution over parameter vectors

- The expected squared error made by a model has two components that add together:
  - Models have systematic bias because they are too simple to fit the data properly.
  - Models have variance because they have many different ways of fitting the data almost equally well. Each way gives different test errors.
- If we make the models more complicated, it reduces bias but increases variance. So it seems that we are stuck with a bias-variance trade-off.
  - But we can beat the trade-off by fitting lots of models and averaging their predictions. The averaging reduces variance without increasing bias. (Its just like holding lots of different stocks instead of one)

Ways to do model averaging

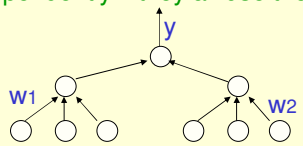
- We want the models in an ensemble to be different from each other.
  - Bagging: Give each model a different training set by using large random subsets of the training data.
  - Boosting: Train models in sequence and give more weight to training cases that the earlier models got wrong.

Two regimes for neural networks

- If we have lots of computer time and not much data, the problem is to get around overfitting so that we get good generalization
  - Use full Bayesian methods for backprop nets.
  - Use methods that combine many different models.
  - Use Gaussian processes (not yet explained)
- If we have a lot of data and a very complicated model, the problem is that fitting takes too long.
  - Backpropagation is competitive in this regime.

Three problems with backpropagation

- Where does the supervision come from?
  - Most data is unlabelled
    - The vestibular-ocular reflex is an exception.
- How well does the learning time scale?
  - Its is impossible to learn features for different parts of an image independently if they all use the same error signal.



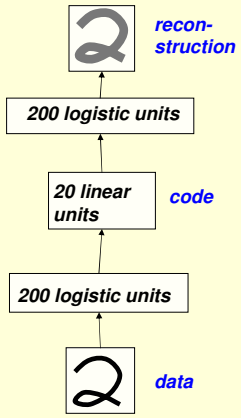
- Can neurons implement backpropagation?
  - Not in the obvious way.
    - but getting derivatives from later layers is so important that evolution may have found a way.

Four ways to use backpropagation without requiring a supervision signal

- Make the desired output be the same as the input and make the middle layer of the network small.
  - This does dimensionality reduction.
- Maximize the mutual information between the scalar output values of two or more networks.
  - This discovers spatial or temporal invariants.
- Make the output change as slowly as possible over time
  - This discovers very neuron-like features.
- Minimize the distance between the outputs of two nets for images of the same person and maximize it for images of different people.
  - This learns a distance metric in which images of the same person are very similar even if they are superficially very different.

Self-supervised backpropagation

- Autoencoders define the desired output to be the same as the input.
  - Trivial to achieve with direct connections
    - The identity is easy to compute!
- It is useful if we can squeeze the information through some kind of bottleneck:
  - If we use a linear network this is very similar to Principal Components Analysis



Self-supervised backprop and PCA

- If the hidden and output layers are linear, it will learn hidden units that are a linear function of the data and minimize the squared reconstruction error.
- The m hidden units will span the same space as the first m principal components
  - Their weight vectors may not be orthogonal
  - They will tend to have equal variances

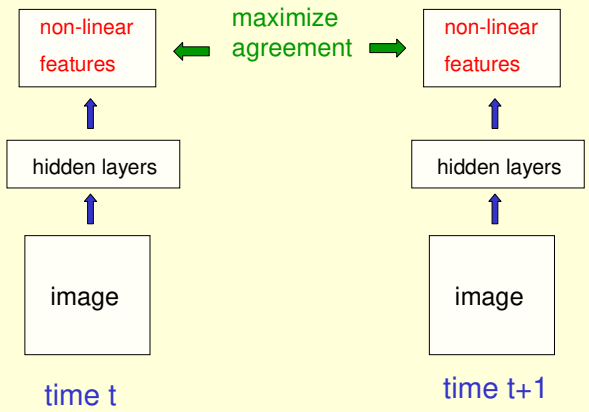
Self-supervised backprop in deep autoencoders

- We can put extra hidden layers between the input and the bottleneck and between the bottleneck and the output.
  - This gives a non-linear generalization of PCA
- It should be very good for non-linear dimensionality reduction.
  - It is very hard to train with backpropagation
  - So deep autoencoders have been a big disappointment.
    - But we recently found a very effective method of training them which will be described later in the course.

Temporally invariant properties

- Consider a rigid object that is moving relative to the retina:
  - Its retinal image changes in predictable ways
  - Its true 3-D shape stays exactly the same. It is invariant over time.
  - Its angular momentum also stays the same if it is in free fall.
- Properties that are invariant over time are usually interesting.

Learning temporal invariances



A new way to get a teaching signal

- Each module uses the output of the other module as the teaching signal.
  - This does not work if the two modules can see the same data. They just report one component of the data and agree perfectly.
  - It also fails if a module always outputs a constant. The modules can just ignore the data and agree on what constant to output.
- We need a sensible definition of the amount of agreement between the outputs.

Mutual information

- Two variables, a and b, have high mutual information if you can predict a lot about one from the other.

$$I(a;b) = H(a) + H(b) - H(a,b)$$

Mutual Information — Individual entropies — Joint entropy

- There is also an asymmetric way to define mutual information:

$$I(a;b) = H(a) - H(a \mid b)$$

Some advantages of mutual information

- If the modules output constants the mutual information is zero.
- If the modules each output a vector, the mutual information is maximized by making the components of each vector be as independent as possible.
- Mutual information exactly captures what we mean by "agreeing".

A problem

- We can never have more mutual information between the two output vectors than there is between the two input vectors.
  - So why not just use the input vector as the output?
- We want to preserve as much mutual information as possible whilst also achieving something else:
  - Dimensionality reduction?
  - A simple form for the prediction of one output from the other?

Simple forms for the relationship

- Assumption: the output of module a equals the output of module b plus noise: $a = b + \varepsilon$

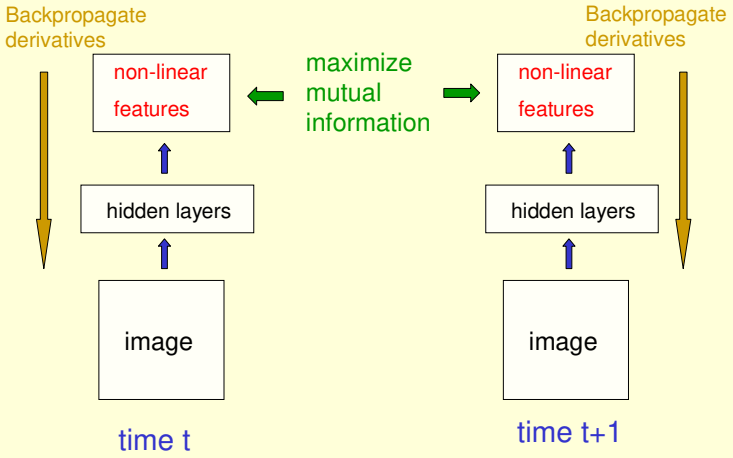
$$I(a;b) = H(a) - H(a \mid b)$$
$$H(a \mid b) = H(\varepsilon) = H(a - b)$$
$$\textit{for Gaussian distributions}$$
$$I(a;b) = \log \sigma_a - \log \sigma_{a-b}$$
$$= \frac{1}{2} \log \frac{\sigma_a^2}{\sigma_{a-b}^2}$$

- Alternative assumption: a and b are both noisy versions of the same underlying signal.

$$a = s + \varepsilon_a \qquad b = s + \varepsilon_b$$

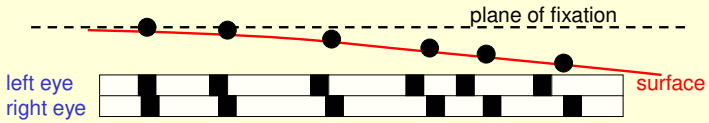
$$\textit{for Gaussian distributions}$$
$$I(a;b) = \frac{1}{2} \log \frac{Var(signal + noise)}{Var(noise)}$$
$$= \frac{1}{2} \log \frac{\sigma_{a+b}^2}{\sigma_{a-b}^2}$$

Learning temporal invariances

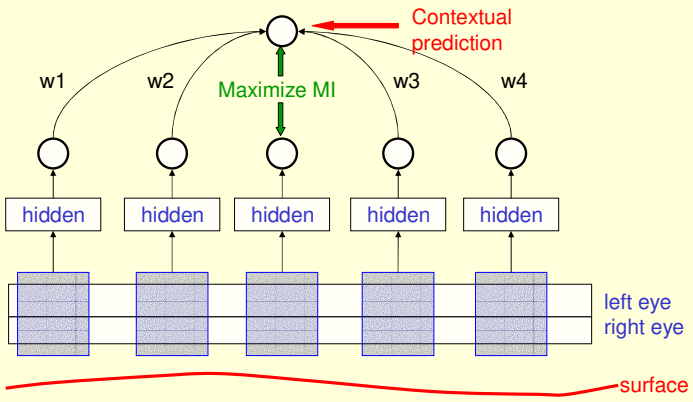


Spatially invariant properties

- Consider a smooth surface covered in random dots that is viewed from two different directions:
  - Each image is just a set of random dots.
  - A stereo pair of images has disparity that changes smoothly over space. Nearby regions of the image pair have very similar disparities.



Maximizing mutual information between a local region and a larger context



How well does it work?

- If we use weight sharing between modules and plenty of hidden units, it works really well.
  - It extracts the depth of the surface fairly accurately.
  - It simultaneously learns the optimal weights of -1/6, +4/6, +4/6, -1/6 for interpolating the depths of the context to predict the depth at the middle module.
- If the data is noisy or the modules are unreliable it learns a more robust interpolator that uses smaller weights in order not to amplify noise.

Slow Feature Analysis (Berges & Wiskott, Wiskott & Sejnowski)

- Use three consecutive time frames from a fake video sequence as the two inputs: [t-1, t] & [t, t+1]
  - The sequence is made from a large, still, natural image by translating, expanding ,and rotating a square window and then pixelating to get sequences of 16x16 images.
  - Two 256 pixel images are reduced to 100 dimensions using PCA then non-linearly expanded by taking pairwise products of components. This provides the 5050 dimensional input to one module.

The SFA objective function

$$\textit{Find weight vectors} \quad y_j = \mathbf{w}_j^T \mathbf{x}$$

$$\textit{to Minimize} \quad < \dot{y}_j^2 >_t \quad \textit{for all } j$$

$$\textit{subject to}:$$

$$< y_j >_t = 0, \quad < y_j^2 >_t = 1$$

$$< y_i y_j >_t = 0 \quad \textit{for all } i, j$$

The solution can be found by solving a generalized eigenvalue problem:

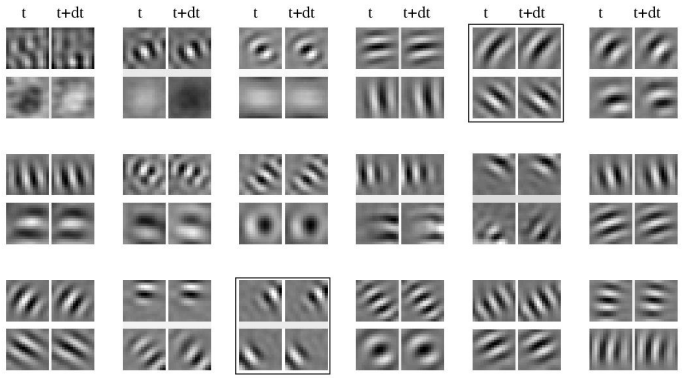
$$\mathbf{AW} = \Lambda \mathbf{BW}$$

$$\textit{where} \quad \mathbf{A} = \dot{\mathbf{x}}\dot{\mathbf{x}}^T, \quad \mathbf{B} = \mathbf{x}\mathbf{x}^T$$

The slow features

- They have a lot of similarities to the features found in the first stage of visual cortex.
- They can be displayed by showing the pair of temporally adjacent images that excite them most and the pair that inhibit them most.

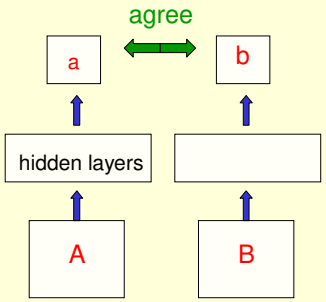
The most excitatory pair of images and the most inhibitory pair of images for some slow features



A way to learn non-linear transformations that maximize agreement between the outputs of two modules

- We want to explain why we observe particular pairs of images rather than observing other pairings of the same set of images.
  - This captures the non "iid-ness" of the data.
- We can formulate this probabilistically using "disagreement" energies

An energy-based model of agreement

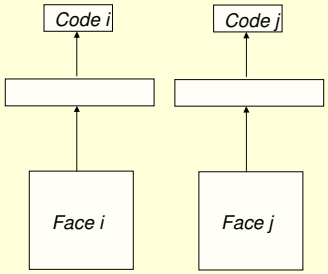


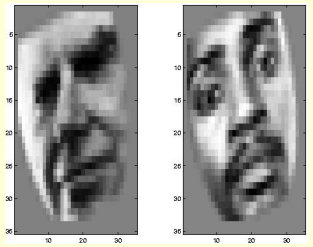
$$p(A^c, B^c) = \frac{e^{-\|a^c - b^c\|^2}}{\sum_{g,h} e^{-\|a^g - b^h\|^2}}$$

$$Maximize \sum_c \log p(A^c, B^c)$$

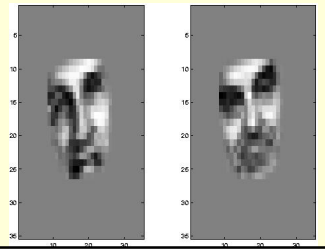
Using agreement to train a feedforward neural net

- Use pairs of face images that have similar orientations and scales but are otherwise quite different.
- Use a feedforward net to map the image to a 2-D code.
- The SNE derivatives are back-propagated through the net.
  - This regularizes the embedding and also makes it easy to apply to new data.
- The aim of the net is to make the codes similar for the pairs it is given.



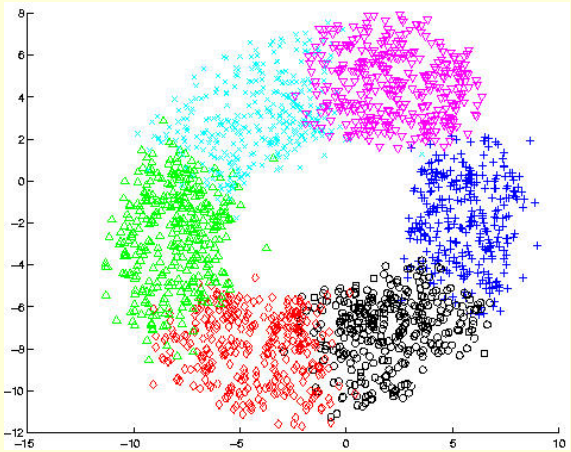
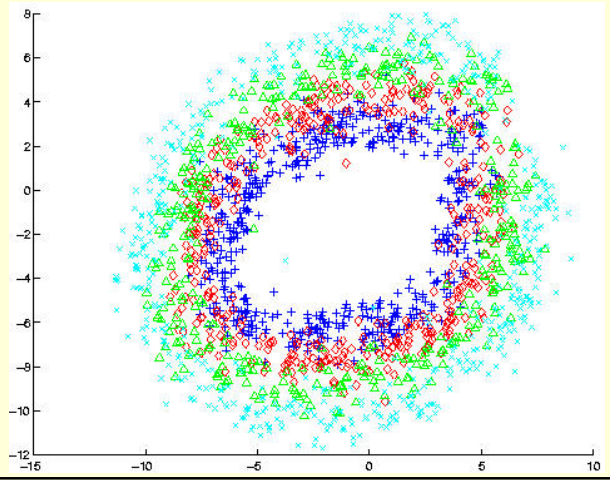


Large pair



Small pair

Each color is for a different band of orientations (from -45 to +45)



Each color is for a different scale (from small to large)