CSC2535

Lecture 2: Some examples of backpropagation learning

Geoffrey Hinton

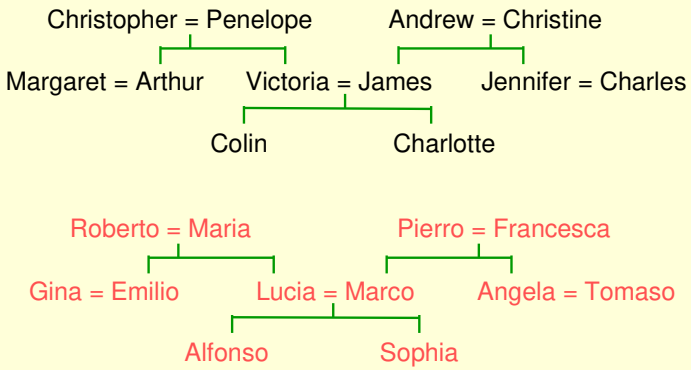
Some Success Stories

- 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.
  - Understanding the effects of brain damage

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
  - 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.
- Inverting computer graphics (if there is time)
  - Using the knowledge in a graphics program to produce a vision program that goes in the opposite direction, even though you don't know the inputs to the graphics program.

An example of relational information



Another way to express the same information

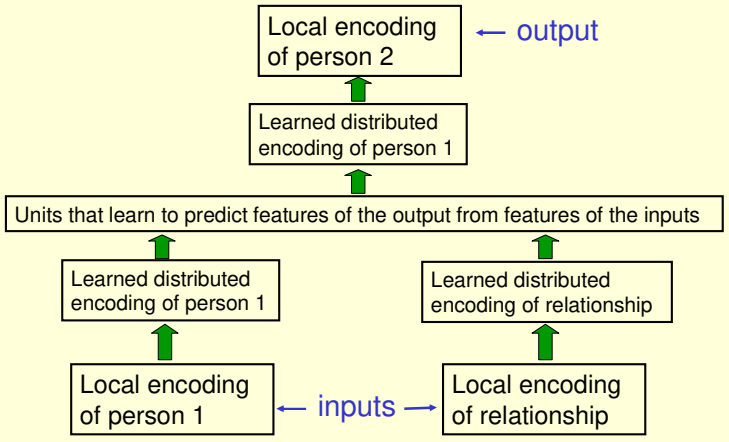
- Make a set of propositions using the 12 relationships:
  - son, daughter, nephew, niece
  - father, mother, uncle, aunt
  - brother, sister, husband, wife
- (colin has-father james)
- (colin has-mother victoria)
- (james has-wife victoria) this follows from the two above
- (charlotte has-brother colin)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

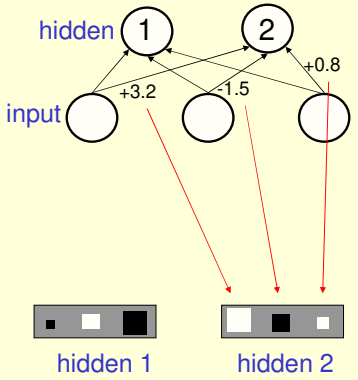
    (x has-mother y) & (y has-husband z) => (x has-father z)
- Finding the symbolic rules involves a difficult search through a very large discrete space of possibilities.
- Can a neural network capture the same knowledge by searching through a continuous space of weights?

The structure of the neural net

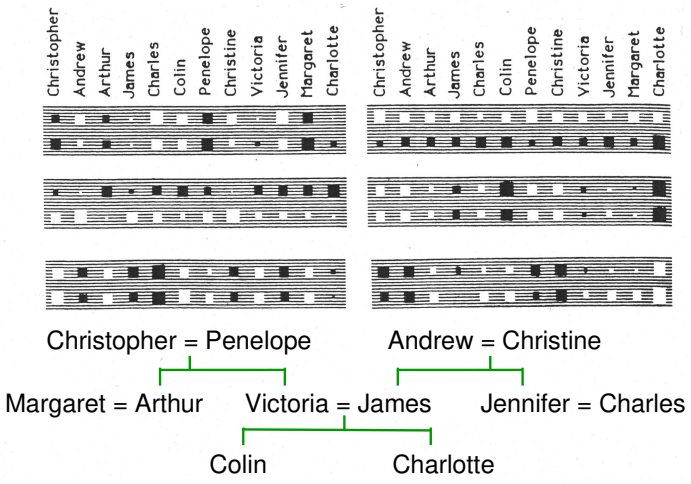


How to show the weights of hidden units

- 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 subtelty

- The obvious way to implement a relational graph in a neural net is to treat a neuron as a node in the graph and a connection as a binary relationship. But this will not work:
  - We need many different types of relationship
    - Connections in a neural net do not have labels.
  - We need ternary relationships as well as binary ones. e.g. (A is between B and C)
  - Its just naïve to think neurons are concepts.

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 ?

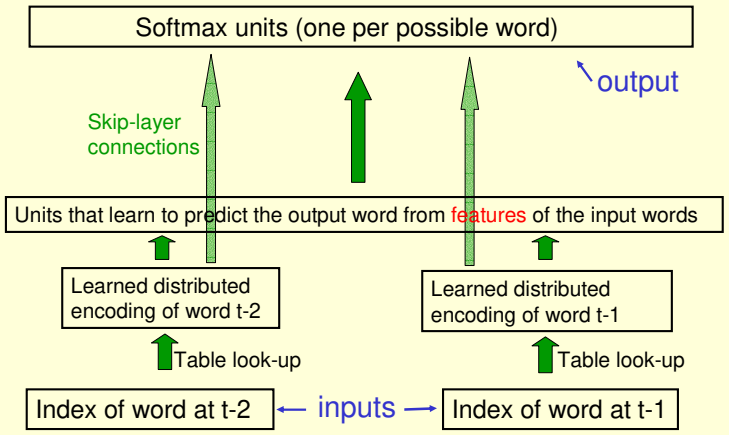
$$\frac{p(w_3 = c \mid w_2 = b, w_1 = a)}{p(w_3 = d \mid w_2 = b, w_1 = a)} = \frac{count(abc)}{count(abd)}$$

- 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.

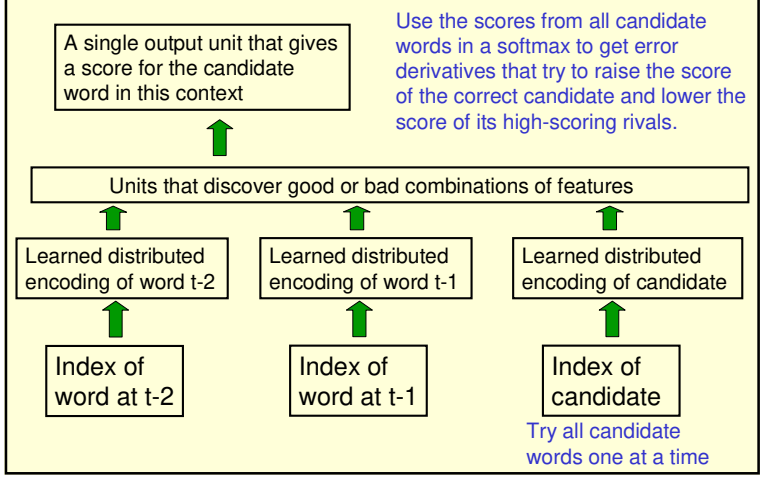
Why the trigram model is silly

- 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



An alternative architecture



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).
  - 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.

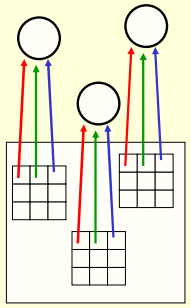
Le Net

- 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
  - These demos are required "reading" for the tests.

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.



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.
  - Segmentation may also be done in multiple stages.

The 82 errors made by LeNet5

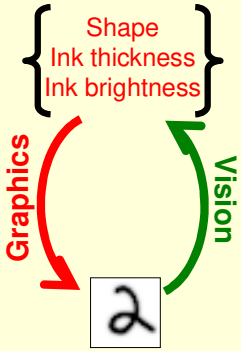
Notice that most of the errors are cases that people find quite easy.

The human error rate is probably 20 to 30 errors

Generative models: A different way to put in prior knowledge

- We know a lot about how digit images are created.
- So instead of using weight constraints to capture prior knowledge, maybe we can somehow make use of our knowledge about the generative process.
- We can easily write a program that creates fairly realistic digit images from a sequence of motor commands.
  - But how can we make use of a generative model to help us create a network that goes the other way?
  - If we could get the motor commands, digit recognition should be easy. This is called "analysis by synthesis".

Modelling Handwritten Digits



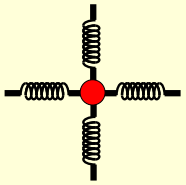
A graphics model is a powerful way of representing prior knowledge for a vision problem.

But this prior knowledge is unusable for supervised learning if a labelled set of images and corresponding graphics inputs is not available.

Given only a set of images and a conditional generative model how can we learn a recognition model that infers generative inputs from an image?

A Generative Model for Digits

A simple simulation of the physics of drawing.

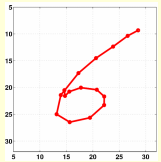


Mass-spring system

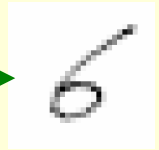
Spring stiffnesses are varied for a fixed number of discrete time steps.

This produces a time-varying force on the mass, causing it to move along a trajectory.

A ***motor program*** is the sequence of stiffness values and two ink parameters.



Compute trajectory from motor program



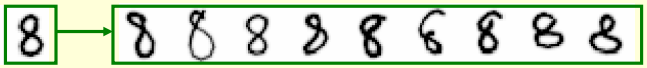
Trace trajectory with ink

Convolve to desired thickness & intensity

Why Motor Programs are a Good Language

Small changes in the spring stiffnesses map to sensible changes in the topology of the digits.

The trajectories live in the "correct" vector space.



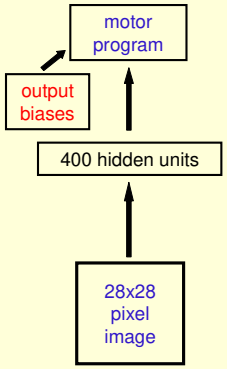
The images on the right were generated by adding Gaussian noise to the motor program of the image on the left. These images are all near each other in motor program space, but they are far apart in pixel space.

Some Ways to Invert a Generator

- Look inside the generator to see how it works (Williams et al.)
  - Too tedious. Not possible for the real motor system.
- Define a prior over codes and generate (code, image) pairs. Then train a "recognition" neural net that does image → code.
  - What about ambiguous images? The average code is bad.
- Define a prior over codes and generate (code, image) pairs. Then train a "generative" neural net that does code → image. Then backpropagate image residuals to get derivatives for the code units and iteratively find a locally optimal code for each test image.
  - But how do we decide what code to start with?
- In both cases we need the prior over codes to generate training examples in the relevant part of the space.
  - But the distribution of codes is what we want to learn from the data!
  - Is there any way to do without a the prior over codes?

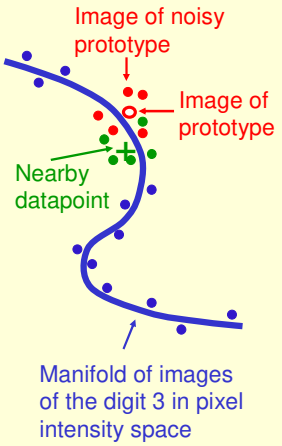
Training a Net to Map Images of 3's to Motor Programs

- Start with a single "prototype" motor program that draws a nice 3.
  - This motor program is created by hand.
- Use the graphics program to make many similar {image, motor-program} pairs by adding a little noise to the motor program.
- Initialize a neural net with very small weights and set the output biases so that it always produces the prototype.
- Train the net until it has an "island of competence" around the prototype.
  - It learns how to convert small variations in the image into small variations in the motor program.



Growing the island of competence along the manifold

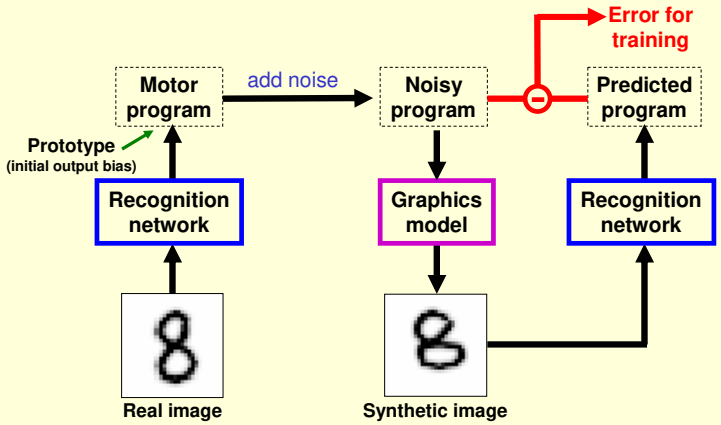
- After learning to invert the generator in the vicinity of the prototype, we can perceive the right motor program for real images that are similar to the prototype.
- Add noise to these perceived motor programs and generate more training data.
- This extends the island of competence along the data manifold.
- Adding noise to a motor program very seldom changes the class of the image. So almost all training data consists of images of 3's.



A Surprise

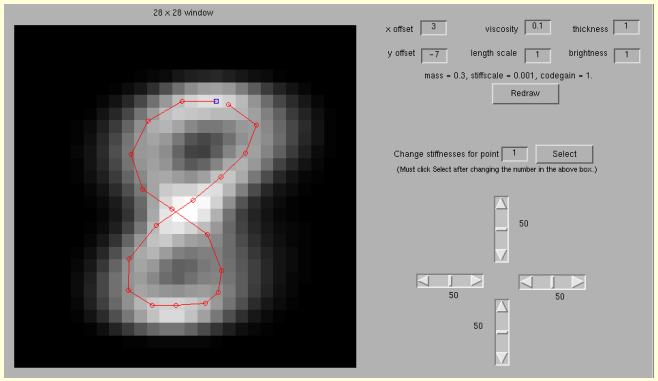
- We initially assumed that we would need to grow the island of competence gradually by only extracting motor programs from images close to the island of competence.
  - This can be done by monitoring the reconstruction error in pixel space and rejecting cases with large error.
- But the learning works fine if we allow the neural net to see all of the MNIST images of 3's right from the start.
- At the start of learning it extracts an incorrect motor program if the image is far from the island of competence. But this doesn't matter!
- The incorrect motor-program is much closer to the island of competence than the correct one. So the incorrect motor programs still produce good 3's for training.

Learning Algorithm

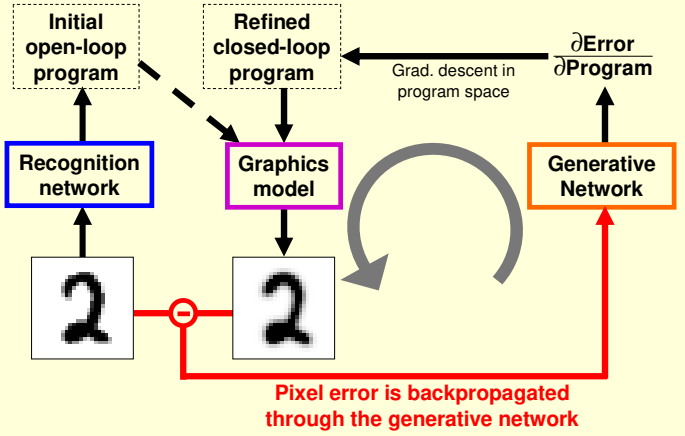


Creating a Prototype

A class prototype is created by manually setting the spring stiffnesses so that the trajectory traces out the average image of the class.

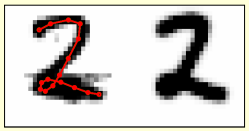


Local Search



Trajectory Prior & Residual Model

A class-specific PCA model of the trajectories acts as a prior on the trajectories. A trajectory's score under the prior is computed as the Euclidean distance between itself and its projection to the PCA subspace.



Reconstruction of the left image as a 2.
Squared pixel error = 24.5
Trajectory prior score = 31.5

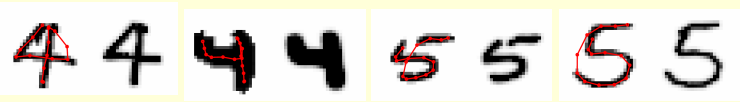
Reconstruction of the left image as a 3.
Squared pixel error = 15.0
Trajectory prior score = 104.2

The prior penalizes a class model for using an unusual trajectory to better explain the ink in an image from the wrong class.

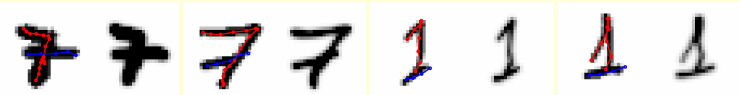
The image residual for each class is also modelled using PCA.

Lifting the Pen & Extra Stroke

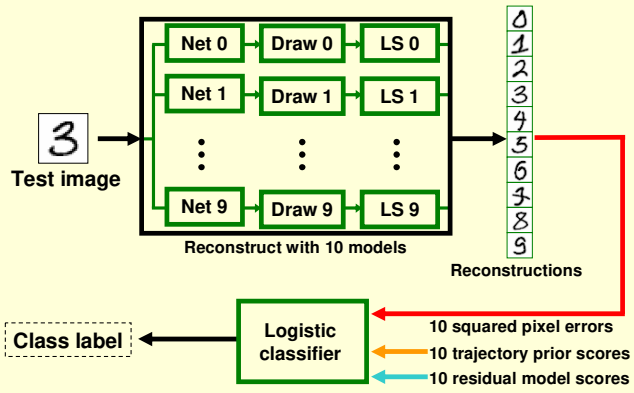
The pen may have to be lifted off the paper in order to draw fours and fives. We simulate this by turning off the ink for two fixed time steps along the trajectory.



Sevens and ones may require an extra stroke to draw a dash. We train an additional net for both seven and one to compute the motor program just for the dash.



Classification on MNIST Database



Error rate on the MNIST test set: **1.82%**