CSC321: Neural Networks

Lecture 5: Applying backpropagation to shape recognition

Geoffrey Hinton

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.

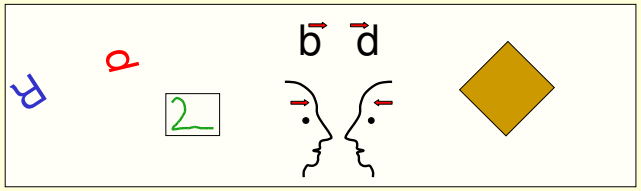
The invariant feature approach

- Extract a large, redundant set of features that are invariant under transformations
  - e.g. "pair of parallel lines with a dot between them.

- With enough of these features, there is only one way to assemble them into an object.
  - we don't need to represent the relationships between features directly because they are captured by other features.
- We must avoid forming features from parts of different objects!

The normalization approach

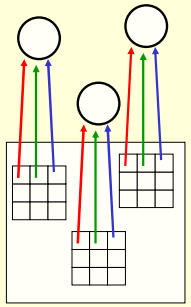
- Do preprocessing to normalize the data
  - e. g. put a box around an object and represent the locations of its pieces relative to this box.
  - Eliminates as many degrees of freedom as the box has.
    - translation, rotation, scale, shear, elongation
  - But its not always easy to choose the box



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.

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

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

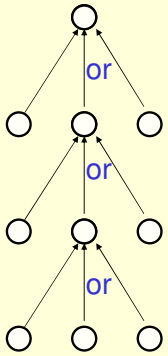
$$use \quad \frac{\partial E}{\partial w_1} + \frac{\partial E}{\partial w_2} \quad for\ w_1\ and\ w_2$$

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 hierarchical partial invariance approach

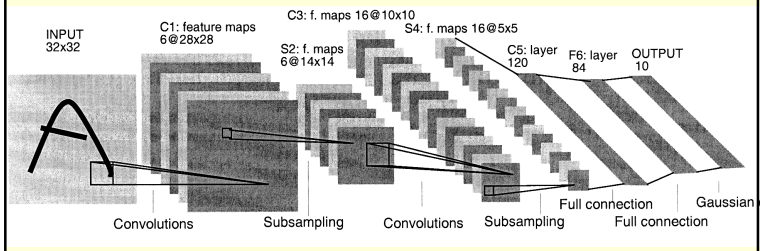
- At each level of the hierarchy, we use an "or" to get features that are invariant across a bigger range of transformations.
  - Receptive fields in the brain look like this.
  - We can combine this approach with an initial approximate normalization.

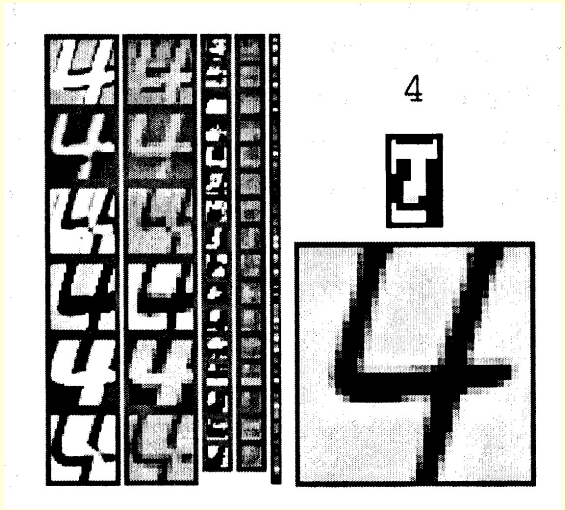


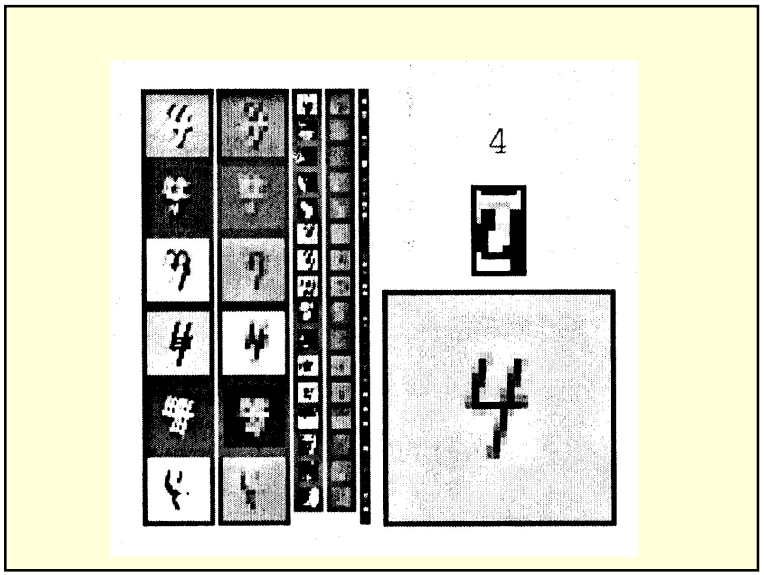
Le Net

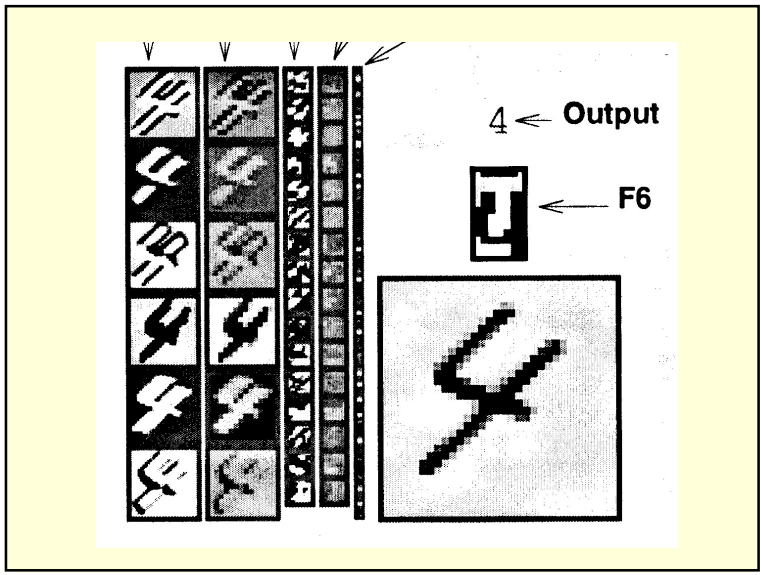
- Yann LeCun and others developed a really good recognizer for handwritten digits by using backpropagation in a feedforward net with:
  - Many hidden layers
  - Many pools of replicated units in each layer.
  - Averaging of the outputs of nearby replicated units.
  - A wide net that can cope with several 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.

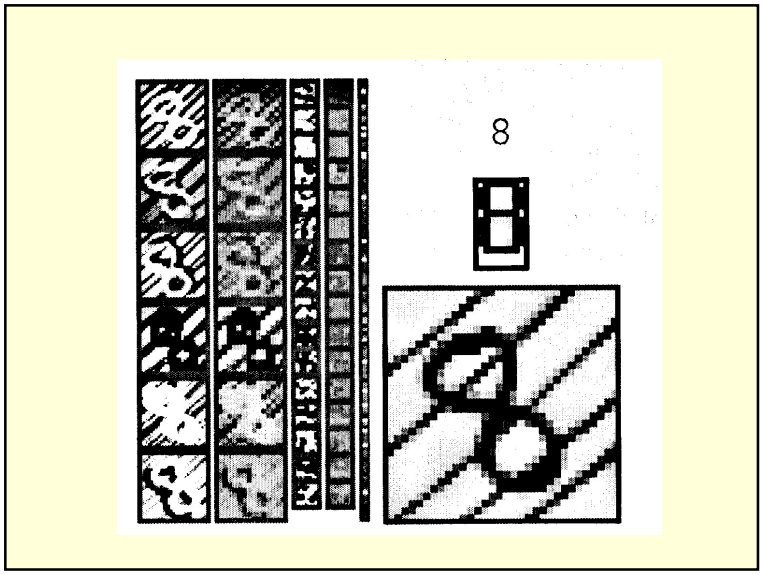
The architecture of LeNet5

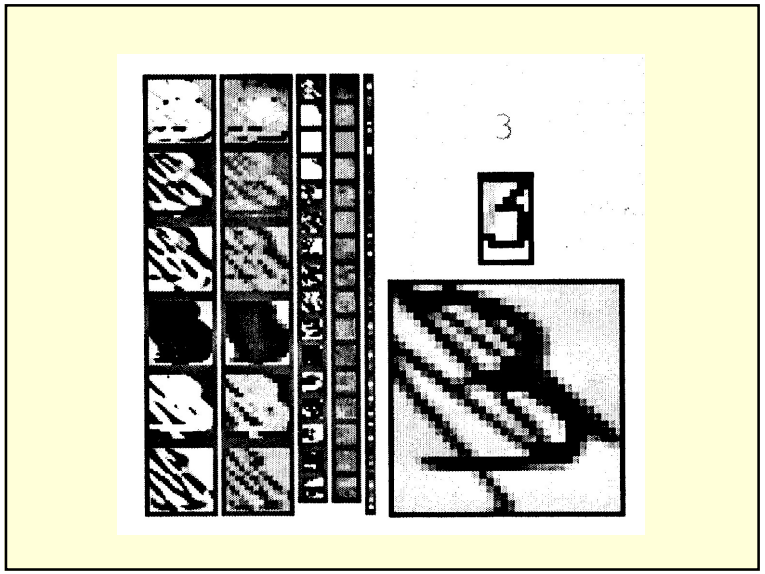





4


4 ← Output
F6


8


3

4

4

3

3

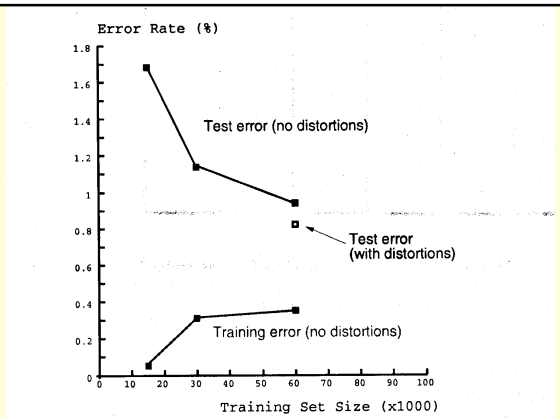


Fig. 6. Training and test errors of LeNet-5 achieved using training sets of various sizes. This graph suggests that a larger training set could improve the performance of LeNet-5. The hollow square show the test error when more training patterns are artificially generated using random distortions. The 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

A brute force approach

- LeNet uses knowledge about the invariances to design:
  - the network architecture
  - or the weight constraints
  - or the types of feature
- But its much simpler to incorporate knowledge of invariances by just creating extra training data:
  - for each training image, produce new training data by applying all of the transformations we want to be insensitive to (Le Net can benefit from this too)
  - Then train a large, dumb net on a fast computer.
  - This works surprisingly well if the transformations are not too big (so do approximate normalization first).

Making dumb backpropagation work really well for recognizing digits

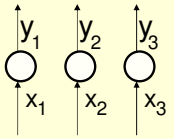
- Using the standard viewing transformations plus local deformation fields to get LOTS of data.
- Use a single hidden layer with very small initial weights:
  - it needs to break symmetry very slowly to find a good local minimum
- Use a more appropriate error measure for multi-class categorization.

Problems with squared error

- The squared error measure has some drawbacks
  - If the desired output is 1 and the actual output is 0.00000001 there is almost no gradient for a logistic unit to fix up the error.
  - If we are trying to assign probabilities to class labels, we know that the outputs should sum to 1, but we are depriving the network of this knowledge.
- Is there a different cost function that is more appropriate and works better?
  - Force the outputs to represent a probability distribution across discrete alternatives.

Softmax

The output units use a non-local non-linearity:



output units

$$y_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

$$\frac{\partial y_i}{\partial x_i} = y_i\,(1 - y_i)$$

The cost function is the negative log prob of the right answer

desired value

$$C = -\sum_j d_j \log y_j$$

The steepness of C exactly balances the flatness of the output non-linearity

$$\frac{\partial C}{\partial x_i} = \sum_j \frac{\partial C}{\partial y_j}\frac{\partial y_j}{\partial x_i} = y_i - d_i$$