

CSC2535: Advanced Machine Learning

Lecture 6a

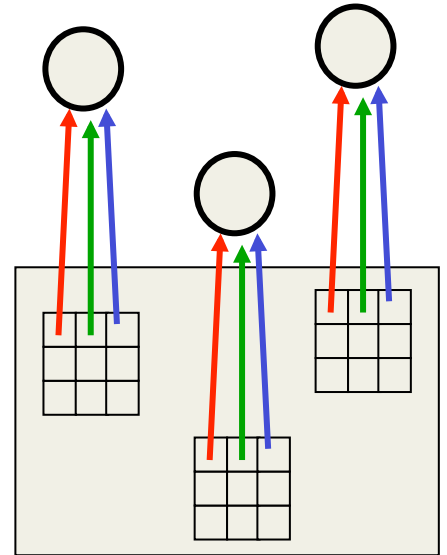
Convolutional neural networks for hand-written digit recognition

Geoffrey Hinton

The replicated feature approach (currently the dominant approach for neural networks)

- Use many different copies of the same feature detector with different positions.
 - Could also replicate across scale and orientation (tricky and expensive)
 - Replication greatly reduces the number of free parameters to be learned.
- Use several different feature types, each with its own map of replicated 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's 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: $w_1 = w_2$

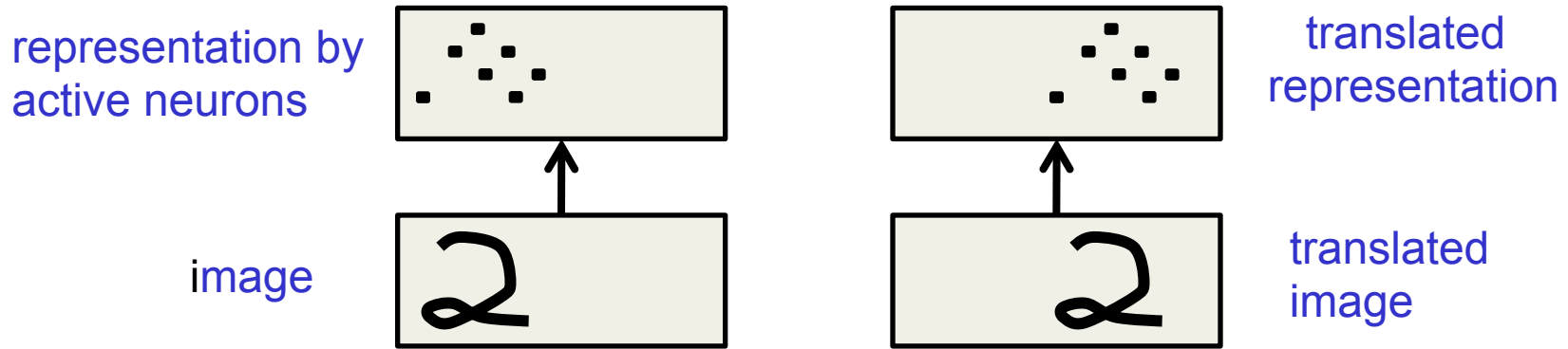
we need: $\Delta w_1 = \Delta w_2$

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

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

What does replicating the feature detectors achieve?

- **Equivariant activities:** Replicated features do **not** make the neural activities invariant to translation. The activities are equivariant.



- **Invariant knowledge:** If a feature is useful in some locations during training, detectors for that feature will be available in all locations during testing.

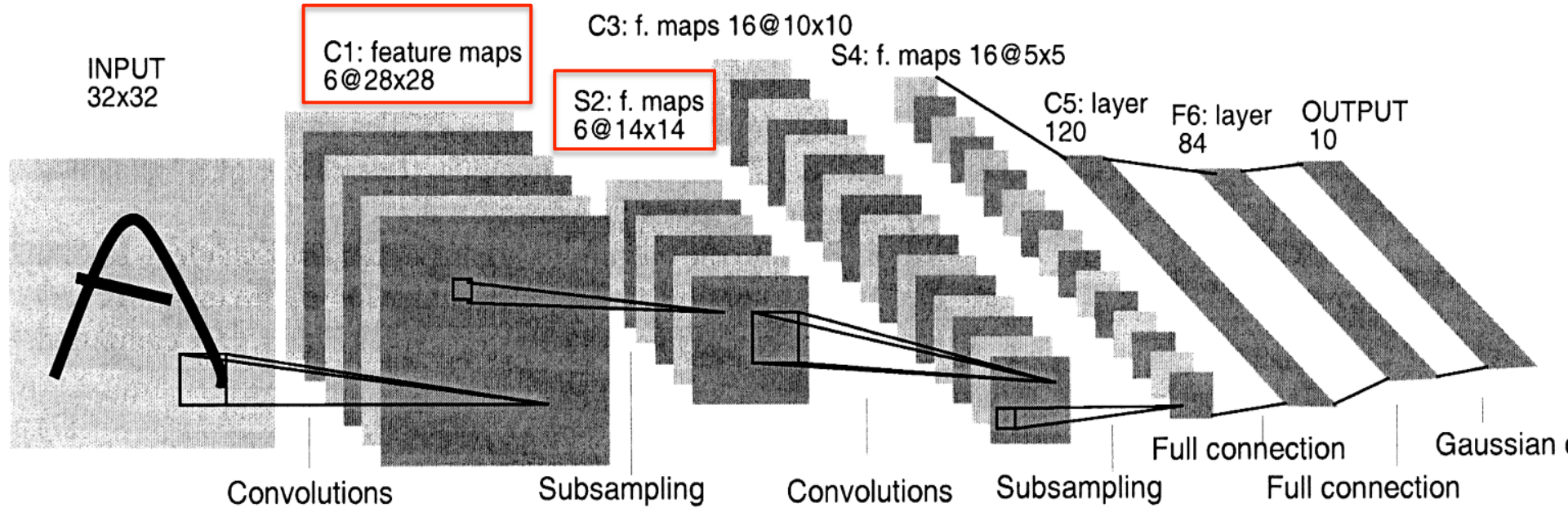
Pooling the outputs of replicated feature detectors

- 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.
 - This reduces the number of inputs to the next layer of feature extraction, thus allowing us to have many more different feature maps.
 - Taking the maximum of the four works slightly better.
- **Problem:** After several levels of pooling, we have lost information about the precise positions of things.
 - This makes it impossible to use the precise spatial relationships between high-level parts for recognition.

Le Net

- Yann LeCun and his collaborators developed a really good recognizer for handwritten digits by using backpropagation in a feedforward net with:
 - Many hidden layers
 - Many maps of replicated units in each layer.
 - Pooling of the outputs of nearby replicated units.
 - A wide net that can cope with several characters at once even if they overlap.
 - A clever way of training a complete system, not just a recognizer.
- This net was used for reading ~10% of the checks in North America.
- Look the impressive demos of LENET at <http://yann.lecun.com>

The architecture of LeNet5



4 4->6	3 3->5	3 8->2	2 2->1	3 5->3	4 4->8	2 2->8	3 3->5	6 6->5	7 7->3
4 9->4	8 8->0	7 7->8	5 5->3	8 8->7	0 0->6	3 3->7	2 2->7	8 8->3	4 9->4
8 8->2	3 5->3	4 4->8	3 3->9	6 6->0	9 9->8	4 4->9	6 6->1	9 9->4	9 9->1
9 9->4	0 2->0	1 6->1	3 3->5	3 3->2	9 9->5	6 6->0	6 6->0	6 6->0	6 6->8
4 4->6	7 7->3	9 9->4	4 4->6	2 2->7	9 9->7	4 4->3	9 9->4	9 9->4	9 9->4
2 8->7	4 4->2	8 8->4	3 3->5	4 8->4	6 6->5	8 8->5	8 3->8	3 3->8	9 9->8
1 1->5	9 9->8	6 6->3	0 0->2	6 6->5	7 9->5	0 0->7	1 1->6	4 4->9	2 2->1
2 2->8	8 8->5	9 4->9	7 7->2	7 7->2	6 6->5	9 9->7	6 6->1	6 5->6	5 5->0
4 4->9	2 2->8								

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 but nobody has had the patience to measure it.




































Priors and Prejudice

- We can put our prior knowledge about the task into the network by designing appropriate:
 - Connectivity.
 - Weight constraints.
 - Neuron activation functions
- This is less intrusive than hand-designing the features.
 - But it still prejudices the network towards the particular way of solving the problem that we had in mind.
- Alternatively, we can use our prior knowledge to create a whole lot more training data.
 - This may require a lot of work (Hofman&Tresp, 1993)
 - It may make learning take much longer.
- It allows optimization to discover clever ways of using the multi-layer network that we did not think of.
 - And we may never fully understand how it does it.

The brute force approach

- LeNet uses knowledge about the invariances to **design**:
 - the local connectivity
 - the weight-sharing
 - the pooling.
- This achieves about 80 errors.
 - This can be reduced to about 40 errors by using many different transformations of the input and other tricks (Ranzato 2008)
- Cirestan *et. al.* (2010) inject knowledge of invariances by creating a huge amount of carefully designed extra training data:
 - For each training image, they produce many new training examples by applying many different transformations.
 - They can then train a large, deep, dumb net on a GPU without much overfitting.
- They achieve about 35 errors.

The errors made by the Ciresan *et. al.* net

 2 17	 1 71	 9 98	 9 59	 9 79	 5 35	 3 23
 4 49	 3 35	 9 97	 4 49	 4 94	 2 02	 3 35
 6 16	 4 94	 0 60	 6 06	 8 86	 1 79	 1 71
 9 49	 0 50	 3 35	 8 98	 9 79	 7 17	 1 61
 2 27	 8 58	 2 78	 6 16	 5 65	 4 94	 0 60

The top printed digit is the right answer. The bottom two printed digits are the network's best two guesses.

The right answer is **almost** always in the top 2 guesses.

With model averaging they can now get about 25 errors.

How to detect a significant drop in the error rate

- Is 30 errors in 10,000 test cases significantly better than 40 errors?
 - It all depends on the particular errors!
 - The McNemar test uses the particular errors and can be much more powerful than a test that just uses the number of errors.

	model 1 wrong	model 1 right
model 2 wrong	29	1
model 2 right	11	9959

	model 1 wrong	model 1 right
model 2 wrong	15	15
model 2 right	25	9945

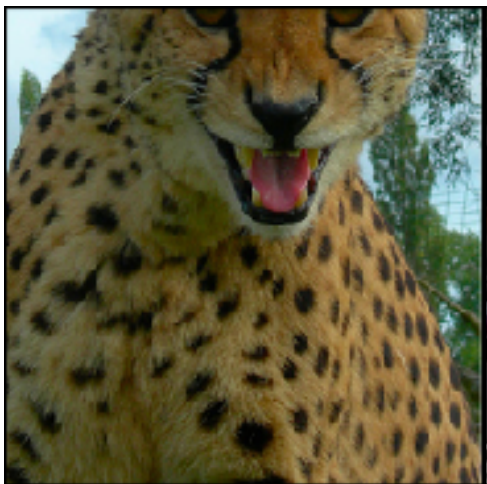
From hand-written digits to 3-D objects

- Recognizing real objects in color photographs downloaded from the web is much more complicated than recognizing hand-written digits:
 - Hundred times as many classes (1000 vs 10)
 - Hundred times as many pixels (256 x 256 color vs 28 x 28 gray)
 - Two dimensional image of three-dimensional scene.
 - Cluttered scenes requiring segmentation
 - Multiple objects in each image.
- Will the same type of convolutional neural network work?

The ILSVRC-2012 competition on ImageNet

- The dataset has 1.2 million high-resolution training images.
- The classification task:
 - Get the “correct” class in your top 5 bets. There are 1000 classes.
- The localization task:
 - For each bet, put a box around the object. Your box must have at least 50% overlap with the correct box.
- Some of the best existing computer vision methods were tried on this dataset by leading computer vision groups from Oxford, INRIA, XRCE, ...
 - Computer vision systems use complicated multi-stage systems.
 - The early stages are typically hand-tuned by optimizing a few parameters.

Examples from the test set (with the network's guesses)



cheetah

cheetah

leopard

snow leopard

Egyptian cat



Jet Train is like a plane, with in-train magazine and a jacket that you can plug your headphones into and listen to

bullet train

bullet train

passenger car

subway train

electric locomotive



hand glass

scissors

hand glass

frying pan

stethoscope

- University of Toronto (Alex Krizhevsky)

• 16.4%

34.1%

Error rates on the ILSVRC-2012 competition

- University of Tokyo
- Oxford University Computer Vision Group
- INRIA (French national research institute in CS) + XRCE (Xerox Research Center Europe)
- University of Amsterdam

classification

classification
& localization

• 26.1%

53.6%

• 26.9%

50.0%

• 27.0%

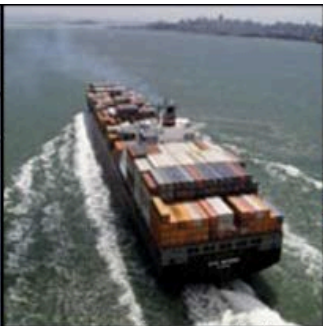
• 29.5%

A neural network for ImageNet

- Alex Krizhevsky (NIPS 2012) developed a very deep convolutional neural net of the type pioneered by Yann Le Cun. Its **architecture** was:
 - 7 hidden layers not counting some max pooling layers.
 - The early layers were convolutional.
 - The last two layers were globally connected.
- The **activation functions** were:
 - Rectified linear units in every hidden layer. These train much faster and are more expressive than logistic units.
 - Competitive normalization to suppress hidden activities when nearby units have stronger activities. This helps with variations in intensity.

Tricks that significantly improve generalization

- Train on random 224x224 patches from the 256x256 images to get more data. Also use left-right reflections of the images.
 - At test time, combine the opinions from ten different patches: The four 224x224 corner patches plus the central 224x224 patch plus the reflections of those five patches.
- Use “dropout” to regularize the weights in the globally connected layers (which contain most of the parameters).
 - Dropout means that half of the hidden units in a layer are randomly removed for each training example.
 - This stops hidden units from relying too much on other hidden units.



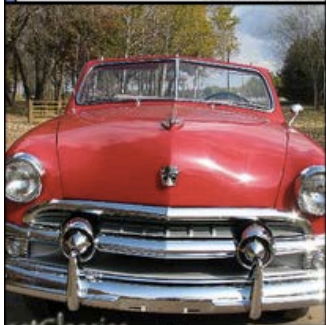
Some more examples of how well the deep net works for object recognition.

mite

container ship

motor scooter

leopard

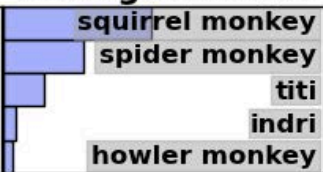
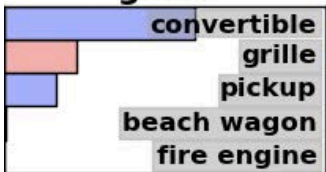


grille

mushroom

cherry

Madagascar cat



The hardware required for Alex's net

- He uses a very efficient implementation of convolutional nets on two Nvidia GTX 580 Graphics Processor Units (over 1000 fast little cores)
 - GPUs are very good for matrix-matrix multiplies.
 - GPUs have very high bandwidth to memory.
 - This allows him to train the network in a week.
 - It also makes it quick to combine results from 10 patches at test time.
- We can spread a network over many cores if we can communicate the states fast enough.
- As cores get cheaper and datasets get bigger, big neural nets will improve faster than old-fashioned (*i.e.* pre Oct 2012) computer vision systems.

Finding roads in high-resolution images

- Vlad Mnih (ICML 2012) used a non-convolutional net with local fields and multiple layers of rectified linear units to find roads in cluttered aerial images.
 - It takes a large image patch and predicts a binary road label for the central 16x16 pixels.
 - There is lots of labeled training data available for this task.
- The task is hard for many reasons:
 - Occlusion by buildings trees and cars.
 - Shadows, Lighting changes
 - Minor viewpoint changes
- The worst problems are incorrect labels:
 - Badly registered maps
 - Arbitrary decisions about what counts as a road.
- Big neural nets trained on big image patches with millions of examples are the only hope.



The best road-finder
on the planet?



Two ways to average models

- MIXTURE: We can combine models by averaging their output probabilities:

Model A: .3 .2 .5

Model B: .1 .8 .1

Combined .2 .5 .3

- PRODUCT: We can combine models by taking the geometric means of their output probabilities:

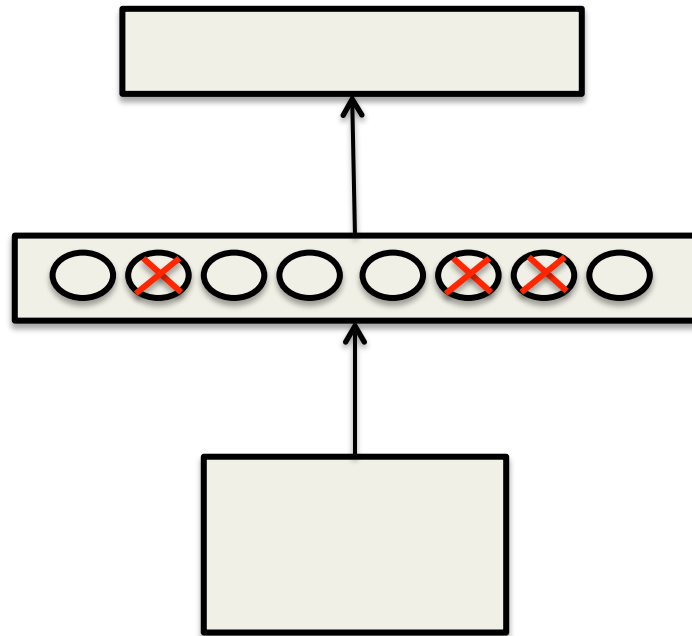
Model A: .3 .2 .5

Model B: .1 .8 .1

Combined $\sqrt{.03}$ $\sqrt{.16}$ $\sqrt{.05}$ /sum

Dropout: An efficient way to average many large neural nets (<http://arxiv.org/abs/1207.0580>)

- Consider a neural net with one hidden layer.
- Each time we present a training example, we randomly omit each hidden unit with probability 0.5.
- So we are randomly sampling from 2^H different architectures.
 - All architectures share weights.



Dropout as a form of model averaging

- We sample from 2^H models. So only a few of the models ever get trained, and they only get one training example.
 - This is as extreme as bagging can get.
- The sharing of the weights means that every model is very strongly regularized.
 - It's a much better regularizer than L2 or L1 penalties that pull the weights towards zero.

But what do we do at test time?

- We could sample many different architectures and take the geometric mean of their output distributions.
- It better to use all of the hidden units, but to halve their outgoing weights.
 - This exactly computes the geometric mean of the predictions of all 2^H models.

What if we have more hidden layers?

- Use dropout of 0.5 in every layer.
- At test time, use the “mean net” that has all the outgoing weights halved.
 - This is not exactly the same as averaging all the separate dropped out models, but it’s a pretty good approximation, and its fast.
- Alternatively, run the stochastic model several times on the same input.
 - This gives us an idea of the uncertainty in the answer.

What about the input layer?

- It helps to use dropout there too, but with a higher probability of keeping an input unit.
 - This trick is already used by the “denoising autoencoders” developed by Pascal Vincent, Hugo Larochelle and Yoshua Bengio.

How well does dropout work?

- The record breaking object recognition net developed by Alex Krizhevsky uses dropout and it helps a lot.
- If your deep neural net is significantly overfitting, dropout will usually reduce the number of errors by a lot.
 - Any net that uses “early stopping” can do better by using dropout (at the cost of taking quite a lot longer to train).
- If your deep neural net is not overfitting you should be using a bigger one!

Another way to think about dropout

- If a hidden unit knows which other hidden units are present, it can co-adapt to them on the training data.
 - But complex co-adaptations are likely to go wrong on new test data.
 - Big, complex conspiracies are not robust.
- If a hidden unit has to work well with combinatorially many sets of co-workers, it is more likely to do something that is individually useful.
 - But it will also tend to do something that is marginally useful given what its co-workers achieve.