CSC 411: Lecture 11: Neural Networks II

Raquel Urtasun & Rich Zemel

University of Toronto

Oct 16, 2015

Today

- Forward propagation
- Backward propagation
- Deep learning

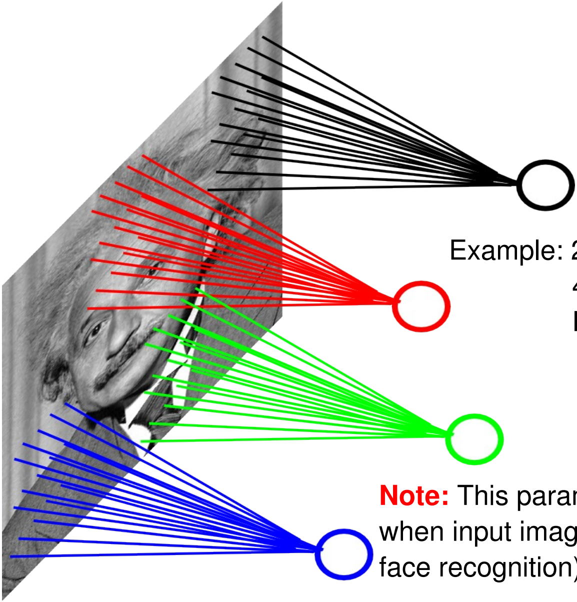
Neural Nets for Object Recognition

- People are very good at recognizing shapes
  - Intrinsically difficult, 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 shape
  - Deformations: Natural shape classes allow variations (faces, letters, chairs)
  - A huge amount of computation is required

How to deal with large Input Spaces

- Images can have millions of pixels, i.e., $\mathbf{x}$ is very high dimensional
- Prohibitive to have fully-connected layer
- We can use a locally connected layer
- This is good when the input is registered

Locally Connected Layer



Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).

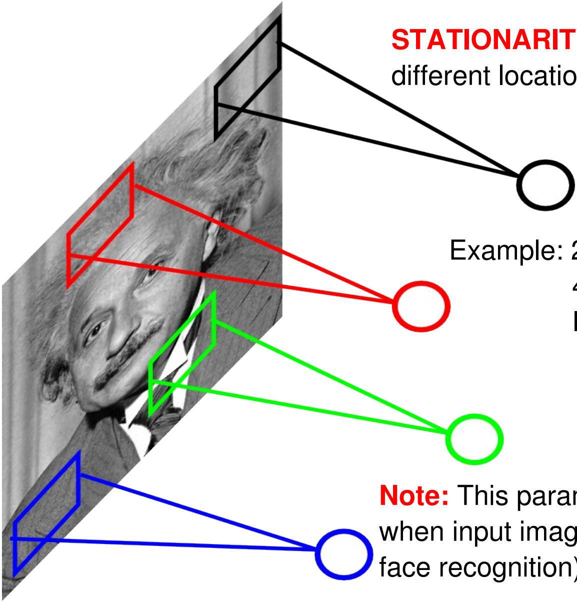
34

Ranzato

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

Locally Connected Layer



STATIONARITY? Statistics is similar at different locations

Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

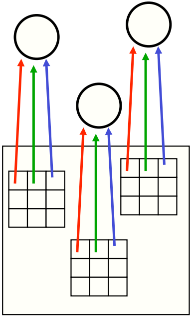
Note: This parameterization is good when input image is registered (e.g., face recognition).

35

Ranzato

The replicated feature approach

The red connections all have the same weight.

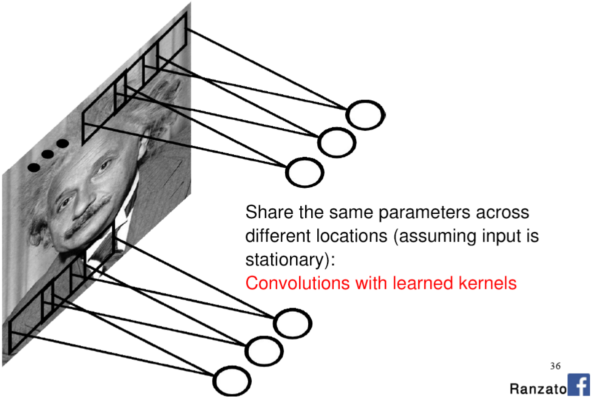


5

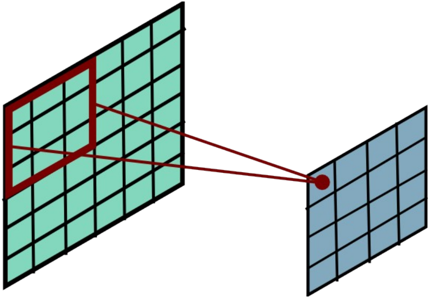
- Adopt approach apparently used in monkey visual systems
- Use many different copies of the same feature detector.
  - Copies have slightly different positions.
  - Could also replicate across scale and orientation.
    - Tricky and expensive
  - Replication reduces number of free parameters to be learned.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

Convolutional Neural Net

- Idea: statistics are similar at different locations (Lecun 1998)
- Connect each hidden unit to a small input patch and share the weight across space
- This is called a convolution layer and the network is a convolutional network



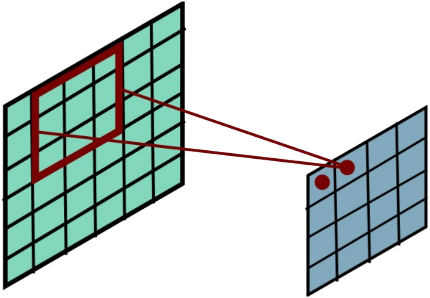
Convolutional Layer



Ranzato

$$h_j^n = \max(0, \sum_{k=1}^{K} h_k^{n-1} * w_{jk}^n)$$

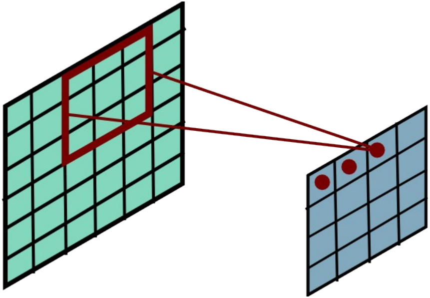
Convolutional Layer



Ranzato

$$h_j^n = \max(0, \sum_{k=1}^{K} h_k^{n-1} * w_{jk}^n)$$

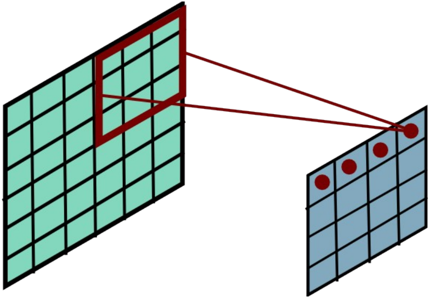
Convolutional Layer



Ranzato

$$h_j^n = \max(0, \sum_{k=1}^{K} h_k^{n-1} * w_{jk}^n)$$

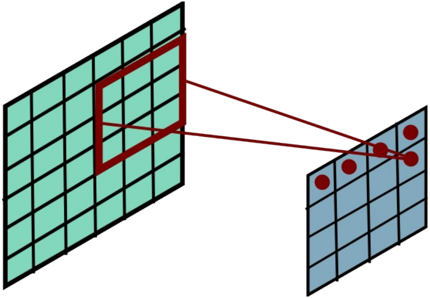
Convolutional Layer



Ranzato

$$h_j^n = \max(0, \sum_{k=1}^{K} h_k^{n-1} * w_{jk}^n)$$

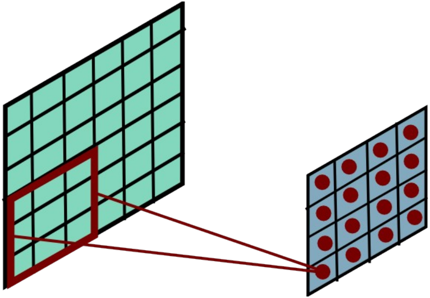
Convolutional Layer



Ranzato

$$h_j^n = \max(0, \sum_{k=1}^{K} h_k^{n-1} * w_{jk}^n)$$

Convolutional Layer



Ranzato

$$h_j^n = \max(0, \sum_{k=1}^{K} h_k^{n-1} * w_{jk}^n)$$

Backpropagation with weight constraints

- It is easy to modify the backpropagation algorithm to incorporate linear constraints between the weights

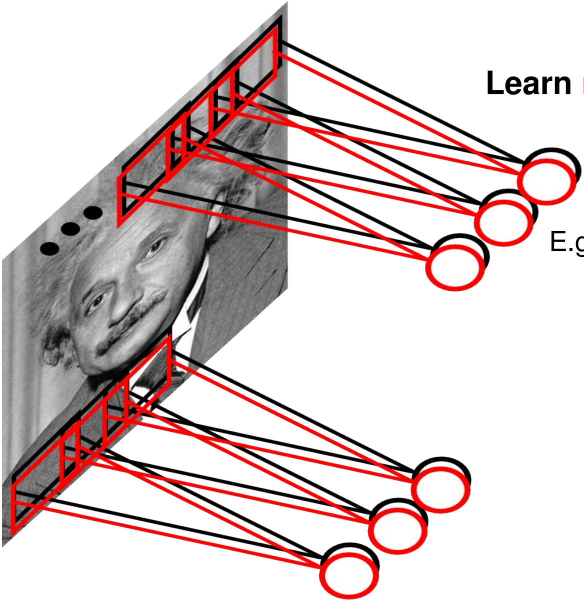
$$\begin{aligned} \text{To constrain:} \quad w_1 &= w_2 \\ \text{we need:} \quad \Delta w_1 &= \Delta w_2 \end{aligned}$$

- We compute the gradients as usual, and then modify the gradients so that they satisfy the constraints.

$$\begin{aligned} \text{compute:} \quad & \frac{\partial E}{\partial w_1} \text{ and } \frac{\partial E}{\partial w_1} \\ \text{use:} \quad & \frac{\partial E}{\partial w_1} + \frac{\partial E}{\partial w_2} \text{ for } w_1 \text{ and } w_2 \end{aligned}$$

- So if the weights started off satisfying the constraints, they will continue to satisfy them.

Convolutional Layer



Learn multiple filters.

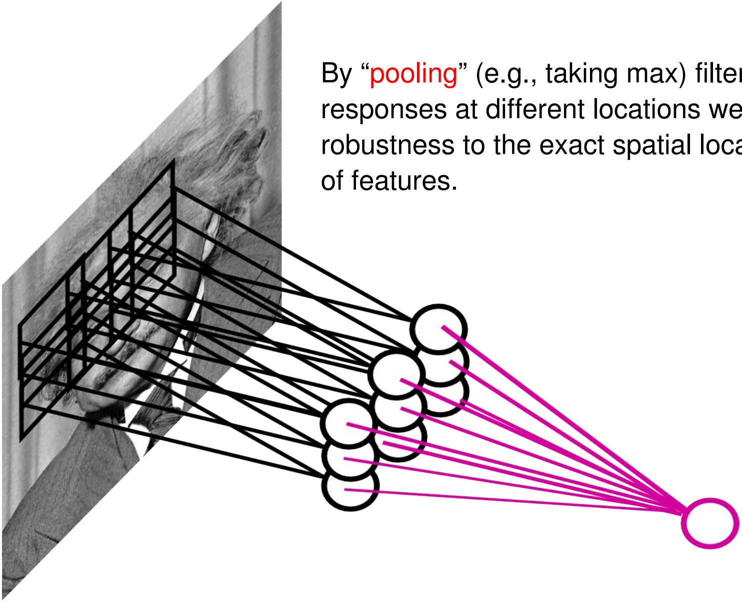
E.g.: 200x200 image
100 Filters
Filter size: 10x10
10K parameters

54

Ranzato

Pooling Layer

By "pooling" (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.



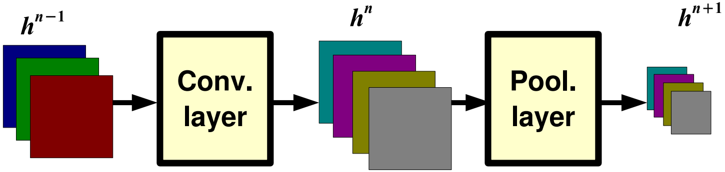
61

Ranzato

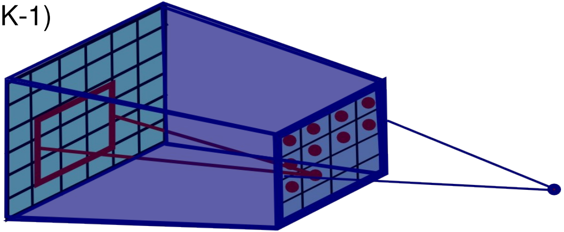
Pooling Options

- Max Pooling: return the maximal argument
- Average Pooling: return the average of the arguments
- Other types of pooling exist.

Pooling Layer: Receptive Field Size



If convolutional filters have size KxK and stride 1, and pooling layer has pools of size PxP, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size: (P+K-1)x(P+K-1)



67

Ranzato

Now let's make this very deep to get a real state-of-the-art object recognition system

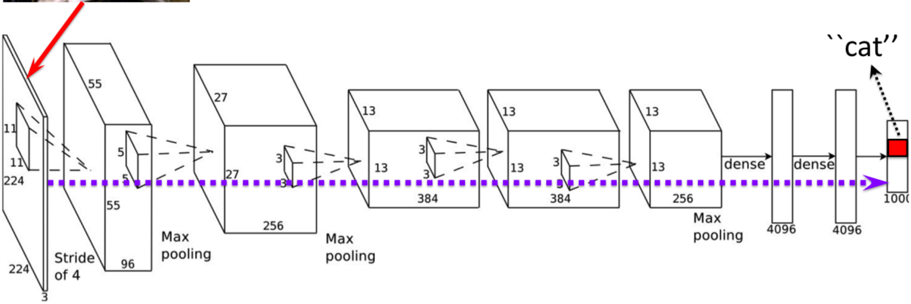
Convolutional Neural Networks (CNN)

- Remember from your image processing / computer vision course about filtering?
- If our filter was $[-1, 1]$, we got a vertical edge detector
- Now imagine we want to have many filters (e.g., vertical, horizontal, corners, one for dots). We will use a filterbank.
- So applying a filterbank to an image yields a cube-like output, a 3D matrix in which each slice is an output of convolution with one filter.
- Do some additional tricks. A popular one is called max pooling. Any idea why you would do this?
- Do some additional tricks. A popular one is called max pooling. Any idea why you would do this? To get invariance to small shifts in position.
- Now add another "layer" of filters. For each filter again do convolution, but this time with the output cube of the previous layer.

Classification

- Once trained we feed in an image or a crop, run through the network, and read out the class with the highest probability in the last (classif) layer.

What's the class of this object?

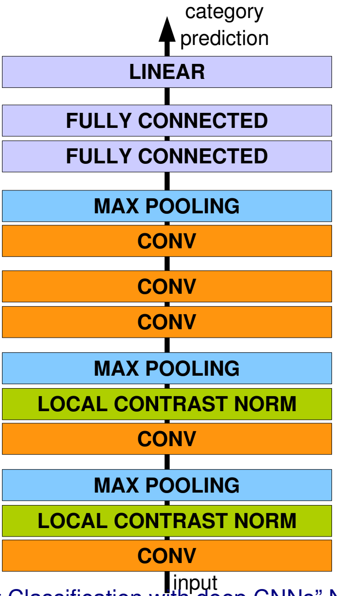


[Slide Credit: Sanja Fidler]

Classification Performance

- Imagenet, main challenge for object classification: `http://image-net.org/`
- 1000 classes, 1.2M training images, 150K for test

Architecture for Classification



Krizhevsky et al. "ImageNet Classification with deep CNNs" NIPS 2012

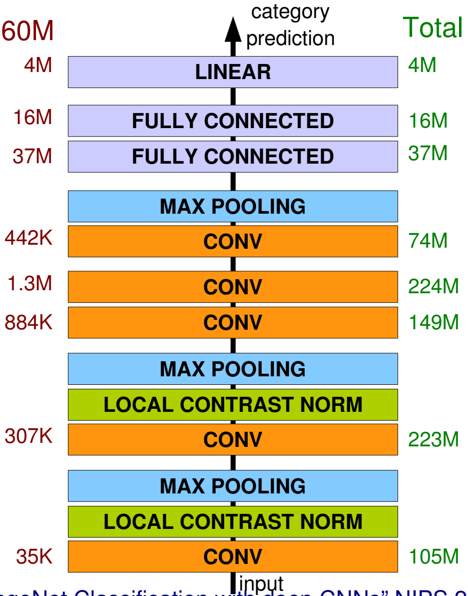
95

Ranzato

Architecture for Classification

Total nr. params: 60M

Total nr. flops: 832M

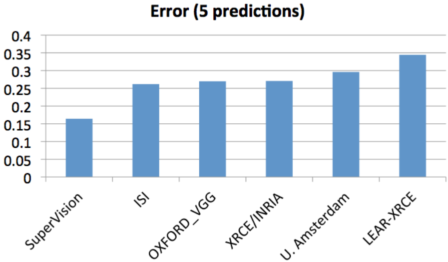


96

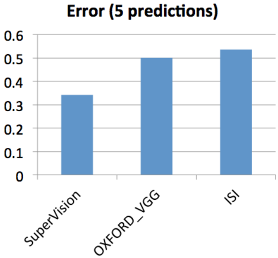
Krizhevsky et al. “ImageNet Classification with deep CNNs” NIPS 2012

Ranzato

The 2012 Computer Vision Crisis



(Classification)



(Detection)

So Neural Networks are Great

- So networks turn out to be great.
- Everything is deep, even if it's shallow!
- Companies leading the competitions as they have more computational power
- At this point Google, Facebook, Microsoft, Baidu "steal" most neural network professors/students from academia

- But to train the networks you need quite a bit of computational power (e.g., GPU farm). So what do you do?

- And train more layers. 16 instead of 7 before. 144 million parameters.
- Buy even more.

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.

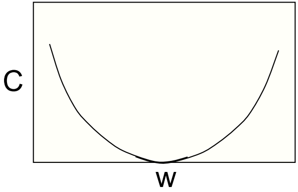
Limiting the size of the weights

- Weight-decay involves adding an extra term to the cost function that penalizes the squared weights.

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

- Keeps weights small unless they have big error derivatives.

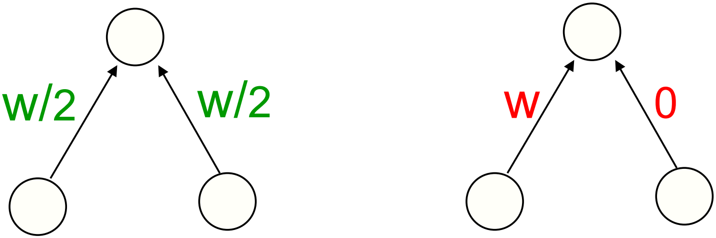
$$\frac{\partial C}{\partial w_i} = \frac{\partial \ell}{\partial w_i} + \lambda w_i$$



$$\text{when } \frac{\partial C}{\partial w_i} = 0, \quad w_i = -\frac{1}{\lambda} \frac{\partial \ell}{\partial w_i}$$

The effect of weight-decay

- It prevents the network from using weights that it does not need
  - This can often improve generalization a lot.
  - It 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.
- But, if the network has two very similar inputs it prefers to put half the weight on each rather than all the weight on one $\rightarrow$ other form of weight decay?



Deciding how much to restrict the capacity

- How do we decide which limit to use and how strong to make the limit?
  - 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 won't do better than chance on another test set.
- So use a separate validation set to do model selection.

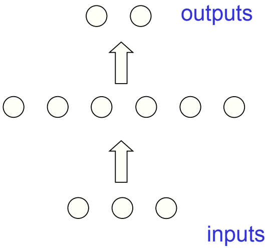
Using a validation set

- Divide the total dataset into three subsets:
  - Training data is used for learning the parameters of the model.
  - Validation data is not used for 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
- 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.