Convolutional Networks

Steve Renals

Machine Learning Practical — MLP Lecture 7
1 November 2017 / 6 November 2017
Recap: Multi-layer network for MNIST

(image from: Michael Nielsen, Neural Networks and Deep Learning,
On MNIST, we can get about 2% error (or even better) using these kind of networks, but

- They ignore the spatial (2-D) structure of the input images – unroll each 28×28 image into a 784-D vector
- Each hidden unit looks at all the units in the layer below, so pixels that are spatially separate are treated the same way as pixels that are adjacent
- There is no simple way for networks to learn the same features (e.g. edges) at different places in the input image
Convolutional networks address these issues through

- **Local receptive fields** in which hidden units are connected to local patches of the layer below,
- **Weight sharing** which enables the construction of feature maps,
- **Pooling** which condenses information from the previous layer.
Fully connected hidden layer – 576 hidden units
Local receptive fields – 24x24 hidden units
Local receptive fields

- Each hidden unit is connected to a small \((m \times m)\) region of the input space – the *local receptive field*
- If we have a \(d \times d\) input space, then we have \((d - m + 1) \times (d - m + 1)\) hidden unit space
- Each hidden unit extracts a feature from “its” region of input space
- Here the receptive field “stride length” is 1, it could be larger
Shared weights

- Constrain each hidden unit $h_{i,j}$ to extract the same feature by sharing weights across the receptive fields.

- For hidden unit $h_{i,j}$

$$h_{i,j} = \text{sigmoid}\left(\sum_{k=0}^{m-1} \sum_{\ell=0}^{m-1} w_{k,\ell} x_{i+k,j+\ell} + b\right)$$

where $w_{k,\ell}$ are elements of the shared $m \times m$ weight matrix $w$, $b$ is the shared bias, and $x_{i+k,j+\ell}$ is the input at $i + k, j + \ell$.

- We use $k$ and $\ell$ to index into the receptive field, whose top left corner is at $x_{i,j}$. 
Shared weights & Receptive Fields

Input 28x28

24x24 Feature Map

h(i,j)
x(i,j)
x(i,j+4)
x(i+4,j+4)
x(i+4,j)
x(i+4,j+4)

k

MLP Lecture 7  Convolutional Networks
Feature Maps

- Local receptive fields with shared weights result in a **feature map**
  - a map showing where the feature corresponding to the shared weight matrix (kernel) occurs in the image
- Feature map encodes **translation invariance**
  - extract the same features irrespective of where an image is located in the input
- **Multiple feature maps**
  - a hidden layer can consist of $F$ different feature maps – in this case $F \times 24 \times 24$ units in total
Feature Maps

Input 28x28

3x24x24 Feature Maps
Weights and Connections

Consider an MNIST hidden layer with feature maps using a 5x5 kernels (resulting in 24x24 feature maps):

- Number of connections per feature map:
  \[ 24 \times 24 \times 5 \times 5 = 14,400 \text{ connections} \]
  \[ 24 \times 24 = 576 \text{ biases} \]

- But since weights are shared within a feature map, we have
  \[ 5 \times 5 = 25 \text{ weights} \]
  1 bias

Consider the case where we have 40 feature maps. We will have

- 1,000 (25x40) weights (+ 40 biases)
- but 576,000 (+ 23,040) connections

In comparison a 100 hidden unit MLP from the first coursework has
\[ 784 \times 100 + 100 = 78,500 \text{ input-hidden weights} \]
Learning image kernels

- Image kernels have been designed and used for feature extraction in image processing (e.g. edge detection)
- However, we can learn multiple kernel functions (feature maps) by optimising the network cost function
- Automating feature engineering

https://en.wikipedia.org/wiki/Kernel_(image_processing)
Convolutional Layer

- This type of feature map is often called a **Convolutional layer**
- We can write the feature map hidden unit equation:

  \[
  h_{i,j} = \text{sigmoid}\left( \sum_{k=1}^{m} \sum_{\ell=1}^{m} w_{k,\ell} x_{i+k,j+\ell} + b \right)
  \]

  \[
  h = \text{sigmoid}(w \otimes x + b)
  \]

  \(\otimes\) is a cross-correlation and is closely related to a **convolution**

- In signal processing a 2D convolution is written as

  \[
  H_{i,j} = \text{sigmoid}\left( \sum_{k=1}^{m} \sum_{\ell=1}^{m} v_{k,\ell} x_{i-k,j-\ell} + b \right)
  \]

  \[
  H = \text{sigmoid}(v \ast x + b)
  \]

- If we “flip” (reflect horizontally and vertically) \(w\) (cross-correlation) then we obtain \(v\) (convolution)
Convolution vs Cross-correlation

- Cross-correlation is often referred to as convolution in deep learning.
- This is not problematic since the specific properties of convolution but not of cross-correlation (commutativity and associativity) are rarely (if ever) required for deep learning.
- In machine learning the network learns the kernel appropriate to its orientation – so if convolution is implemented with a flipped kernel, it will learn that it is a flipped implementation.
- So it is OK to use an efficient (flipped) implementation of convolution for convolutional layers.
Pooling (subsampling)

24x24 Feature Map

12x12 Pooling Layer
Pooling

- Pooling or subsampling takes a feature map and reduces it in size – e.g. by transforming a set of 2x2 regions to a single unit.

- **Pooling functions**
  - Max-pooling – takes the maximum value of the units in the region (c.f. maxout)
  - $L_p$-pooling – take the $L_p$ norm of the units in the region:
    \[ h'_i = \left( \sum_{i \in \text{region}} h_i^p \right)^{1/p} \]
  - Average- / Sum-pooling – takes the average / sum value of the pool

- Information reduction – pooling removes precise location information for a feature

- Apply pooling to each feature map separately
Putting it together – convolutional+pooling layer

Input 28x28

3x24x24 Feature Maps

3x12x12 Pooling Layers
Simple ConvNet:

- Convolutional layer with max-pooling
- Final fully connected hidden layer (no sharing weight)
- Softmax output layer
- With 20 feature maps and a final hidden layer of 100 hidden unit:
  \[20 \times (5 \times 5 + 1) + 20 \times 12 \times 12 \times 100 + 100 + 100 \times 10 + 10 = 289,630\] weights
If we have a colour image, each pixel is defined by 3 RGB values – so our input is in fact 3 images (one R, one G, and one B)

If we want stack convolutional layers, then the second layer needs to take input from all the feature maps in the first layer

**Local receptive fields across multiple input images**

In a second convolutional layer (C2) on top of 20 $12 \times 12$ feature maps, each unit will look at $20 \times 5 \times 5$ input units (combining 20 receptive fields each in the same spatial location)

Typically do not tie weights across feature maps, so each unit in C2 has $20 \times 5 \times 5 = 500$ weights, plus a bias. (Assuming a $5 \times 5$ kernel size)
Stacking convolutional layers

Input 28x28

3x24x24 Feature Maps

3x12x12 Pooling Layers

6x8x8 Feature Maps

6x4x4 Pooling Layers
Example: LeNet5 (LeCun et al, 1997)
MNIST Results (1997)

Fig. 9. Error rate on the test set (%) for various classification methods. [deslant] indicates that the classifier was trained and tested on the deslanted version of the database. [dist] indicates that the training set was augmented with artificially distorted examples. [16x16] indicates that the system used the 16x16 pixel images. The uncertainty in the quoted error rates is about 0.1%.

3) PCA and Polynomial Classifier: Following [53] and [54], a preprocessing stage was constructed which computes the projection of the input pattern on the 40 principal components of the set of training vectors. To compute the principal components, the mean of each input component was first computed and subtracted from the training vectors. The covariance matrix of the resulting vectors was then computed and diagonalized using singular value decomposition. The 40-dimensional feature vector was used as the input of a second degree polynomial classifier. This classifier can be seen as a linear classifier with 821 inputs, preceded by a module that computes all products of pairs of input variables. The error on the regular test set was 3.3%.

4) RBF Network: Following [55], an RBF network was constructed. The first layer was composed of 1000 Gaussian RBF units with 28x28 inputs, and the second layer was a simple 1000 inputs/ten outputs linear classifier. The RBF units were divided into ten groups of 100. Each group of units was trained on all the training examples of one of the ten classes using the adaptive K-means algorithm. The second-layer weights were computed using a regularized pseudoinverse method. The error rate on the regular test set was 3.6%.

5) One-Hidden-Layer Fully Connected Multilayer NN: Another classifier that we tested was a fully connected multilayer NN with two layers of weights (one hidden layer) trained with the version of back-propagation described in Appendix C. Error on the regular test set was 4.7% for a network with 300 hidden units and 4.5% for a network with 1000 hidden units. Using artificial distortions to generate more training data brought only marginal improvement: 3.6% for 300 hidden units and 3.8% for 1000 hidden units. When deslanted images were used, the test error jumped down to 1.6% for a network with 300 hidden units.

It remains somewhat of a mystery that networks with such a large number of free parameters manage to achieve reasonably low testing errors. We conjecture that the dynamics of gradient descent learning in multilayer nets has a "self-regularization" effect. Because the origin of weight space is a saddle point that is attractive in almost every direction, the weights invariably shrink during the first few epochs (recent theoretical analysis seem to confirm this [56]). Small weights cause the sigmoids to operate in the quasi-linear region, making the network essentially equivalent to a low-capacity, single-layer network. As the learning proceeds the weights grow, which
Train convolutional networks with a straightforward but careful application of backprop / SGD

Exercise: prior to the next lecture, write down the gradients for the weights and biases of the feature maps in a convolutional network. Remember to take account of weight sharing.

Next lecture: implementing convolutional networks: how to deal with local receptive fields and tied weights, computing the required gradients...
Convolutional networks include local receptive fields, weight sharing, and pooling leading to:

- Modelling the spatial structure
- Translation invariance
- Local feature detection

Reading:
Michael Nielsen, *Neural Networks and Deep Learning* (ch 6)
http://dx.doi.org/10.1109/5.726791
Ian Goodfellow, Yoshua Bengio & Aaron Courville, *Deep Learning* (ch 9)
http://www.deeplearningbook.org/contents/convnets.html