Neural network programming: convolutional neural networks

Erik Spence

SciNet HPC Consortium

30 April 2019

Today's code and slides

You can get the slides for today's class at the SciNet Education web page.

`https://support.scinet.utoronto.ca/education`

Click on the link for the class, and look under "File Storage" and find the file "CNNs.pdf".

Today's class

This class will cover the following topics:

- Convolutional neural networks
- Feature maps
- Pooling layers
- The latest (and best) versions of our MNIST neural network.

Please ask questions if something isn't clear.

A review of last class

Our story so far:

- We've built a neural network, using Keras, consisting of an input layer, a hidden layer, and an output layer, to classify the MNIST data:
  - We used the tanh function as the activation for the hidden layer.
  - We used the softmax function as the activation for the output ayer.
- We used cross entropy as the cost function, and Stochastic Gradient Descent as the optimization algorithm.
- We trained on the full data set, and achieved an accuracy of 93% on the test data.

We can do better, but it requires a different approach.

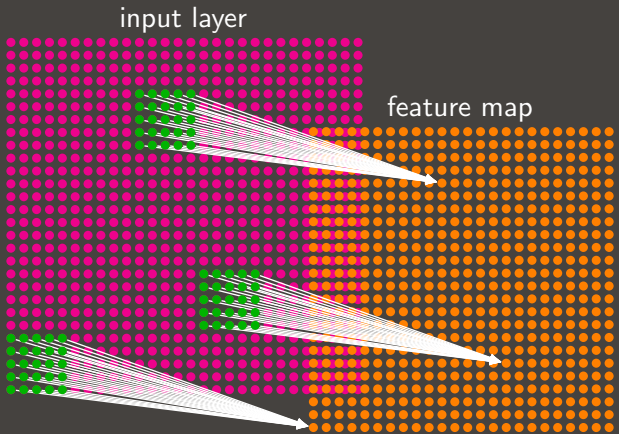
What next?

What we've done so far is pretty good, but it's not going to scale well.

- These are small images, and only black-and-white.
- Imagine we had a more-typical image size (200 x 200) and 3 colours? Now we're up to 120,000 input parameters.
- We need an approach that is more efficient.
- A good place to start would be an approach that doesn't throw away all of the spatial information.
- The data is 28 x 28, not 1 x 784.
- We should redesign our network to account for the spatial information. How do we do that?
- The first step called a Convolutional Layer. This is the bread-and-butter of all neural network image analysis.

Convolutional layers: feature maps

Create a set of neurons that, instead of using all of the data as input, only takes input from a small area of the image. This set of neurons is called a "feature map".



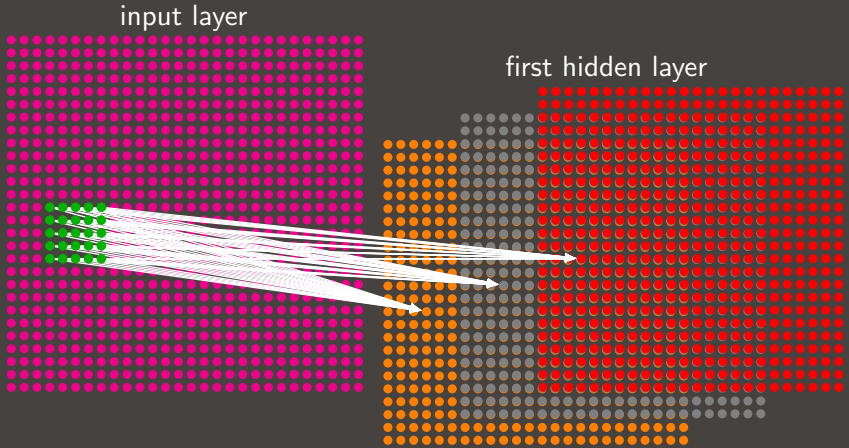
Feature maps

Some notes about feature maps.

- Notice that the feature map is smaller (24 x 24) than the input layer (28 x 28).
- The size of the feature map is partially set by the 'stride', meaning the number of pixels we shift to use as the input to the next neuron. In this case I've used a stride of 1.
- The **weights and biases are shared by all the neurons in the feature map**.
- Why? The goal is to train the feature map to recognize a single feature in the input, regardless of its location in the image.
- Consequently, it makes no sense to have a single feature map as the first hidden layer. Rather, multiple feature maps are used as the first layer.
- Feature maps are also called "filters" and "kernels".

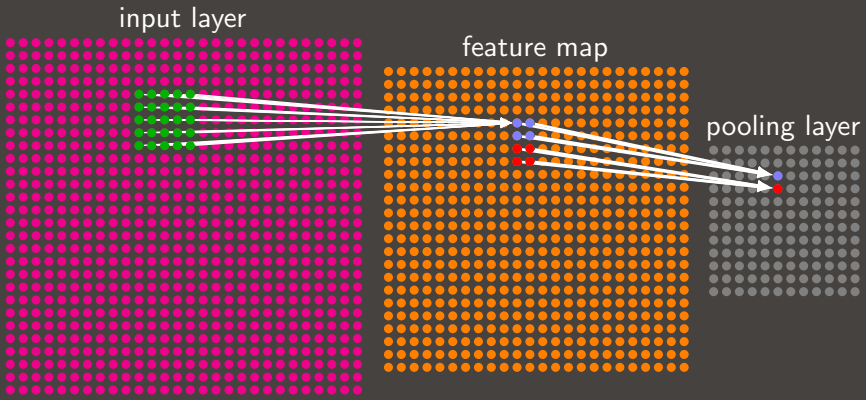
Convolutional layers, continued more

The first hidden layer, a "convolutional layer", consists of multiple feature maps. The same inputs are fed to the neurons in different feature maps.



Pooling layers

Each feature map is often followed by a "pooling layer".



In this case, 2 x 2 feature map neurons are mapped to a single pooling layer neuron.

Pooling layers, continued

Some notes about pooling layers.

- The purpose of a pooling layer is to reduce the size of the data, and thus the number of free parameters in the network.
- The reduction in data also helps with over-fitting.
- Rather than use one of the activation functions we've already discussed, pooling layers use other functions.
- These functions do not have free parameters in them which need to be fit. They are merely functions which operate on the input.
- The most common function used is 'max', simply taking the maximum input value.
- Other functions are sometimes used, average pooling, L2-norm pooling.

2D data formatting

```
In [1]: from keras.datasets import mnist
Using Theano backend.
In [2]: import keras.utils as ku
In [2]:
In [2]: (x_train, y_train), (x_test, y_test) = \
        mnist.load_data()
In [3]:
In [3]: x_train.shape
Out[3]: (60000, 28, 28)
In [4]:
```

Generally, 2D images are actually 3D, to deal with colours. Where the third dimension (the "channels") shows up in the dimensionality is given by the "image_data_format" function.

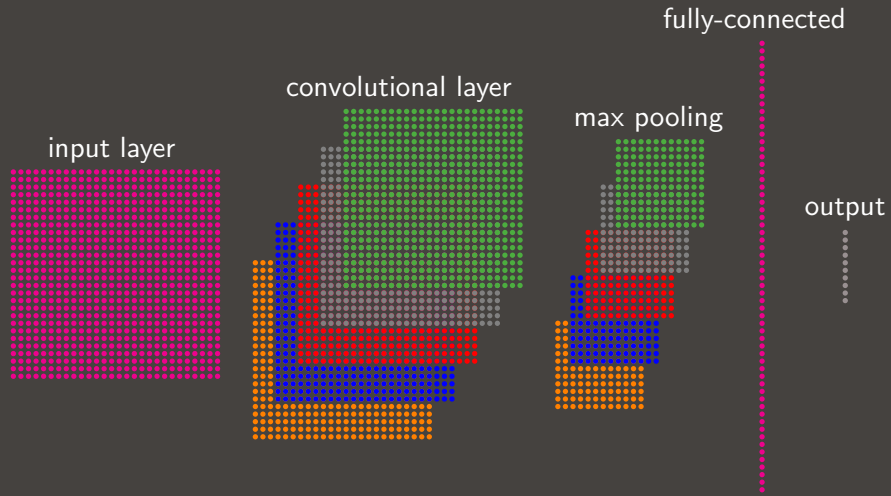
```
In [4]: import keras.backend as K
In [5]: K.image_data_format()
Out[5]: 'channels_last'
In [6]:
In [6]: x_train = \
        x_train.reshape(x_train.shape[0], 28, 28, 1)
In [7]: x_test = \
        x_test.reshape(x_test.shape[0], 28, 28, 1)
In [8]:
In [8]: x_train.shape
Out[8]: (60000, 28, 28, 1)
In [9]:
In [9]: y_train = ku.to_categorical(y_train, 10)
In [10]: y_test = ku.to_categorical(y_test, 10)
In [11]:
```

Backends, an aside

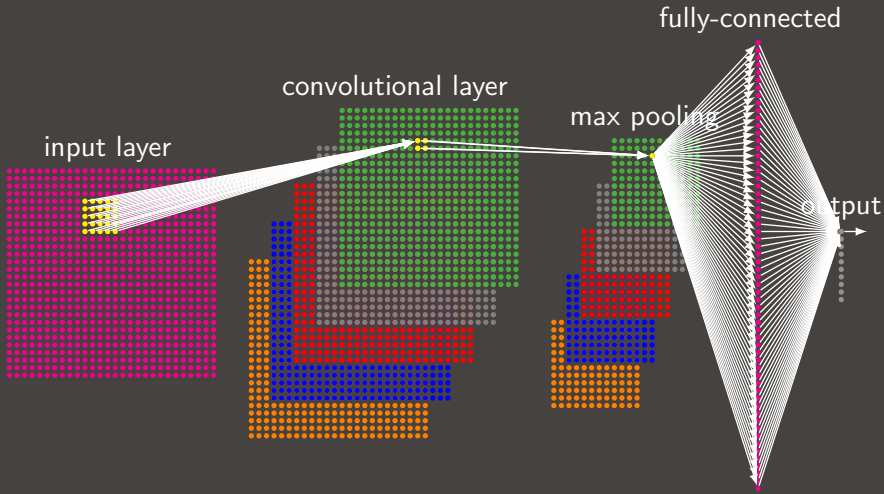
Some notes about the Keras.backend submodule.

- Keras, as you know, uses a standardized API to interface to the 'backend' (Theano, Tensorflow, CNTK, *etc.*).
- The Keras.backend functionality allows you to access the backend functionality directly, but in a backend-independent way.
- Generally speaking, you should stick to the higher-level Keras functionality, but occasionally you need to access Theano or Tensorflow functionality which is not available in Keras.
- In the example on the previous slide, we wanted to know whether the 'channels' (colours) are in the first or last index, so that we know how to reshape the data.
- You should be able to specify whether we are using 'channels_last' or 'channels_first'.

Our network, latest version



Our network, latest version



Our network revisited again

```
# model7.py
import keras.models as km, keras.layers as kl

def get_model(numfm, numnodes):

  model = km.Sequential()
  model.add(kl.Conv2D(numfm, kernel_size = (5, 5),
    input_shape = (28, 28, 1), activation = "relu"))

  model.add(kl.MaxPooling2D(pool_size = (2, 2),
    strides = (2, 2)))

  model.add(kl.Flatten())
  model.add(kl.Dense(numnodes, activation = "tanh"))
  model.add(kl.Dense(10, activation = "softmax"))
  return model
```

```
In [11]:
In [11]: import model7 as m7
In [12]:
In [12]: model = m7.get_model(20, 100)
In [13]:
```

The "Flatten" layer converts the 2D output to 1D, so that the fully-connected layer can handle it.

Our network revisited again, continued

```
In [13]: model.summary()
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 24, 24, 20)        520
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 20)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2880)              0
_________________________________________________________________
dense_1 (Dense)              (None, 100)               288100
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1010
=================================================================
Total params:  289,630
Trainable params:  289,630
Non-trainable params:  0
_________________________________________________________________
```

Our network revisited again, more

```
In [14]:
In [14]: model.compile(loss = "categorical_crossentropy", optimizer = "sgd",
...:           metrics = ['accuracy'])
In [15]:
In [15]: fit = model.fit(x_train, y_train, epochs = 30, batch_size = 100, verbose = 2)
Epoch 1/30
25s - loss:  0.4992 - acc:  0.8638
Epoch 2/30
25s - loss:  0.1973 - acc:  0.9466
⋮
Epoch 30/30
24s - loss:  0.0321 - acc:  0.9911
In [16]:
```

Our network revisited again, some more

Now check against the test data.

98.35%! Only 165 / 10000 wrong!

Not bad!

You can improve this even more by adding another convolutional layer-max pooling layer after the first pair.

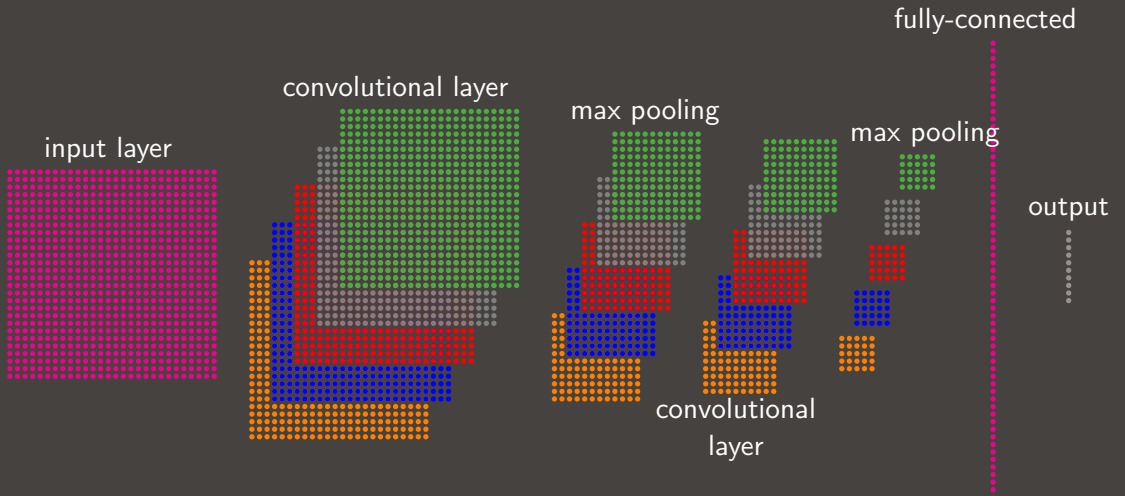
```
In [16]:
In [16]: score = model.evaluate(x_test, y_test)
In [17]:
In [17]: score
Out[17]: [0.053592409740015862, 0.98350000000000004]
In [18]:
```

Notes on Convolutional Networks

The previous network is called a Convolutional Neural Network (CNN), and is quite common in image analysis.

- Often more than a single convolutional layer-pooling layer combination will be used.
- This will lead to improved performance, in this case.
- In practice people come up with all manner of combinations of convolutional, pooling and fully-connected layers in their networks.
- Trial-and-error is a good starting point. Again, hyperparameter optimization techniques should be considered. Reviewing the literature, you will find themes, but also much art.

Our latest network, version 2



Our latest network, version 2, continued

```python
# model8.py
import keras.models as km, keras.layers as kl

def get_model(numfm, numnodes, input_shape = (28, 28, 1)):

  model = km.Sequential()
  model.add(kl.Conv2D(numfm, kernel_size = (5, 5), input_shape = input_shape, activation = "relu"))
  model.add(kl.MaxPooling2D(pool_size = (2, 2), strides = (2, 2)))

  model.add(kl.Conv2D(2 * numfm, kernel_size = (3, 3), activation = "relu"))
  model.add(kl.MaxPooling2D(pool_size = (2, 2), strides = (2, 2)))

  model.add(kl.Flatten())
  model.add(kl.Dense(numnodes, activation = "tanh"))
  model.add(kl.Dense(10, activation = "softmax"))

  return model
```

Our latest network, version 2, summary

```
In [18]: import model8 as m8
In [19]: model = m8.get_model(20, 100)
In [20]:
In [20]: model.summary()
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 24, 24, 20)        520
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 20)        0
conv2d_2 (Conv2D)            (None, 10, 10, 40)        7240
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 40)          0
flatten_1 (Flatten)          (None, 1000)              0
dense_1 (Dense)              (None, 100)               100100
dense_2 (Dense)              (None, 10)                1010
=================================================================
Total params:  108,870
Trainable params:  108,870
Non-trainable params:  0
```

Understanding successive Convolutional Layers

As you noticed, we previously had to change the input data to have dimension (28, 28, 1) rather than (28, 28).

- All convolutional layers assume that its input has a 'channel', which is a third dimension.
- For colour images the three colours of the image (RGB) are the three channels.
- The channel is usually put in the third dimension, though sometimes in the first.
- When the output of one convolutional layer is fed into another layer, the feature maps are the channels.
- How are the channels read by the feature maps?
- If the filter size is, say (3 x 3), and there 20 channels, as in this example, then the number of weights in a given feature map will be (3 x 3) x 20, plus 1 bias.
- Thus, the number of trainable parameters in the second convolutional layer is $(((3 \times 3) \times 20) + 1) \times 40 = 7240$.

Our latest network, version 2, more

```
In [21]:
In [21]: model.compile(loss = "categorical_crossentropy", optimizer = "sgd",
...:           metrics = ['accuracy'])
In [22]:
In [22]: fit = model.fit(x_train, y_train, epochs = 100, batch_size = 100, verbose = 2)
Epoch 1/100
33s - loss:  0.7378 - acc:  0.7966
Epoch 2/100
33s - loss:  0.2010 - acc:  0.9486
⋮
Epoch 99/100
33s - loss:  0.0028 - acc:  0.9996
Epoch 100/100
32s - loss:  0.0028 - acc:  0.9996
In [23]:
```

Our latest network, version 2, even more

Now check against the test data.

99.1%! Only 88 / 10000 wrong! Not bad!

```
In [23]:
In [23]: score = model.evaluate(x_test, y_test)
In [24]:
In [24]: score
Out[24]: [0.028576645734044722, 0.99119999999999997]
In [25]:
```

Using GPUs

An important note. Graphical Processing Units (GPUs) are particularly good at running NN-training calculations.

data size	CPU only		CPU-GPU	
	epoch time	total time	epoch time	total time
50000	41 s	21 min 4 s	4 s	2 min 43s
250000	198 s	100 min	26 s	15 min

These numbers are for the previous network. These were run on a Power 8 CPU, and a P100 GPU.

Multi-GPU functionality is available in Keras running on TensorFlow, though it can be a bit of work to set up.

Deep Learning

What is Deep Learning?

- Quite simply: a neural network with many hidden layers.
- Our last network probably qualified as Deep Learning, though barely.
- Up until the mid-2000s neural network research was dominated by "shallow" networks, networks with only 1 or 2 hidden layers.
- The breakthrough came in discovering that it was practical to train networks with a larger number of hidden layers.
- But it only became practical with the advent of sufficient computing power (GPUs) and easily-accessible huge data sets.
- State-of-the-art networks today can contain dozens of layers.

Most of the rest of this course will involve networks which qualify as Deep Learning.

Linky goodness

Convolutional neural networks:

- http://scs.ryerson.ca/~aharley/vis/conv/flat.html
- http://www.cs.utoronto.ca/~fidler/teaching/2015/CSC2523.html
- https://cs231n.github.io/convolutional-networks
- http://deeplearning.net/tutorial/lenet.html
- https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb