Neural network programming: neural network frameworks

Erik Spence

SciNet HPC Consortium

3 May 2022

SciNet compute • calcul CANADA

Today's code and slides

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

`https://scinet.courses/1210`

Click on the link for the class, and look under "Lectures", click on "Frameworks".

SciNet compute • calcul CANADA

Today's class

This class will cover the following topics:

- Review some of the available neural network frameworks.
- Redo the MNIST example using Keras.
- Introduce Keras' functional syntax.

We will be using Keras for the rest of the course. You will need to install it, by installing Tensorflow. If you have problems, let me know.

SciNet compute • calcul CANADA

A review of last class

Recall what we did last class.

- We built a neural network, consisting of three layers: an input layer, a single hidden layer, and an output layer.
- We defined a cost function, which measured the inaccuracy of the neural network's predictions.
- We used backpropagation to calculate the derivatives of the cost function with respect to the weights and biases.
- Using gradient descent, we trained the network to identify images from the MNIST data set.

However, we built all the parts of the network by hand. There are better ways to do this.

SciNet compute • calcul CANADA

Neural network frameworks

Now that we have a sense of how neural networks work, we're ready to switch gears and use a 'framework'. Why would we do that?

- Coding your own networks from scratch can be a bit of work. (Though it's easier and cleaner if you use classes.)
- Neural network (NN) frameworks have been specifically designed to solve NN problems.
- Python, of course, is not a high-performance language.
- The neural networks which are built using frameworks are compiled before being used, thus being much faster than Python.
- The NN frameworks are also designed to use GPUs, which make things significantly faster than just using CPUs.

The training of neural networks is particularly well suited to GPUs.

SciNet compute • calcul CANADA

Theano

Let's review several of the more-popular NN frameworks out there. First is the Theano programming package.

- Theano is not strictly a NN framework. It was designed to handle multi-dimensional array manipulation.
- Originally developed by the LISA/MILA lab at the Université de Montréal.
- Written in Python; uses a numpy-like syntax, but generates C code which is compiled before being used.
- Supports symbolic differentiation.
- Is the grand-daddy of NN programming approaches.
- Development had been dropped, but has been picked up by PyMC group. Has been renamed Aesara.

Though previously commonly used in NN programming, it's not used any more.

TensorFlow

TensorFlow is Google's NN framework.

- Released as open source in November 2015.
- The second-generation machine-learning framework developed internally at Google, successor to DistBelief.
- More flexible than some other neural network frameworks.
- Capable of running on multiple cores and GPUs.
- Provides APIs for Python, C++, Java and other languages.
- Used to be quite a challenge to learn (many ways to do the same thing).
- With Tensorflow 2.0, Keras has become the main high-level API. A significant consolidation of the API was performed.

This framework is popular, though not necessarily the fastest.

SciNet compute • calcul CANADA

PyTorch

Another framework used for neural networks is PyTorch.

- Based on Torch, which was first released in 2002. Quite mature at this point.
- PyTorch was released by Facebook in January 2018. This is now the most-commonly used interface to Torch, though there is also a C++ interface.
- Like Theano, PyTorch is more flexible than just NN. It is more of a generic scientific computing framework.
- Very strong on GPUs.
- Very fast. Often the fastest depending on the problem being considered.
- Used and maintained by Facebook, Twitter and other high-profile companies.
- PyTorch Lightning was released recently. This gives a more Keras-like interface to PyTorch, making it easier to use.

This framework is the other major player, other than Tensorflow.

SciNet compute • calcul CANADA

Other frameworks

There are other frameworks which you may run into.

- Caffe, previously the leader for image analysis.
- Microsoft Cognitive Toolkit (previously CNTK).
- Apache MXNet, used by Amazon for cloud deep learning.
- Deeplearning4j, a Java-based NN framework.
- PaddlePaddle, Baidu's NN framework.

And of course, many others.

SciNet compute • calcul CANADA

Keras

We will use Keras for the rest of this course.

- Keras is a NN framework, but it's only the top-most level.
- More accurately, it's an API standard for creating neural networks.
- Designed for fast development of networks.
- The original version ran on top of a 'back end', which by default is now TensorFlow, as Keras is being absorbed into TensorFlow.
- Historically it ran on top of many other backends also: Theano, CNTK, MXNet, TypeScript, JavaScript, PlaidML, Scala, CoreML, and others.
- Because it's a proper framework, all of the NN goodies you need are already built into it.
- Because the recommended way is to use Keras through Tensorflow, that is the way we will be using it.

SciNet compute • calcul CANADA

Getting the data

Because it is so commonly used, the MNIST data set is built into most NN frameworks.

Keras is no different, so we'll just grab it from Keras directly.

```
In [1]:
In [1]: from tensorflow.keras.datasets import mnist
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]:
```

Prepping the data

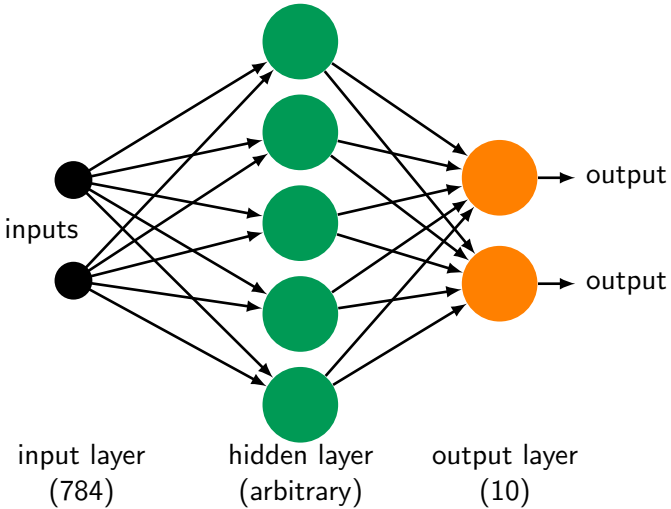
As with last time, we need the data in a specific format:

- Instead of 28 x 28, we flatten the data into a 784-element vector.
- We only take the first 500 data points for training, to be consistent with last class.
- The labels must be changed to a categorical format (one-hot encoding).

```
In [4]:
In [4]: import tensorflow.keras.utils as ku
In [5]:
In [5]: x_train = x_train[0:500, :, :].reshape(500, 784)
In [6]: x_test = x_test[0:100, :, :].reshape(100, 784)
In [7]:
In [7]: y_train = ku.to_categorical(y_train[0:500], 10)
In [8]: y_test = ku.to_categorical(y_test[0:100], 10)
In [9]:
In [9]: y_train.shape
Out[9]: (500, 10)
In [10]:
```

SciNet compute • calcul CANADA

Our neural network



Our network using Keras

Let us re-implement our second network using Keras.

- A "Sequential" model means the layers are stacked on one another in a linear fashion.
- A "Dense" ("fully-connected") layer is the regular layer we've been using.
- Use "input_dim" in the first layer to indicate the shape of the incoming data.
- The "activation" is the output function of the neuron.
- The "name" of the layer is optional.

```
# model1.py
import tensorflow.keras.models as km
import tensorflow.keras.layers as kl

def get_model(numnodes):
  model = km.Sequential()
  model.add(kl.Dense(numnodes, input_dim = 784,
    activation = 'sigmoid', name = 'hidden'))
  model.add(kl.Dense(10, name = 'output',
    activation = 'sigmoid'))
  return model
```

```
In [10]: import model1 as m1
In [11]: model = m1.get_model(30)
In [12]: model.output_shape
Out[12]: (None, 10)
In [13]:
```

Our network using Keras, continued

```
In [13]:
```

```
In [13]: model.summary()
Layer (type)                 Output Shape              Param #
=================================================================
hidden (Dense)               (None, 30)                23550
_________________________________________________________________
output (Dense)               (None, 10)                310
=================================================================
Total params:  23,860
Trainable params:  23,860
Non-trainable params:  0
_________________________________________________________________
```

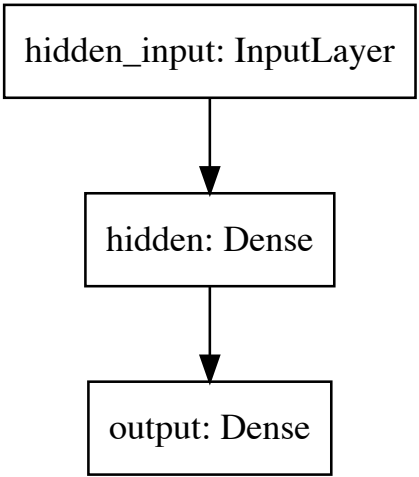
```
In [14]:
```

Plotting the model

Keras allows you the ability to plot your model. This is sometimes helpful if your model gets complicated.

Note that you may need to install the 'pydot' package ("pip install pydot") and the graphviz library for this to work.

```
In [14]:
In [14]: ku.plot_model(model,
                       to_file = 'mymodel.pdf')
In [15]:
```



Our network using Keras, continued more

```
In [15]:
In [15]: model.compile(optimizer = 'sgd', metrics = ['accuracy'], loss = "mean_squared_error")
In [16]:
In [16]: fit = model.fit(x_train, y_train, epochs = 1000, batch_size = 5, verbose = 2)
Epoch 1/1000
0s - loss: 0.1963 - acc: 0.1170
Epoch 2/1000
0s - loss: 0.1338 - acc: 0.1720
⋮
Epoch 999/1000
0s - loss: 0.0394 - acc: 0.8440
Epoch 1000/1000
0s - loss: 0.0394 - acc: 0.8440
In [17]:
```

SciNet compute • calcul CANADA

About that optimization flag

The optimization flag was set to "sgd".

- This stands for "Stochastic Gradient Descent".
- This is similar to regular gradient descent that we used previously.
  - Regular gradient descent is ridiculously slow on large amounts of data.
  - To speed things up, SGD uses a randomly-selected subset of the data (a "batch") to update the weights and biases.
  - This is repeated many times, using different batches, until all of the data has been used. This is called an "epoch".
- In practise, regular gradient descent is never used, stochastic gradient descent is used instead, since it's so much faster.
- The only real advantage of regular gradient descent is that it's easier to code, which is why I used it in previous classes.
- There are many variations on SGD that are also used.

SciNet compute • calcul CANADA

Our network using Keras, notes

Some notes about the compilation of the model.

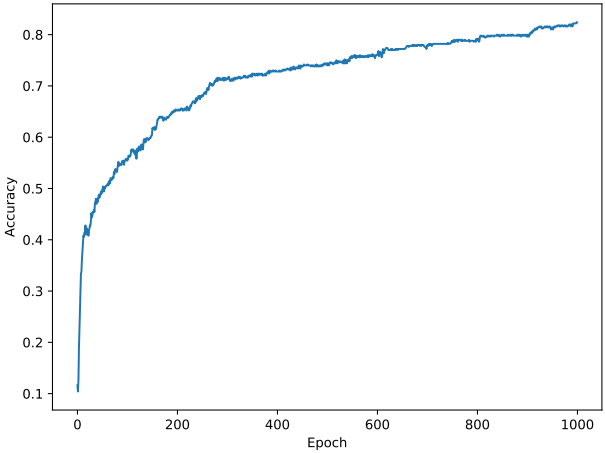
- We must specify the loss (cost) function with the "loss" argument.
- We must specify the optimization algorithm, using the "optimizer" flag.
- The optimizer can be generic ('sgd'), as in this example, or you can specify parameters using the optimizers in the keras.optimizers module.
- I sometimes specify the optimizer explicitly so that I can specify the value of η (using 'lr', the 'learning rate').
- The 'metrics' argument is optional, but is needed if you want the accuracy to be printed.

SciNet compute • calcul CANADA

Plotting the training

The fit.history dictionary contains useful information about the training. Sometimes plotting can give you some insight into the quality of the training, and whether or not it's finished.

```
In [17]:
In [17]: import matplotlib.pyplot as plt
In [18]: plt.plot(fit.history['accuracy'])
In [19]: plt.xlabel('Epoch')
In [20]: plt.ylabel('Accuracy')
In [21]:
```



Our network using Keras, continued even more

Now check against the test data.

We see the over-fitting rearing its head (84% versus 62%).

We can do better! That will be the goal of next class.

```
In [21]:
In [22]: score = model.evaluate(x_test, y_test)
In [23]:
In [23]: score
Out[23]: [0.056402873396873471, 0.62]
In [24]:
```

Our network using Keras, different syntax

```python
# model1.py
import tensorflow.keras.models as km
import tensorflow.keras.layers as kl

def get_model(numnodes):

  model = km.Sequential()

  model.add(kl.Dense(numnodes,
    input_dim = 784, , name = 'hidden',
    activation = 'sigmoid'))

  model.add(kl.Dense(10, name = 'output',
    activation = 'sigmoid'))

  return model
```

Keras has two network-building syntaxes.

```python
# model2.py
import tensorflow.keras.models as km
import tensorflow.keras.layers as kl

def get_model(numnodes):

  input_image = kl.Input(shape = (784,),
    name = 'input')

  x = kl.Dense(numnodes, name = 'hidden',
    activation = 'sigmoid')(input_image)

  x = kl.Dense(10, name = 'output',
    activation = 'sigmoid')(x)

  model = km.Model(inputs = input_image, outputs = x)

  return model
```

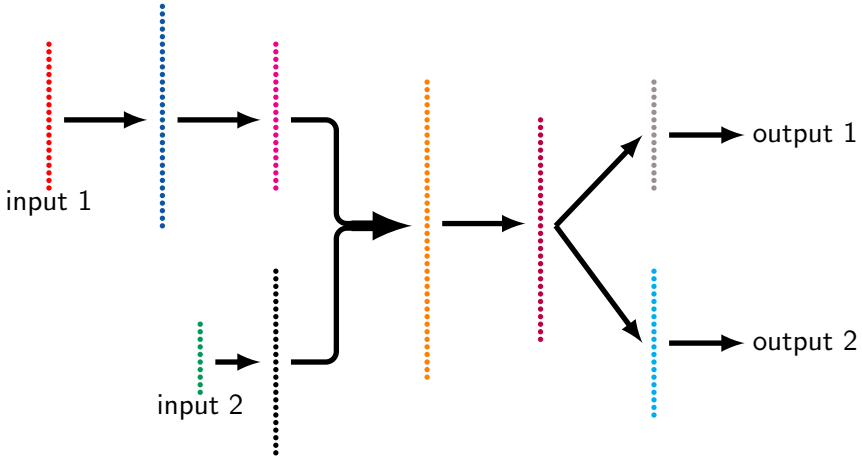
Keras' functional syntax, continued

The second syntax is known as the 'functional' syntax. Why would such a thing be available?

- The original syntax, using model = km.Sequential(), works fine assuming you have a single input, a single output, and all the layers lay in a sequence.
- But what do you do if you have multiple inputs or outputs?
- In a later class, we'll do an example of a network which takes two inputs: input data, and a requested number. The sequential networks can't handle this sort of input without combining the input together at the input stage, which may not make sense.
- The functional syntax allows you to create networks which have multiple inputs and outputs.
- It also gives you the ability to combine the output of layers within the network.
- Options for combining layer outputs include concatenating, multipling/dividing, adding/subtracting, etc.

SciNet compute • calcul CANADA

Keras' functional syntax



Training with multiple outputs: an aside

As you might imagine, if you have multiple outputs the training of your network is going to get complicated. There are several things that must be done to train such networks.

- The first is to specify multiple loss functions, one for each output. This is done by putting a list of loss functions into your compile command (loss = ['mean_squared_error', 'categorical_crossentropy'])
- You can also scale how much emphasis to put on one output versus another, using the 'loss_weights' argument to the compile function (loss_weights = [1.0, 0.2]).
- You can also define your own custom loss functions. These take two arguments (y_true and y_pred), and return the loss value.

There is lots of flexibility available for setting up the loss functions. We will use this functionality later in the course.

SciNet compute • calcul CANADA

The next steps

We can do better. What's the plan? There are a few simple approaches:

- Use more data.
- Change the activation function.
- Change the cost function.
- Change the optimization algorithm.
- Change the way things are initialized.
- Add regularization, to try to deal with the over-fitting.

We'll try some of these next class, but there are also some not-so-simple approaches:

- Completely overhaul the network strategy.

We'll take a look at this next week.

SciNet compute • calcul CANADA

The next steps, an aside

There are alot of things we can tweak to make the network do better on the testing data. How do we know what to do?

- In many ways, implementing a network is an art.
- Certain forms and functions and parameters are known to lead to certain types of behaviour, and thus are used in certain situations.
- Choosing the correct values of parameters can often seem like a matter of trial-and-error.
- And choosing the correct activation functions, number of nodes, can also seem like trial-and-error.
- But there are more-sophisticated ways of finding the optimum parameter choices. We may discuss this in a later class.

Practice is often needed to know how to approach various types of problems. Consult your colleagues, and the literature.

Linky goodness

Keras:

- `https://keras.io`

Other frameworks:

- `https://github.com/Theano/Theano`
- `https://github.com/pymc-devs/aesara`
- `https://www.tensorflow.org`
- `https://pytorch.org`
- `http://torch.ch`
- `http://caffe.berkeleyvision.org`

SciNet compute • calcul CANADA