Solutions 3

Jumping Rivers

The fashion MNIST data is similar to the MNIST digits but instead of images of digits it is made up of images of 10 fashion items. Otherwise it's format is similar. Each image consists of a 28 by 28 pixel, single colour channel image. For the purposes of our example we will treat each of the 784 pixels as independent inputs. There are 60,000 training images and 10,000 test images. The *jrpytorch* package provides a function for downloading the data (the first time) and preprocessing it into an appropriate numpy array structure as below

```
import jrpytorch

X_train,X_test,y_train,y_test,labels = jrpytorch.datasets.load_fashion_mnist('fashiondata',True)
```

For this example we will consider optimiser performance. For this purpose it is not necessary to consider testing loss. Whilst test loss is the better metric of how well our model is doing it is affected by things like overfitting. Since we want to get a feel for how the different optimisers work training loss is a better bet since this is what we ask our optimiser to optimise.

- Create a model similar to that we used in the notes for the MNIST digits example.

```
import torch.nn as nn
class mlp(nn.Module):
  def __init__(self):
    super(mlp,self).__init__()
    self.lin1 = nn.Linear(784,20)
    self.lin2 = nn.Linear(20,10)

  def forward(self,x):
    x = torch.relu(self.lin1(x))
    return self.lin2(x)
```

In the notes we were explicit with returning softmax activation on the final layer as we introduced the topic. However this is not necessary for calculating loss as the `torch.nn.CrossEntropyLoss()` function will do that for us. Given that your forward pass of your created model returns a 1 by 10 (1 observation of 10 class values) for each individual prediction you can train the model using `jrpytorch.train_mnist_example()` function, for example.

```
import torch
```

```
model = mlp()
optimiser = torch.optim.SGD(model.parameters(), lr=0.1)
output = jrpytorch.train_mnist_example(model,optimiser,1000,X_train,y_train)
```

- Try different optimisers and varying the hyperparameters. How do these affect the model fitting process? Remember to recreate your model for each run as this stores the weights as internal state.
- Try saving and reloading your model and an optimiser part way through training