Practical 4

Jumping Rivers

A

Predict a person based on accelerometer data from walking

We have accelerometer data on 15 individuals who all walked for a period of time. Each observation is a set of values from the x,y and z axis of an accelerometer in 5 seconds sampled at a frequency of 52Hz. One sample is 260 observations. We have between 300-600 observations on each individual

```
import jrpytorch

walking = jrpytorch.datasets.load_walking()
```

The following code will produce a visualisation of one sample

```
import matplotlib.pyplot as plt
import numpy as np

sub = walking[walking['sample'] == 400]
sub.loc[:,'time'] = np.arange(260)
fig, (ax1,ax2,ax3) = plt.subplots(1,3, figsize = (18,10))
sub.plot(x = 'time', y = 'acc_x', ax = ax1)
sub.plot(x = 'time', y = 'acc_y', ax = ax2)
sub.plot(x = 'time', y = 'acc_z', ax = ax3)
plt.show()
```

At present this data is not in a particularly convenient structure for training my model. The following code will create the (n,260,3) (n observations of 260 inputs for 3 channels) input shape and class labels as seperate array objects

```
dims = ['acc_x','acc_y','acc_z']
x = np.dstack([walking[[d]].values.reshape(-1,260) for d in dims])
y = walking['person'].values[::260] - 1
```

Each observation is a sequence of 260 values with 3 channels. This is the sort of data structure where we would use a convolutional neural network with 1d convolutions.

Create a model structure that takes in the 260 input features for each of 3 data channels and returns 15 output features (one for each person). We will want a set of convolution and pooling layers to begin with before having some linear layers to get to the final output. The

convolutions and pooling extract features from the 3 channels of sequences, the linear layers then map those features to the final output.

partition your data into training and test sets

pytorch expects a tensor shape of (samples, channels, inputdim) for convolutional nets. we can transpose the dimensions of our data to match this

```
X_train = X_train.transpose((0,2,1))
X_test = X_test.transpose((0,2,1))
```

Create a data loader to create batches from this data for training (Hint: `torch.utils.data.TensorDataset` and `torch.utils.data.DataLoader`)

Train your model using the training set, if you want to you could keep track of the loss for the test set too.

For me this gives 95% plus accuracy on classifying a person purely on the accelerometer data while walking. Pretty neat!

B

Transfer learning

Some images of dogs and cats are available on github `https://github.com/jamieRowen/jrpytorch_data`

Download the images and use a pre trained model as a fixed feature extractor on this data. Try using data augmentation to obtain a model that does well for predicting objects from the images.