CS-345/M45 Lab Class 4 **Release date: 14/03/2022**
Total Marks: 4 **Due date: 28/03/2022**

Support Vector Machines, Neural Networks, and Convolutional Neural Networks

This lab is about utilizing Support Vector Machines, Neural Networks, and Convolutional Neural Networks for classification. We will be looking at applications of the approaches to both the previously seen Wine dataset, and also a new dataset: MNIST Hand-written digits.

There are 4 marked tasks in this lab; implementing SVMs, Neural Networks, and Convolutional Neural Networks. Task 4.1 and 4.2 compare SVMs and neural networks for multi-class prediction on multivariate data. Task 4.3 and 4.4 compare the use of neural networks and convolutional neural networks on the image classification task.

For tasks 4.1 and 4.2, we will be using the Wine dataset. You should re-use the dataset provided in the previous lab. Links to the dataset are provided on the Canvas page. For tasks 4.3 and 4.4, we will use the MNIST hand-written digit recognition dataset. We will download this set using Tensorflow. A description of the data, and how to download it, are provided below.

In order to complete this lab you will need to install the following additional Python packages into your virtual environment: `tensorflow`

Note: Building and developing deep learning models is a big step up from just calling a constructor seen in the previous few labs. Building these neural networks requires understanding how the layers fit together, understanding the shape of the data, the outputs, and the purpose of each layer type. There are also additional considerations to take into account; including the role of optimisers, activation functions, layer hyperparameters, and metrics. To this end, this lab is a bit more involved. Watch the tutorial video, check out the Tensorflow API and really strive to understand what is happening, and what can be tweaked.

After describing the lab tasks, there is also a section at the end of this lab sheet which describes how the Tensorflow/Keras framework can be used. It is by no means exhaustive, but should give you a head-start in completing the lab.

You are also provided with an example Jupyter notebook which implements a fully-connected neural network model on the Fisher Iris dataset. This notebook looks at defining a network of Dense layers, with 2 hidden layers and an output layer. It also defines the hyperparameters, optimisers and metrics used to train the model. It then trains a model and uses it to predict on the test set. It then plots the loss and accuracy curves, to allow us to gain more insight into our model's training.

Tasks Using the Wine Dataset

The first two tasks look to classify a given sample into one of three different classes. The wines all come from the same region of Italy, but have been produced by one of three different cultivators. The data includes a 13-dimensional multivariate chemical analysis of the wine, including things like the acidity and alcohol content. The labels are a numerical ID of which cultivar created the wine (0, 1, 2). The following two tasks will utilise this 13-dimensional space to train models which predict the label.

☐ Task 4.1 – Multiclass SVM for Wine Data

This first task involves using a Support Vector Machine to predict classes labels on the Wine dataset.

Your task here is to create, train and predict using an instance of the `sklearn.svm.SVC` object. sklearn has many Support Vector Machine algorithms implemented, however we are interested in the classification task here, so we will use the `SVC` object.

- Load the full Wine dataset and divide it into a training and testing set.
- Use the `sklearn.preprocessing.StandardScaler` class to standardise the data. First, fit the StandardScaler to the training data, and then apply to both the training and testing data using the `transform()` method.
- Create and train a multiclass SVM on the training set by creating an instance of the `sklearn.svm.SVC` class and the calling its `fit()` method.
- Predict labels for the testing set and report the accuracy of your model. To do this, you can use the model's `score()` method, passing in the testing data and labels.
- Visualise the test data using a scatter plot. Colour the markers with the ground truth labels from the dataset. In an adjacent scatter plot, visualise the test data, but this time colour the markers by predicted class label. You should end up with a plot similar to the following:

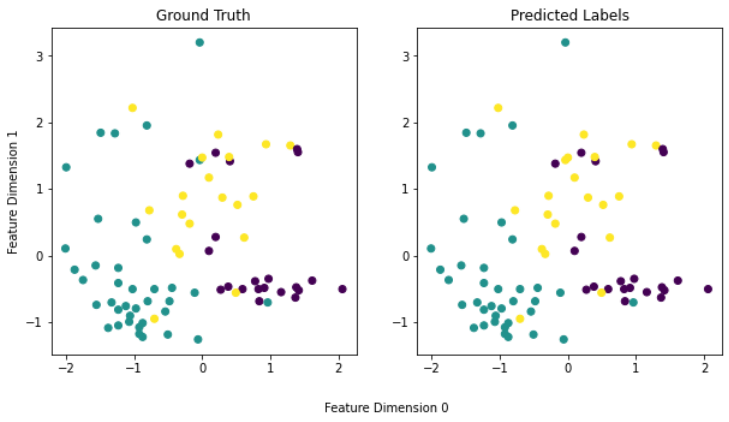


Figure 4: Plot of Ground Truth vs. SVM predictions.

- Go back and explore the different model hyper-parameters (i.e. cost, kernel type etc.), see if you can improve the accuracy.

☐ Task 4.2 – Neural Network for Wine Data

This task looks to apply the Tensorflow framework, and more specifically the Keras submodule to create a deep learning model, showing how layers can be connected to create the model. The Tensorflow API can be found at `https://www.tensorflow.org/api_docs/python/tf` and there are further, deeper tutorials into Tensorflow and Keras here.

To create our neural network we will create a `tensorflow.keras.Sequential` model. A helper notebook is provided to give you a crash-course on the framework.

- Use the same standardised training and testing set from Task 4.1
- Create and train a Tensorflow Fully Connected Neural Network on the training set. See the helper notebook and end of this handout for more guidance.
- Predict labels for the testing set and report the accuracy of your model on the testing set.
- Visualise the test data using a scatter plot. Colour the markers with the ground truth labels. In an adjacent scatter plot, visualise the test data, but this time colour the markers by the predicted class label.
- Go back and explore the model hyperparameters; for example try changing network's layers, or the optimiser. You may not see change in the result, but you can also check the training curves to observe any impact. Does it overfit more? Is it converging faster?
- You should end up with a few plots, as follows:

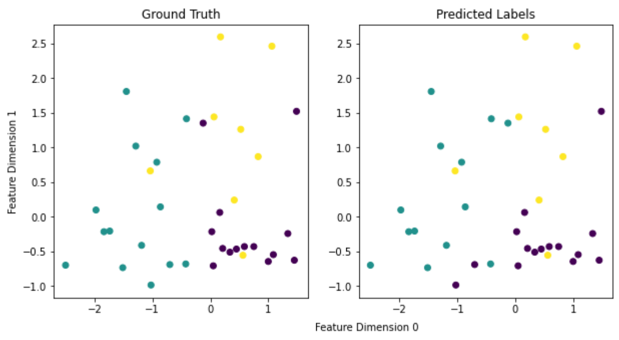


Figure 5: Plot of Ground Truth vs. Neural Network predictions.

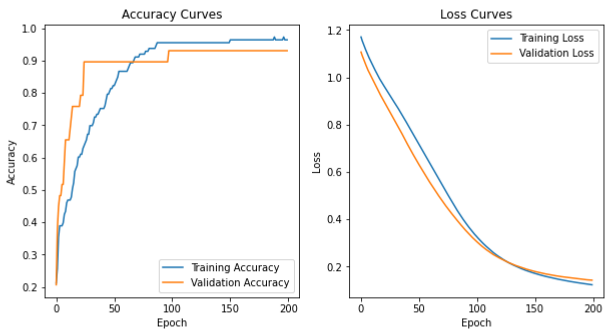


Figure 6: Plot of Neural Network training curves.

☐ Task 4.3 – Neural Network for Digit Recognition

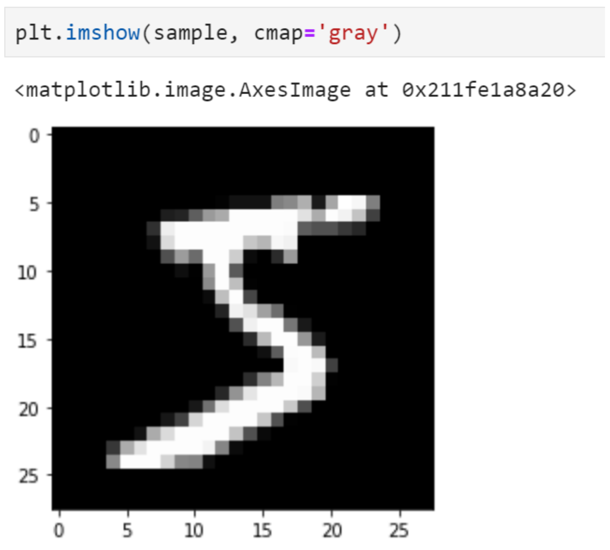
- Load in the MNIST dataset (see below).
- To use a fully connected neural network, you will need to first **flatten** the data so that is able to be passed into a Dense network. To do this, use `np.reshape()` to reshape the training data into 60000-by-784, and the testing data into 10000-by-784.
- Normalise our data by dividing it by 255 (the maximum value in the original data).
- Create and train a Tensorflow Fully Connected Neural Network on the training set. See the helper notebook and end of this handout for more guidance.
- Predict labels for the testing set and report the accuracy of your model on the testing set.
- Plot your model's training curves.
- Go back and explore the model hyperparameters; for example try changing network's layers, or the optimiser.

☐ Task 4.4 – Convolutional Neural Network for Digit Recognition

- Use the MNIST data as loaded before, with its original shape of S-H-W.
- To use a convolutional neural network, you will need to first **expand** the data so that it also has a `channel` dimension. As our data is grayscale, we only need add an additional axis in the last dimension of our data. To do this, use `np.expand_dims()` to make training data 60000-28-28-1, and the testing data 10000-28-28-1.
- Normalise our data by dividing it by 255 (the maximum value in the original data).
- Create and train a Tensorflow Convolutional Neural Network on the training set. To do this, you will need to explore the API to find out about Conv2D and Pooling layers within `tensorflow.keras.layers`.
- Predict labels for the testing set and report the accuracy of your model on the testing set.
- Plot your model's training curves.
- Go back and explore the model hyperparameters; for example try changing network's layers, or the optimiser.

Tasks Using the MNIST Dataset

The tasks here look to classify a given sample into one of 10 different classes. The data is a 28-by-28 grayscale image of a hand-written numerical digit, 0-9, as in Figure 1:



```
plt.imshow(sample, cmap='gray')
```

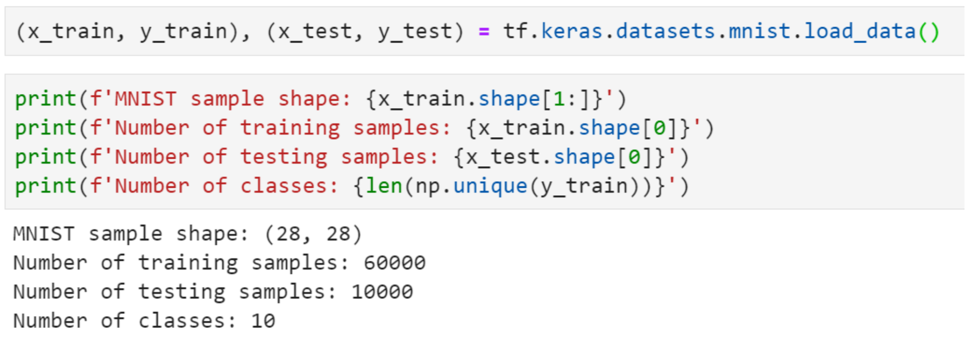
```
<matplotlib.image.AxesImage at 0x211fe1a8a20>
```

Figure 7: matplotlib plot of an example image from the MNIST dataset

In order to utilise the MNIST dataset, we must first load it into our Jupyter notebook. You can use Tensorflow's built-in datasets to do this; loading MNIST requires you to call the following:

`(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()`

This will load the MNIST dataset into 4 variables, `x_train`, `y_train`, `x_test`, and `y_test`. These correspond to the data (`x`) and targets (`y`) for the training and testing sets respectively. You can then verify the shape of the various variables loaded in:



```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

```
print(f'MNIST sample shape: {x_train.shape[1:]}')
print(f'Number of training samples: {x_train.shape[0]}')
print(f'Number of testing samples: {x_test.shape[0]}')
print(f'Number of classes: {len(np.unique(y_train))}')
```

```
MNIST sample shape: (28, 28)
Number of training samples: 60000
Number of testing samples: 10000
Number of classes: 10
```

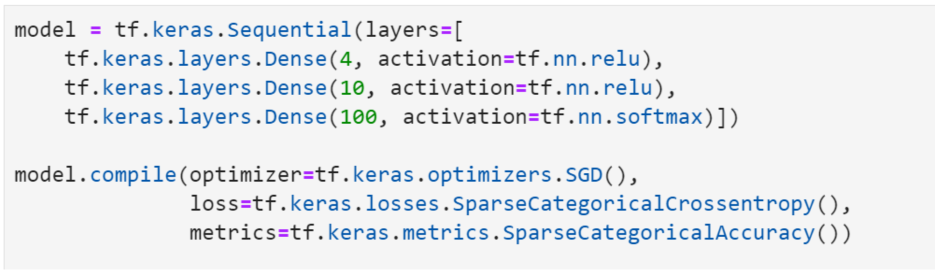
Figure 8: Shapes of the MNIST dataset loaded from `keras.datasets`

Tensorflow and the keras.Sequential model class

We can utilise the keras submodule within Tensorflow to create and train our neural network model. Keras is a part of the Tensorflow framework which is designed to make creating, training, and deploying deep learning models relatively straightforward. It contains a number of common implemented layer types, which can be passed into the constructor of the `Sequential` class.

The `Sequential` object will be a model in which the provided layers are applied one after the other to the input, producing an output. The class, and the Keras framework, provide an underlying computation graph which handles the feedforward and backpropagation required to train the network. Keras also provides the usual `fit`, `predict`, and `evaluate` methods to allow us to train our model and provide inference on a new observation. More detail on the Sequential model can be found at the Tensorflow API at `https://www.tensorflow.org/api_docs/python/tf/keras/Sequential`

To build a basic fully-connected neural network, we will create a `Sequential` model object. The following example creates a fully-connected neural network with 2 hidden layers, and an output layer. The first hidden layer is a Dense layer with 4 neurons, the second hidden layer has 10 neurons, and the output layer has 100 neurons. Each layer has an activation function applied to it once the weights and bias have been applied, the hidden layers have Rectified Linear activation applied, whilst the output layer has a softmax activation (to place the values into probability space).



```
model = tf.keras.Sequential(layers=[
    tf.keras.layers.Dense(4, activation=tf.nn.relu),
    tf.keras.layers.Dense(10, activation=tf.nn.relu),
    tf.keras.layers.Dense(100, activation=tf.nn.softmax)])

model.compile(optimizer=tf.keras.optimizers.SGD(),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=tf.keras.metrics.SparseCategoricalAccuracy())
```

Figure 9: A Sequential model, with 2 hidden layers, and a output layer with 100 outputs.

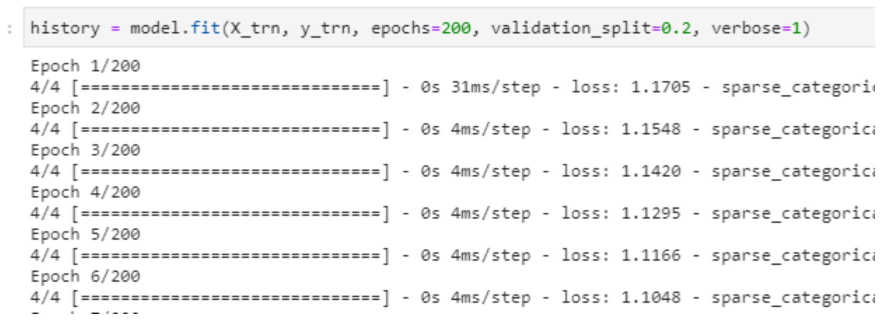
What is happening in Figure 3? First we create a Sequential model, passing instances of tf.keras Dense layers to the constructor as a list. We then compile our model with `model.compile` so that it can be trained. In the compile method we pass in details regarding which optimiser, loss, and training metric we want to use to fit our model to the data. There's a few things to unpack here:

- **tf.keras.optimizers.SGD()** This is an instance of a Stochastic Gradient Descent optimiser, one of the simplest optimisers we can use to train our model. Other optimisers include Adam(), and RMSProp().
- **tf.keras.losses.SparseCategoricalCrossentropy()** This is an instance of the categorical crossentropy loss metric, which works for targets which are labels (rather than one-hot encoded). Our targets are labels (i.e. IDs are 0-9).
- **tf.keras.metrics.SparseCategoricalAccuracy()** This is an instance of the categorical crossentropy metric for checking model accuracy. Again, this works for targets which are labels, like we have loaded in.

Once compiled, our model can be trained using the `fit()` method. Checking the API, at `https://www.tensorflow.org/api_docs/python/tf/keras/Sequential#fit`, we can see that the fit method can take in a number of parameters. The key ones are as follows:

- **x** The data, to be passed through the network in order to train.
- **y** The targets, the ground truth labels for the data in x.
- **epochs** The number of passes through the dataset to complete when training the model.

- **validation_split** A float between 0.0 and 1.0. This will split x and y into a training and validation set during training. This will let us see if over-fitting is occurring.



```
history = model.fit(X_trn, y_trn, epochs=200, validation_split=0.2, verbose=1)
```

```
Epoch 1/200
4/4 [==============================] - 0s 31ms/step - loss: 1.1705 - sparse_categori
Epoch 2/200
4/4 [==============================] - 0s 4ms/step - loss: 1.1548 - sparse_categorica
Epoch 3/200
4/4 [==============================] - 0s 4ms/step - loss: 1.1420 - sparse_categorica
Epoch 4/200
4/4 [==============================] - 0s 4ms/step - loss: 1.1295 - sparse_categorica
Epoch 5/200
4/4 [==============================] - 0s 4ms/step - loss: 1.1166 - sparse_categorica
Epoch 6/200
4/4 [==============================] - 0s 4ms/step - loss: 1.1048 - sparse_categorica
```

Figure 10: A Sequential model being fit to some data.

☐ Challenge Task 4.5

Some questions to consider:

1. What makes a neural network a "Deep Learning" model?
2. How do I make my neural network deeper? How do I make it wider?
3. How do I train my models for longer?
4. Why does running the methods numerous times result in different accuracy rates?
5. What hyperparameters are available to our models? What happens when we alter the penalty in the SVM or the optimisation strategy in the neural network?