An Introduction to Neural Networks for Classification of the Fashion-MNIST Dataset

Jeffrey M Abraham

ABSTRACT

The purpose of this report is to design fully-connected and convolutional neural networks to classify images of clothing into their types.

Keywords: Tensorflow, SGD, MNIST,

I INTRODUCTION AND OVERVIEW

This analysis is being done to classify 28x28 pixel images of articles of clothing into one of ten categories. The MNIST Fashion data set is meant to be a benchmark test for machine learning algorithms, and is mo

II THEORETICAL BACKGROUND

Fully Connected Neural Networks

A fully connected neural network as pictured in Figure 1 works by multiplying each input by a matrix of weights that determines the value of each neuron in the following layer. In this case each neuron in the hidden layer is determined by four weights multiplied by each of the inputs. Not pictured are bias neurons that add constants to the weighted inputs to shift the outputs. This has a similar function to the b term in the equation for a line $y = mx + b$. Activation functions are applied after each layer. The output layer has a specific activation function called the *softmax* activation function. This gives outputs as probability values. The neural network first has to be trained which is done by using some variation of gradient descent to minimize a loss function. For classification purposes, the neural network shown in Figure 1 would be trained using a number of (X,y), with four values in each X and two values in each y. The loss function simply put takes the average predicted probability that each input is its corresponding output in vector y. Gradient descent is done to find the way that changing each weight value or bias value would change the loss function.

$$\vec{y} = A\vec{X} + \vec{b} \tag{1}$$

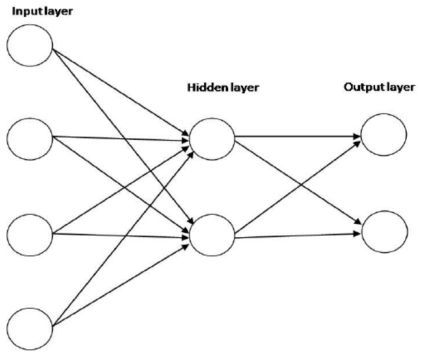


Figure 1. A Fully Connected Neural Network with 1 Hidden Layer

Optimization Functions

There are many varieties of optimization functions used to minimize the loss function. Most of these, however, are just variations of Gradient Descent. The purpose of an optimization function is to successively get closer to some minimum loss. Many hyper-parameters can be used to fine tune these functions including but not limited to learning rates, momentum, and initialization schemes.

Stochastic Gradient Descent

Stochastic gradient descent is a variation of gradient descent that uses a randomly selected data point to calculate gradients rather than the entire data set. This can cut down on processing power, but also can take indirect paths to local minima. Some compromise can be made by splitting data randomly into small batches to be used for training. This is still less intensive on the computer being used to train the network, but also allows for more direct paths to minima and less variation after arriving at a minimum.

Learning Rate

The learning rate sets the magnitude that the weight and bias values move in the direction set by the gradient. Too high learning rates can cause loss functions to skyrocket, or fall quickly and oscillate around a minima. Using something called learning rate scheduling can change the learning rate between iterations to speed up learning, while not compromising on accuracy.

Back Propagation

Back Propagation is a Jargon term used to describe the chain rule used for calculating the gradient of the loss function. It starts at the output and calculates partial derivatives and moves backward through the network to calculate partial derivatives using chain rule and the previously calculated partial derivatives.

Activation Functions

Activation functions are used to shape the output of each layer of a neural network to make them more friendly as inputs of the following layer, or in the case of output activation functions to transform values into probabilities.

Cross Validation

Cross validation is a method for checking accuracy of a predictive model. It requires that you split your data randomly into training and test data. Standard training:test splits are 70:30 and 80:20. Then you can run the test data through the predictive model and see how accurate your model is. This is important because it is possible to fit a model so that it works perfectly for the training data, but as soon as you test your model you get bad results. This is called over-fitting, and cross validation is an effective method for testing the fit of a model. For neural networks, using validation data and test data can further prevent over fitting by allowing you to tune hyper parameters with validation data and check model accuracy with test data. This way if you over-fit your model to the validation data it will still preform poorly on the test data.

Convolutional Layers

Convolutional layers in a nutshell sweep across an input signal and apply filters to areas of the signal to look for features. In a two dimensional context the filters consists of matrices of values between zero and one that exemplify features of interest. In the context of images this can be very useful by identifying features regardless of location within an image. If the training data has all the identifying features in a certain area of the image, a fully connected layer relies on pixel values which would change drastically, but with convolutional layers, these features would still be identified and images classified correctly.

III ALGORITHM IMPLEMENTATION AND DEVELOPMENT

Fully Connected Neural Network

I started development by collapsing the input into one dimension and using a pyramid like network architecture with ReLu activation, the Adam optimizer L2 regularization and a fixed training rate. On this baseline model I was able to achieve validation accuracy of around 89%

Network Architecture

I started with two hidden layers, the first 300 neurons wide and the second 100 neurons wide. the input is 28x28 so when flattened is 784 neurons, so this structure progressively reduces the number of neurons that are used to define each input to gradually move towards a reliable classification. To make the network deeper I experimented with having multiple layers at the same size before moving to a narrower layer. Also adding more different width layers like a 50 neuron wide layer right before the 10 wide output layer.

Activation Functions

Some experimentation was done exchanging the ReLu activation function which just returns zero for all negative output values to something like ELu the exponential linear unit which uses an exponential function on all negative valued tensors. This did not result in better results and rather made for larger variation in accuracy around the minimum achieved.

Optimization

Some experimentation was done with the Stochastic Gradient Descent activation function but I found that this function took too long to normalize for the fully connected layers. Adam optimizer in combination with learning rate scheduling using a constant learning rate for the first 10 epochs and exponentially decaying after that. This allowed for a sped up convergence without sacrificing stability and accuracy.

Adding Convolutional Layers

To try and improve performance further above 90% accuracy, convolutional layers were used first to identify the features within each image and then fully connected layers were used to train and classify based on the features extracted. I designed my model loosely based on the AlexNet architecture. This model was built to classify color images with much higher resolution so it had to be adapted and shortened to work on the data I was using and run faster given the computer power I am working with.

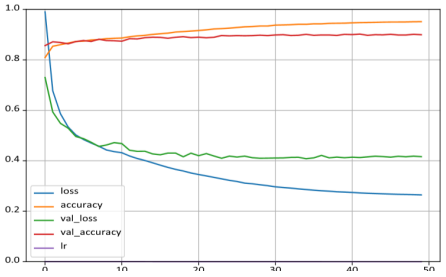
IV COMPUTATIONAL RESULTS

Fully Connected Results

Using only fully connected layers I was able to get results with a maximum validation accuracy of 90.1% using the following hyper-parameters. Two hidden layers of 300 neurons followed by two hidden layers with 100 neurons, both using PReLu activation and L2 regularization with a constant of 0.0008 to prevent over fitting. Scheduled learning rate using 0.0005 for the first 10 epochs and decaying after. After evaluating the test data, an accuracy of 89.3%. The confusion matrix and learning curves can be seen in Figure 2. The confusion matrix shows that the 0 and 6 items are the most often mistaken for each other. This is understandable as these are the T-Shirt/Top and Shirt categories respectively.

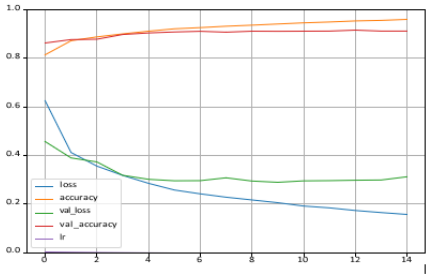
CNN Results

I applied some of the same methods to the convolutional neural network as I did to the fully connected layers. I used scheduling learning rate with the exponential decay beginning at the fifth epoch. Specifics of the network architecture can be seen in the code in the appendix. The best validation accuracy achieved was 91.4% and the test accuracy was 91.2%. The confusion matrix and learning curves can be seen in Figure 3.



	0	1	2	3	4	5	6	7	8	9
0	832	3	19	16	4	2	117	0	7	0
1	2	975	0	14	4	0	4	0	0	1
2	17	0	817	13	82	1	70	0	0	0
3	23	7	10	895	30	2	31	0	2	0
4	1	0	86	32	827	0	53	0	1	0
5	0	0	0	1	0	960	0	21	0	18
6	103	1	82	33	60	0	714	0	7	0
7	0	0	0	0	0	13	0	958	0	29
8	5	2	5	7	4	1	4	8	964	0
9	1	0	0	0	0	7	0	32	0	960

Figure 2. Learning Curves(Left) and Test Confusion Matrix(Right)



	0	1	2	3	4	5	6	7	8	9
0	900	0	18	15	3	2	57	0	5	0
1	0	976	4	12	3	0	3	0	2	0
2	19	0	888	9	44	0	38	0	2	0
3	17	2	12	917	29	0	22	0	1	0
4	4	0	53	23	881	0	38	0	1	0
5	0	0	0	1	0	971	0	15	1	12
6	142	0	75	28	73	0	671	0	11	0
7	0	0	0	0	0	7	0	974	0	19
8	2	0	1	6	6	2	1	4	977	1
9	0	0	0	0	0	4	0	30	1	965

Figure 3. Learning Curves(Left) and Test Confusion Matrix(Right)

V SUMMARY AND CONCLUSIONS

As an introduction to neural networks and hyper-parameter tuning, the limitless options can be overwhelming, but with a basic understanding of what each parameter controls neural network optimizations could be done. With more exploratory time I'm sure that better results could be achieved. The results achieved through this exercises were by no means revolutionary, but I was able to make progress and improve my results.

APPENDIX

Homework 5

March 14, 2020

[142]:
```
import tensorflow as tf
```

[143]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
```

[144]:
```
fashion_mnist = tf.keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()
```

[145]:
```
X_valid = X_train_full[:5000]/255.0
y_valid = y_train_full[:5000]
X_train = X_train_full[5000:]/255.0
y_train = y_train_full[5000:]
X_test = X_test/255.0
```

[146]:
```
from functools import partial
```

[147]:
```
my_dense_layer = partial(tf.keras.layers.Dense, activation="softmax",␣
↪kernel_regularizer=tf.keras.regularizers.l2(0.0008))
```

[148]:
```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=[28,28]),
    my_dense_layer(300,activation=None),
    tf.keras.layers.PReLU(),
    my_dense_layer(300,activation=None),
    tf.keras.layers.PReLU(),
    my_dense_layer(100,activation=None),
    tf.keras.layers.PReLU(),
    my_dense_layer(100,activation=None),
    tf.keras.layers.PReLU(),
    my_dense_layer(50,activation=None),
    tf.keras.layers.ReLU(),
    my_dense_layer(10)
])
```

[149]:
```
def scheduler(epoch):
    if epoch < 10:
        return 0.0005
    else:
        return 0.0005*tf.math.exp(0.1*(10-epoch))
```

[150]:
```
model.compile(loss="sparse_categorical_crossentropy",
              optimizer=tf.keras.optimizers.Adam(),
              metrics=["accuracy"])
```

[151]:
```
callback = tf.keras.callbacks.LearningRateScheduler(scheduler)
history = model.fit(X_train,y_train,epochs = 50, callbacks=[callback],␣
 ↪validation_data=(X_valid,y_valid))
```

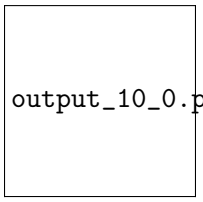
```
Train on 55000 samples, validate on 5000 samples
Epoch 1/50
55000/55000 [==============================] - 7s 120us/sample - loss: 0.9901 -
accuracy: 0.8095 - val_loss: 0.7301 - val_accuracy: 0.8568
Epoch 2/50
55000/55000 [==============================] - 6s 107us/sample - loss: 0.6775 -
accuracy: 0.8541 - val_loss: 0.5936 - val_accuracy: 0.8718
Epoch 3/50
55000/55000 [==============================] - 6s 108us/sample - loss: 0.5868 -
accuracy: 0.8604 - val_loss: 0.5487 - val_accuracy: 0.8692
Epoch 4/50
55000/55000 [==============================] - 6s 108us/sample - loss: 0.5343 -
accuracy: 0.8668 - val_loss: 0.5296 - val_accuracy: 0.8638
Epoch 5/50
55000/55000 [==============================] - 6s 109us/sample - loss: 0.5027 -
accuracy: 0.8733 - val_loss: 0.4965 - val_accuracy: 0.8726
Epoch 6/50
55000/55000 [==============================] - 6s 107us/sample - loss: 0.4846 -
accuracy: 0.8745 - val_loss: 0.4881 - val_accuracy: 0.8768
Epoch 7/50
55000/55000 [==============================] - 6s 110us/sample - loss: 0.4706 -
accuracy: 0.8791 - val_loss: 0.4735 - val_accuracy: 0.8732
Epoch 8/50
55000/55000 [==============================] - 6s 116us/sample - loss: 0.4579 -
accuracy: 0.8807 - val_loss: 0.4569 - val_accuracy: 0.8820
Epoch 9/50
55000/55000 [==============================] - 6s 112us/sample - loss: 0.4429 -
accuracy: 0.8845 - val_loss: 0.4627 - val_accuracy: 0.8768
Epoch 10/50
55000/55000 [==============================] - 6s 109us/sample - loss: 0.4364 -
accuracy: 0.8858 - val_loss: 0.4721 - val_accuracy: 0.8760
Epoch 11/50
55000/55000 [==============================] - 6s 110us/sample - loss: 0.4320 -
```

```
accuracy: 0.8868 - val_loss: 0.4682 - val_accuracy: 0.8744
Epoch 12/50
55000/55000 [==============================] - 6s 109us/sample - loss: 0.4192 -
accuracy: 0.8919 - val_loss: 0.4420 - val_accuracy: 0.8842
Epoch 13/50
55000/55000 [==============================] - 6s 112us/sample - loss: 0.4093 -
accuracy: 0.8952 - val_loss: 0.4375 - val_accuracy: 0.8832
Epoch 14/50
55000/55000 [==============================] - 6s 113us/sample - loss: 0.4013 -
accuracy: 0.8976 - val_loss: 0.4380 - val_accuracy: 0.8882
Epoch 15/50
55000/55000 [==============================] - 6s 113us/sample - loss: 0.3922 -
accuracy: 0.9012 - val_loss: 0.4275 - val_accuracy: 0.8900
Epoch 16/50
55000/55000 [==============================] - 6s 106us/sample - loss: 0.3828 -
accuracy: 0.9038 - val_loss: 0.4239 - val_accuracy: 0.8894
Epoch 17/50
55000/55000 [==============================] - 6s 109us/sample - loss: 0.3739 -
accuracy: 0.9062 - val_loss: 0.4306 - val_accuracy: 0.8860
Epoch 18/50
55000/55000 [==============================] - 6s 110us/sample - loss: 0.3661 -
accuracy: 0.9106 - val_loss: 0.4306 - val_accuracy: 0.8898
Epoch 19/50
55000/55000 [==============================] - 6s 108us/sample - loss: 0.3596 -
accuracy: 0.9123 - val_loss: 0.4160 - val_accuracy: 0.8922
Epoch 20/50
55000/55000 [==============================] - 6s 110us/sample - loss: 0.3518 -
accuracy: 0.9145 - val_loss: 0.4303 - val_accuracy: 0.8884
Epoch 21/50
55000/55000 [==============================] - 6s 107us/sample - loss: 0.3457 -
accuracy: 0.9166 - val_loss: 0.4204 - val_accuracy: 0.8902
Epoch 22/50
55000/55000 [==============================] - 6s 113us/sample - loss: 0.3403 -
accuracy: 0.9192 - val_loss: 0.4287 - val_accuracy: 0.8878
Epoch 23/50
55000/55000 [==============================] - 6s 114us/sample - loss: 0.3343 -
accuracy: 0.9227 - val_loss: 0.4190 - val_accuracy: 0.8898
Epoch 24/50
55000/55000 [==============================] - 6s 111us/sample - loss: 0.3285 -
accuracy: 0.9241 - val_loss: 0.4102 - val_accuracy: 0.8968
Epoch 25/50
55000/55000 [==============================] - 7s 121us/sample - loss: 0.3226 -
accuracy: 0.9261 - val_loss: 0.4186 - val_accuracy: 0.8954
Epoch 26/50
55000/55000 [==============================] - 6s 109us/sample - loss: 0.3181 -
accuracy: 0.9286 - val_loss: 0.4150 - val_accuracy: 0.8966
Epoch 27/50
55000/55000 [==============================] - 6s 109us/sample - loss: 0.3117 -
```

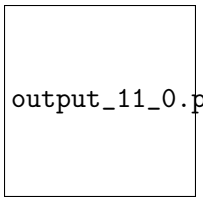
```
accuracy: 0.9312 - val_loss: 0.4185 - val_accuracy: 0.8958
Epoch 28/50
55000/55000 [==============================] - 6s 107us/sample - loss: 0.3090 -
accuracy: 0.9324 - val_loss: 0.4123 - val_accuracy: 0.8966
Epoch 29/50
55000/55000 [==============================] - 6s 106us/sample - loss: 0.3050 -
accuracy: 0.9344 - val_loss: 0.4102 - val_accuracy: 0.8980
Epoch 30/50
55000/55000 [==============================] - 6s 107us/sample - loss: 0.3015 -
accuracy: 0.9345 - val_loss: 0.4106 - val_accuracy: 0.8966
Epoch 31/50
55000/55000 [==============================] - 6s 107us/sample - loss: 0.2969 -
accuracy: 0.9381 - val_loss: 0.4111 - val_accuracy: 0.8990
Epoch 32/50
55000/55000 [==============================] - 6s 110us/sample - loss: 0.2942 -
accuracy: 0.9388 - val_loss: 0.4114 - val_accuracy: 0.9000
Epoch 33/50
55000/55000 [==============================] - 6s 109us/sample - loss: 0.2918 -
accuracy: 0.9396 - val_loss: 0.4136 - val_accuracy: 0.8970
Epoch 34/50
55000/55000 [==============================] - 6s 109us/sample - loss: 0.2887 -
accuracy: 0.9413 - val_loss: 0.4141 - val_accuracy: 0.8978
Epoch 35/50
55000/55000 [==============================] - 6s 112us/sample - loss: 0.2862 -
accuracy: 0.9413 - val_loss: 0.4085 - val_accuracy: 0.9014
Epoch 36/50
55000/55000 [==============================] - 6s 108us/sample - loss: 0.2836 -
accuracy: 0.9432 - val_loss: 0.4117 - val_accuracy: 0.8976
Epoch 37/50
55000/55000 [==============================] - 6s 110us/sample - loss: 0.2814 -
accuracy: 0.9431 - val_loss: 0.4215 - val_accuracy: 0.8990
Epoch 38/50
55000/55000 [==============================] - 6s 110us/sample - loss: 0.2796 -
accuracy: 0.9449 - val_loss: 0.4118 - val_accuracy: 0.8988
Epoch 39/50
55000/55000 [==============================] - 6s 108us/sample - loss: 0.2773 -
accuracy: 0.9454 - val_loss: 0.4148 - val_accuracy: 0.8972
Epoch 40/50
55000/55000 [==============================] - 6s 110us/sample - loss: 0.2760 -
accuracy: 0.9457 - val_loss: 0.4122 - val_accuracy: 0.9014
Epoch 41/50
55000/55000 [==============================] - 6s 109us/sample - loss: 0.2746 -
accuracy: 0.9471 - val_loss: 0.4144 - val_accuracy: 0.9004
Epoch 42/50
55000/55000 [==============================] - 6s 109us/sample - loss: 0.2727 -
accuracy: 0.9480 - val_loss: 0.4131 - val_accuracy: 0.9024
Epoch 43/50
55000/55000 [==============================] - 6s 109us/sample - loss: 0.2714 -
```

```
accuracy: 0.9485 - val_loss: 0.4156 - val_accuracy: 0.8976
Epoch 44/50
55000/55000 [==============================] - 6s 108us/sample - loss: 0.2703 -
accuracy: 0.9491 - val_loss: 0.4183 - val_accuracy: 0.9000
Epoch 45/50
55000/55000 [==============================] - 6s 111us/sample - loss: 0.2692 -
accuracy: 0.9498 - val_loss: 0.4167 - val_accuracy: 0.8994
Epoch 46/50
55000/55000 [==============================] - 6s 105us/sample - loss: 0.2682 -
accuracy: 0.9502 - val_loss: 0.4146 - val_accuracy: 0.9016
Epoch 47/50
55000/55000 [==============================] - 6s 105us/sample - loss: 0.2671 -
accuracy: 0.9504 - val_loss: 0.4180 - val_accuracy: 0.8988
Epoch 48/50
55000/55000 [==============================] - 6s 106us/sample - loss: 0.2665 -
accuracy: 0.9506 - val_loss: 0.4161 - val_accuracy: 0.8988
Epoch 49/50
55000/55000 [==============================] - 6s 105us/sample - loss: 0.2658 -
accuracy: 0.9513 - val_loss: 0.4185 - val_accuracy: 0.9016
Epoch 50/50
55000/55000 [==============================] - 6s 108us/sample - loss: 0.2649 -
accuracy: 0.9516 - val_loss: 0.4164 - val_accuracy: 0.8998
```

[157]:
```
pd.DataFrame(history.history).plot(figsize=(8,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.show
plt.savefig('learning_curve.pdf')
```



[160]:
```
pd.DataFrame(history.history)['lr'].plot(figsize=(8,5))
plt.grid(True)
plt.gca().set_ylim(0,0.001)
plt.show
plt.savefig('learning_rate.pdf')
```



[153]:
```
y_pred = model.predict_classes(X_train)
conf_train = confusion_matrix(y_train,y_pred)
print(conf_train)
print(np.tril(conf_train,-1)+np.triu(conf_train,1))
```

```
[[5040    0   69   60    4    0  359    1    9    1]
 [   0 5407    1   27    7    0    1    0    1    0]
 [  47    0 4958   28  269    0  193    0    1    0]
 [  40    4   13 5286  108    0   46    0    2    0]
 [   2    1  195   91 5036    0  182    0    5    0]
 [   0    0    0    0    0 5507    0    0    0    0]
 [ 277    2  187   66  164    0 4810    0    1    0]
 [   0    0    0    0    0    0    0 5440    1   47]
 [   4    0    2    7    4    0    6    2 5485    0]
 [   0    1    0    0    0    0    0   83    1 5409]]
[[  0   0  69  60   4   0 359   1   9   1]
 [  0   0   1  27   7   0   1   0   1   0]
 [ 47   0   0  28 269   0 193   0   1   0]
 [ 40   4  13   0 108   0  46   0   2   0]
 [  2   1 195  91   0   0 182   0   5   0]
 [  0   0   0   0   0   0   0   0   0   0]
 [277   2 187  66 164   0   0   0   1   0]
 [  0   0   0   0   0   0   0   0   1  47]
 [  4   0   2   7   4   0   6   2   0   0]
 [  0   1   0   0   0   0   0  83   1   0]]
```

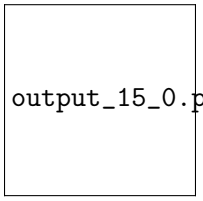
[154]:
```
pd.DataFrame(history.history)['val_accuracy'].max()
```

[154]:
```
0.902400016784668
```

[155]:
```
y_pred = model.predict_classes(X_test)
```

[161]:
```
df = pd.DataFrame(conf_test)
fig, ax = plt.subplots()
fig.patch.set_visible(False)
ax.axis('off')
ax.axis('tight')
ax.table(cellText=df.values,rowLabels=np.arange(10),colLabels=np.
 ↪arange(10),loc='center',cellLoc='center')
```

```
fig.tight_layout()
plt.savefig('conf_matr.pdf')
```



[ ]:

Homework 5_2

March 14, 2020

[30]:
```
import tensorflow as tf
```

[31]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
```

[32]:
```
fashion_mnist = tf.keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()
```

[33]:
```
X_valid = X_train_full[:5000]/255.0
y_valid = y_train_full[:5000]
X_train = X_train_full[5000:]/255.0
y_train = y_train_full[5000:]
X_test = X_test/255.0


X_valid = X_valid[...,np.newaxis]
X_train = X_train[...,np.newaxis]
X_test = X_test[...,np.newaxis]
```

[34]:
```
from functools import partial
```

[35]:
```
my_dense_layer = partial(tf.keras.layers.Dense, activation="tanh",␣
 ↪kernel_regularizer=tf.keras.regularizers.l2(0.001))
my_conv_layer = partial(tf.keras.layers.Conv2D,␣
 ↪activation="tanh",padding="valid")
```

[36]:
```
model = tf.keras.models.Sequential([
    my_conv_layer(128,5,input_shape=[28,28,1]),
    tf.keras.layers.MaxPool2D(pool_size=(3,3),strides=(2,2)),
    my_conv_layer(156,3),
    tf.keras.layers.MaxPool2D(pool_size=(3,3),strides=(2,2)),
    my_conv_layer(156,3),
    tf.keras.layers.Flatten(),
    my_dense_layer(84, activation=None),
```

```
    tf.keras.layers.PReLU(),
    my_dense_layer(84, activation=None),
    tf.keras.layers.PReLU(),
    my_dense_layer(10,activation = "softmax")
])
```

[37]:
```
def scheduler(epoch):
    if epoch>5:
        return 0.0005
    else:
        return 0.0005*tf.math.exp(0.2*(5-epoch))
```

[38]:
```
model.compile(loss="sparse_categorical_crossentropy",
              optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
              metrics=["accuracy"])
```

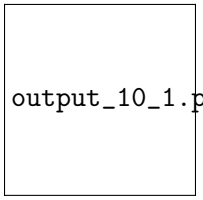
[39]:
```
callback = tf.keras.callbacks.LearningRateScheduler(scheduler)
history = model.fit(X_train,y_train,epochs = 15, callbacks=[callback],␣
 ↪validation_data=(X_valid,y_valid))
```

```
Train on 55000 samples, validate on 5000 samples
Epoch 1/15
55000/55000 [==============================] - 72s 1ms/sample - loss: 0.6231 -
accuracy: 0.8152 - val_loss: 0.4588 - val_accuracy: 0.8644
Epoch 2/15
55000/55000 [==============================] - 71s 1ms/sample - loss: 0.4109 -
accuracy: 0.8735 - val_loss: 0.3917 - val_accuracy: 0.8776
Epoch 3/15
55000/55000 [==============================] - 73s 1ms/sample - loss: 0.3548 -
accuracy: 0.8895 - val_loss: 0.3750 - val_accuracy: 0.8790
Epoch 4/15
55000/55000 [==============================] - 73s 1ms/sample - loss: 0.3166 -
accuracy: 0.9019 - val_loss: 0.3201 - val_accuracy: 0.8974
Epoch 5/15
55000/55000 [==============================] - 71s 1ms/sample - loss: 0.2839 -
accuracy: 0.9131 - val_loss: 0.3035 - val_accuracy: 0.9034
Epoch 6/15
55000/55000 [==============================] - 68s 1ms/sample - loss: 0.2564 -
accuracy: 0.9230 - val_loss: 0.2975 - val_accuracy: 0.9072
Epoch 7/15
55000/55000 [==============================] - 73s 1ms/sample - loss: 0.2407 -
accuracy: 0.9280 - val_loss: 0.2980 - val_accuracy: 0.9098
Epoch 8/15
55000/55000 [==============================] - 68s 1ms/sample - loss: 0.2268 -
accuracy: 0.9337 - val_loss: 0.3097 - val_accuracy: 0.9064
Epoch 9/15
55000/55000 [==============================] - 67s 1ms/sample - loss: 0.2156 -
```

```
accuracy: 0.9381 - val_loss: 0.2964 - val_accuracy: 0.9102
Epoch 10/15
55000/55000 [==============================] - 67s 1ms/sample - loss: 0.2043 -
accuracy: 0.9430 - val_loss: 0.2913 - val_accuracy: 0.9098
Epoch 11/15
55000/55000 [==============================] - 68s 1ms/sample - loss: 0.1905 -
accuracy: 0.9478 - val_loss: 0.2972 - val_accuracy: 0.9102
Epoch 12/15
55000/55000 [==============================] - 68s 1ms/sample - loss: 0.1830 -
accuracy: 0.9514 - val_loss: 0.2981 - val_accuracy: 0.9106
Epoch 13/15
55000/55000 [==============================] - 68s 1ms/sample - loss: 0.1721 -
accuracy: 0.9557 - val_loss: 0.2998 - val_accuracy: 0.9144
Epoch 14/15
55000/55000 [==============================] - 71s 1ms/sample - loss: 0.1636 -
accuracy: 0.9580 - val_loss: 0.3007 - val_accuracy: 0.9106
Epoch 15/15
55000/55000 [==============================] - 75s 1ms/sample - loss: 0.1559 -
accuracy: 0.9621 - val_loss: 0.3140 - val_accuracy: 0.9106
```

[40]:
```
pd.DataFrame(history.history).plot(figsize=(8,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.show
```

[40]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



[41]:
```
y_pred = model.predict_classes(X_train)
conf_train = confusion_matrix(y_train,y_pred)
print(pd.DataFrame(conf_train))
print(np.tril(conf_train,-1)+np.triu(conf_train,1))
```

```
      0     1     2     3     4     5     6     7     8     9
0  5347     0    45    29     3     0   115     0     4     0
1     0  5416     1    20     5     0     0     0     2     0
2    57     0  5274    22   100     0    43     0     0     0
3    32     1     1  5406    43     0    16     0     0     0
4     3     0    86    31  5321     0    68     0     3     0
5     0     0     0     0     0  5484     0     7     1    15
```

```
6   428     0   125    54   118     0  4779     0     3     0
7     0     0     0     0     0     2     0  5441     0    45
8     7     1     1     0     5     0     2     0  5493     1
9     0     0     0     0     0     2     0    61     0  5431
[[  0   0  45  29   3   0 115   0   4   0]
 [  0   0   1  20   5   0   0   0   2   0]
 [ 57   0   0  22 100   0  43   0   0   0]
 [ 32   1   1   0  43   0  16   0   0   0]
 [  3   0  86  31   0   0  68   0   3   0]
 [  0   0   0   0   0   0   0   7   1  15]
 [428   0 125  54 118   0   0   0   3   0]
 [  0   0   0   0   0   2   0   0   0  45]
 [  7   1   1   0   5   0   2   0   0   1]
 [  0   0   0   0   0   2   0  61   0   0]]
```

[42]:
```
pd.DataFrame(history.history)['val_accuracy'].max()
```

[42]:
```
0.9143999814987183
```

[43]:
```
#model.evaluate(X_test,y_test)
```

[ ]:
```

```