**Assignment 2**

*CSCI E-101 Intro to Artificial Intelligence*

Student: Ferran Hueto Puig

Student ID: 00000000

In this report I will run through a set of tests to improve on the current accuracy for the model provided in assignment 2. Before running any tests, I will first record the accuracy for all three models as a benchmark. In every experiment I will try to optimize a specific hyperparameter. For the following experiments, I will use the chosen best hyperparameter, until I get all of my hyperparameters optimized. I will then compare the performance of all three datasets against my benchmark.

I noticed right away that some of my models wouldn’t train the whole way through (which is, the validation accuracy was still going up when the training finished) so I increased the number of epochs to 15 to prevent that.

**EPOCHS: 20**

# Benchmark

Table 1: Benchmark Validation Accuracy and Validation Loss for all datasets

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Computer Vision** | **Speech Recognition** | **NLP** |
| **Val. Accuracy (%)** | 96.92 | 44.08 | 88.27% |
| **Val. Loss** | 0.0993 | 8.3756 | 0.3052 |

The model seems to especially struggle with the speech recognition task, reaching only 44.08% accuracy.

# Experiment 1: Learning Rates

I will now modify the learning rates for all models to see how they behave. In theory, I could just manually change the variable “lr” in my code every time before running the command “python assignment2\_answer.py”. In order to speed up the process, I modified the code to loop over multiple learning rates at once:

learning\_rate = [1, 0.1, 0.001, 0.0001, 0.00001, 0.000001]

# Loop over several learning rates

**for** lr **in** learning\_rate:

# Generate and train model

model = generic\_vns\_function(X\_train.shape[1], layers, y\_train.shape[1],

layer\_units, lr)

trained\_model = train\_model(model, epochs, batch\_size, X\_train, y\_train,

X\_test, y\_test)

# Save model to h5 file

trained\_model.save('models/model\_%s\_a2.h5' % dataset)

To be able to easily compare my results, I set “verbose” in the training function to 0, in order to avoid printing of progress per epoch, and printed accuracy and loss instead standard error:

**def** train\_model(model, epochs, batch\_size, X\_train, y\_train, X\_test, y\_test):

"""Generic Deep Learning Model training function."""

model.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test), epochs=epochs,

batch\_size=batch\_size, verbose=0)

scores = model.evaluate(X\_test, y\_test, verbose=0)

#print("Baseline Error: %.2f%%" % (100-scores[1]\*100))

**print**("Accuracy: %.2f%%" % (scores[1]\*100))

**print**("Loss: %.4f" % (scores[0]))

**return** model

I obtain the following results.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | **Computer Vision** | **Speech Recognition** | **NLP** |
| *lr = 1* | Val. Accuracy (%) | 19.32 | 10.46 | 50.00 |
| Val. Loss | 13.0041 | 14.4318 | 8. 0590 |
| *lr = 0.1* | Val. Accuracy (%) | 9.80 | 9.72 | 50.00 |
| Val. Loss | 14.5385 | 14.5518 | 8.0590 |
| *lr = 0.01* | Val. Accuracy (%) | 29.57 | 9.80 | 86.02 |
| Val. Loss | 11.3520 | 14.5391 | 0.7881 |
| *lr = 0.001* | Val. Accuracy (%) | **97.52** | 10.46 | 87.71 |
| Val. Loss | **0.0883** | 14.4318 | 0.4636 |
| *lr = 0.0001* | Val. Accuracy (%) | 97.14 | **46.79** | **88.01** |
| Val. Loss | 0.0937 | **7.5458** | **0.3079** |
| *lr = 0.00001* | Val. Accuracy (%) | 92.16 | 42.12 | 82.26 |
| Val. Loss | 0.2588 | 7.7030 | 0.4353 |

Learning rate of 0.0001 seems to yield the most performant models overall, as it is the best learning rate for both speech recognition and NLP, and yields a model only 0.4% from the most performant one for computer vision.

**BEST LEARNING RATE: 0.0001**

# Experiment 2: Layers

I will now attempt to add either 1, 2 or 3 extra layers to my model to see how performance changes. Once again I could create a for loop to test all three options in one go. This time I decided against it, so I had to change the layer variable and dataset variable a total of 9 times to test out all possible scenarios.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | **Computer Vision** | **Speech Recognition** | **NLP** |
| *# Layers = 1* | Val. Accuracy (%) | 97.14 | 46.79 | **88.01** |
| Val. Loss | 0.0937 | 7.5458 | **0.3079** |
| *# Layers = 2* | Val. Accuracy (%) | 97.57 | 64.18 | 86.90 |
| Val. Loss | .0802 | 2.8012 | 0.5963 |
| *# Layers = 3* | Val. Accuracy (%) | **97.61** | **75.20** | 86.11 |
| Val. Loss | **.0872** | **1.5408** | 0.7671 |

It seems that as you increase the number of layers, the accuracy of the system for speech and vision keeps on improving. I stopped at 3 layers because otherwise the algorithm would take too long to run.

**BEST # LAYERS = 3**

# Experiment 3: Batch Size

I will now attempt to add either 1, 2 or 3 extra layers to my model to see how performance changes.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | **Computer Vision** | **Speech Recognition** | **NLP** |
| *# Batch Size = 100* | Val. Accuracy (%) | 97.54 | 9.56 | **86.64** |
| Val. Loss | 0.1072 | 14.5770 | **0.5416** |
| *# Batch Size = 200* | Val. Accuracy (%) | 97.61 | **75.20** | 86.11 |
| Val. Loss | .0872 | **1.5408** | 0.7671 |
| *# Batch Size = 400* | Val. Accuracy (%) | **97.76** | 59.84 | 86.11 |
| Val. Loss | **0.0793** | 5.1896 | 0.4333 |
| *# Batch Size = 800* | Val. Accuracy (%) | 97.64 | 62.85% | 85.43 |
| Val. Loss | 0.0848 | 4.2475 | 0.3788 |

As the batch size increased, the training time substantially decreases. A possible alternative would be to increase the number of layers as you increase the batch size. In this case, I decided to keep the layer number and set the batch size to 200.

# Experiment 4: Layer Size

In this case I chose only three layer sizes, as increasing this value makes the training very slow. I evaluated sizes 500, 1000 and 2000.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | **Computer Vision** | **Speech Recognition** | **NLP** |
| *# Layer Size = 500* | Val. Accuracy (%) | 98.26 | 74.80 | **86.31** |
| Val. Loss | 0.0670 | 1.1186 | **0.5887** |
| *# Layer Size = 1000* | Val. Accuracy (%) | 97.61 | **75.20** | 86.11 |
| Val. Loss | .0872 | **1.5408** | 0.7671 |
| *# Layer Size = 2000* | Val. Accuracy (%) | **97.63** | 9.72 | - |
| Val. Loss | **0.1382** | 14.5518 | - |

It appears that I am able to obtain very similar accuracies with much smaller layers (500 instead of 1000). I wasn’t able to compute Layer Size 2000 for NLP because the training took too long and caused my computer to crash.

**BEST # UNITS PER LAYER = 500**

# Final Model

The code below represents my final optimized model:

**def** main():

# Hyperparameters

layers = 3

layer\_units = 500

epochs = 15

batch\_size = 200

# Dataset : "nlp", "computer\_vision" or "speech\_recognition"

dataset = "computer\_vision"

# Import Datasets

(X\_train, y\_train), (X\_test, y\_test) = choose\_dataset(dataset)

learning\_rate = [0.0001]

# Loop over several learning rates

**for** lr **in** learning\_rate:

# Generate and train model

**print**(lr)

model = generic\_vns\_function(X\_train.shape[1], layers, y\_train.shape[1], layer\_units, lr)

trained\_model = train\_model(model, epochs, batch\_size, X\_train, y\_train, X\_test, y\_test)

# Save model to h5 file

trained\_model.save('models/model\_%s\_a2.h5' % dataset)

**return** None

I attach a the folder “models” with files for each model with best accuracies:

* For Speech Recognition model\_speech\_recognition\_a2.h5
* For Computer Vision model\_computer\_vision\_a2.h5
* For NLP model\_nlp\_a2.h5

The table below shows accuracies and losses for all models:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Computer Vision** | **Speech Recognition** | **NLP** |
| Val. Accuracy (%) | 98.26 | 74.80 | 86.31 |
| Val. Loss | 0.0670 | 1.1186 | 0.5887 |
| **Improvement over benchmark acc.** | + 1.34 | + 30.72 | - 1.96 |

# ANNEX: Command history

In this command history, I show how I run my model once. During the assignment, I had to repeat this process many more times, as I manipulated the different hyperparameters.

(base) ferran:~/code/aibook$ conda activate py36

(py36) ferran:~/code/aibook$ touch assignment2\_answer.py

(py36) ferran:~/code/aibook$ python assignment2\_answer.py

Using TensorFlow backend.

0.0001

Established Secure Connection.

WARNING:tensorflow:From /home/ferran/anaconda3/envs/py36/lib/python3.6/site-packages/keras/backend/tensorflow\_backend.py:74: The name tf.get\_default\_graph is deprecated. Please use tf.compat.v1.get\_default\_graph instead.

WARNING:tensorflow:From /home/ferran/anaconda3/envs/py36/lib/python3.6/site-packages/keras/optimizers.py:790: The name tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead.

Ended Secure Connection.

WARNING:tensorflow:From /home/ferran/anaconda3/envs/py36/lib/python3.6/site-packages/tensorflow\_core/python/ops/math\_grad.py:1424: where (from tensorflow.python.ops.array\_ops) is deprecated and will be removed in a future version.

Instructions for updating:

Use tf.where in 2.0, which has the same broadcast rule as np.where

Train on 60000 samples, validate on 10000 samples

Epoch 1/15

60000/60000 [==============================] - 3s 49us/step - loss: 0.3938 - acc: 0.8798 - val\_loss: 0.1759 - val\_acc: 0.9467

Epoch 2/15

60000/60000 [==============================] - 2s 25us/step - loss: 0.1429 - acc: 0.9574 - val\_loss: 0.1245 - val\_acc: 0.9622

Epoch 3/15

60000/60000 [==============================] - 2s 26us/step - loss: 0.0954 - acc: 0.9727 - val\_loss: 0.1044 - val\_acc: 0.9680

Epoch 4/15

60000/60000 [==============================] - 2s 25us/step - loss: 0.0681 - acc: 0.9811 - val\_loss: 0.0989 - val\_acc: 0.9671

Epoch 5/15

60000/60000 [==============================] - 2s 25us/step - loss: 0.0501 - acc: 0.9860 - val\_loss: 0.0871 - val\_acc: 0.9721

Epoch 6/15

60000/60000 [==============================] - 1s 25us/step - loss: 0.0391 - acc: 0.9891 - val\_loss: 0.0760 - val\_acc: 0.9769

Epoch 7/15

60000/60000 [==============================] - 2s 25us/step - loss: 0.0276 - acc: 0.9932 - val\_loss: 0.0772 - val\_acc: 0.9770

Epoch 8/15

60000/60000 [==============================] - 1s 25us/step - loss: 0.0209 - acc: 0.9953 - val\_loss: 0.0752 - val\_acc: 0.9773

Epoch 9/15

60000/60000 [==============================] - 1s 24us/step - loss: 0.0153 - acc: 0.9967 - val\_loss: 0.0728 - val\_acc: 0.9782

Epoch 10/15

60000/60000 [==============================] - 1s 24us/step - loss: 0.0124 - acc: 0.9977 - val\_loss: 0.0774 - val\_acc: 0.9764

Epoch 11/15

60000/60000 [==============================] - 1s 24us/step - loss: 0.0101 - acc: 0.9982 - val\_loss: 0.0728 - val\_acc: 0.9788

Epoch 12/15

60000/60000 [==============================] - 1s 25us/step - loss: 0.0083 - acc: 0.9984 - val\_loss: 0.0718 - val\_acc: 0.9801

Epoch 13/15

60000/60000 [==============================] - 1s 24us/step - loss: 0.0055 - acc: 0.9993 - val\_loss: 0.0752 - val\_acc: 0.9798

Epoch 14/15

60000/60000 [==============================] - 2s 26us/step - loss: 0.0042 - acc: 0.9995 - val\_loss: 0.0747 - val\_acc: 0.9789

Epoch 15/15

60000/60000 [==============================] - 2s 26us/step - loss: 0.0030 - acc: 0.9996 - val\_loss: 0.0748 - val\_acc: 0.9808

Accuracy: 98.08%

Loss: 0.0748

-----------------------------------------------------