**CIS-490I Deep Learning**

**With Dr. Jin Lu**

**HW2**

**Building an MLP Model**

**Student: Demetrius Johnson**

**11 February 2023**

**CIS 490I/590K Homework 2**

**1) Build a MLP Model:**

Download ‘First MLP.ipynb’ on Canvas and try to open it in Jupyter and run it on your computer. Remember to use Conda to install any packages that are missing. Then, build your own neural network with Keras layers to train the model using the dataset 10 class handwritten digits. Your own model should be a three-Dense-layer MLP (1st layer: 100 hidden nodes and activation function ‘tanh’, 2nd layer: with 100 hidden nodes and activation function ‘tanh’, 3rd layer: output\_dim output nodes and activation function ‘softmax’).

Submit your screenshot of the training convergence.

**2) Impact of the Optimizer**

A) Try to decrease the learning rate value by 10 or 100. What do you observe?

Submit your screenshot of the new convergence figure.

B) Try to increase the learning rate value to make the optimization diverge.

Submit your screenshot of the divergence figure.

**\*\*\*\*Notes**:

The keras API documentation is available at:

<https://www.tensorflow.org/api_docs/python/tf/keras>

It is also possible to learn more about the parameters of a class by using the question mark: type and evaluate:

optimizers**.**SGD?

in a jupyter notebook cell.

It is also possible to type the beginning of a function call / constructor and type "shift-tab" after the opening paren:

optimizers**.**SGD(**<**shiff**-**tab**>**

optimizers.SGD**?**

C) Replace the SGD optimizer by the Adam optimizer from keras and run it with the default parameters.

Submit your screenshot of the training convergence.

Hint: use optimizers.<TAB> to tab-complete the list of implemented optimizers in Keras.

D) Use the "Rectified Linear Unit" for each hidden layer. Can you still train the model with Adam with its default global learning rate?

Submit your screenshot of the training convergence/non-convergence figure.

**3) Forward Pass and Generalization**

A) Compute predictions of the model from 1) on test set using

 y\_predicted = np.argmax(model.predict(X\_test, verbose=0), axis=1)

B) Compute average accuracy of the model on the test set: the fraction of test samples for which the model makes a prediction that matches the true label.

# Let's display the first inputs image, the predicted labels and the true labels

fig, axes = plt.subplots(ncols=5, nrows=3, figsize=(12, 9))

for i, ax in enumerate(axes.ravel()):

ax.imshow(scaler.inverse\_transform(X\_test[i:i+1]).reshape(8, 8), cmap=plt.cm.gray\_r, interpolation='nearest')

ax.set\_title("predicted label: %d\n true label: %d" % (y\_predicted[i], y\_test[i]))

print("test acc: %0.4f" % np.mean(y\_predicted == y\_test))

Submit your screenshot of the test accuracy.

**4) Impact of Initialization**

Let us study the impact of a bad initialization when training a deep feed forward network.

To assess the impact of initialization let us plug an alternative init scheme into a 2 hidden layers networks with "tanh" activations. For the sake of the example let's use normal distributed weights with a manually adjustable scale (standard deviation) and see the impact the scale value:

**from** tensorflow.keras **import** initializers

normal\_init **=** initializers**.**TruncatedNormal(stddev**=**0.01)

model **=** Sequential()

model**.**add(Dense(hidden\_dim, input\_dim**=**input\_dim, activation**=**"tanh", kernel\_initializer**=**normal\_init))

model**.**add(Dense(hidden\_dim, activation**=**"tanh", kernel\_initializer**=**normal\_init))

model**.**add(Dense(output\_dim, activation**=**"softmax", kernel\_initializer**=**normal\_init))

model**.**compile(optimizer**=**optimizers**.**SGD(learning\_rate**=**0.1), loss**=**'categorical\_crossentropy', metrics**=**['accuracy'])

model**.**layers

Let's have a look at the parameters of the first layer after initialization but before any training has happened:

model**.**layers[0]**.**weights

w **=** model**.**layers[0]**.**weights[0]**.**numpy()

w

b **=** model**.**layers[0]**.**weights[1]**.**numpy()

b

history **=** model**.**fit(X\_train, Y\_train, epochs**=**15, batch\_size**=**32)

plt**.**figure(figsize**=**(12, 4))

plt**.**plot(history**.**history['loss'], label**=**"Truncated Normal init")

plt**.**legend();

Once the model has been fit, the weights have been updated and notably the biases are no longer 0:

model**.**layers[0]**.**weights

Try the following initialization schemes and see whether the SGD algorithm can successfully train the network or not:

A) a very small e.g. stddev=1e-3

Submit your screenshot of the training convergence.

B) a larger scale e.g. stddev=1 or 10

Submit your screenshot of the training convergence.

C) initialize all weights to 0 (stddev=0)

Submit your screenshot of the training convergence.

D) Find an explanation for those outcomes.

E) Are more advanced solvers such as SGD with momentum or Adam able to deal better with such bad initializations?