Alessandro Allegranzi

MET CS 767

Assignment 2: Neural Nets Intro

5/22/024

Table of Contents

[\_Toc166923437](#_Toc166923437)

[1. How I modified data and/or code to attempt improvement 2](#_Toc166923438)

[1.1 Description of what you did and reason(s) it could be an improvement (include the relevant code fragments) 2](#_Toc166923439)

[1.2 Comparison of the result with the original output 4](#_Toc166923440)

[1.3 URL of your Colab code 4](#_Toc166923441)

[2. New application 5](#_Toc166923443)

[2.1 Description of the application (include description of inputs, functionality, and outputs—in no more than ½ page. Identify a clearly obtainable data source.) 5](#_Toc166923444)

[2.2 Three illustrative input/output pairs from running the implementation 5](#_Toc166923445)

[2.3 Key code snippets, with explanation (the important code—in no more than a page) 6](#_Toc166923446)

[2.4 URL of your code (Colab code--or attach and explain how to run if necessary) 10](#_Toc166923447)

[References 10](#_Toc166923449)

[Evaluation 10](#_Toc166923450)

[Appendix 1 11](#_Toc166923451)

[Appendix 2 11](#_Toc166923452)

5/22/2024

MET CS 767 Assignment 2: Neural Nets Intro

*Alessandro Allegranzi*

The purpose of this exercise is to give you practice with standard neural net implementations and their parameters. Architecture and parameter manipulation are essential elements of neural net application. We typically start with an existing implementation when we can.

## Please use this Word file template for your response. Follow—and retain—these instructions in gray text. Insert your work in black where indicated. Keep in mind the evaluation matrix below as you do the work and use it to guide what you submit. It is recommended that you complete this assignment with the assistance of an AI generator such as ChatGPT, and the assignments are designed to assume this. Your work will be assessed in terms of your *value added* to existing or generated material, as per the evaluation matrix at the end. You will describe your value added in the format provided by this Word template. Your description consists of your prompts together with you edits and additions to AI-generated material. You are not *obliged* to used AI generation, however, in which case please indicate this whereever requested in this template.

You may build on the application you selected in Assignment 1 if you wish.

Use no more than 6 pages of 12-point text excluding figures, the instructions in gray, your AI generator descriptions, and appendices. You can add as many appendices as you like for voluminous material, as per the instructions provided in the first appendix below. It will be read on an as-needed basis.

You may build on the work of others but (1) show clearly that you understand this work and (2) observe all plagiarism rules scrupulously, including clear citations. Use the Reference section at the end.

**Hints**:

* Be organized in your parameter value process; keep track of results as you go.
* For parameter optimization, consider using a binary process (halving and doubling)—explained in live classroom—to identify appropriate parameter values.
* Keep in mind that chatGPT can be an excellent tutor.
* You can’t share your solution, of course, but if you locate a useful website (e.g., to modify a data set), share it with the class. You will benefit from what they share with you.
* Tailor your response to the evaluation criteria (at the end of this) as you respond.
* Leave ample time to edit your work. It is a pity when you do lots of good work but that work is obscured in your submission.

# How I modified data and/or code to attempt improvement

Copy the implementation [here](https://colab.research.google.com/drive/1TI76Pg5RYmlA0sTNthkqDhTjZWfIoZwn?usp=sharing) to your Google drive. Modify this code, attempting to improve the output, and report the results, (using this Word file as a template as below). Since the accuracy of the given implementation is already high, consider reducing the size of the MNIST training set—or using a set like [mnistcorrupted](https://www.tensorflow.org/datasets/catalog/mnist_corrupted)—so that the baseline implementation leaves more room for percentage improvement. (The application is bound to be less effective with fewer data.) If necessary, show changes that make the result worse, with your best explanation.

## 1.1 Description of what you did and reason(s) it could be an improvement (include the relevant code fragments)

**Making model less effective:**

Running the provided implementation as-is produced a model with just under 98% accuracy (accuracy: 0.9781), already high. To lower the baseline model accuracy so that it can be improved, the following steps were taken:

1. Lowered the number of neurons to 25 and used sigmoid function. The sigmoid function is an earlier activation function compared to relu, and is likely less ideal to the task, strictly relatively and possibly [1].

tf.keras.layers.Dense(25, activation='sigmoid'), # fully connected to hidden layer with relu, lowered neurons to 25

1. Lowered the epochs to 2

# Train it with (just) 2 epochs

model.fit(x\_train, y\_train, epochs=2)

1. Halved the training data.

# Calculate half the number of samples.

num\_samples = int(0.5 \* len(x\_train))

# Select a subset of half the original data

x\_train\_subset = x\_train[:num\_samples]

y\_train\_subset = y\_train[:num\_samples]

# Train it with (just) 2 epochs and half the data

model.fit(x\_train\_subset, y\_train\_subset, epochs=2)

The above three steps lowered the accuracy to under 91% (0.9077), which is still not bad, but easier to improve.

**Making model more effective:**

To make the model more effective, the three steps listed above were rolled back.

1. Neurons were bumped to 125 and the activation function was switched to relu.

tf.keras.layers.Dense(150, activation='relu'), # Increased neurons up to 150 and used relu to improve model.

1. Epochs were bumped to 8 and the full training data was used.

# Train it with 8 epochs and full data

model.fit(x\_train, y\_train, epochs=8)

After the above changes, the model’s accuracy jumped to **over 98% (0.9803).**

## 1.2 Comparison of the result with the original output

The original model (after some modification) achieved an accuracy of 0.9289. The probability model produced the following output:

<tf.Tensor: shape=(5, 10), dtype=float32, numpy=

array([[1.1000263e-03, 6.4707501e-04, 2.1277571e-03, 4.5003546e-03,

4.9410516e-04, 3.0914531e-04, 3.0554264e-05, 9.7922391e-01,

2.4208626e-04, 1.1324777e-02],

[1.0976483e-02, 3.7527597e-03, 8.6248595e-01, 6.3958928e-02,

2.1546370e-04, 2.5433626e-02, 1.7427288e-02, 4.6288848e-04,

1.5187029e-02, 9.9573503e-05],

[1.9774670e-04, 9.6232343e-01, 9.5712785e-03, 5.6048646e-03,

8.3665771e-04, 1.5356581e-03, 2.9965965e-03, 6.3423370e-03,

8.5978834e-03, 1.9936161e-03],

[9.8533303e-01, 1.2250569e-04, 2.7189276e-03, 1.5426788e-03,

1.1225416e-04, 3.9215339e-03, 1.7123288e-03, 3.4466460e-03,

2.9701376e-04, 7.9312891e-04],

[7.1512803e-04, 8.9968252e-04, 3.7844738e-03, 6.0112006e-04,

8.6787730e-01, 7.1327132e-03, 1.0090396e-02, 8.2900235e-03,

1.1754143e-02, 8.8855028e-02]], dtype=float32)>

After the modifications specified above, the improved model achieved an accuracy of 0.9803. The probability model produced the following output:

<tf.Tensor: shape=(5, 10), dtype=float32, numpy=

array([[4.69346784e-10, 8.12970871e-12, 5.15578620e-07, 2.08178449e-06,

2.42672167e-11, 1.31815519e-08, 1.05826252e-16, 9.99995947e-01,

4.25251917e-11, 1.59513081e-06],

[6.53053278e-10, 8.27759959e-06, 9.99990582e-01, 1.03168338e-06,

8.59171823e-19, 2.12026894e-08, 2.38588385e-08, 6.55862474e-12,

8.24071336e-08, 6.91888857e-16],

[2.90136626e-08, 9.99329090e-01, 1.91368072e-05, 9.31339230e-07,

2.65980407e-06, 4.85643966e-07, 6.52986182e-06, 6.16225065e-04,

2.48050983e-05, 9.69984555e-08],

[9.99868751e-01, 2.38761060e-11, 2.16848633e-07, 4.92103247e-10,

6.76955665e-08, 6.91781068e-08, 1.24643280e-04, 5.71941291e-06,

7.70536523e-10, 4.00168119e-07],

[5.44614220e-09, 2.61326599e-10, 1.82564435e-07, 3.79575760e-10,

9.98593509e-01, 6.15089935e-09, 1.05871194e-07, 5.17831495e-06,

2.88057089e-09, 1.40100555e-03]], dtype=float32)>

## 1.3 URL of your Colab code

[Colab Notebook Link](https://colab.research.google.com/drive/1lK7Lk6ar98oQpjxZ7_HJ9KjpVef7QE40?usp=sharing)

### >>AI generation for section 1 (or check: I did not use AI generation here \_X\_). Please collapse this section before submitting.

# New application

## 2.1 Description of the application (include description of inputs, functionality, and outputs—in no more than ½ page. Identify a clearly obtainable data source.)

The Movie Review Sentiment Classifier application uses a neural network to perform sentiment analysis on movie reviews. Sentiment analysis, also known as opinion mining, involves building a system to collect and categorize opinions, in this case about the reviewer’s feelings towards a movie.

**Functionality:**

The application could be used by movie producers to gauge public opinion about their movies, by movie recommendation services to recommend movies that have received positive reviews, or by consumers to help decide which movie to watch based on general sentiment towards the movie based on reviews.

**Inputs**:

The input to this application is a movie review in text format. The review could be a single sentence or multiple paragraphs. The source of these reviews is the[IMDB reviews dataset](https://ai.stanford.edu/~amaas/data/sentiment/), which is a dataset of 50,000 movie reviews from the Internet Movie Database (IMDB). The dataset is already divided into 25,000 reviews for training and 25,000 reviews for testing. Each set has an equal number of positive and negative reviews, labeled as such. It is available already as a Keras dataset [2].

**Outputs**:

The output of the application is a sentiment prediction for each review. This prediction will be a binary label, where 1 represents a positive sentiment and 0 represents a negative sentiment. The application could also output a probability score that indicates the confidence of the prediction, as a stretch goal.

## 2.2 Three illustrative input/output pairs from running the implementation

1. Input: A movie review or list of reviews encoded into integer representations of words.

e.g- ["This movie was excellent. I've never loved a movie somuch",

"The movie was ok. I've seen better, but it was mostly fun.",

"I didn't like this movie at all. Terrible film."]

Output: a score from 0 to 1 indicating intent, 1 being most positive and 0 being most negative.

[[0.9282535 ]

[0.3477716 ]

[0.07725603]]

1. Input: an encoded single word of list of words

["excellent", "good", "bad", "terrible"]

Output: a score from 0 to 1 indicating intent, 1 being most positive and 0 being most negative.

[[0.9282535 ]

[0.7777716 ]

[0.12725603]

[0.04353434]]

1. Input: A movie review encoded into integer representations of words.

Output: A binary score of 1 OR 0, with a probability measuring confidence in the decision.

Review 1: Prediction = 1, Confidence = 59.34%

Review 2: Prediction = 0, Confidence = 18.15%

Review 3: Prediction = 0, Confidence = 3.85%

## 2.3 Key code snippets, with explanation (the important code—in no more than a page)

Keras provides the IMDB review dataset to use to train models [2]. The below snippet is taken from the colab notebook and extracts the data to np arrays of training and test data. Setting the num\_words param to 10000 to limit the number of stored words from the default.

# Loading IMDB review data set.

# Available here: https://keras.io/api/datasets/imdb/

imdb\_data = tf.keras.datasets.imdb # data set known to Keras/TensorFlow

# Setting up tensorts for training and testing.

(x\_train, y\_train), (x\_test, y\_test) = imdb\_data.load\_data(num\_words = NUM\_WORDS)

# load\_data() takes several arguments, but starting with mostly defaults. The num\_words

# params limits the size of the word intent dictionary. Lowering it to 10k from the

# standard number to manage size of the dictionary.

# converting everything to np arrays.

x\_train = np.array(x\_train)

y\_train = np.array(y\_train)

x\_test = np.array(x\_test)

y\_train = np.array(y\_train)

Defining the neural net model. Using embedding as it’s recommended for intent classification. Embedding provides a way to convert discrete categorical data into continuous vector representations that can be used in machine learning models. [3]. Additionally, different activation functions are used in the hidden layer and the output layer. The relu function introduces non-linearity into the model, allowing it to learn more complex patterns. The sigmoid activation function is used in the output layer. It squashes the output of the model into the range 0-1. allowing it to be interpreted as a probability [3].

# Define the neural net model

model = tf.keras.models.Sequential([

tf.keras.layers.Embedding(NUM\_WORDS, 2, input\_length=250),

tf.keras.layers.Flatten(),

tf.keras.layers.Dense(250, activation='relu'),

tf.keras.layers.Dense(1, activation='sigmoid')

])

Compiling the model with a cross-entropy cost function, appropriate for classification models. Going with 3 epochs as testing showed 5, my original param, was overfitting the model.

# Compile the model

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model

fit = model.fit(x\_train, y\_train, validation\_data=(x\_test, y\_test), epochs=3, batch\_size=120, verbose=2)

The model info looks like:

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

embedding (Embedding) (None, 250, 2) 20000

flatten (Flatten) (None, 500) 0

dense (Dense) (None, 250) 125250

dense\_1 (Dense) (None, 1) 251

=================================================================

Total params: 145501 (568.36 KB)

Trainable params: 145501 (568.36 KB)

Non-trainable params: 0 (0.00 Byte)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Defining an encode function to convert text reviews into integer representations of words based on the dataset dictionary. Using the converted reviews to test some sample output from the model [2].

# Making some predictions to see sample output.

def encode\_review(review):

#encodes review based on dataset dictionary.

words = review.split()

encoded = [start\_char]

for word in words:

# Check if the word is in the word\_index

if word in word\_index:

encoded.append(word\_index[word] + index\_from)

else:

encoded.append(oov\_char)

return encoded

# Sample reviews. Going for a positive, medium, and negative sentiment review.

sample\_reviews = ["This movie was excellent. I've never loved a movie somuch",

"The movie was ok. I've seen better, but it was mostly fun.",

"I didn't like this movie at all. Terrible film."]

# Encode the reviews

encoded\_reviews = [encode\_review(review) for review in sample\_reviews]

# Pad the sequences

encoded\_reviews = tf.keras.preprocessing.sequence.pad\_sequences(encoded\_reviews, maxlen=500)

# Get predictions

predictions = model.predict(encoded\_reviews)

# Print predictions

print(predictions)

The above samples produced the following output:

[[0.9249888 ]

[0.3462709 ]

[0.07797936]]  
  
Formatted differently with binary choices and confidence scores:

Review 1: Prediction = 1, Confidence = 53.00%

Review 2: Prediction = 1, Confidence = 25.25%

Review 3: Prediction = 0, Confidence = 16.80%

Below is a scatterplot showing the rating of some sample words. Positive words like ‘excellent’ score close to 1, and negative words are towards 0.

1/1 [==============================] - 0s 21ms/step

A graph with blue dots

Description automatically generated

## 2.4 URL of your code (Colab code--or attach and explain how to run if necessary)

The Colab link includes more information on the models and visualizations beyond the code snippets shown above.

[Google Colab Link](https://colab.research.google.com/drive/1OytWG-71g3VZrr4IHG51UwCzYW0PrgBe?usp=sharing)

### >>AI generation for section 2 (or check: I did not use AI generation here \_\_). Please collapse this section before submitting.

1. PARAGRAPH DESCRIBING YOUR VALUE ADDED TO AI-GENERATED MATERIAL

I did not add value to the material in this case. I asked AI for a few options of an application to build, and I selected to work with the first suggestion using the movie review database. I did not use AI working on the code beyond asking it for application ideas.

1. YOUR PROMPT SEQUENCE
2. [1] Given this notebook neural network as a template, can you recommend some ideas for other applications using publicly available data sources?
3. AI Answer:

Sure, there are many publicly available datasets that you can use to train a neural network for various applications. Here are a few ideas:

1. **Sentiment Analysis**: Use the [IMDB reviews dataset](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) to train a neural network to predict the sentiment of a movie review. This is a binary classification problem where the two classes are positive and negative sentiment.
2. **Image Classification**: The [CIFAR-10 and CIFAR-100 datasets](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) are collections of images that are commonly used to train machine learning and computer vision models. They contain 60,000 32x32 color images in 10 and 100 different classes, respectively.
3. **Text Generation**: The [Project Gutenberg](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) offers over 60,000 free eBooks. You could use one of these books to train a recurrent neural network (RNN) to generate text in the style of the book.
4. **Anomaly Detection**: The [KDD Cup 1999 Data](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) is often used for testing anomaly detection methods. It contains a wide variety of intrusions simulated in a military network environment.
5. **Speech Recognition**: The [LibriSpeech dataset](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html" \o "http://www.openslr.org/12/) is a collection of approximately 1000 hours of English speech, which you could use to train a neural network for speech recognition.
6. **Object Detection**: The [COCO dataset](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) is a large-scale object detection, segmentation, and captioning dataset. It contains 330K images (>200K labeled), 1.5 million object instances, 80 object categories, 91 stuff categories, 5 captions per image, 250,000 people with keypoints.

Remember to respect the terms and conditions of these datasets when using them.

# References

You are welcome to use the work of others—but only if you clearly indicate what work is theirs. Failure to do so is plagiarism. Each of your references should occur within the text; so for example [1] should occur below *and* within the body of your response at the relevant location. Include specific sections of the textbooks if used directly.

[1] Geron, Aurelien. *Hands-On Machine Learning with Scikit-learn, Keras, & Tensorflow Third Edition*. “Chapter 10: Introduction to Artificial Neural Networks with Keras”. O’Reilly Media, Inc. October, 2022

[2] *IMDB Movie Review Sentiment Classification Dataset*. Keras Docs. <https://keras.io/api/datasets/imdb/>

[3] The IMDB Dataset: Embedding. https://colab.research.google.com/github/cbernet/maldives/blob/master/imdb/imdb\_embed.ipynb#scrollTo=NuN6jDpv913n

# Evaluation

## 



# Appendix 1

…

# Appendix 2

…