HW9: Introduction to Keras

Due Nov 25 by 11:05am **Points** 100 **Submitting** a file upload **File Types** zip
Available until Nov 25 at 11:05am

This assignment was locked Nov 25 at 11:05am.

Changelog

11/17: Evaluate model output details highlighted

11/18: More explanation about the print_stats()

11/18: Evaluate model's output loss should be formatted with four decimal places (as shown in the sample output).

11/18: Not test the model.metrics_names output

11/20: Change the example in train_model function from "s/example" to "s/step" and removed "Train on X samples" to conform with the CSL machine output

Assignment Goals

- Get TensorFlow (and Keras) set up for your environment
- Familiarize yourself with the tools
- Perform some basic neural network tasks using TF's utilities
- Happy deep learning! :)

Summary

One of the reasons we're using Python in this course is because it has some very powerful machine learning tools, and home-brewing every machine learning solution is not only time-consuming but potentially error-prone (as you may have discovered). It's very useful in practice to use packages like **scikit-learn** (https://scikit-learn.org/stable/), **TensorFlow** (https://www.tensorflow.org/), and even just SciPy and NumPy to support your projects, as their utilities have been developed by a team of professionals and undergo rigorous testing and verification.

In this homework, we'll be exploring the **Keras** (https://www.tensorflow.org/guide/keras) package in TensorFlow and its uses in deep learning.

Part 1: Setting up the Python Virtual Environment

In this assignment, you will familiarize yourself with the Python Virtual Environment. Working in a virtual environment is an important part of working with modern ML platforms, so we want you to get a flavor of that through this assignment. Why do we prefer virtual environments? Virtual environments allow us to install packages within the virtual environment without affecting the host system setup. So you can maintain project-specific packages in respective virtual environments.

We suggest that you use the CS lab computers for this homework. You can also work on your personal system for the initial development, but finally, you will have to test your model on the CSL lab computers.
Find more instructions: **How to access CSL Machines Remotely** **(https://csl.cs.wisc.edu/)**

(https://csl.cs.wisc.edu/) The following are the installation steps for Ubuntu/MacOS (CSL machines are recommended). For Windows installation steps refer to **this.** **(https://www.tensorflow.org/install/pip#windows_1)**

1. You will be working on Python 3 as Tensorflow is not supported on Python 2.

Read more about Tensorflow and Python version **here** **(https://www.tensorflow.org/install)**.

2. To check your Python version use:

```
python -V or python3 -V
```

If you have an alias set for python=python3 then both should show the same versions (3.x.x)

3. Setup a **Python Virtual Environment** **(https://www.tensorflow.org/install/pip#ubuntu-macos)**:

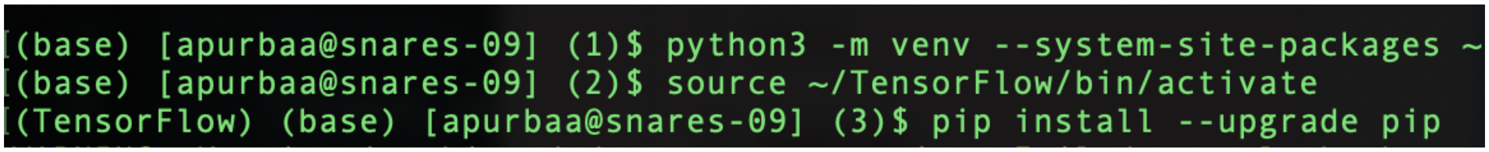
```
python3 -m venv --system-site-packages ~/TensorFlow
```

Here the name of our virtual environment is Tensorflow (you can use any other name if you want).

4. Activate the environment:

```
source ~/TensorFlow/bin/activate
```

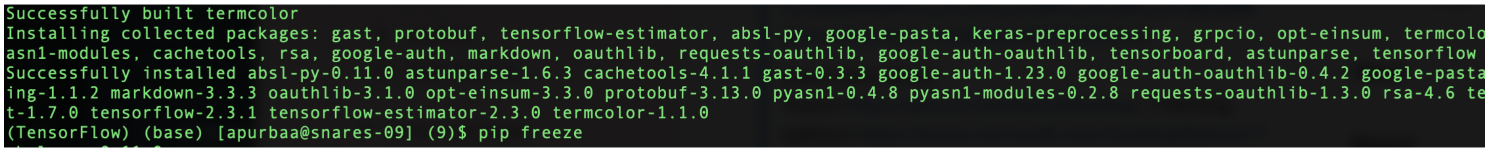
This will activate your virtual environment and your shell prompt is prefixed with `(Tensorflow)`.



```
(base) [apurbaa@snares-09] (1)$ python3 -m venv --system-site-packages ~
(base) [apurbaa@snares-09] (2)$ source ~/TensorFlow/bin/activate
(TensorFlow) (base) [apurbaa@snares-09] (3)$ pip install --upgrade pip
```

5. From your virtual environment shell, run the following commands to successfully upgrade pip and install Tensorflow:

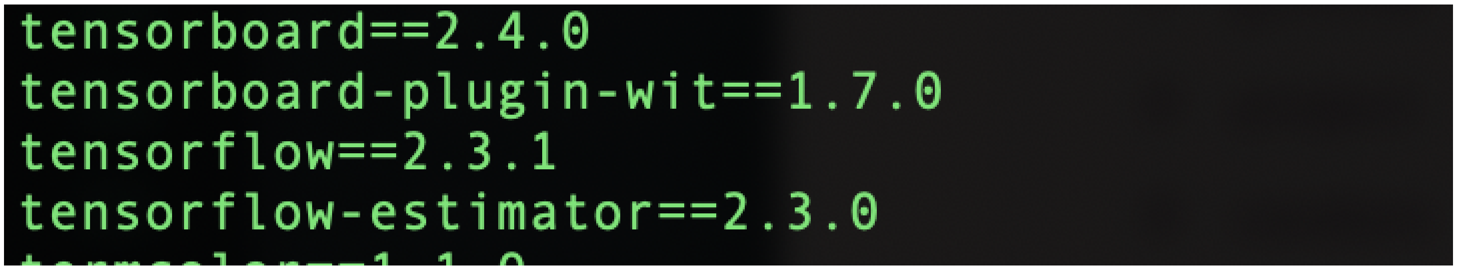
```
pip install --upgrade pip
pip install --upgrade tensorflow
```



```
Successfully built termcolor
Installing collected packages: gast, protobuf, tensorflow-estimator, absl-py, google-pasta, keras-preprocessing, grpcio, opt-einsum, termcolo
asn1-modules, cachetools, rsa, google-auth, markdown, oauthlib, requests-oauthlib, google-auth-oauthlib, tensorboard, astunparse, tensorflow
Successfully installed absl-py-0.11.0 astunparse-1.6.3 cachetools-4.1.1 gast-0.3.3 google-auth-1.23.0 google-auth-oauthlib-0.4.2 google-pasta
ing-1.1.2 markdown-3.3.3 oauthlib-3.1.0 opt-einsum-3.3.0 protobuf-3.13.0 pyasn1-0.4.8 pyasn1-modules-0.2.8 requests-oauthlib-1.3.0 rsa-4.6 te
t-1.7.0 tensorflow-2.3.1 tensorflow-estimator-2.3.0 termcolor-1.1.0
(TensorFlow) (base) [apurbaa@snares-09] (9)$ pip freeze
```

Please make sure that you have the same version of TensorFlow as installed as us. You can check the version of the packages installed using the following command:

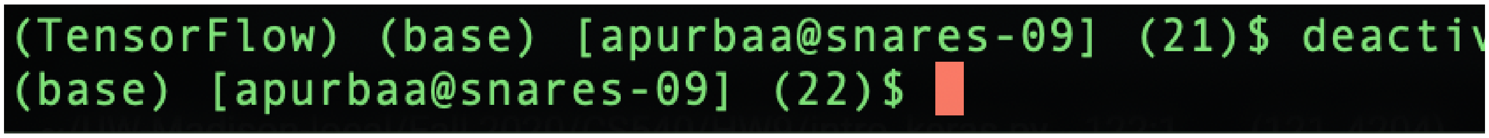
```
pip freeze
```



```
tensorboard==2.4.0
tensorboard-plugin-wit==1.7.0
tensorflow==2.3.1
tensorflow-estimator==2.3.0
```

6. For deactivating the virtual environment:

```
deactivate
```



```
(TensorFlow) (base) [apurbaa@snares-09] (21)$ deactiv
(base) [apurbaa@snares-09] (22)$
```

7. In your implementation you will be using Tensorflow and Keras by importing them as follows:

```
import tensorflow as tf
from tensorflow import keras
```

Deliverable for Part 1:

Once you have set-up the virtual environment, please run the following command:

```
pip freeze > setup_output.txt
```

This command will dump the output of "pip freeze" into a .txt file called **setup_output.txt**. We will be checking that you have installed the correct version of TensorFlow.

Part 2: Program Specification

In this program, you will be using Tensorflow Keras to build a simple deep learning model for predicting labels of images of handwritten images. You will learn how to build, train, evaluate models, and make predictions on test data using this model. This program asks you to implement the following functions in predictions on test data using this model. This program asks you to implement the following functions in Python.

1. **get_dataset(training=True)** —
   Input: an optional boolean argument (default value is True for training dataset)
   Return: two NumPy arrays for the train_images and train_labels
2. **print_stats(train_images, train_labels)** — This function will **print** several statistics about the data
   Input: the dataset and labels produced by the previous function; **does not return anything**
3. **build_model()** — takes no arguments and **returns** an untrained neural network model
4. **train_model(model, train_images, train_labels, T)** — takes the model produced by the previous function and the dataset and labels produced by the first function and trains the data for T epochs; **does not return anything**
5. **evaluate_model(model, test_images, test_labels, show_loss=True)** — takes the trained model produced by the previous function and the test image/labels, and **prints** the evaluation statistics as described below (displaying the loss metric value if and only if the optional parameter has not been set to False); **does not return anything**
6. **predict_label(model, test_images, index)** — takes the trained model and test images, and **prints** the top 3 most likely labels for the image at the given index, along with their probabilities; **does not return anything**

You are free to implement any other utility function. But we will only be testing the functionality using the above 6 APIs, so make sure that each of them follows the exact function signature and returns.
You can also use helper methods to visualize the images from the MNIST dataset for a better understanding of the dataset and the labels. But it is totally optional and does not carry any points.

Get Dataset

Unlike last time, we're not going to make you curate your own dataset — or even download it. Keras contains a few datasets from the National Institute of Standards and Technology (NIST). Typically the "hello world" of datasets is a bunch of images of handwritten numbers, **MNIST (http://yann.lecun.com/exdb/mnist/)** which we will be using for this homework.

For this function, you'll want to use the load_data() function on keras.datasets.mnist to retrieve and return the 2D array of integers and labels representing images of handwritten numbers:

```
mnist = keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

Train images and labels are the data we'll be using to train our neural network; test images and labels are what we'll be using to test that model.

When the optional argument is unspecified, return the **training** images and labels as NumPy arrays. If the optional argument is set to False, return the **testing** images and labels. This function should be the optional argument is set to False, return the **testing** images and labels. This function should be called like so:

```
>>> (train_images, train_labels) = get_dataset()
>>> type(train_images)
<class 'numpy.ndarray'>

>>> type(train_labels)
<class 'numpy.ndarray'>

>>> type(train_labels[0])
<class 'numpy.uint8'>

>>> (test_images, test_labels) = get_dataset(False)
```

Dataset Statistics

This function explores the datasets and labels produced by the previous function.

Note that the labels are themselves just integers, so will want to make use of the following label translations here:

```
class_names = ['Zero', 'One', 'Two', 'Three', 'Four', 'Five', 'Six', 'Seven', 'Eight', 'Nine']
```

Print out the following statistics for the data provided in the arguments:

1. Total number of images in the given dataset
2. Image dimension
3. Number of images corresponding to each of the class labels

For example, the call for print_stats on the training data should output something like this. Similarly, you will be able to get the statistics of the test dataset as well. (note: these numbers are made up):

```
>>> (train_images, train_labels) = get_dataset()
>>> print_stats(train_images, train_labels)
100
10x10
0. Zero - 10
1. One - 10
2. Two - 10
3. Three - 10
4. Four - 10
5. Five - 10
6. Six - 10
7. Seven - 10
8. Eight - 10
9. Nine - 10
```

Your code should work for both the training and testing data, and produce different results for each.

Build Model

Now that you're familiar with the dataset, let's actually build the model we're going to use with that data. Neural networks in Keras are composed of layers. You've heard about these in the lecture, but take a minute to look through the **About Keras Layers** (https://keras.io/layers/about-keras-layers/) documentation page (it's nice and short), to get an idea of what the implementation logistics will look like.

For this assignment, we'll be using some of the **core Keras layers** (https://keras.io/layers/core/) (that's a good resource to check on for layer specifics): Flatten and Dense.

To hold these layers, begin by creating a **Sequential** (https://keras.io/getting-started/sequential-model-guide/) object (that's ALSO a good resource for using Keras models):

```
model = keras.Sequential()
```

and then add the following layers in this order:

1. A Flatten layer to convert the 2D pixel array to a 1D array of numbers; you'll need to specify the input shape here based on your results from `print_stats()` above.
2. A Dense layer with 128 nodes and a ReLU **activation.** (https://keras.io/activations/)
3. A Dense layer with 64 nodes and a ReLU activation.
4. A Dense layer with 10 nodes.

The Flatten layer just serves to reformat the data, but you'll have to train the Dense layers' parameters with the training data before we can use the model.

The model should be compiled using the following params:

- **Optimizer** (https://keras.io/optimizers/): SGD with a learning rate of 0.001.
- **Loss Function** (https://keras.io/losses/): Sparse categorical cross-entropy (with the **from_logits** parameter set to True).
- **Metric** (https://keras.io/metrics/): Accuracy.

Finally, return the compiled model.

Due to some bug in TF v2.2.0 and higher in the model.metrics_names, we will not be testing the output of this model parameter.

```
>>> model = build_model()
>>> print(model)
<tensorflow.python.keras.engine.sequential.Sequential object at 0xADDR>
>>> print(model.loss)
<tensorflow.python.keras.losses.SparseCategoricalCrossentropy object at 0xADDR>
>>> print(model.optimizer)
```

```
<tensorflow.python.keras.optimizer_v2.gradient_descent.SGD object at 0xADDR>
```

(If you don't compile your model, you'll still see the Sequential line, but the other three will not work.)

Train Model

This function is mostly a wrapper function. Use the Keras-provided `model.fit()` function with the training images and labels, with the number of epochs from the parameters, and let Keras do the rest!

This sample output is performed on a subset of the data (for simplicity); your output will be different:

```
>>> train_model(model, train_images, train_labels, 10)

Epoch 1/10
100/100 [==============================] - 0s 2ms/step - loss: 110.3792 - accuracy: 0.1500
Epoch 2/10
100/100 [==============================] - 0s 41us/step - loss: 48.8902 - accuracy: 0.2900
Epoch 3/10
100/100 [==============================] - 0s 38us/step - loss: 9.3743 - accuracy: 0.6200
Epoch 4/10
100/100 [==============================] - 0s 42us/step - loss: 6.6086 - accuracy: 0.6800
Epoch 5/10
100/100 [==============================] - 0s 38us/step - loss: 7.6547 - accuracy: 0.6900
```

Evaluate Model

Now that you've trained the model, you will want to evaluate how good it is. Keras also provides a function to run a test dataset and compare the results to its true labels; again you'll be writing what is effectively a wrapper function for the `model.evaluate().` Please use the function's optional `verbose` parameter to suppress the default output.

The evaluate() function provides two outputs:

```
test_loss, test_accuracy = model.evaluate(images, labels)
```

Format the accuracy output with **two** decimal places and the accuracy should be shown as a percentage. Format the loss with **four** decimal places. (As shown in the sample output below)

Your evaluate function, when using a model trained on the full dataset, should work approximately like this (your results may vary slightly; `trained_model`, `trained_model_1` and `trained_model_2` are versions of the same model trained on exactly the same data):

```
>>> evaluate_model(trained_model_1, test_images, test_labels, show_loss=False)
Accuracy: 94.14%

>>> evaluate_model(trained_model_2, test_images, test_labels, show_loss=False)
Accuracy: 94.30%
```

```
Accuracy: 94.30%

>>> evaluate_model(trained_model, test_images, test_labels)
Loss: 0.1222
Accuracy: 94.30%
```

Predict Label

For testing your model on the test_dataset we will be using this function. **Before** calling this function, please add a **Softmax** (https://keras.io/api/layers/activation_layers/softmax/) layer to interpret the output of your final Dense layer into probabilities. Generally, Softmax is often used as the activation for the last layer of a classification network because the result could be interpreted as a probability distribution of the different categories. You will be adding this Softmax layer for testing your output, but during the grading of assignments, you can assume that this will be provided by the auto-grader. Once inside this function, you may assume the output of the provided model will be in probability form rather than **logits** (https://developers.google.com/machine-learning/glossary#logits) (what the model spits out by default).

Run the `model.predict()` function using the test images, and display the top three most likely class labels for the image at the given index (assumed to be valid) along with their respective probabilities (again, your output will vary in its exact numbers but should be *similar*):

```
>>> predict_label(trained_model, test_images, 1)
Two: 99.88%
Three: 0.08%
Seven: 0.04%
```

Deliverable for Part 2:

A single file named **intro_keras.py** containing the methods mentioned in the program specification section.

Submission Notes

Please submit your files in a zip file named **hw9_<netid>.zip**, where you replace <netid> with your netID (your wisc.edu login). Inside your zip file, there should be two files named: **intro_keras.py** and **setup_output.txt**. Do not submit a Jupyter notebook .ipynb file.

All code should be contained in functions OR under a

```
if __name__=="__main__":
```

check so that it will not run if your code is imported to another program.

Be sure to **remove all debugging output** before submission. Failure to remove debugging output will be **penalized (10pts)**.

If a regrading request isn't justifiable (the initial grade is correct and clear, subject to the instructors' judgment), the request for regrading will be **penalized (10 pts)**.

This assignment is due on Nov 25th, 11:05 AM. We have extended the due date by a day due to requests from some students. There will be no additional extensions beyond this. It is preferable to first submit a version well before the deadline (at least one hour before) and check the content/format of the submission to make sure it's the right version. Then, later update the submission until the deadline if needed.

Introduction to Keras

Criteria	Ratings		Pts
get_dataset() returns the correct values This encompasses both type and contents of the return value and usage of an optional argument.	**20.0 pts Full Marks**	**0.0 pts No Marks**	20.0 pts
print_stats() prints correct values in correct format	**10.0 pts Full Marks**	**0.0 pts No Marks**	10.0 pts
build_model() returns an untrained, compiled Sequential object with the specified components	**10.0 pts Full Marks**	**0.0 pts No Marks**	10.0 pts
train_model() trains the model with the provided dataset for the specified epochs Note: this should produce different results based on whether the provided dataset is the full training data or a random sample	**10.0 pts Full Marks**	**0.0 pts No Marks**	10.0 pts
evaluate_model() displays correctly formatted output	**10.0 pts Full Marks**	**0.0 pts No Marks**	10.0 pts
evaluate_model() displays different output based on its optional argument	**5.0 pts Full Marks**	**0.0 pts No Marks**	5.0 pts
predict_label() displays correctly formatted output	**10.0 pts Full Marks**	**0.0 pts No Marks**	10.0 pts
predict_label() displays expected top-3 categories	**10.0 pts Full Marks**	**0.0 pts No Marks**	10.0 pts
predict_label() displays probabilities, not logits	**5.0 pts Full Marks**	**0.0 pts No Marks**	5.0 pts

Criteria	Ratings		Pts
Python Virtual Environment setup	**10.0 pts Full Marks**	**0.0 pts No Marks**	10.0 pts
Total Points: 100.0			