Lab 11 – Keras

Submission: `lab11.ipynb` pushed to your Git repo on `master`.
Points: 10
Due: Monday, March 30
Pair programming: Optional, but strongly recommended

Objectives

- Learn about the artificial neural nets in Python, and the **Keras** deep learning framework for Python

Installing keras

Before you begin this lab, you will need to configure your conda environment to use the **keras** deep learning framework. Keras is a high level framework that makes it substantially easier to use deep learning for your modeling and prediction needs. It runs on top of tensorflow (and several other deep learning frameworks, though tensorflow is by far the most popular.)

Follow these steps:

1) Open a terminal window
2) Activate your conda environment for csci349
3) `$ conda install keras`
   This will install the `keras` package in your environment. (This will install quite a large number of additional packages, including `tensorflow`, `tensorboard`, `keras-base`, `keras-applications`, `keras-preprocessing`, and several others.)
4) `$ conda install pydot`

That's it!

NOTE: Keep the following note handy - If you have some errors that occur when training the model, reporting an error message about `libiomp5.dylib`, it's possible that you might need to enter this command in your conda environment. (I believe this is Mac dependent. I had to do this on my Mac. You may not.)

```
$ conda install nomkl
```

Exercise 1 – Interview questions about deep learning and keras

Suppose you are interviewing for a job. (If you haven't yet, you will be interviewing sooner than you think!) Interviews for software development positions will assess your personal knowledgebase and your applied skillset. If you do well, then you usually move ahead with a more technical interview, perhaps being required to demonstrate your coding skills on actual problems. So, you need to make a solid impression during that first interview!

Do you pay attention to the science and tech mainstream media sites, blogs, etc. to understand the current trends in industry? If not, you should.

I am a fan of *The Economist*. (https://www.economist.com/) They do a great job not only reporting on current news events around the world, but also giving their readers a good understanding of current and future trends in many different industries. (They also have an app available that aggregates all of their videos in an organized way.) I tend to focus on the *Science and technology* section. It has a nice benefit of giving you a heads up on up-and-coming disruptors in our field. (It also makes for some interesting investment ideas!) Take some time to occasionally watch the videos on their YouTube channel. https://www.youtube.com/channel/UC0p5jTq6Xx_DosDFxVXnWaQ).

If you follow trends in science and technology, then you have probably already heard that many are suggesting that "data is the new oil" (For one example, see: https://www.economist.com/leaders/2017/05/06/the-worlds-most-valuable-resource-is-no-longer-oil-but-data) Data analysis is not just something that research scientists and statisticians are doing. Now, companies in nearly every domain fathomable are working with data! (Yeah, I know. We discussed this the first day of class.) It has also presented some serious challenges to traditional approaches to mining data for patterns. Data has become larger, more complex, more unstructured, capturing many more dimensions than ever. This has made traditional approaches to data analysis (e.g. Excel) ineffective.

You should have also heard about the rise of **deep learning** in this field. If data is the new oil, then deep learning is the equivalent of the best technology to not only extract the oil but also to harness the energy from it in the most efficient way possible that minimizes energy lost. It would work especially well when that oil is deep, convoluted, sparse, and challenging to get to. Ahem… OK, ok, enough of the analogies. Simply put, deep learning has made a huge impact in the data mining community. It has allowed the search for knowledge in data to be taken to the next level.

Before you graduate, you would be doing yourself a huge favor by learning a bit about deep learning. You'll be doing yourself an even greater service if you can get some experience with it. And, that's what this lab (and the next lab, and the next homework) will do.

Deep learning is a very complex, challenging topic in artificial intelligence. Many graduate programs have started to offer entire classes to teach all of the intricacies of deep learning. There is no possible way we can delve deeply into this topic (no pun intended.) We simply do not have the time, given everything else this class covers. However, hopefully this lab and the next homework will whet your appetite a bit and make you hungry to learn more on your own.

For the curious mind, I recommend some of the following resources:

- https://www.deeplearningbook.org/ - Pretty exhaustive, and advanced. But, quite complete and up to date.
- https://github.com/ChristosChristofidis/awesome-deep-learning - A ridiculous number of resources accumulated about deep learning

The questions

Below are some interview questions gathered from numerous online resources related to deep learning, with some questions that are specifically looking at **keras**. Many of these are questions that people contributed from their actual interviewing experiences. Of course, the level of questioning certainly depends on the job you are expected to do! So, if you don't plan on finding a job where any aspect of machine learning or data mining is to be part of your key responsibility, then this will be far more than you need.

Why so many questions? Part of the motivation here is to support the active-learning, guided approach. Part of this is to give you something you can use as a study guide later.

Instructions

Answer the following questions. Again, this is more for you and your future, than it is for me! So, as long as you answer the questions, you may record the answers however you want. If you believe you are solid with the questions, then you should be able to quickly write an answer, and move on. Be as brief or as exhaustive as you believe is enough for you to use as study material (for an exam, or for an interview.) Be sure to include references for yourself, as you will come across some great material online. **It is NOT enough to just copy and paste a URL! Extract the pertinent information from your source to answer the question.** As usual, answer all questions in markdown in your notebook file. (Many of these questions are answered in the Neural Net videos posted, in lieu of lecture.)

1) What is an artificial neural network (ANN)?
2) What is deep learning? How does it relate to an ANN?
3) Name a couple of examples where deep learning has made a tremendous impact.
4) Briefly, what is the *feedforward* algorithm with a neural net?
5) In the context of machine learning, what is a *loss function*?
6) What is gradient descent? And how is the loss function a critical part of gradient descent?
7) Training a neural net involves the *backpropagation* algorithm. In a few sentences, describe what this algorithm does.
8) What is the difference between *batch gradient descent* and *stochastic gradient descent*?
9) In the context of neural network training, explain the terms *epoch* and *batch*.
10) In the context of machine learning, what is a *hyperparameter*?
11) In the context of neural nets training, what are examples of hyperparameters that can affect model performance?
12) What is an activation function?
13) Most agree that the most popular activation functions are sigmoid, hyperbolic tangent (tanh), softmax, and ReLu (rectified linear unit). Compare and contrast each, using whatever resources you want. Again, 1-2 sentences for each Is sufficient.
14) Why is ReLu so popular for large, deep learning networks?
15) Why is softmax most appropriate for the output layer, especially for classification problems?
16) What does ReLu sometimes suffer from, and how does a Leaky ReLu activation address it?

The next few questions are focused around Keras and Tensorflow:

17) What is Tensorflow? Who created it?
18) What are tensors?
19) What is keras? Who created it?
20) Explain the relationship between keras and tensorflow. How are they similar? Different?

The remaining questions all pertain specifically to specific classes or critical methods in Keras. Most of these you can get directly from the Keras documentation. https://keras.io/

21) Describe what the `Sequential` class represents
22) What is a layer? How is a layer added to a model?
23) What is a `Dense` layer?
24) What does the `compile` method do for a model, and what two parameters are required to compile every model?

Clearly, deep learning has countless other topics you could prepare for, including controlling overfitting, regularization, dropout, convolutional neural networks, autoencoders, recurrent networks, and so on. This course will not give you the opportunity to go into those topics. However, for your final project, you are quite welcome, and even *encouraged*, to learn more and use more advanced network and training setups. There are no lack of tutorials available online. I encourage you to keep diving further! (After you finish this lab. ☺)

Exercise 2 – The return of iris

This exercise will rely on you successfully completing lab 10 on classification. If you did not finish that lab, then you will struggle completing this exercise. For this exercise, you will configure the structure of a very basic neural net using `keras`. Once you configure the structure, then you will train the network, evaluate the network on the training data, validate it by generating predictions on test data, and report results. Most of the questions are all in Python, but there will be a couple of reflective exercises to make sure you understand what you are observing.

> A NOTE FOR THOSE WHO HAVE DEEP LEARNING EXPERIENCE from research or another related course: If you have experience programming with keras, then this lab will be SIMPLE! I urge (and even beg) you to make yourself available to others who have never used keras or any neural net framework before. However, even if you've worked on projects using the largest convolutional neural nets, I encourage you to work through this lab anyway, and give feedback to me directly about what you liked, or didn't like. Did you learn anything? Remember when you first learned – was there something you think this lab missed? If you have experience, then work through the lab and feel free to go further. Just make comments about what you are doing differently right in your notebook file, and why you are doing what you are doing. It would be greatly appreciated.

The spirit and approach for this keras lab was modeled by a very simple, but effective tutorial presented on the following page: https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/

You should open a page to the Keras documentation, paying close attention to the section: Getting started: 30 seconds to Keras: https://keras.io/#guiding-principles

Before you get started, be sure to copy over the standard import statements (numpy, pandas, seaborn, and matplotlib.pyplot, etc.) Also add the following:

```
from sklearn.model_selection import train_test_split, KFold
from sklearn.metrics import classification_report, confusion_matrix
from keras.models import Sequential
from keras.layers.core import Dense, Activation
```

1) [P] Copy over your code from the previous lab that read in and pre-processed the **iris** dataset from `seaborn`. You should have a pandas dataframe that contains four numeric variables and one categorical variable representing the target class. You should have one dataframe `X` and a dataframe `y` representing the target class. Do not split your data into training and testing data yet.

   NORMALLY, I would always expect some EDA tasks to be performed to understand the distributions of your data (i.e. the center, shape, spread of your variables, etc). However, you did a lot of that in the previous lab. Generally, these variables have very similar distributions, with centers that are near each other. Thus, technically, you really don't need to standardize your variables. However, for most models we've learned about, it's a good idea. So, if you did not standardize, go ahead and do that to your `X` dataframe now using a z-score standardization. (All variables are numeric, so this is quite straightforward.)

2) [P] Shuffle your data in your data frames. This will be important for later exercises. Read about the `shuffle()` function in `sklearn.utils`. Import it, and use it to shuffle your `X` and `y` data frames. Use `random_state=0`. Remember – it returns the shuffled data! So, be sure to reassign `X` and `y`.

3) [P] Use `train_test_split` to split your data, but this time, let's use an even smaller split, using a 50/50 split, initializing with a random state of 0. (Why? This is a relatively simple dataset. Let's make the problem a bit more challenging by introducing a smaller training data size.)

   Completing this will result in `X_train`, `X_test`, `y_train` and `y_test` data frames, both with 75 instances.

4) [M] How many inputs will your network need to have?

5) [M] Consider the outputs required for a neural network. Remember that the `iris` dataset is a *multi-class* dataset. It has to predict three different, categorical values. How do you represent a multi-class target variable in a model like a neural net? For the `iris` data, what does the final layer of your neural net structure need to look like?

6) [P] Write the code to convert the iris *target* variables (i.e. `y_train` and `y_test`) to a set of *binarized* variables derived from the target class variable (why? Hopefully you figured out why based on your previous answer!)

   With `iris`, this means that the `"species"` variable should be converted to a data frame (or numpy array) of three variables, one representing each species. (HINT: as usual, there are many ways to do this. I like pandas `get_dummies()` or scikit-learn's `OneHotEncoder`.)

OK. You are now going to build a very basic ANN structure with only one hidden layer. Feel free to explore more complex structures, just for practice, but for this simple dataset, you won't need it. For this first exercise, I'll step you through the basics, but you are encouraged to document the &#^!%$ out of each line, as you want to work toward a solid understanding of what you are doing here! Don't just blindly copy and paste what you see online. If you don't document and understand, you will NOT have the guidance and confidence to work through the next assignment. It will be far more challenging.

Each line tells you to complete a task. The process is laid out exhaustively to ensure that you understand every critical step toward building the structure of a neural net, train it, and then use it. (HINT: As stated above, be sure to open a page to the Keras documentation referred to above!)

7) Create an instance of `Sequential()` called `model`.

8) Now, we will sequentially add layers, starting with the hidden layer that receives the input, then you continue adding layers until you get to the output layer. We will keep it simple: one hidden layer, and one output layer.

   Add a `Dense` layer representing the hidden layer. The first layer you add always needs to specify the number of inputs. And, you also need to specify the number of units in the layer (e.g. 9-12 is a good start for these simple data.) Specify an activation function of your choosing. Most basic nets use a `'sigmoid'` or `'tanh'` activation, though deep learning emphasizes `'relu'`. (Be sure you understand why at some point in your near future!) Any of the above is fine.

9) Add one more layer representing the output layer. Be sure to specify the correct number of outputs. Remember, use a `'softmax'` activation here. (NOTE – after you specify your first layer, the number of inputs in further layers added are implied, and thus do not need to be specified.)

10) OK. Now compile your model. Look at the documentation for the `compile()` method. You'll need to specify the following parameters:

a) Choose an `optimizer` – This is a GREATLY HEATED TOPIC on deep learning and ANN blogs. The facts: **Stochastic Gradient Descent** (SGD) is *the* quintessential standard. It is the basis of neural net learning. It is mathematically sound, though it can suffer from the worst computational performance (i.e. timing to convergence.) All enhancements and newer methods are based on SGD. For SGD, an appropriate selection a *momentum* parameter makes a *huge* difference, and papers have come out demonstrating that SGD + a good momentum parameter are as effective as the most hyped optimization techniques, *if you can identify good parameters*. (For example: https://arxiv.org/abs/1705.08292) Selecting good learning rates, momentum and other parameters is not easy. Theoretically, there are an infinite number of combinations you could choose. So, several other methods have come out to help address the challenges of selecting good parameters. Adam is most commonly used optimizer in practice for deep learning. So, use `optimizer='sgd'` to start, but then go back and set it to `optimizer = 'adam'`.

b) Choose the `loss` function – This is the function that gives you the error that is backpropagated. Use `loss='categorical_crossentropy'` for this problem.

c) Choose the performance `metrics` – Typically, you will stick with `metrics=['accuracy']` here. You can do far more with your predictions later.

11) OK, your structure is set. Now you need to train the model. Look at the documentation for the `fit()` method. Use `fit` to train your model with `X_train` and your binarized `y_train` data. There are many additional parameters available that basically control how you perform weight updates. This is where, depending on your data size and your selection of parameters, you could be waiting a while. This is a SIMPLE dataset, and should take no more than a 5-10 seconds to get good results.

    You can start with the following:

a) `epochs` = number of epochs to train the model. An epoch is an iteration over all your training data. You will need to experiment. Start with a value of 100.

b) `batch_size` = number of samples per gradient update. Updating every instance will usually converge the earliest, but with high variance. At the other extreme, batch_size using the entire dataset is slow, but very smooth convergence. Experiment. Use 1, 5, 15. You'll need to select the number of epochs in conjunction with this parameter. For this simple problem, `batch_size = 1` will likely work just fine.

c) `verbose = 1` will show output as training progresses. Very useful!

d) Use `validation_data` to pass your test data. This will make it easy to understand if your model is overfitting your data. (This slows things down a bit more, but it's so important to capture how your model is doing on BOTH training and test data!)

The `fit()` method returns a `History` object. Look up what this is, and store this result! You'll use it in the next step!

Congratulations! You have a trained keras model!

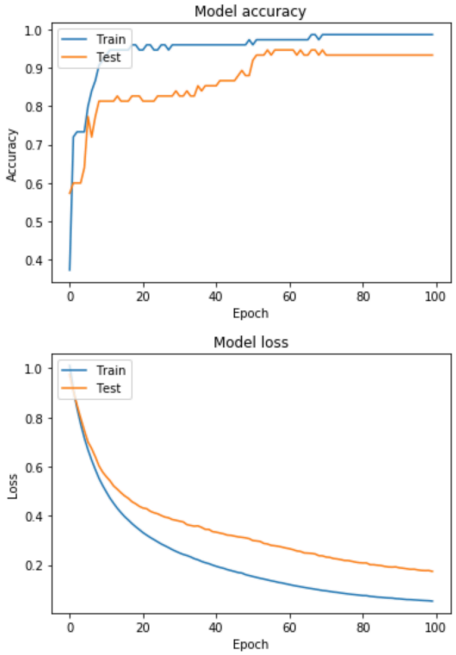
Take a moment, and think about the vast number of different parameters that influence the model that is built. Certainly, the structure of the network itself has a substantial influence. But think about the *hyperparameters*. There are **many**. This is a challenging problem with deep learning models, especially because the problem of performing a search for optimal parameters is nearly impossible. The best you can hope to do is use some type of framework to make hyperparameter tuning a bit easier. (This is coming up in the next lab.)

---

12) [P] It's important to understand your accuracy and loss rates as your model proceeds through training. Visualize the loss on training and test data. Look at the code presented here: https://keras.io/visualization/. You may copy it, or make it more fancy if you choose to do so. Pay attention to the section of code that shows **Training history visualization**. You may also include the code that visualizes the model, as it may be good to know for your future.

    If you did everything correctly, you should show two graphs, one for **model loss**, and one for **model accuracy**.

    Below are my example graphs, based on the code given on the Keras documentation page. I used `'adam'` for optimization, a single hidden layer with 12 units, and `'sigmoid'` activation. I trained with `epochs=100`, and `batch_size=1`. You'll see that these parameters do pretty well for these data. Even the test data performs quite well.



    You can see how useful these plots are. Both loss curves are only *starting* to stabilize! This suggests that it may be worth repeating this with 150 epochs? Or, do we reduce the number of hidden units? Change the optimization? The momentum? Argh! There are so many parameters that can influence getting the best model!

    I shouldn't be telling you this. But, in case you haven't, you should be moving code like this into functions that you can reuse…

13) Try to change some parameters with the model. However, instead of copying and pasting each individual line you wrote above, follow the approach of creating the entire structure in a single line. Use the example laid out in the Keras documention https://keras.io/getting-started/sequential-model-guide/ This will get you started:

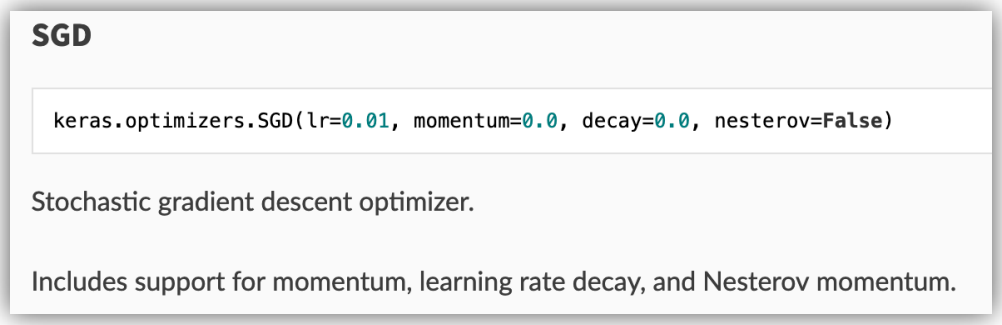
```
model = Sequential([
            Dense(12, input_shape=(4,)),
            Activation( ??? ),
            ...
        ])
```

Clearly, you need to finish the rest. And, you'll need to compile and build your model. However, change the parameters in some way from what you did above. Clearly, as given above, you need 4 inputs, and 3 outputs. Beyond that, you have countless ways to create a different approach! Different structure? An additional hidden layer? Different activation? Or, use the same structure, and use different batch_size for training? Different optimization algorithm? More epochs? Etc. The choices are quite vast!

Generate the same two plots. Compare and contrast your findings.

14) OK, one more time. This time, copy the same model, but use an SGD optimizer. Of course, you may have already chosen this by specifying the `optimizer='sgd'` parameter when you compiled your model. This time, you will instantiate your optimizer.

From Keras documentation:



SGD

```
keras.optimizers.SGD(lr=0.01, momentum=0.0, decay=0.0, nesterov=False)
```

Stochastic gradient descent optimizer.

Includes support for momentum, learning rate decay, and Nesterov momentum.

Copy one of your models above. (**Remember, if you keep using the same model instance, you are continually improving the weights, and thus not evaluating your new model properly! When you experiment with new models, you need to instantiate a new model, or figure out how to reset your weights to random initial values. For now, it's just easy enough to reinstantiate a new model.**)

Now, instantiate `SGD`. Look at the documentation, and choose a different learning rate (`lr`) and a momentum value of some value between 0.5-0.9. Compile and fit your model. Regenerate your accuracy and loss plots. Compare and contrast your findings.

15) Remember, this is a classification problem. Use your model to predict the classes for the test data (using the function `predict_classes`), and store the results as `y_pred`.

16) Finally, using your code from the lab on classification, output the `confusion_matrix` and the `classification_report` (from scikit-learn's `metric` package) to print out the complete performance results.

Nice work!

Exercise 3 – Wrapping keras in the scikit-learn framework.

To make it easier to evaluate your classifier, why not force Keras to play by the rules established by scikit-learn? (Recall your software engineering course – what design pattern is this problem begging for? Keras is an implementation of *Façade* because of how it presents a nice, clean interface to Tensorflow. However, this problem needs an *Adapter*.) Fortunately, we can do this! The Keras framework provides this functionality for you. The class that does this is called **`KerasClassifier`**, and it's in the `keras.wrappers.scikit_learn` package.

Go ahead and pull up the web page in the Keras documentation that describes this class. Before you can instantiate a wrapped class, you must have a function that builds and returns your keras model.

17) Import the above class into your code. Then write a function called `create_keras_model()`. Copy all of your code that creates the `Sequential` instance, adds the layers, activation functions, and compiles it, into this function.

    For example, my function looks as follows:

```
from keras.wrappers.scikit_learn import KerasClassifier

def create_keras_model():
    model = Sequential()
    model.add(Dense(12, input_shape=(4,)))
    model.add(Activation('sigmoid'))
    model.add(Dense(3))
    model.add(Activation('softmax'))
    model.compile(optimizer='adam', loss="categorical_crossentropy",
                  metrics=["accuracy"])

    return model

# Create a wrapped keras model
clf = KerasClassifier(build_fn=create_keras_model, verbose=1, epochs=100,
batch_size=1)
```

18) At this point, you now have a classifier model that behaves like any other scikit-learn classifier! Cool, right?

    So, using the `clf` classifier above, use it just like you would any other classifier. Run the `fit` method on your classifier, just like you did in lab10. Use `X_train` and the one hot encoded `y_train` data. Then, use the `predict()` method to generate class predictions on `X_test`. Store the results in `y_pred`.

19) Use the predictions to generate a `confusion matrix`.

20) Generate a performance report with the `classification_report` function.

21) Now, harness the power of wrapping this class. Use your code from lab10 that performed a full cross validation. For sake of your time, you may set K to 5. Also, you will likely want to disable verbose mode for this, otherwise you'll have a LOT of output. AND, because deep learning models can take a while to train each model, it is a good idea to generate some output in your loop to show that the cross validation is progressing. For example:

```
Starting Fold #1
Fold #1 complete, time = 30.15 sec
Starting Fold #2
Fold #2 complete, time = 30.44 sec
Starting Fold #3
Fold #3 complete, time = 30.52 sec
Starting Fold #4
Fold #4 complete, time = 31.04 sec
Starting Fold #5
Fold #5 complete, time = 31.00 sec
DONE!
```

22) Generate a full confusion matrix and final classification report based on your 5-fold cross validation of the keras model.

Coming up, in the next lab, you will learn how to use `GridSearchCV`.

Deliverables

Commit your **`lab11.ipynb`** file and push them to the remote Git repository. Be sure you have every cell run, and output generated. All plots should have `plt.show()` as the last command to ensure the plot is generated and viewable on Gitlab. DOUBLE CHECK GITLAB AFTER YOU PUSH! I only grade what I can view!