Deep learning using Keras

Ch2. TensorFlow & Keras Libraries

By Eng. Mohammed Marwan Shahin

What is Keras?

Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. It was developed with a focus on enabling fast experimentation.

Because of this, it's very user friendly and allows us to go from idea to implementation with only a few steps.

Keras was a high-level API that sat on top of one of three lower level neural network APIs and acted as a wrapper to to these lower level libraries. These libraries were referred to as Keras backend engines.

You could choose TensorFlow, Theano, or CNTK as the backend engine you'd like to work with.

- TensorFlow
- Theano
- CNTK

TensorFlow

Ultimately, TensorFlow became the most popular backend engine for Keras.

Later, Keras became integrated with the TensorFlow library and now comes completely packaged with it.

Now, when you install TensorFlow, you also automatically get Keras, as it is now part of the TensorFlow library.

Since Keras now comes packaged with TensorFlow, we need to install TensorFlow with the command:

```
pip install tensorflow
```

Keras Model Development Workflow

- **Prepare the data:** Process, filter and select only the required information from the data.
- **Analyze the data :** acquire a good understanding of the data. The better understanding of the data is required to select the correct ANN algorithm.
- **Split data:** Split the data into training and test data set. Test data will be used to evaluate the prediction of the algorithm / Model (once the machine learn) and to cross check the efficiency of the learning process.
- **Compile the model:** Compile the algorithm / model, so that, it can be used further to learn by training and finally do to prediction. This step requires us to choose loss function and Optimizer. loss function and Optimizer are used in learning phase to find the error (deviation from actual output) and do optimization so that the error will be minimized.
- **Fit the model:** The actual learning process will be done in this phase using the training data set.
- **Predict result for unknown value:** Predict the output for the unknown input data (other than existing training and test data)
- **Evaluate model:** Evaluate the model by predicting the output for test data and crosscomparing the prediction with actual result of the test data.
- **Freeze, Modify or choose new algorithm:** Check whether the evaluation of the model is successful. If yes, save the algorithm for future prediction purpose. If not, then modify or choose new algorithm / model and finally, again train, predict and evaluate the model. Repeat the process until the best algorithm (model) is found.

Model

Sequential Model : Sequential model is basically a linear composition of Keras Layers. Sequential model is easy, minimal as well as has the ability to represent nearly all available neural networks.

Layers: Each Keras layer in the Keras model represent the corresponding layer (input layer, hidden layer and output layer) in the actual proposed neural network model. Keras provides a lot of pre-build layers so that any complex neural network can be easily created.

Some of the important Keras layers are specified below,

- Core Layers
- Convolution Layers
- Pooling Layers
- Recurrent Layers

```
1  from tensorflow.keras.models import Sequential
2  from tensorflow.keras.layers import Dense
```

Sequential

The core idea of Sequential API is simply arranging the Keras layers in a sequential order and so, it is called Sequential API. Most of the ANN also has layers in sequential order and the data flows from one layer to another layer in the given order until the data finally reaches the output layer.

A ANN model can be created by simply calling **Sequential()** API as specified below:

```
from keras.models import Sequential

model = Sequential()
```

Layer

A simple python code to represent a neural network model using sequential model is as follows:

1st Way:

```
model = Sequential([Dense(4,activation='relu'),
                    Dense(2,activation='relu'),
                    Dense(1)])
```

2nd Way:

```
model = Sequential()

model.add(Dense(4,activation='relu'))
model.add(Dense(2,activation='relu'))
model.add(Dense(1))
```

Core Modules

Keras also provides a lot of built-in neural network related functions to properly create the Keras model and Keras layers. Some of the function are as follows:

- **Activations module** - Activation function is an important concept in ANN and activation modules provides many activation function like softmax, relu, etc.,
- **Loss module** - Loss module provides loss functions like mean_squared_error, mean_absolute_error, poisson, etc.,
- **Optimizer module** - Optimizer module provides optimizer function like adam, sgd, etc.,

Activations

In machine learning, activation function is a special function used to find whether a specific neuron is activated or not. Basically, the activation function does a nonlinear transformation of the input data and thus enable the neurons to learn better. Output of a neuron depends on the activation function. As you recall the concept of single perception, the output of a perceptron (neuron) is simply the result of the activation function, which accepts the summation of all input multiplied with its corresponding weight plus overall bias, if any available.

relu

Applies Rectified Linear Unit.

```
from keras.models import Sequential
from keras.layers import Activation, Dense

model = Sequential()
model.add(Dense(512, activation='relu' ))
```

Softmax

```
from keras.models import Sequential
from keras.layers import Activation, Dense

model = Sequential()
model.add(Dense(512, activation='softmax'))
```

Sigmoid

Applies Sigmoid function.

```
from keras.models import Sequential
from keras.layers import Activation, Dense

model = Sequential()
model.add(Dense(512, activation='sigmoid'))
```

Loss

Choosing an optimizer and loss ¶

Keep in mind what kind of problem you are trying to solve:

```
# For a multi-class classification problem
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# For a binary classification problem
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# For a mean squared error regression problem
model.compile(optimizer='rmsprop',
              loss='mse')
```

Optimizer

In machine learning, Optimization is an important process which optimize the input weights by comparing the prediction and the loss function. Keras provides quite a few optimizer as a module, optimizers and they are as follows:

SGD: Stochastic gradient descent optimizer.

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

RMSprop: RMSProp optimizer.

```
keras.optimizers.RMSprop(learning_rate=0.001, rho=0.9)
```

Adagrad: Adagrad optimizer.

```
keras.optimizers.Adagrad(learning_rate=0.01)
```

Adadelta: Adadelta optimizer.

```
keras.optimizers.Adadelta(learning_rate=1.0, rho=0.95)
```

Adam: Adam optimizer.

```
keras.optimizers.Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999,
amsgrad=False)
```

Adamax: Adamax optimizer from Adam.

```
keras.optimizers.Adamax(learning_rate=0.002, beta_1=0.9, beta_2=0.999)
```

Nadam: Nesterov Adam optimizer.

```
keras.optimizers.Nadam(learning_rate=0.002, beta_1=0.9, beta_2=0.999)
```

Import the *optimizers* module before using optimizers as specified below:

```
from keras import optimizers
```

Compile the model

Keras model provides a method, compile() to compile the model. The argument and default value of the compile() method is as follows:

```
compile(optimizer, loss=None,
        metrics=None,
        loss_weights=None,
        sample_weight_mode=None,
        weighted_metrics=None,
        target_tensors=None)
```

The important arguments are as follows:

- loss function
- Optimizer
- metrics

Example

A sample code to compile the mode is as follows:

```
from keras import losses
from keras import optimizers
from keras import metrics

model.compile(loss='mean_squared_error',
              optimizer='sgd',
              metrics=[metrics.categorical_accuracy])
```

where,

- loss function is set as **mean_squared_error**
- optimizer is set as **sgd**
- metrics is set as **metrics.categorical_accuracy**

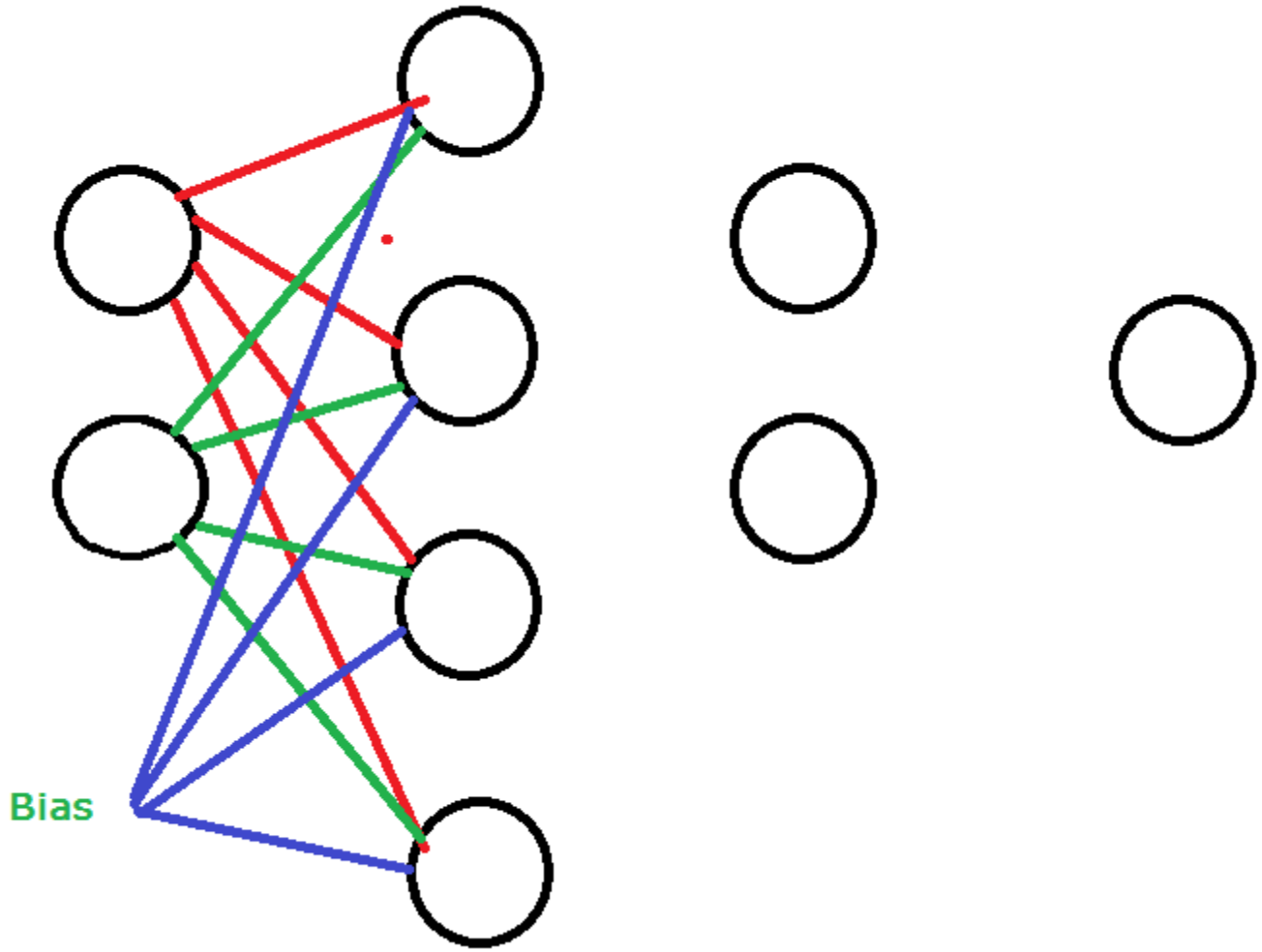
Summaries the model

Understanding the model is very important phase to properly use it for training and prediction purposes. Keras provides a simple method, summary to get the full information about the model and its layers.

A summary of the model created in the previous section is as follows:

```
>>> model.summary()
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_13 (Dense)             (None, 32)                288
_________________________________________________________________
dense_14 (Dense)             (None, 64)                2112
_________________________________________________________________
dense_15 (Dense)             (None, 8)                 520
=================================================================
Total params: 2,920
Trainable params: 2,920
Non-trainable params: 0
_________________________________________________________________
>>>
```

Summaries the model



Train and Predict the model

Model provides function for training, evaluation and prediction process. They are as follows:

- compile: Configure the learning process of the model
- fit: Train the model using the training data
- evaluate: Evaluate the model using the test data
- predict: Predict the results for new input

Model Training

- Models are trained by NumPy arrays using fit(). The main purpose of this fit function is used to evaluate your model on training. This can be also used for graphing model performance. It has the following syntax:

```
model.fit(X, y, epochs=, batch_size=)
```

Here,

- **X, y** - It is a tuple to evaluate your data.
- **epochs** - no of times the model is needed to be evaluated during training.
- **batch_size** - training instances.

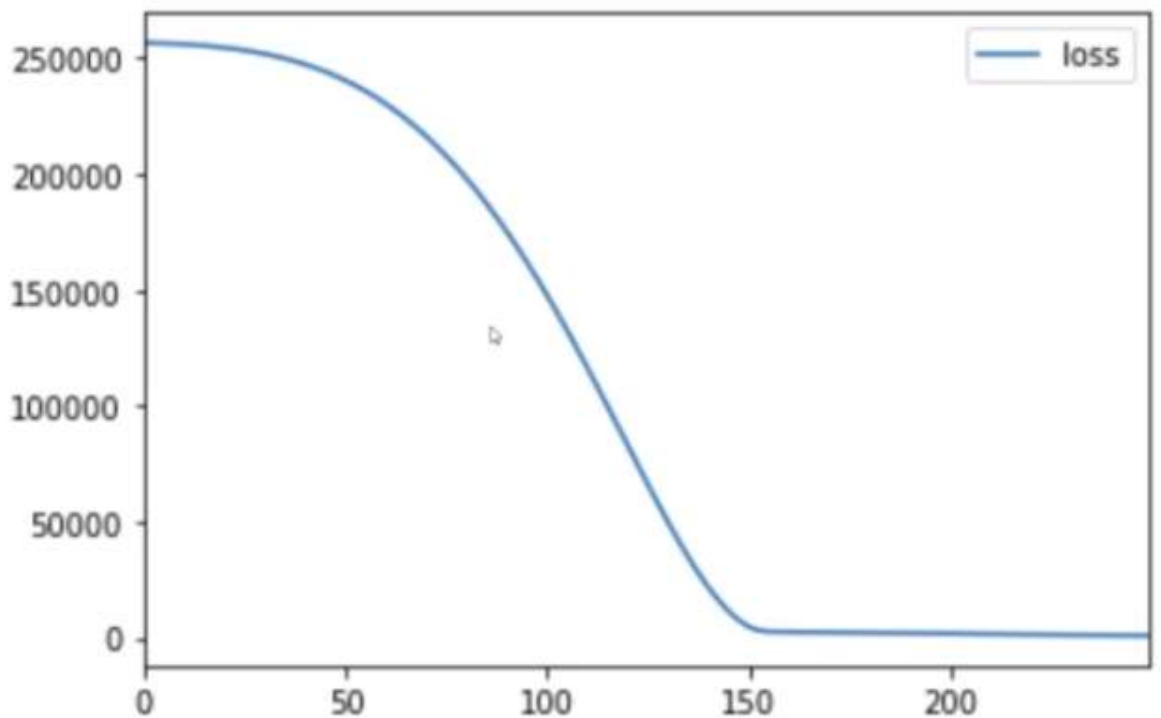
Let us take a simple example of numpy random data to use this concept.

Model.history.history

```
In [79]: loss_df = pd.DataFrame(model.history.history)

In [80]: loss_df.plot()

Out[80]: <matplotlib.axes._subplots.AxesSubplot at 0x1f08676bc88>
```



Metrics

- **In machine learning, Metrics is used to evaluate the performance of your model. It is similar to loss function, but not used in training process. Keras provides quite a few metrics as a module, metrics and they are as follows:**
- Accuracy
- Binary_accuracy
- Categorical_accuracy

Model Evaluation

Evaluation is a process during development of the model to check whether the model is best fit for the given problem and corresponding data. Keras model provides a function, evaluate which does the evaluation of the model.

It has three main arguments:

- Test data
- Test data label
- verbose - true or false

```
score = model.evaluate(x_test, y_test, verbose=0)

print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Demo 1: Create Regression DL model

Model Prediction

Prediction is the final step and our expected outcome of the model generation. Keras provides a method, predict to get the prediction of the trained model. The signature of the predict method is as follows,

```
predict(x,
    batch_size=None,
    verbose=0,
    steps=None,
    callbacks=None,
    max_queue_size=10, workers=1, use_multiprocessing=False)
```

Here, all arguments are optional except the first argument, which refers the unknown input data. The shape should be maintained to get the proper prediction.

Overfitting

Early Stopping: Keras can automatically stop training based on a loss condition on the validation data passed during the model.fit() call

Dropout Layers: Dropout can be added to layers to “turn off” neurons during training to prevent overfitting.

Each dropout layer will “drop” a user-defined percentage of neuron units in the previous layer every batch.

Demo 2: Create DL model for Cancer classification

Project: Create DL model for Bank Loan