I pledge my honor that I have abided by the Stevens Honor System. Kristin Kim

Perceptrons

1. After reading iris data, provide the summary information, by injecting the following code. Include a snapshot.

Iris data is composed of **sepal and petal length and width of 150 iris flowers**

```
iris.data
iris.feature_names
```

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

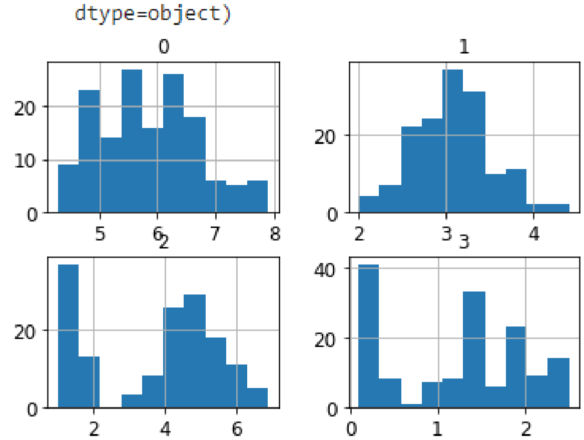
```
iris.target_names
```

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

Based on the histograms, sepal length ranges from about 1 to 8cm, while the width ranges from 2 to 4. The petal length ranges from about 1 to 6 cm, and width ranges from 0.1 cm to 2cm.

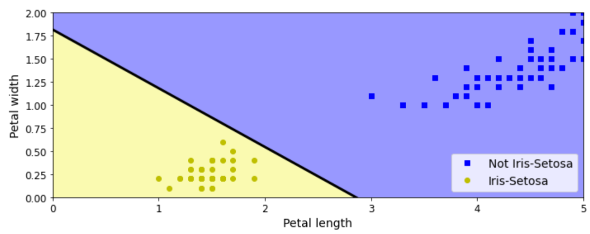
```
import pandas as pd
df_describe = pd.DataFrame(iris.data)
df_describe.hist()
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7ff3d37cca50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7ff3d373b210>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7ff3d36f4810>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7ff3d36a9e10>]],
      dtype=object)
```

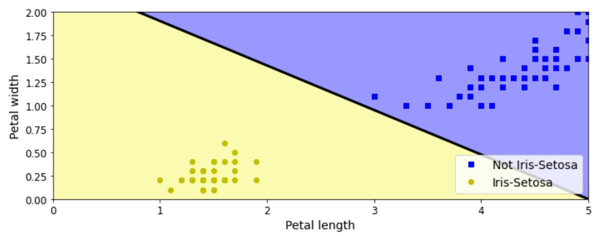


2. Modify the following code to start with a different random seed so that the perceptron_iris_plot in the subsequent code snippet differs. Include snapshots of both classifiers and comment on differences from SVM classfiers.

per_clf = Perceptron(max_iter=1000, tol=1e-3, **random_state=42**)



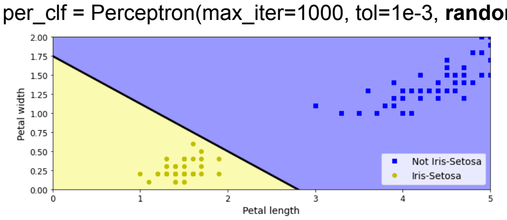
per_clf = Perceptron(max_iter=1000, tol=1e-3, **random_state=20**)



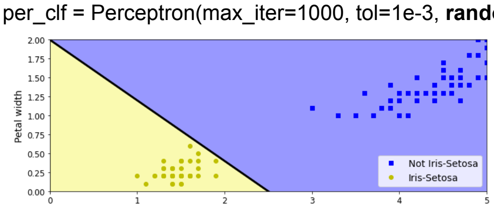
The model of random seed of 42 has a line drawn closer to dots of Iris-Setosa, however, the model with random seed of 20 has the line closed to dots of Not-Iris-Setosa. However, both of them are considered having a bad margin.

Then, I tried few more to see the difference

per_clf = Perceptron(max_iter=1000, tol=1e-3, **random_state=5**)



per_clf = Perceptron(max_iter=1000, tol=1e-3, **random_state=55)**



Building an Image Classifier

3. How many layers are used to classify fashion MNIST dataset? What is the role of softmax in the output layer?

Total 4 layers: **1 flatten layer to flatten the input**, not affecting the input size, and **3 (hidden) dense layers** (dimensionality of the output space of 300, 100, and 10) to generate densely-connected layers

```
[<keras.layers.core.flatten.Flatten at 0x7ffa4b25fb50>,
 <keras.layers.core.dense.Dense at 0x7ffa4b24de90>,
 <keras.layers.core.dense.Dense at 0x7ffa4b24dc50>,
 <keras.layers.core.dense.Dense at 0x7ffa4b24d1d0>]
```

The role of **softmax is to display a probability distribution** at the end when modifying the output layers.

4. How many parameters does the model train? What is the role of biases in the model?

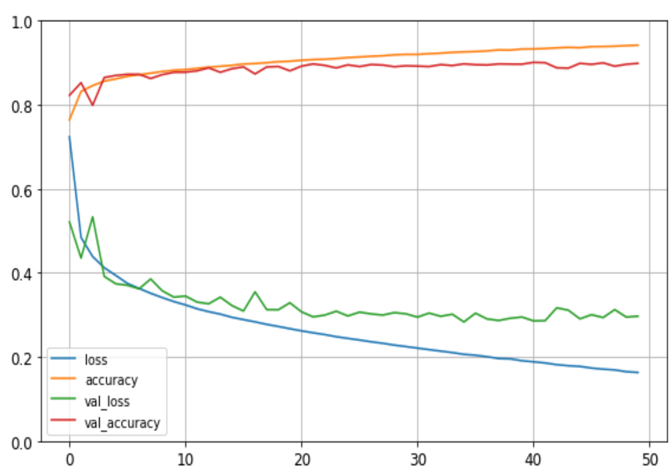
266,610 parameters
By giving biases of the lower layers, the model **learn more of valuable structure** instead of low-level structure that appear in most data

5. Modify the model.fit method to run for 50 epochs. When do you see the model stabilizes? Include a snapshot of the training performance figure.

```
history = model.fit(X_train, y_train, epochs=50,
                    validation_data=(X_valid, y_valid))
```

Based on 4 measures on the keras_learning_curves_plot, the model in the beginning of learning processes is not stable, however, **as it approaches 5 learnings**, the lines seem increasing/decreasing in a stable manner, therefore stabilizing itself.

```
Epoch 30/50
1719/1719 [==============================] - 5s 3ms/step - loss: 0.2247 - accuracy: 0.9199 - val_loss: 0.3026 - val_accuracy: 0.8928
Epoch 31/50
1719/1719 [==============================] - 5s 3ms/step - loss: 0.2211 - accuracy: 0.9199 - val_loss: 0.2948 - val_accuracy: 0.8920
Epoch 32/50
1719/1719 [==============================] - 5s 3ms/step - loss: 0.2175 - accuracy: 0.9216 - val_loss: 0.3043 - val_accuracy: 0.8910
Epoch 33/50
1719/1719 [==============================] - 5s 3ms/step - loss: 0.2139 - accuracy: 0.9228 - val_loss: 0.2967 - val_accuracy: 0.8956
Epoch 34/50
1719/1719 [==============================] - 5s 3ms/step - loss: 0.2104 - accuracy: 0.9247 - val_loss: 0.3017 - val_accuracy: 0.8934
Epoch 35/50
1719/1719 [==============================] - 5s 3ms/step - loss: 0.2063 - accuracy: 0.9260 - val_loss: 0.2833 - val_accuracy: 0.8974
Epoch 36/50
1719/1719 [==============================] - 5s 3ms/step - loss: 0.2041 - accuracy: 0.9267 - val_loss: 0.3041 - val_accuracy: 0.8954
Epoch 37/50
1719/1719 [==============================] - 5s 3ms/step - loss: 0.2006 - accuracy: 0.9282 - val_loss: 0.2904 - val_accuracy: 0.8948
Epoch 38/50
1719/1719 [==============================] - 5s 3ms/step - loss: 0.1963 - accuracy: 0.9306 - val_loss: 0.2868 - val_accuracy: 0.8972
Epoch 39/50
1719/1719 [==============================] - 5s 3ms/step - loss: 0.1955 - accuracy: 0.9302 - val_loss: 0.2921 - val_accuracy: 0.8968
Epoch 40/50
1719/1719 [==============================] - 5s 3ms/step - loss: 0.1910 - accuracy: 0.9327 - val_loss: 0.2950 - val_accuracy: 0.8966
Epoch 41/50
1719/1719 [==============================] - 5s 3ms/step - loss: 0.1885 - accuracy: 0.9329 - val_loss: 0.2858 - val_accuracy: 0.9010
Epoch 42/50
1719/1719 [==============================] - 5s 3ms/step - loss: 0.1858 - accuracy: 0.9339 - val_loss: 0.2863 - val_accuracy: 0.9002
Epoch 43/50
1719/1719 [==============================] - 5s 3ms/step - loss: 0.1816 - accuracy: 0.9355 - val_loss: 0.3170 - val_accuracy: 0.8880
Epoch 44/50
1719/1719 [==============================] - 5s 3ms/step - loss: 0.1792 - accuracy: 0.9366 - val_loss: 0.3113 - val_accuracy: 0.8872
Epoch 45/50
1719/1719 [==============================] - 5s 3ms/step - loss: 0.1776 - accuracy: 0.9359 - val_loss: 0.2907 - val_accuracy: 0.8986
Epoch 46/50
1719/1719 [==============================] - 5s 3ms/step - loss: 0.1736 - accuracy: 0.9383 - val_loss: 0.3005 - val_accuracy: 0.8962
Epoch 47/50
1719/1719 [==============================] - 5s 3ms/step - loss: 0.1710 - accuracy: 0.9386 - val_loss: 0.2938 - val_accuracy: 0.8998
Epoch 48/50
1719/1719 [==============================] - 5s 3ms/step - loss: 0.1692 - accuracy: 0.9394 - val_loss: 0.3125 - val_accuracy: 0.8918
Epoch 49/50
1719/1719 [==============================] - 5s 3ms/step - loss: 0.1650 - accuracy: 0.9408 - val_loss: 0.2948 - val_accuracy: 0.8964
Epoch 50/50
1719/1719 [==============================] - 5s 3ms/step - loss: 0.1632 - accuracy: 0.9419 - val_loss: 0.2970 - val_accuracy: 0.8986
```

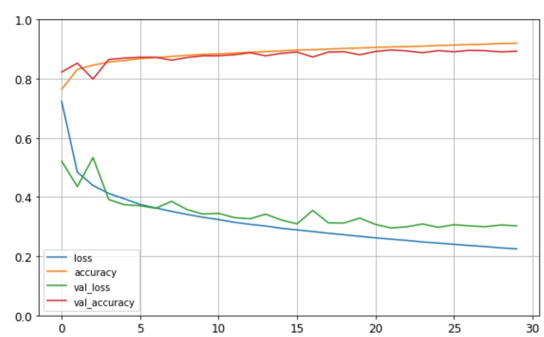


6. Revert model to use 30 epochs. Compare testing performance (include snapshots of both testing results).

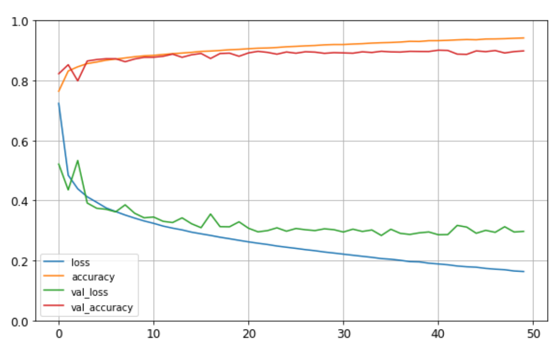
```
history = model.fit(X_train, y_train, epochs=30,
                    validation_data=(X_valid, y_valid))
```

```
Epoch 10/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.3318 - accuracy: 0.8822 - val_loss: 0.3423 - val_accuracy: 0.8774
Epoch 11/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.3238 - accuracy: 0.8838 - val_loss: 0.3449 - val_accuracy: 0.8776
Epoch 12/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.3147 - accuracy: 0.8867 - val_loss: 0.3306 - val_accuracy: 0.8808
Epoch 13/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.3078 - accuracy: 0.8894 - val_loss: 0.3265 - val_accuracy: 0.8880
Epoch 14/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.3019 - accuracy: 0.8915 - val_loss: 0.3422 - val_accuracy: 0.8774
Epoch 15/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.2944 - accuracy: 0.8938 - val_loss: 0.3224 - val_accuracy: 0.8858
Epoch 16/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.2888 - accuracy: 0.8970 - val_loss: 0.3093 - val_accuracy: 0.8902
Epoch 17/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.2835 - accuracy: 0.8981 - val_loss: 0.3549 - val_accuracy: 0.8734
Epoch 18/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.2775 - accuracy: 0.9001 - val_loss: 0.3128 - val_accuracy: 0.8900
Epoch 19/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.2726 - accuracy: 0.9025 - val_loss: 0.3123 - val_accuracy: 0.8912
Epoch 20/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.2672 - accuracy: 0.9037 - val_loss: 0.3289 - val_accuracy: 0.8808
Epoch 21/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.2621 - accuracy: 0.9059 - val_loss: 0.3074 - val_accuracy: 0.8920
Epoch 22/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.2575 - accuracy: 0.9075 - val_loss: 0.2955 - val_accuracy: 0.8972
Epoch 23/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.2533 - accuracy: 0.9084 - val_loss: 0.2995 - val_accuracy: 0.8938
Epoch 24/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.2482 - accuracy: 0.9100 - val_loss: 0.3089 - val_accuracy: 0.8878
Epoch 25/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.2441 - accuracy: 0.9124 - val_loss: 0.2975 - val_accuracy: 0.8950
Epoch 26/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.2403 - accuracy: 0.9137 - val_loss: 0.3066 - val_accuracy: 0.8910
Epoch 27/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.2361 - accuracy: 0.9155 - val_loss: 0.3025 - val_accuracy: 0.8958
Epoch 28/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.2326 - accuracy: 0.9166 - val_loss: 0.2996 - val_accuracy: 0.8946
Epoch 29/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.2282 - accuracy: 0.9189 - val_loss: 0.3055 - val_accuracy: 0.8906
Epoch 30/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.2247 - accuracy: 0.9199 - val_loss: 0.3026 - val_accuracy: 0.8928
```

Comparison of performance:



(30 epochs)



(50 epochs)

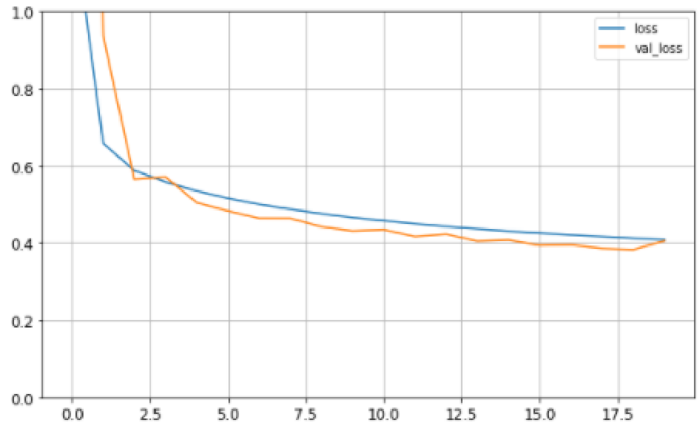
```
Epoch 30/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.2247 - accuracy: 0.9199 - val_loss: 0.3026 - val_accuracy: 0.8928
```

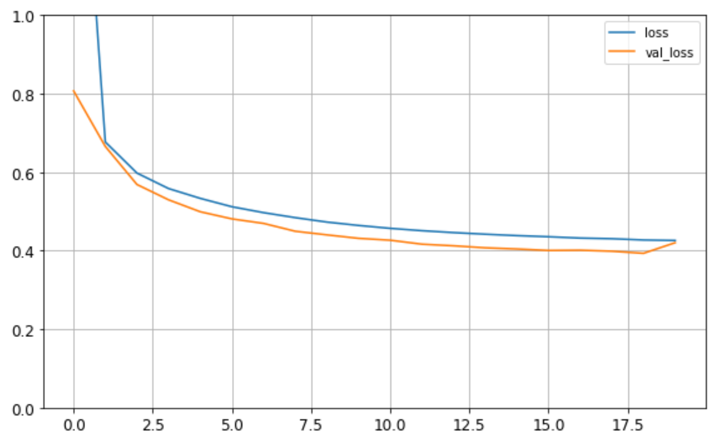
```
Epoch 50/50
1719/1719 [==============================] - 5s 3ms/step - loss: 0.1632 - accuracy: 0.9419 - val_loss: 0.2970 - val_accuracy: 0.8986
```

I re-compiled the model with epochs of 30, the accuracy **didn't reach as high as it did for 50 epochs**, however, it made a good progress on **acquiring a accuracy of 92%**

Functional API & The subclassing API

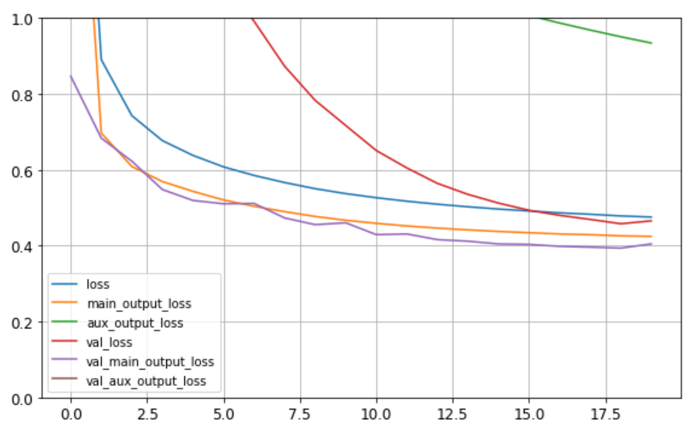
7. [2 points] Copy the following code to show the progression of each model performance. Note that history is updated after each run, so you need to insert the code between model runs. Compare model performances.





The first model's loss drops quicker than the 2nd model's loss

The graph after adding the auxiliary output,



Saving and Restoring

8. What are the benefits of saving (a) the model and (b) the weights? Which methods help save them? Include a snapshot of saved files.

a) Knowing that training a model can take hours, even days, it's useful to have all the details necessary to generate the mode saved for next build
b) In the future, we can load saved weights that have been updated based on the errors or loss.

After constructing and compiling the model by keras.models.Sequential() function, **.save() and .save_weights() functions from keras library** were evoked from that model.

Snapshot of saved files:

checkpoint	11/13/2021 4:48 PM	File
my_keras_model.h5	11/13/2021 4:51 PM	H5 File
my_keras_weights.ckpt.data-00000-of-00...	11/13/2021 4:48 PM	DATA-00000-OF-0...
my_keras_weights.ckpt.index	11/13/2021 4:48 PM	INDEX File

```
[180] model.save("my_keras_model.h5")

[181] model = keras.models.load_model("my_keras_model.h5")

[182] model.predict(X_new)

      array([[0.54002357],
             [1.6505971 ],
             [3.009824  ]], dtype=float32)

[183] model.save_weights("my_keras_weights.ckpt")

      model.load_weights("my_keras_weights.ckpt")

      <tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7fe2699a8390>
```

Exercise Solutions 10

9. What contributed to achieving over 98% accuracy?

The DNN Classifier