Artificial Neural Networks

Artificial Neural Networks (ANN) is a supervised learning system built of a large number of simple elements, called neurons or perceptrons. Each neuron can make simple decisions, and feeds those decisions to other neurons, organized in interconnected layers. Together, the neural network can emulate almost any function, and answer practically any question, given enough training samples and computing power.

To learn more about the basic concepts of neural networks, visit [this page.]()
If you want to learn about backpropagation, go [here.]()

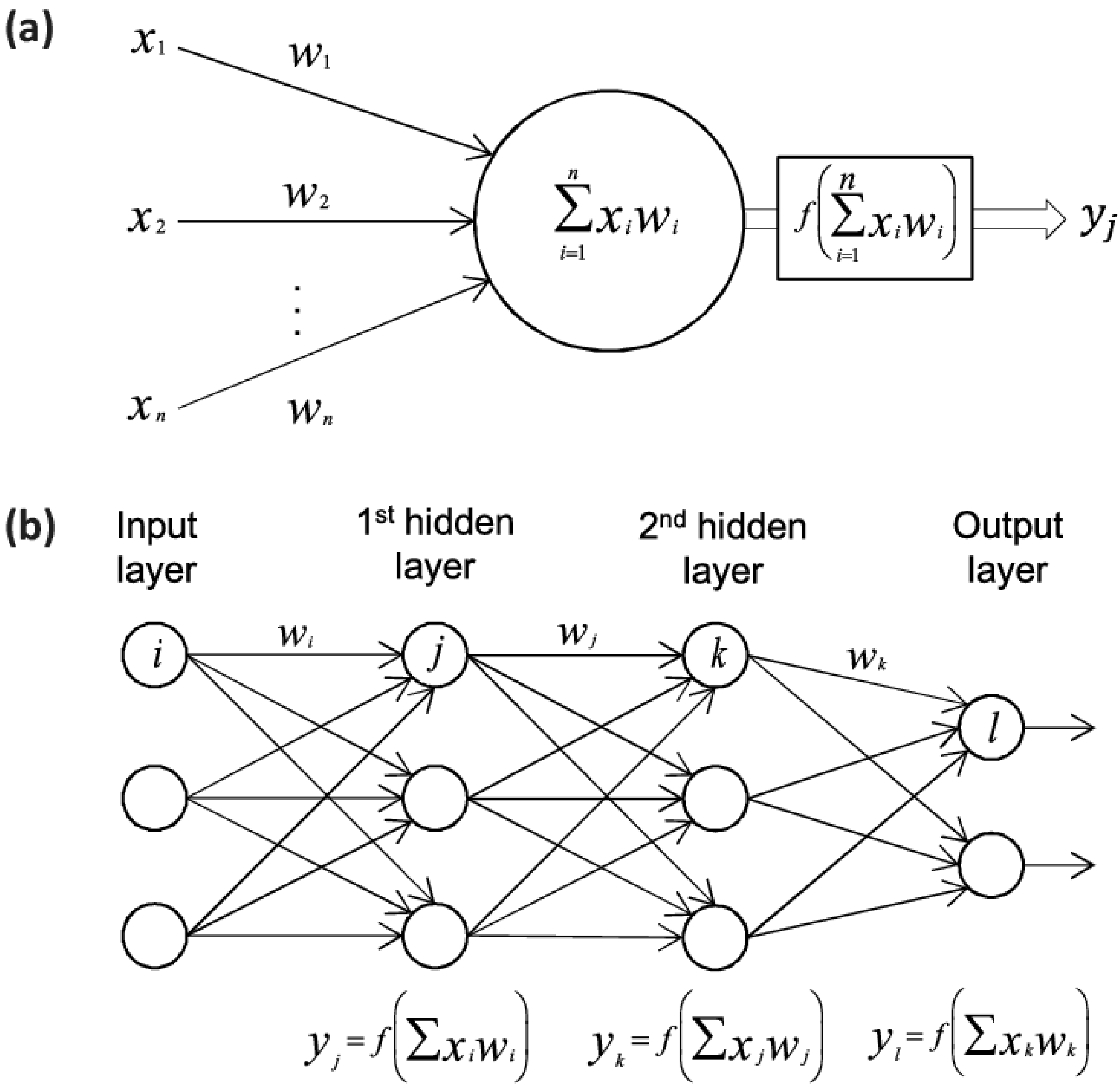


Image of a Perceptron and a Neural Network

Imports

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Load Dataset

```
from sklearn.datasets import load_breast_cancer # Importing dataset from sklearn
```

```
cancer = load_breast_cancer()   # Loading the dataset into cancer variable
```

```
cancer.keys()   # Checking the keys of the dataset dictionary
```

```
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names', 'filename'])
```

```
print(cancer['DESCR'])   # Checking dataset description
```

```
.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class

    :Attribute Information:
        - radius (mean of distances from center to points on the perimeter)
        - texture (standard deviation of gray-scale values)
        - perimeter
        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
        - concave points (number of concave portions of the contour)
        - symmetry
        - fractal dimension ("coastline approximation" - 1)

        The mean, standard error, and "worst" or largest (mean of the three
        largest values) of these features were computed for each image,
        resulting in 30 features.  For instance, field 3 is Mean Radius, field
        13 is Radius SE, field 23 is Worst Radius.

        - class:
                - WDBC-Malignant
                - WDBC-Benign
```

```
:Summary Statistics:

===================================== ====== ======
                                       Min    Max
===================================== ====== ======
radius (mean):                        6.981  28.11
texture (mean):                       9.71   39.28
perimeter (mean):                     43.79  188.5
area (mean):                          143.5  2501.0
smoothness (mean):                    0.053  0.163
compactness (mean):                   0.019  0.345
concavity (mean):                     0.0    0.427
concave points (mean):                0.0    0.201
symmetry (mean):                      0.106  0.304
fractal dimension (mean):             0.05   0.097
radius (standard error):              0.112  2.873
texture (standard error):             0.36   4.885
perimeter (standard error):           0.757  21.98
area (standard error):                6.802  542.2
smoothness (standard error):          0.002  0.031
compactness (standard error):         0.002  0.135
concavity (standard error):           0.0    0.396
concave points (standard error):      0.0    0.053
symmetry (standard error):            0.008  0.079
fractal dimension (standard error):   0.001  0.03
radius (worst):                       7.93   36.04
texture (worst):                      12.02  49.54
```

```
cancer['feature_names']   # Checking all the column names
```

```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23')
```

Feature Selection

```
X = cancer['data']  # Independant variables
```

```
y = cancer['target']  # Dependant variable
```

```
cancer['target_names']
```

```
array(['malignant', 'benign'], dtype='<U9')
```

```
X[0:5]
```

```
array([[1.799e+01, 1.038e+01, 1.228e+02, 1.001e+03, 1.184e-01, 2.776e-01,
        3.001e-01, 1.471e-01, 2.419e-01, 7.871e-02, 1.095e+00, 9.053e-01,
        8.589e+00, 1.534e+02, 6.399e-03, 4.904e-02, 5.373e-02, 1.587e-02,
        3.003e-02, 6.193e-03, 2.538e+01, 1.733e+01, 1.846e+02, 2.019e+03,
        1.622e-01, 6.656e-01, 7.119e-01, 2.654e-01, 4.601e-01, 1.189e-01],
       [2.057e+01, 1.777e+01, 1.329e+02, 1.326e+03, 8.474e-02, 7.864e-02,
        8.690e-02, 7.017e-02, 1.812e-01, 5.667e-02, 5.435e-01, 7.339e-01,
        3.398e+00, 7.408e+01, 5.225e-03, 1.308e-02, 1.860e-02, 1.340e-02,
        1.389e-02, 3.532e-03, 2.499e+01, 2.341e+01, 1.588e+02, 1.956e+03,
        1.238e-01, 1.866e-01, 2.416e-01, 1.860e-01, 2.750e-01, 8.902e-02],
       [1.969e+01, 2.125e+01, 1.300e+02, 1.203e+03, 1.096e-01, 1.599e-01,
        1.974e-01, 1.279e-01, 2.069e-01, 5.999e-02, 7.456e-01, 7.869e-01,
        4.585e+00, 9.403e+01, 6.150e-03, 4.006e-02, 3.832e-02, 2.058e-02,
        2.250e-02, 4.571e-03, 2.357e+01, 2.553e+01, 1.525e+02, 1.709e+03,
        1.444e-01, 4.245e-01, 4.504e-01, 2.430e-01, 3.613e-01, 8.758e-02],
       [1.142e+01, 2.038e+01, 7.758e+01, 3.861e+02, 1.425e-01, 2.839e-01,
        2.414e-01, 1.052e-01, 2.597e-01, 9.744e-02, 4.956e-01, 1.156e+00,
        3.445e+00, 2.723e+01, 9.110e-03, 7.458e-02, 5.661e-02, 1.867e-02,
        5.963e-02, 9.208e-03, 1.491e+01, 2.650e+01, 9.887e+01, 5.677e+02,
        2.098e-01, 8.663e-01, 6.869e-01, 2.575e-01, 6.638e-01, 1.730e-01],
       [2.029e+01, 1.434e+01, 1.351e+02, 1.297e+03, 1.003e-01, 1.328e-01,
        1.980e-01, 1.043e-01, 1.809e-01, 5.883e-02, 7.572e-01, 7.813e-01,
        5.438e+00, 9.444e+01, 1.149e-02, 2.461e-02, 5.688e-02, 1.885e-02,
        1.756e-02, 5.115e-03, 2.254e+01, 1.667e+01, 1.522e+02, 1.575e+03,
        1.374e-01, 2.050e-01, 4.000e-01, 1.625e-01, 2.364e-01, 7.678e-02]])
```

```
y[0:30]
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0])
```

Train Test Split

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=101)
```

Feature Scaling

To learn more about Feature Scaling visit [this link.]()
If you are confused between Standardization and Normalization, check out [this article]()

```
from sklearn.preprocessing import MinMaxScaler
```

```
scaler = MinMaxScaler()
```

```
X_train[0] # Before scaling
```

```
array([1.317e+01, 1.822e+01, 8.428e+01, 5.373e+02, 7.466e-02, 5.994e-02,
       4.859e-02, 2.870e-02, 1.454e-01, 5.549e-02, 2.023e-01, 6.850e-01,
       1.236e+00, 1.689e+01, 5.969e-03, 1.493e-02, 1.564e-02, 8.463e-03,
       1.093e-02, 1.672e-03, 1.490e+01, 2.389e+01, 9.510e+01, 6.876e+02,
       1.282e-01, 1.965e-01, 1.876e-01, 1.045e-01, 2.235e-01, 6.925e-02])
```

```
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

# Alternative
# X_train = scaler.fit_transform(X_train)
# X_test = scaler.transform(X_test)
```

If you are confused regarding why only training data was used to fit the scaler, check out the answers in [this stackoverflow page.]()

```
X_train[0]  # After Scaling
```

```
array([0.30280346, 0.28779168, 0.28292922, 0.16704136, 0.19888056,
       0.12440955, 0.11384724, 0.14264414, 0.1989899 , 0.11647009,
       0.03728187, 0.07124851, 0.02677025, 0.01806076, 0.19688208,
       0.09521735, 0.03949495, 0.16031445, 0.0428885 , 0.02685074,
       0.24795446, 0.31636461, 0.22257085, 0.12347621, 0.37660965,
       0.16416839, 0.14984026, 0.35910653, 0.13207175, 0.09320478])
```

ANN Model

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
# You will see the usage of dropout later
```

```
X_train.shape
```

```
(426, 30)
```

```
model = Sequential()

# Input layer (30 neurons for 30 inputs)
model.add(Dense(units=30, activation='relu'))

# Hidden Layer
```

```
model.add(Dense(units=15, activation='relu'))

# Output Layer
model.add(Dense(units=1, activation='sigmoid'))
# For multi-class units=no. of classes, activation='softmax'

# For binary classification, loss='binary_crossentropy', metrics=['accuracy']
# For multi-class classification, loss='categorical_crossentropy', metrics=['accuracy']
# For regression, loss='mse'
# Adam optimzer is a great algorithm for adaptive gradient descent
# Adam mostly out-performs other optimizers

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

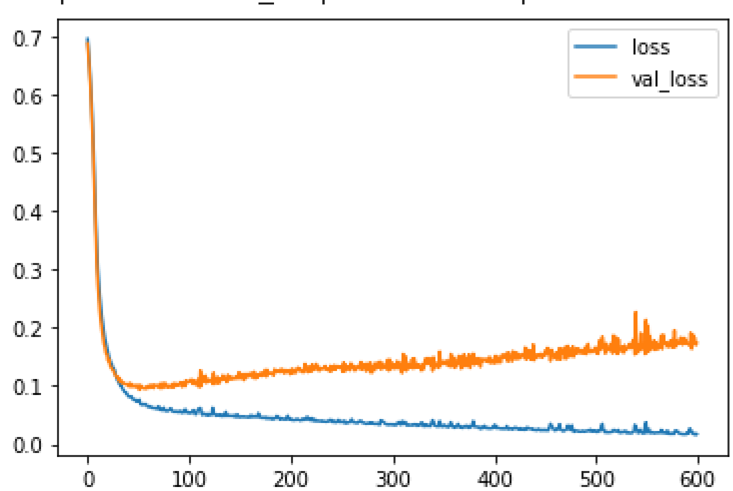
```
model.fit(
  x=X_train,
  y=y_train,
  epochs=600,
  validation_data=(X_test, y_test),
  verbose=1
)
```

```
Epoch 1/600
14/14 [==============================] - 0s 13ms/step - loss: 0.6967 - accuracy: 0.62
Epoch 2/600
14/14 [==============================] - 0s 2ms/step - loss: 0.6744 - accuracy: 0.633
Epoch 3/600
14/14 [==============================] - 0s 2ms/step - loss: 0.6491 - accuracy: 0.725
Epoch 4/600
14/14 [==============================] - 0s 2ms/step - loss: 0.6197 - accuracy: 0.753
Epoch 5/600
14/14 [==============================] - 0s 2ms/step - loss: 0.5869 - accuracy: 0.823
Epoch 6/600
14/14 [==============================] - 0s 2ms/step - loss: 0.5482 - accuracy: 0.852
Epoch 7/600
14/14 [==============================] - 0s 2ms/step - loss: 0.5013 - accuracy: 0.861
Epoch 8/600
14/14 [==============================] - 0s 3ms/step - loss: 0.4409 - accuracy: 0.885
Epoch 9/600
14/14 [==============================] - 0s 2ms/step - loss: 0.3866 - accuracy: 0.896
Epoch 10/600
14/14 [==============================] - 0s 2ms/step - loss: 0.3414 - accuracy: 0.896
Epoch 11/600
14/14 [==============================] - 0s 2ms/step - loss: 0.3038 - accuracy: 0.908
Epoch 12/600
14/14 [==============================] - 0s 2ms/step - loss: 0.2766 - accuracy: 0.915
Epoch 13/600
14/14 [==============================] - 0s 2ms/step - loss: 0.2552 - accuracy: 0.917
Epoch 14/600
14/14 [==============================] - 0s 3ms/step - loss: 0.2360 - accuracy: 0.920
Epoch 15/600
14/14 [==============================] - 0s 2ms/step - loss: 0.2196 - accuracy: 0.917
Epoch 16/600
14/14 [==============================] - 0s 2ms/step - loss: 0.2062 - accuracy: 0.927
```

```
Epoch 17/600
14/14 [==============================] - 0s 2ms/step - loss: 0.1941 - accuracy: 0.924
Epoch 18/600
14/14 [==============================] - 0s 3ms/step - loss: 0.1826 - accuracy: 0.929
Epoch 19/600
14/14 [==============================] - 0s 3ms/step - loss: 0.1747 - accuracy: 0.929
Epoch 20/600
14/14 [==============================] - 0s 2ms/step - loss: 0.1680 - accuracy: 0.941
Epoch 21/600
14/14 [==============================] - 0s 2ms/step - loss: 0.1588 - accuracy: 0.936
Epoch 22/600
14/14 [==============================] - 0s 2ms/step - loss: 0.1506 - accuracy: 0.955
Epoch 23/600
14/14 [==============================] - 0s 3ms/step - loss: 0.1454 - accuracy: 0.953
Epoch 24/600
14/14 [==============================] - 0s 3ms/step - loss: 0.1400 - accuracy: 0.953
Epoch 25/600
14/14 [==============================] - 0s 3ms/step - loss: 0.1370 - accuracy: 0.948
Epoch 26/600
14/14 [==============================] - 0s 3ms/step - loss: 0.1300 - accuracy: 0.955
Epoch 27/600
14/14 [==============================] - 0s 2ms/step - loss: 0.1297 - accuracy: 0.960
Epoch 28/600
14/14 [==============================] - 0s 2ms/step - loss: 0.1249 - accuracy: 0.957
Epoch 29/600
14/14 [==============================] - 0s 2ms/step - loss: 0.1173 - accuracy: 0.960
```

```
loss = pd.DataFrame(model.history.history)
# We get loss, validation loss, accuracy and validation accuracy
# Let us plot training loss and validation (testing) loss
loss.drop(['accuracy', 'val_accuracy'], axis=1).plot()
# We can see that overfitting has taken place
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffa9a66f780>
```



Early Stopping to Prevent Overfitting

Article on Eary Stopping. Visit [here.]()

```
from tensorflow.keras.callbacks import EarlyStopping
```

```
early_stop = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=25)
# The early stopping callback will try to minimize validation loss
# It will continue monitoring for 25 epochs after val_loss starts increasing
```

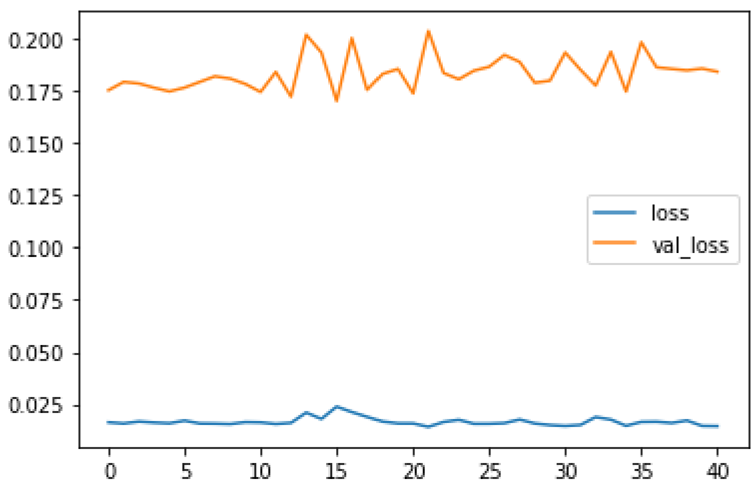
```
model.fit(
  x=X_train,
  y=y_train,
  epochs=600,
  validation_data=(X_test, y_test),
  verbose=1,
  callbacks=[early_stop]
)
# callbacks should be an array
```

```
Epoch 1/600
14/14 [==============================] - 0s 4ms/step - loss: 0.0163 - accuracy: 0.993
Epoch 2/600
14/14 [==============================] - 0s 2ms/step - loss: 0.0158 - accuracy: 0.993
Epoch 3/600
14/14 [==============================] - 0s 2ms/step - loss: 0.0167 - accuracy: 0.993
Epoch 4/600
14/14 [==============================] - 0s 2ms/step - loss: 0.0162 - accuracy: 0.993
Epoch 5/600
14/14 [==============================] - 0s 2ms/step - loss: 0.0159 - accuracy: 0.995
Epoch 6/600
14/14 [==============================] - 0s 2ms/step - loss: 0.0171 - accuracy: 0.993
Epoch 7/600
14/14 [==============================] - 0s 2ms/step - loss: 0.0158 - accuracy: 0.993
Epoch 8/600
14/14 [==============================] - 0s 2ms/step - loss: 0.0157 - accuracy: 0.995
Epoch 9/600
14/14 [==============================] - 0s 2ms/step - loss: 0.0155 - accuracy: 0.993
Epoch 10/600
14/14 [==============================] - 0s 3ms/step - loss: 0.0164 - accuracy: 0.995
Epoch 11/600
14/14 [==============================] - 0s 3ms/step - loss: 0.0163 - accuracy: 0.993
Epoch 12/600
14/14 [==============================] - 0s 2ms/step - loss: 0.0155 - accuracy: 0.993
Epoch 13/600
14/14 [==============================] - 0s 2ms/step - loss: 0.0161 - accuracy: 0.995
Epoch 14/600
14/14 [==============================] - 0s 2ms/step - loss: 0.0210 - accuracy: 0.990
Epoch 15/600
14/14 [==============================] - 0s 3ms/step - loss: 0.0180 - accuracy: 0.993
Epoch 16/600
14/14 [==============================] - 0s 2ms/step - loss: 0.0239 - accuracy: 0.993
Epoch 17/600
14/14 [==============================] - 0s 3ms/step - loss: 0.0213 - accuracy: 0.988
Epoch 18/600
14/14 [==============================] - 0s 3ms/step - loss: 0.0190 - accuracy: 0.990
Epoch 19/600
```

```
14/14 [==============================] - 0s 2ms/step - loss: 0.0167 - accuracy: 0.993
Epoch 20/600
14/14 [==============================] - 0s 3ms/step - loss: 0.0158 - accuracy: 0.995
Epoch 21/600
14/14 [==============================] - 0s 3ms/step - loss: 0.0158 - accuracy: 0.993
Epoch 22/600
14/14 [==============================] - 0s 3ms/step - loss: 0.0142 - accuracy: 0.997
Epoch 23/600
14/14 [==============================] - 0s 3ms/step - loss: 0.0165 - accuracy: 0.990
Epoch 24/600
14/14 [==============================] - 0s 3ms/step - loss: 0.0175 - accuracy: 0.995
Epoch 25/600
14/14 [==============================] - 0s 2ms/step - loss: 0.0156 - accuracy: 0.995
Epoch 26/600
14/14 [==============================] - 0s 3ms/step - loss: 0.0157 - accuracy: 0.993
Epoch 27/600
14/14 [==============================] - 0s 2ms/step - loss: 0.0159 - accuracy: 0.993
Epoch 28/600
14/14 [==============================] - 0s 2ms/step - loss: 0.0177 - accuracy: 0.993
Epoch 29/600
14/14 [==============================] - 0s 2ms/step - loss: 0.0158 - accuracy: 0.997
```

```
loss = pd.DataFrame(model.history.history)
loss.drop(['accuracy', 'val_accuracy'], axis=1).plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffa9a081eb8>
```



Dropout to Reduce Overfitting

Article about Dropout in tensorflow. Visit [here.]()

```
model = Sequential()

model.add(Dense(units=30,activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(units=15,activation='relu'))
```

```
model.add(Dropout(0.5))

model.add(Dense(units=1,activation='sigmoid'))
model.compile(loss='binary_crossentropy', metrics=['Accuracy'], optimizer='adam')
```

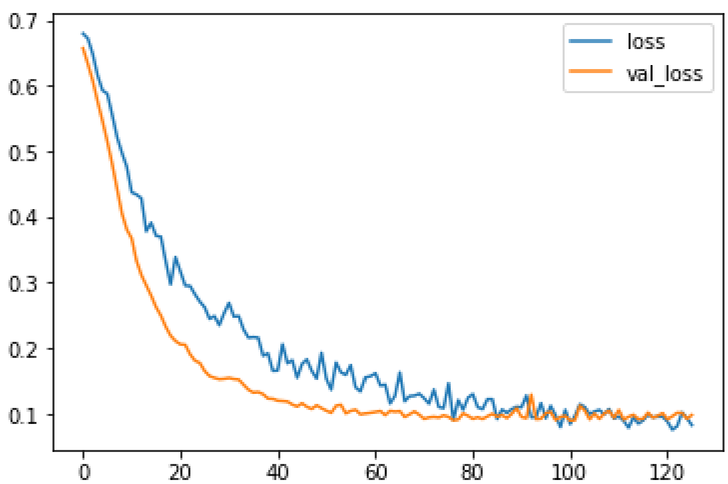
```
model.fit(
  x=X_train,
  y=y_train,
  epochs=600,
  validation_data=(X_test, y_test),
  verbose=1,
  callbacks=[early_stop]
)
```

```
Epoch 1/600
14/14 [==============================] - 0s 10ms/step - loss: 0.6796 - accuracy: 0.00
Epoch 2/600
14/14 [==============================] - 0s 2ms/step - loss: 0.6720 - accuracy: 0.000
Epoch 3/600
14/14 [==============================] - 0s 3ms/step - loss: 0.6483 - accuracy: 0.000
Epoch 4/600
14/14 [==============================] - 0s 2ms/step - loss: 0.6158 - accuracy: 0.000
Epoch 5/600
14/14 [==============================] - 0s 2ms/step - loss: 0.5929 - accuracy: 0.000
Epoch 6/600
14/14 [==============================] - 0s 2ms/step - loss: 0.5876 - accuracy: 0.000
Epoch 7/600
14/14 [==============================] - 0s 3ms/step - loss: 0.5555 - accuracy: 0.000
Epoch 8/600
14/14 [==============================] - 0s 2ms/step - loss: 0.5216 - accuracy: 0.000
Epoch 9/600
14/14 [==============================] - 0s 2ms/step - loss: 0.4975 - accuracy: 0.000
Epoch 10/600
14/14 [==============================] - 0s 2ms/step - loss: 0.4761 - accuracy: 0.000
Epoch 11/600
14/14 [==============================] - 0s 2ms/step - loss: 0.4373 - accuracy: 0.000
Epoch 12/600
14/14 [==============================] - 0s 2ms/step - loss: 0.4338 - accuracy: 0.000
Epoch 13/600
14/14 [==============================] - 0s 2ms/step - loss: 0.4281 - accuracy: 0.000
Epoch 14/600
14/14 [==============================] - 0s 2ms/step - loss: 0.3778 - accuracy: 0.000
Epoch 15/600
14/14 [==============================] - 0s 2ms/step - loss: 0.3908 - accuracy: 0.000
Epoch 16/600
14/14 [==============================] - 0s 3ms/step - loss: 0.3710 - accuracy: 0.000
Epoch 17/600
14/14 [==============================] - 0s 3ms/step - loss: 0.3694 - accuracy: 0.000
Epoch 18/600
14/14 [==============================] - 0s 3ms/step - loss: 0.3327 - accuracy: 0.000
Epoch 19/600
14/14 [==============================] - 0s 3ms/step - loss: 0.2976 - accuracy: 0.000
Epoch 20/600
14/14 [==============================] - 0s 2ms/step - loss: 0.3390 - accuracy: 0.000
Epoch 21/600
```

```
14/14 [==============================] - 0s 2ms/step - loss: 0.3175 - accuracy: 0.00
Epoch 22/600
14/14 [==============================] - 0s 2ms/step - loss: 0.2957 - accuracy: 0.00
Epoch 23/600
14/14 [==============================] - 0s 2ms/step - loss: 0.2951 - accuracy: 0.00
Epoch 24/600
14/14 [==============================] - 0s 2ms/step - loss: 0.2823 - accuracy: 0.00
Epoch 25/600
14/14 [==============================] - 0s 2ms/step - loss: 0.2713 - accuracy: 0.00
Epoch 26/600
14/14 [==============================] - 0s 2ms/step - loss: 0.2625 - accuracy: 0.00
Epoch 27/600
14/14 [==============================] - 0s 2ms/step - loss: 0.2449 - accuracy: 0.00
Epoch 28/600
14/14 [==============================] - 0s 3ms/step - loss: 0.2491 - accuracy: 0.00
Epoch 29/600
14/14 [==============================] - 0s 2ms/step - loss: 0.2352 - accuracy: 0.00
```

```
loss = pd.DataFrame(model.history.history)
loss.drop(['accuracy', 'val_accuracy'], axis=1).plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffa9ad1db00>
```



Classification Report & Confusion Matrix

```
predictions = model.predict(X_test)
predictions[0:10]
```

```
array([[9.9093592e-01],
       [9.9417746e-01],
       [9.9562114e-01],
       [7.5210631e-03],
       [9.9978292e-01],
       [9.9975145e-01],
       [9.9986684e-01],
       [2.3564194e-06],
       [9.9807030e-01],
       [9.9914813e-01]], dtype=float32)
```

```
predictions = model.predict(X_test)[:,0]
predictions[0:10]
```

```
array([9.9093592e-01, 9.9417746e-01, 9.9562114e-01, 7.5210631e-03,
       9.9978292e-01, 9.9975145e-01, 9.9986684e-01, 2.3564194e-06,
       9.9807030e-01, 9.9914813e-01], dtype=float32)
```

```
predictions = np.round(model.predict(X_test)[:,0])
predictions[0:10]
```

```
array([1., 1., 1., 0., 1., 1., 1., 0., 1., 1.], dtype=float32)
```

```
from sklearn.metrics import classification_report,confusion_matrix
```

```
print(confusion_matrix(y_test,predictions))
```

```
[[54  1]
 [ 2 86]]
```

```
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97        55
           1       0.99      0.98      0.98        88

    accuracy                           0.98       143
   macro avg       0.98      0.98      0.98       143
weighted avg       0.98      0.98      0.98       143
```