# Lecture 25 - Our last “model” the DNN

Most of the time when I want to “predict” something from a set of inputs what I want to do is recognize that there is “some” relationship between the set of inputs and the set of outputs.

There are statical models for some of this like multiple linear regression.

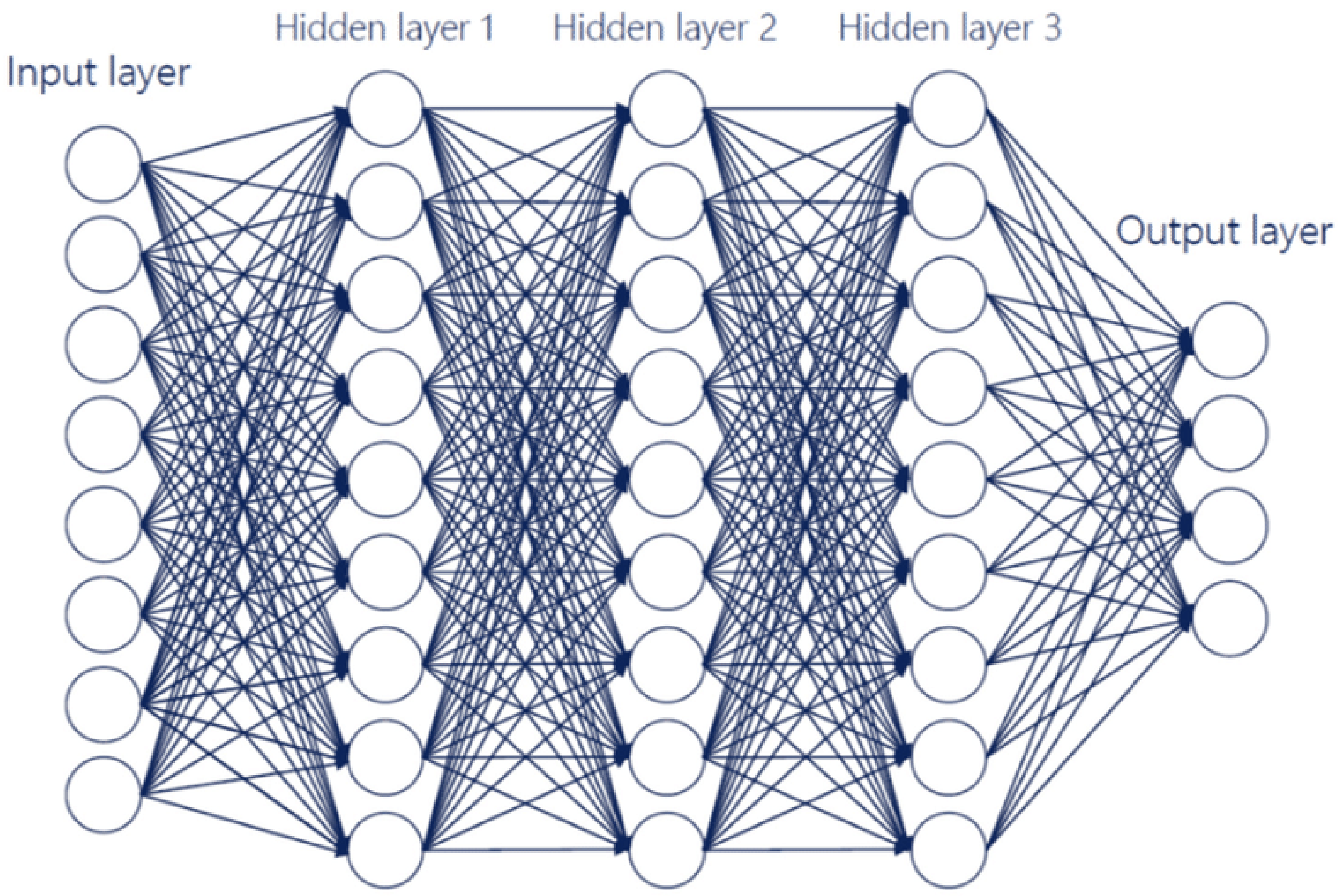
Usually a machine learning model can do what the statistical models can do - only better. It will produce a better “fit” that just pure statistics.

The reason for this is that a machine learning model can be made to recognize “features” that are more complex than just a linear input to output relationship. To do this we need to make the model deeper - we need intermediate layers for the more complex relationships.

The brain uses 6 or 7 layers in the pre-frontal cortex. This gives you a range to work with. Think 3 to 7 layers will do most pattern matching.

Unlike our brain we can do more layers and more training - but - As we do this example I think that you will find that more layers may not be all that useful.

So the model will look something like:



let’s walk through some code for predicting gas millage based on horsepower, weight and other inputs based on this.

```

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

# Make NumPy printouts easier to read.
np.set_printoptions(precision=3, suppress=True)

import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers

print(tf.__version__)

url = 'http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data'
column_names = ['MPG', 'Cylinders', 'Displacement', 'Horsepower', 'Weight',
                'Acceleration', 'Model Year', 'Origin']

raw_dataset = pd.read_csv(url, names=column_names,
                          na_values='?', comment='\t',
                          sep=' ', skipinitialspace=True)

dataset = raw_dataset.copy()
print ( dataset.tail() )

# Get rid of bad data.
print ( dataset.isna().sum() )
dataset = dataset.dropna()

# One hot encode Origin
dataset['Origin'] = dataset['Origin'].map({1: 'USA', 2: 'Europe', 3: 'Japan'})
dataset = pd.get_dummies(dataset, columns=['Origin'], prefix='', prefix_sep='')
print ( dataset.tail() )

# Split into training and testing data sets.
train_dataset = dataset.sample(frac=0.8, random_state=0)
test_dataset = dataset.drop(train_dataset.index)

# Profile the data with some graphs
sns.pairplot(train_dataset[['MPG', 'Cylinders', 'Displacement', 'Weight']], diag_kind='kde')
plt.show()

print ( train_dataset.describe().transpose() )

train_features = train_dataset.copy()
test_features = test_dataset.copy()

train_labels = train_features.pop('MPG')
test_labels = test_features.pop('MPG')

print ( train_dataset.describe().transpose()[['mean', 'std']] )

normalizer = tf.keras.layers.Normalization(axis=-1)
normalizer.adapt(np.array(train_features))

print(normalizer.mean.numpy())

first = np.array(train_features[:1])

with np.printoptions(precision=2, suppress=True):
  print('First example:', first)
  print()
  print('Normalized:', normalizer(first).numpy())
```

```




horsepower = np.array(train_features['Horsepower'])

# Must take data and convert it to values that have a consistent range.
horsepower_normalizer = layers.Normalization(input_shape=[1,], axis=None)
horsepower_normalizer.adapt(horsepower)

# Make horsepower into a sequenctial model input.
horsepower_model = tf.keras.Sequential([
    horsepower_normalizer,
    layers.Dense(units=1)
])

horsepower_model.summary()

print ( horsepower_model.predict(horsepower[:10]) )

horsepower_model.compile(
    optimizer=tf.optimizers.Adam(learning_rate=0.1),
    loss='mean_absolute_error')

history = horsepower_model.fit(
    train_features['Horsepower'],
    train_labels,
    epochs=100,
    verbose=0, # Suppress logging.
    validation_split = 0.2) # Calculate validation results on 20% of the training data.

hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch
print ( hist.tail() )


def plot_loss(history):
    plt.plot(history.history['loss'], label='loss')
    plt.plot(history.history['val_loss'], label='val_loss')
    plt.ylim([0, 10])
    plt.xlabel('Epoch')
    plt.ylabel('Error [MPG]')
    plt.legend()
    plt.grid(True)
    plt.show()

plot_loss(history)

test_results = {}

test_results['horsepower_model'] = horsepower_model.evaluate(
    test_features['Horsepower'],
    test_labels, verbose=0)

x = tf.linspace(0.0, 250, 251)
y = horsepower_model.predict(x)

def plot_horsepower(x, y):
    plt.scatter(train_features['Horsepower'], train_labels, label='Data')
    plt.plot(x, y, color='k', label='Predictions')
    plt.xlabel('Horsepower')
    plt.ylabel('MPG')
    plt.legend()
    plt.show()

```

```
plot_horsepower(x, y)


# --

# The model with multiple layers - try different numbers of layers to see what
# effect this has on the results.
def build_and_compile_model(norm):
  model = keras.Sequential([
      norm,
      layers.Dense(64, activation='relu'),
      layers.Dense(64, activation='relu'),
      layers.Dense(64, activation='relu'),
      layers.Dense(64, activation='relu'),
      layers.Dense(1)
  ])

  model.compile(loss='mean_absolute_error',
                optimizer=tf.keras.optimizers.Adam(0.001))
  return model

dnn_horsepower_model = build_and_compile_model(horsepower_normalizer)

# Print out a text stummary of the behavior of the model.
dnn_horsepower_model.summary()

# Train to this model.
history = dnn_horsepower_model.fit(
    train_features['Horsepower'],
    train_labels,
    validation_split=0.2,
    verbose=0, epochs=100)

# Plot loss values to see how long it took to train the model.
plot_loss(history)

# Use the model to do preditions.
x = tf.linspace(0.0, 250, 251)
y = dnn_horsepower_model.predict(x)

# Plot results
plot_horsepower(x, y)

# Evaluate results of model over the 20% of data that we sved for the test.
test_results['dnn_horsepower_model'] = dnn_horsepower_model.evaluate(
    test_features['Horsepower'], test_labels,
    verbose=0)

dnn_model = build_and_compile_model(normalizer)
dnn_model.summary()

history = dnn_model.fit(
    train_features,
    train_labels,
    validation_split=0.2,
    verbose=0, epochs=100)

# How did our model turn out.
plot_loss(history)


# Save model for later.
test_results['dnn_model'] = dnn_model.evaluate(test_features, test_labels, verbose=0)

pd.DataFrame(test_results, index=['Mean absolute error [MPG]']).T

```

```
test_predictions = dnn_model.predict(test_features).flatten()

a = plt.axes(aspect='equal')
plt.scatter(test_labels, test_predictions)
plt.xlabel('True Values [MPG]')
plt.ylabel('Predictions [MPG]')
lims = [0, 50]
plt.xlim(lims)
plt.ylim(lims)
_ = plt.plot(lims, lims)
plt.show()

error = test_predictions - test_labels
plt.hist(error, bins=25)
plt.xlabel('Prediction Error [MPG]')
_ = plt.ylabel('Count')
plt.show()

dnn_model.save('dnn_model')

reloaded = tf.keras.models.load_model('dnn_model')

test_results['reloaded'] = reloaded.evaluate(
    test_features, test_labels, verbose=0)

pd.DataFrame(test_results, index=['Mean absolute error [MPG]']).T
```