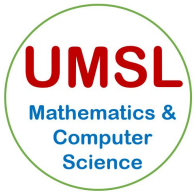


The Basics of Neural Networks

http://jalammar.github.io/visual-interactive-guide-basics-neural-networks/

https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/

https://github.com/badriadhikari/dltutorial/blob/master/Pima-Indians-Diabetes.ipynb

The Problem

Let's start with a simple example. Say you're helping a friend who wants to buy a house. She was quoted $400,000 for a 2000 sq ft house (185 meters). Is this a good price or not?

It's not easy to tell without a frame of reference. So you ask your friends who have bought houses in that same neighborhoods, and you end up with three data points:

Area (sq ft) (x)	Price (y)
2,104	399,900
1,600	329,900
2,400	369,000

What is a good price?

We Have a Model

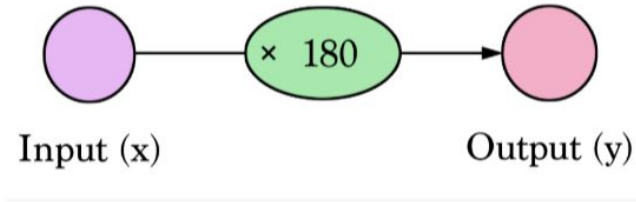
Let's start with a simple example. Say you're helping a friend who wants to buy a house. She was quoted $400,000 for a 2000 sq ft house (185 meters). Is this a good price or not?

It's not easy to tell without a frame of reference. So you ask your friends who have bought houses in that same neighborhoods, and you end up with three data points:

Area (sq ft) (x)	Price (y)
2,104	399,900
1,600	329,900
2,400	369,000

Personally, my first instinct would be to get the average price per sq ft. That comes to $180 per sq ft.

Welcome to your first neural network! Now it's not quite at Siri level yet, but now you know the fundamental building block. And it looks like this:



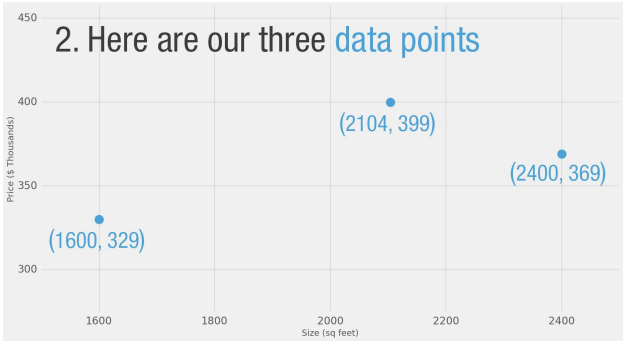
Diagrams like this show you the structure of the network and how it calculates a prediction. The calculation starts from the input node at the left. The input value flows to the right. It gets multiplied by the weight and the result becomes our output.

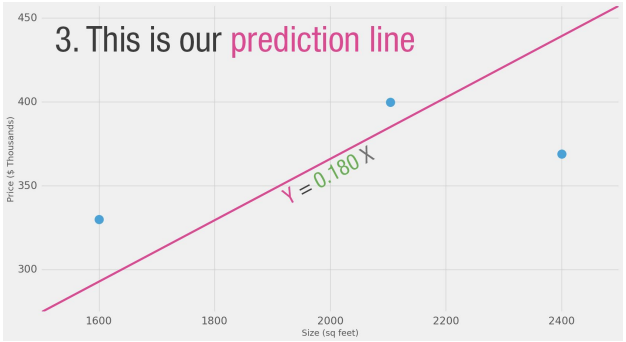
Multiplying 2,000 sq ft by 180 gives us $360,000. That's all there is to it at this level. Calculating the prediction is simple multiplication. But before that, we needed to think about the weight we'll be multiplying by. Here we started with an average, later we'll look at better algorithms that can scale as we get more inputs and more complicated models. Finding the weight is our "training" stage. So whenever you hear of someone "training" a neural network, it just means finding the weights we use to calculate the prediction.

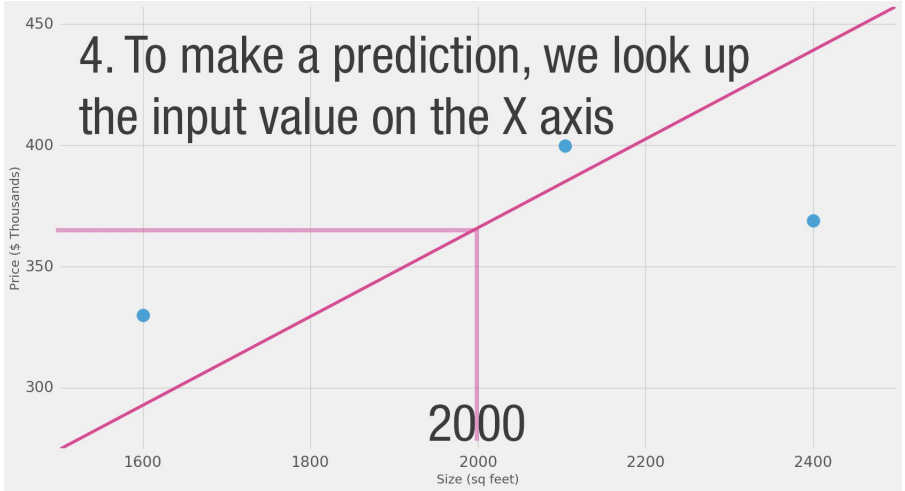
$$y = W x$$

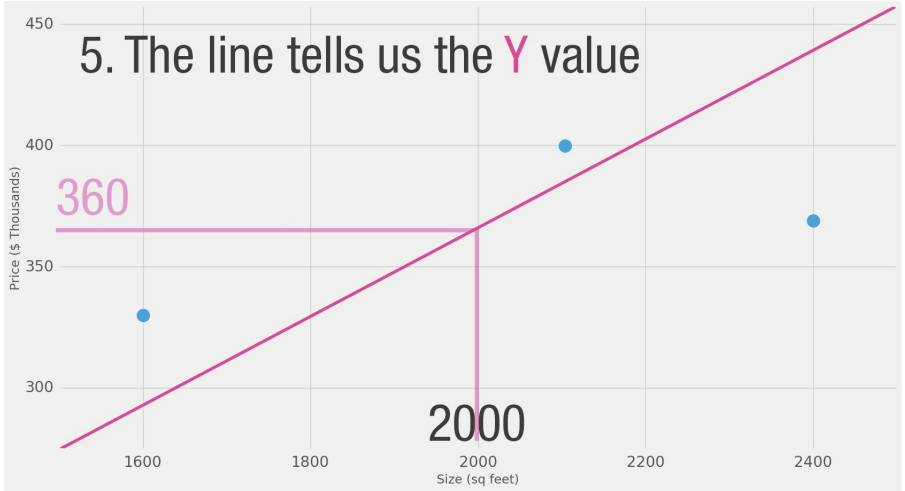
This is a form of prediction. This is a simple predictive model that takes an input, does a calculation, and gives an output (since the output can be of continuous values, the technical name for what we have would be a "regression model")

How to Make a Prediction Using a Model?

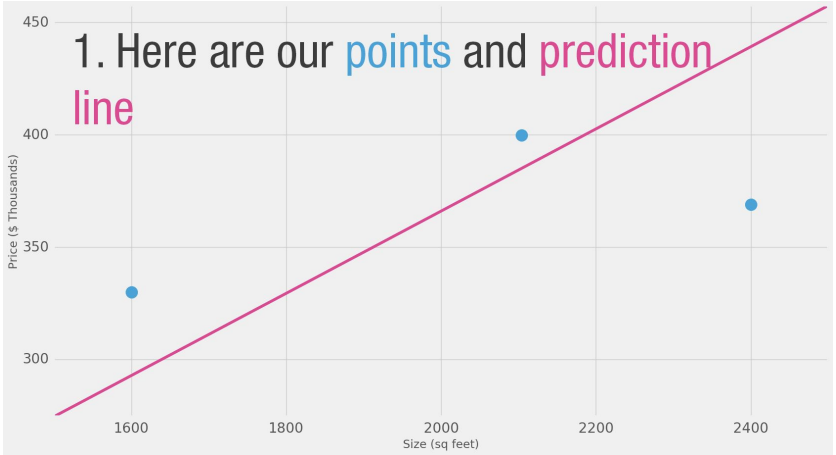


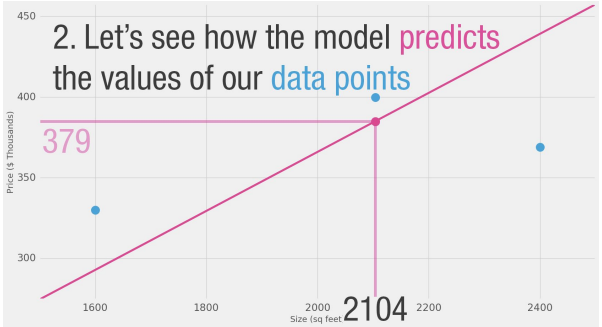


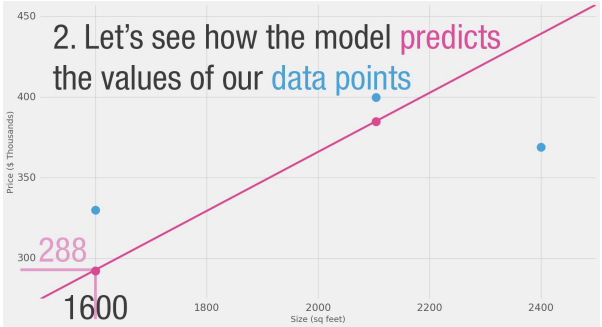


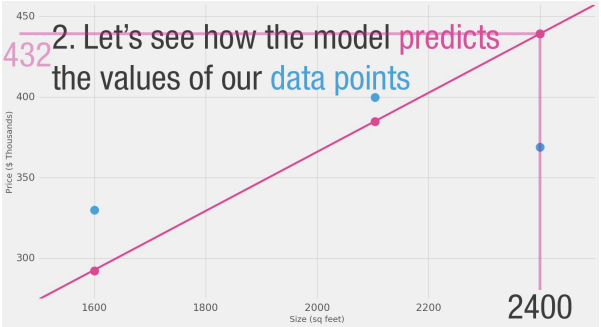


How to Minimize the Error on Training Dataset?

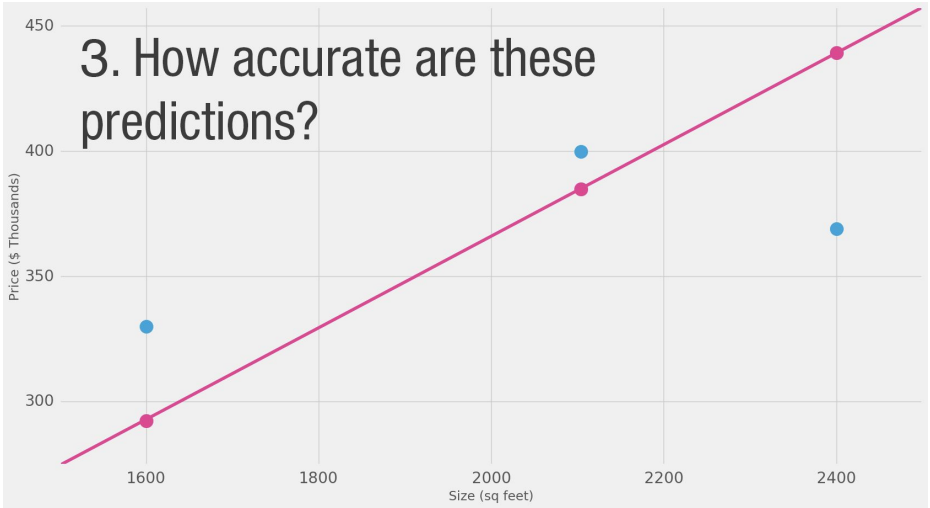


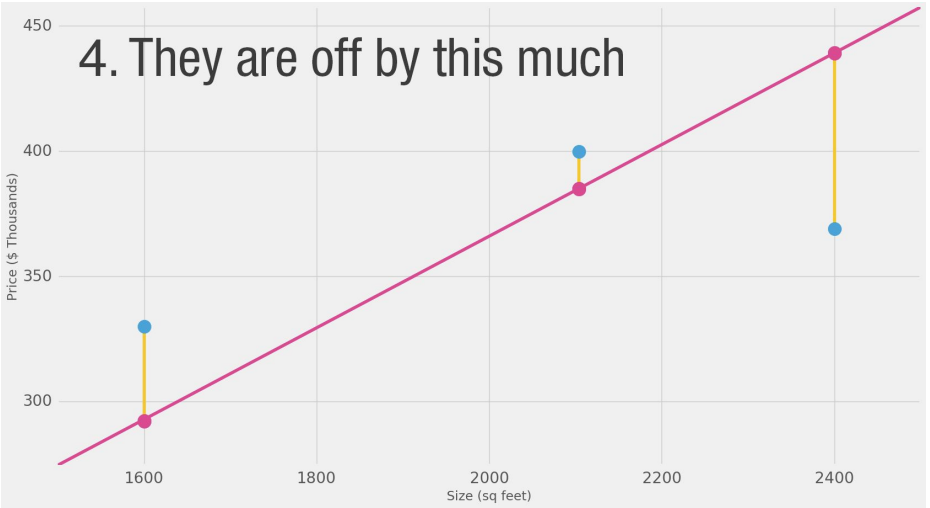




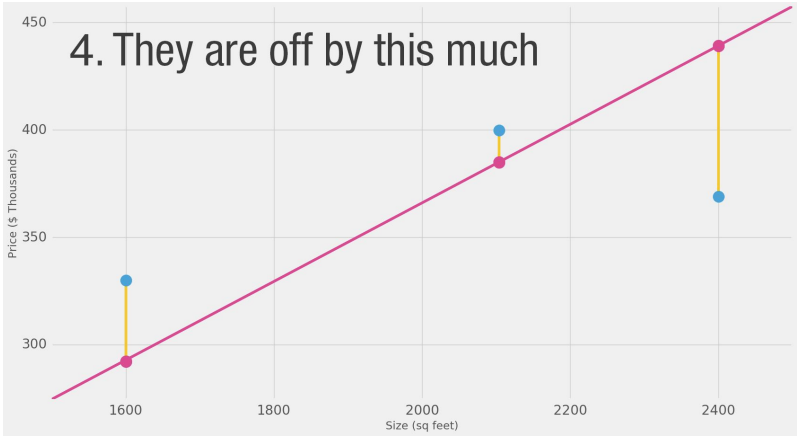


How to Minimize the Error on Training Dataset?





Calculating Mean-Squared Error on Training Dataset

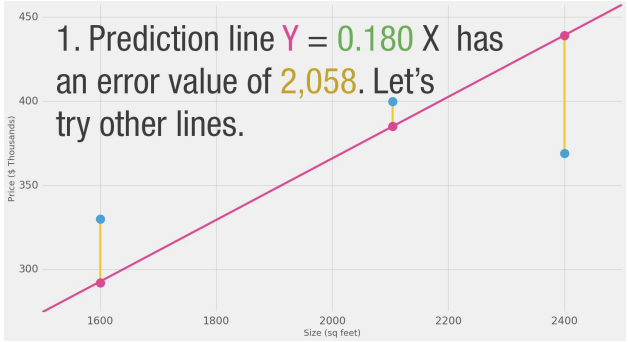


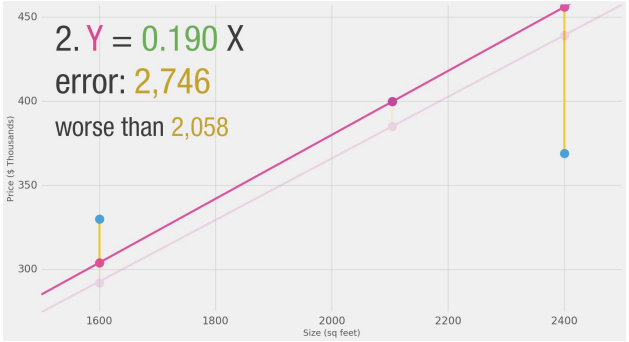
Area (x)	Price ($1000) (y_)	Prediction (y)	y_-y	(y_-y)²
2,104	399.9	379	21	449
1,600	329.9	288	42	1756
2,400	369	432	-63	3969
			Average:	2,058

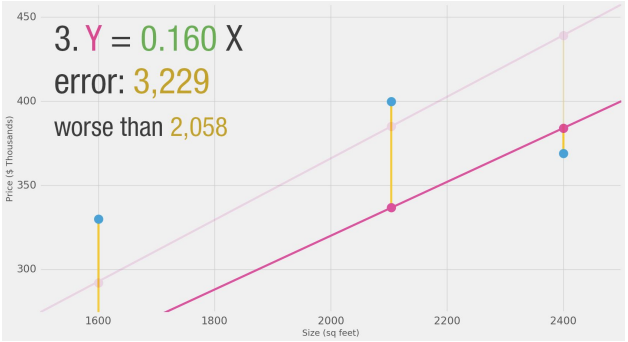
Here we can see the actual price value, the predicted price value, and the difference between them. Then we'll need to average these differences so we have a number that tells us how much error there is in this prediction model. The problem is, the 3rd row has -63 as its value. We have to deal with this negative value if we want to use the difference between the prediction and price as our error measuring stick. That's one reason why we introduce an additional column that shows the error squared, thus getting rid of the negative value.

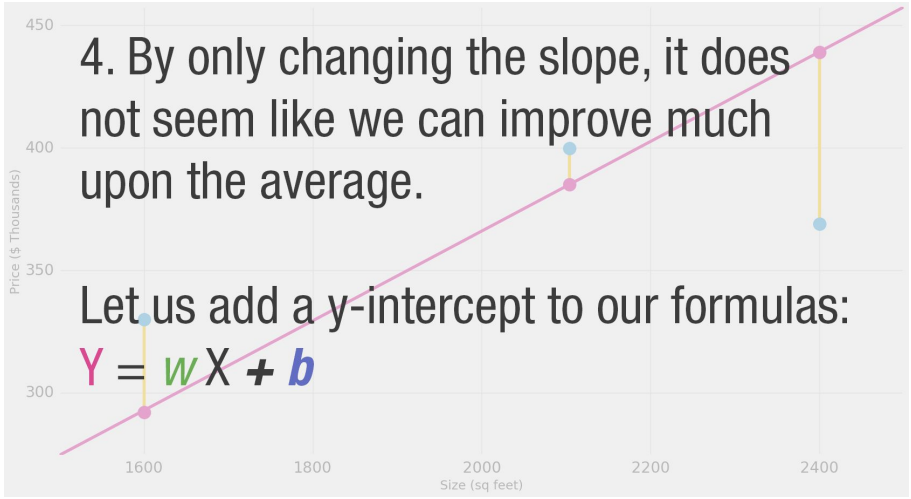
This is now our definition of doing better – a better model is one that has less error. Error is measured as the average of the errors for each point in our data set. For each point, the error is measured by the difference between the actual value and the predicted value, raised to the power of 2. This is called Mean Square Error. Using it as a guide to train our model makes it our **loss function** (also, **cost function**).

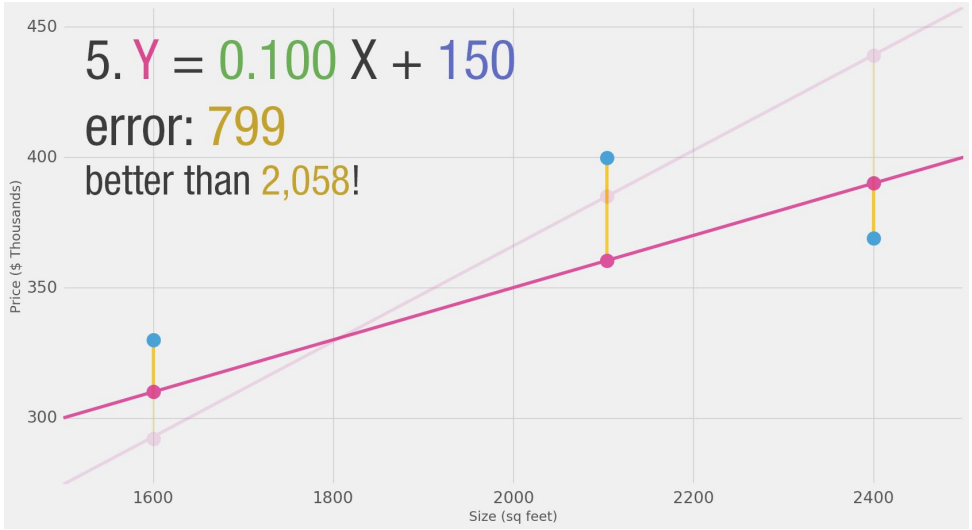
Tweaking the Current Model to Obtain Better Models



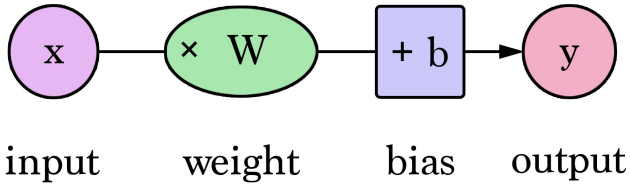








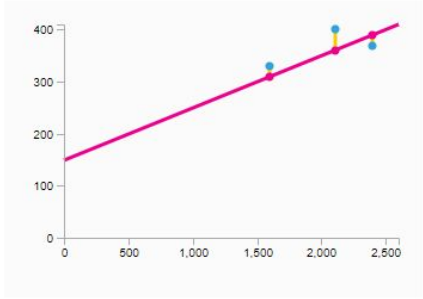
Updated Model



$$y = W x + b$$

$$y = 0.100 x + 150$$

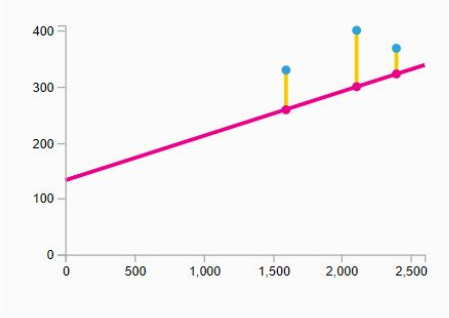
Train Your Dragon

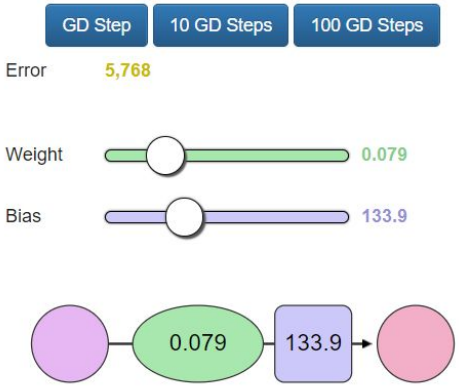


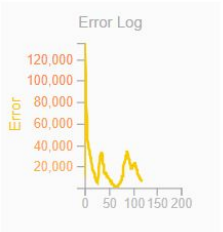
http://jalammar.github.io/visual-interactive-guide-basics-neural-networks/

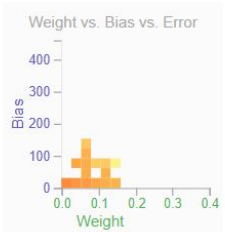
Automation

Congratulations on manually training your first neural network! Let's look at how to automate this training process. Below is another example with an additional autopilot-like functionality. These are the GD Step buttons. They use an algorithm called "Gradient Descent" to try to step towards the correct weight and bias values that minimize the loss function.







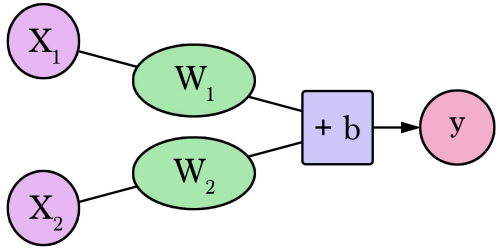


The two new graphs are to help you track the error values as you fiddle with the parameters (weight and bias) of the model. It's important to keep track of the error as the training process is all about reducing this error as much possible.

How does gradient descent know where its next step should be? Calculus. You see, knowing the function we're minimizing (our loss function, the average of $(y\_ - y)^2$ for all our data points), and knowing the current inputs into it (the current weight and bias), the derivatives of the loss function tell us which direction to nudge W and b in order to minimize the error.

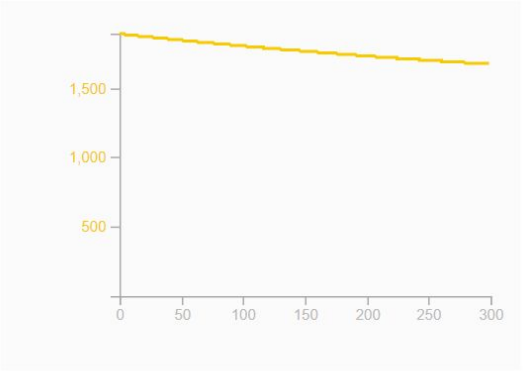
Two Variable (FEATURE) Neural Network

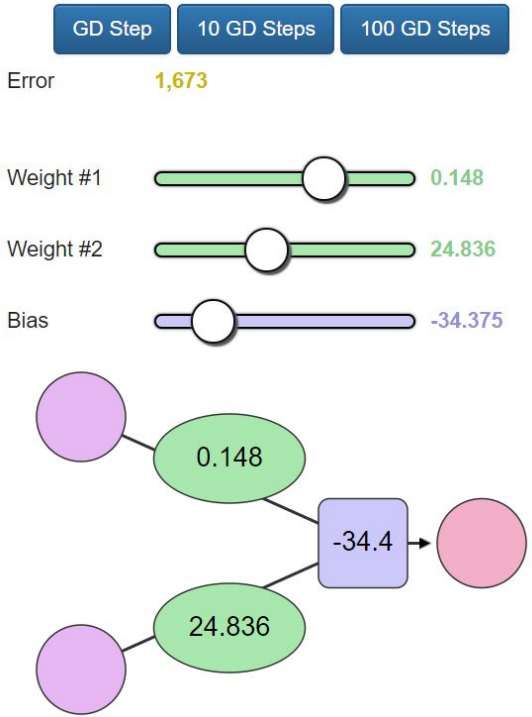
Area (sq ft) (x1)	Bathrooms (x2)	Price (y)
2,104	3	399,900
1,600	3	329,900
2,400	3	369,000
1,416	2	232,000
3,000	4	539,900
1,985	4	299,900
1,534	3	314,900
1,427	3	198,999
1,380	3	212,000
1,494	3	242,500



$$y = W_1 x_1 + W_2 x_2 + b$$

Gradient Descent Again!





http://jalammar.github.io/visual-interactive-guide-basics-neural-networks/

Implementation

Dataset - Pima Indians onset of diabetes dataset. This is a standard machine learning dataset from the UCI Machine Learning repository. It describes patient medical record data for Pima Indians and whether they had an onset of diabetes within five years.

```
# Create your first MLP in Keras
from keras.models import Sequential
from keras.layers import Dense
import numpy
# fix random seed for reproducibility
numpy.random.seed(7)
# load pima indians dataset
dataset = numpy.loadtxt("pima-indians-diabetes.csv", delimiter=",")
# split into input (X) and output (Y) variables
X = dataset[:,0:8]
Y = dataset[:,8]
# create model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Fit the model
model.fit(X, Y, epochs=150, batch_size=10)
# evaluate the model
scores = model.evaluate(X, Y)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
6,148,72,35,0,33.6,0.627,50,1
1,85,66,29,0,26.6,0.351,31,0
8,183,64,0,0,23.3,0.672,32,1
1,89,66,23,94,28.1,0.167,21,0
0,137,40,35,168,43.1,2.288,33,1
5,116,74,0,0,25.6,0.201,30,0
3,78,50,32,88,31.0,0.248,26,1
10,115,0,0,0,35.3,0.134,29,0
2,197,70,45,543,30.5,0.158,53,1
8,125,96,0,0,0.0,0.232,54,1
4,110,92,0,0,37.6,0.191,30,0
...
...
```

1. Number of times pregnant
2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test
3. Diastolic blood pressure (mm Hg)
4. Triceps skin fold thickness (mm)
5. 2-Hour serum insulin (mu U/ml)
6. Body mass index (weight in kg/(height in m)^2)
7. Diabetes pedigree function
8. Age (years)
9. Class variable (0 or 1)

https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/

Splitting Data Into Training Set and Validation Set

```
# load pima indians dataset
dataset = numpy.loadtxt("https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv", delimiter=",")
# split into input (X) and output (Y) variables
X = dataset[:700,0:8]
Y = dataset[:700,8]
XTEST = dataset[700:,0:8]
YTEST = dataset[700:,8]
```

```
# Fit the model
history = model.fit(X, Y, epochs=100, batch_size=5, validation_data = (XTEST, YTEST), verbose=2)
```