# Week 2 – Stanford Machine Learning

## Multivariate Regression via Gradient Descent

Let us create some random data, say housing size, number of bedrooms, and age of the house as a feature matrix, and price levels on y. Although we deal with a multi-dimensional case, traditionally we consider the training features as a part of **x** and the training labels as a part of **y**.

In[669]:=
```
trainX = {{100, 320, 213, 512, 58, 84, 113},
   {1, 4, 2, 4, 0, 1, 1}, {7, 4, 12, 21, 9, 3, 31}};
trainY = {140 000, 400 000, 241 000, 489 000, 78 000, 123 000, 139 000};
grid = Grid[MapThread[Prepend, {Prepend[Transpose@Append[trainX, trainY],
     {"Size (feet^2)", "Bedrooms", "Age (years)", "Price ($)"}],
    Join[{"Example"}, Range[1, 7]]}], Dividers → Center]
```

Out[671]=

| Example | Size $(\text{feet}^2)$ | Bedrooms | Age (years) | Price ($) |
|---|---|---|---|---|
| 1 | 100 | 1 | 7 | 140 000 |
| 2 | 320 | 4 | 4 | 400 000 |
| 3 | 213 | 2 | 12 | 241 000 |
| 4 | 512 | 4 | 21 | 489 000 |
| 5 | 58 | 0 | 9 | 78 000 |
| 6 | 84 | 1 | 3 | 123 000 |
| 7 | 113 | 1 | 31 | 139 000 |

In order to make this data work better with our gradient descent algorithm, we scale all the features through mean normalisation and by dividing by the standard deviation.

In[672]:=
```
trainXScaled = (trainX - Mean[Transpose@trainX]) / StandardDeviation[Transpose@trainX];
```

We could also have used the range of the list as below, but the standard deviation works just as well.

In[673]:=
```
trainXScaledWithRange = (trainX - Mean[Transpose@trainX]) / (Max[#] - Min[#] & /@ trainX);
```

Now, we define a our hypothesis function which is the dot product between any two n-dimensional vectors

In[674]:=
```
Hyp[v_, x_] := Dot[v, x]
```

Then, before defining the cost function, we modify the training data a final time to include the constant in our multivariate model, usually $x_0$. That is, we prepend a column of ones.

In[675]:=
```
trainXFinal = Join[{Table[1, {Length[trainY]}]}, trainXScaled];
```

The cost function is defined below as follows.

In[676]:=
```
Cost[v_] :=  1 / (2 Length[trainXFinal])
    Total[Apply[Hyp, {v, #1} & /@Transpose@trainXFinal, {1}]^2 - trainY];
```

We then create the step function and assign the learning rate.

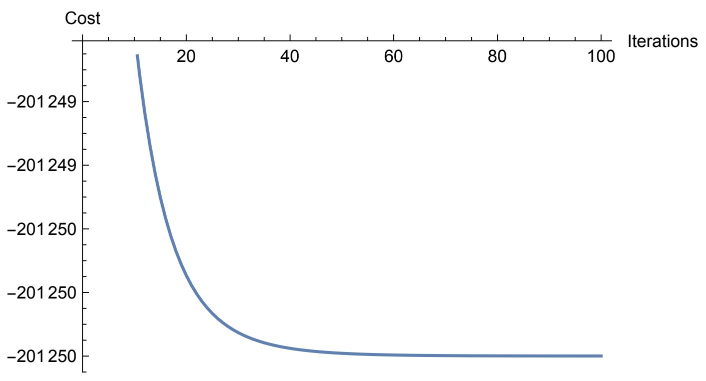
In[681]:=
```
α = 0.03;
Step[v_] :=
  v - α Grad[Cost[{w, x, y, z}], {w, x, y, z}] /.
   {w → v[[1]], x → v[[2]], y → v[[3]], z → v[[4]]}
```

To check that the step function works, we can plot the step function as a function of the iteration number to picture the learning rate. We do so by creating the learning function

In[679]:=
```
Learning[x_] := Cost@Nest[Step, {1, 1, 1, 1}, x];
```

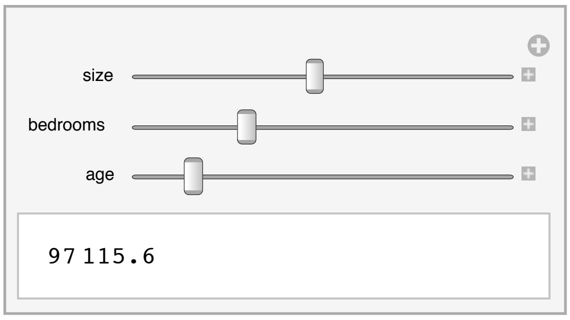
As seen below, we learn, and so our gradient descent algorithm for multiple variables works.

In[683]:=
```
learningRate = ListLinePlot[#, AxesLabel → {"Iterations", "Cost"}] &@
  Transpose@{Range[0, 100], Learning /@ Range[0, 100]}
```

Out[683]=


Thus, we can predict the new housing prices as follows

In[684]:=
```
optimalParameters = Nest[Step, {1, 1, 1, 1}, 10 000]
     Join[{1}, StandardDeviation[Transpose@trainX]] +
    Join[{1}, Mean[Transpose@trainX]];
Model = Manipulate[Hyp[{1, size, bedrooms, age}, optimalParameters],
  {size, 10, 1000}, {bedrooms, 0, 10}, {age, 0, 70}]
```

Out[685]=


```
97 115.6
```

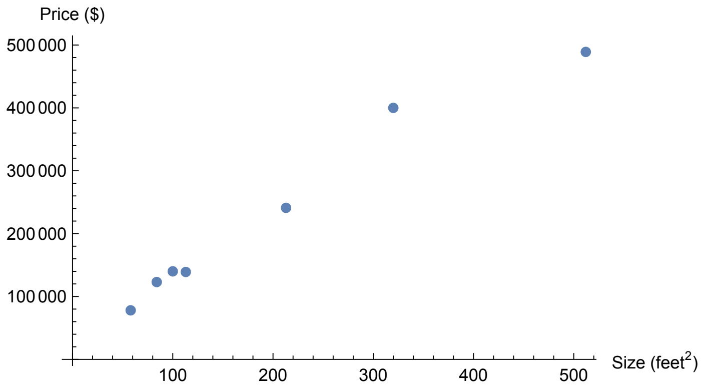
As seen, our model computes the predicted housing price in a very poor manner compared to the sample cases. The reason for this is two-fold; firstly, our data set is not large enough for the model to correctly capture the fact that age is a negative attribute. Moreover, we did a linear fit, which probably wasn't the best possible model.

## Polynomial Regression

Using the dataset from week1,we will try and fit a polynomial model instead. First, we insert and plot the data.

In[686]:=
```
trainX = {100, 320, 213, 512, 58, 84, 113};
trainY = {140 000, 400 000, 241 000, 489 000, 78 000, 123 000, 139 000};
reglp = ListPlot[Transpose@{trainX, trainY},
   AxesLabel → {"Size (feet^2)", "Price ($)"}];
Show[
 reglp]
```

Out[689]=



We then define a hypothesis and a cost function. Note that, this time, we add an extra parameter for the squared term.

In[690]:=
```
Hyp[v_, x_] := v[[1]] + v[[2]] x + v[[3]] x^2;
Cost[v1_, v2_, v3_] := 1/(2 Length[trainX])
   Total[(Hyp[{v1, v2, v3}, #1] - #2)^2 & @@@ Transpose@{trainX, trainY}];
```

We create the step function; again note the very low learning rates, which could have been avoided by normalising the data as above.

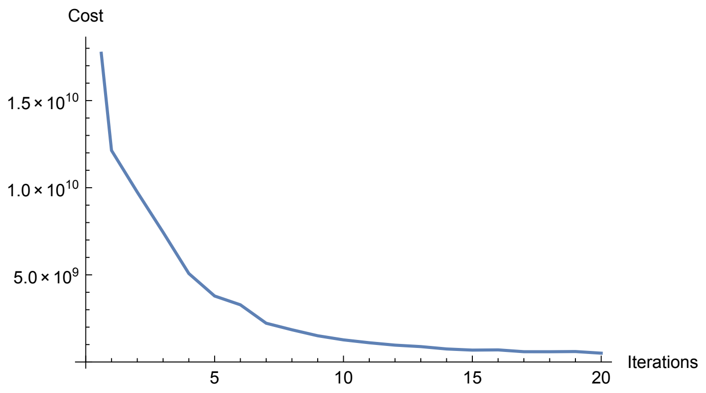
In[692]:=
```
αv1 = 0.1;
αv2  = 0.00001;
αv3 = 0.0000000001;
Step[v1_, v2_, v3_] :=
  {v1 - αv1 D[Cost[x, y, z], x] /. {x → v1, y → v2, z → v3},
   v2 - αv2 D[Cost[x, y, z], y] /. {x → v1, y → v2, z → v3},
   v3 - αv3 D[Cost[x, y, z], z] /. {x → v1, y → v2, z → v3}};
```

We check that it works via the learning rate plot

In[696]:=

```
Learning[x_] :=
  Cost @@ Nest[Apply[Step, #] &, {RandomReal[], RandomReal[], RandomReal[]}, x];
learningRate = ListLinePlot[#, AxesLabel → {"Iterations", "Cost"}] &@
  Transpose@{Range[0, 20], Learning /@ Range[0, 20]}
```

Out[697]=

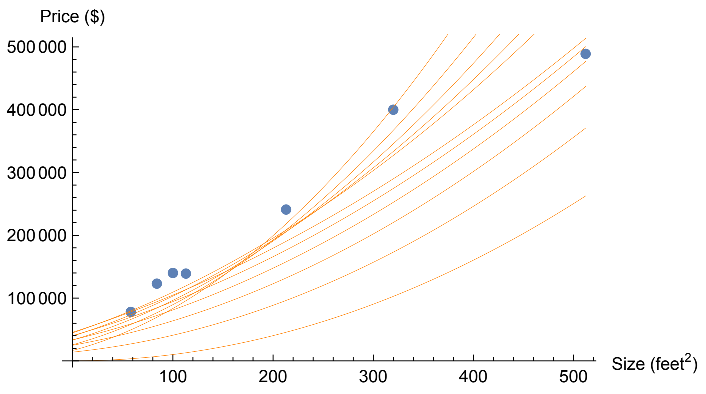


And then step through the algorithm ten times, to plot all intermediate steps.

In[698]:=

```
gd = NestList[Apply[Step, #] &, {1, 1, 1}, 10];
gdpp = Show[
   Plot[Hyp[#1, x], {x, 0, Max[trainX]}, PlotStyle → {Orange, Thin}] & /@ gd];
Show[
 gdlp,
 gdpp]
```

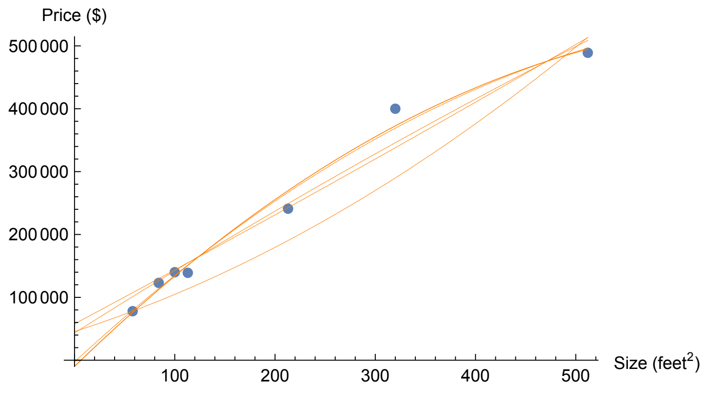
Out[700]=



As seen, we do a very poorly fit over the first 10 iterations. However, if we plot the parameters after higher iterations as seen below, then the fit becomes better.

In[701]:=
```
gdit =
  Nest[Apply[Step, #] &, {1, 1, 1}, #] & /@ {10, 50, 100, 500, 1000, 5000, 10 000};
gdpp = Show[Plot[Hyp[#1, x], {x, 0, Max[trainX]}, PlotStyle → {Orange, Thin}] & /@
    gdit];
Show[
 gdlp,
 gdpp]
```

Out[703]=



## Normal Equation

As seen, our gradient descent algorithm took long time in order for us to actually derive at a useful answer. We could have optimised it by scaling the features and readjusting the weights. However, another method to get the precise parameters for the lowest cost, is the normal equation.

We reuse the same data from the beginning.

In[704]:=
```
trainX = {Table[1, {7}], {100, 319, 215, 511, 57, 83, 123},
   {1, 4, 2, 4, 0, 1, 1}, {7, 4, 12, 23, 7, 3, 33}};
trainY = {140 000, 400 000, 241 000, 489 000, 78 000, 123 000, 139 000};
grid = Grid[MapThread[Prepend, {Prepend[Transpose@Append[trainX, trainY],
      {"x0", "Size (feet2)", "Bedrooms", "Age (years)", "Price ($)"}],
    Join[{"Example"}, Range[1, 7]]}], Dividers → Center]
```

Out[706]=

| Example | $x_0$ | Size (feet$^2$) | Bedrooms | Age (years) | Price ($) |
|---|---|---|---|---|---|
| 1 | 1 | 100 | 1 | 7 | 140 000 |
| 2 | 1 | 319 | 4 | 4 | 400 000 |
| 3 | 1 | 215 | 2 | 12 | 241 000 |
| 4 | 1 | 511 | 4 | 23 | 489 000 |
| 5 | 1 | 57 | 0 | 7 | 78 000 |
| 6 | 1 | 83 | 1 | 3 | 123 000 |
| 7 | 1 | 123 | 1 | 33 | 139 000 |

We then compute the precise values through the normal equation, assuming that the matrix is singular. Note that we apply the transposes differently, due to the way mathematica stores matrices.

In[707]:=
```
theta = Inverse[trainX.Transpose[trainX]].trainX.trainY // N
```

Out[707]=
```
{43 178.9, 545.952, 44 973.9, -512.509}
```

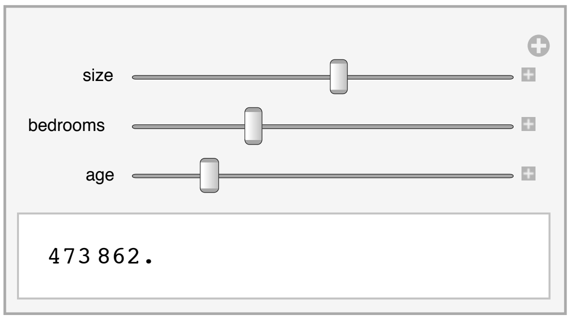
Using these parameters in our hypothesis function

In[708]:=
```
Hyp[v_, x_] := Dot[v, x]
```

we get  much better housing price predictions as seen below. Note that age counts negatively, while bedrooms and size count positively towards the price. This also suggest that we might have done something wrong with the gradient descent algorithm.

In[709]:=
```
Model = Manipulate[Hyp[{1, size, bedrooms, age}, {43178.87263686469`,
     545.9516325435968`, 44973.94772235059`, -512.5092974156712`}],
   {size, 10, 1000}, {bedrooms, 0, 10, 1}, {age, 0, 70, 1}]
```

Out[709]=



```
473 862.
```