Linear Regression

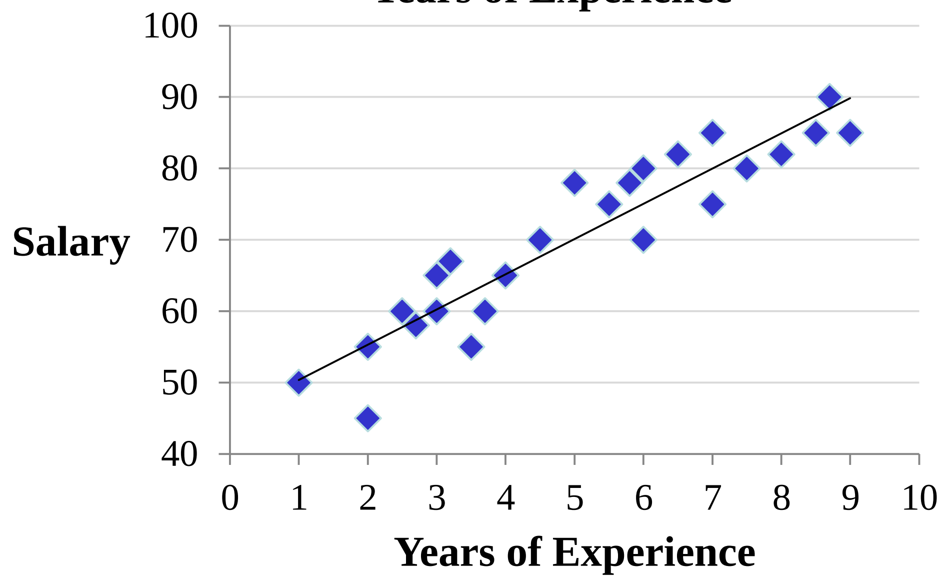
Another kind of prediction

YearsOfExperience (x_1)	**Salary** (y)
1	50
2	55
2	45
2.5	60
2.7	58
…	…

- Suppose we'd like to predict salary based on number of years of experience.

- Different prediction problem because class is a continuous attribute.
  - Called **Regression**

- Linear Regression: Build a prediction **line**.



Linear regression with one variable

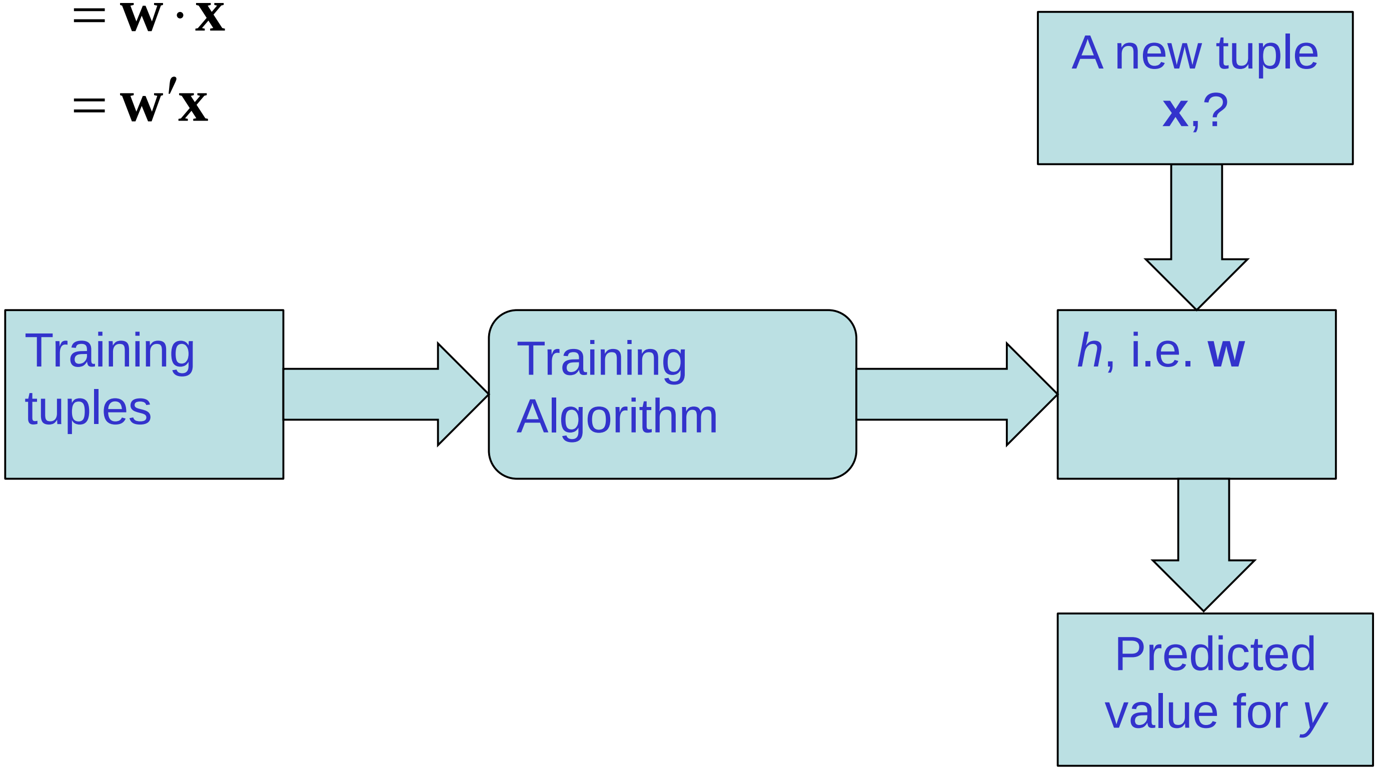
$$w' = w^T$$

$$h(\mathbf{w},\mathbf{x}) = w_0 + w_1 x_1$$

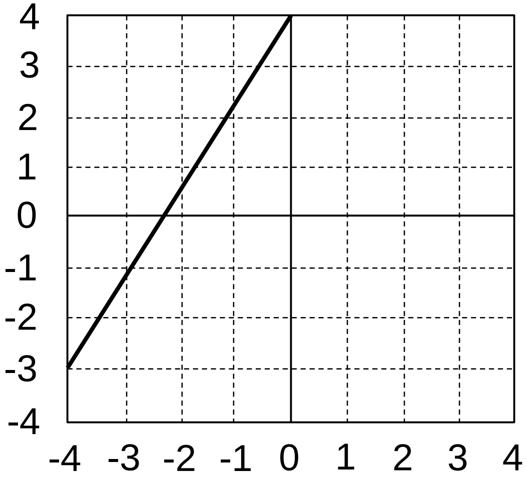
$$= w_0 x_0 + w_1 x_1 \qquad x_0 = 1$$

$$= \mathbf{w} \cdot \mathbf{x}$$

$$= \mathbf{w}'\mathbf{x}$$



Example of line



$$w_0 + w_1 x_1 = y$$

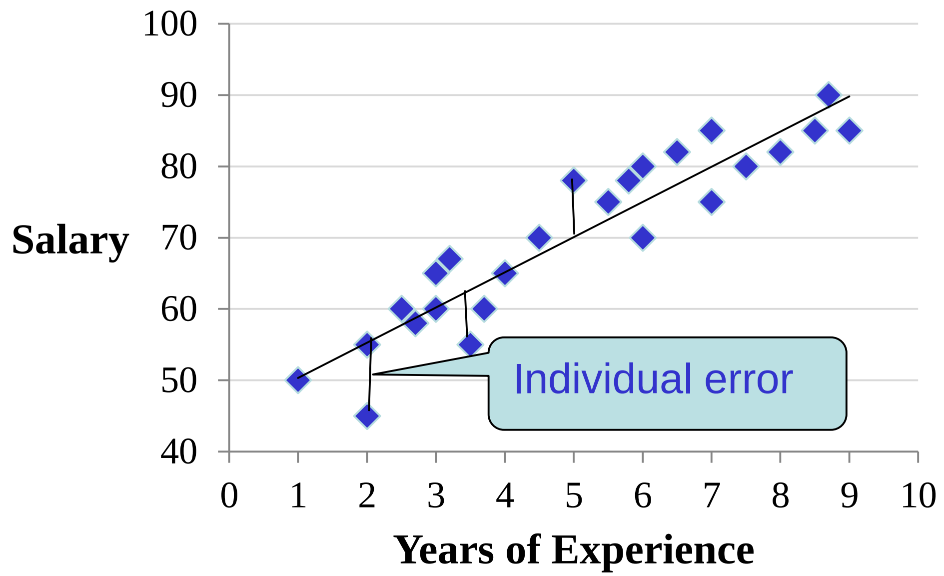
$$w_0 + w_1(-4) = -3$$

$$w_0 + w_1(0) = 4$$

$$w_0 = 4$$

$$w_1 = 7/4$$

Cost/Error/Penalty Function



- Goal: find a line (hypothesis) $h(\mathbf{w},\mathbf{x})$ that for the training tuples gives numbers close to their y's.

n is the number of training tuples

$$E(\mathbf{w}) = \frac{1}{2n}\sum_{k=1}^{n}(y^k - \mathbf{w}'\mathbf{x}^k)^2$$

We want to minimize *E* over possible **w**'s.
x^k are fixed, they are the training tuples.

Average squared error.
½ in the front is just to make the math easier.

Recap

Hypothesis form:

$$h(\mathbf{w}, \mathbf{x}) = \mathbf{w} \cdot \mathbf{x} = w_0 x_0 + w_1 x_1$$

Weights to learn:

$$w_0, w_1$$

Error function:

$$E(\mathbf{w}) = \frac{1}{2n} \sum_{k=1}^{n} (y^k - \mathbf{w}'\mathbf{x}^k)^2$$

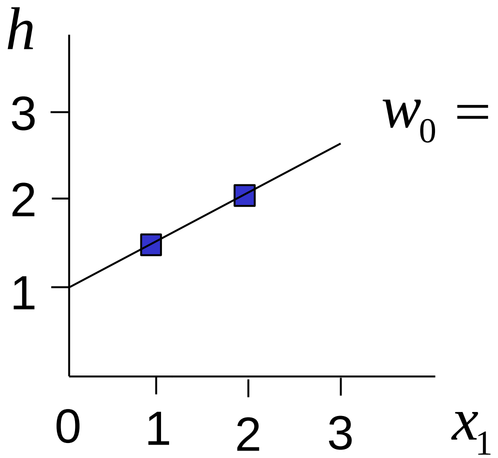
Minimization:

$$\min_{\mathbf{w}} E(\mathbf{w})$$

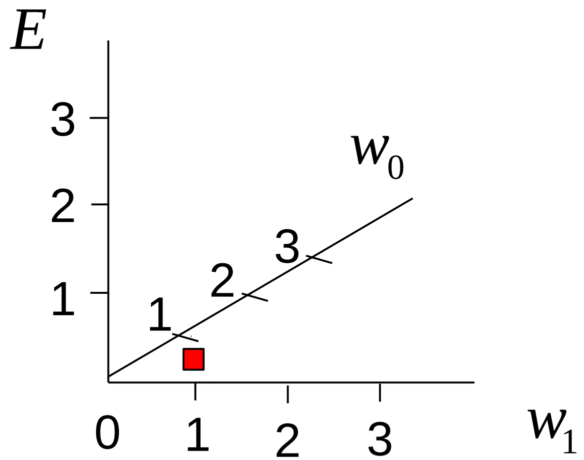
h vs. E

For a fixed w_0,w_1, h is a function of x_1.

For a fixed $x^k{}_1$'s, E is a function of w_1.



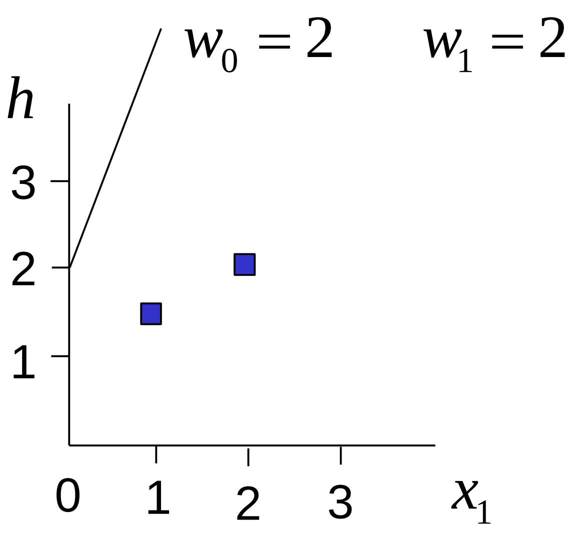
$w_0 = 1 \qquad w_1 = 0.5$



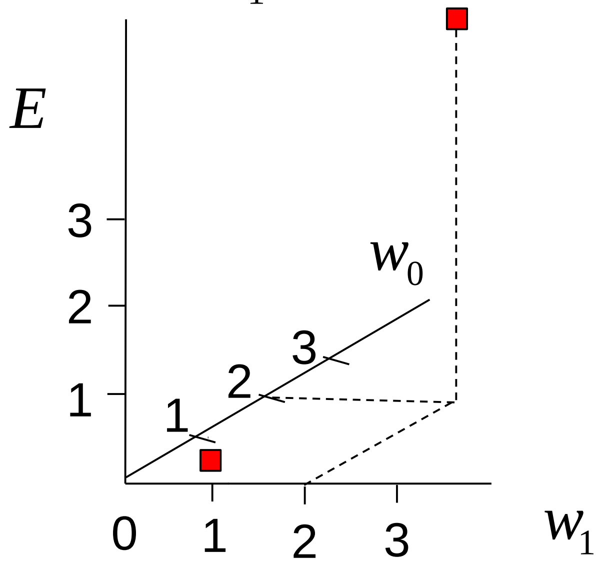
$$E(1{,}0.5) = \frac{1}{2 \cdot 2}\left[(1.5 - 1 - 0.5 \cdot 1)^2 + (2 - 1 - 0.5 \cdot 2)^2\right] = 0$$

h vs. E

For a fixed w_0, w_1, h is a function of x_1.



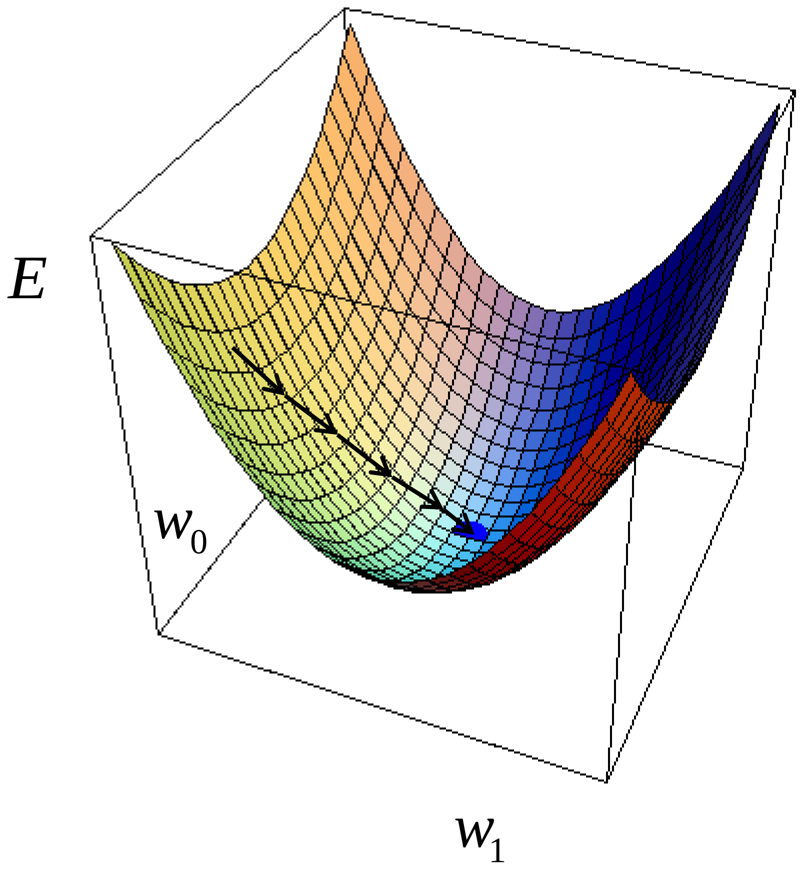
For a fixed x^k_1 's, E is a function of w_1.



$$E(2,2) = \frac{1}{2 \cdot 2}\left[(1.5 - 2 - 2 \cdot 1)^2 + (2 - 2 - 2 \cdot 2)^2\right] = 5.56$$

Minimization

- Start with some w_0, w_1,
- Nudge w_0, w_1 to lower E



Which direction to nudge?

Compute opposite of gradient

$$-\frac{\partial}{\partial w_0}E(w_0,w_1)$$

$$=-\frac{\partial}{\partial w_0}\left(\frac{1}{2n}\sum_{k=1}^{n}(y^k-x_0^k w_0-w_1x_1^k)^2\right)$$

$$=\frac{1}{n}\sum_{k=1}^{n}(y^k-x_0^k w_0-w_1x_1^k)x_0^k$$

$$=\frac{1}{n}\sum_{k=1}^{n}(y^k-\mathbf{w}'\mathbf{x}^k)x_0^k$$

$$-\frac{\partial}{\partial w_1}E(w_0,w_1)$$

$$=-\frac{\partial}{\partial w_1}\left(\frac{1}{2n}\sum_{k=1}^{n}(y^k-x_0^k w_0-w_1x_1^k)^2\right)$$

$$=\frac{1}{n}\sum_{k=1}^{n}(y^k-x_0^k w_0-w_1x_1^k)x_1^k$$

$$=\frac{1}{n}\sum_{k=1}^{n}(y^k-\mathbf{w}'\mathbf{x}^k)x_1^k$$

Vectorization

$$-\nabla_E(\mathbf{w}) = -\begin{bmatrix} \dfrac{\partial}{\partial w_0} E(w_0, w_1) \\ \dfrac{\partial}{\partial w_1} E(w_0, w_1) \end{bmatrix} = \begin{bmatrix} \dfrac{1}{n}\sum_{k=1}^{n}(y^k - \mathbf{w}'\mathbf{x}^k)x_0^k \\ \dfrac{1}{n}\sum_{k=1}^{n}(y^k - \mathbf{w}'\mathbf{x}^k)x_1^k \end{bmatrix} = \frac{1}{n}\sum_{k=1}^{n}\begin{bmatrix} (y^k - \mathbf{w}'\mathbf{x}^k)x_0^k \\ (y^k - \mathbf{w}'\mathbf{x}^k)x_1^k \end{bmatrix}$$

$$= \frac{1}{n}\sum_{k=1}^{n}(y^k - \mathbf{w}'\mathbf{x}^k)\begin{bmatrix} x_0^k \\ x_1^k \end{bmatrix}$$

$$= \frac{1}{n}\sum_{k=1}^{n}(y^k - \mathbf{w}'\mathbf{x}^k)\mathbf{x}^k$$

Gradient Recap

$$-\nabla_E(\mathbf{w}) = \frac{1}{n}\sum_{k=1}^{n}(y^k - \mathbf{w}'\mathbf{x}^k)\mathbf{x}^k$$

$$\mathbf{w} \leftarrow \mathbf{w} + \kappa\,\nabla_E(\mathbf{w})$$

$$\mathbf{w} \leftarrow \mathbf{w} + \kappa\,\frac{1}{n}\sum_{k=1}^{n}\left(y^k - \mathbf{w}'\mathbf{x}^k\right)\mathbf{x}^k$$

Variations

$$\mathbf{w}'\mathbf{x}^k = \mathbf{x}^{k'}\mathbf{w}$$

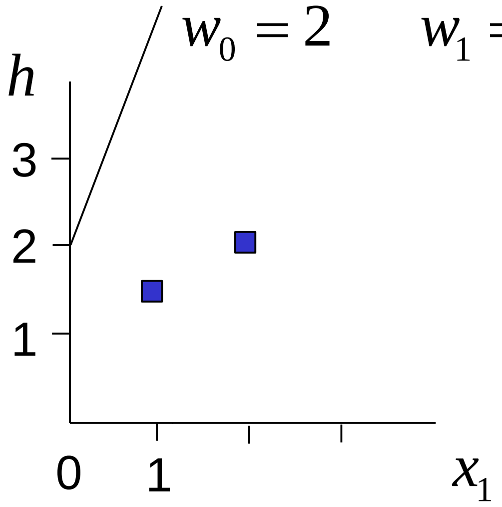
If you consider w and x^k **column** vectors

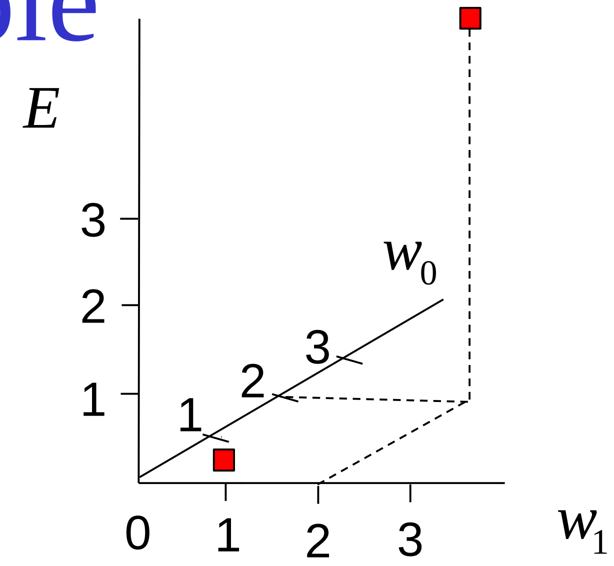
$$\mathbf{w}\mathbf{x}^{k'} = \mathbf{x}^k\mathbf{w}'$$

If you consider w and x^k **row** vectors

Either one is fine!

Example






```
X = np.array([[1,1],
              [1,2]])
y = np.array([[1.5, 2]])
w = np.array([[2,2]])
kappa = 0.1
n = 2

E = (1/(2*n)) * ( (y[0,0]-w@X[0].T)**2 + (y[0,1]-w@X[1].T)**2)
print("E = ", E)

w = w + kappa*( (1/n)*( (y[0,0]-w@X[0].T)*X[0] + (y[0,1]-w@X[1].T)*X[1] ) )
print("new w = ", w)

E = (1/(2*n)) * ( (y[0,0]-w@X[0].T)**2 + (y[0,1]-w@X[1].T)**2)
print("new E = ", E)
```

Output

```
E = [ 5.5625]
new w = [[ 1.675 1.475]]
new E = [ 2.40328125]
```

$$E(\mathbf{w}) = \frac{1}{2n}\sum_{k=1}^{n}(y^k - \mathbf{w}'\mathbf{x}^k)^2$$

$$-\nabla_E(\mathbf{w}) = \frac{1}{n}\sum_{k=1}^{n}(y^k - \mathbf{w}'\mathbf{x}^k)\mathbf{x}^k$$

Another version

```
X = np.array([[1,1],
              [1,2]])

y = np.array([[1.5, 2]])

w = np.array([[2,2]])

kappa = 0.1

n = 2

E = (1/(2*n)) * ( np.sum((y-w@X.T)**2) )
print("E = ", E)

w = w + kappa*( (1/n)*( np.sum( (y-w@X.T).T*X, axis=0, keepdims=True ) ) )
print("new w = ", w)

E = (1/(2*n)) * ( np.sum((y-w@X.T)**2) )
print("new E = ", E)
```

$$E(\mathbf{w}) = \frac{1}{2n}\sum_{k=1}^{n}(y^k - \mathbf{w}'\mathbf{x}^k)^2$$

$$-\nabla_E(\mathbf{w}) = \frac{1}{n}\sum_{k=1}^{n}(y^k - \mathbf{w}'\mathbf{x}^k)\mathbf{x}^k$$

More than one attribute

GPA	YearsOfExperience	Salary
90	1	50
80	3	60
90	2	55
70	8	70
…	…	…

y

More than one attribute

$$h(\mathbf{w},\mathbf{x}) = w_1 x_1 + \ldots + w_m x_m + w_0 x_0 = \mathbf{w}'\mathbf{x}$$

$$E(\mathbf{w}) = \frac{1}{2n}\sum_{k=1}^{n}(y^k - \mathbf{w}'\mathbf{x}^k)^2$$

$$\nabla_E(\mathbf{w}) = -\frac{1}{n}\sum_{k=1}^{n}\left(y^k - \mathbf{w}'\mathbf{x}^k\right)\mathbf{x}^k$$

E(**w**) and gradient same as before.

Gradient Descent Algorithm

Initialize at some $\mathbf{w}_0$

For t=0,1,2,…do

Compute the gradient $$\nabla_E(\mathbf{w}_t) = -\frac{1}{n}\sum_{k=1}^{n}\mathbf{x}^k\left(y^k - \mathbf{w}_t'\mathbf{x}^k\right)$$

Update the weights $$\mathbf{w}_{t+1} = \mathbf{w}_t - \kappa\nabla_E(\mathbf{w}_t) = \mathbf{w}_t + \kappa\frac{1}{n}\sum_{k=1}^{n}\mathbf{x}^k\left(y^k - \mathbf{w}_t'\mathbf{x}^k\right)$$

Iterate with the next step until **w** doesn't change too much

(or for a fixed number of iterations)

Return final **w**.

Attribute Scaling

- In order for Gradient Descent to converge quickly, scale attributes, e.g.

$$f_1' = \frac{f_1 - m_1}{\max_1 - \min_1} \qquad \text{or} \qquad f_1' = \frac{f_1 - m_1}{s_1}$$

where m_1 is the mean (average) of f_1, and $\max_1$, $\min_1$, s_1 are the max, min, and stdev of values for f_1.

Don't scale the all 1's attribute.

Learning Rate

- For small learning rate kappa, the value of *E* should decrease in each iteration.
  - If this doesn't happen, kappa is too big, so decrease kappa.
  - However, don't make kappa too small as GD will be slow to converge.

- Practical advise:
  - Start with kappa=1, and decrease it if too big.

THE MATRIX WAY:
CANONICAL EQUATIONS

Matrix **X** and vector **y**

x_0	GPA	YearsOfExperience	Salary
1	90	1	50
1	80	3	60
1	90	2	55
1	70	8	70

$$\mathbf{X} = \begin{bmatrix} 1 & 90 & 1 \\ 1 & 80 & 3 \\ 1 & 90 & 2 \\ 1 & 70 & 8 \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} 50 \\ 60 \\ 55 \\ 70 \end{bmatrix}$$

The Matrix Way: Canonical Equations

- *E* will have the smallest value when the gradient is equal to zero.

$$\nabla_E(\mathbf{w}) = -\frac{1}{n}\sum_{k=1}^{n}\mathbf{x}^k\left(y^k - \mathbf{w}'\mathbf{x}^k\right) = \mathbf{0}$$

$$\sum_{k=0}^{n}\mathbf{x}^k\left(y^k - \mathbf{w}'\mathbf{x}^k\right) = \mathbf{0}$$

$$\sum_{k=1}^{n}\left(\mathbf{x}^k y^k - \mathbf{x}^k\mathbf{w}'\mathbf{x}^k\right) = \mathbf{0}$$

$$\sum_{k=1}^{n}\mathbf{x}^k y^k = \sum_{k=1}^{n}\mathbf{x}^k\mathbf{w}'\mathbf{x}^k$$

$$\mathbf{X}'\mathbf{y} = \left(\mathbf{X}'\mathbf{X}\right)\mathbf{w}$$

$$\left(\mathbf{X}'\mathbf{X}\right)^{-1}\mathbf{X}'\mathbf{y} = \mathbf{w}$$

- **X** is the $n \times (m+1)$ data matrix
  - one row of $m+1$ elements for each data instance
    - without the y attribute
- **y** is the n-vector of class values
- **X'X** is $(m+1) \times (m+1)$ matrix
  - Good if number m of attributes is not too big.
- **w** is m-vector, i.e. $(m+1) \times 1$

Python

```
X=np.array([
[1,90,1],
[1,80,3],
[1,90,2],
[1,70,8]])

y=np.array([[50],[60],[55],[70]]);

w = np.linalg.pinv(X.T @ X)  @ X.T @ y

Result
w =
   86.5
   -0.4
    1.5
```

n training tuples
m attributes

Discussion:
GD vs. canonical equations

GD

Needs to iterate, sometimes a lot.

Need to play with kappa.

Method of choice when *m* is big (>10,000).

Canonical equations

No need to iterate, adjust kappa, or scale attributes.

However, the challenge is to compute:

$$(\mathbf{X}'\mathbf{X})^{-1}$$

X'X not a problem;

result is $(m+1) \times (m+1)$

Inverting takes ~$O(m^3)$ time.

Fine for m<10,000, difficult after that.

What if $X^T X$ is non-invertible

- Causes:
  - Some columns are linearly dependent
    - E.g. salary is given in two columns; both in CAD and USD
  - Too many attributes, few tuples

- Solution:
  - Delete some attributes
  - Use regularization (developed later in the course)