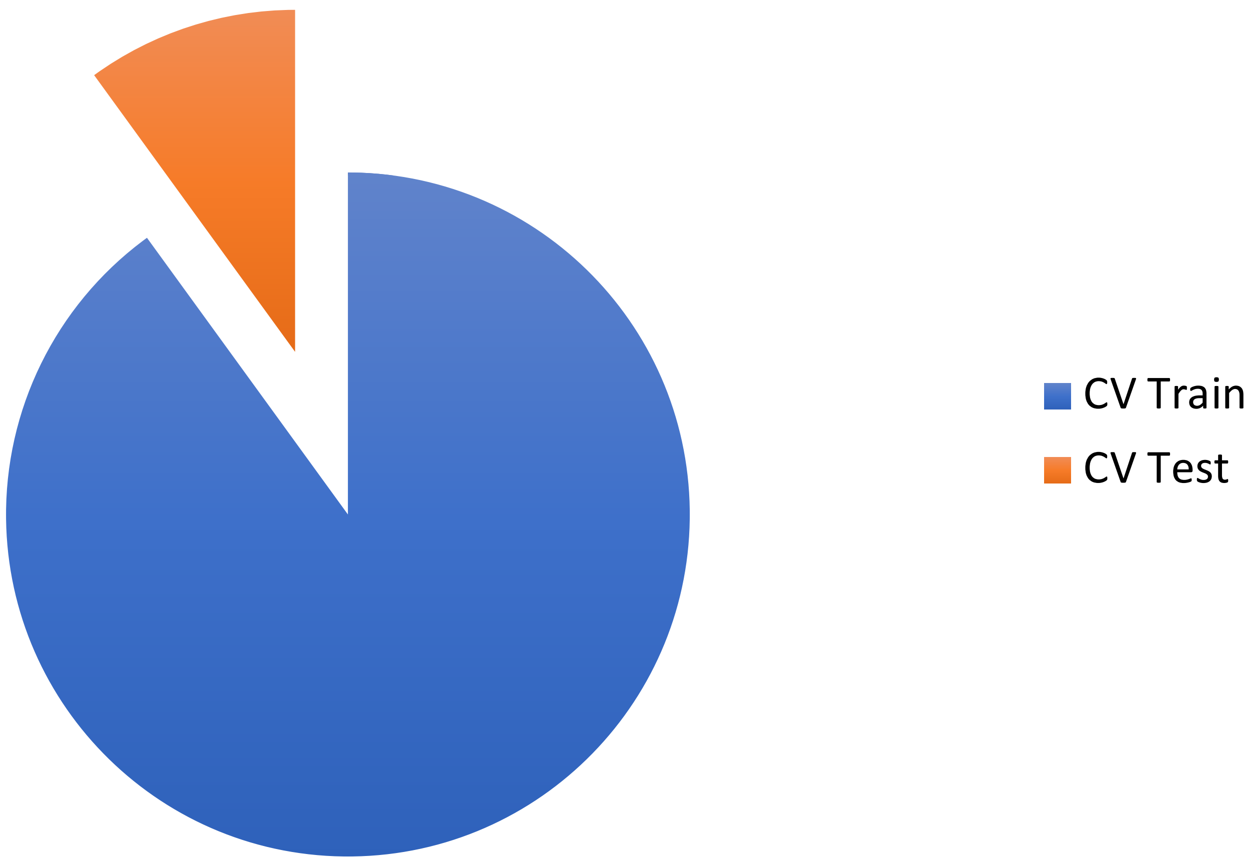
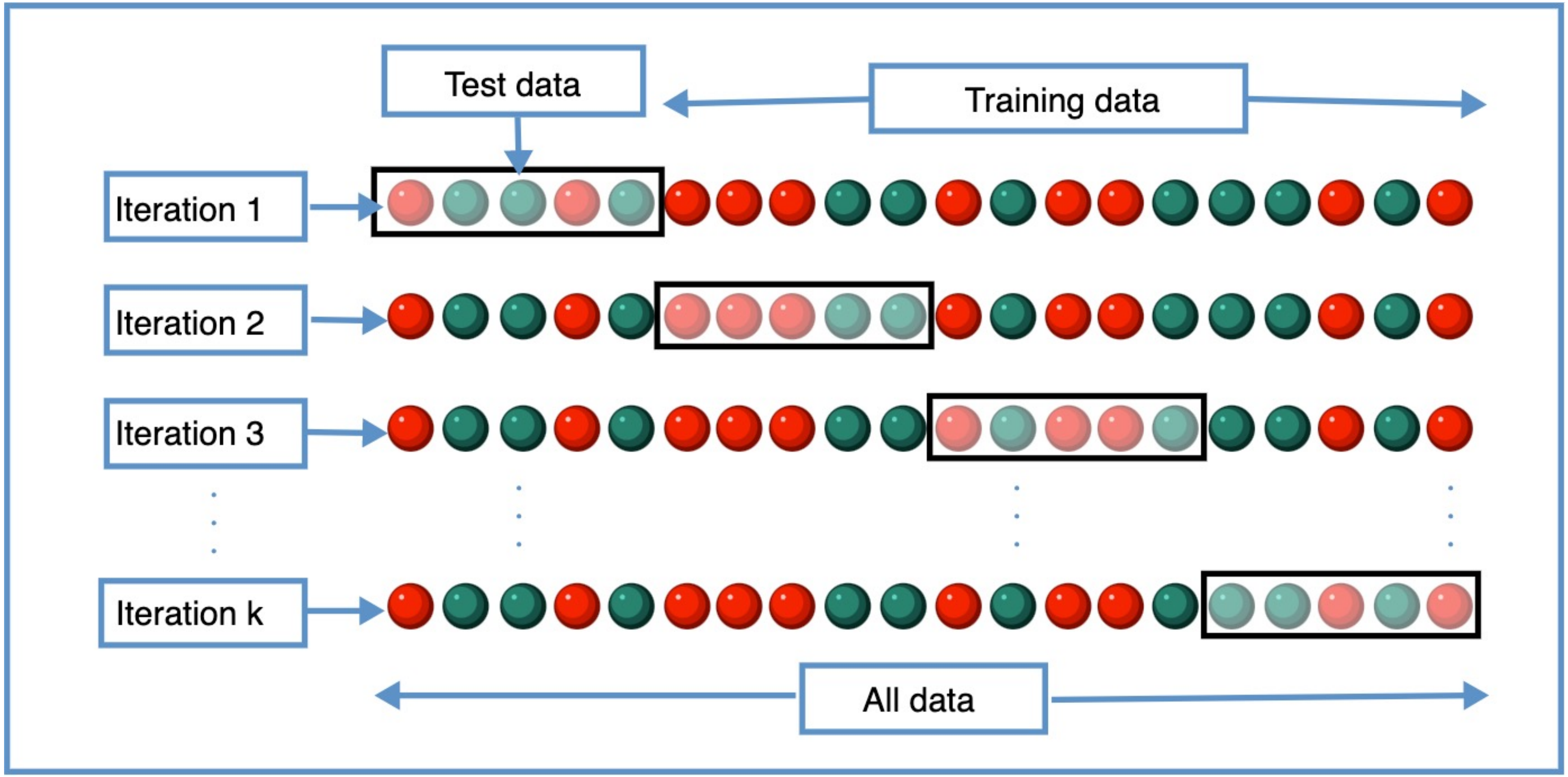
Evaluation cont.
Optimization/Grad descent

Cross Validation



Cross Validation



Practical issues for CV

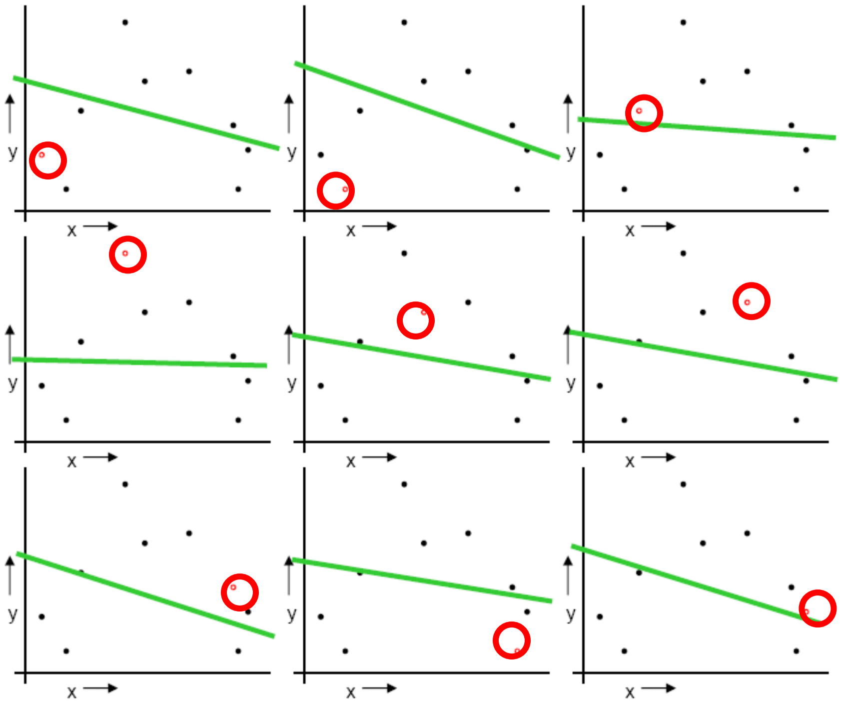
- How to big of a slice of the pie?
  - Commonly used K = 10 folds (thus each fold is 10% of the data)
  - Leave-one-out-cross-validation LOOCV (K=N, number of training instances)

- One important point is that (for a particular fold) the test data is never used for training, because doing so would result in overly (indeed dishonest) optimistic accuracy rates during the testing phase.

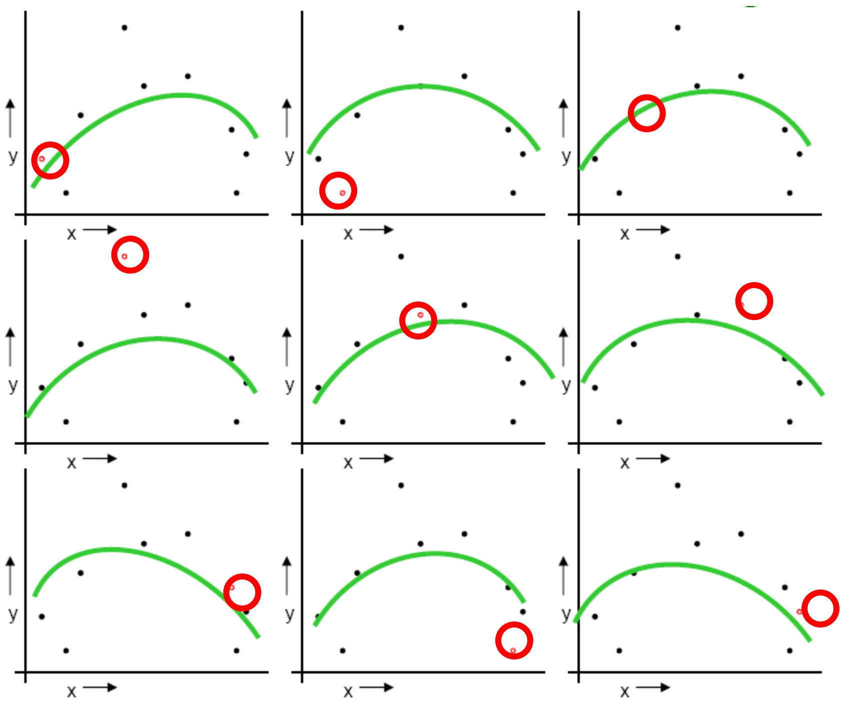
- Stratification – should you balance the classes across the folds?

Example:

- When $k=N$, the algorithm is known as Leave-One-Out-Cross-Validation (LOOCV)



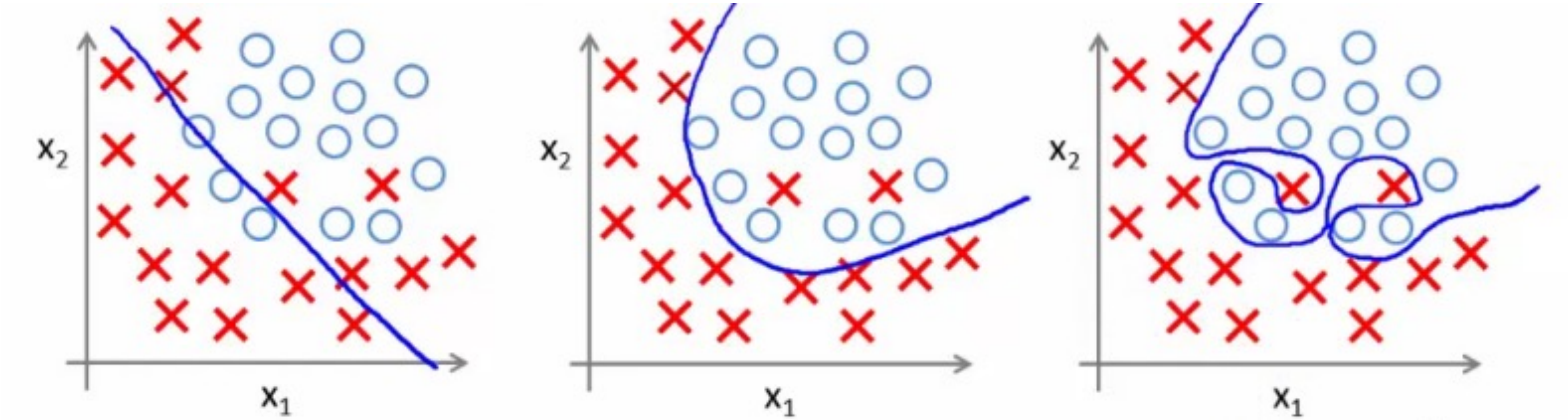
MSELOOCV(M_1)=2.12



MSELOOCV(M_2)=0.962

MSE explained later in these slides

Why is CV so important?



UNDERFITTING (high bias)

OVERFITTING (high variance)

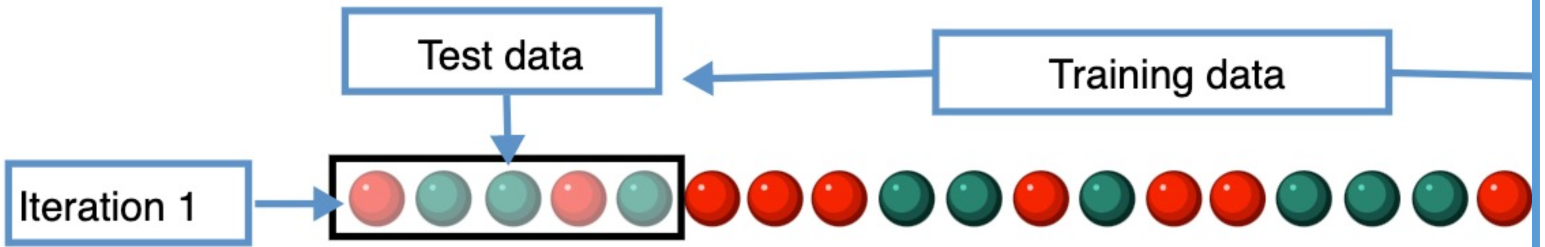
How to handle tuning hyperparameters

- **Key idea: data used to choose hyperparams should not be used to calculate the reported accuracy**

- Two regimes:
  1. Tune with Validation Set (no cross validation, splits fixed)
     - Split into Train/Validation/Test (e.g. 70,10,20%)
     - Train on Training data, test on validation to set hyperparameters or choose an algorithm
     - Report final accuracy by training on all of training data (with your final chosen parameters) and predicting on test data.

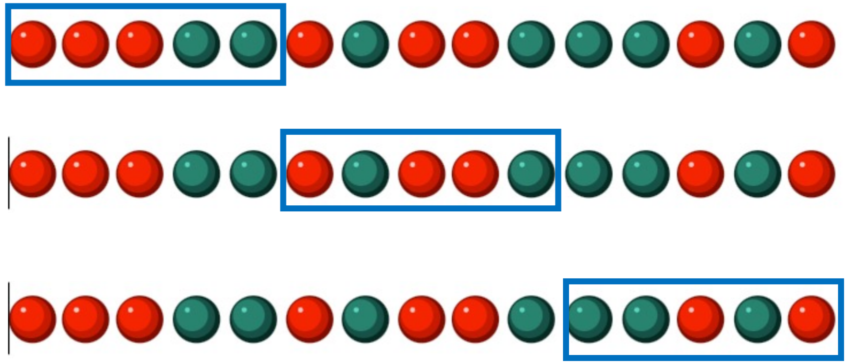
How to handle tuning hyperparameters

2. Nested Cross Validation. Takes two ks: k_1 k_2
   - Partition into k_1 sets
   - For each hyperparameter setting h:
     I. For each set of $(k_1-1)/k_1$ Train, $1/k_1$ Test (e.g. for each 90, 10% split)
        - Partition Train (e.g. 90% of the data from step I) into k_2 sets
        i. For each set of $(k_2-1)/k_2$ *sub-Train*, $1/k_2$ *sub-Test* (e.g. 0.9*0.9=81% of all data, 0.1*0.9=9% of all data)
           a. Train on *sub-Train* from step i using hyperparams h
           b. Test on *sub-Test* from step i
        - Calculate average performance across all k_2 splits for hyperparam h
     - Return hyperparam h' that maximizes performance
     II. Train on all Train data from step I using hyperparam h', test on Test data from step I. Record performance
   - Report average performance across all k_1 folds of Train and Test from step II



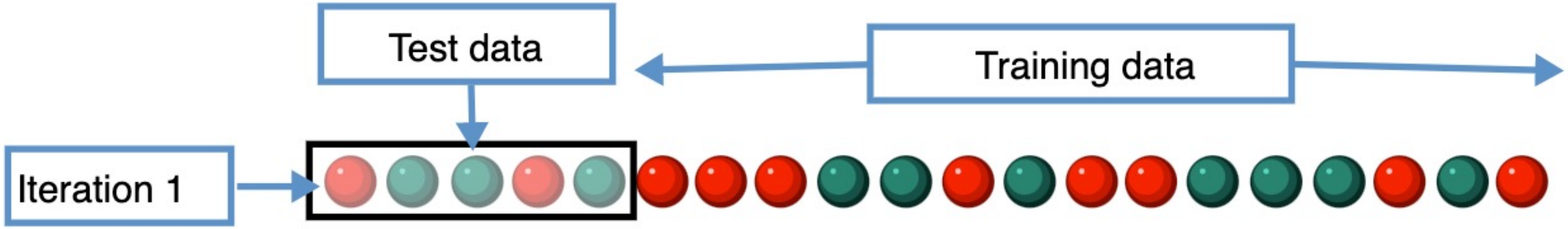
Important: the data I use to choose a hyperparam is **not** used to calculate the **test** accuracy with that hyperparam in the **outer** loop!

Run x-val on this train data, with each hyperparam



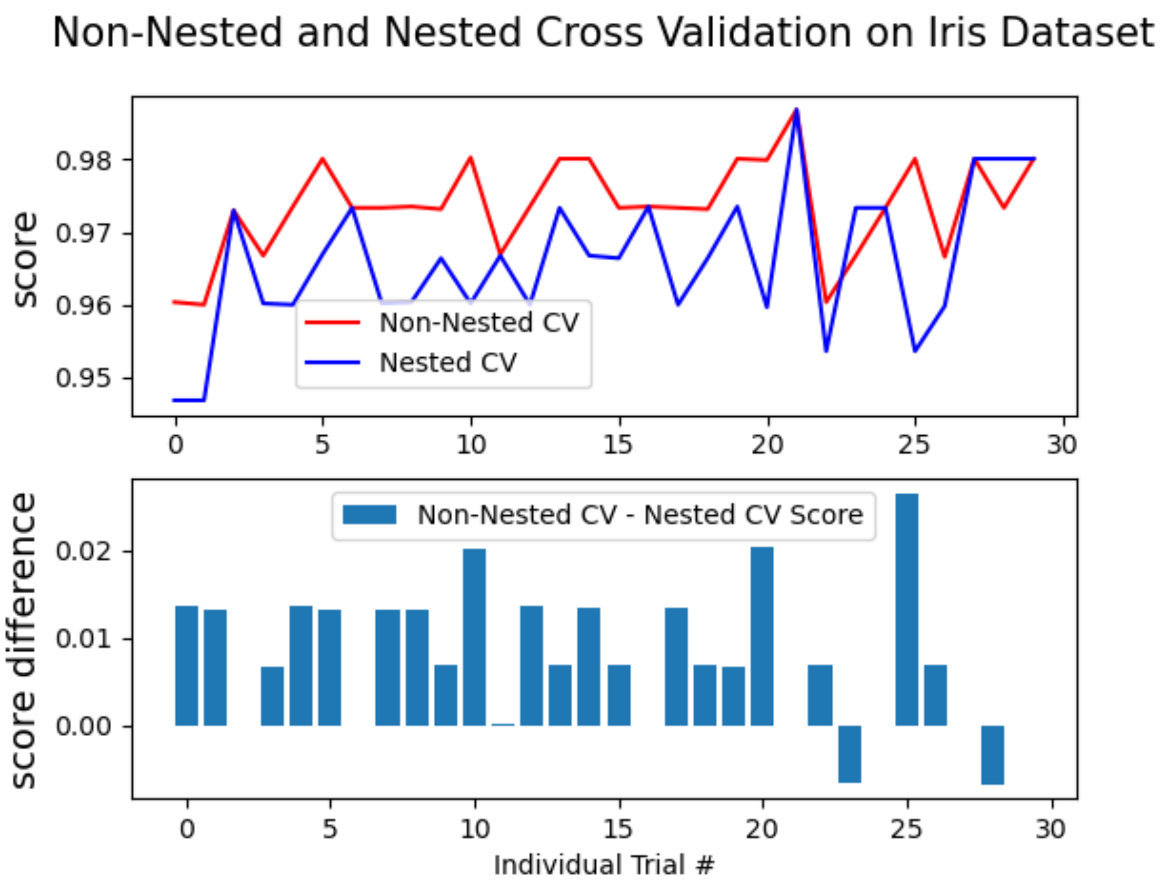
h1: 0.5, h2: 0.7, h3: 0.6, h4: 0.9 -> Best accuracy is with h4

Train on all of the training data using h4, test on test data, and save accuracy for this iteration



Repeat for all outer folds, then report average accuracy

Nested Cross Validation



https://scikit-learn.org/stable/auto_examples/model_selection/plot_nested_cross_validation_iris.html

See also

- https://mlfromscratch.com/nested-cross-validation-python-code/#/

Other Evaluation Methods

- Random subsampling / Monte Carlo cross validation
  - choose a test set randomly and repeatedly, without replacement
  - like cross-validation except test sets need not be disjoint
  - Can be nested
- Bootstrap
  - choose a test set randomly with replacement
  - like random sampling, but with replacement
  - Pessimistic estimate, corrected with .632 bootstrap estimate

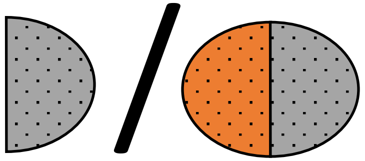
More info: http://web.cs.iastate.edu/~jtian/cs573/Papers/Kohavi-IJCAI-95.pdf

Measuring Performance (Classification)

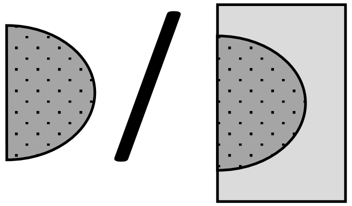
- Accuracy
  - (# test instances correctly labeled)/(# test instances)
- Error
  - 1- accuracy
  - (# test instances incorrectly labeled)/(# test instances)

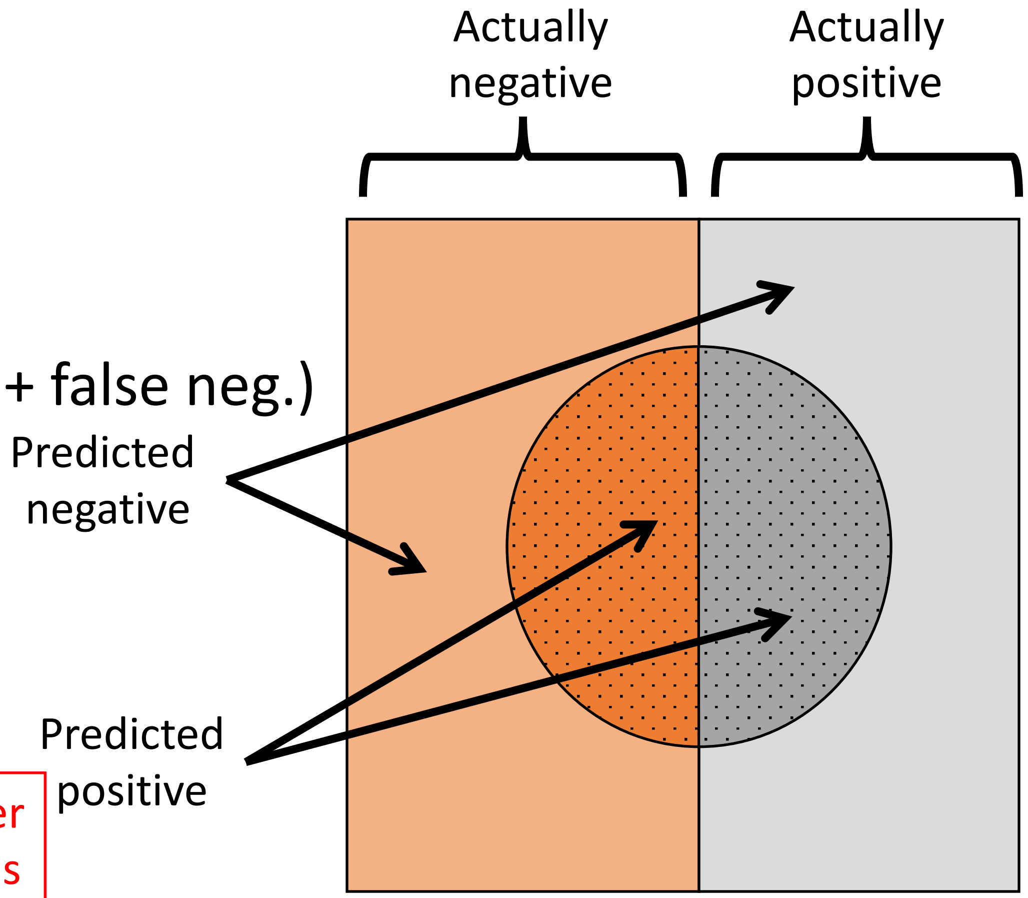
Measuring Performance (Classification)

- Precision
  - true positives/(true pos. + false pos.)



- Recall
  - true positives/(true pos. + false neg.)





How to remember: the first word is whether the prediction is correct, the second word is what the prediction was.

Measuring Performance (Classification)

- F1
  - harmonic mean of precision and recall
  - 2*(p*r)/(p+r)

Measuring Performance (Regression)

- Regression
  - predicting a real number
- Root Mean Squared Error (RMSE)
  - sometimes just MSE (no sqrt)

$$\sqrt{\frac{1}{N}\underbrace{\sum_{i=1}^{N}}_{\text{sum over all N test instances}}\underbrace{(\hat{y}_i - y_i)^2}_{\text{error}}}$$

Measuring Performance (Regression)

$$\sqrt{\frac{1}{N}\underbrace{\sum_{i=1}^{N}}_{\text{sum over all N test instances}}\underbrace{(\hat{y}_i - y_i)}_{\text{error}}}$$

- What is a “good” Root Mean Squared Error (RMSE)?
- That depends entirely on the problem!
  - Are you predicting a large or a small number?
  - How much do they vary?
- To account for this, we can report “percent of variance explained” or R^2

Measuring Performance (Regression)

- Percent of variance explained (POVE) or R^2

$$1 - \frac{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}{\frac{1}{N}\sum_{i=1}^{N}(\bar{y} - y_i)^2}$$

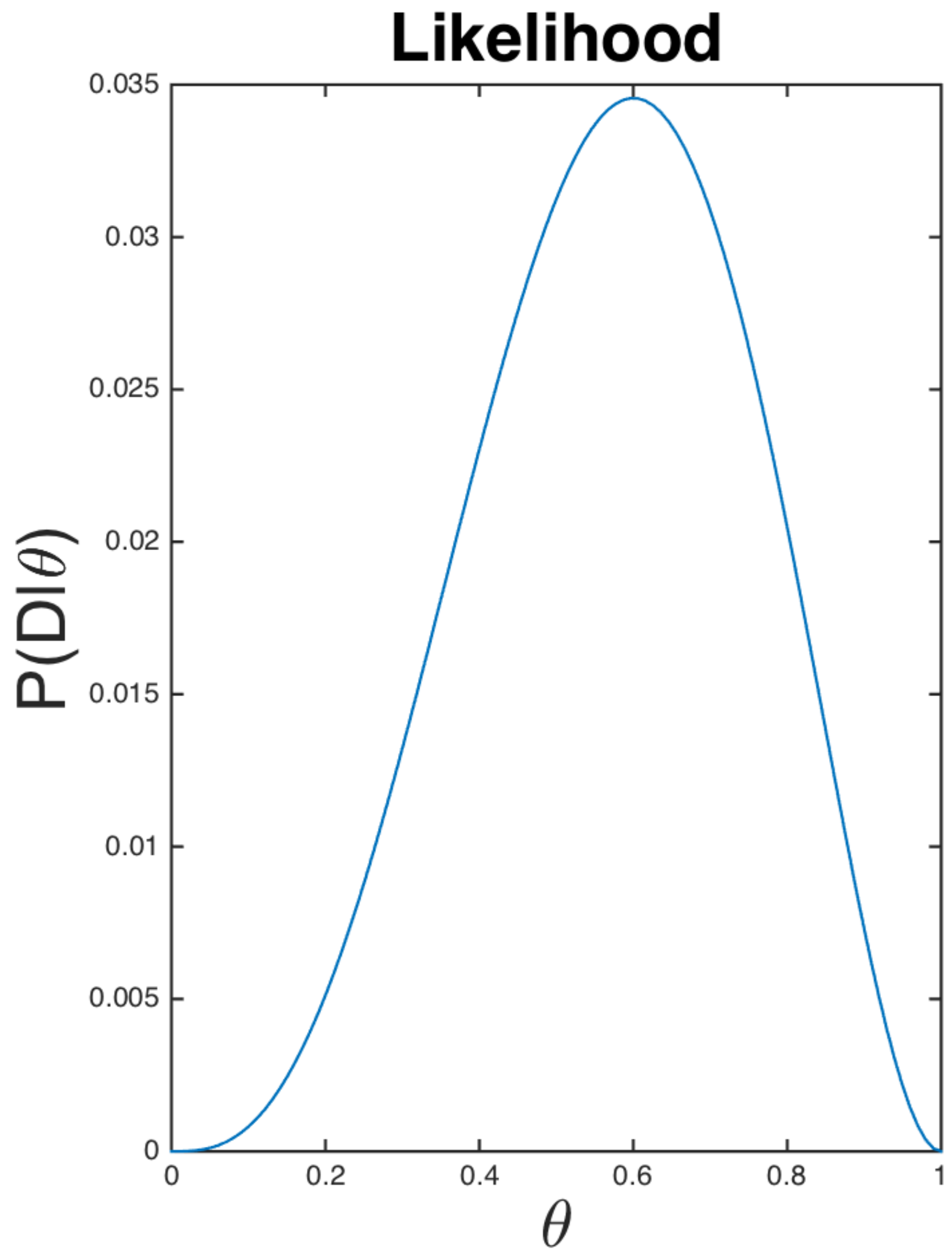
- This is just one minus (MSE scaled by the variance in Y)
  - Max is 1, and when things are performing poorly PVE can actually be negative!

Optimization

A little math $\underset{\theta}{\operatorname{argmax}}\ P(D) = (1-\theta)^{\alpha_P} * \theta^{\alpha_R}$

… take the log, take the derivative, set equal to zero, solve for theta…

$$\frac{\alpha_R}{\alpha_R + \alpha_P}$$


Likelihood
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
P(D|θ)
0
0.2
0.4
0.6
0.8
1
θ

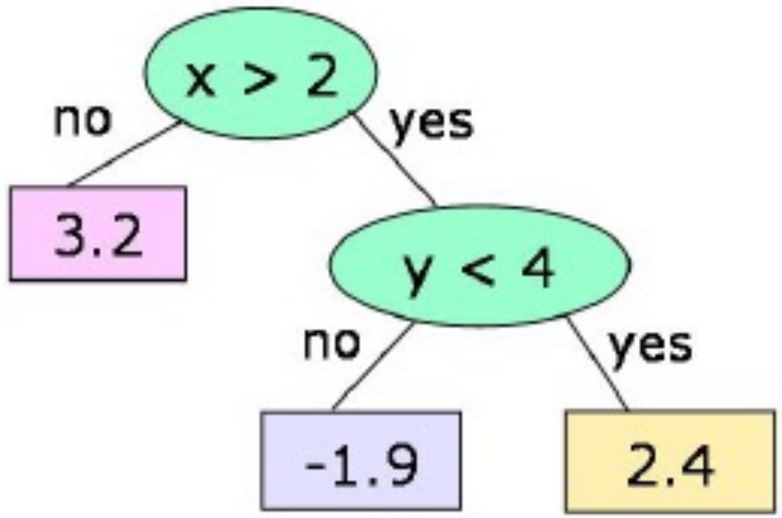
Optimization for linear regression

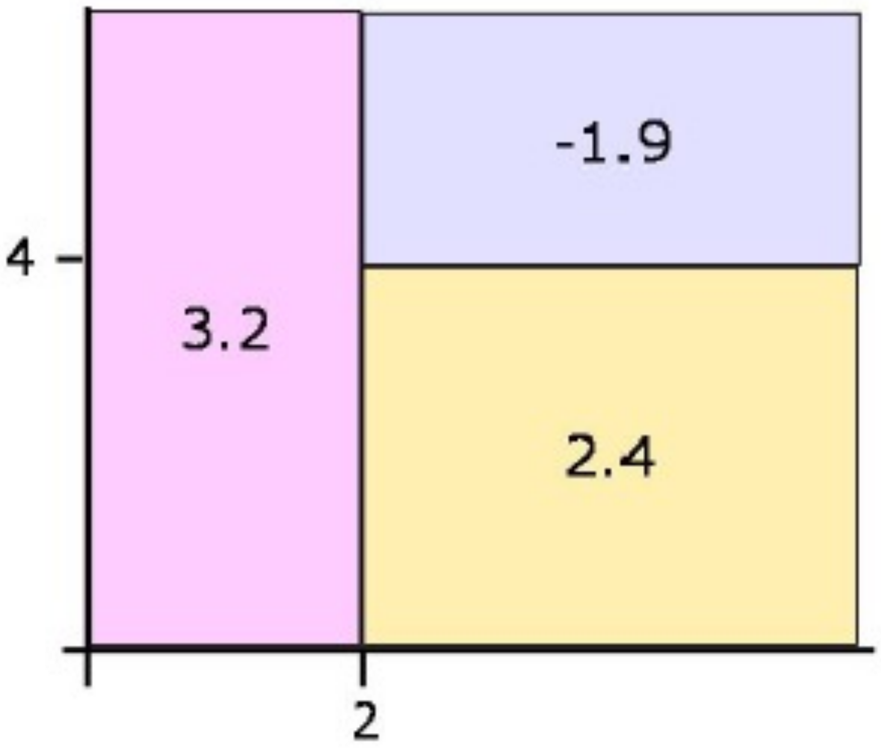
Regression

- Regression is predicting ___________, whereas classification is predicting _____________.

Regression Trees

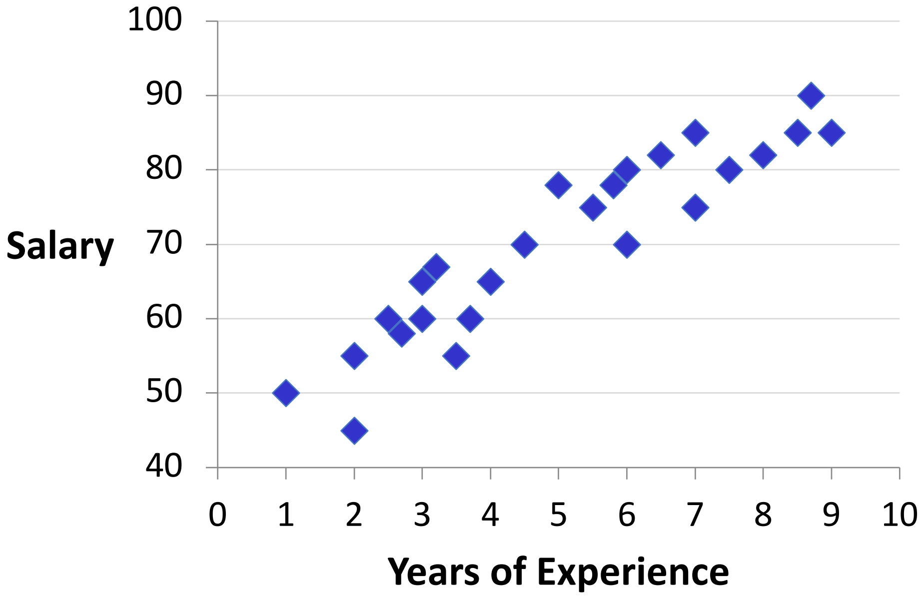
- Like decision trees, but with real-valued outputs at the leaves.





Another way to do regression

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 **target** (the thing to be predicted) is a **continuous attribute**.
  - Called **Regression**
- Linear Regression: Build a prediction **line**.

Linear regression with one variable

equation of the line, bias is w0, slope is w1

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

h is our hypothesis, our learned model

mathematical convenience we often append a 1 to the front of our x vector. So our x vector is [1, x_1], and has the same dim as our w vector [w_0, w_1], and now we can just write **w'x**

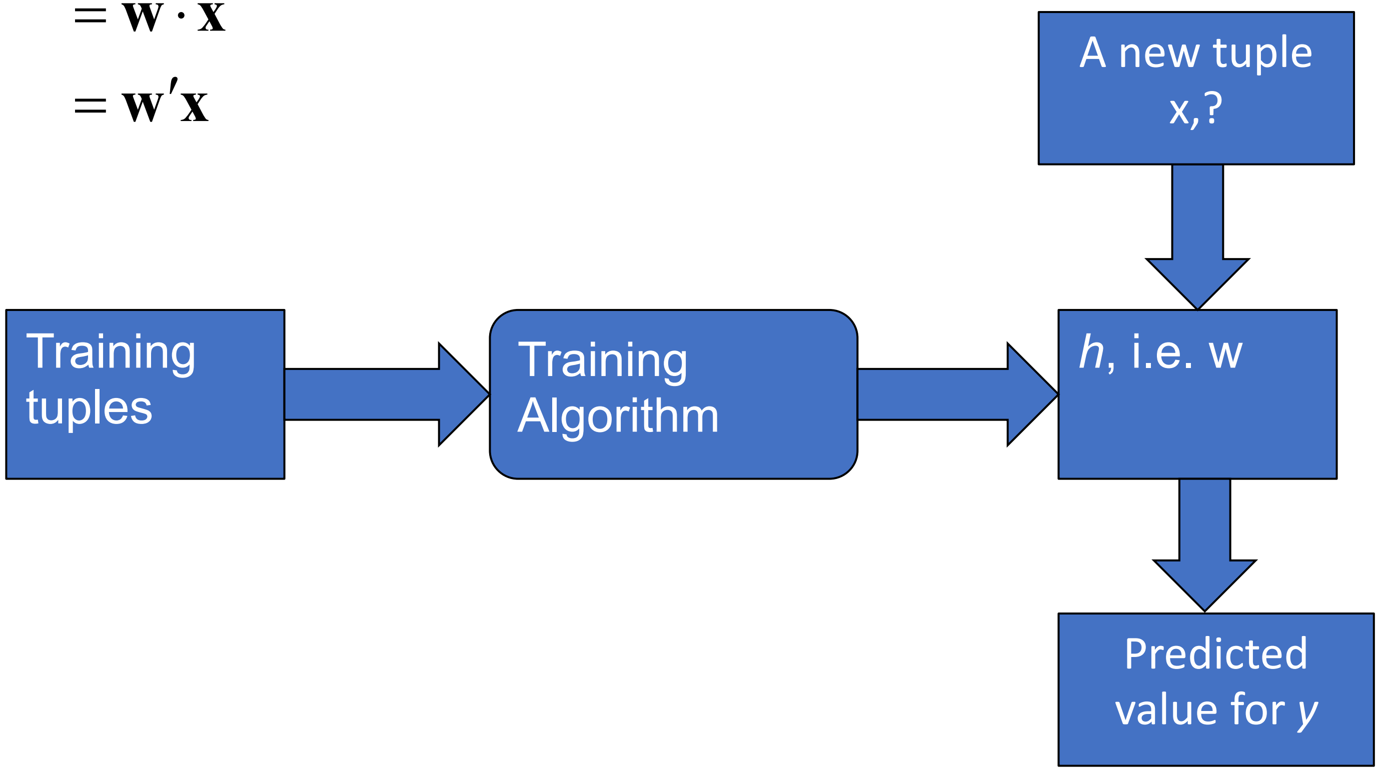
Linear regression with one variable

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

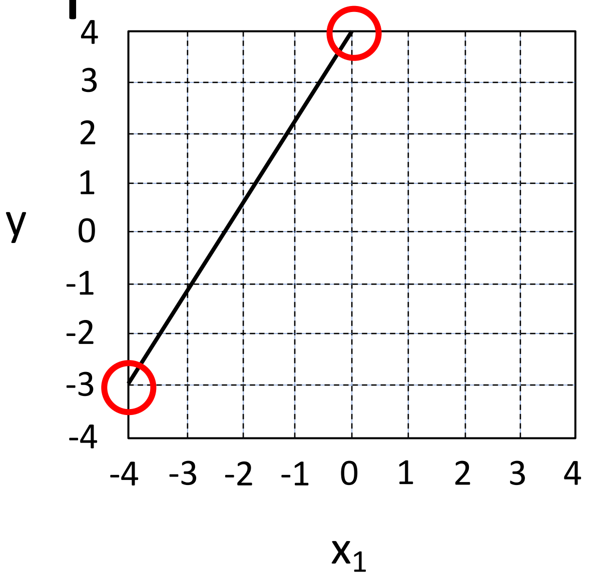
$$= w_0 x_0 + w_1 x_1 \qquad\qquad x_0 = 1$$

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

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



Example of line



$$w_0 + w_1 x_1 = y$$

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

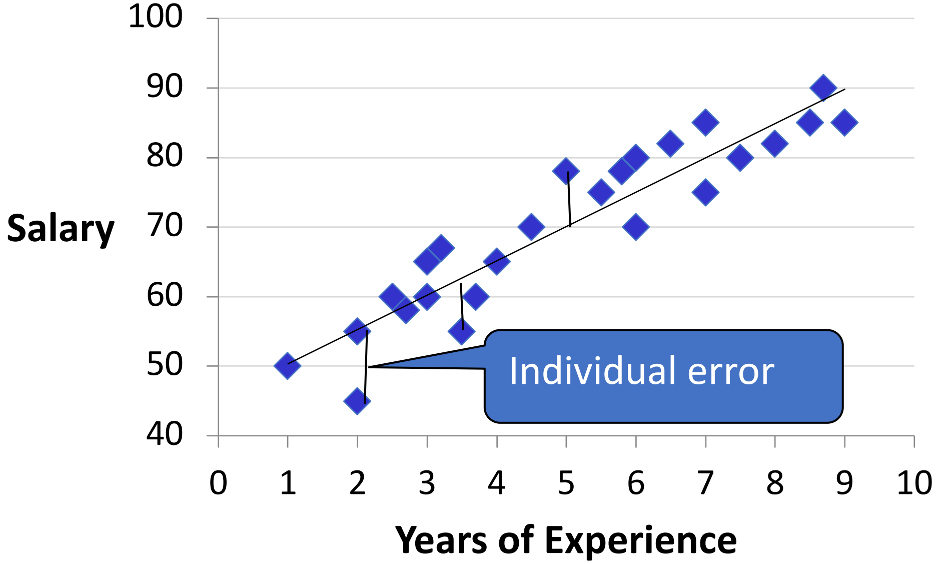
$$w_0 + w_1(0) = 4$$

$$\boxed{\begin{aligned} w_0 &= 4 \\ w_1 &= 7/4 \end{aligned}}$$

So if we knew the line… we could find the w's.

How can we find the line if all we have is data?

Cost/Error/Penalty Function



- Goal: find a line (hypothesis) $h(\mathbf{w},\mathbf{x})$ that for a given **training x**, gives a y value close to the **training *y***.

n is the number of training tuples

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

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

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

Recap

Hypothesis form:

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

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:

$\min_{\mathbf{w}} E(\mathbf{w})$ i.e. $\min_{w_0, w_1} E(w_0, w_1)$ i.e. $E(\mathbf{w}) = \frac{1}{2n} \sum_{k=1}^{n} (y^k - w_0 - w_1 x_1^k)^2$

Simplified *h* (for illustration)

Simplified hypothesis form (w_0=0), i.e. lines passing through the origin:

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

Weights to learn:

$$w_1$$

Error function:

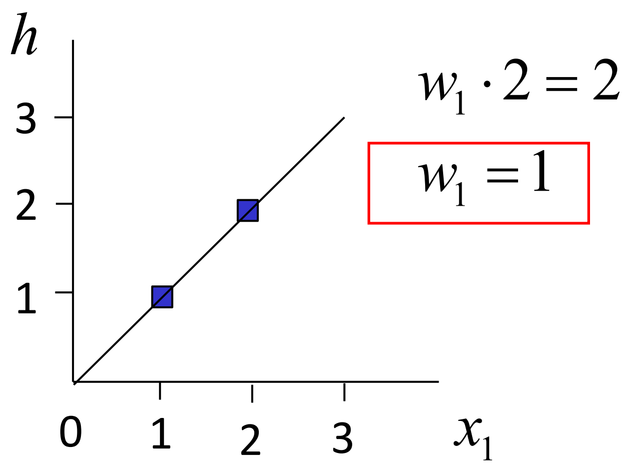
$$E(\mathbf{w}) = \frac{1}{2n} \sum_{k=1}^{n} (y^k - \mathbf{w}'\mathbf{x}^k)^2$$

Minimization:

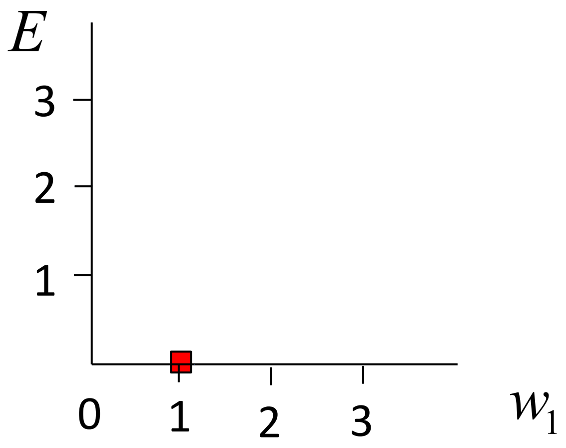
$\min_{\mathbf{w}} E(\mathbf{w})$ i.e. $\min_{w_1} E(w_1)$ i.e. $E(\mathbf{w}) = \frac{1}{2n} \sum_{k=1}^{n} (y^k - w_1 x_1^k)^2$

h vs. E

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



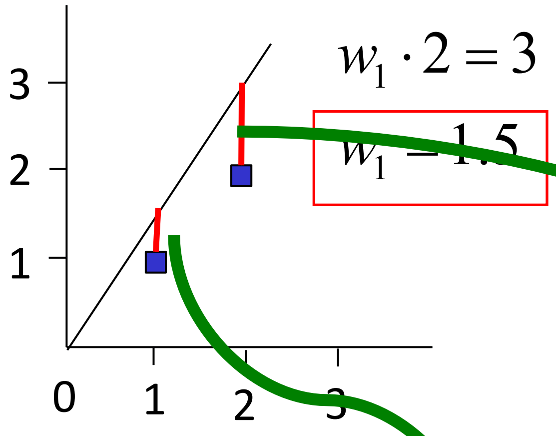
For a fixed x^k_1's, E is a function of w_1.



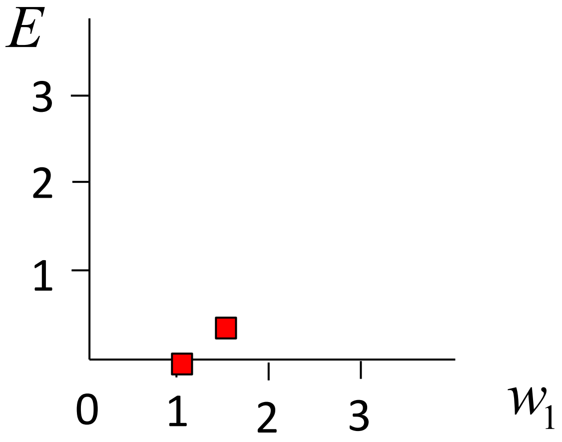
$$E(\mathbf{w}) = \frac{1}{2 \cdot 2}\left[(1 - 1 \cdot 1)^2 + (2 - 1 \cdot 2)^2\right] = 0$$

h vs. E

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



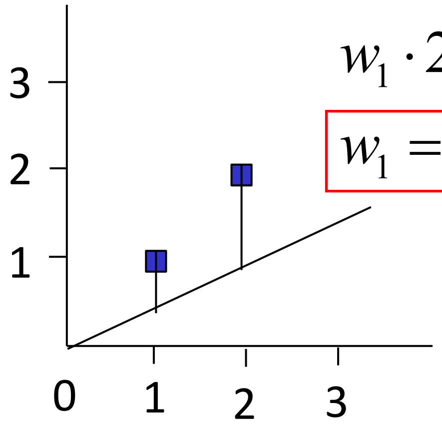
For a fixed $x^k{}_1$'s, E is a function of w_1.



$$E(\mathbf{w}) = \frac{1}{2 \cdot 2}\left[(1 - 1.5 \cdot 1)^2 + (2 - 1.5 \cdot 2)^2\right] = 0.3125$$

h vs. E

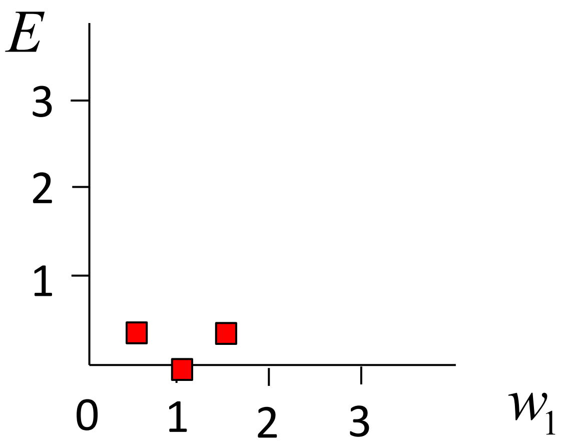
For a fixed w_1, h is a function of x_1.



$w_1 \cdot 2 = 1$

$\boxed{w_1 = 0.5}$

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



$$E(\mathbf{w}) = \frac{1}{2 \cdot 2}\left[(1 - 0.5 \cdot 1)^2 + (2 - 0.5 \cdot 2)^2\right] = 0.3125$$

$$\min_{w_1} E(w_1) \quad ? \qquad\qquad w_1 = 1$$

Let's do it again, now for w_0!=0

Hypothesis form:

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

Weights to learn:

$$w_0, w_1$$

Error function:

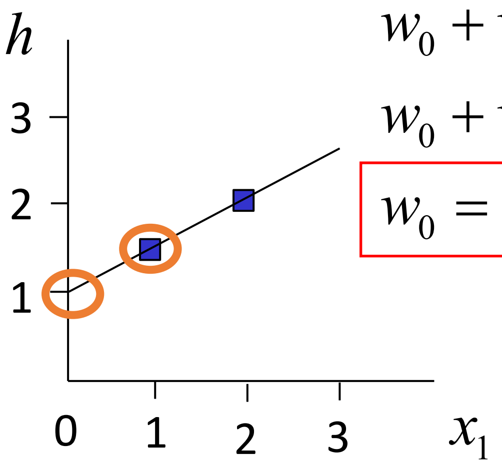
$$E(\mathbf{w}) = \frac{1}{2n}\sum_{k=1}^{n}(y^k - \mathbf{w}'\mathbf{x}^k)^2$$

Minimization:

$\min_{\mathbf{w}} E(\mathbf{w})$ i.e. $\min_{w_0, w_1} E(w_0, w_1)$ i.e. $E(\mathbf{w}) = \frac{1}{2n}\sum_{k=1}^{n}(y^k - w_0 - w_1 x_1^k)^2$

h vs. E

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



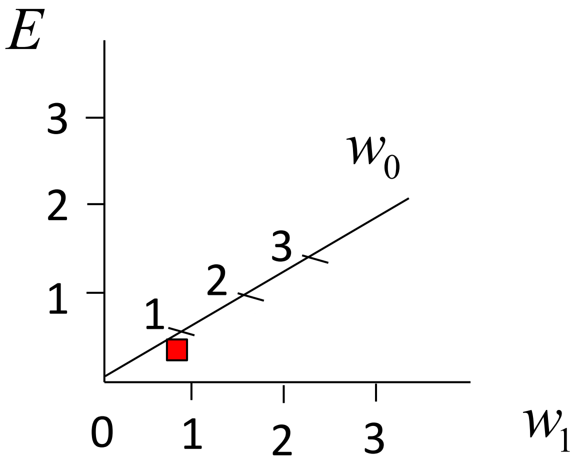
$$w_0 + w_1 \cdot 0 = 1$$

$$w_0 + w_1 \cdot 1 = 1.5$$

$$\boxed{w_0 = 1 \qquad w_1 = 0.5}$$

For a fixed x^k_1's,

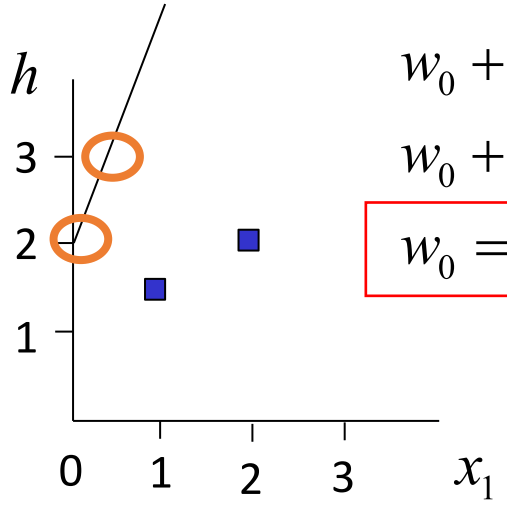
E is a function of w_1, w_0.



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

h vs. E

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

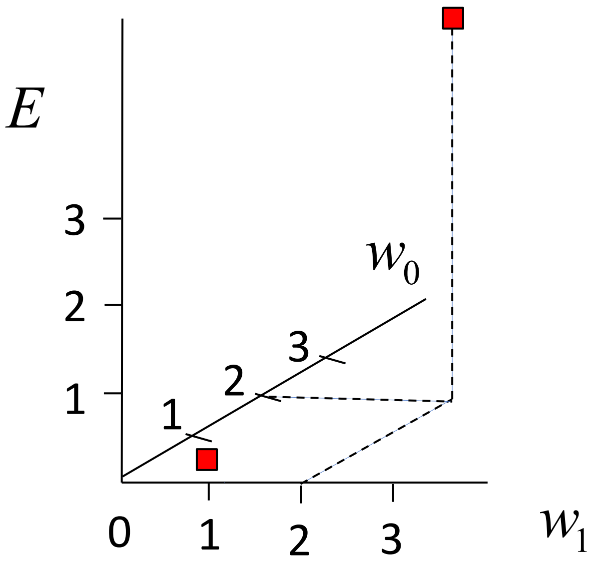


$$w_0 + w_1 \cdot 0 = 2$$

$$w_0 + w_1 \cdot 0.5 = 3$$

$$\boxed{w_0 = 2 \qquad w_1 = 2}$$

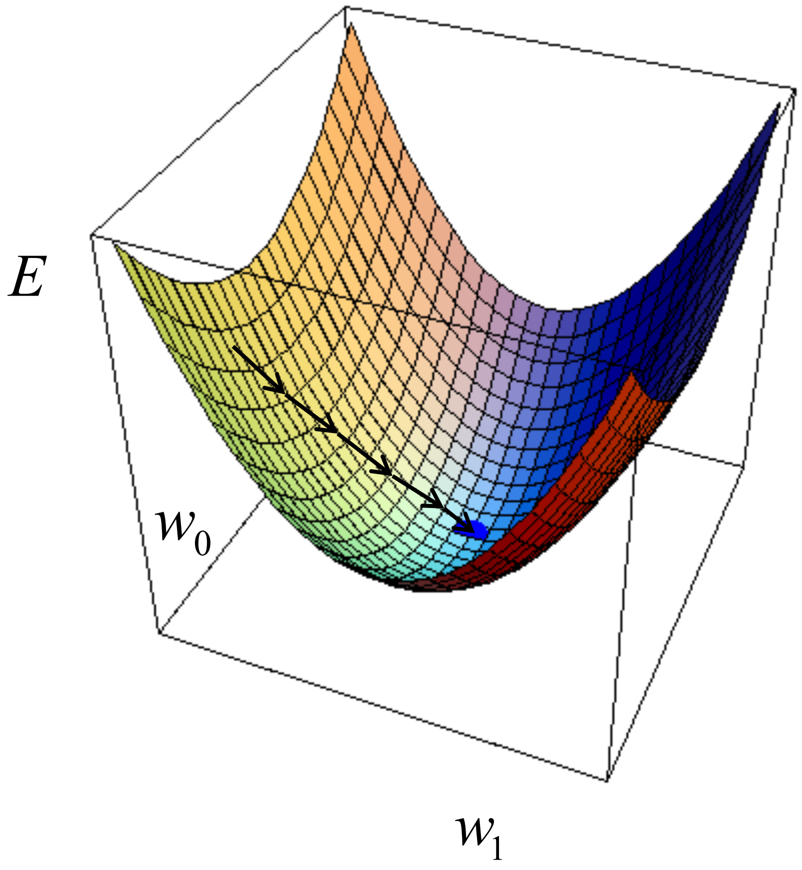
For a fixed $x^k{}_1$'s, E is a function of w_1.



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

Minimization

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



Which direction to nudge?

- Use opposite of gradient direction.

$$
\begin{aligned}
w_0 &\leftarrow w_0 - \kappa \frac{\partial}{\partial w_0} E(w_0, w_1) \\
&= w_0 - \kappa \frac{\partial}{\partial w_0} \left(\frac{1}{2n} \sum_{k=1}^{n} (y^k - w_0 - w_1 x_1^k)^2 \right) \\
&= w_0 - \kappa \frac{1}{2n} \sum_{k=1}^{n} \frac{\partial}{\partial w_0} (y^k - w_0 - w_1 x_1^k)^2 \\
&= w_0 + \kappa \frac{1}{n} \sum_{k=1}^{n} (y^k - w_0 - w_1 x_1^k)
\end{aligned}
$$

Which direction to nudge?

- Use opposite of gradient direction.

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

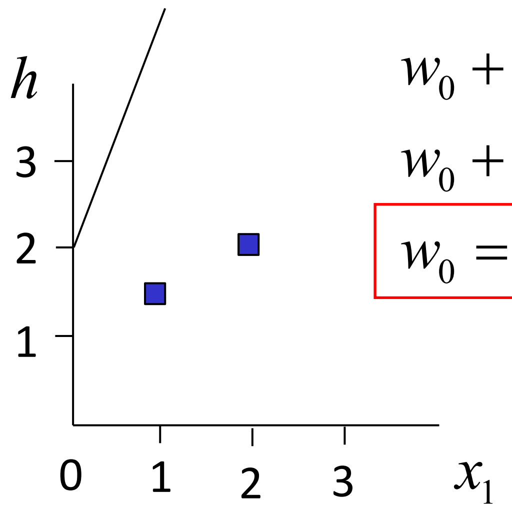
$$= w_1 - \kappa \frac{\partial}{\partial w_1} \left(\frac{1}{2n} \sum_{k=1}^{n} (y^k - w_0 - w_1 x_1^k)^2 \right)$$

$$= w_1 - \kappa \frac{1}{2n} \sum_{k=1}^{n} \frac{\partial}{\partial w_1} (y^k - w_0 - w_1 x_1^k)^2$$

$$= w_1 + \kappa \frac{1}{n} \sum_{k=1}^{n} (y^k - w_0 - w_1 x_1^k) x_1^k$$

Learning rate kappa
If kappa too small, GD is slow.
If kappa too big, GD is too eager and can overshoot min.

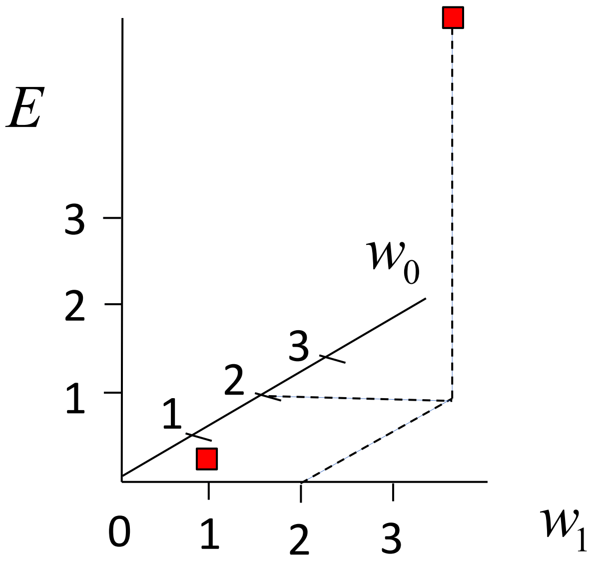
Example



$$w_0 + w_1 \cdot 0 = 2$$

$$w_0 + w_1 \cdot 0.5 = 3$$

$$\boxed{w_0 = 2 \qquad w_1 = 2}$$



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

$$w_0 \leftarrow w_0 + \kappa \frac{1}{n}\sum_{k=1}^{n}(y^k - w_0 - w_1 x_1^k)$$

$$= 2 + 0.1\frac{1}{2}\left(\left(1.5-2-2\cdot 1\right) + \left(2-2-2\cdot 2\right)\right)$$

$$= 1.675$$

$$w_1 \leftarrow w_1 + \kappa \frac{1}{n}\sum_{k=1}^{n}(y^k - w_0 - w_1 x_1^k)x_1$$

$$= 2 + 0.1\frac{1}{2}\left(\left(1.5-2-2\cdot 1\right)\cdot 1 + \left(2-2-2\cdot 2\right)\cdot 2\right)$$

$$= 1.475$$

$$E(1.675,1.475) = \frac{1}{2 \cdot 2}\left[(1.5-1.675-1.475\cdot 1)^2 + (2-1.675-1.475\cdot 2)^2\right] = 2.4$$

Vectorization

$$\frac{\partial}{\partial w_0} E(w_0, w_1)$$
$$= \frac{\partial}{\partial w_0}\left(\frac{1}{2n}\sum_{k=1}^{n}(y^k - w_0 - w_1 x_1^k)^2\right)$$
$$= \frac{1}{n}\sum_{k=1}^{n}(y^k - w_0 x_0^k - w_1 x_1^k)x_0^k$$
$$= \frac{1}{n}\sum_{k=1}^{n}(y^k - \mathbf{w}'\mathbf{x}^k)x_0^k$$

$$\frac{\partial}{\partial w_1} E(w_0, w_1)$$
$$= \frac{\partial}{\partial w_1}\left(\frac{1}{2n}\sum_{k=1}^{n}(y^k - w_0 - w_1 x_1^k)^2\right)$$
$$= \frac{1}{n}\sum_{k=1}^{n}(y^k - w_0 x_0^k - w_1 x_1^k)x_1^k$$
$$= \frac{1}{n}\sum_{k=1}^{n}(y^k - \mathbf{w}'\mathbf{x}^k)x_1^k$$

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

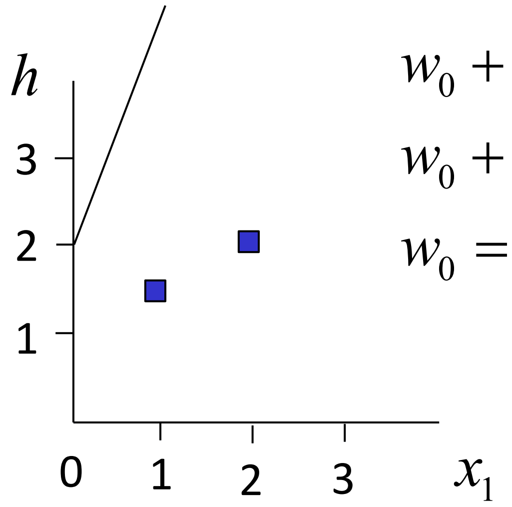
Gradient Recap

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

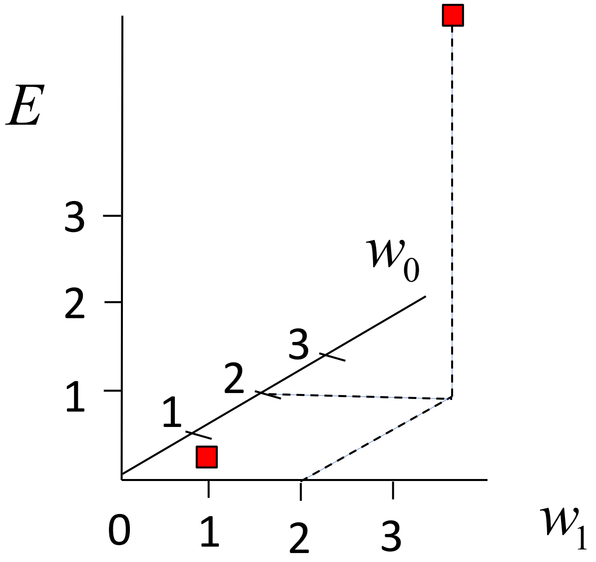
$$\mathbf{w} \leftarrow \mathbf{w} - \kappa \nabla_E(\mathbf{w})$$

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

Example



$$w_0 + w_1 \cdot 0 = 2$$
$$w_0 + w_1 \cdot 0.5 = 3$$
$$w_0 = 2 \qquad w_1 = 2$$



$$E\left(\begin{bmatrix}2\\2\end{bmatrix}\right) = \frac{1}{2\cdot 2}\left(\left(1.5-\begin{bmatrix}2 & 2\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}\right)^2 + \left(2-\begin{bmatrix}2 & 2\end{bmatrix}\begin{bmatrix}1\\2\end{bmatrix}\right)^2\right) = 5.56$$

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

$$= \begin{bmatrix}2\\2\end{bmatrix} + 0.1\frac{1}{2}\left(\left(1.5-\begin{bmatrix}2 & 2\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}\right)\begin{bmatrix}1\\1\end{bmatrix} + \left(2-\begin{bmatrix}2 & 2\end{bmatrix}\begin{bmatrix}1\\2\end{bmatrix}\right)\begin{bmatrix}1\\2\end{bmatrix}\right) = \begin{bmatrix}1.675\\1.475\end{bmatrix}$$

$$E\left(\begin{bmatrix}1.675\\1.475\end{bmatrix}\right) = \frac{1}{2\cdot 2}\left(\left(1.5-\begin{bmatrix}1.675 & 1.475\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}\right)^2 + \left(2-\begin{bmatrix}1.675 & 1.475\end{bmatrix}\begin{bmatrix}1\\2\end{bmatrix}\right)^2\right) = 2.4$$

Matlab/Octave

$$E\left(\begin{bmatrix}2\\2\end{bmatrix}\right)=\frac{1}{2\cdot 2}\left(\left(1.5-\begin{bmatrix}2 & 2\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}\right)^2+\left(2-\begin{bmatrix}2 & 2\end{bmatrix}\begin{bmatrix}1\\2\end{bmatrix}\right)^2\right)=5.56$$

```
E=(1/(2*2))*( (1.5-[2 2]*[1; 1])^2 + (2-[2 2]*[1; 2])^2 )
```

$$\mathbf{w}\leftarrow\begin{bmatrix}2\\2\end{bmatrix}+0.1\frac{1}{2}\left(\left(1.5-\begin{bmatrix}2 & 2\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}\right)\begin{bmatrix}1\\1\end{bmatrix}+\left(2-\begin{bmatrix}2 & 2\end{bmatrix}\begin{bmatrix}1\\2\end{bmatrix}\right)\begin{bmatrix}1\\2\end{bmatrix}\right)=\begin{bmatrix}1.675\\1.475\end{bmatrix}$$

```
w=[2;2]+0.1*(1/2)*
( (1.5-[2 2]*[1;1])*[1;1] + (2-[2 2]*[1;2])*[1;2] )
```

$$E\left(\begin{bmatrix}1.675\\1.475\end{bmatrix}\right)=\frac{1}{2\cdot 2}\left(\left(1.5-\begin{bmatrix}1.675 & 1.475\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}\right)^2+\left(2-\begin{bmatrix}1.675 & 1.475\end{bmatrix}\begin{bmatrix}1\\2\end{bmatrix}\right)^2\right)=2.4$$

```
E=(1/(2*2))*
( (1.5-[1.675 1.475]*[1; 1])^2 + (2-[1.675 1.475]*[1; 2])^2 )
```

More than one x attribute

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

y

Linear Approximation

$$y \approx w_1 x_1 + \ldots + w_m x_m + b$$

Approximate y given the attribute values by a linear function of the attributes.

w_0 is b

For neatness, let $\mathbf{w}' = [w_0, w_1, .., w_m]$ $\mathbf{x}' = [1, x_1, .., x_m]$

1 is an artificial, but completely harmless constant attribute we add to each training instance.

Then we can write the above in a neat form as

$$y \approx \mathbf{w}'\mathbf{x}$$

How to estimate the *w* parameters, i.e. **w**?

Cost function

- Find **w** that gives the lowest approximation error given the training data.
  - Minimize the **sum of square errors**:

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

Same **w** for all the training instances.

Iterative Method

- Start at some $\mathbf{w}_0$; take a step along **steepest slope**.
  - What's the steepest slope?

- Gradient of E:

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

Same form as before

Vectorized form makes it more general!

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**.

GD Matlab/Octave

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

Matlab/Octave:

```
w = w + kappa*(1/n)*(X'*(y-X*w));
```

Code in py notebook

- https://colab.research.google.com/drive/1xg87B_TPhoqlnNfHXIL8122lwSdmwW-6?usp=sharing

```
import numpy as np

# Training Data
X = np.matrix([[1, 1],
[1, 2]])
# These are the values we want to predict as a column vector
y = np.matrix([1.5, 2]).T

# This is the initial value for the weight vector
w = np.matrix([2, 2]).T

# Learning rate. Play with it to see how it changes the outcome
kappa = 0.1

# Loss function
loss = lambda w, X, y: np.mean( 1/2 * np.power((y - (X@w)), 2) )
print(f"Before optimization, loss is {loss(w, X, y) : .4f}")

# Gradient descent process
gradient = lambda w, X, y: 1/len(y) * X.T @ (X@w - y) # Gradient of
||X@w-y||
for t in range(1, 20):
w = w - kappa * gradient(w, X, y) # Move w to decrease ||X@w-y||
if t % 10 == 1:
print(f"Iteration {t}, loss is {loss(w, X, y) : .4f}")
print(f"After optimization, loss is {loss(w, X, y) : .4f}")
```

Summary

- If we have a differentiable loss function, we can use gradient descent to optimize!
- In cases where:
  - the loss function is convex, and
  - our learning rate is set correctly, and
  - we have sufficient data …

…we will find a solution that is “close” to the true global minimum

- If one of the above cases is not true, we can often find something “close enough”

Summary

- Stochastic gradient descent is gradient descent with one difference:

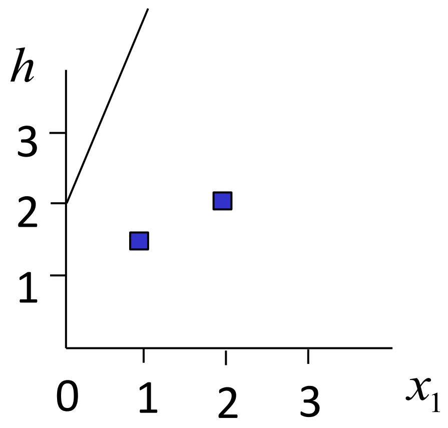
$$\nabla_E(\mathbf{w}_t) = -\frac{1}{n}\sum_{k=1}^{n}\mathbf{x}^k\left(y^k - \mathbf{w}_t'\mathbf{x}^k\right)$$

- We compute the gradient with a batch of data, rather than the whole dataset
  - Batches can be as small as 1 data point, or sets of 10, 50, 100…

Summary

- Stochastic gradient descent is how neural networks are trained!
  - There are multiple elaborations to the update function that we will talk about in a few lectures.

GD example in Matlab




```
 % Training data
X=[1 1;
   1 2];
% These are the values we want to predict
y=[1.5; 2];
% This is the starting assignment for the weight vector.
w=[2; 2];

% Learning rate.  Play with it to see how it changes the outcome!
kappa = 0.1;
n = length(y);

fprintf('Before optimization, loss is %.4f\n',mean(1/2*(y-X*w).^2));
for t=1:20,
    w = w + kappa*(1/n)*(X'*(y-X*w));
    if rem(t,10) ==1,
        fprintf('Iteration %i, loss is %.4f\n',t,mean(1/2*(y-X*w).^2));
    end
end;
fprintf('After optimization, loss is %.4f\n',mean(1/2*(y-X*w).^2));
w
```