Machine Learning

Topic: Gradient Descent

Max Morrison and Bryan Pardo, EECS 349 Machine Learning, 2021

Supervised Machine Learning in one slide

1. Pick data **X**, labels **Y**, model $\mathbf{M}(\boldsymbol{\theta})$ and loss function $\boldsymbol{L}(\mathbf{X}, \mathbf{Y}; \boldsymbol{\theta})$
2. Initialize model parameters $\boldsymbol{\theta}$, somehow
3. Measure model performance with the loss function $\boldsymbol{L}(\boldsymbol{X}, \boldsymbol{Y}; \boldsymbol{\theta})$

HOW?

4. Modify parameters θ somehow, hoping to improve $\boldsymbol{L}(\boldsymbol{X}, \boldsymbol{Y}; \boldsymbol{\theta})$
5. Repeat 3 and 4 until you stop improving or run out of time

A common approach to picking the next parameters

HOW?

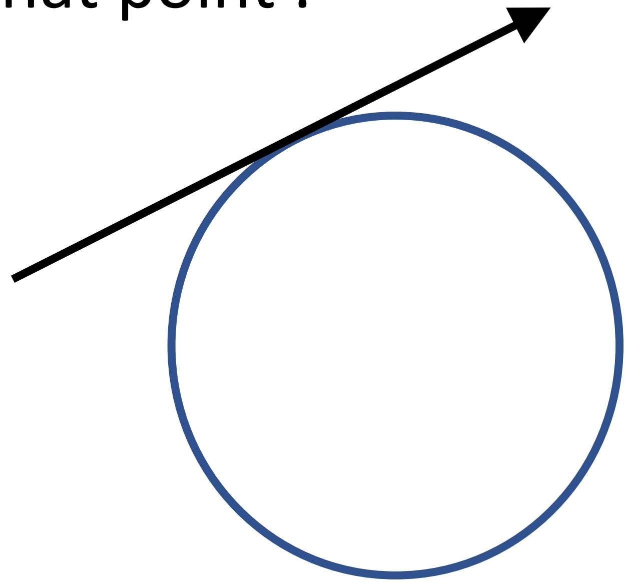
1. Measure how the the loss changes when we change the parameters θ slightly

2. Pick the next set of parameters to be close to the current set, but in the direction that most changes the loss function for the better

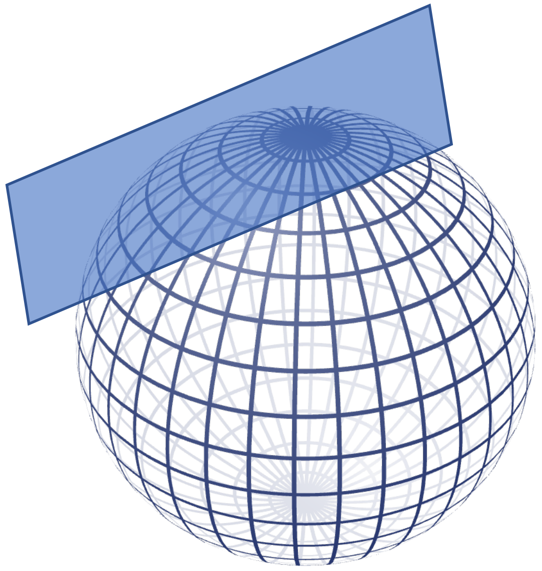
3. Repeat

Slope vs gradient

- Slope of $f(\theta)$ is a scalar describing a line perpendicular to the tangent of the function at that point .

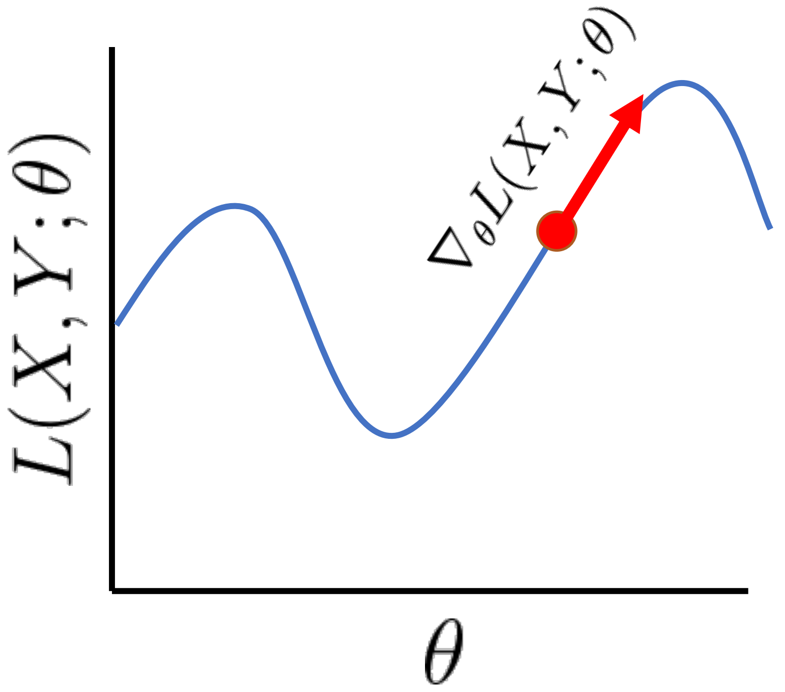


- Gradient $\nabla f(\boldsymbol{\theta})$ is a vector describing a hyperplane perpendicular to the tangent at $\boldsymbol{\theta}$



What does the gradient tell us?

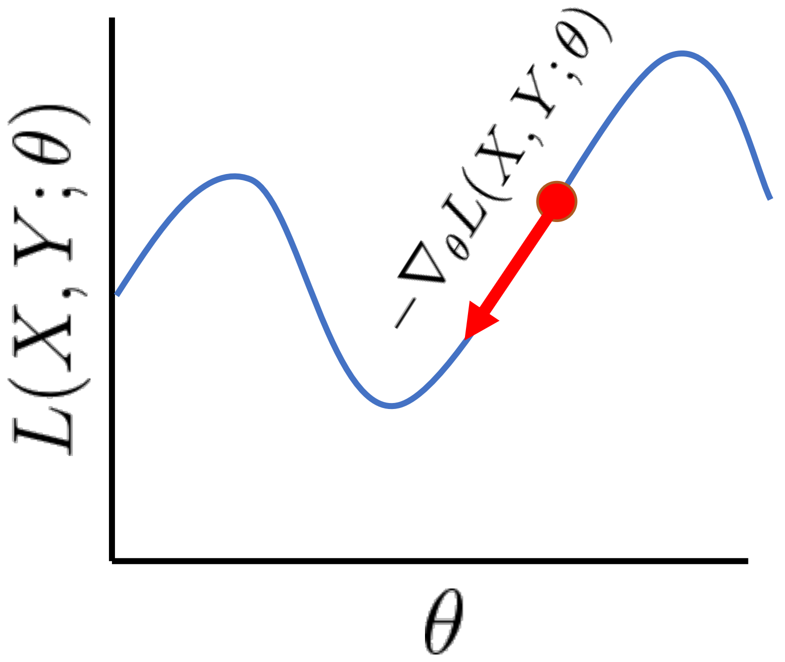
- If the loss function and hypothesis function encoded by the model are differentiable* (i.e., the gradient exists)
- We can evaluate the gradient for some fixed value of our model parameters θ and get the *direction* in which the loss *increases* fastest



*or subdifferentiable

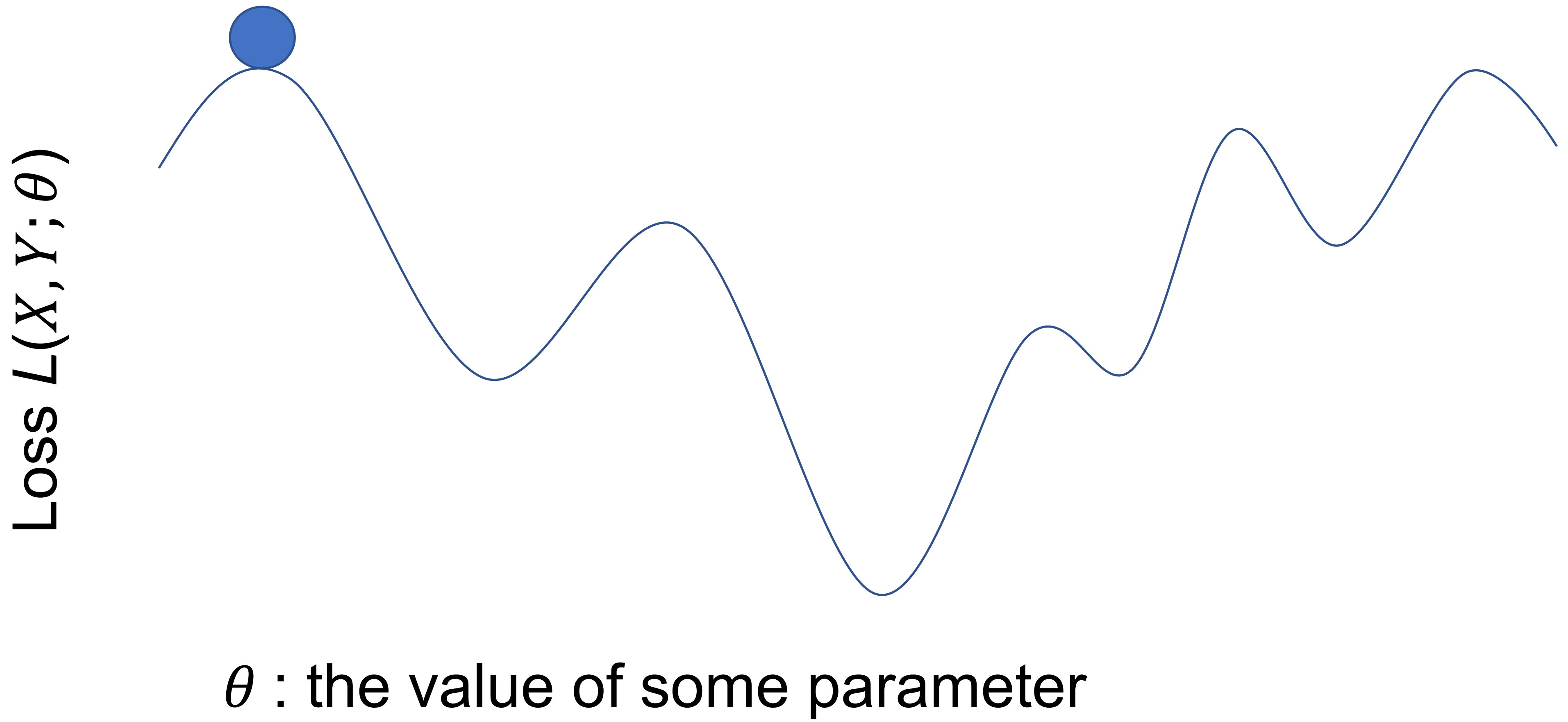
What does the gradient tell us?

- We want to *decrease* our loss, so let’s go the other way instead



Gradient Descent: Promises & Caveats

- Much faster than guessing new parameters randomly
- Finds the global optimum only if the objective function is convex



Gradient Descent Pseudocode

```
Initialize $\theta^{(0)}$
Repeat until stopping condition met:
    $\theta^{(t+1)} = \theta^{(t)} - \eta \nabla L(X, Y; \theta^{(t)})$
Return $\theta^{(t_{max})}$
```

$\theta^{(t)}$ are the parameters of the model at time step t

X, Y are the input data vectors and the output values.

$\nabla L(X, Y; \theta^{(t)})$ is the gradient of the loss function with respect to model parameters $\theta^{(t)}$

η controls the step size

$\theta^{(t_{max})}$ is the set of parameters that did best on the loss function.

Design choices

```
Initialize $\theta^{(0)}$
Repeat until stopping condition met:
    $\theta^{(t+1)} = \theta^{(t)} - \eta \nabla L(X, Y, \theta^{(t)})$
Return $\theta^{(t_{max})}$
```

- Initialization of θ
- Convergence criterion (i.e. when to stop)
- How much data to use (batch size)
- Step size for updating model parameters
- Choosing a loss function

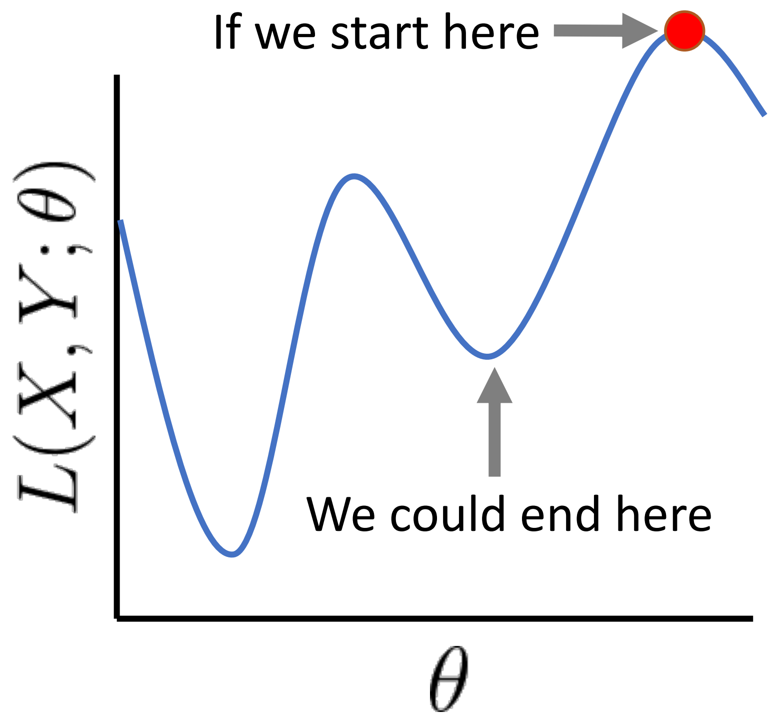
Parameter Initialization

Common initializations:

- $\theta^{(0)} = 0$
- $\theta^{(0)} =$ random values

What happens if our initialization is bad?

- Convergence to a *local* minimum
- No way to determine if you've converged to the global minimum



Convergence criterion: when to stop

- Stop when the gradient is close (within ε) to 0
  (i.e., we reached a minimum)

- Stop after some fixed number of iterations

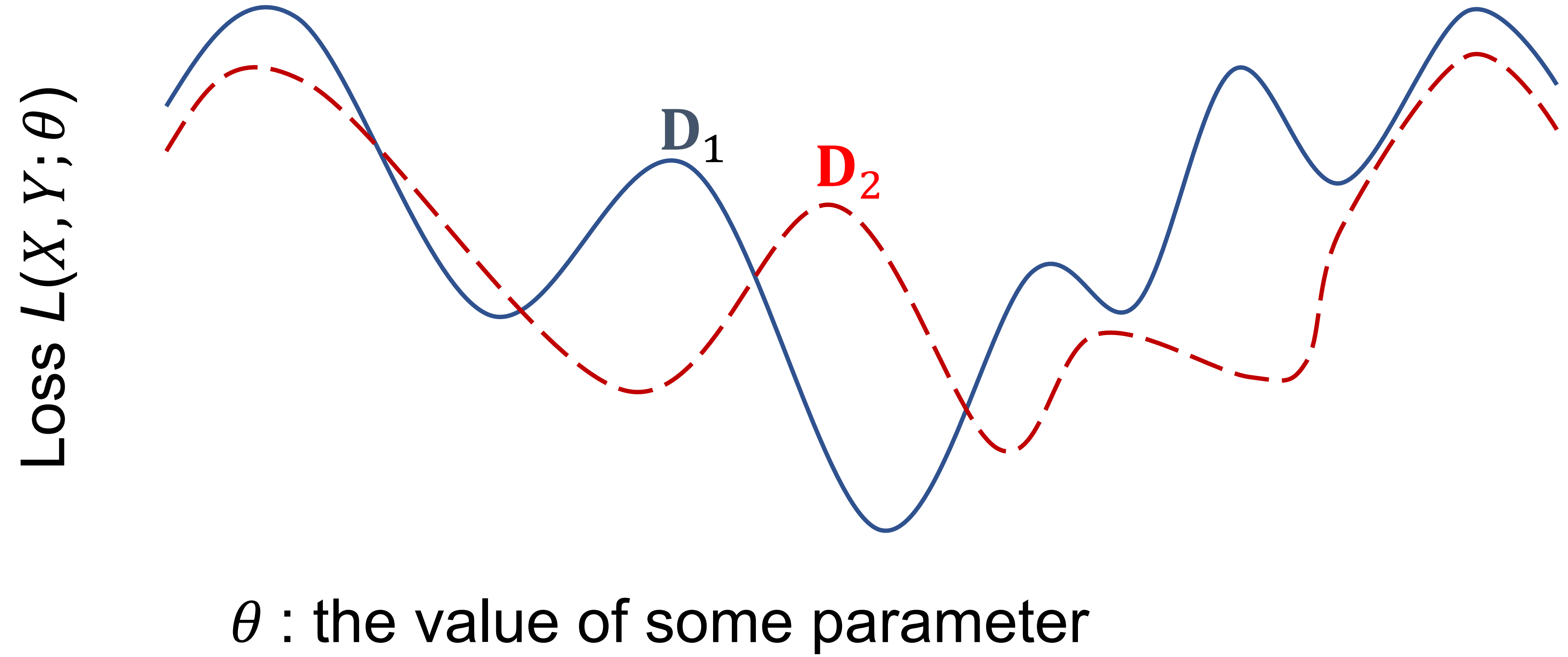
- Stop when the loss on a *validation set* stops decreasing
  (This helps prevent overfitting)

Batch Size: How much data?

- Call D the set of X,Y pairs we measure loss on
- In **batch gradient descent**, the loss is a function of both the parameters θ and the set of all training data **D.**
  (What if if |**D**| > memory?)
- In **stochastic gradient descent**, loss is a function of the parameters and a different single random training sample at each iteration.
- In **mini-batch gradient descent**, random subsets of the data (e.g. 100 examples) are used at each step in the iteration.

Different data, different loss

- Call D the set of X,Y pairs we measure loss on.
- If D changes, then the landscape of the loss function changes
- You typically won't know how it has changed.

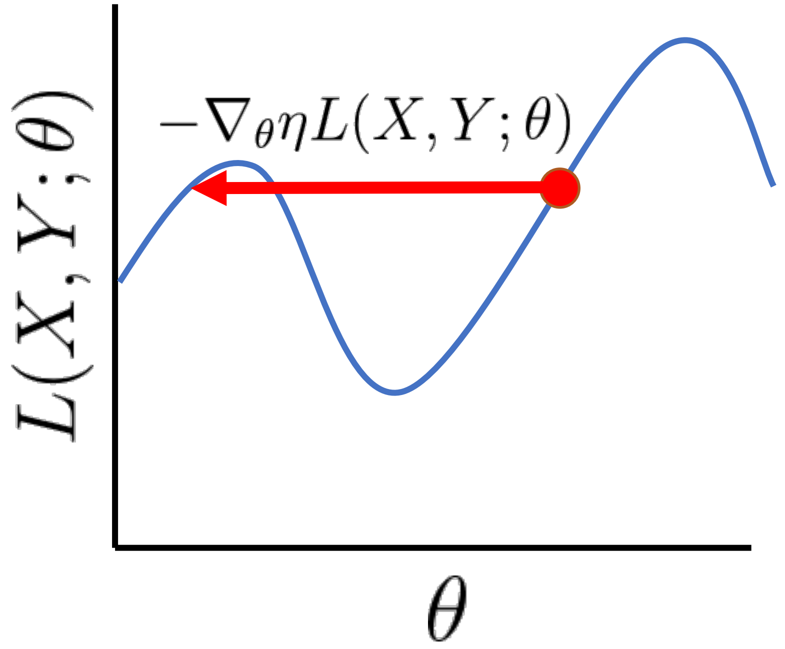


How much data to use in each step?

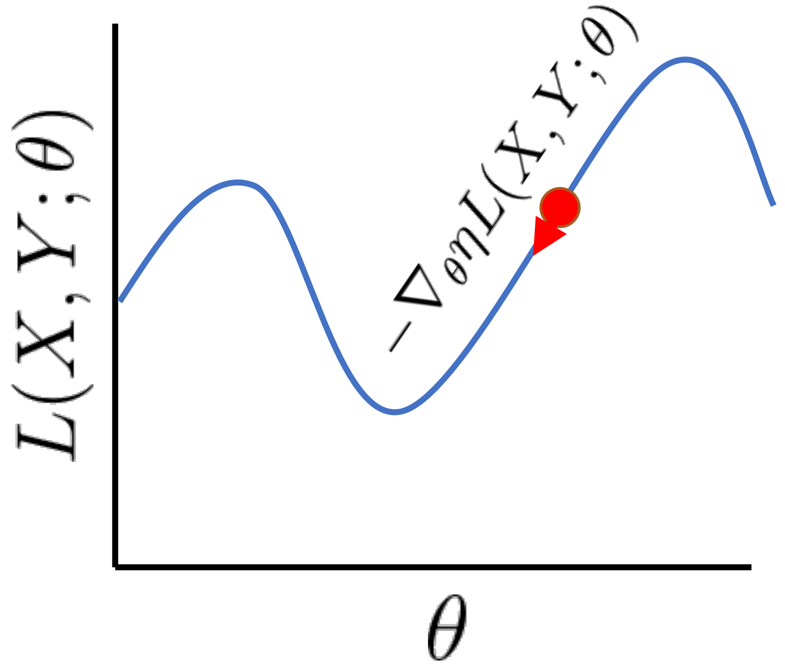
- **All of it (*batch gradient descent*)**
  - The *most accurate* representation of your training loss
  - It can be slow
  - Not possible if data does not fit in RAM
- Just one data point (*stochastic gradient descent*)
  - A *noisy, inaccurate* representation of your training loss
  - *very* fast
  - Random shuffling is important
- More than one data point, but less than all (*mini-batch gradient descent)*
  - Most common approach today
  - Balances *speed* and *accuracy*
  - Random shuffling is important
  - Usually want batch size to be as large as possible for your machine

Step Size: how far should we go?

- The gradient we calculated was based on a fixed value of θ
- As we move away from this point, the gradient changes



If the step size is too large, we may overshoot the minimum



If the step size is too small, we need to take more steps (more computation)

Add Momentum

```
Initialize $\theta^{(0)}, V^{(0)}$
Repeat until stopping condition met:
        $V^{(t+1)} = mV^{(t)} - \eta \nabla L(X, Y, \theta^{(t)})$
        $\theta^{(t+1)} = \theta^{(t)} + V^{(t+1)}$
Return $\theta^{(t_{max})}$
```

There are many variants on gradient descent

- Lots of kinds of momentum/step size selection algorithms (e.g. ADAM)

- Lots of 2nd order algorithms (e.g. BGFS)

- This is an entire field of study.

- Check out classes taught in IEMS on this.

Loss functions

A good objective (loss) function $L(\mathrm{X}, \mathrm{Y}; \theta)$

$$L(\underset{\text{data}}{\mathrm{X}}, \underset{\text{labels}}{\mathrm{Y}}; \underset{\text{parameters}}{\theta})$$

Required

- $L(\mathrm{X}, \mathrm{Y}; \theta) \geq 0$
- $L(\mathrm{X}, \mathrm{Y}; \theta)$ decreases as performance improves

Required for gradient descent

- $L(\mathrm{X}, \mathrm{Y}; \theta)$ is differentiable*, with respect to θ

helpful For gradient descent

- The gradient of L is bounded … $\mathbf{0} < |\nabla L| \ll \infty$

*or subdifferentiable

Notational conventions

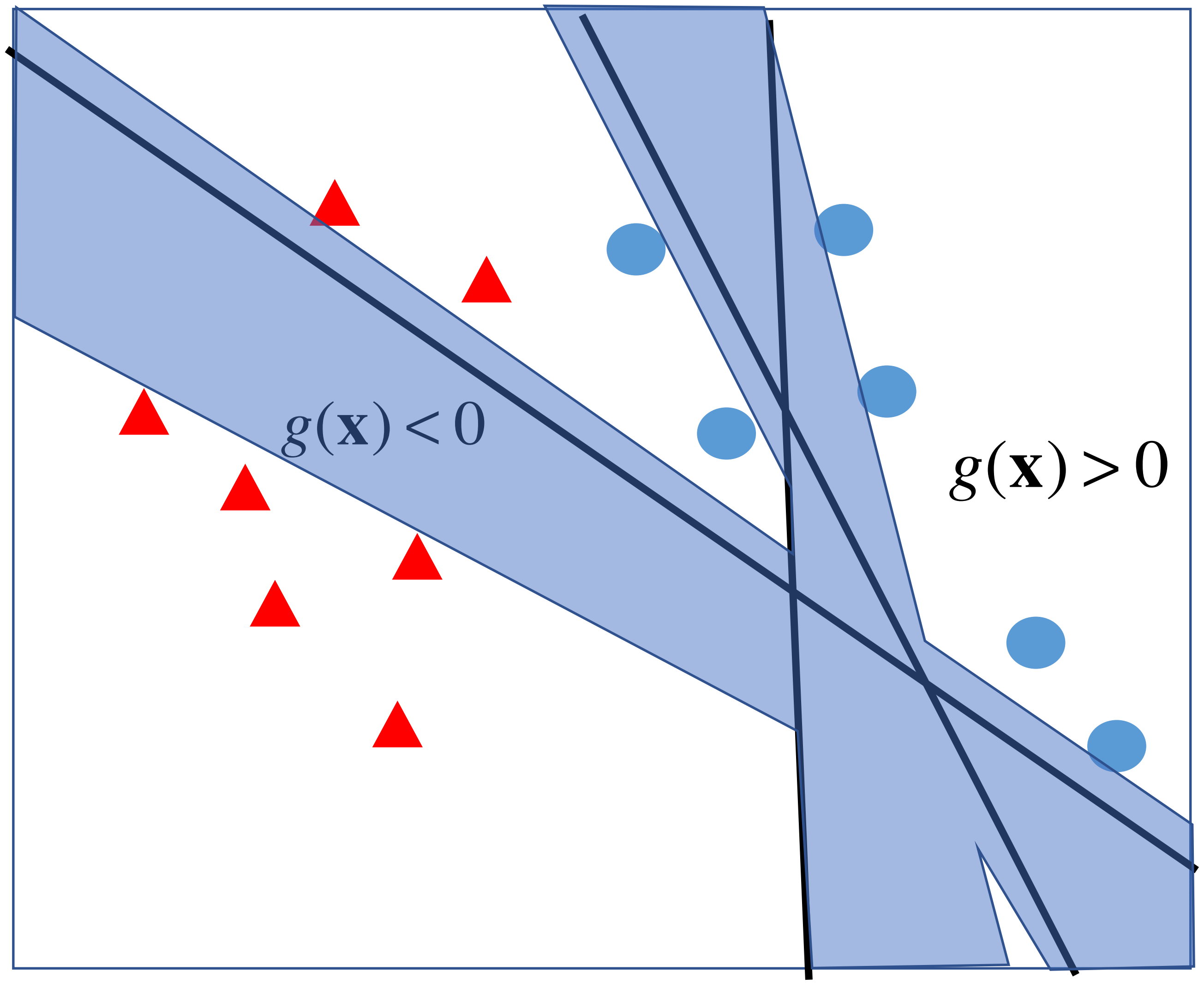
D is the total number of dimensions
d is the current dimension

$\mathbf{w}$ is the *D* dimensional model weight vector (i.e. the model parameters θ)
w_d is the model weight for dimension *d*

$\mathbf{x}$ is one *D* dimensional input example
x_d is the value for $\mathbf{x}$ at dimension *d*
X is a set of examples
$\mathbf{x}_{\boldsymbol{i}}$ is the *i*th example in *X* (note the boldface and use of *i* instead of *d*).

y is one scalar label, drawn from {+1, -1}
Y is a set of labels
y_i is the *i*th example in *Y.*

Example: 0 1 loss



Our linear model

$$g(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 = 0$$

Our hypothesis function

$$h(\mathbf{x}) = \begin{cases} 1 & if\ 0 < \mathbf{w}^T\mathbf{x} \\ -1 & else \end{cases}$$

Our label estimate

$$\hat{y} = h(\mathbf{x})$$

Sum of squared errors loss

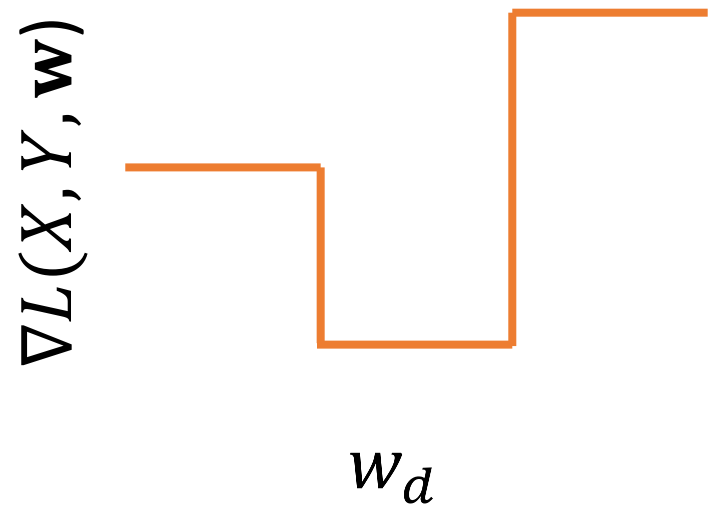
$$L(X, Y, \mathbf{w}) = \frac{1}{2N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

SSE is same everywhere in the blue

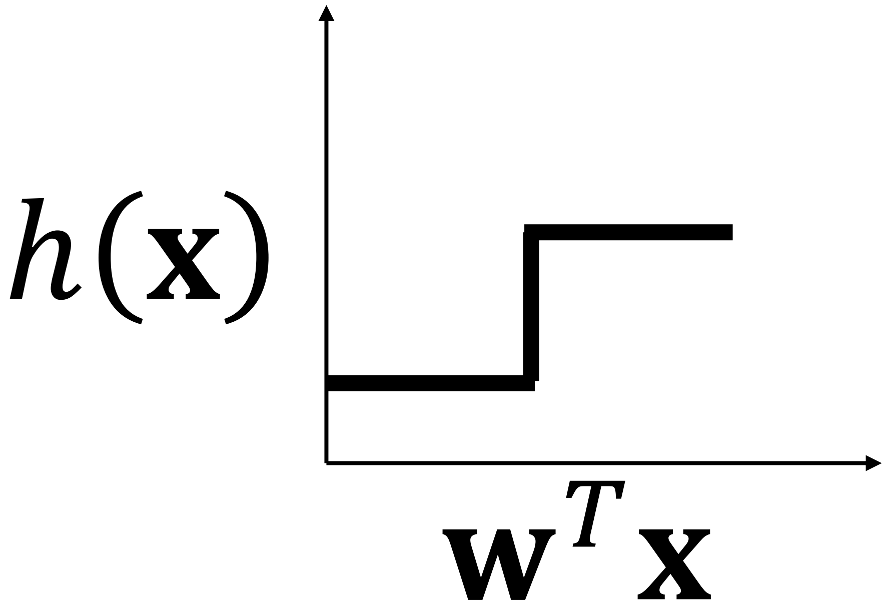
Gradient 0 in the blue region!

The 0 1 Loss function

- Loss $= 1$ if $y \neq h(x)$, else it's 0
- A count of mislabeled items
- Results in a step function
- Not useful for for gradient descent

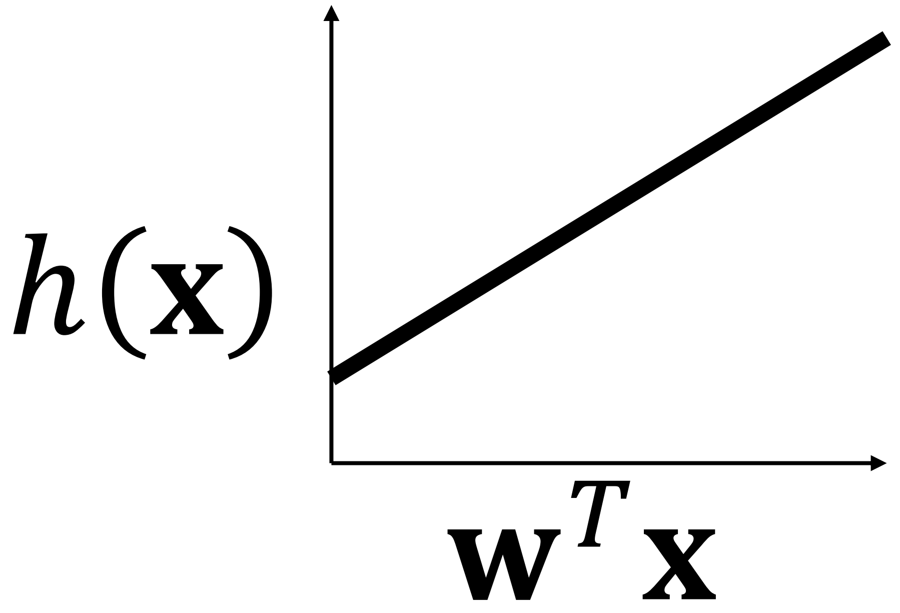


Perceptron Problem: The step function



$$h(x) = \begin{cases} 1 & if\ 0 < \mathbf{w}^T\mathbf{x} \\ -1 & else \end{cases}$$

Solution: Remove the step function



$$h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$$

Squared loss: we now have a gradient

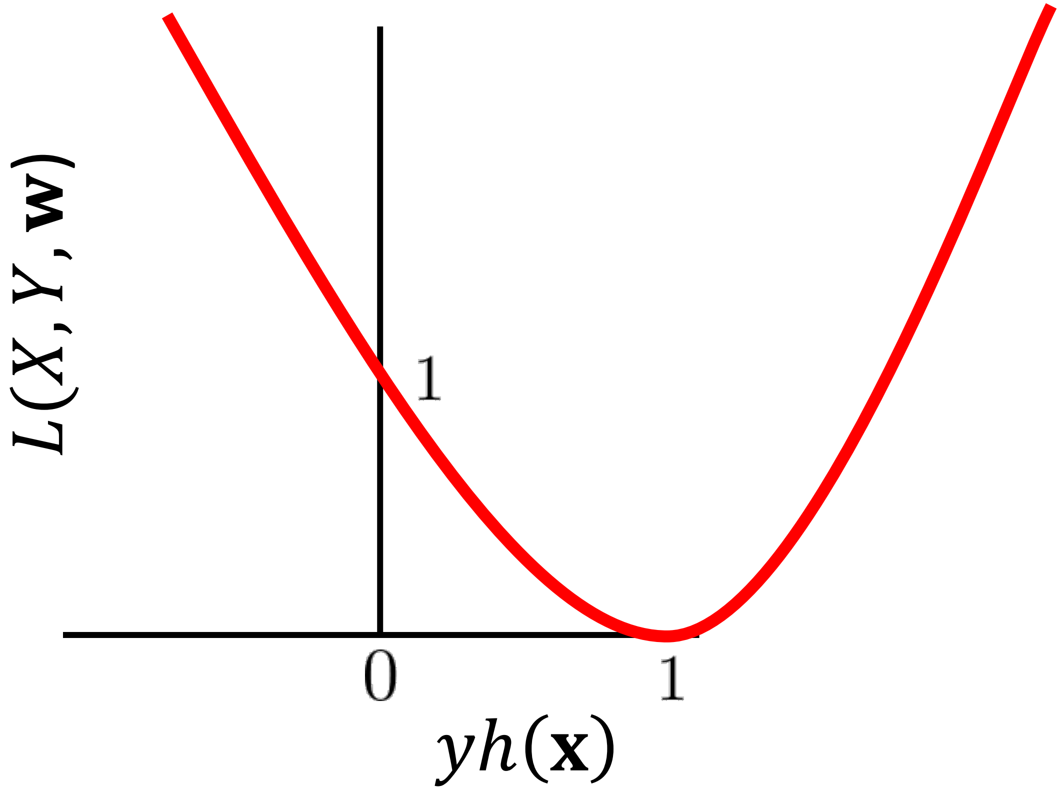
- Our hypothesis function is now $h(\mathbf{x})$ where $\mathbf{w}$ are the model parameters.

- We write our loss function as..

$$L(X, Y, \mathbf{w}) = \frac{1}{2N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^{\mathbf{2}}$$

- If we use a linear model, then..

$$h(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$$



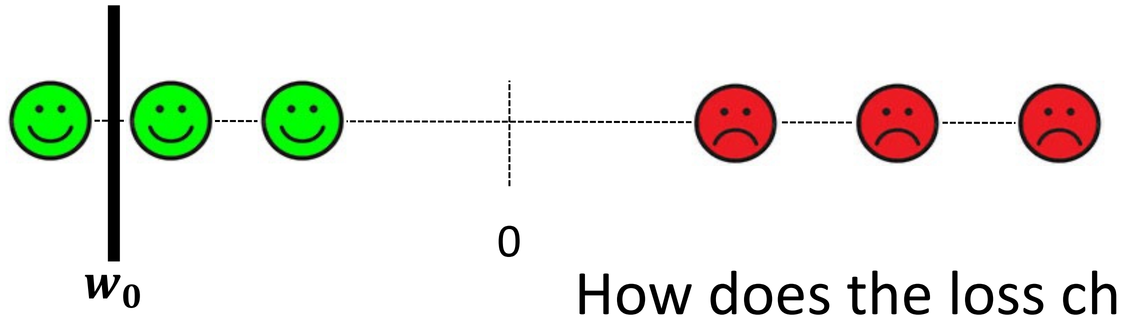
A simple example: where do you draw the line?

Happy faces have label *y* = +1 and sad faces have label y = -1.

We have a linear model with 2 parameters: $\hat{y} = \mathbf{w}^T\mathbf{x} = w_0 x_0 + w_1 x_1$

Our loss function will be sum-of-squared-errors:

$$L(X, Y, \mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^{\mathbf{2}}$$



How does the loss change as we move the line defined by $\boldsymbol{w_0}$?
Can we use that to decide where to move it?
What does $\boldsymbol{w_1}$ do?

Measuring loss for a linear unit

- Model's hypothesis $h(\mathbf{x})$ function outputs a label estimate $\hat{y}$, given its parameters θ. Let's call them the weights, $\mathbf{w}$.

$$\hat{y} = h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$$

- Sum of squared errors loss function:

i is the index to the ith example $\mathbf{x}_i$ and its label $\mathbf{y}_i$

$$L(X, Y, \mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^{\mathbf{2}}$$

This ½ makes the derivative simpler

If we consider a single example, then…

$$L(X, Y, \mathbf{w}) = \frac{1}{2N} \sum_{i=1}^{N} (y_i - \hat{y}_{\boldsymbol{i}})^{\mathbf{2}}$$

Setting the number of data points N = 1 results in…

$$L(\mathbf{x}, y, \mathbf{w}) = \frac{1}{2} (y - \hat{y})^{\mathbf{2}}$$

The example **x** is a *D* dimensional vector
The model weights **w** are also *D* dimensional
Our label *y* is a scalar

For each dimension *d*, take the partial derivative

$\frac{\partial L}{\partial w_d} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_d}$ gives the change of our loss function L with respect to weight w_d

Our loss function is : $L = \frac{1}{2}(y - \hat{y})^{\mathbf{2}}$

$$= \frac{y^2}{2} + \frac{\hat{y}^2}{2} - y\hat{y}$$

therefore… $\frac{\partial L}{\partial \hat{y}} = \hat{y} - y$

For each dimension *d*, take the partial derivative

$\frac{\partial L}{\partial w_d} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_d}$ gives the change of our loss function L with respect to weight w_d

From the previous slide.... $\frac{\partial L}{\partial \hat{y}} = \hat{y} - y$

Our estimator is a linear unit : $\hat{y} = \mathbf{w}^T \mathbf{x}$

therefore… $\frac{\partial L}{\partial \hat{y}} = \mathbf{w}^T \mathbf{x} - y$

Let’s calculate $\frac{\partial \hat{y}}{\partial w_d}$

D is the total number of dimensions
d is the current dimension
$\mathbf{w}$ is the *D* dimensional model weight vector
$\mathbf{x}$ is the *D* dimensional input example
w_d is the model weight for dimension *d*
x_d is the value for $\mathbf{x}$ at dimension *d*

Our estimator is : $\hat{y} = \mathbf{w}^T\mathbf{x} = w_0 x_0 + \ldots\, w_d x_d + \ldots\, w_D x_D$

Now… w_d is the only parameter we’re varying right now.

So all w_j where $j \neq d$ are constant in this partial derivative.

Therefore, $\frac{\partial \hat{y}}{\partial w_d} = x_d$

The gradient for weight *d* **is…**

$$\frac{\partial L}{\partial w_d} = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w_d} = (\mathbf{w}^T\mathbf{x} - y)x_d$$
$$= -(y - \mathbf{w}^T\mathbf{x})x_d$$

So the gradient of the loss for all *D* weights is…

$$\nabla L(\mathbf{x}, y, \mathbf{w}) = \left[\frac{\partial L}{\partial w_0}, \ldots \frac{\partial L}{\partial w_d}, \ldots \frac{\partial L}{\partial w_D}\right]$$

$$= -(y - \mathbf{w}^T\mathbf{x})\mathbf{x}$$

We can now estimate the gradient for a whole set

$$\nabla L(X, Y, \mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} \nabla L(\mathbf{x}_i, y_i, \mathbf{w})$$

X and *Y* are the set of examples and labels.
N is the number of examples.
$\mathbf{x}_i, \mathbf{y}_i$ are a single pair of example and label.

The gradient can now be used here

```
Initialize $\theta^{(0)}$
Repeat until stopping condition met:
    $\theta^{(t+1)} = \theta_t - \eta \nabla L(X, Y; \theta^{(t)})$
Return $\theta^{(t_{max})}$
```

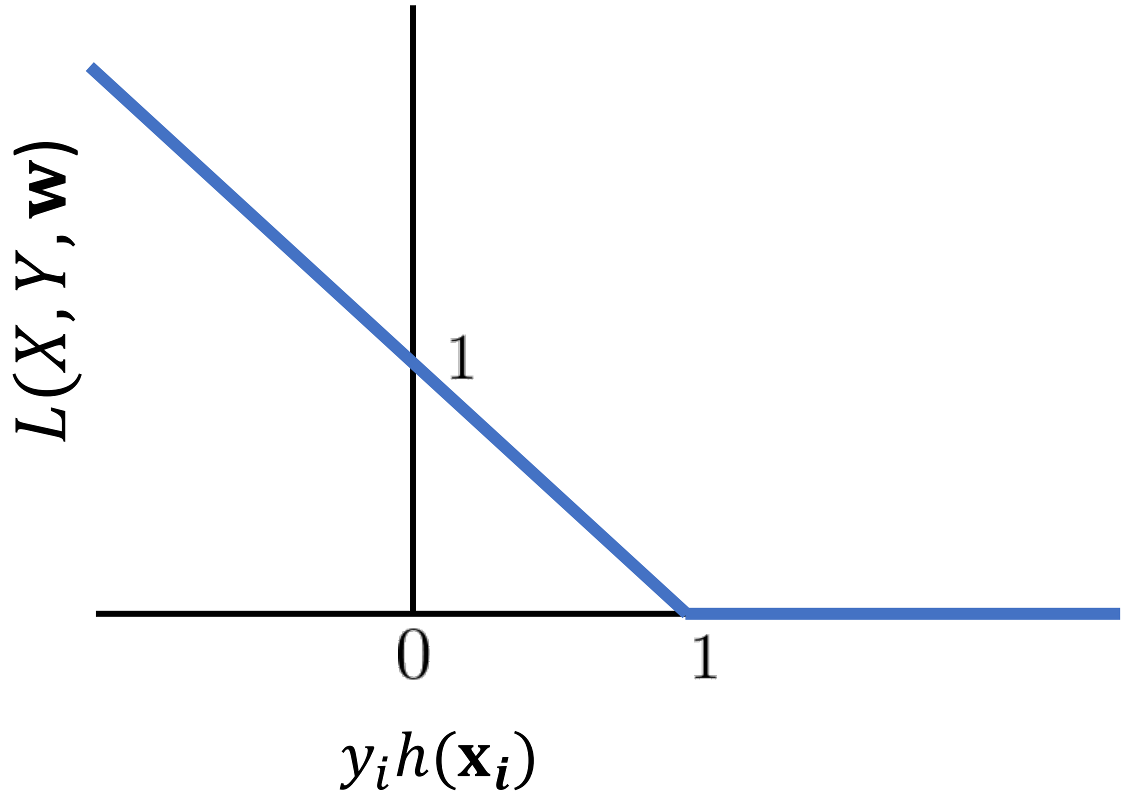
$\theta^{(t)}$ are the parameters of the model at time step *t.*

($NOTE$: $\theta^{(t)}$ corresponds to the model weights **w** from the prev. slide)

Hinge Loss

$$L(X, Y, \mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} max(0, 1 - y_i h(\mathbf{x}_{\boldsymbol{i}}))$$

- Loss only >0 if the data is within 1 of the wrong side of the line.



Hinge Loss Gradient

The loss function is: $L(X, Y, \mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} max(0, 1 - y_i h(\mathbf{x_i}))$

Let's calculate its gradient with a linear model: $h_\theta(x) = \mathbf{w}^T\mathbf{x}$

As we did for squared error, let's consider a single example:

$$L(\mathbf{x}, y, \mathbf{w}) = \max(0, 1 - y\mathbf{w}^T\mathbf{x})$$

Put another way.....

$$L(\mathbf{x}, y, \mathbf{w}) = \begin{cases} 0 & \text{if } y\mathbf{w}^T\mathbf{x} > 1 \\ 1 - y\mathbf{w}^T\mathbf{x} & \text{else} \end{cases}$$

Hinge Loss Gradient

From the previous slide: $L(\mathbf{x}, y, \mathbf{w}) = \begin{cases} 0 & \text{if } y\mathbf{w}^T\mathbf{x} > 1 \\ 1 - y\mathbf{w}^T\mathbf{x} & \text{else} \end{cases}$

The top case is where the model is returning a value that is the right sign (remember our labels are either +1 or -1) and far enough from the decision boundary that we don't need to move the line.

In this case, where $(y\mathbf{w}^T\mathbf{x}) > 1$, our loss function is $L(\mathbf{x}, y, \mathbf{w})$=0. That's easy, the gradient must also be 0 in this case. So we just need to consider the other case.

Hinge Loss Gradient

From the previous slide: $L(\mathbf{x}, y, \mathbf{w}) = \begin{cases} 0 & \text{if } \; y\mathbf{w}^T\mathbf{x} > 1 \\ 1 - y\mathbf{w}^T\mathbf{x} & \text{else} \end{cases}$

The bottom case is where we will need to get the gradient. First, let's put that case into a form where doing the math will be easy and obvious.

$$\begin{aligned} L(\mathbf{x}, y, \mathbf{w}) &= 1 - y\mathbf{w}^T\mathbf{x} \\ &= 1 - y(w_0x_0 + \ldots\, w_dx_d + \ldots\, w_Dx_D) \\ &= 1 - yw_0x_0 \ldots - yw_dx_d \ldots - yw_Dx_D \end{aligned}$$

Hinge Loss Gradient

From prev. slide: $L(\mathbf{x}, y, \mathbf{w}) = 1 - yw_0x_0 \ldots - yw_dx_d \ldots - yw_Dx_D$

Since w_d is the only parameter we're varying right now, all w_j where $j \neq d$ are constant in this partial derivative. Therefore…

$$\frac{\partial L}{\partial w_d} = -yx_d$$

The label is the same, regardless of the dimension we're considering. Thus we can get the full gradient by multiplying *y* by the full vector **x**

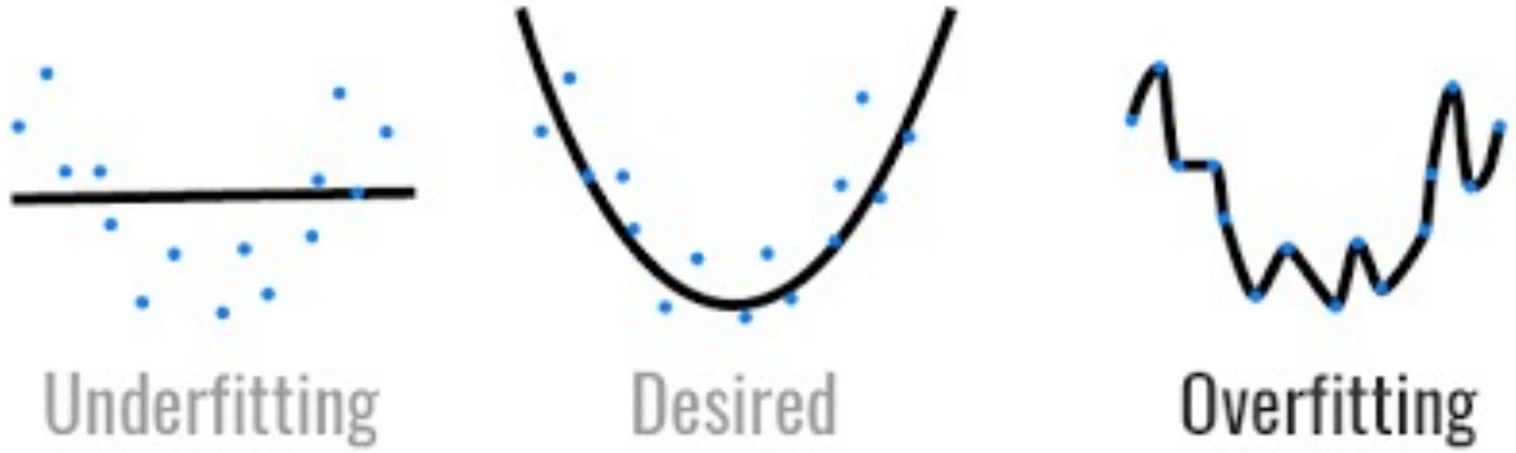
$$\nabla L(\mathbf{x}, y, \mathbf{w}) = -y\mathbf{x}$$

Regularization

(note….the slides are back to using θ for the model parameters)

Revisiting Overfitting

- Overfitting occurs when your model begins to “memorize” the training data
  - Can detect overfitting from an increasing gap between training and validation loss.
  - Performance on the training set improves, but performance on the validation set does not.



Revisiting Overfitting: Regularization

- Big idea (**Occam's Razor**) – Given two models with equal performance, prefer the *simpler* model.
  - E.g., models with fewer parameters or smaller coefficients

- Regularization can be applied to any loss function

$$L_R(X, Y; \theta) = L(X, Y; \theta) + \lambda R(\theta)$$

- The amount of regularization is controlled by the hyperparameter λ

L1- and L2-regularization

- Recall the l_p-norm:

$$\ell_p(\theta) = \sqrt[p]{\sum_{i=1}^{d} |\theta_i|^p}$$

- l_1-regularization penalizes high values of the l_1-norm of the model parameters:

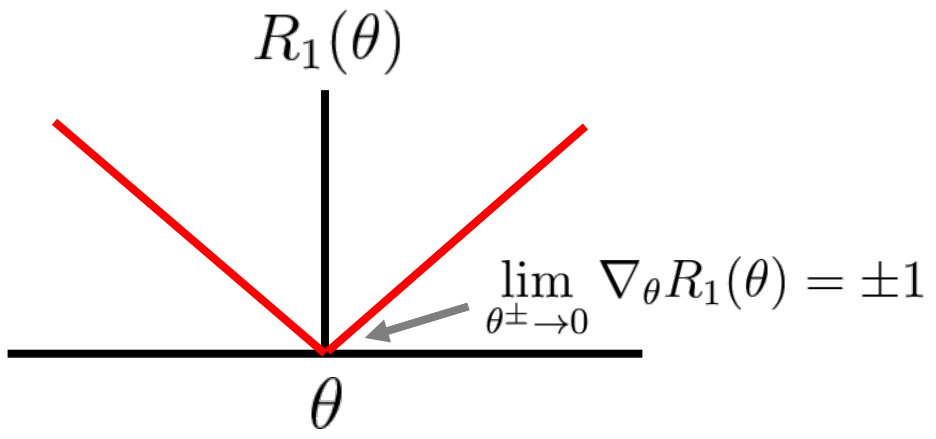
$$R_1(\theta) = \sum_{i=1}^{d} |\theta_i|$$

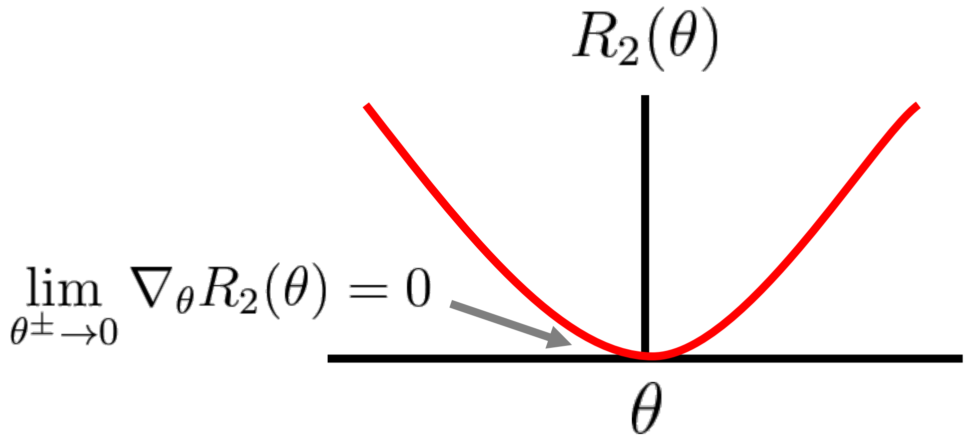
- l_2-regularization penalizes high values of the l_2-norm:

$$R_2(\theta) = \frac{1}{2}\sum_{i=1}^{d} |\theta_i|^2$$

L1-regularization and sparsity

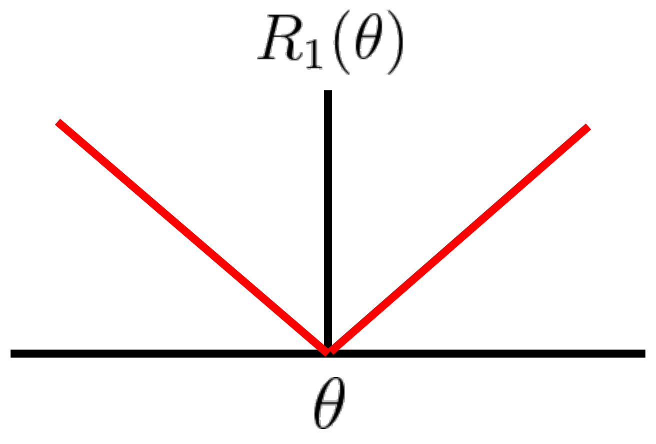
- L1-regularization encourages the model parameters to be *sparse*
  - This is a form of feature selection
  - Only features with non-zero coefficients contribute to the model's prediction
- This is because the gradient of L1-regularization moves model parameters towards 0 at a *constant* rate





L1-regularization and sparsity

- The gradient of the L1-regularizer is bounded (between -1 and +1, inclusive) but not unique at $\theta = 0$.
- Arbitrarily set the gradient at this point to 0.
- The resulting function is the *sign* function



$$\nabla_\theta R_1(\theta) = \mathrm{sign}(\theta) = \begin{cases} +1, & \theta > 0 \\ 0, & \theta = 0 \\ -1, & \theta < 0 \end{cases}$$

Regularization and offset (aka bias)

- Many ML models include a bias term, b.
- Example: A linear model: $h_\theta(x) = \theta^T x + b$
- Or equivalently, by augmenting θ and x, like we did with perceptrons…

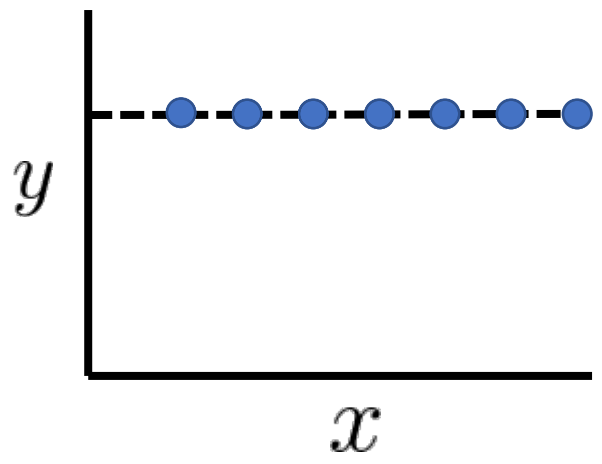
$$\theta' = [\theta_1, \theta_2, \ldots, \theta_d, b],\ x' = [x_1, x_2, \ldots, x_d, 1]$$

- What happens if we regularize the bias term?

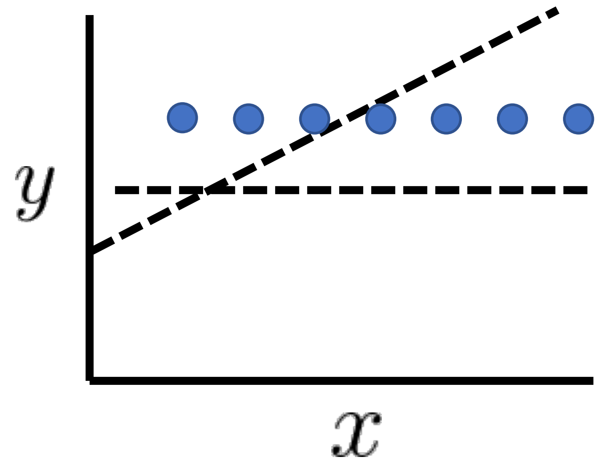
Regularization and offset (aka bias)

- Recall that "regularizing" a model parameter means encouraging that model parameter to tend towards 0.
- How would a linear model represent horizontal line?
- How does shrinking the bias affect its ability to do so?

No bias regularization



Bias regularization



Don't regularize the bias term!

A familiar example

- A lot of algorithms can be constructed by simply combining a loss function and a regularizer

- For example, hinge loss and L2-regularization with a linear hypothesis

$$L_H(S, Y; \theta) + \lambda R_2(\theta) = \frac{1}{N} \sum_{i=1}^{N} \max(0, 1 - y_i \theta^T x_i) + \frac{\lambda}{2} \|\theta\|_2^2$$

- Does this look like something we've studied?

Soft margin SVMs!

Learning a maximum margin separator via gradient descent

- Finding an exact solution for SVMs can be difficult
    - Convex quadratic programming problem
    - $\sim O(n^3)$
- Finding a good approximate solution for SVMs using gradient descent is much easier and computationally faster

Multi-class Classification

Using your linear model, once learned

- Regression:

$$\hat{y} = h_\theta(x)$$

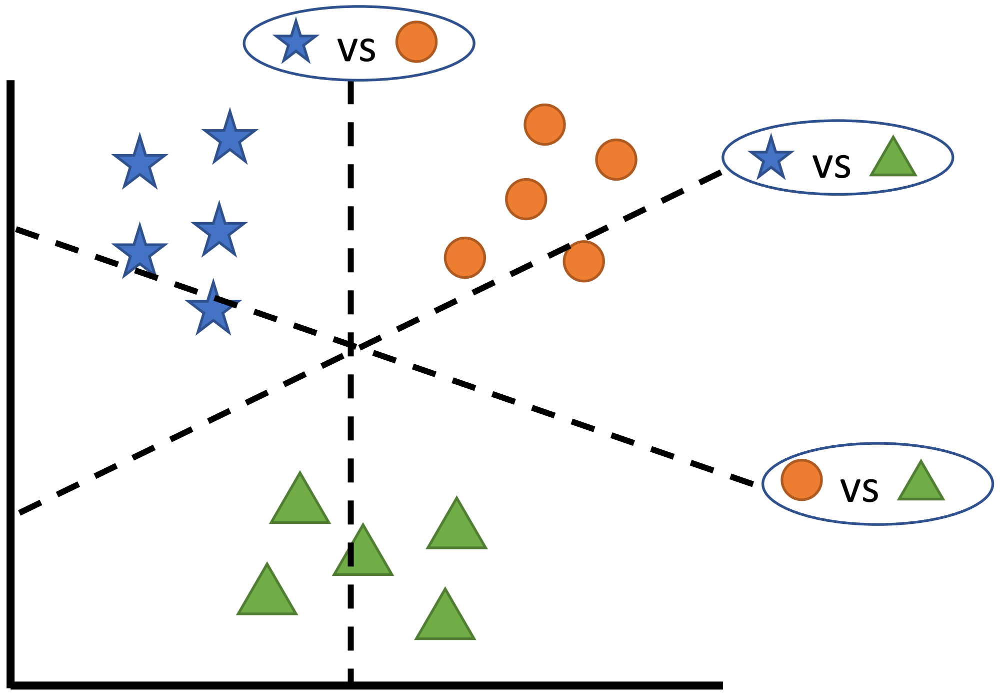
- Binary classification:

$$\hat{y} = \begin{cases} 1, & \text{if } h_\theta(x) > 0 \\ -1, & \text{otherwise} \end{cases}$$

- What about classification with more than 2 classes (e.g., *C* classes)? We will discuss 2 approaches:
    - One-vs-One classification
    - One-vs-All classification

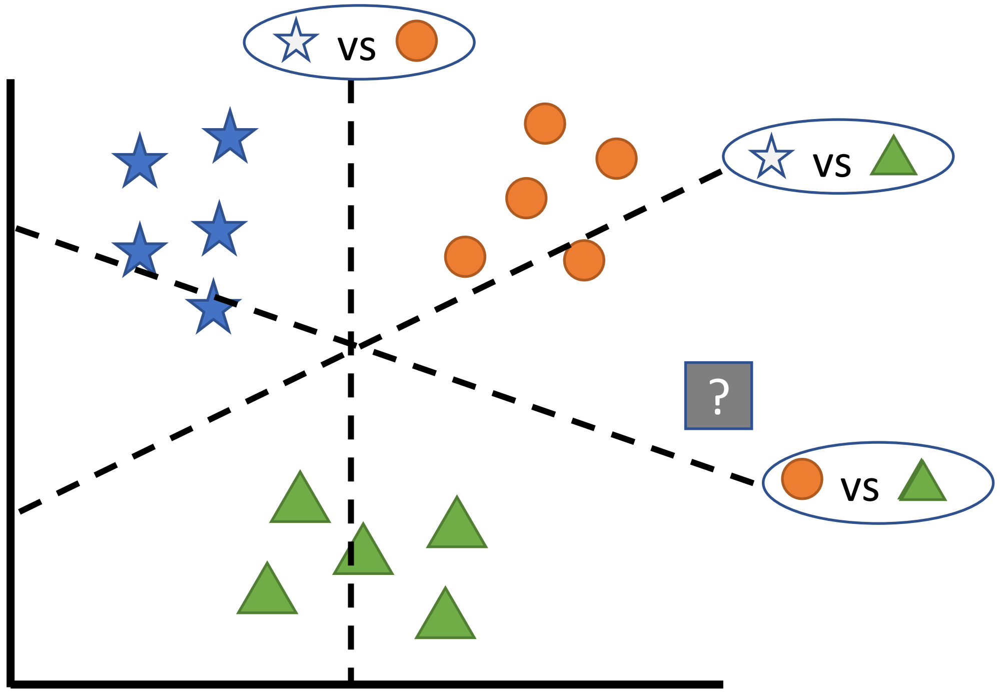
One-vs-One Classification

- Train a binary classifier to disambiguate between each *pair* of classes
- Final prediction is the majority vote among all binary classifiers



One-vs-One Classification

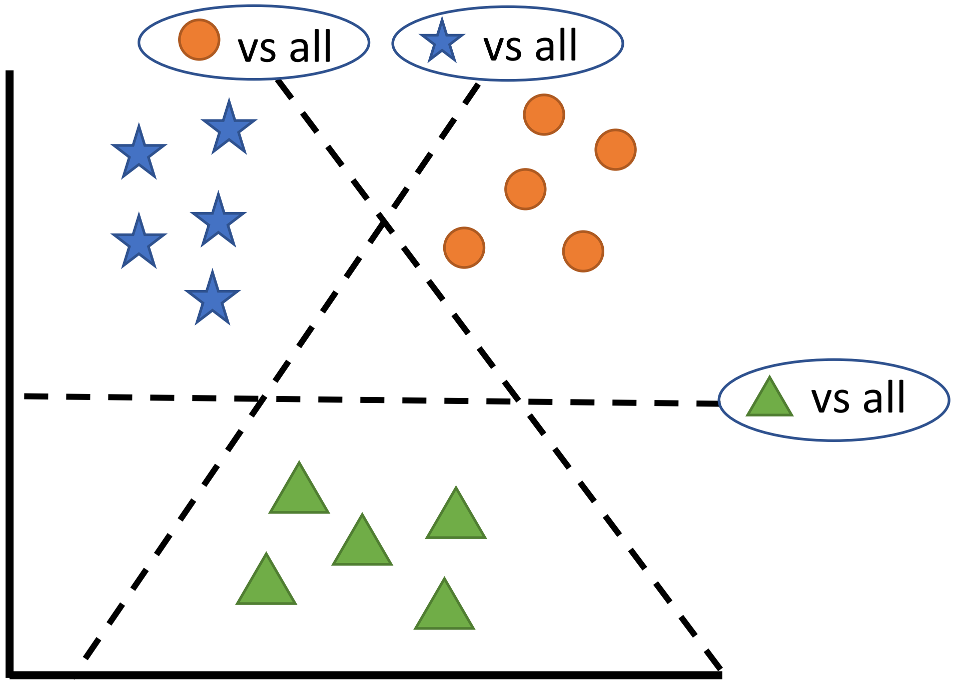
- Train a binary classifier to disambiguate between each *pair* of classes
- Final prediction is the majority vote among all binary classifiers



2 votes for ●
1 vote for ▲
-> classify as ●

One-vs-All Classification

- Train a binary classifier on whether an example does or does not belong to a class
- Predict based on the *highest confidence score* (i.e., the regression output)



$$\hat{y} = \underset{c \in C}{\operatorname{argmax}}\, h_{\theta_c}(x)$$