Lec.5. The need for optimization

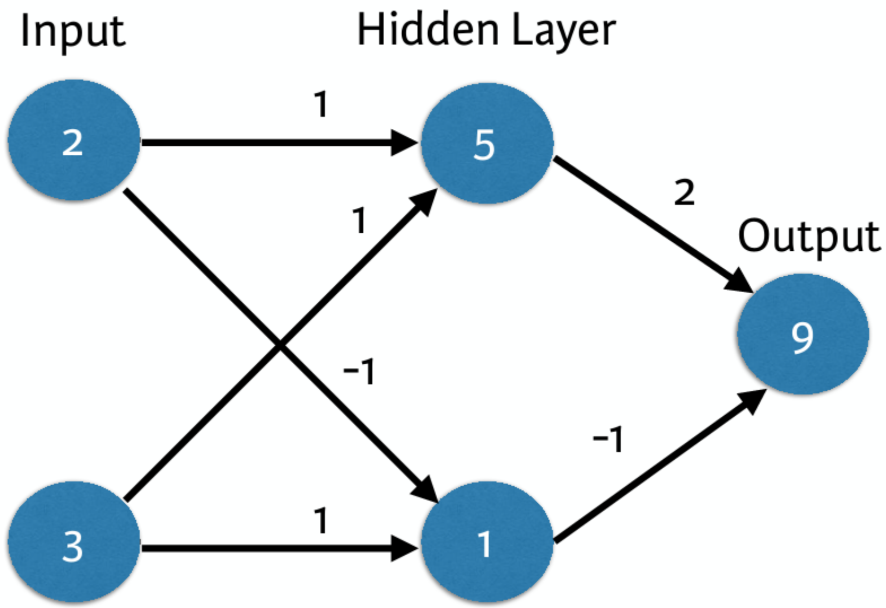
Machine Learning II

Aidos Sarsembayev, IITU, Almaty, 2019

Outline

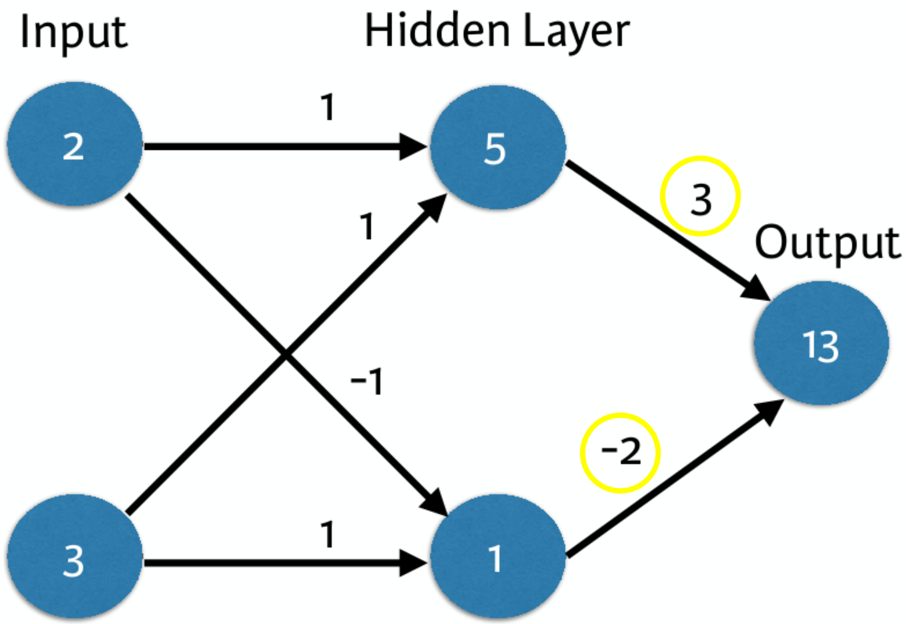
1. Introduction to Deep Learning

A baseline neural network



- NB: here we assume that we use identity activation function
- Actual Value of Target: 13
- Error: Predicted - Actual = -4

A baseline neural network



- NB: here we assume that we use identity activation function
- Actual Value of Target: 13
- Error: Predicted - Actual = 0

Predictions with multiple points

- Making accurate predictions gets harder with more points
- At any set of weights, there are many values of the error
- … corresponding to the many points we make predictions for

Loss function

- Aggregates errors in predictions from many data points into single number
- Measure of model's predictive performance

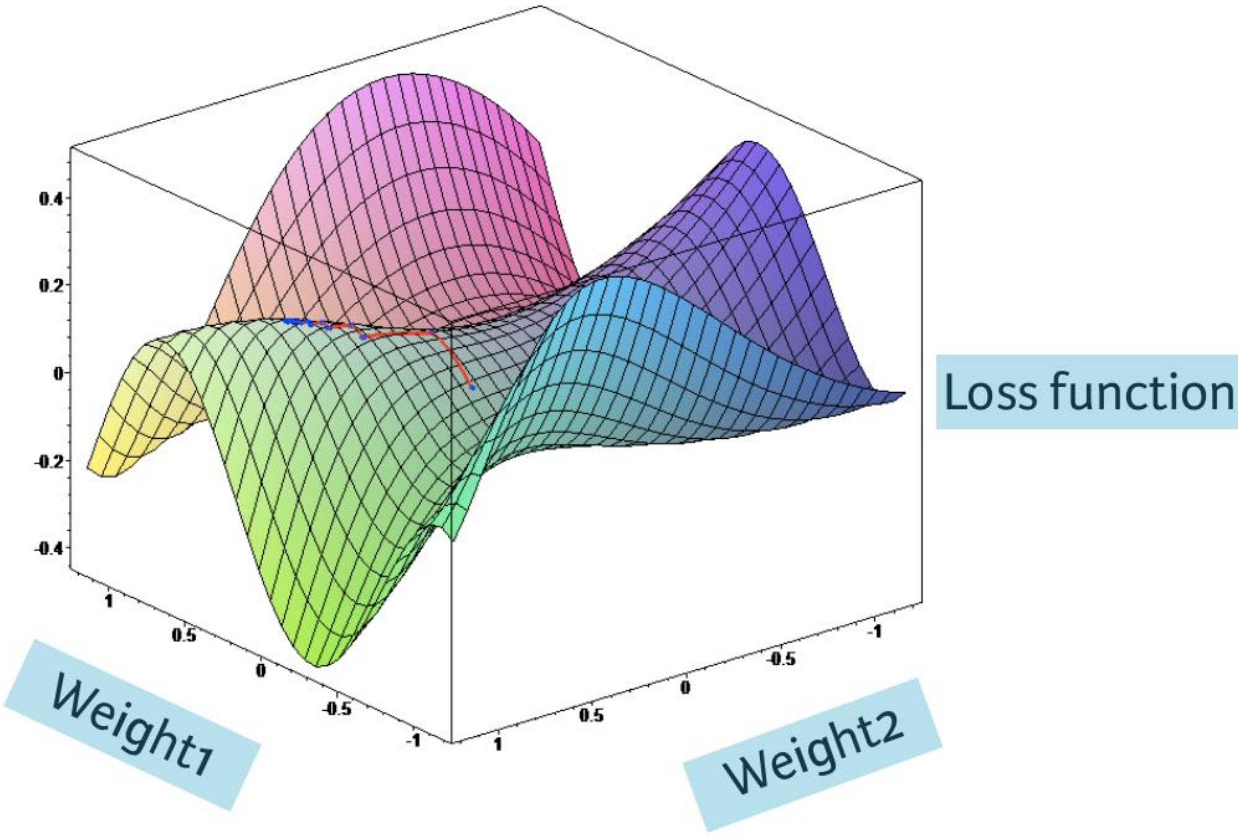
Squared error loss function

Actual	Predicted	Error	RMSE
10	20	-10	100
3	8	-5	25
6	1	5	25

- Total Squared Error: 150
- Mean Squared Error: 50

Loss function

- For instance an example of LF for two weights

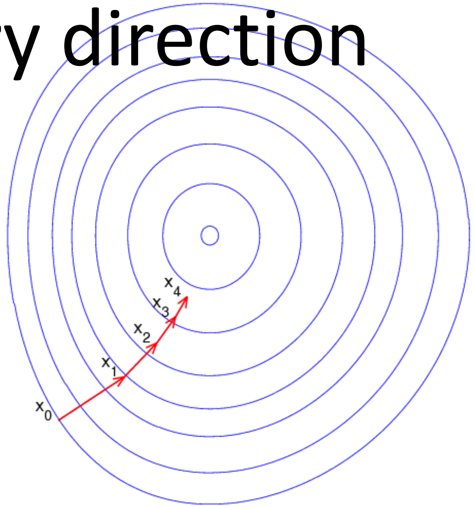


Loss function

- Lower loss function value means a better model
- Goal: Find the weights that give the lowest value for the loss function
- Gradient descent

Gradient descent

- Imagine you are in a pitch dark field
- Want to find the lowest point
- Feel the ground to see how it slopes
- Take a small step downhill
- Repeat until it is uphill in every direction



Gradient descent steps

- Start at random point
- Until you are somewhere flat:
  - Find the slope
  - Take a step downhill

Gradient descent explained

The equation below describes what Gradient Descent does:

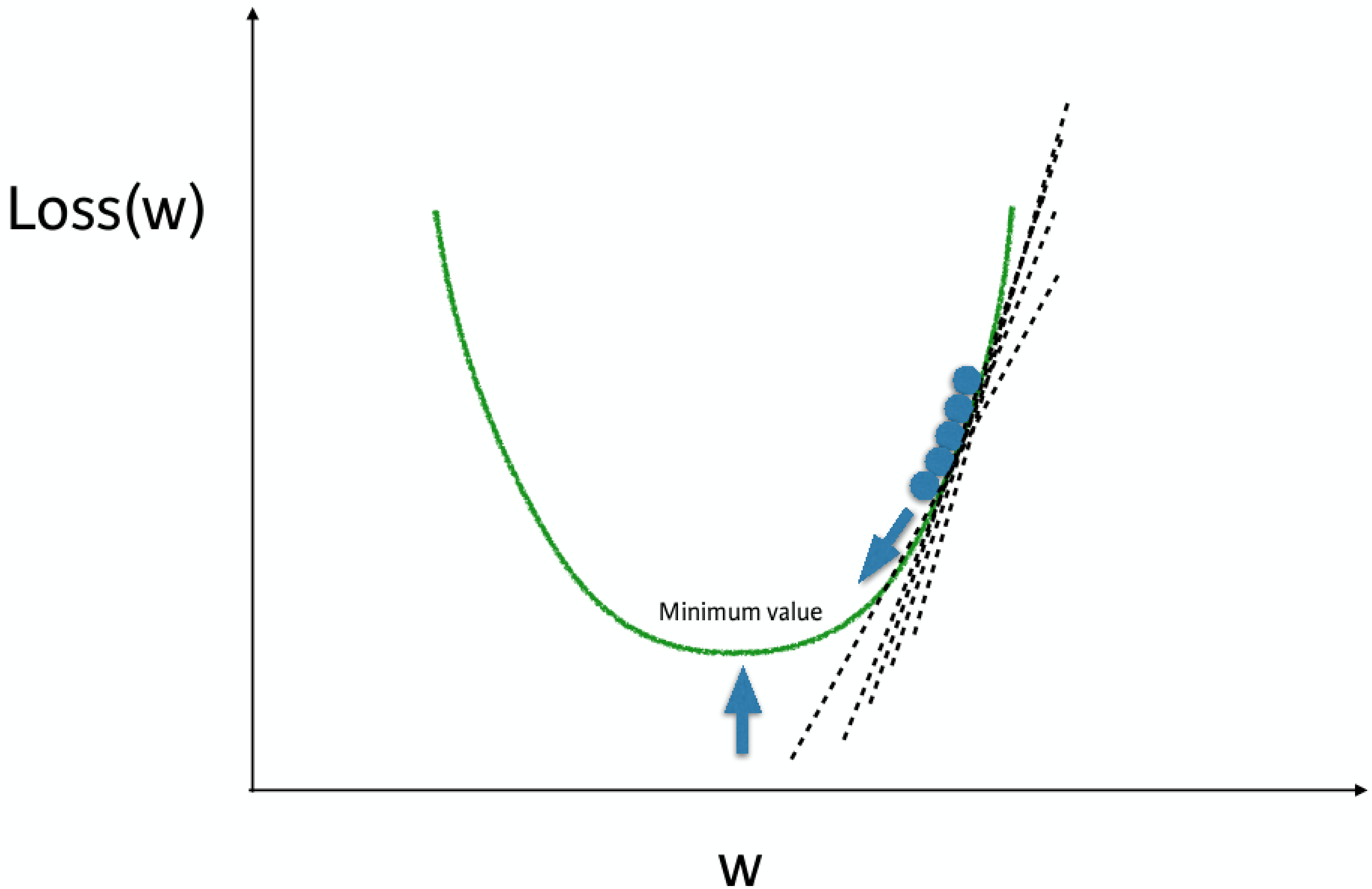
b describes the next position of our climber, while ***a*** represents his current position. The minus sign refers to the minimization part of gradient descent.

The γ in the middle is a waiting factor and the gradient term $\nabla f(a)$ is simply the direction of the steepest descent.

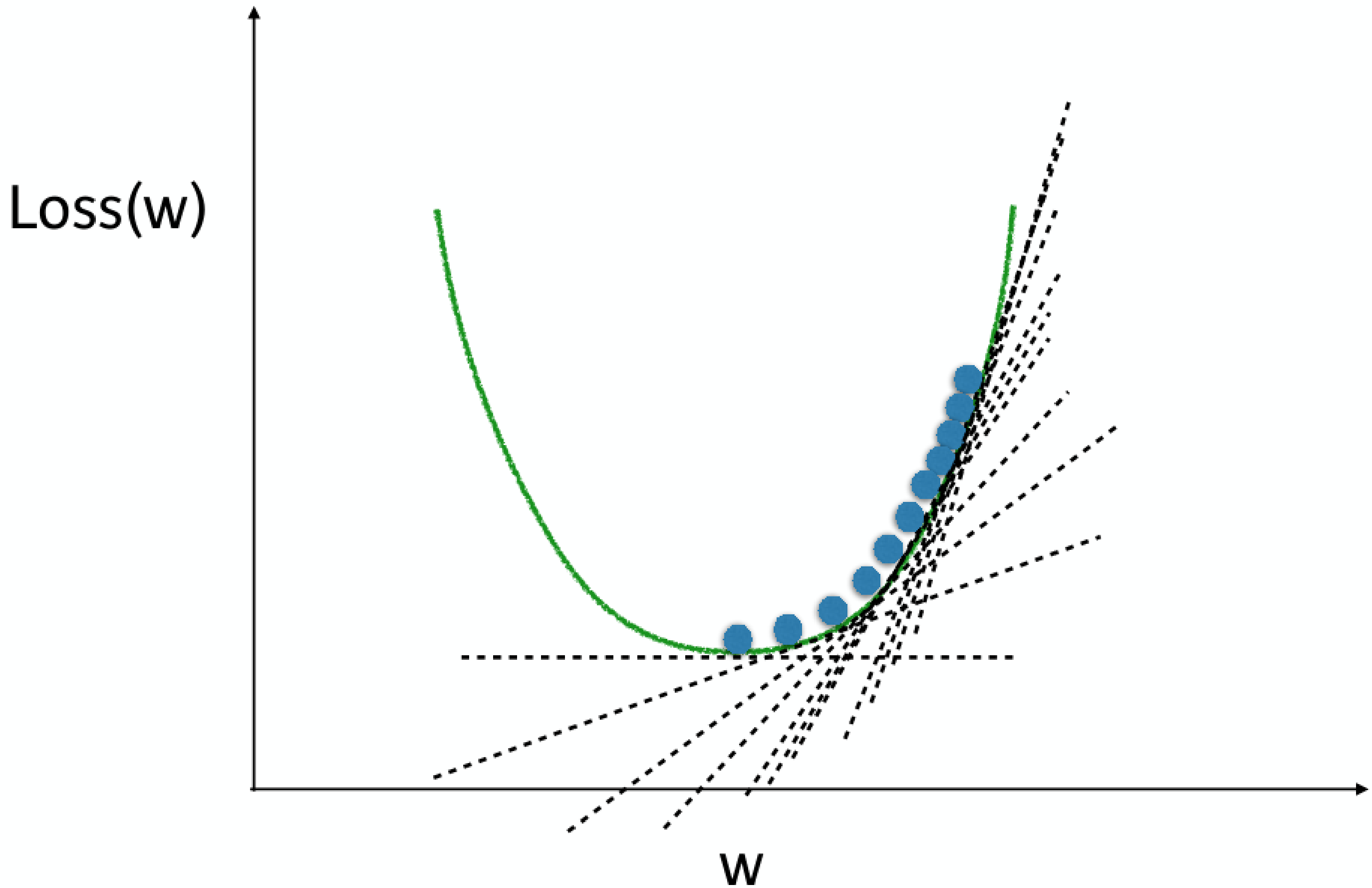
$$b = a - \gamma \nabla f(a)$$

Del, or nabla, is an operator used in mathematics, in particular in vector calculus, as a vector differential operator, usually represented by the nabla symbol ∇. When applied to a function defined on a one-dimensional domain, it denotes its standard derivative as defined in calculus. When applied to a field (a function defined on a multi-dimensional domain), it may denote the gradient (locally steepest slope) of a scalar field (or sometimes of a vector field, as in the Navier–Stokes equations), the divergence of a vector field, or the curl (rotation) of a vector field, depending on the way it is applied.

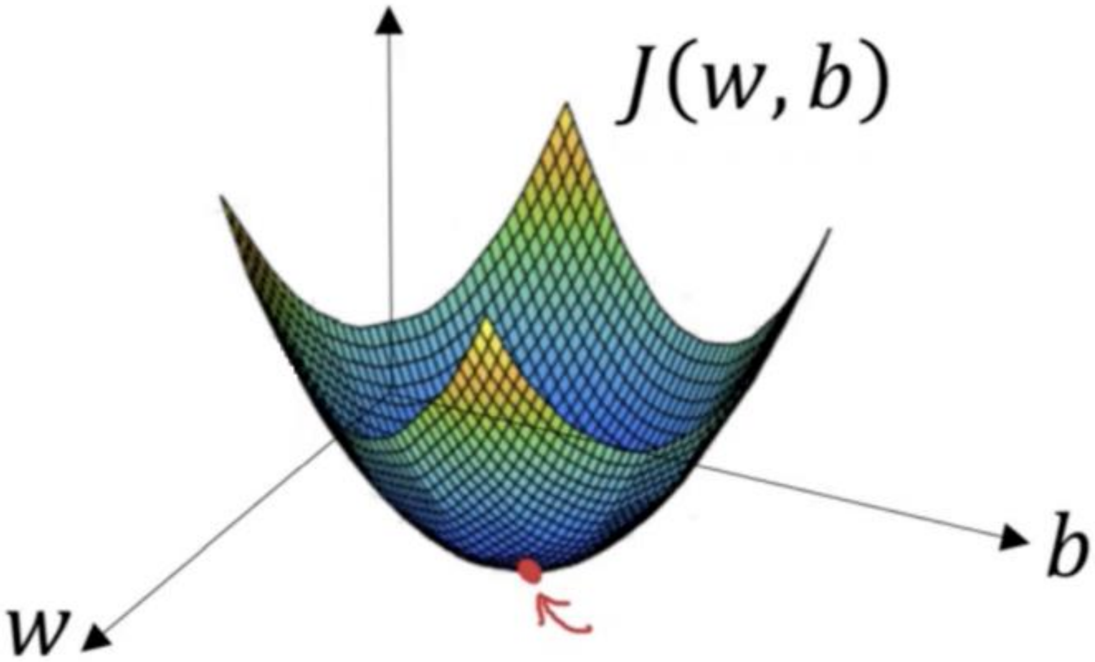
Optimizing a model with a single weight



Optimizing a model with a single weight



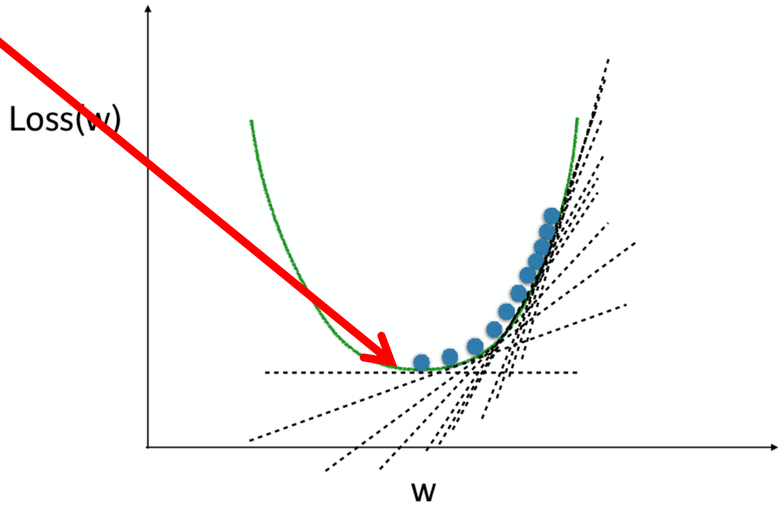
Optimizing a model with a single weight and bias



Gradient descent

With gradient decent, you repeatedly find a slope capturing how your loss function changes as your weights change.

You make a small change to the weight in order to get to the lower point, and you repeat this until you go cannot go downhill any more



Gradient descent

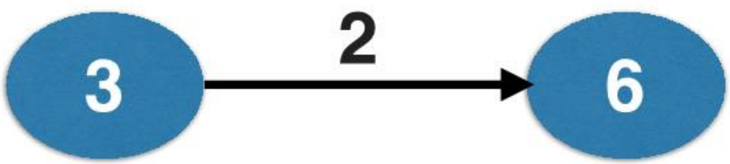
- If the slope is positive:
  - Going opposite the slope means moving to lower numbers
  - Subtract the slope from the current value
  - Too big a step might lead us astray
- Solution: learning rate
  - Update each weight by subtracting learning rate * slope

Gradient descent

- If the slope is positive:
  - Going opposite the slope means moving to lower numbers
  - Subtract the slope from the current value
  - Too big a step might lead us astray
- Solution: learning rate
  - Update each weight by subtracting learning rate * slope

  Learning rate is often a value ~ 0,01

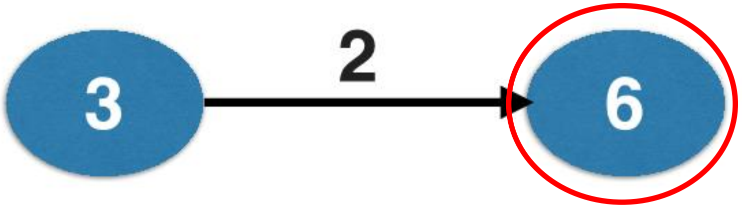
Slope calculation example on single data point



Actual Target Value = 10

- To calculate the slope for a weight, need to multiply three things:
  - Slope of the loss function w.r.t value at the node we feed into
  - The value of the node that feeds into our weight
  - Slope of the activation function w.r.t value we feed into

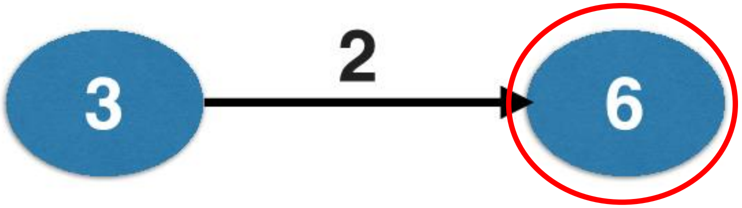
Slope calculation example on single data point



Actual Target Value = 10

- To calculate the slope for a weight, need to multiply three things:
  - Slope of the loss function w.r.t value at the node we feed into
  - The value of the node that feeds into our weight
  - Slope of the activation function w.r.t value we feed into

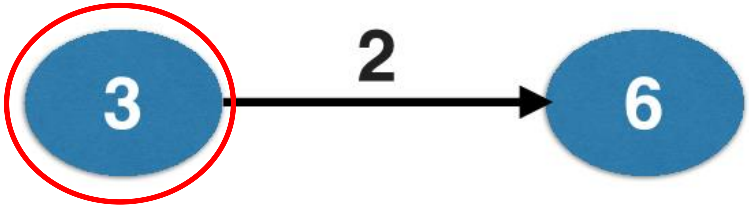
Slope calculation example on single data point



Actual Target Value = 10

- Slope of mean-squared loss function w.r.t prediction:
  - 2 * (Predicted Value - Actual Value) = 2 * Error
  - 2 * -4

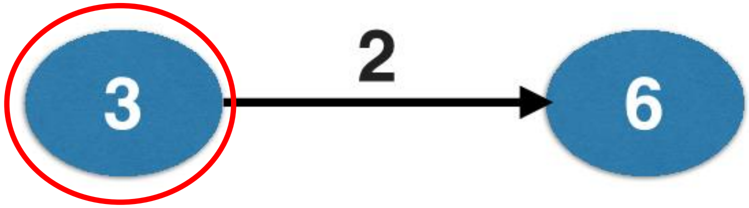
Slope calculation example on single data point



Actual Target Value = 10

- To calculate the slope for a weight, need to multiply three things:
  - Slope of the loss function w.r.t value at the node we feed into
  - The value of the node that feeds into our weight
  - Slope of the activation function w.r.t value we feed into

Slope calculation example on single data point

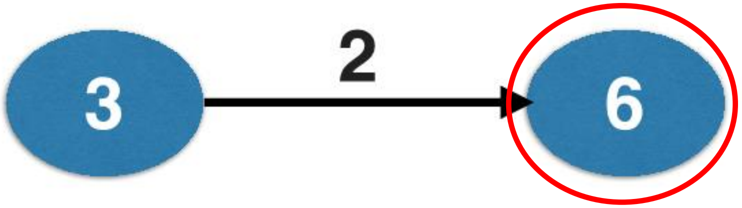


Actual Target Value = 10

- To calculate the slope for a weight, need to multiply three things:
  - Slope of the loss function w.r.t value at the node we feed into
  - The value of the node that feeds into our weight
  - ~~Slope of the activation function w.r.t value we feed into~~

Since we don't have the AF here, we leave this step

Slope calculation example on single data point



Actual Target Value = 10

Weight

- 2 * -4 * 3
- -24 ← Slope of the loss
- If learning rate is 0.01, the new weight would be
- 2 - 0.01(-24) = 2.24

Learning Rate | Slope (or gradient) | Updated weight

- **Cost function**

$$f(m,b) = \frac{1}{N}\sum_{i=1}^{n}(y_i - (mx_i + b))^2$$

- **Gradient function**

$$f'(m,b) = \begin{bmatrix} \frac{df}{dm} \\ \frac{df}{db} \end{bmatrix} = \begin{bmatrix} \frac{1}{N}\sum -2x_i(y_i - (mx_i + b)) \\ \frac{1}{N}\sum -2(y_i - (mx_i + b)) \end{bmatrix}$$

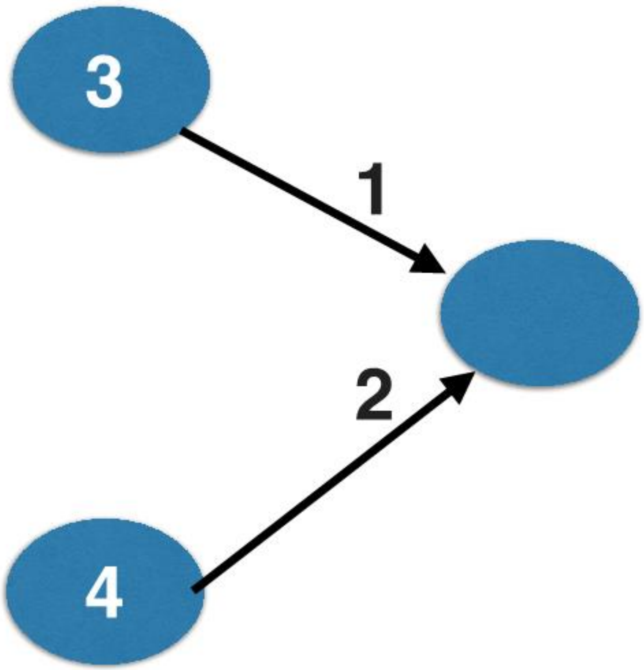
- https://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html

Slope calculation example with two inputs affecting prediction

- For multiple weights we repeat this calculation separately for each weight
- Then we update both weights simultaneously using their respective derivatives

Slope calculation example with two inputs affecting prediction

- Let’s see the code for this example

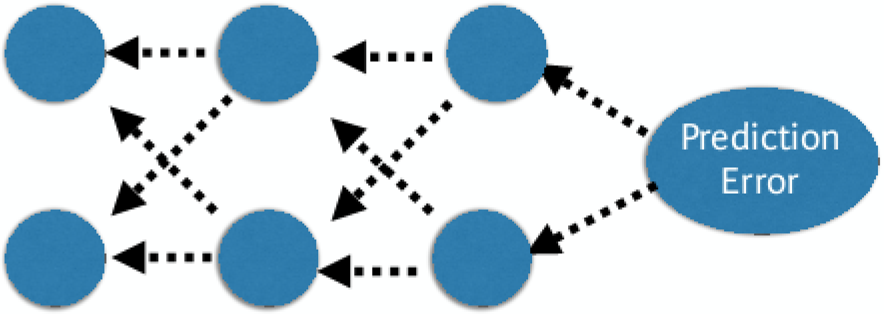


Slope calculation example with two inputs affecting prediction

```
In [1]: import numpy as np

In [2]: weights = np.array([1, 2])

In [3]: input_data = np.array([3, 4])

In [4]: target = 6

In [5]: learning_rate = 0.01

In [6]: preds = (weights * input_data).sum()

In [7]: error = preds - target

In [8]: print(error)
5

In [9]: gradient = 2 * input_data * error

In [10]: gradient
Out[10]: array([30, 40])

In [11]: weights_updated = weights - learning_rate * gradient

In [12]: preds_updated = (weights_updated * input_data).sum()

In [13]: error_updated = preds_updated - target

In [14]: print(error_updated)
-2.5
```

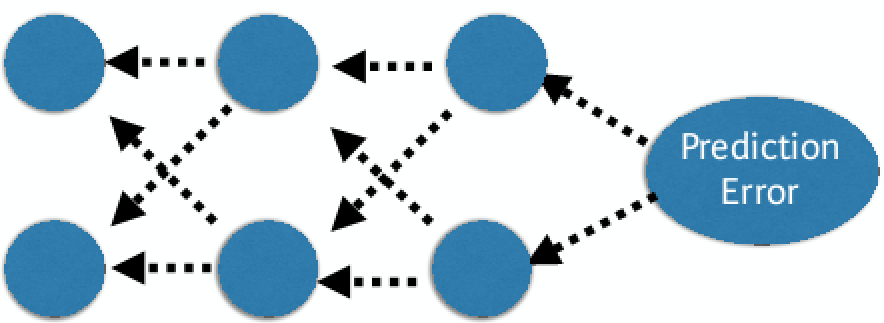
Backpropagation

- Forward propagation (FP) sends the input data through the hidden layers and into the output
- Backpropagation (BP) takes the error from the output layer and propagates it backward towards the input layer



Backpropagation

- It calculates the necessary slopes sequentially from the weights closest to the prediction, through the hidden layers, eventually back to the weights coming from the inputs
- We then use these slopes to update our weights as we have seen before

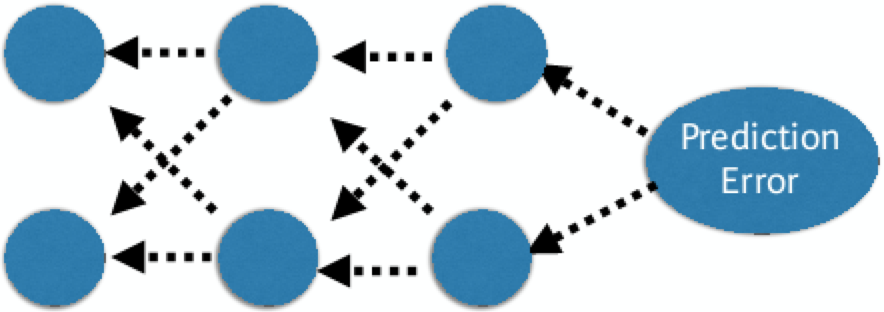


Backpropagation

- In the big picture, we are trying to estimate the slope of the loss function w.r.t. each weight of our network
- We use the prediction errors to calculate some of those slopes
- Therefore we always do FP to make a prediction and get the error, before we do BP

Backpropagation

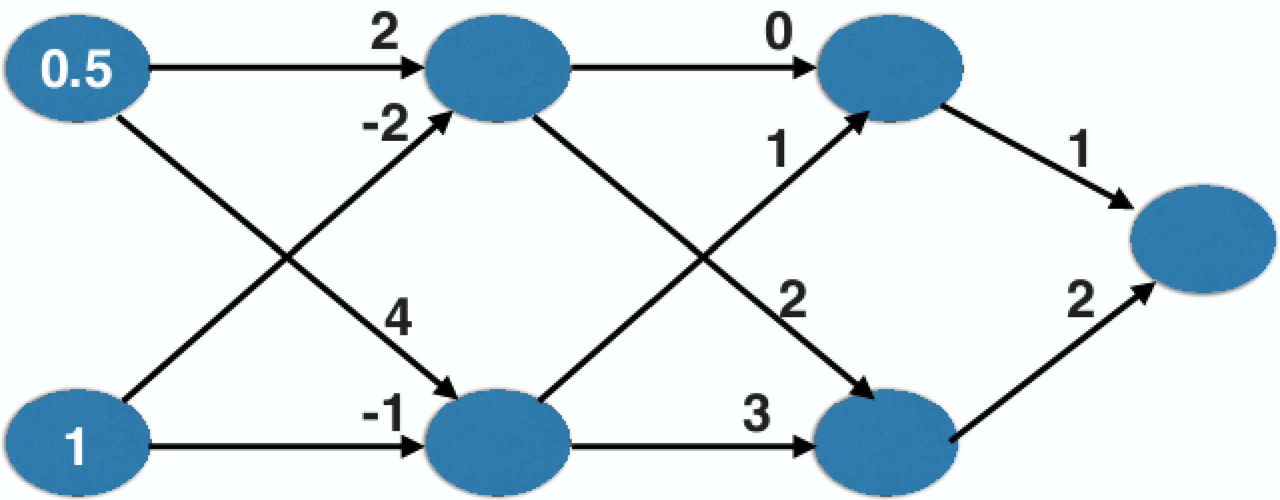
- Allows gradient descent to update all weights in neural network (by getting gradients for all weights)
- Comes from chain rule of calculus
- Important to understand the process, but you will generally use a library that implements this



Backpropagation process

ReLU Activation Function

Actual Target Value = 4

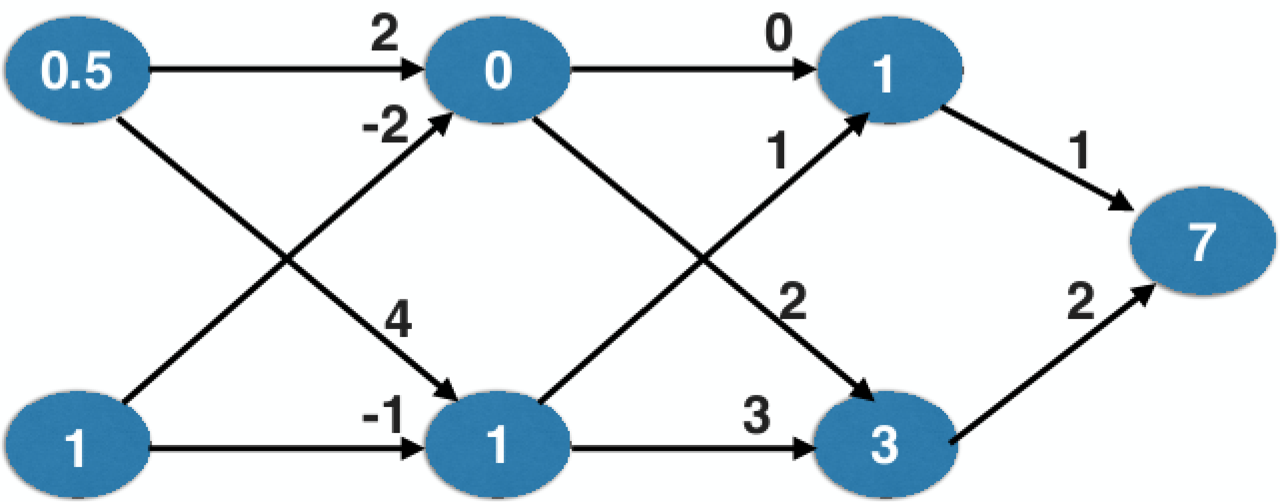


Backpropagation process

ReLU Activation Function

Actual Target Value = 4

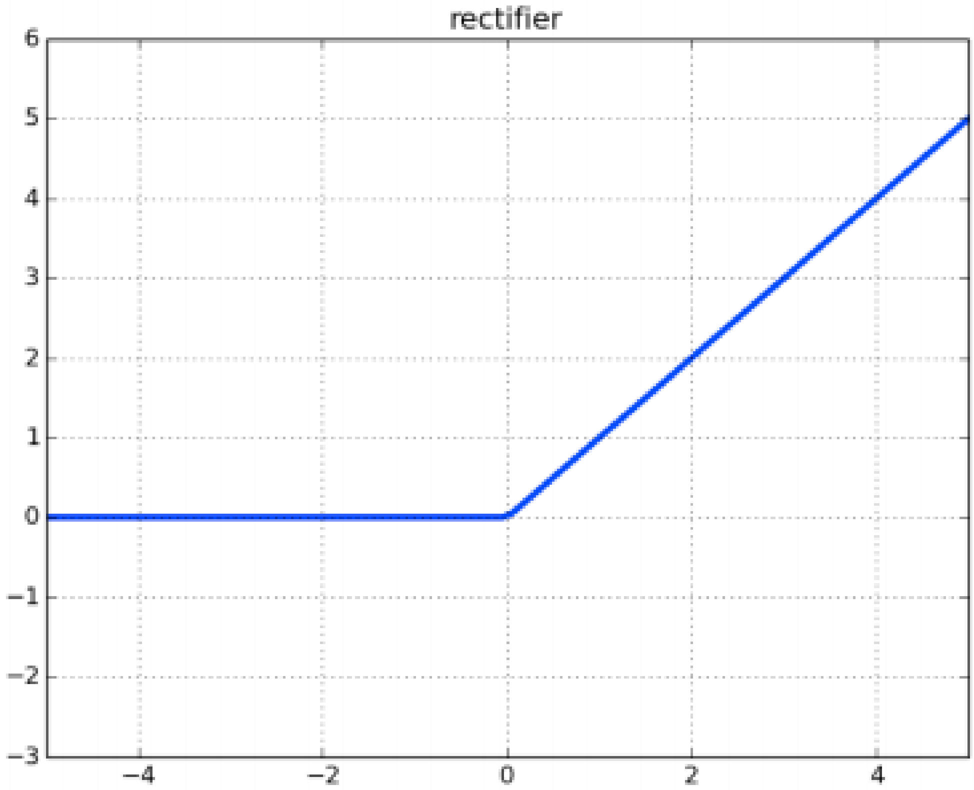
Error = 3



Backpropagation process

- Go back one layer at a time
- Gradients for weight is product of:
  1. Node value feeding into that weight
  2. Slope of loss function w.r.t node it feeds into
  3. Slope of activation function at the node it feeds into

ReLU Activation Function



Backpropagation process

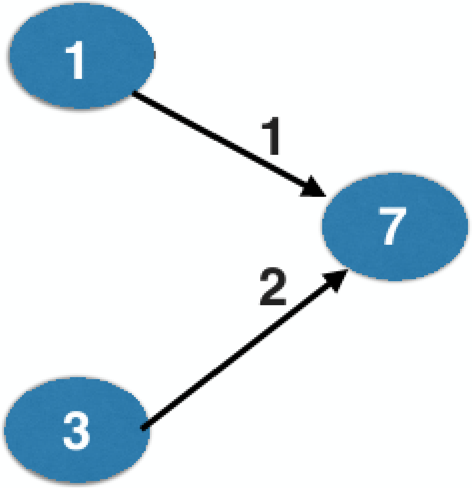
- Need to also keep track of the slopes of the loss function w.r.t node values
- Slope of node values are the sum of the slopes for all weights that come out of them

Backpropagation in practice

ReLU Activation Function

Actual Target Value = 4

Error = 3

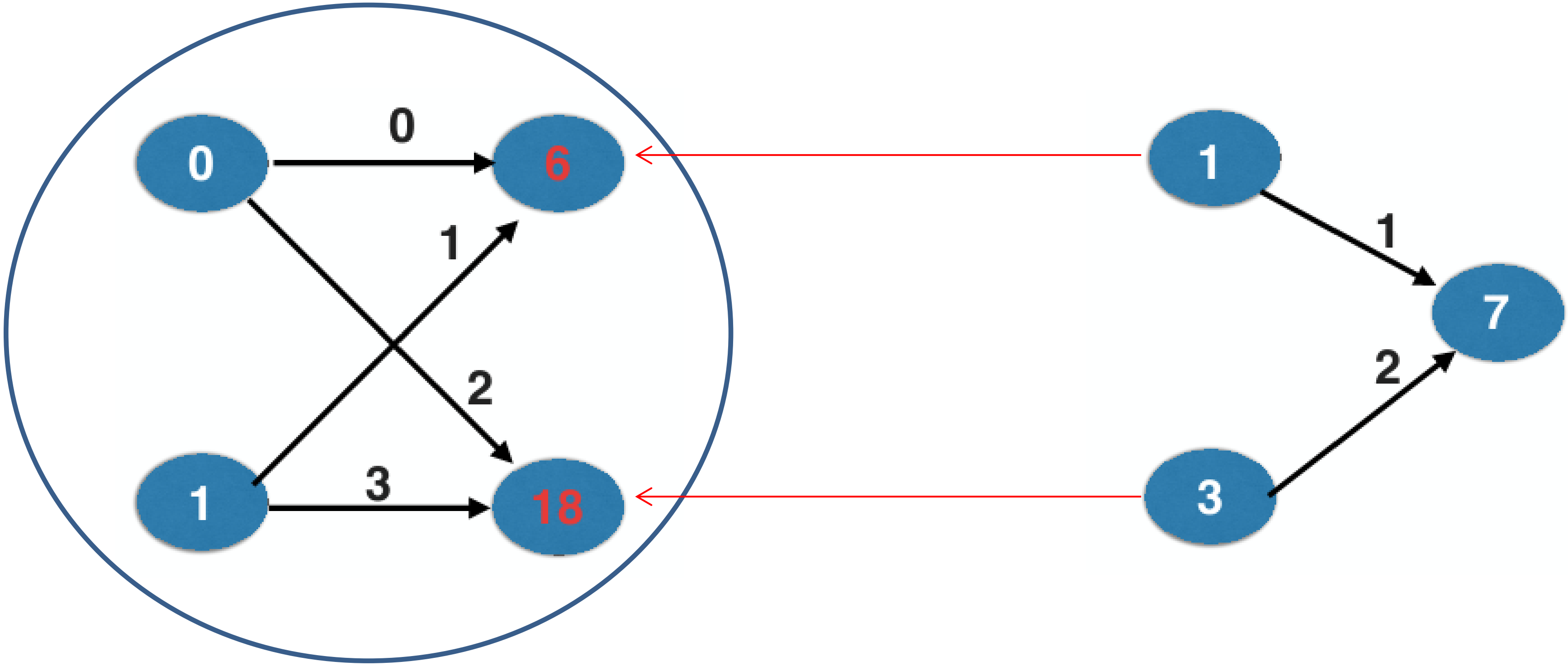


- The relevant slope for the output node is 2*error = 6
- And the slope for the activation function is 1, since the output is positive

So, we have

- Top weight’s slope = 1 * 6 = 6
- Bottom weight’s slope = 3 * 6 = 18

Backpropagation in practice

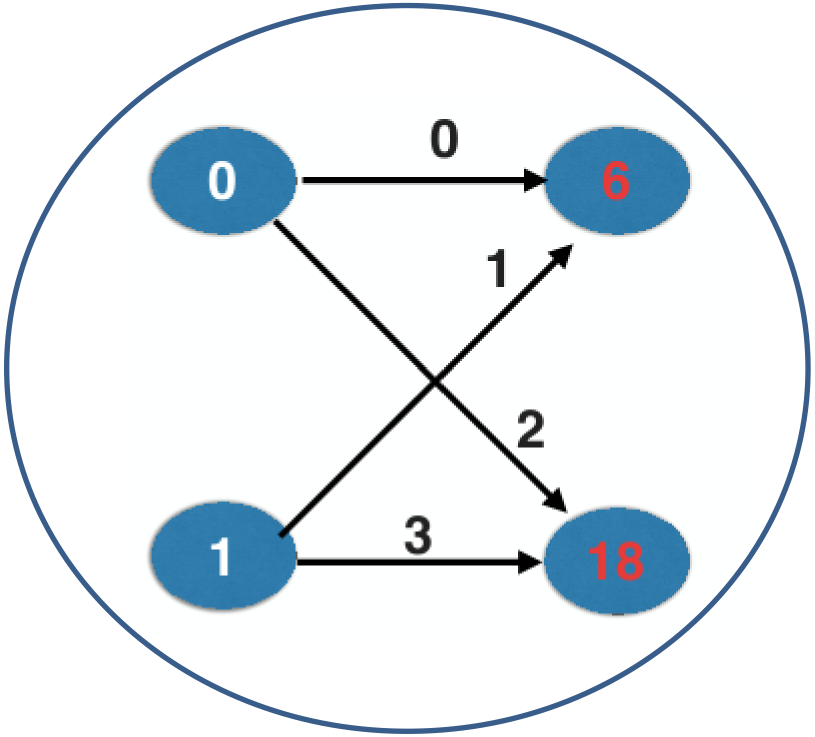


Calculating slopes associated with any weight

- Gradients for weight is product of:

1. Node value feeding into that weight
2. Slope of activation function for the node being fed into
3. Slope of loss function w.r.t output node

Backpropagation in practice



Current Weight Value	Gradient
0	0
1	6
2	0
3	18

Backpropagation: Recap

- Start at some random set of weights
- Use forward propagation to make a prediction
- Use backward propagation to calculate the slope of the loss function w.r.t each weight
- Multiply that slope by the learning rate, and subtract from the current weights
- Keep going with that cycle until we get to a flat part

Stochastic gradient descent

- It is common to calculate slopes on only a subset of the data ('batch')
- Use a different batch of data to calculate the next update
- Start over from the beginning once all data is used
- Each time through the training data is called an epoch
- When slopes are calculated on one batch at a time: stochastic gradient descent

To be continued,

Thanks!