Deep Learning for NLP

Session 2: Basics of loss functions, SGD & Pytorch

Sharid Loáiciga

Linguistics Department
University of Potsdam

April 28th, 2020

Slides credits: Miikka Silfverberg & Hande Celikkanat

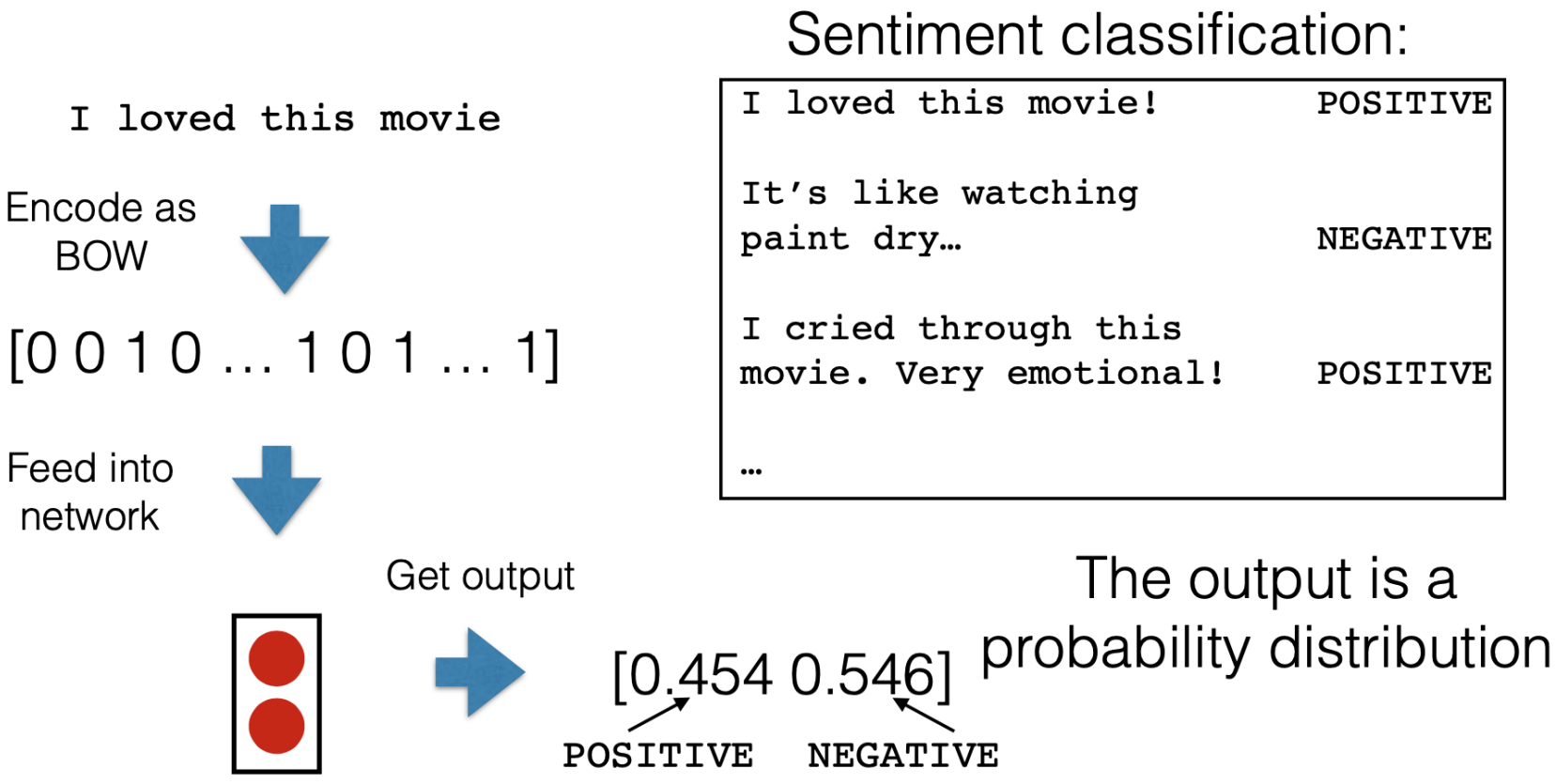
Outline

1. Recap
2. Pytorch basics notebook
3. Brief intro to SGD and loss functions
4. Get started on assignment 1

Linear regression: models continuous variables, not the most useful for NLP

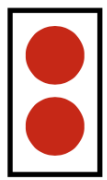
Logistic "regression": models categorical variables, used for classification

Logistic “regression”



Logistic “regression”

x [0 0 1 0 … 1 0 1 … 1]



w_{POS} [0.123 -1.1 3.02 -0.97 …]

w_{NEG} [-0.123 1.1 3.02 0.97 …]

softmax

$$p_{\text{POS}} = \frac{\exp(x \cdot w_{\text{POS}})}{\exp(x \cdot w_{\text{POS}}) + \exp(x \cdot w_{\text{NEG}})}$$

$$p_{\text{NEG}} = \frac{\exp(x \cdot w_{\text{NEG}})}{\exp(x \cdot w_{\text{POS}}) + \exp(x \cdot w_{\text{NEG}})}$$

Linear regression: models continuous variables, not the most useful for NLP

Logistic "regression": models categorical variables, used for classification

- equivalent to single neuron **perceptron** with a sigmoid activation function
- **softmax "regression":** normalized exponentiation that squeezes more than 1 class into a distribution [1]

Pytorch basics

Brief intro to loss functions & SGD

Loss functions: the bread and butter of ML

Learning happens because of loss functions

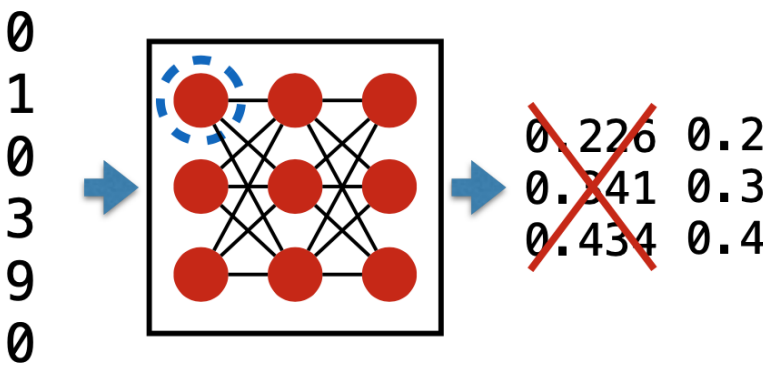
The loss can be a general function which measures the fit of the model to the training data. It should = 0 when the model fits the data perfectly.

For practical reasons, the loss is usually an easily differentiable function.

Training NN

A neural network maps an input vector into an output vector.

A change in parameter values influences the mapping.



It has parameters ● :

[0.365, 0.482, 0.156, 0.294, 0.339, 0.053]

0.220

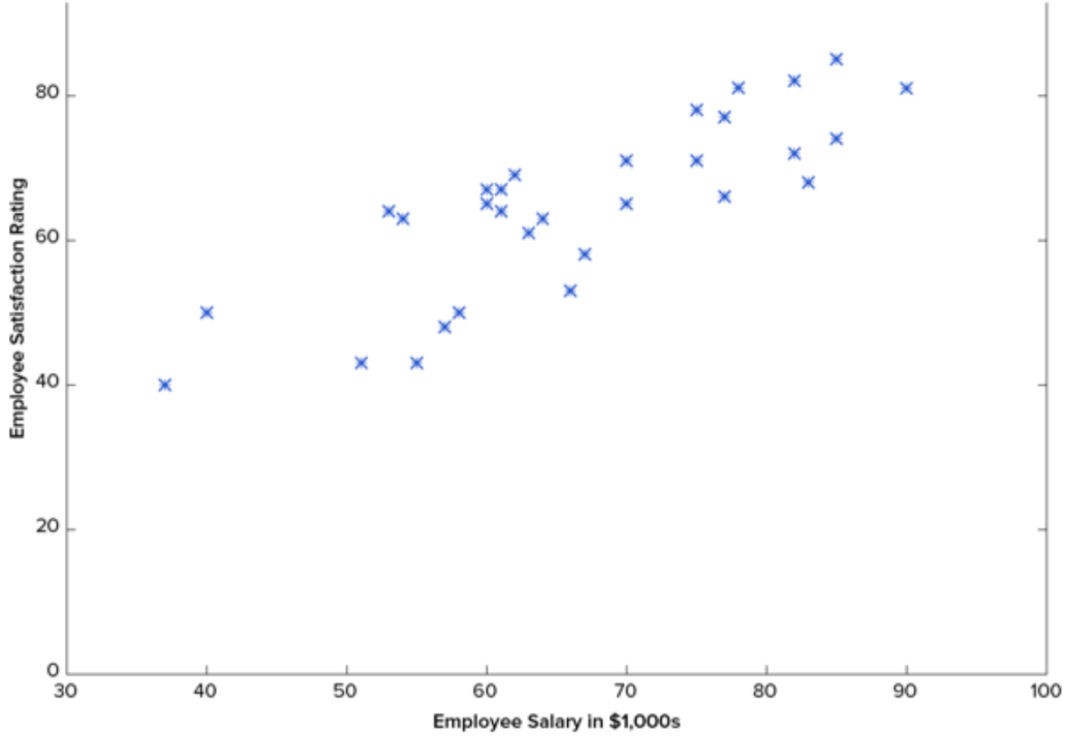
Training = Adjusting the parameters so that **the loss is minimized.**

How do we decrease the loss?

Example: linear regression

The main trick in ML: Change your parameters, change your loss!
Find **the best parameters** that minimize your loss.

Let's say we have this data: Try to find a linear regression model that fits the data as well as possible.

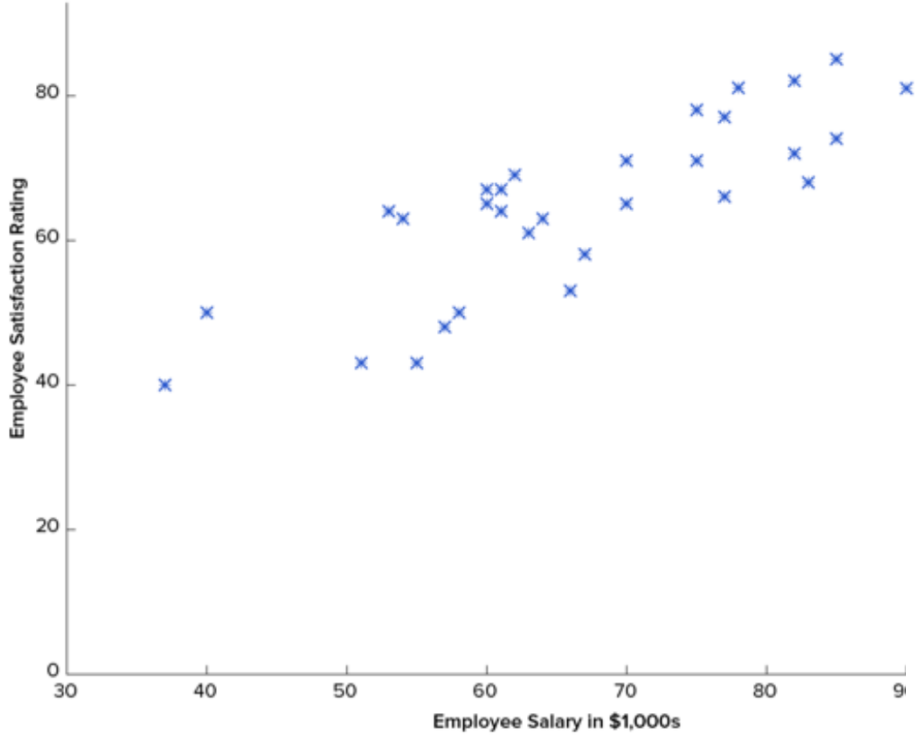


How do we decrease the loss?

Example: linear regression

The main trick in ML: Change your parameters, change your loss!
Find **the best parameters** that minimize your loss.

Let's say we have this data: Try to find a linear regression model that fits the data as well as possible.



The usual loss function for linear regression is Mean Square Error:

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

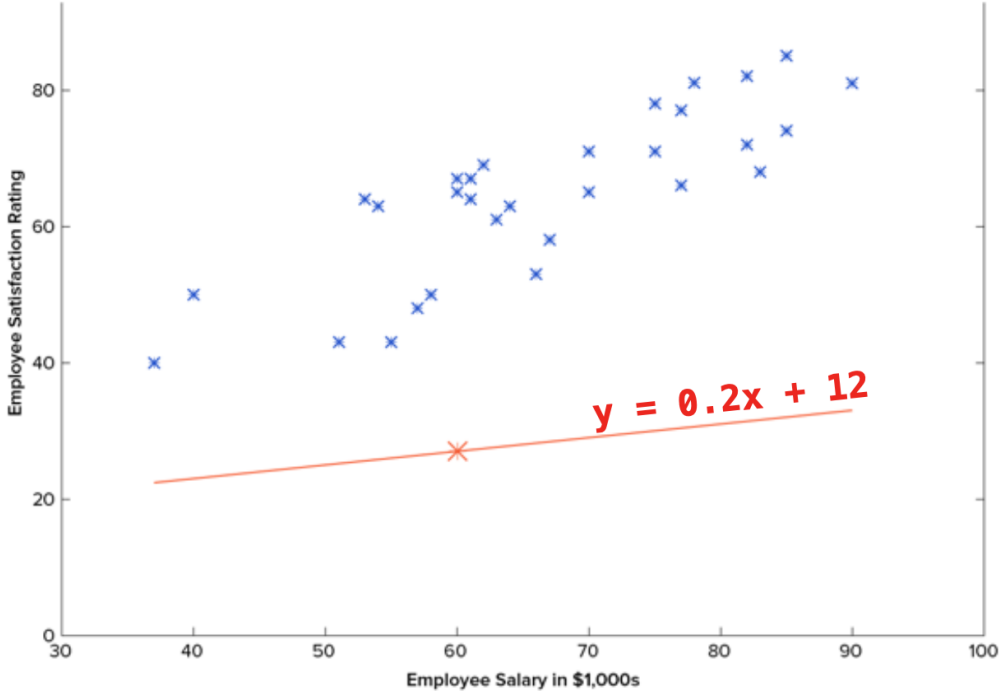
$\hat{y}_i$ - predicted y for example i

y_i - gold standard y for example i

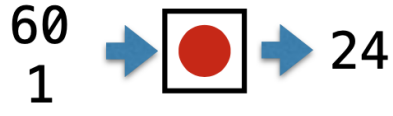
How do we decrease the loss?

Let's start with a **random** initial model, say slope=0.2, intercept=12:

`y = 0.2x + 12`



We now have a neural network like this:

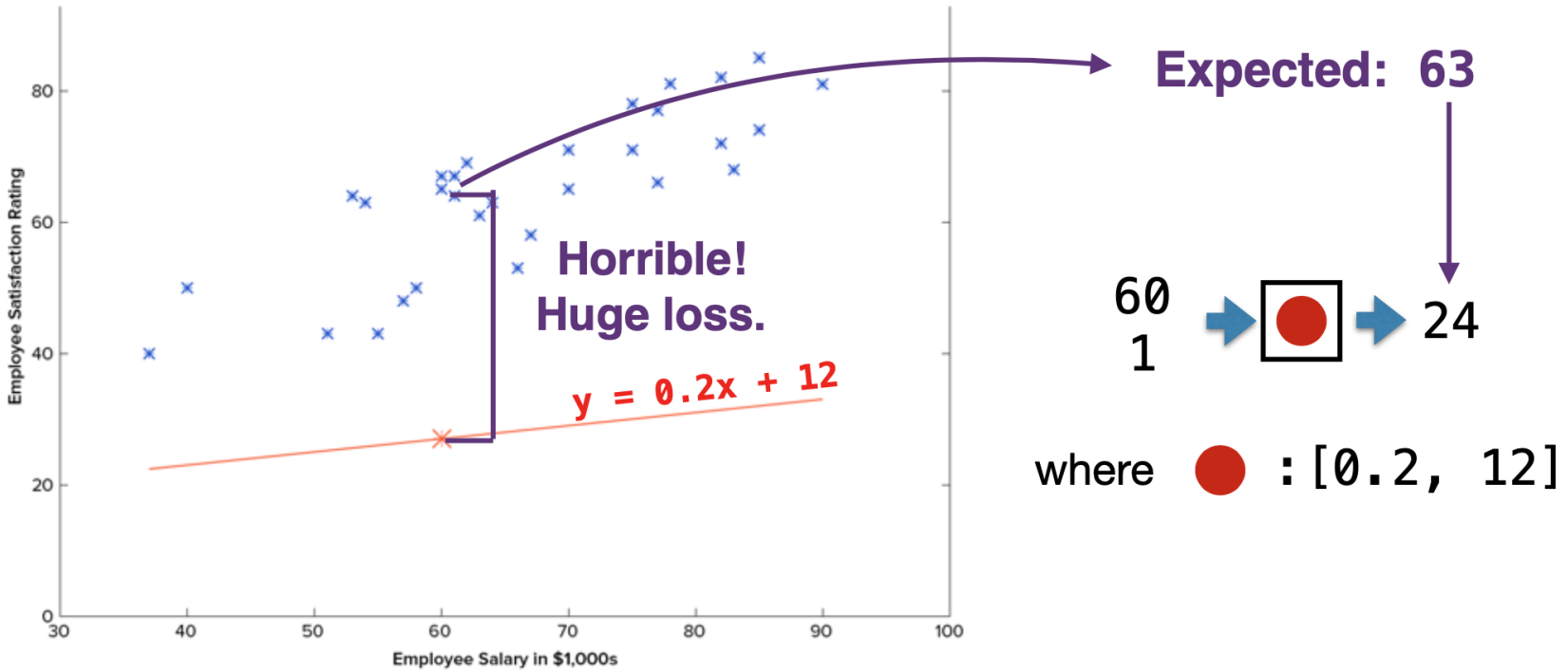


where ● : `[0.2, 12]`

Graph from: https://www.toptal.com/machine-learning/machine-learning-theory-an-introductory-primer

How do we decrease the loss?

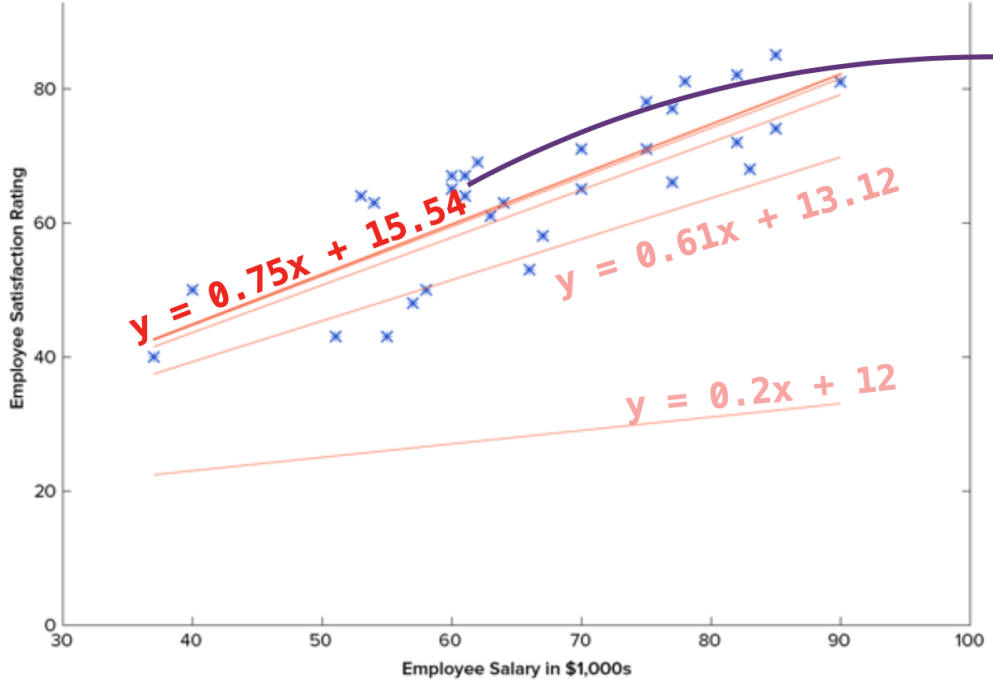
`y = 0.20x + 12`: How good is this model?
Let's use it for prediction:

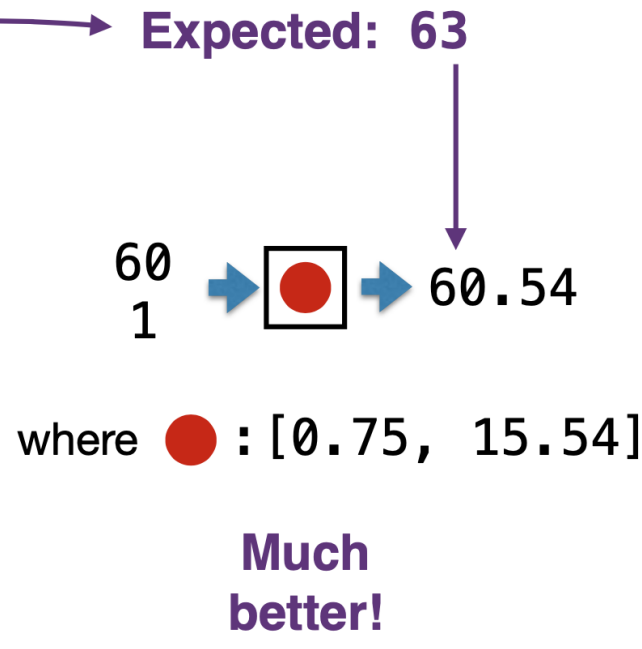


Graph from: https://www.toptal.com/machine-learning/machine-learning-theory-an-introductory-primer

How do we decrease the loss?

Say `y = 0.75x + 15.54` : This is really good!
Actually, it is as good as we can get with this model.
(What does it mean that it's as good as it can get?)





Graph from: https://www.toptal.com/machine-learning/machine-learning-theory-an-introductory-primer

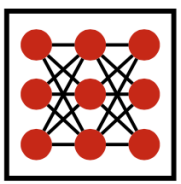
Loss function for logistic regression

`I loved this movie`

Encode as BOW

[0 0 1 0 … 1 0 1 … 0]

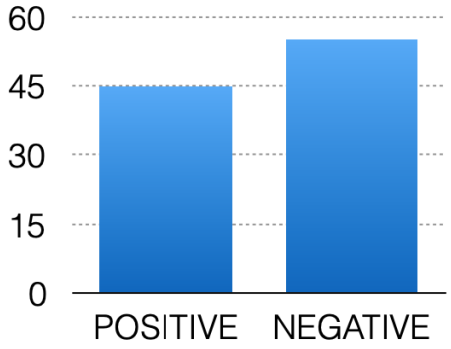
Feed into network

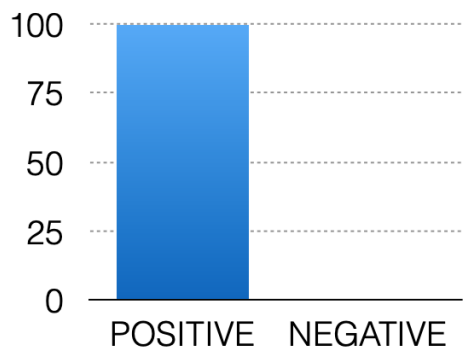


Get output

[0.454 0.546]

`POSITIVE` `NEGATIVE`



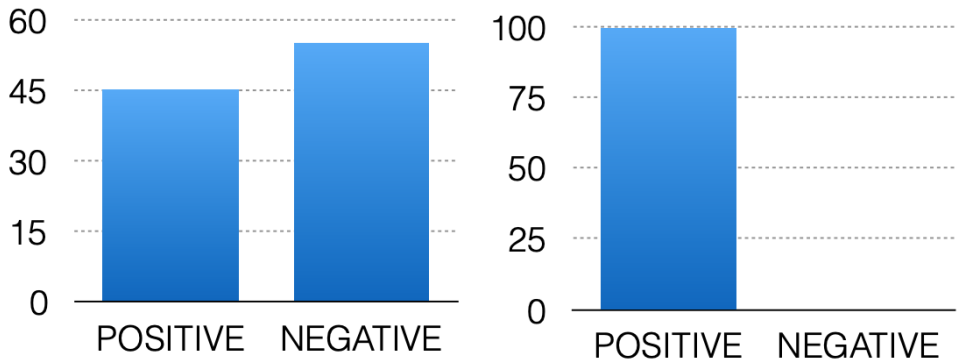


Gold standard:

[1.000 0.000]

Loss function for logistic regression

Cross-entropy



A common way to compare two distributions is cross entropy:

$$-y_{\text{POS}} \cdot \log \hat{y}_{\text{POS}} - y_{\text{NEG}} \cdot \log \hat{y}_{\text{NEG}}$$

`http://colah.github.io/posts/2015-09-Visual-Information/`

Loss function for logistic regression

Cross-entropy

y_{POS} : 1.0
y_{NEG} : 0.0

$$\mathrm{L}(\hat{y}, y) = -y_{\text{POS}} \cdot \log \hat{y}_{\text{POS}} - y_{\text{NEG}} \cdot \log \hat{y}_{\text{NEG}}$$

$\hat{y}_{\text{POS}}$: 0.9
$\hat{y}_{\text{NEG}}$: 0.1

$$\mathrm{L}(\hat{y}, y) = \text{-1.0*log(0.9) - 0.0 * log(0.1) = 0.105}$$

$\hat{y}_{\text{POS}}$: 0.1
$\hat{y}_{\text{NEG}}$: 0.9

$$\mathrm{L}(\hat{y}, y) = \text{-1.0*log(0.1) - 0.0 * log(0.9) = 2.303}$$

$\hat{y}_{\text{POS}}$: 0.5
$\hat{y}_{\text{NEG}}$: 0.5

$$\mathrm{L}(\hat{y}, y) = \text{-1.0*log(0.5) - 0.0 * log(0.5) = 0.693}$$

Stochastic Gradient Descend

A principled approach to minimizing the loss

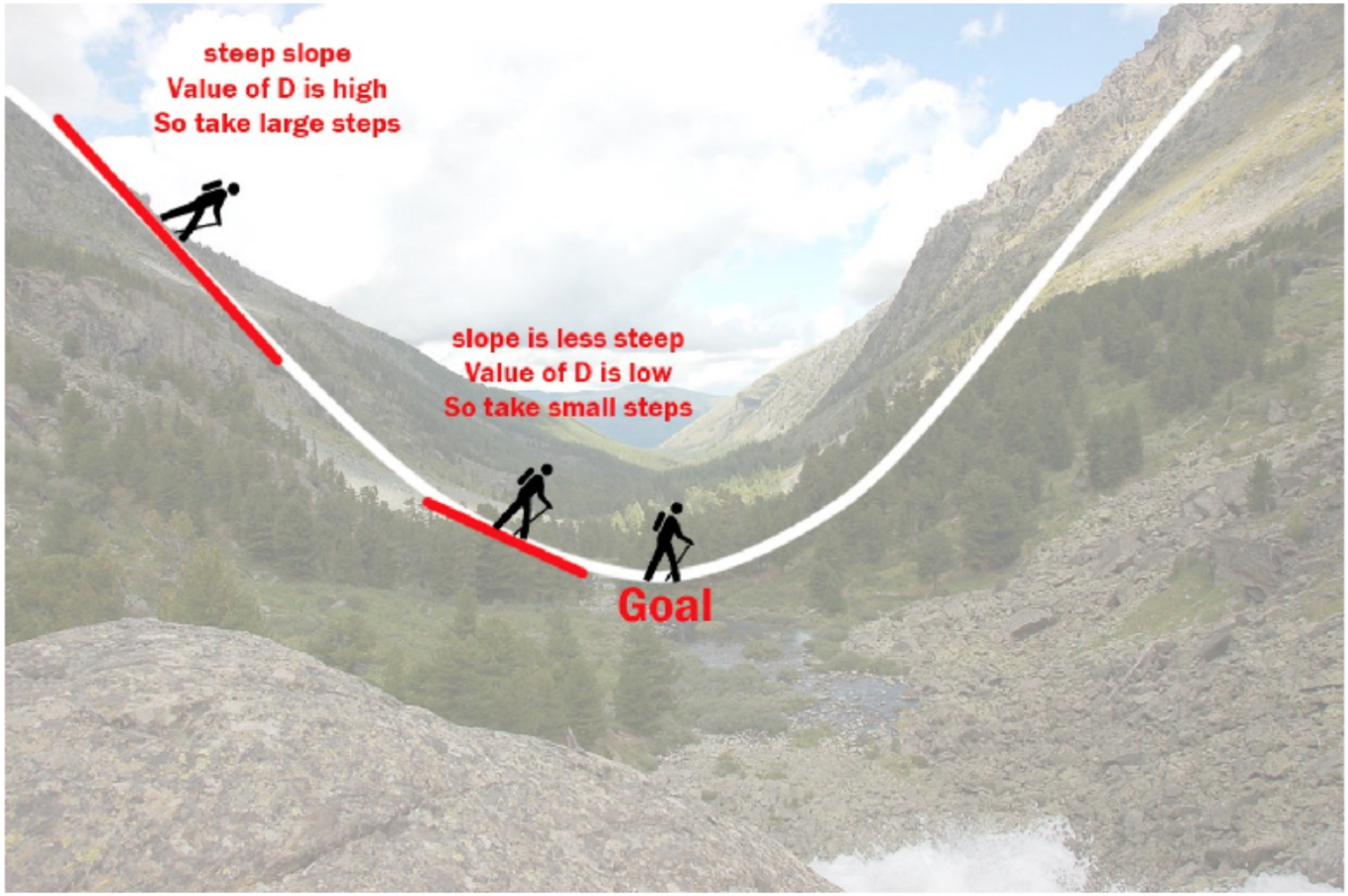


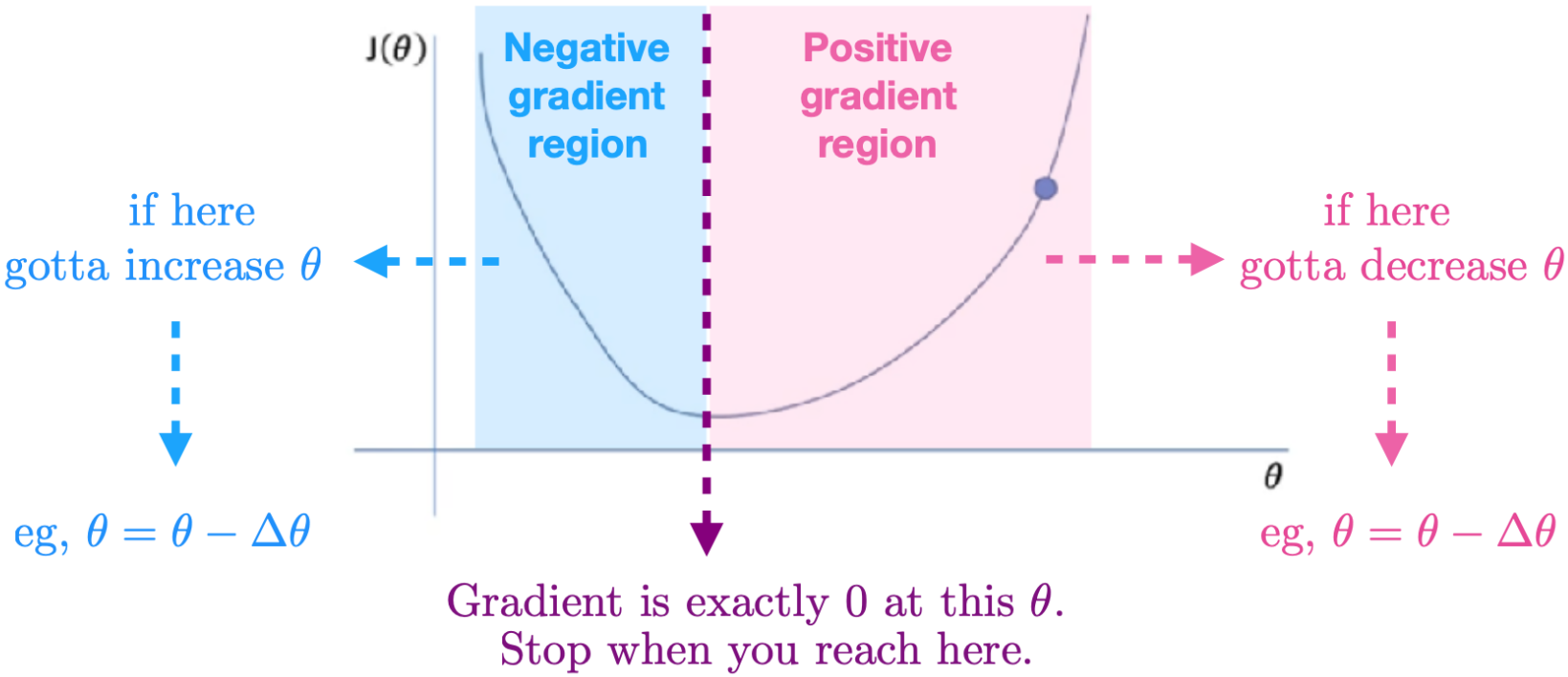
Illustration of how the gradient descent algorithm works

https://towardsdatascience.com/linear-regression-using-gradient-descent-97a6c8700931

SGD

How can we make sure that we find the minimum of the loss function?

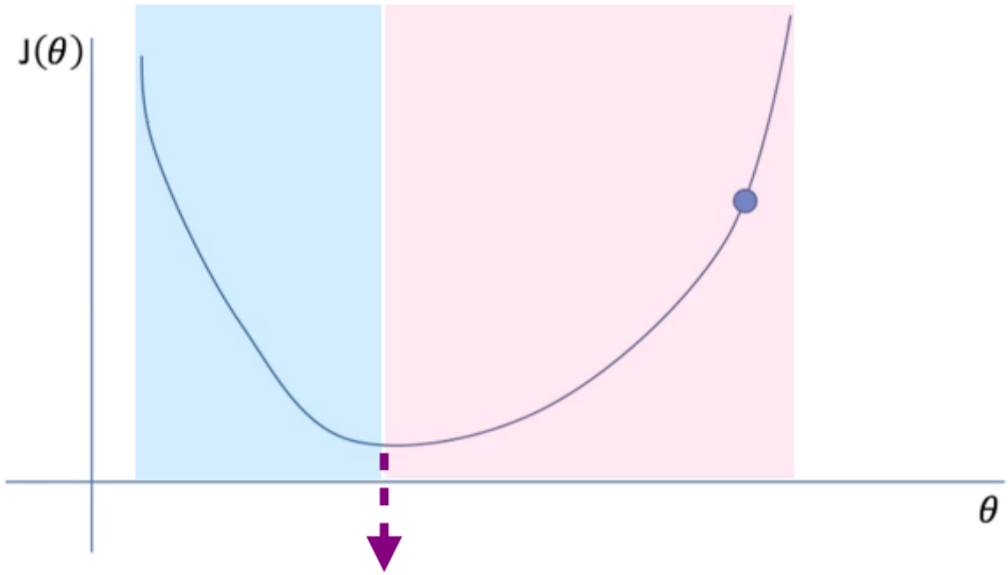
Question: How can you find the minimum of a function?



SGD

How can we make sure that we find the minimum of the loss function?

Question: How can you find the minimum of a function?



Stop here!

This is our ideal θ: The global "minimum" of the loss function.

https://www.jeremyjordan.me/gradient-descent/

SGD

example: SGD for linear regression

Model: `y = m.x + c`

Let's try to minimize Mean Squared Loss.
(Question: Why do we need to decide this in advance?)

$$L = \frac{1}{N}\sum_{i=0}^{N}(y_i - \hat{y}_i)^2$$

SGD

example: SGD for linear regression

Model: `y = m.x + c`

$$L = \frac{1}{N}\sum_{i=0}^{N}(y_i - (m.x_i + c))^2$$

Derivative wrt. m:

$$\frac{\partial L}{\partial m} = \frac{1}{N}\sum_{i=0}^{N} 2(y_i - (m.x_i + c))(-x_i)$$

$$\frac{\partial L}{\partial m} = \frac{-2}{N}\sum_{i=0}^{N} x_i(y_i - \hat{y}_i)$$

Derivative wrt. c:

$$\frac{\partial L}{\partial c} = \frac{1}{N}\sum_{i=0}^{N} 2(y_i - (m.x_i + c))$$

$$\frac{\partial L}{\partial c} = \frac{-2}{N}\sum_{i=0}^{N} (y_i - \hat{y}_i)$$

SGD

example: SGD for linear regression

While **`Loss`** is still not acceptable:

(Say it's time t and our parameters atm are $\theta^t = [\theta_1^t, \theta_2^t]$

Draw **one** input-output pair from training data: **`(xi, yi)`**

Compute **`Loss`** for it: $L(f(x_i; \theta_1^t, \theta_1^t), y_i)$

Calculate gradients wrt. $[\theta_1^t, \theta_2^t]$: $[\frac{\partial L}{\partial \theta_1^t}, \frac{\partial L}{\partial \theta_2^t}]$

$\hat{g}_1$ $\hat{g}_2$

Update both parameters:

$$\theta_1^t \leftarrow \theta_1^t - \mu.\hat{g}_1 \qquad \theta_2^t \leftarrow \theta_2^t - \mu.\hat{g}_2$$

SGD

example: SGD for linear regression

While **`Loss`** is still not acceptable:

(Say it's time t and our parameters atm are $\theta^t = [\theta_1^t, \theta_2^t]$)

Draw **one** input-output pair from training data: **`(xi, yi)`**

Question: Only one data point at a time?
Does this guarantee convergence at all?

SGD

example: SGD for linear regression

While **`Loss`** is still not acceptable:

(Say it's time t and our parameters atm are $\theta^t = [\theta_1^t, \theta_2^t]$)

Draw **one** input-output pair from training data: **`(xi, yi)`**

**Yes! As the number of training examples approach infinity,
as we sample each of them once,
we get closer and closer to the best model!**

SGD

example: SGD for linear regression

While **`Loss`** is still not acceptable:

(Say it's time t and our parameters atm are $\theta^t = [\theta_1^t, \theta_2^t]$)

Draw **one** input-output pair from training data: **`(xi, yi)`**

In practice though,
we must make *multiple passes* over the *same* training points.
Each pass is called an *epoch*.

SGD

example: SGD for linear regression

Derivative of Loss wrt. m:

$$\frac{\partial L}{\partial m} = \frac{-2}{N}\sum_{i=0}^{N} x_i(y_i - \hat{y}_i)$$

Derivative of Loss wrt. c:

$$\frac{\partial L}{\partial c} = \frac{-2}{N}\sum_{i=0}^{N} (y_i - \hat{y}_i)$$

Model at t=0

x0	ŷ0	y0
60 1	24	63

Loss

$$L_0 = (y_i - \hat{y}_i)^2 = (63 - 24)^2 = 1521$$

Derivatives

$$\frac{\partial L_0}{\partial m} = -2x_i(y_i - \hat{y}_i) = -2.60.(63 - 24) = -4680$$

$$\frac{\partial L_0}{\partial c} = -2(y_i - \hat{y}_i) = -2.(63 - 24) = -78$$

● :[0.2, 12]

−μ.−4680

−μ.−78

SGD

example: SGD for linear regression

Derivative of Loss wrt. m:

$$\frac{\partial L}{\partial m} = \frac{-2}{N}\sum_{i=0}^{N} x_i(y_i - \hat{y}_i)$$

Derivative of Loss wrt. c:

$$\frac{\partial L}{\partial c} = \frac{-2}{N}\sum_{i=0}^{N} (y_i - \hat{y}_i)$$

Model at t=1

x0		ŷ0	y0	Loss
60 1		52	63	$L_1 = (y_i - \hat{y}_i)^2 = (63 - 52)^2 = 121$

●:[0.668, 12.0078]