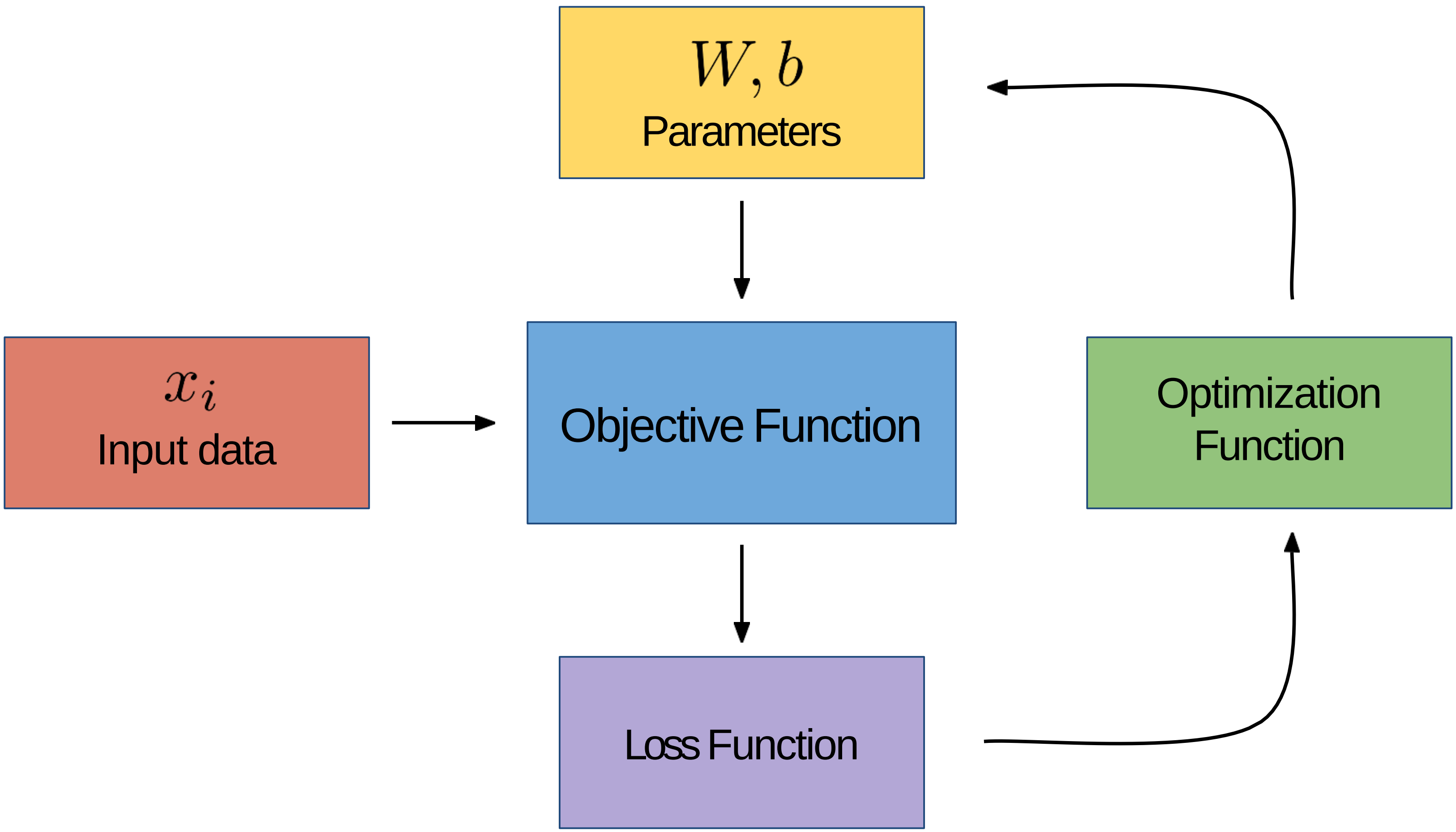
Machine Learning

Lecture # 2

Loss Functions

Lifecycle of Training a Machine Learning Model

Learning Lifecycle



Objective Function

$$f(x, W, b) = W \cdot x + b$$

Objective function defines our *goal*

f ( , W, b) = N Numbers

Input

Parameters

P

Objective Function

$$f(x, W, b) = W \cdot x + b$$

Objective function defines our *goal*

f ( , W, b) = N Numbers

Input

Parameters
P

N can be 1 as in our previous example

Objective Function

f ( , W, b) = 1 Number

This 1 number can be a real valued output (for example depicting price, age etc). This is called regression.

This 1 number can also be used in the special case of binary classification (two classes) like we did in the previous exercise - i.e. **Class 1 if f > 0 and Class 2 if f ≤ 0**

Objective Function

$$f(x, W, b) = W \cdot x + b$$

Objective function defines our *goal*

f ( , W, b) = N Numbers

Input

Parameters
P

Outputs
N=2 can be used for binary classification
Example: scores for the two classes - *accident prone* and *not accident prone*

Objective Function

$$f(x, W, b) = W \cdot x + b$$

Objective function defines our *goal*

f( , W, b) = N Numbers

Input

Parameters
P

Outputs
N=M can be used for M-class classification
Example: M = 40 for POS tags

Objective Function

$$f(x, W, b) = W \cdot x + b$$

Objective function defines our *goal*

f ( , W, b) = 2 Numbers

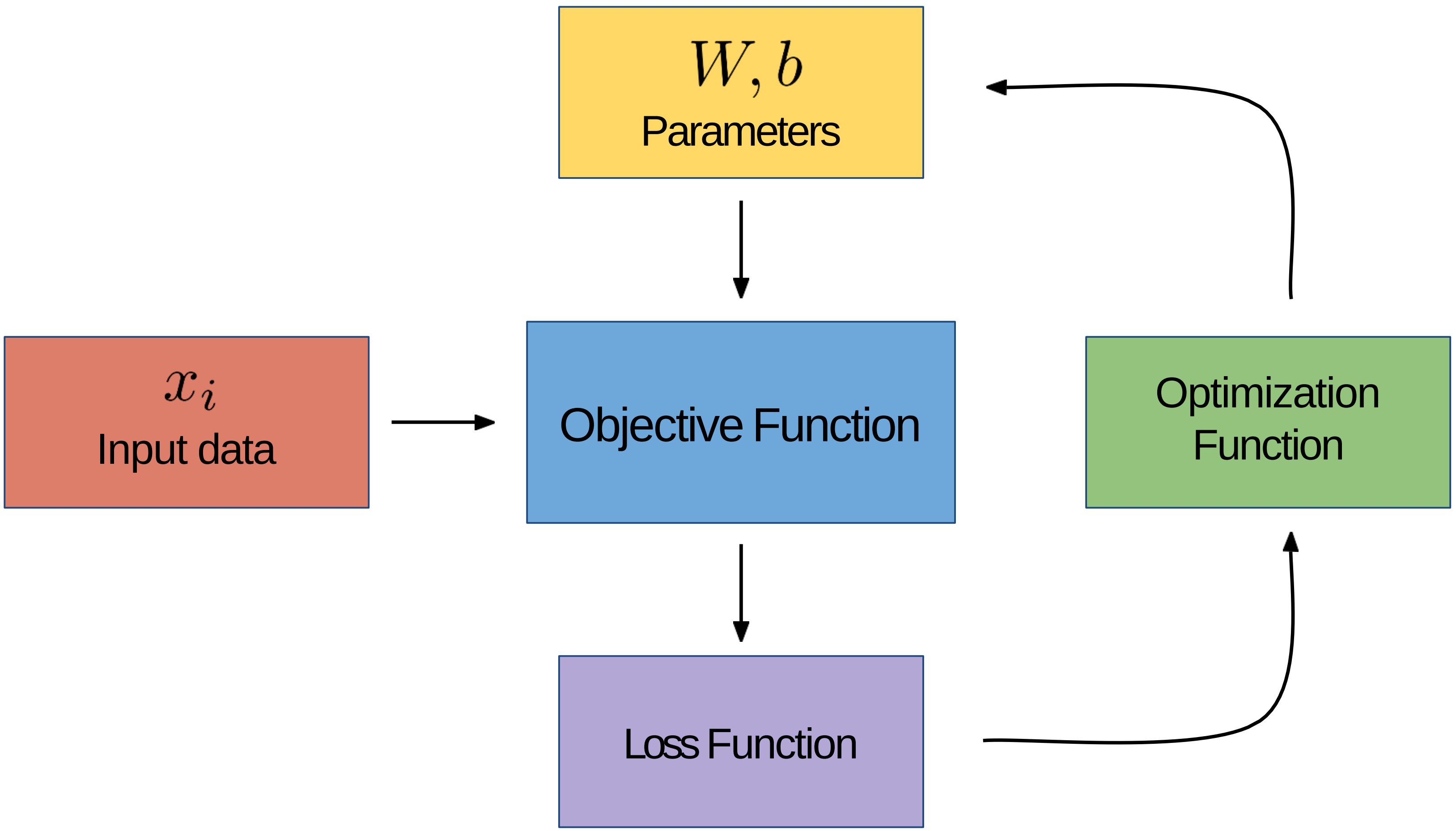
Input

Parameters
P

Outputs

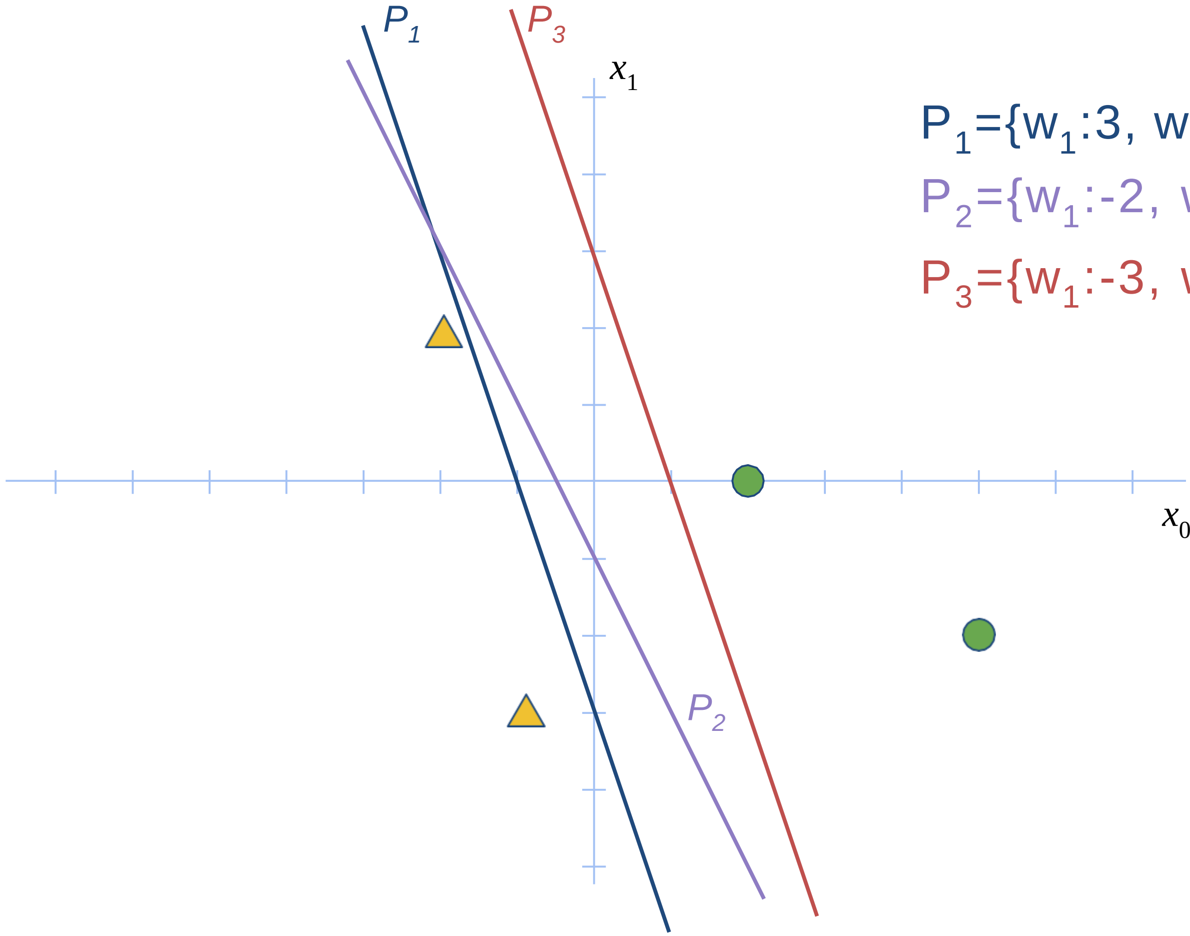
Learned by the algorithm, just like we learned W and b in the previous exercise!

Learning Lifecycle



Loss Function

Given a set of parameters $P=\{P_1, P_2, \ldots\}$, how do you know which one to use?



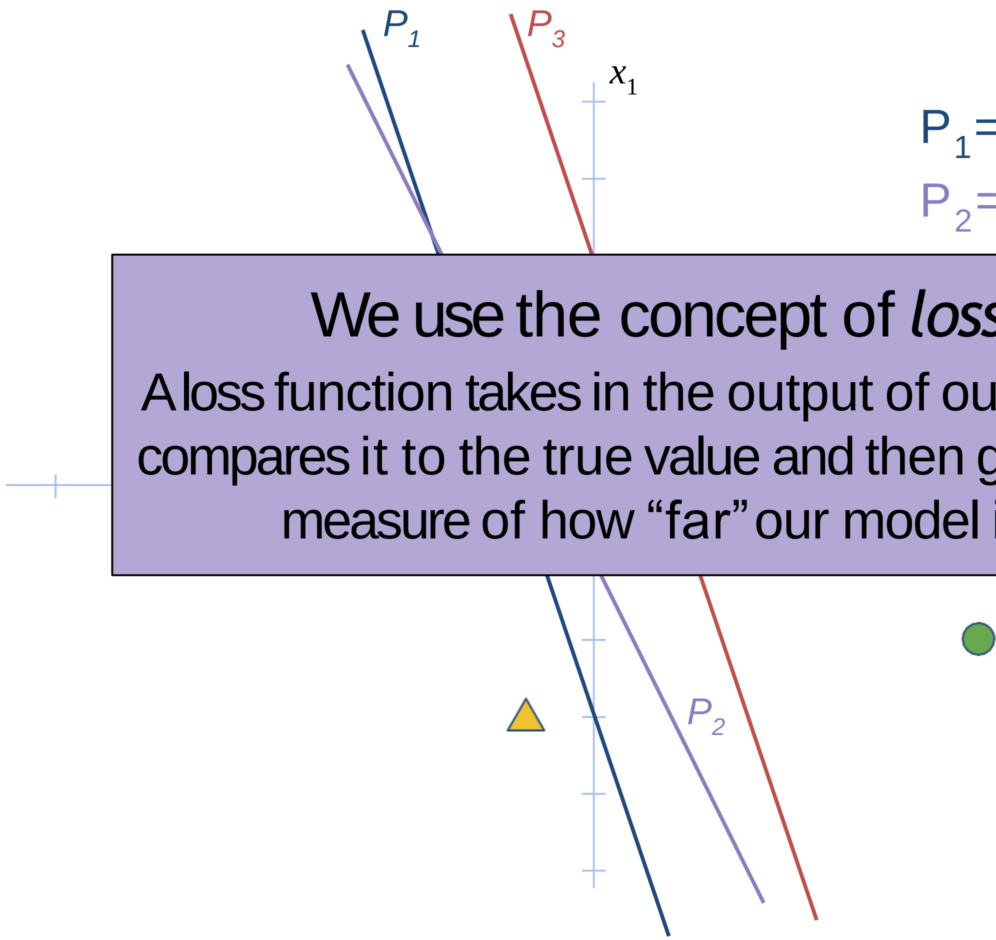
$P_1=\{w_1:3,\ w_2:1,\ b:3\}$

$P_2=\{w_1:-2,\ w_2:-1,\ b:-1\}$

$P_3=\{w_1:-3,\ w_2:-1,\ b:3\}$

Loss Function

Given a set of parameters P={P_1, P_2, …}, how do you know which one to use?



P_1={w_1:3, w_2:1, b:3}

P_2={w_1:-2, w_2:-1, b:-1}

v_2:-1, b:3}

We use the concept of *loss*

A loss function takes in the output of our model, compares it to the true value and then gives us a measure of how "far" our model is.

Loss Function

A loss function is *any function that gives a measure* of how far your scores are from their true values

Loss Function

A loss function is *any function that gives a measure* of how far your scores are from their true values

Not accident prone

Accident prone

`[1.0, 0.0]` ← True values → `[0.0, 1.0]`

Loss Function Exercise

Consider two cars and three sets of parameters

Not accident prone

Accident prone

P_1 P_2 P_3

Which set of parameters is the best?

f ( , P_1) = [0.5, 0.5]

f ( , P_2) = [0.7, 0.3]

f ( , P_3) = [0.1, 0.9]

f( , P_1) = [0.1, 0.9]

f( , P_2) = [0.3, 0.7]

f( , P_3) = [0.9, 0.1]

Loss Function Exercise

Consider two cars and three sets of parameters

Not accident prone

P_1 P_2 P_3

Which set of parameters is the best?

f ( , P_1) = [0.5, 0.5]   f( , P_1) = [0.1, 0.9]

f ( , P_2) = [0.7, 0.3]   f( , P_2) = [0.3, 0.7]

f ( , P_3) = [0.1, 0.9]   f( , P_3) = [0.9, 0.1]

Confused model

Less confident but correct model

Very confident but wrong model

Loss Function

A potential loss function in this case is the *sum of the absolute difference* of scores:

L( , P_1) = sum(f( , P_1) - [1.0, 0.0])

= sum([|-0.5| , |0.5|]) = 1

L( , P_1) = sum(f( , P_1) - [0.0, 1.0])

= sum([|0.1|, |-0.1|]) = 0.2

$$L = \sum_{i=1}^{n} |x_i - y_i|$$

Loss Function

A potential loss function in this case is the *sum of the absolute difference* of scores:

L( , P_1) = sum(f( , P_1) - [1.0, 0.0])

= sum([|-0.5| , |0.5|]) = 1

L( , P_1) = sum(f( , P_1) - [0.0, 1.0])

= sum([|0.1|, |-0.1|]) = 0.2

$$L = \sum_{i=1}^{\boxed{n}} |x_i - y_i|$$

Number of classes

Loss Function

A potential loss function in this case is the *sum of the absolute difference* of scores:

L( , P_1) = sum(f( , P_1) - [1.0, 0.0])

= sum([|-0.5| , |0.5|]) = 1

L( , P_1) = sum(f( , P_1) - [0.0, 1.0])

= sum([|0.1|, |-0.1|]) = 0.2

L( , P_2) = 0.6

L( , P_3) = 1.8

L( , P_2) = 0.6

L( , P_3) = 1.8

Loss Function

A potential loss function in this case is the *sum of the absolute difference* of scores:

L( , P_1) = sum(f( , P_1) - [1.0, 0.0])

= sum([|-0.5| , |0.5|]) = 1

L( , P_1) = sum(f( , P_1) - [0.0, 1.0])

= sum([|0.1|, |-0.1|]) = 0.2

L( , P_2) = 0.6

L( , P_3) = 1.8

L( , P_2) = 0.6

L( , P_3) = 1.8

Average loss for both cars

$L(P_1) = 0.6$ $L(P_2) = 0.6$ $L(P_3) = 1.8$

Loss Function

Average loss for both cars

$L(P_1) = 0.6$ $L(P_2) = 0.6$ $L(P_3) = 1.8$

A lower value of the loss indicates a better model

i.e. we are closer to the true values

In this case, P_1 and P_2 have the *lower value* of 0.6, so we know they are better than P_3. However, we also know that P_2 is better than P_1, and this implies our loss function is not very good right now!

Loss Function

Better loss function:

Mean Squared Error

Loss is equal to the sum of the square of the differences in the scores

$$L = \sum_{i=1}^{n} (x_i - y_i)^2$$

Loss Function

Better loss function:

Mean Squared Error

Loss is equal to the sum of the square of the differences in the scores

f( , P_1) = [0.5, 0.5]

f( , P_2) = [0.7, 0.3]

f( , P_3) = [0.1, 0.9]

f( , P_1) = [0.1, 0.9]

f( , P_2) = [0.3, 0.7]

f( , P_3) = [0.9, 0.1]

Loss Function

Better loss function:

Mean Squared Error

Loss is equal to the sum of the square of the differences in the scores

MSE( , P_1) = 0.50

MSE( , P_2) = 0.18

MSE( , P_3) = 1.62

MSE( , P_1) = 0.02

MSE( , P_2) = 0.18

MSE( , P_3) = 1.62

Loss Function

Better loss function:

Mean Squared Error

Loss is equal to the sum of the square of the differences in the scores

MSE( , P_1) = 0.50

MSE( , P_2) = 0.18

MSE( , P_3) = 1.62

MSE( , P_1) = 0.02

MSE( , P_2) = 0.18

MSE( , P_3) = 1.62

Average loss

$L(P_1) = 0.26$ $L(P_2) = 0.18$ $L(P_3) = 1.62$

Loss Function

Mean Squared Error works better, as it penalizes values that are further away from the true value

Loss Function

Many other choices for loss functions:

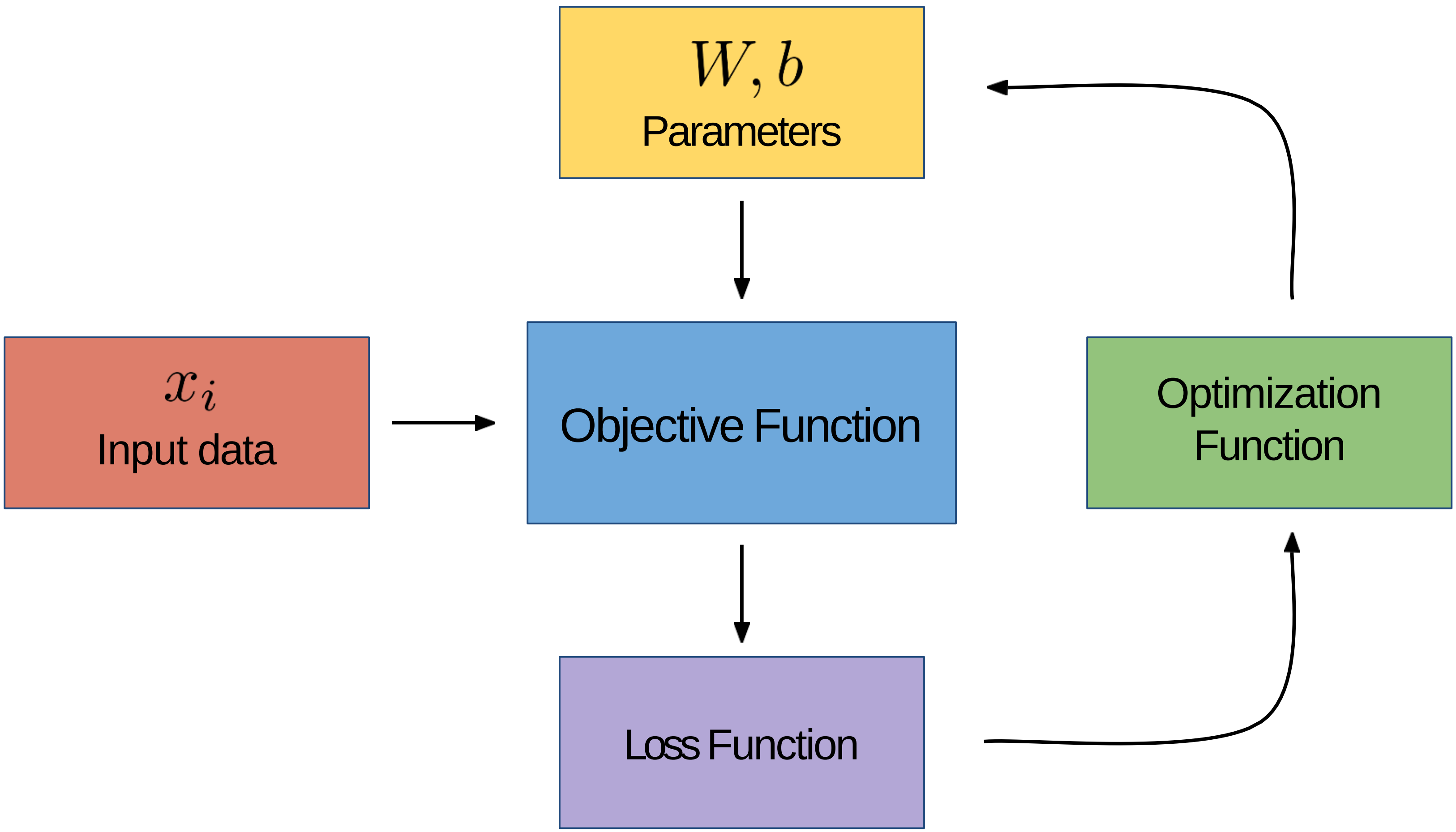
- Absolute Distance loss
- Hinge loss
- Logistic loss
- Cross Entropy loss

⋮

Loss Function

Loss function is also known as the *cost function* in some literature

Learning Lifecycle



Optimization

Now that we have a way of defining loss, we need a way to use it to improve our parameters

This process is called optimization - where our goal is to “minimize” the loss function, i.e. bring it as close to zero as possible

Optimization Exercise

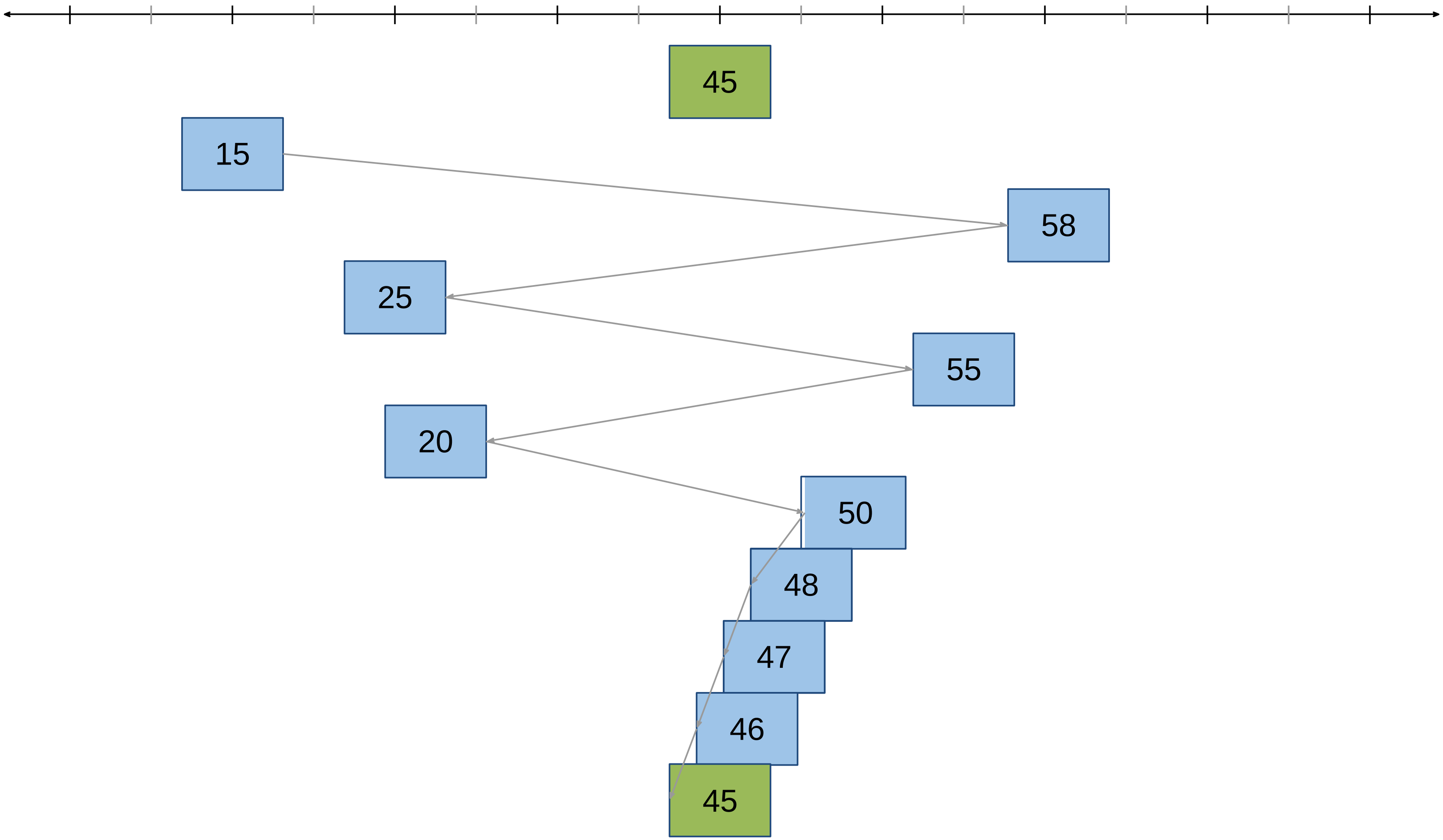
Find the value of x in the following equation:

$$x + 5 = ?$$

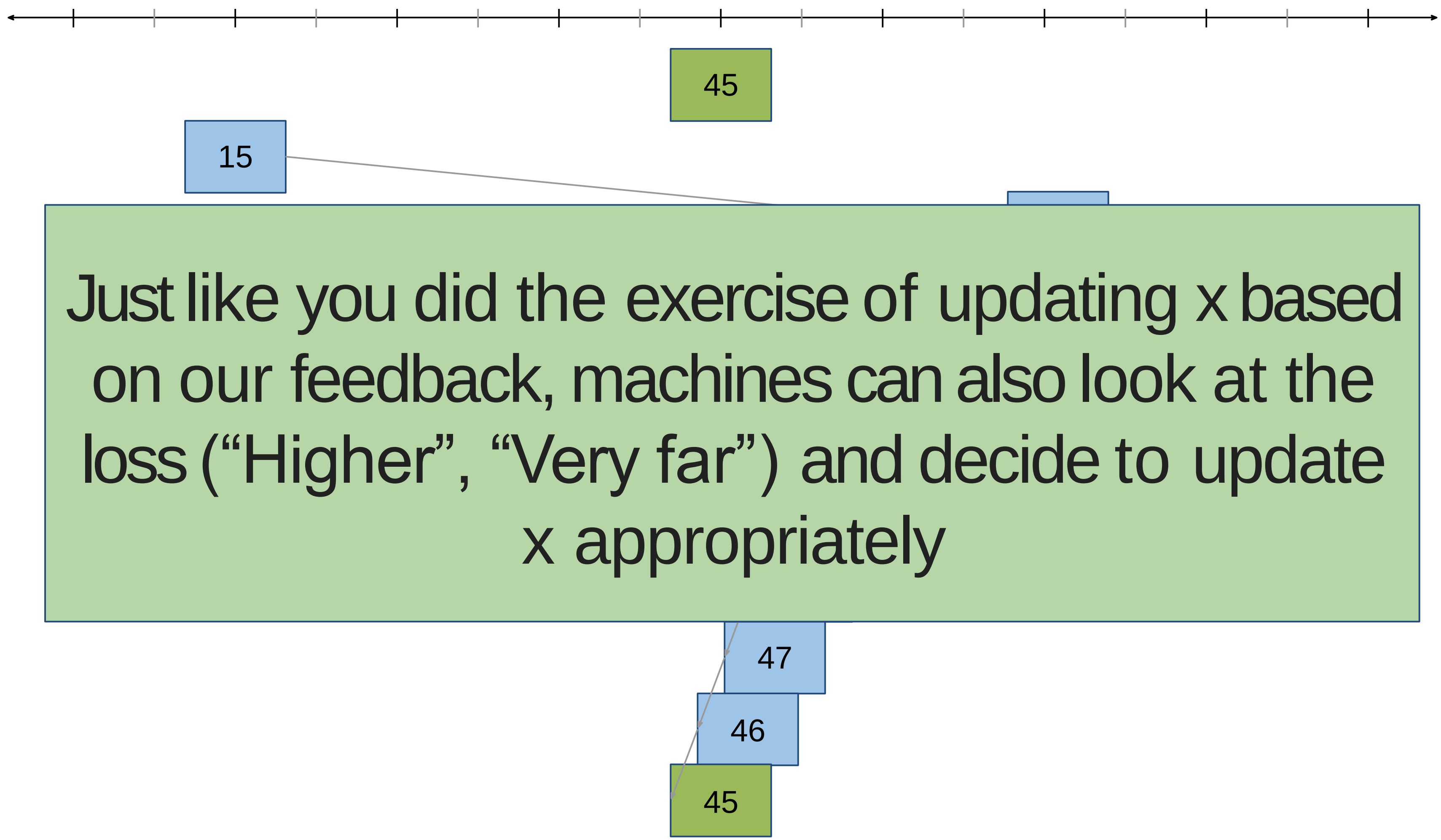
For every guess you will get the following hints:

Direction:	Higher	Lower		
Error:	Very far	Far	Close	Very close

Optimization Exercise

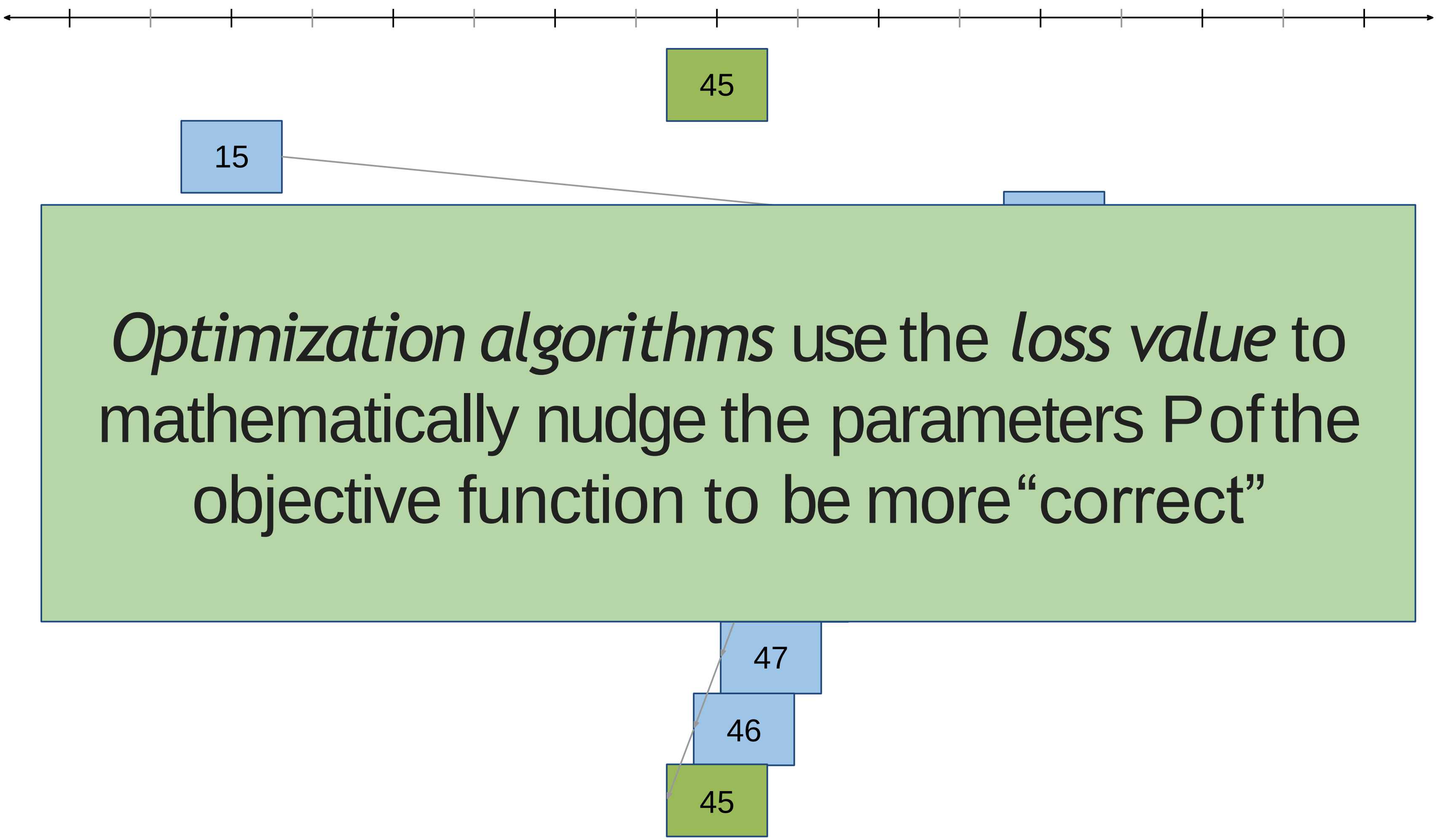


Optimization Exercise



Just like you did the exercise of updating x based on our feedback, machines can also look at the loss ("Higher", "Very far") and decide to update x appropriately

Optimization Exercise



Optimization algorithms use the *loss value* to mathematically nudge the parameters P of the objective function to be more “correct”

Optimization Exercise

What are some strategies you used to optimize x?

Optimization: Random Search

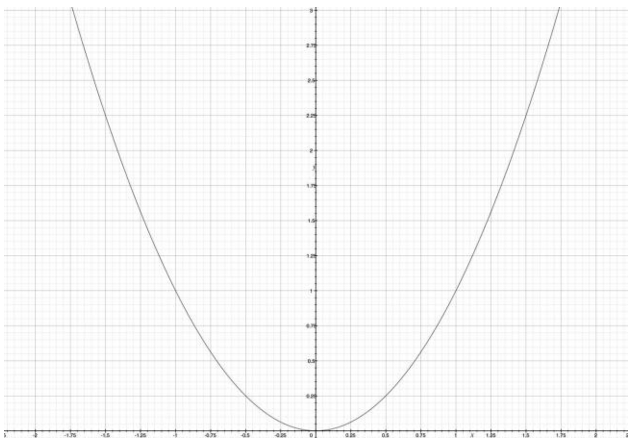
- Potential Solution: Guess randomly each time
- Pros:
  - Very simple
- Cons:
  - Not very efficient
  - *loss* value is unused
  - Potentially may never find a good solution

Optimization: Gradient Search

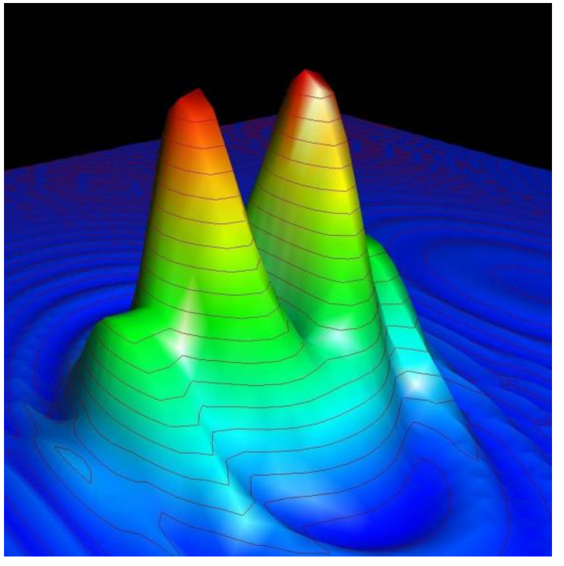
- Better Solution: Gradient based search

Optimization: Gradient Search

- Better Solution: Gradient based search
- Every function can be represented in space



$$y = x^2$$

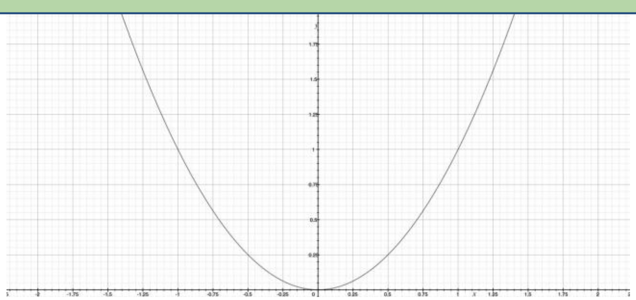


$$z = \frac{\sin(x^2 + 3y^2)}{0.1 + r^2} + (x^2 + 5y^2) \cdot \frac{\exp(1 - r^2)}{2}, r = \sqrt{x^2 + y^2}$$

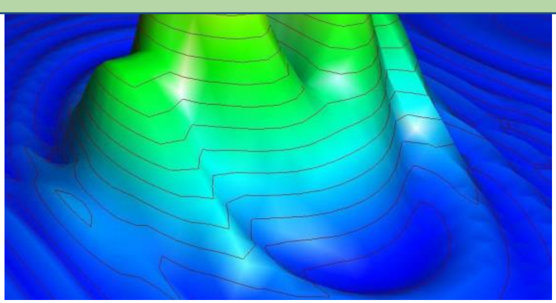
Optimization: Gradient Search

- Better Solution: Gradient based search
- Every function can be represented in space

Any *loss* function can also be represented in space



$$y = x^2$$



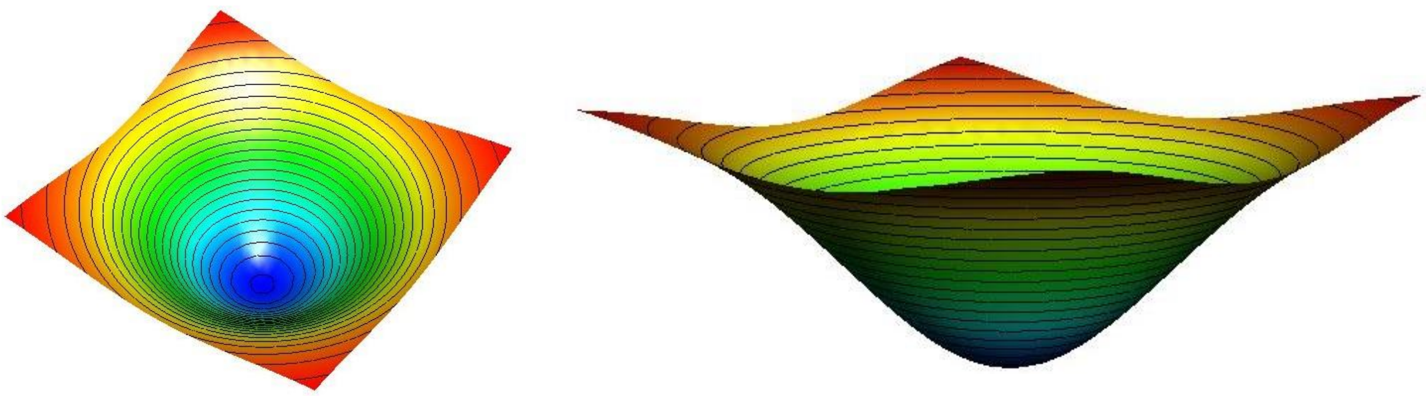
$$z = \frac{\sin(x^2 + 3y^2)}{0.1 + r^2} + (x^2 + 5y^2) \cdot \frac{\exp(1 - r^2)}{2}, r = \sqrt{x^2 + y^2}$$

Optimization: Gradient Search

- Better Solution: Gradient based search
- Our goal is to minimize the loss, i.e. find a set of parameters P such that the loss is close to zero

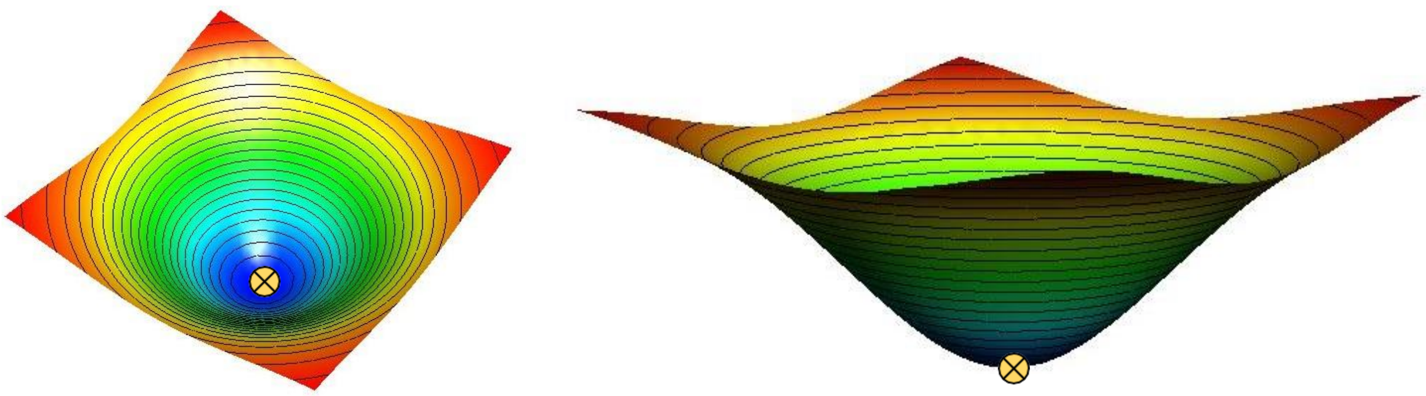
Optimization: Gradient Search

- Better Solution: Gradient based search
- Our goal is to minimize the loss, i.e. find a set of parameters P such that the loss is close to zero



Optimization: Gradient Search

- Better Solution: Gradient based search
- Our goal is to minimize the loss, i.e. find a set of parameters P such that the loss is close to zero



⊗ Minimum value of the loss function

Optimization: Gradient Search

Functions are just like terrain - they have mountains and valleys

We want to minimize loss,

i.e. go to the bottom of the terrain

Optimization: Gradient Search

Q: Imagine you are blindfolded on a mountain, how will you go to the bottom?

Optimization: Gradient Search

Q: Imagine you are blindfolded on a mountain, how will you go to the bottom?

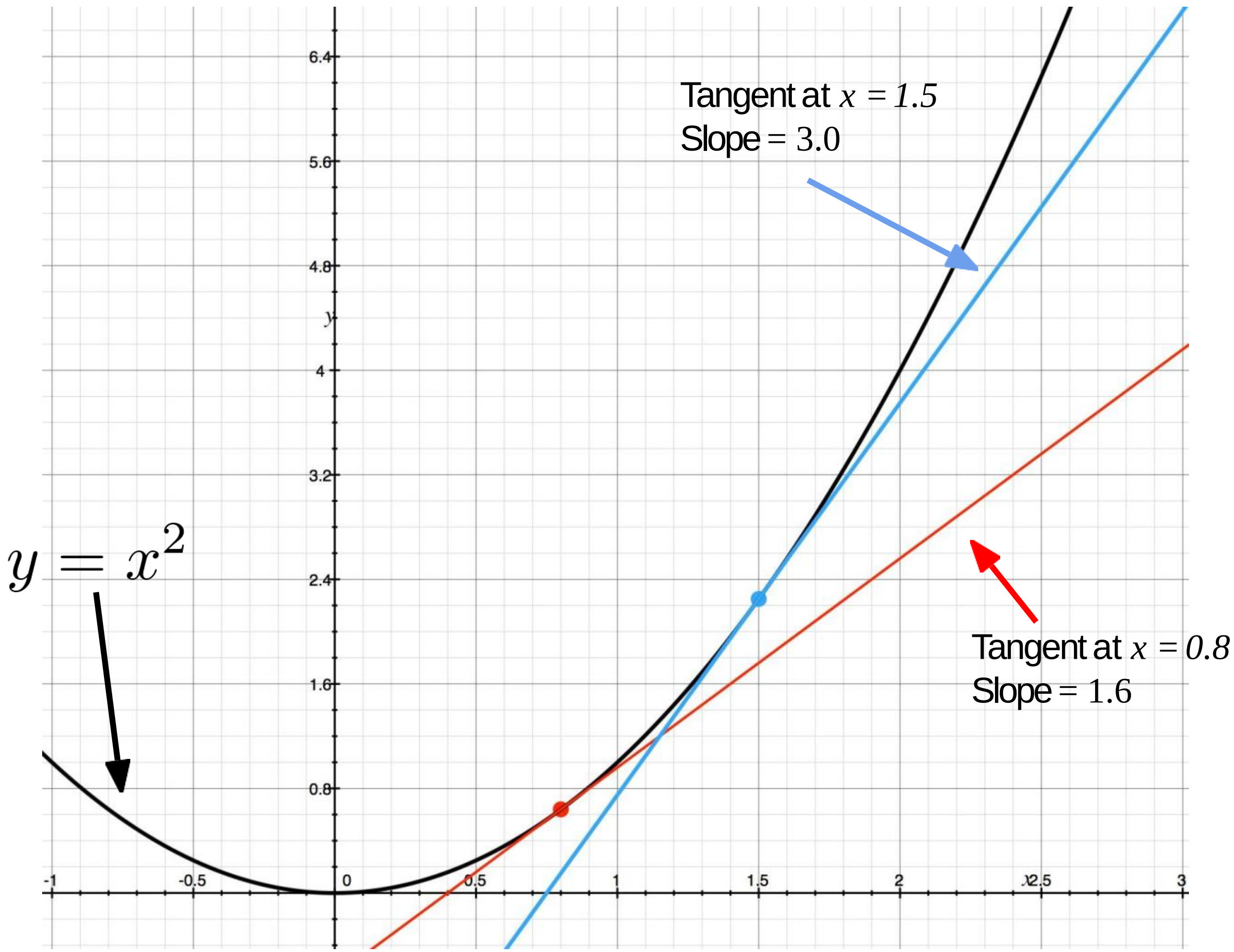
A: Sense the slope around you, and move in the direction where the slope points downwards

Optimization: Gradient Search

Concept of gradient == “your sense of slope” for the loss function

The gradient of a function is mathematically defined as the slope of the tangent i.e. slope at any given point on the function

Optimization: Gradient Search



Optimization: Gradient Search

Once we know the direction, we can move towards the minimum.

Are we done?

Optimization: Learning Rate

How far should we move?

The *step size* or *learning rate* defines how big a step we should take in the direction of the gradient

Optimization: Learning Rate

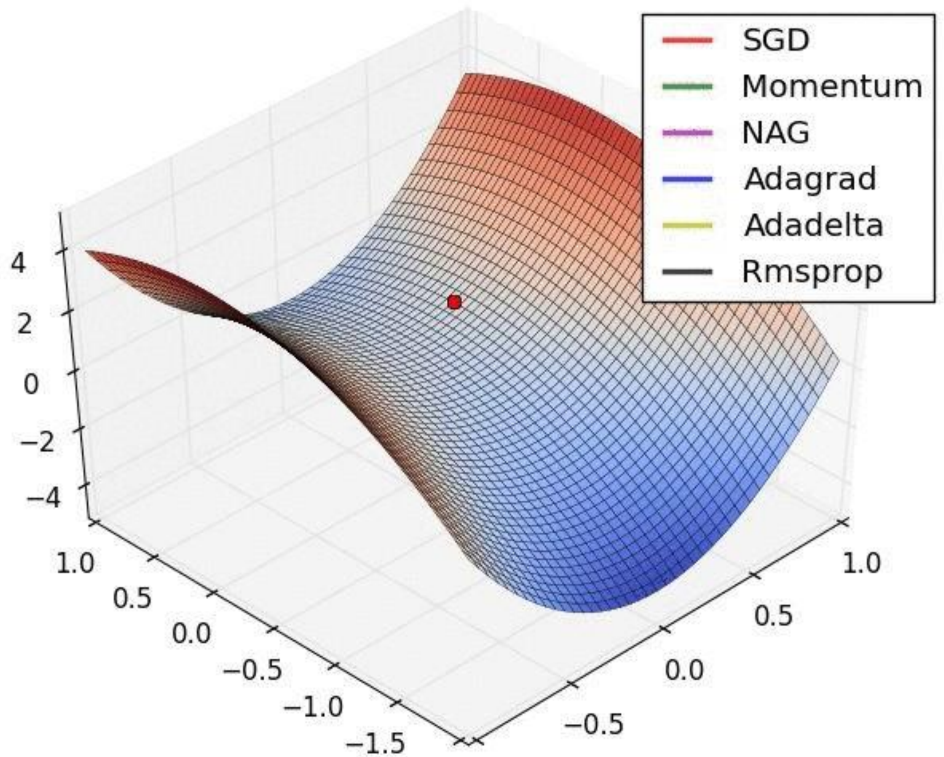
How far should we move?

The *step size* or *learning rate* defines how big a step we should take in the direction of the gradient

It must be well controlled - too small a step and it may take a long time to reach the bottom - too big a step and we may miss the minimum all together!

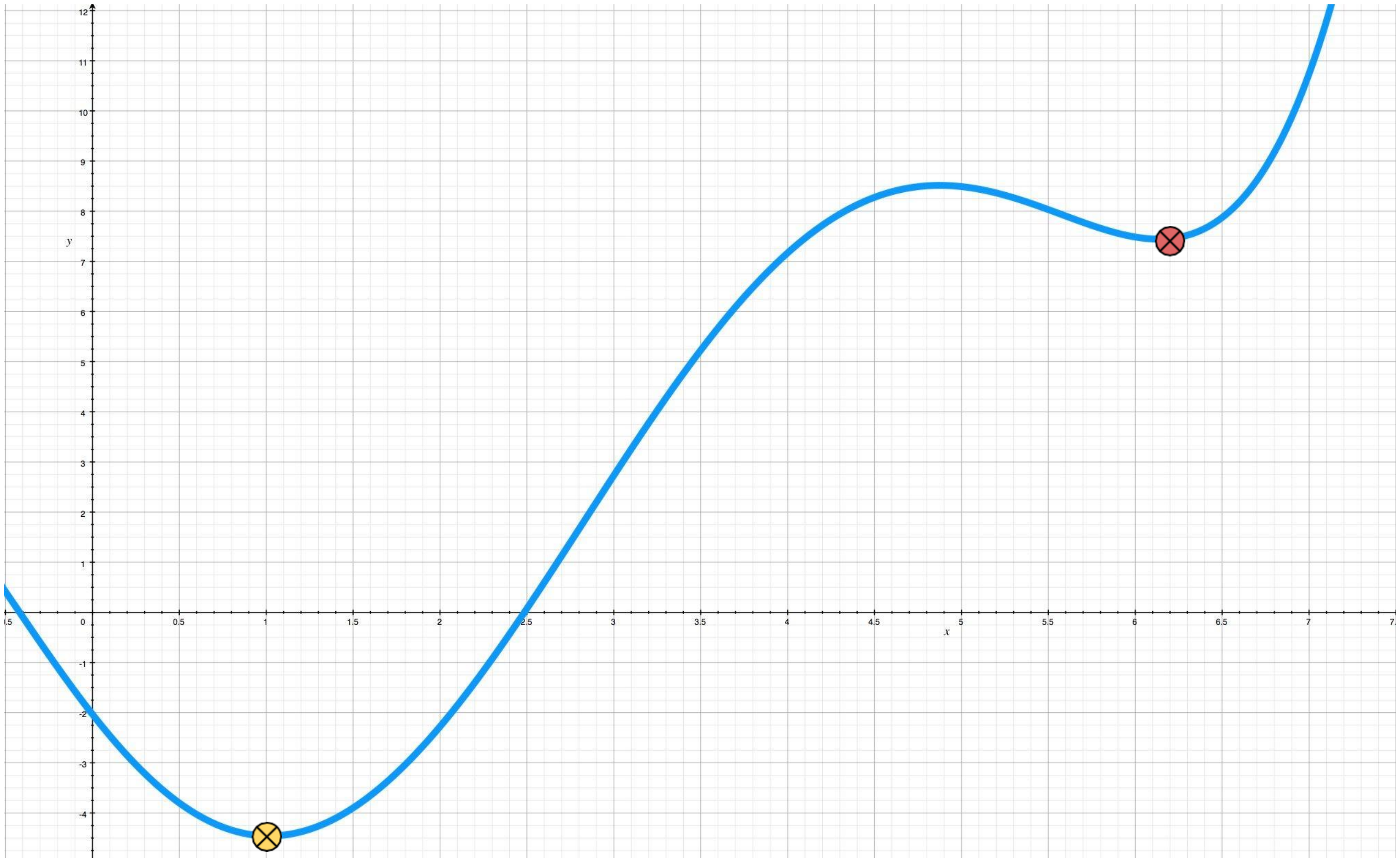
Optimization

Various optimization algorithms



Alec Radford (Reddit)

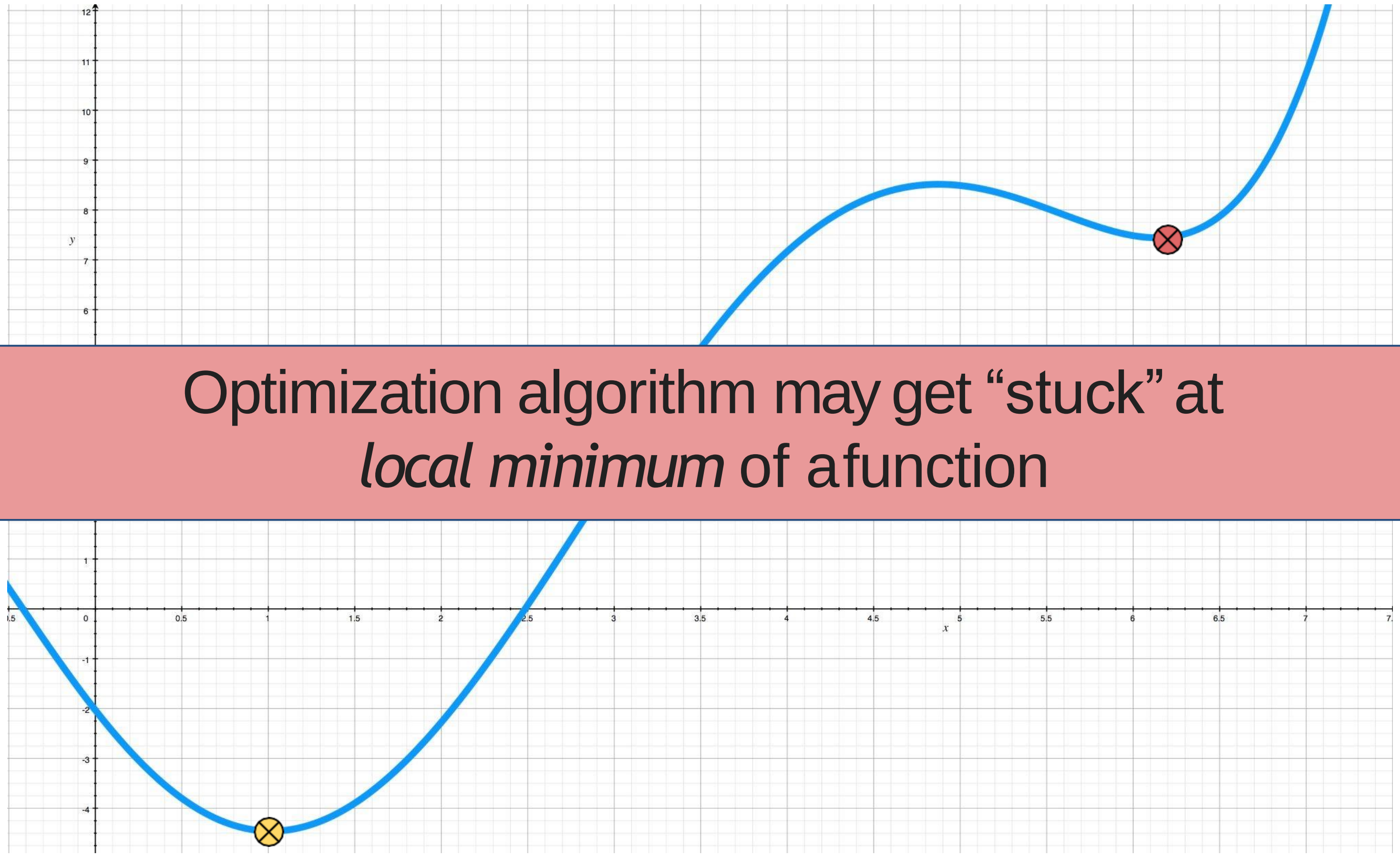
Local Minima



Minimum value of the function

Local minimum value of the function

Local Minima



Optimization algorithm may get "stuck" at
local minimum of a function

Minimum value of the function

Local minimum value of the function

Optimization

- How can we compute the slope of the function?
  - Compute gradients analytically
  - Backpropagation

Optimization

Let us compute the gradient of MSE analytically

$$\mathcal{L} = (f(x, W, b) - y)^2$$

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w_0} &= \frac{\partial}{\partial w_0}(f(x, W, b) - y)^2 \\
&= 2 \cdot (f(x, W, b) - y) \cdot \frac{\partial}{\partial w_0}(f(x, W, b) - y) \\
&= 2 \cdot (f(x, W, b) - y) \cdot \frac{\partial}{\partial w_0}(w_0 \cdot x_0 + w_1 \cdot x_1 + b - y) \\
&= 2 \cdot (f(x, W, b) - y) \cdot (x_0 + 0 + 0 - 0) \\
&= 2 \cdot x_0 \cdot (f(x, W, b) - y)
\end{aligned}$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = 2 \cdot x_1 \cdot (f(x, W, b) - y)$$

$$\frac{\partial \mathcal{L}}{\partial b} = 2 \cdot (f(x, W, b) - y)$$

Optimization

But what if the function was slightly more complicated:

$$f(x, w) = \left(\frac{e^{x \cdot w}}{x} \right)^3$$

$$\begin{aligned}
\frac{\partial}{\partial w} \left(\frac{e^{xw}}{xw} \right)^3 &= 3 \left(\frac{e^{xw}}{xw} \right)^2 \cdot \frac{\partial}{\partial w} \left(\frac{e^{xw}}{xw} \right) \\
&= 3 \left(\frac{e^{xw}}{xw} \right)^2 \cdot \frac{\frac{\partial}{\partial w} e^{xw} \cdot xw - e^{xw} \cdot \frac{\partial}{\partial w} xw}{x^2 w^2} \\
&= 3 \left(\frac{e^{xw}}{xw} \right)^2 \cdot \frac{e^{xw} x \cdot xw - e^{xw} \cdot x}{x^2 w^2} \\
&= 3 \left(\frac{e^{2xw}}{x^2 w^2} \right) \cdot \frac{e^{xw} \cdot x^2 w - e^{xw} \cdot x}{x^2 w^2} \\
&= 3 \left(\frac{e^{2xw}}{x^2 w^2} \right) \cdot \frac{e^{xw} \cdot xw - e^{xw}}{xw^2} \\
&= 3 \frac{e^{3xw} \cdot (xw - 1)}{x^3 w^4}
\end{aligned}$$

Optimization

But what if the function was slightly more complicated:

$$f(x, w) = \left(\frac{e^{x \cdot w}}{x}\right)^3$$

Analytical gradients become much more complicated and tedious to compute!

$$= 3\left(\frac{e^{2xw}}{x^2w^2}\right) \cdot \frac{e^{xw} \cdot xw - e^{xw}}{xw^2}$$

$$= 3\frac{e^{3xw} \cdot (xw - 1)}{x^3w^4}$$

Optimization

But what if the function was slightly more complicated:

$$f(x, w) = \left(\frac{e^{x \cdot w}}{x} \right)^3$$

Backpropagation to the rescue!

$$= 3 \left(\frac{e^{2xw}}{x^2 w^2} \right) \cdot \frac{e^{xw} \cdot xw - e^{xw}}{xw^2}$$

$$= 3 \frac{e^{3xw} \cdot (xw - 1)}{x^3 w^4}$$

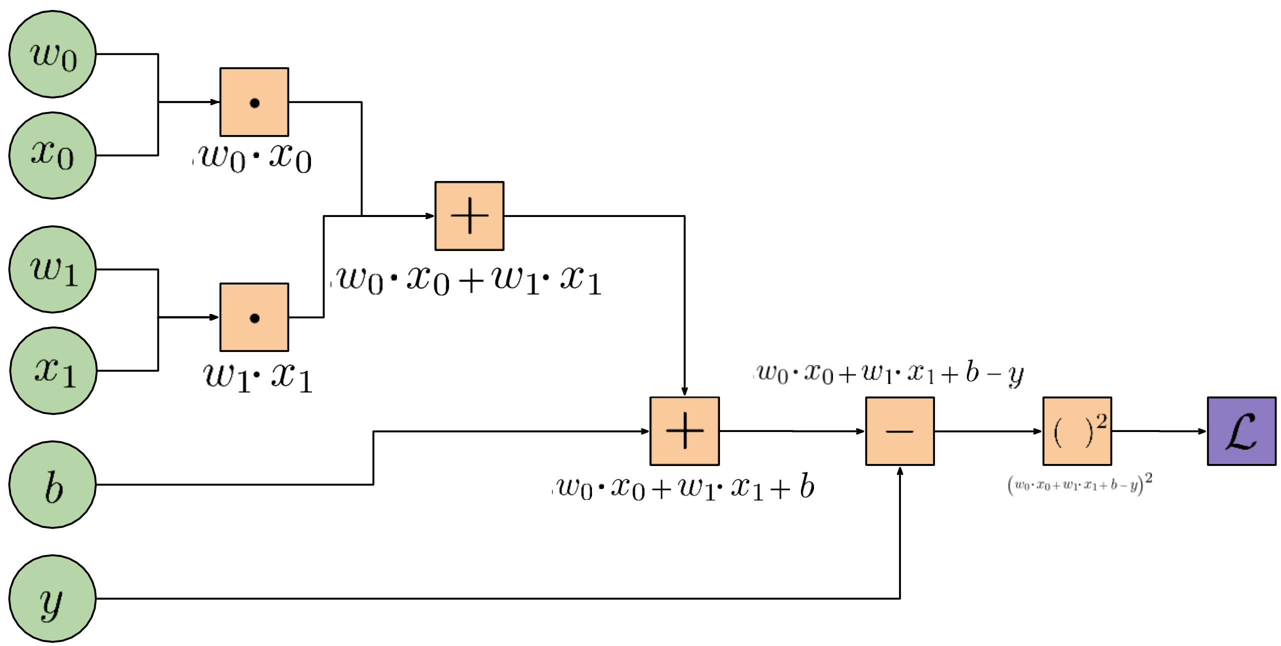
Backpropagation

Backpropagation is a technique to compute gradients of any function with respect to a variable using the concept of a *computation graph*

Backpropagation

Computation graph: Graphical way of describing any function:

$$\mathcal{L} = (f(x, W, b) - y)^2$$



Backpropagation

Intuition

- Divide the loss function into small differentiable steps
- Calculate the gradient of each small step and use chain rule to calculate the gradient of your input parameters

Optimization

To complete the picture, we can then use the gradients to update the parameters using gradient descent

Optimization

To complete the picture, we can then use the gradients to update the parameters using gradient descent

Recall: We want to take a “step” in the direction of the slope

Optimization

To complete the picture, we can then use the gradients to update the parameters using gradient descent

$$w_0 = w_0 - \eta \cdot \frac{\partial \mathcal{L}}{\partial w_0}$$

$$w_1 = w_1 - \eta \cdot \frac{\partial \mathcal{L}}{\partial w_1}$$

$$b = b - \eta \cdot \frac{\partial \mathcal{L}}{\partial b}$$

Optimization

To complete the picture, we can then use the gradients to update the parameters using gradient descent

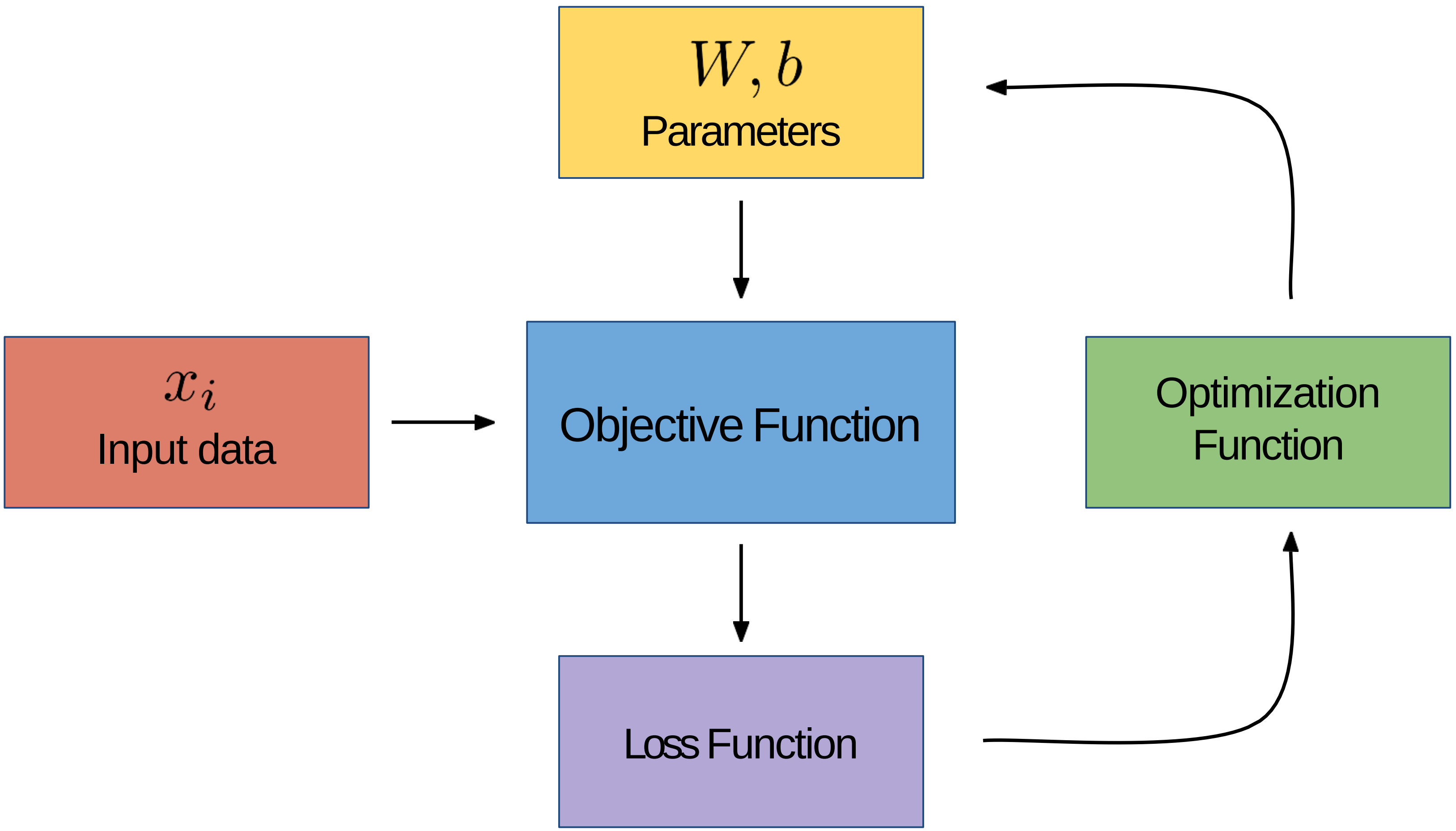
$$w_0 = w_0 - \boxed{\eta} \cdot \frac{\partial \mathcal{L}}{\partial w_0}$$

$$w_1 = w_1 - \boxed{\eta} \cdot \frac{\partial \mathcal{L}}{\partial w_1}$$

Step size
Learning rate

$$b = b - \boxed{\eta} \cdot \frac{\partial \mathcal{L}}{\partial b}$$

Learning Lifecycle



Multiclass Classification

Let us now look at another complete example of classification

Multiclass Classification

Recall that we have been using *linear regression* so far and making decisions based on the sign of the output

f( , W, b) = 1 Real Number

Output
If the number is less than 0, it is *accident prone*, else it is *not accident prone*

Multiclass Classification

In general, we design our function f such that we output one number per class:

f ( , W, b) = 2 Numbers

Outputs
The scores for the two classes - *accident prone* and *not accident prone*

Multiclass Classification

From now on, we will use this generalized technique, since it can be easily extended to more than two classes

f ( , W, b) = Numbers

Outputs
M scores for the
M classes

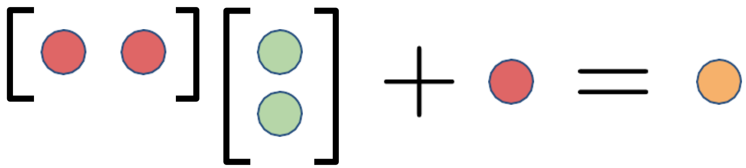
Multiclass Classification

From now on, we will use this generalized technique, since it can be easily extended to more than two classes

f ( , W, b) = Numbers (vector)

Everything else remains the same - the loss functions now operates on vectors instead of real numbers

Multiclass Classification



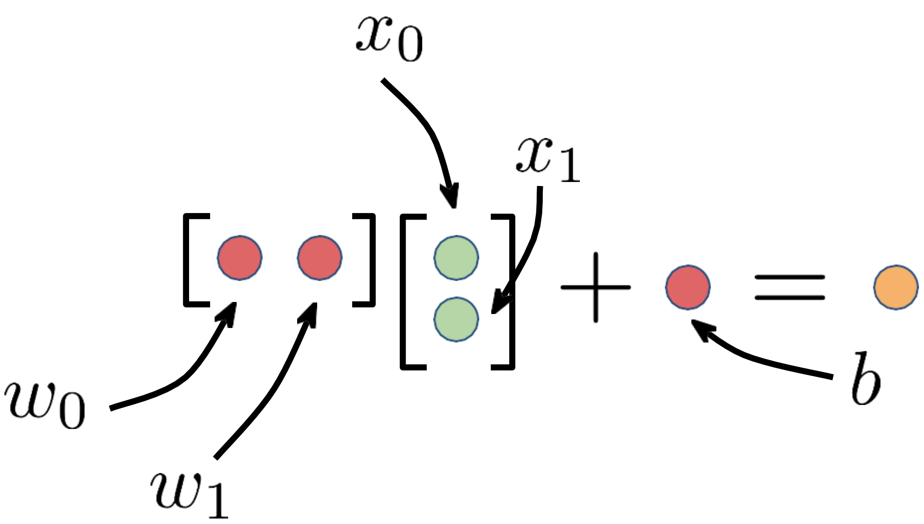
Vector

$$f(x, w, b) = \boxed{w} \cdot \boxed{x} + \boxed{b}$$

Real number

Linear Regression

Multiclass Classification



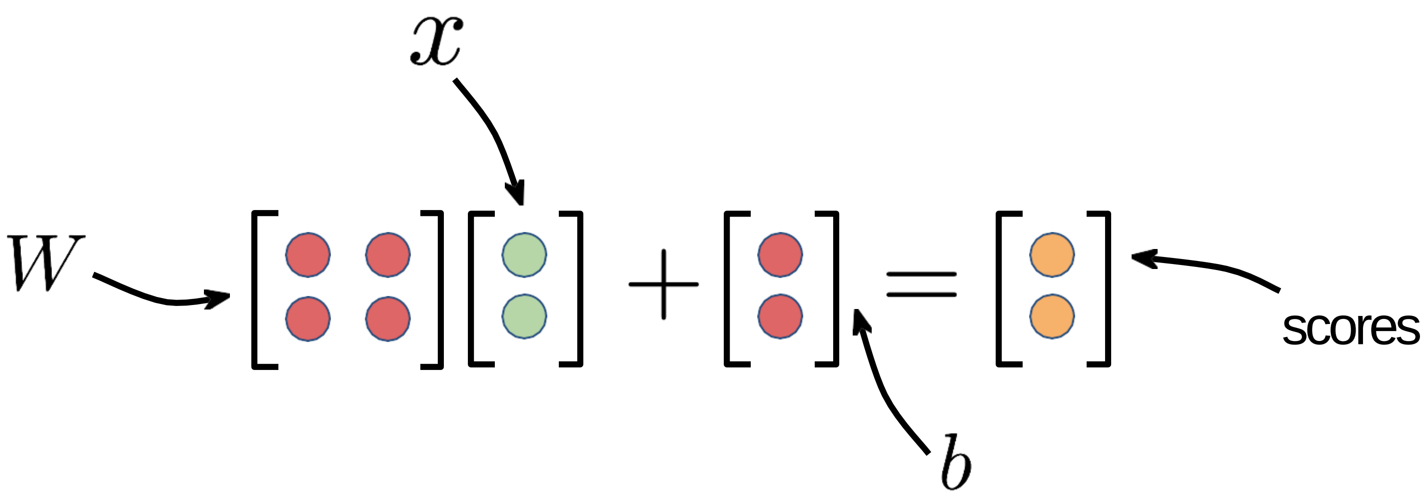
Vector

$$f(x, w, b) = \boxed{w} \cdot \boxed{x} + \boxed{b}$$

Real number

Linear Regression

Multiclass Classification



Matrix Vector

$$f(x, W, b) = \boxed{W} \cdot \boxed{x} + \boxed{b}$$

Multi-class Linear Classification

Multiclass Classification

Prediction

In regression:

f( , W, b) >= 0

In classification:

argmax(f( , W, b))

Pick the class with the highest score

Softmax function

Softmax

With the argmax function, our classifier has always output some “scores”, and we just pick whichever score is higher:

f( , W, b) = 2 Numbers

Outputs
The scores for the two classes - *accident prone* and *not accident prone*

Softmax

However, these scores are not *interpretable*. Their absolute values don't give us any insight, we can only compare them relatively

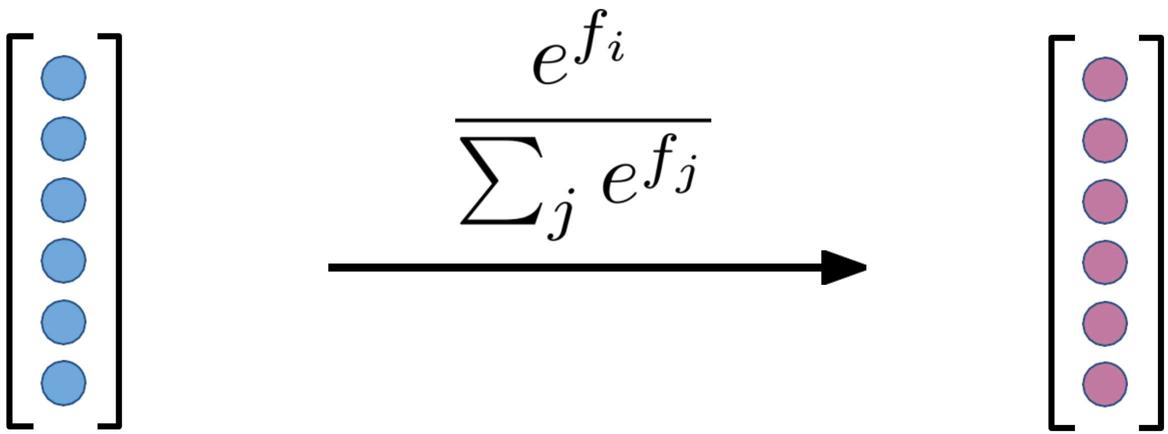
f ( , W, b) = 2 Numbers

Outputs

The scores for the two classes - *accident prone* and *not accident prone*

Softmax

The softmax function helps us transform these values into probability distributions:



$$\frac{e^{f_i}}{\sum_j e^{f_j}}$$

Scores from the classifier

f

Scores as a probability distribution

Softmax

The softmax function helps us transform these values into probability distributions:

$$\begin{bmatrix} -1.85 \\ 0.42 \\ 0.15 \end{bmatrix} \longrightarrow \boxed{\text{Softmax}} \longrightarrow \begin{bmatrix} 0.06 \\ 0.54 \\ 0.40 \end{bmatrix}$$

$$\frac{e^{f_i}}{\sum_j e^{f_j}}$$

Softmax

The softmax function helps us transform these values into probability distributions:

each output can be treated as the probability of that class

$$\begin{bmatrix} -1.85 \\ 0.42 \\ 0.15 \end{bmatrix} \longrightarrow \boxed{\text{Softmax}} \longrightarrow \begin{bmatrix} \boxed{\begin{matrix} 0.06 \\ 0.54 \\ 0.40 \end{matrix}} \end{bmatrix}$$

$$\frac{e^{f_i}}{\sum_j e^{f_j}}$$

scores sum to one

Softmax

The softmax function helps us transform these values into probability distributions:

each output can be treated as the probability of that class

$$\begin{bmatrix}-1.85\\ 0.42\\ 0.15\end{bmatrix} \longrightarrow \boxed{\text{Softmax}} \longrightarrow \begin{bmatrix}\boxed{\begin{matrix}0.06\\ 0.54\\ 0.40\end{matrix}}\end{bmatrix}$$

$$\frac{e^{f_i}}{\sum_j e^{f_j}}$$

scores sum to one

The Softmax function also acts as a *normalizer*, i.e. we can now compare scores from different models and examples directly

Cross Entropy Loss

Cross Entropy Loss

Recall MSE:

Mean Squared Error

$$L = \sum_{i=1}^{n} (f_i - y_i)^2$$

Cross Entropy Loss

Recall MSE:

Mean Squared Error

We saw that MSE is better than just taking the absolute difference:

$$\boxed{L = \sum_{i=1}^{n} |f_i - y_i|}$$

Cross Entropy Loss

Recall MSE:

Mean Squared Error

In practice, we use *Cross Entropy loss*, which generally performs better for more complex models.

Cross Entropy Loss

$$H_y(f) = -\sum_i y_i \log(f_i)$$

Here, y represents the true probability distribution (so $y_i = 1$ for the correct class i, and 0 otherwise)

f_i represents the score of class i from our classifier

Cross Entropy Loss

$$H_y(f) = -\sum_i y_i \log(f_i)$$
$$= -y_c \log(f_c)$$

Simplifying for our case,
if c is the correct class, then $y_c = 1$, and all other y_i's are 0
Therefore, we only have one element left from the summation

Cross Entropy Loss

$$
\begin{aligned}
H_y(f) &= -\sum_i y_i \log(f_i) \\
&= -y_c \log(f_c) \\
&= -\log(f_c)
\end{aligned}
$$

Cross Entropy Loss

Mean Squared Error

$$L = \sum_{i=1}^{n} (f_i - y_i)^2$$

Cross Entropy

$$L = -\log(f_c)$$

Cross Entropy Loss

Why cross entropy?

Consider three people, Person1 is a *Democrat*, Person2 is a *Republican* and Person3 is *Other*. We have two models to classify these people:

	S_{Other}	$S_{Republican}$	$S_{Democrat}$
Person1	0.3	0.3	**0.4**
Person2	0.3	**0.4**	0.3
Person3	0.1	0.2	**0.7**

Model 1

	S_{Other}	$S_{Republican}$	$S_{Democrat}$
Person1	0.1	0.2	**0.7**
Person2	0.1	**0.7**	0.2
Person3	0.3	**0.4**	0.3

Model 2

Cross Entropy Loss

	s_{Other}	$s_{Republican}$	$s_{Democrat}$
Person1	0.3	0.3	**0.4**
Person2	0.3	**0.4**	0.3
Person3	0.1	0.2	**0.7**

Model 1

	s_{Other}	$s_{Republican}$	$s_{Democrat}$
Person1	0.1	0.2	**0.7**
Person2	0.1	**0.7**	0.2
Person3	0.3	**0.4**	0.3

Model 2

Both models misclassify *Person3*, but is one model better than the other?

Cross Entropy Loss

	S_{Other}	$S_{Republican}$	$S_{Democrat}$
Person1	0.3	0.3	**0.4**
Person2	0.3	**0.4**	0.3
Person3	0.1	0.2	**0.7**

Model 1

	S_{Other}	$S_{Republican}$	$S_{Democrat}$
Person1	0.1	0.2	**0.7**
Person2	0.1	**0.7**	0.2
Person3	0.3	**0.4**	0.3

Model 2

Model 2 is better, since it classifies *Person1* and *Person2* with higher scores on the correct class, and mis-classifies *Person3* with a smaller error in the scores

Cross Entropy Loss

	S_{Other}	$S_{Republican}$	$S_{Democrat}$
Person1	0.3	0.3	**0.4**
Person2	0.3	**0.4**	0.3
Person3	0.1	0.2	**0.7**

Model 1

	S_{Other}	$S_{Republican}$	$S_{Democrat}$
Person1	0.1	0.2	**0.7**
Person2	0.1	**0.7**	0.2
Person3	0.3	**0.4**	0.3

Model 2

Person1: 0.54

Person2: 0.54

Person3: 1.34

Model 1 Average: 0.81

Person1: 0.14

Person2: 0.14

Person3: 0.74

Model 2 Average: 0.34

Mean Squared Error

Cross Entropy Loss

	S_{Other}	$S_{Republican}$	$S_{Democrat}$
Person1	0.3	0.3	**0.4**
Person2	0.3	**0.4**	0.3
Person3	0.1	0.2	**0.7**

Model 1

	S_{Other}	$S_{Republican}$	$S_{Democrat}$
Person1	0.1	0.2	**0.7**
Person2	0.1	**0.7**	0.2
Person3	0.3	**0.4**	0.3

Model 2

Person1: -log(0.4) = 0.92

Person2: -log(0.4) = 0.92

Person3: -log(0.1) = 2.30

Model 1 Average: 1.38

Person1: 0.36

Person2: 0.36

Person3: 1.20

Model 2 Average: 0.64

Cross Entropy

Cross Entropy Loss

	S_{Other}	$S_{Republican}$	$S_{Democrat}$
Person1	0.3	0.3	**0.4**
Person2	0.3	**0.4**	0.3
Person3	0.1	0.2	**0.7**

Model 1

	S_{Other}	$S_{Republican}$	$S_{Democrat}$
Person1	0.1	0.2	**0.7**
Person2	0.1	**0.7**	0.2
Person3	0.3	**0.4**	0.3

Model 2

Mean Squared Error

Model 1 Average: 0.81

Model 2 Average: 0.34

Cross Entropy

Model 1 Average: 1.38

Model 2 Average: 0.64

Cross Entropy Loss

Mean Squared Error

Model 1 Average: 0.81 **Model 2 Average:** 0.34

Cross Entropy

Model 1 Average: 1.38 **Model 2 Average:** 0.64

Cross Entropy Loss difference between the two models is greater than the Mean Squared Error!

Cross Entropy Loss

In general, *Mean Squared Error* penalizes incorrect predictions much more than *Cross Entropy*

Cross Entropy Loss

A more principled reason arises from the underlying mathematics of MSE and Cross Entropy

MSE causes the gradients to become very small as the network scores become better, so learning slows down!

Cross Entropy and Softmax

Cross Entropy and Softmax

Cross Entropy is mathematically defined to compare two probability distributions

Cross Entropy and Softmax

Cross Entropy is mathematically defined to compare two probability distributions

Our ground truth is already represented as a probability distribution (with all the probability mass on the correct class)

$$y = \begin{bmatrix} 0.00 \\ 1.00 \\ 0.00 \end{bmatrix}$$

Cross Entropy and Softmax

Cross Entropy is mathematically defined to compare two probability distributions

However, the scores directly from a linear classifier do not form any such distribution:

$$f = \begin{bmatrix} -1.85 \\ 0.42 \\ 0.15 \end{bmatrix}$$

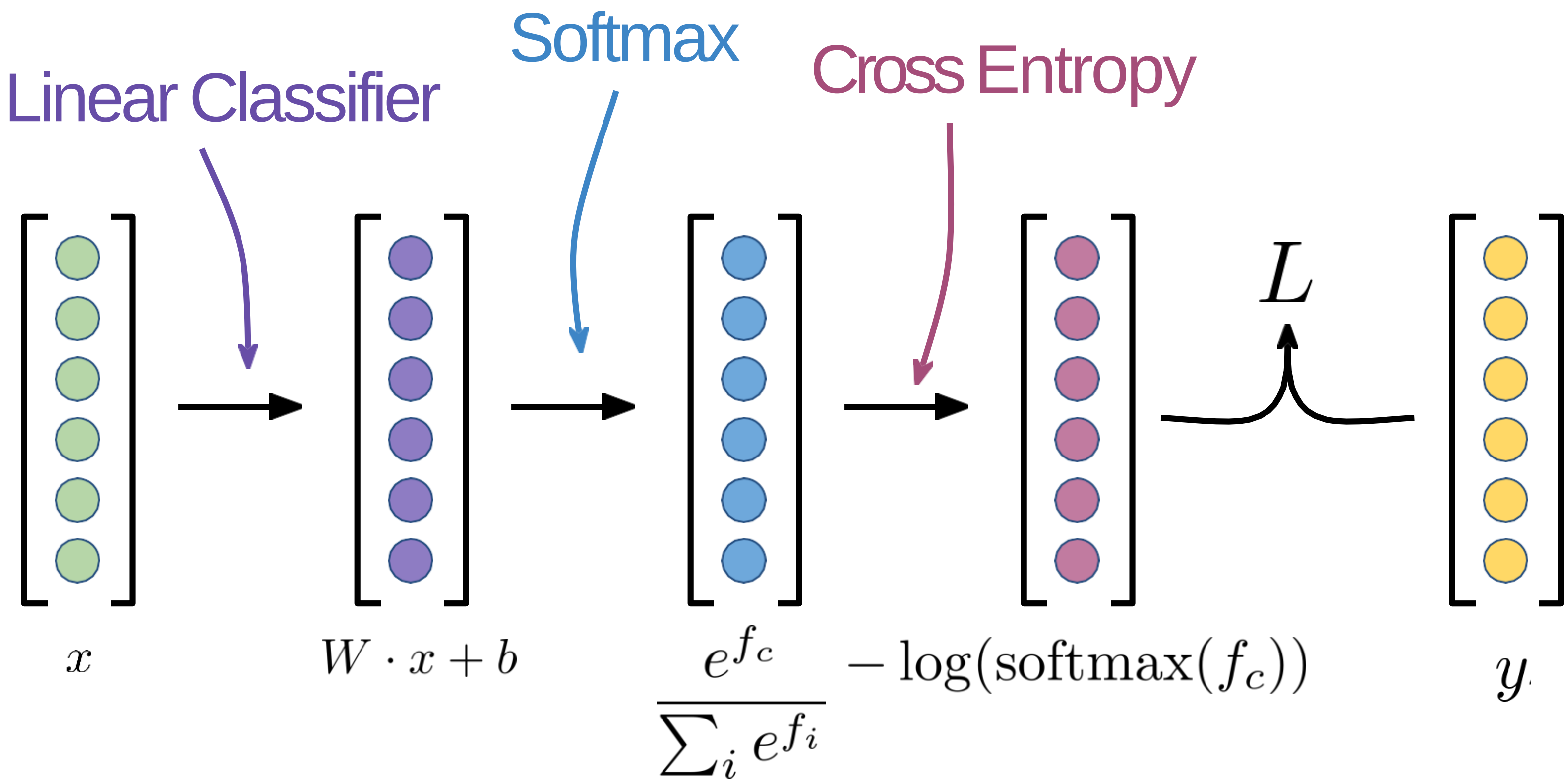
Cross Entropy and Softmax

Cross Entropy is mathematically defined to compare two probability distributions

Solution: Use softmax!

$$\text{softmax}(f) = \begin{bmatrix} 0.06 \\ 0.54 \\ 0.40 \end{bmatrix}$$

Putting it all together



Summary

- Classification
- Objective function
- Loss function
  - sum of absolute differences
  - mean squared error
- Optimization
  - random search
  - gradient search
  - backpropagation