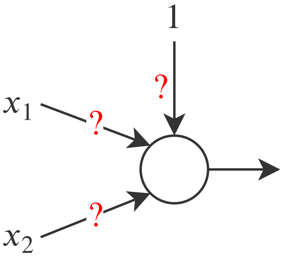
Deep Learning

Syed Irtaza Muzaffar

Training a Perceptron

What is training?



AND

x_1	x_2	t
0	0	0
0	1	0
1	0	0
1	1	1

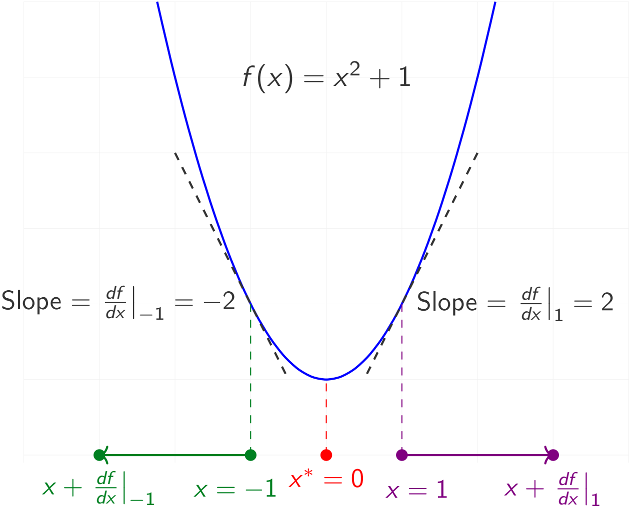
OR

x_1	x_2	t
0	0	0
0	1	1
1	0	1
1	1	1

Find weights $\mathbf{w}$ and bias b that maps input vectors $\mathbf{x}$ to given targets t.

- A perceptron is a function $f : \mathbf{x} \rightarrow t$ with parameters $\mathbf{w}, b$.
- Formally written as $f(\mathbf{x}; \mathbf{w}, b)$.
- Training corresponds to *minimizing a loss function*.
- So let's take a detour to understand function minimization.

Minimization

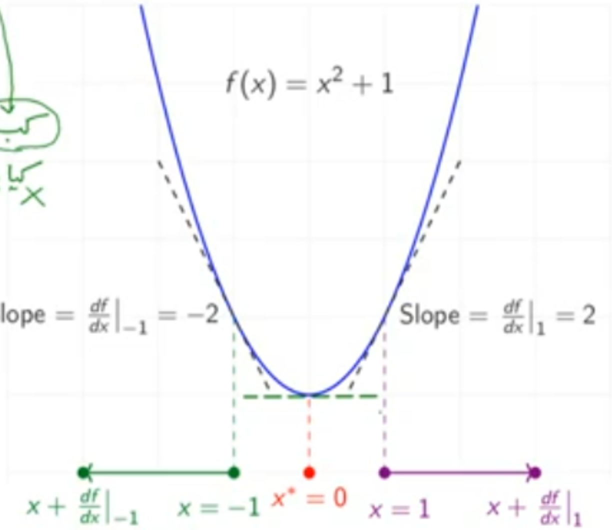


What is the slope/derivative/gradient at the minimizer $x^* = 0$?

Minimization

$x^* = 0$ کب؟

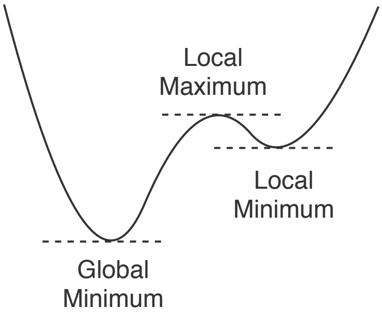
$f(0) = 0^2 + 1 = 1$ کیا؟



rate of change/

What is the slope/derivative/gradient at the minimizer $x^* = 0$?

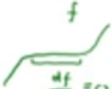
Minimization
Local vs. Global Minima



- *Stationary point*: where derivative is 0.
- A stationary point can be a minimum or a maximum.
- A minimum can be local or global. Same for maximum.

Minimization
Local vs. Global Minima

- *Stationary point*: where derivative is 0.
- A stationary point can be a minimum or a maximum. or a saddle point.
- A minimum can be local or global. Same for maximum.



$\frac{df}{dx} = 0$

saddle points

Gradient Descent

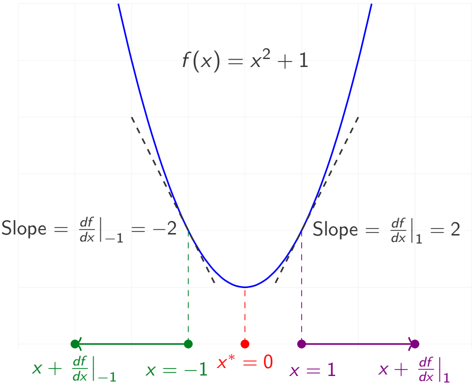
- Gradient is the direction, in input space, of maximum rate of increase of a function.

$$f\left(x + \frac{df}{dx}\right) \geq f(x)$$

- To minimize function $f(x)$ with respect to x, move in negative gradient direction.

$$x^{\text{new}} = x^{\text{old}} - \left.\frac{df}{dx}\right|_{x^{\text{old}}}$$

- Try it! Start from $x^{\text{old}} = -1$. Do you notice any problem?



Gradient Descent

- Gradient is the direction, in input space, of maximum rate of increase of a function.

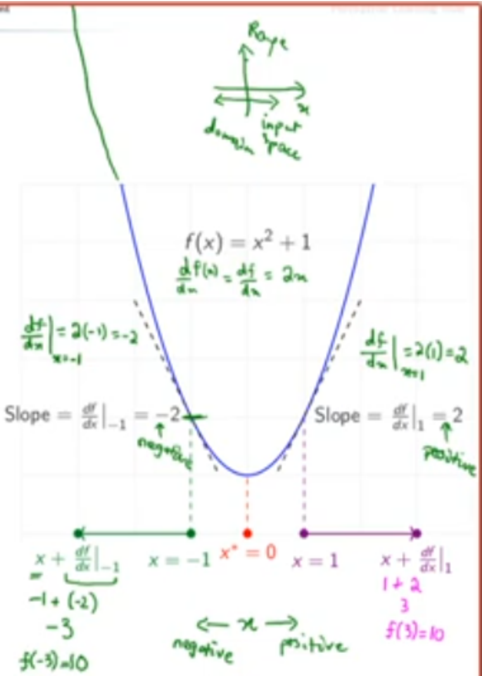
$$f\left(x + \frac{df}{dx}\right) \geq f(x)$$

- To minimize function $f(x)$ with respect to x, move in negative gradient direction.

$f(x^{new}) \leq f(x^{old})$

$$x^{new} = x^{old} - \left.\frac{df}{dx}\right|_{x^{old}}$$

- Try it! Start from $x^{old} = -1$. Do you notice any problem?



Minimization via Gradient Descent

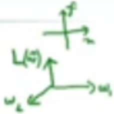
- To minimize loss $L(\mathbf{w})$ with respect to weights $\mathbf{w}$

$$\mathbf{w}^{\text{new}} = \mathbf{w}^{\text{old}} - \eta \nabla_{\mathbf{w}} L(\mathbf{w})$$

  where scalar $\eta > 0$ controls the step-size. It is called the *learning rate*.
- Also known as *gradient descent*.

  Repeated applications of gradient descent find the closest local minimum.

Minimization via Gradient Descent



$f(x)$
$f(\vec{x}; \vec{w}, b)$ } $\frac{df}{dx}$ $\quad \vec{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ $f(\vec{x})$

$L(\vec{w}, b)$ $\quad$ nabla $\rightarrow \nabla_{\vec{x}} f(\vec{x}) = \begin{bmatrix} \frac{df}{dx_1} \\ \frac{df}{dx_2} \end{bmatrix}$

$L(\vec{w}^*, b^*)$ is minimum.

$\nabla_{\vec{w}} L(\vec{w}) = \begin{bmatrix} \frac{dL}{dw_1} \\ \frac{dL}{dw_2} \end{bmatrix} \leftarrow$ gradient vector

- To minimize loss $L(\mathbf{w})$ with respect to weights $\mathbf{w}$

$$\mathbf{w}^{\text{new}} = \mathbf{w}^{\text{old}} - \underbrace{\eta}_{\text{eta}} \nabla_{\mathbf{w}} L(\mathbf{w})$$

where scalar $\eta > 0$ controls the step-size. It is called the *learning rate.*

- Also known as *gradient descent*.

> Repeated applications of gradient descent find the closest local minimum.

Gradient Descent

1. Initialize $\mathbf{w}^{\text{old}}$ randomly.
2. do
   2.1 $\mathbf{w}^{\text{new}} \leftarrow \mathbf{w}^{\text{old}} - \eta \, \nabla_{\mathbf{w}} L(\mathbf{w})\big|_{\mathbf{w}^{\text{old}}}$
3. while $\left| L(\mathbf{w}^{\text{new}}) - L(\mathbf{w}^{\text{old}}) \right| > \epsilon$

- Learning rate η needs to be reduced gradually to ensure *convergence to a local minimum*.
- If η is too large, the algorithm can *overshoot* the local minimum and keep doing that indefinitely *(oscillation)*.
- If η is too small, the algorithm will take too long to reach a local minimum.

Gradient Descent

1. Initialize $\mathbf{w}^{\text{old}}$ randomly.
2. do

   2.1 $\mathbf{w}^{\text{new}} \leftarrow \mathbf{w}^{\text{old}} - \overset{>0}{\eta} \nabla_{\mathbf{w}} L(\mathbf{w})\big|_{\mathbf{w}^{\text{old}}}$
3. while $\left| L(\mathbf{w}^{\text{new}}) - L(\mathbf{w}^{\text{old}}) \right| > \epsilon$ 1e^{-3}, 1e^{-4},

$\eta=1$

$\eta<1$

- Learning rate η needs to be reduced gradually to ensure *convergence to a local minimum.*
- If η is too large, the algorithm can *overshoot* the local minimum and keep doing that indefinitely *(oscillation)*.
- If η is too small, the algorithm will take too long to reach a local minimum.

Gradient Descent

- Different types of gradient descent:

Batch	$\mathbf{w}^{\text{new}} = \mathbf{w}^{\text{old}} - \eta \nabla_{\mathbf{w}} L$
Sequential	$\mathbf{w}^{\text{new}} = \mathbf{w}^{\text{old}} - \eta \nabla_{\mathbf{w}} L_n$
Stochastic	same as sequential but n is chosen randomly
Mini-batches	$\mathbf{w}^{\text{new}} = \mathbf{w}^{\text{old}} - \eta \nabla_{\mathbf{w}} L_{\mathcal{B}}$

- Most common variations are stochastic gradient descent (SGD) and SGD using mini-batches.

Gradient Descent

AND

x_1	x_2	t
0	0	0
0	1	0
1	0	0
1	1	1

$L(\vec{w}) = \sum_{n=1}^{4} L_n(\vec{w})$

Decompose

- Different types of gradient descent:

Batch GD	$\mathbf{w}^{new} = \mathbf{w}^{old} - \eta \nabla_{\mathbf{w}} L$	$\longrightarrow \nabla_{\vec{w}} L(\vec{w}) = \sum_{n=1}^{4} \nabla_{\vec{w}} L_n(\vec{w})$
Sequential GD	$\mathbf{w}^{new} = \mathbf{w}^{old} - \eta \nabla_{\mathbf{w}} L_n$	1, 2, 3, 4
Stochastic GD	same as sequential but n is chosen randomly	
Mini-batches	$\mathbf{w}^{new} = \mathbf{w}^{old} - \eta \nabla_{\mathbf{w}} L_{\mathcal{B}}$	$\mathcal{B} = \{2, 4\}$ $\mathcal{B} = \{1, 4\}$

- Most common variations are stochastic gradient descent (SGD) and SGD using mini-batches.

Perceptron Algorithm
Two-class Classification

- Let $(\mathbf{x}_n, t_n)$ be the n-th training example pair.
- Mathematical convenience: replace Boolean target (0/1) by binary target (−1/1).

AND

x_1	x_2	t
0	0	−1
0	1	−1
1	0	−1
1	1	1

OR

x_1	x_2	t
0	0	−1
0	1	1
1	0	1
1	1	1

- Do the same for perceptron output.

$$y(\mathbf{x}_n) = \begin{cases} 1 & \text{if } \mathbf{w}^T\mathbf{x}_n + b \geq 0 \\ -1 & \text{if } \mathbf{w}^T\mathbf{x}_n + b < 0 \end{cases}$$

Perceptron Algorithm
Two-class Classification

input vector target

- Let $(\mathbf{x}_n, t_n)$ be the n-th training example pair.
- Mathematical convenience: replace Boolean target (0/1) by binary target (−1/1).

AND			OR		
x_1	x_2	t	x_1	x_2	t
0	0	−1	0	0	−1
0	1	−1	0	1	1
1	0	−1	1	0	1
1	1	1	1	1	1

- Do the same for perceptron output.

$$y(\mathbf{x}_n) = \begin{cases} 1 & \text{if } \mathbf{w}^T\mathbf{x}_n + b \geq 0 \\ -1 & \text{if } \mathbf{w}^T\mathbf{x}_n + b < 0 \end{cases}$$

Perceptron Algorithm
Two-class Classification

- Notational convenience: append b at the end of $\mathbf{w}$ and append 1 at the end of $\mathbf{x}_n$ to write pre-activation simply as $\mathbf{w}^T\mathbf{x}_n$.
- A perceptron classifies its input via the non-linear step function

$$y(\mathbf{x}_n) = \begin{cases} 1 & \text{if } \mathbf{w}^T\mathbf{x}_n \geq 0 \\ -1 & \text{if } \mathbf{w}^T\mathbf{x}_n < 0 \end{cases}$$

- *Perceptron criterion*: $\mathbf{w}^T\mathbf{x}_n t_n > 0$ for correctly classified point.

Perceptron Algorithm
Two-class Classification

$\vec{w}^T \vec{x} + b = w_1 x_1 + w_2 x_2 + 1b$

$= \underbrace{[w_1 \;\; w_2 \;\; b]}_{\text{parameters}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix}}_{\text{input}} = \vec{w}^T \vec{x}$

$\vec{w}$ $\quad$ $\vec{x}$

- Notational convenience: append b at the end of $\mathbf{w}$ and append 1 at the end of $\mathbf{x}_n$ to write pre-activation simply as $\mathbf{w}^T\mathbf{x}_n$.
- A perceptron classifies its input via the non-linear step function

$$y(\mathbf{x}_n) = \begin{cases} 1 & \text{if } \underline{\mathbf{w}^T\mathbf{x}_n} \geq 0 \\ -1 & \text{if } \mathbf{w}^T\mathbf{x}_n < 0 \end{cases}$$

- *Perceptron criterion*: $\mathbf{w}^T\mathbf{x}_n t_n > 0$ for correctly classified point.

<u>AND</u>

0	0	-1
0	1	-1
1	0	-1
1	1	1

$$\underbrace{\begin{gathered} \vec{w}^T \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} < 0 \\ \vec{w}^T \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} < 0 \\ \vec{w}^T \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} < 0 \end{gathered}}_{t=-1} \qquad \underbrace{\vec{w}\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \geq 0}_{t=1}$$

Perceptron Algorithm
Two-class Classification

- Loss can be defined on the set $\mathcal{M}(\mathbf{w})$ of misclassified points.

$$L(\mathbf{w}) = \sum_{n \in \mathcal{M}(w)} -\mathbf{w}^T \mathbf{x}_n t_n$$

- Optimal $\mathbf{w}$ minimizes the value of the loss function $L(\mathbf{w})$.

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} L(\mathbf{w})$$

- Gradient is computed as

$$\nabla_{\mathbf{w}} L(\mathbf{w}) = \sum_{n \in \mathcal{M}(w)} -\mathbf{x}_n t_n$$

Perceptron Algorithm

Two-class Classification

$\mathcal{M}$

- Loss can be defined on the set $\mathcal{M}(\mathbf{w})$ of misclassified points.

$\vec{w}$ کوکتی سنرا
دربني بے؟

$$L(\mathbf{w}) = \sum_{n \in \mathcal{M}(w)} -\mathbf{w}^T \mathbf{x}_n t_n \quad = \sum_{n \in \mathcal{M}(w)} -(w_1 x_{n1} + w_2 x_{n2} + b) t_n$$

$$\frac{\partial L}{\partial w_1} = -\sum_{n \in \mathcal{M}(w)} x_{n1} t_n$$

- Optimal $\mathbf{w}$ minimizes the value of the loss function $L(\mathbf{w})$.

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} L(\mathbf{w})$$

$f(\overset{\text{argument}}{x})$

- Gradient is computed as

$$\nabla_{w} L(w) = \begin{bmatrix} \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial w_2} \\ \frac{\partial L}{\partial b} \end{bmatrix} = \begin{bmatrix} -\sum x_{n1} t_n \\ -\sum x_{n2} t_n \\ -\sum t_n \end{bmatrix}$$

$$\nabla_{\mathbf{w}} L(\mathbf{w}) = \sum_{n \in \mathcal{M}(w)} -\mathbf{x}_n t_n$$

Perceptron Algorithm
Two-class Classification

- Optimal $\mathbf{w}^*$ can be learned via gradient descent.
- Corresponds to the following rule at the n-th training sample *if it is misclassified*.

$$\mathbf{w}^{\text{new}} = \mathbf{w}^{\text{old}} + \mathbf{x}_n t_n$$

- Known as the *perceptron learning rule*.
- For *linearly separable data*, perceptron learning is guaranteed to find the decision boundary in finite iterations.
  - Try it for the AND or OR problems.
- For data that is *not linearly separable*, this algorithm will never converge.
  - Try it for the XOR problem.

Perceptron Algorithm
Two-class Classification

- Optimal $\mathbf{w}^*$ can be learned via gradient descent.
- Corresponds to the following rule at the n-th training sample *if it is misclassified*.

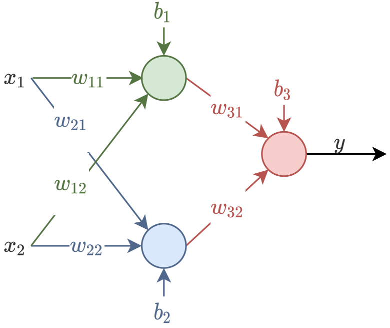
$\nabla_{w} L_n = -\vec{x}_n t_n$

$$\mathbf{w}^{\text{new}} = \mathbf{w}^{\text{old}} + \vec{\mathbf{x}}_n t_n$$

- Known as the *perceptron learning rule*.
- For *linearly separable data*, perceptron learning is guaranteed to find the decision boundary in finite iterations.
  - Try it for the AND or OR problems.
- For data that is *not linearly separable*, this algorithm will never converge.
  - Try it for the XOR problem.

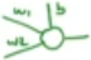
Perceptron Algorithm
Weaknesses

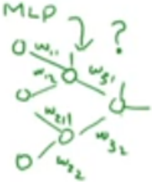
- Only works if training data is linearly separable.
- Cannot be generalized to MLPs.
  - Because t_n will be available for output perceptron only.
  - Hidden layer perceptrons will have no intermediate targets.



- Next lecture: Training MLPs.

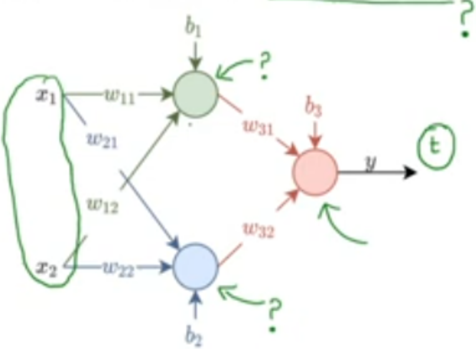
Perceptron Algorithm
Weaknesses





- Only works if training data is linearly separable.
- Cannot be generalized to MLPs.
  - Because t_n will be available for output perceptron only.
  - Hidden layer perceptrons will have no intermediate targets.

$w_1^{new} = w_1^{old} + x_n \underbrace{t_n}$



- Next lecture: Training MLPs.