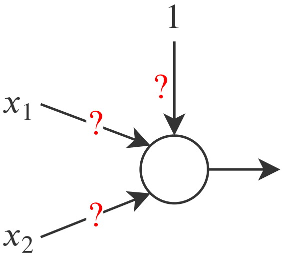
CS-568 Deep Learning

Training a Perceptron

Nazar Khan
Department of Computer Science
University of the Punjab

What is training?



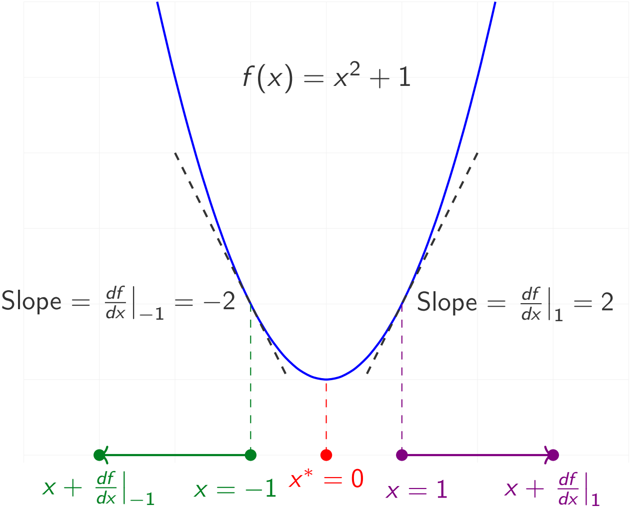
AND		
x_1	x_2	t
0	0	0
0	1	0
1	0	0
1	1	1

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.

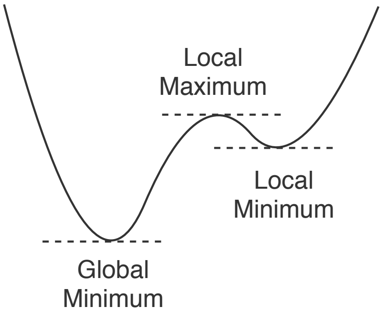
- A perceptron is a function $f : \mathbf{x} \rightarrow t$ with parameters $\mathbf{w}, b$.
- Formally written as $f(\mathbf{x}; \mathbf{w}, b)$.
- Training corresponds to *minimizing a loss function*.
- So let's take a detour to understand function minimization.

Minimization



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

Minimization
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.

Gradient Descent

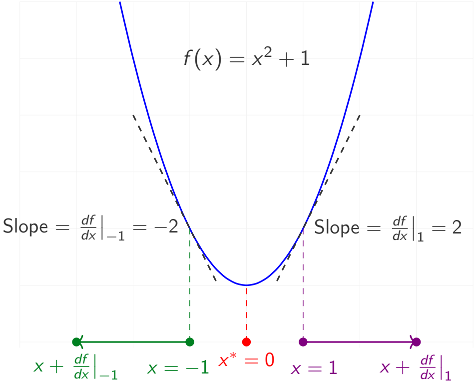
- Gradient is the direction, in input space, of maximum rate of increase of a function.

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

- 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?



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.

Gradient Descent

1. Initialize $\mathbf{w}^{\text{old}}$ randomly.
2. do
   2.1 $\mathbf{w}^{\text{new}} \leftarrow \mathbf{w}^{\text{old}} - \eta \, \nabla_{\mathbf{w}} L(\mathbf{w})|_{\mathbf{w}^{\text{old}}}$
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

- 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.

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

- 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

- 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

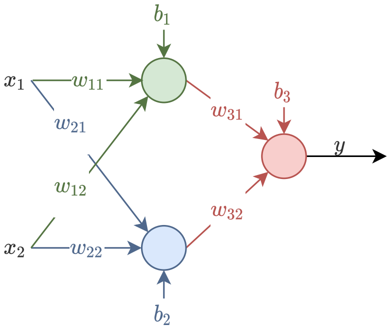
- Optimal $\mathbf{w}^*$ can be learned via gradient descent.
- Corresponds to the following rule at the n-th training sample *if it is misclassified*.

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

- 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
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.



Summary

- Perceptron training corresponds to minimizing a loss function.
- Gradient at minimum of a function is zero.
- Gradient descent: to find minimum, repeatedly move in negative of the gradient direction.
- Perceptron training algorithm only works if training data is linearly separable.
    - Cannot be generalized to MLPs.
- Next lecture: loss and activation functions for ML.