Deep Learning

Perceptron - Gradient Descent

Edgar Roman-Rangel.
edgar.roman@itam.mx

Department of Computer Science.
Instituto Tecnológico Autónomo de México, ITAM.

Outline

Linear regression

Approximate y from the input data $\mathbf{x}$, using the set of weigths $\mathbf{w} = \{\omega_i\}$,

$$\mathbf{y} = \mathbf{X}\mathbf{w}.$$

We could learn $\mathbf{w}$ using the normal equation (least squares):

$$\mathbf{w} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}.$$

Logistic regression

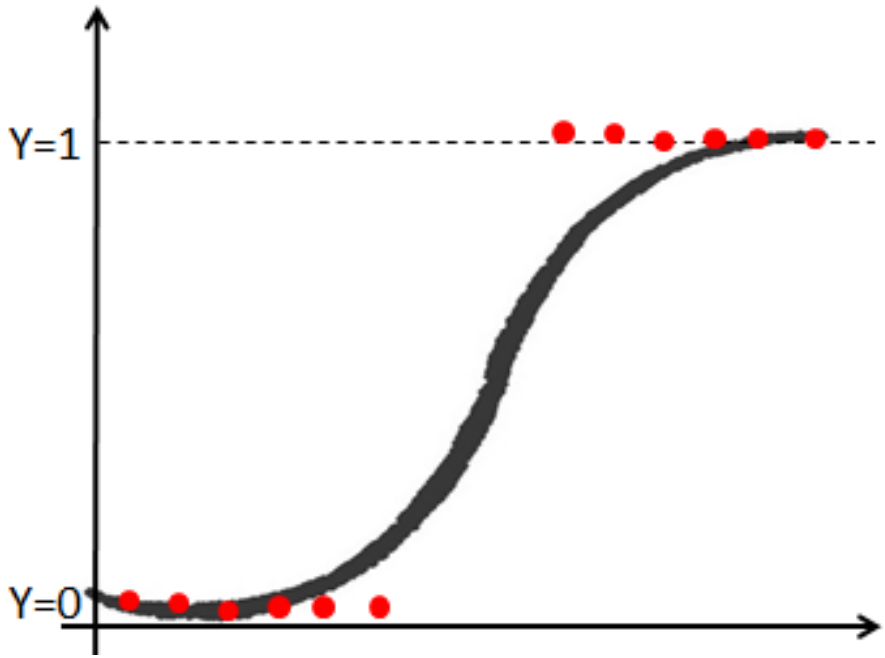
Similarly, we could fit a logistic function to perform binary classification: true vs false (0 vs 1).

$$s = \mathbf{w}^T \mathbf{x},$$
$$y = \sigma(s),$$

where,

$$\sigma(s) = \frac{1}{1 + \exp^{-s}},$$

is the sigmoid function.

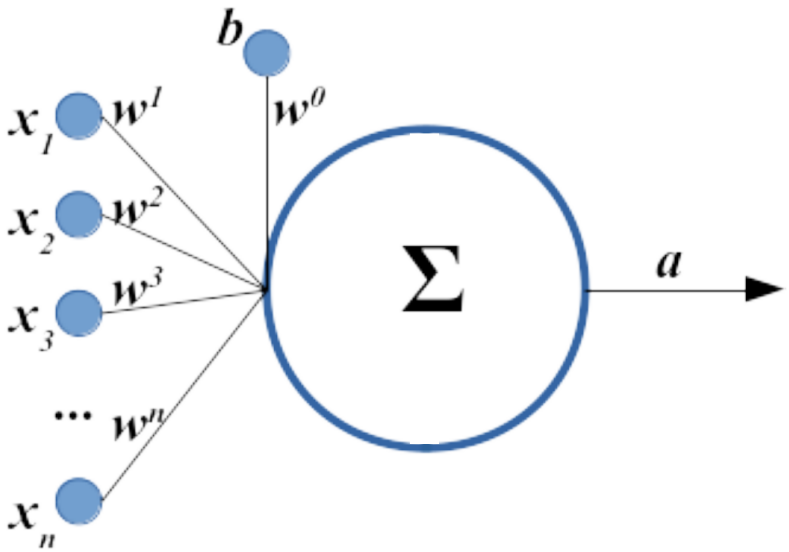


It actually, gives the probability of $y = 1$.

Linear perceptron

Another formulation for regression problems.

$$
\begin{aligned}
y &= \mathbf{w}^T \mathbf{x}, \\
&= \sum_{n=0}^{N} \omega_n x_n, \\
&= \sum_{n=1}^{N} \omega_n x_n + \omega_0 x_0,
\end{aligned}
$$

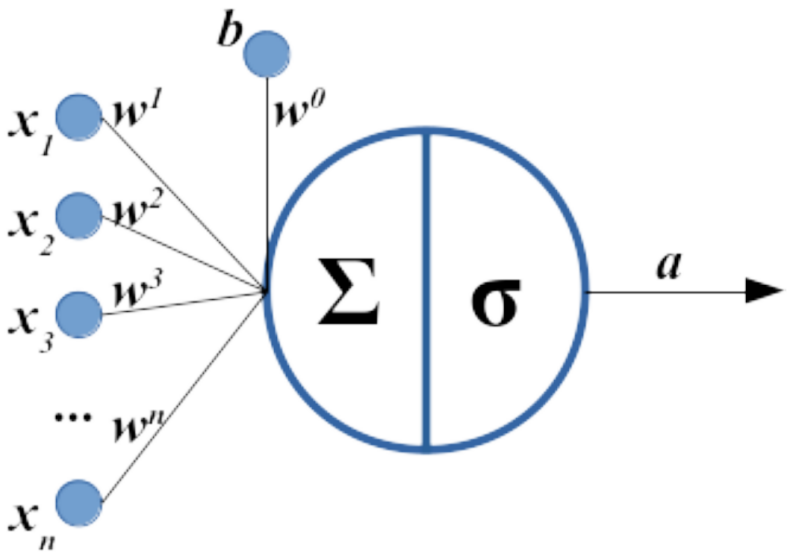


where, $\omega_0 = b$ and $x_0 = 1$.

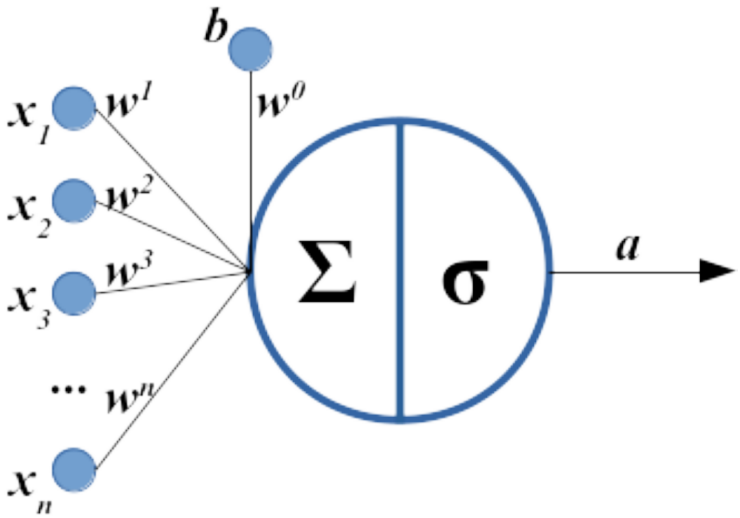
Perceptron

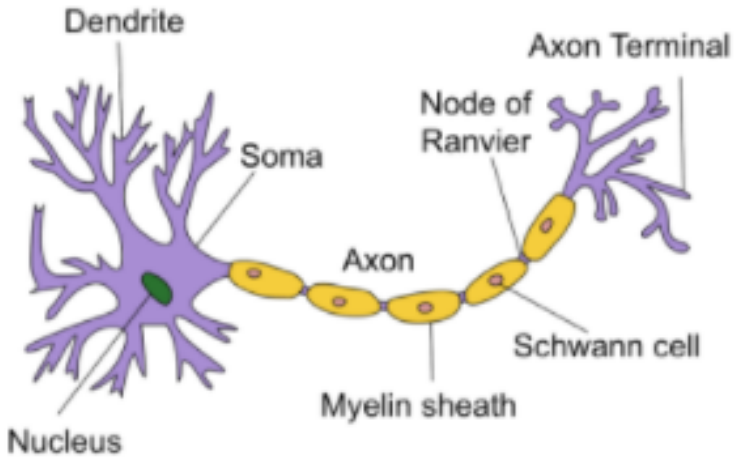
Let's use the sigmoid activation function.

$$s = \mathbf{w}^T \mathbf{x},$$
$$a = \sigma(s).$$



Artificial neuron





Outline

Weights estimation

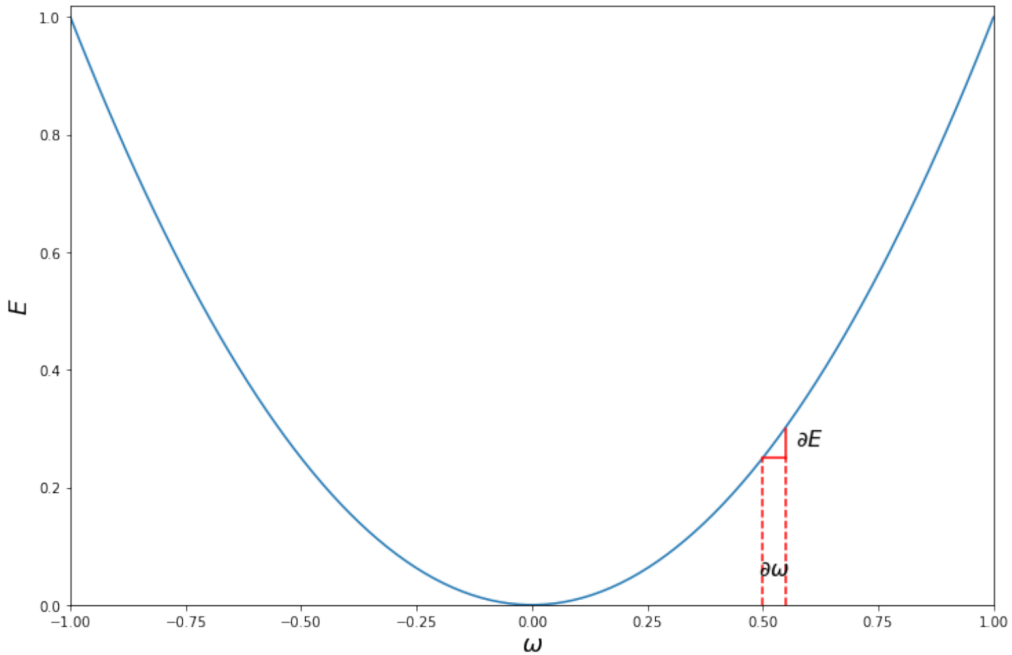
To estimate values for $\{w_i\}$ we use an iterative minization approach termed *Gradient Descent* (GD).

- Most complex problems have no closed-form solution.
- Iterative approaches reach fairly good approximations.
- Risk of getting trapped in local minima.

Gradient descent (GD)

We require a *loss function*. e.g., $E = (y - \hat{y})^2$.

Remeber: relation between derivative, tangent, and direction.



And we can move in the opposite direction of the derivative,

$$\omega_i = \omega_i - \eta \frac{\partial E}{\partial \omega_i}.$$

GD example, I

Consider first only a linear perceptron:

- $\hat{y} = \mathbf{w}^T \mathbf{x} = \sum_{n=0}^{N} \omega_n x_n$,
- $E = (y - \hat{y})^2$.

Then,

$$\begin{aligned}
\frac{\partial E}{\partial \omega_n} &= \frac{\partial (y - \hat{y})^2}{\partial \omega_n}, \\
&= 2(y - \hat{y}) \frac{\partial (y - \hat{y})}{\partial \omega_n}, \\
&= 2(y - \hat{y}) \left[0 - \frac{\partial \sum_{n=0}^{N} \omega_n x_n}{\partial \omega_n} \right], \\
&= -2(y - \hat{y}) x_n.
\end{aligned}$$

Therefore,

$$\omega_n = \omega_n + 2\eta(y - \hat{y}) x_n.$$

GD example, II

Consider now a non-linar perceptron:

- $\hat{y} = \sigma(s)$,
- $s = \mathbf{w}^T\mathbf{x} = \sum_{n=0}^{N} \omega_n x_n$,
- $E = 0.5(y - \hat{y})^2$.

The derivative of the sigmoid function is: $\sigma'(s) = \sigma(s)(1 - \sigma(s))$.

Then,

$$\begin{aligned}\frac{\partial E}{\partial \omega_n} &= \frac{\partial (y - \hat{y})^2}{\partial \omega_n}, \\ &= -(y - \hat{y})\sigma(s)(1 - \sigma(s))x_n.\end{aligned}$$

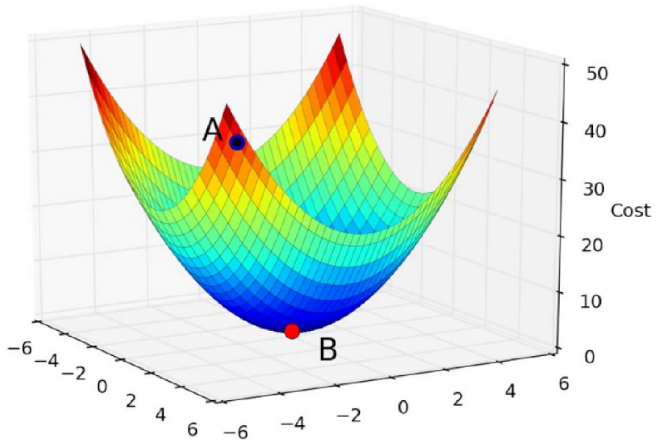
Therefore,

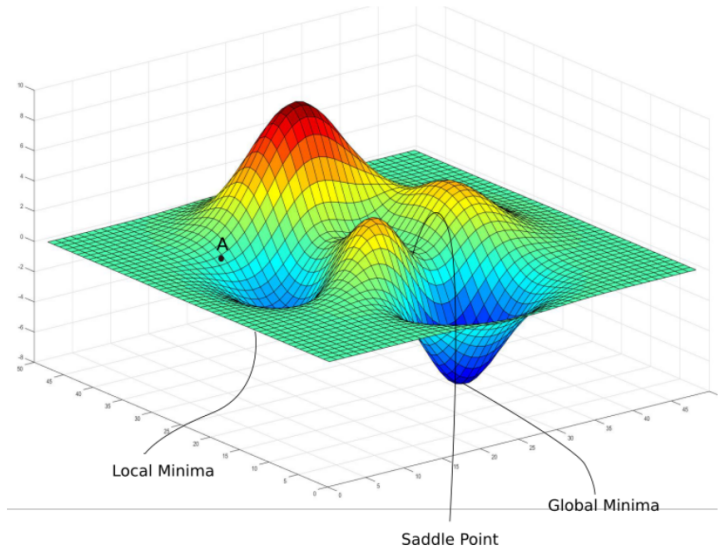
$$\omega_n = \omega_n + \eta(y - \hat{y})\sigma(s)(1 - \sigma(s))x_n.$$

GD multivariado

We can use it for multiple parameters.

- We always must move in the direction of the stepest descent, so first compute the all partial derivatives and then update.





GD procedure

Random initialization

For each epoch

For each training point

- Forward pass
- Error computation.
- Gradient estimation.
- Backward pass (weight adjustment).

Q&A

Thank you!

edgar.roman@itam.mx