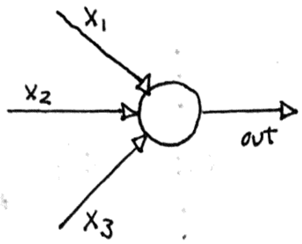
·DEEP LEARNING

PERCEPTRON

(Neurona Binaria)

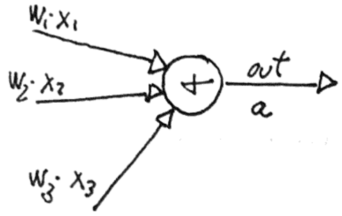


$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

b = bias = −threshold

w = weights

SIGMOID NEURON



sigmoid function $\rightarrow \sigma(z) = \dfrac{1}{1+e^{-z}} \in [0,1]$

$$z = \sum_j w_j \cdot x_j + b \quad \text{(componente a componente)}$$

$$\Downarrow$$

$$\boxed{z = w \cdot x + b}$$

GRADIENT DESCENT

Cost function $\rightarrow C(w,b) = \dfrac{1}{2n} \sum_x \| y(x) - a \|^2$ where $\begin{cases} y(x) = \text{test vector } (y = \text{out}, x = \text{input}) \\ a = \text{neuron output} \end{cases}$

$$\Delta \text{out} \simeq \sum_j \frac{\partial \text{out}}{\partial w_j} \Delta w_j + \frac{\partial \text{out}}{\partial b} \Delta b$$

$$\Delta C \simeq \frac{\partial C}{\partial v_1} \Delta v_1 + \frac{\partial C}{\partial v_2} \Delta v_2 \quad (v_1, v_2) \rightarrow \text{direccions}$$

$v \equiv (w, b)$

$$\nabla C = \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2} \right)^T \longrightarrow \Delta C = \nabla C \cdot \Delta v$$

$$\Delta v \equiv -\nabla C \cdot \eta \qquad \eta = \text{learning rate}$$

SGD → Stocastic Gradient descent

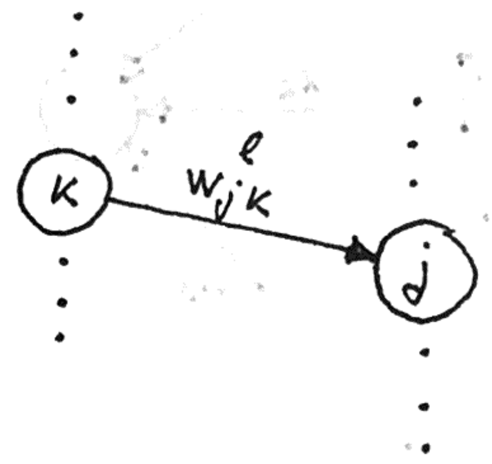
SVM → Support Vector Machine (NOT MACHINE LEARNING ALGORITHM)

sophisticated algorithm ≤ simple learning algorithm + Good Training Data

Deep Neural Networks → Networks with many-layers structure (two or more hidden layers)

CHAPTER 2. BACKPROPAGATION

Notation → w_{jk}^{l} $\begin{cases} l \equiv \text{layer} \\ k \equiv \text{Neurona en capa } (l-1) \\ j \equiv \text{Neurona en capa } l \end{cases}$



$b_j^l \left\{ \begin{matrix} l \to \text{capa} \\ j \to \text{Neuron} \end{matrix} \right\} a_j^l$ [Bias & Activations]

$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right) \equiv \sigma(\underbrace{w^l \cdot a^{l-1} + b^l}_{z^l}) \leftarrow \text{MATRIX FORM}$$

↑ weighted input for neurons in layer l.

BACKPROPAGATION

Objetive → $\left(\frac{\partial C}{\partial w}, \frac{\partial C}{\partial b}\right)$

Quadratic Cost → $C = \frac{1}{2n} \sum_x \| y(x) - a^L(x) \|^2$ $\begin{cases} n = \text{Total number of training examples} \\ L = \text{Number of layers (Final layer)} \\ x = \text{input} \\ y = \text{output (desired)} \end{cases}$

$$C = \frac{1}{n} \sum_x C_x$$

$$C_x = \frac{1}{2} \| y(x) - a^L(x) \|^2 \leftarrow \text{Average}$$

$C = C(a_L)$ ← Depends on activations

$\delta_j^l \equiv$ Error in j^{th} neuron in l^{th} layer → $\delta_j^l = \frac{\partial C}{\partial z_j^l}$

4 EQUATIONS

1.- ERROR IN THE OUTPUT LAYER

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L) = \underbrace{\nabla_a C \bullet \sigma'(z^L)}_{\text{MATRIX-BASED FORM}} = (a^L - y) \bullet \sigma'(z^L)$$

↑ Substituyendo el gradiente del coste respecto de la activación.

2.- ERROR IN LAYER L REFERRED TO ERROR IN LAYER L+1 $\rightarrow \delta^{l} = ((w^{l+1})^{T}\delta^{l+1}) \odot \sigma'(z^{l})$

· with 1 an 2, we can compute every error of the net from L backwords.

3.- RATE OF CHANGE OF THE COST WITH RESPECT TO ANY BIAS $\rightarrow \dfrac{\partial C}{\partial b_j^l} = \delta_j^l$

4.- RATE OF CHANGE OF THE COST WITH RESPECT TO ANY WEIGHT $\rightarrow \dfrac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \cdot \delta_j^l \rightarrow \dfrac{\partial C}{\partial w} = a_{in} \cdot \delta_{out}$

$$\bigcirc \xrightarrow[w]{a_{in}\cdot\delta_{out}} \bigcirc$$

"Weights output from low activations neurons learn slowly (the output neuron is saturated 0 or 1)"

PROOFS

1. $$\delta_j^L = \frac{\partial C}{\partial z_j^L} = \sum_k \frac{\partial C}{\partial a_k^L}\cdot\frac{\partial a_k^L}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L}\cdot\frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L)$$

$a_j^L = \sigma(z_j^L)$

2. $$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}}\cdot\frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l}\cdot\delta_k^{l+1} = \sum_k w_{kj}^{l+1}\cdot\delta_k^{l+1}\cdot\sigma'(z_j^l)$$

$$z_k^{l+1} = \sum_j w_{kj}^{l+1}\cdot a_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1}\sigma(z_j^l) + b_k^{l+1} \underset{\frac{\partial}{\partial z_j^l}}{=} w_{kj}^{l+1}\sigma'(z_j^l)$$

3. $$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \frac{\partial C}{\partial b_j^l}\cdot\frac{\partial b_j^l}{\partial z_j^l} = \frac{\partial C}{\partial z_j^l}\cdot 1$$

$$z_k^l = \sum_j w_{kj}^l a_j^{l-1} + b_k^l = \sum_j w_{kj}^l \sigma(z_j^{l-1}) + b_k^l \underset{\frac{\partial}{\partial z_k^l}}{=} 1$$

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \frac{\partial C}{\partial w_{jk}^l} \cdot \frac{\partial w_{jk}^l}{\partial z_j^l} =$$

$\boxed{\text{ALGORITHM}}$

1. INPUT. x → set a^l for input layers

2. FEEDFORWARD → $z^l = w^l \cdot a^{l-1} + b^l \quad \forall\, l \in (2, L)$

3. OUTPUT ERROR → $\delta^L = \nabla_a C \bullet f'(z^L)$

4. BACK PROPAGATE ERROR → $\delta^l = ((w^{l+1})^T \cdot \delta^{l+1}) \bullet f'(z^l) \quad \forall\, l \in (L-1, 1)$.

5. OUTPUT → $\dfrac{\delta C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \;;\; \dfrac{\partial C}{\partial b_j^l} = \delta_j^l$

6. GRADIENT DESCENT (update)

$$\left.\begin{cases} w^l \to w^l - \dfrac{\eta}{m} \displaystyle\sum_x \delta^{x,l} (a^{x,l-1})^T \\ b^l \to b^l - \dfrac{\eta}{m} \displaystyle\sum_x \delta^{x,l} \end{cases}\right\}$$

Se hace con grupos de vectores de test (mini-batch)

CHAPTER 3.

CROSS-ENTROPY COST FUNCTION

$$C = \frac{-1}{n} \sum_x \left[y \ln(a) + (1-y) \ln(1-a) \right]$$

Avoids the problem of learning slow-down near saturation.
Better than quadratic cost for using with sigmoid neurons.

$$\frac{\partial C}{\partial w_j} = \frac{1}{n} \sum_x x_j (\sigma(z) - y)$$

error, learning depends on error.

$$\frac{\partial C}{\partial b} = \frac{1}{n} \sum_x (\sigma(z) - y)$$

Many neurons $\rightarrow$ $C = \frac{-1}{n} \sum_x \sum_y \left[y_j \ln a_j^L + (1-y_j) \ln (1-a_j^L) \right]$

SOFTMAX

(Inciso, solución alternativa a slowdown problem)

* New type of output ~~neuron~~ layer in neuronal network.

$$z_j^l = \sum_k w_{jk}^L a_k^{L-1} + b_j^L$$

softmax function $\rightarrow$ $a_j^L = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}}$ $\quad \left[\sum_j a_j^L = 1 \right]$ ← Can be understood as a probability distribution.

Log-likelihood Cost Function $\rightarrow$ $C \equiv -\ln a_y^L$ ← Desired output for the network

$\downarrow$

$$\left. \begin{aligned} \frac{\partial C}{\partial b_j^L} &= a_j^L - y_j \\ \frac{\partial C}{\partial w_k^l} &= a_k^{L-1} (a_j^L - y_j) \end{aligned} \right\}$$

Works as sigmoids neurons + cross-entropy.

OVERFITTING AND REGULARIZATION

· OVERFITTING → What the network learns, no longer generalizes to the test data.
It is a mayor problem on modern neuronal networks.

↓

Strategy to prevent: EARLY STOPPING

HOLD-OUT METHOD → To use a validation data set. When the classification accuracy on that set stops, we stop training.

* Validation data: Training data that helps us to learn good hyper-parameters.

→ One of the best ways of reduccing overfitting is to increase test data size. ←

· REGULARIZATION → Techniques to reduce over-fitting.

* WEIGHT DECAY TECHNIQUE / L2 REGULARIZATION

To add a "regularization term" to the cost function.

- Regularization Cross-Entropy → $C = \frac{-1}{n} \sum_{x_j} [y_j \ln(a_j^L) + (1-y_j) \ln(1-a_j^L)] + \underbrace{\frac{\lambda}{2n} \sum_w w^2}$

$\lambda > 0$ (Regularization Parameter)

- Regularized Cuadratic cost → $C = \frac{1}{2n} \sum_x \| y - a^L \|^2 + \frac{\lambda}{2n} \sum_w w^2$

- In general → $C = C_0 + \frac{\lambda}{2n} \sum_w w^2$

Compromise between finding small weights and minimizing the original cost function.

$$\begin{cases} \lambda \downarrow : \text{Minimize original cost function} \\ \lambda \uparrow : \text{Find small weights.} \end{cases}$$

$\left(\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{n} w \ ; \ \frac{\partial C}{\partial b} = \frac{\partial C_0}{\partial b} \right)$ ← To Apply SGD

$w' \to \left(1 - \frac{\eta \lambda}{n}\right) w - \eta \frac{\partial C_0}{\partial w}$; $b' \to b - \eta \frac{\partial C_0}{\partial b}$

↑

Weight decay.

· OTHER TECHNIQUES

- L1 REGULARIZATION ⇒ $C = C_0 + \frac{\lambda}{n} \sum_w |w|$

(It also penalize large weights)

$$\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{n} \operatorname{sgn}(w)$$

$$w' \to w - \frac{\eta \lambda}{n} \operatorname{sgn}(w) - \eta \frac{\partial C_0}{\partial w}$$

- DROPOUT ⇒ Modify the network itself.

→ 1. Randomly (and temporarily) deleting half the hidden neurons.

2. Apply the backpropagation algorithm

3. Back to 1.

- ARTIFIAL EXPANDING THE TRAINING DATA.

Modify the training data to obtain a new one (i.e. rotating the set of images a little amount). It can be applied to many fields like noise recognition (i.e adding noise or changing velocity of speech).

WEIGHT INITIALIZATION

We initilized the weights with a Gaussian distribution with ($\bar{x} = 0$ & $\sigma = 1$)

⇓ (σ = standard deviation)

New Approach → $\bar{x} = 0$, $\sigma = 1/\sqrt{n_{in}}$ (n_{in} = number of input weights.)

HOW TO CHOOSE NEURAL NETWORK HYPERPARAMETERS?

→ BROAD STRATEGY

- Learning Rate: The only parameter that can be tunned using the cost, because it represents the step in SGD.
- Number of epochs ⇒ Early stopping (automatic)
  - When the classification accuracy has not improved in quite some time.
- Learning Rate Schedule → start with a big learning rate and when accuracy is getting worse, reduce it by a factor (2, 10, ...).
- Regularization Parameter → Start with $\lambda = 0$ and tunned η.
  Put $\lambda = 1.0$ and tunned it in factors of 10.
  Then tunned again η.
- Minibatch Size → Compromise value that maximizes the speed of learning (most rapid improvement in performance).

→ AUTOMATED TECHNIQUES → Grid Search.

READ → "Practical Bayesian optimization of machine learning algorithm" Jasper Snoek, Hug Larochelle & Ryan Adams.

OTHER TECHNIQUES

- Hessian Technique → $C(w+\Delta w) \simeq C(w) + \nabla C \cdot \Delta w + \frac{1}{2} \Delta w^T H \Delta w$

  H = Hessian Matrix

  $$\Delta w = -H^{-1} \nabla C$$

  In practice → $\Delta w = -\eta H^{-1} \nabla C$

  $$w' \rightarrow w + (-\eta H^{-1} \nabla C)$$

Muy difícil de llevar a la práctica.

- MOMEMTUM-BASED GRADIENT DESCENT

Introduce velocity $\to v$ (one per variable)

$$v \to v' = \mu v - \eta \nabla C \qquad (\mu \equiv \text{friction}) \quad \mu = 1 \to \text{NO FRICTION}$$

$$w \to w' = w + v'$$

momentum coefficient.

$\boxed{\text{OTHER MODELS OF NEURONS}}$

- tanh neurons $\to a = \tanh(w \cdot x + b)$
- Rectified linear neurons $\to a = \max(0, w \cdot x + b)$