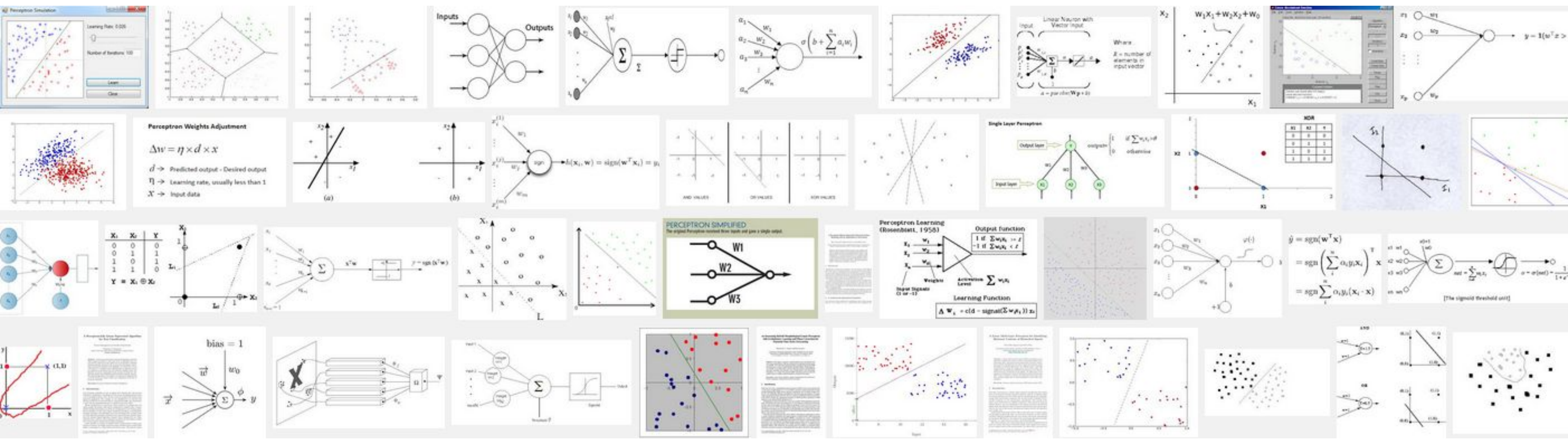
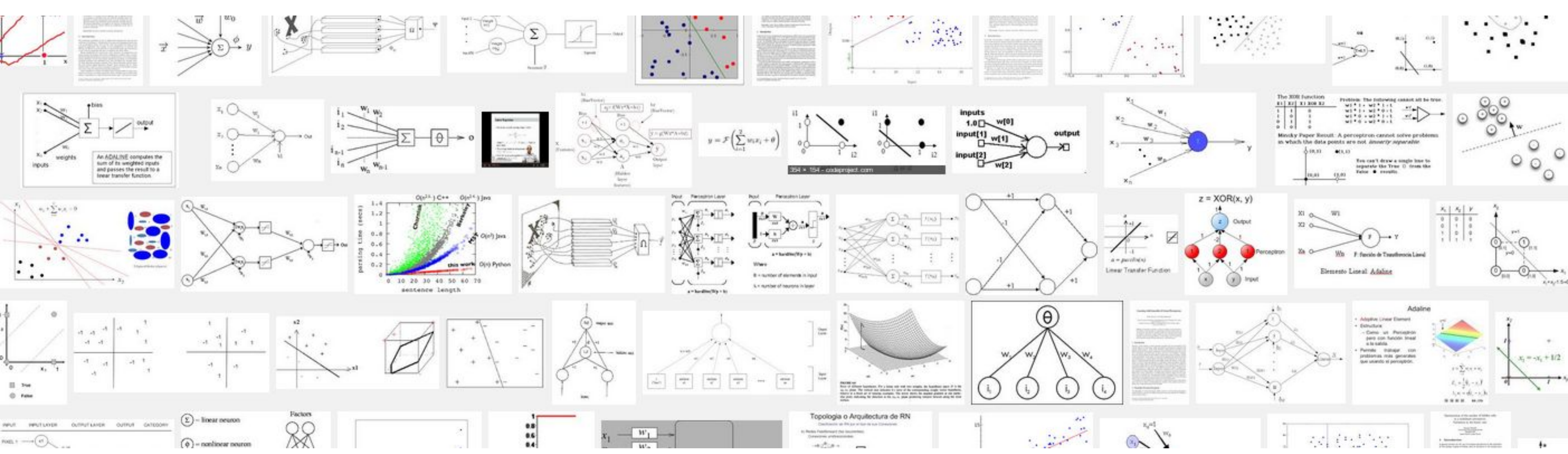
Unidad 2: Redes Neuronales Artificiales

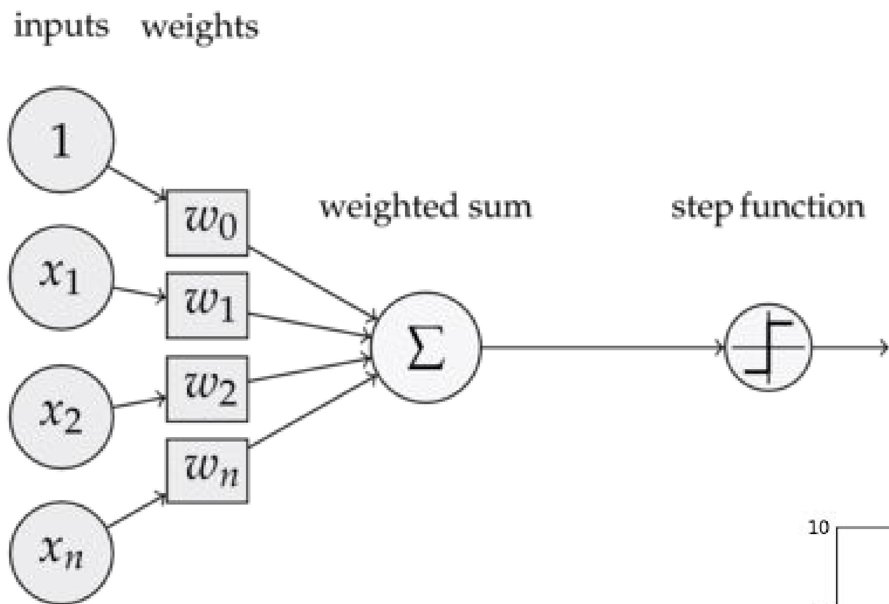
Curso: Redes Neuronales Profundas



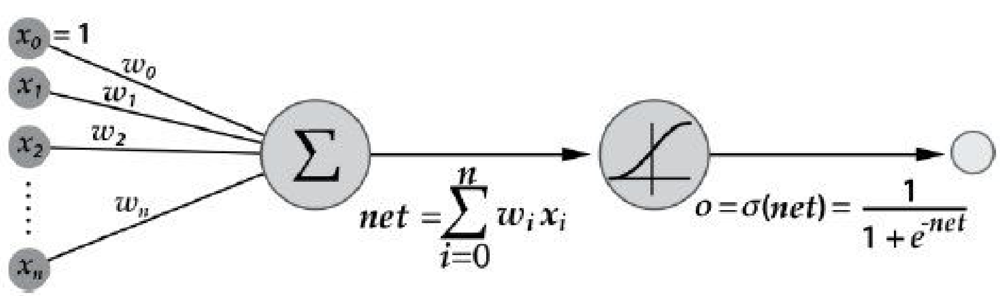
Linear Perceptron

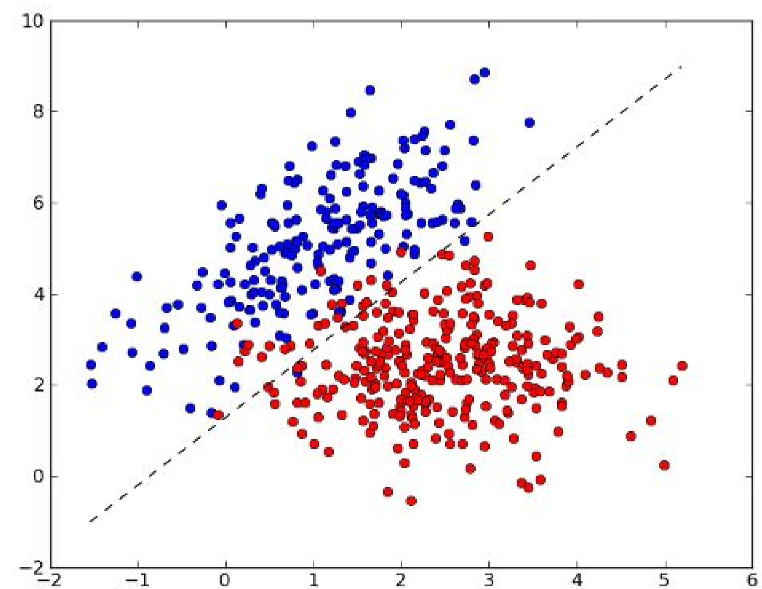


Linear Perceptron

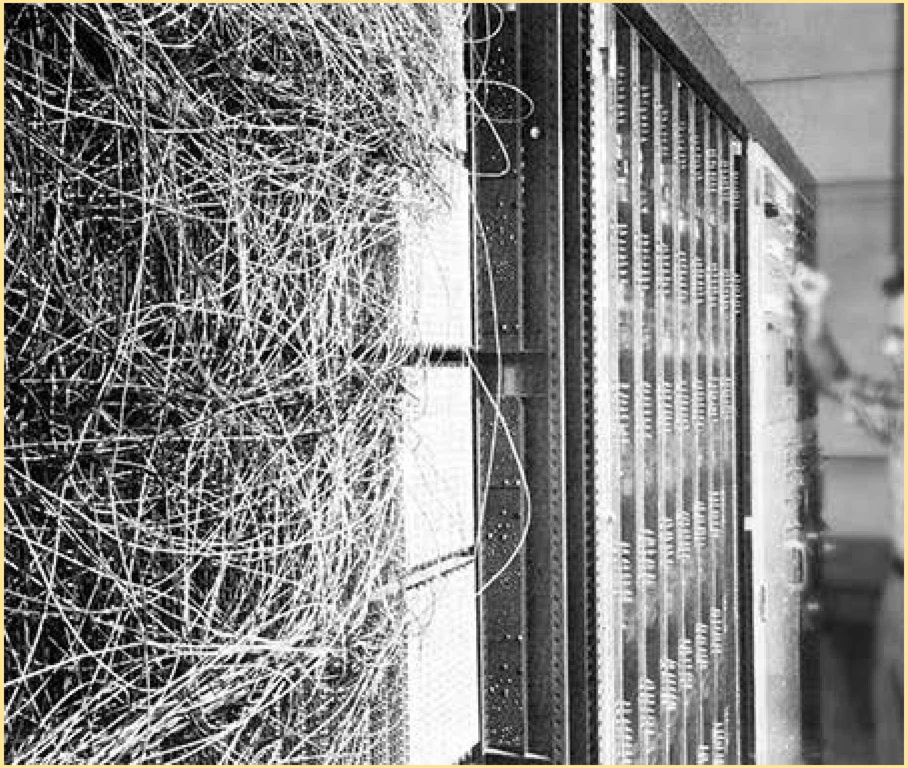


$$g_0(x) = \sigma(x^T W^{(0)} + b^{(0)})$$





Mark I Perceptron (1957)



Input: Imágenes de 20x20 pixeles (photocells)
Pesos adaptativos: arreglo de potenciómetros
Updates: motores eléctricos

Frank Rosenblatt
Cornell Aeronautical Laboratory

THE NEW YORK TIMES REPORTED THE PERCEPTRON TO BE "THE EMBRYO OF AN ELECTRONIC COMPUTER THAT [THE NAVY] EXPECTS WILL BE ABLE TO

- WALK
- TALK
- SEE
- WRITE
- REPRODUCE ITSELF
- AND BE CONSCIOUS OF ITS EXISTENCE

Softmax Regression

$$\log \overbrace{p(\underset{\text{output (multiple categories)}}{y}|\underset{\text{input}}{x})}^{\text{class probability vector}} = \overbrace{\underbrace{x^T W + b}_{\text{affine transform.}} + \underbrace{c(x)}_{\text{normalization}}}^{\text{parametric model}}$$

$$p(y|x) = \frac{\exp(x^T W + b)}{\sum_i \exp(x^T W + b)_i} = \mathrm{softmax}(x^T W + b)$$

Softmax Regression Training

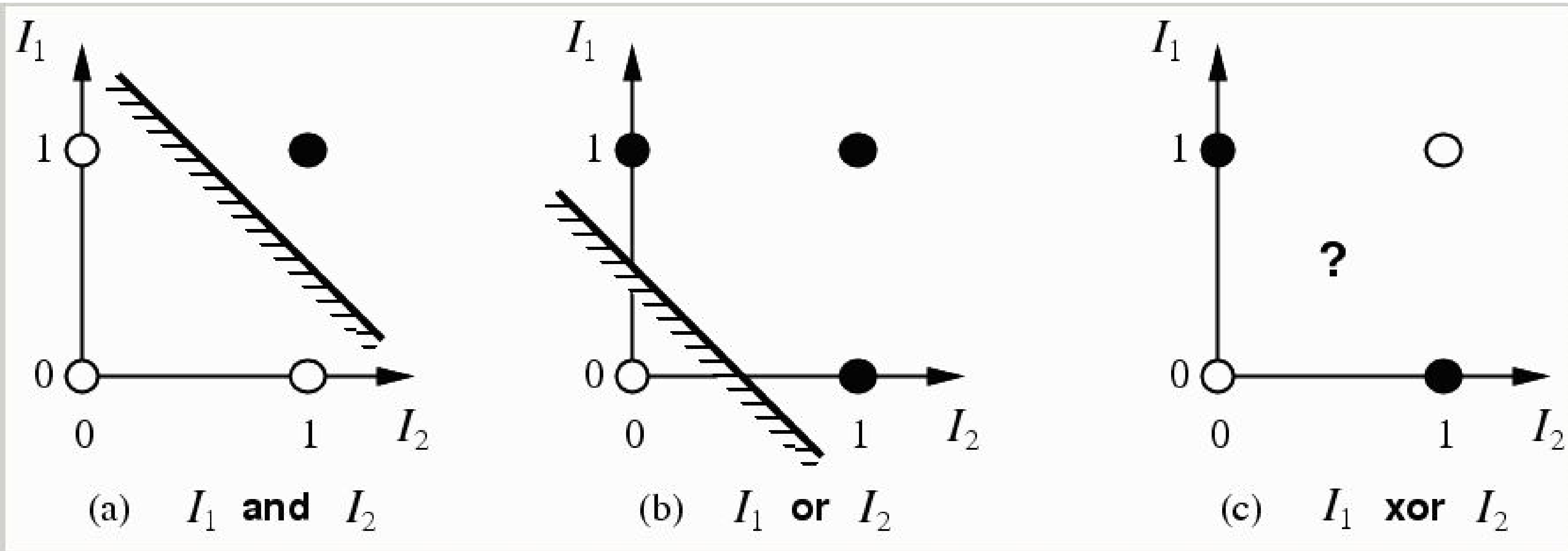
$$J(\mathcal{D}, W, b) = \prod_{x,y \in \mathcal{D}} p(y|x)$$ Maximum Likelihood Estimation (MLE)

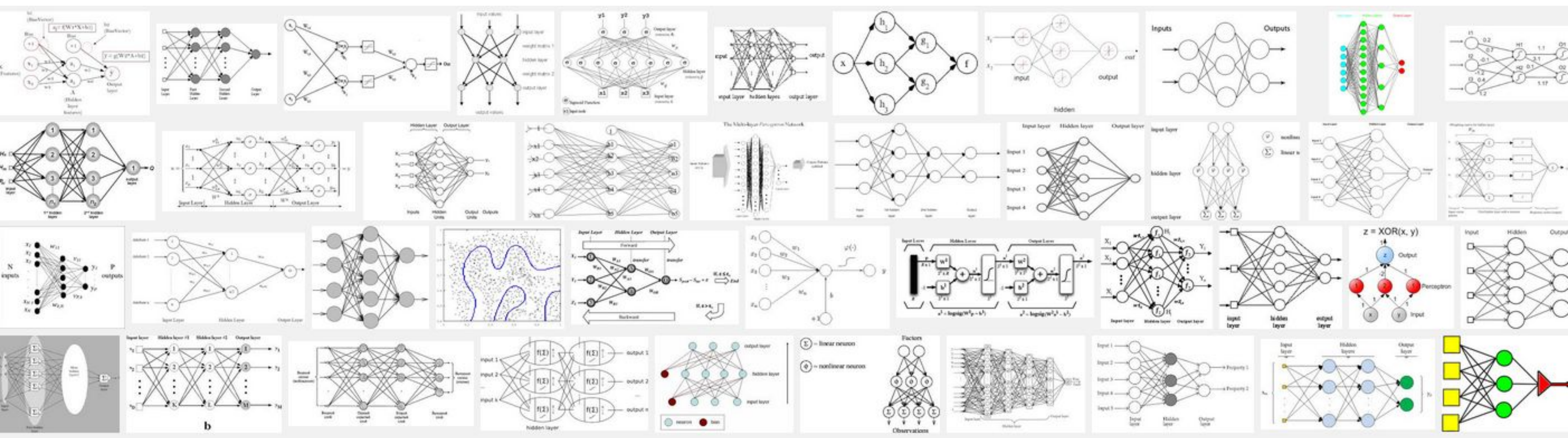
training dataset

$$J(\mathcal{D}, W, b) = \sum_{x,y \in \mathcal{D}} \log p(y|x)$$ Log-Likelihood

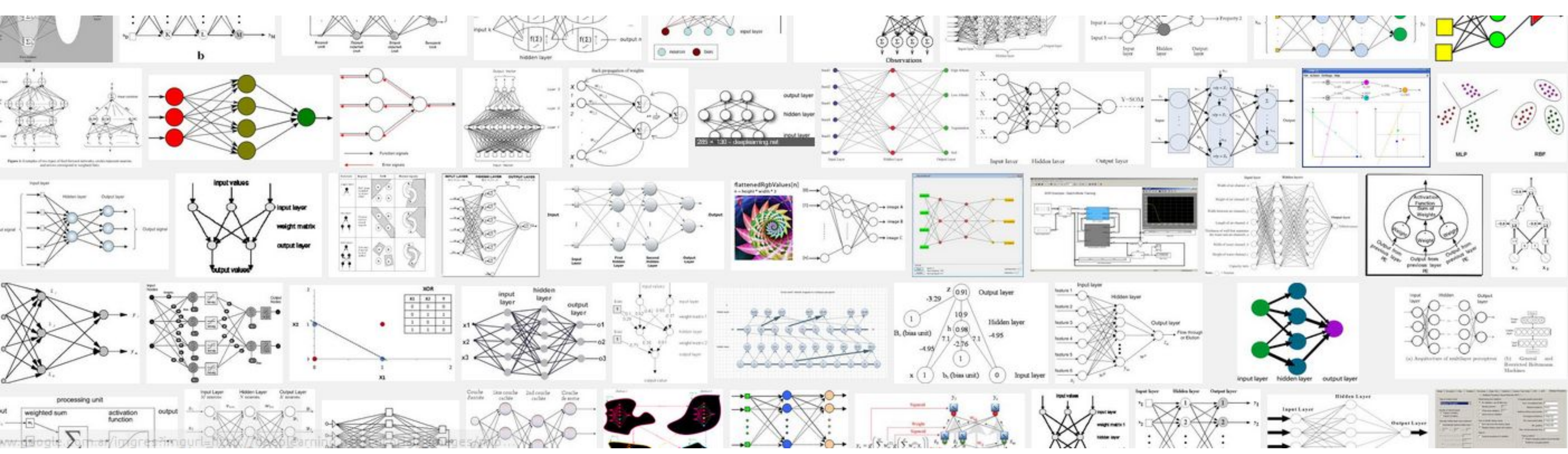
$$NLL(\theta, \mathcal{D}) = -\sum_{i=0}^{|\mathcal{D}|} \log P(Y = y^{(i)} | x^{(i)}, \theta)$$ Negative Log-Likelihood

Linear Perceptron





Multilayer Perceptrons (MLP)



Multilayer Perceptrons (MLP)

$$J(\mathcal{D}, W, b) = \sum_{x,y \in \mathcal{D}} \log p(y|x)$$

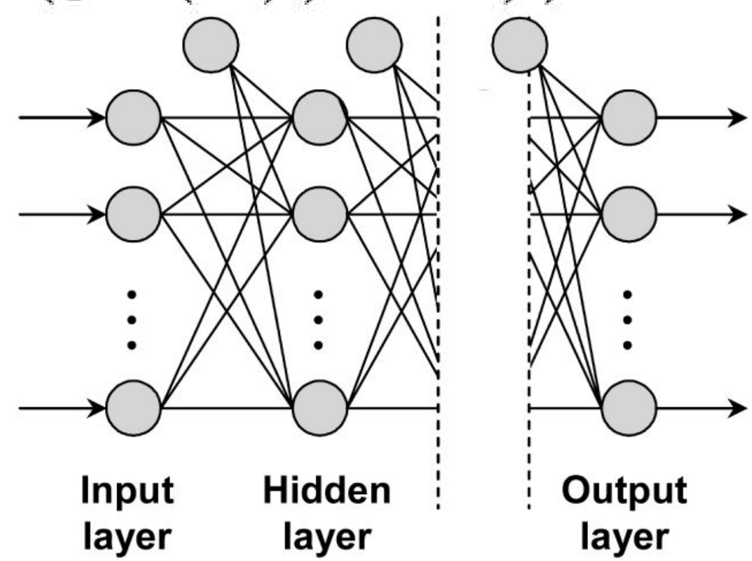
Log-Likelihood

$$\log p(y|x) = \cancel{x^T W + b + c(x)}$$

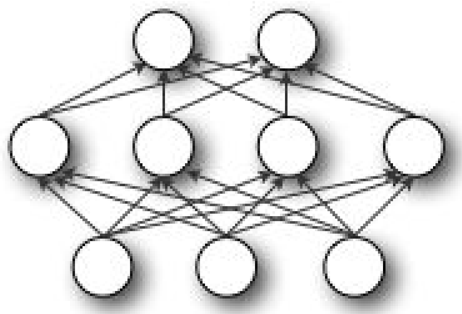
Single layer softmax

$$\overbrace{f(x)}^{\text{MLP output}} = g_L(g_{L-1}(\cdots g_2(g_1(x))\ldots))$$

$$g_\ell(x) = \sigma(x^T W^{(\ell)} + b^{(\ell)})$$

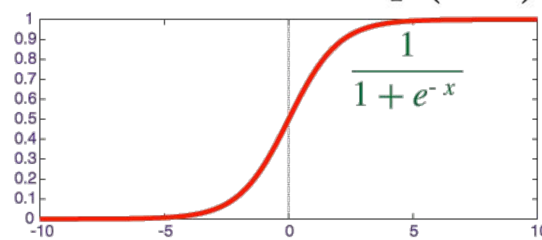


Multilayer Perceptrons (MLP)



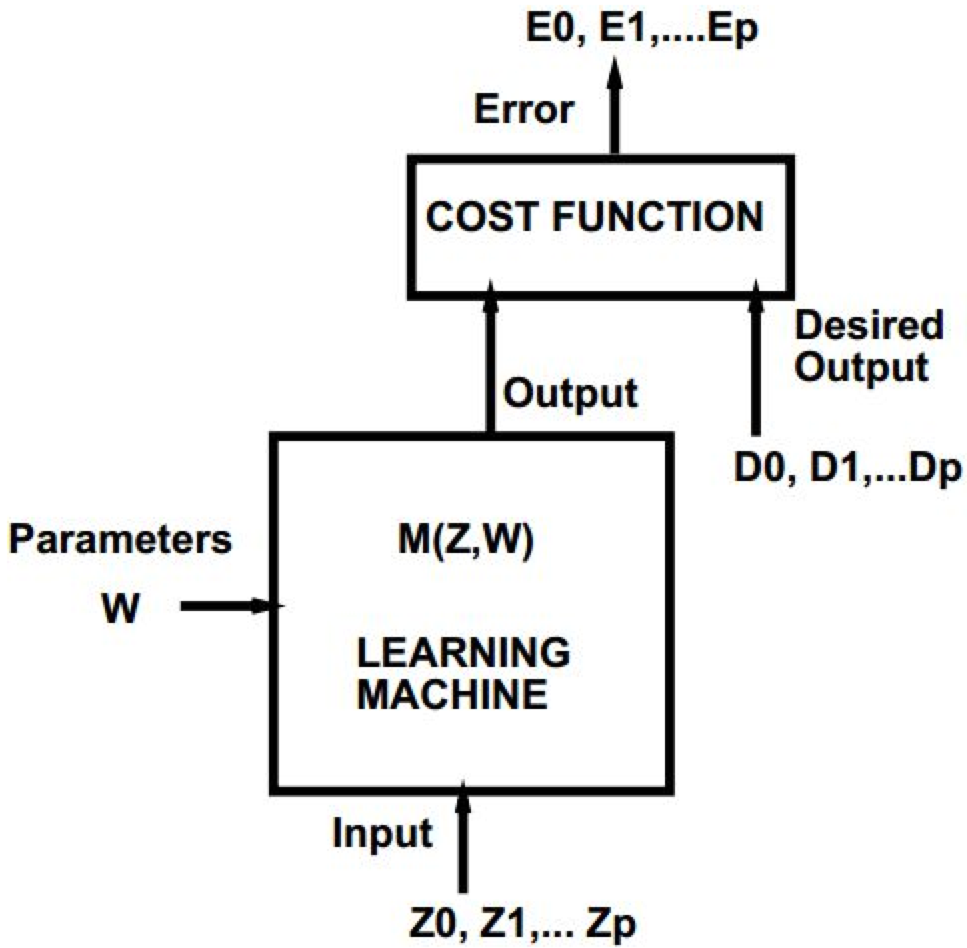
$$g_2(g_1) = \text{softmax}(g_1^T W^{(2)} + b^{(2)})$$

$$g_1(x) = \sigma(x^T W^{(1)} + b^{(1)}) \qquad \sigma(z) = \frac{1}{1+exp(-z)}$$



$$f(x) = \text{softmax}\left(\sigma(x^T W^{(1)} + b^{(1)})^T W^{(2)} + b^{(2)}\right)$$

Gradient Based Learning Machine



$$E^p = \frac{1}{2}(D^p - M(Z^p, W))^2$$

$$E_{train} = \frac{1}{P}\sum_{p=1} E^p$$

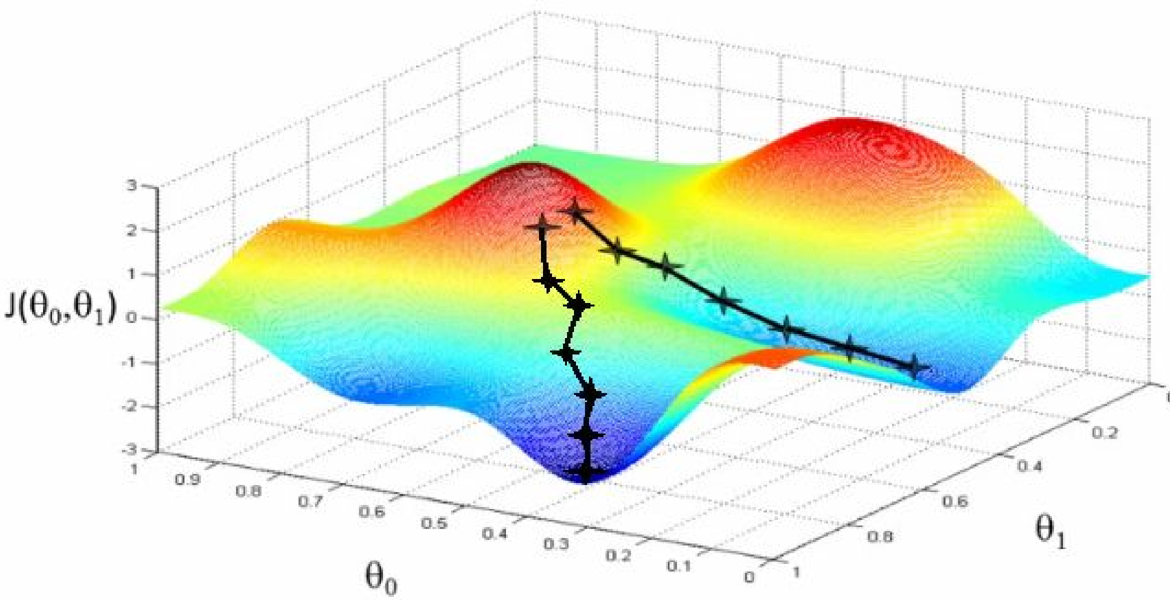
$$W_k = W_{k-1} - \epsilon \frac{\partial E(W)}{\partial W}$$

LeCun, Yann A et al. "Efficient backprop." *Neural networks: Tricks of the trade* (2012): 9-48.

Gradient Descent

Algorithm 1 GRADIENT DESCENT

while True **do**
  loss = f(params)
  d_loss_wrt_params = ... ▷ compute gradient
  params -= learning_rate * d_loss_wrt_params
  if stopping condition is met **then return** params
  end if
end while



Stochastic Gradient Descent

Algorithm 1 GRADIENT DESCENT

while True **do**
  loss = f(params)
  d_loss_wrt_params = ... ▷ compute gradient
  params -= learning_rate * d_loss_wrt_params
  if stopping condition is met **then return** params
  end if
end while

Algorithm 2 STOCHASTIC GRADIENT DESCENT

for $(x_i, y_i) \in \mathcal{D}_{train}$ **do**
  ▷ imagine an infinite generator that may repeat
  ▷ examples (if there is only a finite training set)
  loss = f(params, x_i, y_i)
  d_loss_wrt_params = ... ▷ compute gradient
  params $- =$ learning_rate * d_loss_wrt_params
  if stopping condition is met **then return** params
  end if
end for

Mini-batch Gradient Descent

Algorithm 2 STOCHASTIC GRADIENT DESCENT

for $(x_i, y_i) \in \mathcal{D}_{train}$ **do**
▷ imagine an infinite generator that may repeat
▷ examples (if there is only a finite training set)
loss = f(params, x_i, y_i)
d_loss_wrt_params = ... ▷ compute gradient
params − = learning_rate * d_loss_wrt_params
if stopping condition is met **then return** params
end if
end for

Algorithm 3 MINIBATCH SGD

for (x_batch,y_batch) ∈ train_batches **do**
▷ imagine an infinite generator
▷ that may repeat examples
loss = f(params, x_batch, y_batch)
d_loss_wrt_params = ... ▷ compute gradient
params − = learning_rate * d_loss_wrt_params
if stopping condition is met **then return** params
end if
end for

Hyperparameters: Learning Rate

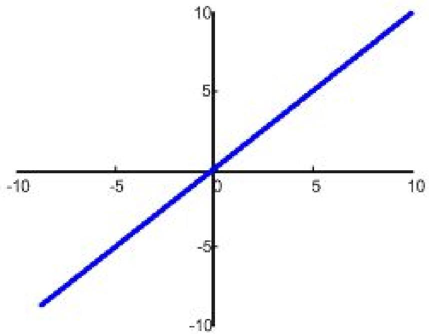
$$W_i \leftarrow W_i - \underbrace{\eta}_{\text{learning rate}} \frac{\partial \mathrm{E}(W_i)}{\partial W_i}$$

- constant learning rate (simplest solution)
- logarithmic grid search (10^{-1}, 10^{-2}, ...)
- decreasing learning rate over time:

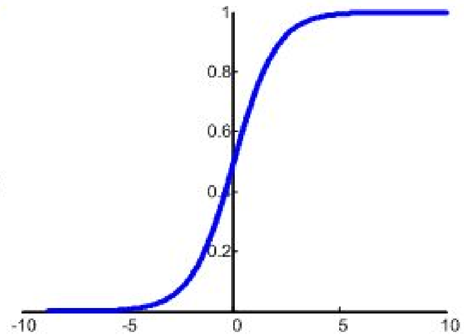
$$\eta_t = \frac{\eta_0}{1 + at}$$

For adaptive learning rate see: LeCun, Yann A et al. "Efficient backprop." *Neural networks: Tricks of the trade* (2012): 9-48.

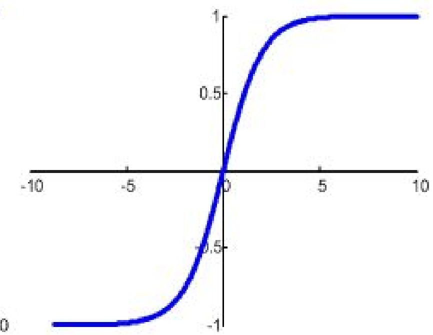
Hyperparameters: Nonlinearity



linear



logistic

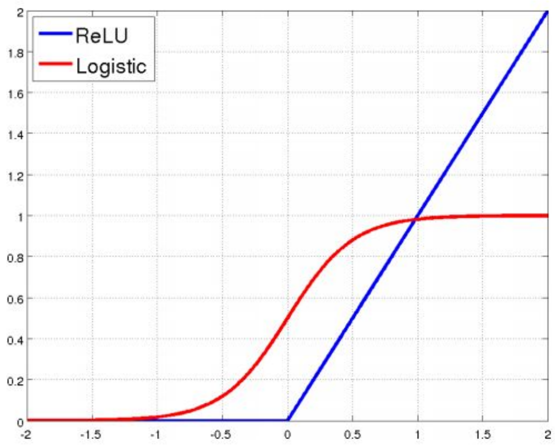


tanh

$$f(u) = 1/(1+\exp(-u))$$

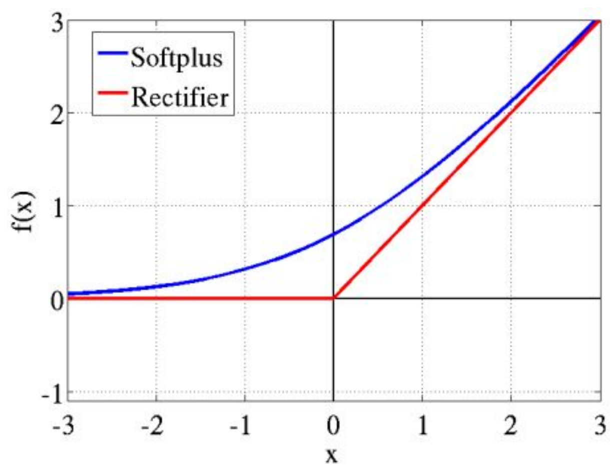
ReLU

$$f(u) = \max(0, u)$$

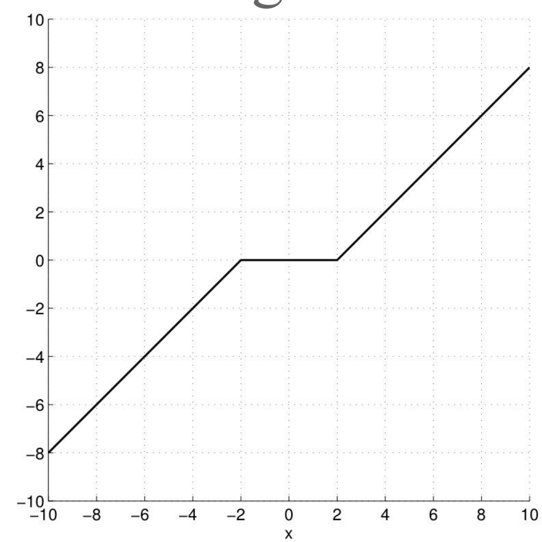


Softplus

$$softplus(x) = log(1+e^{x})$$



Shrinkage



Hyperparameters: Weight Initialization

$$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right]$$

for tanh activations

$$W \sim U\left[-\frac{4\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{4\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right]$$

for logistic activations

Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." *International Conference on Artificial Intelligence and Statistics* 2010: 249-256.

Hyperparameters: momentum

$$\Delta\theta_i(t) = v_i(t) = \alpha v_i(t-1) - \epsilon \frac{dE}{d\theta_i}(t)$$

momentum

learning rate

Hinton, Geoffrey E. "A practical guide to training restricted boltzmann machines." *Neural Networks: Tricks of the Trade* (2012): 599-619.