Programando con numpy, la teoría detrás de la red neuronal clásica

Road to Python Day #2

Historia

- **IA Simbólica desde 1950 – mediados y finales de la década de 1980 (Sistemas expertos).**
- **Minsky y Papert en su libro Perceptron desacreditan la utilidad del perceptrón (1969).**
- **Se retrasa en una década el desarrollo de métodos de optimización para el aprendizaje efectivo.**

Historia

- **1986 Se publica en nature el artículo que revivió el hype con el uso de aprendizaje en base a gradientes.**
- **Geoffrey Hinton y Yann LeCu popularizan el método del gradiente aplicado los MLP (Muerte de la IA Simbólica).**
- **Se propone el modelo LeNet y la historia continúa . . .**

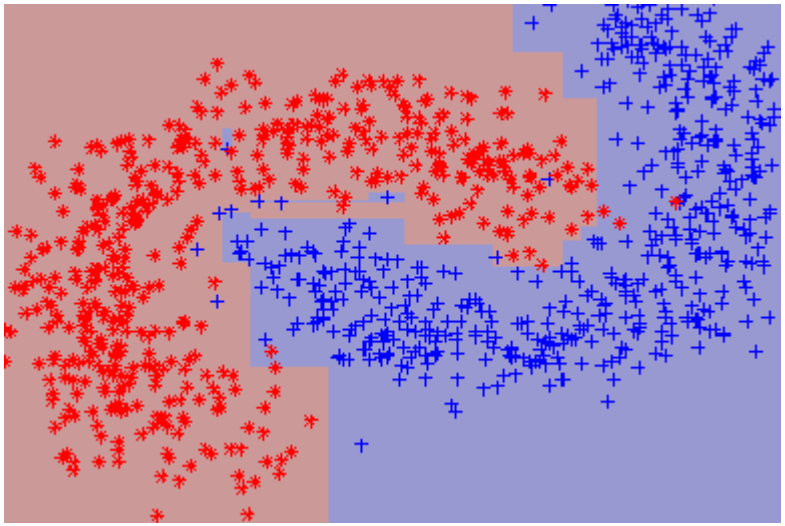
Problemas

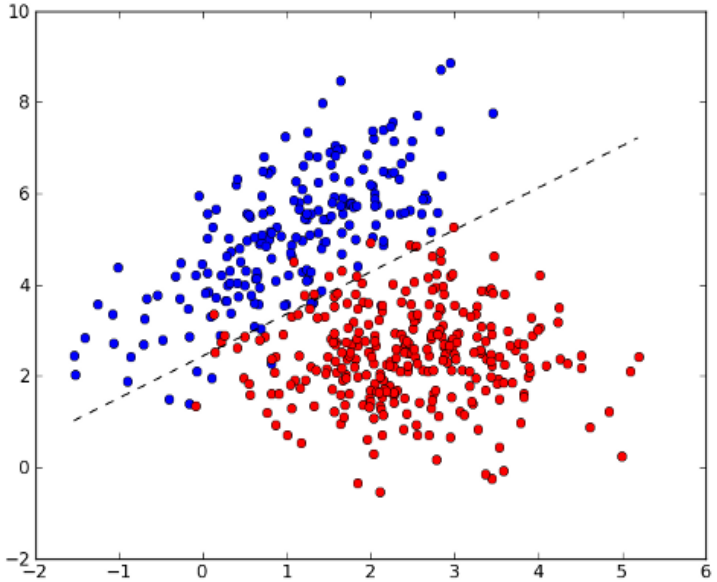
- **El primer modelo de perceptron tenía demasiadas limitaciones y se hizo con una mala reputación.**
- **Minsky y Papert expusieron las limitaciones del perceptrón.**
- **Se asumió que esto se aplicaba a todos los modelos de redes neuronales.**

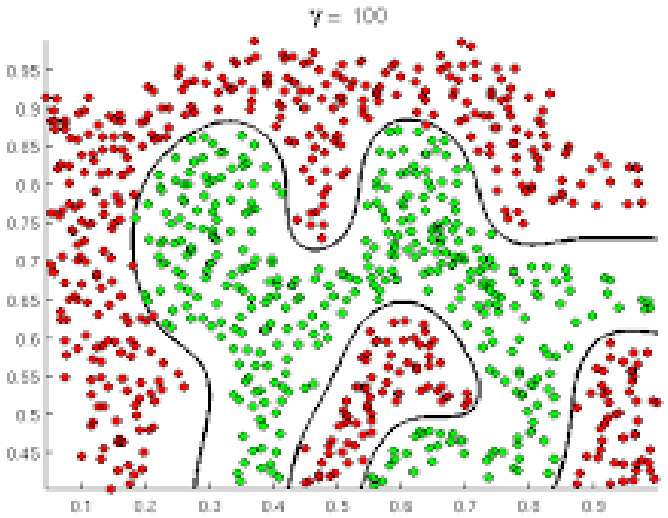
Modelos de clasificación más comunes

- KNN
- Regresión Logística
- Árbol de decisiones
- SVM
- Perceptrón multicapa
- Redes convolucionales

Clasificación lineal y no lineal







Modelos de clasificación más comunes

- **KNN**
- **Regresión Logística**
- **Árbol de decisiones**
- **SVM**
- **Perceptrón multicapa**
- **Redes convolucionales**

Perceptrón multicapa

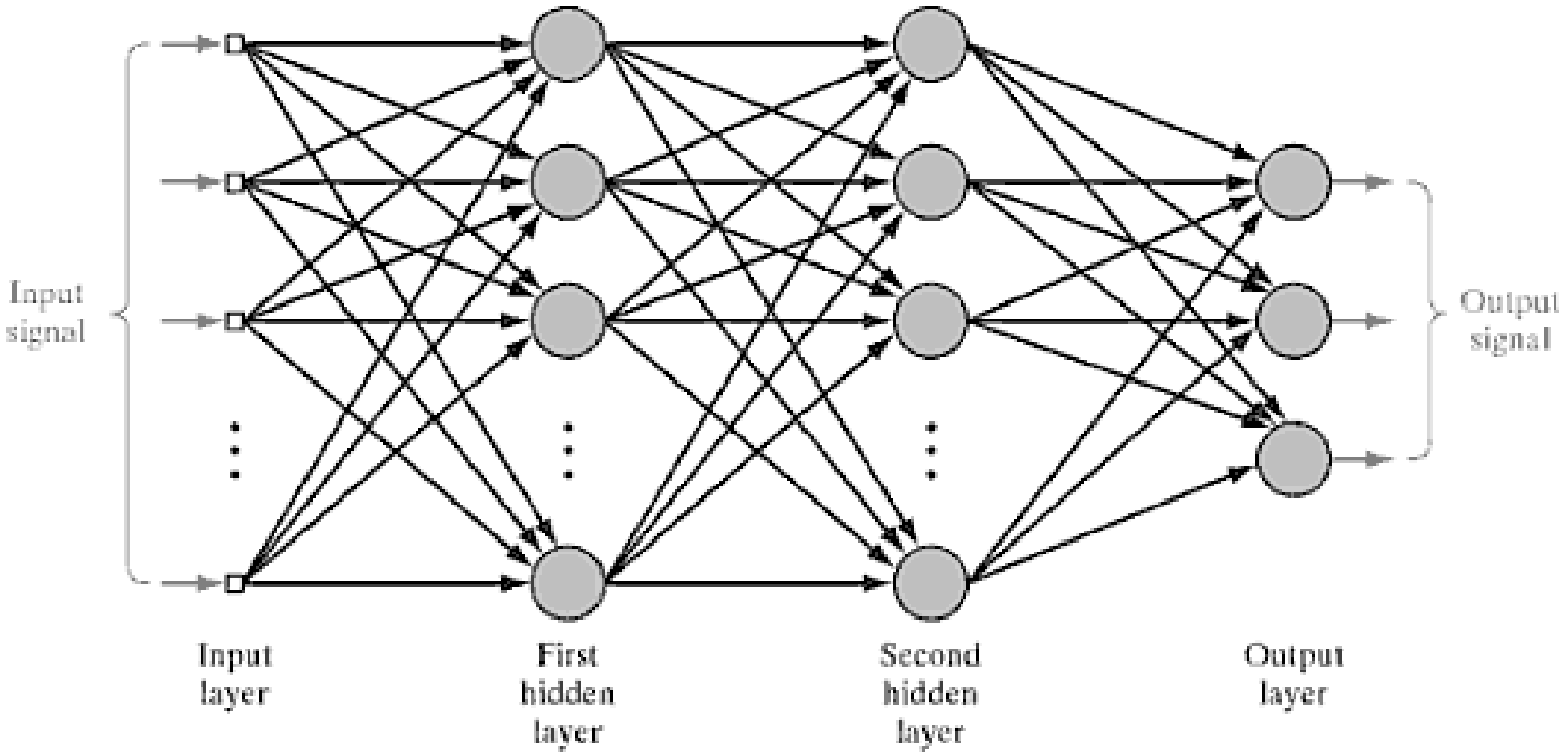


FIGURE 4.1 Architectural graph of a multilayer perceptron with two hidden layers.

El modelo

- **El aprendizaje para ajustar los valores de una red neuronal consiste en los siguientes pasos.**
  - Propagación a las siguientes capas
  - Retropropagación (calculo de la derivada del error)
  - Actualización de parámetros.

Propagación a la siguiente capa

$$Z =< W, A^{[l-1]} > +b$$

$$A^{[l]} = g(Z)$$

Donde $< W, A^{[l-1]} >$ significa producto matricial, y g(x) representa la aplicación de una función de activación

- **Una neurona contiene la información propagada de la anterior capa de neuronas a través de los pesos aplicando una activación.**
- **De esta forma se propaga el conocimiento simulando la interacción sináptica de las neuronas en nuetro cerebro.**

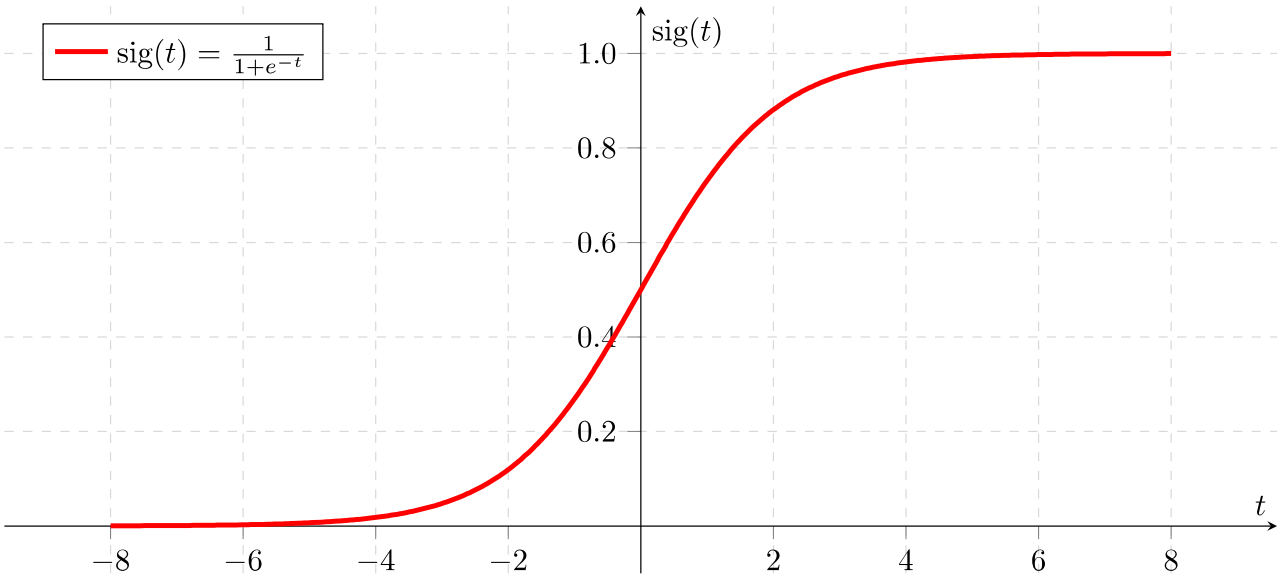
Funciones de activación

- **Sigmoid (σ)**
  - Función no lineal en rango (0,1).
  - Función altamente saturada.
- **Relu (Rectified Linear Units)**
  - Equivalente a varias unidades logísticas apiladas.
  - Es una función no linear no saturada.

Sigmoid (σ)

- **Es una función no lineal que nos devuelve una probabilidad o vector de probabilidades (valores entre 0 y 1).**

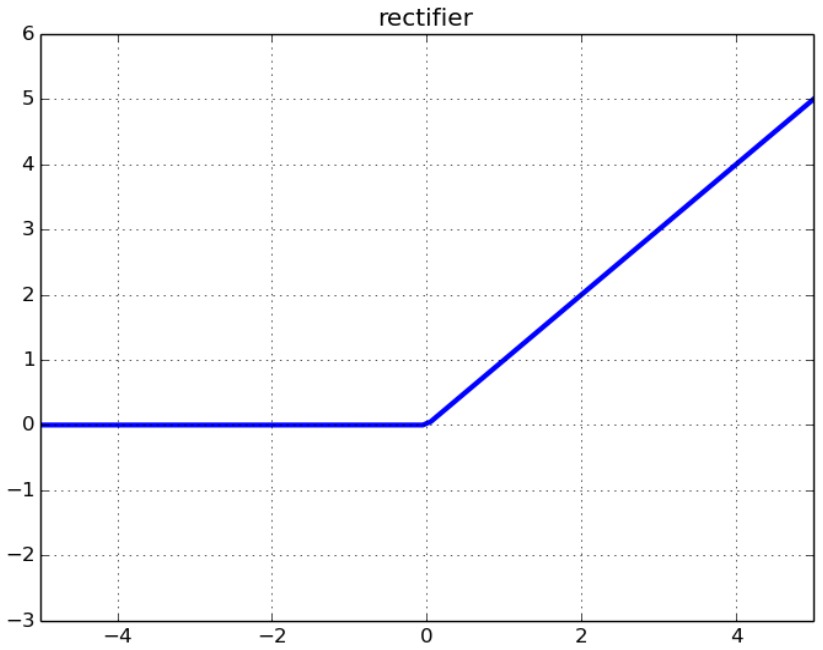
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



Relu (Rectified Linear Units)

- **Función no lineal no saturada que devuelve valores positivos.**

$$Relu(x) = max(0, x)$$



Retropropagación

- **El error de la salida se calcula usando la función de error de entropía cruzada.**

$$L = -\sum_{i=1}^{n} y_i * log|\hat{y}_i|$$

$$L = -y * log|\hat{y}| - (1 - y) * log|1 - \hat{y}|$$

- **Se deben calcular las derivadas del error con respecto a cada uno de los parámetros o pesos en las distintas capas**

Retropropagación

- **Se debe propagar el error usando derivadas iterativamente capa por capa.**

$$\frac{\partial L}{\partial Z^{[l]}} = \frac{\partial L}{\partial A^{[l]}} * \frac{\partial A^{[l]}}{\partial Z^{[l]}}$$

$$\frac{\partial L}{\partial W^{[l]}} = \frac{\partial L}{\partial Z^{[l]}} * \frac{\partial Z^{[l]}}{\partial W^{[l]}}$$

$$\frac{\partial L}{\partial b^{[l]}} = \frac{\partial L}{\partial Z^{[l]}} * \frac{\partial Z^{[l]}}{\partial b^{[l]}}$$

$$\frac{\partial L}{\partial A^{[l-1]}} = \frac{\partial L}{\partial Z^{[l]}} * \frac{\partial Z^{[l]}}{\partial A^{[l-1]}}$$

Derivadas

- **Binary cross entropy**

$$\frac{\partial L}{\partial \hat{y}} = \frac{(1-y)}{(1-\hat{y})} - \frac{y}{\hat{y}}$$

- **Sigmoid**

$$d\sigma(Z) = \sigma(Z) * (1 - \sigma(Z))$$

- **Relu**

$$dRelu(Z) = \begin{cases} 1, & \text{si } Z_i > 0 \\ 0, & \text{en otro caso} \end{cases}$$

Derivadas

- **Sustituyendo valores**

Para la última capa

$$\frac{\partial L}{\partial Z^{[l]}} = [\frac{(1-y)}{(1-\hat{y})} - \frac{y}{\hat{y}}] * [\sigma(Z) * [1-\sigma(Z)]]$$

Para otras capas

$$\frac{\partial L}{\partial Z^{[l]}} = \frac{\partial L}{\partial A^{[l]}} * \begin{cases} 1, & \text{si } Z_i > 0 \\ 0, & \text{en otro caso} \end{cases}$$

$$\frac{\partial L}{\partial W^{[l]}} = \frac{1}{m} * < \frac{\partial L}{\partial Z^{[l]}}, (A^{[l-1]})^T >$$

$$\frac{\partial L}{\partial b^{[l]}} = \frac{1}{m} * \sum_{j=1}^{batch} \frac{\partial L}{\partial Z^{[l]}}$$

$$\frac{\partial L}{\partial A^{[l-1]}} =< (W^{[l]})^T, \frac{\partial L}{\partial Z^{[l]}} >$$

Train, Dev, Test

- **Train**
  - Dataset de entrenamiento siendo una gran parte de todo el data set.
- **Dev**
  - Dataset de validación usado para mejorar la elección de hiperparámetros.
- **Test**
  - Dataset de prueba sólo utilizado al final para verificar la capacidad de predicción en datos jamás vistos.

Optimización

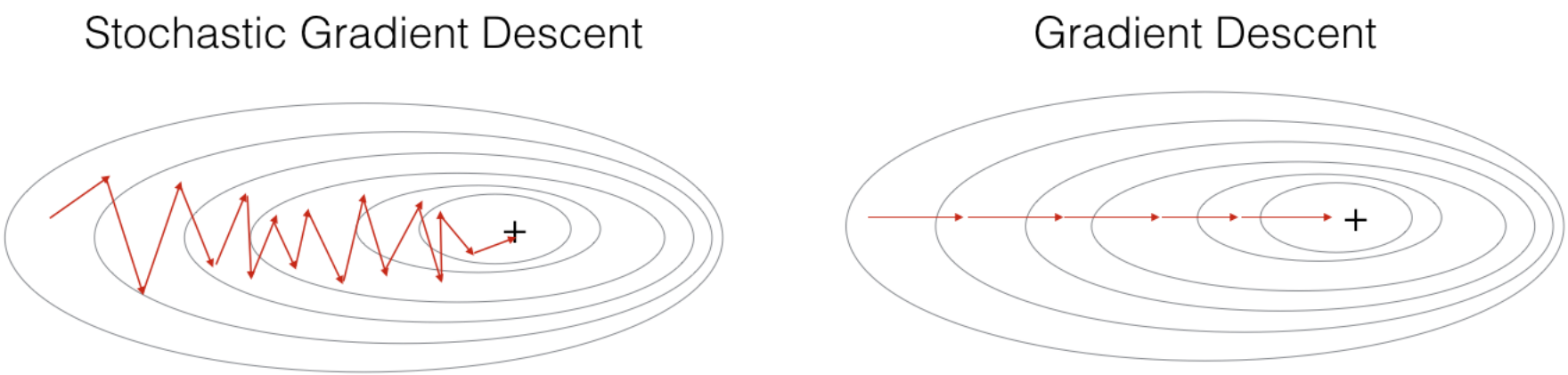
- **Gradient descent**
- **Stochastic gradient descent**
  - Divide el training set en batches (pedazos de tamaño definido).
- **Adam**
  - Usa dos momentos, para suavizar la oscilación de los gradientes.
  - Usa correción de sesgo de inicialización.

Optimización

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

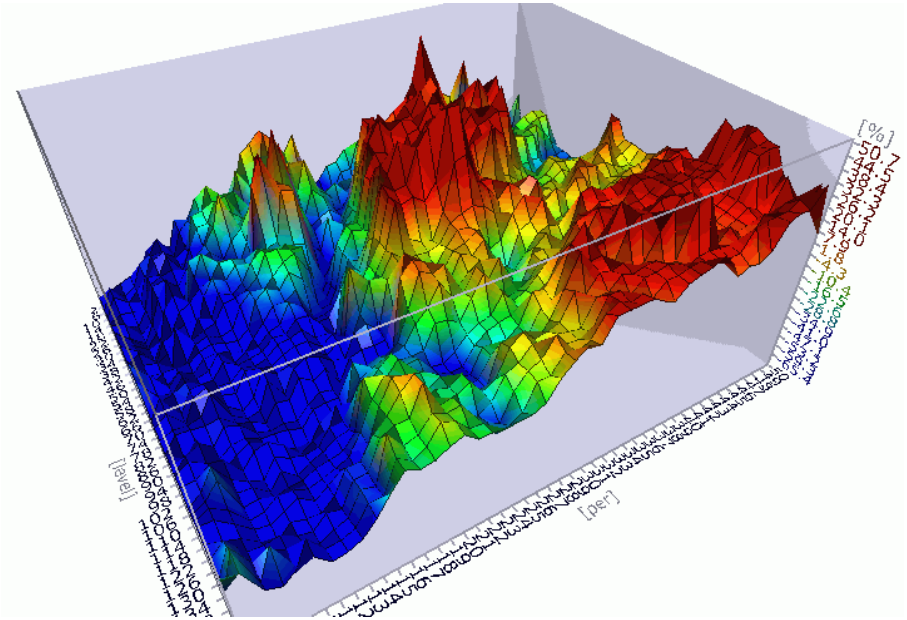
Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1$^{\text{st}}$ moment vector)
$v_0 \leftarrow 0$ (Initialize 2$^{\text{nd}}$ moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

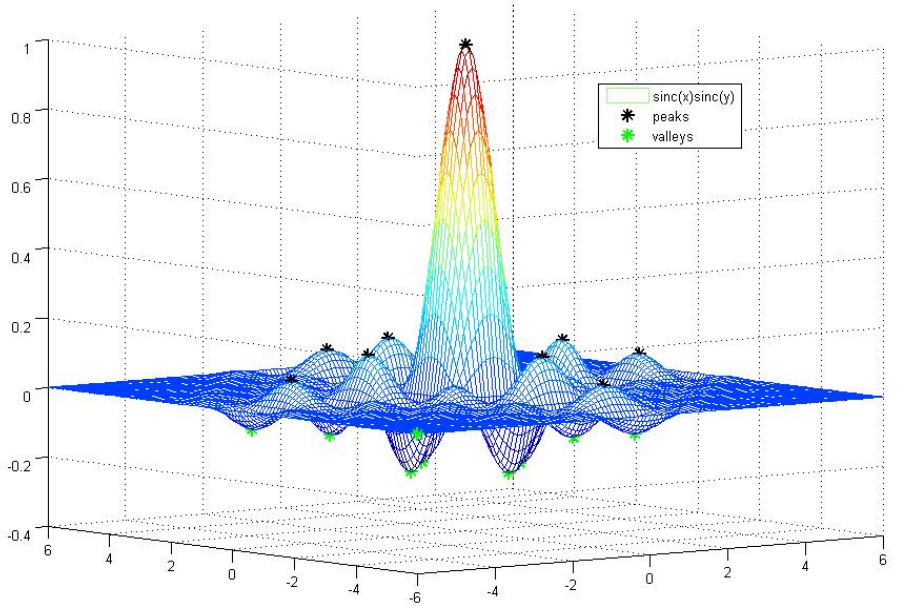
Optimización



- **Función de coste o pérdida**
  - Para neuronas lineales es un paraboloide.
  - Para funciones de activación no lineales multicapa, se vuelve una superficie hiperdimensional llena de minimos locales.

Optimización





Sobre-entrenamiento y sub-entrenamiento

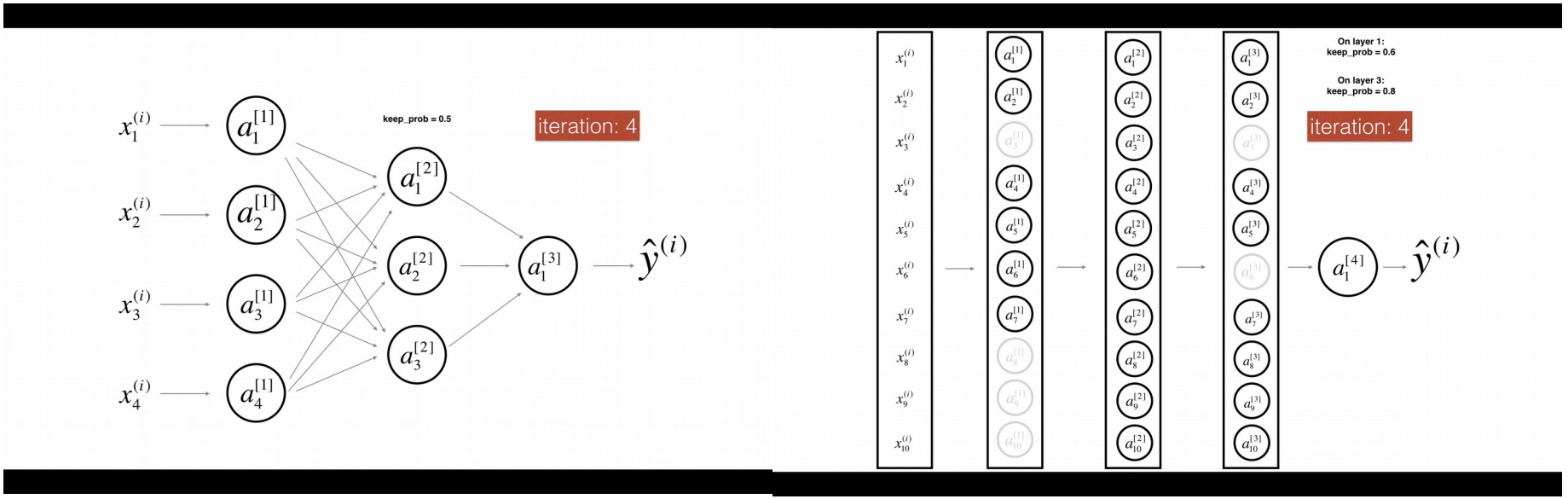
- **Sobre-entrenamiento**
  - Ocurre cuando la función de aproximación se memoriza los datos y no generaliza correctamente.
- **Sub-entrenamiento**
  - Ocurre cuando la función de aproximación no se entrena lo suficiente y no aprende las cualidades de los datos, siendo incapaz de realizar inferencias.

Regularización

- **Un método muy útil para combatir el sobre entrenamiento es la regularización**
- **Regularización L2**
  - Permite al modelo que automáticamente limite las neuronas no útiles mediante sus gradientes.
  - Hiperparámetro extra
  - Aproxima los valores a 0 para evitar sobre entrenamiento

Regularización

- **Dropout**
  - Apaga neuronas aleatoriamente en cada iteración
  - Permite aprender múltiples modelos en una sola sesión de entrenamiento.



Referencias

- **Backpropagation (Nature)**
  - https://www.iro.umontreal.ca/~vincentp/ift3395/lectures/backprop_old.pdf
- **Relu**
  - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.165.6419&rep=rep1&type=pdf
  - https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf
- **Adam**
  - https://arxiv.org/pdf/1412.6980.pdf
- **Dropout**
  - https://arxiv.org/pdf/1207.0580.pdf