Clase 2.A

Redes Neuronales Artificiales

Enzo Ferrante

eferrante@sinc.unl.edu.ar

@enzoferrante

Universidad de SanAndrés

UNL CONICET

s i n c (i)

Neurona biológica

Schematic of a biological neuron.

Fuente: https://sebastianraschka.com/Articles/2015_singlelayer_neurons.html

Perceptrón

(Rosenblatt, 1958)

Psychological Review
Vol. 65, No. 6, 1958

THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN[1]

F. ROSENBLATT

Cornell Aeronautical Laboratory

If we are eventually to understand the capability of higher organisms for perceptual recognition, generalization, recall, and thinking, we must first have answers to three fundamental questions:

1. How is information about the physical world sensed, or detected, by the biological system?
2. In what form is information stored, or remembered?
3. How does information

and the stored pattern. According to this hypothesis, if one understood the code or "wiring diagram" of the nervous system, one should, in principle, be able to discover exactly what an organism remembers by reconstructing the original sensory patterns from the "memory traces" which they have left, much as we might develop a photographic negative, or translate the pattern of electrical

Neurona artificial

Neurona artificial

Neurona artificial

Neurona artificial

(Bias trick)

Perceptrón simple

Perceptrón simple

$$y = f(\sum_{i=1}^{d} w_i x_i - b) = f(\sum_{i=0}^{d} w_i x_i) = f(\langle \mathbf{w}, \mathbf{x} \rangle)$$

Funciones de activación

Función signo

$$y = f(\langle \mathbf{w}, \mathbf{x} \rangle)$$

$$y = \mathrm{sgn}(\langle \mathbf{w}, \mathbf{x} \rangle)$$

$$\mathrm{sgn}(z) = \begin{cases} -1 & \text{si } z < 0 \\ +1 & \text{si } z \geq 0 \end{cases}$$

Funciones de activación

Función signo lineal

$$y = f(\langle \mathbf{w}, \mathbf{x} \rangle)$$

$$y = \mathrm{sln}(\langle \mathbf{w}, \mathbf{x} \rangle)$$

$$\mathrm{sln}(z) = \begin{cases} -1 & \text{si } z < -a \\ \alpha z & \text{si } -a < z < a \\ +1 & \text{si } z \geq a \end{cases}$$

Funciones de activación

Función Sigmoidea

$$y = f(\langle \mathbf{w}, \mathbf{x} \rangle)$$

$$y = \sigma(\langle \mathbf{w}, \mathbf{x} \rangle)$$

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

Funciones de activación

Ejemplo

Perceptrón simple con 2 entradas

Ejemplo

Representación gráfica de la frontera de decisión

$$w_1 x_1 + w_2 x_2 = 0$$

$$x_2 = -\frac{w_1}{w_2} x_1$$

Ej:

$$w_1 = 1, w_2 = 1$$

$$x_2 = -x_1$$

$$y = \mathrm{sgn}(x_1 w_1 + x_2 w_2)$$

Ejemplo

Representación gráfica de la frontera de decisión al clasificar imágenes

Ejemplo

Modelando la operación **x1 OR x2**

A	B	Out
0	0	0
0	1	1
1	0	1
1	1	1

Fuente: http://hyperphysics.phy-astr.gsu.edu/hbase/Electronic/

Ejemplo

Modelando la operación **x1 OR x2** (cambiando 0 por -1)

$$w_1 x_1 + w_2 x_2 = 0$$

Se necesita un término de bias

$$w_1 x_1 + w_2 x_2 - w_0 = 0$$

$$-\frac{w_1}{w_2} x_1 + \frac{w_0}{w_2} = x_2$$

Ej: $w_1 = w_2 = 1, w_0 = -1$

$$-x_1 - 1 = x_2$$

Ejemplo

Perceptrón simple con 2 entradas y bias

Ejemplo

Perceptrón simple con 2 entradas y bias modelando la operación **x1 OR x2**

Cómo entrenar mi perceptrón?

Cómo entrenar mi perceptrón?

Dataset de entrenamiento

$$\mathcal{D} = \{(\mathbf{x}, d)_n\} = \{(\mathbf{x}, d)_1, ...(\mathbf{x}, d)_N\}$$

En nuestro problema de ejemplo: $\mathcal{D} =$

x = (-1,-1)	*d* = -1
x = (1,-1)	*d* = 1
x = (-1,1)	*d* = 1
x = (1,1)	*d* = 1

Cómo entrenar mi perceptrón?

Función de pérdida (o función de costo)

- En el contexto del aprendizaje automática, es una función que **mide la calidad de una predicción basándose en el acuerdo entre dicha predicción y la etiqueta provista** en el dataset de entrenamiento

- La función de costo **devuelve un valor alto si nuestra predicción es incorrecta, y bajo si la predicción es correcta**

- Sirve como **guía en el proceso de búsqueda de los parámetros** de nuestro modelo a entrenar

Cómo entrenar mi perceptrón?

Función de pérdida

$$\mathcal{L}((\mathbf{x}, d); \mathbf{w})$$

Función de pérdida

Error cuadrático instantáneo

$$\mathcal{L}((\mathbf{x}, d); \mathbf{w}) = (d - \boxed{\langle \mathbf{w}, \mathbf{x} \rangle})^2$$

Caso Lineal
(sin función de activación)

Aprendizaje como un problema de optimización

Algoritmo de optimización

Función de pérdida

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}((\mathbf{x}, d)_n; \mathbf{w})$$

Parámetros del modelo entrenado

Estrategias para buscar los parámetros óptimos

1. Búsqueda aleatoria

$\mathcal{L}_{\text{Best}}$ = **inf**

for i = 1 to #iteraciones:

$\mathbf{w_i}$ = generar pesos aleatorios

$$\hat{\mathcal{L}} = \frac{1}{N}\sum_{n=1}^{N}\mathcal{L}((\mathbf{x}, d)_n; \mathbf{w_i})$$

If $\hat{\mathcal{L}} < \mathcal{L}_{\text{Best}}$

$\mathcal{L}_{\text{Best}} = \hat{\mathcal{L}}$

$\mathbf{w_{best}} = \mathbf{w_i}$

Y si en lugar de buscar aleatoreamente vamos refinando en cada paso?

Estrategias para buscar los parámetros óptimos

2. Búsqueda aleatoria en un entorno local

$\mathcal{L}_{\text{Best}}$ = **inf,** **$\mathbf{w}_0$** = generar pesos aleatorios

for i = 1 to #iteraciones:

[$\mathbf{w}_{\text{local}}$] = generar pesos aleatorios alrededor de **$\mathbf{w}_{\text{best}}$**

Calcular $\hat{\mathcal{L}} = \frac{1}{N}\sum_{n=1}^{N}\mathcal{L}((\mathbf{x}, d)_n; \mathbf{w}_{\text{local}})$ por cada **[$\mathbf{w}_{\text{local}}$]** y elegir minimo

If $\hat{\mathcal{L}}$ **<** $\mathcal{L}_{\text{Best}}$

$\mathcal{L}_{\text{Best}}$ = $\hat{\mathcal{L}}$

$\mathbf{w}_{\text{best}}$ = $\mathbf{w}_{\text{local}}$

Estrategias para buscar los parámetros óptimos

3. Siguiendo el gradiente

Actualizar los pesos en la dirección en **que garantice una reducción de la función de pérdida**

El vector **gradiente** apunta en la dirección de mayor crecimiento de la función

$$\nabla_{\mathbf{w}}\mathcal{L} = \left(\boxed{\frac{\partial \mathcal{L}}{\partial w_1}}, \ldots, \frac{\partial \mathcal{L}}{\partial w_D}\right)$$

Indica cómo cambiará el valor de la función en un entorno infinitesimalmente pequeño alrededor de un punto al variar el peso w_1

Estrategias para buscar los parámetros óptimos

3. Siguiendo el gradiente

- El método del gradiente descendente **garantiza la convergencia a un mínimo local** de la función a optimizar.

- Si la función es **convexa**, **el mínimo local es a su vez el mínimo global.**

- El **error cuadrático** utilizado como función de pérdida del **perceptrón simple en el caso lineal** resulta en una función **convenxa**.

Método del gradiente descendente

Tasa de aprendizaje

1. Inicializar los pesos **W** aleatoriamente

2. Dado el dataset $\mathcal{D} = \{(\mathbf{x}, d)_n\} = \{(\mathbf{x}, d)_1, ...(\mathbf{x}, d)_N\}$

Actualizar en
la dirección opuesta
al gradiente de la
pérdida promedio

$$\mathbf{w} = \mathbf{w} - \boxed{\delta} \nabla_{\mathbf{w}} \boxed{\mathcal{L}(\mathcal{D}; \mathbf{w})}$$

3. Iterar hasta satisfacer algún criterio de convergencia

$$\boxed{\mathcal{L}(\mathcal{D}; \mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}((\mathbf{x}, d)_n; \mathbf{w})}$$

Sumatoria sobre todos los datos de entrenamiento $\mathcal{D} = \{(\mathbf{x}, d)_n\} = \{(\mathbf{x}, d)_1, ...(\mathbf{x}, d)_N\}$

Ejemplo de cálculo del gradiente

Caso lineal del perceptrón

Función de pérdida

$$\mathcal{L}((\mathbf{x}, d); \mathbf{w}) = (d - \langle \mathbf{w}, \mathbf{x} \rangle)^2$$

Gradiente de la función de pérdida

$$\nabla \mathcal{L}_{\mathbf{w}} = 2(d - \langle \mathbf{w}, \mathbf{x} \rangle)(-\mathbf{x})$$

Método del gradiente descendente

1. Inicializar los pesos **W** aleatoriamente
2. Dado el dataset $\mathcal{D} = \{(\mathbf{x}, d)_n\} = \{(\mathbf{x}, d)_1, ...(\mathbf{x}, d)_N\}$

   Actualizar en la dirección opuesta al gradiente de la pérdida promedio

$$\mathbf{w} = \mathbf{w} - \delta \nabla_{\mathbf{w}} \mathcal{L}(\mathcal{D}; \mathbf{w})$$

3. Iterar hasta satisfacer algún criterio de convergencia

Método del gradiente descendente

1. Inicializar los pesos **W** aleatoriamente

2. Dado el dataset $\mathcal{D} = \{(\mathbf{x}, d)_n\} = \{(\mathbf{x}, d)_1, ...(\mathbf{x}, d)_N\}$

   Actualizar en la dirección opuesta al gradiente de la pérdida promedio

$$\mathbf{w} = \mathbf{w} - \delta \frac{1}{N} \sum_{n=1}^{N} 2(d_i - \langle \mathbf{w}, \mathbf{x}_i \rangle)(-\mathbf{x}_i)$$

3. Iterar hasta satisfacer algún criterio de convergencia

$\mathcal{D} =$

x = (-1,-1)	*d* = -1
x = (1,-1)	*d* = 1
x = (-1,1)	*d* = 1
x = (1,1)	*d* = 1

El perceptrón simple puede modelar el XOR?

A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

Fuente: http://hyperphysics.phy-astr.gsu.edu/hbase/Electronic/

El perceptrón simple puede modelar el XOR?

Perceptrón Multicapa

Multi layer perceptron (MLP)

Regiones de decisión del MLP

Estructura	*Tipos de regiones de decisión*	*Problema XOR*	*Separación en clases*	*Formas regiones más generales*
Una capa	*hemiplano limitado por hiperplano*			
Dos capas	*Regiones convexas abiertas o cerradas*			
Tres capas	Arbitrarias (Complejidad limitada por N°. de Nodos)			

Teorema de aproximación universal

Hornik et al, 1989

Neural Networks, Vol. 2, pp. 359–366, 1989
Printed in the USA. All rights reserved.

0893-6080/89 $3.00 + .00
Copyright © 1989 Pergamon Press plc

ORIGINAL CONTRIBUTION

Multilayer Feedforward Networks are Universal Approximators

KURT HORNIK

Technische Universität Wien

MAXWELL STINCHCOMBE AND HALBERT WHITE

University of California, San Diego

(*Received* 16 *September* 1988; *revised and accepted* 9 *March* 1989)

Abstract—*This paper rigorously establishes that standard multilayer feedforward networks with as few as one hidden layer using arbitrary squashing functions are capable of approximating any Borel measurable function from one finite dimensional space to another to any desired degree of accuracy, provided sufficiently many hidden units are available. In this sense, multilayer feedforward networks are a class of universal approximators.*

Keywords—Feedforward networks, Universal approximation, Mapping networks, Network representation capability, Stone-Weierstrass Theorem, Squashing functions, Sigma-Pi networks, Back-propagation networks.

any function encountered in applications leads one
about the ultimate capabilities of such

Teorema de aproximación universal

Hornik et al, 1989

Establece que una **red feed-forward con una sola capa intermedia** es suficiente para **aproximar, con una precisión arbitraria, cualquier función** con un número finito de discontinuidades, **siempre y cuando las funciones de activación de las neuronas ocultas sean no lineales**.

Es un teorema de existencia (dice que la solución existe pero no dice que sea facil encontrarla)

Función de pérdida no-convexa

Perceptrón simple, caso lineal
Convexa

Perceptrón multicapa con
funciones de activación no lineales
No convexa

Cómo entrenar mi perceptrón multicapa?

Función de pérdida no-convexa en MLP

- Gradient descent **no garantiza encontrar el mínimo global** en funciones no-convexas
- Sin embargo, **para redes neuronales grandes, la mayoría de los mínimos locales son similares** y presentan **performance similar en los dataset de test**
- La probabilidad de encontrar mínimos locales 'malos' decrese con el tamaño de la red
- **Focalizarse demasiado en la búsqueda del mínimo global en el dataset de entrenamiento** no es últil en la práctica, y puede terminar en **sobreajuste** del modelo

Choromanska et al. *"The Loss Surfaces of Multilayer Networks".* AISTATS 2015.

Método del gradiente para MLP

Matriz con un vector de pesos $\mathbf{w}_i$ por capa

1. Inicializar los pesos **W** aleatoriamente

2. Dado el dataset $\mathcal{D} = \{(\mathbf{x}, d)_n\} = \{(\mathbf{x}, d)_1, \ldots (\mathbf{x}, d)_N\}$

Actualizar en
la dirección opuesta
al gradiente de la
pérdida promedio

$$\mathbf{W} = \mathbf{W} - \delta \nabla_{\mathbf{W}} \mathcal{L}(\mathcal{D}; \mathbf{W})$$

3. Volver a 2 hasta satisfacer algún criterio de convergencia

$$\mathcal{L}(\mathcal{D}; \mathbf{W}) = \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}((\mathbf{x}, d)_n; \mathbf{W})$$

Cómputo del gradiente sobre el
dataset de entrenamiento
completo

Variantes del método del gradiente

- **Gradiente descendiente clásico**
- **Gradiente descendiente estocástico**
- **Gradiente descendiente por mini-batches**

Método del gradiente descendiente por mini-batches

1. Inicializar los pesos **w** aleatoriamente

2. Dado el dataset $\mathcal{D} = \{(\mathbf{x}, d)_n\} = \{(\mathbf{x}, d)_1, ...(\mathbf{x}, d)_N\}$

Generar mini-batch $\mathcal{D}_M = \{(\mathbf{x}, d)_m\} = \{(\mathbf{x}, d)_1, ...(\mathbf{x}, d)_M\}, M < N$

Actualizar en
la dirección opuesta
al gradiente de la
pérdida promedio

$$\mathbf{W} = \mathbf{W} - \delta \nabla_{\mathbf{W}} \mathcal{L}(\mathcal{D}_M; \mathbf{W})$$

3. Volver a 2 hasta satisfacer algún criterio de convergencia

$$\mathcal{L}(\mathcal{D}_M; \mathbf{W}) = \frac{1}{M} \sum_{m=1}^{M} \mathcal{L}((\mathbf{x}, d)_m; \mathbf{W})$$

Cómputo del gradiente sobre el
mini-batch de tamaño M < N

Resumen de percetrón

Perceptrón
con 2 neuronas
de salida

3 muestras
con solo una feature
cada una

Imagen extraída de: https://ml-cheatsheet.readthedocs.io/en/latest/forwardpropagation.html

Resumen de percetrón

4 muestras
con 2 features
de entrada cada una

Resumen de percetrón

1 muestras
con 2 features
de entrada

El tamaño de las matrices
W es independiente
de la cantidad de
elementos en el batch!

Cómo calcular el gradiente?

- **1. Derivación analítica**: derivar a mano y escribir el código

- **2. Derivación numérica :** diferencias finitas

- **3. Derivación simbólica:** se realiza utilizando las reglas estudiadas en Análisis matemático pero automatizadas (ej: Maple, Mathematica)

Backpropagation

- **4. Derivación automática:**
  - Se definen las derivadas para las operaciones 'primitivas' (matemáticas y de control)
  - Se construye un grafo de operaciones y se deriva siguiendo la regla de la cadena.

Baydin et al. *"Automatic differentiation in machine learning: a survey".* Journal of Machine Learning Research. 2017.

Cálculo del gradiente en redes feed forward

Backpropagation (o algortímo de retropropagación)

Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton† & Ronald J. Williams*

* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure[1].

There have been many attempts to design self-organizing neural networks. The aim is to find a powerful synaptic modification rule that will allow an arbitrarily connected neural network to develop an internal structure that is appropriate for [illegible] specified by giving the

more difficult when we introduce hidden units whose actual or desired states are not specified by the task. (In perceptrons, there are 'feature analysers' between the input and output that are not true hidden units because their input connections are fixed by hand, so their states are completely determined by the input vector: they do not learn representations.) The learning procedure must decide under what circumstances the hidden units should be active in order to help achieve the desired input–output behaviour. This amounts to deciding what these units should represent. We demonstrate that a general purpose and relatively simple procedure is powerful enough to construct appropriate internal representations.

The simplest form of the learning procedure is for layered networks which have a layer of input units at the bottom; any number of intermediate layers; and a layer of output units at the top. Connections within a layer or from higher to lower layers are forbidden, but connections can skip intermediate layers. An input vector is presented to the network by setting the states of the input units. Then the states of the units in each layer are determined by applying equations (1) and (2) to the connections coming from lower layers. All units within a layer have their states set in parallel, but different layers have their states set sequentially, starting at the bottom and working upwards until the states of the output units are determined.

The total input, x_j, to unit j is a linear function of the outputs, y_i, of the units that are connected to j and of the weights, w_{ji}, on these connections

$$x_j = \sum_i y_i w_{ji} \tag{1}$$

[illegible] be given biases by introducing an extra input to each [illegible] this extra

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533.

Backpropagation

Empecemos por recordar la definición de derivada

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Cuando h es muuuuy pequeña, la función puede ser bien aproximada por una recta.
La pendiente de esa recta, es la derivada.

Intuición de derivada

La derivada nos indica entonces **la 'sensibilidad' de la función al cambio en la variable**. Esto se observa al reordenar la fórmula (considerando un h lo suficientemente pequeño):

$$f(x+h) = f(x) + h\frac{df(x)}{dx}$$

Ejemplo producto: $f(x,y) = xy \qquad \frac{df}{dy} = x \qquad \frac{df}{dx} = y$

Si tomamos *x=4* e *y=-3*, qué pasa con *f* , alrededor de ese punto, al variar x?

Al variar el valor de *x* una pequeña cantidad *h*, el valor de la función f se decrementará 3h.

Intuición de derivada

La derivada nos indica entonces **la 'sensibilidad' de la función al cambio en la variable**.

Ejemplo suma:

$$f(x,y) = x + y \qquad \frac{df}{dy} = 1 \qquad \frac{df}{dx} = 1$$

Si tomamos *x=4* e *y=-3*, qué pasa con *f* , alrededor de ese punto, al variar x e y?

En ambos casos la función se incrementa de igual forma, independientemente del valor de *x* y de *y* (a diferencia del caso del producto).

Intuición de derivada

La derivada nos indica entonces **la 'sensibilidad' de la función al cambio en la variable**.

Ejemplo Máximo: $$f(x, y) = \max(x, y)$$

$$\frac{\partial f}{\partial x} = \mathbb{1}(x >= y)$$

$$\frac{\partial f}{\partial y} = \mathbb{1}(y >= x)$$

Función no diferenciable, por lo tanto hablamos de **sub-gradiente** → *Generalización del gradiente para funciones no diferenciables*

Cálculo del gradiente en redes feed forward

Backpropagation (o algortímo de retropropagación)

Regla de la cadena

$f = (x + y) * z$

$q = x + y$

$f = q * z$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

Pizarrón!

Cálculo del gradiente en redes feed forward

Backpropagation (o algortímo de retropropagación)

Nuestro enfoque: Backpropagation como flujo de gradientes encadenados en un circuito de operaciones aritméticas

(Explicación basada en http://cs231n.github.io)

$f = (x + y) * z$

$q = x + y$

$f = q * z$

Flujo de gradientes

Flujo de gradientes

De esta forma, las operaciones a derivar de forma analítica se mantienen simples

Esta regla se aplica recursivamente, hacia atrás, durante la fase backward.

Ejemplos en pizarrón

$$f(\mathbf{w}, \mathbf{x}) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$

$$f(x, y, z, w) = ((xy) + \max(z, w)) * 2$$

Flujo de gradientes

Cuando la salida de un nodo es compartida, los gradientes retropropagados se suman antes de multiplicarlos con el gradiente local

Patrones en el grafo de operaciones

- La **suma** distribuye el gradiente de igual forma a todas sus entradas.
- La operación **max** envía el gradiente sólo hacia la mayor entrada.
- La operación **producto** multiplica la otra entrada por el gradiente retropropagado.

Frameworks que implementan diferenciación automática

TensorFlow

PYTORCH

Gradiente descendiente por mini-batches + backpropagation

1. Inicializar los pesos **w** aleatoriamente

2. Dado el dataset $\mathcal{D} = \{(\mathbf{x}, d)_n\} = \{(\mathbf{x}, d)_1, ...(\mathbf{x}, d)_N\}$

   Generar mini-batch $\mathcal{D}_M = \{(\mathbf{x}, d)_m\} = \{(\mathbf{x}, d)_1, ...(\mathbf{x}, d)_M\}, M < N$

   Actualizar en
   la dirección opuesta
   al gradiente de la
   pérdida promedio

$$\mathbf{W} = \mathbf{W} - \boxed{\delta \nabla_{\mathbf{W}} \mathcal{L}(\mathcal{D}_M; \mathbf{W})}$$

3. Volver a 2 hasta satisfacer algún criterio de convergencia

En la práctica se utiliza gradiente descendiente por minibatches, y el gradiente es calculado utilizando backprogation

Clase 2.A

Redes Neuronales Artificiales

Enzo Ferrante

eferrante@sinc.unl.edu.ar

@enzoferrante

Universidad de SanAndrés

UNL CONICET

s i n c (i)