Redes Neuronales Multicapa

Reconocimiento de Patrones

Departamento de Procesamiento de Señales
Instituto de Ingeniería Eléctrica
Facultad de Ingeniería, UdelaR

2018

Bishop, Cap. 5 – Duda et al., Cap. 6

Outline

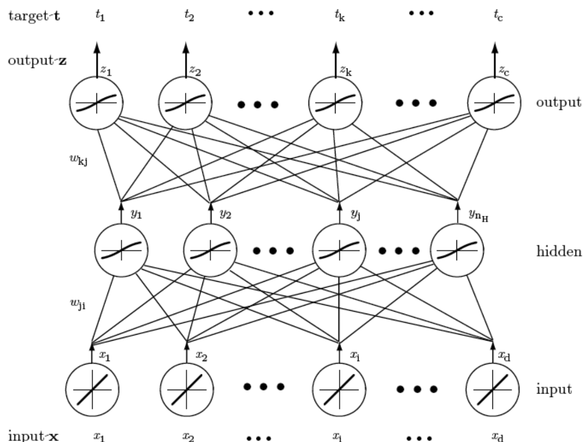
Redes neuronales multicapa (MLNN)

- Las funciones discriminantes lineales puras tienen limitaciones: las fronteras de decisión son hiperplanos.
- Con funciones no lineales ϕ se pueden obtener fronteras de decisión complejas: mayor poder de expresión.
- ¿Cómo determinar o aprender las no linealidades ϕ para lograr bajos errores de clasificación?

Idea de MLNN: aprender las no-linealidades al mismo tiempo que los discriminantes lineales.

Estructura de una red multicapa

Ejemplo: red de 3 capas – entrada, capa oculta y capa de salida.



Redes de 3 capas

Salida de la capa de entrada:

$$a_j = \sum_{i=1}^{d} w_{ji}^{(1)} x_i + w_{j0}^{(1)} = \sum_{i=0}^{d} w_{ji}^{(1)} x_i + w_{j0}^{(1)} = \mathbf{w}_j^T \mathbf{x}, \quad \text{con } x_0 = 1.$$

$y_j = f(a_j)$, e.g. f : función ~~signo~~ o sigmoide (veremos otras).

Unidades de la capa de salida: idem pero tomando las salidas de la capa oculta como entradas:

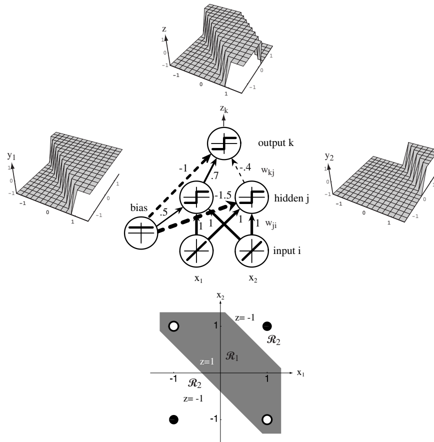
$$a_k = \sum_{j=1}^{n_H} w_{kj}^{(2)} y_j + w_{k0}^{(2)} = \mathbf{w}^T \mathbf{y}$$

$z_k = f(a_k)$, f : en general (no necesariamente) igual a las de la capa oculta.

a_j, a_k : activaciones; f : funciones de activación.

Redes de 3 capas

Ejercicio Problema XOR: mostrar que se resuelve con esta red.



(2-2-1 fully connected)

Funciones *feedforward*

La forma general de las salidas de una red de tres capas es:

$$z_k(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=1}^{n_H} w_{kj}^{(2)} f\left(\sum_{i=1}^{d} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

La evaluación de esta función se puede ver como una propagación de información en una red, hacia adelante (*feedforward*).

Interesa saber la capacidad expresiva de este modelo: **¿Se puede aproximar cualquier función?**

Poder de expresión

Las redes neuronales multicapa son **aproximadores universales**:

Teorema de Kolmogorov: Para toda función $g : [0,1]^n \to \mathbb{R}$ continua ($n \geq 2$), existen funciones Ξ_j y ψ_{ij} tales que

$$g(\mathbf{x}) = \sum_{j=1}^{2n+1} \Xi_j \left(\sum_{i=1}^{d} \psi_{ij}(x_i) \right).$$

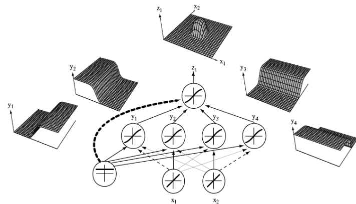
- Viabilidad teórica, poco utilidad práctica: las Ξ_j y ψ_{ij} son complejas y muy irregulares, no sabemos como elegirlas ni aprenderlas de los datos, ni determinar n.

Teorema de Cybenko 1989, Hornik 1991: con una única capa oculta con número finito de neuronas, y condiciones débiles sobre f, se puede aproximar cualquier función continua en un compacto.

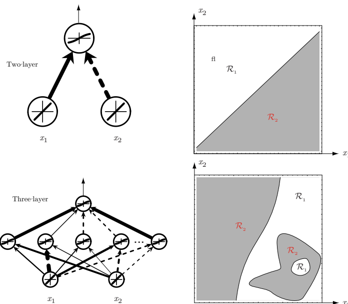
- Viabilidad teórica, poco utilidad práctica: ¿Cómo optimizamos o determinamos los parámetros?

Aproximadores universales

Intuitición



Poder de expresión



Entrenamiento de la red

- c clases. Muestras etiquetadas iid: $\{\mathbf{x}_n, \mathbf{t}_n\}$, $\mathbf{t}_n = (t_{n1}, t_{n2}, \ldots, t_{nc})^T$.
- Salidas de la red: $\mathbf{z}_n = (z_{n1}, z_{n2}, \ldots, z_{nc})^T$, $z_{nk} = g_k(\mathbf{x}_n)$.

Dos funciones de costo típicas son: Norma L_2 y entropía cruzada.

Norma L_2 (para regresión y clasificación):

$$J(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \|\mathbf{z}_n - \mathbf{t}_n\|^2.$$

Corresponde al estimador ML: $p(\mathbf{t}|\mathbf{x}, \mathbf{w}) = \prod_{n=1}^{N} \mathcal{N}(\mathbf{t}|\mathbf{z}, \mathrm{I})$.

Entrenamiento de la red

Entropía cruzada (para clasificación; training más rápido y mejor performance):

(i) Dos clases $\mathcal{C}_1$, $\mathcal{C}_2$: $t_n = 1$ si $x_n \in \mathcal{C}_1$, $t_n = 0$ si $x_n \in \mathcal{C}_2$.

- Una sola salida con sigmoide:
  $z(\mathbf{x}, \mathbf{w}) = \sigma(a(\mathbf{x}, \mathbf{w})) = \frac{1}{1+\exp(-a)}$.
- Podemos interpretar
  $z_n = \Pr(\mathcal{C}_1|\mathbf{x}_n), 1 - z_n = 1 - \Pr(\mathcal{C}_1|\mathbf{x}_n) = \Pr(\mathcal{C}_2|\mathbf{x}_n)$.
- Con esto la verosimilitud es $p(t_n|\mathbf{x}_n, \mathbf{w}) = z_n^{t_n}(1 - z_n)^{1-t_n}$ y la verosimilitud de toda la muestra es $\prod_{n=1}^{N} p(t_n|\mathbf{x}_n, \mathbf{w})$.
- El estimador ML corresponde a minimizar

$$J(\mathbf{w}) = -\sum_{n=1}^{N}\{t_n \ln z_n + (1 - t_n)\ln(1 - z_n)\}.$$

Entrenamiento de la red

(ii) c **clases** $\mathcal{C}_k$**:** $\mathbf{t}_n = (t_{n1}, t_{n2}, \ldots, t_{nc})^T$**,** $t_{nk} = \mathbb{1}_{[\mathbf{x}_n \in \mathcal{C}_k]}$**.**

- Salida softmax: $z_k(\mathbf{x}, \mathbf{w}) = \frac{\exp(a_k(\mathbf{x},\mathbf{w}))}{\sum_{j=1}^{c} \exp(a_j(\mathbf{x},\mathbf{w}))}$.
- Podemos interpretar $z_{nk} = \Pr(\mathcal{C}_k | \mathbf{x}_n)$.
- Con esto la verosimilitud es $p(\mathbf{t}_n | \mathbf{x}_n, \mathbf{w}) = \prod_{k=1}^{c} z_{nk}^{t_{nk}}$ y la verosimilitud de toda la muestra es $\prod_{n=1}^{N} p(\mathbf{t}_n | \mathbf{x}_n, \mathbf{w})$.
- El estimador ML corresponde a minimizar

$$J(\mathbf{w}) = -\sum_{n=1}^{N} \sum_{k=1}^{c} t_{nk} \ln z_{nk}.$$

Optimización de los parámetros

- Costo fuertemente no lineal en $\mathbf{w}$. Optimización local con varias inicializaciones.
- Se pueden usar métodos de orden 1 (máxima pendiente) o 2 (métodos de tipo Newton, gradiente conjugado, etc).
- Por costo computacional (muchas muestras, muchos parámetros) $\rightarrow$ método de máxima pendiente:

$$\Delta \mathbf{w} = -\eta \frac{\partial J}{\partial \mathbf{w}}, \quad \Delta w_{pq} = -\eta \frac{\partial J}{\partial w_{pq}}$$

$$\mathbf{w}(m+1) = \mathbf{w}(m) + \Delta \mathbf{w}(m)$$

Optimización: mejoras al descenso por gradiente

Elección de η: se puede ajustar en base a una aproximación local cuadrática del costo entorno al mínimo (Hessiana, inversos de valores propios).

- η chico: convergencia lenta; η grande: puede diverger.

Momento o inercia: $\mathbf{w}(m+1) = \mathbf{w}(m) + \Delta\mathbf{w}(m) + \alpha\Delta\mathbf{w}(m-1)$. en general con α cercano a 1.

- Suaviza el descenso, ayuda a no estancarse en mesetas.
- Otros más sofisticados: momento de Nesterov, Adam, etc.

Regularización en redes neuronales

Algunas muy usadas:

Weight Decay: amortiguar el peso de $\mathbf{w}(m)$ antes de actualizar:

$$\mathbf{w}(m+1) = (1-\varepsilon)\mathbf{w}(m) + \Delta\mathbf{w}(m), \quad 0 < \varepsilon < 1.$$

Equivale a regularización L^2: $J_{ef}(\mathbf{w}) = J(\mathbf{w}) + \frac{\varepsilon}{2\eta}\mathbf{w}^T\mathbf{w}$.

Terminación temprana (*early stopping*):

- En training muchos algoritmos de optimización pueden asegurar que el error decrece o se mantiene.
- En validación: a menudo el error decrece al principio, y luego crece a medida que la red empieza a sobre-entrenar.

$\Rightarrow$ Criterio: cortar el training en la iteración que corresponde al menor error de validación.

Dropout

Propagación de errores hacia atrás (*backpropagation*)
Rumelhart et al., 1986

Técnica eficiente para evaluar derivadas de funciones de costo (del orden que sea) en redes feed-forward, en entrenamiento.

La mayor parte de los algoritmos de training tienen dos etapas:

1. Evaluar las derivadas que sean necesarias (gradientes, hessianas, ...).
2. Ajuste de pesos usando las derivadas evaluadas, según algún esquema de optimización (Newton, máxima pendiente, etc.)

Backpropagation:

- Para evaluar derivadas de la función de error (o costo), los errores se propagan por la red hacia atrás.
- No es otra cosa que una implementación de la regla de la cadena para las redes neuronales.

Propagación de errores hacia atrás (*backpropagation*)

Ejemplo con red de tres capas, costo L^2. Vale para cualquier topología feed-forward y para cualquier costo diferenciable.

Previo a la propagación backwards: propagación forward para evaluar las activaciones a_j, a_k.

Pesos de la capa oculta a la capa de salida:

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial a_k}\frac{\partial a_k}{\partial w_{kj}} = -\delta_k \frac{\partial a_k}{\partial w_{kj}}$$

donde $\delta_k = -\frac{\partial J}{\partial a_k}$ es la sensibilidad del nodo k.

$$\frac{\partial a_k}{\partial w_{kj}} = y_j,$$

$$\delta_k = -\frac{\partial J}{\partial a_k} = -\frac{\partial J}{\partial z_k}\frac{\partial z_k}{\partial a_k} = (t_k - z_k) f'(a_k)$$

$$\Rightarrow \boxed{\Delta w_{kj} = \eta \delta_k y_j = \eta (t_k - z_k) f'(a_k) y_j}$$

Propagación de errores hacia atrás (*backpropagation*)

Pesos de la capa de entrada a la capa oculta:

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} = \frac{\partial J}{\partial y_j} f'(a_j) x_i$$

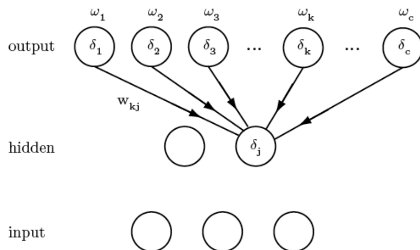
$$\frac{\partial J}{\partial y_j} = \frac{\partial}{\partial y_j}\left(\frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2\right) = -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial y_j} = -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial a_k}\frac{\partial a_k}{\partial y_j}$$

$$= -\sum_{k=1}^{c}(t_k - z_k) f'(a_k) w_{jk} = -\sum_{k=1}^{c} w_{kj}\delta_k$$

Sensibilidad, nodo de capa oculta: $\delta_j = f'(a_j)\sum_{k=1}^{c} w_{kj}\delta_k$.

$$\Rightarrow \boxed{\Delta w_{ji} = \eta x_i \delta_j = \eta x_i f'(a_j)\sum_{k=1}^{c} w_{kj}\delta_k}$$

Backpropagation - Aprendizaje de la capa oculta



$$\delta_k = (t_k - z_k) f'(a_k), \quad \delta_j = f'(a_j) \sum_{k=1}^{c} w_{kj} \delta_k$$

Protocolos de Aprendizaje

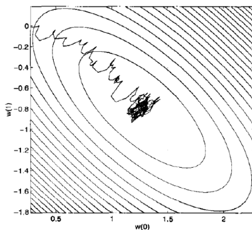
Tres esquemas básicos de aprendizaje:

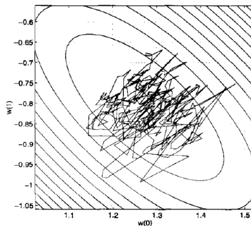
- Estocástico: se elige un subconjunto aleatorio de los patrones
- Batch: se usan todos los patrones
- En línea: uno a uno a medida que aparecen

Esquema Estocástico

Algorithm 1 (Stochastic backpropagation)

1 **begin initialize** network topology (# hidden units), $\mathbf{w}$, criterion $\theta, \eta, m \leftarrow 0$
2   **do** $m \leftarrow m + 1$
3       $\mathbf{x}^m \leftarrow$ randomly chosen pattern
4       $w_{ij} \leftarrow w_{ij} + \eta\delta_j x_i; \quad w_{jk} \leftarrow w_{jk} + \eta\delta_k y_j$
5   **until** $\nabla J(\mathbf{w}) < \theta$
6 **return** $\mathbf{w}$
7 **end**





Esquema Batch Backpropagation

Se considera todos los patrones y se actualiza con una suma del aporte de todos, es decir tomando como error:

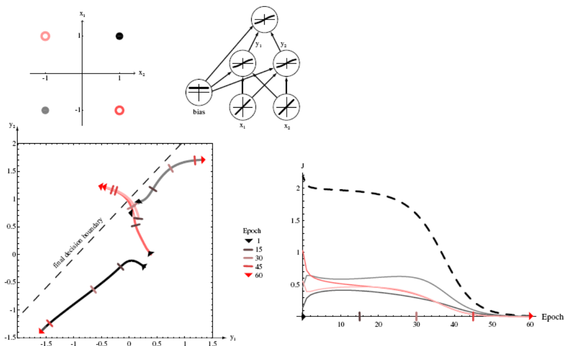
$$\sum_{p=1}^{n} J_p$$

Algorithm 2 (Batch backpropagation)

1 **begin initialize** network topology (# hidden units), $\mathbf{w}$, criterion $\theta, \eta, r \leftarrow 0$
2 **do** $r \leftarrow r + 1$ (increment epoch)
3 $m \leftarrow 0;\ \Delta w_{ij} \leftarrow 0;\ \Delta w_{jk} \leftarrow 0$
4 **do** $m \leftarrow m + 1$
5 $\mathbf{x}^m \leftarrow$ select pattern
6 $\Delta w_{ij} \leftarrow \Delta w_{ij} + \eta \delta_j x_i;\quad \Delta w_{jk} \leftarrow \Delta w_{jk} + \eta \delta_k y_j$
7 **until** $m = n$
8 $w_{ij} \leftarrow w_{ij} + \Delta w_{ij};\quad w_{jk} \leftarrow w_{jk} + \Delta w_{jk}$
9 **until** $\nabla J(\mathbf{w}) < \theta$
10 **return** $\mathbf{w}$
11 **end**

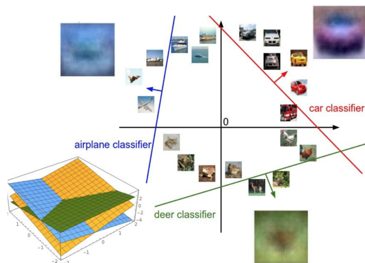
Backpropagation como un mapeo del espacio de caractesrísticas

Como la última etapa es un discriminante lineal, el entrenamiento se puede ver como una deformación del espacio en otro separable. Graficamente para el XOR:



Interpretación de los pesos aprendidos

Los pesos en uno de los nodos de la capa oculta representan los valores que dan máxima respuesta de ese nodo. Se puede explicar la solución como haber encontrado los pesos que corresponden a un filtro matcheado.



Interpretation of linear classifiers as template matching. Another interpretation for the weights W is that each row of W corresponds to a *template* (or sometimes also called a *prototype*) for one of the classes. The score of each class for an image is then obtained by comparing each template with the image using an *inner product* (or *dot product*) one by one to find the one that "fits" best. With this terminology, the linear classifier is doing template matching, where the templates are learned. Another way to think of it is that we are still effectively doing Nearest Neighbor, but instead of having thousands of training images we are only using a single image per class (although we will learn it, and it does not necessarily have to be one of the images in the training set), and we use the (negative) inner product as the distance instead of the L1 or L2 distance.

Skipping ahead a bit: Example learned weights at the end of learning for CIFAR-10. Note that, for example, the ship template contains a lot of blue pixels as expected. This template will therefore give a high score once it is matched against images of ships on the ocean with an inner product.

(Imágenes del curso "Convolutional Neural Networks for Visual Recognition", Stanford)

Reders neuronales y discriminantes de Bayes

Se puede probar (Duda, sec. 6.6.1) que en el límite con infinitos datos, la red entrenada aproxima en mínimos cuadrados las probabilidades a posteriori:

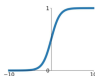
$$z_k(\mathbf{x}, \mathbf{w}) \approx P(w_k|\mathbf{x})$$

(asumiendo que hay suficientes nodos en la capa oculta como para que la red pueda representar a la función $P(w_k|\mathbf{x})$.)

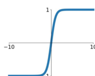
La función de activación: $f(.)$ debe ser:

- No lineal
- Que sature (acotada)
- Continua y derivable (para poder hacer descenso por el gradiente)
- monotonía

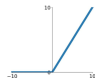
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$



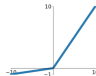
tanh
$\tanh(x)$



ReLU
$\max(0, x)$

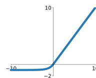


Leaky ReLU
$\max(0.1x, x)$



Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU
$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$



Consideraciones prácticas

- Normalizar la entrada: es conveniente modificar la entrada para que las caracterísiticas tengan igual media y varianza.
- Entrenar con ruido: puede ser útil entrenar agregando ruido de hasta 10% en la entrada para evitar sobre-entrenamiento.
- Transformar los datos de entrada (*data augmentation*:
    - Si se conoce transformaciones que describan la variabilidad de los datos: por ej en OCR: rotación, escala,negrita... entonces se puede hacer crecer el conjunto de entrenamiento aplicando estas transformaciones.
    - Es una forma de hacer cualquier red invariante ante cierto grupo de transformaciones.

Consideraciones - Tamaño de la capa oculta

Relacionada con la complejidad de la superficie de decisión.
En general funciona bien la regla empírica:

> Tantos nodos para que haya $n/10$ parámetros por determinar.

Backpropagation - Inicialización de los pesos

Se quiere aprender las clases a la misma velocidad:
Elección de pesos dist uniforme:

$$-\hat{w} < w < \hat{w}$$

con $\hat{w}$ de forma que la salida de los nodos ocultos esté en un rango no saturado, $-1 < a_i < 1$, si no es difícil que esos nodos se ajusten en el aprendizaje.
Ejemplo, d entradas:

$$\hat{w} = \frac{1}{\sqrt{d}}$$

Lo mismo se aplica a los pesos de la capa final:
inicializar con

$$-1/\sqrt{n_H} < w_{kj} < 1/\sqrt{n_H}$$

Otros tipos de redes

- Redes convolucionales (el jueves)
- Redes recurrentes (texto, procesamiento de lenguaje natural)
- Redes adversarias generativas (GANs) (el jueves)
- Autoencoders (son no supervisadas)
- ...