Deep Learning
Seminario Minería de Datos

Álvaro Riascos
Mónica Ribero

2 de mayo de 2017

Contenido

Introducción

Problema

- Sea $(\vec{x_1}, y_1), (\vec{x_1}, y_2), ..., (\vec{x_1}, y_n)$ una muestra marcada con variables independientes $\vec{x_i} = (x_{i1}, x_{i2}, ..., x_{im})$ y dependiente y_i, $i = 1, ..., n$.
- Queremos predecir el valor de y_i dado x_i.
- Asumimos que existe una función tal que $f(x_i) = y_i$ y queremos encontrar su mejor aproximación

Problema – Regresión

$\vec{x_i} = (\text{salario}, \text{cantidad de personas con las que vive})$

$y_i = \text{precio de su vivienda}$

Problema – Clasificación

$\vec{x_i}$ = información pixeles en imagen de resonancia

y_i = tiene tumor

Problema – Clasificación

$\vec{x_i}$ = (cuentas de palabras en un documento)

y_i = sentimiento (positivo, neutro, negativo)

Problema – Clasificación

$\vec{x_i}$ = (calificaciones de un usuario a varios restaurantes)

y_i = perfil de cliente

Modelos

- Regresión lineal
- Regresión logística
- Árboles de decisión
- Random Forests y Boosting de árboles

Modelo Logit

Regresión Logística – Clasificación

- En vez de representar el valor de y, representar la Probabilidad de que y perteneza a cierta categoría dado x

$$Pr(y = 1|x) = g(\theta^t x)$$
$$= \frac{1}{1 + e^{-\theta^t x}}$$

- $\hat{y}_i = h_\theta(x_i) = \begin{cases} 1 & g(\theta^t x_i) \geq 0{,}5 \\ 0 & g(\theta^t x_i) < 0{,}5 \end{cases}$

Regresión Logística

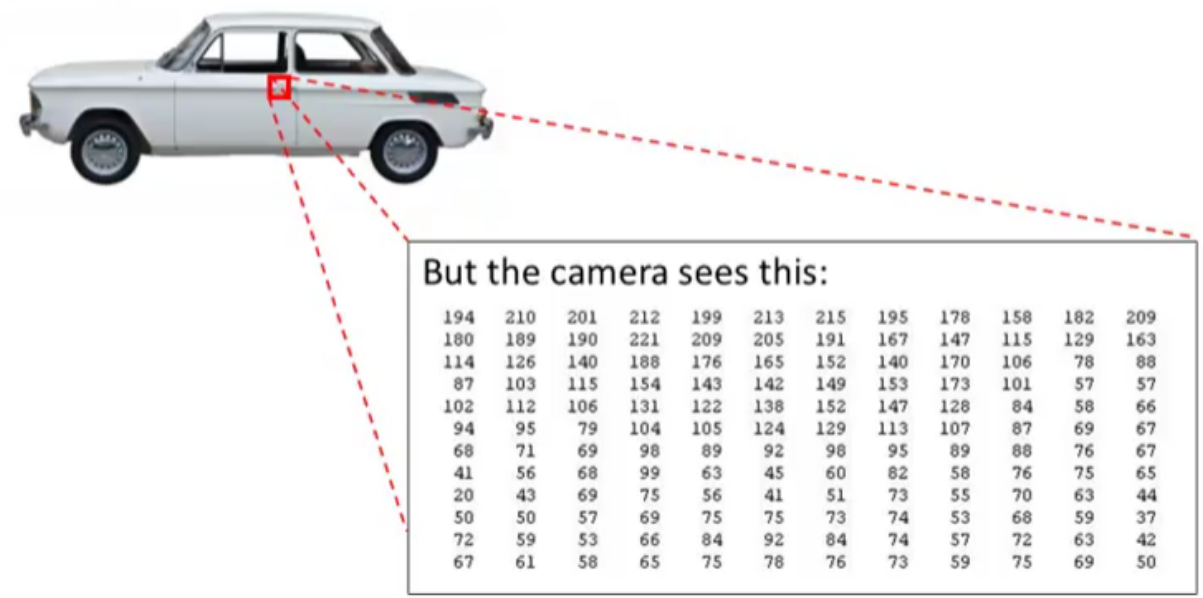
Función de costo

$$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}(y_i \log(h_\theta(x_i))) + (1 - y_i)(1 - \log(h_\theta(x_i)) + \frac{\lambda}{2m}\sum_{i=1}^{m}\theta_i^2$$

$$\hat{\theta} = argmin(J(\theta))$$

Desventajas del modelo Logit

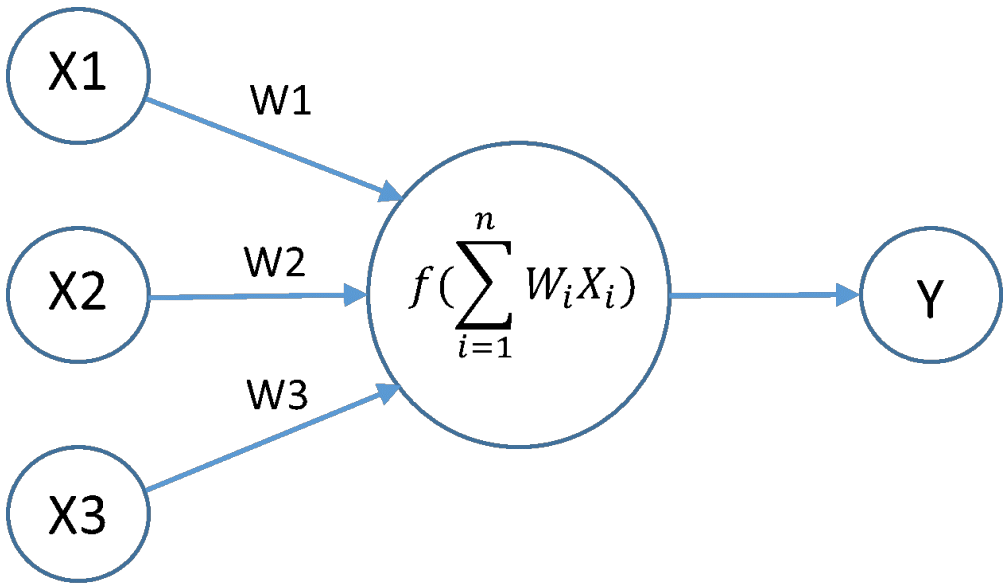
- Problemas de reconocimiento visual pequeños tienen 50×50 pixeles.
- Hipótesis no lineales requieren muchas *features*
- 2500 *features* + términos cuadráticos $(\frac{n^2}{2}) \approx 3{,}1$ millones



Redes Neuronales

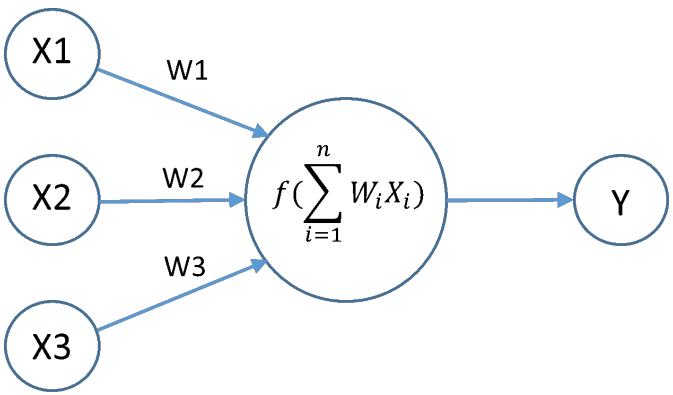
Neurona

- Unidad que recibe inputs y devuelve un output



- Por ejemplo una regresión logística

Feed Forward Networks



Cada neurona

1. Recibe un input x
2. Realiza una transformación lineal $(w_i x + b_i)$
3. Aplica una transformación no lineal g, para obtener $out = g(w_i x + b_i)$

Redes Neuronales

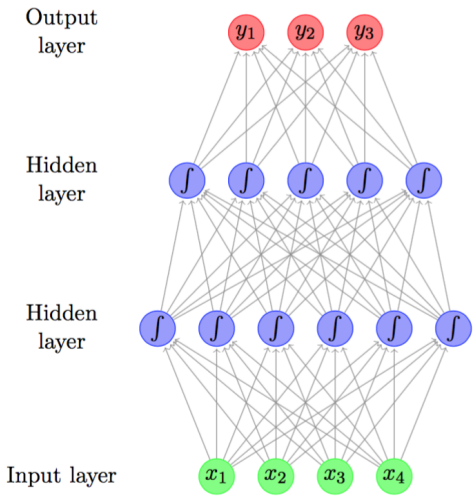


Figure 2: Feed-forward neural network with two hidden layers.

- Modelos no paramétricos de Machine Learning compuestos de unidades computacionales (**neuronas**)
- Aproximar la función $f : \mathcal{F} \rightarrow \mathcal{O}$ que relaciona un conjunto de features $\mathcal{F} \subset \mathbb{R}^n$ con un conjunto de outputs $O \subset \mathbb{R}$

Redes Neuronales

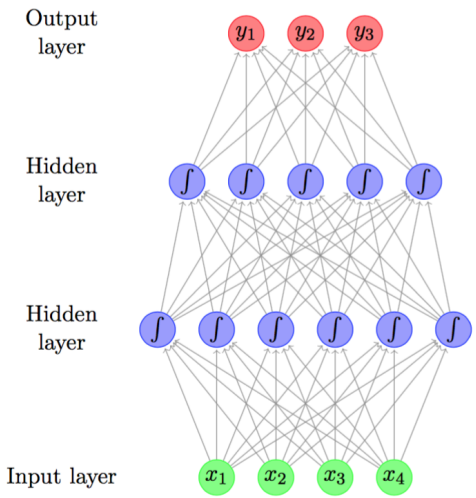


Figure 2: Feed-forward neural network with two hidden layers.

- Concatenación de capas de neuronas
  - Capa *input*
  - Capas ocultas
  - Capa *output*

Capa externa

- Regresión: Una neurona
- Clasificación Binaria: Probabilidad de éxito
- Clasificación $k-$categorías: Probabilidad de pertenecer a cada categoría. Para el entrenamiento, las etiquetas se convierten en vectores:

$$y_i = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

Feed Forward Networks

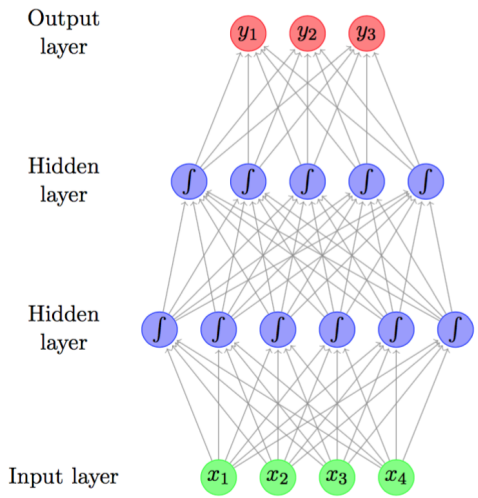
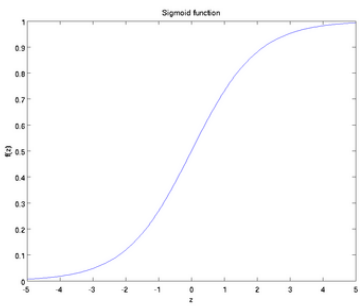


Figure 2: Feed-forward neural network with two hidden layers.

$$NN_{MLP2}(x) = y$$
$$\mathbf{h^1} = g^1(\mathbf{W^1 x} + \mathbf{b^1})$$
$$\mathbf{h^2} = g^2(\mathbf{W^2 h^1} + \mathbf{b^2})$$
$$\mathbf{y} = h^2 W^3$$

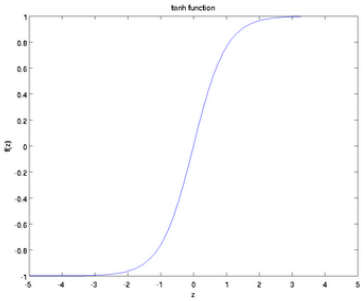
- Si las capa j y $j+1$ tienen s_j y s_{j+1} neuronas respectivamente, $W_{j+1} \in \mathbb{R}^{s_{j+1} \times s_j}$
- Si usa un término de sesgo, $W_{j+1} \in \mathbb{R}^{s_{j+1} \times s_j + 1}$

Funciones de Activación



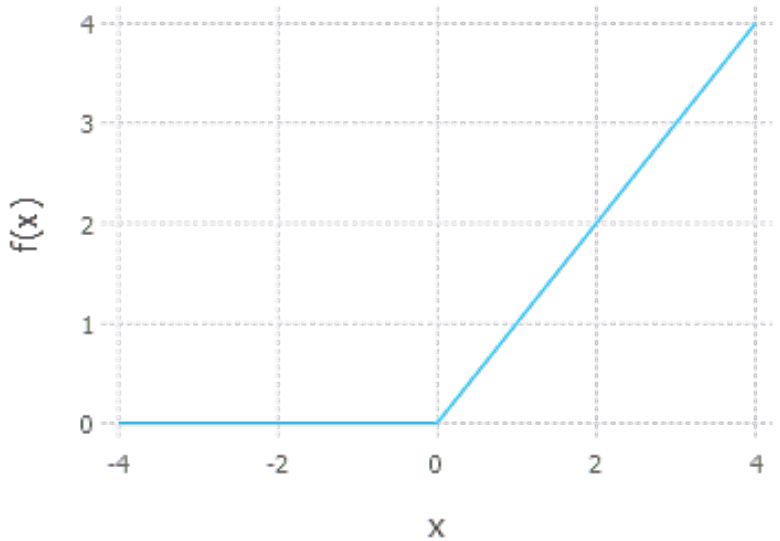
$$sigmoid(x) = \frac{1}{1 + e^{-x}}$$

Funciones de Activación



$$tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

Funciones de Activación



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

Función de Costo

$$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K}(y_i^k \log(h_\theta(x_i)))^k + (1 - y_i^k)(1 - \log(h_\theta(x_i))_k$$
$$+ \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_j}\sum_{j=1}^{s_{j+1}}\Theta^{(l)}_{i,j}$$

$$\hat{\theta} = argmin(J(\theta))$$

Entrenamiento

- Mejor más neuronas que menos, capturar más *no linearidades*
- Evitar overfitting con regularización
- Escalar las variables
- Mínimos locales: Hacer varias inicializaciones.

Minimización de la función de costo

Algoritmo *Backpropagation*

Aplicaciones

Carros autónomos

https://www.coursera.org/learn/machine-learning/lecture/zYS8T/autonomous-driving

Medicina

- Diagnóstico a partir de imágenes
- Diagnóstico a partir de lenguaje
- Genómica

Redes en el contexto de NLP

Input x

- En las aplicaciones de NLP, el input **x** codifica palabras, POS, información lingüistica.
- Para cada documento se cuenta, en general, con un conjunto de vectores.
- Se utiliza una función $c : \mathcal{F} \rightarrow \mathbb{R}^d$
- c's comunes incluyen concatenación y suma.

Embeddings

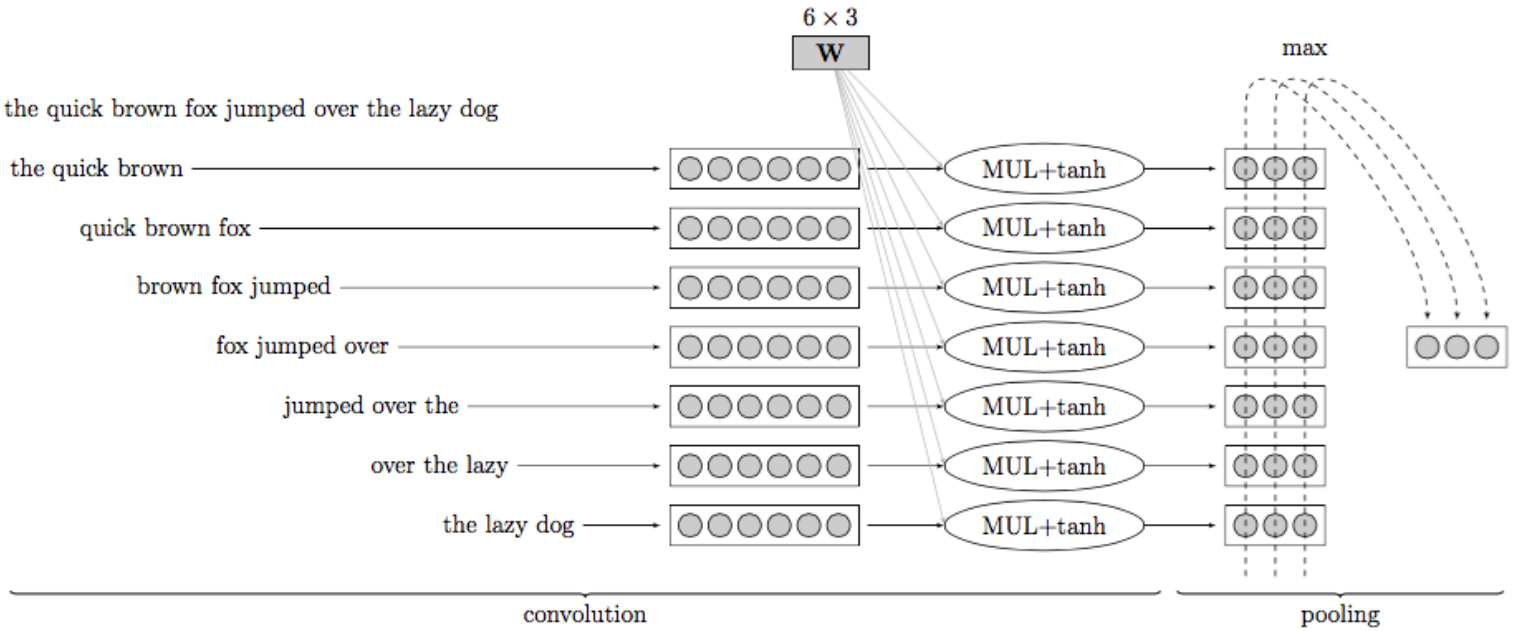
- Supervisados: Entrenar redes para contexto particular ('one hot representations")
- No supervisados: GloVe, LDA y modelos de tópicos.

Otras arquitecturas

Convolutional Networks

- Arquitectura que permite detectar features importantes sin importar la ubicación
- Ha permitido avances importantes en NLP y procesamiento de imágenes
- Aplica un *filtro* (función no lineal aprendida) a cada $k-$ventana de palabras para crear un vector
- Se hace *pooling* para combinar estos vectores en uno solo que concentre las características más importantes sin importar la ubicación

Convolutional Networks



Convolutional Networks

- Sea $x_1, x_2, ..., x_n$ una secuencia de palabras con vectores $v(x_1), ..., v(x_n)$
- Para cada ventana i sea $w_i = [v_{i+1}, ..., v_{i+k}]$
- Convolución: Calcule $p_i = g(w_i W + b)$
- Pooling: Calcule

$$\mathbf{c} = [c_j]$$

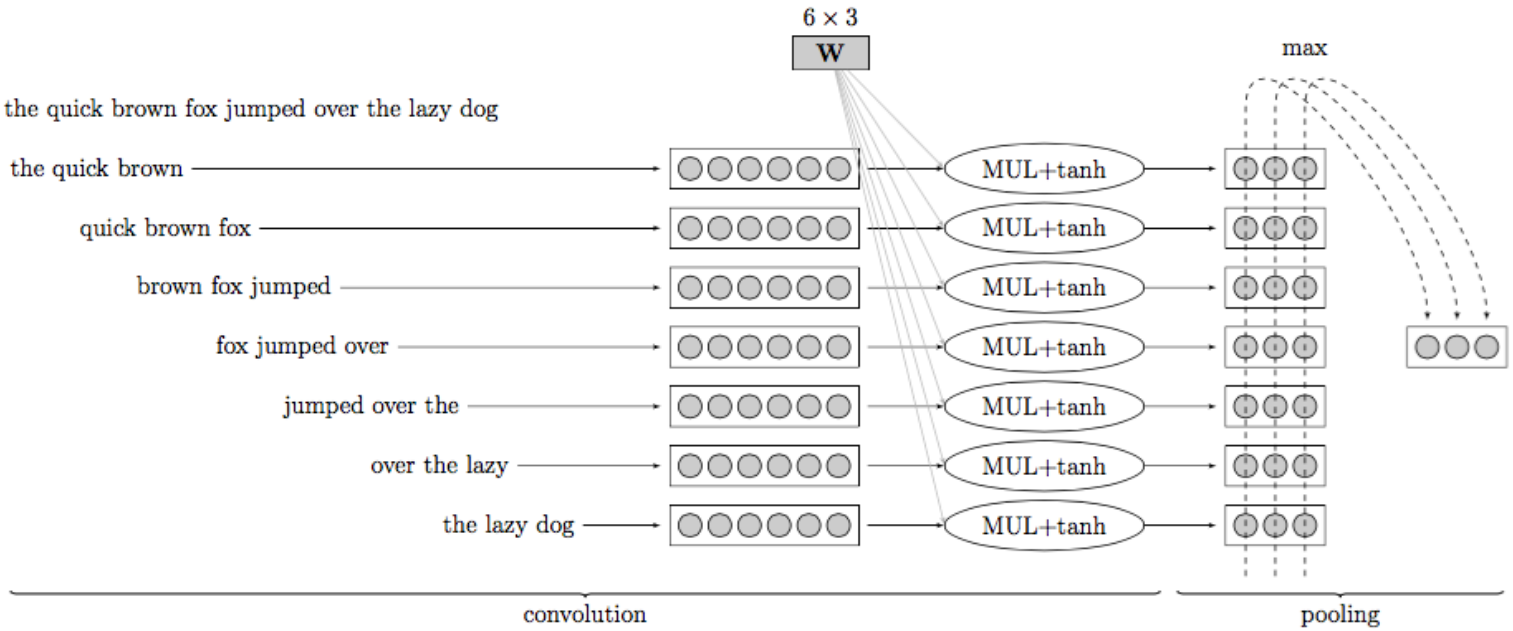
donde

$$c_j = max_{1<i\leq m} p_{ij}$$

Convolutional Networks

- Cada dimensión se especializará en una clase de predictores y la operación *max* escogerá el más importante de cada tipo

Convolutional Networks



Recurrent Networks

- Si bien las redes convolucionales tienen en cierta forma el orden, se restringen a patrones locales.
- Las Redes Neuronales Recurrentes permiten representar inputs de tamaños arbitrarios en un vector de tamaño fijo teniendo en cuenta propiedades estructurales.

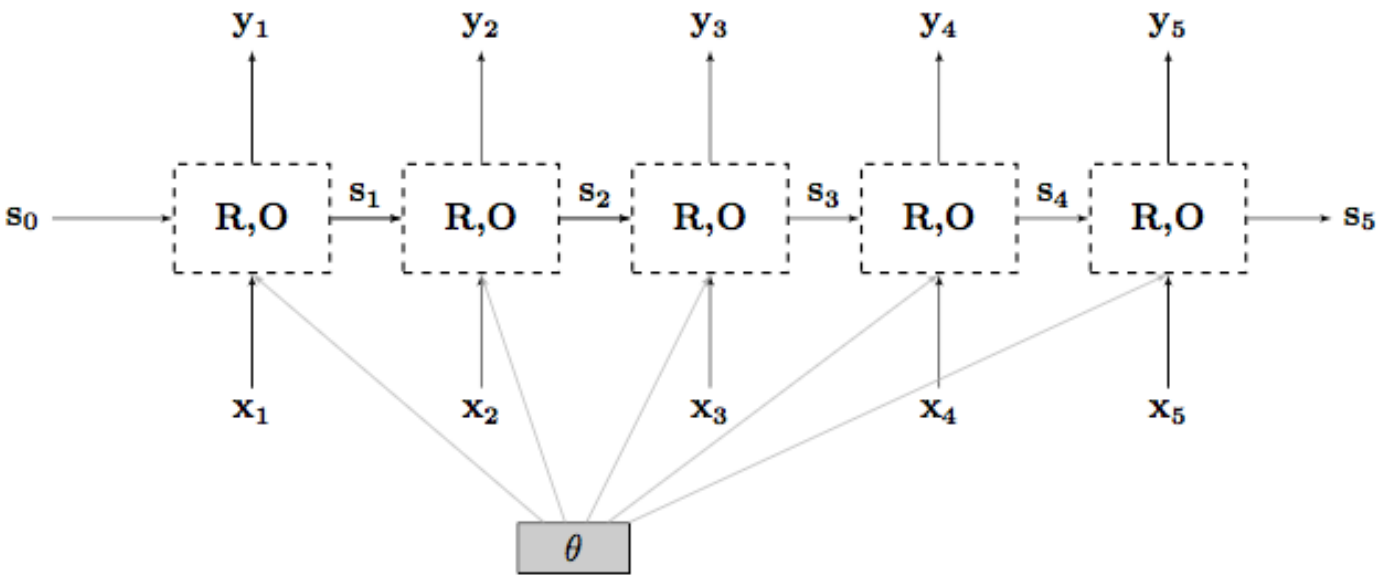
Recurrent Networks

- Dados vectores $x_1, ..., x_n$ y un vector s_0 que será el estado inicial

$$
\begin{aligned}
RNN(s_0, x_{1:n}) &= s_{1:n}, y_{1_n} \\
s_i &= R(s_{i-1}, x_{1:i}) \\
y_i &= O(s_i)
\end{aligned}
$$

Recurrent Networks

Figura: Red Neuronal recurrente



Recurrent Networks

El output puede ser únicamente el último estado o utilizar también los intermedios

- **Acceptor:** Solo se utiliza el último estado para determinar y calcular la función de pérdida a partir de y_n(Análisis de sentimiento)
- **Encoder:** Creación de features para otra tarea (por ejemplo resumir un documento). Solo se utiliza y_n
- **Transducer:** Se utilizan todos los estados y outputs. Por ejemplo, encontrar la distribución de la palabra ii a partir de las palabras $1 : (i - 1)$
- **Encoder - Decoder:** Se utiliza un encoder para producir y_n que se entrega como input a otra RNN que utiliza las palabras ya traducidas y el útlimo output para decodificar. Muy buenos resultados para traducir (o poner tags a las palabras)

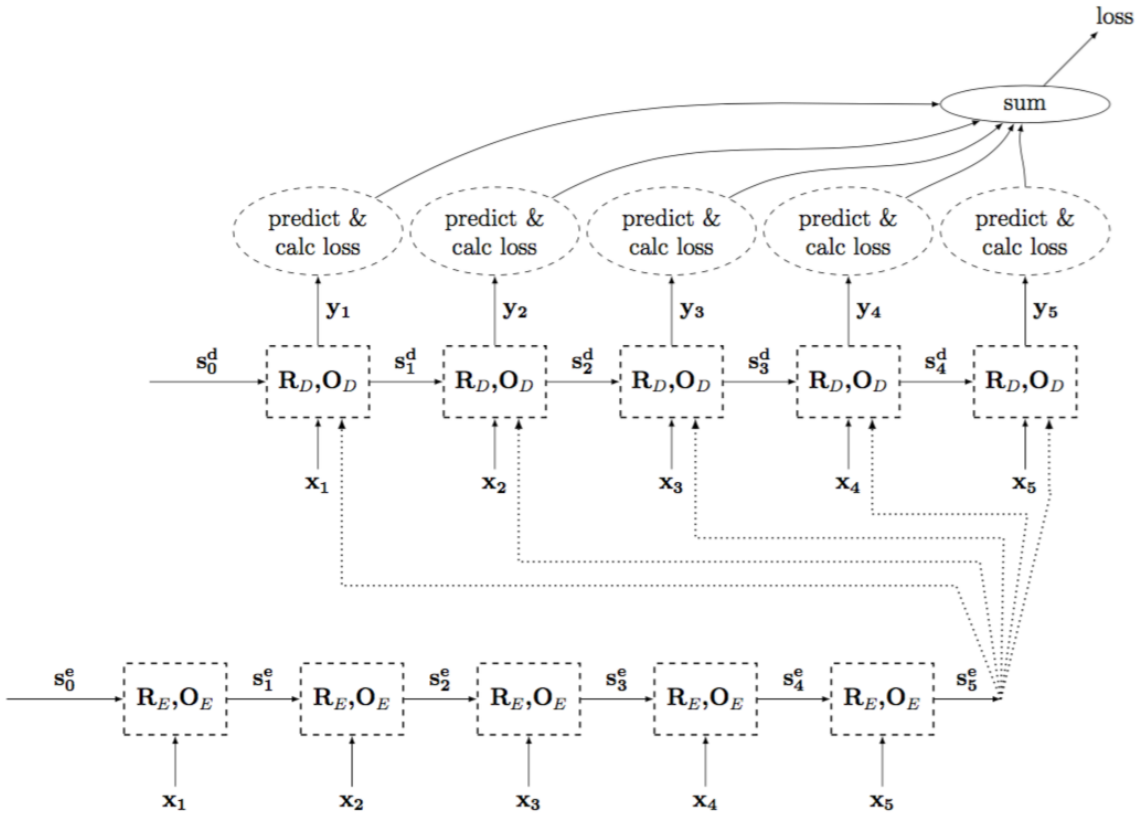


Figure 9: Encoder-Decoder RNN Training Graph.

Recurrent Networks – S-RNN

- **Red Recurrente Simple:** Muy buenos resultados en *sequence tagging* y modelaje de lenguaje.

$$s_i = g(x_i, W^x + s_{i-1} W^s + b)$$
$$y_i = O(s_i) = s_i$$

Recurrent Networks –S-RNN

- Simple RNN es difícil de entrenar por problema de *vanishing gradients*
- Difícil capturar dependencias muy lejanas.

Recurrent Networks – LSTM

- Introducir en el estado s celdas de memoria que preserven los gradientes.
- El acceso a la memoria está controlado por *puertas*. Estas son funciones con rango en $[0, 1]^n$

Recurrent Networks – LSTM

- **Long Short Term Memory:**

$$
\begin{aligned}
\mathbf{s_j} &= R_{LSTM}(\mathbf{s_{j-1}}, \mathbf{x_j}) && = [\mathbf{c_j}; \mathbf{h_j}] \\
\mathbf{c_j} &= \mathbf{c_{j-1}} \odot \mathbf{f} + \mathbf{g} \odot \mathbf{i} \\
\mathbf{h_j} &= \tanh(\mathbf{c_j}) \odot \mathbf{o} \\
\mathbf{i} &= \sigma(\mathbf{x_j}\mathbf{W^{xi}} + \mathbf{h_{j-1}}\mathbf{W^{hf}}) \\
\mathbf{f} &= \sigma(\mathbf{x_j}\mathbf{W^{xf}} + \mathbf{h_{j-1}}\mathbf{W^{hf}}) \\
\mathbf{o} &= \sigma(\mathbf{x_j}\mathbf{W^{xo}} + \mathbf{h_{j-1}}\mathbf{W^{ho}}) \\
\mathbf{g} &= \tanh(\mathbf{x_j}\mathbf{W^{xg}} + \mathbf{h_{j-1}}\mathbf{W^{hg}}) \\
y_j &= O(s_j) = h_j
\end{aligned}
$$

Recurrent Networks – GRU

- LSTM es efectiva pero complicada y computacionalmente costosa.
- **GRU** (Gated Recurrent Unit)

Recurrent Networks – GRU

$$\mathbf{s_j} = R_{GRU}(\mathbf{s_{j-1}}, \mathbf{x_j}) = (\mathbf{1} - \mathbf{z}) \odot \mathbf{s_{j-1}} + \mathbf{z} \odot \mathbf{h}$$

$$\mathbf{z} = \sigma(\mathbf{x_j W^{xz}} + \mathbf{h_{j-1} W^{hz}})$$

$$\mathbf{r} = \sigma(\mathbf{x_j W^{xr}} + \mathbf{h_{j-1} W^{hr}})$$

$$\mathbf{h} = \tanh(\mathbf{x_j W^{xh}} + (\mathbf{h_{j-1}} \odot \mathbf{r})\mathbf{W^{hg}})$$

$$y_j = O(s_j) = h_j$$

GRACIAS