1. Introducción

Que es “representation learing” y que es la relación con DL?

Un elemento clave de DL es la **extracción de patrones** a partir de los datos. El “representation learning” es el proceso de no solo encontrar estos patrones, pero además encontrar que representación (features) hay en **los datos bruto**.

Así un sistema DL en el mejor caso, basado en solo los datos, puede encontrar las features que mejor describen estos datos, y aprende como combinar estos features para clasificar o analizar los datos.

Existe la teoría, que una red con múltiples capas aprenda en cada capa, características distintas que dependen de caparas anteriores; generando así una pirámide de características simples a más y más complejas características.

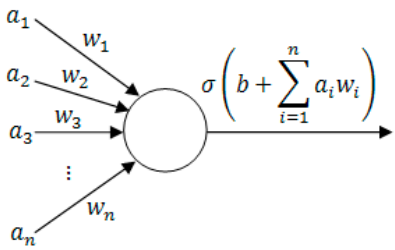
La historia de DL

1950s-60s: perceptrón, Adaptive Linear Element
80s: connexionismo (combinar muchos elementos tontos), back-propagation
90s: LSTM
2006>: Deep Learning, Geoffrey Hinton, Yoshua Bengio, Yann LeCun

Factores más importantes ahora: Muchos datos, mucha potencia computacional

Arquitectura de una Neurona

a … input
w … input weights
b … bias (permite modelar un constante, independiente de las entradas)
σ … función de activación



Para que sirven múltiples capas?

Se puede modelizar la interacción de distintas entradas!

2. Entrenamiento

Que es la función de coste en una RNN?

La “cost function” se utiliza para medir la diferencia entre la predicción y el valor varadero.

$$C(w,b) \equiv \frac{1}{2n}\sum_x \left\| y(x) - a \right\|^2$$

Que es el algoritmo “Gradient Decent” y para que sirve?

La red neuronal tiene dos parámetros que esta aprendiendo los “**weights**” (w_i) y los **biase** (b_i). Estos se aprenden de los ejemplos, ajustan dándolos para que redusquan el valor de la función de coste. El algoritmo “Gradient Decent” define la dirección de este adjuste en función del gradiente de la función de coste al respecto de los pesos y del bias.

$$w'_k = w_k - \eta \frac{\partial C}{\partial w_k}$$
$$b'_l = b_l - \eta \frac{\partial C}{\partial b_l}$$

Donde η es el ‘learning rate’ que esta escalando la fuerza del efecto de una iteración del algorimo.

Stochastic Gradient Decent

Es una aproximación de gradient decent donde en vez de calcular el gradiente en función de todos los datos de entrenamiento (que sea una iteración del gradient decent normal) se utilizan solo m ejemplos de los todos que hay.

$$w'_k = w_k - \frac{\eta}{m}\sum_j \frac{\partial C_{X_j}}{\partial w_k}$$

$$b'_l = b_l - \frac{\eta}{m}\sum_j \frac{\partial C_{X_j}}{\partial b_l}$$

Que es el algoritmo “Backpropagation” y para que sirve?

Backpropagation se utiliza para calcular los gradientes de cada nodo de la red, que se necesitan para aplicar el algoritmo de gradient decent.

Se ejecuta den dos fases:
Forward pass: donde se calculan las funciones de actividad de cada neurona y se pasan al siguiente nivel (hacia a tras)
Backward pass: ahora se calculan las gradientes en función del valor de coste en la última capa. Capa por capa se calcula el gradiente y se pasa a la capa anterior (hacia adelante)

El gradiente de la función de actividad se puede calcular analíticamente (si se conoce) o utilizando un grafo de computación (secuencia de operaciones simples).

3. Frameworks

Que es un “framework” y para que sirve?

Frameworks son libarías de software que ayudan en desarollar RNN, que automáticamente generan **un grafo de computación** y ejecutan **backpropagation**.

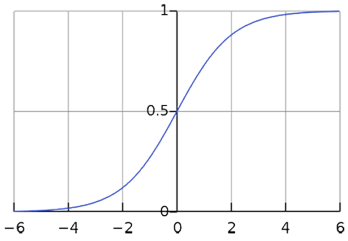
El grafo de computación dispone:

- **Optimización de los cálculos a realizar**
- **La ejecución se divide en trozos, lo que facilita la autodiferenciación**
- **Facilita la ejecución distribuida y en GPU**

Ejemplos: Tensorflow, Theano, pytorch, caffe

4. Aspectos Prácticos del entrenamiento

Que son funciones de activación típicas?



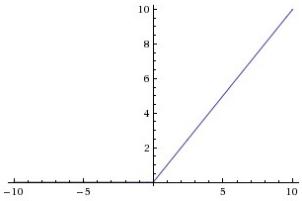
Sigmoid:

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

- **Matan los gradientes** (para valores altos de entrada)
- **No esta centrada a 0.0** (el resultado es siempre positivo)

ReLu (rectified linear unit)

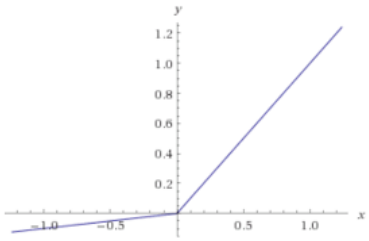
f(x)=max(0,x)



+ No satura con valores altos de entrada
+ Computacionalmente fácil
-- Puede morir (cuando los pesos de entrada no dejan la entrada subir, así se queda la salida en cero)

Leaky ReLu

$$f(x) = \begin{cases} x \\ 0.01x \end{cases} if\ x > 0\ otherwise$$



+ Evita que muere

Que es la saturación de la función de la activación?

Si la función de activación independiente de la entrada produce su valor máximo. (Por ejemplo, de sigmoid)

Como se inicialen los pesos?

La inicialización de los pesos es muy importante, en vez de poner cero (fatal) o hacerlo aleatorio, se utiliza: **Xavier Initialization**

```
w = np.random.randn(n) * sqrt(2/(n_in + n_out))
w = np.random.randn(n) * sqrt(2/n_in) [para ReLu]
```

Es importante que la distribución de los pesos sea escalada de manera que los pesos quedan pequeños, con una media cerca cero y una varianza pequeña.

Que es Batch normalization?

BN es un método que adapta la entrada de cada capa para que la distribución de la entrada para un mini-batch sigue una distribución normal y no cambia mucho entre los mini-batches. (reduce el 'Covariance shift')

Eso facilita la capa aprender. Además, se añaden dos parámetros (gamma, beta) para cada capa que se pueden entrenan también y que controlan la fuerza de BN para cada capa.

normalization: $\widehat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{(k)}]}}$

scale and shift: $y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}$

SGD con momentum

Momentum añada un componente de 'memoria' de gradientes anteriores al gradiente actual. Así esta acelerando el aprendizaje en función de valores antiguas del gradiente.

SGD normal: $x_{t+1}=x_t-\alpha\nabla f(x_t)$

SGD + momento

$x_{t+1} = x_t - \alpha(pv_t + \nabla f(x_t))$

$= x_t - \alpha\nabla f(x_t) + \alpha pv_t$

$= x_t - \alpha\nabla f(x_t) + \alpha p(pv_{t-1} \nabla f(x_{t-1}))$

El Nesterov Momentum hace lo mismo, pero solo con la diferencia del gradiente.

AdaGrad & RMSProp

Esta adaptando/escalando la learning rate actual al respecto de LR anteriores
AdaGrad tiene el problema que la learning rate baja con cada cyclo y puede llegar a un valor muy bajo.

RMSProp evita eso

AdaGrad

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

RMSProp

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

Regularización L1 & L2 & Dropout & Early Stopping

Regularización es un mecanismo que se utiliza para evitar overfitting, evitando que los pesos de un circulo de entrenamiento se cambian mucho en función de los datos de entrada. Eso se puede hacer penalizando valores altos de los pesos en la función de coste.

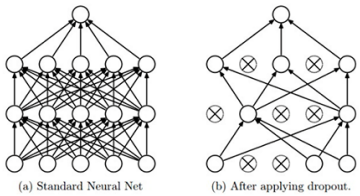
L2

$$C = C_0 + \frac{\lambda}{2n}\sum_w w^2$$

L1

$$C = C_0 + \frac{\lambda}{n}\sum_w |w|$$

Dropout es otro método de regularización que evita que una neurona depende mucho de sus entradas. En cada ciclo de entrenamiento, se desactiven neuronas con cierta probabilidad (25%-50%).



(a) Standard Neural Net (b) After applying dropout.

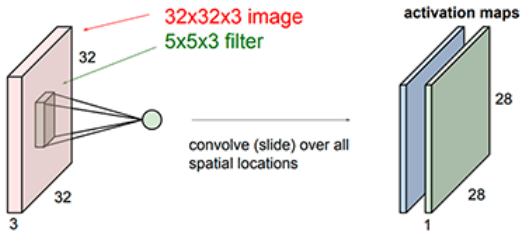
Early stopping
Simplemente para de entrenar una vez que el modelo no mejora en los datos de validación, solo en los de test.

5. Convolutional Neural Networks

Que son CNN?

Convolutional Neural Networks son una arquitectura de FFN que se ha hecho popular desde 2010/2012 en el ámbito de análisis de imágenes.

En vez de capas FC, se aplican Filtros pequeños a la entrada, generando como salida una 'activation map'. Durante el entrenamiento se aprenden los valores de estos filtros.



Historicamente redes importantes son:

- 1998: LeNet-5
- 2012: AlexNet

Se aplican para

- Clasificación
- Búsqueda (similares)
- Detección de objetos
- Segmentación

Que es 'Stride' y zero-padding?

Stride es la distancia entre los filtros aplicados.

Cuantas veces se aplica un filtro (así el tamaño del mapa de activación) se calcula con

$$\text{Output\_size: } ((N - F + 2P) / S) + 1$$

Donde P … es el número de pixeles zero (padding) que se añaden.
N …Input NxN
F … Tamaño del Filtro

Que es 'data augmentation'?

Es un método de ampliar los datos de entrenamiento, en cual se modifican las imágenes de entrada, multiplicando el número de imágenes de entrada.
Métodos (rotación, ruido, translación, …)

Que es Transfer Learning?

Un método para utilizar recursos invierto previos en entrenar una red con un dataset muy grande y diverso, y utilizarlo para acelerar el entrenamiento a un dataset nuevo.

Se coge el parte delantera de una red CNN (sin los últimos capas FC), se añadan capas FC nuevos y se entrenan en función de los datos disponible.

6. Word Vectors

Representación de una palabra

RLN empieza con una representación de sentido de una palabra. Representations de Tipo Wordnet (Taurus) o bag of words (one-hot encoding) son costosos y no permiten comparar la **similitud** de dos palabras.

Una representación de tipo vector (i.e. **representación distribuida**); generado por algoritmos como Word2Vec es preferible.

Word2Vec

Genera dos vectores para cada palabra (V_w … palabra central, U_w … palabra central)

Se busquen vectores/embedding que maximiza la Probabilidad (palabra context | palabra centro)

$$P(o|c) = \frac{exp(u_o^T v_c)}{\sum_{w \in V} exp(u_w^T v_c)}$$

El embedding de una palabra es **la media** de ambos vectores.

Así un buen embedding sirva para predecir que palabras sean parte del contexto, dado un vector de una palabra central.

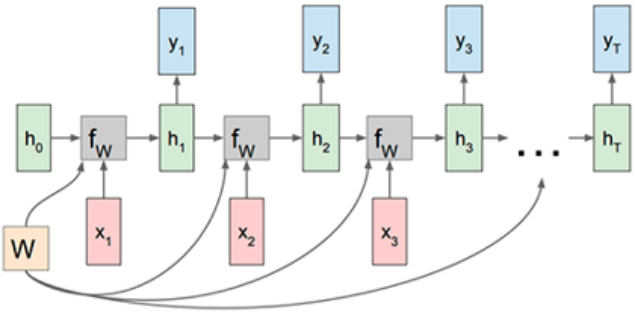
La representación vectorial permite calcular la similitud de dos palabras utilizando **el coseno** del Angulo entre los vectores.

$$similarity = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \sum_{i=1}^{n} A_i B_i \Big/ \sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}$$

Además, permite cálculos aritméticos con el significado de palabras.

El Word2Vec se ejecuta de tipo ANN con SGD. Se aplica '**Negative Sampling**' que sirve como aproximación de todas las palabras que no sean contexto de una palabra central. En vez de calcular la distancia a todos las palabras del diccionario para definir el contexto de una palabra, se cogen los palabras que si son parte del contexto (palabras ceras) y se hace un sampling del diccionario para aproximar todos las palabras que no son.

7. Redes Recurrentes



Recurrent Neural Networks (RNN) son uno de tres tipos de redes que hemos visto, especializado en modelar relaciones temporales entre elementos de una secuencia.

En la versión vanilla, el resultado depende del input y un estado interno (h)

$$y_t = W_{hy}h_t$$
$$h_t = \tanh\left(W_{hh}h_{t-1} + W_{xh}x_t\right)$$

El estado interno h_t esta cambiando en función de la secuencia del input.

Para entrenar RNN se utiliza una variación de Backpropagation que es
Backpropagation throug time (BPTT), o una versión aproximada **Truncated BPPT (TBPTT)**

La versión básica de RNN sufre dos problemas con el BPTT

- **Exploding gradient**
- **Vanishing gradient**

Eso pasa en función de aplicar los mismos pesos (W) múltiples veces a los gradientes.

Como solución se han desarrollado (LSTM y GRU)

Long short term memory (LSTM)

Una LSTM es un nodo que sirve como una celda de memoria programada.
En vez de tener solo un valor interno, tiene dos (nuevo **cell state**), y cuatro/tres 'gates' para manipular la célula.

- h … **internal state**
- c … **cell state**

- f … forget gate (cuanto se olvida del valor c_{t-1} anterior)
- i … input gate (cuanto del input x efecto c)
- g … similar a i, regla cuanto de tanh(x) llega a c
- o … output gate (cuanto del estado c se aplica al estado h)

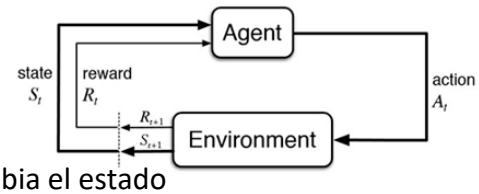
Gated Recurrent Unit (GRU)

Parecido a LSTM, pero en vez de dos estados internos hay solo uno.

Modelos de Lenguajes

Un modelo de un lenguaje se utiliza para calcular la probabilidad de que una secuencia de letras o palabras es valida.
Se puede aplicar para elegir entre predicciones.
Sistemas tradicionales generan el modelo basado en la distribución de palabras, pero son limitados, por el número de pasos a tras.

8. Deep Reinforcement Learning



Los componentes claves son

- Un **agente** que puede actual
- Un **Ambiente** que evalúa las acciones de Agente y cambia el estado
- Una **recompensa** en función de las acciones del agente

Markov Decision Processes (MDP)

Se modela como que define que conocer el estado actual es suficiente para decidir la acción optima en cada estado.

El sistema esta definido por (S, A, Psa, γ, R)

- S … Estados
- A … Actiones
- Psa … La probabilidad de llegar al estado s con la acción a
- γ … es el ***discount factor*** (cuanto nos vale una recompensa futura)
- R … la función de recompensa

Función del valor

Value function: Calcular el valor de un estado

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t\geq 0}\gamma^t r_t | s_0 = s, \pi\right]$$

Q-value function: Estado + Acción

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{t\geq 0}\gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

Q*(s, a) nos dice que acción hay que elegir en el estado s, para maximizar la recompensa total.

Ecuación de Bellman

La ecuación de Bellman es un algoritmo iterativo (dinámico) que permite calcular Q*.

Aplicando Bellmann se hace **"Value iteration"**, que es el proceso de calcular Q*, calculando todos los posibles Q(s, a).

Para cada problema real, el número de combinaciones (s, a) es computacionalmente intratable.

Deep Q-Learning

En vez de calcular Q(s, a) se utiliza una red neuronal para aproximar la función. Esta aproximación se utiliza en la ecuación Bellman.

Ecuacuión de Bellman $$Q^*(s,a) = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s',a') | s, a\right]$$

9. DL en el ámbito de Big Data

Las redes modernas cada vez son más profundas y necesitan más datos de entrenamiento. Los avances de DL siguen por hacer cálculos con **GPU's** y **Entrenamiento distribuido**.

GPU

Los GPU's como hardware especializado tiene los siguiente aventajes

- **Muchos núcleos**: Permite algoritmos paralelos de gran escala
- **Mucha memoria en el circuito**: Una ancha de banda alta para cargar los datos de entrenamiento

CUDA: una plataforma de desarrollo para computación paralela con GPU
cuDNN: una liberia CUDA que implementa primitivas altamente optimizadas para el entrenamiento de redes neuronales profunda

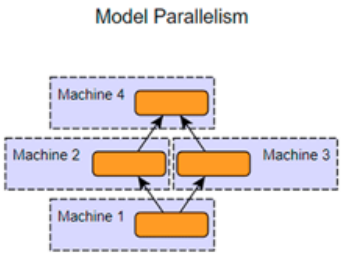
Entrenamiento distribuido

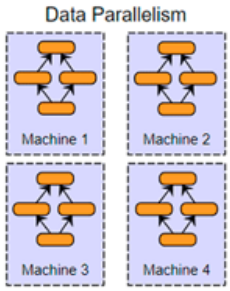
Se necesita un entrenamiento distribuido en el caso que los datos de entrenamiento no caben en una única máquina o que hay múltiples máquinas disponibles.

Model paralelismo

En el caso que no cabe en un única maquina.

- Introduce dependencias
- Necesita comunicación entre máquinas

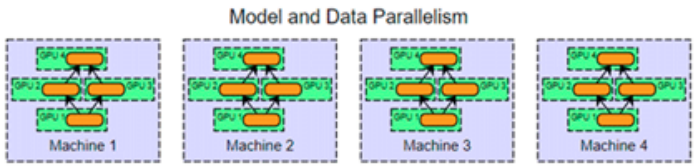




Data Paralelismo

Como combinar los modelos entrenados?

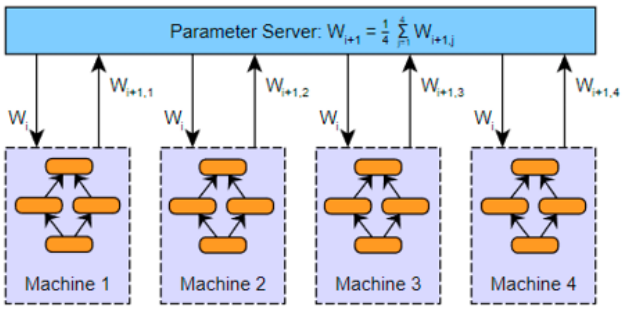
Combinaciones



Parametere Averaging

Hay que juntar los conocimientos de las redes distintas, una manera hacerlo es calcular la media de los distintos pesos.

- Desventaja: El sistema depende de la máquina más lenta.

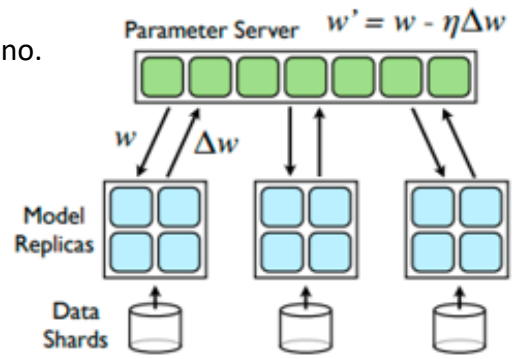


Asynchronous SGD y Downpour SGD

En vez de enviar los pesos, se envía los gradientes a un Servidos central que calcula el SGD. Eso puede pasar asíncrono. Los updates que reciben los workers son distintos.

Asynchronous SGD es el sistema genérico de calcular SGD en cual los gradientes o pesos vienen de origines distintos.

Downpour SGD es una implementación concreta desarrollado por Google.



10. Ecosistemas en la nube

Servidores de modelos

Son aplicación especializadas en hacer un modelo accesible (normalmente por web/api). (ejemplos: Tensorflow Serve, Clipper)

- **Concurrencia:** hacer múltiples inferencias al mismo tiempo
- **Batching:** juntas inferencias para ejecutarlas más eficiente
- **Versionar los modelos:** ser capaz de seguir cambios y restablecer modelos

Parte de un sistema ML

- **Extracción de Datos** -> **Extracción de Features**
- **Código ML**
- **Validación**: evaluar y validar el funcionamiento del modelo y de datos
- **Servidores** (serving)
- **Monitorización**

Continous training pipelines

Son frameworks que automaticen el proceso de creación de un modelo desde la extracción de datos hasta serving.

Problemas comunes

Training-serving skew: los resultados de inferencia son peores que durante el entrenamiento

- Verificar la extracción de datos y features
- Distribución de datos es distinta

Comportamiento del modelo en subpoblaciones: los resultados son malos o falsos para una subpoblación de los casos

- Monitorización y validación
- Tener todos grupos en los datos de entrenamiento

Ecosistemas en la Nube

- NLP
- Visión
- Speech2Text, Text2Speech
- Tradución Automática