TEC | Tecnológico de Costa Rica

Maestría en Ingeniería Electrónica

Reconocimiento de patrones

Apuntes de clase 3

Estudiante: Esteban Martínez Valverde

Profesor: Felipe Meza Obando

II Cuatrimestre

Agosto, 2018

Redes Neuronales Articiales (ANN)

Redes Neuronales Artificiales (ANN)

1943 McCulloch/Pitts proponen un modelo neuronal MP.

1958 Rosenblatt introduce el concepto de "perceptrón".

1969 Minsky/Papert lanzan un libro en el que se demuestran las limitaciones del "perceptrón".

1986 Con el concepto de Back-Propagation (Hinton) y MLP, se vuelve a popularizar el tema de ANN's.

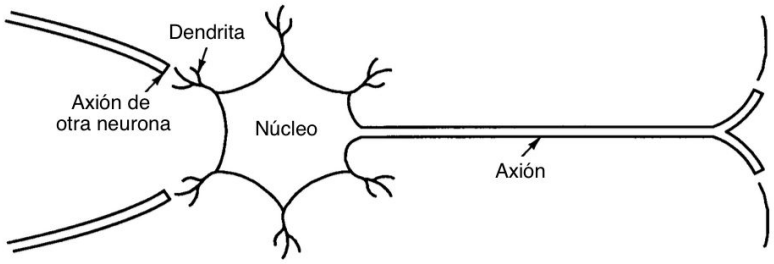
1998 Yann LeCun, CNN's.

2006 Deep Learning (Hinton), como concepto.

2014 Ian Goodfellow, GAN's.

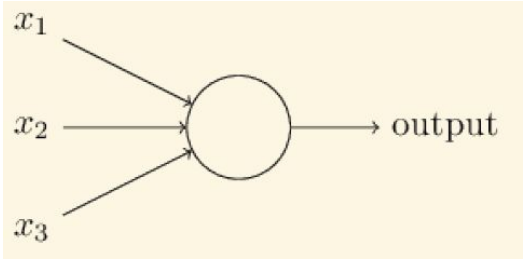
Modelo neuronal biológico

- Codifica las salidas en una serie de impulsos electroquimicos
- **Dendritas** son los receptores que se conectan con otras neuronas
- **Axones** Son las líneas de transmisión hacia otras neuronas
- **Núcleo** es donde se procesa las señales de entrada y las convierte en señales de salida
- El aprendizaje ocurre por activaciones (señales) repetidas de ciertas conexiones a otras

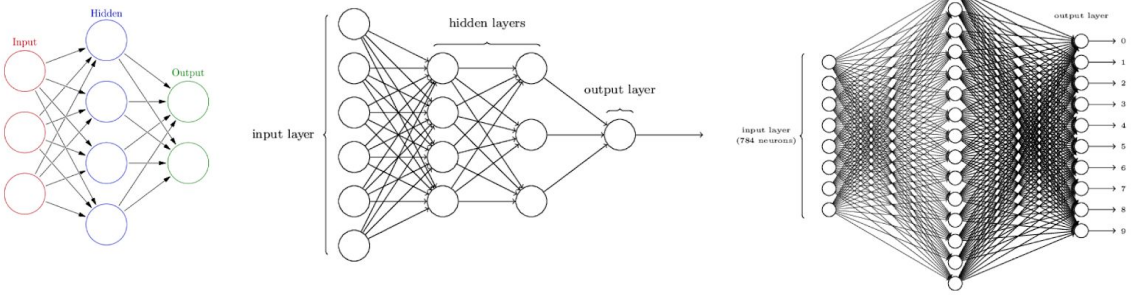


Modelo neuronal artificial

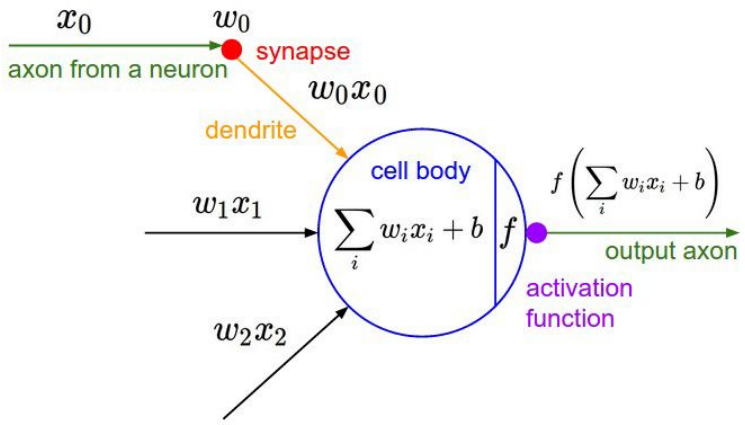
- Un **Perceptrón simple** es la disposición de una **capa de entradas** de activaciones que alimentan a una **capa de salida**



- También conocidas como las neuronas de *McCulloch-Pitts*
- La Red Neuronal Artificial (ANN) más simple se basa en las neuronas *McCulloch-Pitts* etiquetadas con los indices *k,i,j*
- El flujo de activación entre la red de neuronas se realiza a traves de la sipnapsis utilizando **pesos** Wki,Wij

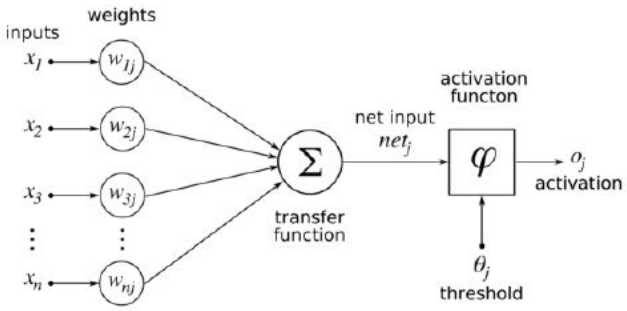


- Posee las capas de **Inputs, Hidden y output layers**.
- Provee una salida deseada a partir de una(s) entrada(s), en mecanismos de aprendizaje supervisado y no supervisado.
- A pesar de que parezca que hay varias salidas, en realidad es una sola salida que se reparte a diferentes perceptrones
- La interpretación biológica de un perceptrón es la siguiente: cuando emite un **1**, esto equivale a **"disparar" un impulso eléctrico**, y cuando es **0**, es cuando no está disparando. El **Bias** indica qué tan difícil es para este nodo en particular enviar una señal.



Funciones de Activación, Capas, Nodos y pesos *(feedfoward)*

Se llama red de ***feed-forward*** a la red de nodos que envía señales en una dirección. Como se observa en la figura, en este tipo de red, se utiliza diferentes formas en la función de activación.



Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^z)$	Multi-layer Neural Networks	

Estos son los tipos de redes más estudiados, sin embargo existen redes neuronales recurrentes **(RNN)** que permiten ciclos en la red. La naturaleza unidireccional de las redes de feed-forward es probablemente es la mayor diferencia entre las redes neuronales artificiales y su equivalente biológico.

La regla de aprendizaje del perceptrón

La regla de aprendizaje Perceptron se desplaza iterativamente alrededor de los pesos Wij y en los límites de decisión para dar las salidas (objetivo) para cada entrada. Si el problema es linealmente separable, los pesos requeridos se encontrarán en un número finito de iteraciones.

Network Activations:

$$out_j = \text{sgn}(\sum_{i=1}^{n} in_i w_{ij} - \theta_j)$$

Perceptron Learning Rule:

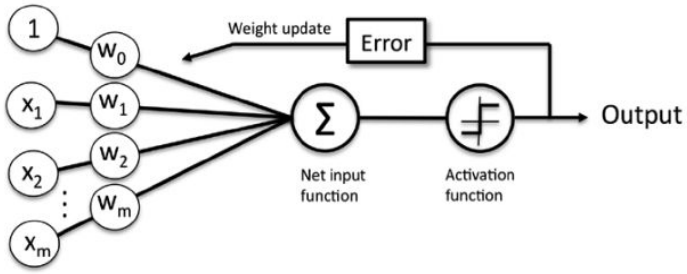
$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$$

$$\Delta w_{ij} = \eta.(targ_j - out_j).in_i$$

Back-Propagation

1. Se hace feedforward.
2. Comparación de la salida obtenida con la deseada.

3. Cálculo del error.
4. Se hace feedforward al reves (backpropagation) del error.
5. Se actualizan los pesos para un mejor modelo.
6. Se continua hasta tener el mejor modelo.



$$E = -\frac{1}{m}\sum_{i=1}^{m}\left(y_i \ln(\hat{y}_i) + (1-y_i)\ln(1-\hat{y}_i)\right)$$

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

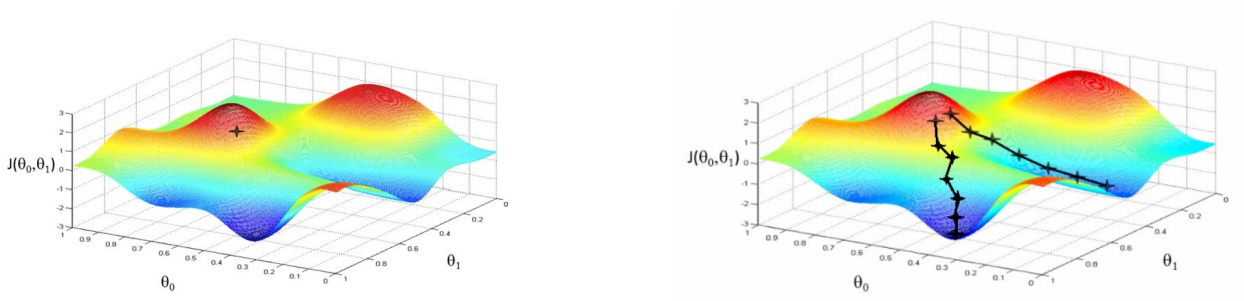
Construcción Red Neuronal Artificial (ANN)

Usando Redes Neuronales Artificiales para resolver problemas reales en procesos multi-etapas :

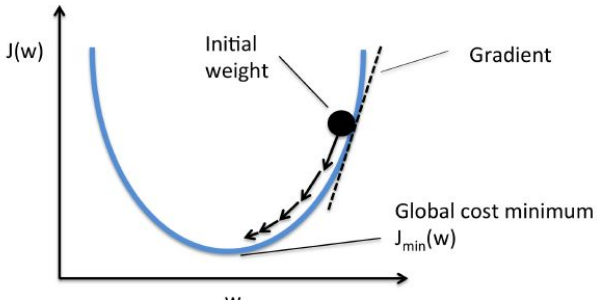
1. Comprender y especificar el problema en términos de entradas y salidas requeridas.
2. Tome la forma más simple de red que pueda resolver el problema.
3. Trate de encontrar pesos de conexión y umbrales neuronales adecuados para que la red produzca salidas apropiadas para cada entrada, en sus datos de entrenamiento.
4. Pruebe la red en sus datos de entrenamiento, y también en datos nuevos (validación / prueba).
5. Si la red no funciona lo suficientemente bien, regrese a la etapa 3 y trabaje más.
6. Si la red aún no funciona lo suficientemente bien, regrese a la etapa 2 y trabaje más.
7. Si la red aún no funciona lo suficientemente bien, regrese a la etapa 1 y trabaje más.
8. Problema resuelto - pasar al siguiente problema.

Después del entrenamiento, generalmente se espera que la red **generalice** bien, es decir, produzca **salidas apropiadas** para patrones de prueba que nunca antes se haya visto.

Gradiente descendiente



- Minimizar el error en la salida al variar el valor de los pesos (W). Se utiliza el el gradiente descendiente, para determinar hacia cual dirección cambiar los pesos
- Por ejemplo el gradiente puede representarse como las derivadas parciales entre los pesos.



$$\nabla E = \left(\frac{\partial}{\partial w_1} E, \cdots, \frac{\partial}{\partial w_n} E, \frac{\partial}{\partial b} E \right)$$

Consideraciones prácticas

Hay una serie de importantes consideraciones prácticas/de implementación que deben tomarse en cuenta al entrenar redes neuronales:

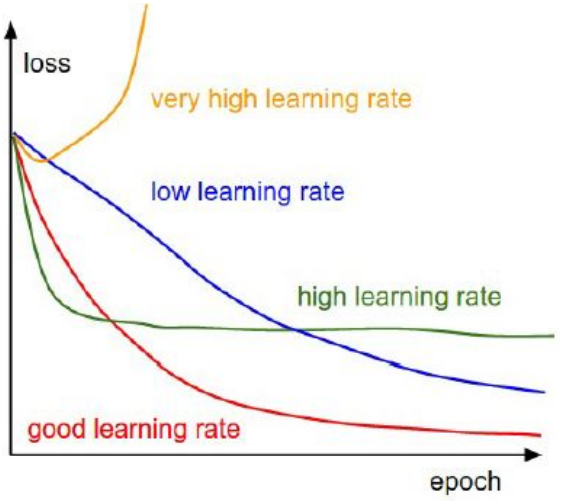
1. ¿Necesitamos preprocesar los datos de entrenamiento? ¿Si es así, cómo?
2. ¿Cómo pueden las unidades ocultas necesitamos?
3. ¿Son algunas funciones de activación mejores que otras?
4. ¿Cómo elegimos los pesos iniciales desde los cuales comenzamos el entrenamiento?
5. ¿Deberíamos tener diferentes tasas de aprendizaje para las diferentes capas?
6. ¿Cómo elegimos las tasas de aprendizaje?
7. ¿Cambiamos los pesos después de cada patrón de entrenamiento, o después de todo el conjunto?
8. ¿Cómo evitamos los puntos planos en la función de error?
9. ¿Cómo evitamos los mínimos locales en la función de error?
10. ¿Cuándo dejamos de entrenar?

En general, las respuestas a estas preguntas dependen en gran medida del problema

Tasa de aprendizaje

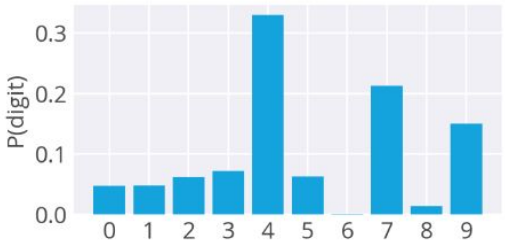
- Variable denominada como α que se usa de la mano el gradiente descendiente para dar pasos más pequeños.

$$w_i(final) = w_i(inicial) + \alpha \times \nabla E_i$$



Normalización (Función SOFTMAX)

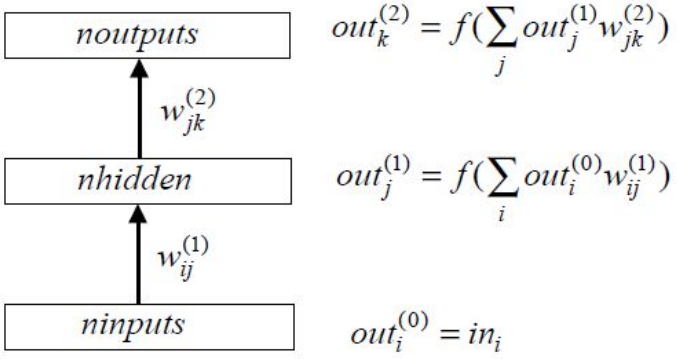
- La salida queda distribuida entre 0 y 1, como una distribución de probabilidad.
- Se normaliza respecto a la totalidad de las salidas.
- Para clasificación múltiple se usa en las capas de salida.




```
>>> import numpy as np
>>> z = [1.0, 2.0, 3.0, 4.0, 1.0, 2.0, 3.0]
>>> softmax = np.exp(z)/np.sum(np.exp(z))
>>> softmax
array([0.02364054, 0.06426166, 0.1746813 , 0.474833  , 0.02364054,
       0.06426166, 0.1746813 ])
```

El clasificador MLP

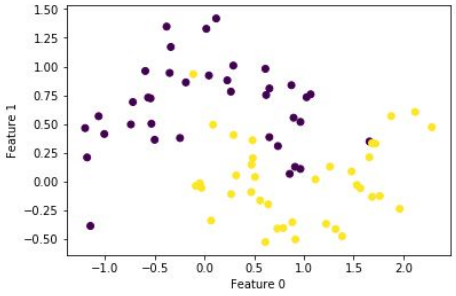
- Para resolver problemas no-lineales, se utilizan funciones de activación **No-Monotonicas**
- Más convenientemente, se puede extender un perceptron simple a un **Multi-Layer Perceptrons,** donde incluye al menos una capa oculta de neuronas con **funciones de activación no lineales** (i.e sigmods)



- Si la activación en la capa oculta fuera lineal, la red sería equivalente a una red de una sola capa, y no podría hacer frente a problemas separables no lineales

```
mlp = MLPClassifier(solver='lbfgs', random_state=0, hidden_layer_sizes=[10, 10])
mlp.fit(X_train, y_train)

plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train)

plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
```

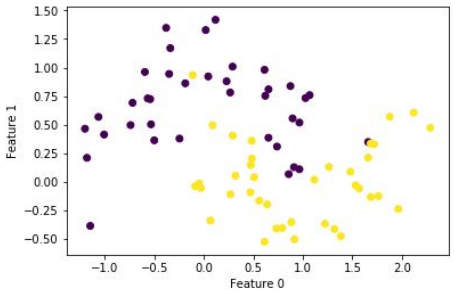
```
<matplotlib.text.Text at 0x10ed9a748>
```



(a) MLP Función de activación **RELU**

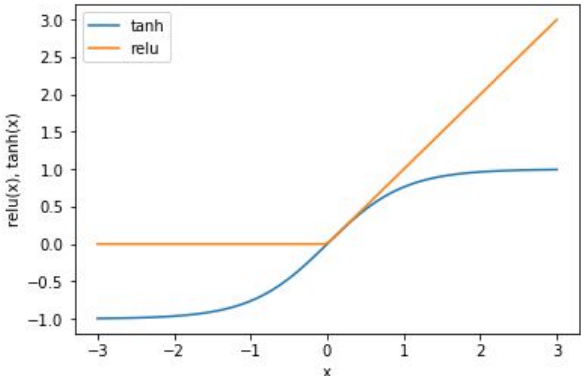
```
mlp = MLPClassifier(solver='lbfgs', activation='tanh', random_state=0, hidden_layer_sizes=[10, 10])
mlp.fit(X_train, y_train)

plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train)

plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
```

```
<matplotlib.text.Text at 0x10ed8d748>
```



(b) MLP Función de activación **TANH**

Ambos casos utilizan 2 capas ocultas de 10 nodos.

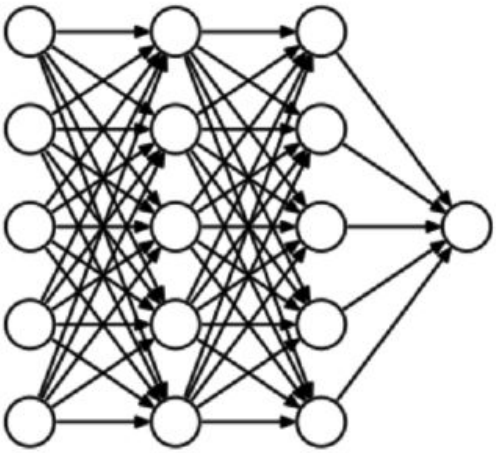


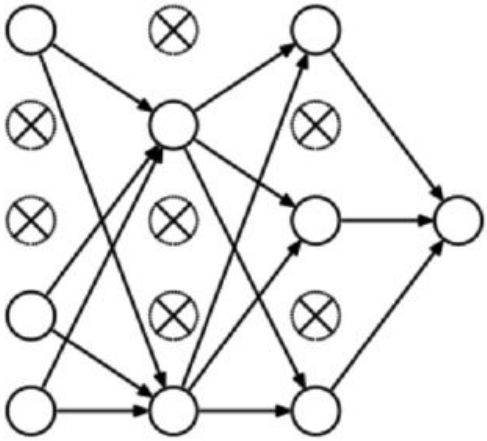
Regularización

- Método para prevenir overtting.
- Consiste en penalizar pesos grandes que acaparen el entrenamiento.
- **L1:** Se reduce el número de pesos al enviar a cero los cercanos a cero.
- **L2:** Se mantienen los pesos pero en valores bajos (cercanos a cero)

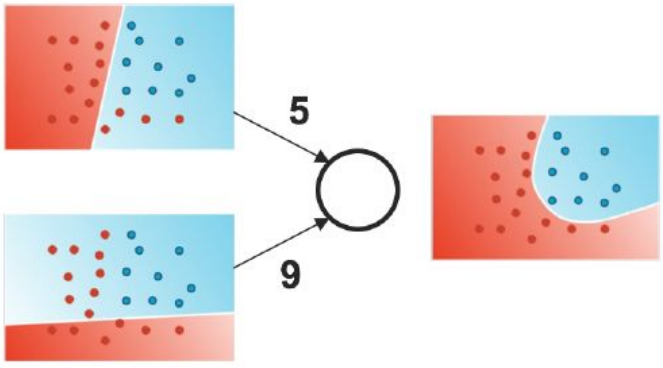
Dropout

- Neuronas con altos pesos pueden acaparar el entrenamiento.
- Consiste en “apagar" algunas neuronas para darle oportunidad a otras de ser parte del entrenamiento.





Proyección ANN



http://playground.tensorflow.org/

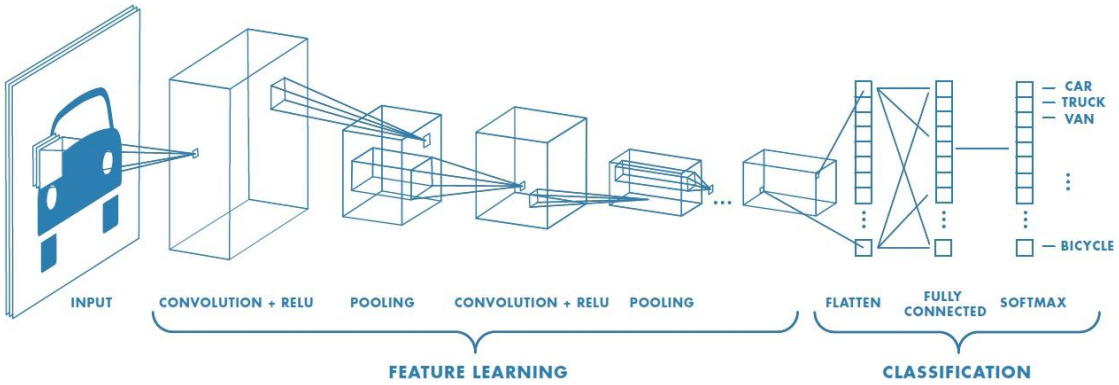
ANN - Ejemplo Clasificación

https://github.com/elisiojsj/Handwritten-Digit-Recognition-with-TFLearn-and-MNIST

ANN - Ejemplo Regresión

https://github.com/fmezacr/DeepLearning/blob/master/dlnd-your-first-neural-network.ipynb

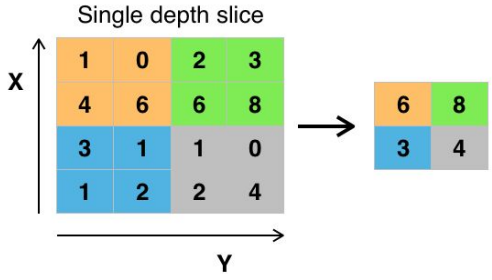
CNN - Convolutional Neural Networks



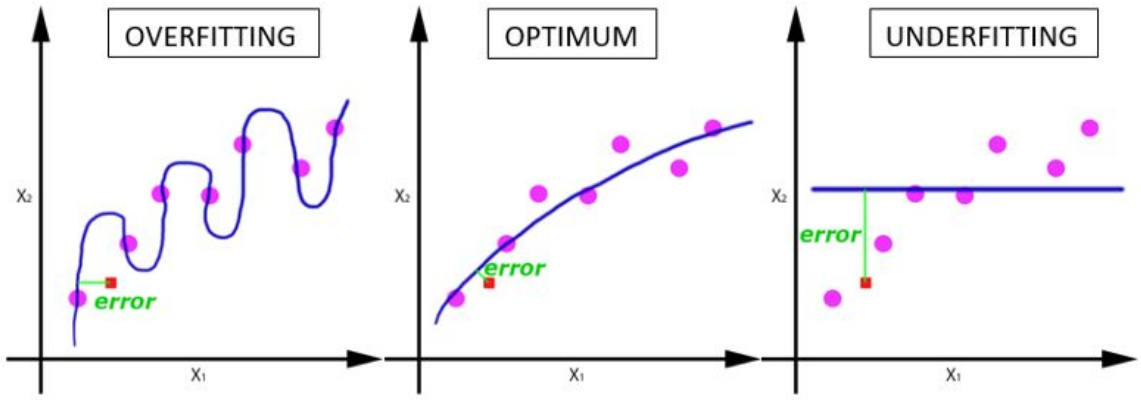
- Arquitectura propuesta con 3 capas:
  - **Convolución:**

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) g(t - \tau)\, d\tau$$

    - Incluye la ecuación para no-linealidad, típicamente **RELU**
  - **Pooling:** Consiste en una reducción de dimensiones, después de la convolución.



  - **Capa de conexión (FCL):** Capa totalmente conectada, modelo convencional MLP.
  - **EPOCH:**
    - Un Epoch es cuando un conjunto de datos ENTERO se pasa hacia adelante y hacia atrás a través de la red neuronal solo UNA VEZ.
    - A medida que el número de EPOCHS aumenta , se cambia el número de veces que se cambia el peso en la red neuronal y la curva pasa de estar de **underfitting** del nivel adecuado a una curva óptima para lograr un **overfitting**



- **BATCH**
  - no puede pasar todo el conjunto de datos a la red neuronal de una vez. Entonces, divide el conjunto de datos en Número de BATCHs o conjuntos de partes
- **BATCH SIZE**
  - Número total de ejemplos de entrenamiento presentes en un solo BATCH.
- **ITERATIONS**
  - Es la cantidad de lotes necesarios para completar un EPOCH.
  - EJEMPLO
    - Se puede dividir el conjunto de datos de 2000 ejemplos en lotes de 500 y luego tomará 4 iteraciones para completar 1 época

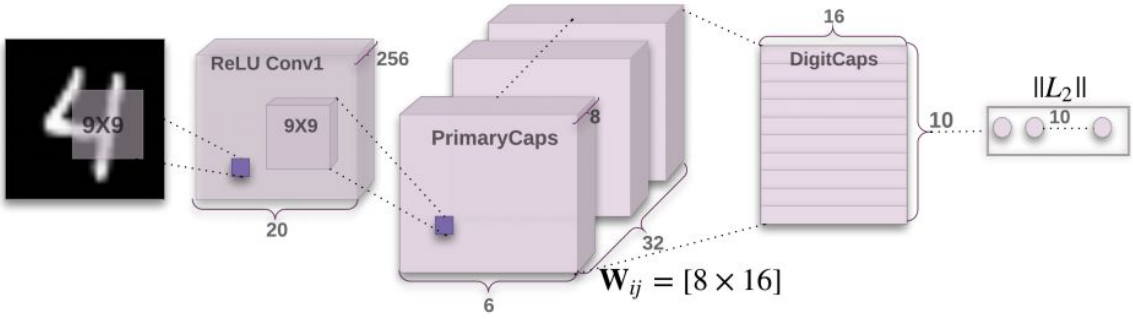
CNN más comunes

- AlexNet: https://en.wikipedia.org/wiki/AlexNet
- LeNet:http://deeplearning.net/tutorial/lenet.html
- ZF Net.
- GoogLeNet:https://github.com/rugbyprof/5443-Data-Mining/wiki/GoogleNet
- VGGNet. https://cv-tricks.com/cnn/understand-resnet-alexnet-vgg-inception/
- ResNet.

CapsNets - Capsule NN

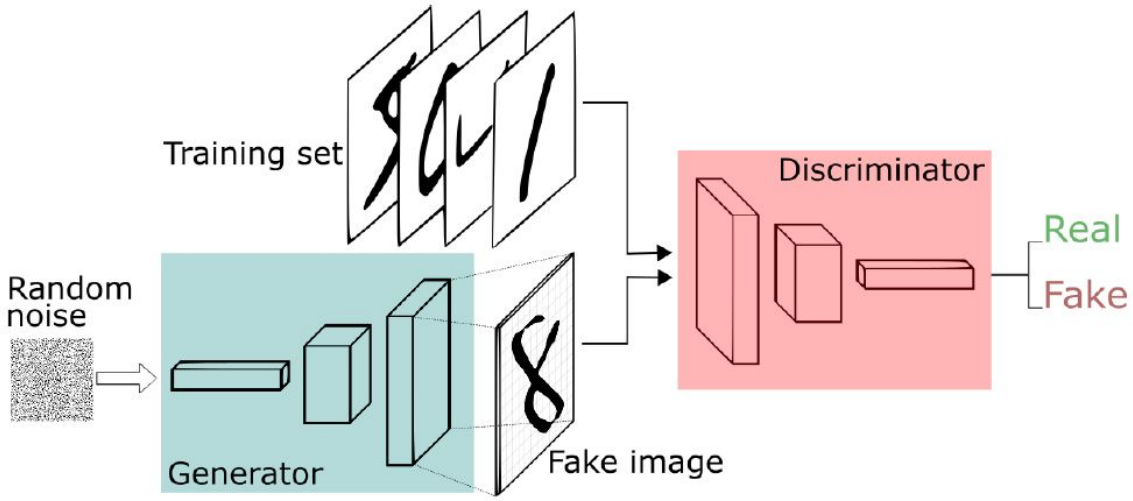
Propuesta reciente (27OCT17) del Prof. Hinton, para atacar algunos problemas de las CNN convencionales:

- El pooling usado por las CNN pierde precision en relaciones espaciales entre diferentes capas.
- Las CNN no son buenas en cuanto al reconocimiento de objetos o formas que varen su posicion (e.g rotacion).



GAN - Generative Adversial Networks

https://github.com/fmezacr/DeepLearning/blob/master/dlnd_face_generation.ipynb



LIbrerías comunes

- TENSORFLOW
  - Librera de uso libre desarrollada por Google y liberada en 2015 para programación basada en flujo de datos y util para aplicaciones de Machine Learning en especial para ANN.
- KERAS
  - API de alto nivel, se ejecuta una capa arriba de TensorFlow (entre otros). De facil implementacion y amigable.
- SciFlow
  - Se ejecuta por arriba de TensorFlow y se presenta con el formato habitual de Scikit-Learn.

Herramientas Utiles

-

Recomendaciones

- Validar que los datos tengan sentido, datos invertidos?, ceros por error?. Desplegar grupos de datos aleatorios.
- En caso de un error, intentar el uso de datos aleatorios, identificar si el error se repite.
- Eliminar ruido en el conjunto de datos (NaNs).
- Intentar re-ordenar los datos (shue).
- Reducir el des-balance de datos.
- Buscar en lo posible una cantidad de datos considerable.
- No usar batches muy grandes.
- Considerar la normalización en caso de ser necesario.
- Intentar con una versión “simple” del problema.
- No depender de una sola métrica.
- Problemas de generalización? Intentar con más capas.
- Revisar los pesos de inicialización.
- Comprender el efecto de variación en los hiper-parámetros.
- Probar con varios optimizadores.
- Comprender el efecto de variación del learning rate.
- Validar el efecto de la regularización.