




Aprendizaje Supervisado

Matías Marenchino - Cristian Cardellino



Tercera Clase

Temario de la Clase

- ¿Qué es aprendizaje supervisado?
- Repaso general de introducción al aprendizaje automático.
 - Regresión Lineal y Polinomial, Regresión Logística, Perceptrón.
- Árboles de decisión
- Naive Bayes
- Support Vector Machines
- Ensemble learning.
 - Random Forest, Bagging, Boosting.
- Redes neuronales.
 - Perceptrón multicapa.
- Sistemas de recomendación.
 - Filtrado colaborativo, máquinas de factorización.
- Prácticas de reproducibilidad

Ensemble Learning

Árboles de decisión

Aprende a diferenciar los datos en base a reglas de decisión.

Los nodos del árbol representan las reglas. Las hojas asignan la clase o el valor.

El árbol se particiona recursivamente. Para obtener el resultado, simplemente se siguen los nodos de decisión de acuerdo a los datos y se asigna la clase final.

Es un algoritmo de “caja blanca”, ya que puede visualizarse fácilmente e interpretarse por los humanos (a diferencia de un algoritmo de “caja negra” como son las redes neuronales).

Son buenos con datos de mucha dimensionalidad (high dimensional data).

Elimina de la
clase pasada

Árboles de decisión: Algoritmo

Selecciona el mejor atributo de acuerdo a alguna métrica (e.g. ganancia de información).

Hacer un nodo de decisión con ese atributo, que particione los datos en subconjuntos de menor tamaño.

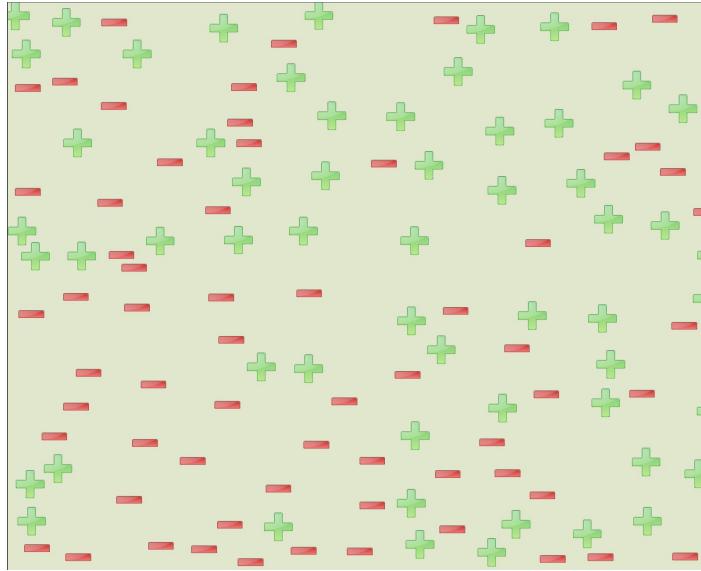
Repetir recursivamente el procedimiento para cada nodo hijo.

El algoritmo se detiene si:

- Todos los ejemplos del subconjunto son de la misma clase.
- Todos los elementos del subconjunto son constantes con respecto al atributo/s de interés del nodo actual.

Filminas de la
clase pasada

Árboles de decisión: Cómo son las fronteras



"Frontera de
Decisión"



Ensemble Learning: Motivación

Se basan en la idea de que el **trabajo en conjunto** debería dar mejores resultados.

De tal forma, un conjunto de modelos, al combinarse, deberían tener mejor performance.

Consideremos el caso de tres modelos: M1, M2 y M3:

Cuando las predicciones son iguales, es sencillo definir cómo predecir; qué hacemos en el caso en que difieran?

- Le creemos al mejor de los modelos?
- Votamos por mayoría?

Ensemble Learning

Un modelo "ensemble" se constituyen como un conjunto de diferentes modelos.

Habitualmente, un modelo "ensemble" es más preciso que los modelos que lo constituyen. Intuitivamente, esto se debe a que "dos aprenden mejor que uno".

Ensemble Learning: Aproximación de justificación

Asumimos que tenemos un modelo M , formado por n modelos: M_1, M_2, \dots, M_n .

Cuando un modelo recibe un dato x , el modelo predice $M(x)$ a partir de las predicciones $M_i(x)$, a partir de votación (clase más votada).

Cómo determinamos cuán bien funciona el modelo?

Para responder, consideramos:

- Todos los clasificadores M_1, M_2, \dots, M_n son igualmente precisos (precisión p)
- Los errores en la clasificación hecha por cada clasificador son independientes: $\mathbf{P}(M_j \text{ erróneo} \mid M_k \text{ erróneo}) = \mathbf{P}(M_j \text{ erróneo})$

Ensemble Learning: Aproximación de justificación

Para hacer el ejemplo más concreto, consideremos:

- $p = 0.8$
- $n = 5$

Entonces,

$$\begin{aligned} P(\text{la predicción de } M \text{ es correcta}) &= \\ &= P(\text{al menos 3 de los 5 } M_i \text{ predican correctamente}) \\ &= \binom{5}{5} 0.8^5 0.2^0 + \binom{5}{4} 0.8^4 0.2^1 + \binom{5}{3} 0.8^3 0.2^2 \\ &= 0.942 \end{aligned}$$

Ensemble Learning: Bagging

Bagging

Es un método para hacer aprendizaje por "ensemble".

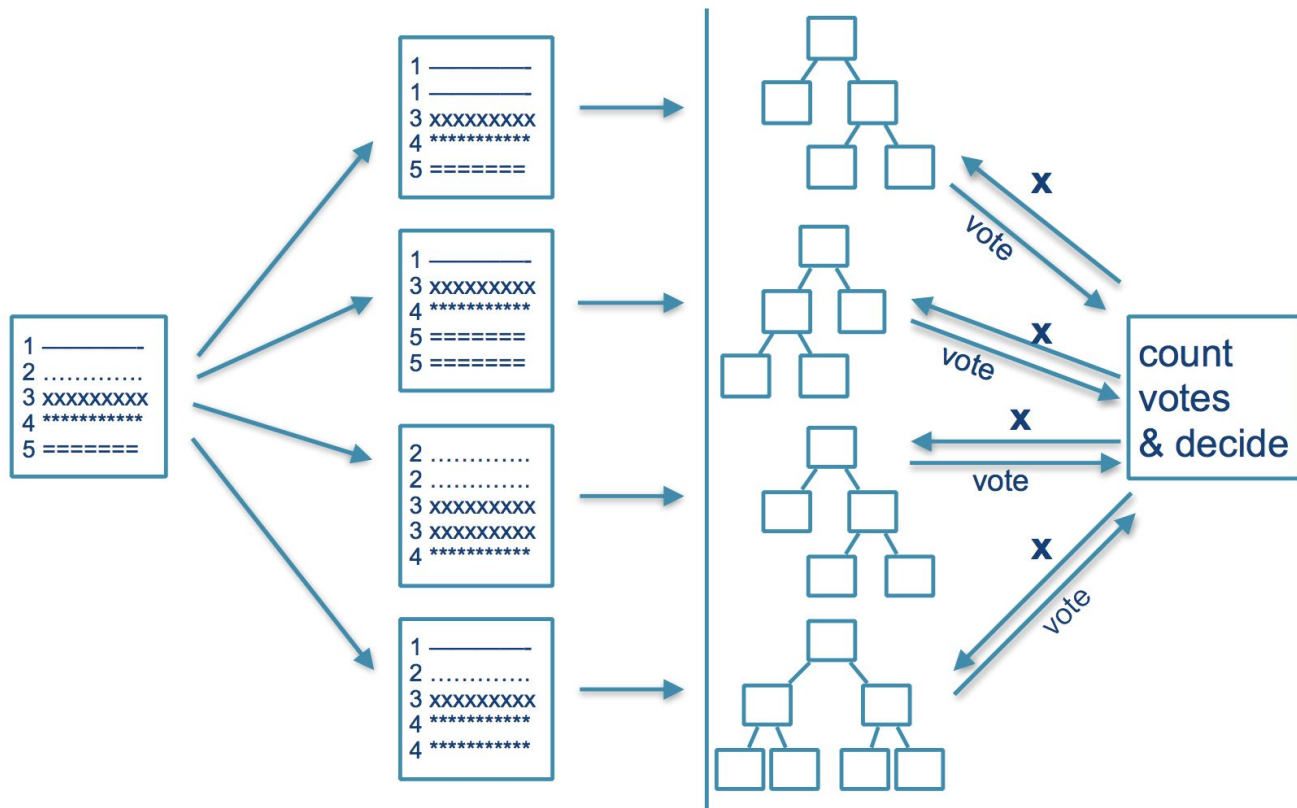
Si usamos el mismo modelo sobre los mismos datos, obtendremos los mismos resultados (salvo inicializaciones aleatorias). Entonces, de cierta forma, debemos introducir variaciones en los datos:

Si \mathbf{D} es el dataset inicial; repetimos k veces lo siguiente:

- Generar \mathbf{D}_i a partir de las entradas de \mathbf{D} , seleccionando aleatoriamente y con reposición $|\mathbf{D}|$ instancias de \mathbf{D} .
- Entrenamos el modelo \mathbf{M}_i a partir de \mathbf{D}_i .

Nuestro modelo \mathbf{M} selecciona las predicciones más frecuentes de $\{\mathbf{M}_i\}_i$.

Bagging gráficamente (para árboles de decisión)



Bagging para árboles de decisión

Bagging generalmente funciona bien para algoritmos "inestables":

- Un algoritmo es inestable si pequeñas variaciones en el dataset pueden generar modelos muy diferentes.

Resulta que los árboles de decisión son inestables.

Si bien lo anterior incentiva al uso de bagging para árboles de decisión, tenemos una desventaja importante: entrenar **k** árboles es **k** veces más caro que entrenar uno solo.

Random Forests

Random Forests

Las Random Forests son una modificación a Bagging para Árboles de Decisión. Para evitar la sobrecarga, se simplifican los modelos:

- Cuando se crean los árboles (cuando se entrenan), todas las features son consideradas o evaluadas al crear un nodo.
- Para los Random Forests, en cada nodo se consideran sólo M atributos elegidos aleatoriamente (parámetro *max_features* en sklearn).
- Usualmente, se toma:
$$M = \sqrt{\text{number of attributes}}$$

Random Forests

El modelo se puede resumir en los siguientes pasos:

Repetir **k** veces:

- Construir un dataset a partir del original, como se hace con bagging.
- Construir un árbol de decisión:
 - Para cada nodo del árbol, seleccionar **M** variables y construir el nodo "óptimo" entre esas features.
 - Repetir hasta que el árbol esté completamente construido (no se hace pruning).

Random Forests

"Los Random Forests son uno de los métodos de aprendizaje más eficientes y precisos a la fecha" (2008): Caruana: *An empirical evaluation of supervised learning in high dimensions. ICML 2008.*

El algoritmo es sencillo, fácil de implementar, fácil de usar y requiere de poco ajuste de parámetros.

Es relativamente sencillo debuggear los Random Forests; aunque comparado con los Árboles de Decisión, pueda resultar menos interpretable.

Demo Time (demo 7)

Boosting

Boosting

Método para hacer aprendizaje por "ensemble".

Para el ejemplo de los árboles de decisión, vimos que con bagging, entrenamos grandes árboles sobre versiones del material de entrenamiento generados con reposición; y luego votamos por mayoría.

Boosting pretende darle mayor importancia a los modelos que mejor performance tienen sobre los datos. Así, podemos decir que, con boosting, se entrenan árboles sobre versiones "pesadas" de los datos de entrenamiento. Y luego se clasifica por mayoría pero pesando los modelos.


Boosting

Cada modelo, de cierta forma, define qué features considerará el siguiente modelo.


Se seleccionan datos como en bagging (bootstrapping), pero en este caso, los datos son pesados con cierto criterio. De hecho, algunos registros serán usados más frecuentemente.

El proceso es el siguiente:

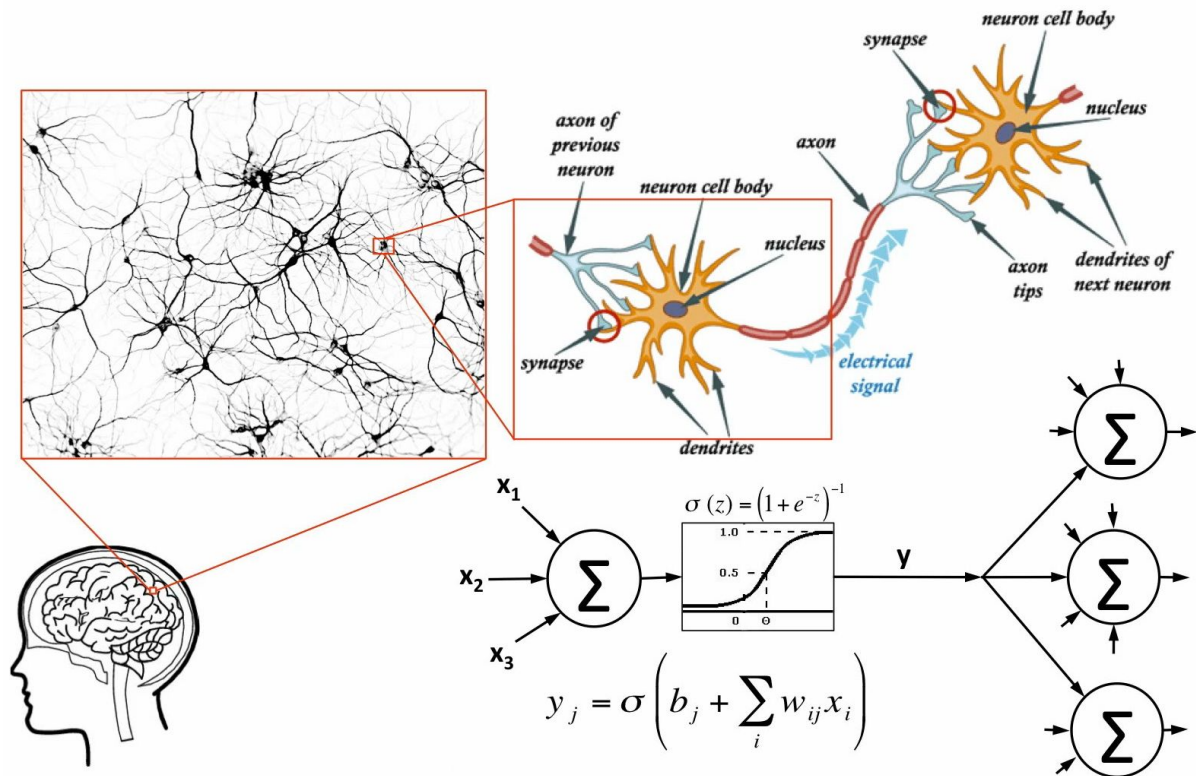
- Dado un modelo, determinar qué registros son más "erróneos" y darles mayor pesos para los modelos siguientes).
- Dado un modelo, determinar su performance para darle mayor peso a los "mejores" modelos.



Redes Neuronales (Introducción)



Por qué redes "neuronales"?



Repaso de la Regresión Logística

- Dado x , el objetivo es encontrar: $\hat{y} = P(y = 1|x)$
- Cuál es la forma más sencilla de transformar un vector x ?

Repaso de la Regresión Logística

- Dado x , el objetivo es encontrar: $\hat{y} = P(y = 1|x)$
- Cuál es la forma más sencilla de transformar un vector x ?

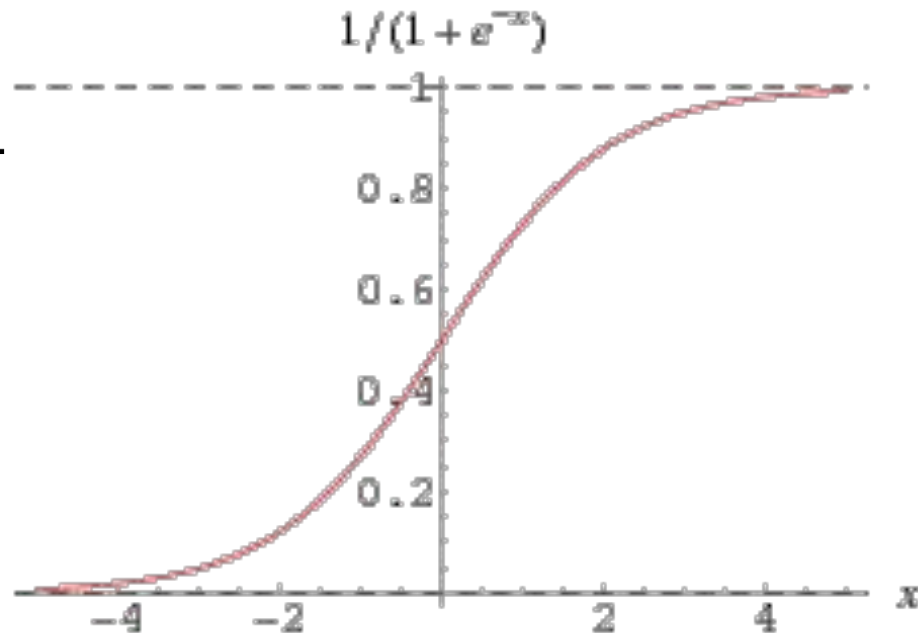
$$\hat{y} = w^T x + b$$

- Pero nos gustaría que \hat{y} fuese una probabilidad: $0 \leq \hat{y} \leq 1$

Repaso de la Regresión Logística: Sigmoid

Función de Sigmoid

$$f(x) = \frac{1}{1 + \exp(-x)}$$



Repaso de la Regresión Logística: Coste

Se disponen de m instancias $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots (x^{(m)}, y^{(m)})\}$

Se pretende predecir $\hat{y}^{(i)} \approx y^{(i)}$

En general, se busca minimizar $\mathcal{L}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$

Para la regresión logística, en su lugar se usa

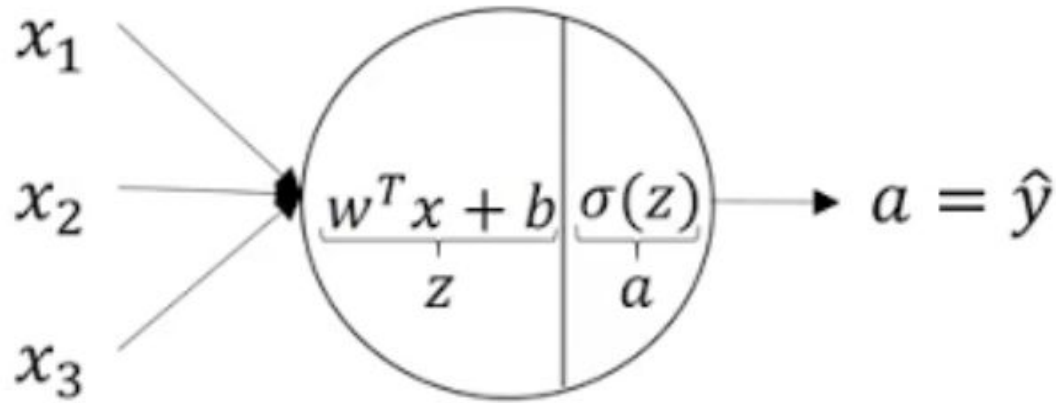
$$\mathcal{L}(\hat{y}, y) = -[y \log \hat{y} + (1 - y) \log (1 - \hat{y})]$$

Siendo entonces la función de coste: $\mathcal{J}(w, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$

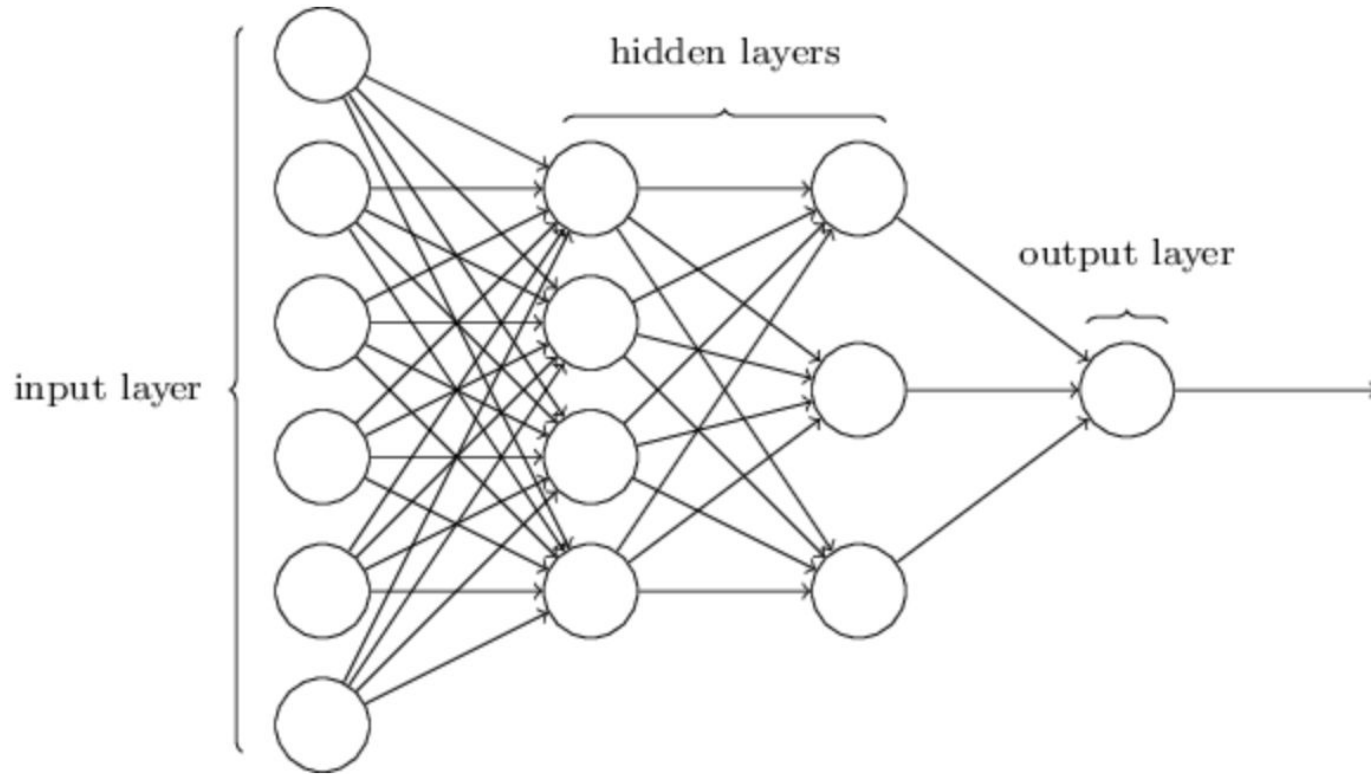
Para minimizarla, usamos descenso por gradientes. Necesitaremos:

$$\frac{\partial}{\partial w_1} \mathcal{J}(w, b) = \frac{1}{m} \sum_{i=1}^m \frac{\partial}{\partial w_1} \mathcal{L}(a^{(i)}, y^{(i)})$$

La Regresión Logística como una neurona



Redes Neuronales



Redes Neuronales

Funciones de Activación:

- Sigmoid (como en la regresión logística)
- tanh: $\frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$
- Rectified Linear Unit (ReLU): $\max(0, x)$
- Leaky ReLU




Demo Time
(demo 8)



Demo Time

(demo 8)



Home Installation Documentation ▾ Examples

Google Custom Search

Previous	Next	Up
1.16. Probabl...	2. Unsupervis...	1. Supervised...

scikit-learn v0.19.1
[Other versions](#)

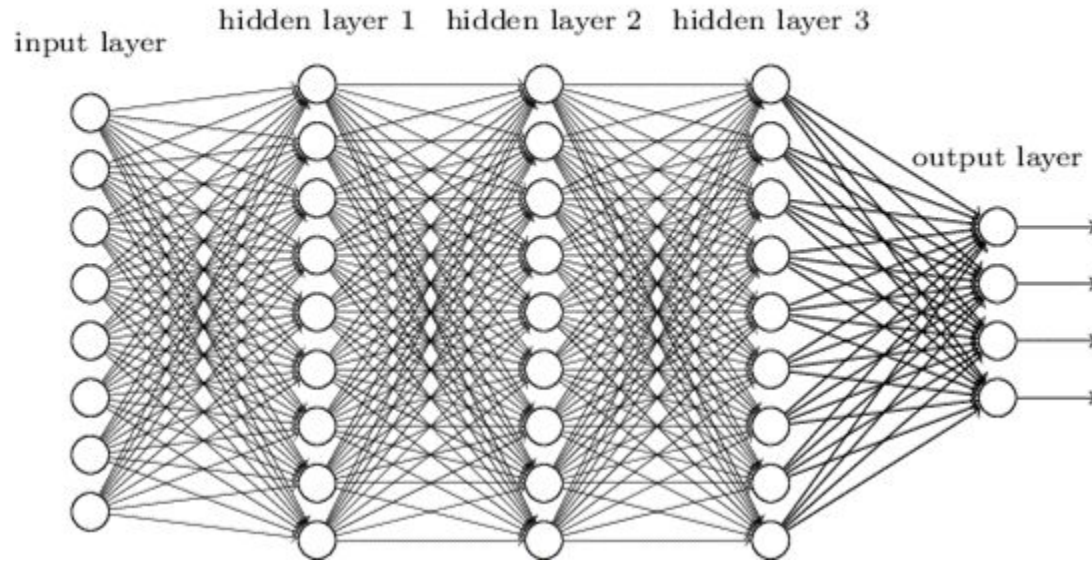
Please [cite us](#) if you use the software.

1.17. Neural network models (supervised)

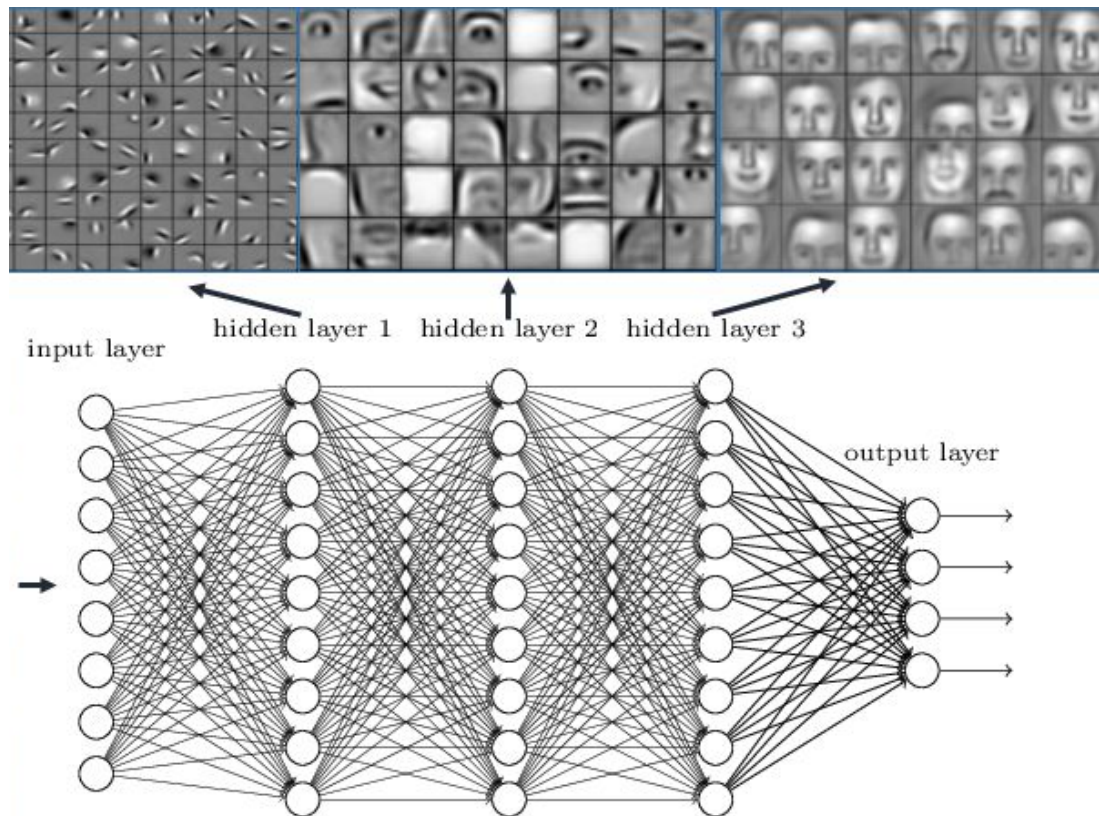
Warning: This implementation is not intended for large-scale applications. In particular, scikit-learn offers no GPU support. For much faster, GPU-based implementations, as well as frameworks offering much more flexibility to build deep learning architectures, see [Related Projects](#).

1.17.1. Multi-layer Perceptron

Redes Neuronales Profundas (Deep Learning)



Redes Neuronales Profundas



Redes Neuronales

Dataset:

- Train/Test/Validation



Redes Neuronales

Dataset:

- Train/Test/Validation



- Ahora?
 - Muchísimos datos ($>> 10.000.000$ registros)



**Asegurarse que train/ test /
validation vienen de la
misma distribución**

Redes Neuronales: sesgo y varianza

Underfitting (high bias):

- Ampliar la red
- Cambiar la arquitectura de la red

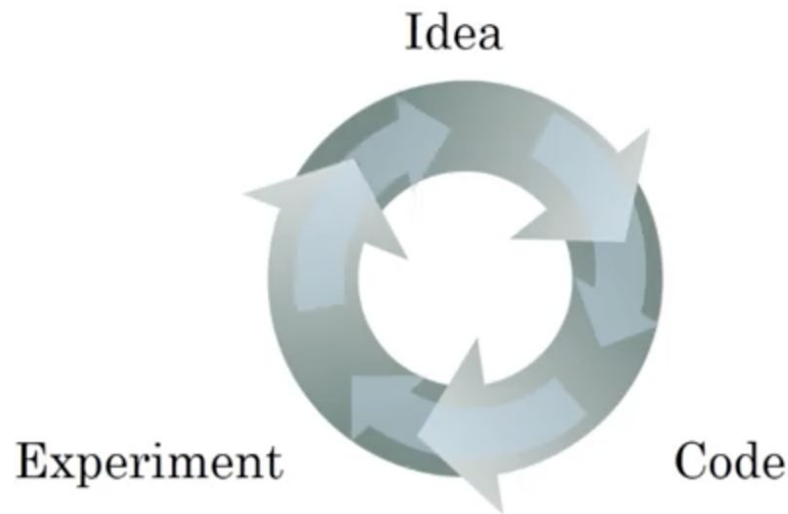
Overfitting (high variance):

- Agregar más datos
- Regularización
- Cambiar la arquitectura de la red

Redes Neuronales

Cómo se determinan?

- el número de capas ocultas (hidden layers)?
- el número de unidades (units)?
- qué función de activación usar?
- ...



Redes Neuronales: Regularización

Logistic regression:

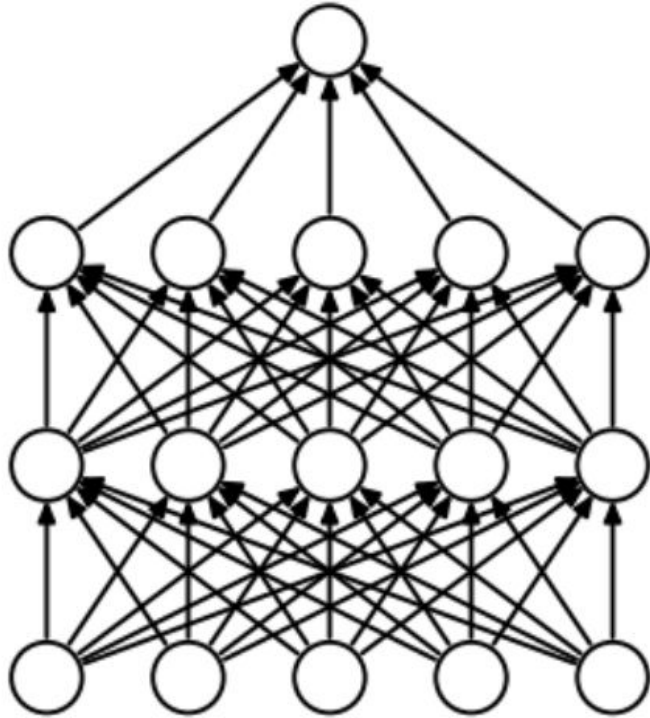
$$J(\theta) = -\frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \log h_{\theta}(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

Neural network:

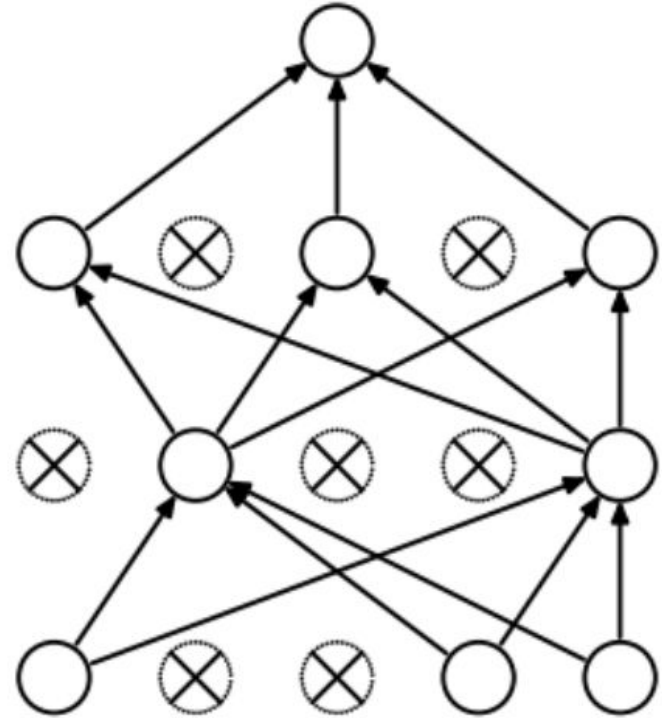
$$h_{\Theta}(x) \in \mathbb{R}^K \quad (h_{\Theta}(x))_i = i^{th} \text{ output}$$

$$J(\Theta) = -\frac{1}{m} \left[\sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \log(h_{\Theta}(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_{\Theta}(x^{(i)}))_k) \right] \\ + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{ji}^{(l)})^2$$

Redes Neuronales: Dropout



(a) Standard Neural Net



(b) After applying dropout.



Fin de la tercera clase

