{desafío}
latam_

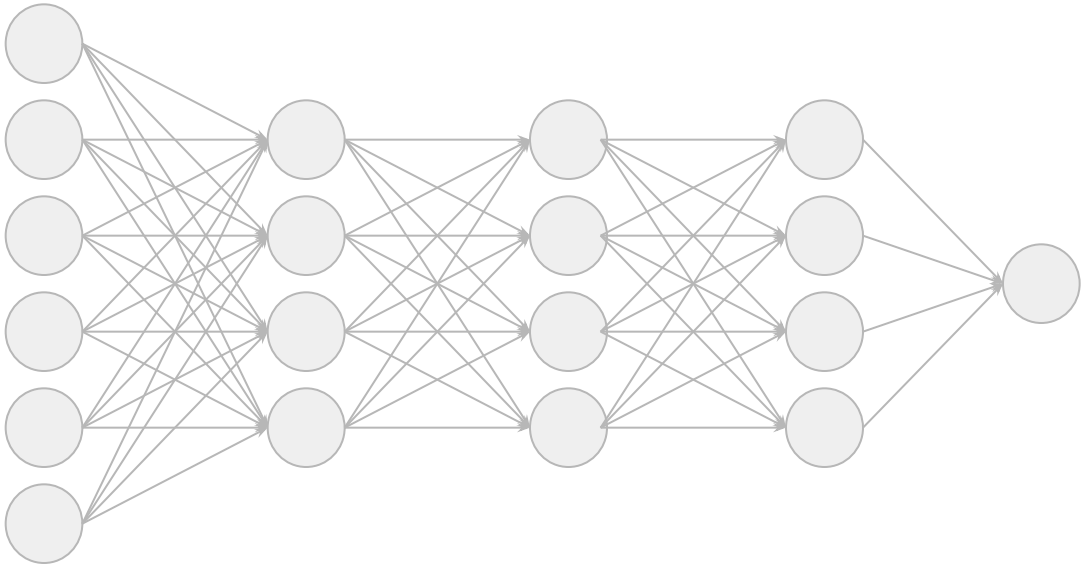
Diseño de Redes Neuronales _

Motivación

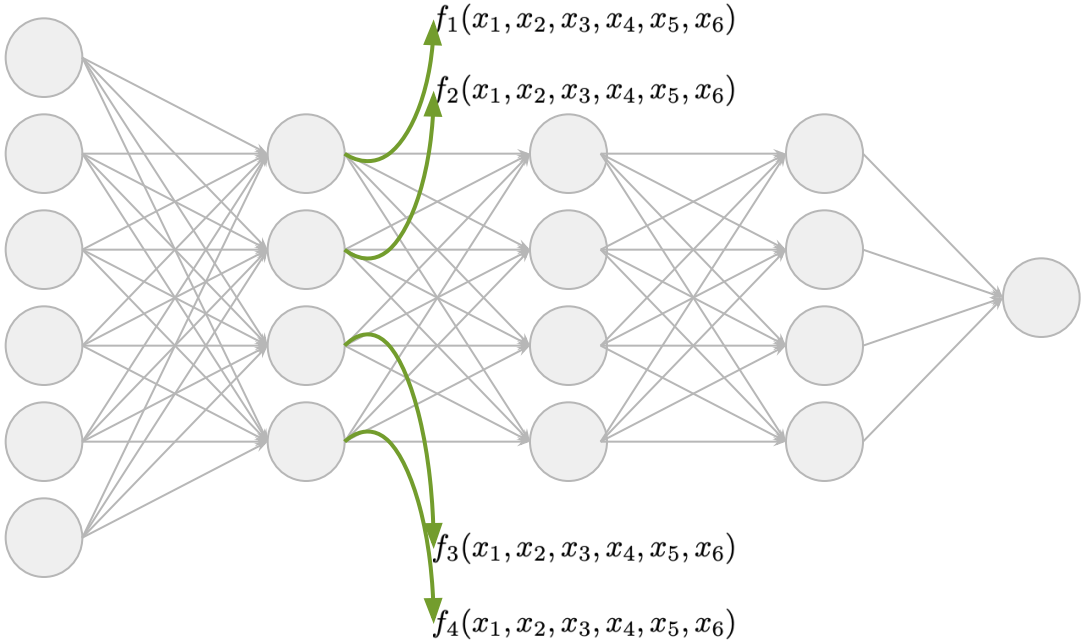
¿Qué sabemos hasta ahora?

- Sabemos que la unidad base de una red neuronal es un perceptrón.
- Podemos entenderlo como un proceso donde ponderamos múltiples observaciones con pesos, los sumamos y en base a ello lo reexpresamos en un término no-lineal.
- Cuando juntamos múltiples perceptrones en paralelo, hablamos de una capa de neuronas.
- Estas capas de neuronas permite generar representaciones complejas del fenómeno de estudio.
- Resulta que también podemos implementar múltiples capas dentro de una red neuronal.
- Antes de entrar a las redes con múltiples capas, debemos mencionar un supuesto teórico que justifica su implementación.

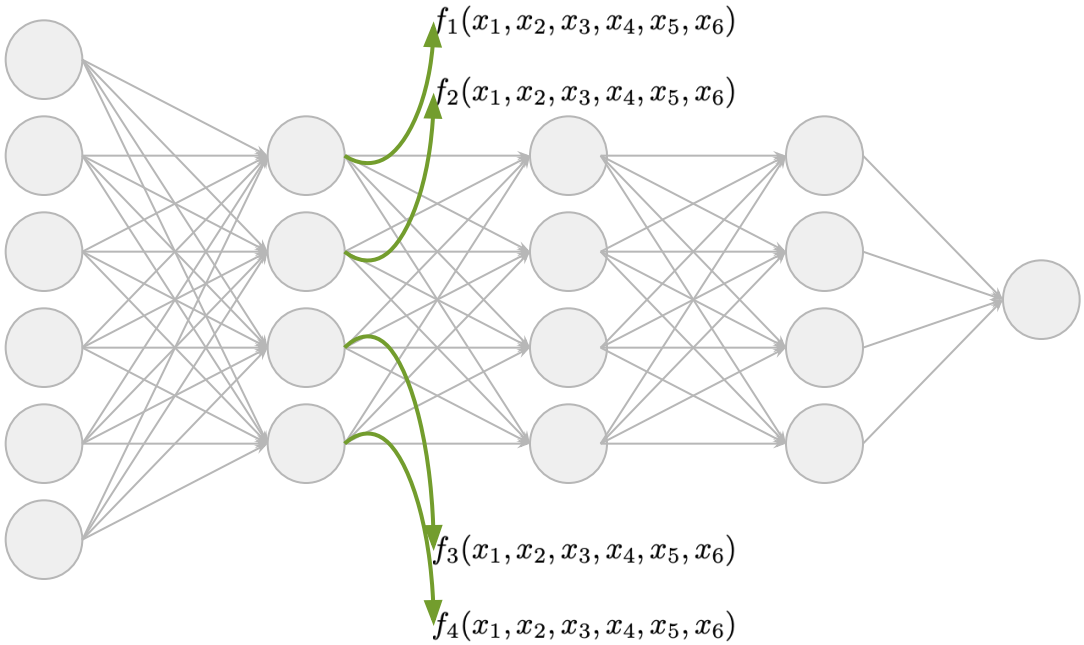
Teorema de Aproximación Universal



Teorema de Aproximación Universal

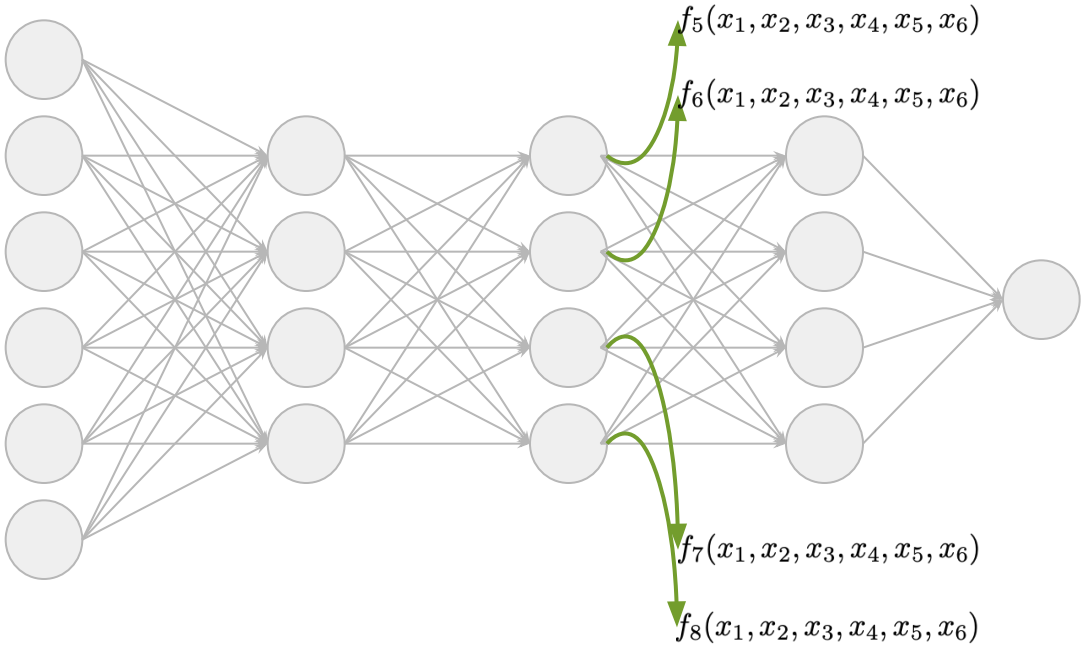


Teorema de Aproximación Universal



En su versión más simple, una red neuronal con una capa oculta aprende representaciones estrictamente lineales.

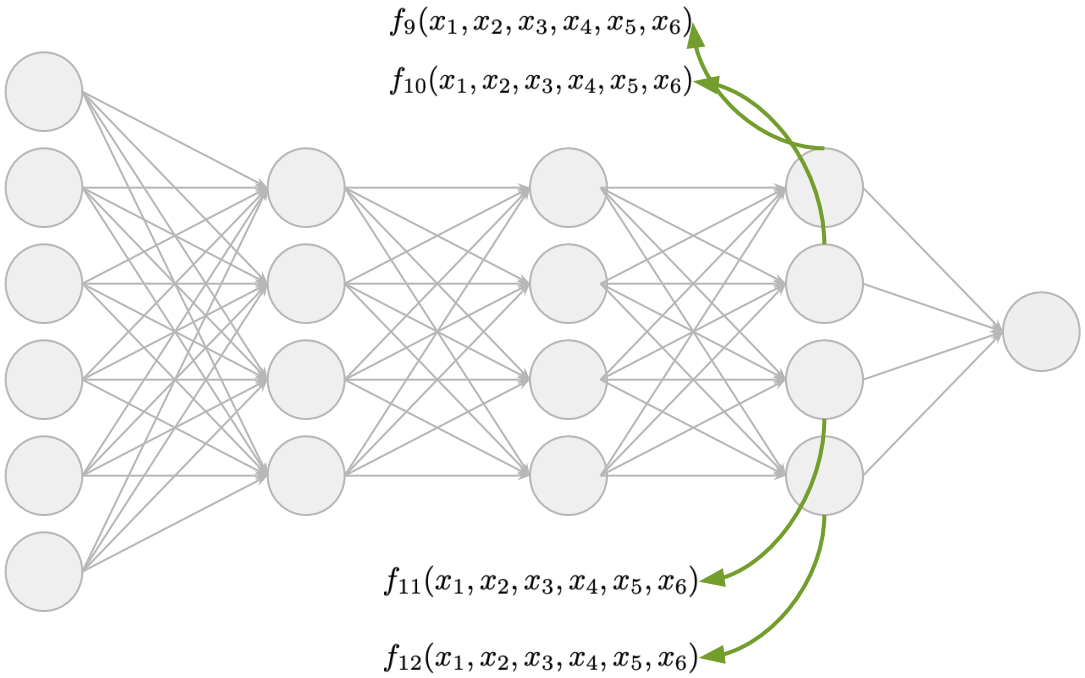
Teorema de Aproximación Universal



En su versión más simple, una red neuronal con una capa oculta aprende representaciones estrictamente lineales.

El teorema de la aproximación universal establece que **independiente de la función que deseamos representar mediante una red neuronal multicapa, una red neuronal con una profundidad substancial permitirá representar esta función.**

Teorema de Aproximación Universal

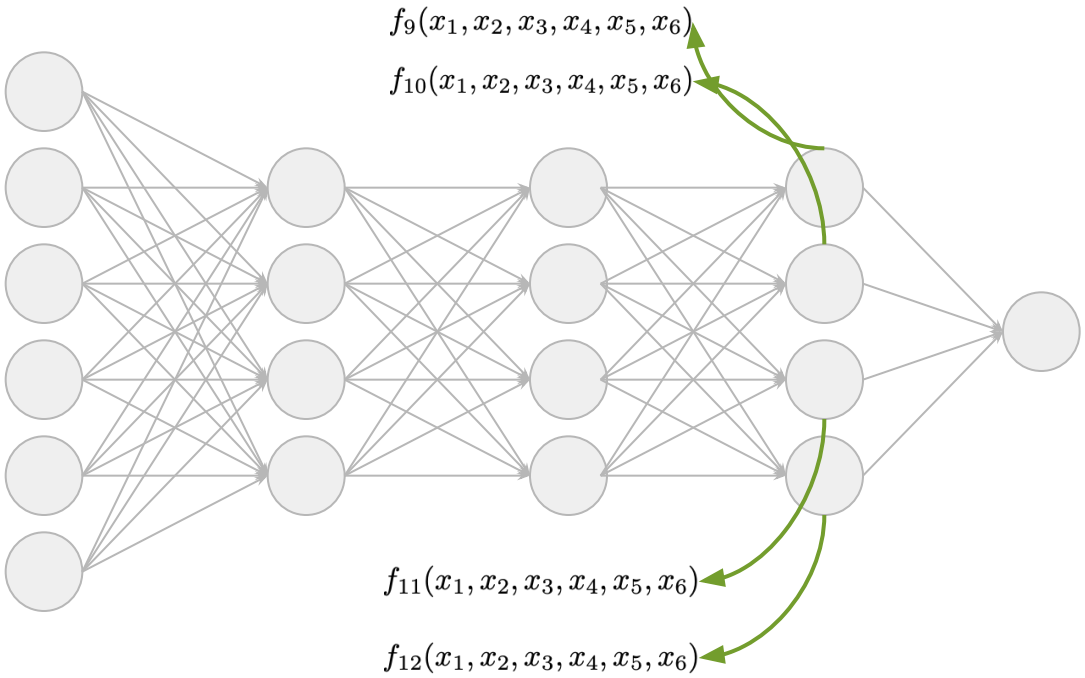


En su versión más simple, una red neuronal con una capa oculta aprende representaciones estrictamente lineales.

El teorema de la aproximación universal establece que **independiente de la función que deseamos representar mediante una red neuronal multicapa, una red neuronal con una profundidad substancial permitirá representar esta función.**

Esto no asegura que la red sea capaz de aprender la función. Esto surge en base a dos elementos:

Teorema de Aproximación Universal

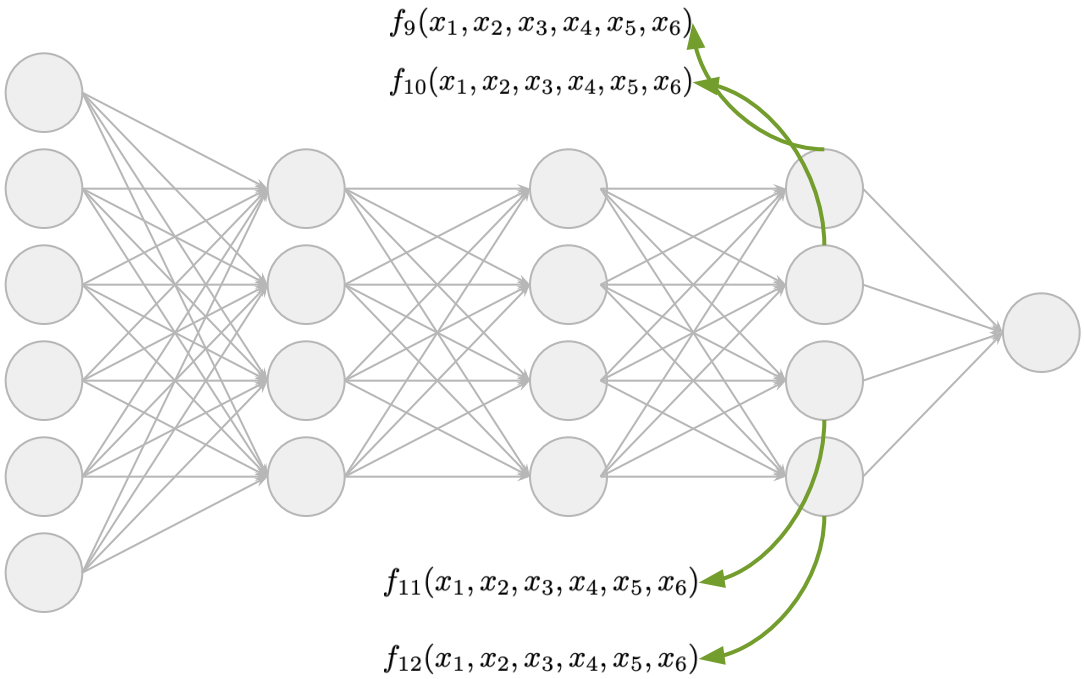


En su versión más simple, una red neuronal con una capa oculta aprende representaciones estrictamente lineales.

El teorema de la aproximación universal establece que **independiente de la función que deseamos representar mediante una red neuronal multicapa, una red neuronal con una profundidad substancial permitirá representar esta función.**

Esto no asegura que la red sea capaz de aprender la función. Esto surge en base a dos elementos:
El **algoritmo de optimización** puede fallar en cuanto a la búsqueda de los parámetros inferidos.

Teorema de Aproximación Universal



En su versión más simple, una red neuronal con una capa oculta aprende representaciones estrictamente lineales.

El teorema de la aproximación universal establece que **independiente de la función que deseamos representar mediante una red neuronal multicapa, una red neuronal con una profundidad substancial permitirá representar esta función.**

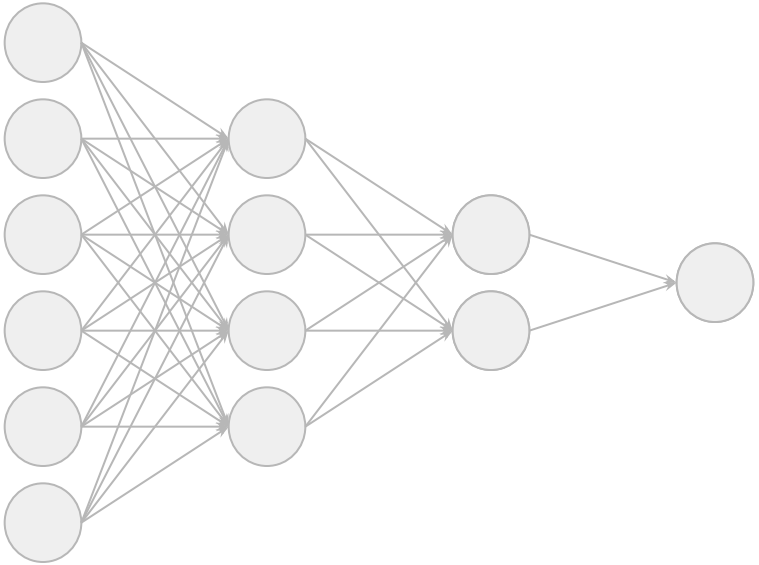
Esto no asegura que la red sea capaz de aprender la función. Esto surge en base a dos elementos:
El **algoritmo de optimización** puede fallar en cuanto a la búsqueda de los parámetros inferidos.
La función representada puede ser incorrecta dado al **overfitting**.

Backpropagation

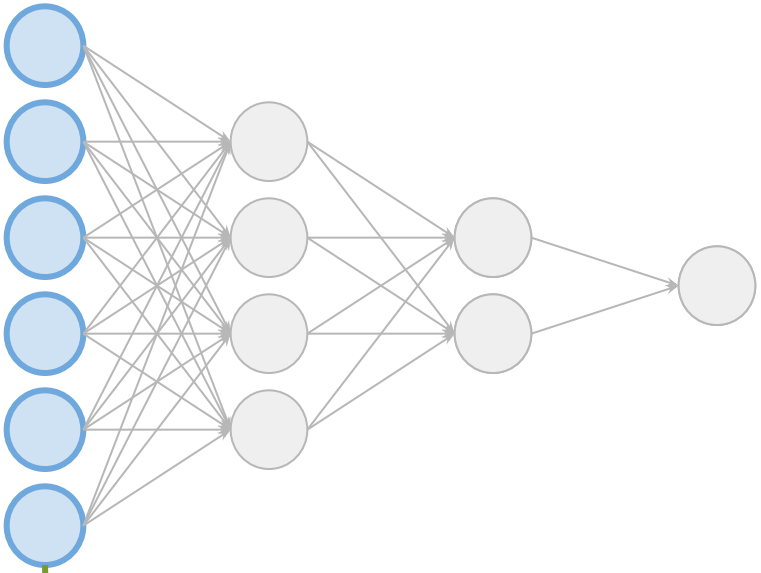
¿Qué es?

- Hasta el momento ignoramos cómo logramos que una red neuronal aprenda los pesos a nivel de capa.
- A grandes rasgos, el proceso de aprendizaje de una red neuronal parte con una selección “aleatoria” de los pesos, que son actualizados posteriormente por un método de retroalimentación.
- Este método contiene dos fases:
  - **Feed-Forward propagation:** Emitimos los impulsos de las neuronas que son captados y procesados por otras, hasta que generamos un output.
  - **Back propagation:** Corregimos los pesos de cada neurona para disminuir la función de pérdida.

Paso Feed Forward

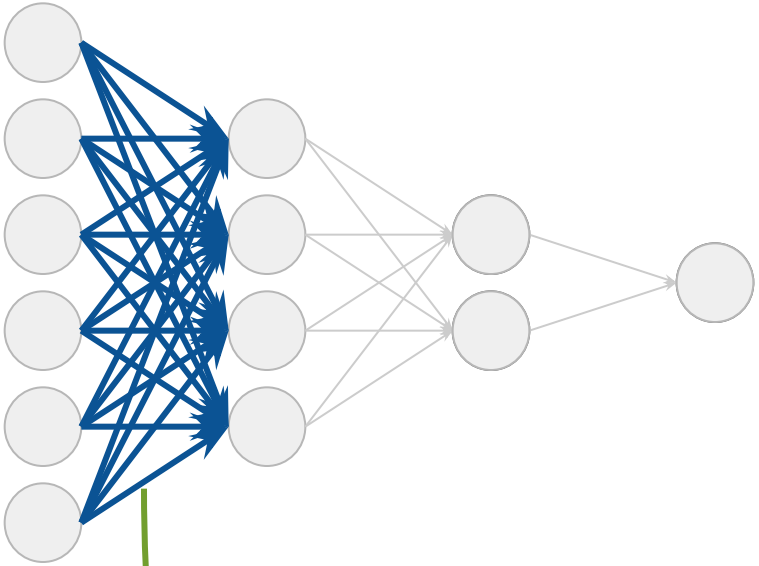


Paso Feed Forward



Input:
Ingresamos un ejemplo (serie de atributos) a nuestra red neuronal.

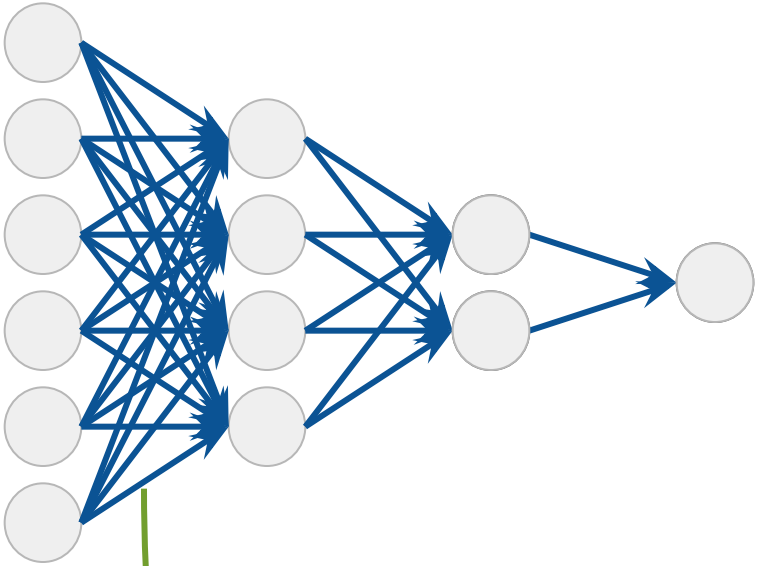
Paso Feed Forward



Pesos:
Los pesos mediante un número pseudoaleatorio que surge de algún **“inicializador de pesos”.**

Alternativas de inicialización:
Normal
Uniforme
Truncada
Ortonormal
Glorot
He
LeCunn

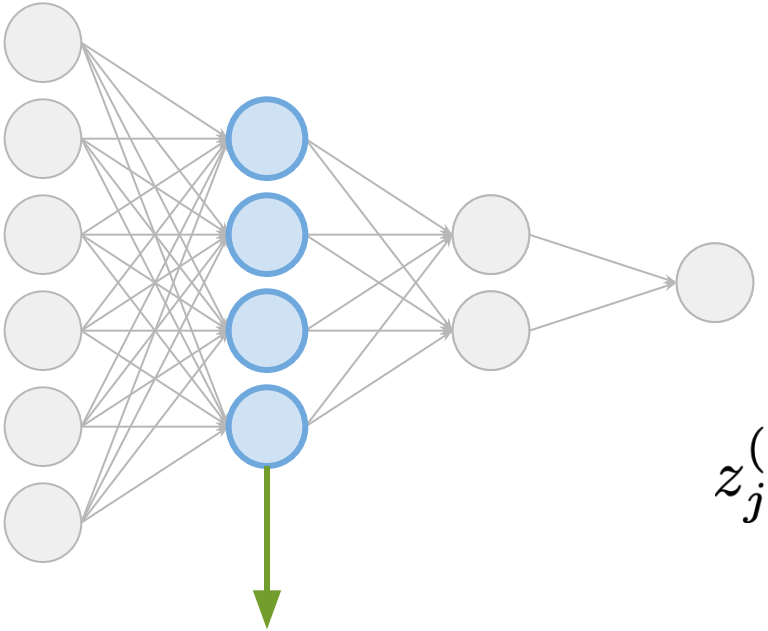
Paso Feed Forward



Pesos:
Los pesos mediante un número pseudoaleatorio que surge de algún **“inicializador de pesos”.**

Alternativas de inicialización:
Normal
Uniforme
Truncada
Ortonormal
Glorot
He
LeCunn

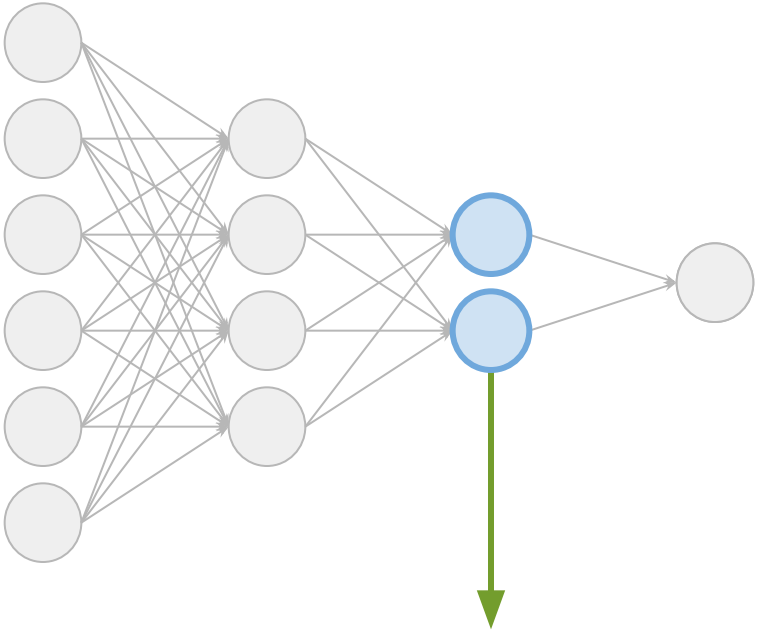
Paso Feed Forward



$$z_j^{(1)} = \sigma(\mathbf{w}_j^{\mathsf{T}} \cdot \mathbf{x})$$

Primera Capa oculta:
Genera una reexpresión de los pesos en base a una función de activación y un sesgo.

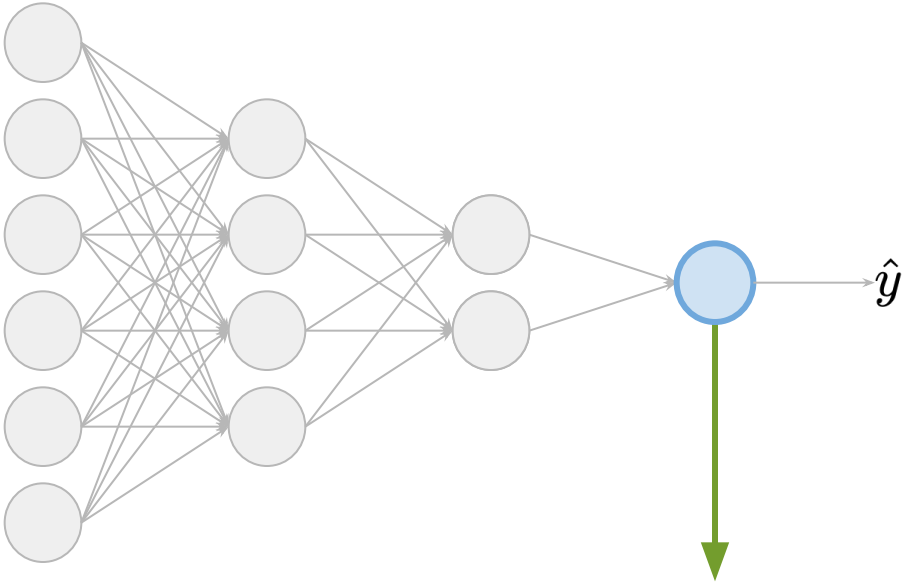
Paso Feed Forward



Segunda Capa oculta:
Genera una reexpresión de los pesos en base a una función de activación y un sesgo.

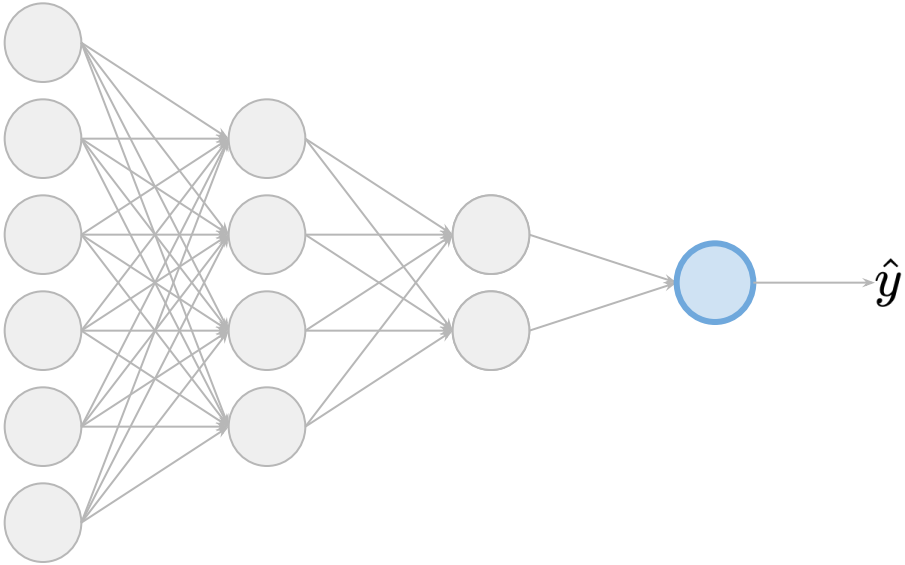
$$z_j^{(2)} = \sigma(\mathbf{w}_j^{\mathsf{T}} \cdot \mathbf{x})$$

Paso Feed Forward

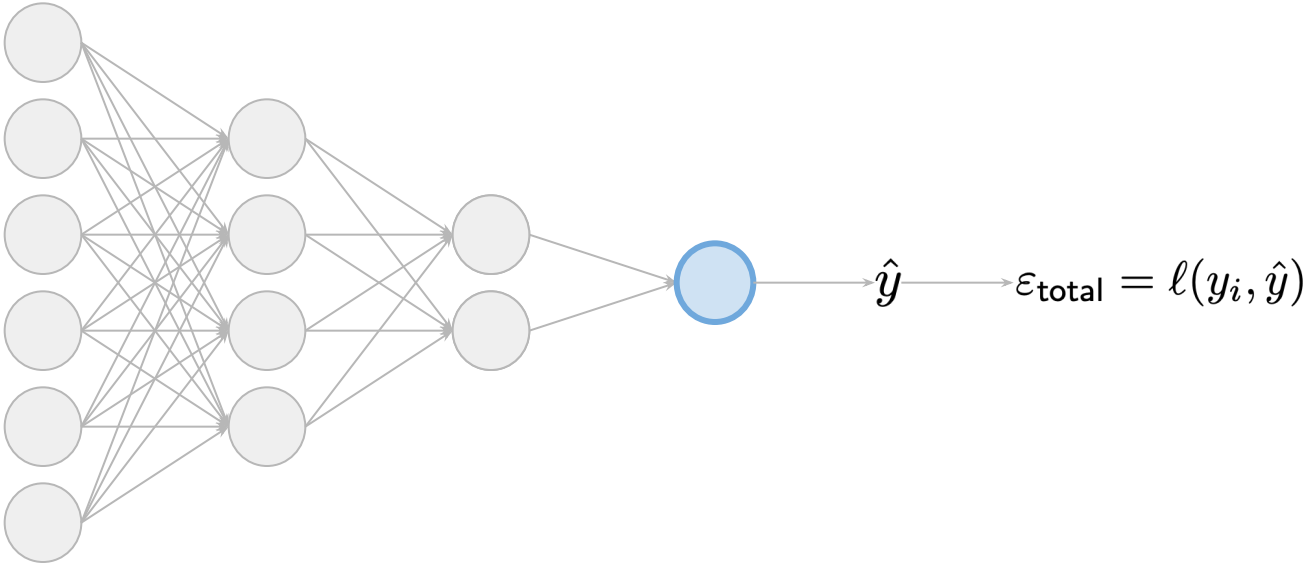


Capa de salida:
Expresa los impulsos de las capas previas.

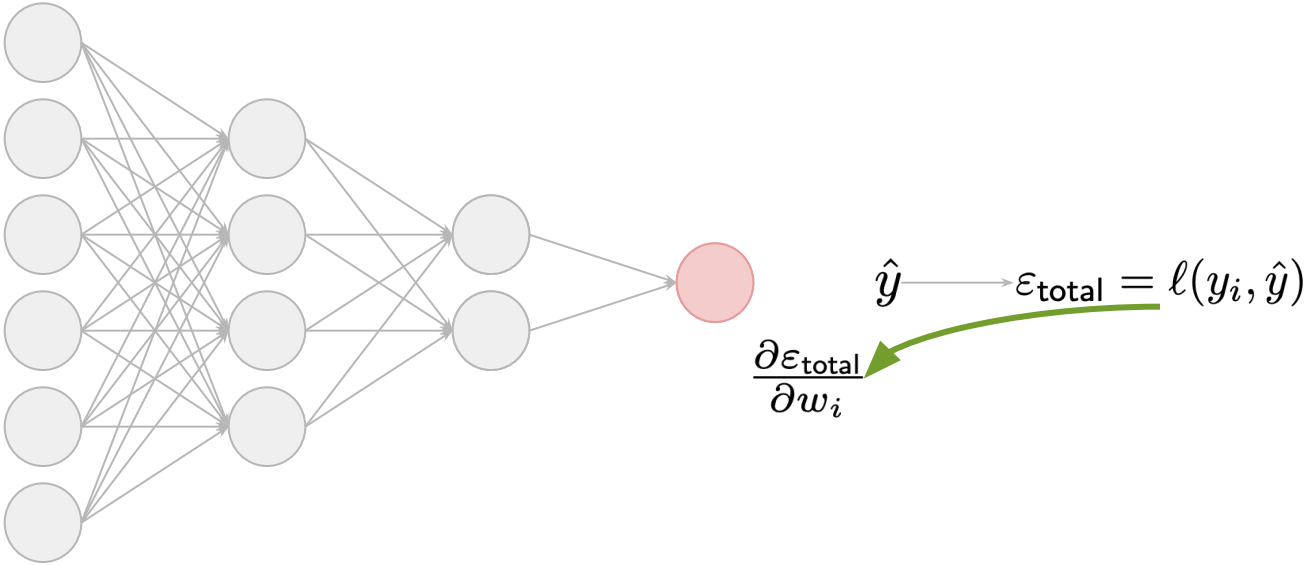
Paso Backpropagation



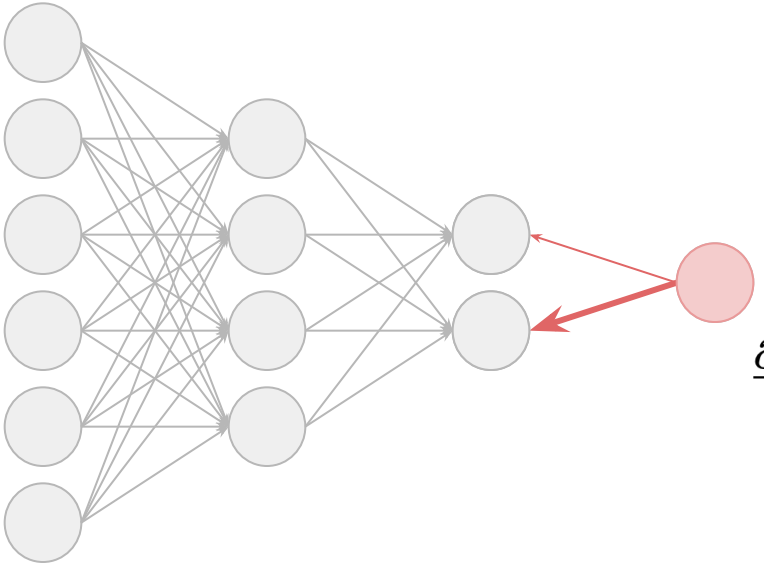
Paso Backpropagation



Paso Backpropagation



Paso Backpropagation

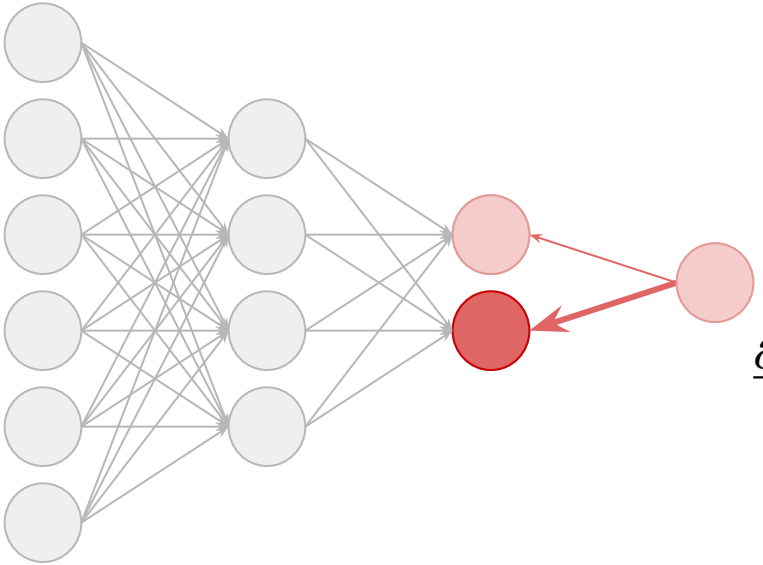


$$\hat{y} \longrightarrow \varepsilon_{\text{total}} = \ell(y_i, \hat{y})$$

$$\frac{\partial \varepsilon_{\text{total}}}{\partial w_i}$$

Evaluamos el gradiente de la función de pérdida respecto a todos los pesos de las capas.

Paso Backpropagation



$$\hat{y} \longrightarrow \varepsilon_{\text{total}} = \ell(y_i, \hat{y})$$

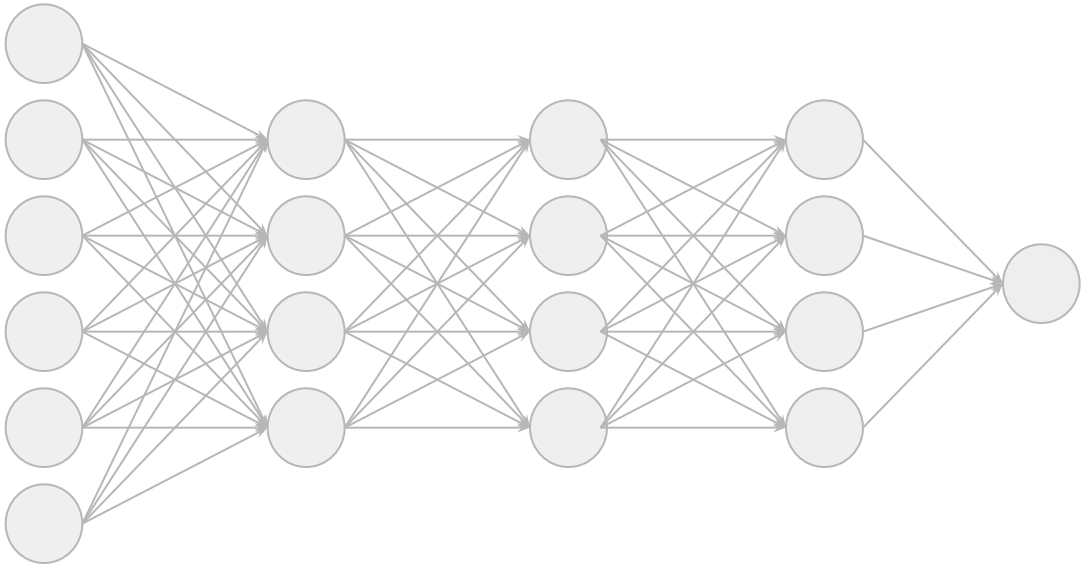
$$\frac{\partial \varepsilon_{\text{total}}}{\partial w_i}$$

Evaluamos el gradiente de la función de pérdida respecto a todos los pesos de las capas.

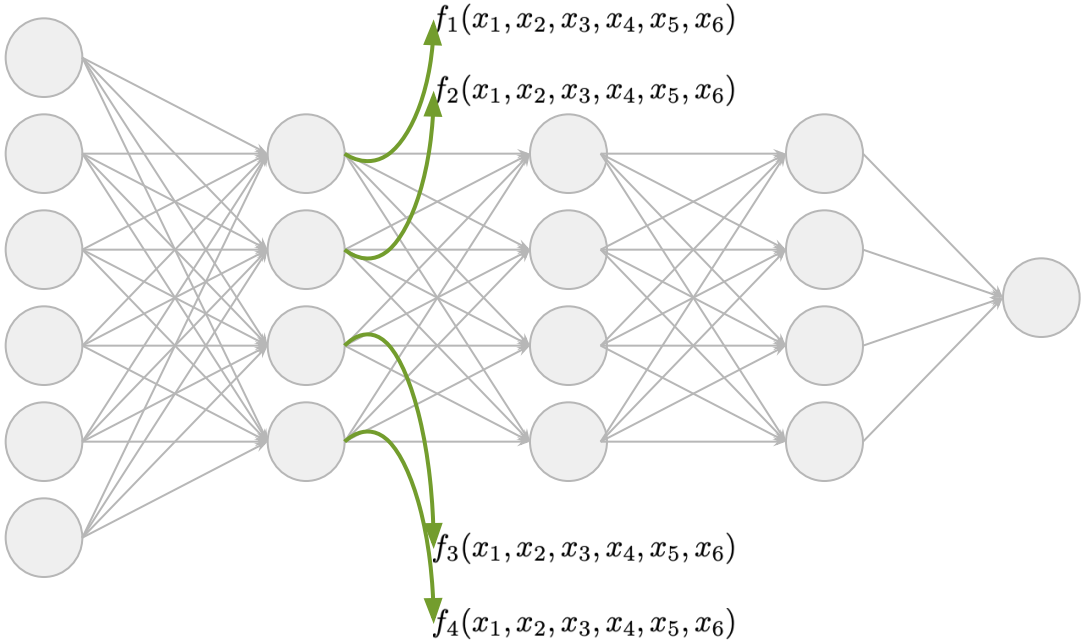
¿Qué sabemos hasta ahora?

- Sabemos que la unidad base de una red neuronal es un perceptrón.
- Podemos entenderlo como un proceso donde ponderamos múltiples observaciones con pesos, los sumamos y en base a ello lo reexpresamos en un término no-lineal.
- Cuando juntamos múltiples perceptrones en paralelo, hablamos de una capa de neuronas.
- Estas capas de neuronas permite generar representaciones complejas del fenómeno de estudio.
- Resulta que también podemos implementar múltiples capas dentro de una red neuronal.
- Antes de entrar a las redes con múltiples capas, debemos mencionar un supuesto teórico que justifica su implementación.

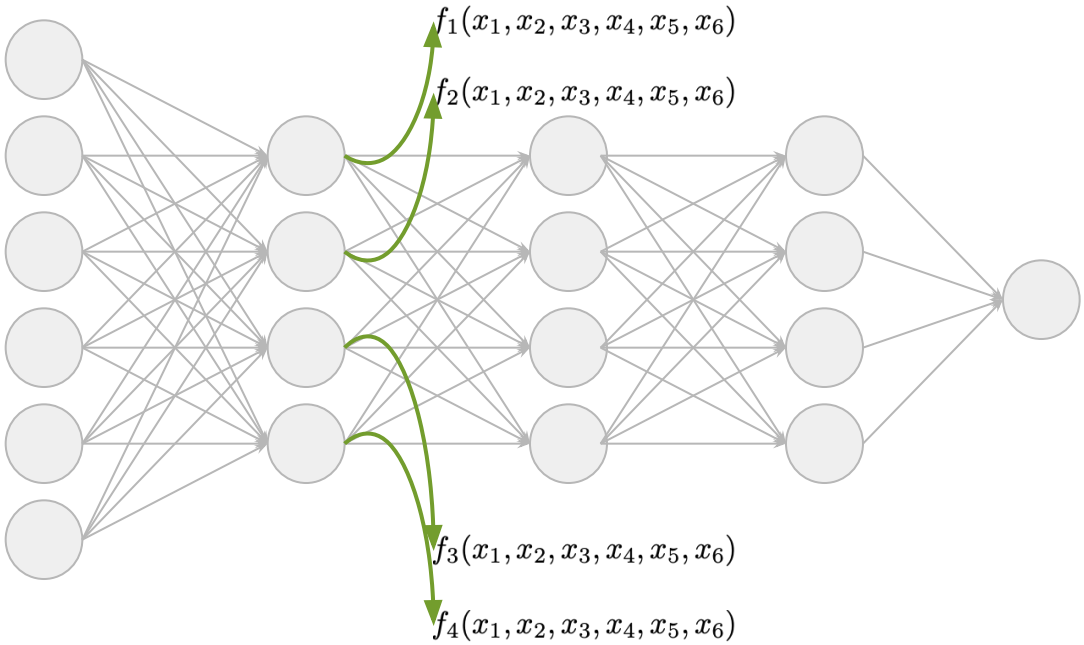
Teorema de Aproximación Universal



Teorema de Aproximación Universal

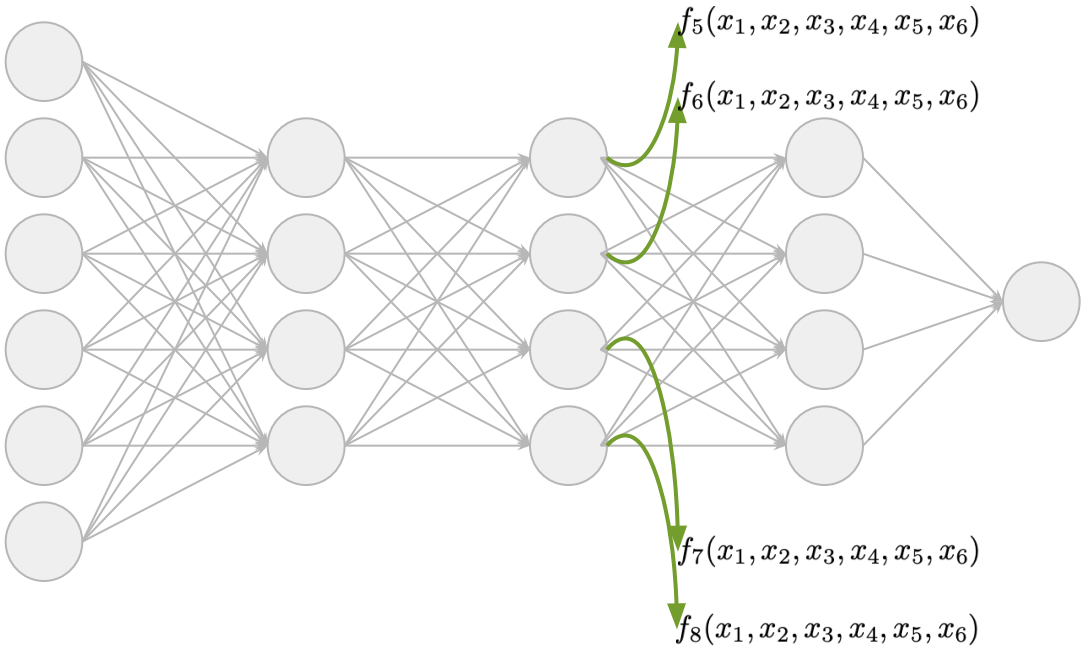


Teorema de Aproximación Universal



En su versión más simple, una red neuronal con una capa oculta aprende representaciones estrictamente lineales.

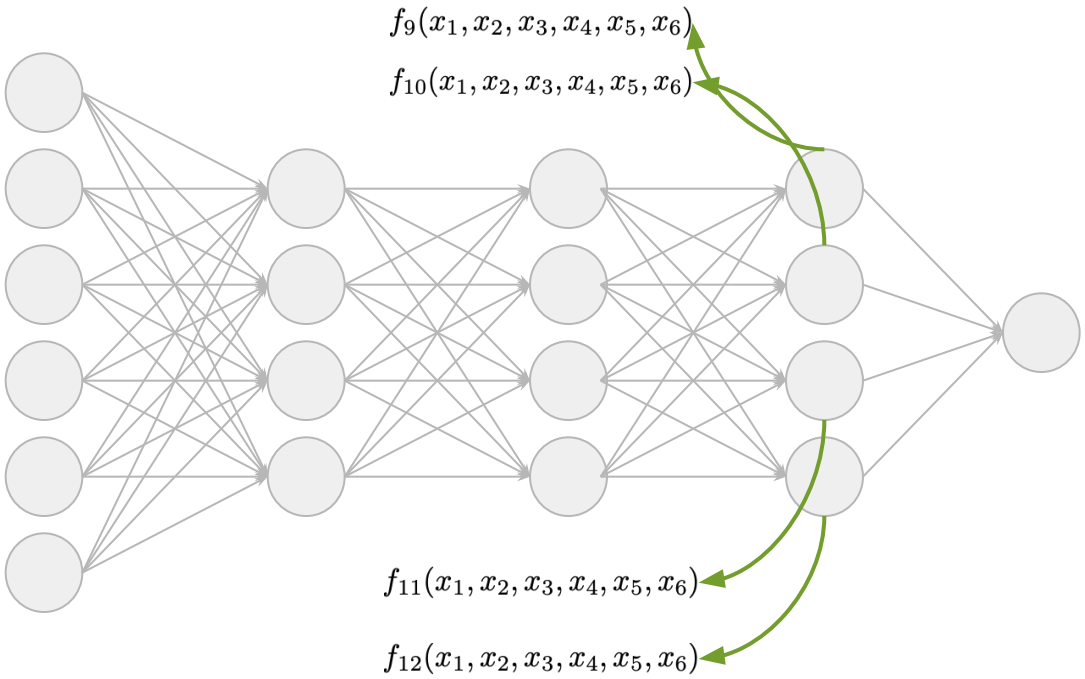
Teorema de Aproximación Universal



En su versión más simple, una red neuronal con una capa oculta aprende representaciones estrictamente lineales.

El teorema de la aproximación universal establece que **independiente de la función que deseamos representar mediante una red neuronal multicapa, una red neuronal con una profundidad substancial permitirá representar esta función.**

Teorema de Aproximación Universal

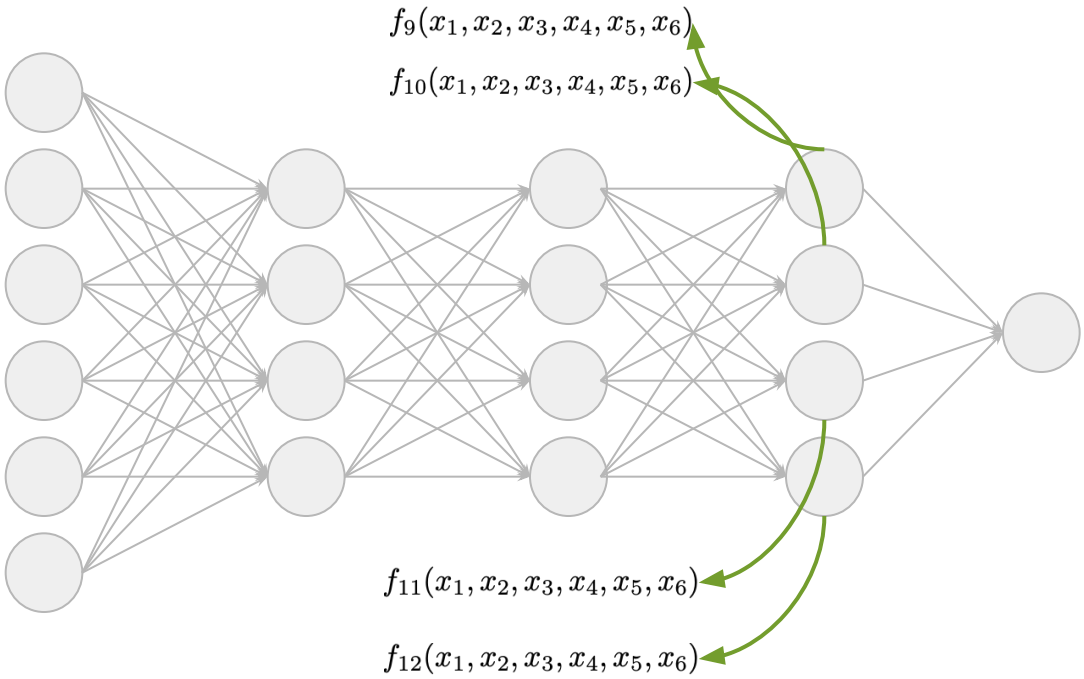


En su versión más simple, una red neuronal con una capa oculta aprende representaciones estrictamente lineales.

El teorema de la aproximación universal establece que **independiente de la función que deseamos representar mediante una red neuronal multicapa, una red neuronal con una profundidad substancial permitirá representar esta función.**

Esto no asegura que la red sea capaz de aprender la función. Esto surge en base a dos elementos:

Teorema de Aproximación Universal

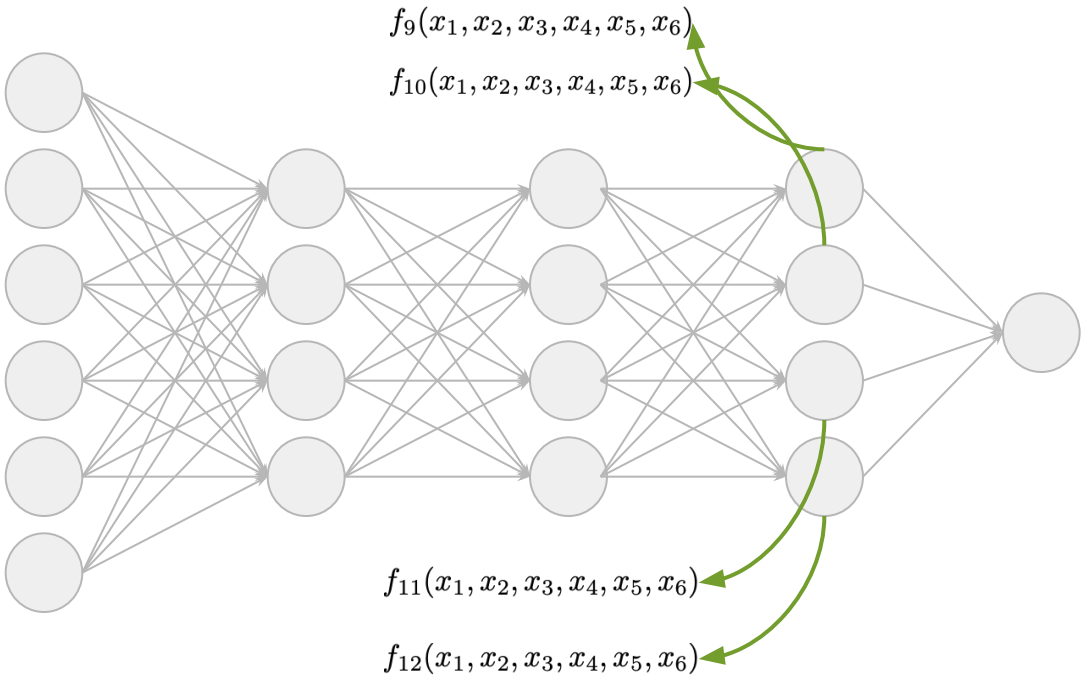


En su versión más simple, una red neuronal con una capa oculta aprende representaciones estrictamente lineales.

El teorema de la aproximación universal establece que **independiente de la función que deseamos representar mediante una red neuronal multicapa, una red neuronal con una profundidad substancial permitirá representar esta función.**

Esto no asegura que la red sea capaz de aprender la función. Esto surge en base a dos elementos:
El **algoritmo de optimización** puede fallar en cuanto a la búsqueda de los parámetros inferidos.

Teorema de Aproximación Universal



En su versión más simple, una red neuronal con una capa oculta aprende representaciones estrictamente lineales.

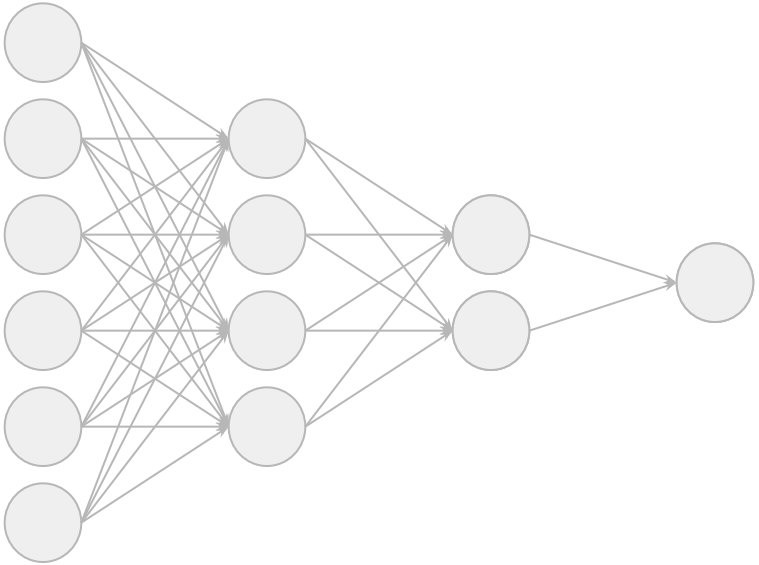
El teorema de la aproximación universal establece que **independiente de la función que deseamos representar mediante una red neuronal multicapa, una red neuronal con una profundidad substancial permitirá representar esta función.**

Esto no asegura que la red sea capaz de aprender la función. Esto surge en base a dos elementos:
El **algoritmo de optimización** puede fallar en cuanto a la búsqueda de los parámetros inferidos.
La función representada puede ser incorrecta dado al **overfitting**.

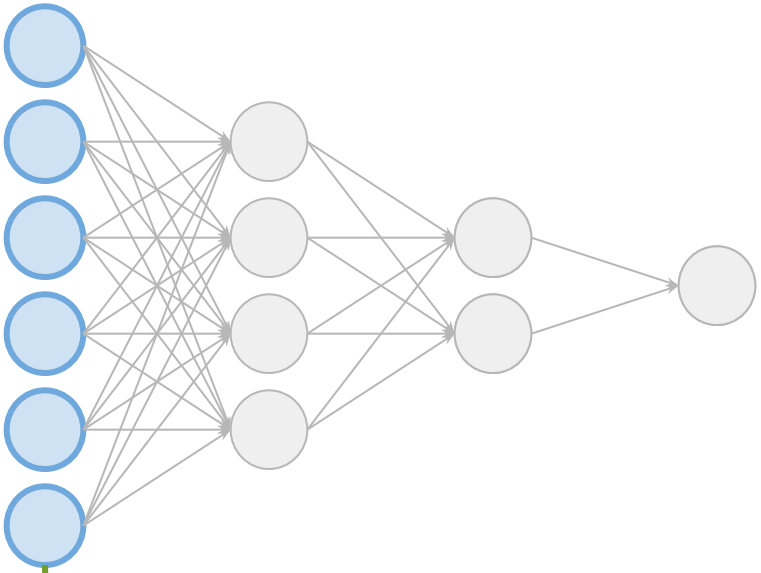
¿Qué es?

- Hasta el momento ignoramos cómo logramos que una red neuronal aprenda los pesos a nivel de capa.
- A grandes rasgos, el proceso de aprendizaje de una red neuronal parte con una selección “aleatoria” de los pesos, que son actualizados posteriormente por un método de retroalimentación.
- Este método contiene dos fases:
    - **Feed-Forward propagation:** Emitimos los impulsos de las neuronas que son captados y procesados por otras, hasta que generamos un output.
    - **Back propagation:** Corregimos los pesos de cada neurona para disminuir la función de pérdida.

Paso Feed Forward

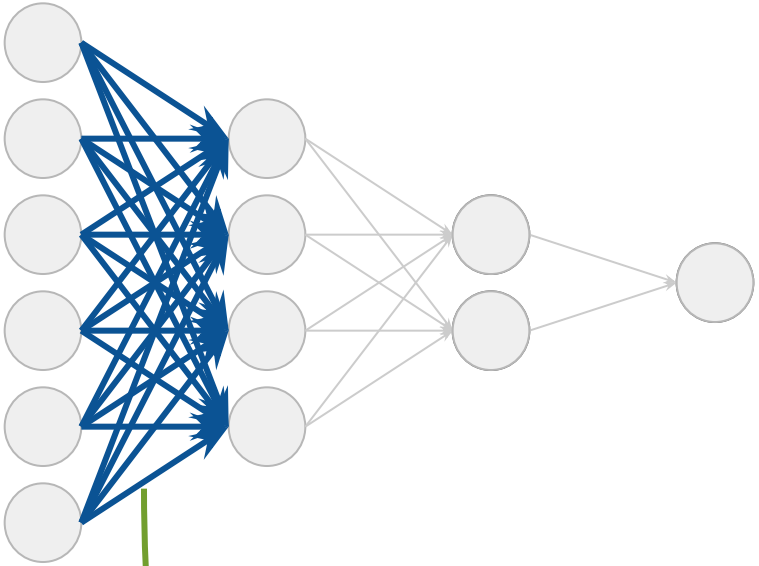


Paso Feed Forward



Input:
Ingresamos un ejemplo (serie de atributos) a nuestra red neuronal.

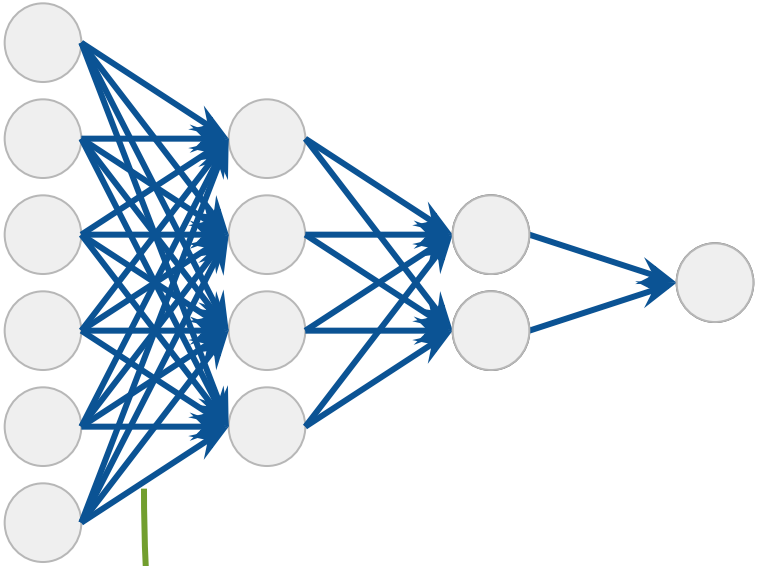
Paso Feed Forward



Pesos:
Los pesos mediante un número pseudoaleatorio que surge de algún **"inicializador de pesos".**

Alternativas de inicialización:
Normal
Uniforme
Truncada
Ortonormal
Glorot
He
LeCunn

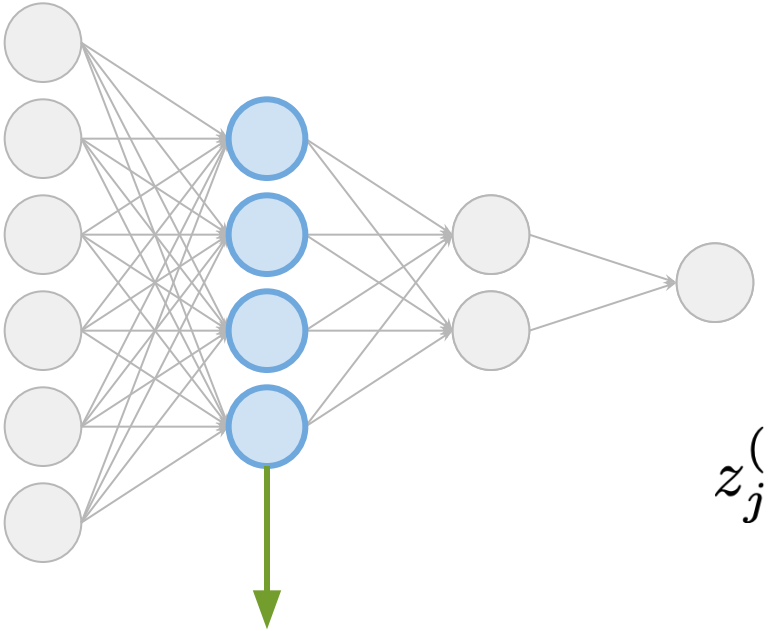
Paso Feed Forward



Pesos:
Los pesos mediante un número pseudoaleatorio que surge de algún **"inicializador de pesos".**

Alternativas de inicialización:
Normal
Uniforme
Truncada
Ortonormal
Glorot
He
LeCunn

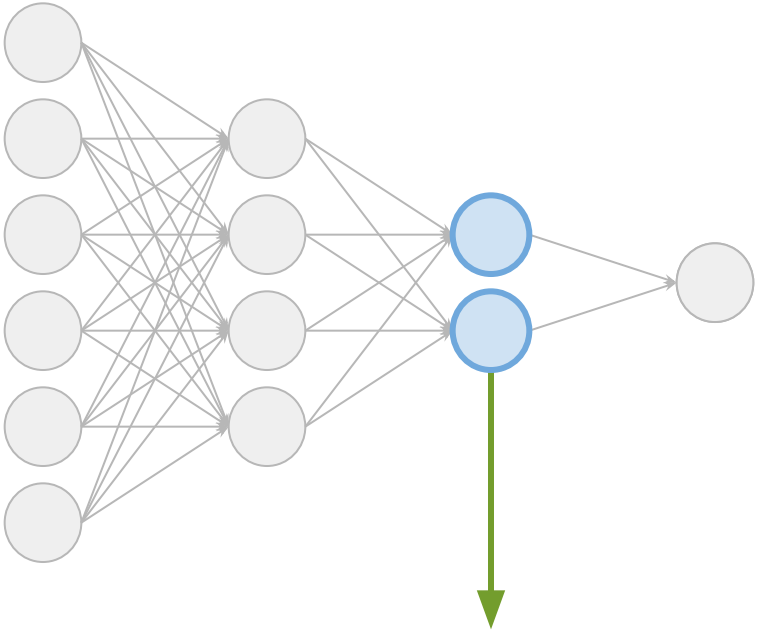
Paso Feed Forward



$$z_j^{(1)} = \sigma(\mathbf{w}_j^{\mathsf{T}} \cdot \mathbf{x})$$

Primera Capa oculta:
Genera una reexpresión de los pesos en base a una función de activación y un sesgo.

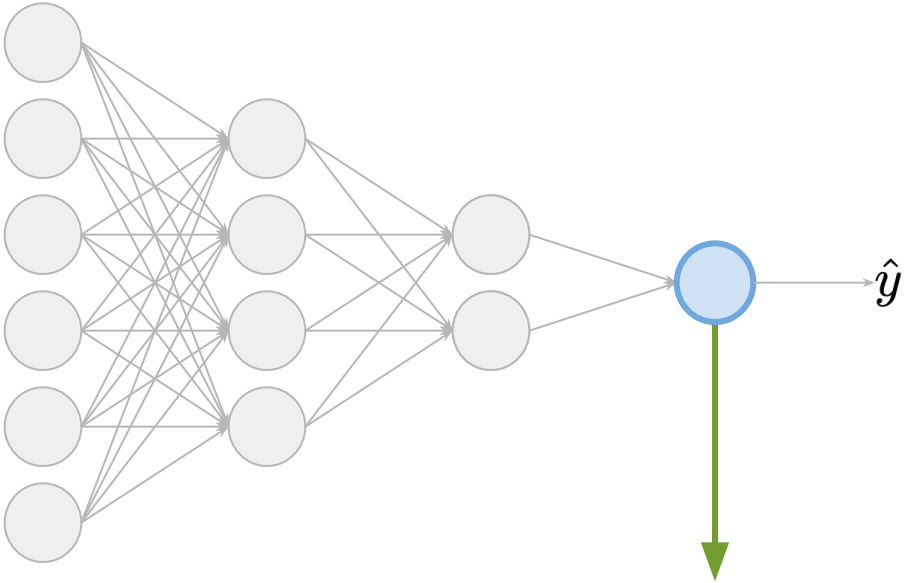
Paso Feed Forward



Segunda Capa oculta:
Genera una reexpresión de los pesos en base a una función de activación y un sesgo.

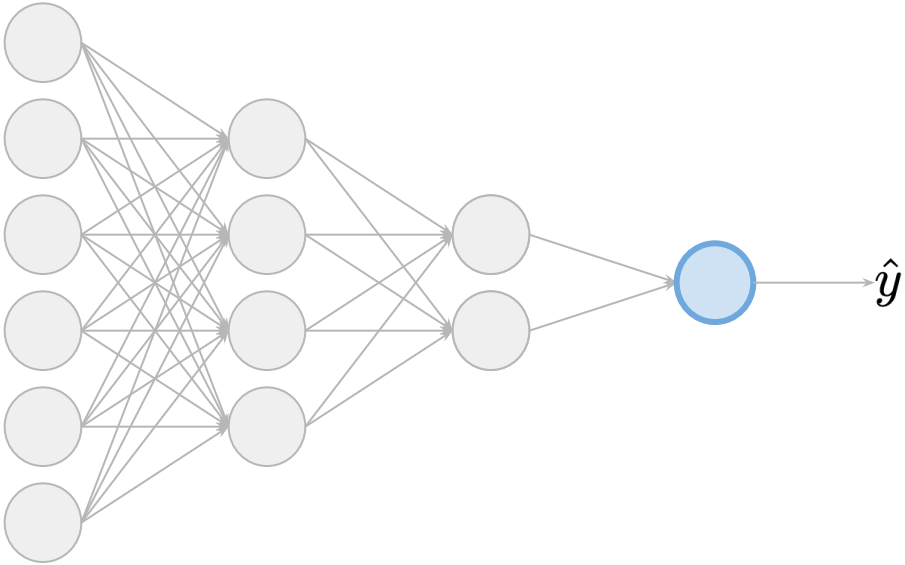
$$z_j^{(2)} = \sigma(\mathbf{w}_j^{\mathsf{T}} \cdot \mathbf{x})$$

Paso Feed Forward

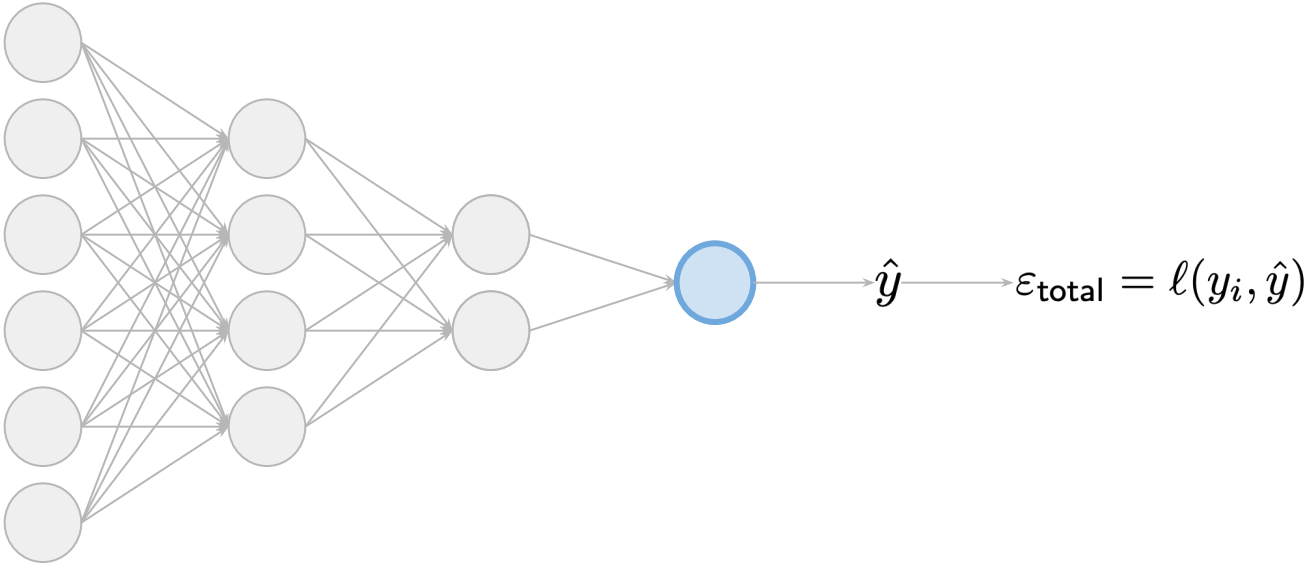


Capa de salida:
Expresa los impulsos de las capas previas.

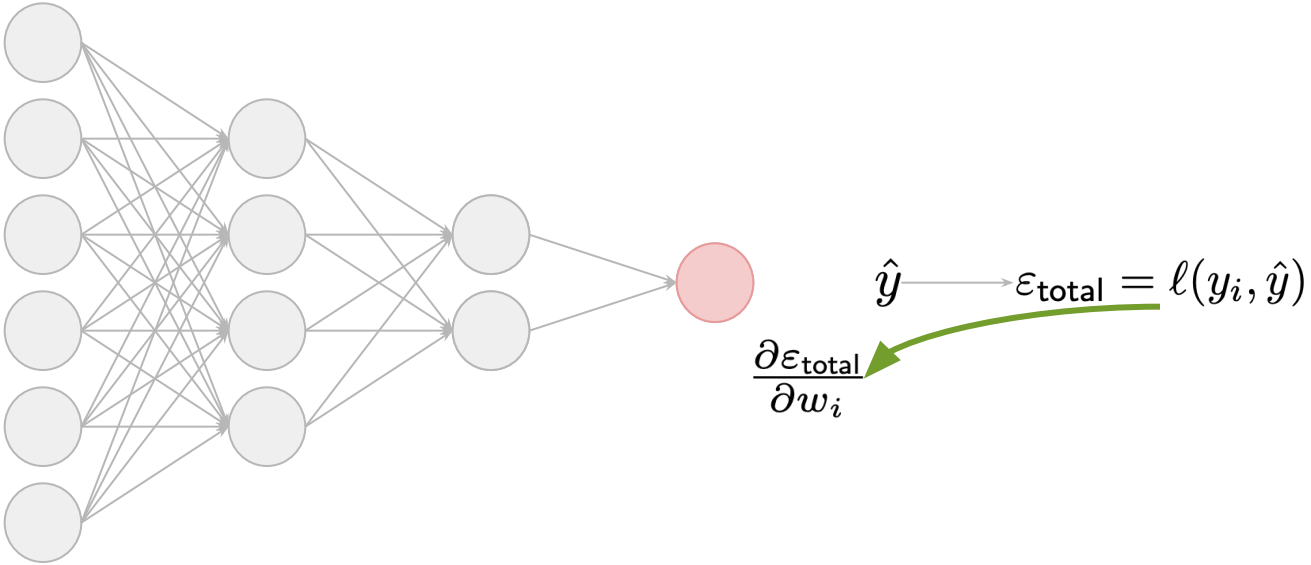
Paso Backpropagation



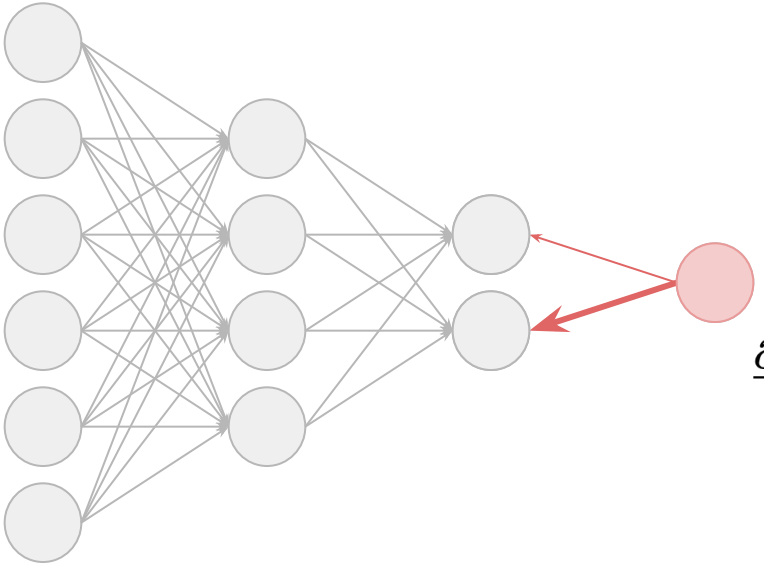
Paso Backpropagation



Paso Backpropagation



Paso Backpropagation

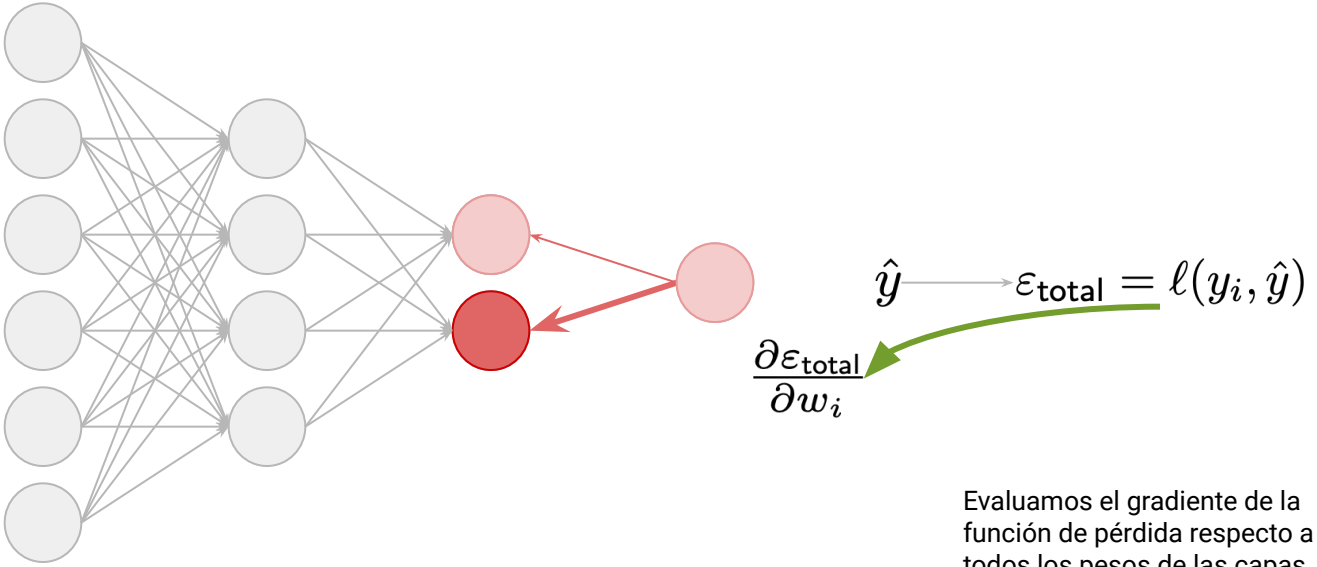


$$\hat{y} \longrightarrow \varepsilon_{\text{total}} = \ell(y_i, \hat{y})$$

$$\frac{\partial \varepsilon_{\text{total}}}{\partial w_i}$$

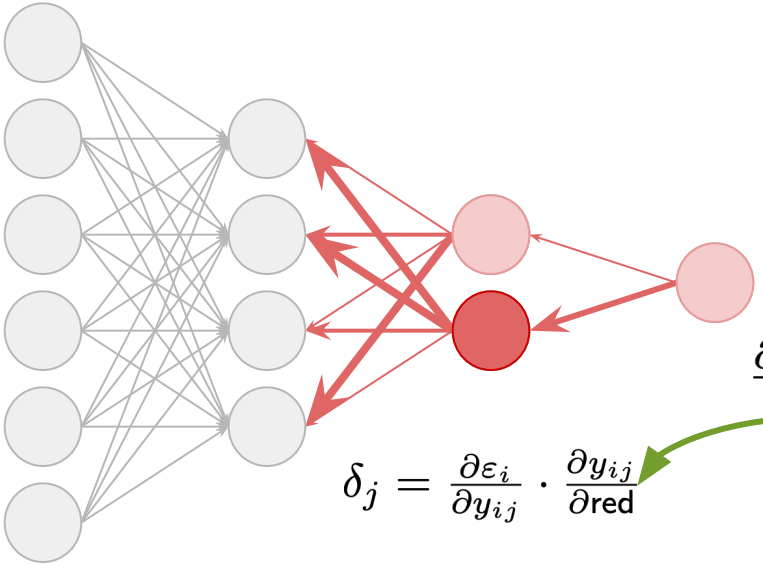
Evaluamos el gradiente de la función de pérdida respecto a todos los pesos de las capas.

Paso Backpropagation



Evaluamos el gradiente de la función de pérdida respecto a todos los pesos de las capas.

Paso Backpropagation



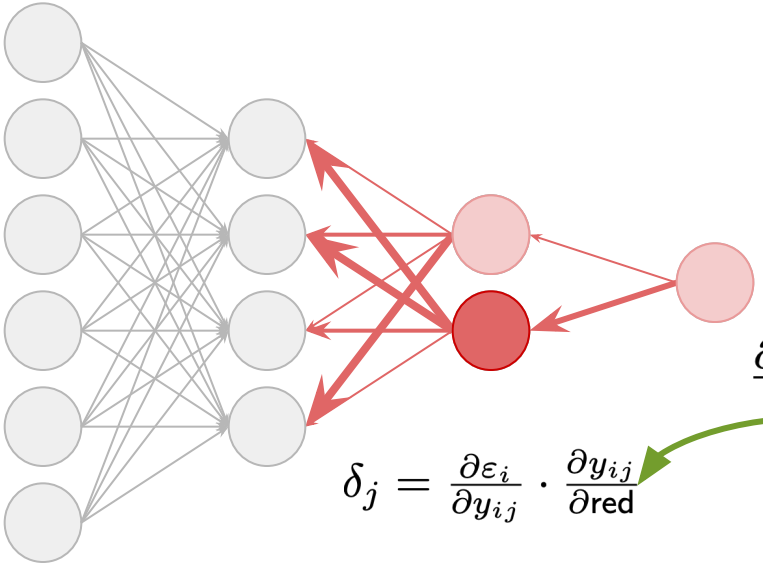
$$\hat{y} \longrightarrow \varepsilon_{\text{total}} = \ell(y_i, \hat{y})$$

$$\frac{\partial \varepsilon_{\text{total}}}{\partial w_i}$$

$$\delta_j = \frac{\partial \varepsilon_i}{\partial y_{ij}} \cdot \frac{\partial y_{ij}}{\partial \text{red}}$$

Evaluamos el gradiente de la función de pérdida respecto a todos los pesos de las capas.

Paso Backpropagation



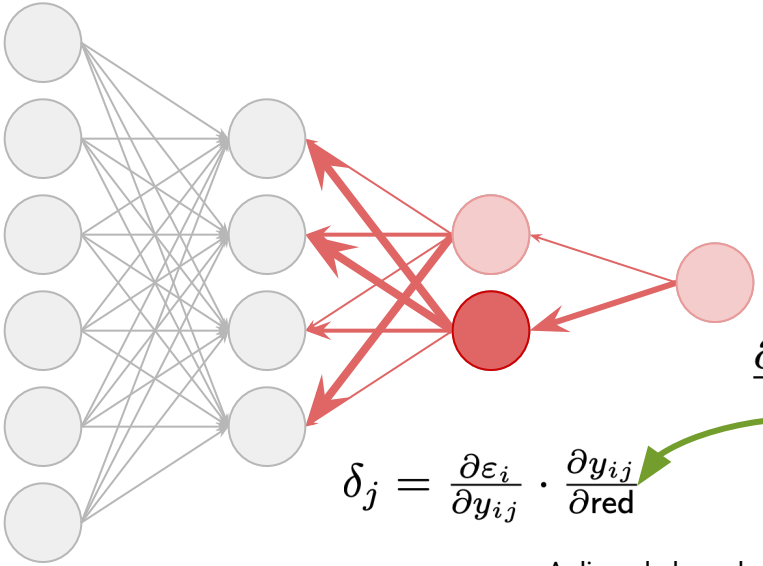
$$\hat{y} \longrightarrow \varepsilon_{\text{total}} = \ell(y_i, \hat{y})$$

$$\frac{\partial \varepsilon_{\text{total}}}{\partial w_i}$$

$$\delta_j = \frac{\partial \varepsilon_i}{\partial y_{ij}} \cdot \frac{\partial y_{ij}}{\partial \text{red}}$$

Evaluamos el gradiente de la función de pérdida respecto a todos los pesos de las capas.

Paso Backpropagation



$$\hat{y} \longrightarrow \varepsilon_{\text{total}} = \ell(y_i, \hat{y})$$

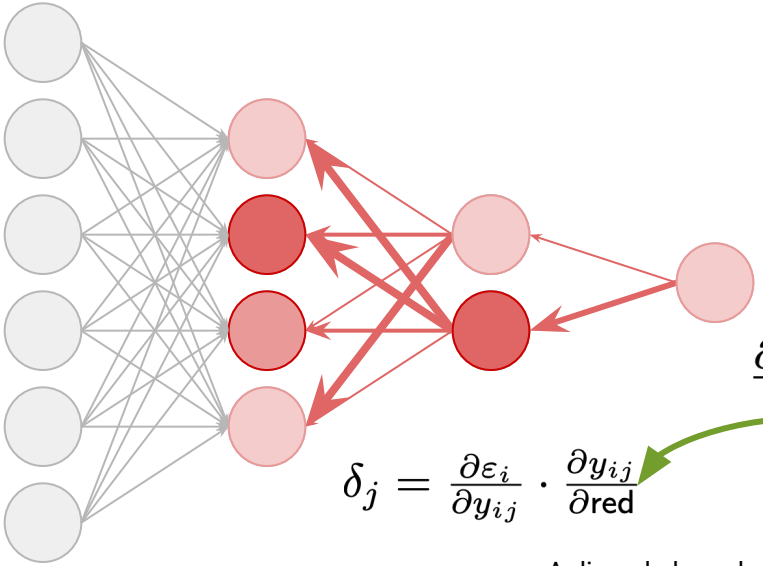
$$\frac{\partial \varepsilon_{\text{total}}}{\partial w_i}$$

$$\delta_j = \frac{\partial \varepsilon_i}{\partial y_{ij}} \cdot \frac{\partial y_{ij}}{\partial \text{red}}$$

Aplicando la regla de la cadena, obtenemos un actualizador de pesos que penalize de mayor manera aquellos pesos responsables del aumento de la pérdida.

Evaluamos el gradiente de la función de pérdida respecto a todos los pesos de las capas.

Paso Backpropagation



$$\hat{y} \longrightarrow \varepsilon_{\text{total}} = \ell(y_i, \hat{y})$$

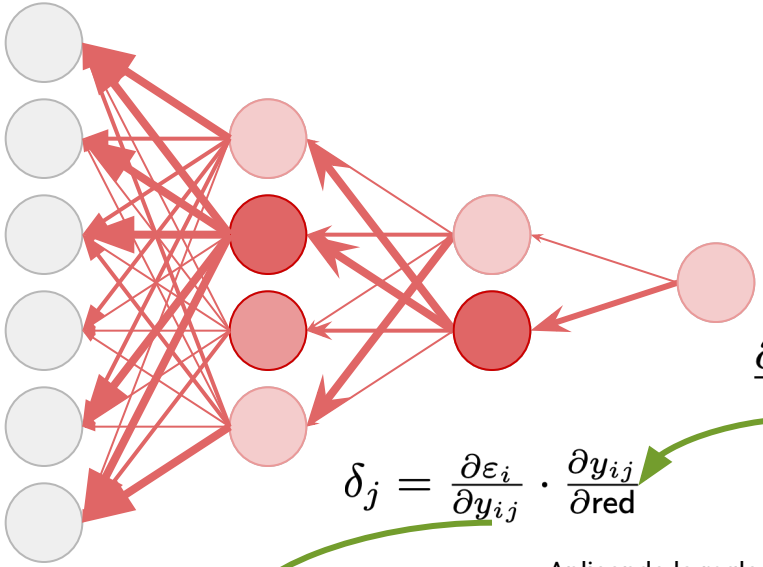
$$\frac{\partial \varepsilon_{\text{total}}}{\partial w_i}$$

$$\delta_j = \frac{\partial \varepsilon_i}{\partial y_{ij}} \cdot \frac{\partial y_{ij}}{\partial \text{red}}$$

Aplicando la regla de la cadena, obtenemos un actualizador de pesos que penalize de mayor manera aquellos pesos responsables del aumento de la pérdida.

Evaluamos el gradiente de la función de pérdida respecto a todos los pesos de las capas.

Paso Backpropagation



$$\hat{y} \longrightarrow \varepsilon_{\text{total}} = \ell(y_i, \hat{y})$$

$$\frac{\partial \varepsilon_{\text{total}}}{\partial w_i}$$

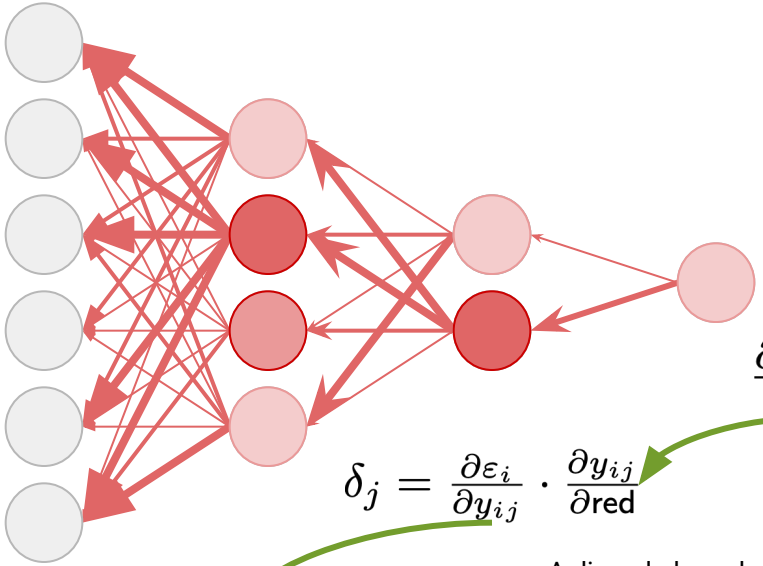
$$\delta_j = \frac{\partial \varepsilon_i}{\partial y_{ij}} \cdot \frac{\partial y_{ij}}{\partial \text{red}}$$

$$\Delta w_{ji} = -\alpha \delta_j y_i$$

Evaluamos el gradiente de la función de pérdida respecto a todos los pesos de las capas.

Aplicando la regla de la cadena, obtenemos un actualizador de pesos que penalize de mayor manera aquellos pesos responsables del aumento de la pérdida.

Paso Backpropagation



$$\hat{y} \longrightarrow \varepsilon_{\text{total}} = \ell(y_i, \hat{y})$$

$$\frac{\partial \varepsilon_{\text{total}}}{\partial w_i}$$

$$\delta_j = \frac{\partial \varepsilon_i}{\partial y_{ij}} \cdot \frac{\partial y_{ij}}{\partial \text{red}}$$

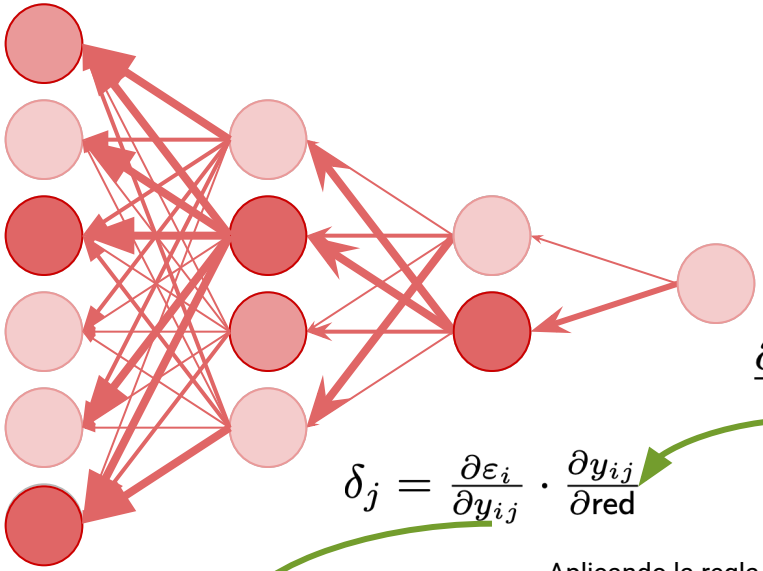
$$\Delta w_{ji} = -\alpha \delta_j y_i$$

Evaluamos el gradiente de la función de pérdida respecto a todos los pesos de las capas.

Aplicando la regla de la cadena, obtenemos un actualizador de pesos que penalize de mayor manera aquellos pesos responsables del aumento de la pérdida.

En la medida que los errores se propaguen, actualizaremos los pesos por este penalizador según el Learning Rate implementado.

Paso Backpropagation



$$\hat{y} \longrightarrow \varepsilon_{\text{total}} = \ell(y_i, \hat{y})$$

$$\frac{\partial \varepsilon_{\text{total}}}{\partial w_i}$$

$$\delta_j = \frac{\partial \varepsilon_i}{\partial y_{ij}} \cdot \frac{\partial y_{ij}}{\partial \text{red}}$$

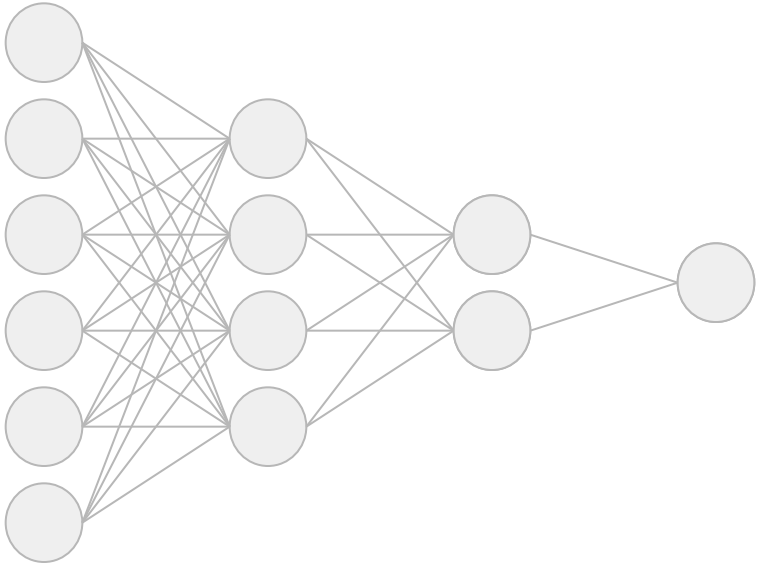
$$\Delta w_{ji} = -\alpha \delta_j y_i$$

Evaluamos el gradiente de la función de pérdida respecto a todos los pesos de las capas.

Aplicando la regla de la cadena, obtenemos un actualizador de pesos que penalize de mayor manera aquellos pesos responsables del aumento de la pérdida.

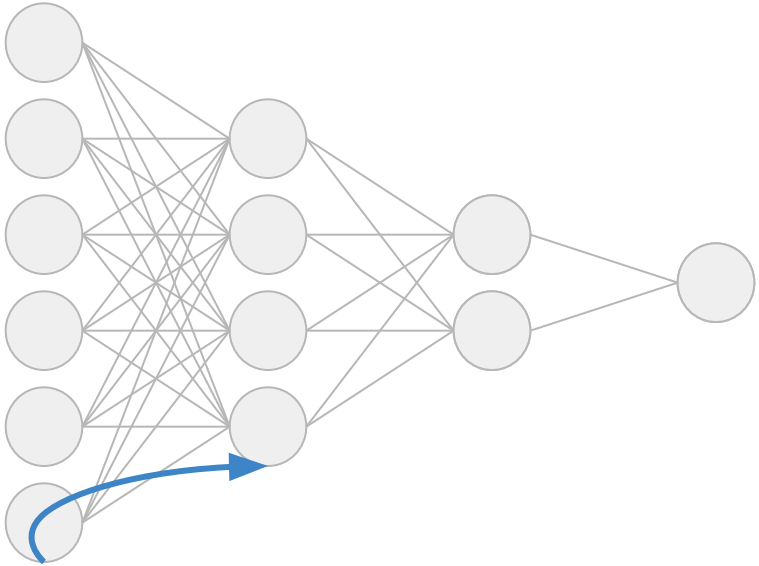
En la medida que los errores se propaguen, actualizaremos los pesos por este penalizador según el Learning Rate implementado.

Épocas (Ciclos) de Entrenamiento



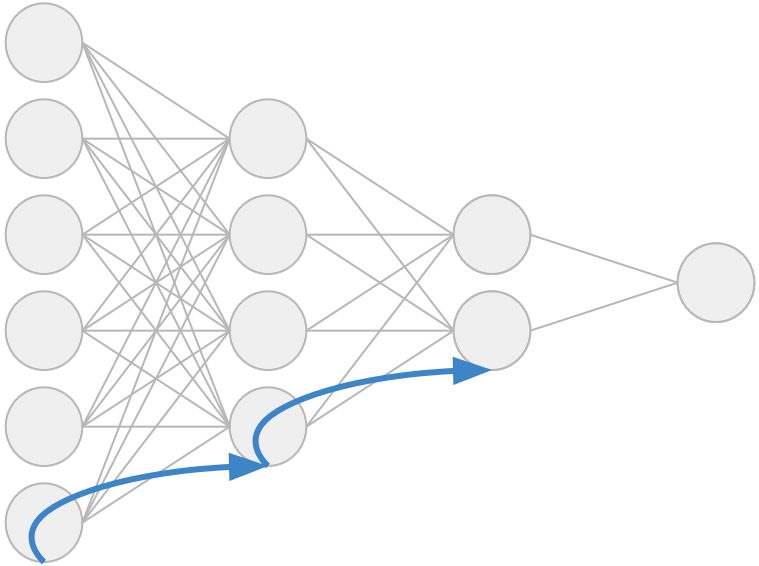
{desafío}
latam_

Épocas (Ciclos) de Entrenamiento



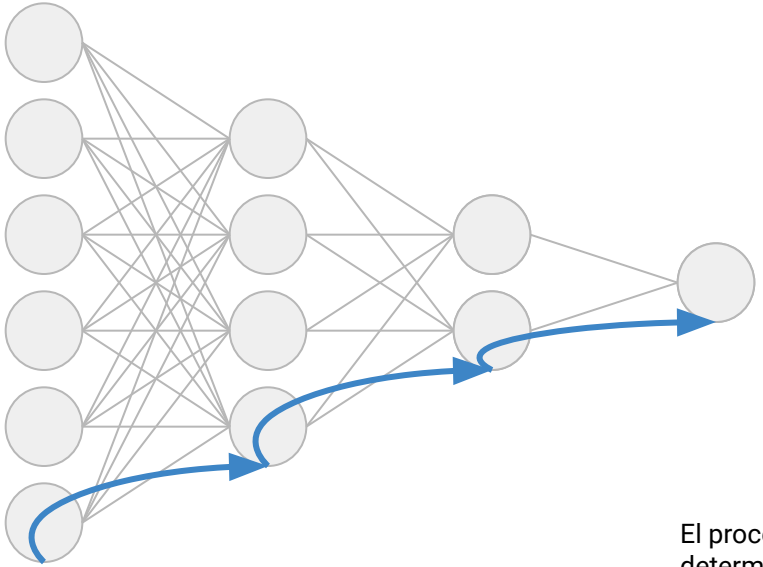
{desafío}
latam_

Épocas (Ciclos) de Entrenamiento



{desafío}
latam_

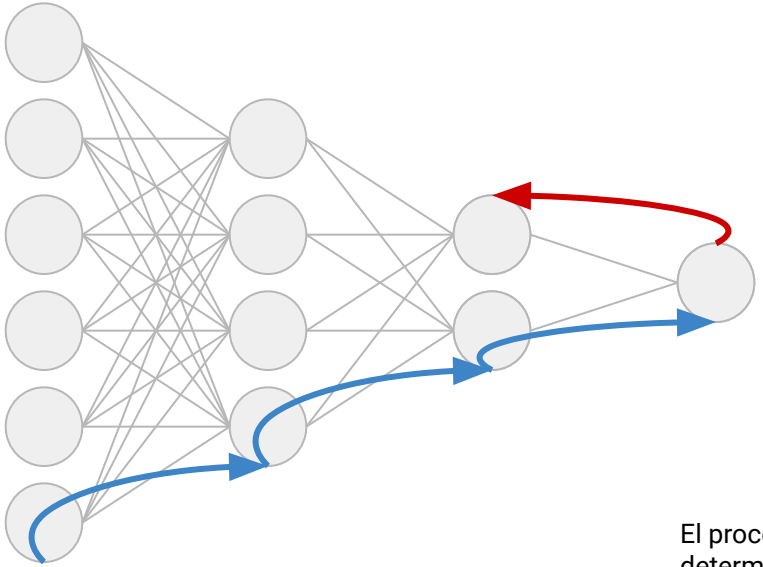
Épocas (Ciclos) de Entrenamiento



El proceso de predecir un output para un ejemplo determinado es el paso **Feed Forward.**

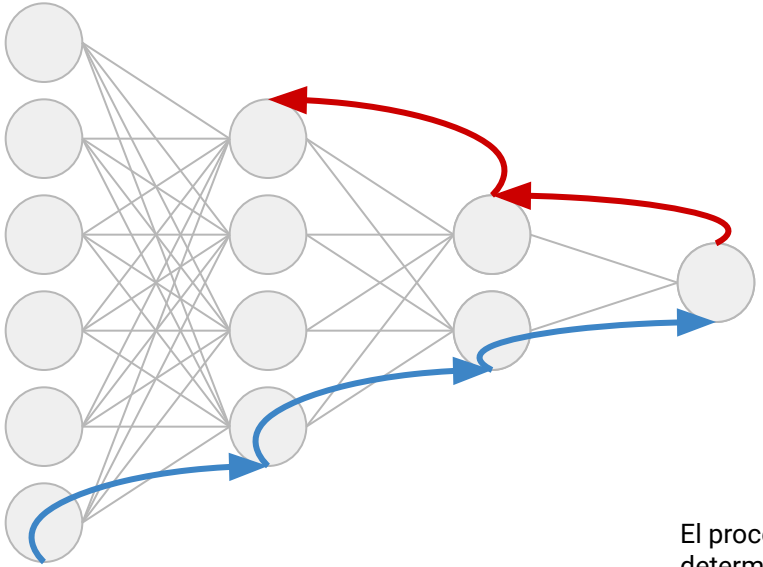
{desafío} latam_

Épocas (Ciclos) de Entrenamiento



El proceso de predecir un output para un ejemplo determinado es el paso **Feed Forward.**

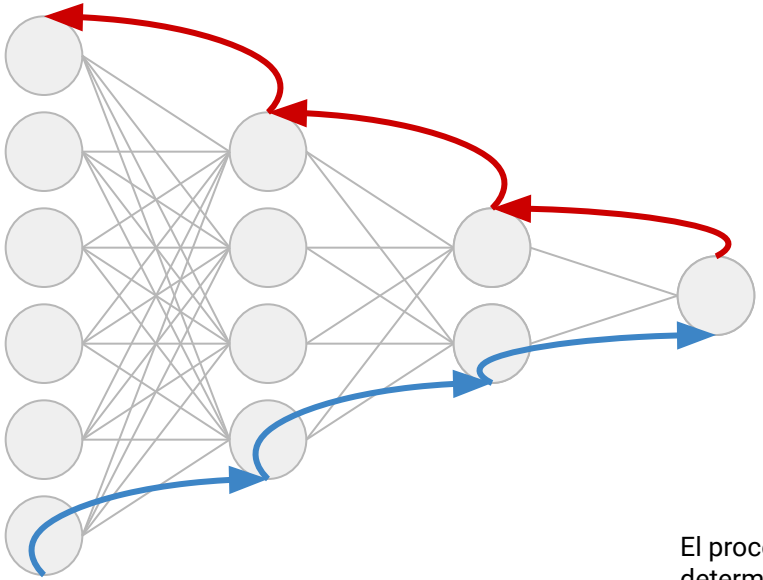
Épocas (Ciclos) de Entrenamiento



El proceso de predecir un output para un ejemplo determinado es el paso **Feed Forward.**

{desafío} latam_

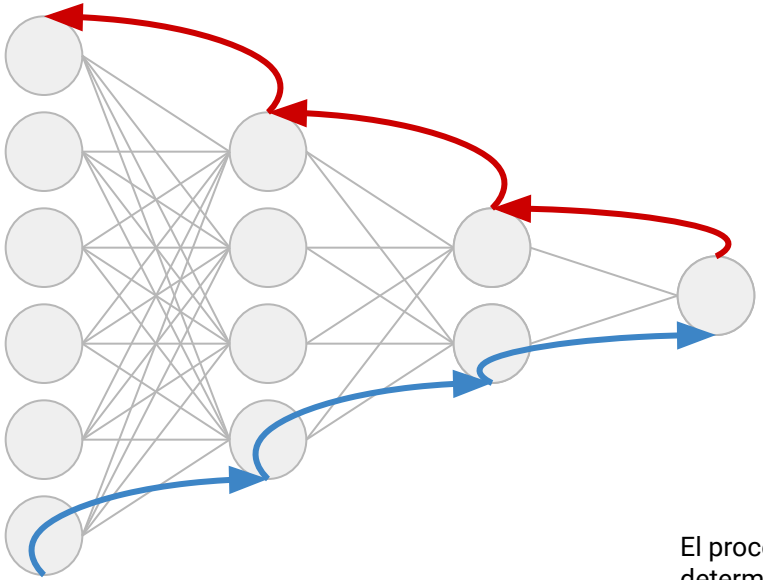
Épocas (Ciclos) de Entrenamiento



El proceso de predecir un output para un ejemplo determinado es el paso **Feed Forward.**

Épocas (Ciclos) de Entrenamiento

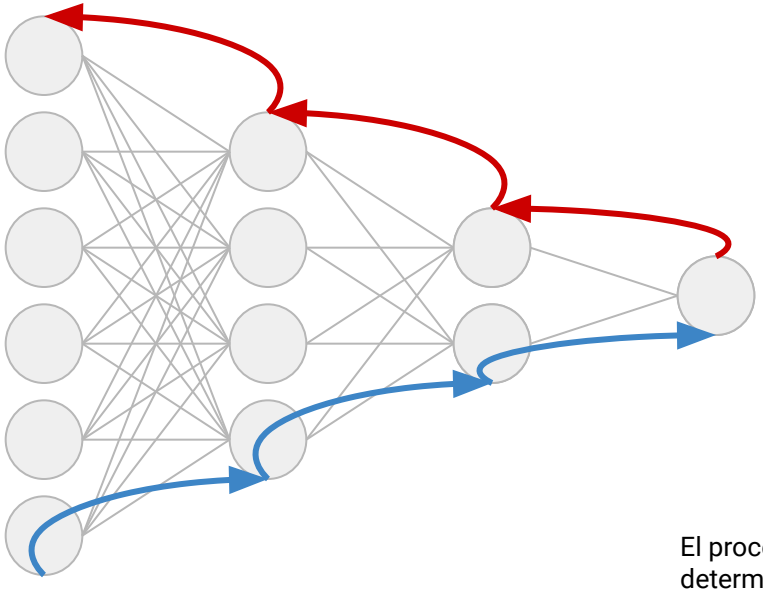
El proceso de actualizar los pesos en función a su contribución al error se conoce como **Back Propagation**.



El proceso de predecir un output para un ejemplo determinado es el paso **Feed Forward.**

Épocas (Ciclos) de Entrenamiento

El proceso de actualizar los pesos en función a su contribución al error se conoce como **Back Propagation**.

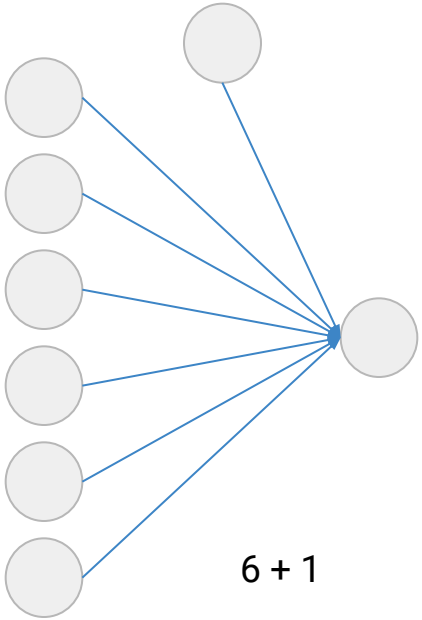


El flujo entre paso 1 a paso 2 se conoce como una **época** de entrenamiento.

El proceso de predecir un output para un ejemplo determinado es el paso **Feed Forward.**

Redes Neuronales Multicapas

Redes Neuronales Multicapas



6 + 1

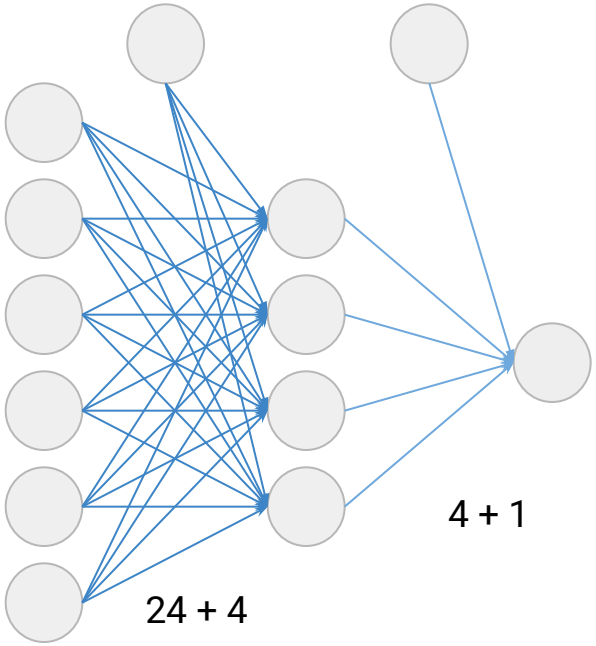
Punto de partida: **Single Layer Perceptron.**

Cada conexión va a representar un peso.

La cantidad de parámetros se formula en base a la cantidad de atributos en la capa de entrada, más un sesgo existente a nivel de capa de salida.

Cantidad de parámetros estimables: 7

Redes Neuronales Multicapas

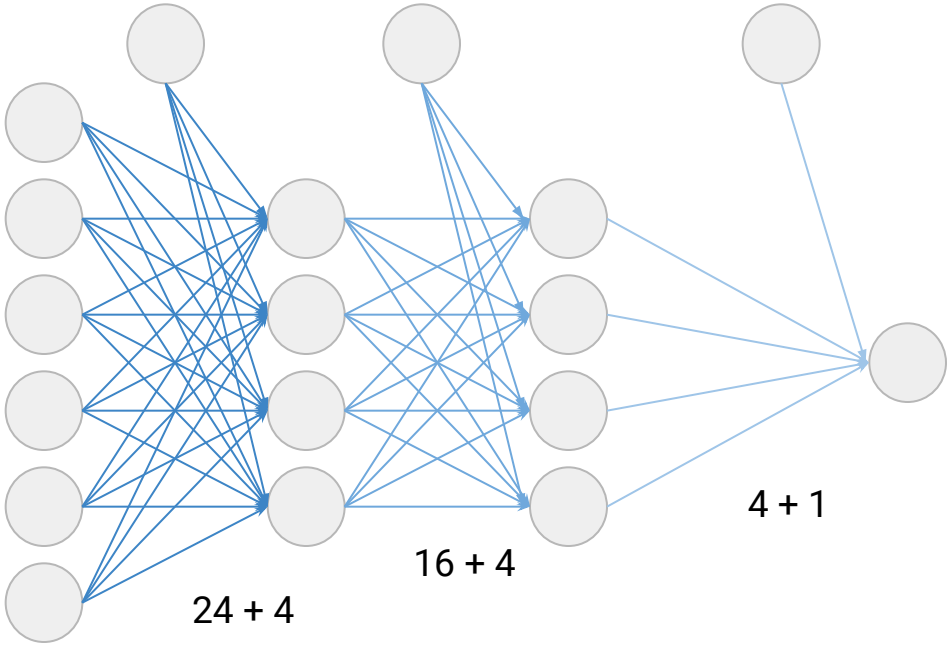


Siguiente Iteración: **Perceptrón Multicapas.**

Primera configuración con una capa intermedia. En esta generamos una representación latente de nuestra función a aproximar.

Cantidad de parámetros estimables: 33

Redes Neuronales Multicapas

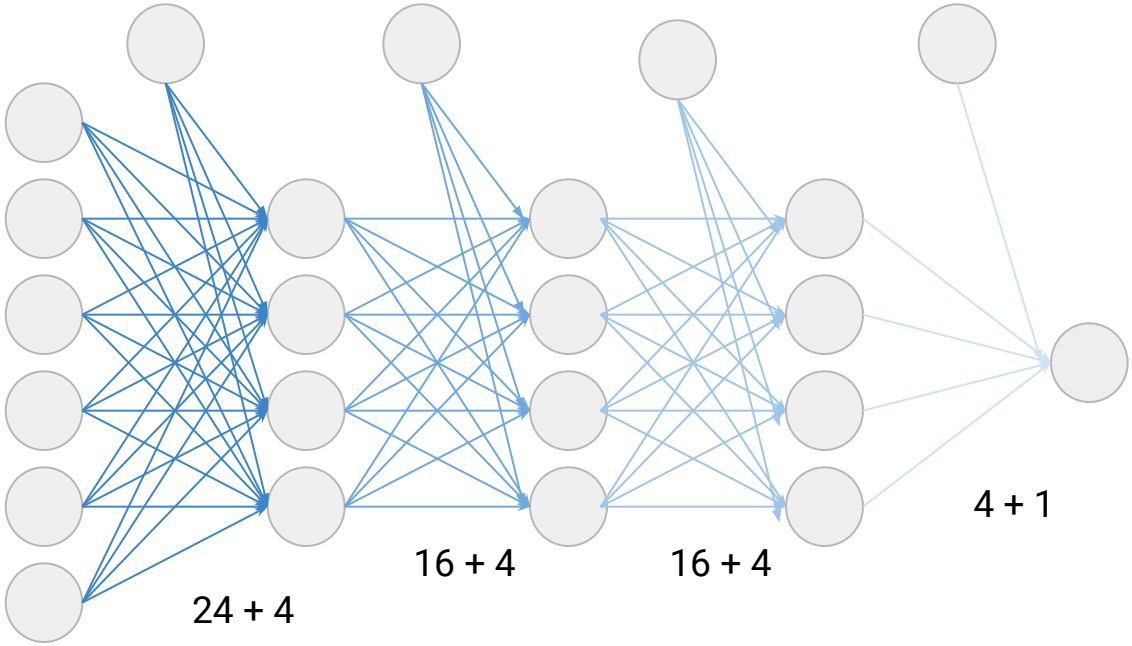


Siguiente Iteración: **Perceptrón Multicapas.**

En la medida que agregamos más capas con una cantidad definida de neuronas, aumentamos la cantidad de representaciones distribuídas en cuanto al fenómeno.

Cantidad de parámetros estimables: 53

Redes Neuronales Multicapas



Siguiente Iteración: **¿"Deep Learning"?**

Idea seminal: Generar un máximo de representaciones distribuídas en base al teorema del aproximador universal.

Cantidad de parámetros estimables: 73

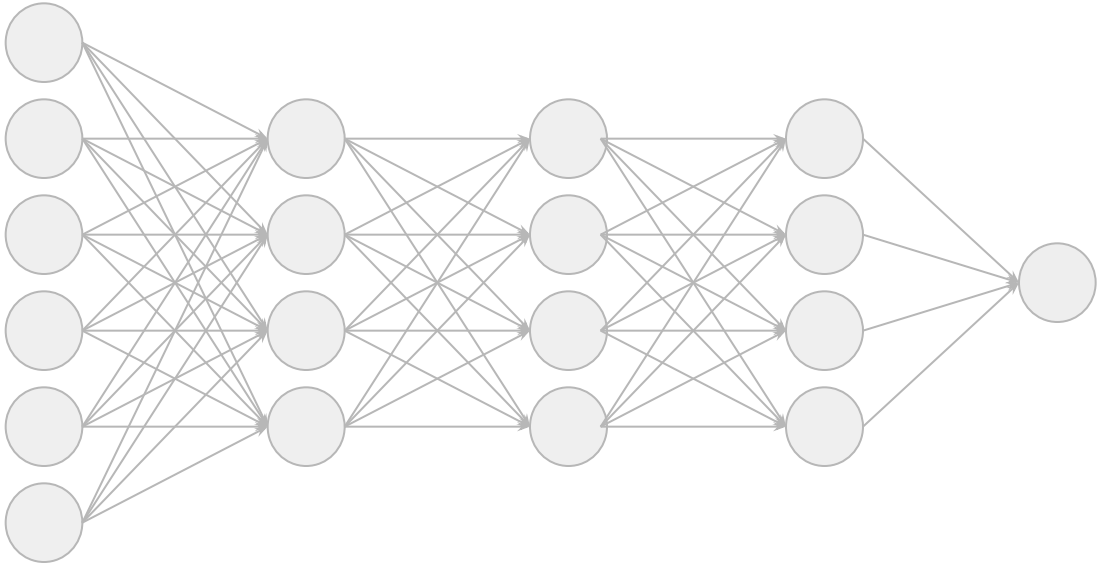
Regularización para Redes Neuronales

{desafío} latam_

Objetivo

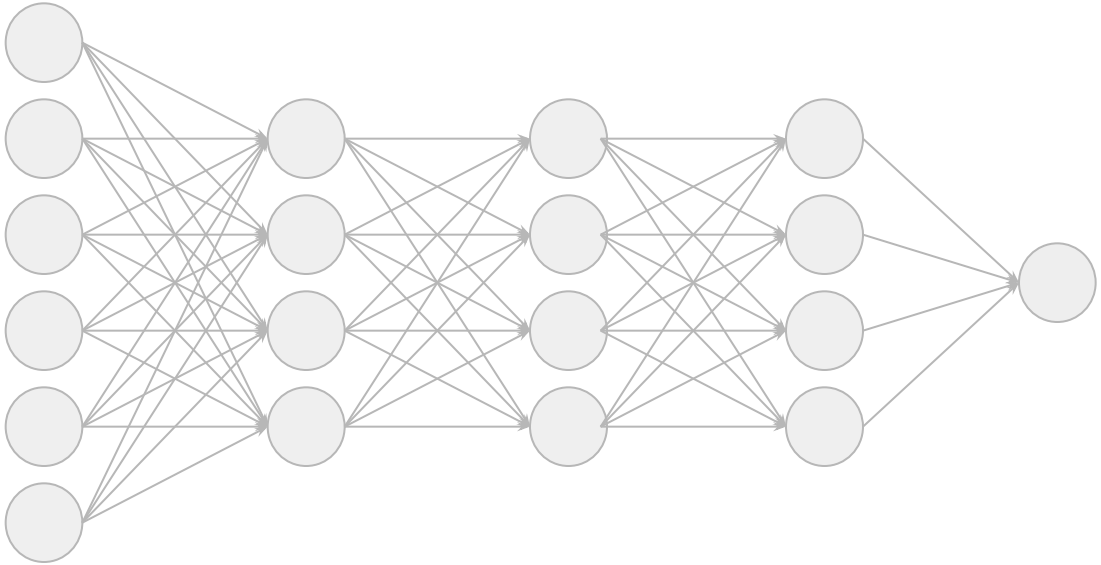
- Problema recurrente con redes neuronales “profundas”: su tendencia a generar soluciones proclives al overfitting.
- Existen dos formas de regularizar el overfitting:
  - Mediante regularización paramétrica.
  - Mediante dropout (cancelación de pesos en las capas ocultas).

Regularización Paramétrica



Punto de partida: cada peso procesado como impulso es una combinación lineal de parámetros.

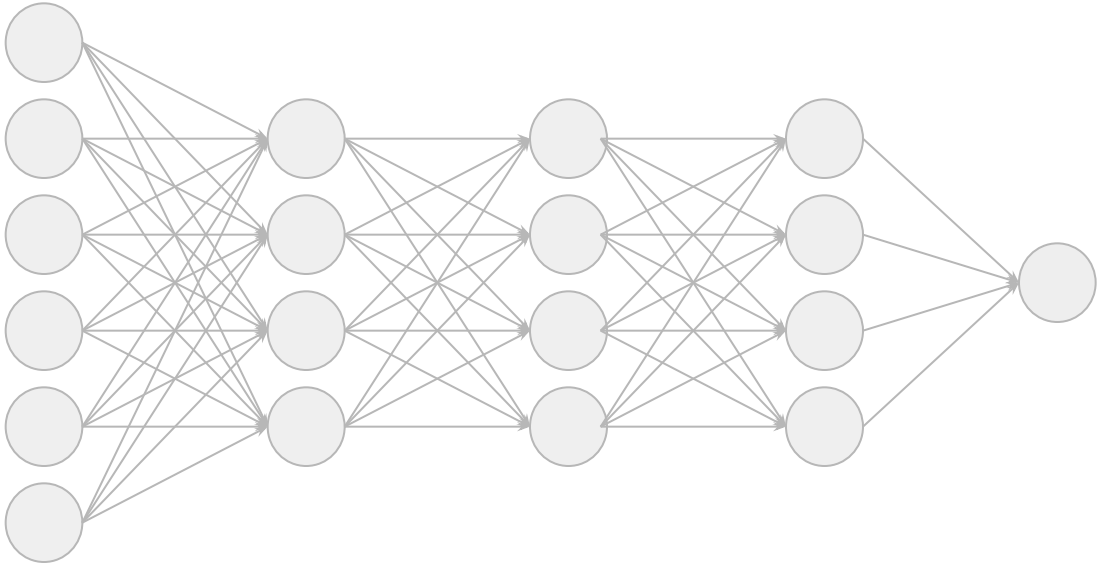
Regularización Paramétrica



Punto de partida: cada peso procesado como impulso es una combinación lineal de parámetros.

Teniendo esto en consideración, podemos implementar técnicas de regularización paramétrica como Ridge, Lasso y Elastic Net.

Regularización Paramétrica

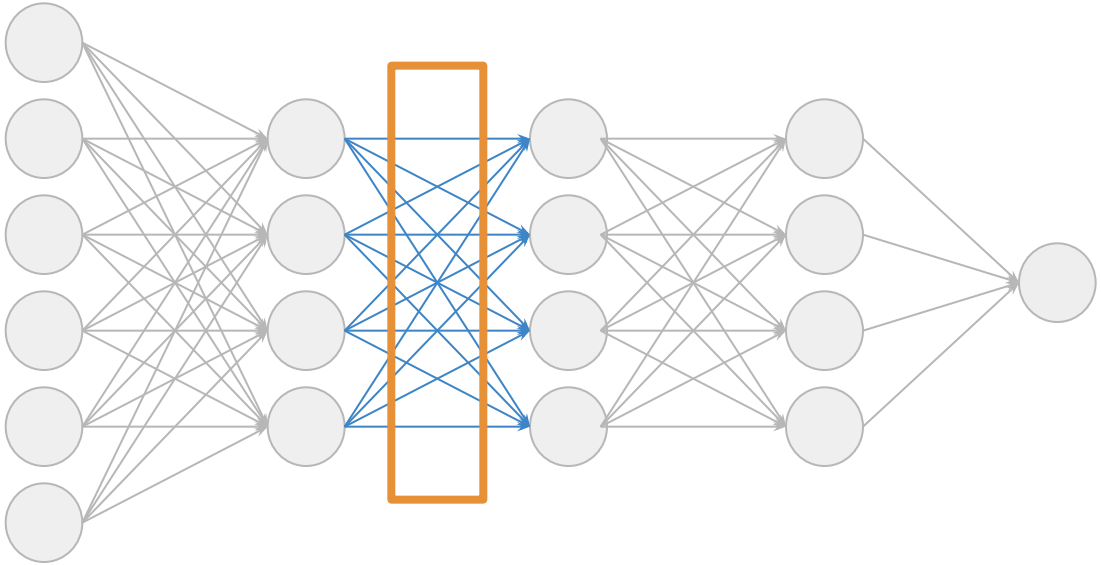


Punto de partida: cada peso procesado como impulso es una combinación lineal de parámetros.

Teniendo esto en consideración, podemos implementar técnicas de regularización paramétrica como Ridge, Lasso y Elastic Net.

La implementación de la regularización se realiza en consideración a la **función de pérdida** definida en nuestra red neuronal.

Regularización - Norma L1



Punto de partida: cada peso procesado como impulso es una combinación lineal de parámetros.

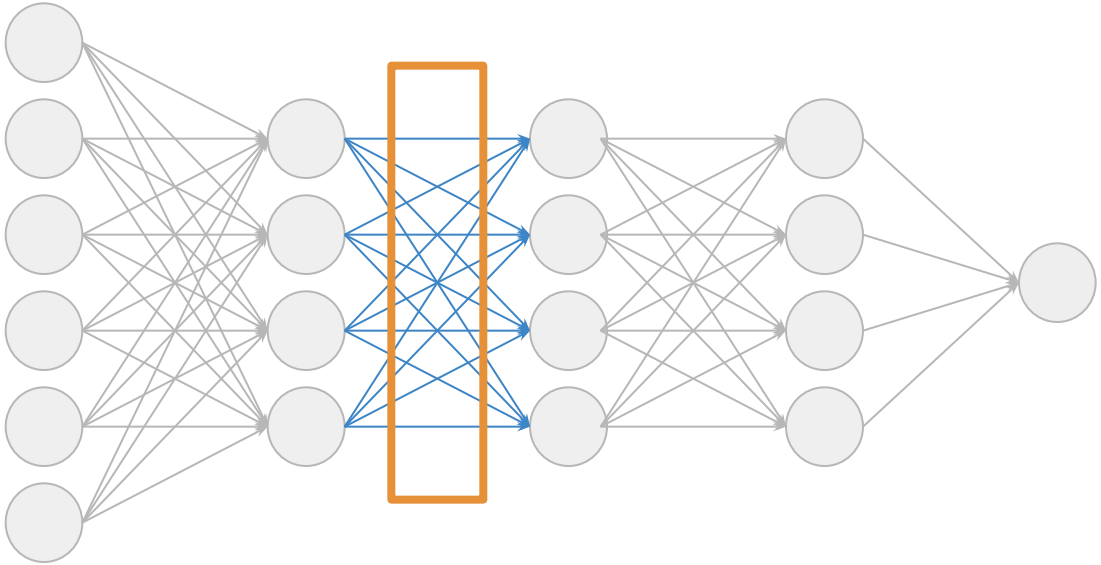
Teniendo esto en consideración, podemos implementar técnicas de regularización paramétrica como Ridge, Lasso y Elastic Net.

La implementación de la regularización se realiza en consideración a la **función de pérdida** definida en nuestra red neuronal.

$$w^{(1)} + \frac{\lambda}{2m} \sum_{w}^{m} \|w\|$$

Norma L1: Genera penalización de coeficientes mediante la norma absoluta. Tiende a entregar soluciones más agresivas, eliminando pesos irrelevantes.

Regularización - Norma L2



Punto de partida: cada peso procesado como impulso es una combinación lineal de parámetros.

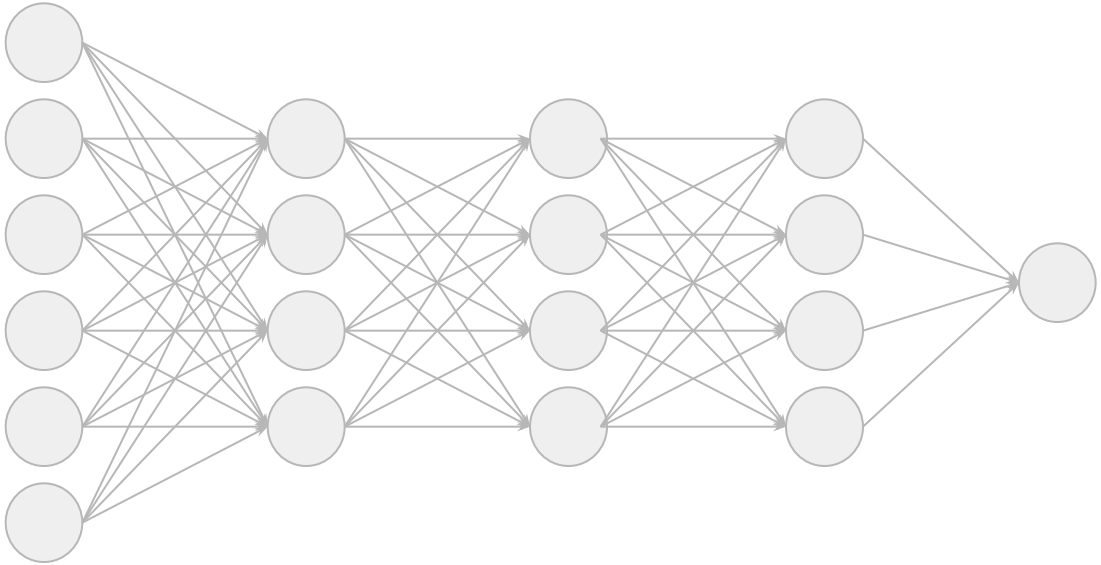
Teniendo esto en consideración, podemos implementar técnicas de regularización paramétrica como Ridge, Lasso y Elastic Net.

La implementación de la regularización se realiza en consideración a la **función de pérdida** definida en nuestra red neuronal.

$$w^{(1)} + \frac{\lambda}{2m} \sum_{w}^{m} w^2$$

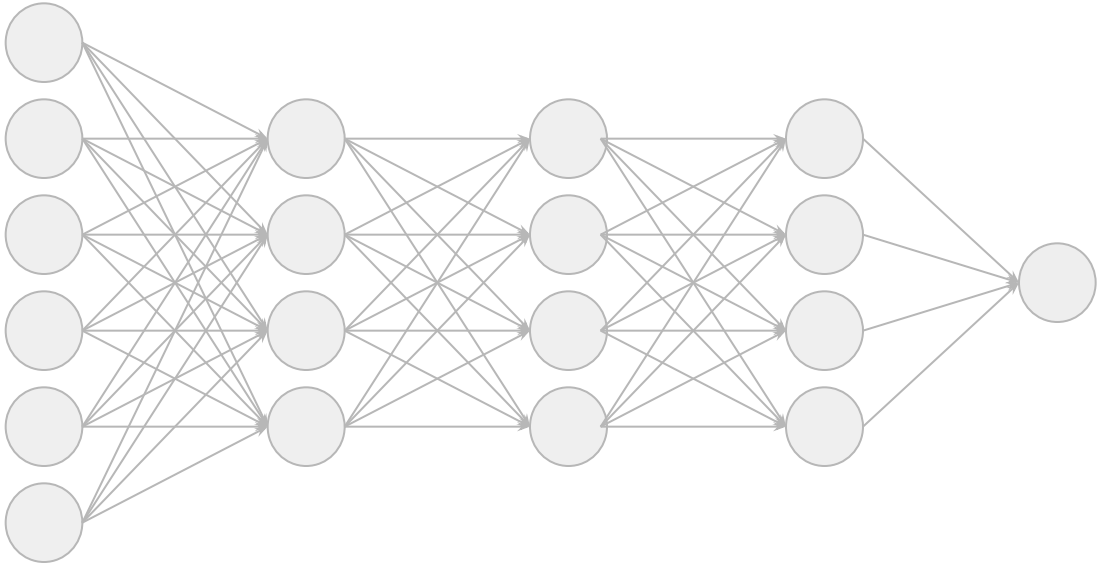
Norma L2: Genera penalización mediante la norma cuadrática. Tiende a regularizar coeficientes pero no eliminarlos.

Dropout



Se puede entender como una segunda forma de regularización entre capas ocultas.

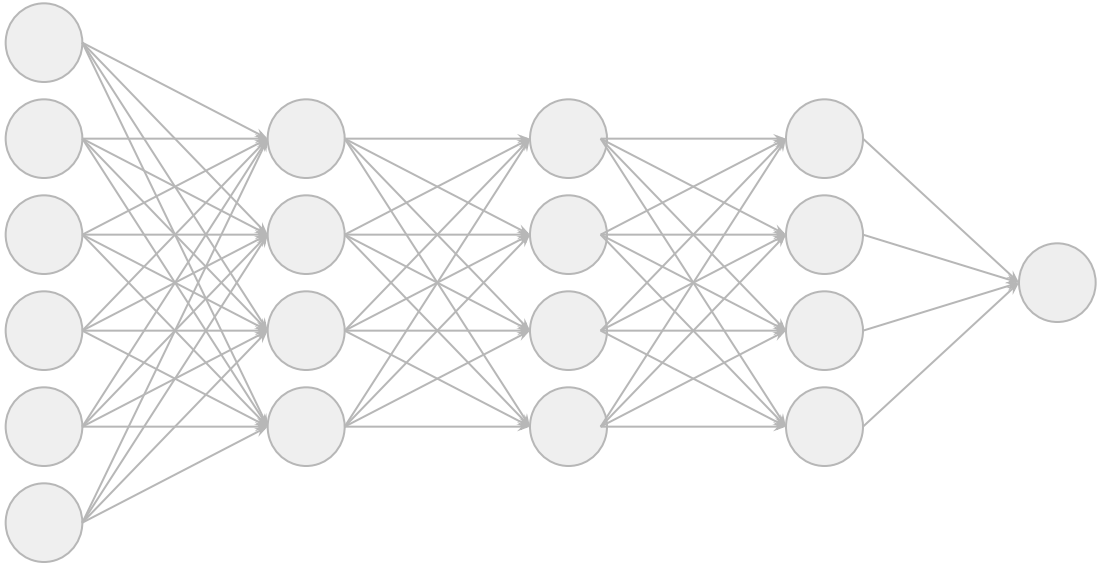
Dropout



Se puede entender como una segunda forma de regularización entre capas ocultas.

Dropout modifica la configuración de las neuronas dentro de la red durante entrenamiento.

Dropout

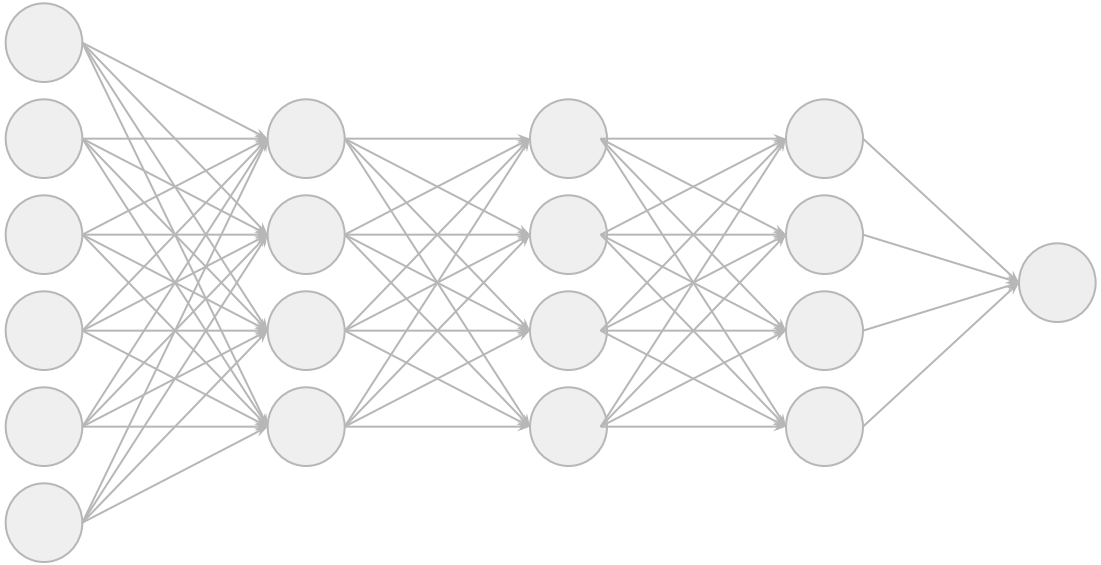


Se puede entender como una segunda forma de regularización entre capas ocultas.

Dropout modifica la configuración de las neuronas dentro de la red durante entrenamiento.

Principal ventaja de Dropout: No implica modificar la función de costo.

Dropout



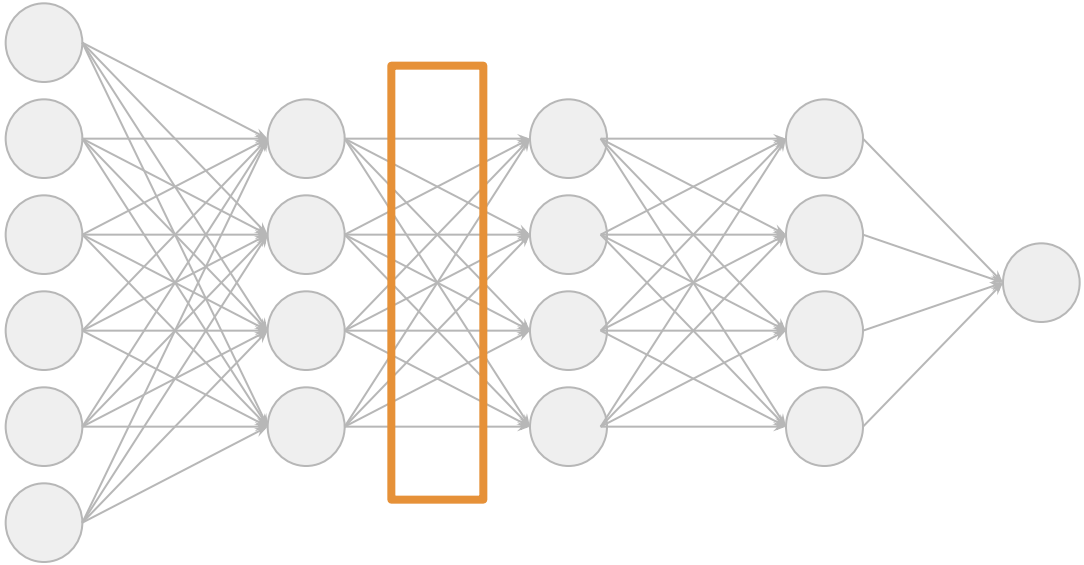
Se puede entender como una segunda forma de regularización entre capas ocultas.

Dropout modifica la configuración de las neuronas dentro de la red durante entrenamiento.

Principal ventaja de Dropout: No implica modificar la función de costo.

También permite agilizar el entrenamiento de la red neuronal.

Dropout



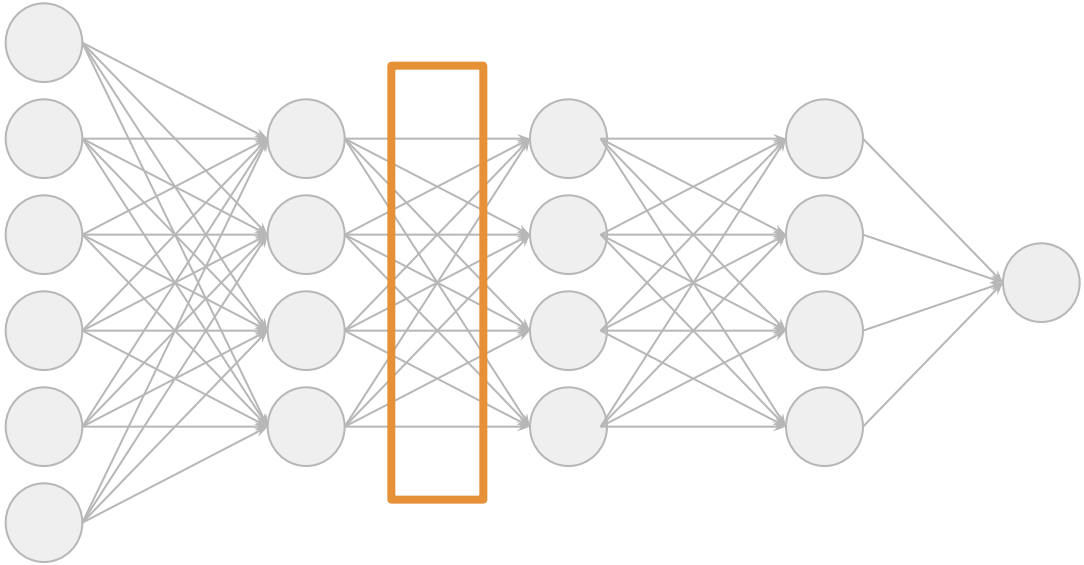
Se puede entender como una segunda forma de regularización entre capas ocultas.

Dropout modifica la configuración de las neuronas dentro de la red durante entrenamiento.

Principal ventaja de Dropout: No implica modificar la función de costo.

También permite agilizar el entrenamiento de la red neuronal.

Dropout



$$\text{Dropout}(w^{(1)}) = .5 = (16 \times .5) \rightarrow 8$$

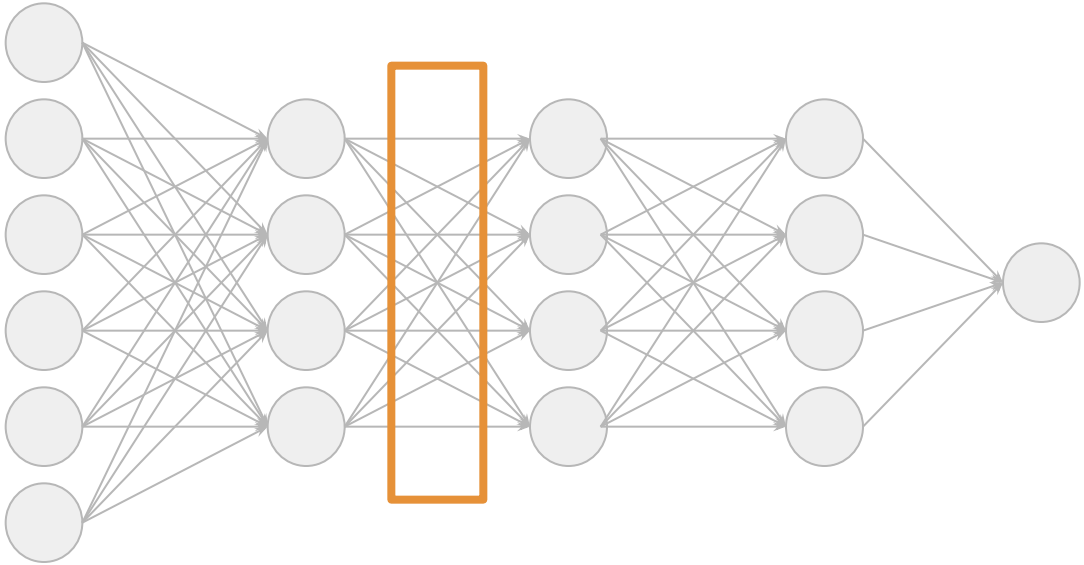
Se puede entender como una segunda forma de regularización entre capas ocultas.

Dropout modifica la configuración de las neuronas dentro de la red durante entrenamiento.

Principal ventaja de Dropout: No implica modificar la función de costo.

También permite agilizar el entrenamiento de la red neuronal.

Dropout



$$\text{Dropout}(w^{(1)}) = .5 = (16 \times .5) \rightarrow 8$$

Se puede entender como una segunda forma de regularización entre capas ocultas.

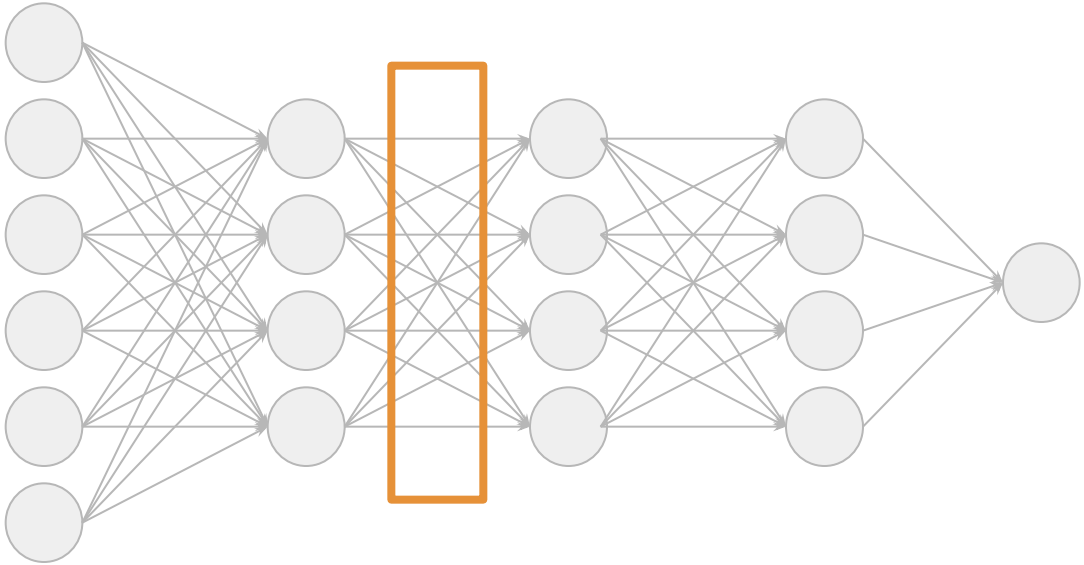
Dropout modifica la configuración de las neuronas dentro de la red durante entrenamiento.

Principal ventaja de Dropout: No implica modificar la función de costo.

También permite agilizar el entrenamiento de la red neuronal.

Mecánica del Dropout:
Definimos una probabilidad de eliminación (p) al comienzo de cada época de entrenamiento para la capa específica.

Dropout



$$\text{Dropout}(w^{(1)}) = .5 = (16 \times .5) \rightarrow 8$$

Se puede entender como una segunda forma de regularización entre capas ocultas.

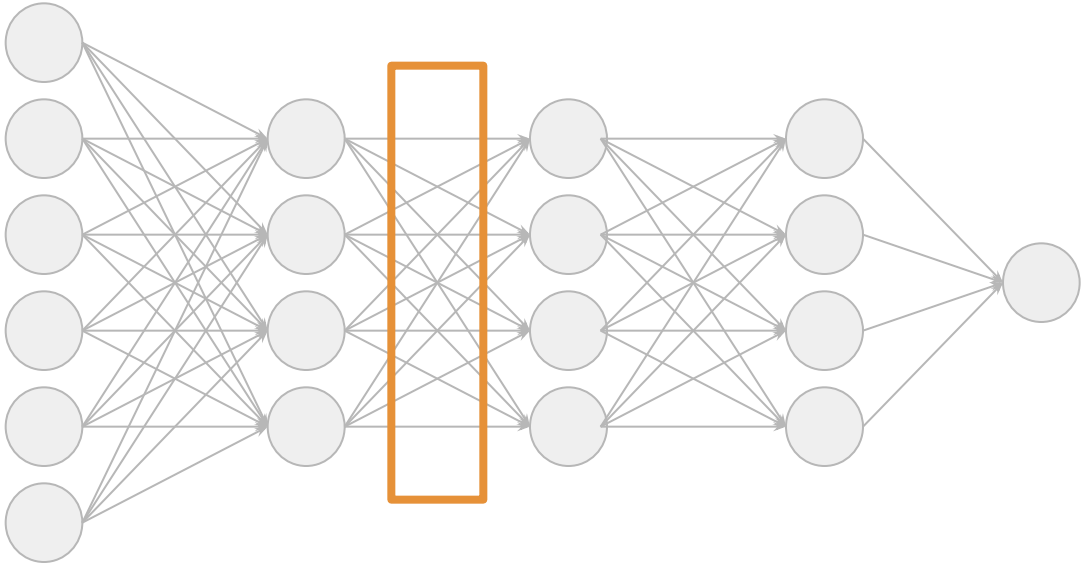
Dropout modifica la configuración de las neuronas dentro de la red durante entrenamiento.

Principal ventaja de Dropout: No implica modificar la función de costo.

También permite agilizar el entrenamiento de la red neuronal.

Mecánica del Dropout:
Definimos una probabilidad de eliminación (p) al comienzo de cada época de entrenamiento para la capa específica.
Evaluamos la tasa de mantención de cada neurona con (1-p).

Dropout



$$\text{Dropout}(w^{(1)}) = .5 = (16 \times .5) \rightarrow 8$$

Se puede entender como una segunda forma de regularización entre capas ocultas.

Dropout modifica la configuración de las neuronas dentro de la red durante entrenamiento.

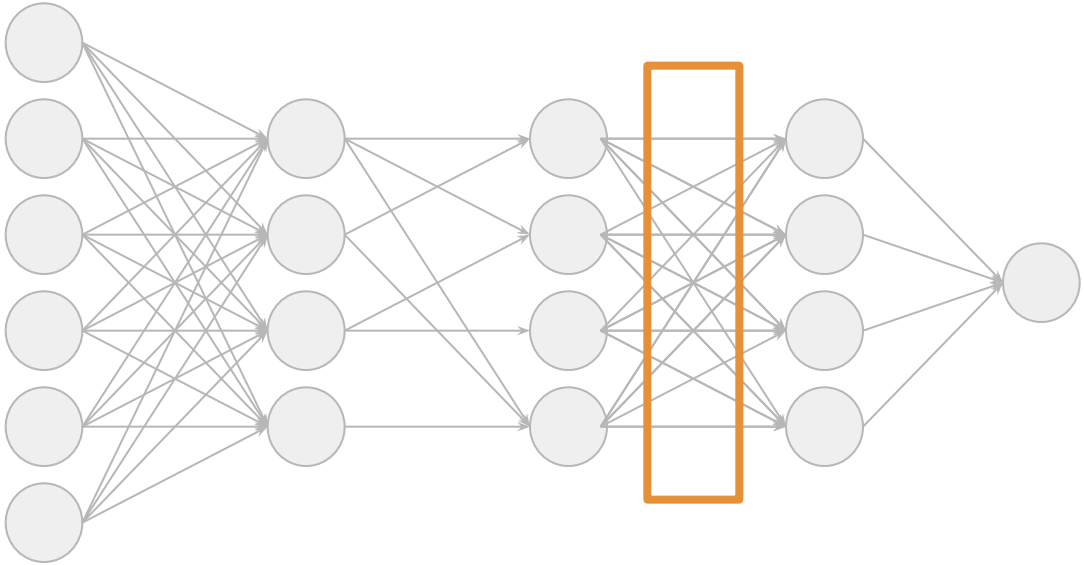
Principal ventaja de Dropout: No implica modificar la función de costo.

También permite agilizar el entrenamiento de la red neuronal.

Mecánica del Dropout:

Definimos una probabilidad de eliminación (p) al comienzo de cada época de entrenamiento para la capa específica.
Evaluamos la tasa de mantención de cada neurona con (1-p).
Las neuronas ignoradas actualizan su peso a 0, cancelando el impulso.

Dropout



$$\text{Dropout}(w^{(2)}) = .25 = (16 \times .25) \rightarrow 4$$

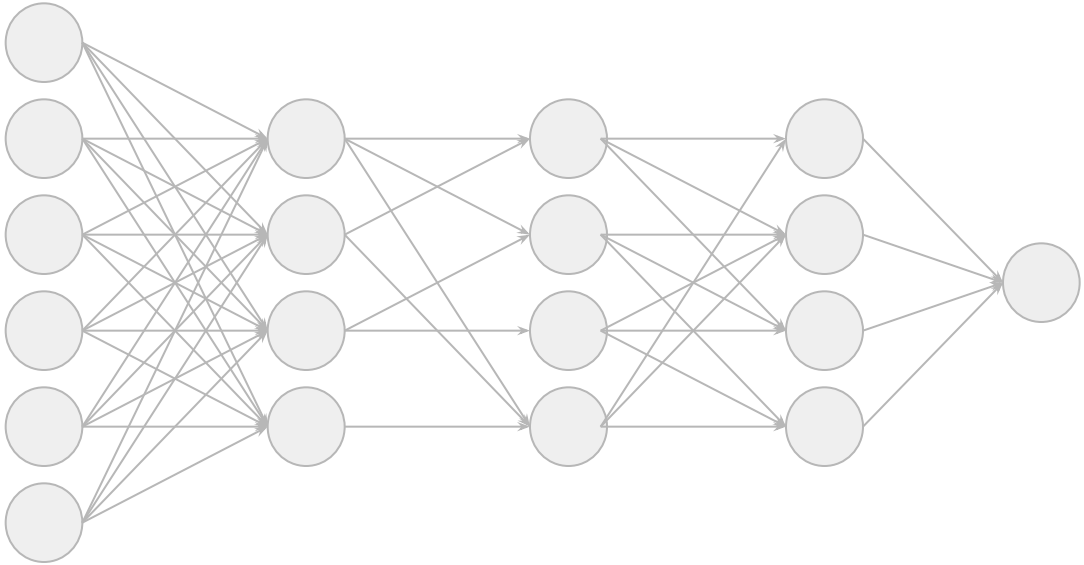
Se puede entender como una segunda forma de regularización entre capas ocultas.

Dropout modifica la configuración de las neuronas dentro de la red durante entrenamiento.

Principal ventaja de Dropout: No implica modificar la función de costo.

También permite agilizar el entrenamiento de la red neuronal.

Dropout



$$\text{Dropout}(w^{(2)}) = .25 = (16 \times .25) \rightarrow 4$$

Se puede entender como una segunda forma de regularización entre capas ocultas.

Dropout modifica la configuración de las neuronas dentro de la red durante entrenamiento.

Principal ventaja de Dropout: No implica modificar la función de costo.

También permite agilizar el entrenamiento de la red neuronal.

{desafío} latam_

Academia de talentos digitales

www.desafiolatam.com