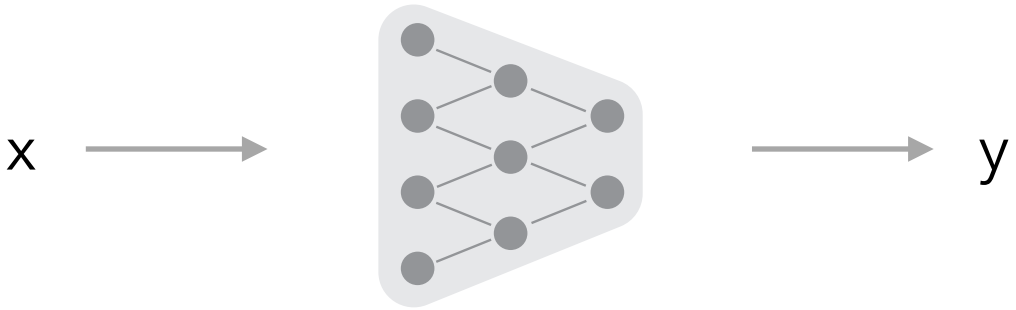
Arquitectura Básica de Redes Neuronales

Capas y Redes Feedforward

Redes Neuronales como Modelos de Computación

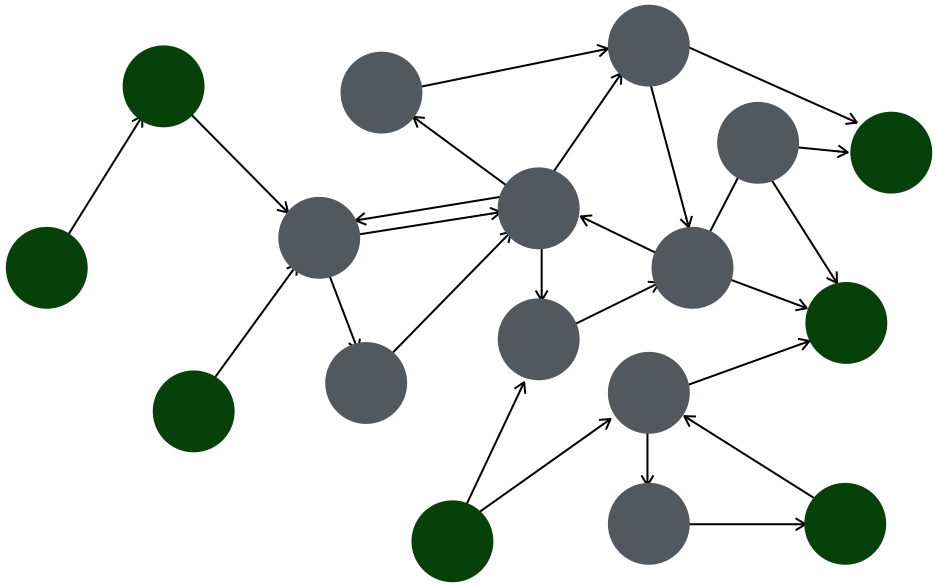
- Hemos dicho que una **red neuronal** es un grafo de computación que permite transformar datos de entrada en datos de salida, implemento una función parametrizada entre dos espacios (X,Y).



- Hemos especificado el modelo de cómputo que usa cada **unidad del grafo** pero para que la red esté completa es necesario especificar el patrón de conectividad: **quién es vecino de quién?**

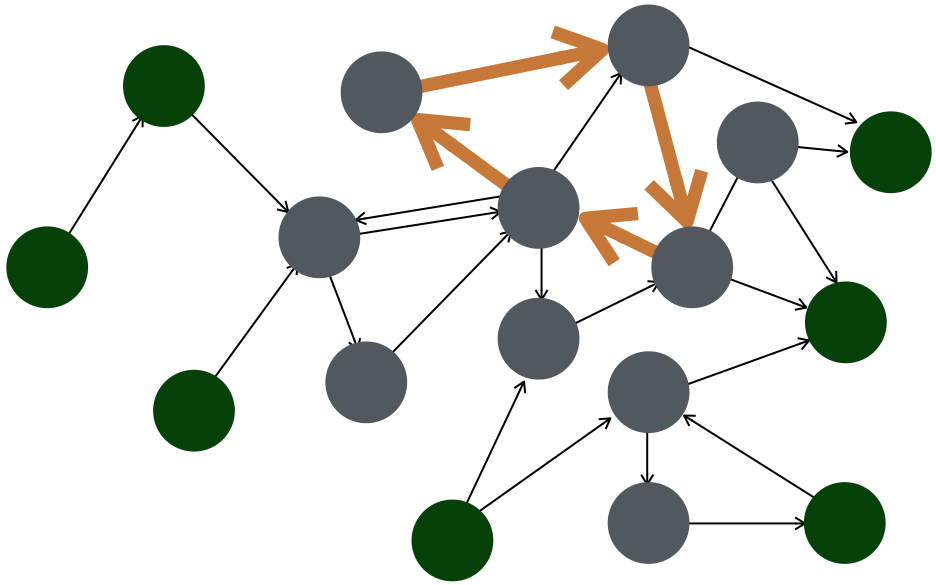
Arquitectura de la Red

- El patrón de conectividad es lo que se denomina **la arquitectura** de la red neuronal. Si la arquitectura es **completamente libre** como hasta el momento, **resulta difícil implementarla** eficientemente en un programa.



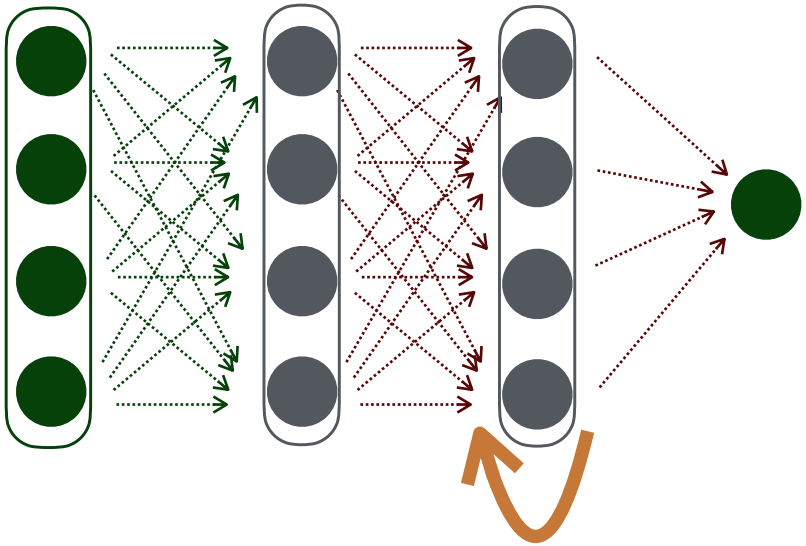
Ciclos o Recurrencias

- La red podría tener por ejemplo **ciclos**. Estos ciclos son muy importantes para implementar ideas (ya sugeridas por Pitts) como las de **contexto y memoria**, pero hacen más compleja la notación y la implementación, ya que nos obligan a considerar explícitamente un reloj para sincronizar/ordenar la computación.

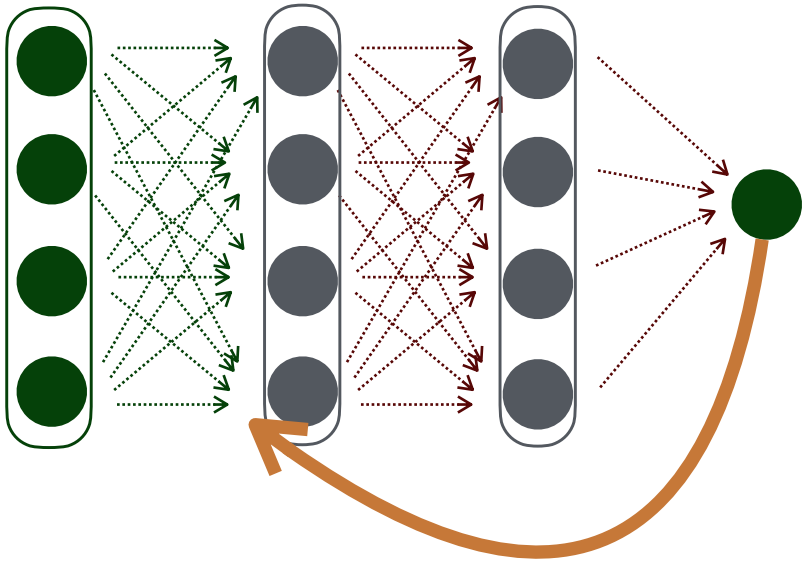


Ciclos o Recurrencias

- Consideraremos formas restringidas de estos ciclos cuando estudiemos redes recurrentes.



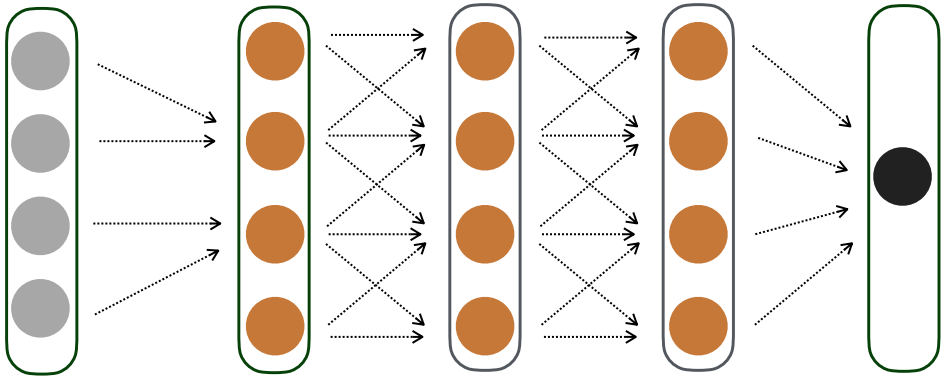
Elman



Jordan

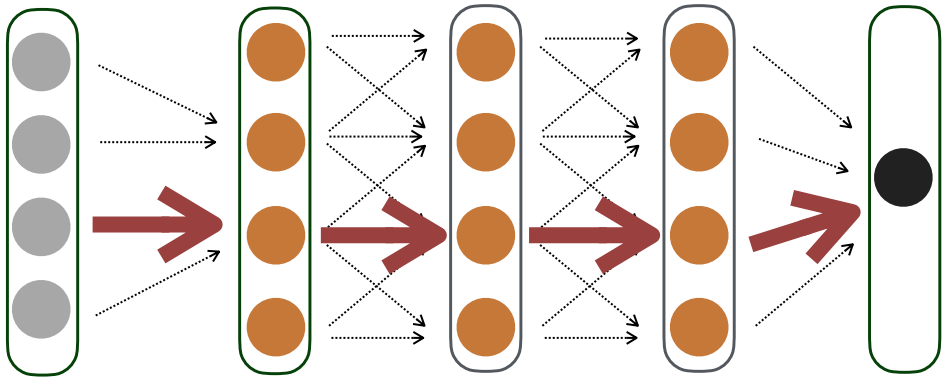
Capas

- Por ahora, el objetivo es **restringir la arquitectura** para permitir una implementación simple y eficiente, limitando lo menos posible la expresividad del modelo. La idea que adoptaremos será organizar las neuronas en **capas, dispuestas secuencialmente.**



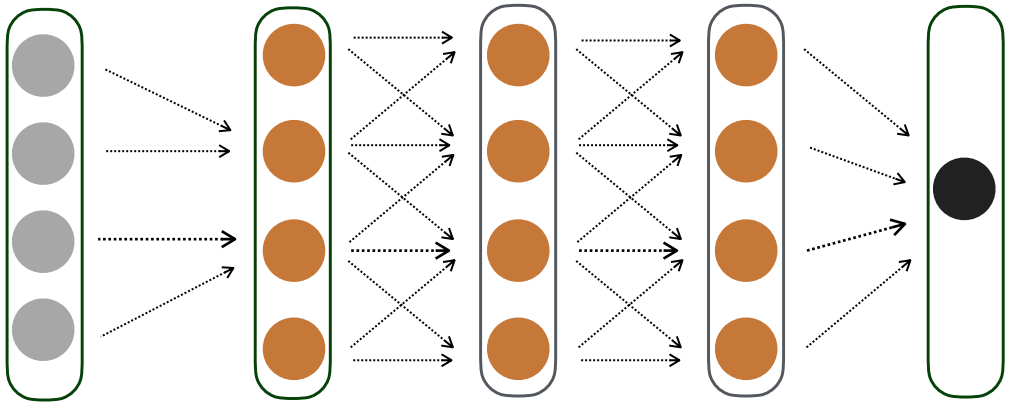
Arquitectura Feed-Forward

- Las neuronas de una capa no se comunicarán entre sí, **se conectarán sólo con neuronas de la capa anterior y de la capa siguiente.**
- Este tipo de red se denomina ***red feed-forward***, por la forma en que se propaga la información: las neuronas de una capa sólo leen información de la capa anterior, y sólo transmiten información hacia la capa siguiente.



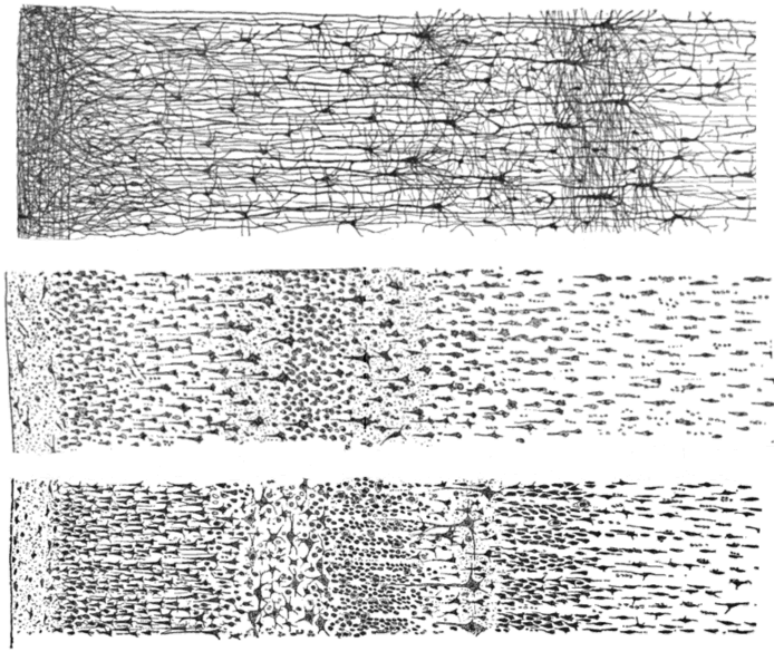
Nomenclatura de las Capas

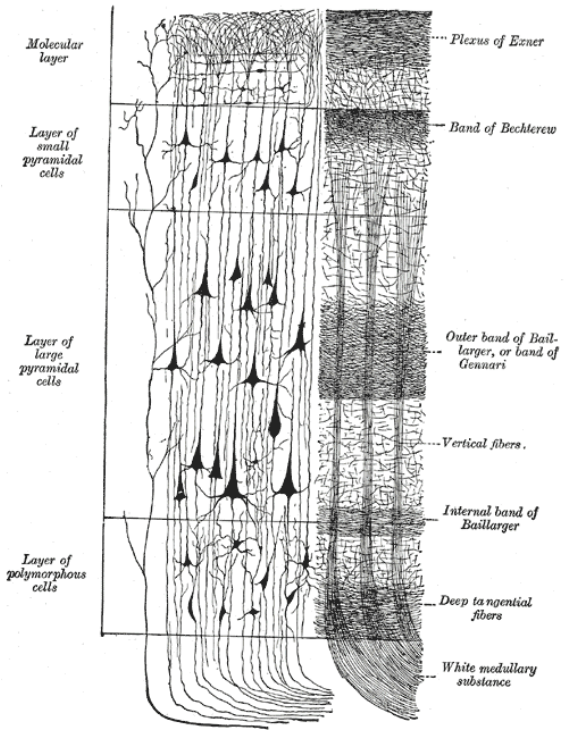
- Las capas de la red se nombrarán de acuerdo a su función: tendremos siempre una ***capa de entrada***, que acomoda los datos de entrada, una ***capa de salida*** desde donde se lee el resultado de la computación, y un número variable de ***capas ocultas o capas escondidas,*** que efectúan cálculos intermedios.



Capas en Redes Neuronales Naturales

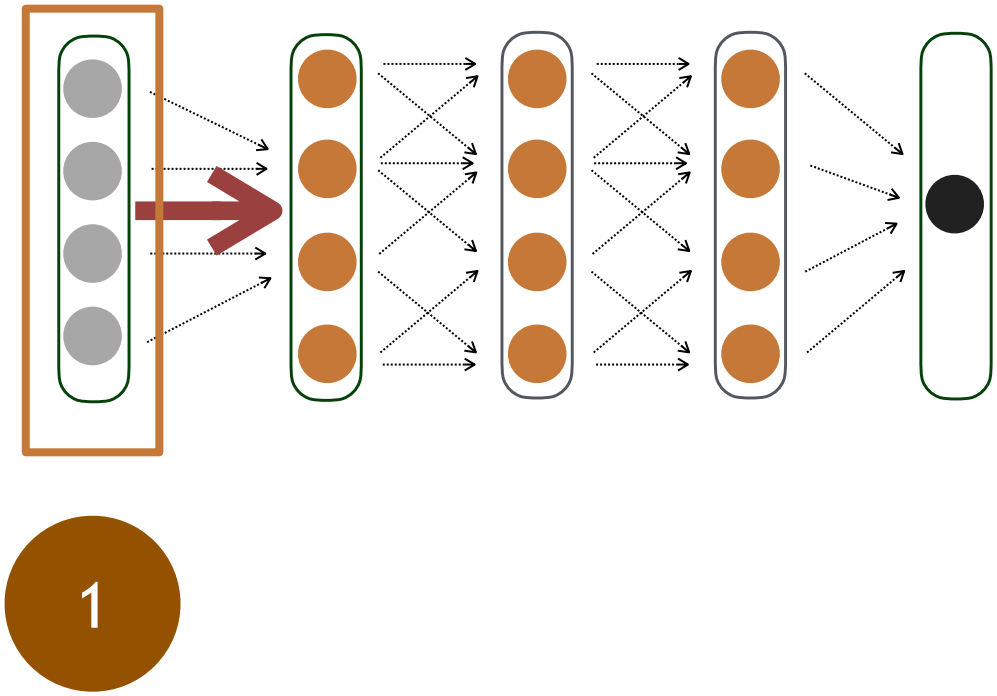
- En redes neuronales biológicas se ha observado que el patrón de conectividad suele ser muy regular. En zonas como la corteza visual y motora se ha sugerido la existencia de capas.





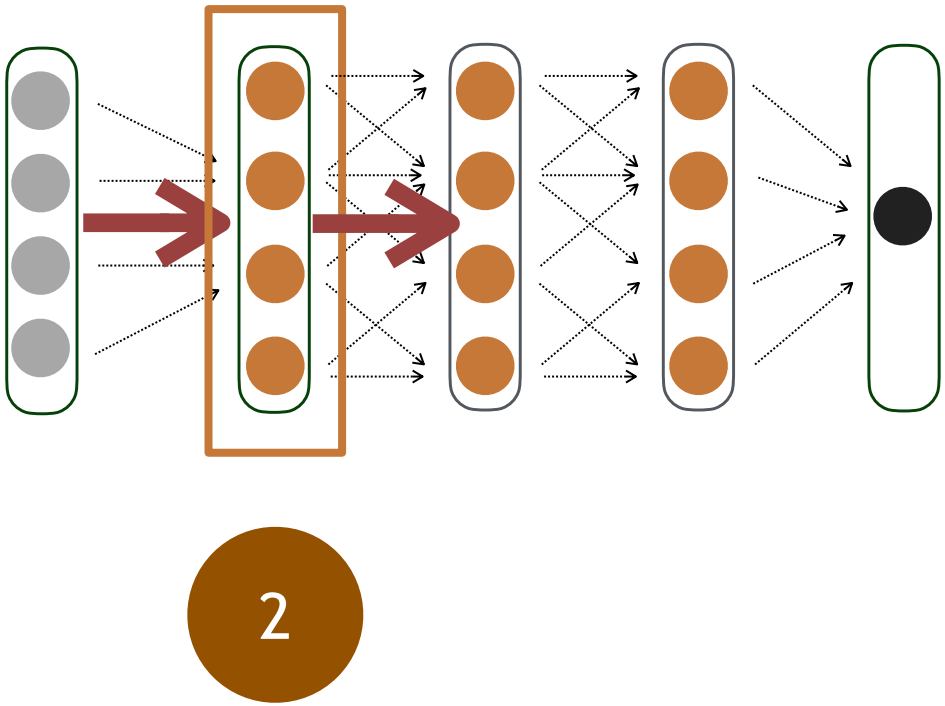
Flujo de la Computación

- Notemos que una arquitectura feed-foward la sincronización del cómputo es muy simple. Primero, la capa de entrada lee los datos de entrada.



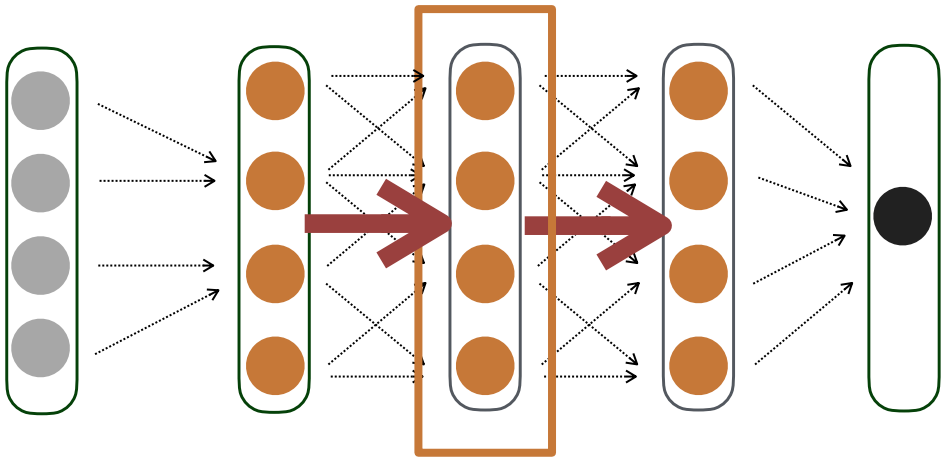
Flujo de la Computación

- Luego, las neuronas de la primera capa escondida hacen sus cálculos. Como estas neuronas son independientes, pueden operar en paralelo.



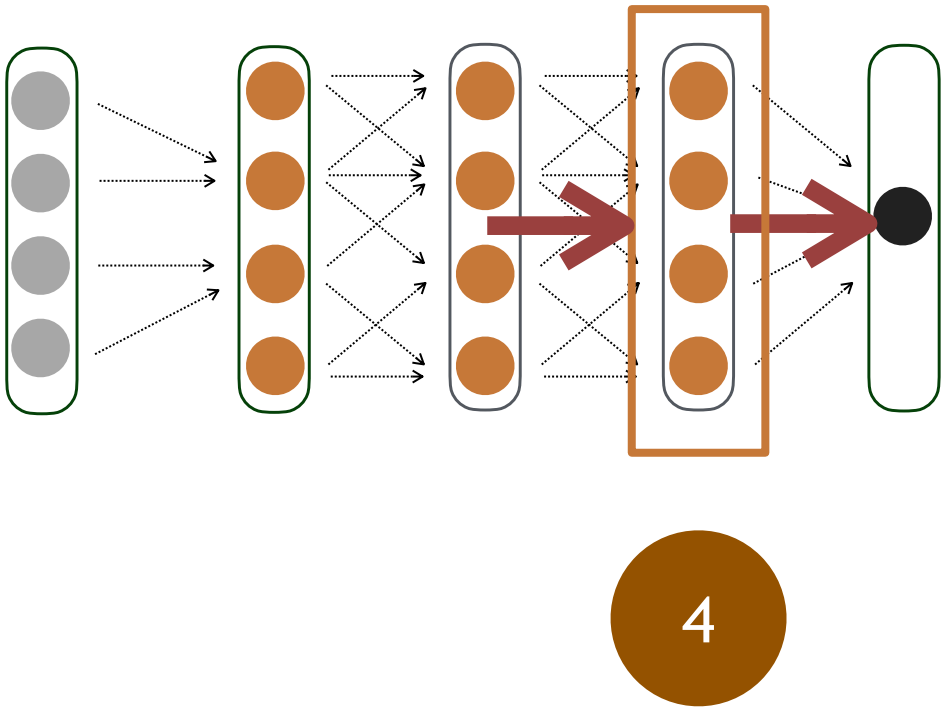
Flujo de la Computación

- Luego, las neuronas de la segunda capa escondida hacen sus cálculos. Estas neuronas también pueden operar en paralelo.



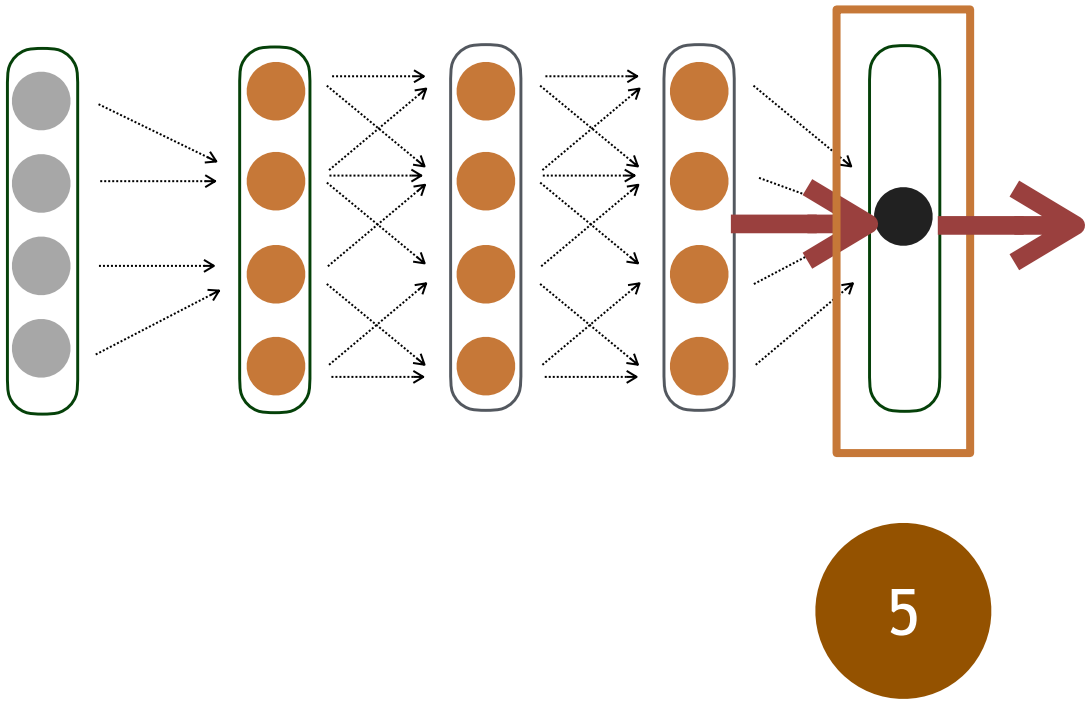
Flujo de la Computación

- Y así sucesivamente …



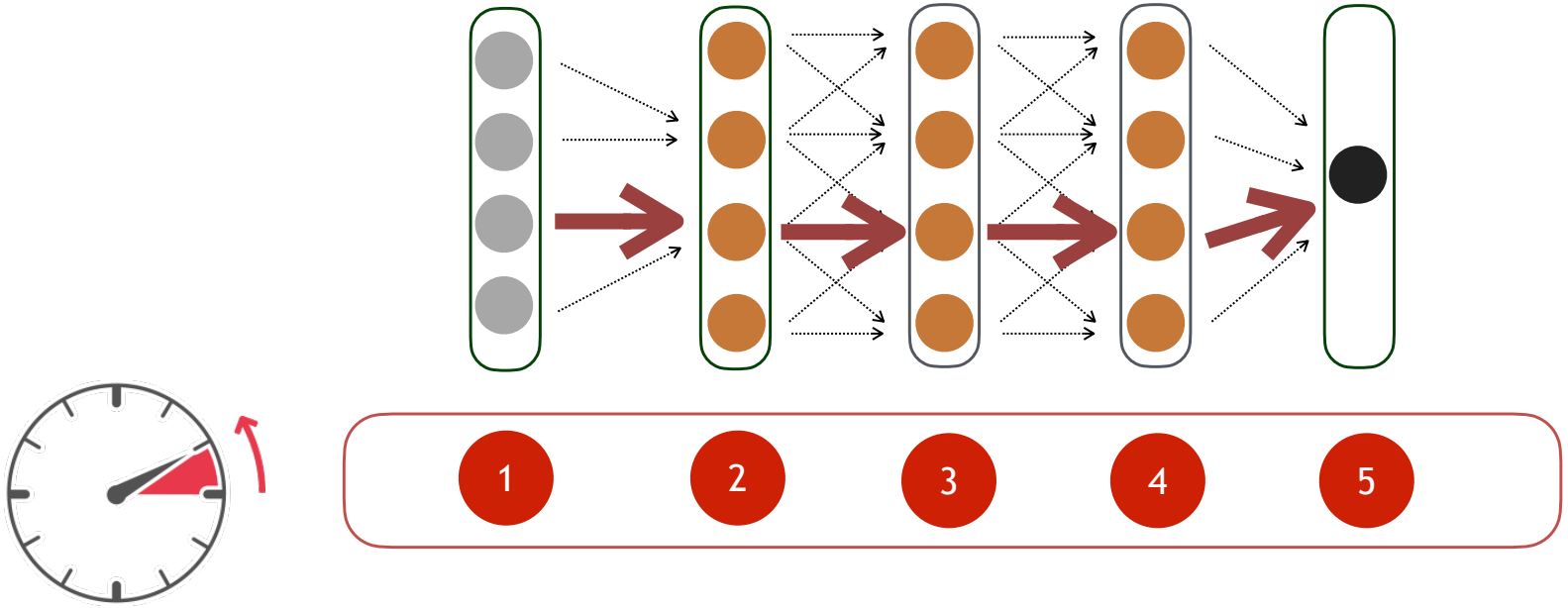
Flujo de la Computación

- Hasta que la capa de salida está lista para hacer los cálculos que se entregarán como salida de la red (resultado de la computación).



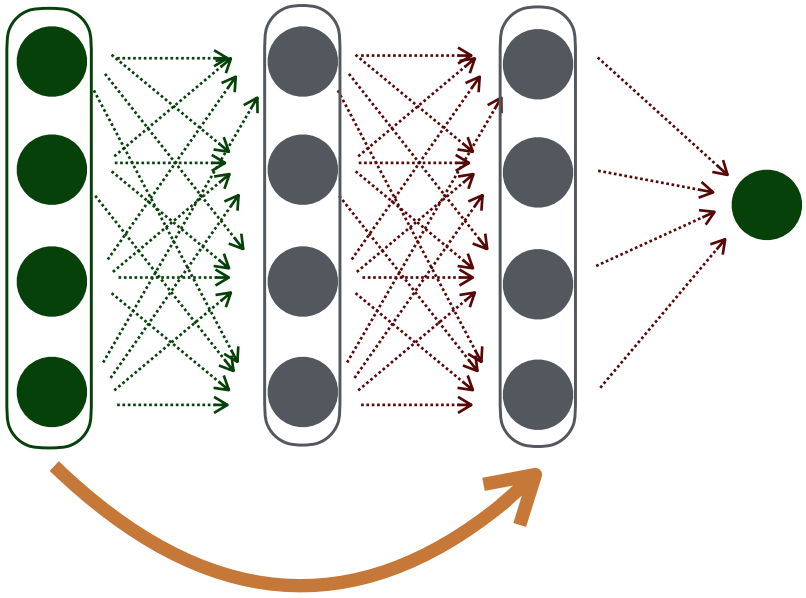
Arquitectura Feed-Forward

- En resumen, cada capa “espera” que la capa anterior haya “terminado” sus cálculos y las neuronas de una misma capa pueden operar **en paralelo**.
- No es necesaria la introducción explícita de un reloj: el tiempo está implícitamente dado por la secuencialidad de las capas.



Saltos Inter-Capa (Skip Connections)

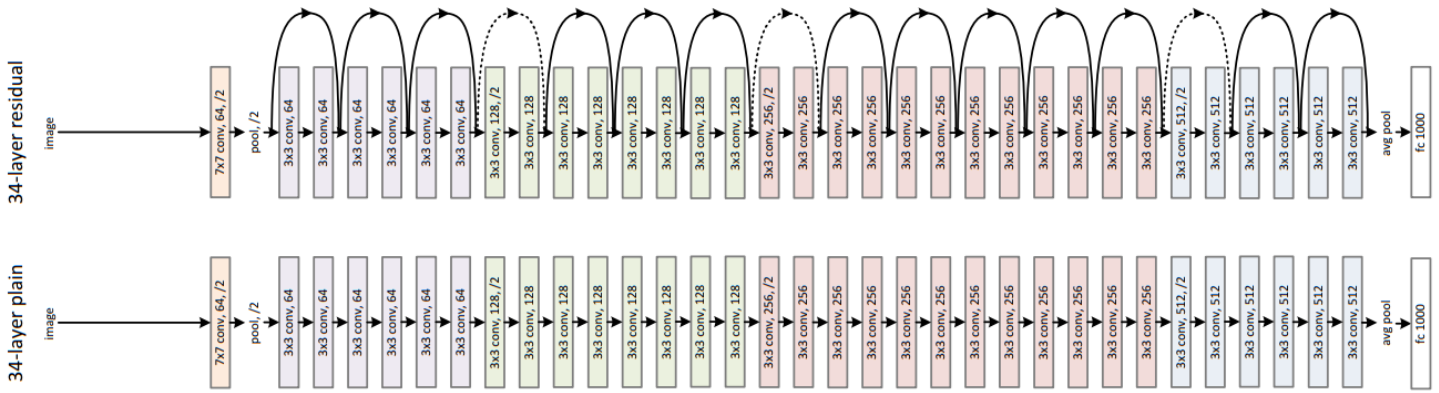
- No es difícil demostrar que cualquier red neuronal sin ciclos puede organizarse en capas tales que el flujo de la computación uni-direccional. Sin embargo, el resultado podría contener saltos entre capas no consecutivas (skip connections).



skip connection

Saltos Inter-Capa (Skip Connections)

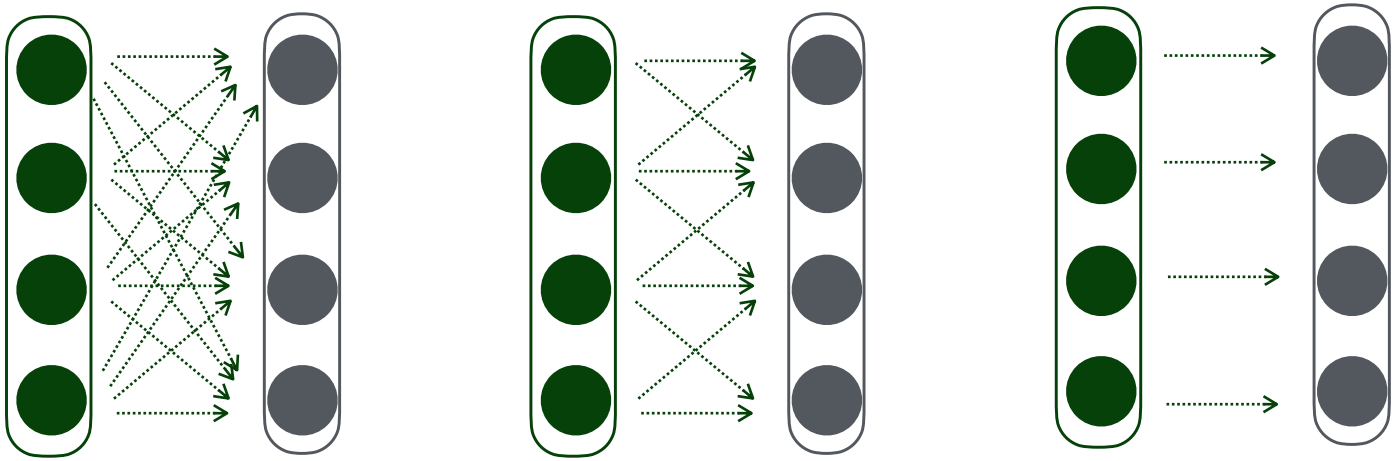
- Estas **skip connections** han demostrado ser muy útiles en redes complejas para tareas como detección, localización y segmentación de objetos en imágenes, pero por simplicidad una red feed-forward convencional no las considera.



Microsoft's ResNet (2015)

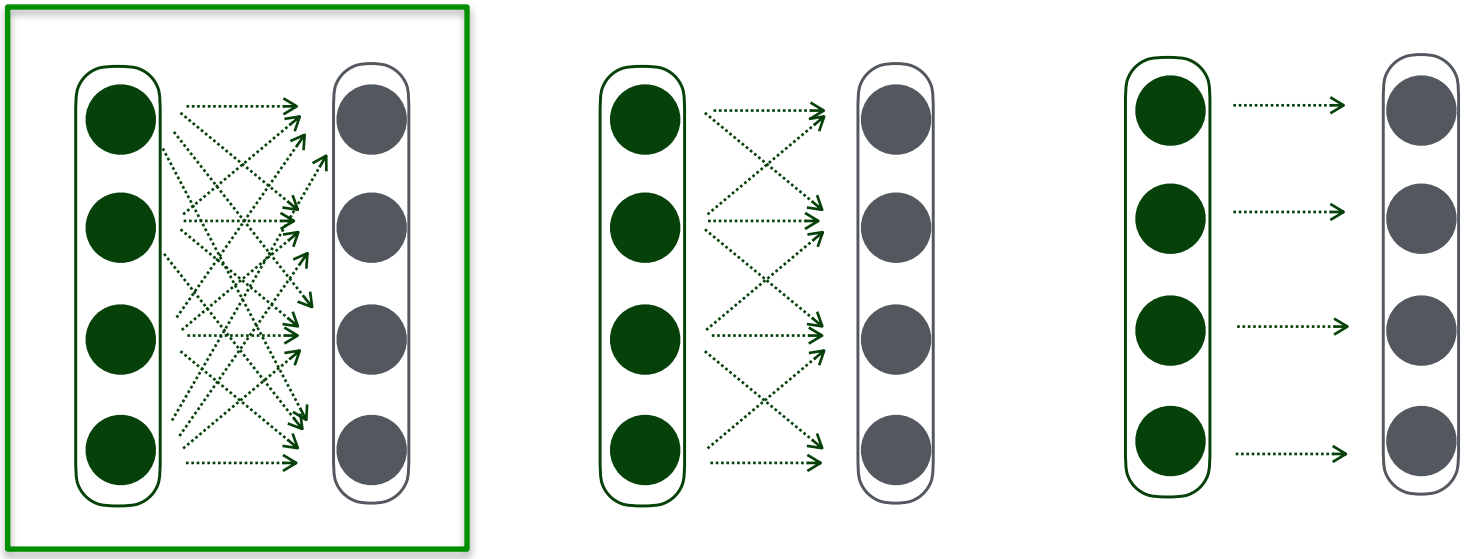
Capas Densas versus Conectividad Dispersa

- ¿Cómo se comunican las neuronas de capas sucesivas?



Capas Densas versus Conectividad Dispersa

- En una arquitectura feed-forward tradicional las neuronas de una capa están conectadas con ***todas las neuronas de la capa anterior*** y en general, usan la misma función de activación.



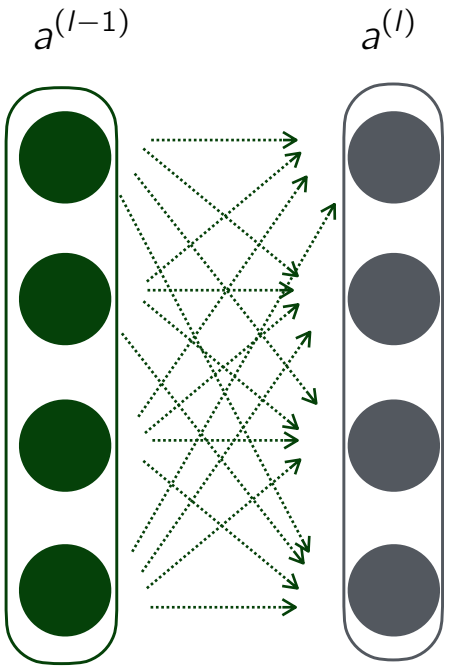
- Consideraremos patrones dispersos de comunicación cuando veamos redes convolucionales.

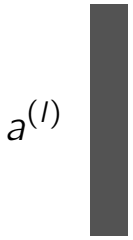
Capas Densas versus Conectividad Dispersa

- Ventajas:
    - Esta forma ***densa*** de conectividad nos permitirá escribir de manera sencilla **la activación de toda una capa a partir de toda la activación de la capa anterior.**
    - En términos de implementación, podremos **vectorizar** el cálculo de las capas y gozar de la eficiencia que las operaciones de álgebra lineal tienen hoy en muchos lenguajes de programación.

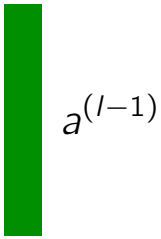
Vectorización de una Red FF

- La **activación** (resultado del cálculo) de cada capa se representará usando un **vector** que tiene tantos elementos como neuronas haya en esa capa.





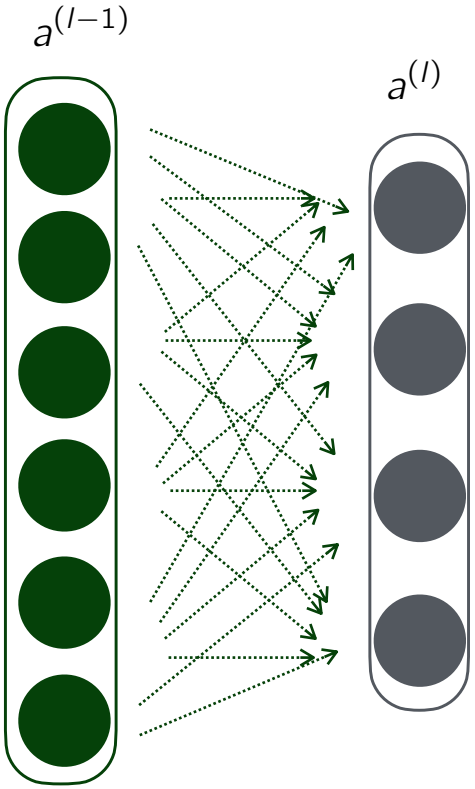
activación de la capa l



activación de la capa l-1

Vectorización de una Red FF

- Por ejemplo:



$a^{(l)} \in \mathbb{R}^4$

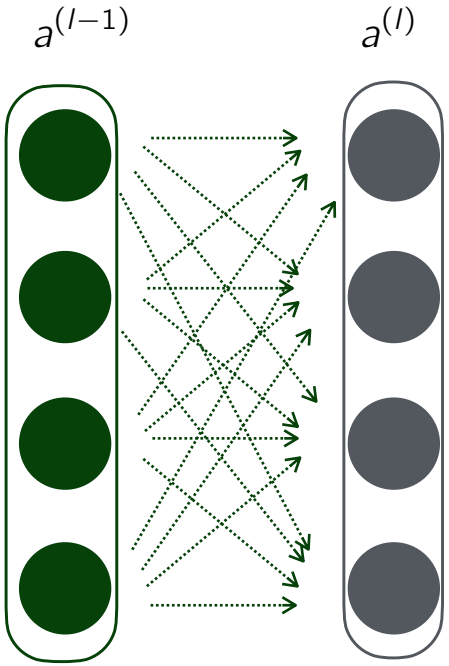
activación de la capa l

$a^{(l-1)} \in \mathbb{R}^6$

activación de la capa l-1

Vectorización de las Capas

- El modelo de neurona MP nos lleva a la siguiente ecuación entre capas sucesivas:



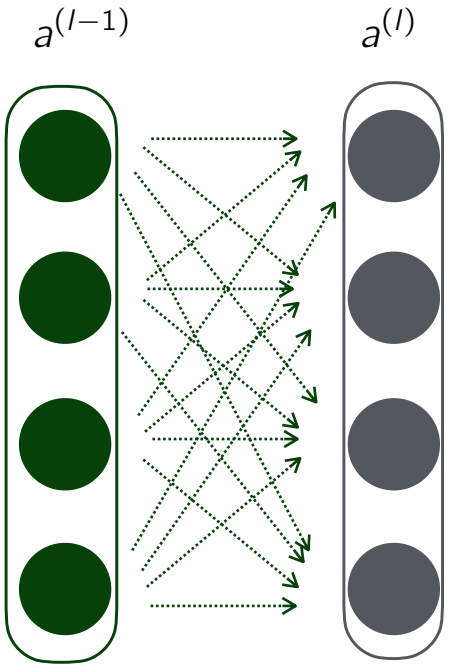
$$a^{(l)} = \sigma_l\left(W^{(l)}a^{(l-1)} - b^{(l)}\right)$$

activación de la capa l

activación de la capa l-1

Vectorización de las Capas

- Para obtener la activación de la capa l, primero multiplicamos la activación de la capa l-1 por una matriz $W^{(l)}$ y restamos un vector $b^{(l)}$. Luego, aplicamos una función no-lineal $\sigma_{(l)}$ a cada elemento del resultado.



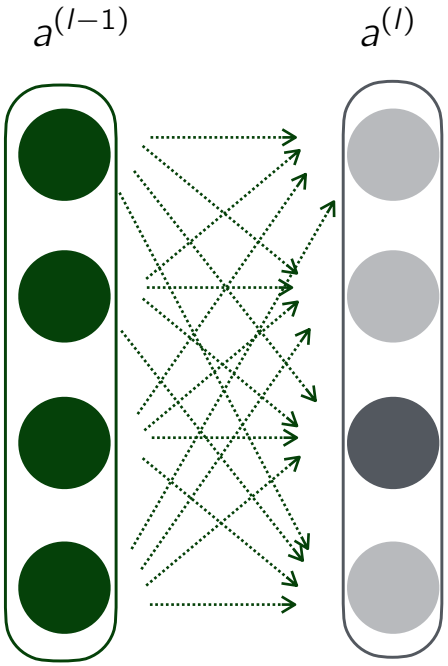
$$a^{(l)} = \sigma_l\left(W^{(l)}a^{(l-1)} - b^{(l)}\right)$$

activación de la capa l

activación de la capa l-1

Vectorización de las Capas

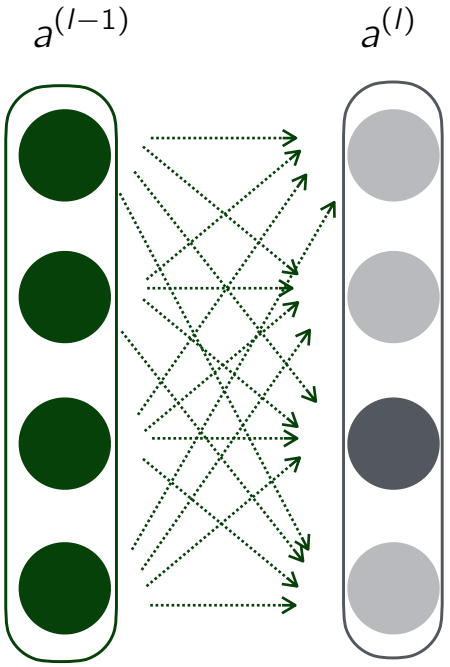
- Si miramos en detalle lo que sucede con el elemento i-ésimo del vector que corresponde a la capa l, vemos que ese trata simplemente de la ecuación de la neurona MP.



$$a_i^{(l)} = \sigma_l\left(W_{i,:}^{(l)} a^{(l-1)} - b^{(l)}\right)$$
$$= \sigma_l\left(W_1^{(l)} a_1^{(l-1)} + W_2^{(l)} a_2^{(l-1)} + \ldots W_d^{(l)} a_d^{(l-1)} - b^{(l)}\right)$$

Vectorización de las Capas

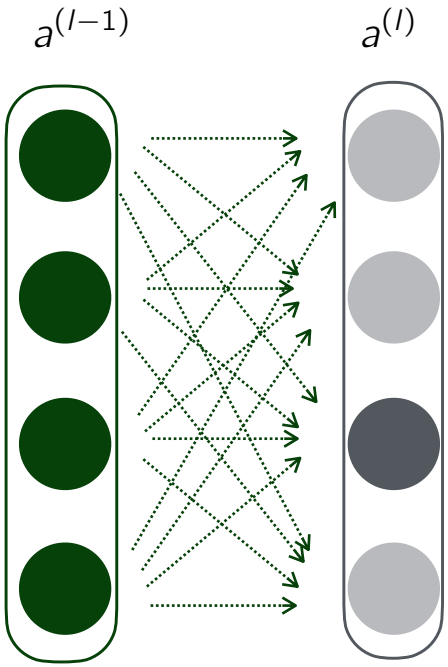
- El vector **b** corresponde entonces a **vector de umbrales de activación**.
- La matriz **W** es simplemente **la matriz de pesos de conexión**: el elemento **Wij** de la matriz es el peso de conexión entre la neurona j de la capa l-1 y la neurona i de a capa l.



$$a_i^{(l)} = \sigma_l\left(W_{i,:}^{(l)} a^{(l-1)} - b^{(l)}\right)$$
$$= \sigma_l\left(W_1^{(l)} a_1^{(l-1)} + W_2^{(l)} a_2^{(l-1)} + \ldots W_d^{(l)} a_d^{(l-1)} - b^{(l)}\right)$$

Vectorización de las Capas

- La no-linealidad que se aplica es simplemente la **función de activación** (compartida) de las neuronas de esa capa.

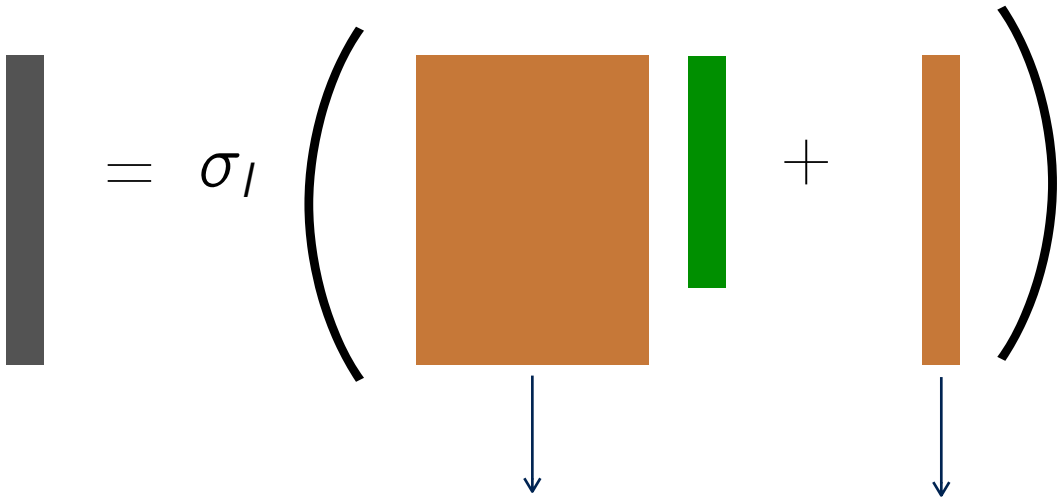


$$a_i^{(l)} = \sigma_l\left(W_{i,:}^{(l)} a^{(l-1)} - b^{(l)}\right)$$
$$= \sigma_l\left(W_1^{(l)} a_1^{(l-1)} + W_2^{(l)} a_2^{(l-1)} + \ldots W_d^{(l)} a_d^{(l-1)} - b^{(l)}\right)$$

Vectorización de las Capas

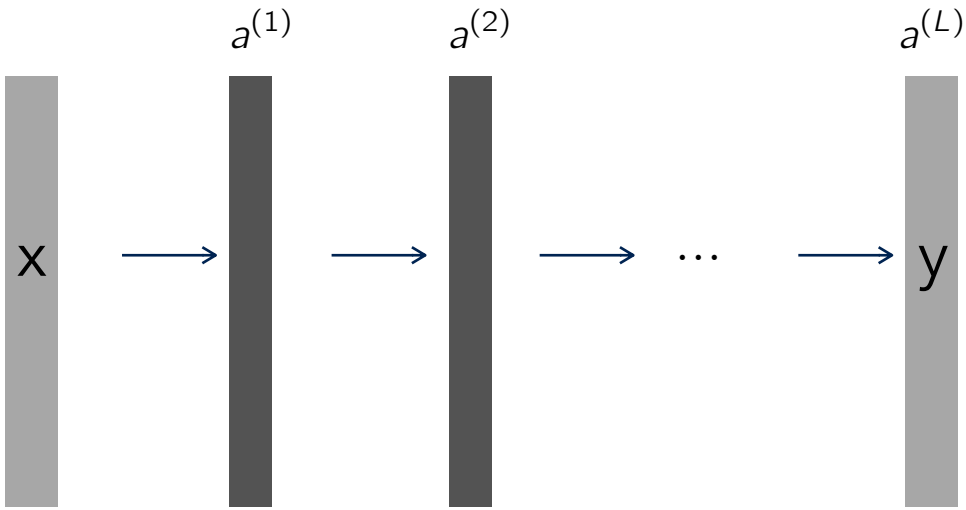
- Si la capa l tiene N neuronas y la capa l-1 tiene M neuronas, ¿Qué dimensiones tiene las matriz W y el vector b?

$$a^{(l)} = \sigma_l \left(W^{(l)} a^{(l-1)} - b^{(l)} \right)$$



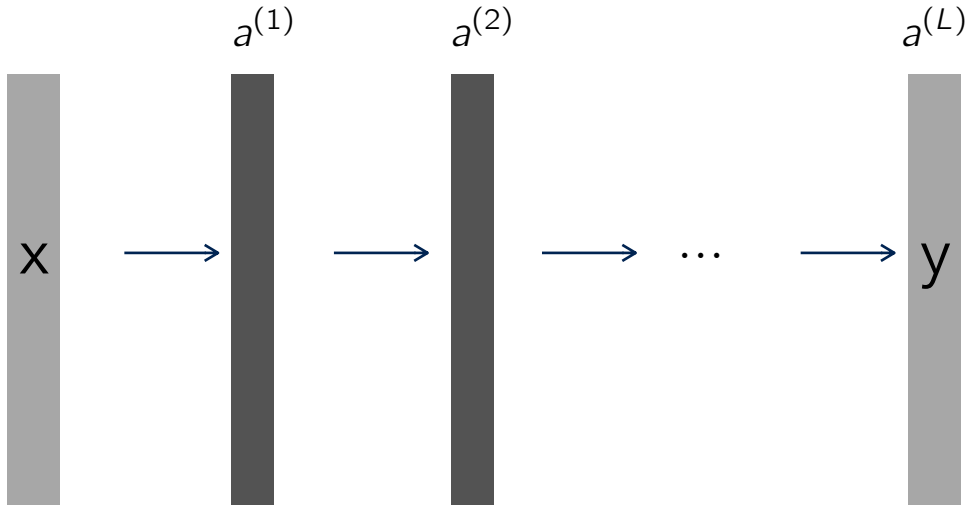
Red Feed-Forward Estándar

- Obtenemos así un versión práctica y conveniente del modelo general …
- Una red feed-foward implementa una función entre X e Y **componiendo una serie de transformaciones vector-vector** (entra una vector, sale un vector).



Red Feed-Forward Estándar

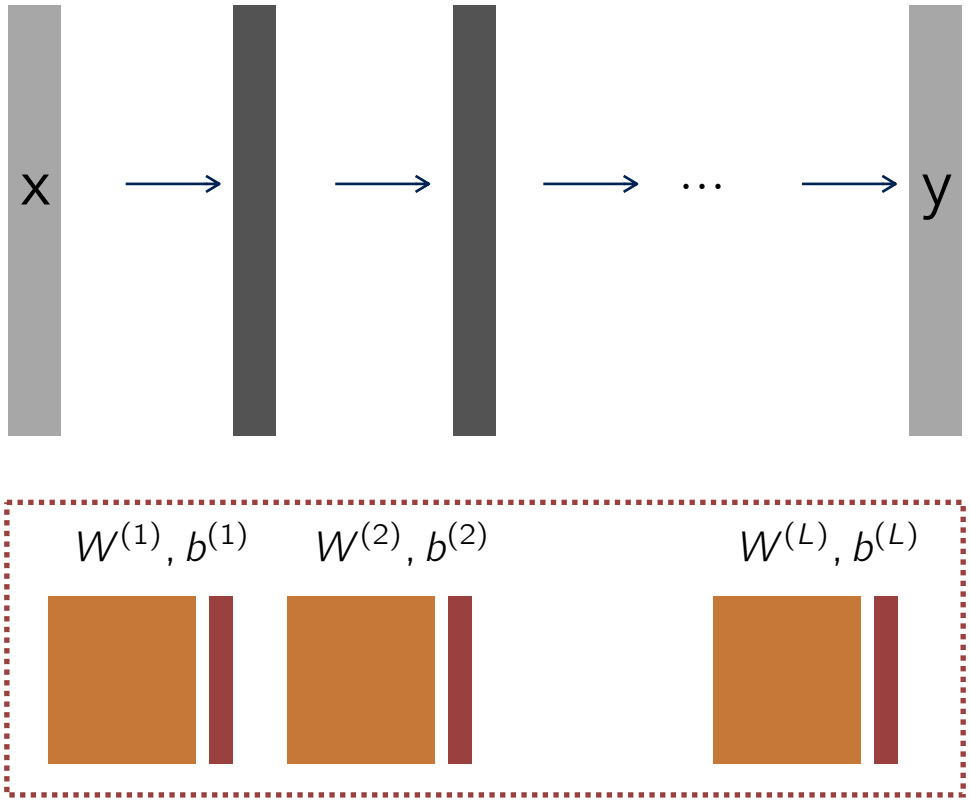
- La secuencia de transformaciones parte desde el vector de entrada y termina en un vector que representa el cálculo efectuado por la red.
- Cada transformación tiene la misma forma: **transformación lineal + no linealidad** element-wise.



$$a^{(l)} = \sigma_l\left(W^{(l)}a^{(l-1)} - b^{(l)}\right)$$

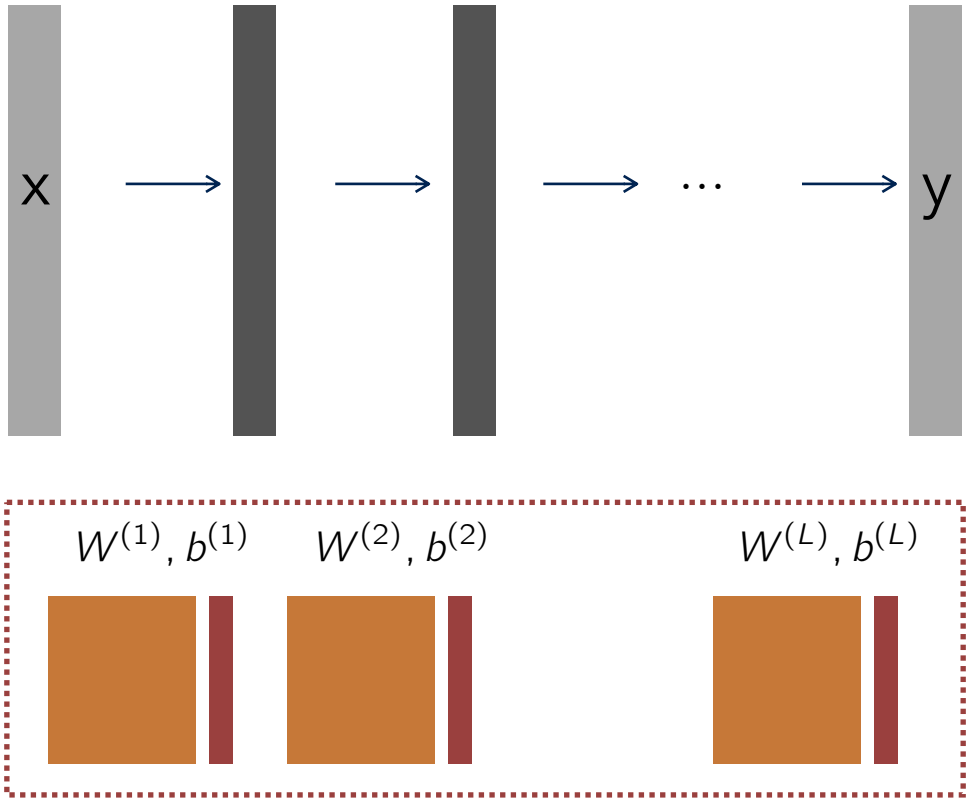
Red Feed-Forward Estándar

- Los parámetros que controlan la forma de a transformación son dos: una matriz de pesos y un vector de umbrales de excitación.



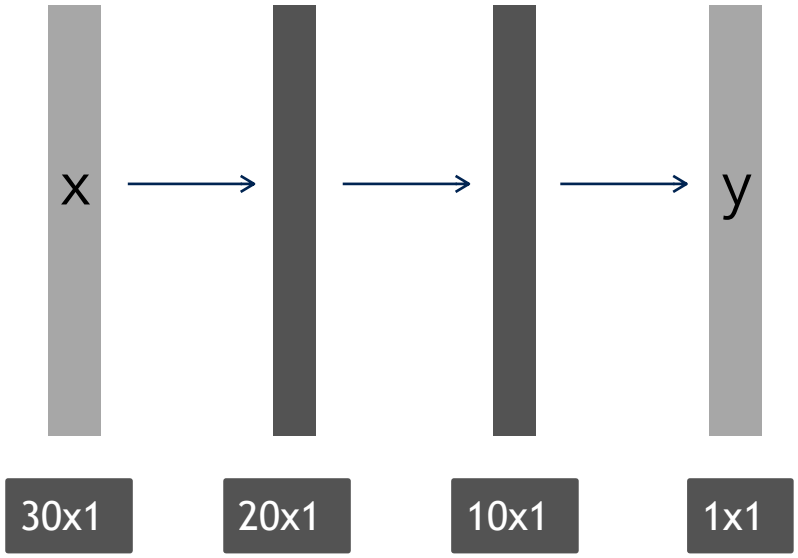
Red Feed-Forward Estándar

- Entrenar la red significará determinar valores para esos parámetros de modo que la red produzca las salidas que queremos.



Ejercicio

- Consideremos una red FF con 4 capas como en el dibujo.
- El vector de entrada tiene 30 atributos y la salida es un escalar.
- Si la primera capa oculta tiene 20 neuronas y la segunda tiene 10 neuronas: **cuántos parámetros entrenables tiene el modelo?**



Entonces ...

- Una **red neuronal feed-forward (FF)** es una restriccción del modelo general que permite obtener una implementación simple y eficiente.
- El modelo queda definido como una serie de transformaciones vector-vector que dependen de "2" parámetros entrenables (una matriz y un vector). Algorítmicamente (*):

1 $\boldsymbol{a}^{(1)} = \boldsymbol{x}$;
2 **for** $\ell = 1, \ldots, L-1$ **do**
3 $\quad \boldsymbol{a}^{(\ell+1)} = \sigma(\boldsymbol{W}^{(\ell)\top}\boldsymbol{a}^{(\ell)} + \boldsymbol{w}_0^{(\ell)})$;
4 **end**
5 return $\boldsymbol{a}^{(L)}$

- Tarea: programe una red FF.

(*) hay que decidir si indexaremos desde la capa de entrada o desde la primera oculta.