Neural Methods for NLP

Course 2: Introduction to Deep Learning

Master LiTL --- 2021-2022
chloe.braud@irit.fr

https://github.com/chloebt/m2-litl-students

30/11/2021

(Tentative) Schedule

- S47 - C1: 23.11 13h30-15h30 (2H) → ML reminder + TP1
- S48 - C2: 30.11 13h30-15h30 (2H) → Intro DL + TP2
~~**S49 - C3: 07.12 13h30-16h30 (3H) →**~~
- S50 - C3: 14.12 13h30-16h30 (**3H**) → Embeddings + TP3
- S51 - 21.12: break
- S52 - 28.12: break
- S1 - C4: 04.01 10h-12h (2H) → Training a NN
- **?? S1 - C5: 06.01 13h-15h (2H) → TP4**
- S2 - C6: 14.01 10h-12h (2H) → [Start projects]
- S3 - C7: 18.01 13h-16h (**3H**) → CNN, RNN + TP5
- S4 - C8: 25.01 13h-16h (**3H**) → NLP applications and NN + TP6
- S5 - C9: 01.02 10h-12h (2H) → [Work on projects]
- **S6 - C10: 15.02 10h-12h (2H→ 3H) → Current challenges + project defense**

14.02: Assignments due (code + report)

Neural methods for NLP

- 1980’s: Symbolic NLP
  - rule-based approach, hand-written rules
  - advantages: based on linguistics expertise, very precise
  - inconvenients: lack of coverage, time consuming
- 1990’s: ‘Statistical’ NLP
  - learn rules automatically = (mostly linear) functions, with high-dimensional, sparse feature vectors
  - large annotated corpora
  - handcrafted features
  - rather fast to train, still good baselines
- ≃ 2010: ‘Neural’ NLP
  - combine linear and non-linear functions, over dense inputs
  - (very) large annotated corpora and very large unannotated corpora
  - improved performance (in general), no feature engineering
  - harder to interpret (“black box”)

Neural methods for NLP

- 1980’s: Symbolic NLP
  - rule-based approach, hand-written rules
  - advantages: based on linguistics expertise, very precise
  - inconvenients: lack of coverage, time consuming
- 1990’s: ‘Statistical’ NLP
  - learn rules automatically = (mostly linear) functions, with high-dimensional, sparse feature vectors
  - large annotated corpora
  - handcrafted features
  - rather fast to train, still good baselines
- **≃ 2010: ‘Neural’ NLP**
  - **combine linear and non-linear functions, over dense inputs**
  - (very) large annotated corpora and very large unannotated corpora
  - improved performance (in general), no feature engineering
  - harder to interpret (“black box”)

Neural methods for NLP

- 1980’s: Symbolic NLP
  - rule-based approach, hand-written rules
  - advantages: based on linguistics expertise, very precise
  - inconvenients: lack of coverage, time consuming
- 1990’s: ‘Statistical’ NLP
  - learn rules automatically = (mostly linear) functions, with high-dimensional, sparse feature vectors
  - large annotated corpora
  - handcrafted features
  - rather fast to train, still good baselines
- **≃ 2010: ‘Neural’ NLP**
  - **combine linear and non-linear functions, over dense inputs**
  - (very) large annotated corpora and very large unannotated corpora
  - improved performance (in general), no feature engineering
  - harder to interpret (“black box”)

scikit learn

MALLET machine learning for language toolkit

WEKA The University of Waikato

Keras

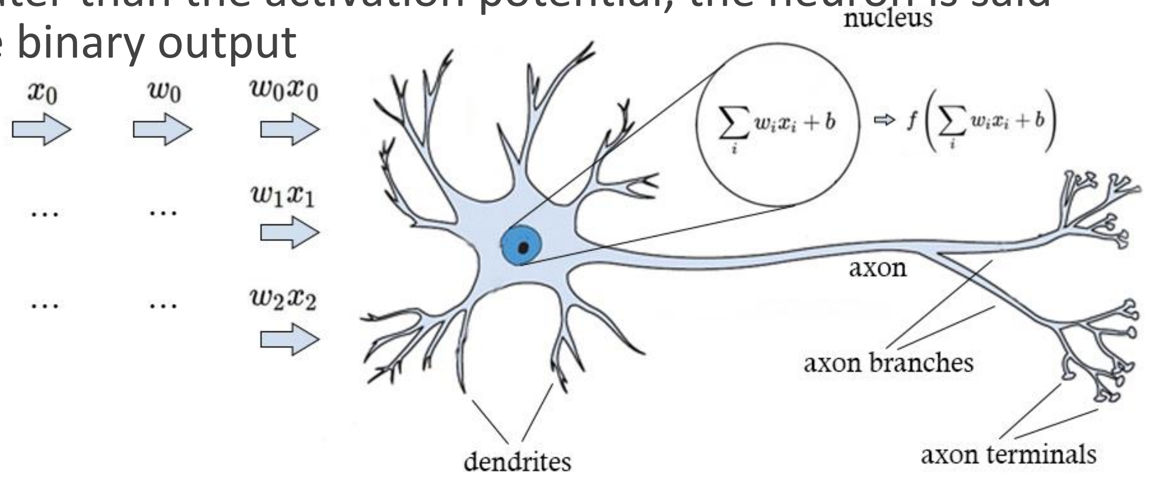
PYTORCH

TensorFlow

A bit more of history: The brain inspired metaphor

Brain’s computation is based on computation units called neurons (Perceptron, Rosenblatt, 1957):

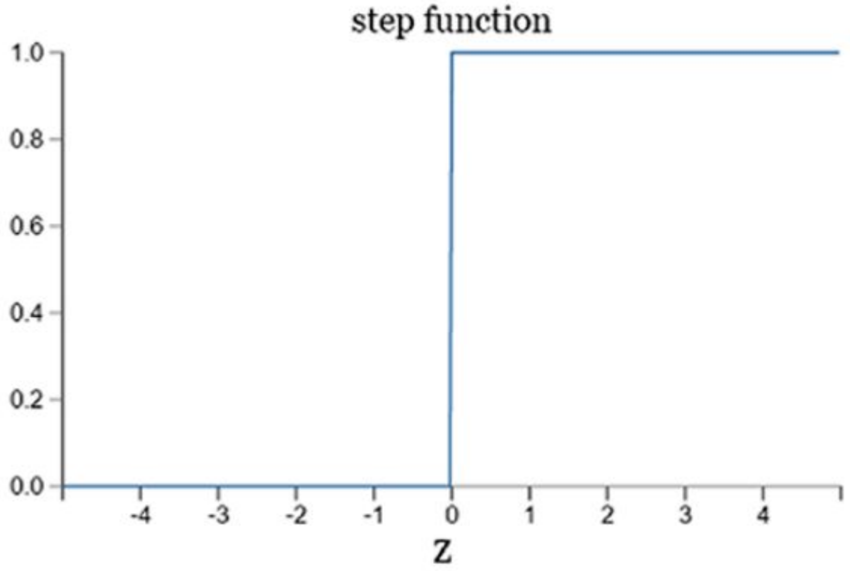
- A neuron has scalar inputs and outputs
- Each input has an associated weight to control its importance
- The neuron multiplies each input by its weight and then sums them
- If the weighted sum is greater than the activation potential, the neuron is said to “fire” = produce a single binary output
- The neurons are connected to each other, forming a network: the output of a neuron may feed into the inputs of one or more neurons

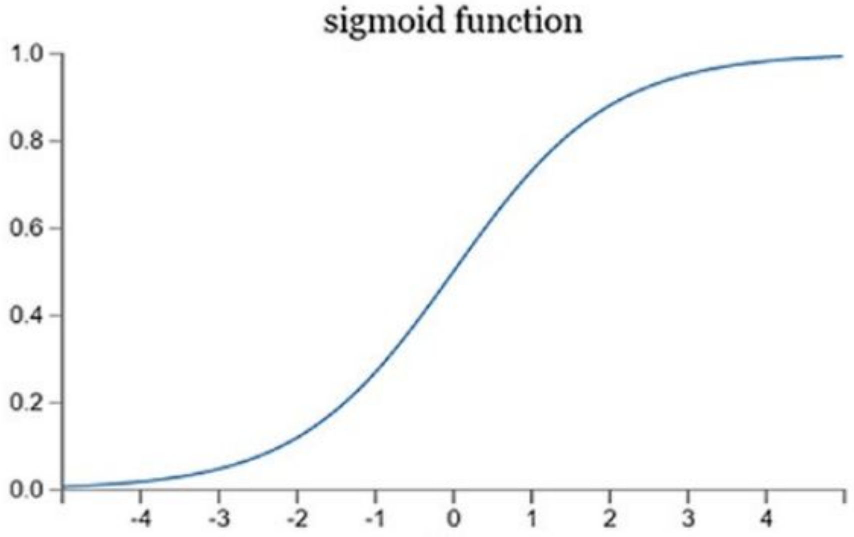


Artificial neuron

Using a binary output: not very practical

→ We prefer having small change in weight leading to small change in output





‘Statistical’ vs ‘neural’ models

Standard approach:

- **linear model** trained over high-dimensional but **very sparse** feature vectors

Neural approach:

- **non-linear** neural networks over **dense input vectors**

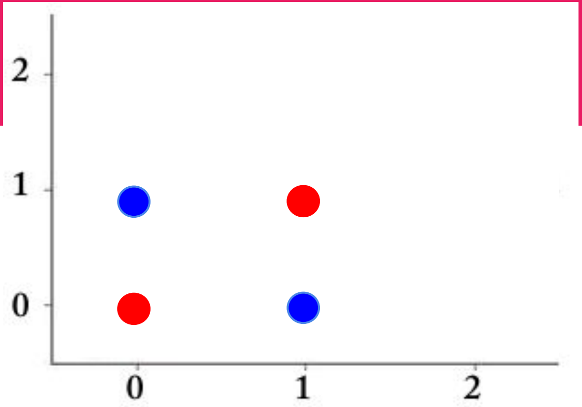
Content

Introduction to Deep Learning

1. Introducing non linearity
2. Feed-forward architecture
3. Common activation functions
4. Output Transformation functions

Practical session 2: walk through code in PyTorch

Introducing non-linearity

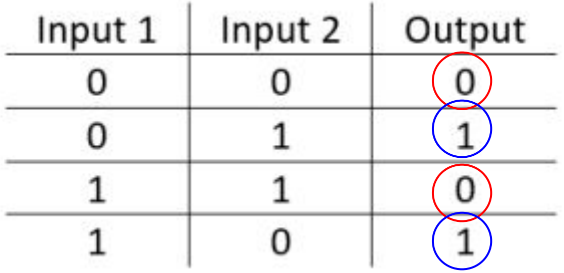


XOR problem: exclusive “or”

⟶ return a true value if the two inputs are not equal and a false value if they are equal.

But it’s impossible to find **w,** b such that:

- $(0,0).w + b < 0$
- $(0,1).w + b >= 0$
- $(1,0).w + b >= 0$
- $(1,1).w + b < 0$



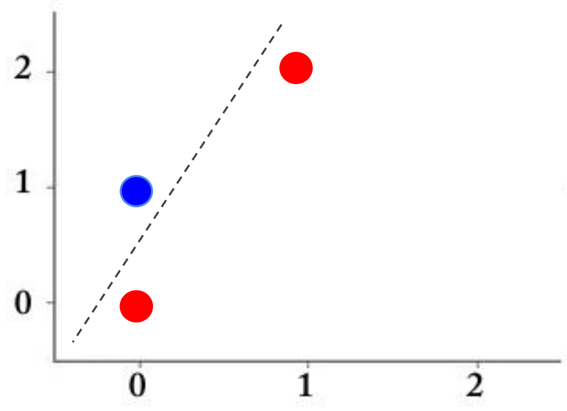
Input 1	Input 2	Output
0	0	0
0	1	1
1	1	0
1	0	1

Solution: non-linear input transformations

If we transform the points using: **ф(x1, x2) = [x1 x x2, x1 + x2]**

The problem becomes linearly separable:

- ф(0,0) = (0, 0) ⟶ **0**
- ф(0,1) = (0, 1) ⟶ **1**
- ф(1,0) = (0, 1) ⟶ **1**
- ф(1,1) = (1, 2) ⟶ **0**



The function ф mapped the data into a representation that is suitable for linear classification, we can find:

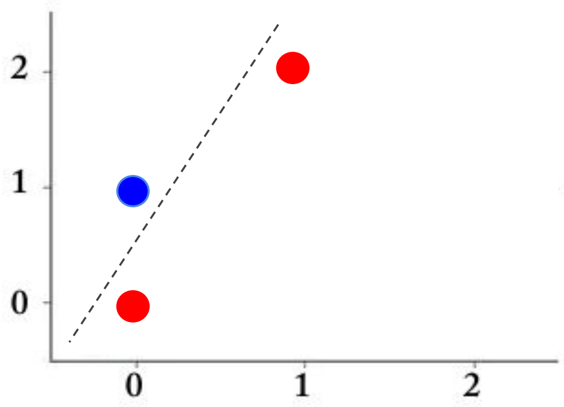
f(**x**) = ф(**x**)**.W** + **b**

Solution: non-linear input transformations

If we transform the points using: **ɸ(x1, x2) = [x1 x x2, x1 + x2]**

The problem becomes linearly separable:

- ɸ(0,0) = (0, 0) ⟶ **0**
- ɸ(0,1) = (0, 1) ⟶ **1**
- ɸ(1,0) = (0, 1) ⟶ **1**
- ɸ(1,1) = (1, 2) ⟶ **0**



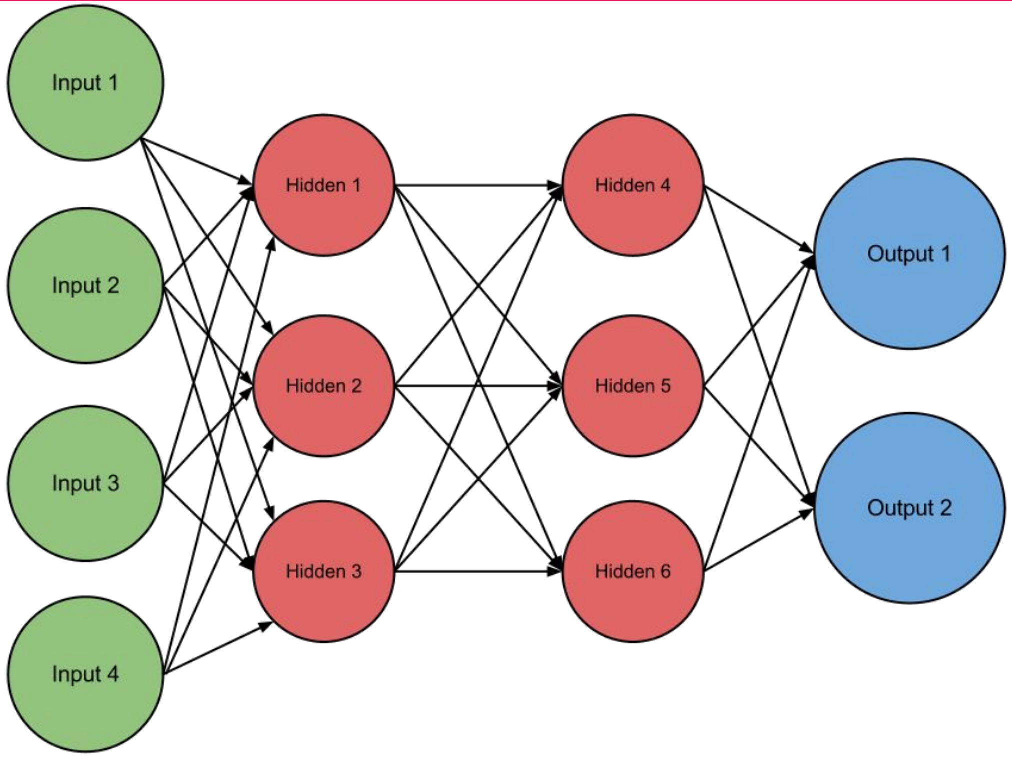
The function ɸ mapped the data into a representation that is suitable for linear classification, we can find:

f(**x**) = ɸ(**x**).**W** + **b**

Note: here the transformed data has the same dimension as the original, but often we'll need to map to a higher dimensional space.

Note: SVM = defining *a priori* generic mapping functions *vs* NN = trainable mapping functions

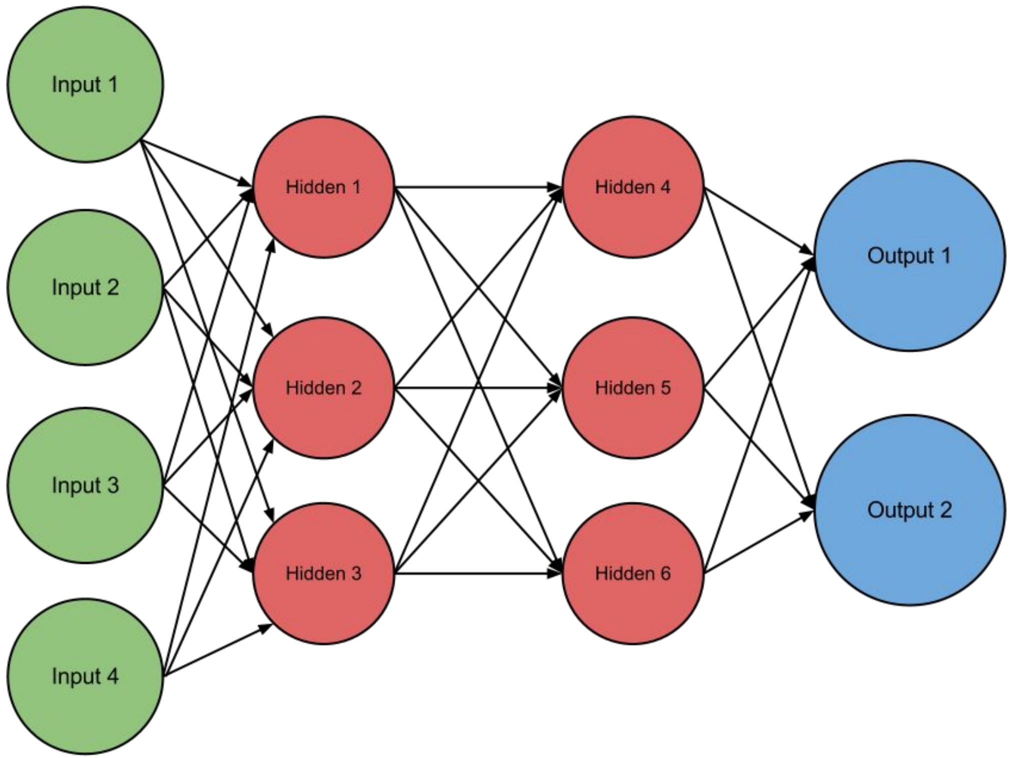
Feed-Forward Architecture



Feed-Forward Architecture

Multi-layer Perceptron

- best known, standard neural network approach
- Fully connected layers
- Can be used as drop-in replacement for typical classifiers



input layer

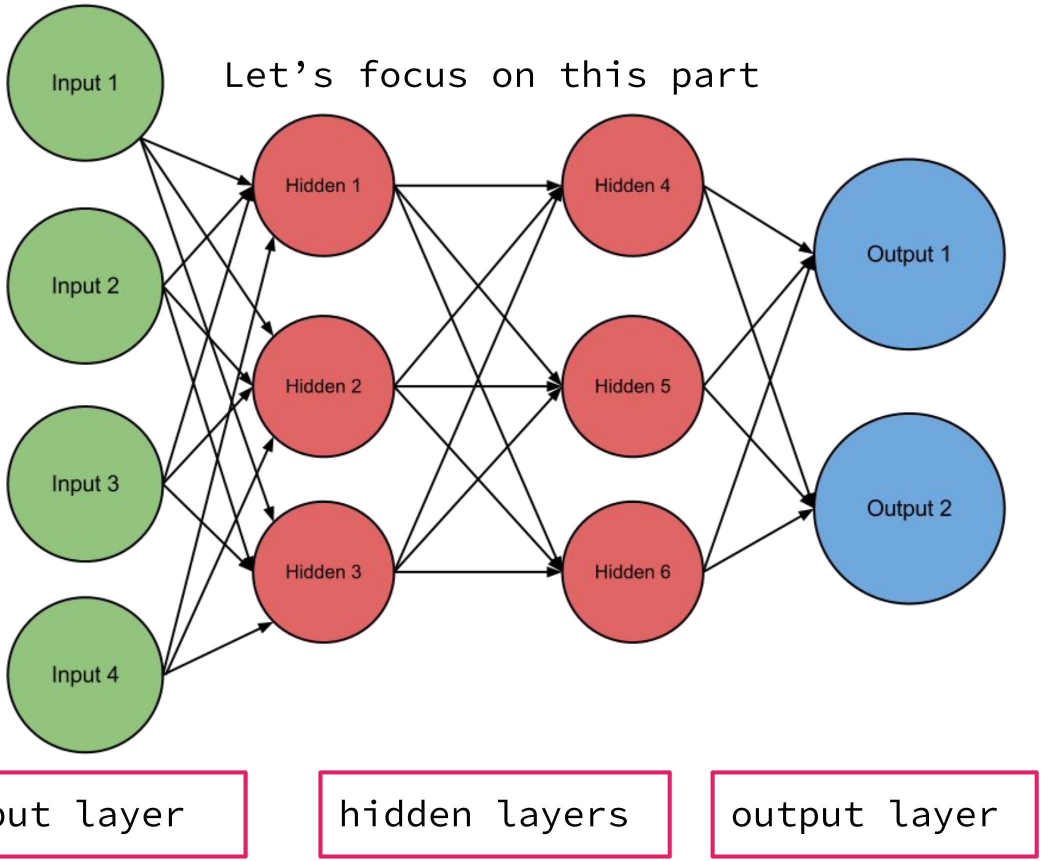
hidden layers

output layer

Feed-Forward Architecture

Multi-layer Perceptron

- best known, standard neural network approach
- Fully connected layers
- Can be used as drop-in replacement for typical classifiers



Neural networks: basics

- Layers are made of neurons = building blocks of neural networks
- A single neuron works like logistic regression, we know that!

Neural networks: basics

- Layers are made of neurons = building blocks of neural networks
- A **single neuron works like logistic regression**, we know that!

Reminder: LR model yields **probability** of an instance belonging to a particular class **based on the instance’s features weighted by the model’s parameters**

Logistic Regression

- Spam (binary) classification
- We take word frequency as features

feature f_i	weight w_i
pharmacie	0.4
viagra	1.2
meilleure	0.2
offre	0.2
demande	-0.8
transmets	-1.7
bias	0.1

Logistic Regression

- Spam (binary) classification
- We take word frequency as features

x_1: "Pharmacie en ligne: **viagra meilleure offre**!

score($\mathbf{x_1}$) = 0.4×1+1.2×1+0.2×1+0.2×1+(−0.8)×0+(−1.7)×0+0.1 = 2.1

feature f_i	weight w_i
pharmacie	0.4
viagra	1.2
meilleure	0.2
offre	0.2
demande	-0.8
transmets	-1.7
bias	0.1

Logistic Regression

- Spam (binary) classification
- We take word frequency as features

x_1: "**Pharmacie** en ligne: **viagra meilleure offre**!

```
score(x₁) =  0.4×1+1.2×1+0.2×1+0.2×1+(−0.8)×0+(−1.7)×0+0.1 = 2.1
```

feature f_i	weight w_i
pharmacie	**0.4**
viagra	1.2
meilleure	0.2
offre	0.2
demande	-0.8
transmets	-1.7
bias	0.1

Logistic Regression

- Spam (binary) classification
- We take word frequency as features

$\mathbf{x}_1$: "**Pharmacie** en ligne: **viagra meilleure offre**!

```
score(x₁) =  0.4×1+1.2×1+0.2×1+0.2×1+(-0.8)×0+(-1.7)
×0+0.1 = 2.1
```

feature f_i	weight w_i
pharmacie	**0.4**
viagra	1.2
meilleure	0.2
offre	0.2
demande	**-0.8**
transmets	-1.7
bias	0.1

Logistic Regression

- Spam (binary) classification
- We take word frequency as features

feature f_i	weight w_i
pharmacie	**0.4**
viagra	1.2
meilleure	0.2
offre	0.2
demande	**-0.8**
transmets	-1.7
bias	0.1

$\mathbf{x}_1$: "**Pharmacie** en ligne: **viagra meilleure offre**!"

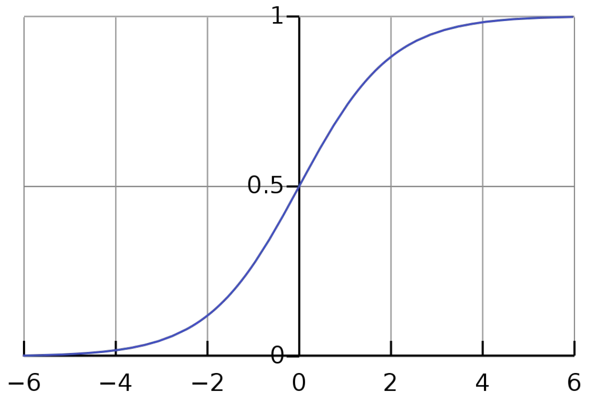
$\text{score}(\mathbf{x}_1) = \mathbf{0.4\times1}+1.2\times1+0.2\times1+0.2\times1+\mathbf{(-0.8)\times0}+(-1.7)\times0+0.1 = 2.1$

$\mathbf{x}_2$: "Suite à votre demande, je vous transmets notre meilleure offre"

$score(\mathbf{x}_2) = 0.4\times0+1.2\times0+0.2\times1+0.2\times1+(-0.8)\times1+(-1.7)\times1+0.1 = -2.0$

Logistic Regression

- Linear scores (range **−**∞ to ∞) difficult to interpret, we'd rather have **probabilities**
- Linear scores are transformed using **non-linear** *logistic function* → range [0,1]



$$f(score(x1)) = \frac{1}{1+e^{-2.1}} = .89$$

$$f(score(x2)) = \frac{1}{1+e^{2.0}} = .12$$

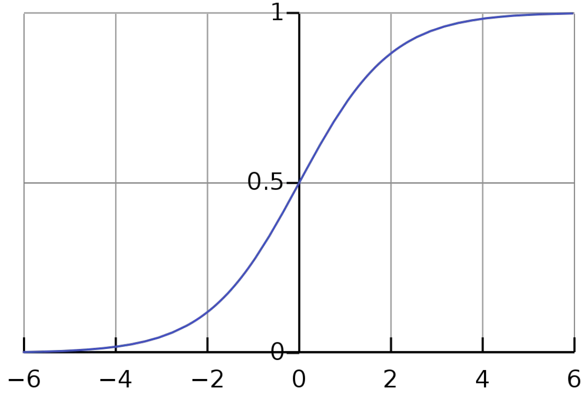
logistic function $(\frac{1}{1+e^{-x}})$

Input number ⟶ [0, 1]

- Large negative number ⟶0
- Large positive number ⟶ 1

Logistic Regression

- Linear scores (range $-\infty$ to ∞) difficult to interpret, we'd rather have **probabilities**
- Linear scores are transformed using **non-linear** *logistic function* → range [0,1]



logistic function $\left(\frac{1}{1+e^{-x}}\right)$

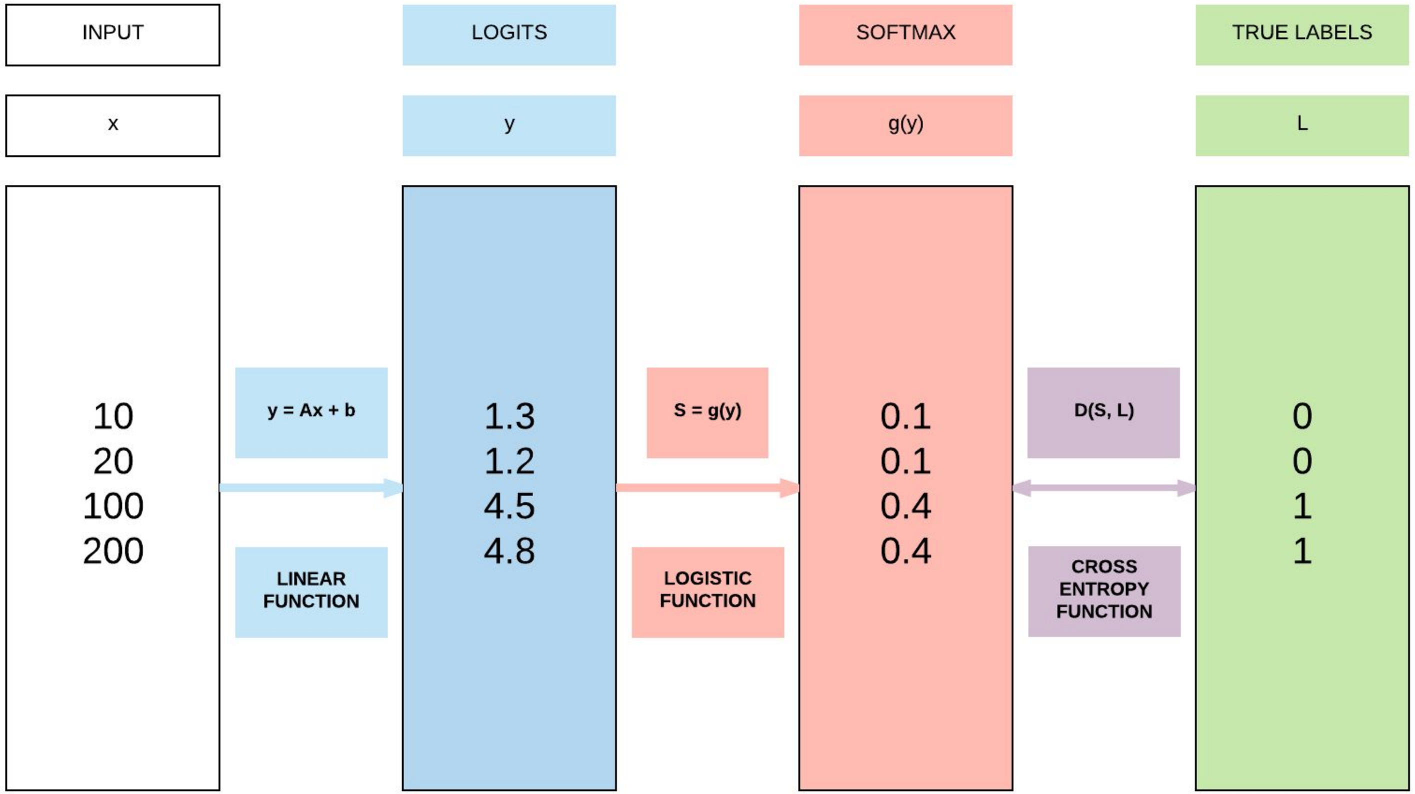
$$f(score(x1)) = \frac{1}{1+e^{-2.1}} = .89$$
$$f(score(x2)) = \frac{1}{1+e^{2.0}} = .12$$

x_1: **"Pharmacie** en ligne: **viagra meilleure offre**!"

Input number → [0, 1]

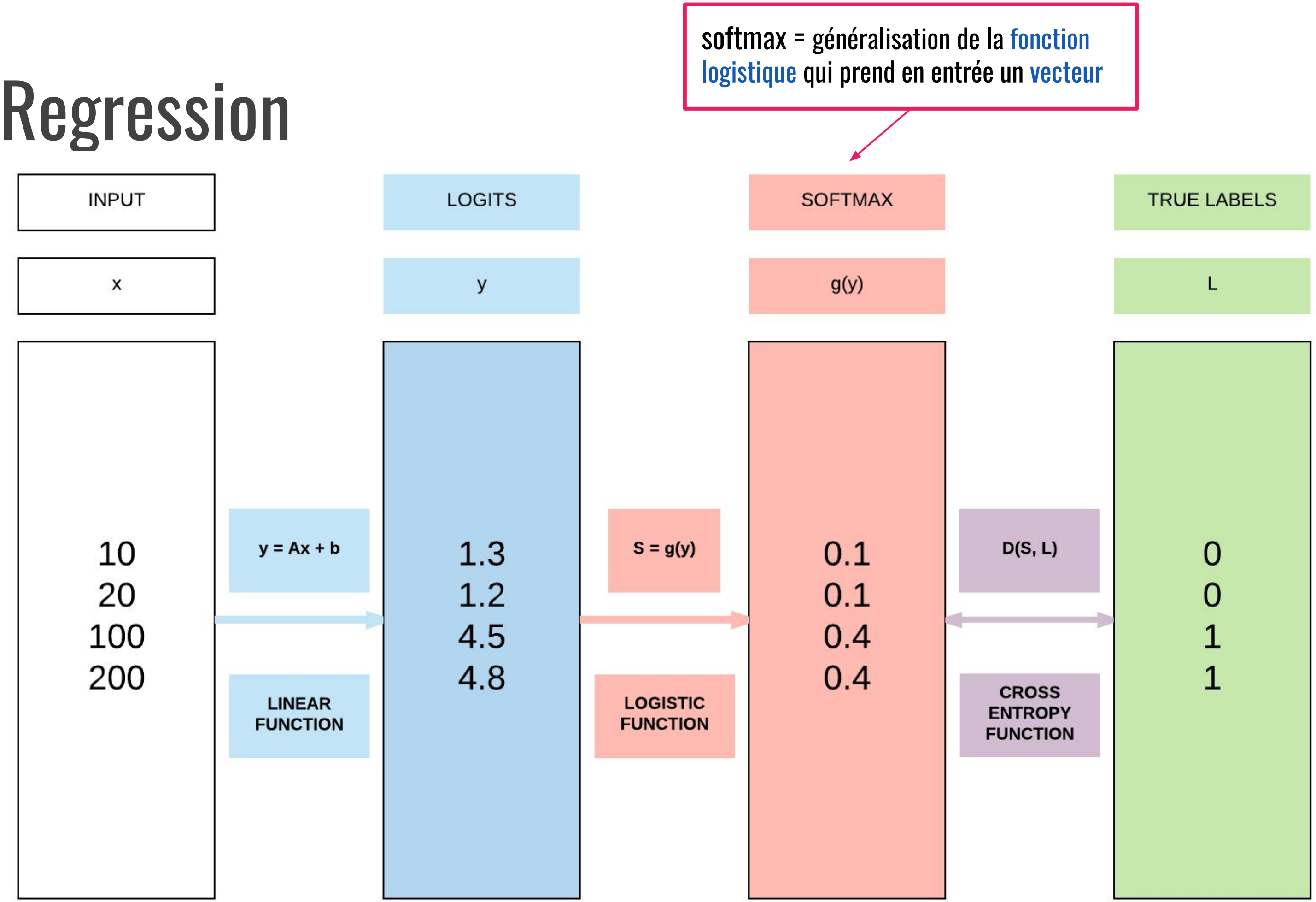
- Large negative number →0
- Large positive number → 1

Logistic Regression



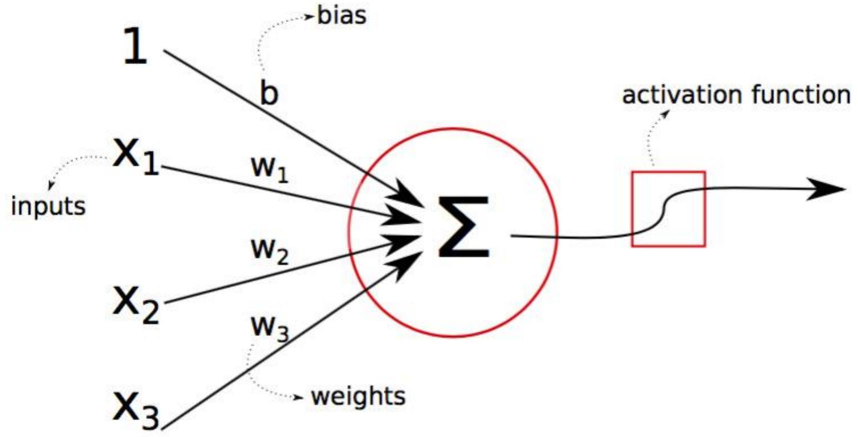
Logistic Regression

softmax = généralisation de la fonction logistique qui prend en entrée un vecteur



Neuron = Logistic Regression

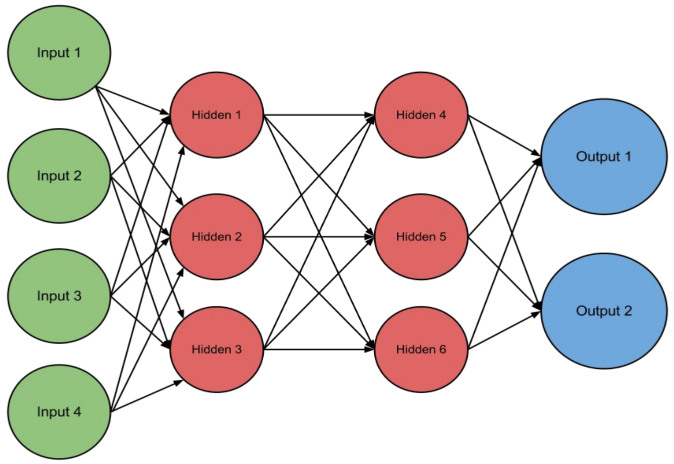
These are the exact computations of a single neuron



- We can feed an input vector to a bunch of LR functions and get an output vector
- which can be fed to another layer of LR functions

Feed-forward architecture

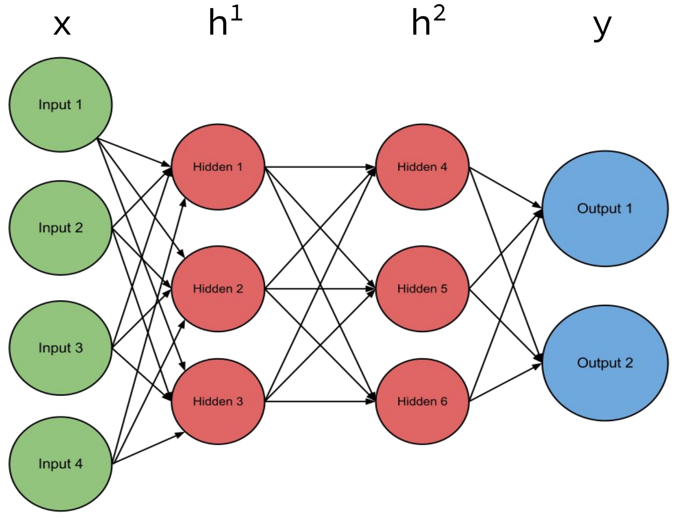
Multi-layer perceptron with 2 hidden layers



`input layer` `hidden layers` `output layer`

Feed-forward architecture

Multi-layer perceptron with 2 hidden layers



`input layer` `hidden layers` `output layer`

$$NN_{MLP2}(\mathbf{x}) = \mathbf{y} \quad (1)$$

$$\mathbf{h}^1 = g(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1) \quad (2)$$

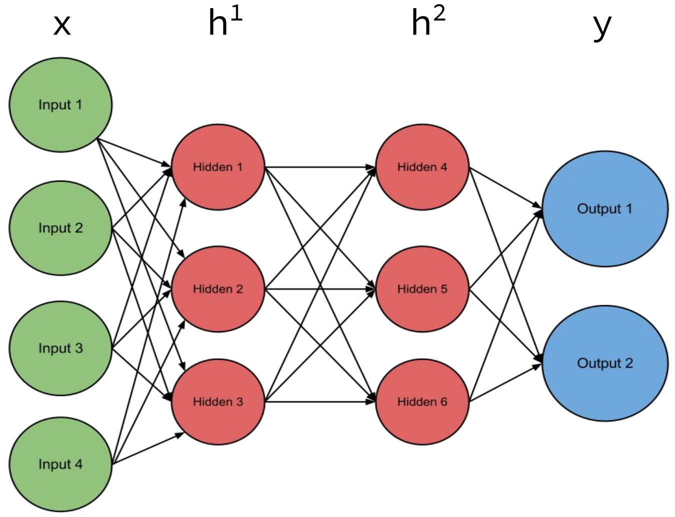
$$\mathbf{h}^2 = g(\mathbf{h}^1\mathbf{W}^2 + \mathbf{b}^2) \quad (3)$$

$$\mathbf{y} = \mathbf{h}^2\mathbf{W}^3 \quad (4)$$

$\mathbf{x}$: vector of size $d_{in} = 4$
$\mathbf{y}$: vector of size $d_{out} = 2$
$\mathbf{h}^1, \mathbf{h}^2$: vectors of size $d_{hidden} = 3$

Feed-forward architecture

Multi-layer perceptron with 2 hidden layers



input layer

hidden layers

output layer

$$NN_{MLP2}(\mathbf{x}) = \mathbf{y} \quad (1)$$

$$\mathbf{h}^1 = g(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1) \quad (2)$$

$$\mathbf{h}^2 = g(\mathbf{h}^1\mathbf{W}^2 + \mathbf{b}^2) \quad (3)$$

$$\mathbf{y} = \mathbf{h}^2\mathbf{W}^3 \quad (4)$$

$\mathbf{x}$: vector of size $d_{in} = 4$
$\mathbf{y}$: vector of size $d_{out} = 2$
$\mathbf{h}^1, \mathbf{h}^2$: vectors of size $d_{hidden} = 3$

$\mathbf{W}^1, \mathbf{W}^2, \mathbf{W}^3$: matrices of size $[4 \times 3], [3 \times 3], [3 \times 2]$
$\mathbf{b}^1, \mathbf{b}^2$: 'bias' vectors of size $d_{hidden} = 4$
$g(\cdot)$: non-linear activation function (elementwise)

Feed-forward architecture

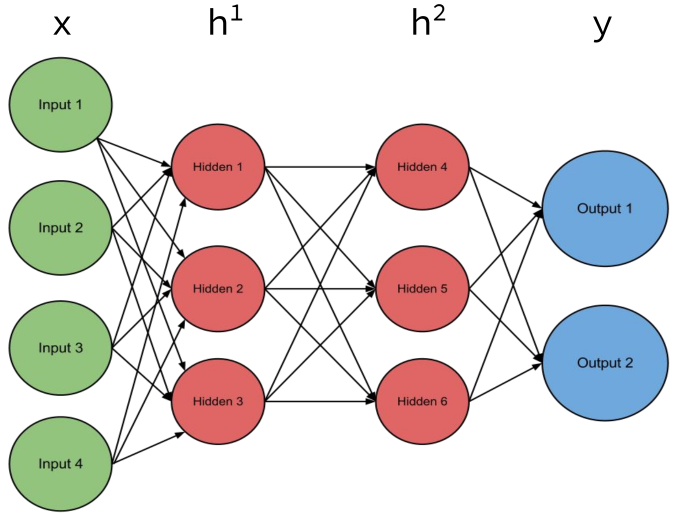
Multi-layer perceptron with 2 hidden layers

$$NN_{MLP2}(\mathbf{x}) = \mathbf{y} \quad (1)$$

$$\mathbf{h}^1 = g(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1) \quad (2)$$

$$\mathbf{h}^2 = g(\mathbf{h}^1\mathbf{W}^2 + \mathbf{b}^2) \quad (3)$$

$$\mathbf{y} = \mathbf{h}^2\mathbf{W}^3 \quad (4)$$



$\mathbf{x}$: vector of size $d_{in} = 4$
$\mathbf{y}$: vector of size $d_{out} = 2$
$\mathbf{h}^1, \mathbf{h}^2$: vectors of size $d_{hidden} = 3$

```
Parameters of the network (θ)
```

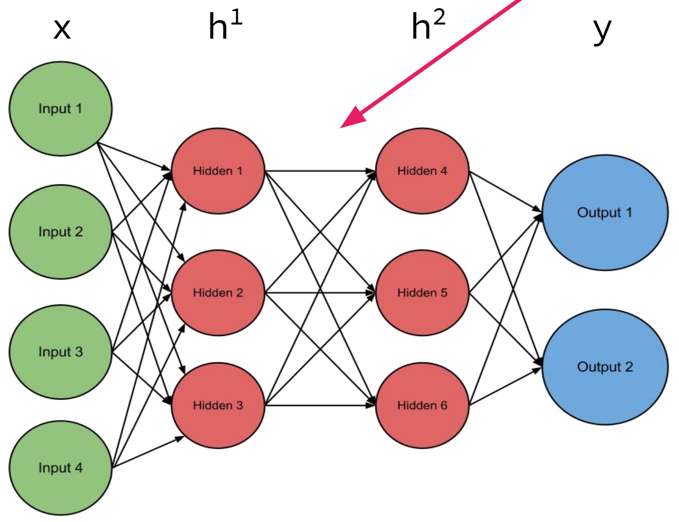
$\mathbf{W}^1, \mathbf{W}^2, \mathbf{W}^3$: matrices of size $[4 \times 3], [3 \times 3], [3 \times 2]$
$\mathbf{b}^1, \mathbf{b}^2$: 'bias' vectors of size $d_{hidden} = 4$
$g(\cdot)$: non-linear activation function (elementwise)

```
input layer
```
```
hidden layers
```
```
output layer
```

Feed-forward architecture

Multi-layer perceptron with 2 hidden layers

Deep learning



input layer

hidden layers

output layer

$$NN_{MLP2}(\mathbf{x}) = \mathbf{y} \quad (1)$$

$$\mathbf{h}^1 = g(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1) \quad (2)$$

$$\mathbf{h}^2 = g(\mathbf{h}^1\mathbf{W}^2 + \mathbf{b}^2) \quad (3)$$

$$\mathbf{y} = \mathbf{h}^2\mathbf{W}^3 \quad (4)$$

$\mathbf{x}$: vector of size $d_{in} = 4$
$\mathbf{y}$: vector of size $d_{out} = 2$
$\mathbf{h}^1, \mathbf{h}^2$: vectors of size $d_{hidden} = 3$

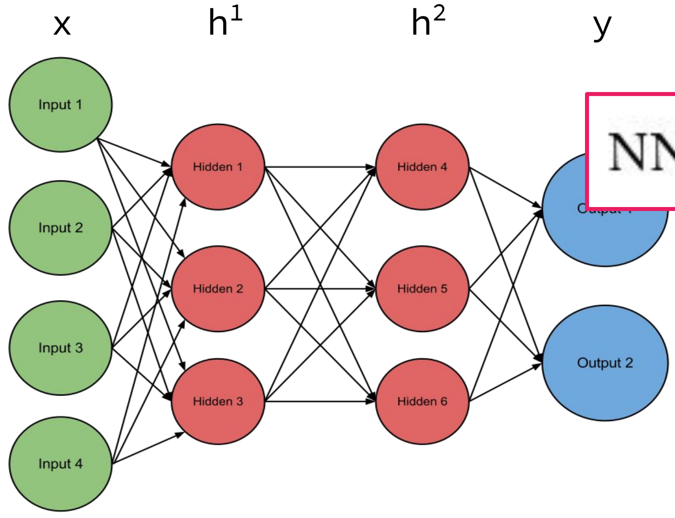
Parameters of the network (θ)

$\mathbf{W}^1, \mathbf{W}^2, \mathbf{W}^3$: matrices of size $[4 \times 3]$, $[3 \times 3]$, $[3 \times 2]$
$\mathbf{b}^1, \mathbf{b}^2$: 'bias' vectors of size $d_{hidden} = 4$
$g(\cdot)$: non-linear activation function (elementwise)

Feed-forward architecture

Multi-layer perceptron with 2 hidden layers

$$NN_{MLP2}(\mathbf{x}) = \mathbf{y} \quad (1)$$
$$\mathbf{h}^1 = g(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1) \quad (2)$$
$$\mathbf{h}^2 = g(\mathbf{h}^1\mathbf{W}^2 + \mathbf{b}^2) \quad (3)$$
$$\mathbf{y} = \mathbf{h}^2\mathbf{W}^3 \quad (4)$$



$$\boxed{\mathrm{NN}_{\mathrm{MLP2}}(\boldsymbol{x}) = (g^2(g^1(\boldsymbol{x}\boldsymbol{W}^1 + \boldsymbol{b}^1)\boldsymbol{W}^2 + \boldsymbol{b}^2))\boldsymbol{W}^3}$$

input layer

hidden layers

output layer

Linear algebra (what? why?)

***Linear algebra** is the branch of mathematics concerning **linear** equations and **linear** functions and their representations through matrices and vector spaces.* (Wikipedia)

→ "Under the hood, the feed forward neural network is just **a composite function, that multiplies some matrices and vectors together**. It is not that vectors and matrices are the only way to do these operations but they become highly **efficient** if you do so. (..) neural networks are computationally expensive, so they require this nice trick to make them compute faster. **It's called vectorization. They make computations extremely faster. This is one of the main reasons *why GPUs are required for deep learning*, as they are specialized in vectorized operations like matrix multiplication.**"

https://towardsdatascience.com/linear-algebra-explained-in-the-context-of-deep-learning-8fcb8fca1494

Linear algebra

- Scalar: A single number (rank 0 tensor)
- Vector : A list of values (rank 1 tensor)
- Matrix: A two dimensional list of values (rank 2 tensor)
- Tensor: A multi dimensional matrix with rank n.

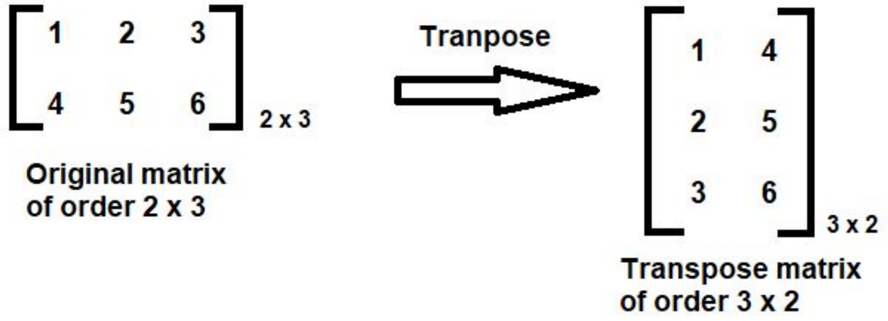
Operations:

- Transpose of a matrix: mirror image of the matrix across the diagonal line (from top left to the bottom right of the matrix)
- Dot product: between 2 vectors, return a scalar
- Dimension / shape of a matrix: row by column
- Norm is the size of the vector, e.g. L2 = $\sqrt{\sum(x_i)^2}$
- Matrix multiplication

https://ml-cheatsheet.readthedocs.io/en/latest/linear_algebra.html

Linear algebra

- Scalar: A single number (rank 0 tensor)
- Vector : A list of values (rank 1 tensor)
- Matrix: A two dimensional list of values (rank 2 ten
- Tensor: A multi dimensional matrix with rank n.



1 2 3
4 5 6
2 x 3
Original matrix of order 2 x 3
Tranpose
1 4
2 5
3 6
3 x 2
Transpose matrix of order 3 x 2

Operations:

- **Transpose of a matrix**: mirror image of the matrix across the diagonal line (from top left to the bottom right of the matrix)
- Dot product: between 2 vectors, return a scalar
- Dimension / shape of a matrix: row by column
- Norm is the size of the vector, e.g. L2 = $\sqrt{\sum(x_i)^2}$
- Matrix multiplication

https://ml-cheatsheet.readthedocs.io/en/latest/linear_algebra.html

Linear algebra

- Scalar: A single number (rank 0 tensor)
- Vector : A list of values (rank 1 tensor)
- Matrix: A two dimensional list of values (rank 2 tensor)
- Tensor: A multi dimensional matrix with rank n.

$$\begin{bmatrix} a & b \end{bmatrix} \bullet \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + by \end{bmatrix}$$

Operations:

- Transpose of a matrix: mirror image of the matrix across the diagonal line (from top left to the bottom right of the matrix)
- **Dot product**: between 2 vectors, return a scalar
- Dimension / shape of a matrix: row by column
- Norm is the size of the vector, e.g. L2 = $\sqrt{\sum(x_i)^2}$
- Matrix multiplication

https://ml-cheatsheet.readthedocs.io/en/latest/linear_algebra.html

Linear algebra

- Scalar: A single number (rank 0 tensor)
- Vector : A list of values (rank 1 tensor)
- Matrix: A two dimensional list of values (rank 2 tensor)
- Tensor: A multi dimensional matrix with rank n.

$$\underset{2\times3}{\begin{bmatrix} 3 & 1 & 4 \\ 2 & 5 & 0 \end{bmatrix}} \quad \underset{4\times2}{\begin{bmatrix} 1 & 3 \\ 2 & 8 \\ 0 & 4 \\ 5 & 6 \end{bmatrix}} \quad \underset{1\times3}{\begin{bmatrix} 1.6 & 0.2 & 1.0 \end{bmatrix}}$$

$$\underset{2\times2}{\begin{bmatrix} 2 & 0 \\ 0 & 3.5 \end{bmatrix}} \quad \underset{2\times1}{\begin{bmatrix} 5 \\ 3 \end{bmatrix}} \quad \underset{3\times3}{\begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 40 & 20 & 1 \end{bmatrix}}$$

Operations:

- Transpose of a matrix: mirror image of the matrix across the diagonal line (from top left to the bottom right of the matrix)
- Dot product: between 2 vectors, return a scalar
- **Dimension / shape of a matrix**: row by column
- Norm is the size of the vector, e.g. L2 = $\sqrt{\sum (x_i)^2}$
- Matrix multiplication

https://ml-cheatsheet.readthedocs.io/en/latest/linear_algebra.html

Linear algebra

- Scalar: A single number (rank 0 tensor)
- Vector : A list of values (rank 1 tensor)
- Matrix: A two dimensional list of values (rank 2 tensor)
- Tensor: A multi dimensional matrix with rank n.

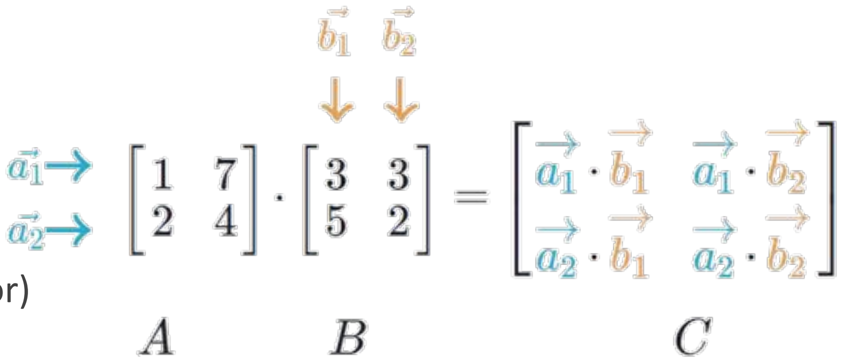
Operations:

- Transpose of a matrix: mirror image of the matrix across the diagonal line (from top left to the bottom right of the matrix)
- Dot product: between 2 vectors, return a scalar
- Dimension / shape of a matrix: row by column
- **Norm is the size of the vector, e.g. L2 = $\sqrt{\sum(x_i)^2}$**
- Matrix multiplication

https://ml-cheatsheet.readthedocs.io/en/latest/linear_algebra.html

Linear algebra

- Scalar: A single number (rank 0 tensor)
- Vector : A list of values (rank 1 tensor)
- Matrix: A two dimensional list of values (rank 2 tensor)
- Tensor: A multi dimensional matrix with rank n.

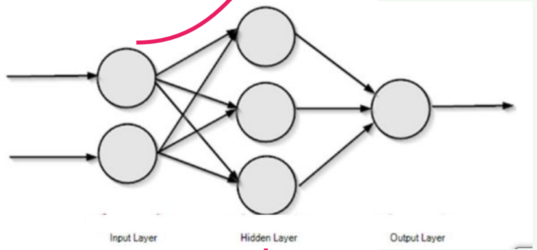


$$\begin{array}{cccccc} & & & \vec{b_1}\downarrow & \vec{b_2}\downarrow & \\ \vec{a_1}\rightarrow & \begin{bmatrix}1 & 7\\2 & 4\end{bmatrix} & \cdot & \begin{bmatrix}3 & 3\\5 & 2\end{bmatrix} & = & \begin{bmatrix}\vec{a_1}\cdot\vec{b_1} & \vec{a_1}\cdot\vec{b_2}\\ \vec{a_2}\cdot\vec{b_1} & \vec{a_2}\cdot\vec{b_2}\end{bmatrix} \\ \vec{a_2}\rightarrow & A & & B & & C \end{array}$$

Operations:

- Transpose of a matrix: mirror image of the matrix across the diagonal line (from top left to the bottom right of the matrix)
- Dot product: between 2 vectors, return a scalar
- Dimension / shape of a matrix: row by column
- Norm is the size of the vector, e.g. L2 = $\sqrt{\sum (x_i)^2}$
- **Matrix multiplication**:
  - The number of columns of the 1st matrix must equal the number of rows of the 2nd
  - The product of an M x N matrix and an N x K matrix is an M x K matrix. The new matrix takes the rows of the 1st and columns of the 2nd

Linear algebra



$$X = \begin{bmatrix} x1 \\ x2 \end{bmatrix} \text{ vector (input layer)}$$

$$W = \begin{bmatrix} w1 & w2 \\ w4 & w5 \\ x3 & w6 \end{bmatrix} \text{ matrix (the weights for hidden layer 1)}$$

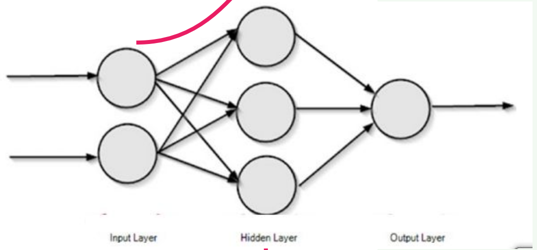
the output is given by

- 1 input layer **x** ; 1 hidden layer **H**
- weight matrix **W** ; bias scalar **b**
- In a computer, the layers of the neural network are represented as vectors/matrices(/tensors)
- $\mathbf{H = f(x.W + b)}$

$\boldsymbol{W.x}$; this is a **matrix-vector product**

https://towardsdatascience.com/linear-algebra-explained-in-the-context-of-deep-learning-8fcb8fca1494

Linear algebra



$$X = \begin{bmatrix} x1 \\ x2 \end{bmatrix} \quad \textit{vector (input layer)}$$

$$W = \begin{bmatrix} w1 & w2 \\ w4 & w5 \\ x3 & w6 \end{bmatrix} \quad \textit{matrix (the weights for hidden layer 1)}$$

the output is given by

- 1 input layer **x** ; 1 hidden layer **H**
- weight matrix **W** ; bias scalar **b**
- In a computer, the layers of the neural network are represented as vectors/matrices(/tensors)
- **H = f(x.W + b)**

W.x ; this is a **matrix-vector product**

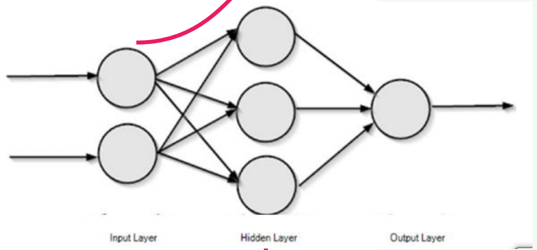
Shape of W.x?

Shape of W.x + b?

Shape of H?

https://towardsdatascience.com/linear-algebra-explained-in-the-context-of-deep-learning-8fcb8fca1494

Linear algebra



$$X = \begin{bmatrix} x1 \\ x2 \end{bmatrix} \quad \textit{vector (input layer)}$$

$$W = \begin{bmatrix} w1 & w2 \\ w4 & w5 \\ x3 & w6 \end{bmatrix} \quad \textit{matrix (the weights for hidden layer 1)}$$

the output is given by

$$\begin{bmatrix} h1 \\ h2 \\ h3 \end{bmatrix} = \begin{bmatrix} w1 & w2 \\ w4 & w5 \\ x3 & w6 \end{bmatrix} \cdot \begin{bmatrix} x1 \\ x2 \end{bmatrix} \quad \textit{(the product of vector and matrices)}$$

- 1 input layer **x** ; 1 hidden layer **H**
- weight matrix **W** ; bias scalar **b**
- In a computer, the layers of the neural network are represented as vectors/matrices(/tensors)
- **H = f(x.W + b)**

this is done by taking each row of the first matrix and doing elemnt wise multiplication with each column of the second matrix. thus,

$$\begin{aligned} h1 &= w1.x1 + w2.x2 \\ h2 &= w3.x1 + w3.x2 \\ h3 &= w5.x1 + w6.x2 \end{aligned}$$

W.x ; this is a **matrix-vector product**

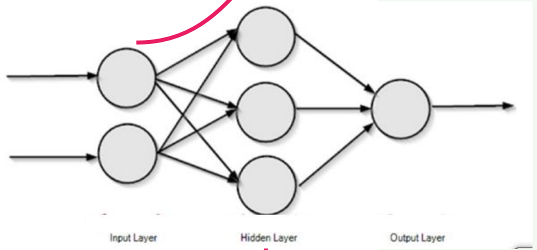
Shape of W.x?

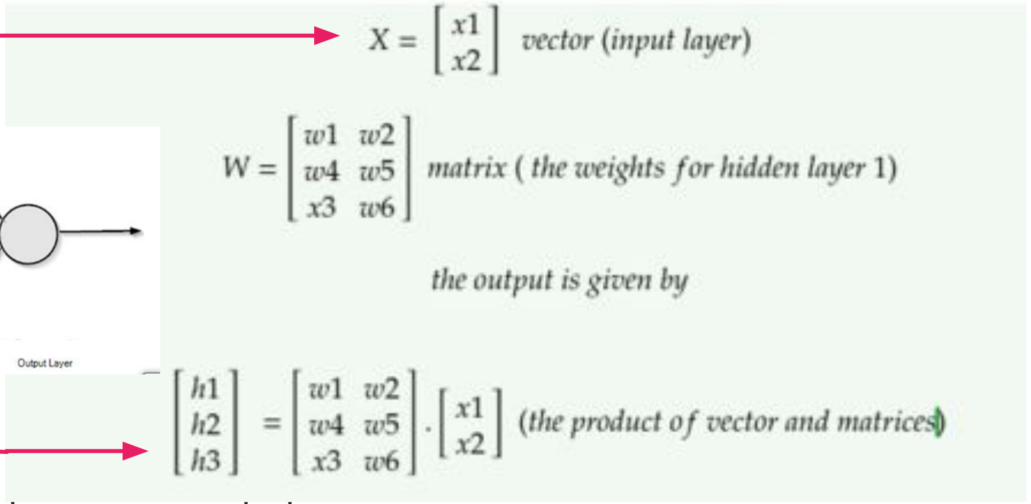
Shape of W.x + b?

Shape of H?

https://towardsdatascience.com/linear-algebra-explained-in-the-context-of-deep-learning-8fcb8fca1494

Linear algebra





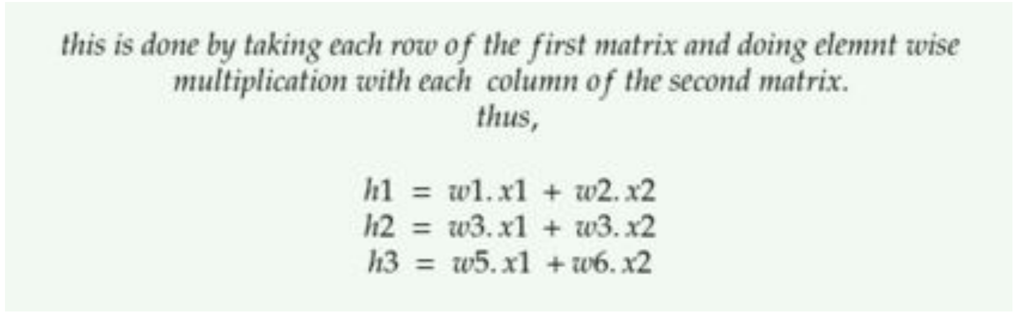
$$X = \begin{bmatrix} x1 \\ x2 \end{bmatrix}$$ *vector (input layer)*

$$W = \begin{bmatrix} w1 & w2 \\ w4 & w5 \\ x3 & w6 \end{bmatrix}$$ *matrix (the weights for hidden layer 1)*

the output is given by

$$\begin{bmatrix} h1 \\ h2 \\ h3 \end{bmatrix} = \begin{bmatrix} w1 & w2 \\ w4 & w5 \\ x3 & w6 \end{bmatrix} \cdot \begin{bmatrix} x1 \\ x2 \end{bmatrix}$$ *(the product of vector and matrices)*

- 1 input layer **x** ; 1 hidden layer **H**
- weight matrix **W** ; bias scalar **b**
- In a computer, the layers of the neural network are represented as vectors/matrices(/tensors)
- **H = f(x.W + b)**



this is done by taking each row of the first matrix and doing elemnt wise multiplication with each column of the second matrix. thus,

$$\begin{aligned} h1 &= w1.x1 + w2.x2 \\ h2 &= w3.x1 + w3.x2 \\ h3 &= w5.x1 + w6.x2 \end{aligned}$$

W.x ; this is a **matrix-vector product**

Shape of W.x?

Shape of W.x + b?

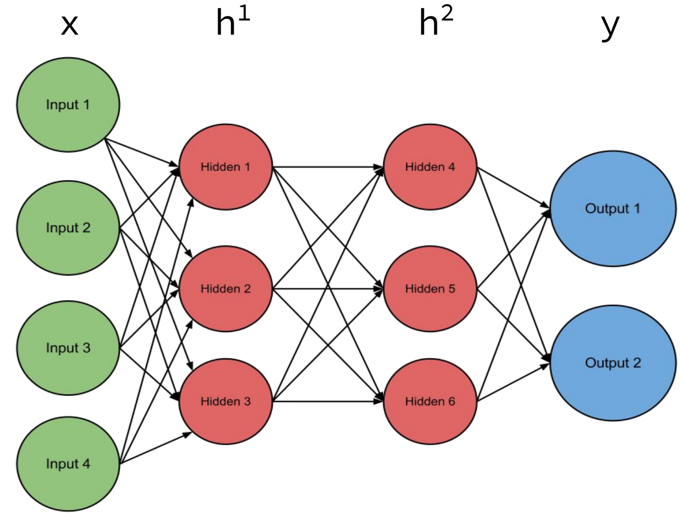
3x1

Shape of H?

https://towardsdatascience.com/linear-algebra-explained-in-the-context-of-deep-learning-8fcb8fca1494

Feed-forward architecture

Multi-layer perceptron with 2 hidden layers



input layer | hidden layers | output layer

$$NN_{MLP2}(\mathbf{x}) = \mathbf{y} \quad (1)$$

$$\mathbf{h}^1 = g(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1) \quad (2)$$

$$\mathbf{h}^2 = g(\mathbf{h}^1\mathbf{W}^2 + \mathbf{b}^2) \quad (3)$$

$$\mathbf{y} = \mathbf{h}^2\mathbf{W}^3 \quad (4)$$

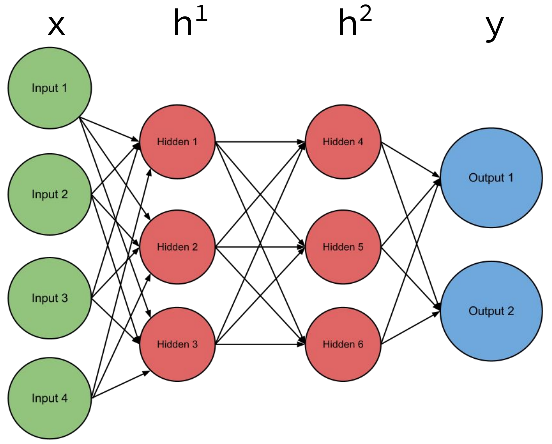
$\mathbf{x}$: vector of size $d_{in} = 4$
$\mathbf{y}$: vector of size $d_{out} = 2$
$\mathbf{h}^1, \mathbf{h}^2$: vectors of size $d_{hidden} = 3$

```
h1 -> x.W1 = (1xd_in).(d_in xd_h1) -> (1x3)
h2 -> h1.W2 = (1xd_h1).(d_h1 xd_h2) -> (1x3)
y  -> h2.W3 = (1xd_h2).(d_h2 xd_out) -> (1x2)
```

$\mathbf{W}^1, \mathbf{W}^2, \mathbf{W}^3$: matrices of size $[4 \times 3], [3 \times 3], [3 \times 2]$
$\mathbf{b}^1, \mathbf{b}^2$: 'bias' vectors of size $d_{hidden} = 4$
$g(\cdot)$: non-linear activation function (elementwise)

Feed-forward architecture

Multi-layer perceptron with 2 hidden layers



input layer

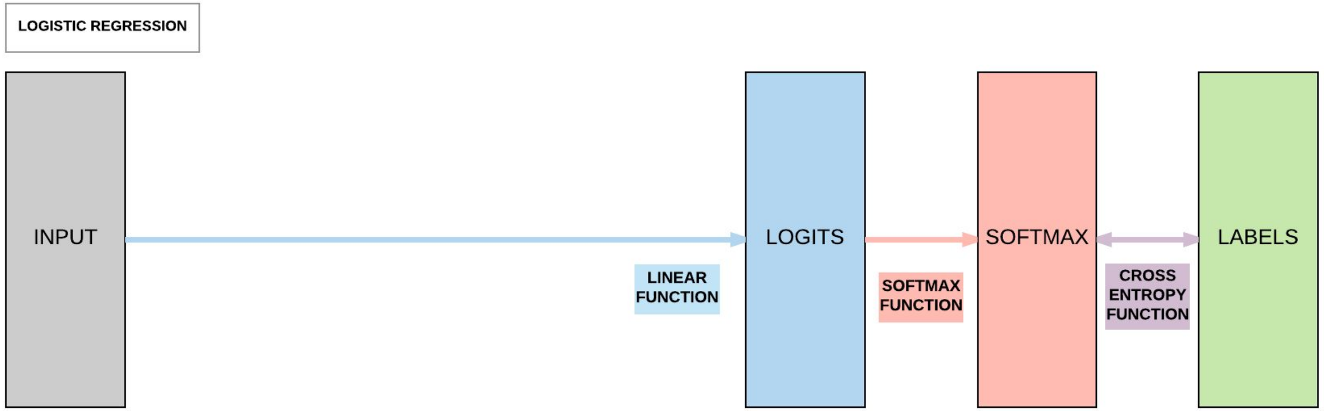
hidden layers

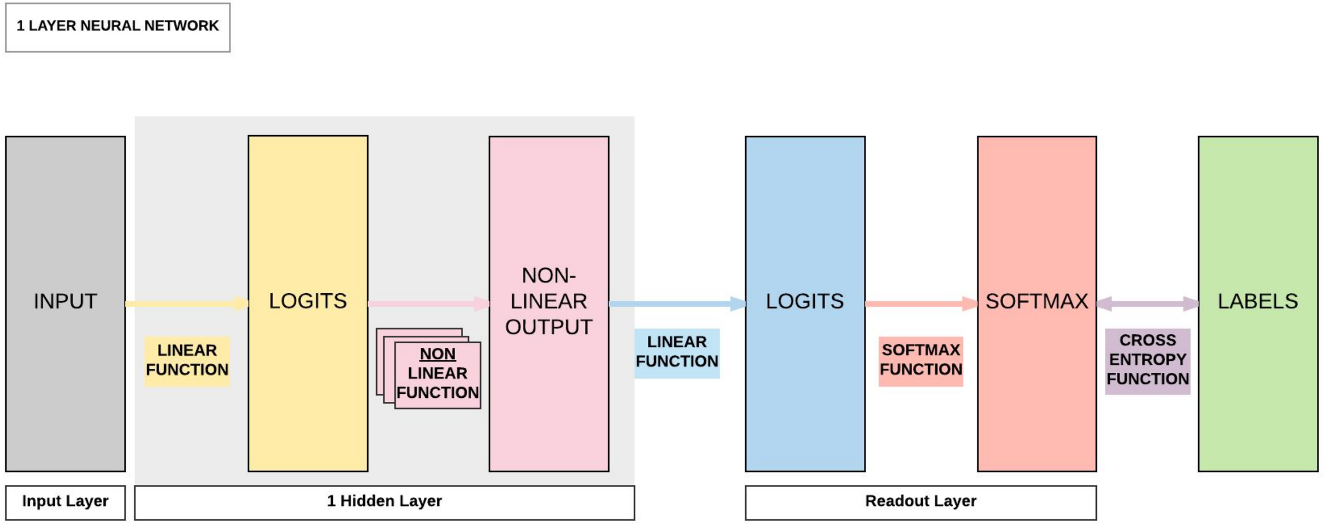
output layer

- **layer** = vector resulting from each linear transformation
- the outer-most linear transform results in the **output layer**
  - if dout = 1 : regression or binary classif
  - if dout > 1: classif MC
- the other linear transforms result in **hidden layers**
- each hidden layer is followed by a **non-linear activation**
- the bias vector can be forced to 0 (= "**dropped**") as here in the last layer
- layers resulting from linear transformations are often referred to as ***fully connected*** or ***affine*** (other types: *pooling* or *convolutional* layers)

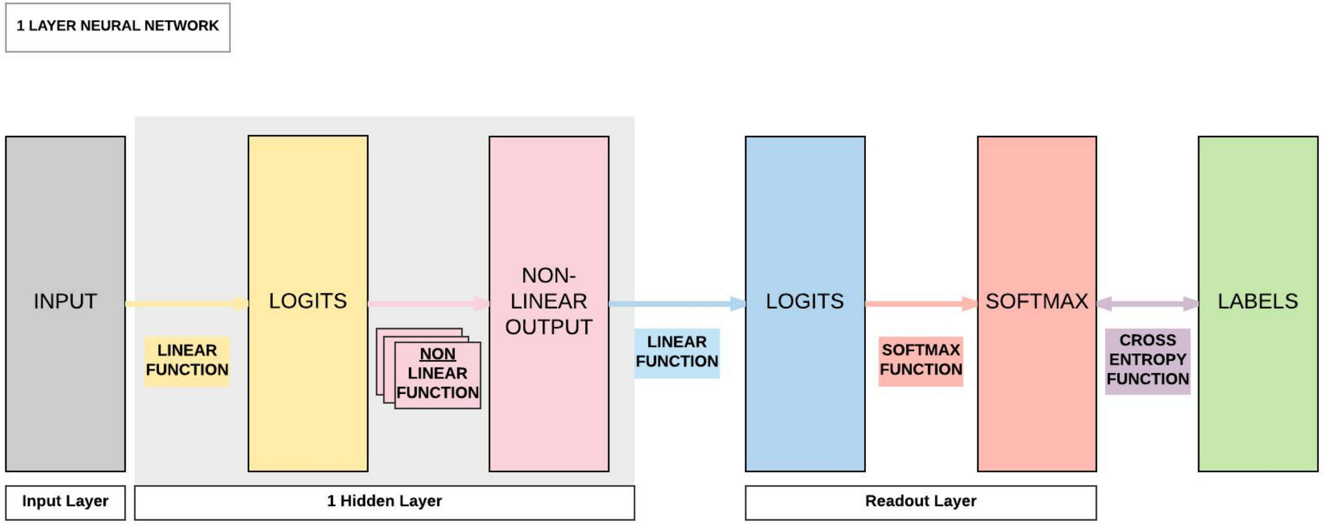
Summary

a feed-forward network is simply a stack of linear models separated by nonlinear functions





Summary



Representation power:

- MLP1 = universal approximator: it can approximate a large family of functions [Hornik et al. 1989; Cybenko, 1989]. Why going beyond MLP1?

  - theoretical results do not discuss the learnability of the NN: a representation exists, but we don’t know how easy or hard it is to set the parameters based on training data and learning algorithm
  - + does not guarantee that a training algorithm will find the correct function
  - + it does not state how large the hidden layer should be

→ in real worlds conditions: there is benefit at trying more complex architectures

Practical session

Walk through code in PyTorch:

- sentiment classification
- feed-forward Neural Network, with BoW representation of documents

Sources

- Course borrowed, with a few modifications, to P. Muller
- Softmax: https://www.wikiwand.com/fr/Fonction_softmax
- https://www.deeplearningbook.org/
- https://towardsdatascience.com/linear-algebra-explained-in-the-context-of-deep-learning-8fcb8fca1494
- https://ml-cheatsheet.readthedocs.io/en/latest/linear_algebra.html
- https://blog.paperspace.com/dataloaders-abstractions-pytorch/
- https://www.deeplearningwizard.com/deep_learning/practical_pytorch/pytorch_feedforward_neuralnetwork/
- https://www.i2tutorials.com/explain-softmax-activation-function-and-difference-between-sigmoid-and-softmax-function/
- http://perso.ens-lyon.fr/jacques.jayez/Cours/LHPST/Deep_Learning_in_NLP_1.pdf
- https://krisbolton.com/a-quick-introduction-to-artificial-neural-networks-part-1
-