# Introduction to Reinforcement Learning
# Session 3

Jean Martinet, based on the course of
Diane Lingrand and Frédéric Precioso

Polytech SI4 / EIT Digital DSC

2020 - 2021

# Outline

# Outline

# Last week

SFFF (S : starting point, safe)
FHFH (F : frozen surface, safe)
FFFH (H : hole, fall to your doom)
HFFG (G : goal, where the frisbee is located)

- 1. Use your code and play with the parameters $\alpha$, $\gamma$, $\epsilon$
  - Plot the time needed for convergence, when varying parameters
- 2. Try gym with the FrozenLake
  - 16 states, 4 actions, a pinch of stochasticity :-)
  - score of $+1$ if the agent achieves the goal (end of game)
  - score of 0 if the agent falls into a hole (end of game)
  - have a try using code from Adesh Gautam
  - if you want to play using the Q-table learned :

```
with open("frozenLake_qTable.pkl", 'rb') as f:
   Q = pickle.load(f)

def choose_action(state):
   action = np.argmax(Q[state, :])
   return action
```

# Mini-quizz

- What is the difference between on-policy and off-policy ?
  -
  -
- Do Q-learning and SARSA both belong to TD learning ?
  -
- Which are they ?
  - $Q(s_t, a_t) += \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$
  - $Q(s_t, a_t) += \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$
- Given $Q^\pi(-, L) = [0, 0, 1, 1, 0, 0]$ and $Q^\pi(-, R) = [0, 1, 1, 0, 0, 0]$, $\epsilon = 0.9$, what is $\pi(s)$ ?
  -

# Mini-quizz

- What is the difference between on-policy and off-policy ?
  - On-policy : same algo. that we are improving generates next actions
  - Off-policy : the algo. that makes decisions is different from the policy we are learning
- Do Q-learning and SARSA both belong to TD learning ?
  - Yes, both improve $\pi$ based on the difference between $V^\pi(s_t)$ and $r_{t+1} + \gamma V^\pi(s_{t+1})$
- Which are they ?
  - $Q(s_t, a_t) += \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$ is Q-learning
  - $Q(s_t, a_t) += \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$ is SARSA
- Given $Q^\pi(-, L) = [0, 0, 1, 1, 0, 0]$ and $Q^\pi(-, R) = [0, 1, 1, 0, 0, 0]$, $\epsilon = 0.9$, what is $\pi(s)$ ?
  - Take a random move with p $= 0.9$, else $\pi(s2) = R$, $\pi(s4) = L$, tie for other states

# Model-based vs model-free methods

- Do we know the transition model and reward ?
  - If yes (model-based), we can use **Dynamic Programming**
  - Use Bellman equations, iterate until convergence $||V_k^\pi - V_{k-1}^\pi|| < \theta$
  - Compute exactly the expectation of sum of future rewards
  - Requires a Markov world : $V^\pi(s_t)$ does not depend on the history

# Model-based vs model-free methods

- If no (model-free), all we can do is interact with the environement
  - Evaluate a policy with **Monte Carlo**
    - Sample trajectories and average return
    - Not Markov, depends on the history
    - (adds up the current return till the end of episode)
    - ($\rightarrow$ the return may be different depending on when the agent arrived on $s_t$)
    - Requires episodes, that terminate
  - Use **Temporal Difference** (Q-learning and SARSA)
    - That we already know
    - Needs Markov

# Outline

# Motivation

- DP, MC, and TD are **tabular methods**
  - When we can discretise states and actions, limited number, write value as a table
- In real life, things are more complex
- Continuous / very large state and /or action spaces
  - Go game : $2^{170}$ states ($>$ nb atoms in universe !), 400 actions
  - Atari games : $240 \times 160$ dimensions
  - Robotics : multiple degrees of freedom
  - etc.
- Polynomial complexity, tabular RL does not scale up

# How to proceed ?

$$\hat{V}^{\pi}(s) = V_{\theta}(s) = V(s, \theta)$$

$$\hat{Q}^{\pi}(s, a) = V_{\theta}(s, a) = V(s, a, \theta)$$

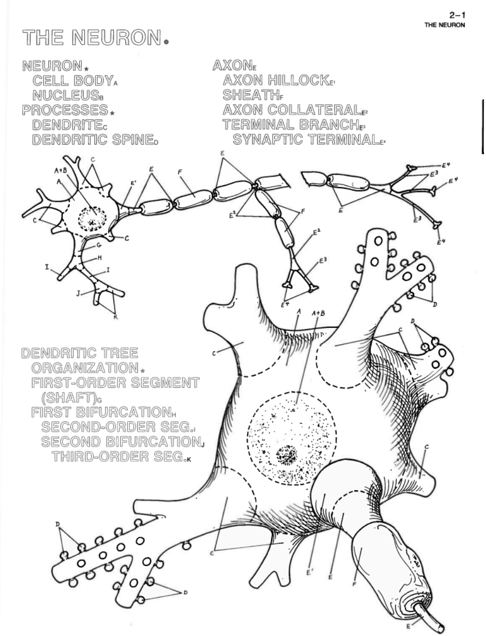
- Use a parametrised function to estimate value function and state-action value fuction
  - Compact representation that generalises across states and actions
  - Reduce memory, computation, experience (data) needed to learn
  - We do not want to store information for all state-action individual pairs
- Which approximator ?
  - Any can do (Decision Tree, Nearest Neighbour, Artificial Neural Networks, etc.)
  - Better if differentiable

# Outline

# Disclamer

From now : CVML slides (Semester 1)
Intended for students who have no background in neural networks.

# Biological neuron

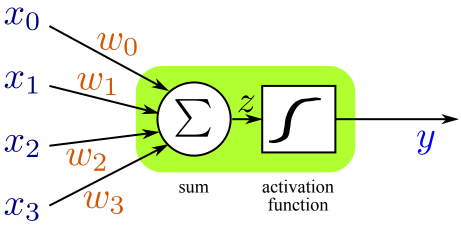



- dendrites : input. multiplication with synaptic weight
- soma or cell body : summation function
- axon : activation function and output

*The Human Brain Coloring Book*
by Diamond etal, 1985

# Outline

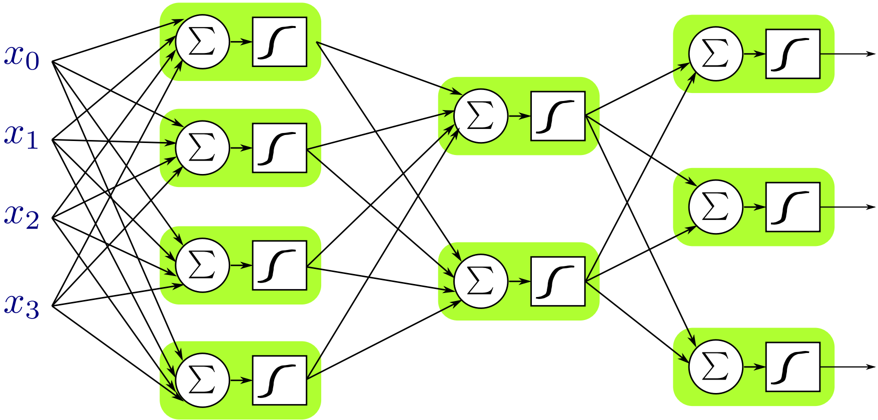
# Artificial neuron



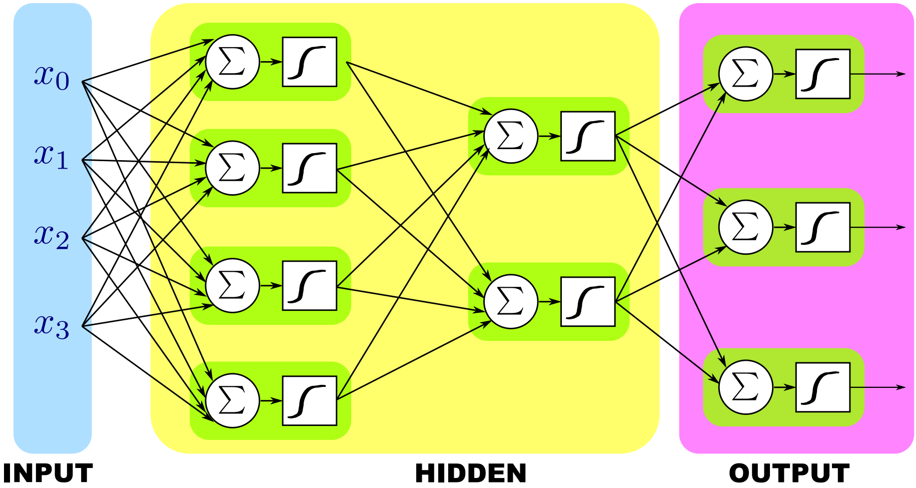

$$y = s(z) = s\left(\sum_{i=0}^{n} w_i x_i\right)$$

- inputs $x_i$ (also outputs of other neurons)
- associate weights $w_i$ (synaptic modulation)
- bias : special input $x_0 = 1$ with weight $w_0$
- sum of the weighted inputs (cell body)
- activation function (axon hillock)
- output $y$

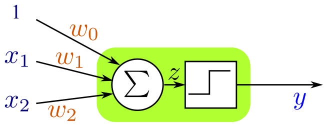
# Feed forward neural network

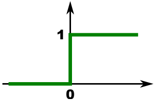


# Feed forward neural network

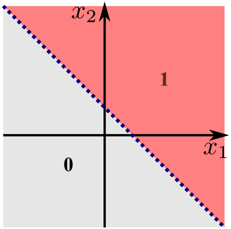


# A simple single neuron




- single neuron with 3 inputs :
- activation function : 

- output : $y = s(z) = s(w_0 + w_1 x_1 + w_2 x_2)$
- equivalent to divide the 2D plane by a line of equation : $w_0 + w_1 x_1 + w_2 x_2 = 0$ or w.x $= 0$



- It is proven that (Cybenko and followers) :
  - all boolean functions can be represented using a single hidden layer
  - all borned continuous functions can be represented using a single hidden layer, with an arbitrary precision
  - all functions can be represented using 2 hidden layers, with an arbitrary precision
- But :
  - How to determine the topology of the neural network ?
  - How many neurons per layer ?
  - How to tune the weights ? (learning phase)

# Outline

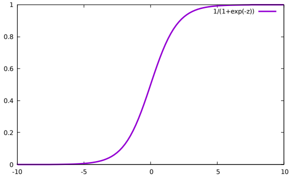
1 Summary of previous contents

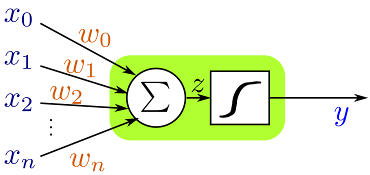
2 Value function approximation

3 A bit of ANN theory (and practice)
- A single neuron
- Perceptron
- Multi layer perceptron (MLP)

# Activation function ?

- For the use of gradient, the activation function must be continuous and differentiable.
- Sigmoids (or S-like)
  - logistic function $s(z) = \frac{1}{1+e^{-z}}$ and
    $s'(z) = \frac{ds}{dz}(z) = \frac{e^{-z}}{(1+e^{-z})^2} = s(z)(1 - s(z))$





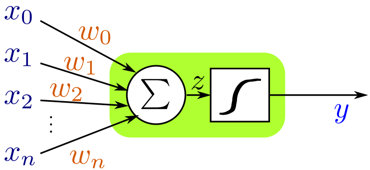

with

$$y = s(z) = s\left(\sum_{i=0}^{n} w_i x_i\right)$$

For a given data $j$, the output of the neuron $y(\mathrm{w}, \mathrm{x}^j)$ should be equal to $y^j$.

$$E = \frac{1}{2}(y(\mathrm{w}, \mathrm{x}^j) - y^j)^2$$

# A single neuron




with

$$y = s(z) = s\left(\sum_{i=0}^{n} w_i x_i\right)$$

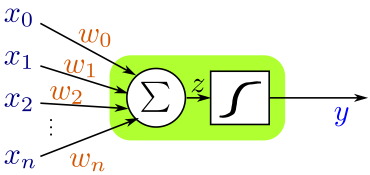
For a given data $j$, the output of the neuron $y(\mathrm{w}, \mathrm{x}^j)$ should be equal to $y^j$.

$$E = \frac{1}{2}(y(\mathrm{w}, \mathrm{x}^j) - y^j)^2$$

Gradient descent :

$$\nabla_{\mathrm{w}} E = \left[\frac{\partial E}{\partial w_0}\ \frac{\partial E}{\partial w_1} \dots \frac{\partial E}{\partial w_n}\right]$$

# A single neuron




with

$$y = s(z) = s\left(\sum_{i=0}^{n} w_i x_i\right)$$

For a given data $j$, the output of the neuron $y(\mathrm{w}, \mathrm{x}^j)$ should be equal to $y^j$.

$$E = \frac{1}{2}(y(\mathrm{w}, \mathrm{x}^j) - y^j)^2$$

Gradient descent :

$$\nabla_{\mathrm{w}} E = \left[\frac{\partial E}{\partial w_0}\ \frac{\partial E}{\partial w_1} \dots \frac{\partial E}{\partial w_n}\right] = (y(\mathrm{w}, \mathrm{x}^j) - y^j)\nabla_{\mathrm{w}} y(\mathrm{w}, \mathrm{x}^j)$$

where

$$\nabla_{\mathrm{w}} y(\mathrm{w}, \mathrm{x}^j) = \left[\frac{\partial y}{\partial w_0}\ \frac{\partial y}{\partial w_1} \dots \frac{\partial y}{\partial w_n}\right]$$

# A single neuron (2)

We compute :

$$\nabla_{\mathrm{w}} E = (y(\mathrm{w}, \mathrm{x}^j) - y^j) \nabla_{\mathrm{w}} y(\mathrm{w}, \mathrm{x}^j) \text{ where } \nabla_{\mathrm{w}} y(\mathrm{w}, \mathrm{x}^j) = \left[ \frac{\partial y}{\partial w_0} \; \frac{\partial y}{\partial w_1} \ldots \frac{\partial y}{\partial w_n} \right]$$

knowing that :

$$y = s(z) = s\left( \sum_{i=0}^{n} w_i x_i \right)$$

# A single neuron (2)

We compute :

$$\nabla_{\mathrm{w}} E = (y(\mathrm{w}, \mathrm{x}^j) - y^j)\nabla_{\mathrm{w}} y(\mathrm{w}, \mathrm{x}^j) \text{ where } \nabla_{\mathrm{w}} y(\mathrm{w}, \mathrm{x}^j) = \left[\frac{\partial y}{\partial w_0}\ \frac{\partial y}{\partial w_1} \dots \frac{\partial y}{\partial w_n}\right]$$

knowing that :

$$y = s(z) = s\left(\sum_{i=0}^{n} w_i x_i\right)$$

$$\frac{\partial y}{\partial w_i}(\mathrm{w}, \mathrm{x}^j) =$$

# A single neuron (2)

We compute :

$$\nabla_{\mathrm{w}} E = (y(\mathrm{w}, \mathrm{x}^j) - y^j)\nabla_{\mathrm{w}} y(\mathrm{w}, \mathrm{x}^j) \text{ where } \nabla_{\mathrm{w}} y(\mathrm{w}, \mathrm{x}^j) = \left[\frac{\partial y}{\partial w_0} \ \frac{\partial y}{\partial w_1} \dots \frac{\partial y}{\partial w_n}\right]$$

knowing that :

$$y = s(z) = s\left(\sum_{i=0}^{n} w_i x_i\right)$$

$$\frac{\partial y}{\partial w_i}(\mathrm{w}, \mathrm{x}^j) = \frac{\partial y}{\partial z}\frac{\partial z}{\partial w_i}(\mathrm{w}, \mathrm{x}^j)$$

# A single neuron (2)

We compute :

$$\nabla_{\mathrm{w}} E = (y(\mathrm{w}, \mathrm{x}^j) - y^j)\nabla_{\mathrm{w}} y(\mathrm{w}, \mathrm{x}^j) \text{ where } \nabla_{\mathrm{w}} y(\mathrm{w}, \mathrm{x}^j) = \left[\frac{\partial y}{\partial w_0} \; \frac{\partial y}{\partial w_1} \ldots \frac{\partial y}{\partial w_n}\right]$$

knowing that :

$$y = s(z) = s\left(\sum_{i=0}^{n} w_i x_i\right)$$

$$\frac{\partial y}{\partial w_i}(\mathrm{w}, \mathrm{x}^j) = \frac{\partial y}{\partial z}\frac{\partial z}{\partial w_i}(\mathrm{w}, \mathrm{x}^j) = s'(z)\underbrace{\frac{\partial\left(\sum_{k=0}^{n} w_k x_k^j\right)}{\partial w_i}}_{x_i^j}$$

# A single neuron (2)

We compute :

$$\nabla_{\mathrm{w}} E = (y(\mathrm{w}, \mathrm{x}^j) - y^j) \nabla_{\mathrm{w}} y(\mathrm{w}, \mathrm{x}^j) \text{ where } \nabla_{\mathrm{w}} y(\mathrm{w}, \mathrm{x}^j) = \left[ \frac{\partial y}{\partial w_0} \; \frac{\partial y}{\partial w_1} \dots \frac{\partial y}{\partial w_n} \right]$$

knowing that :

$$y = s(z) = s\left( \sum_{i=0}^{n} w_i x_i \right)$$

$$\frac{\partial y}{\partial w_i}(\mathrm{w}, \mathrm{x}^j) = \frac{\partial y}{\partial z} \frac{\partial z}{\partial w_i}(\mathrm{w}, \mathrm{x}^j) = s'(z) \underbrace{\frac{\partial \left( \sum_{k=0}^{n} w_k x_k^j \right)}{\partial w_i}}_{x_i^j} = y(1 - y) x_i^j$$

with $y = y(\mathrm{w}, \mathrm{x}^j)$.

# A single neuron (2)

We compute :

$$\nabla_{\mathrm{w}} E = (y(\mathrm{w}, \mathrm{x}^j) - y^j)\nabla_{\mathrm{w}} y(\mathrm{w}, \mathrm{x}^j) \text{ where } \nabla_{\mathrm{w}} y(\mathrm{w}, \mathrm{x}^j) = \left[\frac{\partial y}{\partial w_0} \; \frac{\partial y}{\partial w_1} \dots \frac{\partial y}{\partial w_n}\right]$$

knowing that :

$$y = s(z) = s\left(\sum_{i=0}^{n} w_i x_i\right)$$

$$\frac{\partial y}{\partial w_i}(\mathrm{w}, \mathrm{x}^j) = \frac{\partial y}{\partial z}\frac{\partial z}{\partial w_i}(\mathrm{w}, \mathrm{x}^j) = s'(z)\underbrace{\frac{\partial\left(\sum_{k=0}^{n} w_k x_k^j\right)}{\partial w_i}}_{x_i^j} = y(1-y)x_i^j$$

with $y = y(\mathrm{w}, \mathrm{x}^j)$. Thus :

$$\nabla_{\mathrm{w}} y = y(1-y)\mathrm{x}^{\mathrm{j}}$$

# A single neuron (3)

We obtained :

$$\nabla_{\mathrm{w}} E = (y - y^j) y (1 - y) \mathrm{x}^{\mathrm{j}}$$

# A single neuron (3)

We obtained :

$$\nabla_{\mathrm{w}} E = (y - y^j) y (1 - y) \mathrm{x}^{\mathrm{j}}$$

Let us note :

$$\delta = (y - y^j) y (1 - y)$$

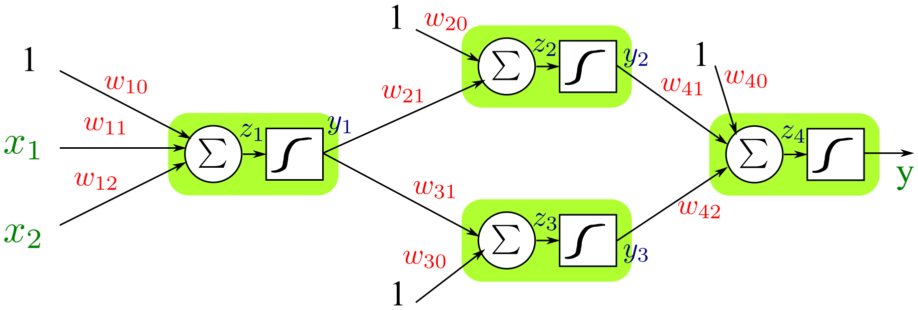
Gradient descent rule :

$$\mathrm{w} \leftarrow \mathrm{w} - \eta\, \delta\, \mathrm{x}^{\mathrm{j}}$$

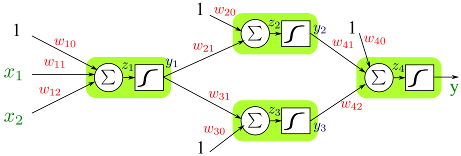
# Outline

- It is proven that (Cybenko and followers) :
  - all boolean functions can be represented using a single hidden layer
  - all borned continuous functions can be represented using a single hidden layer, with an arbitrary precision
  - all functions can be represented using 2 hidden layers, with an arbitrary precision
- But :
  - How to determine the topology of the neural network ?
  - How many neurons per layer ?
  - How to tune the weights ? (learning phase)

# Training a multilayer perceptron

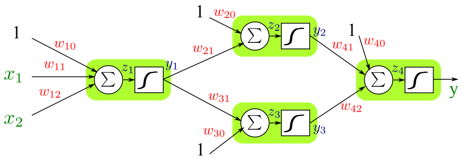


# Forward propagation



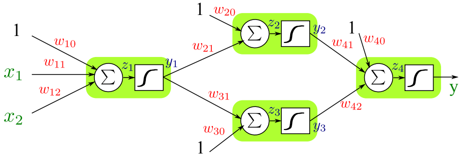

- $y_1 = s(z_1) = s(w_{10} + w_{11}x_1 + w_{12}x_2)$

# Forward propagation



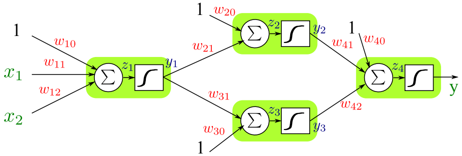

- $y_1 = s(z_1) = s(w_{10} + w_{11}x_1 + w_{12}x_2)$
- $y_2 = s(z_2) = s(w_{20} + w_{21}y_1)$

# Forward propagation




- $y_1 = s(z_1) = s(w_{10} + w_{11}x_1 + w_{12}x_2)$
- $y_2 = s(z_2) = s(w_{20} + w_{21}y_1)$
- $y_3 = s(z_3) = s(w_{30} + w_{31}y_1)$

# Forward propagation



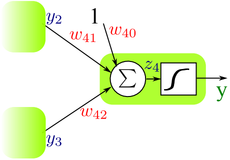

- $y_1 = s(z_1) = s(w_{10} + w_{11}x_1 + w_{12}x_2)$
- $y_2 = s(z_2) = s(w_{20} + w_{21}y_1)$
- $y_3 = s(z_3) = s(w_{30} + w_{31}y_1)$
- $y = s(z_4) = s(w_{40} + w_{41}y_2 + w_{42}y_3)$

# Error minimization

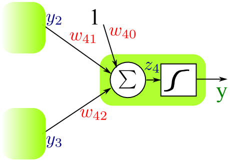
- Error : $E = \dfrac{1}{2}(y(\mathrm{w}, \mathrm{x}^j) - y^j)^2$ with
  $\mathrm{w} = [w_{10}\ w_{11}\ w_{12}\ w_{20}\ w_{21}\ w_{30}\ w_{31}\ w_{40}\ w_{41}\ w_{42}]$ and $\mathrm{x}^j = [x_1\ x_2]$
- Minimization : computation of the gradient
  $\nabla_{\mathrm{w}} E = (y(\mathrm{w}, \mathrm{x}^j) - y^j) \nabla_{\mathrm{w}} y(\mathrm{w}, \mathrm{x}^j)$ where $\nabla_{\mathrm{w}} y = \left[ \frac{\partial y}{\partial w_{10}}\ \frac{\partial y}{\partial w_{11}} \dots \frac{\partial y}{\partial w_{42}} \right]$

# Error minimization

- Error : $E = \frac{1}{2}(y(\mathrm{w}, \mathrm{x}^j) - y^j)^2$ with
  $\mathrm{w} = [w_{10}\ w_{11}\ w_{12}\ w_{20}\ w_{21}\ w_{30}\ w_{31}\ w_{40}\ w_{41}\ w_{42}]$ and $\mathrm{x}^j = [x_1\ x_2]$
- Minimization : computation of the gradient
  $\nabla_{\mathrm{w}} E = (y(\mathrm{w}, \mathrm{x}^j) - y^j)\nabla_{\mathrm{w}} y(\mathrm{w}, \mathrm{x}^j)$ where $\nabla_{\mathrm{w}} y = \left[\frac{\partial y}{\partial w_{10}}\ \frac{\partial y}{\partial w_{11}} \ldots \frac{\partial y}{\partial w_{42}}\right]$
- Output layer :



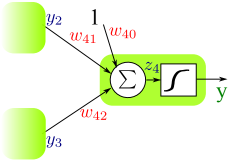
- Error : $E = \frac{1}{2}(y(\mathrm{w}, \mathrm{x}^j) - y^j)^2$ with
  $\mathrm{w} = [w_{10}\ w_{11}\ w_{12}\ w_{20}\ w_{21}\ w_{30}\ w_{31}\ w_{40}\ w_{41}\ w_{42}]$ and $\mathrm{x}^j = [x_1\ x_2]$
- Minimization : computation of the gradient
  $\nabla_{\mathrm{w}} E = (y(\mathrm{w}, \mathrm{x}^j) - y^j)\nabla_{\mathrm{w}} y(\mathrm{w}, \mathrm{x}^j)$ where $\nabla_{\mathrm{w}} y = \left[\frac{\partial y}{\partial w_{10}}\ \frac{\partial y}{\partial w_{11}} \dots \frac{\partial y}{\partial w_{42}}\right]$
- Output layer :




$$\frac{\partial y}{\partial w_{42}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial w_{42}} = y(1-y)y_3$$
$$\frac{\partial y}{\partial w_{41}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial w_{41}} = y(1-y)y_2$$
$$\frac{\partial y}{\partial w_{40}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial w_{40}} = y(1-y)$$
$$\Rightarrow \delta_4 = (y - y^j)y(1-y)$$

# Error minimization

- Error : $E = \frac{1}{2}(y(\mathrm{w}, \mathrm{x}^j) - y^j)^2$ with
  $\mathrm{w} = [w_{10}\ w_{11}\ w_{12}\ w_{20}\ w_{21}\ w_{30}\ w_{31}\ w_{40}\ w_{41}\ w_{42}]$ and $\mathrm{x}^j = [x_1\ x_2]$
- Minimization : computation of the gradient
  $\nabla_{\mathrm{w}} E = (y(\mathrm{w}, \mathrm{x}^j) - y^j)\nabla_{\mathrm{w}} y(\mathrm{w}, \mathrm{x}^j)$ where $\nabla_{\mathrm{w}} y = \left[\frac{\partial y}{\partial w_{10}}\ \frac{\partial y}{\partial w_{11}} \dots \frac{\partial y}{\partial w_{42}}\right]$
- Output layer :

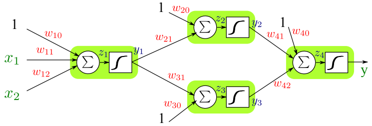



$$
\begin{aligned}
\frac{\partial y}{\partial w_{42}} &= \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial w_{42}} = y(1-y)y_3 \\
\frac{\partial y}{\partial w_{41}} &= \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial w_{41}} = y(1-y)y_2 \\
\frac{\partial y}{\partial w_{40}} &= \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial w_{40}} = y(1-y) \\
\Rightarrow \delta_4 &= (y - y^j)y(1-y)
\end{aligned}
$$

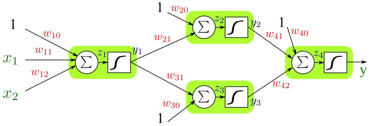
- Gradient descent :

$$
\begin{aligned}
w_{42} &\leftarrow w_{42} - \eta\delta_4 y_3 \\
w_{41} &\leftarrow w_{41} - \eta\delta_4 y_2 \\
w_{40} &\leftarrow w_{40} - \eta\delta_4
\end{aligned}
$$

# Gradient descent on hidden layers



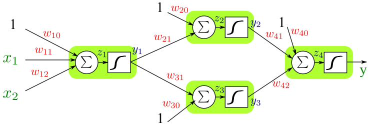

$$\frac{\partial y}{\partial w_{31}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial w_{31}} \text{ with } \begin{cases} z_4 = w_{40} + w_{41}y_2 + w_{42}y_3 \\ y_3 = s(z_3) = s(w_{30} + w_{31}y_1) \end{cases}$$




$$\frac{\partial y}{\partial w_{31}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial w_{31}} \text{ with } \begin{cases} z_4 = w_{40} + w_{41}y_2 + w_{42}y_3 \\ y_3 = s(z_3) = s(w_{30} + w_{31}y_1) \end{cases}$$

- Then $\dfrac{\partial y}{\partial w_{31}} = \dfrac{\partial y}{\partial z_4}\dfrac{\partial z_4}{\partial y_3}\dfrac{\partial y_3}{\partial z_3}\dfrac{\partial z_3}{\partial w_{31}} = y(1-y)w_{42}y_3(1-y_3)y_1$

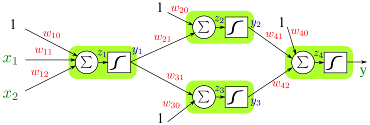
# Gradient descent on hidden layers




$$\frac{\partial y}{\partial w_{31}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial w_{31}} \text{ with } \begin{cases} z_4 = w_{40} + w_{41}y_2 + w_{42}y_3 \\ y_3 = s(z_3) = s(w_{30} + w_{31}y_1) \end{cases}$$

- Then $\frac{\partial y}{\partial w_{31}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial y_3}\frac{\partial y_3}{\partial z_3}\frac{\partial z_3}{\partial w_{31}} = y(1-y)w_{42}y_3(1-y_3)y_1$
- And, similarly :

$$\frac{\partial y}{\partial w_{30}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial w_{30}} = y(1-y)w_{42}y_3(1-y_3)$$



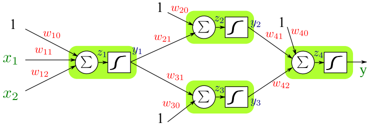

$$\frac{\partial y}{\partial w_{31}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial w_{31}} \text{ with } \begin{cases} z_4 = w_{40} + w_{41}y_2 + w_{42}y_3 \\ y_3 = s(z_3) = s(w_{30} + w_{31}y_1) \end{cases}$$

- Then $\frac{\partial y}{\partial w_{31}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial y_3}\frac{\partial y_3}{\partial z_3}\frac{\partial z_3}{\partial w_{31}} = y(1-y)w_{42}y_3(1-y_3)y_1$
- And, similarly :

$$\frac{\partial y}{\partial w_{30}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial w_{30}} = y(1-y)w_{42}y_3(1-y_3)$$

- Thus : $\delta_3 = \delta_4 w_{42} y_3 (1-y_3)$ (recursion !)

# Gradient descent on hidden layers

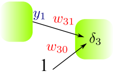


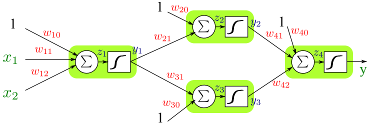

$$\frac{\partial y}{\partial w_{31}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial w_{31}} \text{ with } \begin{cases} z_4 = w_{40} + w_{41}y_2 + w_{42}y_3 \\ y_3 = s(z_3) = s(w_{30} + w_{31}y_1) \end{cases}$$

- Then $\dfrac{\partial y}{\partial w_{31}} = \dfrac{\partial y}{\partial z_4}\dfrac{\partial z_4}{\partial y_3}\dfrac{\partial y_3}{\partial z_3}\dfrac{\partial z_3}{\partial w_{31}} = y(1-y)w_{42}y_3(1-y_3)y_1$
- And, similarly :

$$\frac{\partial y}{\partial w_{30}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial w_{30}} = y(1-y)w_{42}y_3(1-y_3)$$

- Thus : $\delta_3 = \delta_4 w_{42} y_3(1-y_3)$ (recursion !)
- Gradient descent : $\begin{array}{l} w_{30} \leftarrow w_{30} - \eta\delta_3 \\ w_{31} \leftarrow w_{31} - \eta\delta_3 y_1 \end{array}$



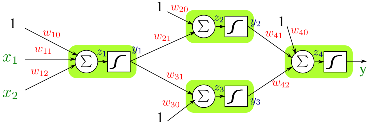



- Similarly :

$$\frac{\partial y}{\partial w_{21}} =$$

- and

$$\frac{\partial y}{\partial w_{20}} =$$




- Similarly :

$$\frac{\partial y}{\partial w_{21}} =$$

- and

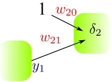
$$\frac{\partial y}{\partial w_{20}} =$$

- Thus :

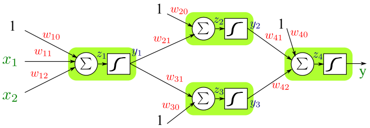
$$\delta_2 = \delta_4 w_{41} y_2 (1 - y_2)$$

- Gradient descent :

$$w_{20} \leftarrow w_{20} - \eta \delta_2$$
$$w_{21} \leftarrow w_{21} - \eta \delta_2 y_1$$



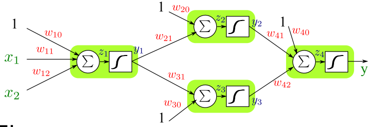
# Gradient descent on first hidden layer




$$\frac{\partial y}{\partial w_{12}} = \frac{\partial y}{\partial z_4} \frac{\partial z_4}{\partial w_{12}}$$

with $z_4 = w_{40} + w_{41} y_2 + w_{42} y_3$

# Gradient descent on first hidden layer



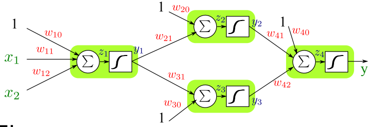

$$\frac{\partial y}{\partial w_{12}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial w_{12}}$$

with $z_4 = w_{40} + w_{41}y_2 + w_{42}y_3$

Thus :

$$\begin{aligned}
\frac{\partial y}{\partial w_{12}} &= \frac{\partial y}{\partial z_4}\left(\frac{\partial z_4}{\partial y_2}\frac{\partial y_2}{\partial z_2}\frac{\partial z_2}{\partial y_1}\frac{\partial y_1}{\partial z_1}\frac{\partial z_1}{\partial w_{12}} + \frac{\partial z_4}{\partial y_3}\frac{\partial y_3}{\partial z_3}\frac{\partial z_3}{\partial y_1}\frac{\partial y_1}{\partial z_1}\frac{\partial z_1}{\partial w_{12}}\right) \\
&= y(1-y)(w_{41}y_2(1-y_2)w_{21}y_1(1-y_1)x_2 \\
&\quad + w_{42}y_3(1-y_3)w_{31}y_1(1-y_1)x_2) \\
&= y(1-y)\left(w_{41}y_2(1-y_2)w_{21} + w_{42}y_3(1-y_3)w_{31}\right)y_1(1-y_1)x_2
\end{aligned}$$

# Gradient descent on first hidden layer




$$\frac{\partial y}{\partial w_{12}} = \frac{\partial y}{\partial z_4} \frac{\partial z_4}{\partial w_{12}}$$
with $z_4 = w_{40} + w_{41} y_2 + w_{42} y_3$

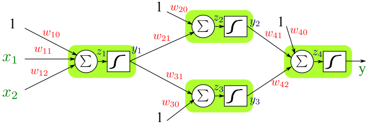
Thus :

$$\begin{aligned}
\frac{\partial y}{\partial w_{12}} &= \frac{\partial y}{\partial z_4} \left( \frac{\partial z_4}{\partial y_2} \frac{\partial y_2}{\partial z_2} \frac{\partial z_2}{\partial y_1} \frac{\partial y_1}{\partial z_1} \frac{\partial z_1}{\partial w_{12}} + \frac{\partial z_4}{\partial y_3} \frac{\partial y_3}{\partial z_3} \frac{\partial z_3}{\partial y_1} \frac{\partial y_1}{\partial z_1} \frac{\partial z_1}{\partial w_{12}} \right) \\
&= y(1-y)(w_{41} y_2 (1-y_2) w_{21} y_1 (1-y_1) x_2 \\
&\quad + w_{42} y_3 (1-y_3) w_{31} y_1 (1-y_1) x_2) \\
&= y(1-y)\left(w_{41} y_2 (1-y_2) w_{21} + w_{42} y_3 (1-y_3) w_{31}\right) y_1 (1-y_1) x_2
\end{aligned}$$

And :

$$\delta_1 = (\delta_2 w_{21} + \delta_3 w_{31}) y_1 (1-y_1)$$

# Gradient descent on first hidden layer




$$\frac{\partial y}{\partial w_{12}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial w_{12}}$$

with $z_4 = w_{40} + w_{41}y_2 + w_{42}y_3$

Thus :

$$\begin{aligned}
\frac{\partial y}{\partial w_{12}} &= \frac{\partial y}{\partial z_4}\left(\frac{\partial z_4}{\partial y_2}\frac{\partial y_2}{\partial z_2}\frac{\partial z_2}{\partial y_1}\frac{\partial y_1}{\partial z_1}\frac{\partial z_1}{\partial w_{12}} + \frac{\partial z_4}{\partial y_3}\frac{\partial y_3}{\partial z_3}\frac{\partial z_3}{\partial y_1}\frac{\partial y_1}{\partial z_1}\frac{\partial z_1}{\partial w_{12}}\right) \\
&= y(1-y)(w_{41}y_2(1-y_2)w_{21}y_1(1-y_1)x_2 \\
&\quad + w_{42}y_3(1-y_3)w_{31}y_1(1-y_1)x_2) \\
&= y(1-y)\left(w_{41}y_2(1-y_2)w_{21} + w_{42}y_3(1-y_3)w_{31}\right)y_1(1-y_1)x_2
\end{aligned}$$

And :

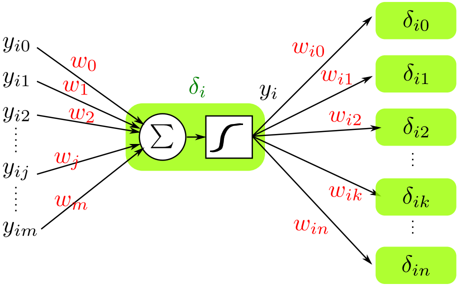
$$\delta_1 = (\delta_2 w_{21} + \delta_3 w_{31})y_1(1-y_1)$$

Gradient descent :

$$\begin{aligned}
w_{10} &\leftarrow w_{10} - \eta\delta_1 \\
w_{11} &\leftarrow w_{11} - \eta\delta_1 x_1 \\
w_{12} &\leftarrow w_{12} - \eta\delta_1 x_2
\end{aligned}$$

# Generalized $\delta$ rule

Considering neuron $i$ connected to other neurons :




- Delta rule :

$$\delta_i = y_i(1 - y_i) \sum_{k=0}^{n} w_{ik}\delta_{ik}$$

- Gradient descent :

$$\forall j \in [0 \;\; m] \;\; w_j \leftarrow w_j - \eta \delta_i y_{ij}$$

# Backpropagation Algorithm (*Retropropagation*)

1. Initialize weights to small random values
2. Choose a random sample training item, say $(x^j, y^j)$
3. Compute total input $z_i$ and output $y_i$ for each unit (**forward prop**)
4. Compute $\delta_p$ for output layer $\delta_p = y_p(1 - y_p)(y_p - y_j)$
5. Compute $\delta_i$ for all preceding layers by **backprop rule**
6. Compute weight change by **descent rule** (repeat for all weights)

- Note that each expression involves data local to a particular unit, we do not have to look around summing things over the whole network.
- It is for this reason, simplicity, locality and, therefore, efficiency that backpropagation has become the dominant paradigm for training neural nets.

# Input encoding

- All values are real values from [0 1].
- If not, scale the values : $x = \frac{u - u_{\min}}{u_{\max} - u_{\min}}$
- For discrete values, unary encoding form. As for example :

| | $x_1$ | $x_2$ | $x_3$ |
|---|---|---|---|
| flour | 1 | 0 | 0 |
| butter | 0 | 1 | 0 |
| sugar | 0 | 0 | 1 |

# Output encoding

- for a binary classification :
  - one output neuron
    - if $y < 0.5$ : class 0
    - else class 1
- for a multi-class classification ($n$ classes) :
  - one output neuron : divide [0 1] by $n$
  - $n$ output neurons
    - each neuron corresponds to probability over classes

# Training

1. Split data set (randomly) into three subsets :
   - Training set : used for adjusting weights
   - Validation set : used to stop training
   - Test set : used to evaluate performance
2. Pick random, small weights as initial values
3. Perform iterative minimization of error over training set
4. Stop when error on validation set reaches a minimum (to avoir overfitting)
5. Repeat training (from step 2) several times (to avoir local minima)
6. Use best weights to compute error on the test set.

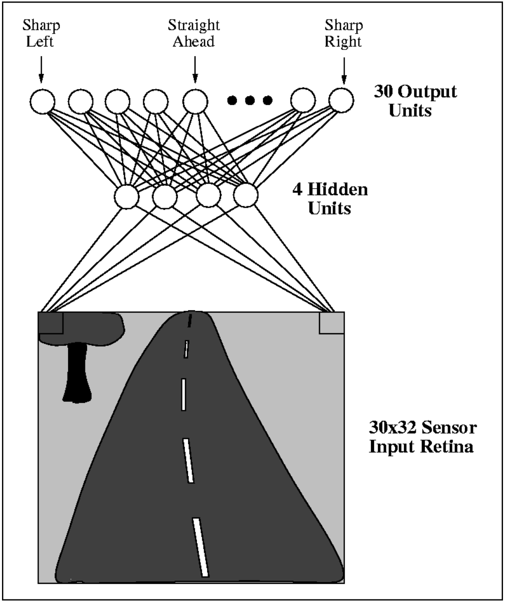
If the data set is too small to be divided into 3 subsets, use cross-validation.

# ALVINN : Autonomous Land Vehicle In a Neural Network

- 960 inputs (a 30x32 array derived from the pixels of an image)
- 4 hidden units
- 30 output units (each representing a steering command)

https://youtu.be/ilP4aPDTBPE

Dean Pomerleau, CMU, 1989



# Activation functions

- Main activation functions :
  - Sigmoid
  - ReLU : Rectified Linear Unit
  - Softmax : transforms output values into values that can be interpreted as probabilities
- Other activation functions :
  - tanh, LeakyReLU, PReLU, ELU, ...

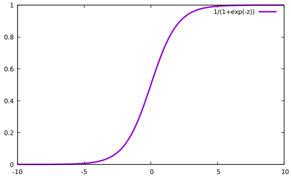
# Why ReLU ?

- problem of vanishing gradient
  - remember the delta rule and gradient descent :

$$\delta_i = y_i(1 - y_i) \sum_{k=0}^{n} w_{ik} \delta_{ik}$$

and

$$\forall j \in [0 \;\; m] \; w_j \leftarrow w_j - \eta \delta_i y_{ij}$$




- different solutions :
  - change to activation function to ReLU
  - residual networks
  - batch normalisation

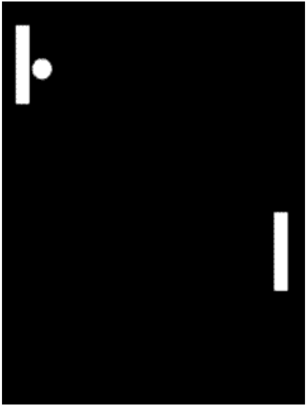
# Softmax layer

- smooth approximation to the arg max function
  - assign probabilities to indices of having the highest response $z = [z_0 z_1 z_2 ... z_n]$ for each $i$,

$$y_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

  - every value in [0 1] and the sum is equal to 1

# Playing Pong using a neural network



- input : images
- output : up or down command

# Neural network with tensorflow.keras library and python3

- installation : `pip3 install --upgrade pip ; pip3 install tensorflow`
- choosing a layer type
  - fully connected or `dense`
  - convolutional : `conv`
- choosing the activation function
  - hidden layer : sigmoid, tanh ($\tanh(t) = 2\text{sig}(2t) - 1$), relu
  - output layer : linear (regression) or softmax (classification)
- choosing a loss function
  - `categorical_crossentropy` for multiclass classification
  - `binary_crossentropy` for binary classification
  - `mse` for regression
- choosing a learning rate
  - fixed : not too small, not too large. Could be $\eta = 0.01$
  - adaptive : Momentum, Adagrad, RMSProp, Adam, ...
- ways of creating a simple network :
  - sequential model
  - functional API
  - subclassing (next year ?)

# keras sequential model

```
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import utils
model = keras.Sequential()
model.add(layers.Dense(16, input_dim=N_features, activation='relu'))
model.add(layers.Dense(8, activation='relu'))
model.add(layers.Dense(3))
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')
model.fit(X_train, y_train, epochs=100, validation_split=0.33)
y_pred = model.predict(X_test)
```

# keras functionnal API (more complex network)

```
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import utils
inputs = keras.Input(shape=(N_features, ))
x = layers.Dense(16, activation='relu')(inputs)
x = layers.Dense(8, activation='relu')(x)
outputs = layers.Dense(3)(x)
model = keras.Model(inputs, outputs)
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')
model.fit(X_train, y_train, epochs=100, validation_split=0.33)
y_pred = model.predict(X_test)
```