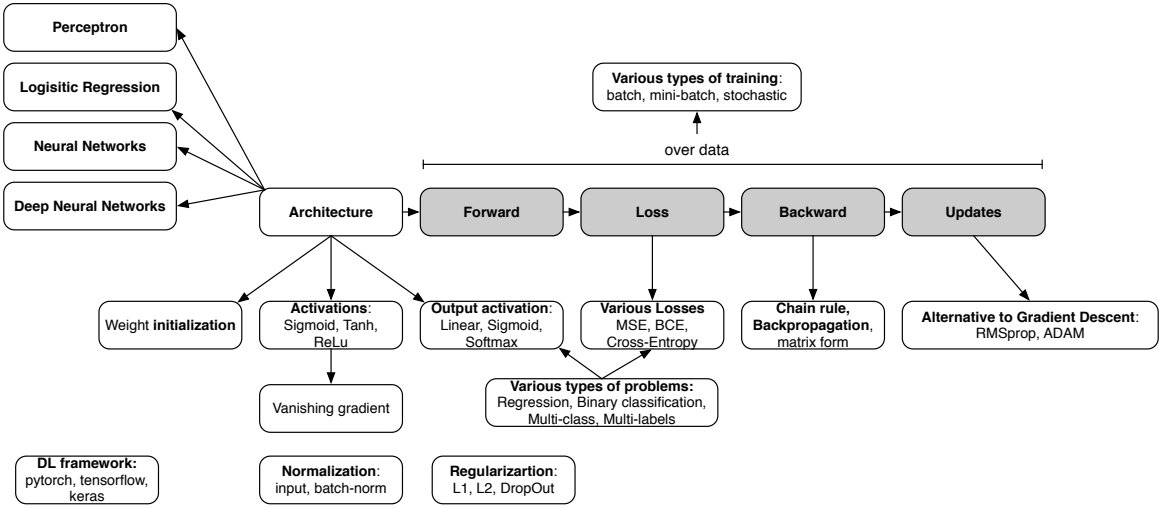
Deep Learning
Topics: Multi-Layers-Perceptron

Geoffroy Peeters

LTCI, Télécom Paris, IP Paris

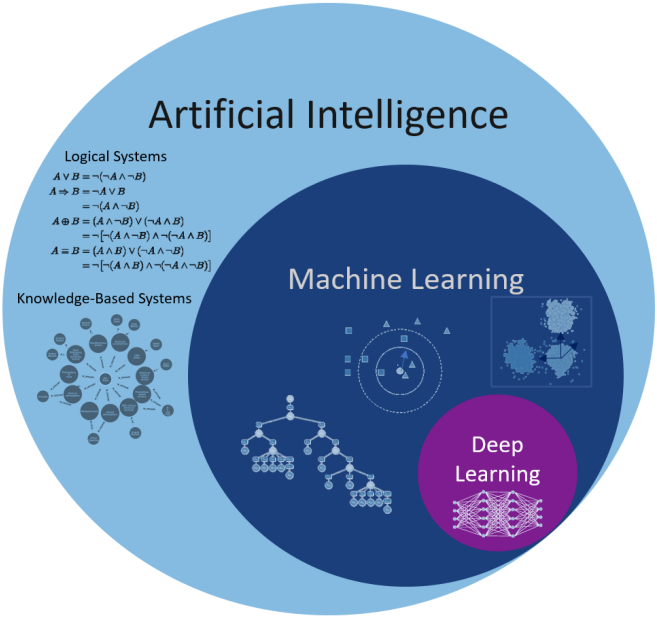
-

Overview

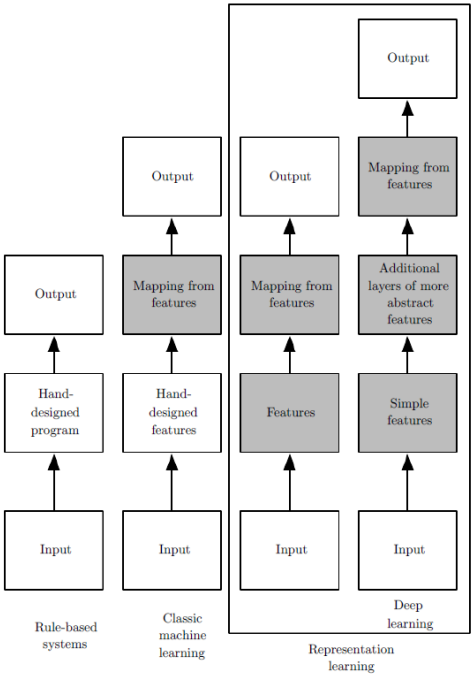

Perceptron
Logisitic Regression
Neural Networks
Deep Neural Networks
Various types of training: batch, mini-batch, stochastic
over data
Architecture
Forward
Loss
Backward
Updates
Weight initialization
Activations: Sigmoid, Tanh, ReLu
Output activation: Linear, Sigmoid, Softmax
Various Losses MSE, BCE, Cross-Entropy
Chain rule, Backpropagation, matrix form
Alternative to Gradient Descent: RMSprop, ADAM
Vanishing gradient
Various types of problems: Regression, Binary classification, Multi-class, Multi-labels
DL framework: pytorch, tensorflow, keras
Normalization: input, batch-norm
Regularizartion: L1, L2, DropOut

Deep Learning and Neural Networks : History

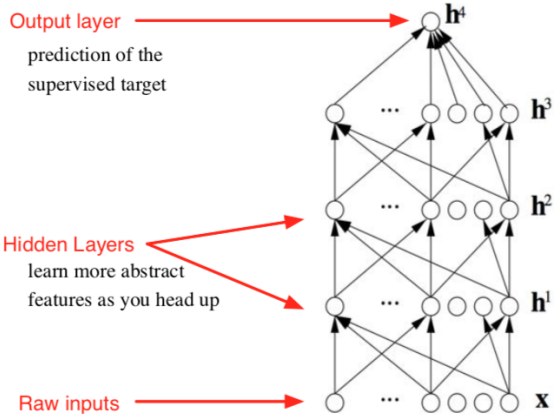
Deep learning (a subset of machine learning)



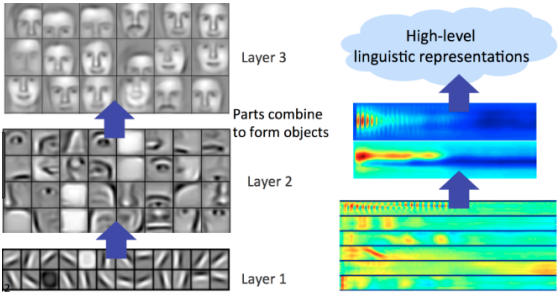
From http://canvart.club/atmose-11_21_13.html



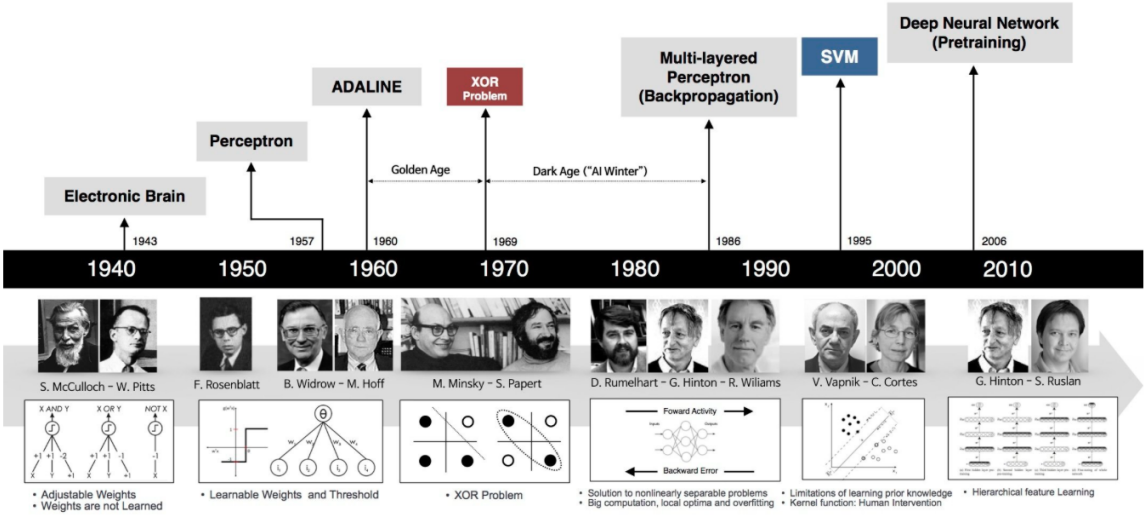
From Deep Learning by Ian Goodfellow and Yoshua Bengio and Aaron Courville

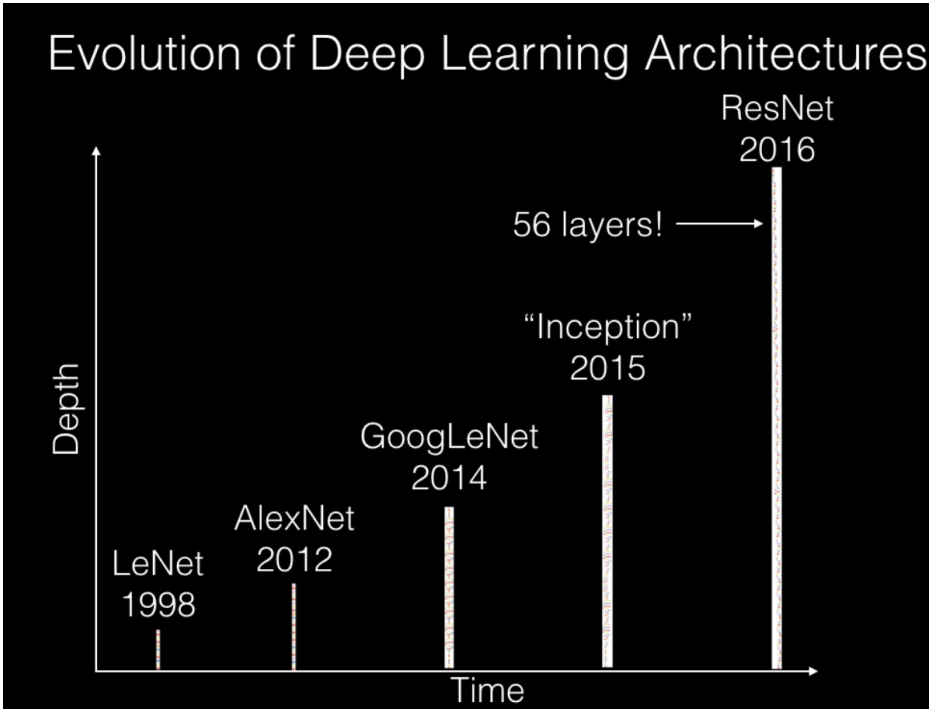


From Bengio 2015



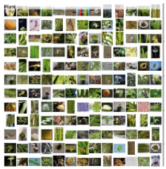
From Bengio 2015


Electronic Brain
1943
Perceptron
1957
ADALINE
1960
XOR Problem
1969
Golden Age
Dark Age ("AI Winter")
Multi-layered Perceptron (Backpropagation)
1986
SVM
1995
Deep Neural Network (Pretraining)
2006
1940
1950
1960
1970
1980
1990
2000
2010
S. McCulloch - W. Pitts
F. Rosenblatt
B. Widrow - M. Hoff
M. Minsky - S. Papert
D. Rumelhart - G. Hinton - R. Wiliams
V. Vapnik - C. Cortes
G. Hinton - S. Ruslan
X AND Y
X OR Y
NOT X
Foward Activity
Backward Error
• Adjustable Weights
• Weights are not Learned
• Learnable Weights and Threshold
• XOR Problem
• Solution to nonlinearly separable problems
• Big computation, local optima and overfitting
• Limitations of learning prior knowledge
• Kernel function: Human Intervention
• Hierarchical feature Learning


Evolution of Deep Learning Architectures
ResNet
2016
56 layers!
"Inception"
2015
GoogLeNet
2014
AlexNet
2012
LeNet
1998
Depth
Time

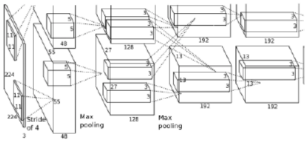
***Deep Learning* =**
Lots of training data* + *Parallel Computation* + *Scalable, smart algorithms

The Deep Learning "Computer Vision Recipe"



Big Data: ImageNet

+



Deep Convolutional Neural Network

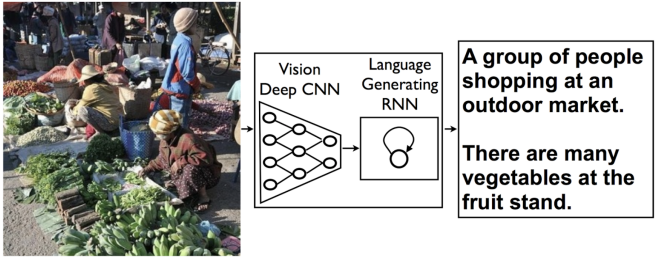
+

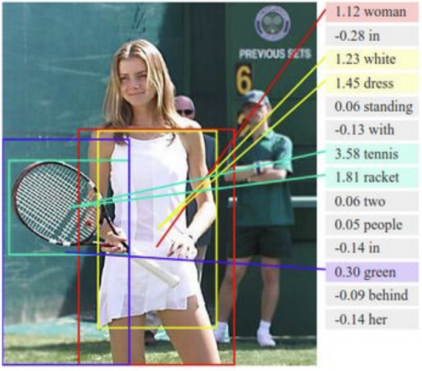
Backprop on GPU

=

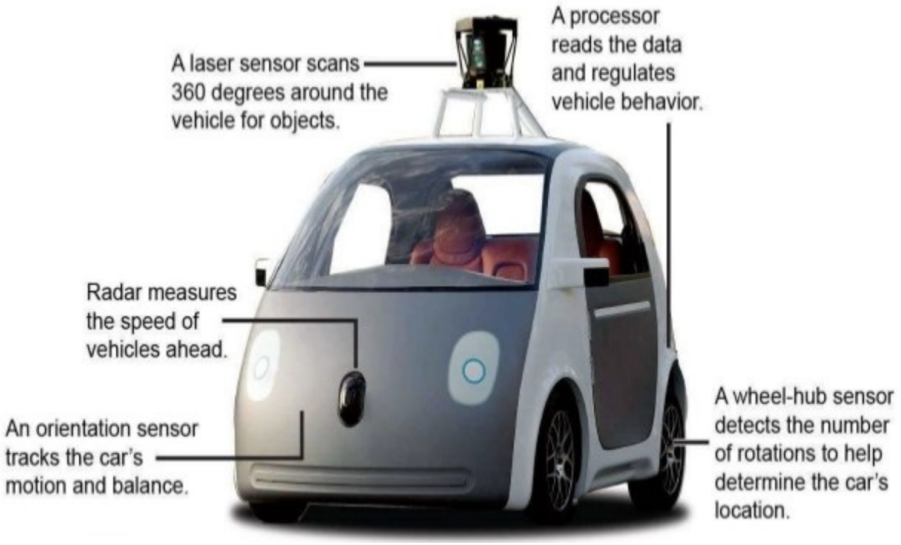
Learned Weights

Automatic picture captioning





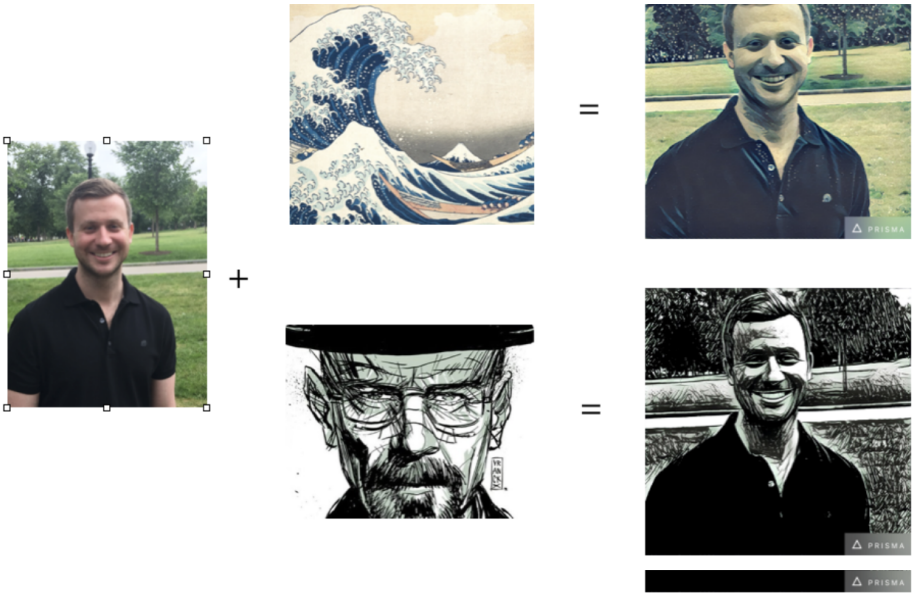
Autonomous car



Google Alpha Go

DEEP LEARNING HAS MASTERED GO

Style Transfer



- Yann LeCun (french)
  - Facebook AI (FAIR), University of New-York

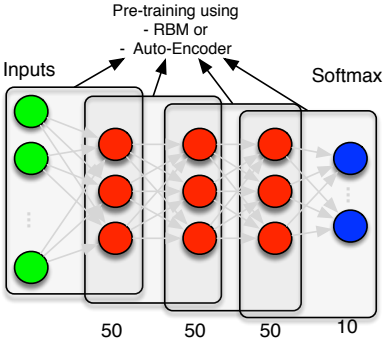
- Geoffrey Hinton (canadian)
  - Google Brain, University of Toronto

- Yoshua Bengio (french)
  - University of Montreal, MILA

Three main types of Nets

Multi Layers Perceptron (MLP), Fully-Connected (FC), Feed-Forward

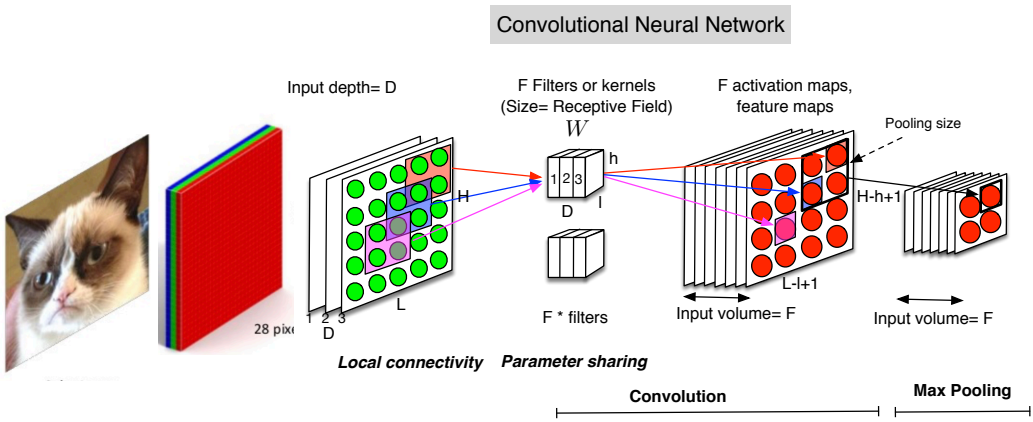
Multi Layers Perceptron (Fully Connected)



$$\underline{z}^{[l]} = \underline{\underline{W}}^{[l]} \underline{a}^{[l-1]} + \underline{b}^{[l]}$$

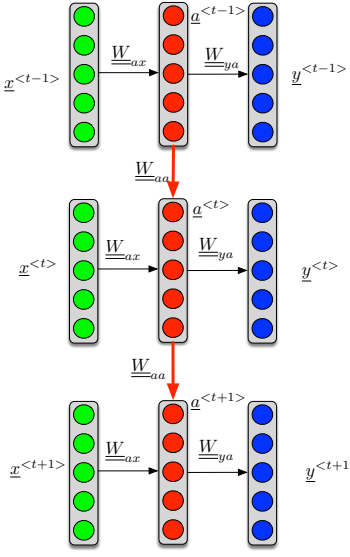
$$\underline{a}^{[l]} = g^{[l]}(\underline{z}^{[l]})$$

Convolutional Neural Networks (CNN)



Recurrent Neural Networks (RNN)

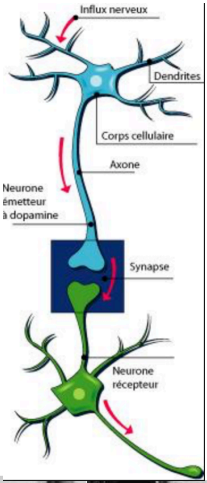
Recurrent Neural Network

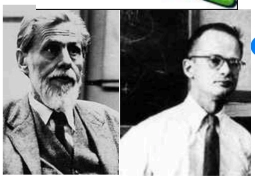


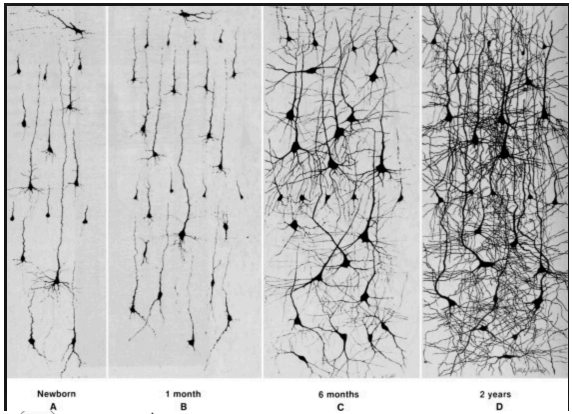
Perceptron

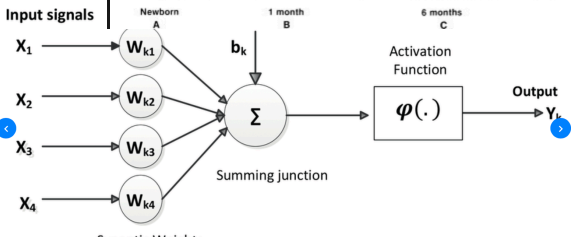
Mc Culloch - Pitts model of a neuron

Warren S. McCulloch and Walter H. Pitts "A logical calculus of the ideas immanent in nervous activity", Bulletin of Mathematical Biophysisc, vol. 5, 1943, p. 115-133









McCulloch-Pitts computational model of a neuron.

Perceptron model

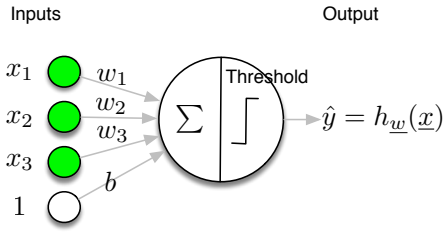
Frank Rosenblatt, "The perceptron : a perceiving and recognizing automation", Projet PARA, Cornell Aeronautical Laboratory, Inc. 1957

- Problem :
  - Given $\underline{x} \in \mathbb{R}^{n_x}$, predict $\hat{y} \in \{0, 1\}$
    - binary classification
- Model :
  - linear function followed by a threshold

    $$\hat{y} = h_{\underline{w}}(\underline{x})$$
  - Formulation 1

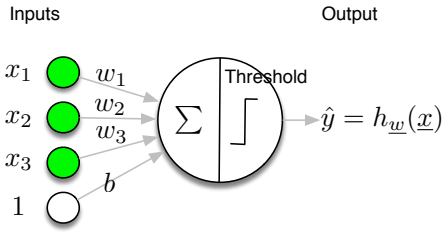
    $$\hat{y} = \text{Threshold}(\underline{x}^T \cdot \underline{w}) \quad \text{with } \underline{x}_0 = 1, \underline{w}_0 = b$$
  - Formulation 2

    $$\hat{y} = \text{Threshold}(\underline{x}^T \cdot \underline{w} + b)$$
  - where
    - $\text{Threshold}(z) = 1$ if $z \geqslant 0$
    - $\text{Threshold}(z) = 0$ if $z < 0$
- Parameters :
  - $\theta = \underline{w} \in \mathbb{R}^{n_x}, b \in \mathbb{R}$
    - $\underline{w}$: weights
    - b : bias



Perceptron training

- **Training ?**
  - Adapt $\underline{w}$ and b such that $\hat{y} = h_{\underline{w}}(\underline{x})$ equal y on training data
- **Algorithm :**
- $\forall i (\underline{x}^{(i)}, y^{(i)})$
  - compute $\hat{y}^{(i)} = h_{\underline{w}}(\underline{x}^{(i)}) = \text{Threshold}(\underline{x}^{(i)\,T} \cdot \underline{w})$
  - if $\hat{y}^{(i)} \neq y(t)^{(i)}$ update
    - $w_d \leftarrow w_d + \alpha(y^{(i)} - \hat{y}^{(i)}) x_d^{(i)} \qquad \forall d$
      - where α is the learning rate



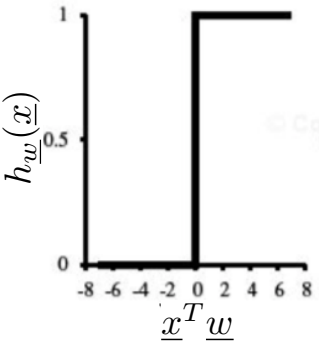
- **Three cases :**

$(y^{(i)} - \hat{y}^{(i)}) = 0$	$\Rightarrow$ no update	-
$(y^{(i)} - \hat{y}^{(i)}) = 1$	$\Rightarrow$ the weights are too low	$\Rightarrow$ increase w_d for positive $x_d^{(i)}$
$(y^{(i)} - \hat{y}^{(i)}) = -1$	$\Rightarrow$ the weights are too high	$\Rightarrow$ decrease w_d for positive $x_d^{(i)}$

Perceptron training (minimizing a Loss function)

- **Perceptron :**
  - predicts a class
  - model :
    - $\hat{y}^{(i)} = h_{\underline{w}}(\underline{x}^{(i)}) = \text{Threshold}(\underline{x}^{(i)\,T} \cdot \underline{w})$
  - update rule :
    - $w_d \leftarrow w_d + \alpha(y^{(i)} - \hat{y}^{(i)})x_d^{(i)} \qquad \forall d$
- It corresponds to **minimizing a Loss function**

$$\mathcal{L}(y^{(i)}, \hat{y}^{(i)}) = -(y^{(i)} - \hat{y}^{(i)})\underline{x}^{(i)\,T} \cdot \underline{w}$$



- Gradient descent ?

$$w_d \leftarrow w_d - \alpha \frac{\partial \mathcal{L}(y^{(i)}, \hat{y}^{(i)})}{\partial w_d} \qquad \forall d$$

- Gradient ?

$$\frac{\partial \mathcal{L}}{\partial w_d} = -\left(\frac{\partial \left(y^{(i)} - \hat{y}^{(i)}\right)}{\partial \underline{w}_d} \underline{x}^{(i)\,T} \cdot \underline{w} + \left(y^{(i)} - \hat{y}^{(i)}\right) \cdot x_d^{(i)} \right) \cong -\left(0 + \left(y^{(i)} - \hat{y}^{(i)}\right) \cdot x_d^{(i)}\right)$$

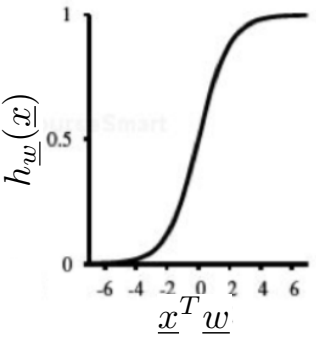
  - Not exactly, since the derivative of $\hat{y}^{(i)} = h_{\underline{w}}(\underline{x}^{(i)})$ does not exist at $\underline{w} \cdot \underline{x} = 0$
  - $\Rightarrow$ Using the Threshold function can lead to instability during training

From Perceptron to Logistic Regression

- **Logistic regression**
  - predicts a **probability to belong to this class**
  - model

$$\hat{y} = p(Y = 1|\underline{x}) = h_{\underline{w}}(\underline{x}) = \text{Logistic}(\underline{x}^T \cdot \underline{w}) = \frac{1}{1 + e^{-\underline{x}^T \cdot \underline{w}}}$$

  - decision : choose the class with the largest probability
    - If $h_{\underline{w}}(\underline{x}) \geqslant 0.5$ choose 1
    - If $h_{\underline{w}}(\underline{x}) < 0.5$ choose 0



- We define the Loss as (binary cross-entropy)

$$\mathcal{L}(y^{(i)}, \hat{y}^{(i)}) = -y^{(i)} \log(\hat{y}^{(i)}) - (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

- Gradient descent ?

$$w_d \leftarrow w_d - \alpha \frac{\partial \mathcal{L}(y^{(i)}, \hat{y}^{(i)})}{\partial w_d} \qquad \forall d$$

- Gradient ?

$$\frac{\partial \mathcal{L}(y^{(i)}, \hat{y}^{(i)})}{\partial w_d} = -(y^{(i)} - \hat{y}^{(i)}) \cdot x_d \qquad \forall d$$

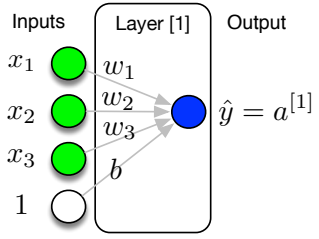
  - The update rule is therefore the same as for the Perceptron but the definition of the Loss and of $h_{\underline{w}}(\underline{x})$ is different

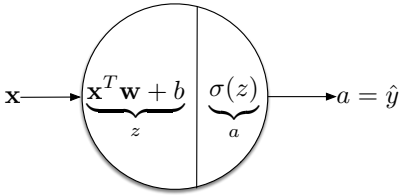
Logistic Regression (0 hidden layers)

$$\underline{x} \rightarrow \boxed{w_1 x_1 + w_2 x_2 + b} \xrightarrow{z} \boxed{\sigma(z)} \xrightarrow{a} \boxed{\mathcal{L}(\hat{y} = a, y)}$$

Logistic regression model

- Problem :
  - Given $\underline{x} \in \mathbb{R}^{n_x}$, predict $\hat{y} = P(y = 1|\underline{x}) \quad \hat{y} \in [0, 1]$
- Model :
  - $\hat{y} = \sigma(\underline{x}^T \underline{w} + b)$
  - Sigmoid function : $\sigma(z) = \frac{1}{1+e^{-z}}$
    - If z is very large then $\sigma(z) \simeq \frac{1}{1+0} = 1$
    - If z is very (negative) small then $\sigma(z) = 0$
- Parameters :
  - $\theta = \{\underline{w} \in \mathbb{R}^{n_x}, b \in \mathbb{R}\}$





Loss / Cost function (Empirical Risk Minimization)

- **Training data :**
  - Given $\{(\underline{x}^{(1)}, y^{(1)}), (\underline{x}^{(2)}, y^{(2)}), \ldots, (\underline{x}^{(m)}, y^{(m)})\}$,
  - We want to find the parameters θ of the network such that $\hat{y}^{(i)} \simeq y^{(i)}$

- How to measure ?
  - Define a **Loss $\mathcal{L}$ (error) function** which needs to be minimized
    - if $y \in \mathbb{R}$: **Mean Square Error (MSE)** $\mathcal{L}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$
    - if $y \in \{0, 1\}$: **Binary Cross-Entropy** $\mathcal{L}(\hat{y}, y) = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$
    - if $y \in \{1, \cdots, K\}$: **Cross-Entropy** $\mathcal{L}(\hat{y}, y) = -\sum_{c=1}^{K} (y_c \log(\hat{y}_c))$

- **Cost J function** : sum of the Loss for all training examples

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

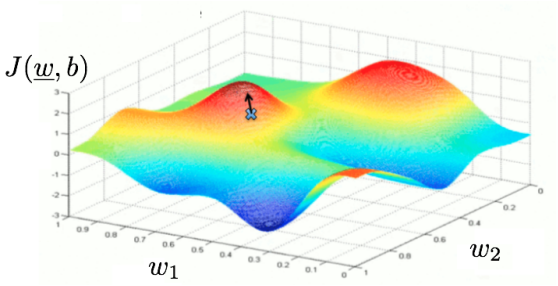
  - In the case of the Binary Cross-Entropy

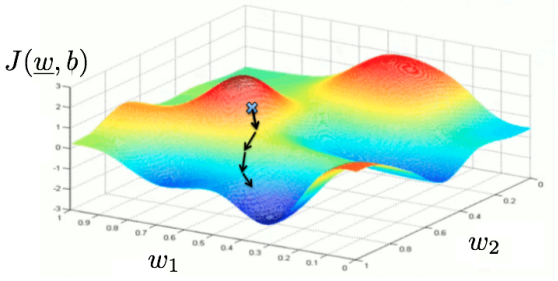
$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \left(y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right)$$

- We want to find the parameters θ of the network that minimize $J(\theta)$

Gradient Descent

- **How to minimize $J(\underline{w}, b)$?**
- The gradient $\frac{\partial J(\underline{w},b)}{\partial \underline{w}}$ points in the direction of the greatest rate of increase of the function
- We will go in the opposite direction : $-\frac{\partial J(\underline{w},b)}{\partial \underline{w}}$
  - We move down-hill in the steepest direction
- **Gradient descent :**
  - Repeat
    - $\underline{w} \leftarrow \underline{w} - \alpha \cdot \frac{\partial J(\underline{w},b)}{\partial \underline{w}}$
    - $b \leftarrow b - \alpha \cdot \frac{\partial J(\underline{w},b)}{\partial b}$
  - where α is the "learning rate"





Gradient Descent

- **Parameters :** to update
  - $\theta = \{\underline{w}, b\}$
- **Gradient Descent :**
  - (i) **Initialize** the parameters
  - Repeat for # iterations
    - Repeat for all training examples $\forall i = 1, \ldots, m$
      - (f) **Forward propagation** : compute prediction $\hat{y}^{(i)}$
      - (l) **Compute the loss** $\mathcal{L}_\theta(\hat{y}^{(i)}, y^{(i)})$
      - (b) **Backward propagation** : compute gradients $\frac{\partial \mathcal{L}}{\partial \underline{w}}$, $\frac{\partial \mathcal{L}}{\partial b}$
    - Compute the Cost : J_θ
    - (u) **Update the parameters** θ using the learning rate α

$$\underline{x} \rightarrow \boxed{w_1 x_1 + w_2 x_2 + b} \xrightarrow{z} \boxed{\sigma(z)} \xrightarrow{a} \boxed{\mathcal{L}(\hat{y} = a, y)}$$

ⓕ Forward propagation

$$z^{(i)} = w_1 x_1^{(i)} + w_2 x_2^{(i)} + b$$

$$a^{(i)} = \sigma(z^{(i)}) \text{ with } \sigma(z) = \frac{1}{1 + e^{-z}}$$

$$\hat{y}^{(i)} = a^{(i)}$$

$$\underline{x} \rightarrow \boxed{w_1 x_1 + w_2 x_2 + b} \xrightarrow{z} \boxed{\sigma(z)} \xrightarrow{a} \boxed{\mathcal{L}(\hat{y} = a, y)}$$

① Loss

$$\hat{y}^{(i)} = a^{(i)}$$

$$\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\left(y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})\right)$$

$$\underline{x} \rightarrow \boxed{w_1 x_1 + w_2 x_2 + b} \xrightarrow{z} \boxed{\sigma(z)} \xrightarrow{a} \boxed{\mathcal{L}(\hat{y} = a, y)}$$

ⓑ Backward propagation

(we ommit $^{(i)}$ in the following)

$$\frac{\partial \mathcal{L}}{\partial a} = -\left(\frac{y}{a} - \frac{1-y}{1-a}\right) = \frac{a-y}{a(1-a)}$$

$$\frac{\partial \mathcal{L}}{\partial z} = \frac{\partial \mathcal{L}}{\partial a}\frac{da}{dz} = \frac{a-y}{a(1-a)} \cdot a(1-a) = a - y$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial z}\frac{\partial z}{\partial w_1} = x_1 \cdot \frac{\partial \mathcal{L}}{\partial z}$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = \frac{\partial \mathcal{L}}{\partial z}\frac{\partial z}{\partial w_2} = x_2 \cdot \frac{\partial \mathcal{L}}{\partial z}$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial z}\frac{\partial z}{\partial b} = \frac{\partial \mathcal{L}}{\partial z}$$

Gradient of the Cost / Gradient of the Loss

- **For one training example i**
  - Forward propagation :

$$\hat{y}^{(i)} = a^{(i)} = \sigma(z^{(i)}) = \sigma(\underline{x}^{(i)\,T}\,\underline{w} + b)$$

  - Computing the Loss :

$$\mathcal{L}_\theta(\hat{y}^{(i)}, y^{(i)})$$

- **For all training examples**
  - Computing the Cost (sum of the Loss $\mathcal{L}$ over all training examples m)

$$J_\theta = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}_\theta(\hat{y}^{(i)}, y^{(i)})$$

- **Minimizing the Cost**
  - We need to compute the gradient of the Cost w.r.t. the parameters θ

$$\frac{\partial J_\theta}{\partial \theta_1} = \frac{\partial}{\partial \theta_1}\left(\frac{1}{m}\sum_{i=1}^{m} \mathcal{L}_\theta(\hat{y}^{(i)}, y^{(i)})\right) = \frac{1}{m}\sum_{i=1}^{m} \frac{\partial \mathcal{L}_\theta(\hat{y}^{(i)}, y^{(i)})}{\partial \theta_1}$$

  - $\Rightarrow$ gradient of the Cost = average (over the m training examples) of gradient of the Loss

$$\underline{x} \rightarrow \boxed{w_1 x_1 + w_2 x_2 + b} \xrightarrow{z} \boxed{\sigma(z)} \xrightarrow{a} \boxed{\mathcal{L}(\hat{y} = a, y)}$$

ⓤ Parameters update

- At iteration t :

$$w_1^{[t]} = w_1^{[t-1]} - \alpha \frac{\partial J}{\partial w_1}$$

$$w_2^{[t]} = w_2^{[t-1]} - \alpha \frac{\partial J}{\partial w_2}$$

$$b^{[t]} = b^{[t-1]} - \alpha \frac{\partial J}{\partial b}$$

where α is the learning rate

Example code in python

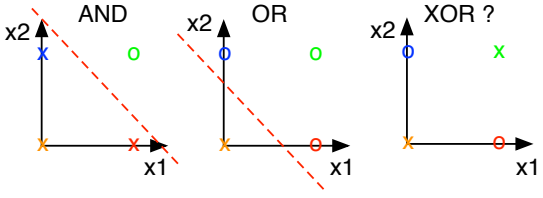
```
# --- initialisation
w1 =  np.random.randn(1)*0.01
w2 =  np.random.randn(1)*0.01
b = 0

# --- loop over epochs
for epoch in range(1000):
  J, dw1, dw2, db = 0, 0, 0, 0

  # --- loop over training examples
  for i in range(m):
     # --- forward
     z[i] = w1 * x[i,0] + w2 * x[i,1] + b
     a[i] = sigmoid(z[i])
     # --- cost
     J += -( y[i] * np.log(a[i]) + (1-y[i]) * np.log(1-a[i]) )
     # --- backward
     dz[i] = a[i] - y[i]
     dw1 += x[i,0] * dz[i]
     dw2 += x[i,1] * dz[i]
     db += dz[i]

  J /= m
  dw1 /= m
  dw2 /= m
  db /= m
  # --- parameters update
  w1 = w1 - alpha * dw1
  w2 = w2 - alpha * dw2
  b = b - alpha * dw2
```

- **Batch Gradient descent** $\Rightarrow$ the gradient is estimated using all m training examples
- One **Epoch** $\Rightarrow$ one pass over all training examples

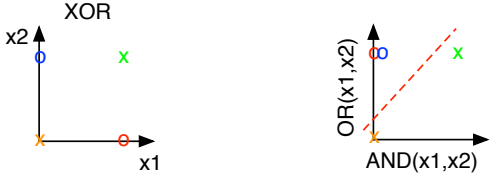
Limitation of linear classifiers

- Perceptron and Logistic Regression are linear classiers
  - What if classes are not linearly separable ?
  - What about the XOR function ?
    - Need 2 layers
      - $x_1^{[1]} = AND(x_1^{[0]}, x_2^{[0]})$
      - $x_2^{[1]} = OR(x_1^{[0]}, x_1^{[0]})$

- **1-Layer** :



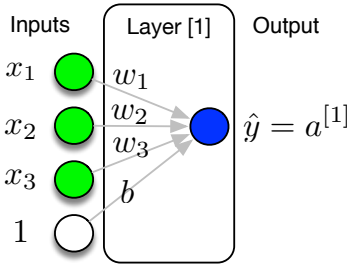
- **2-Layers** :



Neural Networks (1 hidden layer)

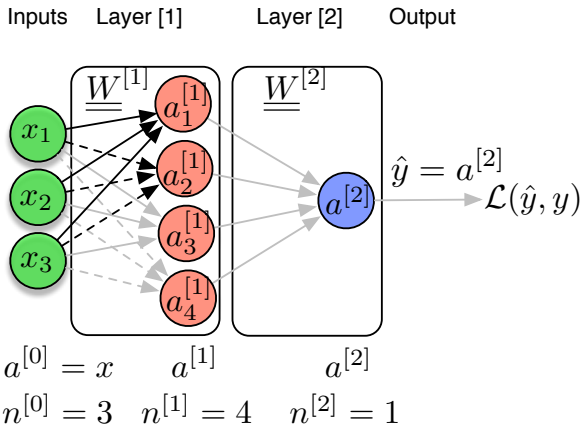
- **Logistic Regression** (1 layer, **0 hidden layer**)

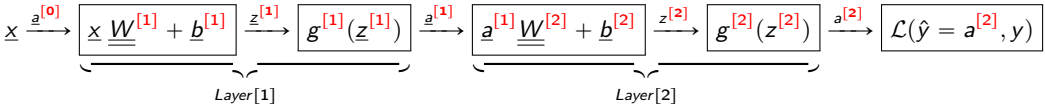
$$\underline{x} \rightarrow \boxed{w_1 x_1 + w_2 x_2 + b} \xrightarrow{z} \boxed{\sigma(z)} \xrightarrow{a} \boxed{\mathcal{L}(\hat{y} = a, y)}$$



- **Neural Network** (2 layers, **1 hidden layer**)

$$\underline{x} \xrightarrow{\underline{a}^{[0]}} \underbrace{\boxed{\underline{x}\,\underline{\underline{W}}^{[1]} + \underline{b}^{[1]}} \xrightarrow{\underline{z}^{[1]}} \boxed{g^{[1]}(\underline{z}^{[1]})}}_{Layer[1]} \xrightarrow{\underline{a}^{[1]}} \underbrace{\boxed{\underline{a}^{[1]}\,\underline{\underline{W}}^{[2]} + \underline{b}^{[2]}} \xrightarrow{z^{[2]}} \boxed{g^{[2]}(z^{[2]})}}_{Layer[2]} \xrightarrow{a^{[2]}} \boxed{\mathcal{L}(\hat{y} = a^{[2]}, y)}$$





Gradient Descent

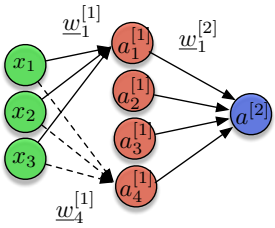
- **Parameters :** to update
  - $\theta = \{ \underset{(n^{[0]}, n^{[1]})}{\underline{\underline{W}}^{[1]}}, \underset{(1, n^{[1]})}{\underline{b}^{[1]}}, \underset{(n^{[1]}, n^{[2]})}{\underline{\underline{W}}^{[2]}}, \underset{(1, n^{[2]})}{\underline{b}^{[2]}} \}$
- **Gradient Descent :**
  - ⓘ **Initialize** the parameters θ
  - Repeat for # iterations
    - Repeat for all training examples $\forall i = 1, \ldots m$
      - ⓕ **Forward propagation** : compute prediction $\hat{y}^{(i)}$
      - ⓛ **Compute the Loss** $\mathcal{L}_\theta(\hat{y}^{(i)}, y^{(i)})$ (binary classification)
      - ⓑ **Backward propagation** : compute gradients $\frac{\partial \mathcal{L}}{\partial \underline{\underline{W}}^{[1]}}$, $\frac{\partial \mathcal{L}}{\partial \underline{b}^{[1]}}$, $\frac{\partial \mathcal{L}}{\partial \underline{\underline{W}}^{[2]}}$, $\frac{\partial \mathcal{L}}{\partial \underline{b}^{[2]}}$
    - Compute the Cost : J_θ
    - ⓤ **Update the parameters** θ using the learning rate α

$$\underline{x} \xrightarrow{\underline{a}^{[0]}} \underbrace{\boxed{\underline{x}\,\underline{\underline{W}}^{[1]} + \underline{b}^{[1]}} \xrightarrow{\underline{z}^{[1]}} \boxed{g^{[1]}(\underline{z}^{[1]})}}_{Layer[1]} \xrightarrow{\underline{a}^{[1]}} \underbrace{\boxed{\underline{a}^{[1]}\,\underline{\underline{W}}^{[2]} + \underline{b}^{[2]}} \xrightarrow{z^{[2]}} \boxed{g^{[2]}(z^{[2]})}}_{Layer[2]} \xrightarrow{a^{[2]}} \boxed{\mathcal{L}(\hat{y} = a^{[2]}, y)}$$

ⓕ Forward propagation (each dimension d, each training examples i)

`for i=1 to m`

- Input
  - $\underline{a}^{[0](i)} = \underline{x}^{(i)}$
- Output
  - $\hat{y}^{(i)} = \underline{a}^{[2](i)}$



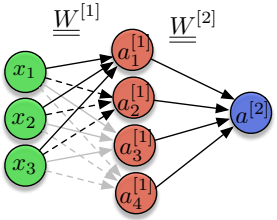
- Layer 1
  - `for d=1 to 4`
    - $z_d^{[1](i)} = \underline{a}^{[0](i)}\,\underline{w}_d^{[1]} + b_d^{[1]}$
    - $a_d^{[1](i)} = g^{[1]}(z_d^{[1](i)})$
- Layer 2
  - `for d'=1 to 1`
    - $z_{d'}^{[2](i)} = \underline{a}^{[1](i)}\,\underline{w}_{d'}^{[2]} + b_{d'}^{[2]}$
    - $a_{d'}^{[2](i)} = g^{[2]}(z_{d'}^{[2](i)})$

$$\underline{x} \xrightarrow{\underline{a}^{[0]}} \underbrace{\boxed{\underline{x}\,\underline{\underline{W}}^{[1]} + \underline{b}^{[1]}} \xrightarrow{\underline{z}^{[1]}} \boxed{g^{[1]}(\underline{z}^{[1]})}}_{Layer[1]} \xrightarrow{\underline{a}^{[1]}} \underbrace{\boxed{\underline{a}^{[1]}\,\underline{\underline{W}}^{[2]} + \underline{b}^{[2]}} \xrightarrow{z^{[2]}} \boxed{g^{[2]}(z^{[2]})}}_{Layer[2]} \xrightarrow{a^{[2]}} \boxed{\mathcal{L}(\hat{y} = a^{[2]}, y)}$$

ⓕ Forward propagation (all dimensions, each i training examples)

```
for i=1 to m
```

- Input
  - $\underset{(1,3)}{\underline{a}^{[0](i)}} = \underset{(1,3)}{\underline{x}^{(i)}}$
- Output
  - $\hat{y}^{(i)} = \underline{a}^{[2](i)}$



- Layer 1

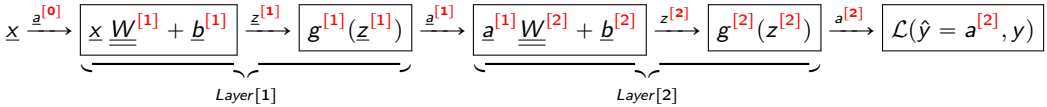
$$\underset{(1,4)}{\underline{z}^{[1](i)}} = \underset{(1,3)}{\underline{a}^{[0](i)}}\,\underset{(3,4)}{\underline{\underline{W}}^{[1]}} + \underset{(1,4)}{\underline{b}^{[1]}}$$

$$\underset{(1,4)}{\underline{a}^{[1](i)}} = g^{[1]}(\underset{(1,4)}{\underline{z}^{[1](i)}})$$

- Layer 2

$$\underset{(1,1)}{\underline{z}^{[2](i)}} = \underset{(1,4)}{\underline{a}^{[1](i)}}\,\underset{(4,1)}{\underline{\underline{W}}^{[2]}} + \underset{(1,1)}{\underline{b}^{[2]}}$$

$$\underset{(1,1)}{\underline{a}^{[2](i)}} = g^{[2]}(\underset{(1,1)}{\underline{z}^{[2](i)}})$$



ⓕ Forward propagation (all dimensions, all m training examples)

- Input
  - $\underline{\underline{A}}^{[0]} = \underline{\underline{X}}$
- Output
  - $\underline{\underline{\hat{Y}}} = \underline{\underline{A}}^{[2]}$

- Layer 1

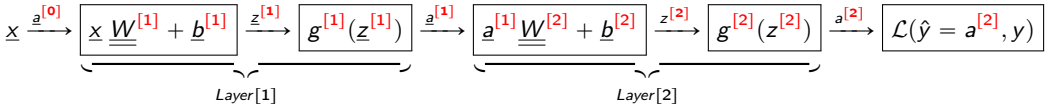
$$\underset{(m,n^{[1]})}{\underline{\underline{Z}}^{[1]}} = \underset{(m,n^{[0]})}{\underline{\underline{A}}^{[0]}} \underset{(n^{[0]},n^{[1]})}{\underline{\underline{W}}^{[1]}} + \underset{(n^{[1]})}{\underline{b}^{[1]}}$$

$$\underset{(m,n^{[1]})}{\underline{\underline{A}}^{[1]}} = g^{[1]}\left(\underline{\underline{Z}}^{[1]}\right)$$

- Layer 2

$$\underset{(m,n^{[2]})}{\underline{\underline{Z}}^{[2]}} = \underset{(m,n^{[1]})}{\underline{\underline{A}}^{[1]}} \underset{(n^{[1]},n^{[2]})}{\underline{\underline{W}}^{[2]}} + \underset{(n^{[2]})}{\underline{b}^{[2]}}$$

$$\underset{(m,n^{[2]})}{\underline{\underline{A}}^{[2]}} = g^{[2]}\left(\underline{\underline{Z}}^{[2]}\right)$$



ⓑ Backward propagation (each training example)

$$\frac{\partial \mathcal{L}}{\partial \underline{a}^{[2]}} = \text{derivative of the loss} = -\left(\frac{y}{a^{[2]}} + \frac{(1-y)}{(1-a^{[2]})}\right) = \frac{a^{[2]} - y}{a^{[2]}(1-a^{[2]})}$$

- Layer 2 (input $da^{[2]}$)

$$\frac{\partial \mathcal{L}}{\partial \underline{z}^{[2]}} = \frac{\partial \mathcal{L}}{\partial \underline{a}^{[2]}} \frac{\partial \underline{a}^{[2]}}{\partial \underline{z}^{[2]}} = \frac{\partial \mathcal{L}}{\partial \underline{a}^{[2]}} \odot g^{[2]\prime}(\underline{z}^{[2]})$$

$$= \frac{\partial \mathcal{L}}{\partial \underline{a}^{[2]}} \odot \underline{a}^{[2]}(1-\underline{a}^{[2]}) = \underline{a}^{[2]} - y$$

$$\underset{(n^{[1]}, n^{[2]})}{\frac{\partial \mathcal{L}}{\partial \underline{\underline{W}}^{[2]}}} = \frac{\partial \mathcal{L}}{\partial \underline{z}^{[2]}} \frac{\partial \underline{z}^{[2]}}{\partial \underline{\underline{W}}^{[2]}} = \underset{(\mathbf{1}, n^{[1]})}{\underline{a}^{[1]\,T}} \underset{(\mathbf{1}, n^{[2]})}{\frac{\partial \mathcal{L}}{\partial \underline{z}^{[2]}}}$$

$$\frac{\partial \mathcal{L}}{\partial \underline{b}^{[2]}} = \frac{\partial \mathcal{L}}{\partial \underline{z}^{[2]}} \frac{\partial \underline{z}^{[2]}}{\partial \underline{b}^{[2]}} = \frac{\partial \mathcal{L}}{\partial \underline{z}^{[2]}}$$

$$---$$

$$\frac{\partial \mathcal{L}}{\partial \underline{a}^{[1]}} = \frac{\partial \mathcal{L}}{\partial \underline{z}^{[2]}} \frac{\partial \underline{z}^{[2]}}{\partial \underline{a}^{[1]}} = \underset{(\mathbf{1}, n^{[2]})}{\frac{\partial \mathcal{L}}{\partial \underline{z}^{[2]}}} \underset{(n^{[1]}, n^{[2]})}{\underline{\underline{W}}^{[2]\,T}}$$

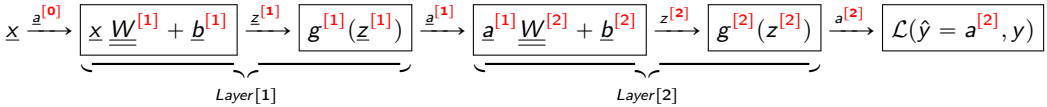
- Layer 1 (input $da^{[1]}$)

$$\frac{\partial \mathcal{L}}{\partial \underline{z}^{[1]}} = \frac{\partial \mathcal{L}}{\partial \underline{a}^{[1]}} \frac{\partial \underline{a}^{[1]}}{\partial \underline{z}^{[1]}} = \frac{\partial \mathcal{L}}{\partial \underline{a}^{[1]}} \odot g^{[1]\prime}(\underline{z}^{[1]})$$

$$\underset{(n^{[0]}, n^{[1]})}{\frac{\partial \mathcal{L}}{\partial \underline{\underline{W}}^{[1]}}} = \frac{\partial \mathcal{L}}{\partial \underline{z}^{[1]}} \frac{\partial \underline{z}^{[1]}}{\partial \underline{\underline{W}}^{[1]}} = \underset{(\mathbf{1}, n^{[0]})}{\underline{a}^{[0]\,T}} \underset{(\mathbf{1}, n^{[1]})}{\frac{\partial \mathcal{L}}{\partial \underline{z}^{[1]}}}$$

$$\frac{\partial \mathcal{L}}{\partial \underline{b}^{[1]}} = \frac{\partial \mathcal{L}}{\partial \underline{z}^{[1]}} \frac{\partial \underline{z}^{[1]}}{\partial \underline{b}^{[1]}} = \frac{\partial \mathcal{L}}{\partial \underline{z}^{[1]}}$$

- where $\odot$ is the element-wise (also named Hadamard) product.



ⓑ Backward propagation (all m training examples)

- Layer 2

$$\underset{(m,n^{[2]})}{\frac{\partial \mathcal{L}}{\partial \underline{\underline{Z}}^{[2]}}} = \underset{(m,n^{[2]})}{\underline{\underline{A}}^{[2]}} - \underset{(m,n^{[2]})}{\underline{\underline{Y}}}$$

$$\underset{(n^{[1]},n^{[2]})}{\frac{\partial \mathcal{L}}{\partial \underline{\underline{W}}^{[2]}}} = \frac{1}{m} \underset{(m,n^{[1]})}{\underline{\underline{A}}^{[1]T}} \underset{(m,n^{[2]})}{\frac{\partial \mathcal{L}}{\partial \underline{\underline{Z}}^{[2]}}}$$

$$\underset{(\mathbf{1},n^{[2]})}{\frac{\partial \mathcal{L}}{\partial \underline{b}^{[2]}}} = \frac{1}{m} \sum_{i=\mathbf{1}}^{m} \underset{(m,n^{[2]})}{\frac{\partial \mathcal{L}}{\partial \underline{\underline{Z}}^{[2]}}}$$

$$- - -$$

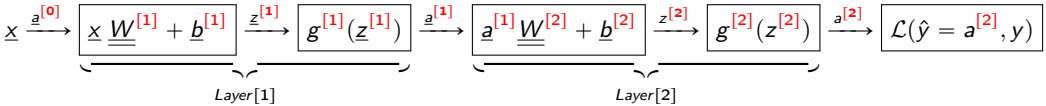
$$\underset{(m,n^{[1]})}{\frac{\partial \mathcal{L}}{\partial \underline{\underline{A}}^{[1]}}} = \underset{(m,n^{[2]})}{\frac{\partial \mathcal{L}}{\partial \underline{\underline{Z}}^{[2]}}} \underset{(n^{[1]},n^{[2]})}{\underline{\underline{W}}^{[2]T}}$$

- Layer 1

$$\underset{(m,n^{[1]})}{\frac{\partial \mathcal{L}}{\partial \underline{\underline{Z}}^{[1]}}} = \underset{(m,n^{[1]})}{\frac{\partial \mathcal{L}}{\partial \underline{\underline{A}}^{[1]}}} \odot g^{[1]\prime}\Big(\underset{(m,n^{[1]})}{\underline{\underline{Z}}^{[1]}}\Big)$$

$$\underset{(n^{[0]},n^{[1]})}{\frac{\partial \mathcal{L}}{\partial \underline{\underline{W}}^{[1]}}} = \frac{1}{m} \underset{(m,n^{[0]})}{\underline{\underline{A}}^{[0]T}} \underset{(m,n^{[1]})}{\frac{\partial \mathcal{L}}{\partial \underline{\underline{Z}}^{[1]}}}$$

$$\underset{(\mathbf{1},n^{[1]})}{\frac{\partial \mathcal{L}}{\partial \underline{b}^{[1]}}} = \frac{1}{m} \sum_{i=\mathbf{1}}^{m} \underset{(m,n^{[1]})}{\frac{\partial \mathcal{L}}{\partial \underline{\underline{Z}}^{[1]}}}$$



ⓤ Parameters update

- **Parameters update :**

$$\underline{\underline{W}}^{[l]} = \underline{\underline{W}}^{[l]} - \alpha \cdot \frac{\partial \mathcal{L}}{\partial \underline{\underline{W}}^{[l]}}$$

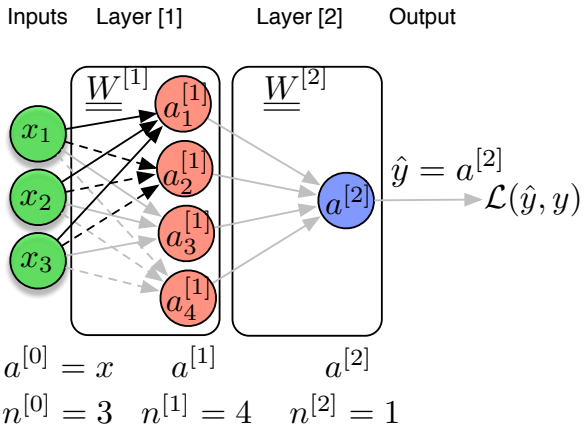
$$\underline{b}^{[l]} = \underline{b}^{[l]} - \alpha \cdot \frac{\partial \mathcal{L}}{\partial \underline{b}^{[l]}}$$

where α is the learning rate

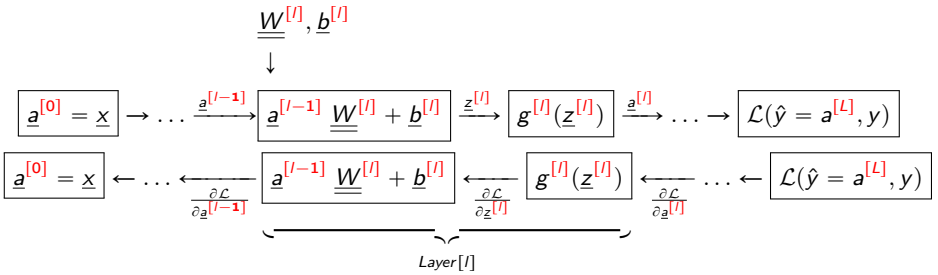
Deep Neural Networks (> 2 hidden layers)

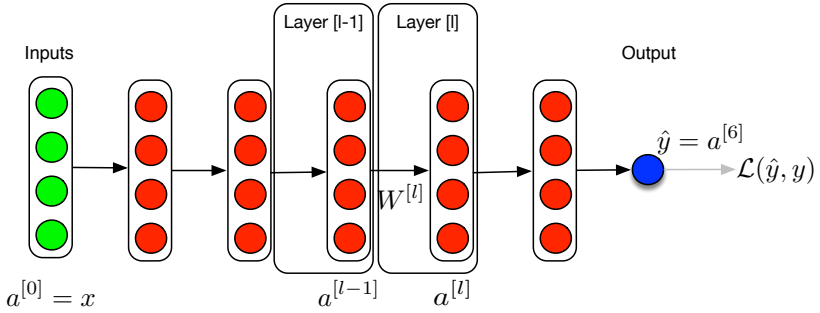
- **Neural Network** (2 layers, **1 hidden layer**)

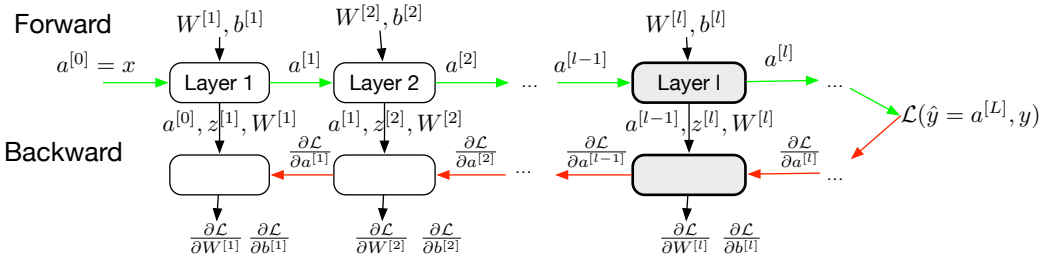
$$\underline{x} \xrightarrow{\underline{a}^{[0]}} \underbrace{\boxed{\underline{x}\,\underline{\underline{W}}^{[1]} + \underline{b}^{[1]}} \xrightarrow{\underline{z}^{[1]}} \boxed{g^{[1]}(\underline{z}^{[1]})}}_{Layer[1]} \xrightarrow{\underline{a}^{[1]}} \underbrace{\boxed{\underline{a}^{[1]}\,\underline{\underline{W}}^{[2]} + \underline{b}^{[2]}} \xrightarrow{z^{[2]}} \boxed{g^{[2]}(z^{[2]})}}_{Layer[2]} \xrightarrow{a^{[2]}} \boxed{\mathcal{L}(\hat{y} = a^{[2]}, y)}$$



- **Deep Neural Networks** (many layers, **>2 hidden layer**)







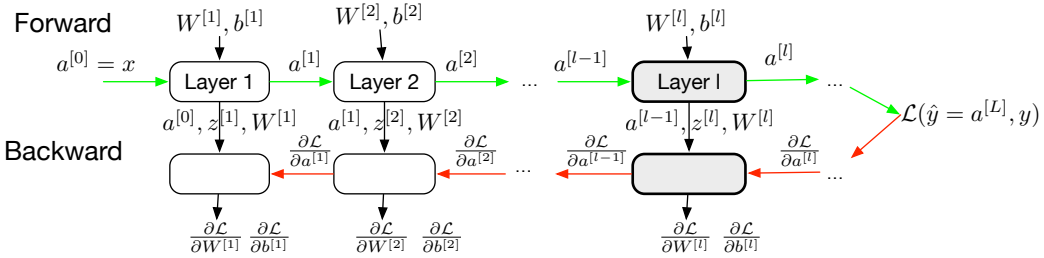
ⓕ Forward (general formulation for layer l, all m training examples)

Input : $\underset{\underline{\underline{}}}{A}^{[l-1]}$

$$\underset{(m,n^{[l]})}{\underline{\underline{Z}}^{[l]}} = \underset{(m,n^{[l-1]})}{\underline{\underline{A}}^{[l-1]}} \; \underset{(n^{[l-1]},n^{[l]})}{\underline{\underline{W}}^{[l]}} + \underset{(1,n^{[l]})}{\underline{b}^{[l]}}$$

$$\underset{(m,n^{[l]})}{\underline{\underline{A}}^{[l]}} = g^{[l]}(\underline{\underline{Z}}^{[l]})$$

Output : $\underline{\underline{A}}^{[l]}$ **Output for Backpropagation :** $\underline{\underline{A}}^{[l-1]}$, $\underline{\underline{Z}}^{[l]}$, $\underline{\underline{W}}^{[l]}$



ⓑ Backward (general formulation for layer l, all m training examples)

Input : $\frac{\partial \mathcal{L}}{\partial \underline{\underline{A}}^{[l]}}$ **Input from Forward :** $\underline{\underline{A}}^{[l-1]}$, $\underline{\underline{Z}}^{[l]}$, $\underline{\underline{W}}^{[l]}$

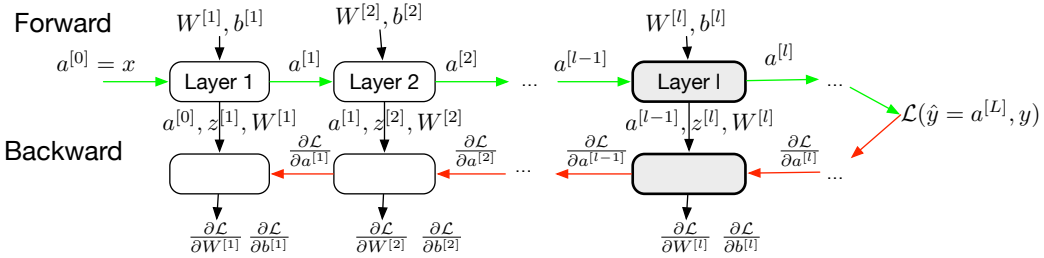
$$\frac{\partial \mathcal{L}}{\partial \underline{\underline{Z}}^{[l]}} = \frac{\partial \mathcal{L}}{\partial \underline{\underline{A}}^{[l]}} \odot g^{[l]\prime}(\underline{\underline{Z}}^{[l]}) = \left(\frac{\partial \mathcal{L}}{\partial \underline{\underline{Z}}^{[l+1]}} \, \underline{\underline{W}}^{[l+1]\,T} \right) \odot g^{[l]\prime}(\underline{\underline{Z}}^{[l]})$$

$$\frac{\partial \mathcal{L}}{\partial \underline{\underline{W}}^{[l]}} = \frac{1}{m} \underline{\underline{A}}^{[l-1]\,T} \, \frac{\partial \mathcal{L}}{\partial \underline{\underline{Z}}^{[l]}}$$

$$\frac{\partial \mathcal{L}}{\partial \underline{b}^{[l]}} = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial \underline{\underline{Z}}^{[l]}}$$

$$\frac{\partial \mathcal{L}}{\partial \underline{\underline{A}}^{[l-1]}} = \frac{\partial \mathcal{L}}{\partial \underline{\underline{Z}}^{[l]}} \, \underline{\underline{W}}^{[l]\,T}$$

Output for backpropagation : $\frac{\partial \mathcal{L}}{\partial \underline{\underline{A}}^{[l-1]}}$ **Output for parameters update :** $\frac{\partial \mathcal{L}}{\partial \underline{\underline{W}}^{[l]}}$, $\frac{\partial \mathcal{L}}{\partial \underline{b}^{[l]}}$



ⓤ Parameters update

$$\underline{\underline{W}}^{[l]} = \underline{\underline{W}}^{[l]} - \alpha \frac{\partial \mathcal{L}}{\partial \underline{\underline{W}}^{[l]}}$$

$$\underline{b}^{[l]} = \underline{b}^{[l]} - \alpha \frac{\partial \mathcal{L}}{\partial \underline{b}^{[l]}}$$

where α is the learning rate

Chain rule and Back-propagation

- **What is the chain rule ?**
  - formula for computing the derivative of the composition of two or more functions

$$\frac{d}{dt}f(x(t)) = \frac{df}{dx}\frac{dx}{dt}$$
$$\frac{d}{dt}f(x(t),y(t)) = \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt}$$

  - Example 1 :

$$z_1 = z_1(x_1, x_2)$$
$$z_2 = z_2(x_1, x_2)$$
$$p = p(z_1, z_2)$$
$$\frac{\partial p}{\partial x_1} = \frac{\partial p}{\partial z_1}\frac{\partial z_1}{\partial x_1} + \frac{\partial p}{\partial z_2}\frac{\partial z_2}{\partial x_1}$$

  - Example 2 :

$$h(x) = f(x)g(x)$$
$$\frac{\partial h}{\partial x} = \frac{\partial h}{\partial f}\frac{\partial f}{\partial x} + \frac{\partial h}{\partial g}\frac{\partial g}{\partial x} = f'g + fg'$$

- **What is back-propagation ?**
  - an efficient algorithm to compute the chain-rule by storing intermediate (and re-use derivatives)

Ex 1 : Logisitic regression / least square (single output)

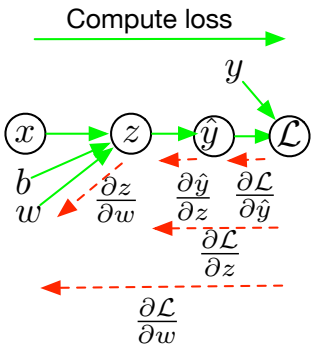
$$z = xw + b$$
$$\hat{y} = a = \sigma(z)$$
$$\mathcal{L} = \frac{1}{2}(\hat{y} - y)^2$$

- **Computing derivative as in calculus class**

$$\mathcal{L} = \frac{1}{2}(\sigma(wx + b) - y)^2$$
$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial}{\partial w}\left[\frac{1}{2}(\sigma(wx + b) - y)^2\right]$$
$$= \frac{1}{2}\frac{\partial}{\partial w}(\sigma(wx + b) - y)^2$$
$$= (\sigma(wx + b) - y)\frac{\partial}{\partial w}(\sigma(wx + b) - y)$$
$$= (\sigma(wx + b) - y)\sigma'(wx + b)\frac{\partial}{\partial w}(wx + b)$$
$$= (\sigma(wx + b) - y)\sigma'(wx + b)x$$

Ex 1 : Logisitic regression / least square (single output)

$$z = xw + b$$
$$\hat{y} = a = \sigma(z)$$
$$\mathcal{L} = \frac{1}{2}(\hat{y} - y)^2$$



- **Computing derivative using back-propagation**

$$\frac{\partial \mathcal{L}}{\partial \hat{y}} = \hat{y} - y$$
$$\frac{\partial \mathcal{L}}{\partial z} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \sigma'(z)$$
$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial z} \frac{\partial z}{\partial w} = \frac{\partial \mathcal{L}}{\partial z} x$$
$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial z} \frac{\partial z}{\partial b} = \frac{\partial \mathcal{L}}{\partial z}$$

- We can diagram out the computations using a **computation graph**
  - nodes represent all the inputs and computed quantities,
  - edges represent which nodes are computed directly as a function of which other nodes

Ex 2 : MLP / least square (1 hidden layer, multiple outputs)

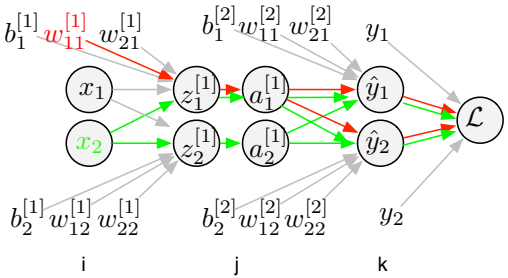
$$z_j^{[1]} = \sum_i x_i w_{ij}^{[1]} + b_j^{[1]}$$

$$a_j^{[1]} = \sigma(z_j^{[1]})$$

$$z_k^{[2]} = \sum_j a_j^{[1]} w_{jk}^{[2]} + b_k^{[2]}$$

$$\hat{y}_k = a_k^{[2]} = z_k^{[2]}$$

$$\mathcal{L} = \frac{1}{2} \sum_k (\hat{y}_k - y_k)^2$$



- **Computing derivative using back-propagation**
  - How much changing w_{11} or x_2 affect $\mathcal{L}$
    - We need to take into account all possible paths

$$\frac{\partial \mathcal{L}}{\partial \hat{y}_k} = \hat{y}_k - y_k$$

$$\frac{\partial \mathcal{L}}{\partial w_{jk}^{[2]}} = \frac{\partial \mathcal{L}}{\partial \hat{y}_k} a_j^{[1]} \qquad \frac{\partial \mathcal{L}}{\partial b_k^{[2]}} = \frac{\partial \mathcal{L}}{\partial \hat{y}_k}$$

$$\frac{\partial \mathcal{L}}{\partial a_j^{[1]}} = \sum_k \frac{\partial \mathcal{L}}{\partial \hat{y}_k} w_{jk}^{[2]}$$

$$\frac{\partial \mathcal{L}}{\partial z_j^{[1]}} = \frac{\partial \mathcal{L}}{\partial a_j^{[1]}} \sigma'(z_j)$$

$$\frac{\partial \mathcal{L}}{\partial w_{ij}^{[1]}} = \frac{\partial \mathcal{L}}{\partial z_j^{[1]}} x_i \qquad \frac{\partial \mathcal{L}}{\partial b_j^{[1]}} = \frac{\partial \mathcal{L}}{\partial z_j^{[1]}}$$

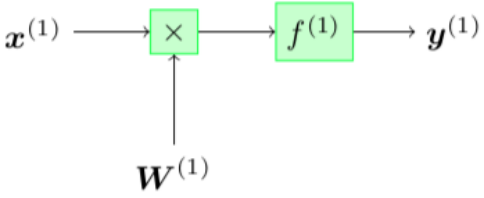
Computation Graph

(slide from from Alexandre Allauzen)

A convenient way to represent a complex mathematical expressions :

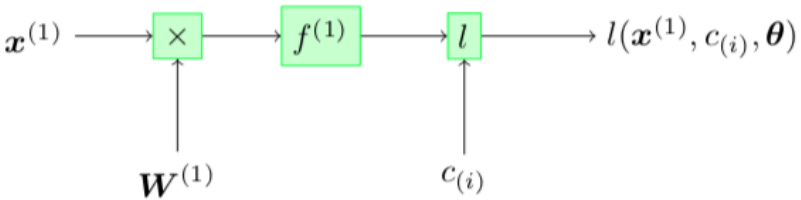
- each node is an operation or a variable
- an operation has some inputs / outputs made of variables

Example 1 : A single layer network



- Setting $x^{(1)}$ and $W^{(1)}$
- Forward pass $\rightarrow y^{(1)}$

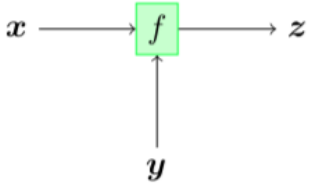
$$y^{(1)} = f^{(1)}(W^{(1)}x^{(1)})$$



- A variable node encodes the label
- To compute the output for a given input
  → forward pass
- To compute the gradient of the loss *wrt* the parameters ($\boldsymbol{W}^{(1)}$)
  → backward pass

A function node

Forward pass

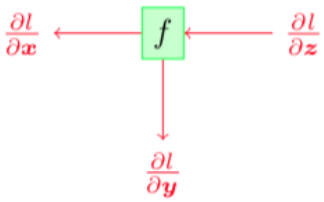


This node implements :

$$z = f(x, y)$$

A function node - 2

Backward pass



A function node knows :

- the "local gradients" computation

$$\frac{\partial z}{\partial x}, \frac{\partial z}{\partial y}$$

- how to return the gradient to the inputs :

$$(\frac{\partial l}{\partial z}\frac{\partial z}{\partial x}), (\frac{\partial l}{\partial z}\frac{\partial z}{\partial y})$$

Summary of a function node

f :

$$\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \quad \text{\# store the values}$$

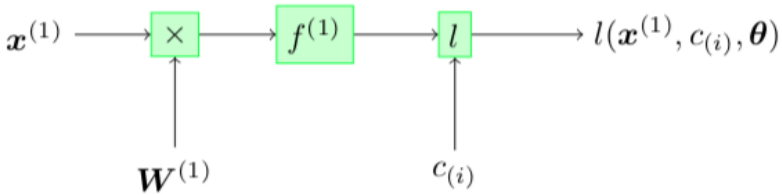
$$\boldsymbol{z} = f(\boldsymbol{x}, \boldsymbol{y}) \quad \text{\# forward}$$

$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} \rightarrow \frac{\partial f}{\partial \boldsymbol{x}} \quad \text{\# local gradients}$$

$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{y}} \rightarrow \frac{\partial f}{\partial \boldsymbol{y}}$$

$$(\frac{\partial l}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}), (\frac{\partial l}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{y}}) \quad \text{\# backward}$$

Example of a single layer network



Forward

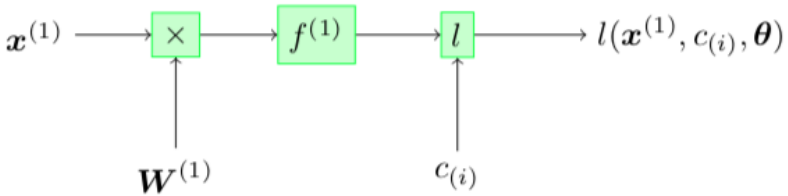
For each function node in topological order

- forward propagation

Which means :

1. $a^{(1)} = W^{(1)} x^{(1)}$
2. $y^{(1)} = f^{(1)}(a^{(1)})$
3. $l(y^{(1)}, c_{(i)})$

Example of a single layer network



Backward

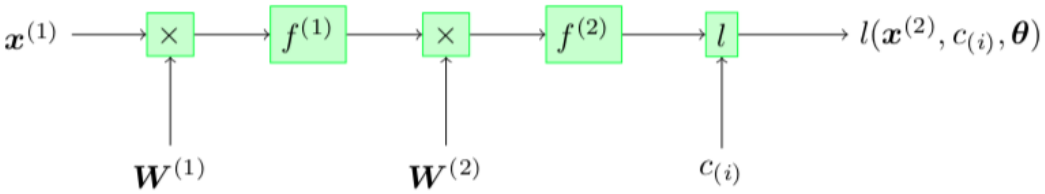
For each function node in reversed topological order

- backward propagation

Which means :

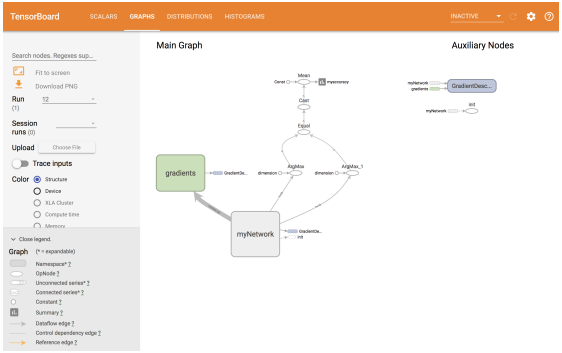
1. $\nabla_{y^{(1)}}$
2. $\nabla_{a^{(1)}}$
3. $\nabla_{W^{(1)}}$

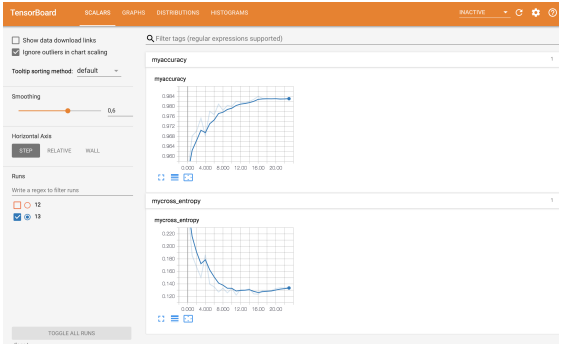
Example of a two layers network



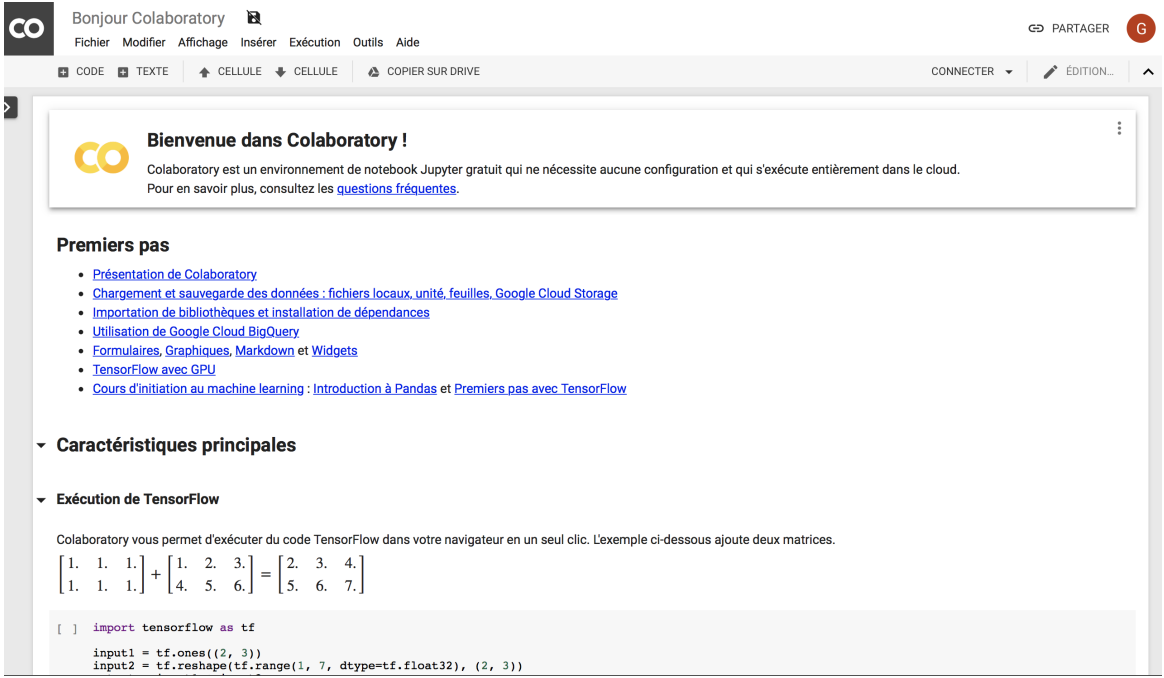
- The algorithms remain the same,
- even for more complex architectures
- Generalization by coding your own function node or by
- Wrapping a layer in a module

Tensorboard

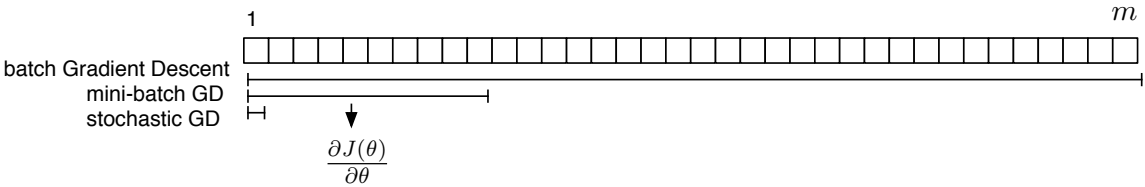




https://colab.research.google.com/



Various types of training



- m training examples :
  - $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$
- Various types of training :
  - Batch Gradient Descent
  - Mini Batch Gradient Descent
  - Stochastic Gradient Descent (SGD)

Alternatives to gradient descent

(mini-batch) Gradient Descent

- Notations :
  - iteration : t
  - parameter at iteration t : $\theta^{[t]}$,
    - θ can be either $\underline{\underline{W}}$ (weight matrix) or $\underline{b}$ (bias)
  - gradient of the loss with respect to θ
    - $\frac{\partial \mathcal{L}(\theta, x, y)}{\partial \theta}$
- **Mini-batch Gradient Descent**

$$\theta^{[t]} = \theta^{[t-1]} - \alpha \frac{\partial \mathcal{L}(\theta^{[t-1]}, x^{(i:i+n)}, y^{(i:i+n)})}{\partial \theta}$$

- **Problems :**
  - does not guarantee good convergence
  - neural network = highly non-convex error functions
    - avoid getting trapped in their numerous suboptimal local minima or saddle points (points where one dimension slopes up and another slopes down) which are usually surrounded by a plateau
  - choosing a proper learning rate can be difficult, need to adapt the learning rate to dataset's characteristics
  - each parameter may require a different learning rate (sparse data)
- **Alternatives to Gradient Descent**
  - first-order methods : Momentum, Nesterov (NAG), Adagrad, Adadelta/RMSprop, Adam
  - second-order methods : Newton

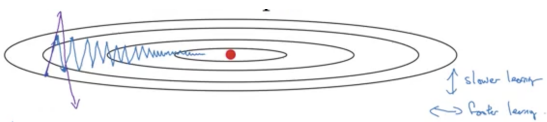
Momentum

- Goal ?
  - helps accelerating gradient descent in the relevant direction and dampens oscillations
- How ?
  - add a fraction β of the update vector of the past time step to the current step
- **Momentum**
  - On iteration t, compute $\frac{\partial \mathcal{L}(\theta(t-1),x,y)}{\partial \theta}$ on current mini-batch

$$V_{d\theta}^{[t]} = \beta V_{d\theta}^{[t-1]} + (1-\beta)\frac{\partial \mathcal{L}(\theta^{[t-1]}, x, y)}{\partial \theta}$$
$$\theta^{[t]} = \theta^{[t-1]} - \alpha V_{d\theta}^{[t]}$$

  - usual choice : $\beta = 0.9$
- Explanation : the momentum term
  - increases for dimensions whose gradients point in the same directions
  - reduces updates for dimensions whose gradients change directions
  - gain faster convergence and reduced oscillation

- β plays the role of a friction parameter
  - $V_{d\theta}$ plays the role of velocity
  - $\frac{\partial \mathcal{L}}{\partial \theta}$ plays the role of acceleration



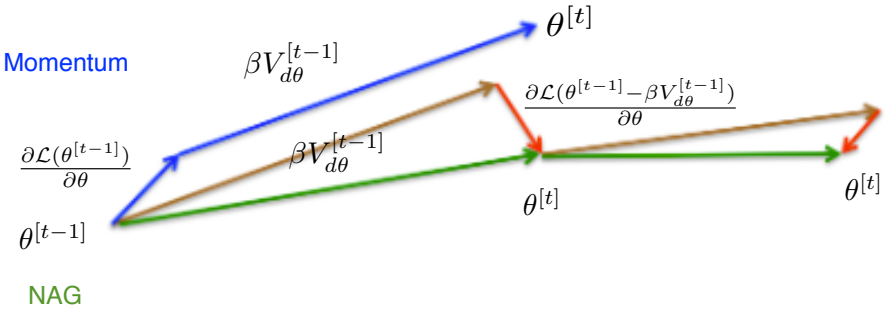
Nesterov Accelerated Gradient (NAG)

- Problem :
  - "A ball that rolls down a hill, blindly following the slope, is highly unsatisfactory."
- Solution :
  - "We'd like to have a smarter ball, a ball that has a notion of where it is going so that it knows to slow down before the hill slopes up again.""
  - We know that we will use our momentum term $\beta V_{d\theta}^{[t-1]}$ to move θ
    - $\theta^{[t]} = \theta^{[t-1]} - \alpha[\beta V_{d\theta}^{[t-1]} + (1-\beta)\frac{\partial \mathcal{L}}{\partial \theta}]$
  - We therefore compute the derivative of the loss at $\theta^{[t-1]} - \alpha\beta V_{d\theta}^{[t-1]}$ instead of $\theta^{[t-1]}$
  - This gives us an approximation of the next position of θ.
- **Nesterov Accelerated Gradient**
  - On iteration t

$$V_{d\theta}^{[t]} = \beta V_{d\theta}^{[t-1]} + (1-\beta)\frac{\partial \mathcal{L}(\theta^{[t-1]} - \alpha\beta V_{d\theta}^{[t-1]}, x, y)}{\partial \theta}$$
$$\theta^{[t]} = \theta^{[t-1]} - \alpha V_{d\theta}^{[t]}$$

Nesterov Accelerated Gradient (NAG)

- **Momentum**
  - 1) computes the current gradient $\frac{\partial \mathcal{L}(\theta^{[t-\mathbf{1}]})}{\partial \theta}$
  - 2) big jump in the direction of the previous accumulated gradient $\beta V_{d\theta}^{[t-\mathbf{1}]}$
- **Nesterov Accelerated Gradient (NAG)**
  - 1) big jump in the direction of the previous accumulated gradient $\beta V_{d\theta}^{[t-\mathbf{1}]}$
  - 2) measures the gradient $\frac{\partial \mathcal{L}(\theta^{[t-\mathbf{1}]} - \alpha\beta V_{d\theta}^{[t-\mathbf{1}]})}{\partial \theta}$
  - 3) makes a correction
- Prevents us from going too fast and results in increased responsiveness



AdaGrad

- Goal :
  - adapt the updates to each individual parameters
    - we want smaller update (lower learning rate) for frequently occuring features
    - we want larger update (higher learning rate) for infrequent features
- Notation :
  - $d\theta_i^{[t]} = \frac{\partial \mathcal{L}(\theta^{[t]}, x, y)}{\partial \theta_i}$
- SGD :
  - $\theta_i^{[t]} = \theta_i^{[t-1]} - \alpha d\theta_i^{[t-1]}$
- Compute the past gradients that have been computed for θ_i : $G_{i,i}^{[t]}$

$$G_{i,i}^{[t]} = \sum_{\tau=0}^{t} d\theta_i^{[\tau]^2}$$

- **Adagrad :**

$$G_{i,i}^{[t]} = \sum_{\tau=0}^{t} d\theta_i^{[\tau]^2}$$
$$\theta_i^{[t]} = \theta_i^{[t-1]} - \frac{\alpha}{\sqrt{G_{i,i}^{[t-1]} + \epsilon}} g_i^{[t-1]}$$

- Problem
  - the gradient are accumulated since the beginning
    - the learning rates will schrink

Adadelta

- Extension of Adagrad : instead of accumulating all past squared gradients (as in Adagrad) in Adadelta we restrict the window of accumulated past gradients to some fixed size
- **Adadelta**

$$\mathbb{E}[d\theta^2]^{[t]} = \gamma \mathbb{E}[d\theta^2]^{[t-1]} + (1-\gamma) d\theta^{[t-1]2} \quad (1)$$

$$\theta^{[t]} = \theta^{[t-1]} - \frac{\alpha}{\sqrt{\mathbb{E}[g^2]^{[t]} + \epsilon}} d\theta^{[t-1]} \quad (2)$$

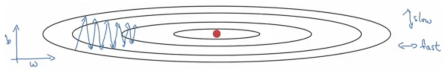
RMSprop (Root Mean Square prop)

- **RMSprop**
  - On iteration t compute $d\theta$ on current mini-batch

$$S_{d\theta}^{[t]} = \gamma S_{d\theta}^{[t-1]} + (1-\gamma) d\theta^{[t-1]2}$$

$$\theta^{[t]} = \theta^{[t-1]} - \frac{\alpha}{\sqrt{S_{d\theta}^{[t]} + \epsilon}} d\theta^{[t-1]}$$

- Want speed up in horizontal (W)
  - $S_{dW}^{[t]}$ small $\Rightarrow \frac{1}{\sqrt{S_{dW}^{[t]}+\epsilon}}$ large $\Rightarrow$ speed up
- Want slow down (damping) oscillation in vertical (b)
  - $S_{db}^{[t]}$ large $\Rightarrow \frac{1}{\sqrt{S_{db}^{[t]}+\epsilon}}$ small $\Rightarrow$ slow down



Adam (Adaptive moment estimation)

- Adam = Momentum with Adadelta/RMSprop
- **Adam**
  - On iteration t compute $d\theta$ on current mini-batch

$$V_{d\theta}^{[t]} = \beta_1 V_{d\theta}^{[t-1]} + (1-\beta_1) d\theta^{[t-1]}$$
$$S_{d\theta}^{[t]} = \beta_2 S_{d\theta}^{[t-1]} + (1-\beta_2) d\theta^{[t-1]2}$$
$$\theta^{[t]} = \theta^{[t-1]} - \alpha \frac{V_{d\theta}^{[t]}}{\sqrt{S_{d\theta}^{[t]} + \epsilon}}$$

- Hyperparameters
  - α : learning rate (needs to be tuned)
  - $\beta_1 = 0.9$ (momentum term, first moment)
  - $\beta_2 = 0.999$ (RMSprop term, second moment)
  - $\epsilon = 10^{-8}$ (avoid divide-by-zero)

Loss functions

Loss function : Binary Cross-Entropy

- The ground-truth output y is a **binary** variable $\in \{0, 1\}$
  - y follows a **Bernoulli distribution** : $P(Y = y) = p^y(1-p)^{1-y} \quad y \in \{0, 1\}$

$$\begin{cases} P(Y = 1) = p \\ P(Y = 0) = 1 - p \end{cases}$$

- In logisitic regression, the output of the network $\hat{y}$ estimates p : $\hat{y} = P(Y = 1|x, \theta)$

$$P(Y = y|x, \theta) = \hat{y}^y(1-\hat{y})^{1-y} \quad y \in \{0, 1\}$$

- We want to
  - find the θ that ... maximize the **likelihood** of the y given the input x

$$\max_\theta P(Y = y|x, \theta) = \hat{y}^y(1-\hat{y})^{1-y} \quad y \in \{0, 1\}$$

  - ... that maximize the **log-likelihood**

$$\max_\theta \log p(y|x) = \log(\hat{y}^y \ (1-\hat{y})^{(1-y)}) = y \log(\hat{y}) + (1-y)\log(1-\hat{y}) = -\mathcal{L}(\hat{y}, y)$$

  - ... that minimize the **binary cross-entropy** (minimize the loss)

$$\min_\theta \boxed{\mathcal{L}(\hat{y}, y) = -\left(y \log(\hat{y}) + (1-y)\log(1-\hat{y})\right)}$$

- ... maximizing **log-likelihood** on the whole training set

$$p(\text{labels }) = \prod_{i=1}^{m} p(y^{(i)}|x^{(i)})$$

$$\log p(\text{labels }) = \log\left(\prod_{i=1}^{m} p(y^{(i)}|x^{(i)})\right) = \sum_{i=1}^{m} \log p(y^{(i)}|x^{(i)}) = \sum_{i=1}^{m} -\mathcal{L}(y^{(i)}, y^{(i)})$$

  - ... is equivalent to minimizing the **Cost** $J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(y^{(i)}|x^{(i)})$

Activation functions $a = g(z)$

Output activation function : sigmoid/ logistic

- **Usage** : binary classification (0 or 1)
  - (logistic regression or deep neural network with binary output)

$$\begin{cases} P(y=1|\underline{x}) = \sigma(\underline{x}^T \underline{w}) = \boxed{\dfrac{1}{1+e^{-(\underline{x}^T \underline{w})}}} \\ P(y=0|\underline{x}) = 1 - P(y=1|\underline{x}) \end{cases}$$

- **Models the log-odds** $\log \frac{p}{1-p}$ (posterior probability) of the value "1" using a linear model of the inputs $\underline{x}$

$$P(y=1|\underline{x}) = \frac{1}{1+e^{-\underline{x}^T \underline{w}}}$$

$$\frac{1}{P(y=1|\underline{x})} = 1 + e^{-\underline{x}^T \underline{w}}$$

$$\frac{1 - P(y=1|\underline{x})}{P(y=1|\underline{x})} = e^{-\underline{x}^T \underline{w}}$$

$$\log\left(\frac{P(y=0|\underline{x})}{P(y=1|\underline{x})}\right) = -\underline{x}^T \underline{w}$$

$$\log\left(\frac{P(y=1|\underline{x})}{P(y=0|\underline{x})}\right) = \underline{x}^T \underline{w}$$

Output activation function : softmax

- **Usage** : multi-class classification $(1 \cdots K)$
  - (softmax regression or deep neural network with several mutually exclusive outputs)

$$P(y=1|\underline{x}) = \frac{e^{\underline{w}_1 \cdot \underline{x}}}{\sum_{c=1}^{K} e^{\underline{w}_c \cdot \underline{x}}}$$

$$\cdots$$

$$P(y=o|\underline{x}) = \boxed{\frac{e^{\underline{w}_o \cdot \underline{x}}}{\sum_{c=1}^{K} e^{\underline{w}_c \cdot \underline{x}}}}$$

$$\cdots$$

$$P(y=K|\underline{x}) = \frac{e^{\underline{w}_K \cdot \underline{x}}}{\sum_{c=1}^{K} e^{\underline{w}_c \cdot \underline{x}}}$$

- Has a "redundant" set of parameters
  - if we subtract some fixed vector ψ from each $\underline{w}_o$, we get the same results

$$P(y=o|\underline{x}) = \frac{e^{(\underline{w}_o - \psi)\cdot \underline{x}}}{\sum_{c=1}^{K} e^{(\underline{w}_c - \psi)\cdot \underline{x}}} = \frac{e^{\underline{w}_o \cdot \underline{x}} \; e^{-\psi \cdot \underline{x}}}{\sum_{c=1}^{K} e^{\underline{w}_c \cdot \underline{x}} \; e^{-\psi \cdot \underline{x}}} = \frac{e^{\underline{w}_o \cdot \underline{x}}}{\sum_{c=1}^{K} e^{\underline{w}_c \cdot \underline{x}}}$$

  - Common choice : $\psi = \underline{w}_K$ hence $e^{(\underline{w}_K - \psi)\cdot \underline{x}} = 1$

$$P(y=o|\underline{x}) = \frac{e^{\underline{w}_o \underline{x}}}{1 + \sum_{c=1}^{K-1} e^{\underline{w}_c \underline{x}}} \quad \forall o \in [1 \ldots K-1]$$

$$P(y=K|\underline{x}) = \frac{1}{1 + \sum_{c=1}^{K-1} e^{\underline{w}_c \cdot \underline{x}}}$$

Output activation function : softmax

- We of course have
  $$P(y = 1|\underline{x}) + P(y = 2|\underline{x}) + ... + P(y = K|\underline{x}) = 1$$
- **Models the log-odds** $\log \frac{p_c}{p_K}$ (posterior probability) using linear models of the inputs $\underline{x}$

$$\log\left(\frac{P(y = 1|\underline{x})}{P(y = K|\underline{x})}\right) = \underline{w_1}\underline{x}$$

$$\dots$$

$$\log\left(\frac{P(y = o|\underline{x})}{P(y = K|\underline{x})}\right) = \underline{w_o}\underline{x}$$

$$\dots$$

$$\log\left(\frac{P(y = K-1|\underline{x})}{P(y = K|\underline{x})}\right) = \underline{w}_{K-1}\underline{x}$$

- If $K = 2$ their is an equivalenve with Logistic Regression (binary classification)
  - we choose $\psi = \underline{w_1}$

$$P(y = 1|\underline{x}) = \frac{e^{\underline{w_1}\underline{x}}}{e^{\underline{w_1}\underline{x}} + e^{\underline{w_2}\underline{x}}} = \frac{e^{(\underline{w_1}-\psi)\underline{x}}}{e^{(\underline{w_1}-\psi)\underline{x}} + e^{(\underline{w_2}-\psi)\underline{x}}} = \frac{1}{1 + e^{(\underline{w_2}-\underline{w_1})\underline{x}}}$$

$$P(y = 2|\underline{x}) = \frac{e^{\underline{w_2}\underline{x}}}{e^{\underline{w_1}\underline{x}} + e^{\underline{w_2}\underline{x}}} = \frac{e^{(\underline{w_2}-\psi)\underline{x}}}{e^{(\underline{w_1}-\psi)\underline{x}} + e^{(\underline{w_2}-\psi)\underline{x}}} = \frac{e^{(\underline{w_2}-\underline{w1})\underline{x}}}{1 + e^{(\underline{w_2}-\underline{w1})\underline{x}}} = 1 - P(y = 1|\underline{x})$$

Why non-linear activation functions ?

- Consider the following network

$$\underline{z}^{[1]} = \underline{x}\,\underline{\underline{W}}^{[1]} + \underline{b}^{[1]}$$
$$\underline{a}^{[1]} = g^{[1]}(\underline{z}^{[1]})$$
$$\underline{z}^{[2]} = \underline{a}^{[1]}\underline{\underline{W}}^{[2]} + \underline{b}^{[2]}$$
$$\underline{a}^{[2]} = g^{[2]}(\underline{z}^{[2]})$$

- If $g^{[1]}$ and $g^{[2]}$ are **linear activation functions** (identity function),

$$\begin{aligned}\underline{a}^{[1]} &= \underline{z}^{[1]} = \underline{x}\,\underline{\underline{W}}^{[1]} + \underline{b}^{[1]}\\ \underline{a}^{[2]} &= \underline{z}^{[2]} = \underline{a}^{[1]}\underline{\underline{W}}^{[2]} + \underline{b}^{[2]}\\ &= \left(\underline{x}\,\underline{\underline{W}}^{[1]} + \underline{b}^{[1]}\right)\underline{\underline{W}}^{[2]} + \underline{b}^{[2]}\\ &= \underline{x}\underline{\underline{W}}^{[1]}\underline{\underline{W}}^{[2]} + \underline{b}^{[1]}\underline{\underline{W}}^{[2]} + \underline{b}^{[2]}\\ &= \underline{x}\,\underline{\underline{W}}' + b'\end{aligned}$$

  - then the network **the network reduces to a simple linear function**
- Linear activation ? only interesting for regression problem : $y \in \mathbb{R}$
  - linear function for last layer : $g^{[L]}$

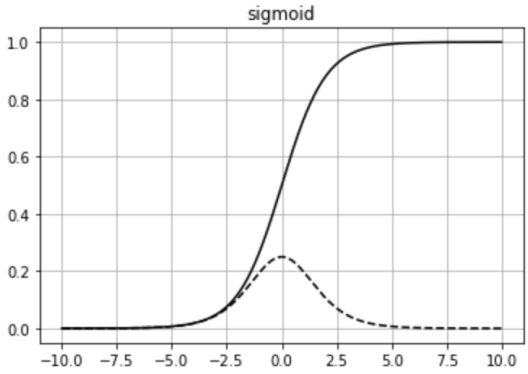
Sigmoid σ

- **Sigmoid function**

$$a = g(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$

  - **Derivative**

$$\begin{aligned} g'(z) &= - - e^{-z} \frac{1}{(1 + e^{-z})^2} \\ &= \frac{1 + e^{-z} - 1}{(1 + e^{-z})^2} \\ &= \frac{1}{1 + e^{-z}} \left(1 - \frac{1}{1 + e^{-z}}\right) \\ &= \sigma(z)(1 - \sigma(z)) \\ g'(z) &= a(1 - a) \end{aligned}$$



Hyperbolic tangent g

- **Hyperbolic tangent function**

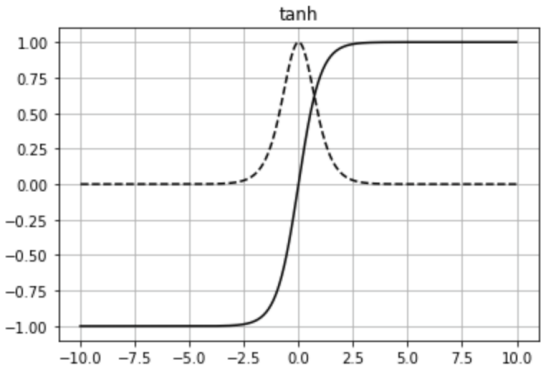
$$a = g(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

  - **Derivative**

$$g'(x) = 1 - (\tanh(z))^2$$

$$g'(z) = 1 - a^2$$

  - **Usage**
    - $\tanh(z)$ better than $\sigma(z)$ in middle hidden layers because its mean = zero ($a \in [-1, 1]$).
  - **Problem** with σ and tanh :
    - if z is very small (negative) or very large (positive)
      - $\Rightarrow$ slope becomes zero
      - $\Rightarrow$ slow down Gradient Descent



Vanishing gradient

- Reminder :

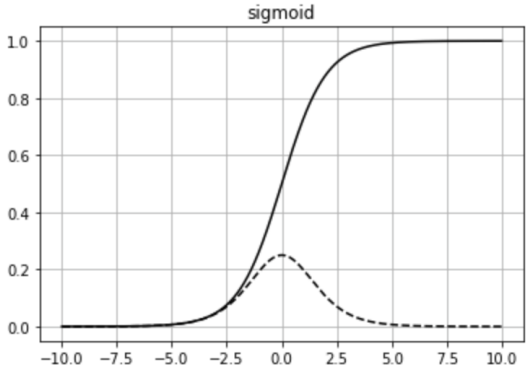
$$\frac{\partial \mathcal{L}}{\partial \underline{\underline{Z}}^{[l]}} = \frac{\partial \mathcal{L}}{\partial \underline{\underline{Z}}^{[l+1]}} \underbrace{\underline{\underline{W}}^{[l+1]\,T} \odot g^{[l]\prime}(Z^{[l]})}$$

$$\frac{\partial \mathcal{L}}{\partial \underline{b}^{[l]}} = \frac{1}{m} \sum_m \frac{\partial \mathcal{L}}{\partial \underline{\underline{Z}}^{[l]}}$$

- Hence for a deep network (supposing $g^{[l]}(z) = \sigma(z)$)

$$\frac{\partial \mathcal{L}}{\partial b^{[1]}} = \frac{\partial \mathcal{L}}{\partial a^{[4]}} \sigma'(z^{[4]}) \underbrace{W^{[4]} \sigma'(z^{[3]})} \underbrace{W^{[3]} \sigma'(z^{[2]})} \underbrace{W^{[2]} \sigma'(z^{[1]})}$$

- But $\max_z \sigma'(z) = \frac{1}{4}$!
  - Therefore, the deeper the network, the fastest the gradient diminishes/vanishes during backpropagation
    - Consequence ?
      - The network stop learning
- Solution ?
  - Use a ReLu activation



ReLU (Rectified Linear Unit)

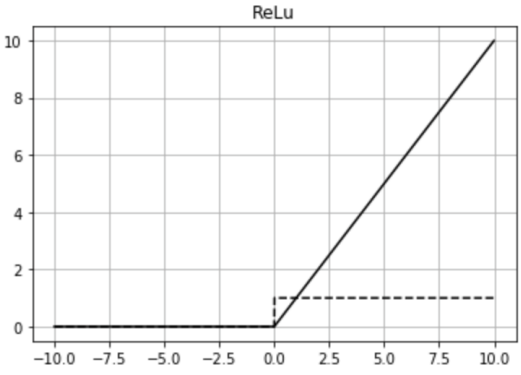
- **ReLU function**

  $a = g(z) = \max(0, z)$

  - **Derivative**

    $g'(x) = 1 \quad \text{if } z > 0$

    $= 0 \quad \text{if } z \leqslant 0$



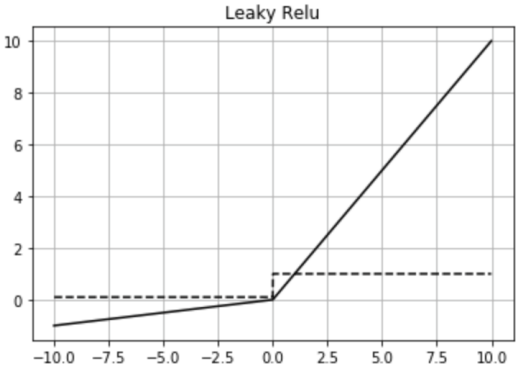
Variations of ReLU

- **Leaky ReLU function**

  $a = g(x) = \max(0.01z, z)$

  - allows to avoid the zero slope of the ReLU for $z < 0$ ("the neuron dies")
  - **Derivative**

$$\begin{aligned} g'(x) &= 1 && \text{if } z > 0 \\ &= 0.01 && \text{if } z \leqslant 0 \end{aligned}$$



- **PReLU function**

  $a = g(x) = \max(\alpha z, z)$

  - same as Leaky ReLU but α is a parameter to be learnt
  - **Derivative**

$$\begin{aligned} g'(x) &= 1 && \text{if } z > 0 \\ &= \alpha && \text{if } z \leqslant 0 \end{aligned}$$

- **Softplus function**

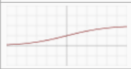
  $g(x) = \log(1 + e^x)$

  - continuous appriximation of ReLU
  - **Derivative**

$$g'(x) = \frac{1}{1 + e^{-x}}$$

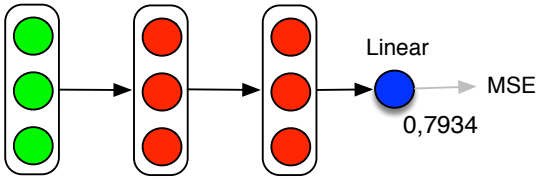
  - the derivative of the Softplus function is the Logistic function (smooth approximation of the derivative of the rectifier, the Heaviside step function.)

List of possible activation functions

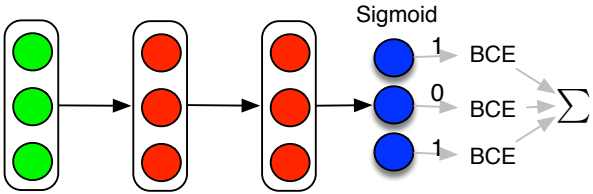
Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1+e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

Various types of problems

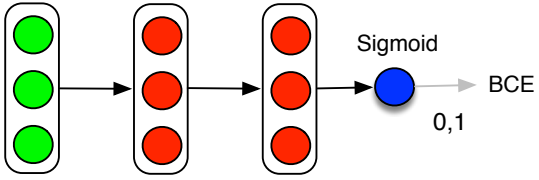
Regression



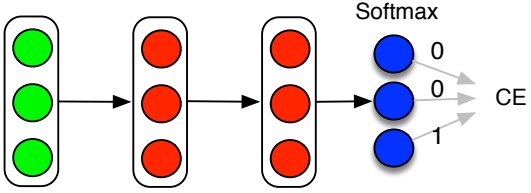
Multi-label



Binary Classification



Multi-class



Weight initialization

Weight initialization with zero ?

- **Initialize with zeros**

$$W^{[1]} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$W^{[2]} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

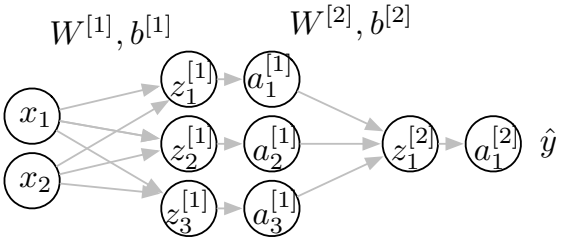
  - then for any example $a_1^{[1]} = a_2^{[1]} = a_3^{[1]}$
  - the three hidden units are computing exactly the same function
    - they are symetric

$$\underline{z}^{[1]} = \underline{a}^{[0]} \underline{\underline{W}}^{[1]}$$

$$\begin{pmatrix} z_1^{[1]} z_2^{[1]} z_3^{[1]} \end{pmatrix} = \begin{pmatrix} a_1^{[0]} a_2^{[0]} \end{pmatrix} \begin{pmatrix} w_{11}^{[1]} w_{12}^{[1]} w_{13}^{[1]} \\ w_{21}^{[1]} w_{22}^{[1]} w_{23}^{[1]} \end{pmatrix}$$

$$\underline{z}^{[2]} = \underline{a}^{[1]} \underline{\underline{W}}^{[2]}$$

$$z_1^{[2]} = \begin{pmatrix} a_1^{[1]} a_2^{[1]} a_3^{[1]} \end{pmatrix} \begin{pmatrix} w_{11}^{[2]} \\ w_{21}^{[2]} \\ w_{31}^{[2]} \end{pmatrix}$$



Weight initialization with zero ?

- Backpropagation

$$\frac{\partial \mathcal{L}}{\partial \underline{z}^{[2]}} = a^{[2]} - y$$

$$\frac{\partial \mathcal{L}}{\partial \underline{\underline{W}}^{[2]}} = a^{[1]^T} \frac{\partial \mathcal{L}}{\partial \underline{z}^{[2]}} = \begin{pmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \end{pmatrix} \frac{\partial \mathcal{L}}{\partial \underline{z}^{[2]}}_1$$

$$\frac{\partial \mathcal{L}}{\partial \underline{a}^{[1]}} = \frac{\partial \mathcal{L}}{\partial \underline{z}^{[2]}} \underline{\underline{W}}^{[2]^T} = \frac{\partial \mathcal{L}}{\partial \underline{z}^{[2]}} \left(w_{11}^{[2]} w_{21}^{[2]} w_{31}^{[2]} \right)$$

$$\frac{\partial \mathcal{L}}{\partial \underline{z}^{[1]}} = \frac{\partial \mathcal{L}}{\partial \underline{a}^{[1]}} \odot g^{[1]'}(z^{[1]}) = \frac{\partial \mathcal{L}}{\partial \underline{z}^{[2]}} \left(w_{11}^{[2]} w_{21}^{[2]} w_{31}^{[2]} \right) \odot g^{[1]'} \left(z_1^{[1]} z_2^{[1]} z_3^{[1]} \right)$$

$$\frac{\partial \mathcal{L}}{\partial \underline{\underline{W}}^{[1]}} = a^{[0]^T} \frac{\partial \mathcal{L}}{\partial \underline{z}^{[1]}} = \begin{pmatrix} a_1^{[0]} \\ a_2^{[0]} \end{pmatrix} \left(\frac{\partial \mathcal{L}}{\partial \underline{z}^{[1]}}_1 \frac{\partial \mathcal{L}}{\partial \underline{z}^{[1]}}_2 \frac{\partial \mathcal{L}}{\partial \underline{z}^{[1]}}_3 \right) = \begin{pmatrix} \frac{\partial \mathcal{L}}{\partial \underline{z}^{[1]}}_1 a_1^{[0]} & \frac{\partial \mathcal{L}}{\partial \underline{z}^{[1]}}_2 a_1^{[0]} & \frac{\partial \mathcal{L}}{\partial \underline{z}^{[1]}}_3 a_1^{[0]} \\ \frac{\partial \mathcal{L}}{\partial \underline{z}^{[1]}}_1 a_2^{[0]} & \frac{\partial \mathcal{L}}{\partial \underline{z}^{[1]}}_2 a_2^{[0]} & \frac{\partial \mathcal{L}}{\partial \underline{z}^{[1]}}_3 a_2^{[0]} \end{pmatrix} = \begin{pmatrix} u & u & u \\ v & v & v \end{pmatrix}$$

- When we compute backpropagation : $\frac{\partial \mathcal{L}}{\partial \underline{z}^{[1]}}_1 = \frac{\partial \mathcal{L}}{\partial \underline{z}^{[1]}}_2 = \frac{\partial \mathcal{L}}{\partial \underline{z}^{[1]}}_3$
- Therefore $\frac{\partial \mathcal{L}}{\partial \underline{\underline{W}}^{[1]}}$ is in the form

$$\frac{\partial \mathcal{L}}{\partial \underline{\underline{W}}^{[1]}} = \begin{pmatrix} u & u & u \\ v & v & v \end{pmatrix} \rightarrow W^{[1]} = W^{[1]} - \alpha \frac{\partial \mathcal{L}}{\partial \underline{\underline{W}}^{[1]}}$$

  - $\Rightarrow$ the update will keep the symetricity : $z_1^{[1]} = z_2^{[1]} = z_3^{[1]} = a_1^{[0]} u + a_2^{[0]} v$
- so this is not usefull since we want the different units to compute difference functions
  - $\Rightarrow$ initialize the parameters randomly

Weight initialization with random values

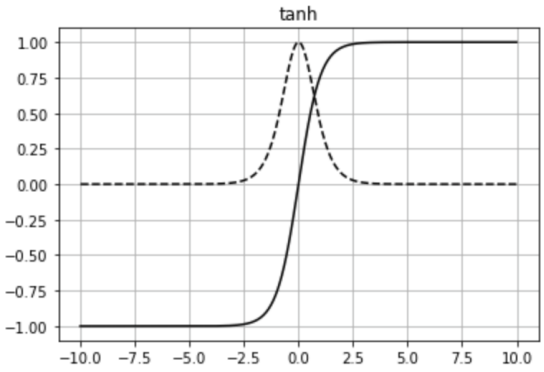
- **Random initialization**
  - $W^{[1]} =$ `np.random.randn((2,3))*0.01`
  - $b^{[1]} =$ `0`
  - $W^{[2]} =$ `np.random.randn((3,1))*0.01`
  - $b^{[1]} =$ `0`
- Remark : b doesn't have the symetry problem

- **Why 0.01 ?**
  - If W is big $\Rightarrow$ Z is also big

$$Z^{[1]} = X\ W^{[1]} + b^{[1]} \tag{3}$$

$$A^{[1]} = g^{[1]}(Z^{[1]}) \tag{4}$$

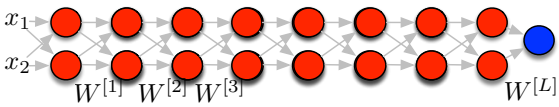
    - $\Rightarrow$ we are in the flat part of the sigmoid/tanh
      - $\Rightarrow$ slope is small
      - $\Rightarrow$ gradient descent slow
      - $\Rightarrow$ learning slow
  - Better to initialize to a very small value (valid for sigmoid and tanh)



Vanishing/ exploding gradients

- For a very deep neural network
  - suppose $g^{[l]}(z) = z$ (linear activation) and $b^{[l]} = 0$
  - then

$$y = \underbrace{\underbrace{X \underline{\underline{W}}^{[1]}}_{a^{[1]}=g(z^{[1]})=z^{[1]}} \underline{\underline{W}}^{[2]}}_{a^{[2]}=g(z^{[2]})=z^{[2]}} \ldots \underline{\underline{W}}^{[L]}$$



- **Exploding gradient**
  - suppose

$$\underline{\underline{W}}^{[l]} = \begin{pmatrix} 1.5 & 0 \\ 0 & 1.5 \end{pmatrix}$$

  - then

$$\hat{y} = X \begin{pmatrix} 1.5 & 0 \\ 0 & 1.5 \end{pmatrix}^{L-1} \underline{\underline{W}}^{[L]}$$

  - if L is large $\Rightarrow (1.5)^{L-1}$ is very large
    - $\Rightarrow$ the value of $\hat{y}$ will explode
- Similar arguments can be used for the gradient

- **Vanishing gradient**
  - suppose

$$\underline{\underline{W}}^{[l]} = \begin{pmatrix} 0.5 & 0 \\ 0 & 0.5 \end{pmatrix}$$

  - if L is large $\Rightarrow (0.5)^{L-1}$ is very small
    - $\Rightarrow$ the value of $\hat{y}$ will vanish

Weight initialization for Deep Neural Networks

- Suppose a single neuron network
  - $z = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b$
- In order to avoid vanishing/exploiding gradient) $\Rightarrow$
  - The larger n is $\Rightarrow$ the smallest w_i should be
- Solution ?
  - set $\mathrm{Var}(w_i) = \frac{1}{n}$
- In practice
  - $\underline{\underline{W}}^{[l]} =$ `np.random.randn(shape) * np.sqrt` $\left(\frac{1}{n^{[l-1]}}\right)$
- Other possibilities
  - For a ReLu
    - $\mathrm{Var}(w_i) = \frac{2}{n}$ works a bit better
  - For a tanh
    - $\mathrm{Var}(w_i) = \frac{1}{n^{[l-1]}}$: Xavier initialization
    - $\mathrm{Var}(w_i) = \frac{2}{n^{[l-1]} n^{[l]}}$: Bengio initialization

Regularization

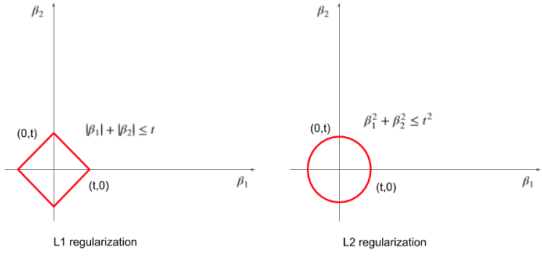
L1 and L2 regularization

- **Goal ?**
  - avoid over-overfitting (high variance)
- **How ?**
  - reduce model complexity
- In logistic regression

$$J(\underline{w}, b) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda_1}{2m}||w||_1 + \frac{\lambda_2}{2m}||w||_2^2 +$$

$$\text{with } ||w||_1 = \sum_{j=1}^{n_x}|w_j| \text{ and } ||w||_2^2 = \sum_{j=1}^{n_x} w_j^2 = \underline{w}^T\underline{w}$$

- **L1 regularization** (Lasso) :
  - will end up with sparse $\underline{w}$ (many zero)
- **L2 regularization** (Ridge) :
  - will end up with small values of $\underline{w}$
- L1+L2 : ELastic Search
- λ is the regularization parameter (hyperparameter)



L1 and L2 regularization

- In neural network

$$J(\ldots) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L}||\underline{\underline{W}}^{[l]}||^2$$

where $||\underline{\underline{W}}||_2^2 = ||\underline{\underline{W}}||_F$ is the "Frobenius norm"

$$||\underline{\underline{W}}||^2 = \sum_{i=1}^{n^{[l-1]}}\sum_{j=1}^{n^{[l]}}(\underline{\underline{W}}_{i,j}^{[l]})^2$$

- In gradient descent ?

$$d'\underline{\underline{W}}^{[l]} = \frac{\partial\mathcal{L}}{\partial\underline{\underline{W}}^{[l]}} + \frac{\lambda}{m}\underline{\underline{W}}^{[l]}$$

Therefore

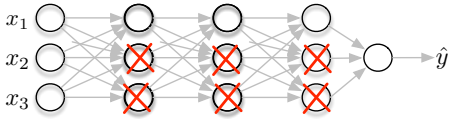
$$\begin{aligned}\underline{\underline{W}}^{[l]} &\leftarrow \underline{\underline{W}}^{[l]} - \alpha d'\underline{\underline{W}}^{[l]}\\ &\leftarrow \underline{\underline{W}}^{[l]} - \alpha(\frac{\partial\mathcal{L}}{\partial\underline{\underline{W}}^{[l]}} + \frac{\lambda}{m}\underline{\underline{W}}^{[l]})\\ &\leftarrow \underline{\underline{W}}^{[l]} - \frac{\alpha\lambda}{m}\underline{\underline{W}}^{[l]} - \alpha\frac{\partial\mathcal{L}}{\partial\underline{\underline{W}}^{[l]}}\\ &\leftarrow \underbrace{\underline{\underline{W}}^{[l]}\underbrace{(1-\frac{\alpha\lambda}{m})}_{\leqslant 1}}_{\text{weight decay}} - \alpha\frac{\partial\mathcal{L}}{\partial\underline{\underline{W}}^{[l]}}\end{aligned}$$

Why regularization reduces overfitting ?

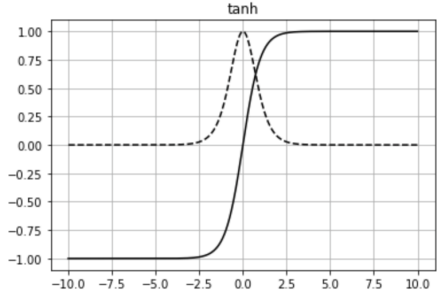
- **Intuition 1**

$$J = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L}||\underline{\underline{W}}^{[l]}||^2$$

  - If we set λ very very big then $\underline{\underline{W}}^{[l]} \simeq 0$
    - $\Rightarrow$ many hidden units are not active
    - $\Rightarrow$ the network becomes much simpler $\Rightarrow$ avoid over-fitting

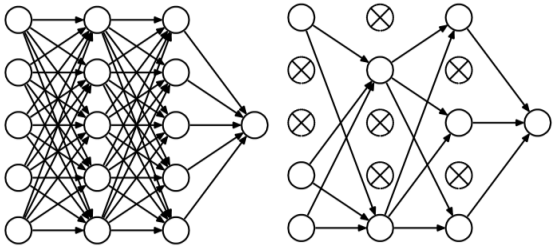


- **Intuition 2**
  - Suppose we are using a tanh
    - If z is small, the tanh is linear
  - If λ is large,
    - then $\underline{\underline{W}}^{[l]}$ is small,
    - then $z^{[l]}$ is small
    - then every layer is almost linear,
    - $\Rightarrow$ the whole network is linear



DropOut regularization

- **During training**
  - for each training example **randomly turn-off** the neurons of hidden units (with $p = 0.5$)
    - this also removes the connections
  - for different training examples, turn-off different units
  - possible to vary the probability across layers
    - for large matrix $\underline{\underline{W}} \Rightarrow p$ is higher
    - for small matrix $\underline{\underline{W}} \Rightarrow p$ is lower



- **During testing**
  - no drop out
- **Dropout effects :**
  - prevents co-adaptation between units
  - can be seen as averaging different models that share parameters
  - acts as a powerful regularization scheme
  - since the network is smaller, it is easier to train (as regularization)
  - The network cannot rely on any feature, it has to spread out weights
    - Effect : shrinking the squared norm of the weights (similar to L2 regularization)
    - Can be shown to be an adaptive form of L2-regularization

Data augmentation


Car_A_0_345


Car_A_0_1193


Car_A_0_1589


Car_A_0_2933


Car_A_0_3228


Car_A_0_3274


Car_A_0_3614


Car_A_0_3686


Car_A_0_3894


Car_A_0_5212


Car_A_0_5528


Car_A_0_5574

Normalization

Normalizing the inputs

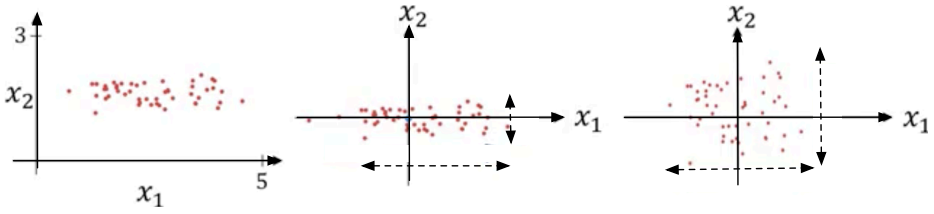
- **"Standardizing" :**
  - subtracting a measure of location and dividing by a measure of scale
  - subtract the mean and divide by the standard deviation

$$\mu_d = \frac{1}{m}\sum_{i=1}^{m} x_d^{(i)} \rightarrow x_d = x_d - \mu_d$$

$$\sigma_d^2 = \frac{1}{m}\sum_{i=1}^{m}(x_d^{(i)} - \mu_d)^2 \rightarrow x_d = \frac{x_d}{\sigma_d}$$

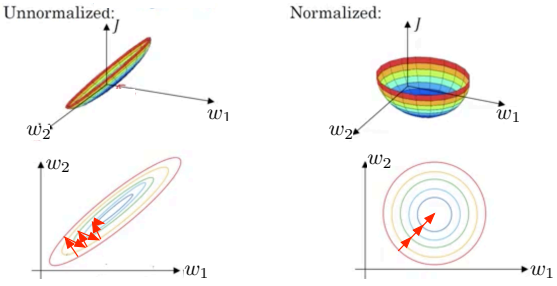
- We use also μ_{train} and σ^2_{train} to standardize the test set

- **"Normalizing" :**
  - rescaling by the minimum and range of the vector
  - make all the elements lie between 0 and 1.



Normalizing the inputs

- Suppose : $x_1 \in [1 \dots 1000]$ and $x_2 \in [0 \dots 1]$
  - Then w_1 and w_2 will take very different value
- if no normalization
  - many oscillations
  - need to use a small learning rate
- **Why use input normalization ?**
  - get similar values for w_1 and w_2
    - gradient descent can go straight to the minimum
    - can use large learning rate
  - avoid each activation to be large (see sigmoid activation)
  - numerical instability



Batch Normalization (BN)

Sergey Ioffe and Christian Szegedy, "Batch Normalization : Accelerating Deep Network Training by Reducing Internal Covariate Shift" [ICML2015]

- **Objective ?**
  - Apply the same normalization for the input of each layer $[l]$
    - allows to learn faster
  - Try to **reduce the "covariate shift"**
    - the inputs of a given layer $[l]$ is the outputs of the previous layer $[l-1]$
      - these outputs $[l-1]$ depends on the parameters of the previous layer which change over training !
    - **normalize the output of the previous layer** $a^{[l-1]}$
    - in practice normalize the pre-activation $z^{[l-1]}$
  - Don't want all units to always have mean 0 and standard-deviation 1
    - Learn an appropriate bias β and scale γ to apply to $z^{[l-1]}$ before the non-linear function $g^{[l]}$

Batch Normalization (BN)

- **Batch Normalization**
  - Given some intermediate values in the network : $z^{[l](1)}, z^{[l](2)}, \ldots, z^{[l](m)}$
  - Compute

- **Normalization**

$$\mu^{[l]} = \frac{1}{m}\sum_i z^{[l](i)}$$

$$\sigma^{2[l]} = \frac{1}{m}\sum_i (z^{[l](i)} - \mu^{[l]})^2$$

$$z_{norm}^{[l](i)} = \frac{z^{[l](i)} - \mu^{[l]}}{\sqrt{\sigma^{2[l]} + \epsilon}}$$

- **New parameters to be trained : γ and β**
  - β allows to set the mean of $\tilde{z}^{[l]}$
  - γ allows to set the variance of $\tilde{z}^{[l]}$

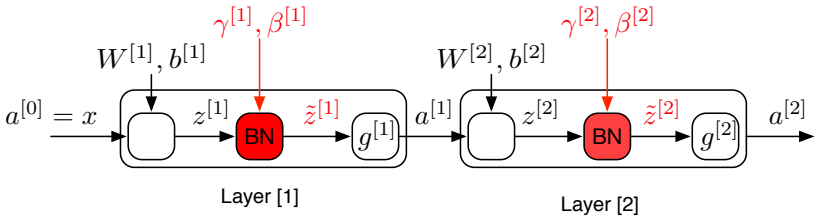
$$\tilde{z}^{[l](i)} = \gamma \cdot z_{norm}^{[l](i)} + \beta$$

- Non-linearity

$$a^{[l](i)} = g^{[l]}(\tilde{z}^{[l](i)})$$

- **Note** : if $\gamma = \sqrt{\sigma^{2[l]} + \epsilon}$ and $\beta = \mu$ then $\tilde{z}^{[l](i)} = z^{[l](i)}$

Batch Normalization (BN)



- **New parameters :**
  - $\beta^{[1]}, \gamma^{[1]}, \beta^{[2]}, \gamma^{[2]}, \ldots$
- How to estimate $\beta^{[l]}$, $\gamma^{[l]}$?
  - $\beta^{[l]}$, $\gamma^{[l]}$ are estimated using **gradient-descent**
    - **Gradient** : $\frac{\partial \mathcal{L}}{\partial \beta^{[l]}}$, $\frac{\partial \mathcal{L}}{\partial \gamma^{[l]}}$
    - **Update** : $\beta^{[l]} = \beta^{[l]} - \alpha \frac{\partial \mathcal{L}}{\partial \beta^{[l]}}$, $\gamma^{[l]} = \gamma^{[l]} - \alpha \frac{\partial \mathcal{L}}{\partial \gamma^{[l]}}$
- **With mini-batches** $\{1\}$, $\{2\}$, ... :
  - $a^{[0]\{1\}} \xrightarrow{W^{[1]}, b^{[1]}} z^{[1]\{1\}} \xrightarrow{BN:\gamma^{[1]}, \beta^{[1]}} \tilde{z}^{[1]\{1\}} \rightarrow a^{[1]\{1\}} = g^{[1]}(z^{[1]\{1\}})$
    - μ and σ are computed over the minibatch $\{1\}$
  - $a^{[0]\{2\}} \xrightarrow{W^{[1]}, b^{[1]}} z^{[1]\{2\}} \xrightarrow{BN:\gamma^{[1]}, \beta^{[1]}} \tilde{z}^{[1]\{2\}} \rightarrow a^{[1]\{2\}} = g^{[1]}(z^{[1]\{2\}})$
    - μ and σ are computed over the minibatch $\{2\}$