



UNIVERSITÉ  
CÔTE D'AZUR

# Lecture 3: A primer on fundamentals of deep learning

Advanced deep learning

Rémy Sun  
[remy.sun@inria.fr](mailto:remy.sun@inria.fr)

*inria*



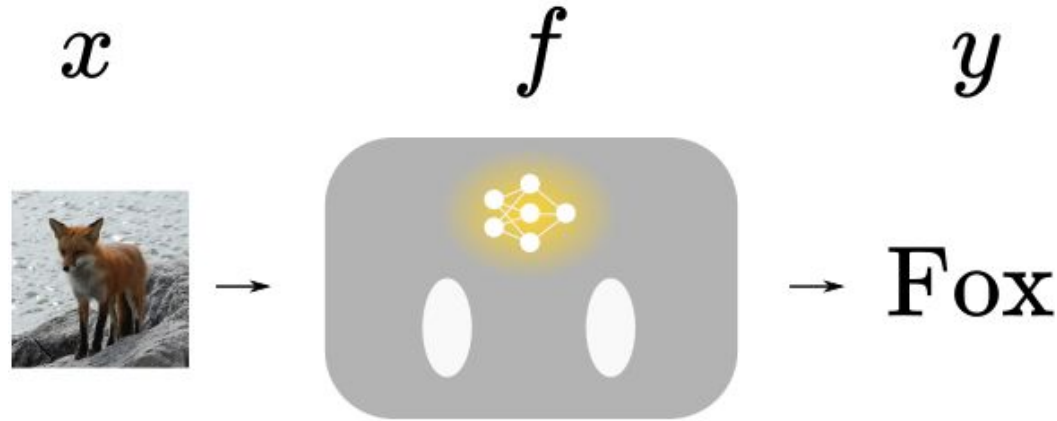
# About this course

- Goal: In-depth understanding of important Deep Learning staples
  - Reinforce what you have already seen
  - Introduce state of the art models
- This is a hands-on course in pytorch
  - Minimal math
    - Enough to understand
  - Quite a bit of coding
  - Get comfortable with the standard pipeline

- L1-2: Overview of Deep Learning (F. Precioso)
- L3-4: Fundamentals of Deep Learning (R. Sun)
- L5-6: Transformers (R. Sun)
- L7: Large models (LLMs, VLMs, Generators) (R. Sun)
- L8-9: Intro to generative models (P-A. Mattei)
- Tricks of the trade (R. Sun)

- Quick recap of deep learning fundamentals
  - Problem Statement and Empirical Risk Minimization
  - Neural Networks as classes of functions
  - Gradient Descent
  - Backpropagation
- Practical in pytorch: Training a MLP
  - Manual implementation
  - Progressive pytorch automation

# 1. Problem statement and risk



- Find (robot)  $f$  that classifies images well
  - Often based on neural networks

$$\forall (x, y) \in \mathcal{D}, f(x) = y$$

- Definitions
  - $X$  set of inputs
  - $Y$  set of labels
  - $\Omega = X \times Y$
  - $\mathcal{D}$  Distribution over  $\Omega$  with probability measure  $p$
- Find function  $f: X \rightarrow Y$  such that

$$\forall (x, y) \in \mathcal{D}, f(x) = y$$



- Finding exact correspondence functions is not always the thing to do
  - No exact matching
  - Other definitions of good solutions
  - Need to use restricted function space
    - Parametric function space

$$\mathcal{F} = \{f_{\theta} | \theta \in \mathbb{R}^d\}$$

- Introduce an assessment of how “good”  $f$  is with a loss  $l$  so that we try to have the lowest quantity  $l(f(x), y)$

- Definitions
  - $X$  set of inputs
  - $Y$  set of labels
  - $\Omega = X \times Y$
  - $\mathcal{D}$  Distribution over  $\Omega$  with probability measure  $p$
  - $l$  loss function assessing fit of  $f(x)$  to  $y$
  - Find  $f$  in function space  $\mathcal{F} = \{f_\theta | \theta \in \mathbb{R}^d\}$
- Minimize the **Risk** over the distribution

$$\min_{\theta} \mathbb{E}_{x, y \sim \mathcal{D}} [l(f_{\theta}(x), y)]$$

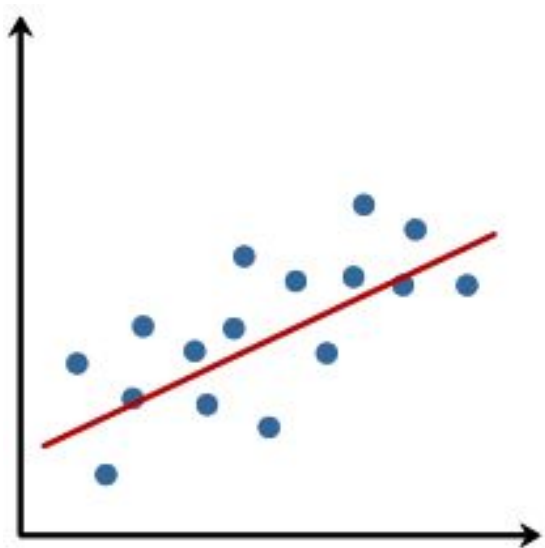


- Problem: we do not know  $\mathcal{D}$  !
  - Solved problem otherwise...
  - Evaluating the risk requires this distribution
- Solution: Use a dataset  $D$  of  $(x,y)$  sampled from  $\mathcal{D}$ 
  - ✓ ○ **Empirical Risk Minimization**
  - ✓ ○ If the  $(x,y)$  are i.i.d drawn from  $\mathcal{D}$  can be expressed as a mean over the dataset

$$\min_{\theta} \hat{\mathcal{R}}_{\theta} = \frac{1}{N} \sum_{i=0, \dots, N-1} l(f_{\theta}(x_i), y_i)$$

- Core problem: Find function matching inputs to outputs for any  $(x,y)$  of the target distribution
- Optimize over family of parametric functions
  - Assess functions with loss criterion
- Minimize the Risk function
  - Empirical Risk Minimization in practice

## 2. Example: linear regression

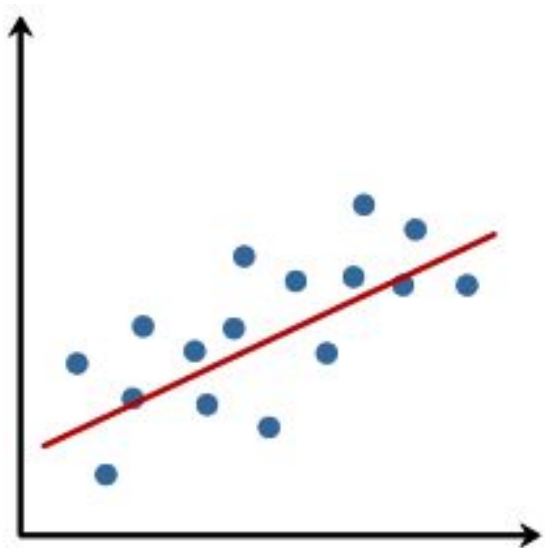


- Linearly correlated data
  - Input  $x$  (e.g. Voltage)
  - Output  $y$  (e.g. Intensity)
- Simple family of linear functions
  - Find linear coefficient

$$f_{\theta}(x) = \theta x$$

$$f_{\theta}(x) = \theta x$$

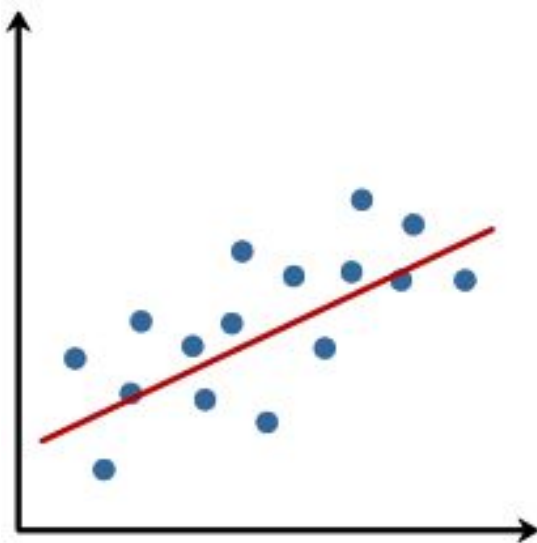
- Minimize the risk



$$f_{\theta}(x) = \theta x$$

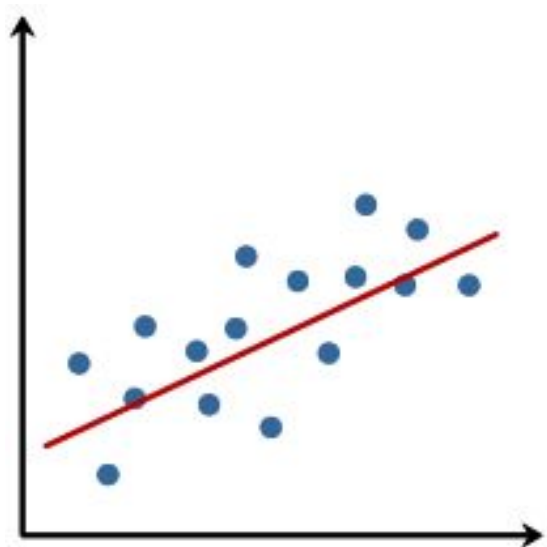
- Minimize the risk

$$\min_{\theta} \hat{\mathcal{R}}_{\theta} = \frac{1}{N} \sum_{i=0, \dots, N-1} (y_i - \theta x_i)$$





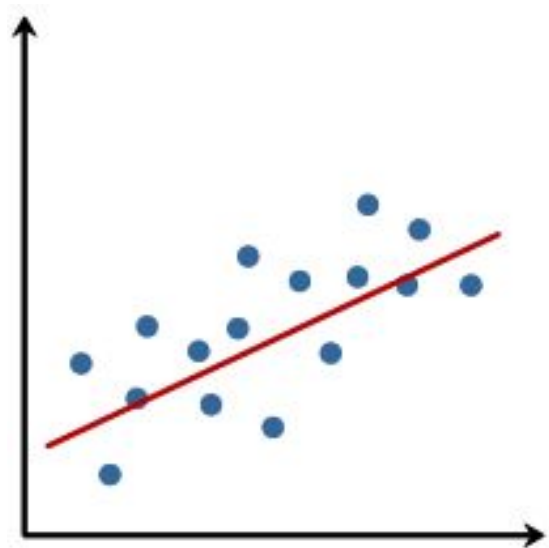
$$f_{\theta}(x) = \theta x$$



- Minimize the risk

$$\min_{\theta} \hat{\mathcal{R}}_{\theta} = \frac{1}{N} \sum_{i=0, \dots, N-1} (y_i - \theta x_i)$$

- How?

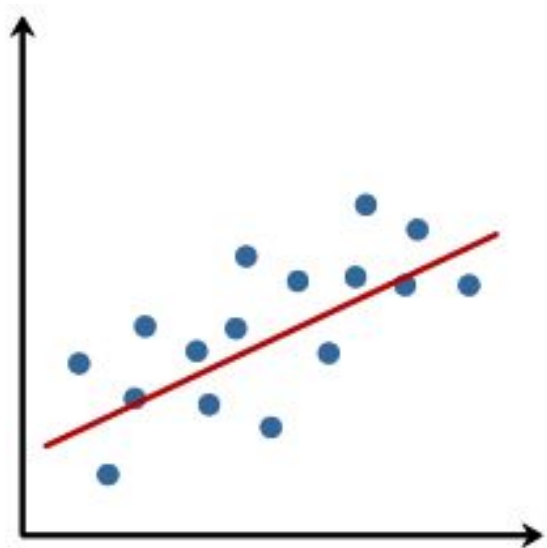


$$f_{\theta}(x) = \theta x$$

- Minimize the risk

$$\min_{\theta} \hat{\mathcal{R}}_{\theta} = \frac{1}{N} \sum_{i=0, \dots, N-1} (y_i - \theta x_i)^2$$

- How?
  - Convex function!

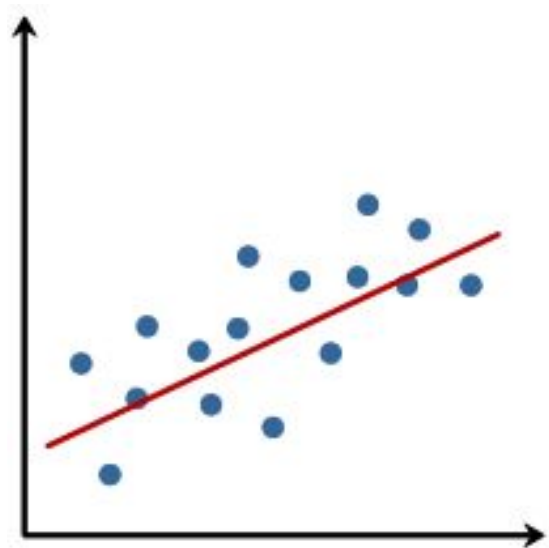


$$f_{\theta}(x) = \theta x$$

- Minimize the risk

$$\min_{\theta} \hat{\mathcal{R}}_{\theta} = \frac{1}{N} \sum_{i=0, \dots, N-1} (y_i - \theta x_i)^2$$

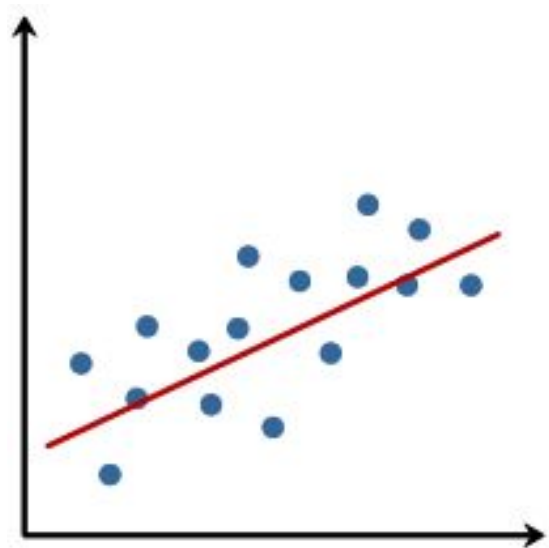
- How?
  - Convex function!
  - Zero out the gradient!



$$\min_{\theta} \mathcal{R}_{\theta} = \frac{1}{N} \sum_{i=0, \dots, N-1} (y_i - \theta x_i)^2$$

- Deriving gives condition:

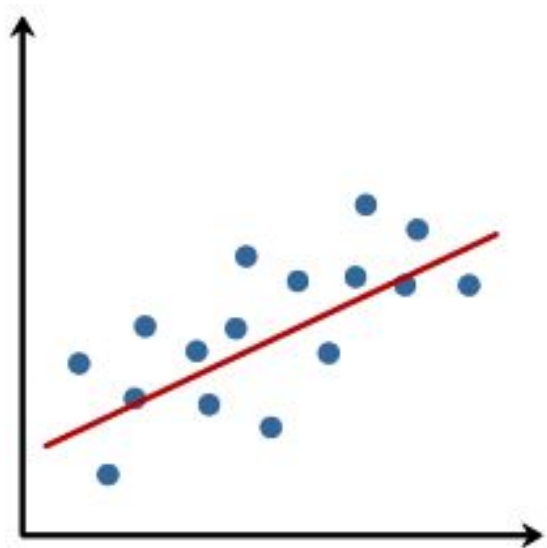




$$\min_{\theta} \mathcal{R}_{\theta} = \frac{1}{N} \sum_{i=0, \dots, N-1} (y_i - \theta x_i)^2$$

- Deriving gives condition:

$$-\frac{2}{N} \sum_{i=0, \dots, N-1} (y_i - \theta x_i) x_i = 0$$

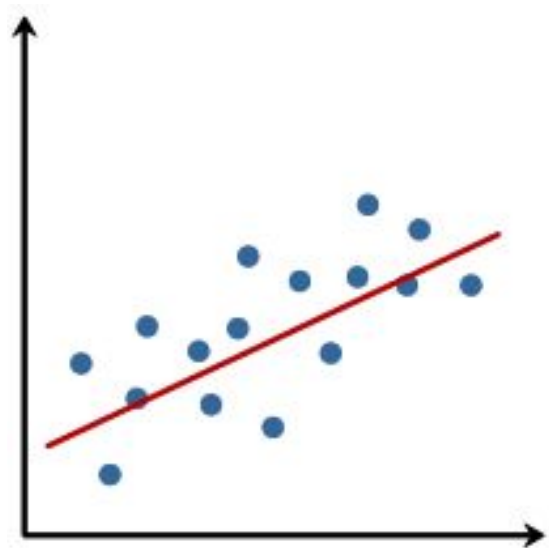


$$\min_{\theta} \mathcal{R}_{\theta} = \frac{1}{N} \sum_{i=0, \dots, N-1} (y_i - \theta x_i)^2$$

- Deriving gives condition:

$$-\frac{2}{N} \sum_{i=0, \dots, N-1} (y_i - \theta x_i) x_i = 0$$

- Solve for  $\theta$




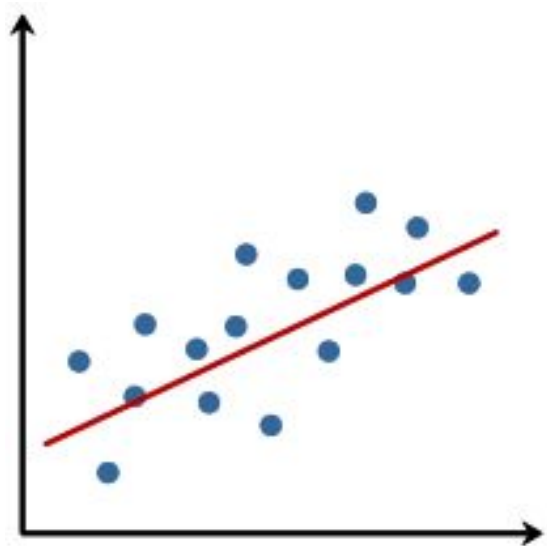
$$\min_{\theta} \mathcal{R}_{\theta} = \frac{1}{N} \sum_{i=0, \dots, N-1} (y_i - \theta x_i)^2$$

- Deriving gives condition:

$$-\frac{2}{N} \sum_{i=0, \dots, N-1} (y_i - \theta x_i) x_i = 0$$

- Solve for  $\theta$


$$\theta = \frac{\sum_{i=0, \dots, N-1} y_i x_i}{\sum_{i=0, \dots, N-1} x_i^2}$$



$$f_{\theta}(x) = \theta x$$

$$\theta = \frac{\sum_{i=0, \dots, N-1} y_i x_i}{\sum_{i=0, \dots, N-1} x_i^2}$$

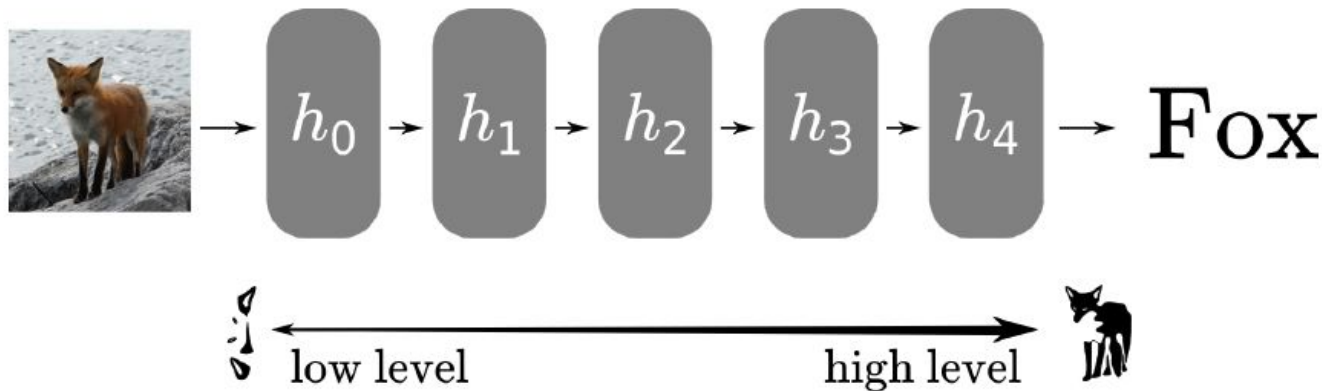
- If perfectly linear correlation

$$\theta = a \frac{\sum_{i=0, \dots, N-1} x_i^2}{\sum_{i=0, \dots, N-1} x_i^2} = a$$



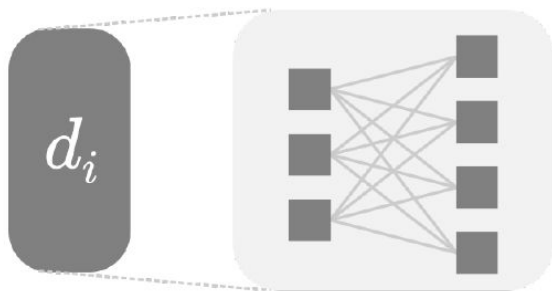
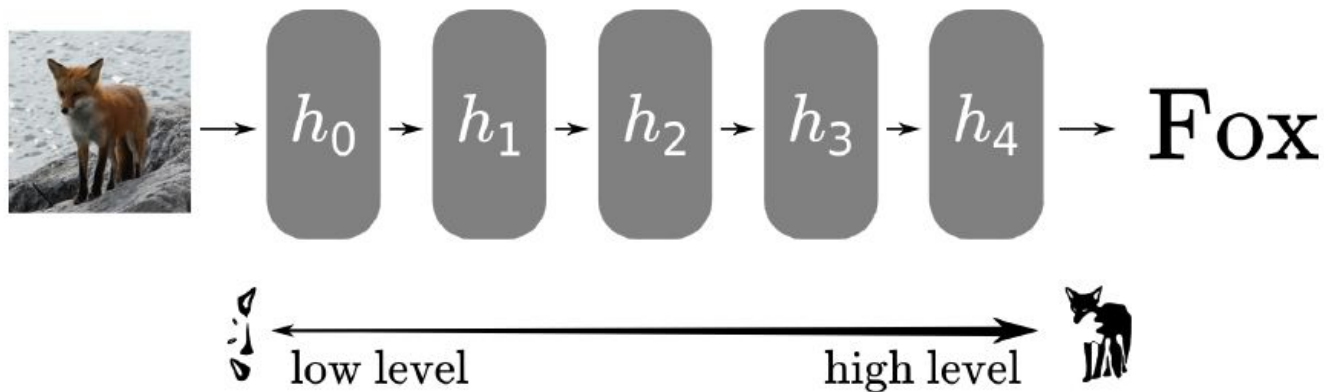
- Core problem: Find the right function in a family
  - Boils down to finding the right parameters
  - Depends on the data available
- Minimizing the risk is finding the best fit solution
  - Shown on univariate linear regression
  - Generalizes to multiple dimensions
  - ***Pointless if the data does not fit!***

### 3. Neural network functions

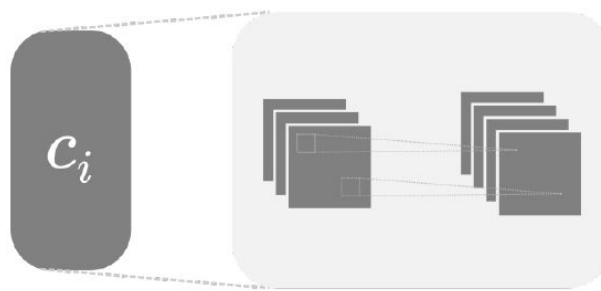


- Neural networks are sequences of simple functions

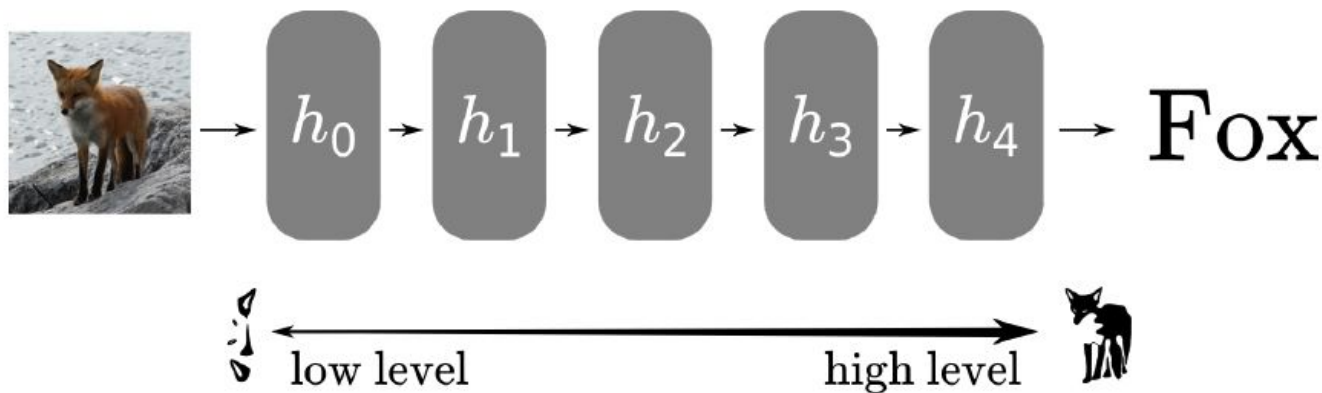
$$f_{\theta} = h_{\theta}^0 \circ h_{\theta}^1 \circ \dots \circ h_{\theta}^{L-1}$$



(a) Dense layer



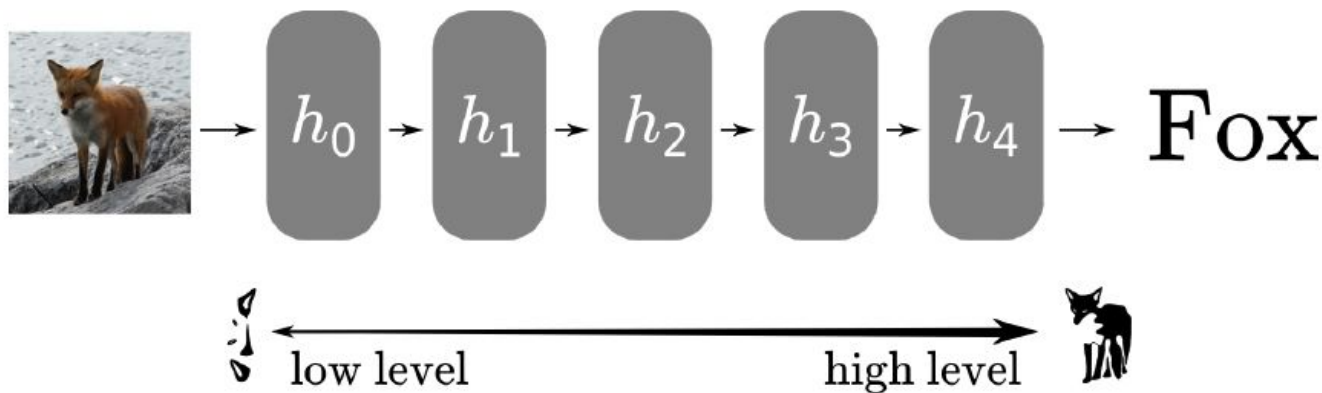
(b) Convolutional layer



(a) Dense layer

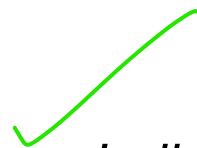
$$d_{\theta}(x) = \sigma(W_{\theta}x^T + b_{\theta})$$

$$\sigma(x) = ReLU(x) = \max(0, x)$$



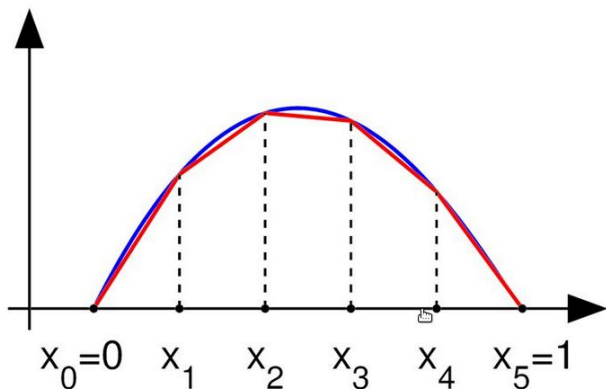
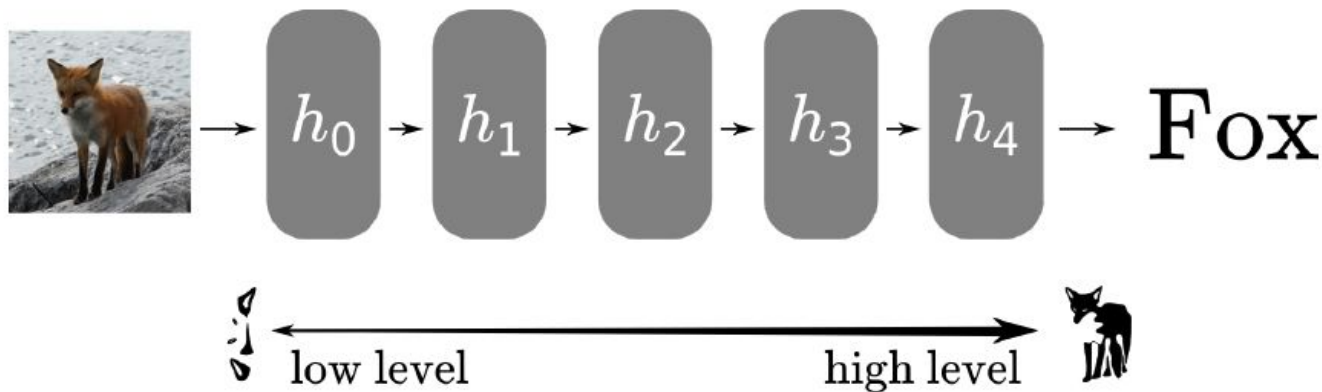
$$d_{\theta}(x) = \sigma(W_{\theta}x^T + b_{\theta})$$

$$\sigma(x) = \text{ReLU}(x) = \max(0, x)$$



**Piecewise linear!**

*Individual layers are piecewise linear, composition preserves piecewise linearity*



- Highly expressive
  - Can fit many types of distributions

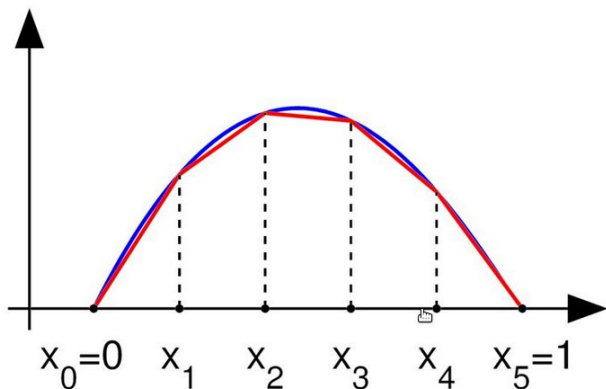
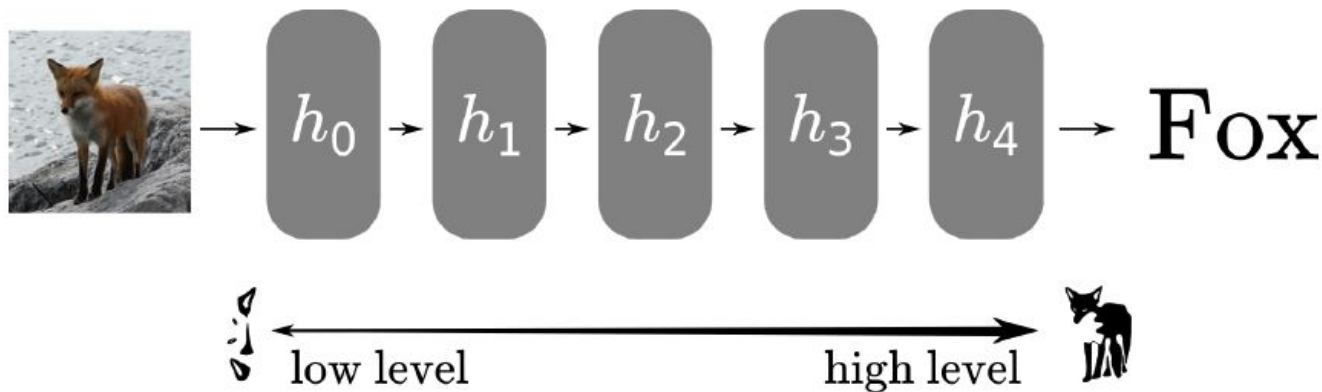
- Upper bound on number of linear pieces wrt number of layers and units per layer
  - Exercise: Proof by recurrence
- Similar properties with other deep networks
  - Piecewise polynomial with other  $\sigma$
  - Similar reasoning on convolutional layers
- ✓ ● Universal approximation theorem [Cybenko '89]
  - Proof by contradiction



- Neural networks composed of simple functions
  - Typical linear layer operations
  - Non-linear activation functions
- High expressive power
  - Universal approximation with enough neurons
  - ReLU Feedforward networks are piecewise linear

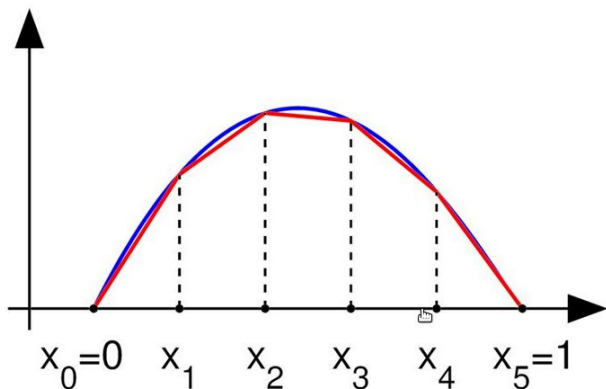
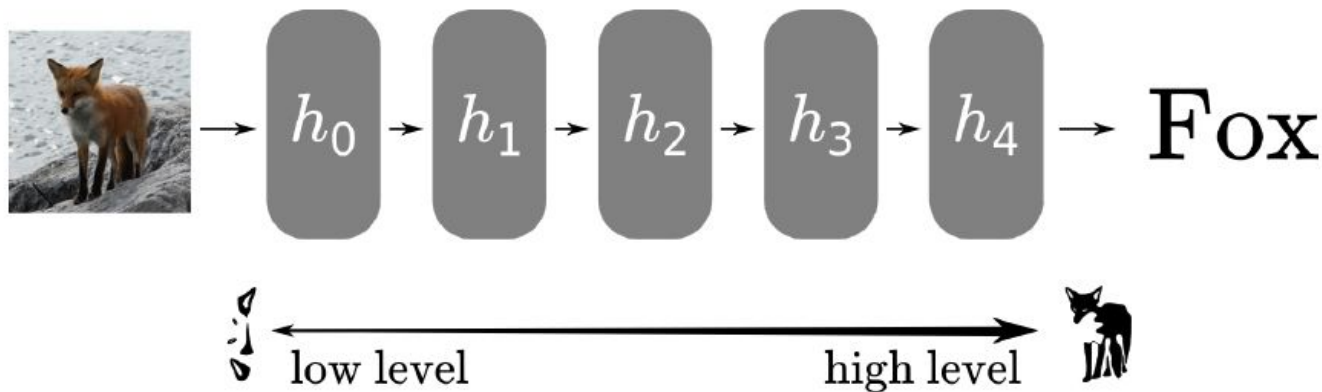
## 4. Gradient descent

# Let's find the best network then!



$$\min_{\theta} \hat{\mathcal{R}}_{\theta} = \frac{1}{N} \sum_{i=0, \dots, N-1} l(f_{\theta}(x_i), y_i)$$

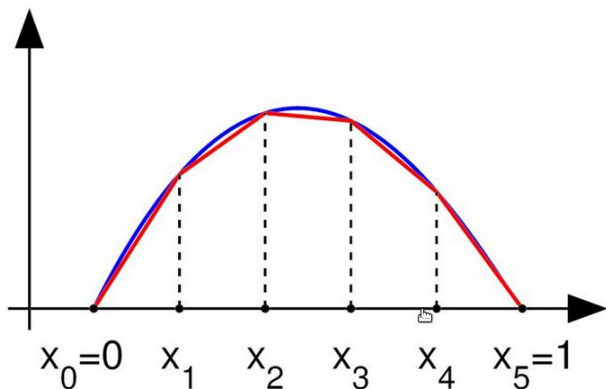
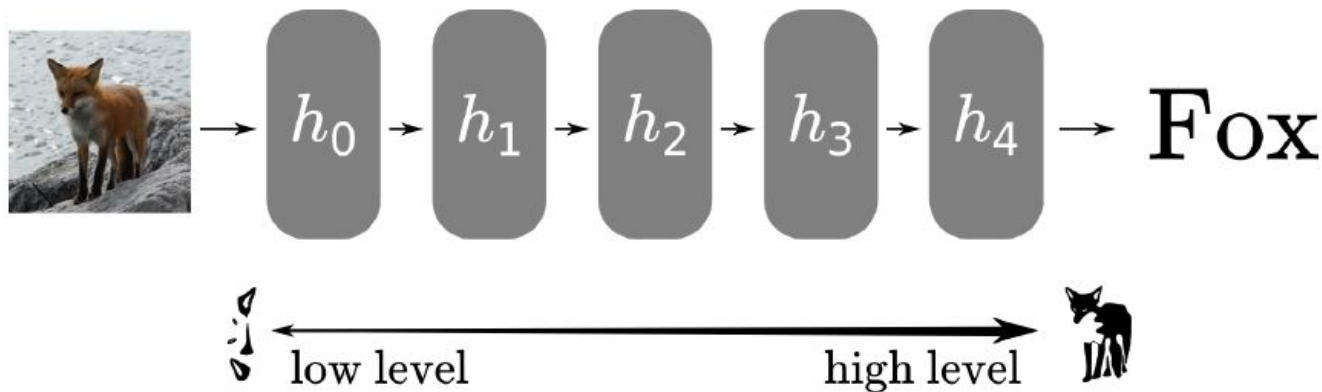
# Let's find the best network then!



$$\min_{\theta} \hat{\mathcal{R}}_{\theta} = \frac{1}{N} \sum_{i=0, \dots, N-1} l(f_{\theta}(x_i), y_i)$$

????????????????

# Let's find the best network then!



$$\min_{\theta} \hat{\mathcal{R}}_{\theta} = \frac{1}{N} \sum_{i=0, \dots, N-1} l(f_{\theta}(x_i), y_i)$$

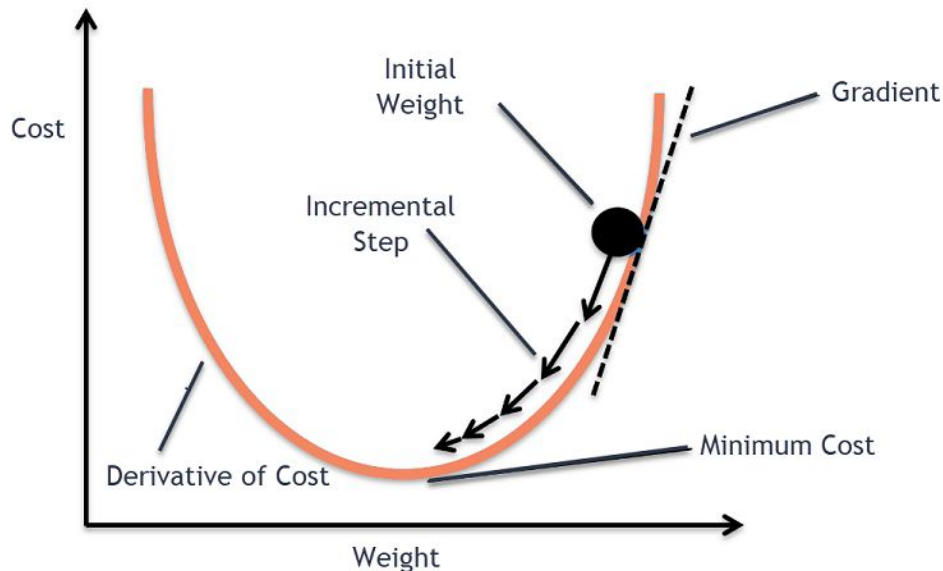
**No closed form!**

- Iteratively make steps of size  $\eta$  to minimize risk

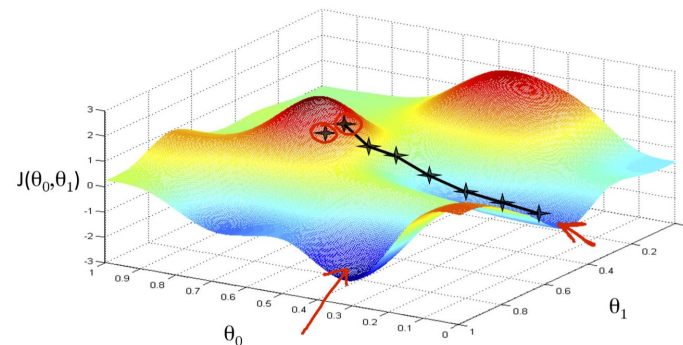
$$\theta^{t+1} := \theta^t - \eta \nabla_{\theta} \hat{\mathcal{R}}_{\theta}$$

- Elementwise form:

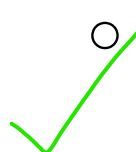
$$\theta_i^{t+1} := \theta_i^t - \eta \frac{\partial \hat{\mathcal{R}}_{\theta}}{\partial \theta_i}$$



- Guarantees on convergence under conditions
  - Lipschitz gradient gives nice upper bounds
  - Adaptive gradient steps can offer guarantees
    - Steps traditionally fixed
- No guarantee to find a global optimum
  - Quite unlikely
  - Gravitates towards Local optimum



- Deep learning deals in large-scale
  - Big datasets
  - Big models
  - GD can get expensive!
- Stochastic Gradient Descent
  - Work on small batches of data  $B$  instead of  $D$



$$\theta^{t+1} := \theta^t - \eta \nabla_{\theta} \mathcal{R}_{\theta}(\hat{B})$$

- Noisier process
- 




- No closed form Risk Minimization solution
  - Networks are too complex
  - Much worse behaved than linear regression
- Solution: Gradient Descent
  - Find direction of  $\theta$  that minimizes the risk
  - Make a step in the direction
  - Repeat

# 5. Backprop


$$\theta^{t+1} := \theta^t - \eta \nabla_{\theta} \mathcal{R}_{\theta}(\hat{B})$$

- Requires finding the risk gradient wrt parameters



$$\nabla_{\theta} \mathcal{R}_{\theta}(\hat{B}) = \frac{1}{\#B} \sum_{k=0, \dots, B-1} \nabla_{\theta} l(f_{\theta}(x_k), y_k)$$

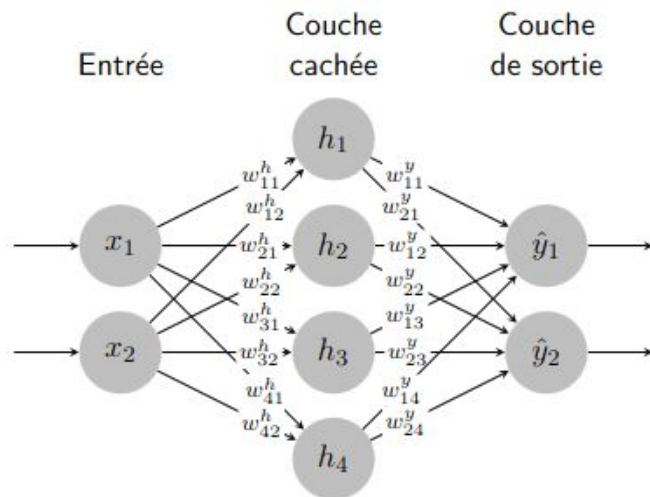
- 
- Boils down to computing gradients for one sample

$$\nabla_{\theta} l(f_{\theta}(x), y)$$

$$l := l(f_{\theta}(x), y)$$

- Networks are complex but made of simple parts!
  - Simple gradients of component functions
  - Chain-rule allows decomposition into simple gradients  $\frac{\partial l}{\partial w} = \frac{\partial l}{\partial a} \frac{\partial a}{\partial w}$

- 
- Need to store intermediate activations “a” to evaluate partial derivatives  $\frac{\partial a}{\partial w}$

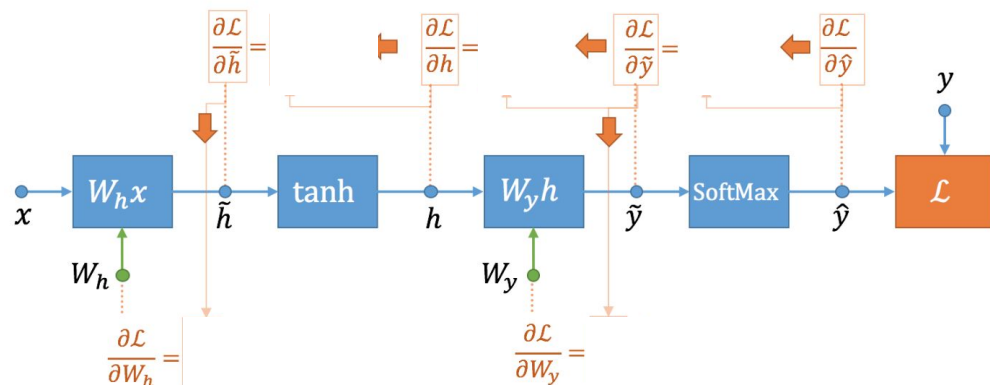
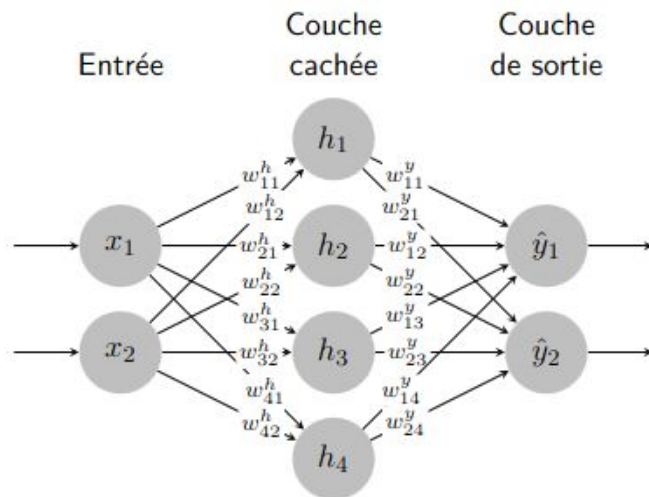


- Simple 1 hidden layer MLP
  - 2 inputs
  - 2 outputs
  - 4 hidden activations
- ✓ ● Classification problem
  - Outputs probabilities
  - Cross-entropy loss

$$l_{CE}(\hat{y}, y) = - \sum_{i=0}^{\#Classes-1} y_i \log(\hat{y}_i)$$

# Example: Tanh MLP

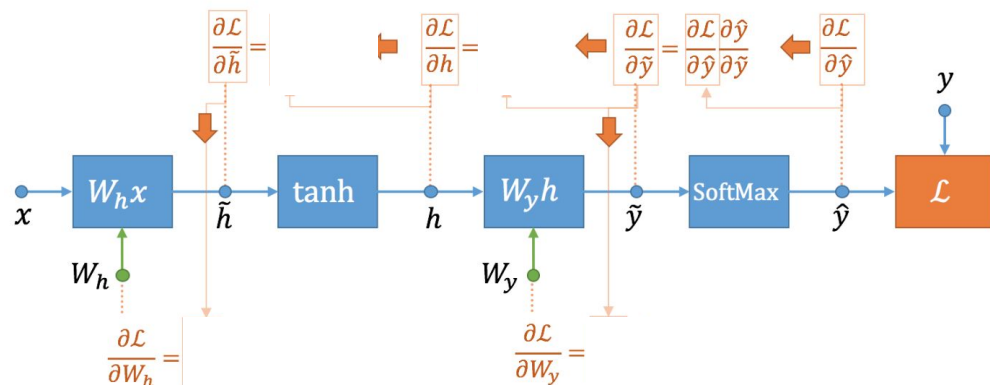
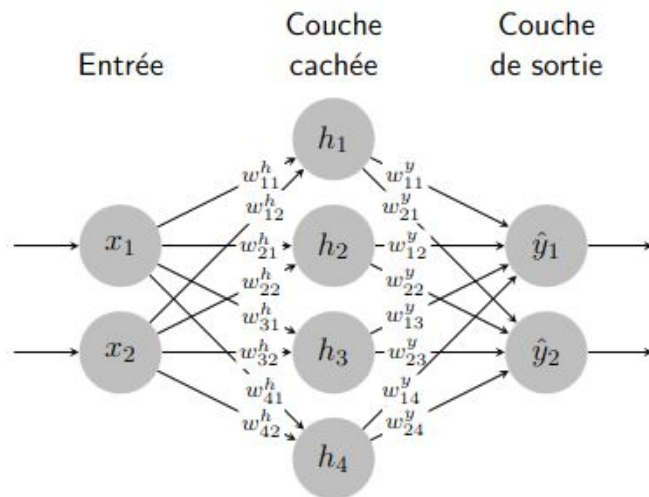
$$l_{CE}(\hat{y}, y) = - \sum_{i=0}^{\#Classes-1} y_i \log(\hat{y}_i)$$



$$\begin{cases} \tilde{h}_i = \sum_{j=1}^{n_x} W_{i,j}^h x_j + b_i^h \\ h_i = \tanh(\tilde{h}_i) \\ \tilde{y}_i = \sum_{j=1}^{n_h} W_{i,j}^y h_j + b_i^y \\ \hat{y}_i = \text{SoftMax}(\tilde{y}_i) = \frac{e^{\tilde{y}_i}}{\sum_{j=1}^{n_y} e^{\tilde{y}_j}} \end{cases}$$

# Example: Tanh MLP

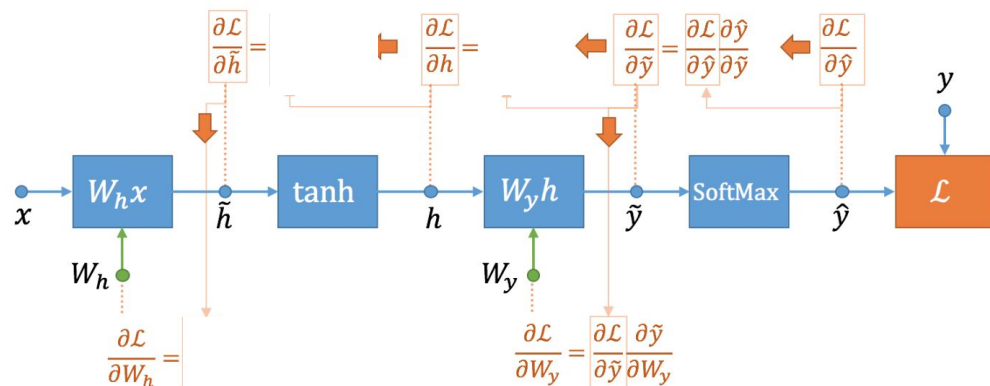
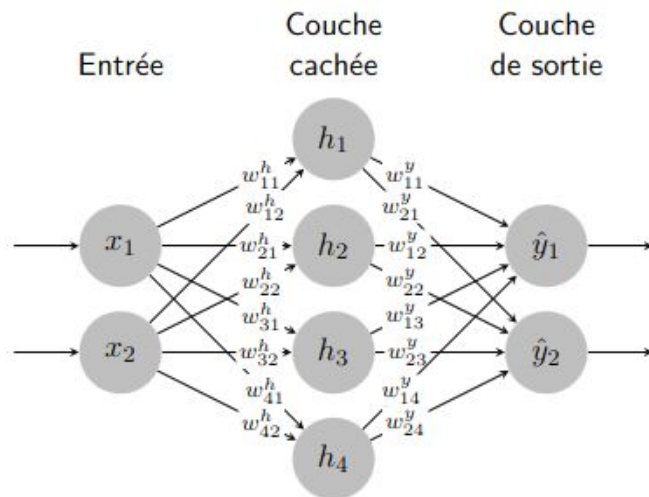
$$l_{CE}(\hat{y}, y) = - \sum_{i=0}^{\#Classes-1} y_i \log(\hat{y}_i)$$



$$\begin{cases} \tilde{h}_i = \sum_{j=1}^{n_x} W_{i,j}^h x_j + b_i^h \\ h_i = \tanh(\tilde{h}_i) \\ \tilde{y}_i = \sum_{j=1}^{n_h} W_{i,j}^y h_j + b_i^y \\ \hat{y}_i = \text{SoftMax}(\tilde{y}_i) = \frac{e^{\tilde{y}_i}}{\sum_{j=1}^{n_y} e^{\tilde{y}_j}} \end{cases}$$

# Example: Tanh MLP

$$l_{CE}(\hat{y}, y) = - \sum_{i=0}^{\#Classes-1} y_i \log(\hat{y}_i)$$

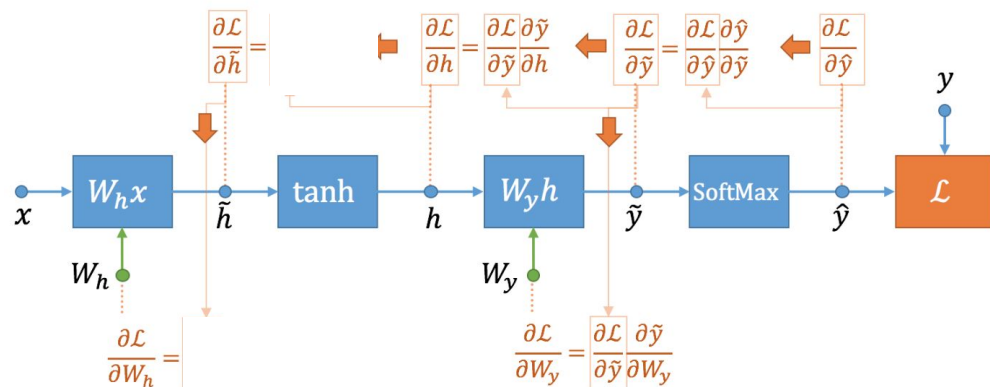
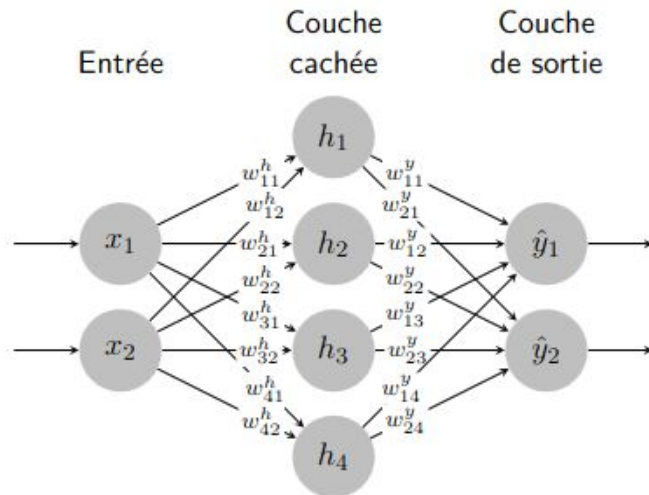


$$\begin{cases} \tilde{h}_i = \sum_{j=1}^{n_x} W_{i,j}^h x_j + b_i^h \\ h_i = \tanh(\tilde{h}_i) \\ \tilde{y}_i = \sum_{j=1}^{n_h} W_{i,j}^y h_j + b_i^y \\ \hat{y}_i = \text{SoftMax}(\tilde{y}_i) = \frac{e^{\tilde{y}_i}}{\sum_{j=1}^{n_y} e^{\tilde{y}_j}} \end{cases}$$



# Example: Tanh MLP

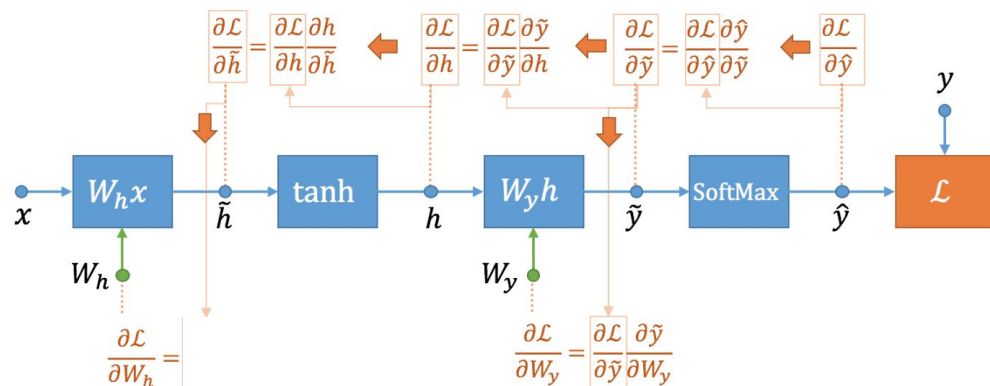
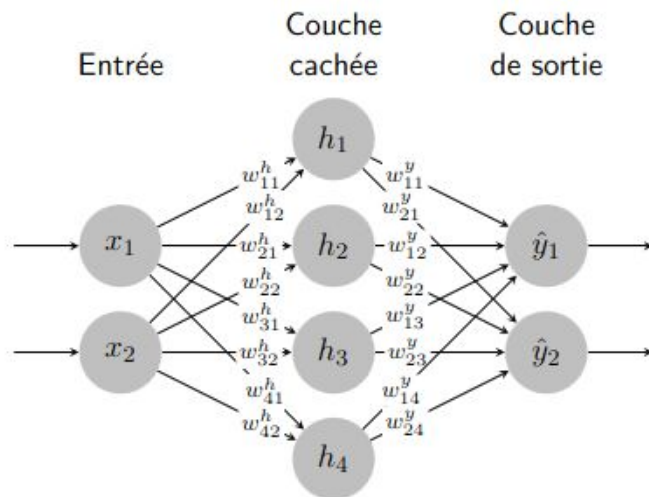
$$l_{CE}(\hat{y}, y) = - \sum_{i=0}^{\#Classes-1} y_i \log(\hat{y}_i)$$



$$\begin{cases} \tilde{h}_i = \sum_{j=1}^{n_x} W_{i,j}^h x_j + b_i^h \\ h_i = \tanh(\tilde{h}_i) \\ \tilde{y}_i = \sum_{j=1}^{n_h} W_{i,j}^y h_j + b_i^y \\ \hat{y}_i = \text{SoftMax}(\tilde{y}_i) = \frac{e^{\tilde{y}_i}}{\sum_{j=1}^{n_y} e^{\tilde{y}_j}} \end{cases}$$

# Example: Tanh MLP

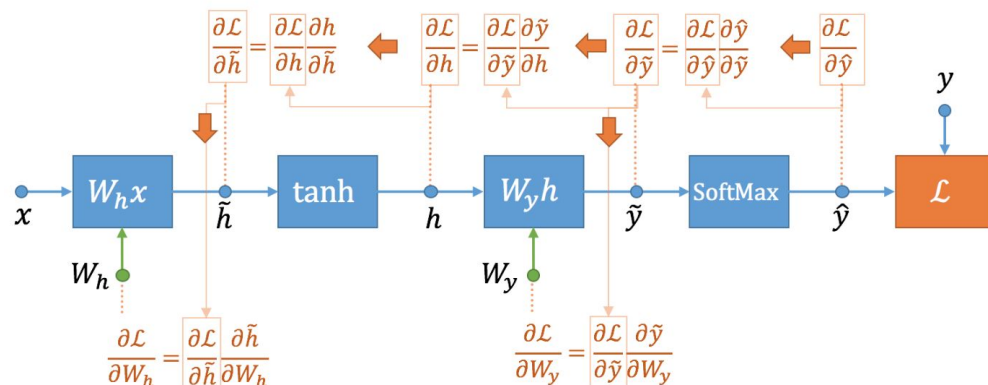
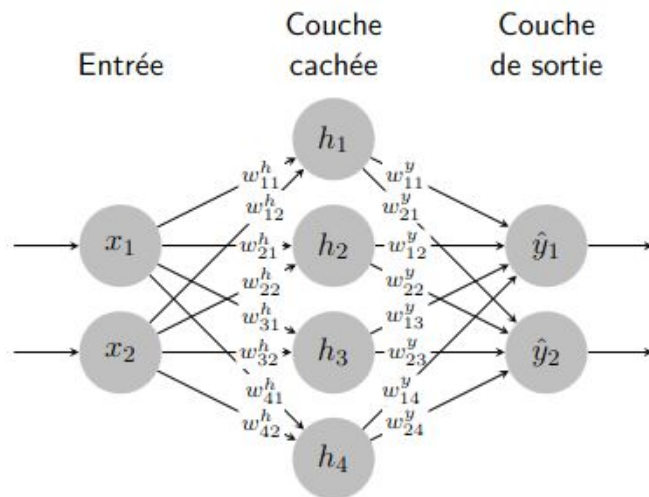
$$l_{CE}(\hat{y}, y) = - \sum_{i=0}^{\#Classes-1} y_i \log(\hat{y}_i)$$



$$\begin{cases} \tilde{h}_i = \sum_{j=1}^{n_x} W_{i,j}^h x_j + b_i^h \\ h_i = \tanh(\tilde{h}_i) \\ \tilde{y}_i = \sum_{j=1}^{n_h} W_{i,j}^y h_j + b_i^y \\ \hat{y}_i = \text{SoftMax}(\tilde{y}_i) = \frac{e^{\tilde{y}_i}}{\sum_{j=1}^{n_y} e^{\tilde{y}_j}} \end{cases}$$

# Example: Tanh MLP

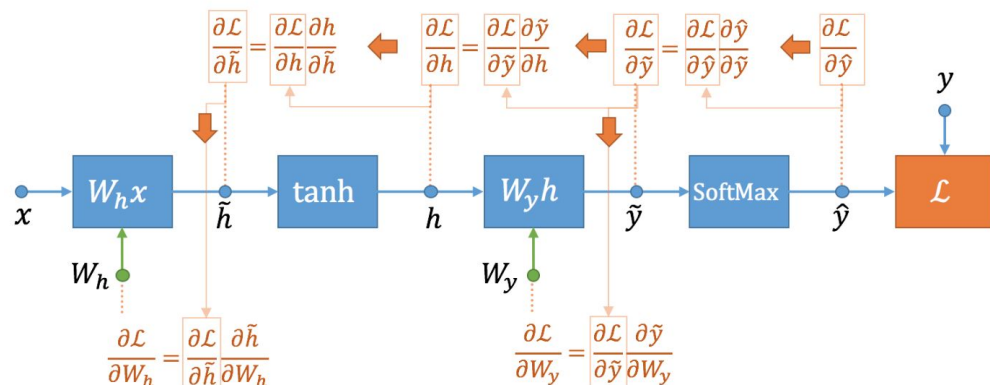
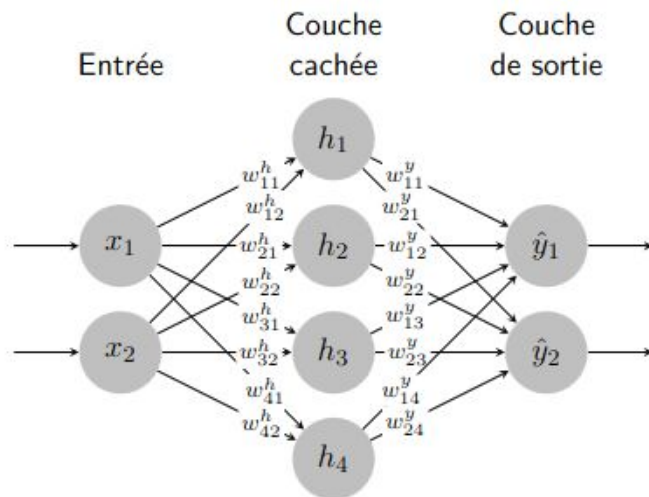
$$l_{CE}(\hat{y}, y) = - \sum_{i=0}^{\#Classes-1} y_i \log(\hat{y}_i)$$



$$\left\{ \begin{array}{l} \tilde{h}_i = \sum_{j=1}^{n_x} W_{i,j}^h x_j + b_i^h \\ h_i = \tanh(\tilde{h}_i) \\ \tilde{y}_i = \sum_{j=1}^{n_h} W_{i,j}^y h_j + b_i^y \\ \hat{y}_i = \text{SoftMax}(\tilde{y}_i) = \frac{e^{\tilde{y}_i}}{\sum_{j=1}^{n_y} e^{\tilde{y}_j}} \end{array} \right\} \quad \left\{ \begin{array}{l} \delta_i^y = \frac{\partial \ell}{\partial \tilde{y}_i} = \\ \frac{\partial \ell}{\partial W_{i,j}^y} = \\ \frac{\partial \ell}{\partial b_i^y} = \end{array} \right.$$

# Example: Tanh MLP

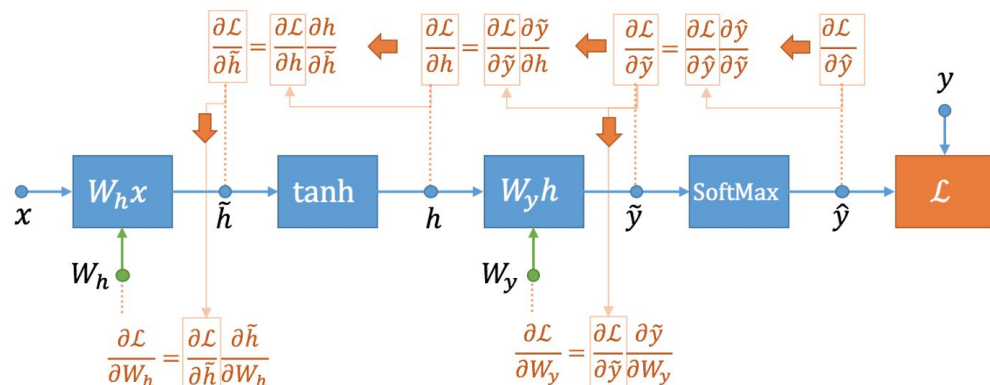
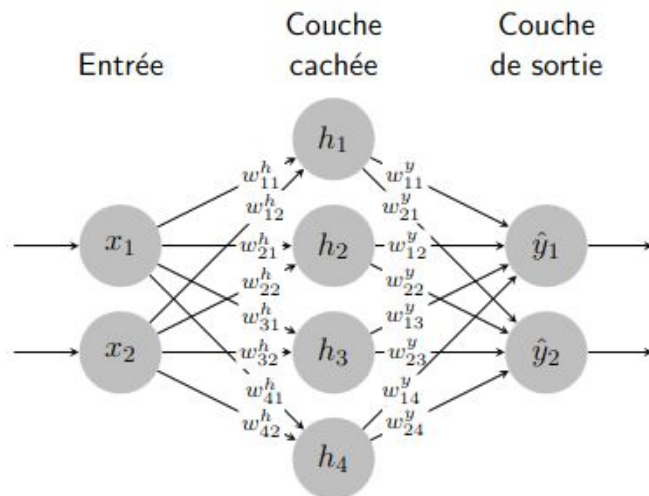
$$l_{CE}(\hat{y}, y) = - \sum_{i=0}^{\#Classes-1} y_i \log(\hat{y}_i)$$



$$\left\{ \begin{array}{l} \tilde{h}_i = \sum_{j=1}^{n_x} W_{i,j}^h x_j + b_i^h \\ h_i = \tanh(\tilde{h}_i) \\ \tilde{y}_i = \sum_{j=1}^{n_h} W_{i,j}^y h_j + b_i^y \\ \hat{y}_i = \text{SoftMax}(\tilde{y}_i) = \frac{e^{\tilde{y}_i}}{\sum_{j=1}^{n_y} e^{\tilde{y}_j}} \end{array} \right\} \quad \left\{ \begin{array}{l} \delta_i^y = \frac{\partial \ell}{\partial \tilde{y}_i} = \hat{y}_i - y_i \\ \frac{\partial \ell}{\partial W_{i,j}^y} = \delta_i^y h_j \\ \frac{\partial \ell}{\partial b_i^y} = \delta_i^y \end{array} \right.$$

# Example: Tanh MLP

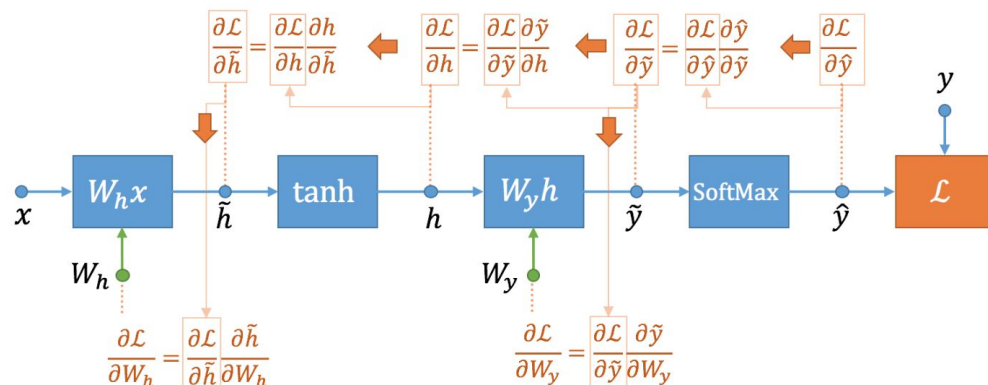
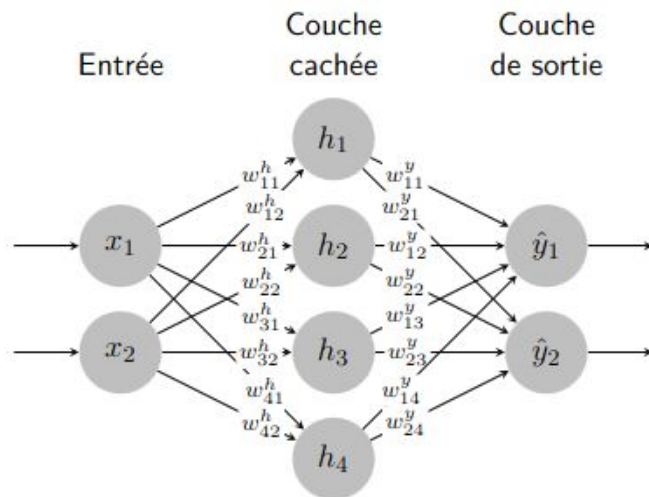
$$l_{CE}(\hat{y}, y) = - \sum_{i=0}^{\#Classes-1} y_i \log(\hat{y}_i)$$



$$\left\{ \begin{array}{l} \tilde{h}_i = \sum_{j=1}^{n_x} W_{i,j}^h x_j + b_i^h \\ h_i = \tanh(\tilde{h}_i) \\ \tilde{y}_i = \sum_{j=1}^{n_h} W_{i,j}^y h_j + b_i^y \\ \hat{y}_i = \text{SoftMax}(\tilde{y}_i) = \frac{e^{\tilde{y}_i}}{\sum_{j=1}^{n_y} e^{\tilde{y}_j}} \end{array} \right. \quad \left\{ \begin{array}{l} \delta_i^y = \frac{\partial \ell}{\partial \tilde{y}_i} = \hat{y}_i - y_i \\ \frac{\partial \ell}{\partial W_{i,j}^y} = \delta_i^y h_j \\ \frac{\partial \ell}{\partial b_i^y} = \delta_i^y \\ \delta_i^h = \frac{\partial \ell}{\partial \tilde{h}_i} = \\ \frac{\partial \ell}{\partial W_{i,j}^h} = \\ \frac{\partial \ell}{\partial b_i^h} = \end{array} \right.$$

# Example: Tanh MLP

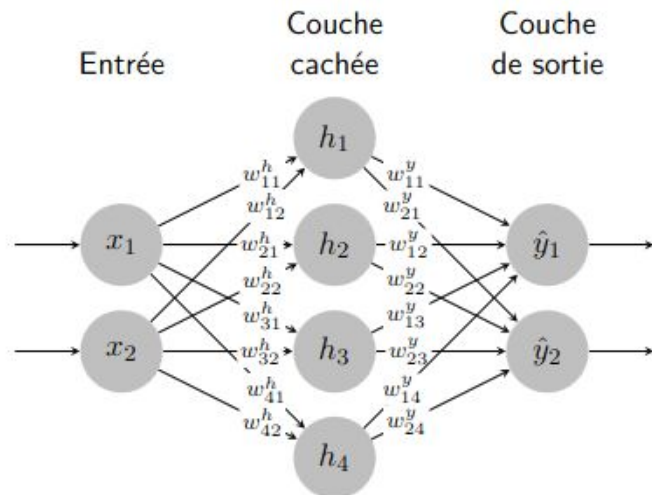
$$l_{CE}(\hat{y}, y) = - \sum_{i=0}^{\#Classes-1} y_i \log(\hat{y}_i)$$



$$\left\{ \begin{array}{l} \tilde{h}_i = \sum_{j=1}^{n_x} W_{i,j}^h x_j + b_i^h \\ h_i = \tanh(\tilde{h}_i) \\ \tilde{y}_i = \sum_{j=1}^{n_h} W_{i,j}^y h_j + b_i^y \\ \hat{y}_i = \text{SoftMax}(\tilde{y}_i) = \frac{e^{\tilde{y}_i}}{\sum_{j=1}^{n_y} e^{\tilde{y}_j}} \end{array} \right. \quad \left\{ \begin{array}{l} \delta_i^y = \frac{\partial \ell}{\partial \tilde{y}_i} = \hat{y}_i - y_i \\ \frac{\partial \ell}{\partial W_{i,j}^y} = \delta_i^y h_j \\ \frac{\partial \ell}{\partial b_i^y} = \delta_i^y \\ \delta_i^h = \frac{\partial \ell}{\partial \tilde{h}_i} = (1 - h_i^2) \sum_{j=1}^{n_y} \delta_j^y W_{j,i}^y \\ \frac{\partial \ell}{\partial W_{i,j}^h} = \delta_i^h x_j \\ \frac{\partial \ell}{\partial b_i^h} = \delta_i^h \end{array} \right.$$

- Core problem: Find gradient updates
- Gradient can be computed efficiently with backpropagation
  - Chain rule starting from the “end” of the network
  - Keep network activation to evaluate gradients
  - Simple layers mean simple gradient blocks





$$\begin{cases} \tilde{\mathbf{h}} = \mathbf{x}\mathbf{W}^h{}^\top + \mathbf{b}^h \\ \mathbf{h} = \tanh(\tilde{\mathbf{h}}) \\ \tilde{\mathbf{y}} = \mathbf{h}\mathbf{W}^y{}^\top + \mathbf{b}^y \\ \hat{\mathbf{y}} = \text{SoftMax}(\tilde{\mathbf{y}}) \end{cases}$$

$$\begin{cases} \nabla_{\hat{\mathbf{y}}} = \hat{\mathbf{y}} - \mathbf{y} \\ \nabla_{\mathbf{W}^y} = \nabla_{\hat{\mathbf{y}}}{}^\top \mathbf{h} \\ \nabla_{\mathbf{b}^y} = \nabla_{\hat{\mathbf{y}}}{}^\top \\ \nabla_{\tilde{\mathbf{h}}} = (\nabla_{\hat{\mathbf{y}}} \mathbf{W}^y) \odot (1 - \mathbf{h}^2) \\ \nabla_{\mathbf{W}^h} = \nabla_{\tilde{\mathbf{h}}}{}^\top \mathbf{x} \\ \nabla_{\mathbf{b}^h} = \nabla_{\tilde{\mathbf{h}}}{}^\top \end{cases}$$

- Lab 3 practical notebook on Moodle
  - Implement this by hand with basic torch!
  - Careful with batch dimension!