Lecture 9: Backpropagation

Optimization for data sciences

Rémy Sun

remy.sun@inria.fr

Course organization

Course organization

- Introduction to optimization
  - A few problems of interest
  - Quick mathematical refresher

- Convex problems (Following Stephen Boyd)
  - Convex sets
  - Convex functions
  - Convex problems
  - Simplex algorithm for Linear Programming

Course organization

- Duality (for convex problems)
  - Lagrangian and dual function
  - Dual problem
  - Qualification constraints
  - KKT conditions

- Newton’s Descent and Barrier methods for convex case
  - Descent for the unconstrained problems
  - Equality constrained problems
  - Interior point methods
  - Lab session!

Course organization

UNIVERSITÉ CÔTE D'AZUR

- What about the real (neural) world?
  - Problem statement
  - Let's try to solve it!
  - Gradient descent with(out) convexity
  - Gradient descent variants

- **Backpropagation**
  - **Chain rule derivation**
  - **Dynamic programming**
  - **Backpropagation**
  - **Lab session 2!**

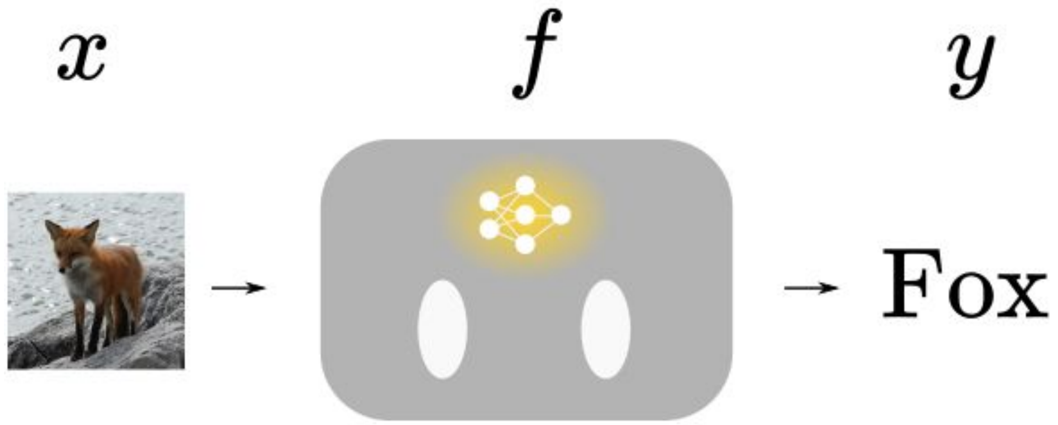
Evaluation

- Reports on lab sessions
  - Labs on jupyter notebooks
    - Not every session
  - Explain the code done in the session
  - Summarize what is done in the practical

- Written Exam
  - Theoretical questions
  - We will do exercises in class

Refresher on convexity!

Problem statement: Ideal case



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

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

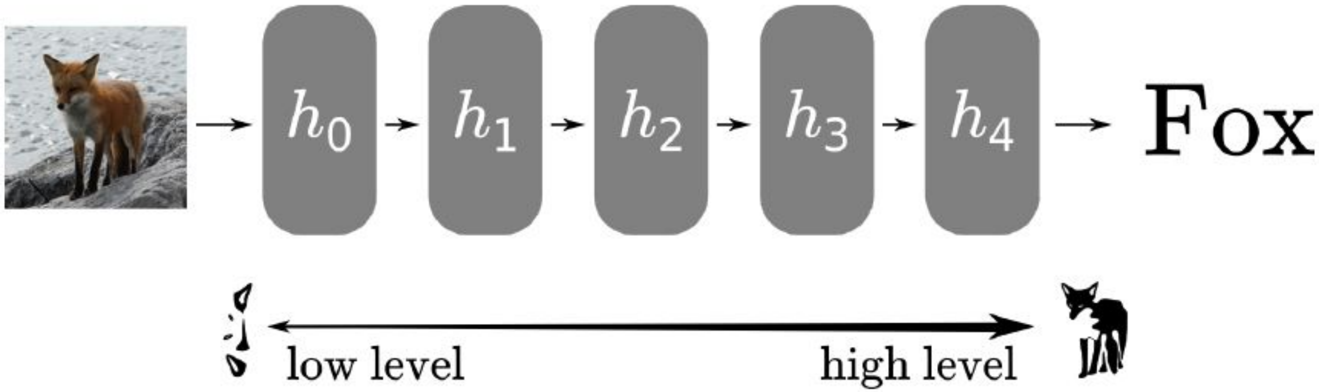
Minimizing Risk

- 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 $\mathcal{D}$
  - ***<u>Empirical Risk Minimization</u>***
  - If the (x,y) are i.i.d drawn from $\mathcal{D}$ can be expressed as a mean over the dataset

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

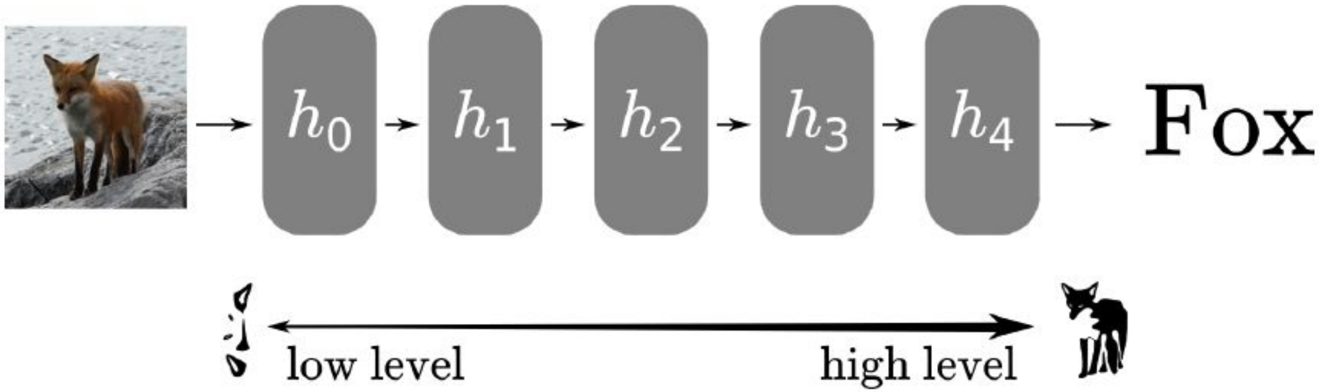
Neural network functions

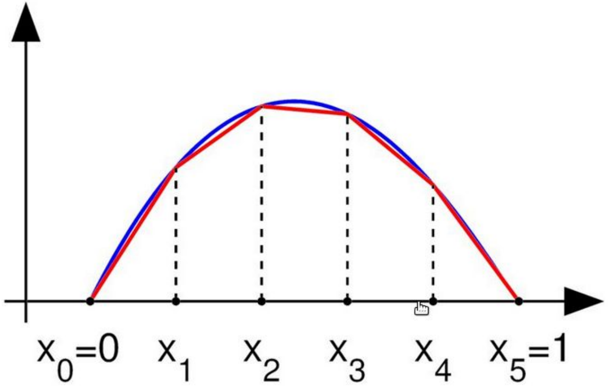


- Neural networks are sequences of simple functions

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

Interlude: Formalization of dense networks





- Highly expressive
  - Can fit many types of distributions

(Non-convex) Unconstrained problem

$$
\begin{array}{ll}
\text{minimize} & f_0(x) \\
\text{subject to} & f_i(x) \leq 0, \quad i = 1, \ldots, m \\
& h_i(x) = 0, \quad i = 1, \ldots, p
\end{array}
$$

- Cost function is given by neural network and loss
- No constraints!
  - Easy unconstrained problem!

- **Not convex, for a number of reasons.**

Let's do discount descent: Gradient descent

UNIVERSITÉ CÔTE D'AZUR

- What do we need?
  - Step size
    - Fixed step size
  - Gradient
    - Could get pretty expensive too…
    - Obtained through backpropagation!
  - No Hessian!

Takeaway

- Under reasonable L-smooth assumption
  - Gradient descent converges!
    - To something
    - With a minuscule fixed step size
- General deep learning heuristics
  - Most local minimums are similarly good/bad
  - Big networks have very few really bad mins

1. Some context

We need the gradient…

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

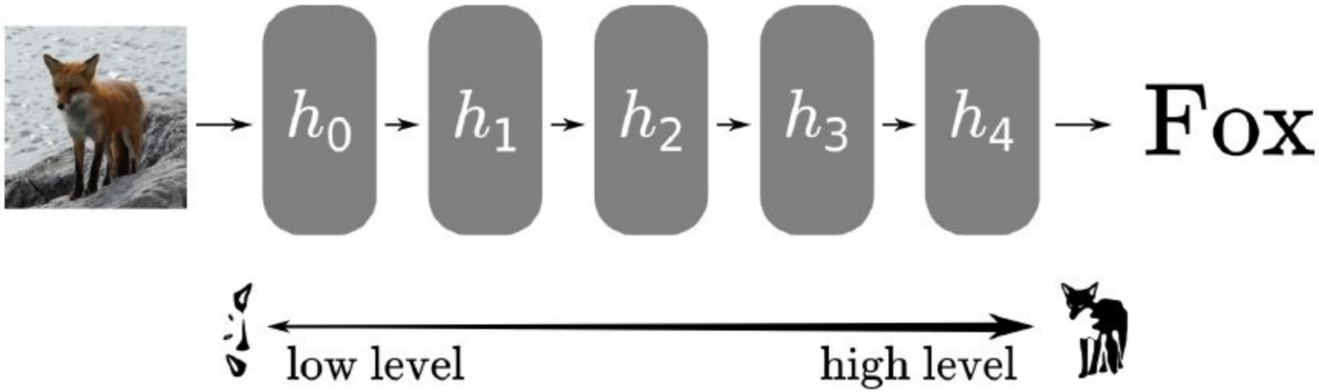
- Requires finding the risk gradient wrt parameters

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

- Boils down to computing gradients for one sample

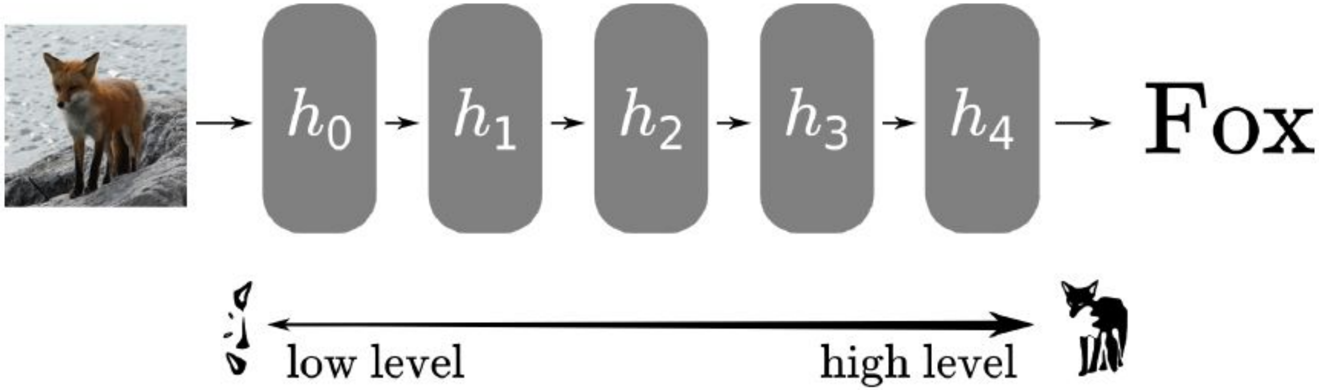
$$\nabla_\theta l(f_\theta(x), y)$$

Let's get the gradients!



- Neural networks embed complex functions
    - Reason we use them

Let's get the gradients!



- Neural networks embed complex functions
  - Reason we use them

- **How do we get the gradients?!**

Gradients for neural networks

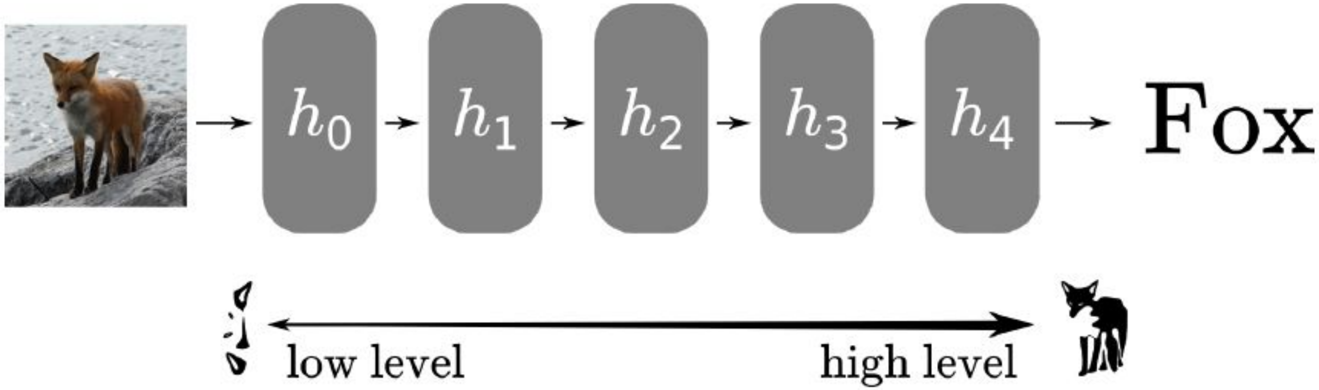
- Not an easy problem originally

- First neural networks do not use gradients
    - "Hard" value perceptron
    - Rosenblatt's algorithm

- 1970: Modern Backpropagation (reverse mode)
    - Computes gradients
    - Uses Chain rule derivation
    - With bottom-up dynamic programming

2. Chain rule

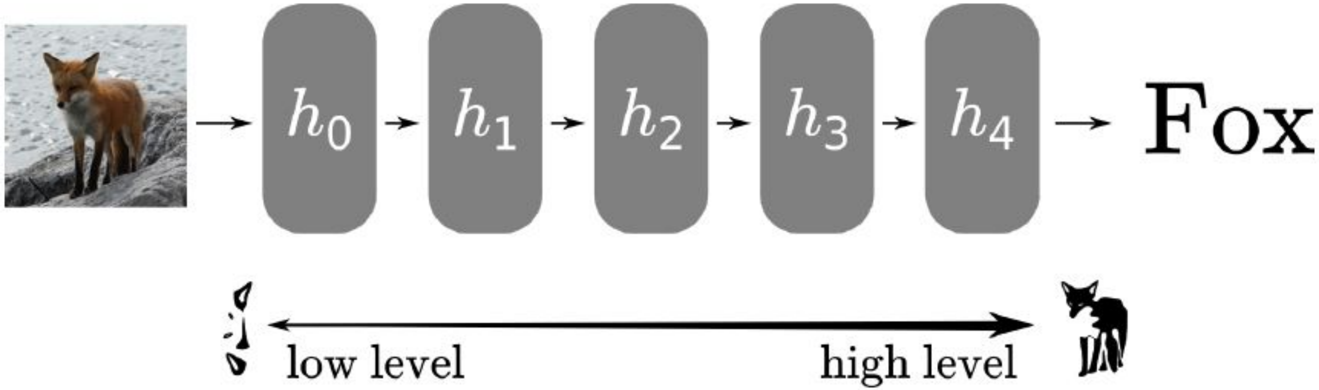
Neural network functions



- The complete network is complicated

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

Neural network functions



- But individual derivatives are simple!

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

Leibniz said: Chain rule derivation

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$$

- Derivatives can be broken down into simpler parts
  - As a product!

- Allows us to use the very simple functions
  - E.g. $\tilde{h}_i = \sum_{j=1}^{n_x} W_{i,j}^h \, x_j + b_i^h$ has trivial partial derivatives

- Known for a very long time (17th century)

Chain rule in neural networks

- Chain rule derivation powers backpropagation
  - Leverages simple component functions
  - Easy to break down

- Works as long as we know how to derivate layers

- Backpropagation is not just the chain rule
  - Exploding costs
  - Lots of redundant computations

Chain rule in neural networks

- Chain rule derivation powers backpropagation
  - Leverages simple component functions
  - Easy to break down

- Works as long as we know how to derivate layers

- **Backpropagation is not just the chain rule**
  - Exploding costs
  - Lots of redundant computations
  - Backprop is **efficient** derivation

3. Dynamic Programming

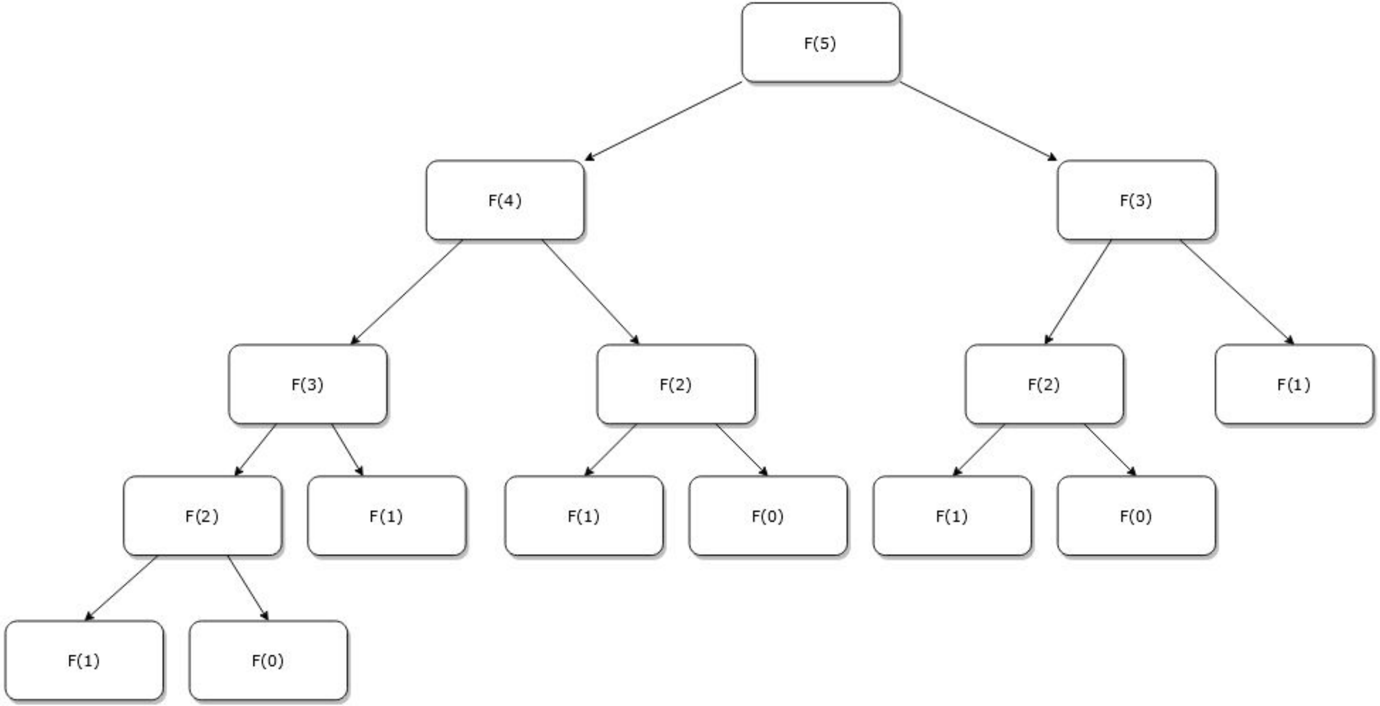
Dynamic programming

- Efficient computations of complex problem
  - Bellman (1950)
  - Break down into simpler (re-occuring) problems
  - And remember your solutions

- Avoid redundant computations
  - By using additional memory to store results
  - By properly structuring computations

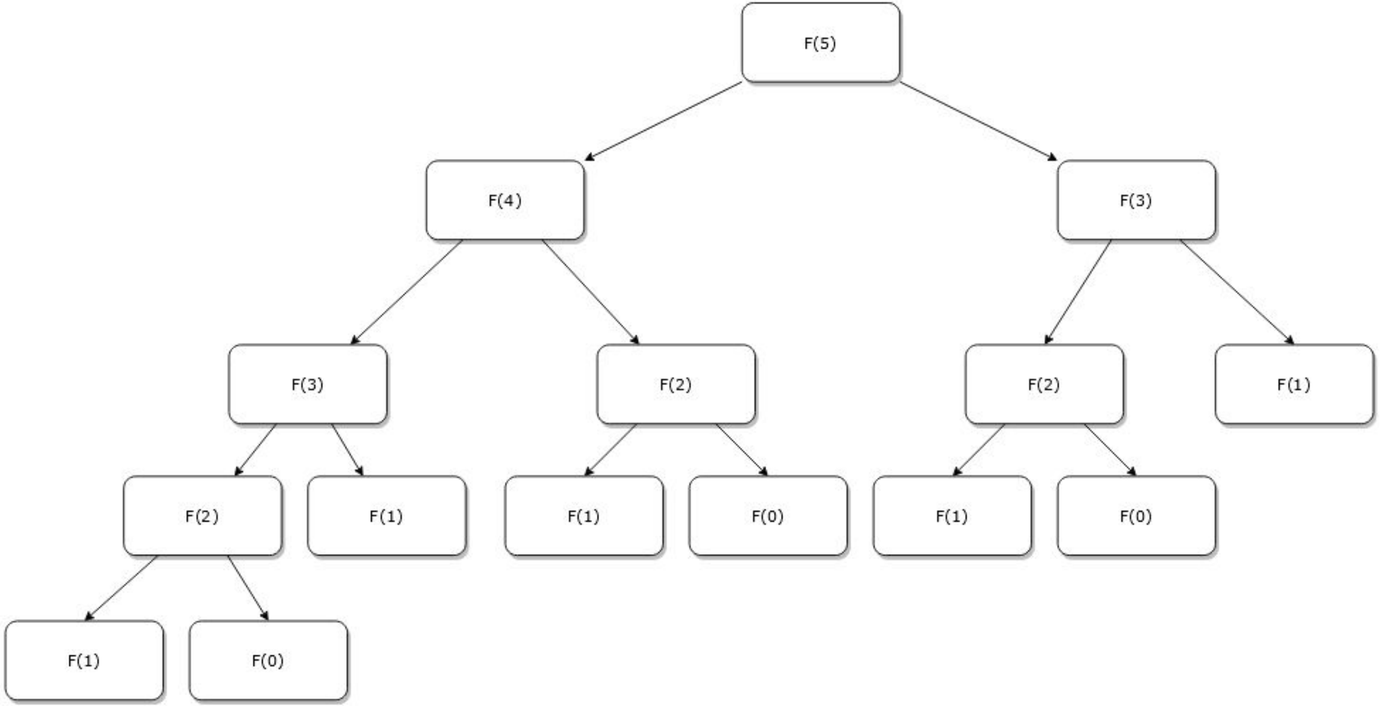
Example: Fibonacci sequence

- F(0)=0, F(1)=1, F(N+2)=F(N)+F(N+1)



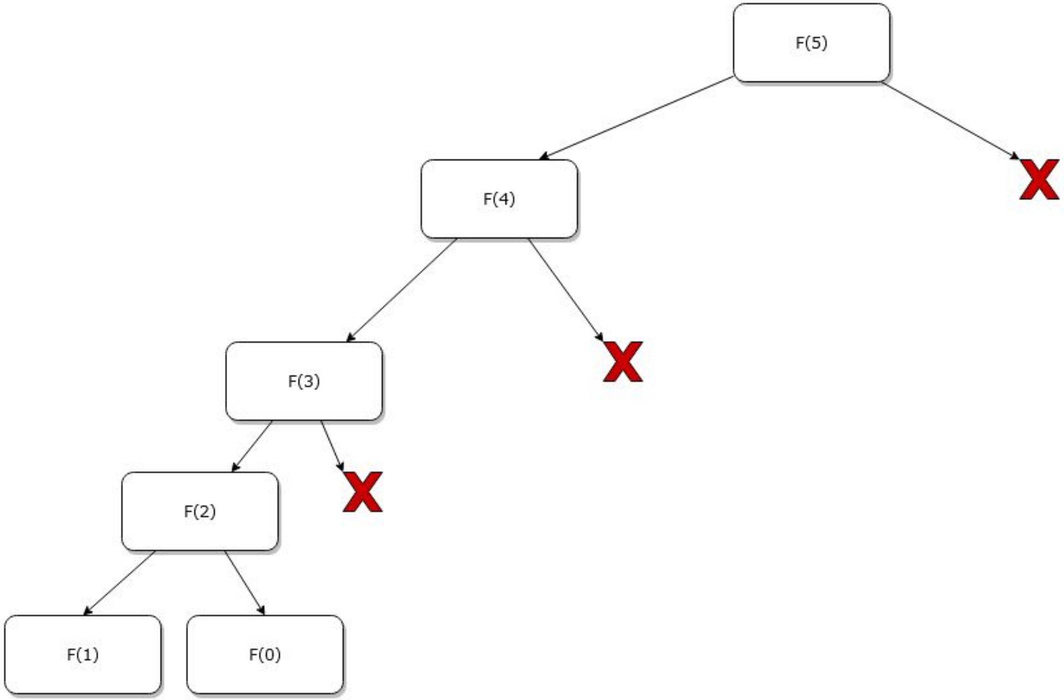
Example: Fibonacci sequence

Lots and lots and lots of computations…



Example: Fibonacci sequence (Top-down)

- Memoization: Store sub-problem solutions as needed

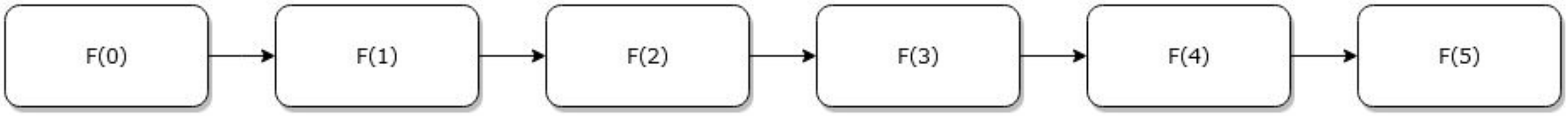


Example: Fibonacci sequence (Bottom-up)

- Tabulation: Start from basic problems
  - Tabulate results as we go along

- One single forward pass this way!

- Memoization vs. Tabulation depends on the problem



Takeaway

- Efficient computations of complex problem
  - Bellman (1950)
- Avoid redundant computations
  - By using additional memory to store results
  - By properly structuring computations
- Top-down (Memoization): Start from final problem
- Bottom-up (Tabulation): Start from basic problem

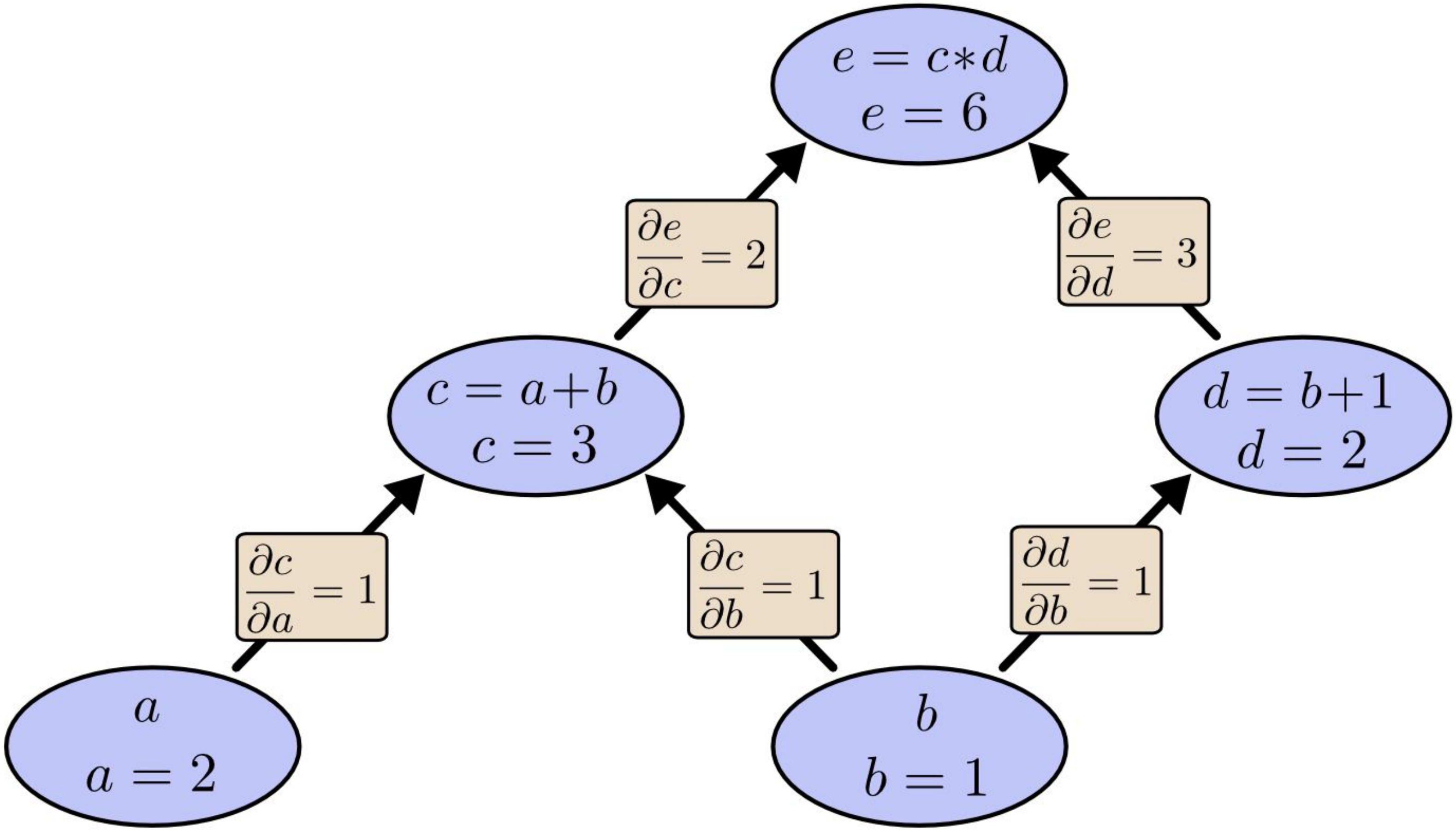
4. Backpropagation

Backpropagation

- Modern Backpropagation: Linnainmaa (1970)
  - In his Master's thesis!
  - Some precursor efforts before
  - Theorized by Rosenblatt for perceptrons

- Efficient differentiation in a computational graph
  - Reverse mode autodiff
  - Not for neural network per se
  - Applied to neural networks later on

Backpropagation



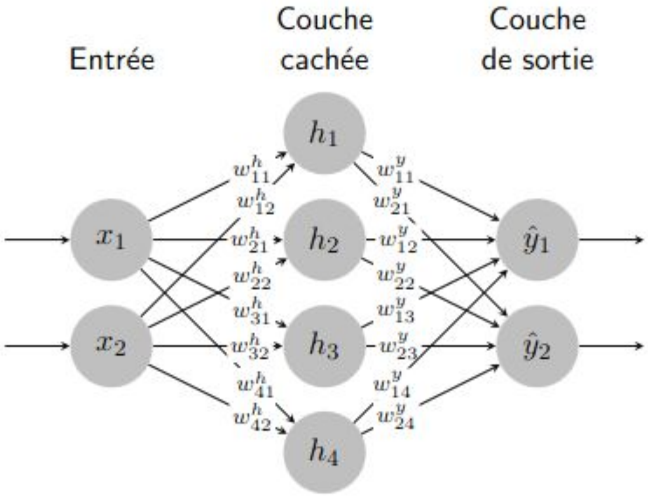
Backpropagation on networks (Informal)

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

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

- Only one pass (in backward)!

Example: Tanh MLP

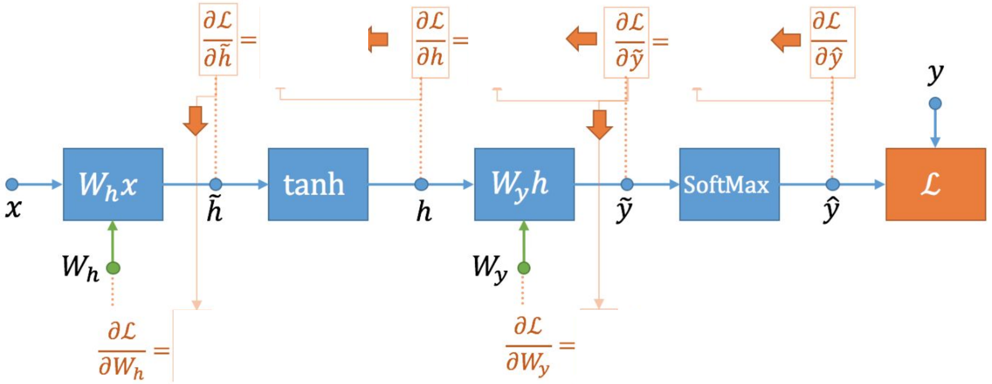


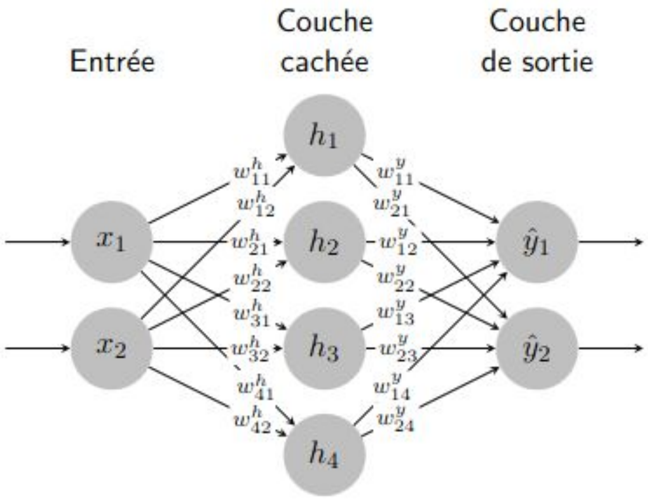
- Simple 1 hidden layer MLP
  - 2 inputs
  - 2 outputs
  - 4 hidden activations
- Classification problem
  - Outputs probabilities
  - Cross-entropy loss

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

Example: Tanh MLP

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



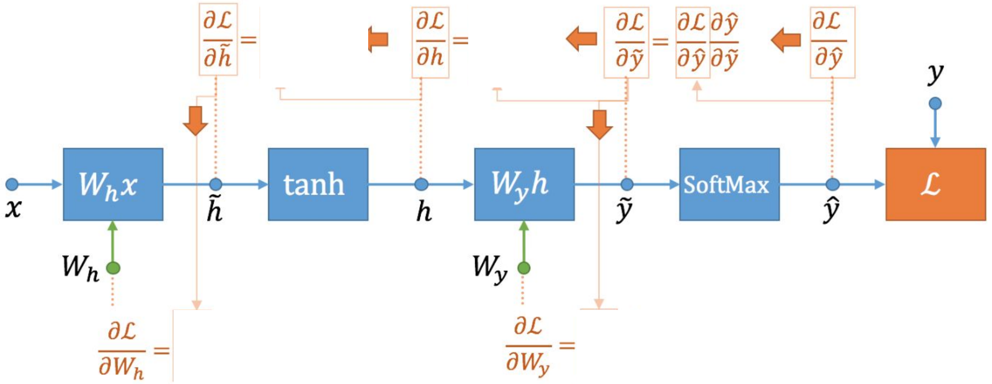


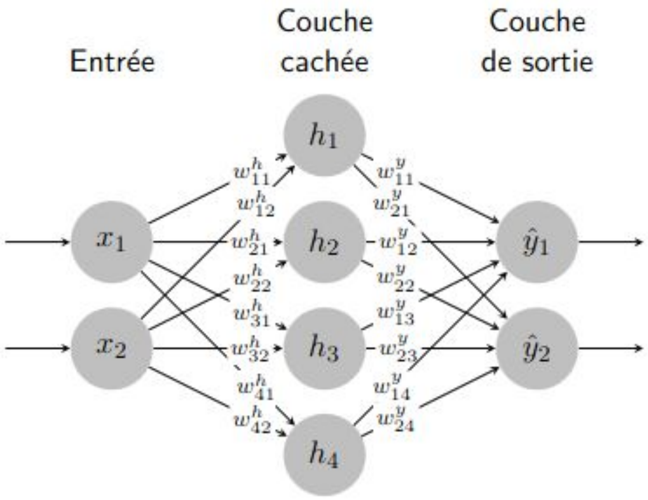
$$\begin{cases} \tilde{h}_i = \sum_{j=1}^{n_x} W^h_{i,j}\, x_j + b^h_i \\ h_i = \tanh(\tilde{h}_i) \\ \tilde{y}_i = \sum_{j=1}^{n_h} W^y_{i,j}\, h_j + b^y_i \\ \hat{y}_i = \mathrm{SoftMax}(\tilde{y}_i) = \dfrac{e^{\tilde{y}_i}}{\sum_{j=1}^{n_y} e^{\tilde{y}_j}} \end{cases}$$

Example: Tanh MLP

UNIVERSITÉ CÔTE D'AZUR

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

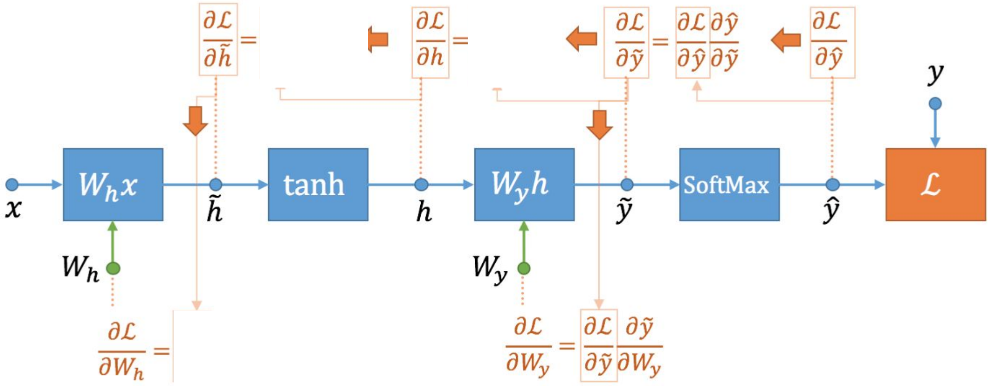


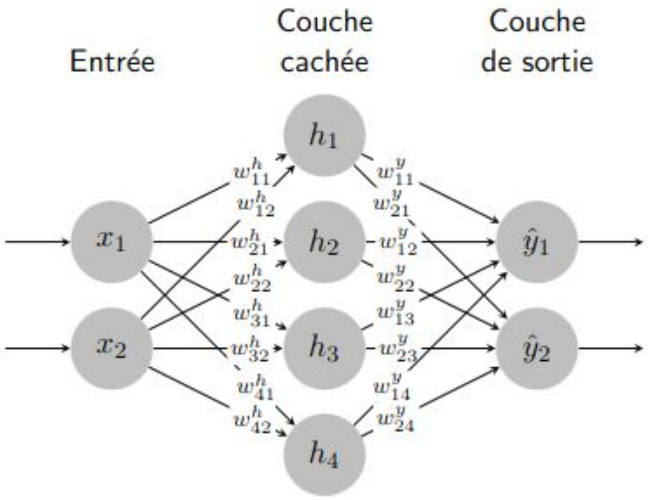


$$\begin{cases} \tilde{h}_i = \sum_{j=1}^{n_x} W^h_{i,j} \; x_j + b^h_i \\ h_i = \tanh(\tilde{h}_i) \\ \tilde{y}_i = \sum_{j=1}^{n_h} W^y_{i,j} \; h_j + b^y_i \\ \hat{y}_i = \mathrm{SoftMax}(\tilde{y}_i) = \dfrac{e^{\tilde{y}_i}}{\sum_{j=1}^{n_y} e^{\tilde{y}_j}} \end{cases}$$

Example: Tanh MLP

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

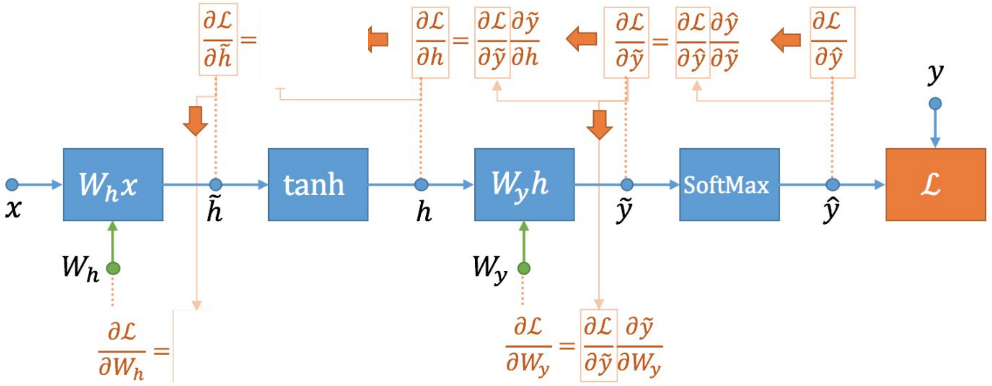


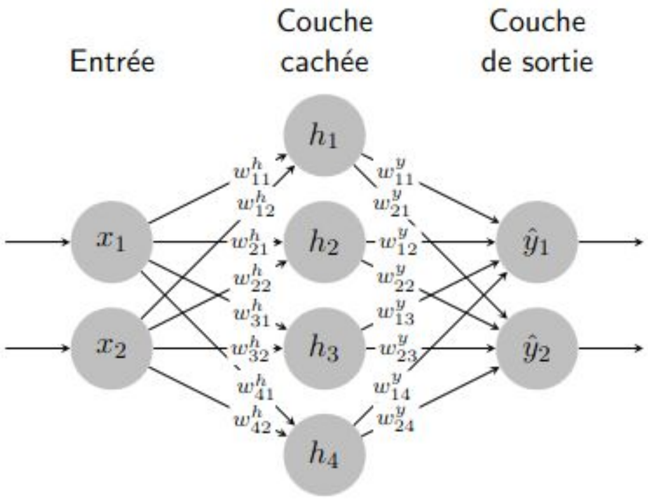


$$\begin{cases} \tilde{h}_i = \sum_{j=1}^{n_x} W^h_{i,j}\ x_j + b^h_i \\ h_i = \tanh(\tilde{h}_i) \\ \tilde{y}_i = \sum_{j=1}^{n_h} W^y_{i,j}\ h_j + b^y_i \\ \hat{y}_i = \mathrm{SoftMax}(\tilde{y}_i) = \dfrac{e^{\tilde{y}_i}}{\sum_{j=1}^{n_y} e^{\tilde{y}_j}} \end{cases}$$

Example: Tanh MLP

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

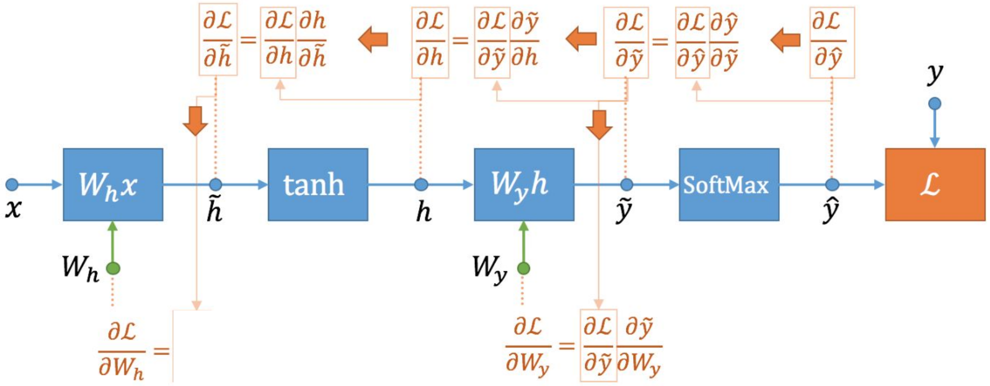


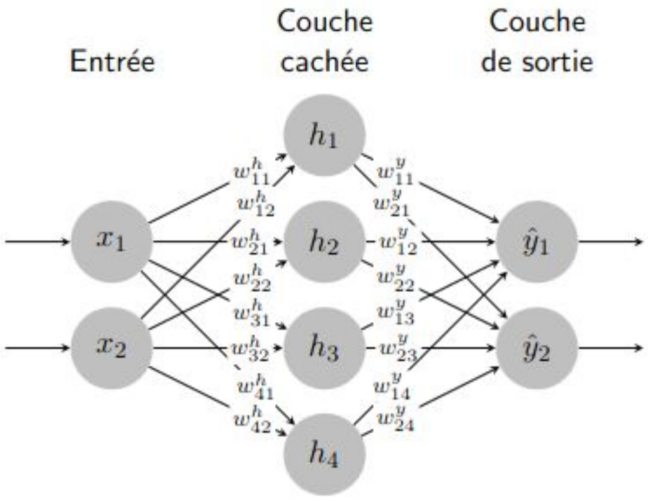


$$
\begin{cases}
\tilde{h}_i = \sum_{j=1}^{n_x} W^h_{i,j} \, x_j + b^h_i \\
h_i = \tanh(\tilde{h}_i) \\
\tilde{y}_i = \sum_{j=1}^{n_h} W^y_{i,j} \, h_j + b^y_i \\
\hat{y}_i = \mathrm{SoftMax}(\tilde{y}_i) = \dfrac{e^{\tilde{y}_i}}{\sum_{j=1}^{n_y} e^{\tilde{y}_j}}
\end{cases}
$$

Example: Tanh MLP

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

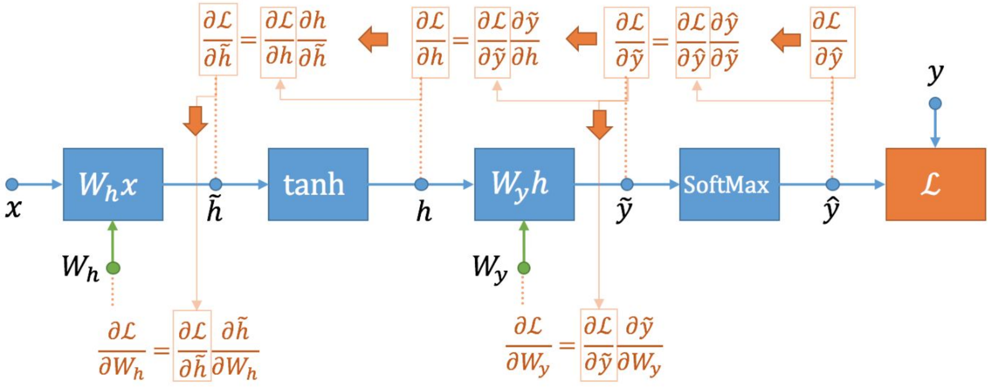


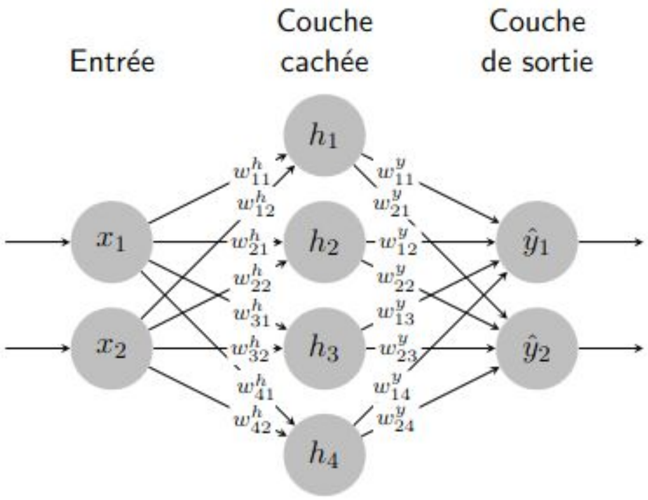


$$\begin{cases} \tilde{h}_i = \sum_{j=1}^{n_x} W^h_{i,j}\ x_j + b^h_i \\ h_i = \tanh(\tilde{h}_i) \\ \tilde{y}_i = \sum_{j=1}^{n_h} W^y_{i,j}\ h_j + b^y_i \\ \hat{y}_i = \mathrm{SoftMax}(\tilde{y}_i) = \dfrac{e^{\tilde{y}_i}}{\sum_{j=1}^{n_y} e^{\tilde{y}_j}} \end{cases}$$

Example: Tanh MLP

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



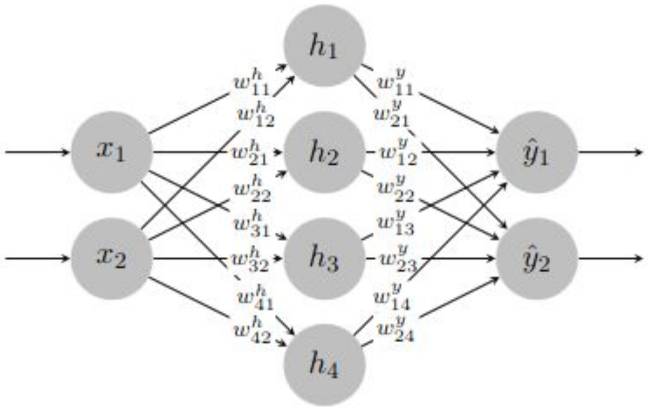


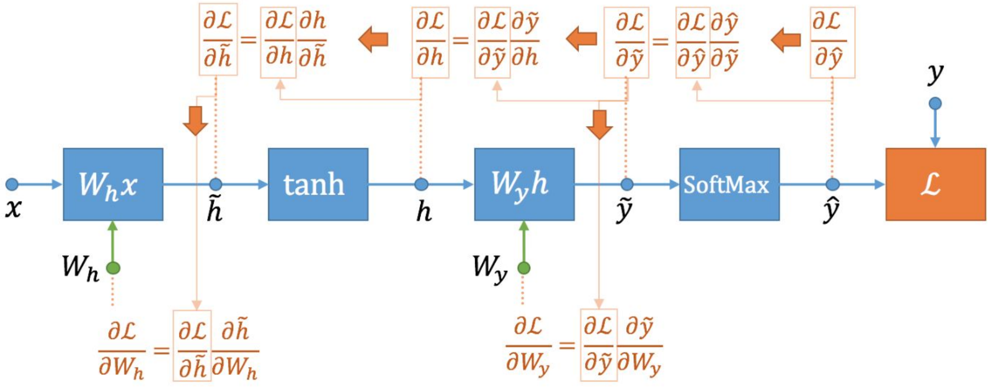
$$\begin{cases} \tilde{h}_i = \sum_{j=1}^{n_x} W^h_{i,j}\ x_j + b^h_i \\ h_i = \tanh(\tilde{h}_i) \\ \tilde{y}_i = \sum_{j=1}^{n_h} W^y_{i,j}\ h_j + b^y_i \\ \hat{y}_i = \mathrm{SoftMax}(\tilde{y}_i) = \dfrac{e^{\tilde{y}_i}}{\sum_{j=1}^{n_y} e^{\tilde{y}_j}} \end{cases}$$

$$\begin{cases} \delta^y_i = \dfrac{\partial \ell}{\partial \tilde{y}_i} = \\ \dfrac{\partial \ell}{\partial W^y_{i,j}} = \\ \dfrac{\partial \ell}{\partial b^y_i} = \end{cases}$$

Example: Tanh MLP

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



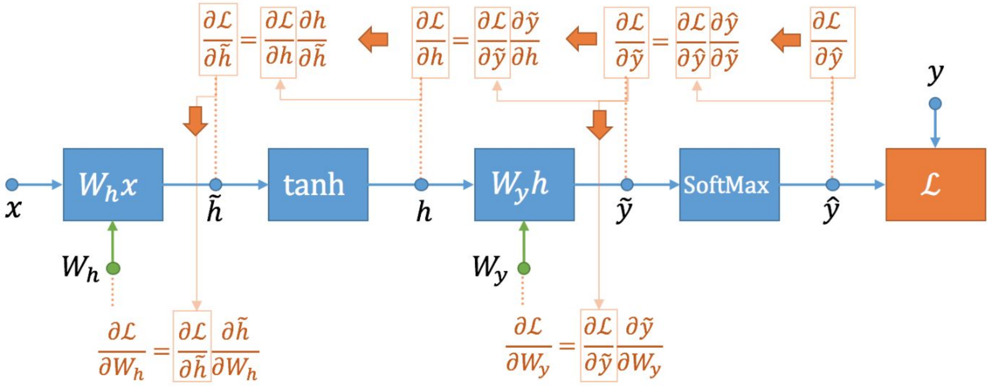


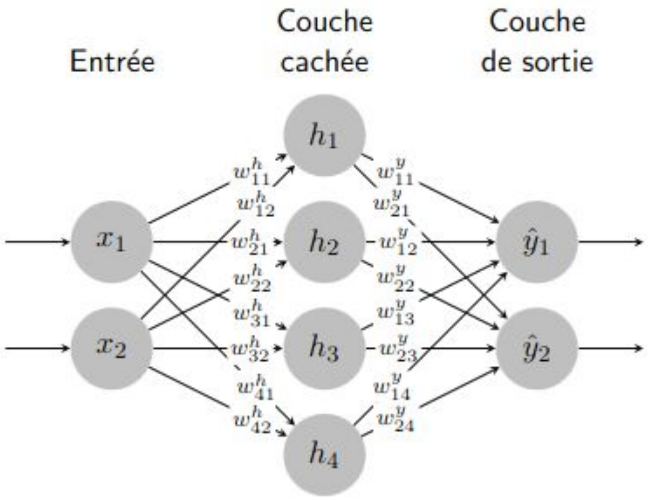
$$\begin{cases} \tilde{h}_i = \sum_{j=1}^{n_x} W^h_{i,j}\ x_j + b^h_i \\ h_i = \tanh(\tilde{h}_i) \\ \tilde{y}_i = \sum_{j=1}^{n_h} W^y_{i,j}\ h_j + b^y_i \\ \hat{y}_i = \text{SoftMax}(\tilde{y}_i) = \dfrac{e^{\tilde{y}_i}}{\sum_{j=1}^{n_y} e^{\tilde{y}_j}} \end{cases}$$

$$\begin{cases} \delta^y_i = \dfrac{\partial \ell}{\partial \tilde{y}_i} = \hat{y}_i - y_i \\ \dfrac{\partial \ell}{\partial W^y_{i,j}} = \delta^y_i\ h_j \\ \dfrac{\partial \ell}{\partial b^y_i} = \delta^y_i \end{cases}$$

Example: Tanh MLP

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



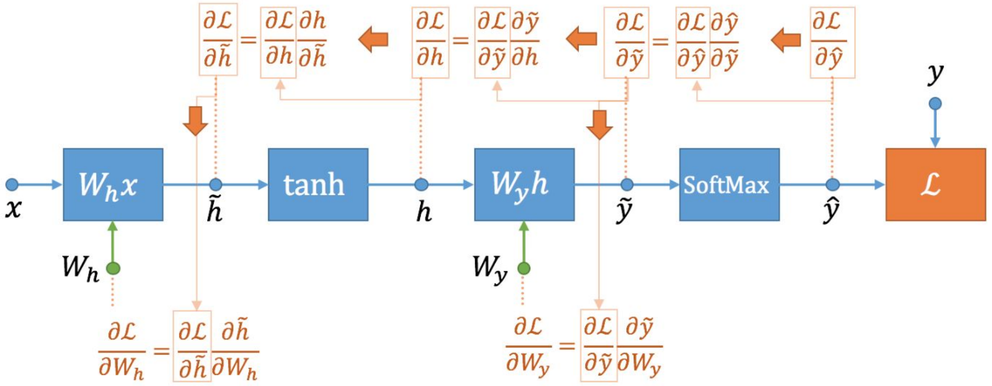


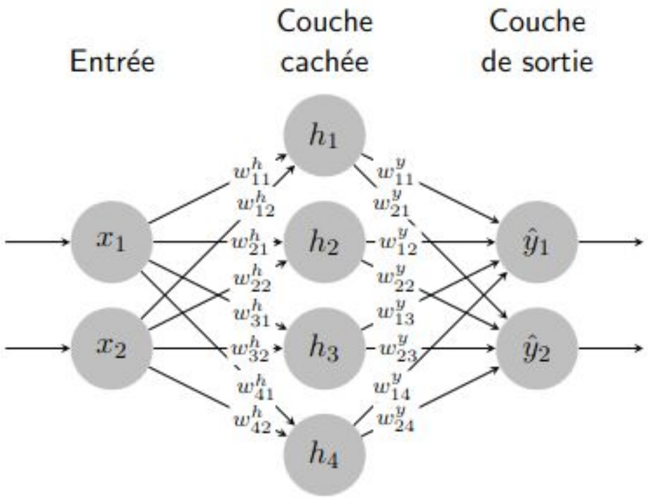
$$\begin{cases} \tilde{h}_i = \sum_{j=1}^{n_x} W^h_{i,j}\ x_j + b^h_i \\ h_i = \tanh(\tilde{h}_i) \\ \tilde{y}_i = \sum_{j=1}^{n_h} W^y_{i,j}\ h_j + b^y_i \\ \hat{y}_i = \mathrm{SoftMax}(\tilde{y}_i) = \dfrac{e^{\tilde{y}_i}}{\sum_{j=1}^{n_y} e^{\tilde{y}_j}} \end{cases}$$

$$\begin{cases} \delta^y_i = \dfrac{\partial \ell}{\partial \tilde{y}_i} = \hat{y}_i - y_i \\ \dfrac{\partial \ell}{\partial W^y_{i,j}} = \delta^y_i\ h_j \\ \dfrac{\partial \ell}{\partial b^y_i} = \delta^y_i \\ \delta^h_i = \dfrac{\partial \ell}{\partial \tilde{h}_i} = \\ \dfrac{\partial \ell}{\partial W^h_{i,j}} = \\ \dfrac{\partial \ell}{\partial b^h_i} = \end{cases}$$

Example: Tanh MLP

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





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

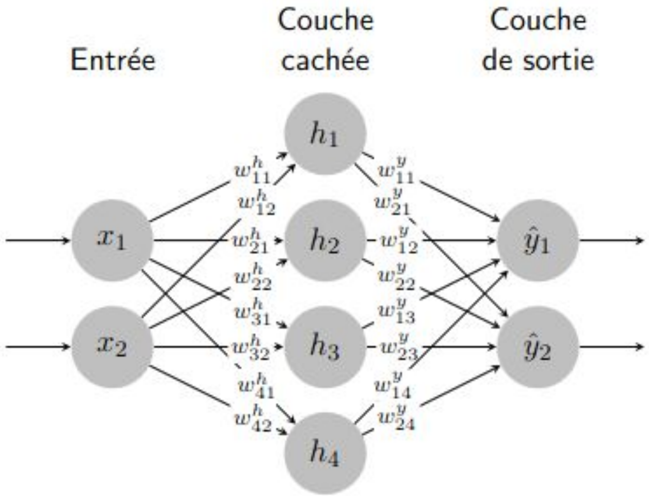
$$\begin{cases} \delta_i^y = \dfrac{\partial \ell}{\partial \tilde{y}_i} = \hat{y}_i - y_i \\ \dfrac{\partial \ell}{\partial W_{i,j}^y} = \delta_i^y \; h_j \\ \dfrac{\partial \ell}{\partial b_i^y} = \delta_i^y \\ \delta_i^h = \dfrac{\partial \ell}{\partial \tilde{h}_i} = (1 - h_i^2) \sum_{j=1}^{n_y} \delta_j^y W_{j,i}^y \\ \dfrac{\partial \ell}{\partial W_{i,j}^h} = \delta_i^h \; x_j \\ \dfrac{\partial \ell}{\partial b_i^h} = \delta_i^h \end{cases}$$

Takeaway

Université Côte d'Azur

- Core problem: Find gradient updates
- Gradient can be computed efficiently with backpropagation
  - Chain rule starting from the “end”
  - Re-use computed gradients (Bottom-up DP)
  - Keep forward activations for gradients
  - Simple layers mean simple gradient blocks

Lab2: Manual Backprop



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

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

- Lab2 on Moodle
  - Implement this by hand with basic torch!
  - Careful with batch dimension!

Lab 2: Autograd backward

UNIVERSITÉ CÔTE D'AZUR

```python
def backward(params, outputs, Y):
    bsize = Y.shape[0]
    grads = {}

    Y_tilde_grad = outputs["yhat"] - Y
    h_tilde_grad = torch.mm(Y_tilde_grad, params['Wy']
                            ) * (1 - torch.pow(outputs['h'], 2))

    grads["Wy"] = torch.mm(Y_tilde_grad.T, outputs["h"])
    grads["Wh"] = torch.mm(h_tilde_grad.T, outputs['X'])
    grads["by"] = Y_tilde_grad.sum(dim=0,keepdim=True).T
    grads["bh"] = h_tilde_grad.sum(0, keepdim=True).T

    grads['Wy'] /= bsize
    grads['by'] /= bsize
    grads['Wh'] /= bsize
    grads['bh'] /= bsize

    return grads
```

- Torch.tensor object
  - Np.array like
  - Tracked on a computational graph
  - .grad variable to track gradients
  - .backward to backpropagate gradients through the graph
  - Activate .autograd!

Lab 2: Autograd backward

UNIVERSITÉ CÔTE D'AZUR

```python
def backward(params, outputs, Y):
    bsize = Y.shape[0]
    grads = {}

    Y_tilde_grad = outputs["yhat"] - Y
    h_tilde_grad = torch.mm(Y_tilde_grad, params['Wy']
                            ) * (1 - torch.pow(outputs['h'], 2))

    grads["Wy"] = torch.mm(Y_tilde_grad.T, outputs["h"])
    grads["Wh"] = torch.mm(h_tilde_grad.T, outputs['X'])
    grads["by"] = Y_tilde_grad.sum(dim=0,keepdim=True).T
    grads["bh"] = h_tilde_grad.sum(0, keepdim=True).T

    grads['Wy'] /= bsize
    grads['by'] /= bsize
    grads['Wh'] /= bsize
    grads['bh'] /= bsize

    return grads
```

```python
params['Wh'] = torch.randn(nh, nx) * 0.3
params['Wh'].requires_grad = True
params['bh'] = torch.zeros(nh, 1, requires_grad=True)
params['Wy'] = torch.randn(ny, nh) * 0.3
params['Wy'].requires_grad = True
params['by'] = torch.zeros(ny, 1, requires_grad=True)
```

```python
with torch.no_grad():
    params['Wy'] -= eta * params['Wy'].grad
    params['Wh'] -= eta * params['Wh'].grad
    params['by'] -= eta * params['by'].grad
    params['bh'] -= eta * params['bh'].grad

    params['Wy'].grad.zero_()
    params['Wh'].grad.zero_()
    params['by'].grad.zero_()
    params['bh'].grad.zero_()
```