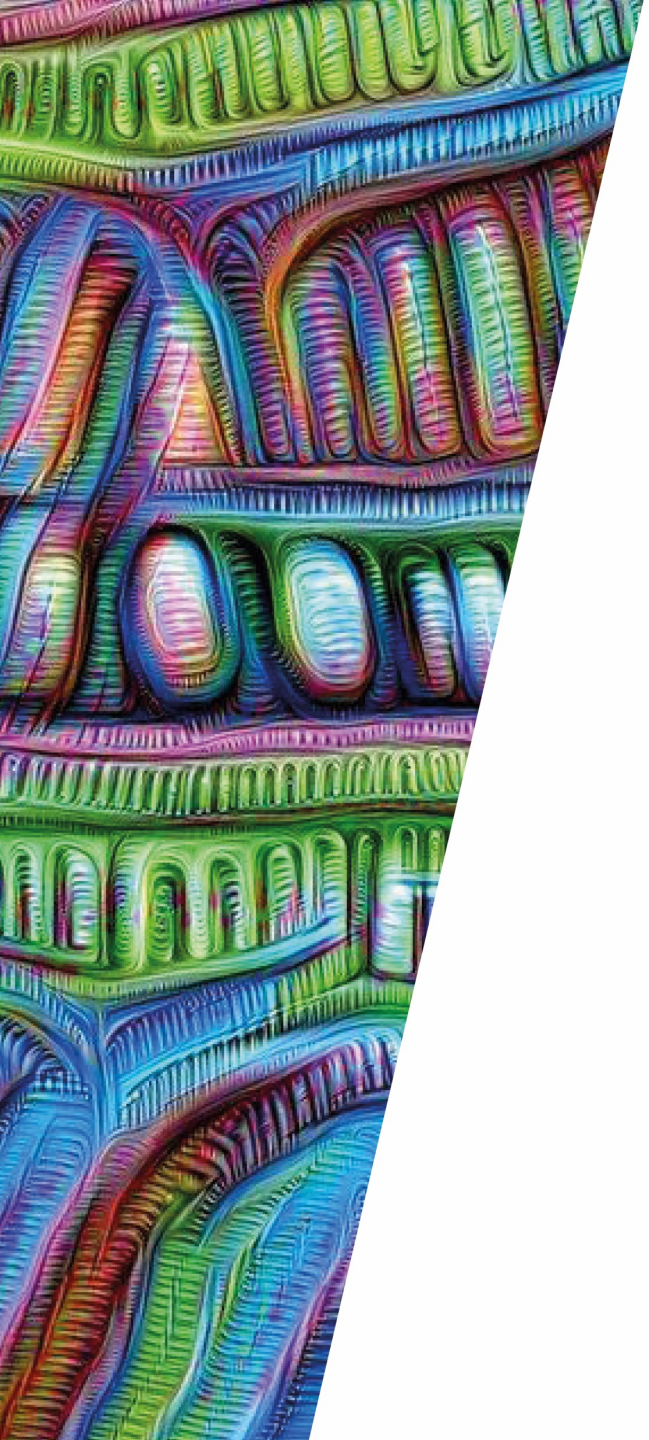


*Bienvenue!*

# ÉCOLE D'ÉTÉ FRANCOPHONE EN APPRENTISSAGE PROFOND

21-25 août 2017

IVADO

HEC Montréal
Polytechnique Montréal
Université de Montréal

MILA

# Graphes de calcul et rétro-propagation du gradient

**Yoshua Bengio**

22 août 2017

École d'Été en Apprentissage Profond

# Recap: Machine Learning 101

- Family of functions $f_\theta$
- Tunable parameters $\theta$
- Examples *(x,y)* sampled fromunknown data generating distribution *P(x,y)*
- Loss fn *L* compares target y and output $f_\theta(x)$, returns a number
- Regularizer *R* (typically depends on $\theta$ but possibly also on *x* & *y*)
- Training criterion for supervised learning:

$$C(\theta) = \text{average}_{(x,y)\sim\text{dataset}} L(f_\theta(x), y) + R(\theta, x, y)$$

- Approximate minimization algorithm to search for good $\theta$

# Motivations pour la rétropropagation (backpropagation, backprop)

- Si on peut calculer comment un petit changement des paramètres pourrait réduire l'erreur, on pourrait faire ce petit changement, et améliorer graduellement l'erreur totale

- Le gradient $\frac{\partial L}{\partial \theta}$ mesure le ratio d'un changement de l'erreur *L* associé à un changement des paramètres $\theta$

- Il indique aussi la direction dans l'espace des paramètres qui donnerait lieu à la plus grande amélioration de *L*, si on faisait un changement infinitésimal des paramètres dans cette direction

# Pourquoi le gradient est puissant

- Avec *n* paramètres ça prend en général O(*n*) calculs pour calculer *L* et aussi O(*n*) pour le gradient par rétropropagation

- Si on ne pouvait pas calculer le gradient (e.g. par rétropropagation), on pourrait l'estimer par différence finie

$$\frac{\partial L(\theta_i, \theta_{-i})}{\partial \theta_i} \approx \frac{L(\theta_i + \epsilon, \theta_{-i}) - L(\theta_i, \theta_{-i})}{\epsilon}$$

- Mais si on a *n* paramètres, ça prendrait un nombre de calculs quadratique en *n*, *n* fois le calcul de *L* (dont le calcul est linéaire en *n*)

# Confusion sur le sens du mot BACKPROP

- Backprop: la procédure pour calculer le gradient efficacement dans un réseau profond, en partant des noeuds de sortie et en traversant les couches dans le sens inverse du calcul normal des activations.

- Ce n'est pas la même chose que la **descente de gradient**, qu'on applique une fois qu'on a calculé le gradient. Il y a beaucoup de variantes d'algorithmes d'optimisation qui utilisent le gradient: la rétroprogation (backprop) est donc une sous-routine pour ces algorithmes d'optimisation.

# Au cœur de l'apprentissage profond: la rétropropagation (*backprop*)

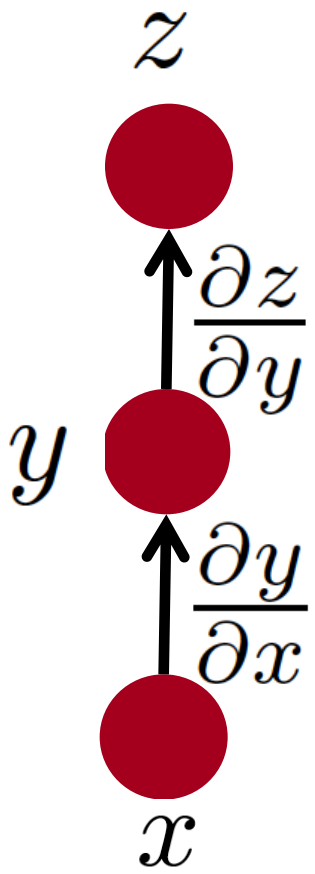
# Back-Prop & Chain Rule

- Compute gradient of example-wise loss wrt parameters

- **Simply applying the derivative chain rule wisely**

$$z = f(y) \quad y = g(x) \quad \frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

# Dérivée en chaîne



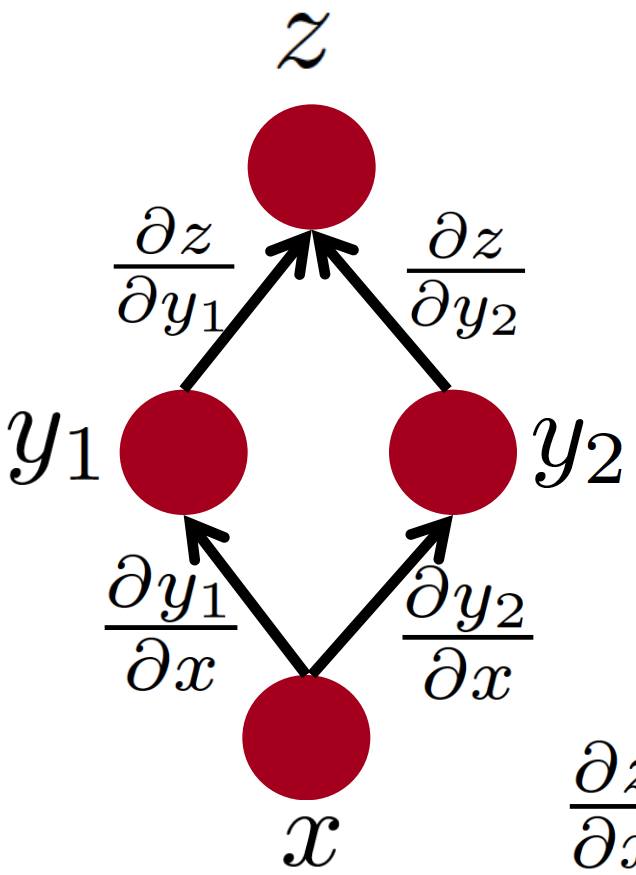

$$\Delta z = \frac{\partial z}{\partial y}\Delta y$$

$$\Delta y = \frac{\partial y}{\partial x}\Delta x$$

$$\Delta z = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\Delta x$$

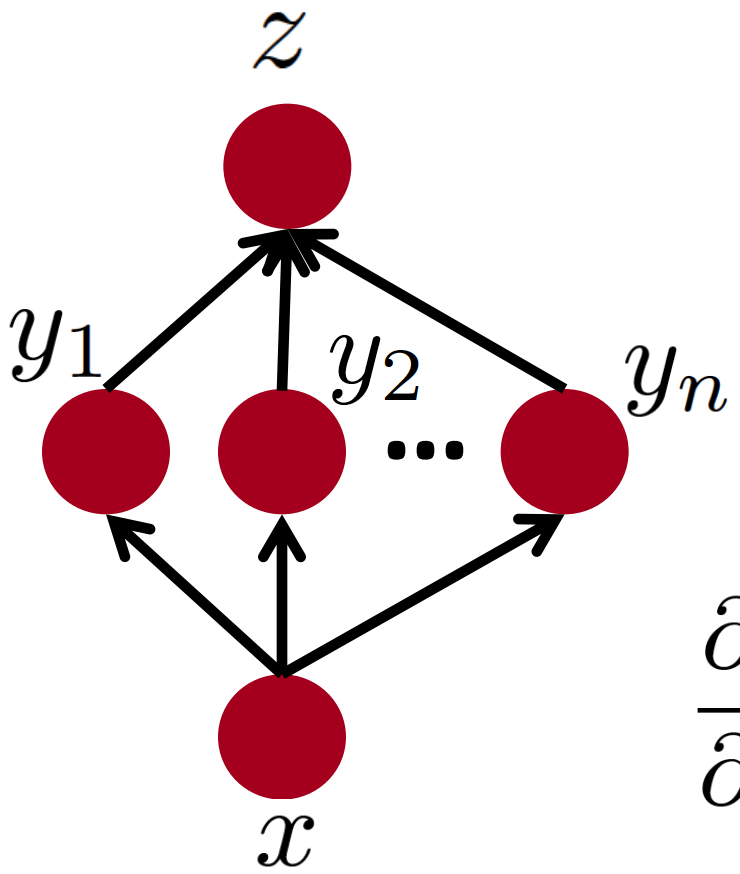
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

# Plusieurs chemins entre deux noeuds




$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_1}\frac{\partial y_1}{\partial x} + \frac{\partial z}{\partial y_2}\frac{\partial y_2}{\partial x}$$

# Plusieurs chemins - cas général

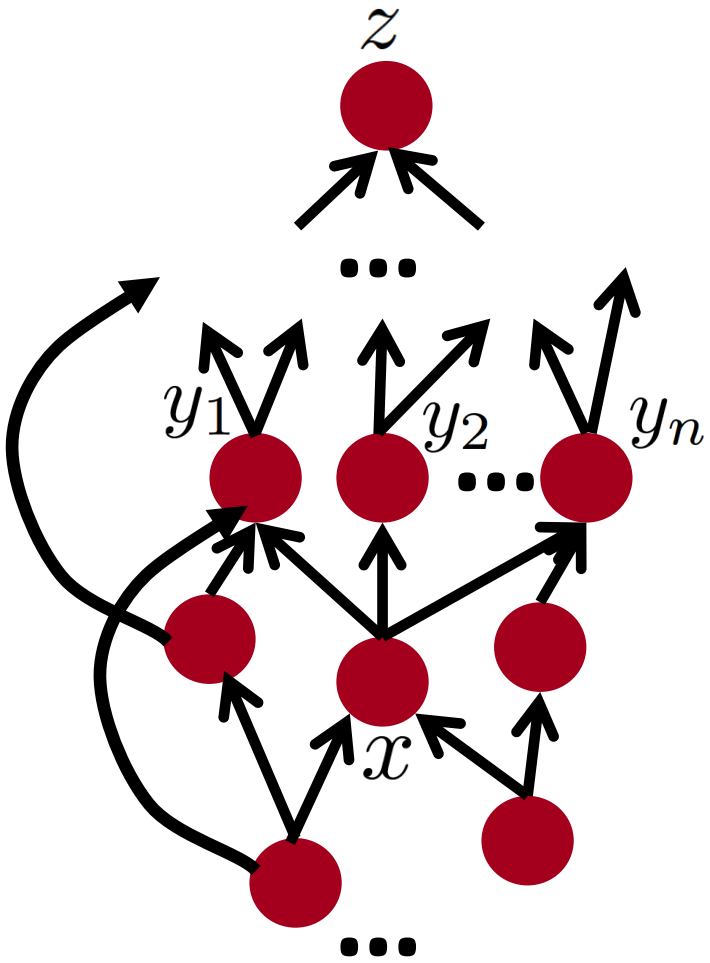



On additionne les contributions provenant de chaque chemin entre *x* et *z*

Sur chaque chemin on multiplie les dérivées partielles

$$\frac{\partial z}{\partial x} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

# Rétropropagation dans un graphe de flots

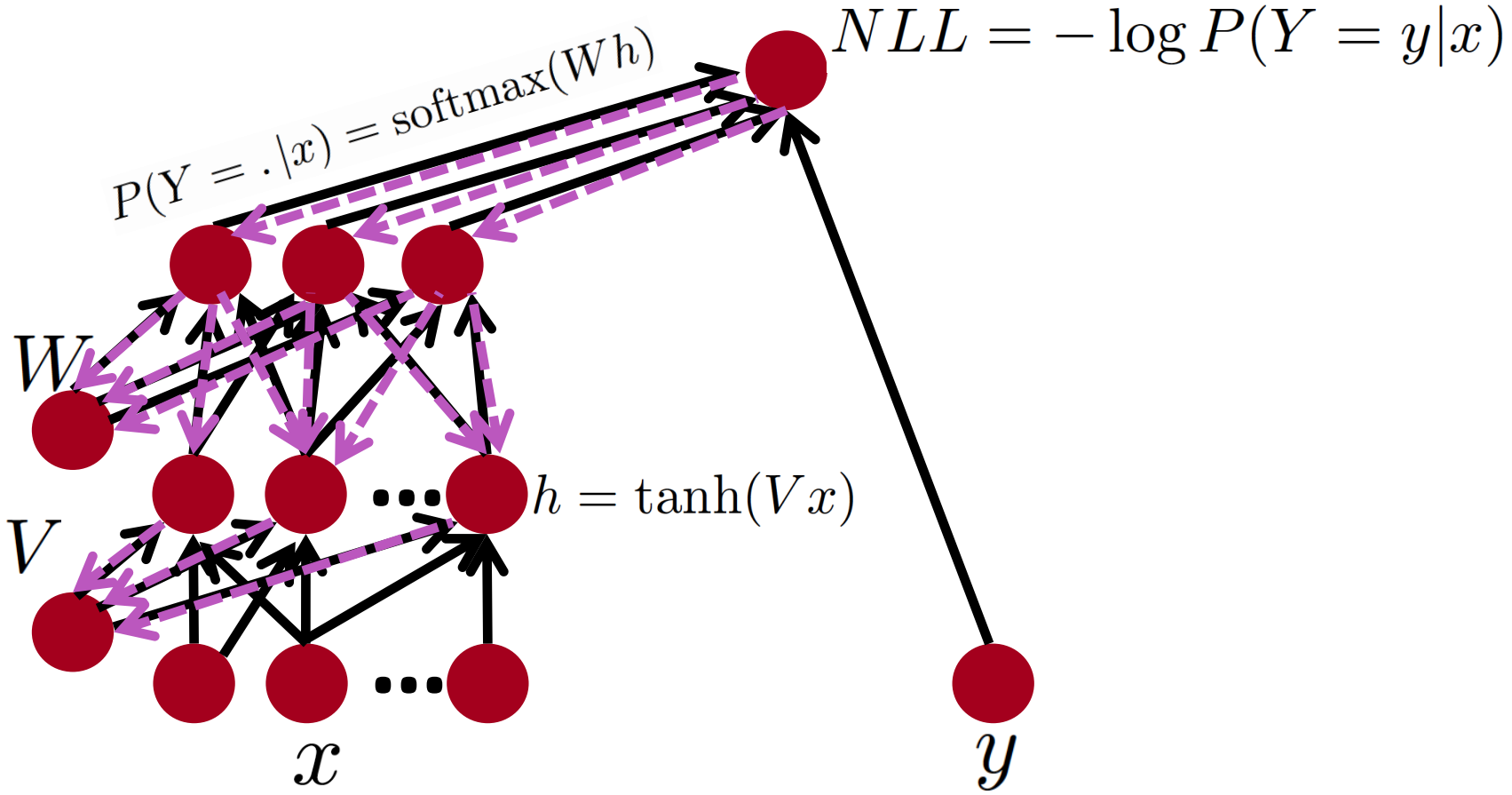


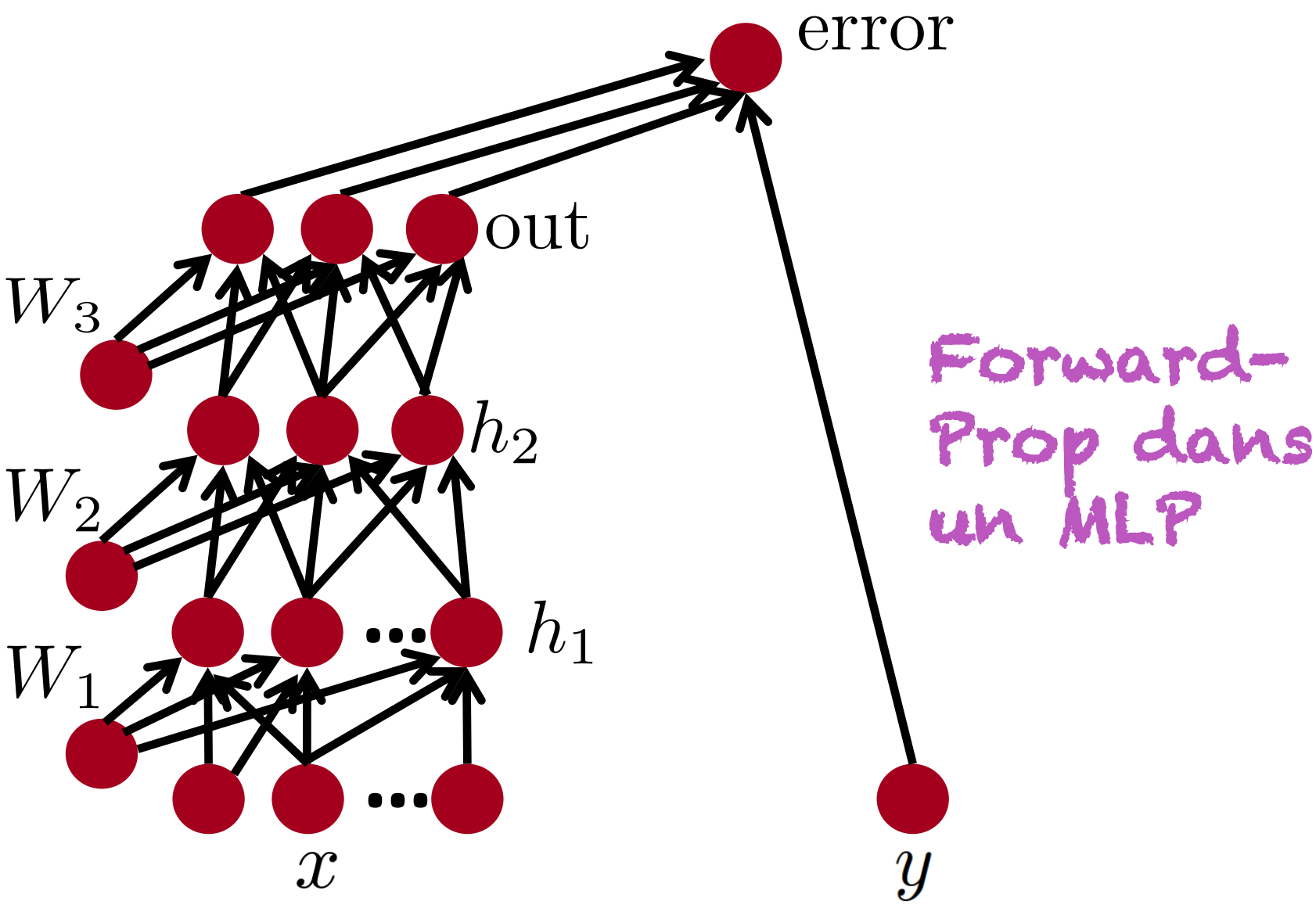

Flow graph: any directed acyclic graph
node = computation result
arc = computation dependency

$\{y_1, y_2, \ldots y_n\}$ = successors of $x$

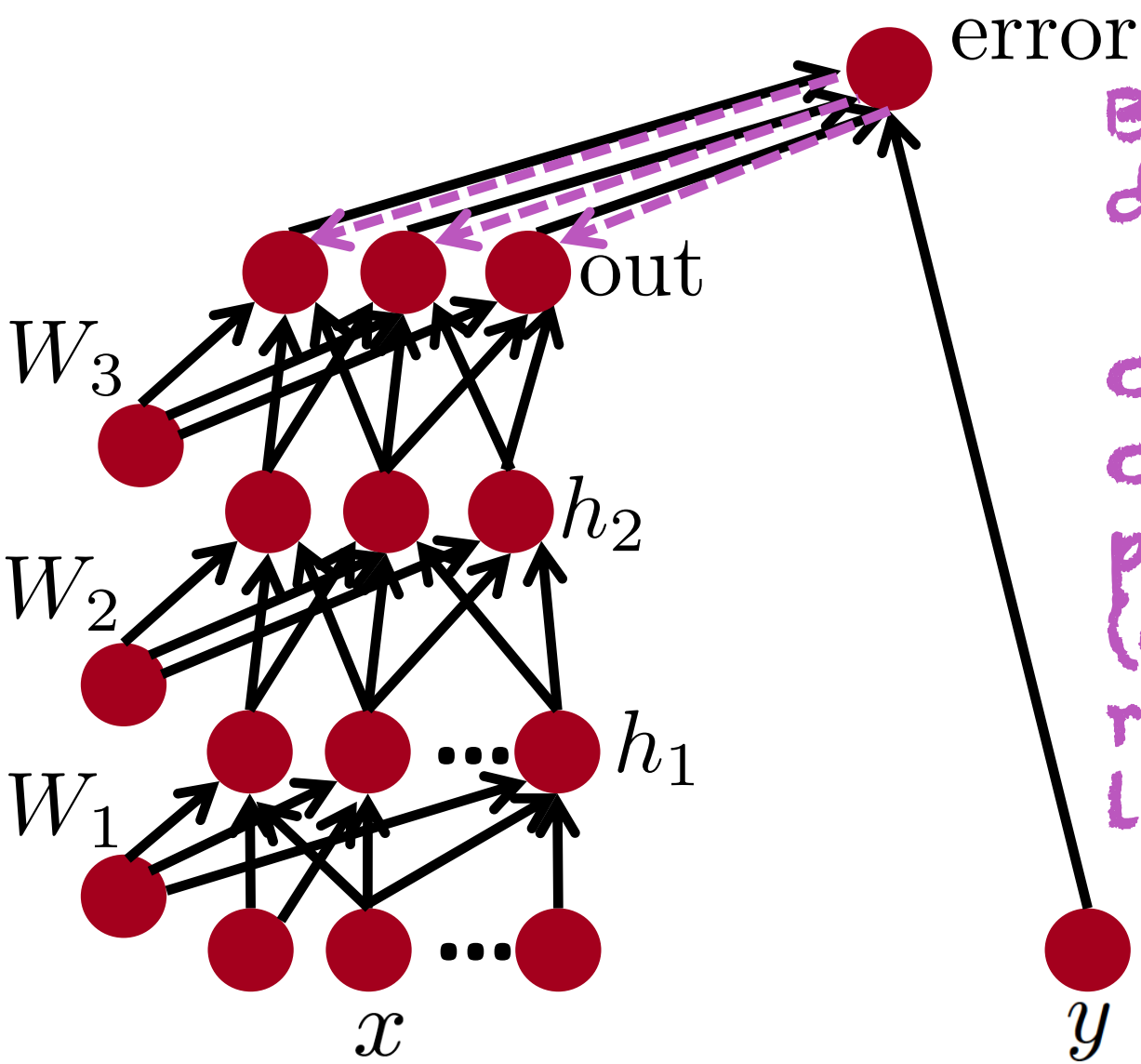
$$\frac{\partial z}{\partial x} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

# Rétropropagation dans un MLP



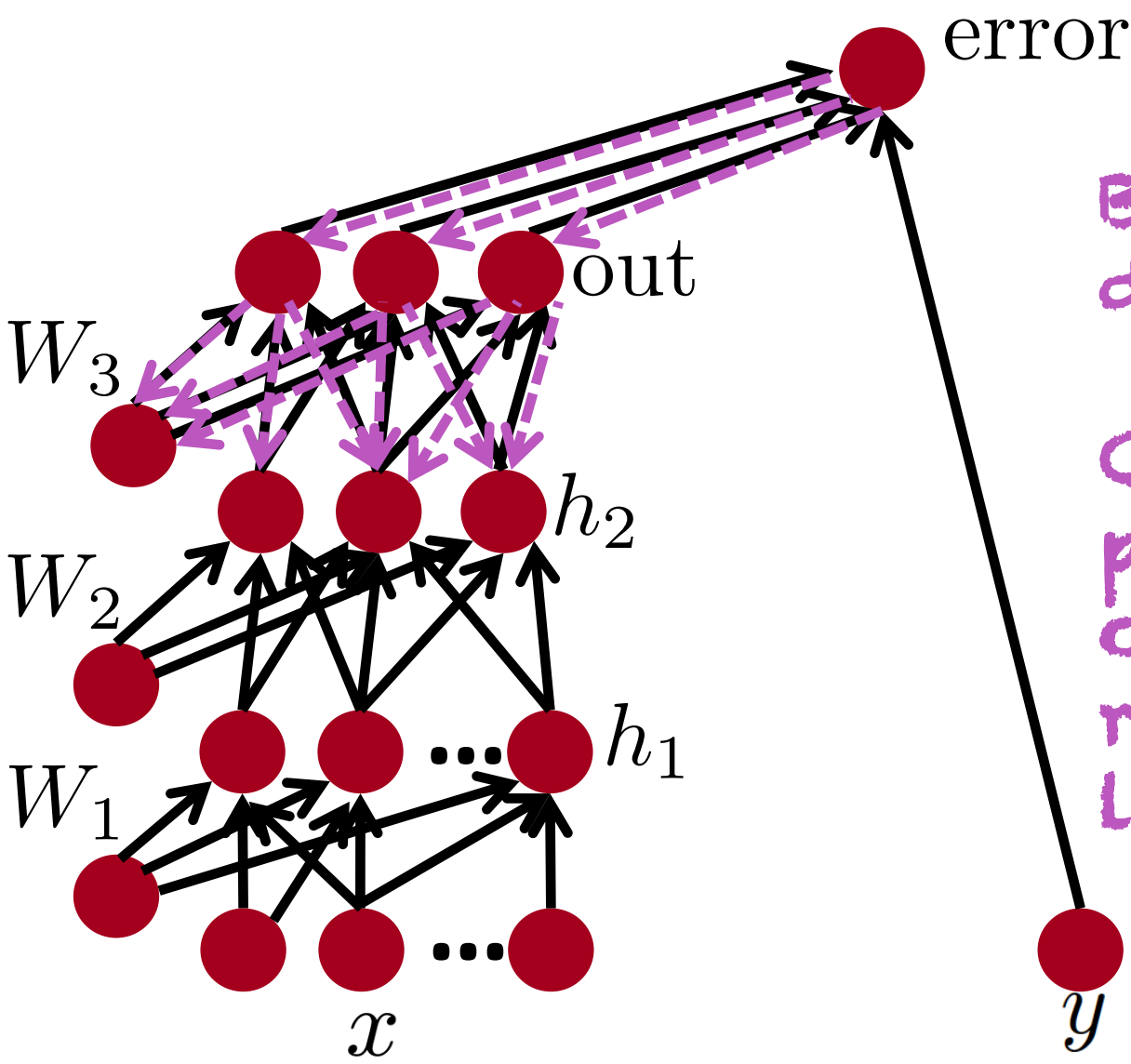



Forward-Prop dans un MLP



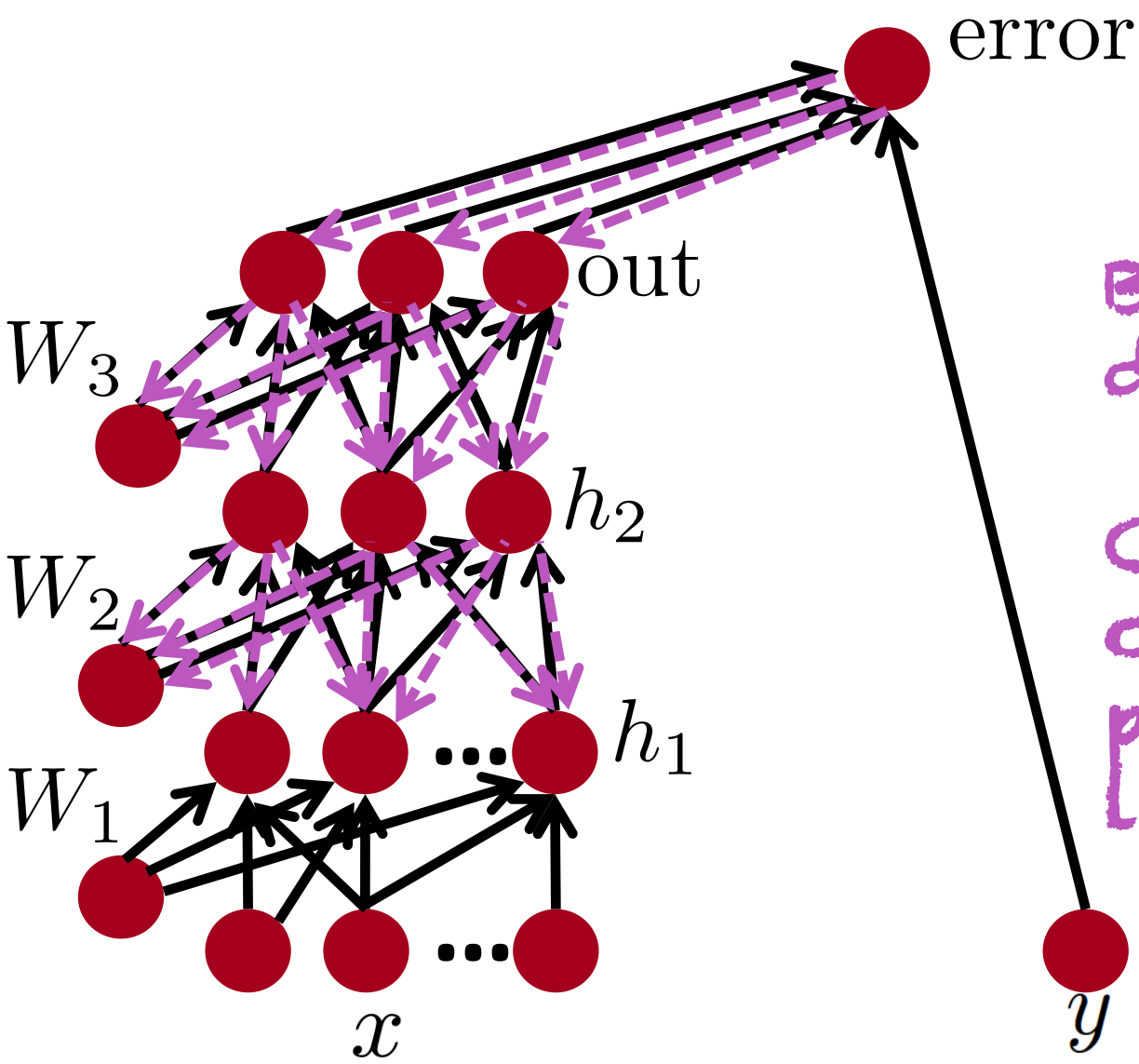

Backprop dans un MLP:

comment changer un peu la sortie (out) pour réduire l'erreur?



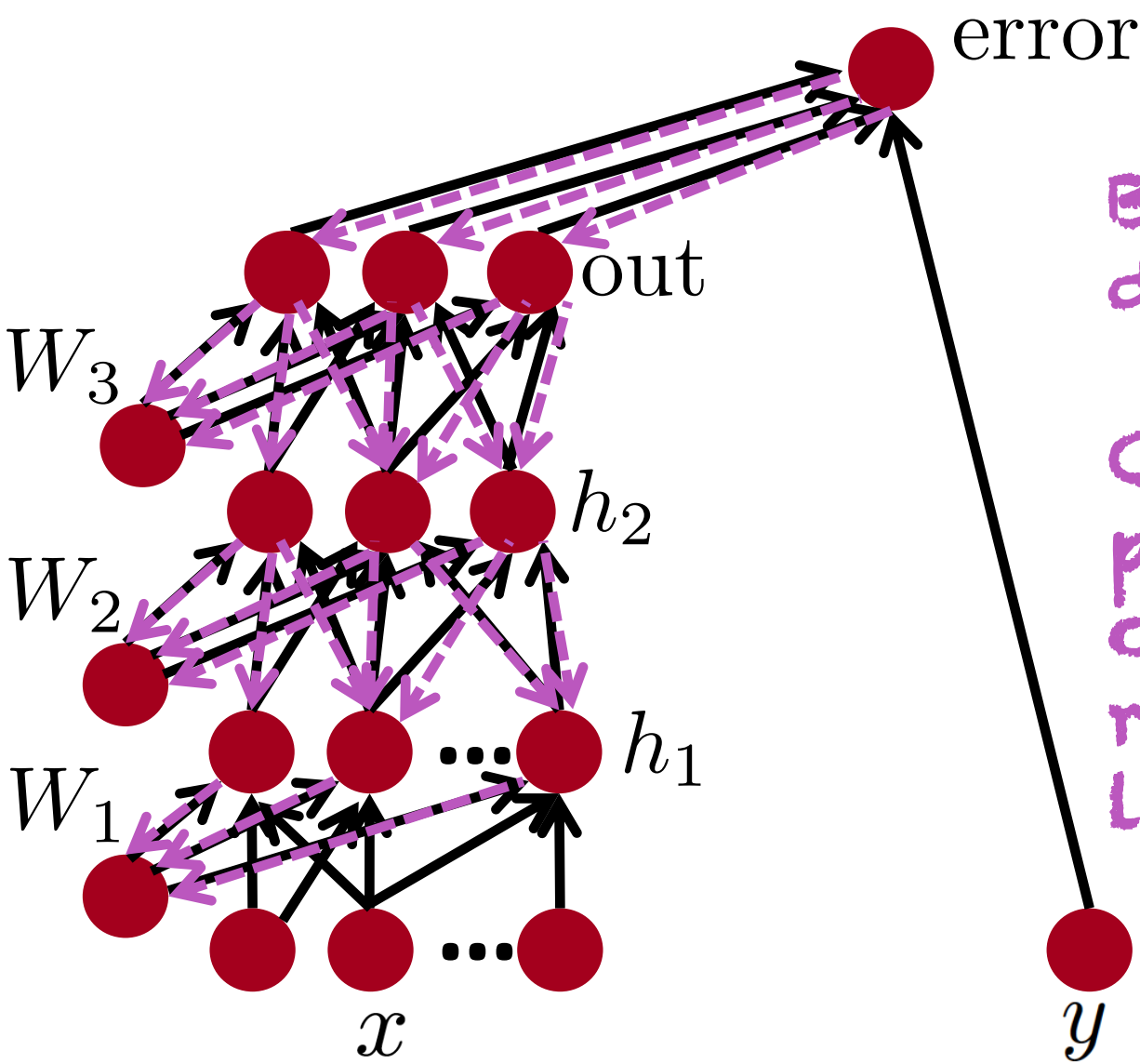

Backprop dans un MLP:

Comment $h_2$ pourrait changer pour réduire l'erreur




Backprop dans un MLP:

Comment changer $h_1$ pour réduire l'erreur

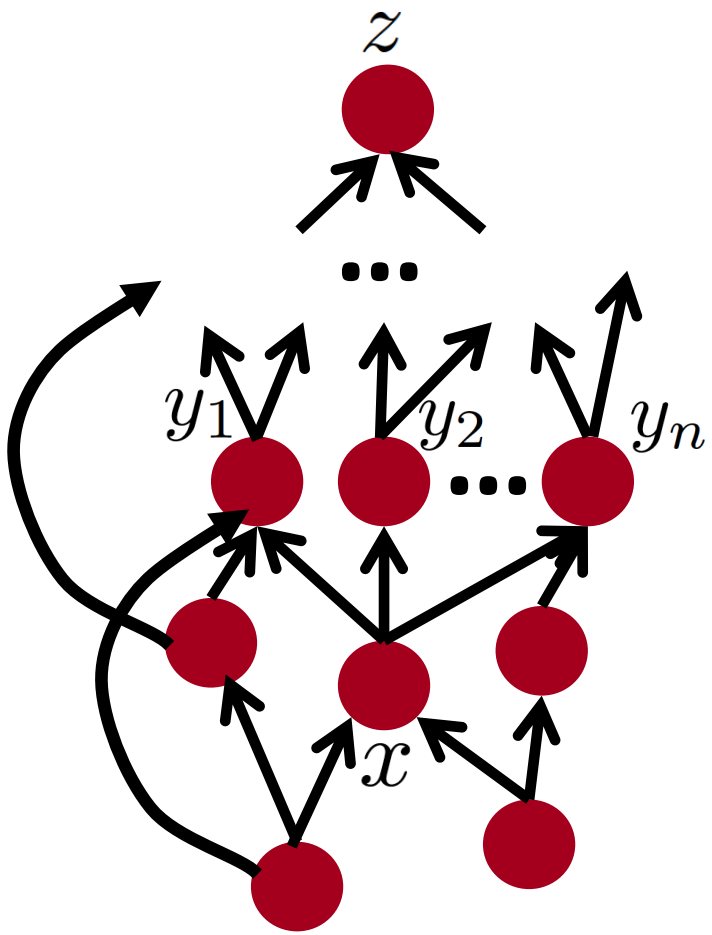



Backprop dans un MLP:

Comment $W_1$ pourrait changer pour réduire l'erreur

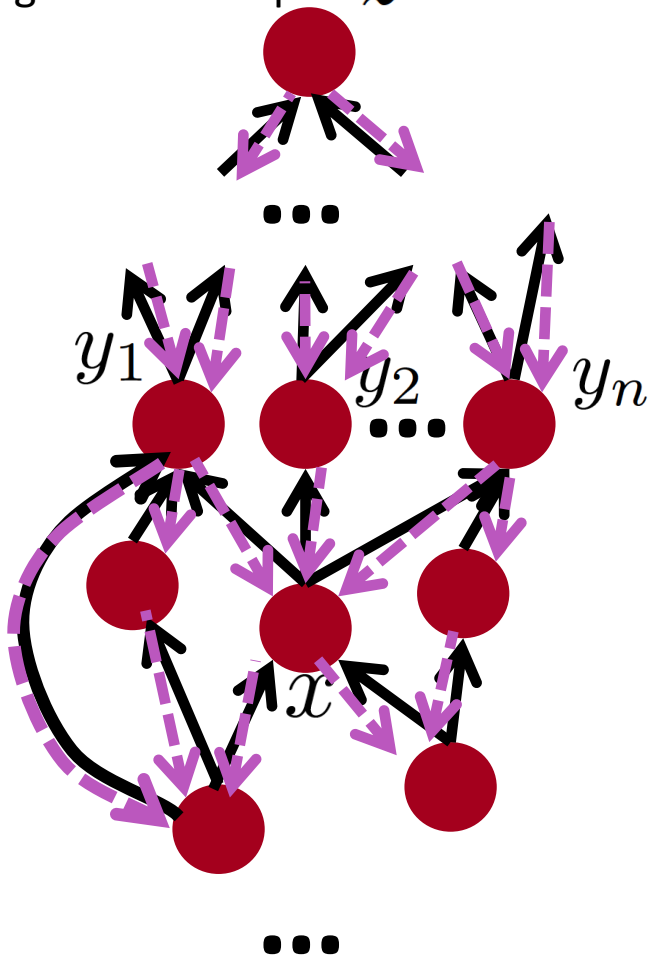
# Backprop: algorithme récursif

- Si on connait le gradient de l'erreur par rapport aux noeuds du graphe de calcul qui sont les successeurs d'un noeud, on peut calculer le gradient de l'erreur par rapport à ce noeud
  - Calcul récursif
  - Une forme de programmation dynamique (pour éviter de refaire le même calcul plusieurs fois)
- L'algorithme n'est pas spécifique aux réseaux multi-couches: s'applique aussi pour n'importe quel graphe de calcul, ce qui permet d'écrire du code de **différentiation automatique**



# Cas Général: Back-Prop dans un Graphe de Flot

Single scalar output $z$



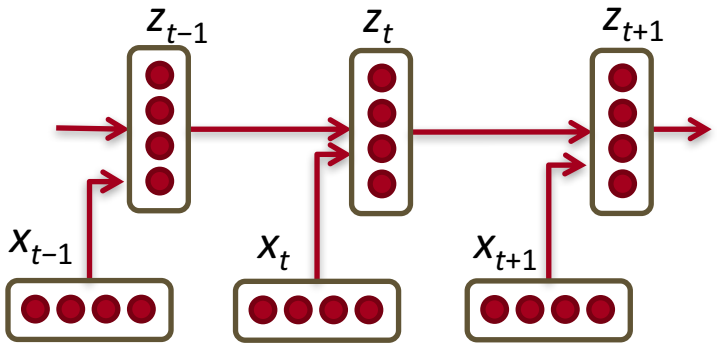

1. Fprop: visit nodes in topo-sort order
   - Compute value of node given predecessors
2. Bprop:
   - initialize output gradient = 1
   - visit nodes in reverse order:
     Compute gradient wrt each node using gradient wrt successors

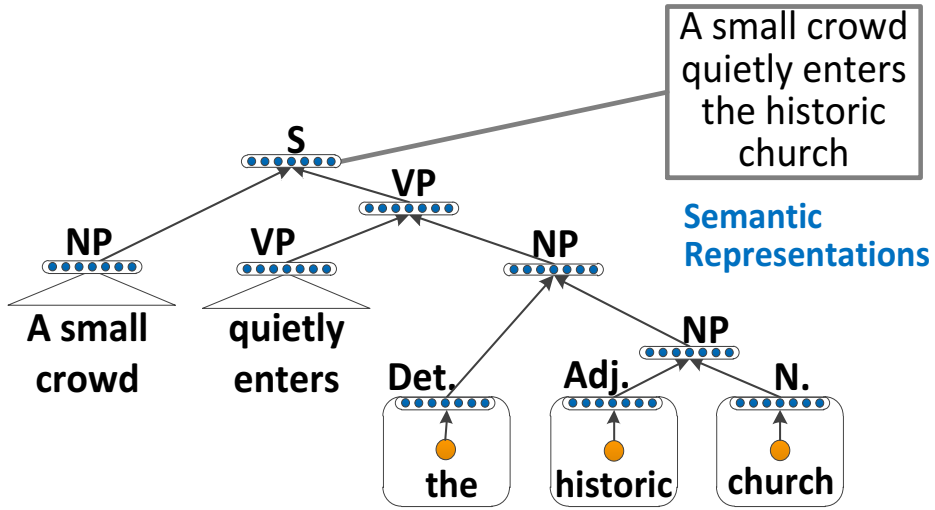
$\{y_1, y_2, \ldots y_n\}$ = successors of $x$

$$\frac{\partial z}{\partial x} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

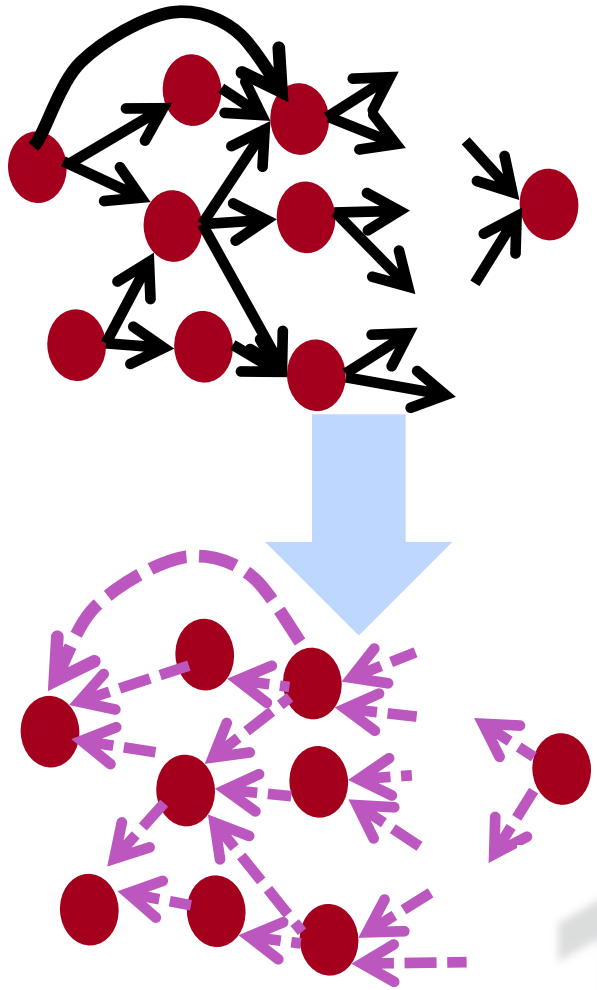
# Back-Prop in Recurrent & Recursive Nets

- Replicate a parameterized function over different time steps or nodes of a DAG
- Output state at one time-step / node is used as input for another time-step / node



# Automatic Differentiation



- The gradient computation can be automatically inferred from the symbolic expression of the fprop.
- Each node type needs to know how to compute its output and how to compute the gradient wrt its inputs given the gradient wrt its output
- Easy and fast prototyping

theano TensorFlow PYTORCH