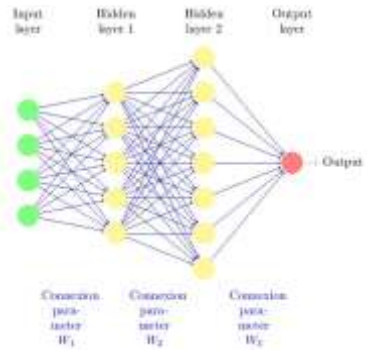
Gradient calculation: example of backpropagation

Idea: NN= **composition of functions**, use **chain rule (**f(g(x)))' = f'(g(x)) g'(x), multivariable form : D(f∘g)= Df ∘ Dg

Information flow : backwards: need Df to have D(f∘g)



$Y_0 \to \tilde{Y}_1 = W_1 Y_0 + b_1 \to Y_1 = A_1(\tilde{Y}_1)(RELU)$, …, softmax, loss

Backpropagation: $\delta L = Id, \delta Y_3 = \frac{\partial L}{\partial Y_3}$, $\partial \tilde{Y}_3 = \frac{\partial L}{\partial \tilde{Y}_3} = \frac{\partial L}{\partial Y_3}\frac{\partial Y_3}{\partial \tilde{Y}_3}$= …

Goal: obtain the gradients $\delta W_k = \frac{\partial L}{\partial W_k}$ et $\delta b_k = \frac{\partial L}{\partial b_k}$ for k=1,2,3.

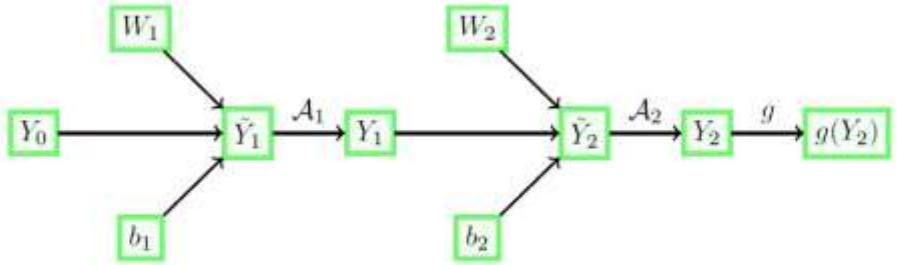


FIGURE 3.5 – Graphe de propagation forward du réseau de neurones

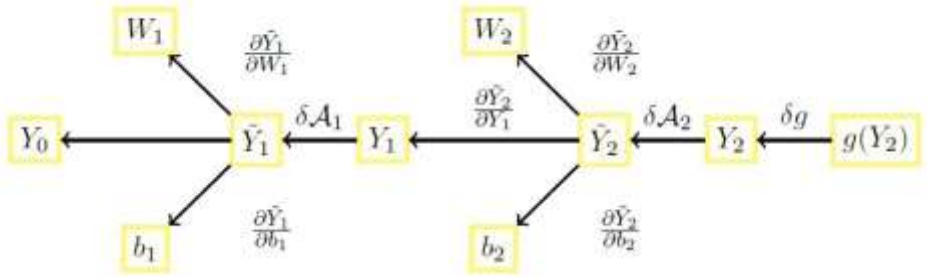
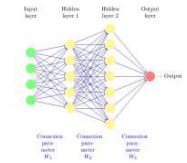


FIGURE 3.6 – Graphe de propagation backward du réseau de neurones

Backprop: formules (cf. documents for proofs)



Notation $\partial(Tom) = Derivative\ of\ the\ final\ loss\ with\ respect\ to\ "Tom"$

Gradient of softmax+cross-entropy

opération: Y -> S= np.exp(Y)/np.sum(np.exp(Y)) (softmax)

S -> Loss(Y)=- np.sum(P*np.log(S)) (P= true labels)

$Ex: prove\ the\ formula:\ \partial Y = S - P\ (denoted\ \tilde{Y}\ in\ the\ code)$

Gradient of ReLu operation $\widetilde{Y_k} \mapsto Y_k = ReLu(\widetilde{Y_k}), formula: \delta\tilde{Y}_k = \delta Y_k \cdot 1_{\tilde{Y}_k \geq 0}$

Gradient of the linear part $operation$: $\widetilde{Y}_{k+1} = W_{k+1}Y_k + b_{k+1}$

$formula: \delta Y_k = W_{k+1}^T \cdot \delta\tilde{Y}_{k+1}, \delta W_{k+1} = \delta\tilde{Y}_{k+1} \cdot Y_k^T, \delta b_{k+1} = \delta\tilde{Y}_{k+1}$ (note: proof uses Kronecker product)