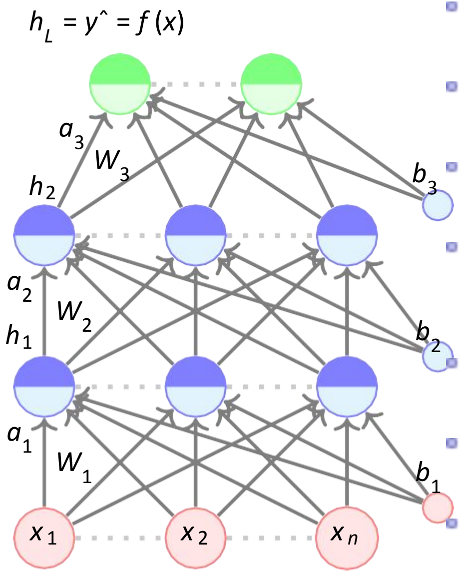
CS7015 (Deep Learning): Lecture 4

Feedforward Neural Networks, Backpropagation

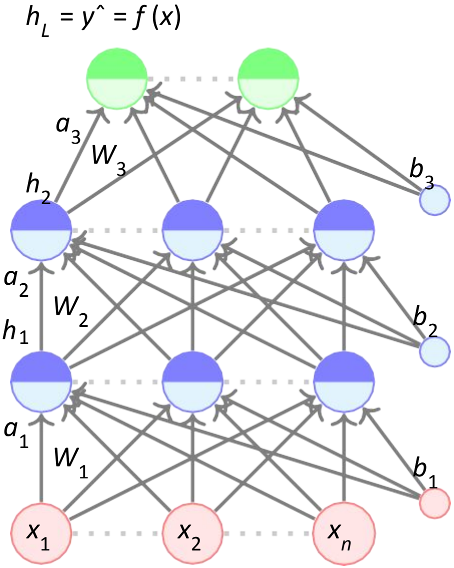
Mitesh M. Khapra

Department of Computer Science and Engineering Indian Institute of Technology Madras

Module 4.1: Feedforward Neural Networks (a.k.a. multilayered network of neurons)



- The input to the network is an **n**-dimensional vector
- The network contains **L − 1** hidden layers (2, in this case) having **n** neurons each
- Finally, there is one output layer containing **k** neurons (say, corresponding to **k** classes)
- Each neuron in the hidden layer and output layer can be split into two parts : pre-activation and activation (a_i and h_i are vectors)
- The input layer can be called the 0-th layer and the output layer can be called the (L)-th layer
- $W_i \in \mathbb{R}^{n \times n}$ and $b_i \in \mathbb{R}^n$ are the weight and bias between layers $i - 1$ and i ($0 < i < L$)
- $W_L \in \mathbb{R}^{n \times k}$ and $b_L \in \mathbb{R}^k$ are the weight and bias between the last hidden layer and the output layer ($L = 3$ in this case)



- The pre-activation at layer i is given by

$$a_i(x) = b_i + W_i h_{i-1}(x)$$

- The activation at layer i is given by

$$h_i(x) = g(a_i(x))$$

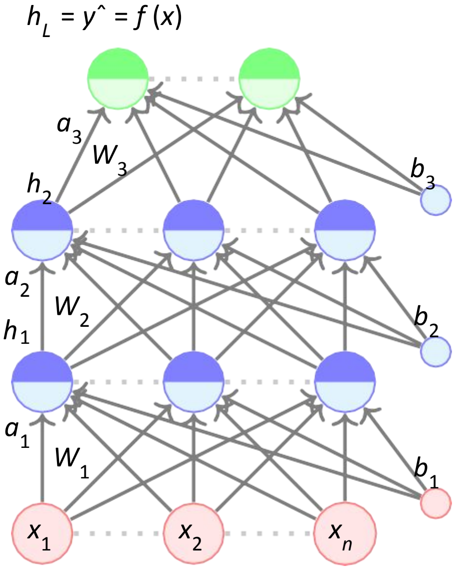
where g is called the activation function (for example, logistic, tanh, linear, *etc.*)

- The activation at the output layer is given by

$$f(x) = h_L(x) = O(a_L(x))$$

where O is the output activation function (for example, softmax, linear, *etc.*)

- To simplify notation we will refer to $a_i(x)$ as a_i and $h_i(x)$ as h_i



- The pre-activation at layer i is given by

$$a_i = b_i + W_i h_{i-1}$$

- The activation at layer i is given by

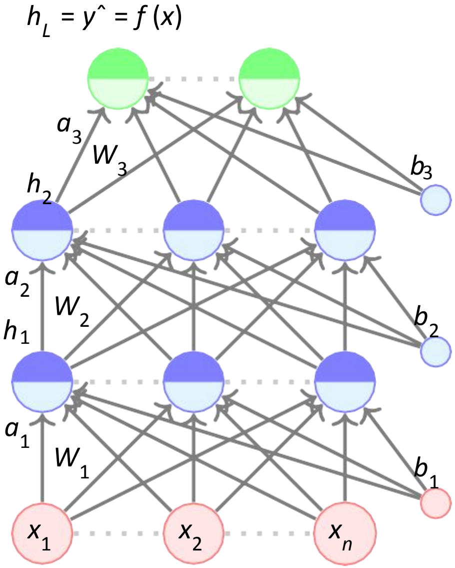
$$h_i = g(a_i)$$

where g is called the activation function (for example, logistic, tanh, linear, *etc.*)

- The activation at the output layer is given by

$$f(x) = h_L = O(a_L)$$

where O is the output activation function (for example, softmax, linear, *etc.*)



- **Data:** $\{x_i, y_i\}_{i=1}^{N}$
- **Model:**

$$\hat{y}_i = f(x_i) = O(W_3 g(W_2 g(W_1 x + b_1) + b_2) + b_3)$$

- **Parameters:**

$$\vartheta = W_1, .., W_L, b_1, b_2, ..., b_L (L = 3)$$

- **Algorithm:** Gradient Descent with Back-propagation (we will see soon)
- **Objective/Loss/Error function:** Say,

$$\min \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{k} (\hat{y}_{ij} - y_{ij})^2$$

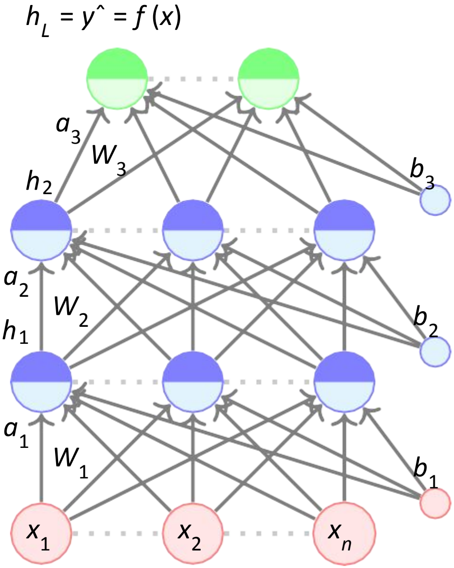
In general, $\min \mathcal{L}(\vartheta)$

where $\mathcal{L}(\vartheta)$ is some function of the parameters

Module 4.2: Learning Parameters of Feedforward Neural Networks (Intuition)

The story so far...

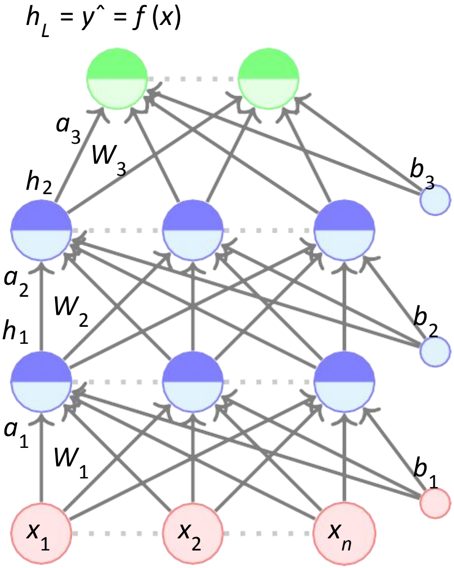
- We have introduced feedforward neural networks
- We are now interested in finding an algorithm for learning the parameters of this model



- Recall our gradient descent algorithm

Algorithm: gradient_descent()

$t \leftarrow 0$;
$max\_iterations \leftarrow 1000$;
Initialize w_0, b_0;
while $t{+}{+} < max\_iterations$ **do**
  $w_{t+1} \leftarrow w_t - \eta \nabla w_t$;
  $b_{t+1} \leftarrow b_t - \eta \nabla b_t$;
end



- Recall our gradient descent algorithm
- We can write it more concisely as

Algorithm: gradient_descent()

$t \leftarrow 0;$

$max\_iterations \leftarrow 1000;$

Initialize $\vartheta_0 = [w_0, b_0];$

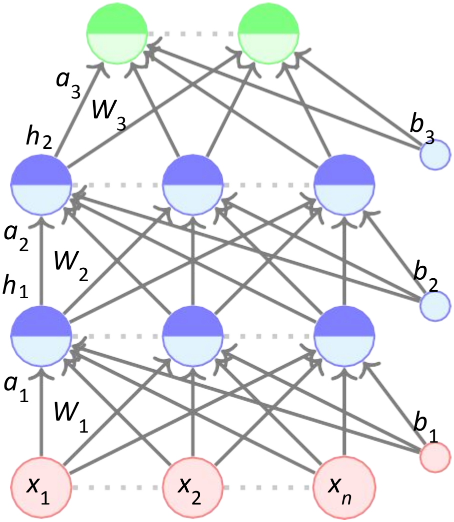
while $t{+}{+} < max\_iterations$ **do**

$\quad \vartheta_{t+1} \leftarrow \vartheta_t - \eta \nabla \vartheta_t;$

end

- where $\nabla \vartheta_t = \left[\frac{\partial \mathscr{L}(\vartheta)}{\partial w_t}, \frac{\partial \mathscr{L}(\vartheta)}{\partial b_t}\right]^T$
- Now, in this feedforward neural network, instead of $\vartheta = [w, b]$ we have $\vartheta = [W_1, W_2, .., W_L, b_1, b_2, .., b_L]$
- We can still use the same algorithm for learning the parameters of our model

$h_L = \hat{y} = f(x)$



- Recall our gradient descent algorithm
- We can write it more concisely as

Algorithm: gradient_descent()

$t \leftarrow 0$;
$max\_iterations \leftarrow 1000$;
Initialize $\theta_0 = [W_1^0, ..., W_L^0, b_1^0, ..., b_L^0]$;
while $t{+}{+} < max\_iterations$ **do**
  $\theta_{t+1} \leftarrow \theta_t - \eta \nabla \theta_t$;
end

- where $\nabla \theta_t = \left[\frac{\partial L(\theta)}{\partial W_{1,t}}, ..., \frac{\partial L(\theta)}{\partial W_{L,t}}, \frac{\partial L(\theta)}{\partial b_{1,t}}, ..., \frac{\partial L(\theta)}{\partial b_{L,t}} \right]^T$
- Now, in this feedforward neural network, instead of $\theta = [w, b]$ we have $\theta = [W_1, W_2, .., W_L, b_1, b_2, .., b_L]$
- We can still use the same algorithm for learning the parameters of our model

- Except that now our $\nabla\vartheta$ looks much more nasty

$$
\begin{bmatrix}
\frac{\partial \mathscr{L}(\vartheta)}{\partial W_{111}} & \dots & \frac{\partial \mathscr{L}(\vartheta)}{\partial W_{11n}} & \frac{\partial \mathscr{L}(\vartheta)}{\partial W_{211}} & \dots & \frac{\partial \mathscr{L}(\vartheta)}{\partial W_{21n}} & \dots & \frac{\partial \mathscr{L}(\vartheta)}{\partial W_{L,11}} & \dots & \frac{\partial \mathscr{L}(\vartheta)}{\partial W_{L,1k}} & \frac{\partial \mathscr{L}(\vartheta)}{\partial W_{L,1k}} & \frac{\partial \mathscr{L}(\vartheta)}{\partial b_{11}} & \dots & \frac{\partial \mathscr{L}(\vartheta)}{\partial b_{L1}} \\
\frac{\partial \mathscr{L}(\vartheta)}{\partial W_{121}} & \dots & \frac{\partial \mathscr{L}(\vartheta)}{\partial W_{12n}} & \frac{\partial \mathscr{L}(\vartheta)}{\partial W_{221}} & \dots & \frac{\partial \mathscr{L}(\vartheta)}{\partial W_{22n}} & \dots & \frac{\partial \mathscr{L}(\vartheta)}{\partial W_{L,21}} & \dots & \frac{\partial \mathscr{L}(\vartheta)}{\partial W_{L,2k}} & \frac{\partial \mathscr{L}(\vartheta)}{\partial W_{L,2k}} & \frac{\partial \mathscr{L}(\vartheta)}{\partial b_{12}} & \dots & \frac{\partial \mathscr{L}(\vartheta)}{\partial b_{L2}} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\frac{\partial \mathscr{L}(\vartheta)}{\partial W_{1n1}} & \dots & \frac{\partial \mathscr{L}(\vartheta)}{\partial W_{1nn}} & \frac{\partial \mathscr{L}(\vartheta)}{\partial W_{2n1}} & \dots & \frac{\partial \mathscr{L}(\vartheta)}{\partial W_{2nn}} & \dots & \frac{\partial \mathscr{L}(\vartheta)}{\partial W_{L,n1}} & \dots & \frac{\partial \mathscr{L}(\vartheta)}{\partial W_{L,nk}} & \frac{\partial \mathscr{L}(\vartheta)}{\partial W_{L,nk}} & \frac{\partial \mathscr{L}(\vartheta)}{\partial b_{1n}} & \dots & \frac{\partial \mathscr{L}(\vartheta)}{\partial b_{Lk}}
\end{bmatrix}
$$

- $\nabla\vartheta$ is thus composed of
$\nabla W_1, \nabla W_2, \dots \nabla W_{L-1} \in \mathbb{R}^{n\times n}$, $\nabla W_L \in \mathbb{R}^{n\times k}$,
$\nabla b_1, \nabla b_2, \dots, \nabla b_{L-1} \in \mathbb{R}^{n}$ and $\nabla b_L \in \mathbb{R}^{k}$

We need to answer two questions

- How to choose the loss function $\mathscr{L}(\theta)$?
- How to compute $\nabla\theta$ which is composed of $\nabla W_1, \nabla W_2, \ldots, \nabla W_{L-1} \in \mathbb{R}^{n\times n}$, $\nabla W_L \in \mathbb{R}^{n\times k}$
$\nabla b_1, \nabla b_2, \ldots, \nabla b_{L-1} \in \mathbb{R}^n$ and $\nabla b_L \in \mathbb{R}^k$?

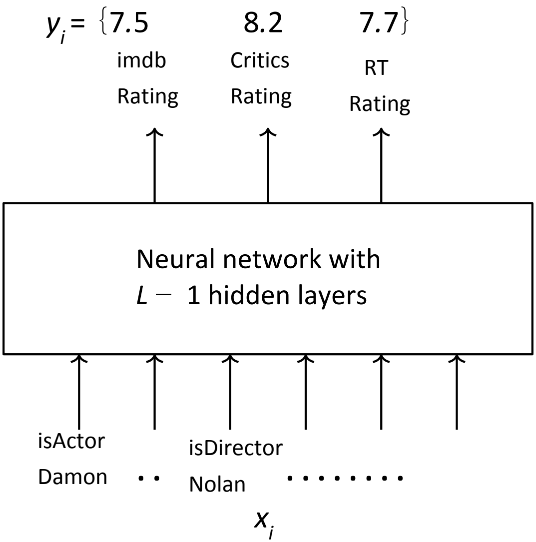
Module 4.3: Output Functions and Loss Functions

We need to answer two questions

- How to choose the loss function $L(\vartheta)$?
- How to compute $\nabla\vartheta$ which is composed of:

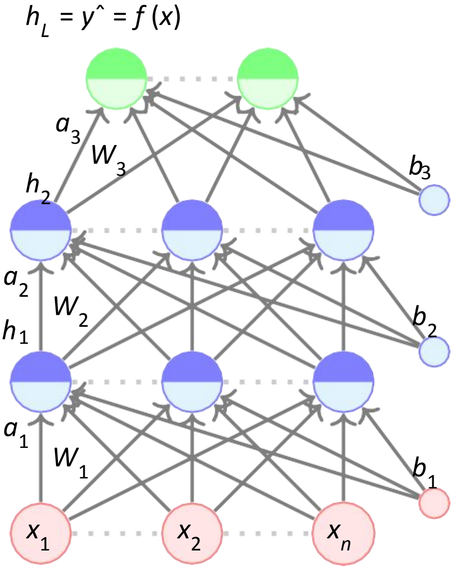
  $\nabla W_1, \nabla W_2, ..., \nabla W_{L-1} \in \mathbb{R}^{n\times n}, \nabla W_L \in \mathbb{R}^{n\times k}$

  $\nabla b_1, \nabla b_2, ..., \nabla b_{L-1} \in \mathbb{R}^{n}$ and $\nabla b_L \in \mathbb{R}^{k}$?



- The choice of loss function depends on the problem at hand
- We will illustrate this with the help of two examples
- Consider our movie example again but this time we are interested in predicting ratings
- Here $y_i \in \mathbb{R}^3$
- The loss function should capture how much $\hat{y}_i$ deviates from y_i
- If $y_i \in \mathbb{R}^n$ then the squared error loss can capture this deviation

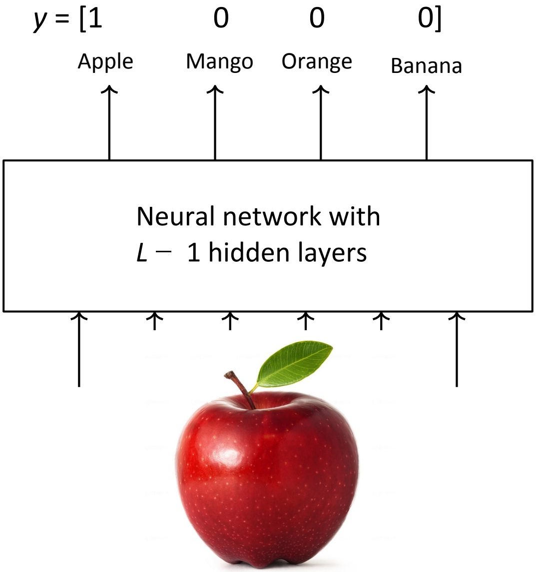
$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{3}(\hat{y}_{ij} - y_{ij})^2$$



- A related question: What should the output function 'O' be if $y_i \in \mathbb{R}$?
- More specifically, can it be the logistic function?
- No, because it restricts $\hat{y}_i$ to a value between 0 & 1 but we want $\hat{y}_i \in \mathbb{R}$
- So, in such cases it makes sense to have 'O' as linear function

$$f(x) = h_L = O(a_L)$$
$$= W_O a_L + b_O$$

- $\hat{y}_i = f(x_i)$ is no longer bounded between 0 and 1



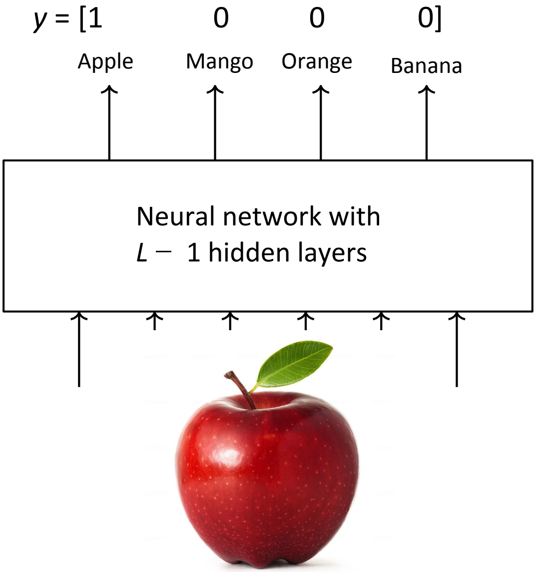
- Notice that y is a probability distribution
- Therefore we should also ensure that $\hat{y}$ is a probability distribution
- What choice of the output activation 'O' will ensure this ?

$$a_L = W_L h_{L-1} + b_L$$

$$\hat{y}_j = O(a_L)_j = \frac{e^{a_{L,j}}}{\sum_{i=1}^{k} e^{a_{L,i}}}$$

$O(a_L)_j$ is the j^{th} element of $\hat{y}$ and $a_{L,j}$ is the j^{th} element of the vector a_L.

- This function is called the *softmax* function

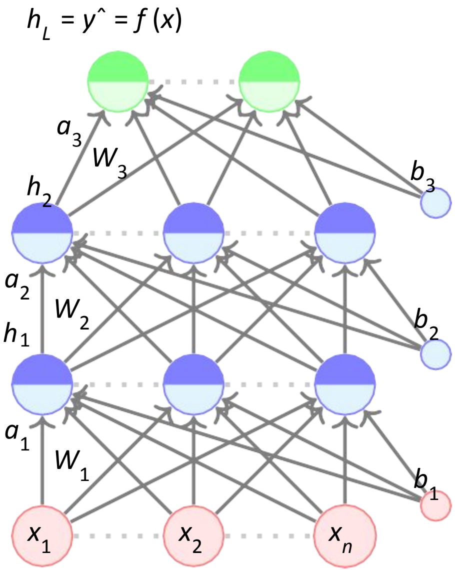


- Now that we have ensured that both y & $\hat{y}$ are probability distributions can you think of a function which captures the difference between them?
- Cross-entropy

$$L(\vartheta) = -\sum_{c=1}^{k} y_c \log \hat{y}_c$$

- Notice that

$$y_c = 1 \text{ if } c = l \text{ (the true class label)}$$
$$= 0 \text{ otherwise}$$
$$\therefore \; L(\vartheta) = -\log \hat{y}_l$$



- So, for classification problem (where you have to choose 1 of K classes), we use the following objective function

$$\underset{\vartheta}{\text{minimize}} \quad L(\vartheta) = -\log \hat{y}_l$$

$$\text{or} \quad \underset{\vartheta}{\text{maximize}} \quad -L(\vartheta) = \log \hat{y}_l$$

- But wait!

  Is $\hat{y}_l$ a function of $\vartheta = [W_1, W_2, ., W_L, b_1, b_2, ., b_L]$?
- Yes, it is indeed a function of ϑ

$$\hat{y}_l = [O(W_3 g(W_2 g(W_1 x + b_1) + b_2) + b_3)]_l$$

- What does $\hat{y}_l$ encode?
- It is the probability that x belongs to the l^{th} class (bring it as close to 1).
- $\log \hat{y}_l$ is called the *log-likelihood* of the data.

	Outputs	
	Real Values	Probabilities
Output Activation	Linear	Softmax
Loss Function	Squared Error	Cross Entropy

- Of course, there could be other loss functions depending on the problem at hand but the two loss functions that we just saw are encountered very often
- For the rest of this lecture we will focus on the case where the output activation is a softmax function and the loss function is cross entropy

Module 4.4: Backpropagation (Intuition)

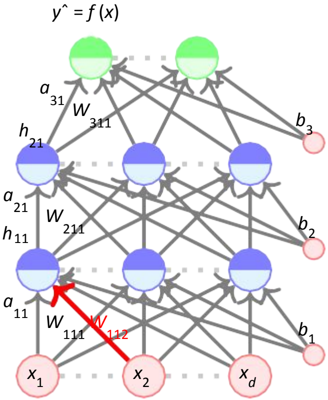
We need to answer two questions

- How to choose the loss function $\mathscr{L}(\theta)$?
- How to compute $\nabla\theta$ which is composed of:

  $\nabla W_1, \nabla W_2, ..., \nabla W_{L-1} \in \mathbb{R}^{n\times n}, \nabla W_L \in \mathbb{R}^{n\times k}$

  $\nabla b_1, \nabla b_2, ..., \nabla b_{L-1} \in \mathbb{R}^{n}$ and $\nabla b_L \in \mathbb{R}^{k}$?

- Let us focus on this one weight (W_{112}).
- To learn this weight using SGD we need a formula for $\frac{\partial \mathscr{L}(\theta)}{\partial W_{112}}$.
- We will see how to calculate this.



Algorithm: gradient descent()

$t \leftarrow 0$;
$max\_iterations \leftarrow 1000$;
Initialize θ_0;
while $t{+}{+} < max\_iterations$ **do**
$\quad \theta_{t+1} \leftarrow \theta_t - \eta \nabla \theta_t$;
end

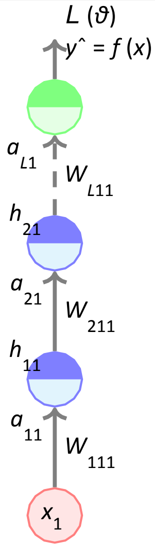
- First let us take the simple case when we have a deep but thin network.
- In this case it is easy to find the derivative by chain rule.

$$\frac{\partial \mathscr{L}(\theta)}{\partial W_{111}} = \frac{\partial \mathscr{L}(\theta)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_{L11}} \frac{\partial a_{L11}}{\partial h_{21}} \frac{\partial h_{21}}{\partial a_{21}} \frac{\partial a_{21}}{\partial h_{11}} \frac{\partial h_{11}}{\partial a_{11}} \frac{\partial a_{11}}{\partial W_{111}}$$

$$\frac{\partial \mathscr{L}(\theta)}{\partial W_{111}} = \frac{\partial \mathscr{L}(\theta)}{\partial h_{11}} \frac{\partial h_{11}}{\partial W_{111}} \qquad \text{(just compressing the chain rule)}$$

$$\frac{\partial \mathscr{L}(\theta)}{\partial W_{211}} = \frac{\partial \mathscr{L}(\theta)}{\partial h_{21}} \frac{\partial h_{21}}{\partial W_{211}}$$

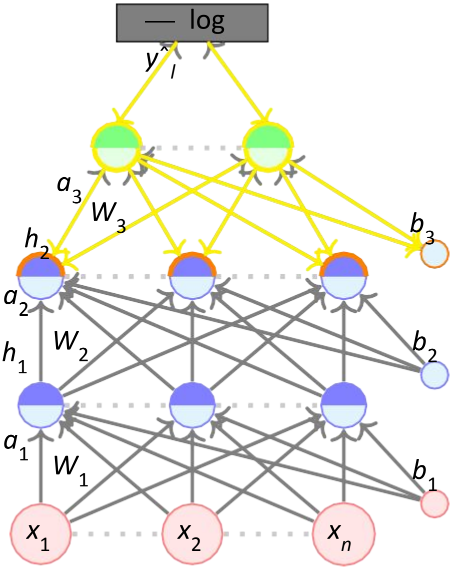
$$\frac{\partial \mathscr{L}(\theta)}{\partial W_{L11}} = \frac{\partial \mathscr{L}(\theta)}{\partial a_{L1}} \frac{\partial a_{L1}}{\partial W_{L11}}$$



Let us see an intuitive explanation of backpropagation before we get into the mathematical details

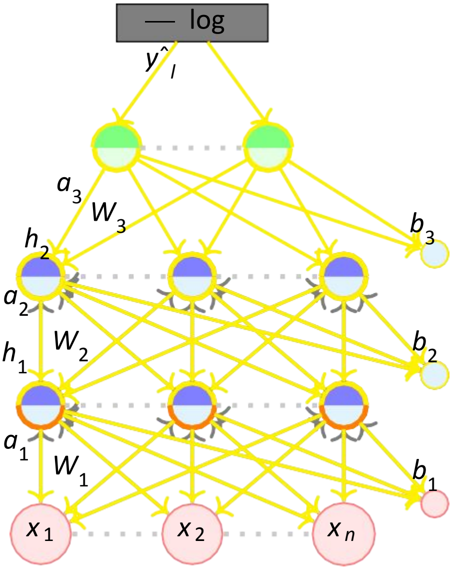
- We get a certain loss at the output and we try to figure out who is responsible for this loss
- So, we talk to the output layer and say "Hey! You are not producing the desired output, better take responsibility".
- The output layer says "Well, I take responsibility for my part but please understand that I am only as the good as the hidden layer and weights below me". After all . . .

$$f(x) = \hat{y} = O(W_L h_{L-1} + b_L)$$



- So, we talk to W_L, b_L and h_L and ask them "What is wrong with you?"
- W_L and b_L take full responsibility but h_L says "Well, please understand that I am only as good as the pre- activation layer"
- The pre-activation layer in turn says that I am only as good as the hidden layer and weights below me.
- We continue in this manner and realize that the responsibility lies with all the weights and biases (i.e. all the parameters of the model)
- But instead of talking to them directly, it is easier to talk to them through the hidden layers and output layers (and this is exactly what the chain rule allows us to do)

$$\underbrace{\frac{\partial L(\theta)}{\partial W_{111}}}_{\text{Talk to the weight directly}} = \underbrace{\frac{\partial L(\theta)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_3}}_{\text{Talk to the output layer}} \underbrace{\frac{\partial a_3}{\partial h_2} \frac{\partial h_2}{\partial a_2}}_{\text{Talk to the previous hidden layer}} \underbrace{\frac{\partial a_2}{\partial h_1} \frac{\partial h_1}{\partial a_1}}_{\text{Talk to the previous hidden layer}} \underbrace{\frac{\partial a_1}{\partial W_{111}}}_{\text{and now talk to the weights}}$$



Quantities of interest (roadmap for the remaining part):

- Gradient w.r.t. output units
- Gradient w.r.t. hidden units
- Gradient w.r.t. weights and biases

$$\underbrace{\frac{\partial \mathscr{L}(\theta)}{\partial W_{111}}}_{\text{Talk to the weight directly}} = \underbrace{\frac{\partial \mathscr{L}(\theta)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_3}}_{\text{Talk to the output layer}} \underbrace{\frac{\partial a_3}{\partial h_2} \frac{\partial h_2}{\partial a_2}}_{\text{Talk to the previous hidden layer}} \underbrace{\frac{\partial a_2}{\partial h_1} \frac{\partial h_1}{\partial a_1}}_{\text{Talk to the previous hidden layer}} \underbrace{\frac{\partial a_1}{\partial W_{111}}}_{\text{and now talk to the weights}}$$

- Our focus is on *Cross entropy loss* and *Softmax* output.