CS7015 (Deep Learning): Lecture 4

Feedforward Neural Networks, Backpropagation

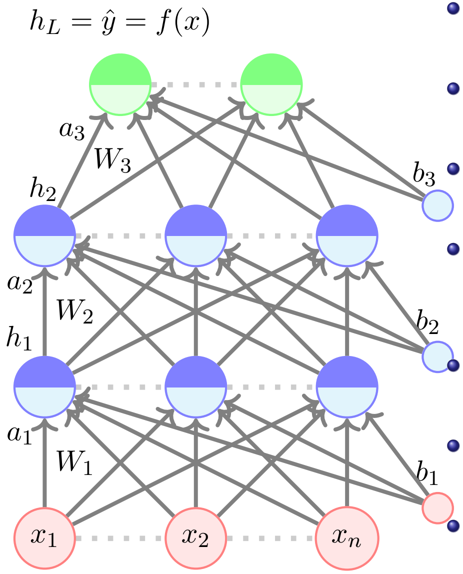
Mitesh M. Khapra

Department of Computer Science and Engineering
Indian Institute of Technology Madras

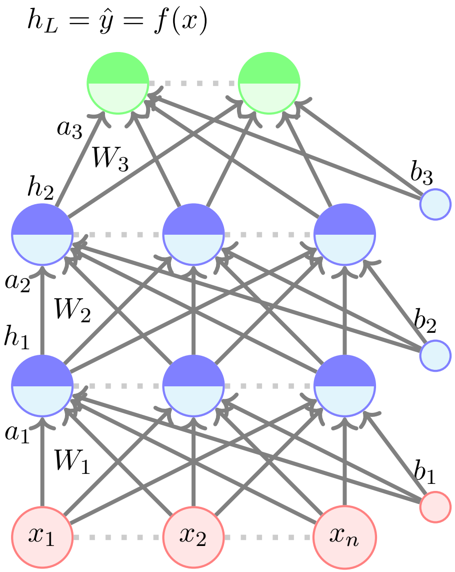
References/Acknowledgments

See the excellent videos by Hugo Larochelle on Backpropagation

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 $\mathbf{L-1}$ 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

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

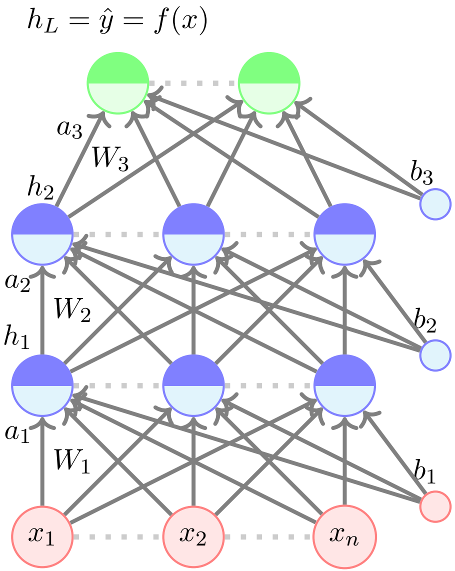
where g is called the activation function (for example, logistic, tanh, linear, *etc.*)

- 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

$$h_i = g(a_i)$$

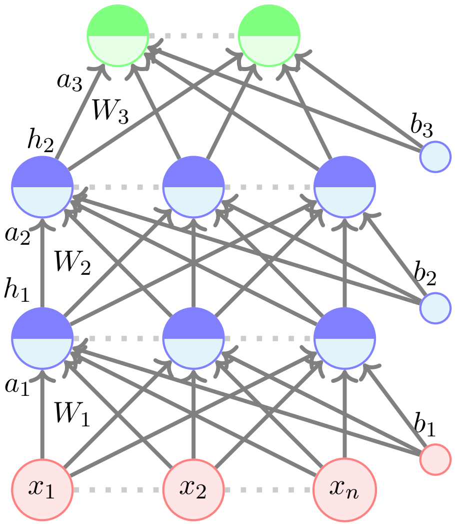
where g is called the activation function (for example, logistic, tanh, linear, *etc.*)

- 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.*)

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



- **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:**
  $\theta = W_1, .., W_L, b_1, b_2, ..., b_L (L = 3)$
- **Algorithm:** Gradient Descent with Back-propagation (we will see soon)
- **Objective/Loss/Error function:** Say,

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

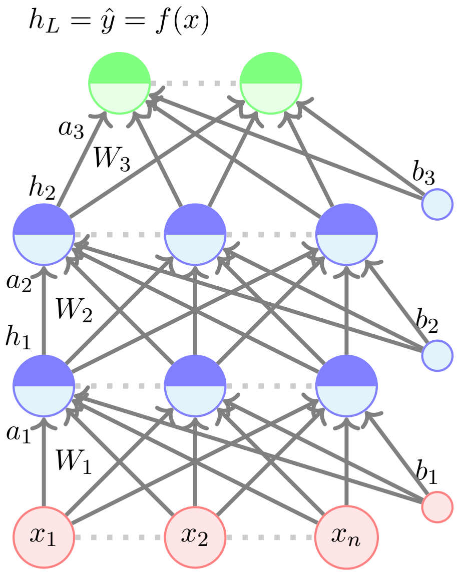
$$\textit{In general, min } \mathscr{L}(\theta)$$

where $\mathscr{L}(\theta)$ 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

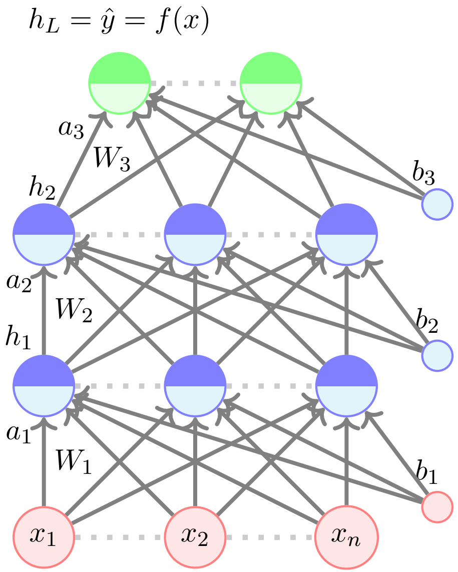


- Recall our gradient descent algorithm

Algorithm: gradient_descent()

```
$t \leftarrow 0$;
$max\_iterations \leftarrow 1000$;
$Initialize \quad w_0, b_0$;
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
```

-

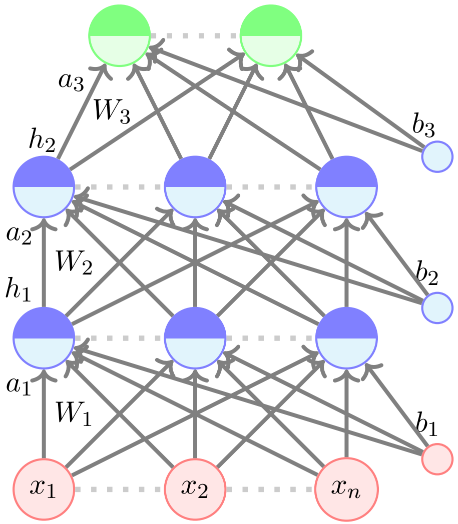


- Recall our gradient descent algorithm
- We can write it more concisely as

```
Algorithm: gradient_descent()
$t \leftarrow 0$;
$max\_iterations \leftarrow 1000$;
$Initialize \quad \theta_0 = [w_0, b_0]$;
while $t{+}{+} < max\_iterations$ do
    $\theta_{t+1} \leftarrow \theta_t - \eta \nabla \theta_t$;
end
```

- where $\nabla \theta_t = \left[\frac{\partial \mathscr{L}(\theta)}{\partial w_t}, \frac{\partial \mathscr{L}(\theta)}{\partial b_t}\right]^T$
- 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

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



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

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

- where $\nabla \theta_t = \left[\frac{\partial \mathscr{L}(\theta)}{\partial W_{1,t}}, .., \frac{\partial \mathscr{L}(\theta)}{\partial W_{L,t}}, \frac{\partial \mathscr{L}(\theta)}{\partial b_{1,t}}, .., \frac{\partial \mathscr{L}(\theta)}{\partial b_{L,t}}\right]^T$
- 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 $\nabla\theta$ looks much more nasty

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

- $\nabla\theta$ is thus 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 $\nabla b_L \in \mathbb{R}^k$

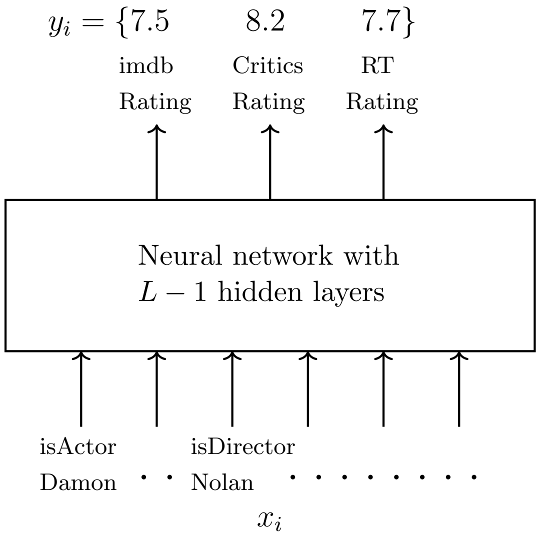
We need to answer two questions

- 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 $\nabla b_L \in \mathbb{R}^{k}$?

Module 4.3: Output Functions and Loss Functions

We need to answer two questions

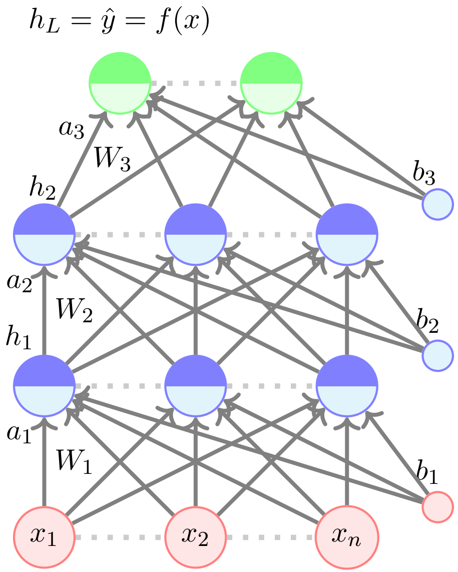
- How to choose the loss function $\mathscr{L}(\theta)$?
- How to compute $\nabla\theta$ which is composed of:
  $\nabla W_1, \nabla W_2, ..., \nabla W_{L-1} \in \mathbb{R}^{n\times n}, \nabla W_L \in \mathbb{R}^{n\times k}$
  $\nabla b_1, \nabla b_2, ..., \nabla b_{L-1} \in \mathbb{R}^{n}$ and $\nabla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



$y_i = \{7.5 \quad 8.2 \quad 7.7\}$
imdb Rating
Critics Rating
RT Rating
Neural network with $L-1$ hidden layers
isActor Damon
isDirector Nolan
x_i

- 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

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

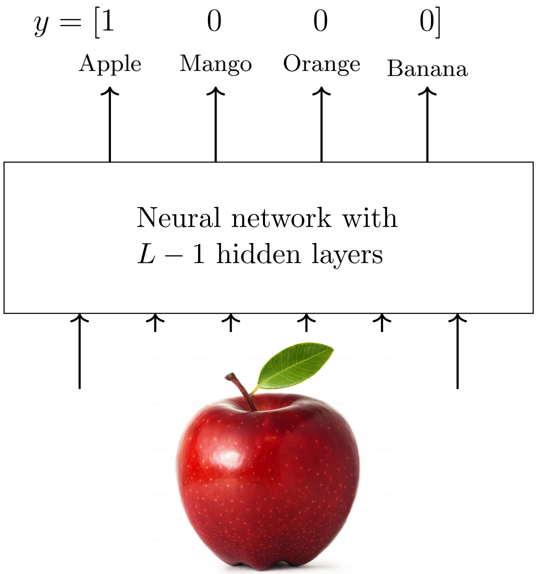


- A related question: What should the output function 'O' be if $y_i \in \mathbb{R}$?
- More specifically, can it be the logistic function?
- No, because it restricts $\hat{y}_i$ to a value between 0 & 1 but we want $\hat{y}_i \in \mathbb{R}$
- So, in such cases it makes sense to have 'O' as linear function

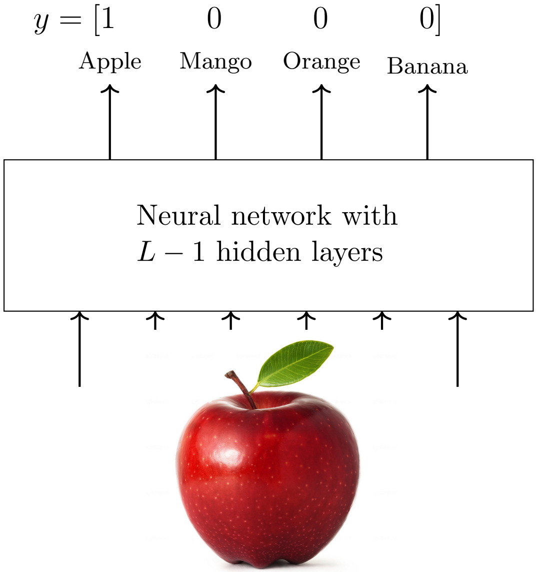
$$\begin{aligned} f(x) = h_L &= O(a_L) \\ &= W_O a_L + b_O \end{aligned}$$

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

Intentionally left blank



- Now let us consider another problem for which a different loss function would be appropriate
- Suppose we want to classify an image into 1 of k classes
- Here again we could use the squared error loss to capture the deviation
- But can you think of a better function?

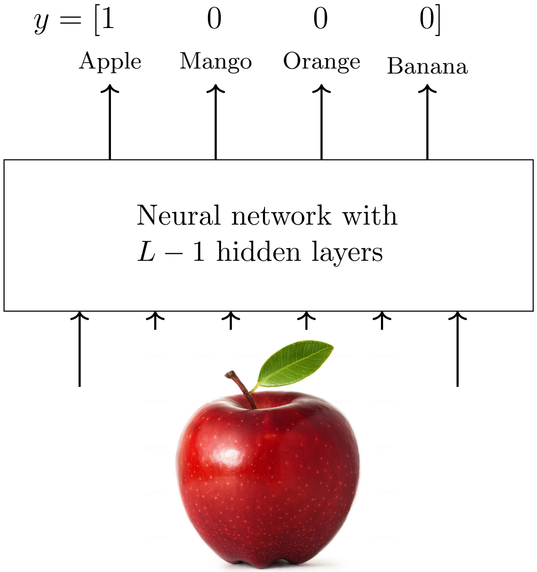


- Notice that y is a probability distribution
- Therefore we should also ensure that $\hat{y}$ is a probability distribution
- What choice of the output activation 'O' will ensure this ?

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

$O(a_L)_j$ 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

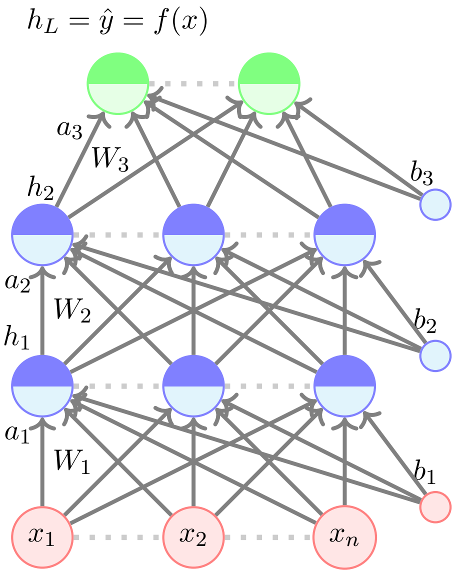


- Now that we have ensured that both y & $\hat{y}$ are probability distributions can you think of a function which captures the difference between them?
- Cross-entropy

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

- Notice that

$$\begin{aligned} y_c &= 1 \quad \text{if } c = \ell \text{ (the true class label)} \\ &= 0 \quad \text{otherwise} \\ \therefore \quad \mathscr{L}(\theta) &= -\log \hat{y}_\ell \end{aligned}$$



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

$$\begin{array}{lll} & \underset{\theta}{\text{minimize}} & \mathscr{L}(\theta) = -\log \hat{y}_\ell \\ \text{or} & \underset{\theta}{\text{maximize}} & -\mathscr{L}(\theta) = \log \hat{y}_\ell \end{array}$$

- But wait!
  Is $\hat{y}_\ell$ a function of $\theta = [W_1, W_2, ., W_L, b_1, b_2, ., b_L]$?
- Yes, it is indeed a function of θ

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

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

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

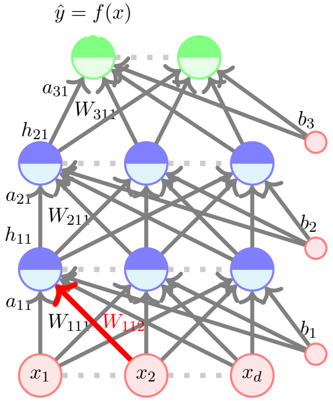
- Of course, there could be other loss functions depending on the problem at hand but the two loss functions that we just saw are encountered very often
- For the rest of this lecture we will focus on the case where the output activation is a softmax function and the loss function is cross entropy

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()

```
$t \leftarrow 0$;
$max\_iterations \leftarrow 1000$;
$Initialize \quad \theta_0$;
while $t{++} < max\_iterations$ do
    $\theta_{t+1} \leftarrow \theta_t - \eta \nabla \theta_t$;
end
```

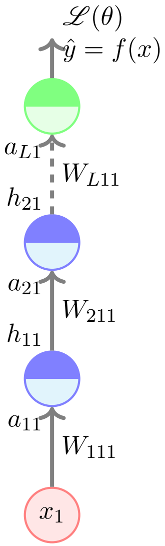
- First let us take the simple case when we have a deep but thin network.
- In this case it is easy to find the derivative by chain rule.

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

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

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

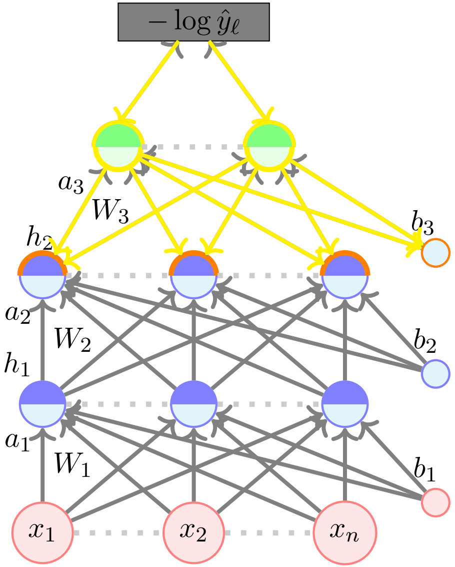
$$\frac{\partial \mathscr{L}(\theta)}{\partial W_{L11}} = \frac{\partial \mathscr{L}(\theta)}{\partial a_{L1}} \frac{\partial a_{L1}}{\partial W_{L11}}$$



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

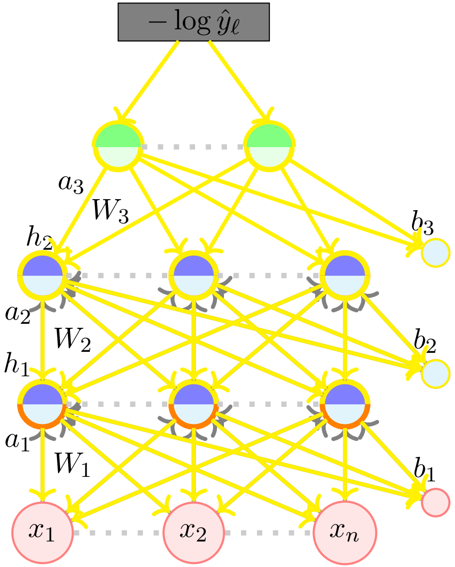
- We get a certain loss at the output and we try to figure out who is responsible for this loss
- So, we talk to the output layer and say "Hey! You are not producing the desired output, better take responsibility".
- The output layer says "Well, I take responsibility for my part but please understand 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)

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



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.

Module 4.5: Backpropagation: Computing Gradients w.r.t. the Output Units

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

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

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

Let us first consider the partial derivative w.r.t. i-th output

$$\mathscr{L}(\theta) = -\log \hat{y}_\ell \quad (\ell = \text{true class label})$$

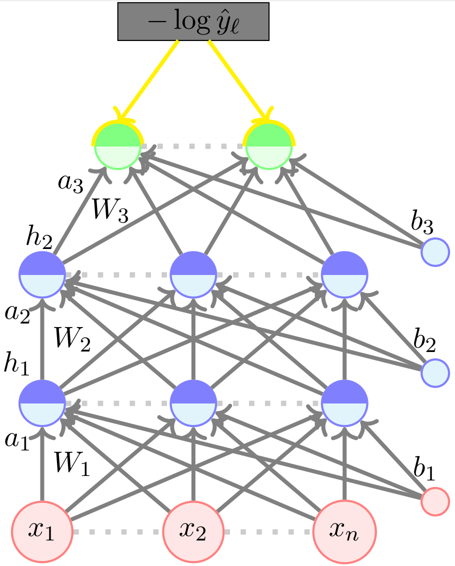
$$\frac{\partial}{\partial \hat{y}_i}(\mathscr{L}(\theta)) = \frac{\partial}{\partial \hat{y}_i}(-\log \hat{y}_\ell)$$

$$= -\frac{1}{\hat{y}_\ell} \quad \text{if } i = \ell$$

$$= \quad 0 \qquad \textit{otherwise}$$

More compactly,

$$\frac{\partial}{\partial \hat{y}_i}(\mathscr{L}(\theta)) = -\frac{\mathbb{1}_{(i=\ell)}}{\hat{y}_\ell}$$



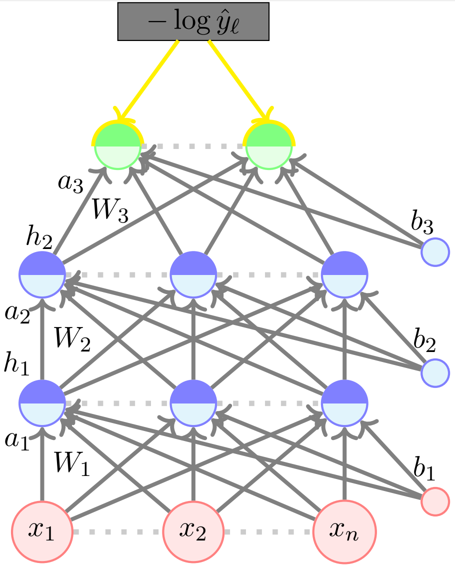
$$\frac{\partial}{\partial \hat{y}_i}(\mathscr{L}(\theta)) = -\frac{\mathbb{1}_{(\ell=i)}}{\hat{y}_\ell}$$

We can now talk about the gradient w.r.t. the vector $\hat{y}$

$$\nabla_{\hat{\mathbf{y}}}\mathscr{L}(\theta) = \begin{bmatrix} \frac{\partial \mathscr{L}(\theta)}{\partial \hat{y}_1} \\ \vdots \\ \frac{\partial \mathscr{L}(\theta)}{\partial \hat{y}_k} \end{bmatrix} = -\frac{1}{\hat{y}_\ell}\begin{bmatrix} \mathbb{1}_{\ell=1} \\ \mathbb{1}_{\ell=2} \\ \vdots \\ \mathbb{1}_{\ell=k} \end{bmatrix}$$

$$= -\frac{1}{\hat{y}_\ell} e_\ell$$

where $e(\ell)$ is a k-dimensional vector whose ℓ-th element is 1 and all other elements are 0.



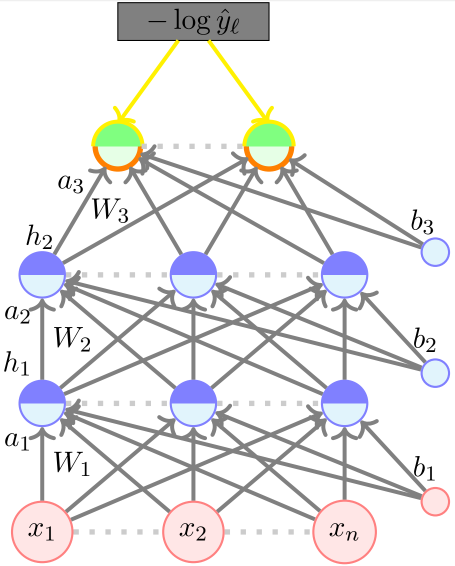
What we are actually interested in is

$$\frac{\partial \mathscr{L}(\theta)}{\partial a_{Li}} = \frac{\partial(-\log \hat{y}_\ell)}{\partial a_{Li}}$$
$$= \frac{\partial(-\log \hat{y}_\ell)}{\partial \hat{y}_\ell} \frac{\partial \hat{y}_\ell}{\partial a_{Li}}$$

Does $\hat{y}_\ell$ depend on a_{Li} ? Indeed, it does.

$$\hat{y}_\ell = \frac{exp(a_{L\ell})}{\sum_i exp(a_{Li})}$$

Having established this, we will now derive the full expression on the next slide



$$\begin{aligned}
\frac{\partial}{\partial a_{Li}} - \log \hat{y}_\ell &= \frac{-1}{\hat{y}_\ell}\frac{\partial}{\partial a_{Li}}\hat{y}_\ell \\
&= \frac{-1}{\hat{y}_\ell}\frac{\partial}{\partial a_{Li}} softmax(\mathbf{a}_L)_\ell \\
&= \frac{-1}{\hat{y}_\ell}\frac{\partial}{\partial a_{Li}} \frac{\exp(\mathbf{a}_L)_\ell}{\sum_{i'}\exp(\mathbf{a}_L)_\ell} \\
&= \frac{-1}{\hat{y}_\ell}\left(\frac{\frac{\partial}{\partial a_{Li}}\exp(\mathbf{a}_L)_\ell}{\sum_{i'}\exp(\mathbf{a}_L)_{i'}} - \frac{\exp(\mathbf{a}_L)_\ell\left(\frac{\partial}{\partial a_{Li}}\sum_{i'}\exp(\mathbf{a}_L)_{i'}\right)}{(\sum_{i'}(\exp(\mathbf{a}_L)_{i'})^2}\right) \\
&= \frac{-1}{\hat{y}_\ell}\left(\frac{\mathbb{1}_{(\ell=i)}\exp(\mathbf{a}_L)_\ell}{\sum_{i'}\exp(\mathbf{a}_L)_{i'}} - \frac{\exp(\mathbf{a}_L)_\ell}{\sum_{i'}\exp(\mathbf{a}_L)_{i'}}\frac{\exp(\mathbf{a}_L)_i}{\sum_{i'}\exp(\mathbf{a}_L)_{i'}}\right) \\
&= \frac{-1}{\hat{y}_\ell}\left(\mathbb{1}_{(\ell=i)} softmax(\mathbf{a}_L)_\ell - softmax(\mathbf{a}_L)_\ell softmax(\mathbf{a}_L)_i\right) \\
&= \frac{-1}{\hat{y}_\ell}\left(\mathbb{1}_{(\ell=i)}\hat{y}_\ell - \hat{y}_\ell\hat{y}_i\right) \\
&= -\left(\mathbb{1}_{(\ell=i)} - \hat{y}_i\right)
\end{aligned}$$

$$\frac{\partial\frac{g(x)}{h(x)}}{\partial x} = \frac{\partial g(x)}{\partial x}\frac{1}{h(x)} - \frac{g(x)}{h(x)^2}\frac{\partial h(x)}{\partial x}$$

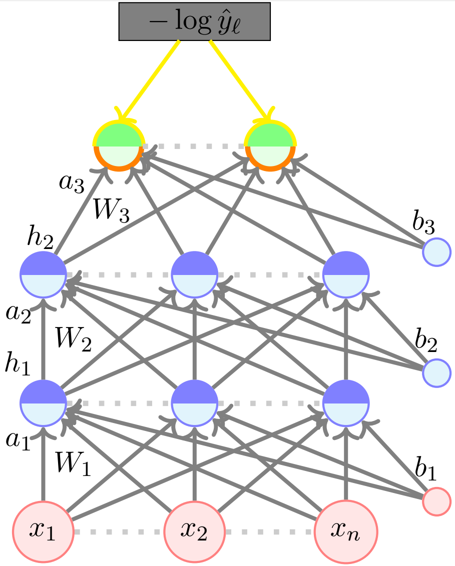
So far we have derived the partial derivative w.r.t. the i-th element of $\mathbf{a}_L$

$$\frac{\partial \mathscr{L}(\theta)}{\partial a_{L,i}} = -(\mathbb{1}_{\ell=i} - \hat{y}_i)$$

We can now write the gradient w.r.t. the vector $\mathbf{a}_L$

$$\nabla_{\mathbf{a_L}} \mathscr{L}(\theta) = \begin{bmatrix} \frac{\partial \mathscr{L}(\theta)}{\partial a_{L1}} \\ \vdots \\ \frac{\partial \mathscr{L}(\theta)}{\partial a_{Lk}} \end{bmatrix} = \begin{bmatrix} -(\mathbb{1}_{\ell=1} - \hat{y}_1) \\ -(\mathbb{1}_{\ell=2} - \hat{y}_2) \\ \vdots \\ -(\mathbb{1}_{\ell=k} - \hat{y}_k) \end{bmatrix}$$

$$= -(\mathbf{e}(\ell) - \hat{y})$$



Module 4.6: Backpropagation: Computing Gradients w.r.t. Hidden Units

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

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

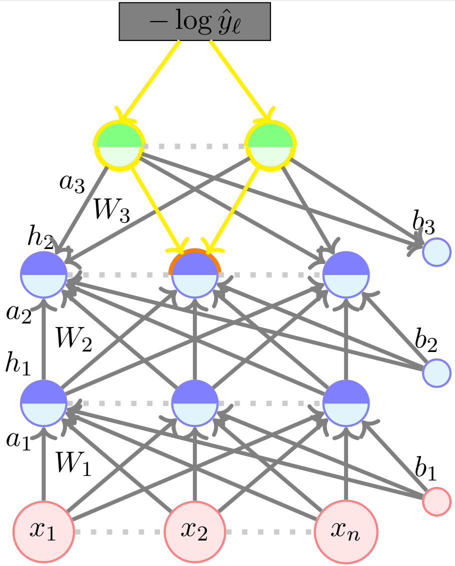
- Our focus is on *Cross entropy loss* and *Softmax* output.

Chain rule along multiple paths: If a function $p(z)$ can be written as a function of intermediate results $q_i(z)$ then we have :

$$\frac{\partial p(z)}{\partial z} = \sum_m \frac{\partial p(z)}{\partial q_m(z)} \frac{\partial q_m(z)}{\partial z}$$

In our case:

- $p(z)$ is the loss function $\mathscr{L}(\theta)$
- $z = h_{ij}$
- $q_m(z) = a_{Lm}$



Intentionally left blank

$$\frac{\partial \mathscr{L}(\theta)}{\partial h_{ij}} = \sum_{m=1}^{k} \frac{\partial \mathscr{L}(\theta)}{\partial a_{i+1,m}} \frac{\partial a_{i+1,m}}{\partial h_{ij}}$$

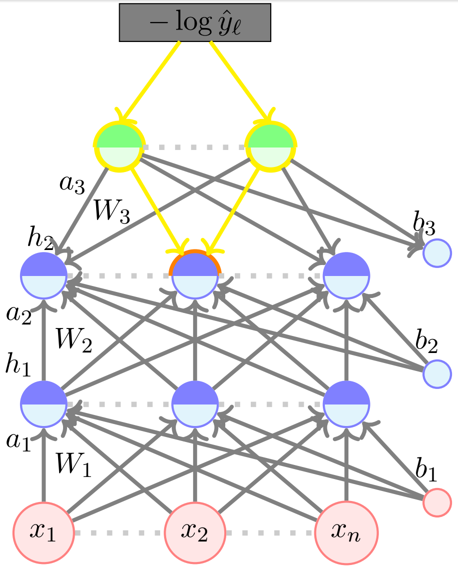
$$= \sum_{m=1}^{k} \frac{\partial \mathscr{L}(\theta)}{\partial a_{i+1,m}} W_{i+1,m,j}$$

Now consider these two vectors,

$$\nabla_{a_{i+1}} \mathscr{L}(\theta) = \begin{bmatrix} \frac{\partial \mathscr{L}(\theta)}{\partial a_{i+1,1}} \\ \vdots \\ \frac{\partial \mathscr{L}(\theta)}{\partial a_{i+1,k}} \end{bmatrix} ; W_{i+1,\,\cdot\,,j} = \begin{bmatrix} W_{i+1,1,j} \\ \vdots \\ W_{i+1,k,j} \end{bmatrix}$$

$W_{i+1,\,\cdot\,,j}$ is the j-th column of W_{i+1}; see that,

$$(W_{i+1,\,\cdot\,,j})^T \nabla_{a_{i+1}} \mathscr{L}(\theta) = \sum_{m=1}^{k} \frac{\partial \mathscr{L}(\theta)}{\partial a_{i+1,m}} W_{i+1,m,j}$$



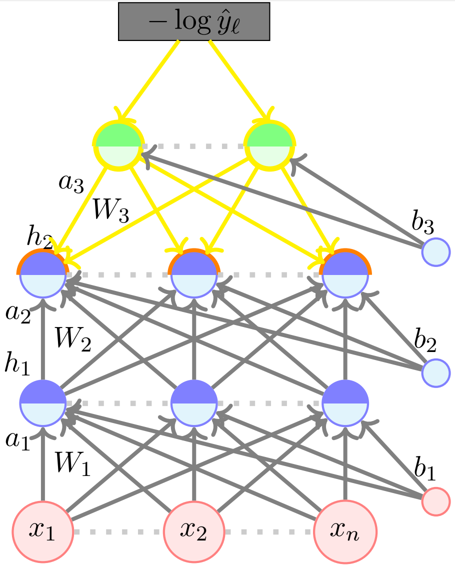
$$a_{i+1} = W_{i+1} h_{ij} + b_{i+1}$$

$$We\ have, \frac{\partial \mathscr{L}(\theta)}{\partial h_{ij}} = (W_{i+1,.,j})^T \nabla_{a_{i+1}} \mathscr{L}(\theta)$$

We can now write the gradient w.r.t. h_i

$$\nabla_{\mathbf{h_i}} \mathscr{L}(\theta) = \begin{bmatrix} \frac{\partial \mathscr{L}(\theta)}{\partial h_{i1}} \\ \frac{\partial \mathscr{L}(\theta)}{\partial h_{i2}} \\ \vdots \\ \frac{\partial \mathscr{L}(\theta)}{\partial h_{in}} \end{bmatrix} = \begin{bmatrix} (W_{i+1,\cdot,1})^T \nabla_{a_{i+1}} \mathscr{L}(\theta) \\ (W_{i+1,\cdot,2})^T \nabla_{a_{i+1}} \mathscr{L}(\theta) \\ \vdots \\ (W_{i+1,\cdot,n})^T \nabla_{a_{i+1}} \mathscr{L}(\theta) \end{bmatrix}$$
$$= (W_{i+1})^T (\nabla_{a_{i+1}} \mathscr{L}(\theta))$$

- We are almost done except that we do not know how to calculate $\nabla_{a_{i+1}} \mathscr{L}(\theta)$ for $i < L-1$
- We will see how to compute that



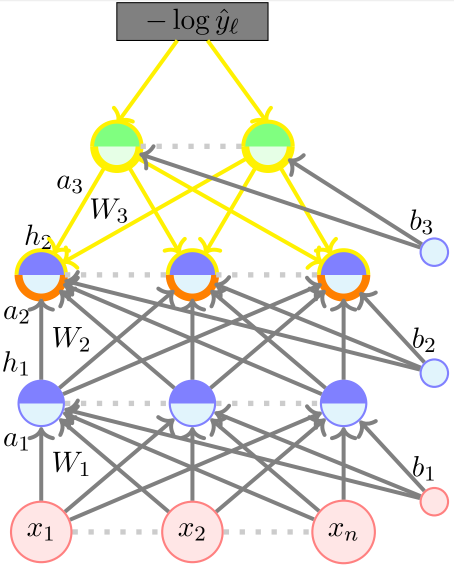
$$\nabla_{\mathbf{a_i}} \mathscr{L}(\theta) = \begin{bmatrix} \frac{\partial \mathscr{L}(\theta)}{\partial a_{i1}} \\ \vdots \\ \frac{\partial \mathscr{L}(\theta)}{\partial a_{in}} \end{bmatrix}$$

$$\frac{\partial \mathscr{L}(\theta)}{\partial a_{ij}} = \frac{\partial \mathscr{L}(\theta)}{\partial h_{ij}} \frac{\partial h_{ij}}{\partial a_{ij}}$$

$$= \frac{\partial \mathscr{L}(\theta)}{\partial h_{ij}} g^{'}(a_{ij}) \quad [\because h_{ij} = g(a_{ij})]$$

$$\nabla_{\mathbf{a_i}} \mathscr{L}(\theta) = \begin{bmatrix} \frac{\partial \mathscr{L}(\theta)}{\partial h_{i1}} g^{'}(a_{i1}) \\ \vdots \\ \frac{\partial \mathscr{L}(\theta)}{\partial h_{in}} g^{'}(a_{in}) \end{bmatrix}$$

$$= \nabla_{h_i} \mathscr{L}(\theta) \odot [\ldots, g^{'}(a_{ik}), \ldots]$$



Module 4.7: Backpropagation: Computing Gradients w.r.t. Parameters

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.

Recall that,

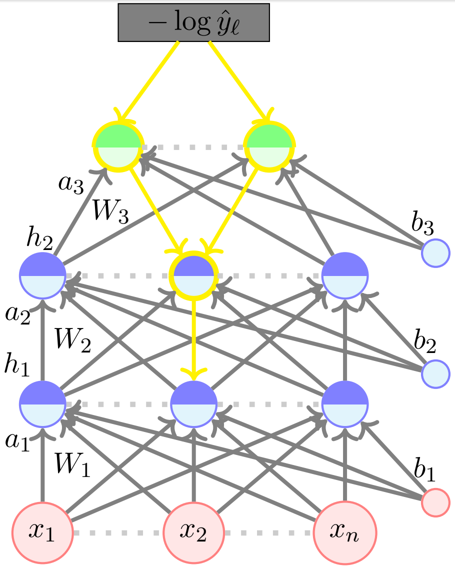
$$\mathbf{a_k} = \mathbf{b_k} + W_k \mathbf{h_{k-1}}$$

$$\frac{\partial a_{ki}}{\partial W_{kij}} = h_{k-1,j}$$

$$\frac{\partial \mathscr{L}(\theta)}{\partial W_{kij}} = \frac{\partial \mathscr{L}(\theta)}{\partial a_{ki}} \frac{\partial a_{ki}}{\partial W_{kij}}$$

$$= \frac{\partial \mathscr{L}(\theta)}{\partial a_{ki}} h_{k-1,j}$$

$$\nabla_{W_k} \mathscr{L}(\theta) = \begin{bmatrix} \frac{\partial \mathscr{L}(\theta)}{\partial W_{k11}} & \frac{\partial \mathscr{L}(\theta)}{\partial W_{k12}} & \cdots & \cdots & \frac{\partial \mathscr{L}(\theta)}{\partial W_{k1n}} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & \cdots & \cdots & \cdots & \frac{\partial \mathscr{L}(\theta)}{\partial W_{knn}} \end{bmatrix}$$



Intentionally left blank

Lets take a simple example of a $W_k \in \mathbb{R}^{3\times3}$ and see what each entry looks like

$$\nabla_{W_k}\mathscr{L}(\theta) = \begin{bmatrix} \frac{\partial \mathscr{L}(\theta)}{\partial W_{k11}} & \frac{\partial \mathscr{L}(\theta)}{\partial W_{k12}} & \frac{\partial \mathscr{L}(\theta)}{\partial W_{k13}} \\ \frac{\partial \mathscr{L}(\theta)}{\partial W_{k21}} & \frac{\partial \mathscr{L}(\theta)}{\partial W_{k22}} & \frac{\partial \mathscr{L}(\theta)}{\partial W_{k23}} \\ \frac{\partial \mathscr{L}(\theta)}{\partial W_{k31}} & \frac{\partial \mathscr{L}(\theta)}{\partial W_{k32}} & \frac{\partial \mathscr{L}(\theta)}{\partial W_{k33}} \end{bmatrix} \quad \frac{\partial \mathscr{L}(\theta)}{\partial W_{kij}} = \frac{\partial \mathscr{L}(\theta)}{\partial a_{ki}} \frac{\partial a_{ki}}{\partial W_{kij}}$$

$$\nabla_{W_k}\mathscr{L}(\theta) = \begin{bmatrix} \frac{\partial \mathscr{L}(\theta)}{\partial a_{k1}} h_{k-1,1} & \frac{\partial \mathscr{L}(\theta)}{\partial a_{k1}} h_{k-1,2} & \frac{\partial \mathscr{L}(\theta)}{\partial a_{k1}} h_{k-1,3} \\ \frac{\partial \mathscr{L}(\theta)}{\partial a_{k2}} h_{k-1,1} & \frac{\partial \mathscr{L}(\theta)}{\partial a_{k2}} h_{k-1,2} & \frac{\partial \mathscr{L}(\theta)}{\partial a_{k2}} h_{k-1,3} \\ \frac{\partial \mathscr{L}(\theta)}{\partial a_{k3}} h_{k-1,1} & \frac{\partial \mathscr{L}(\theta)}{\partial a_{k3}} h_{k-1,2} & \frac{\partial \mathscr{L}(\theta)}{\partial a_{k3}} h_{k-1,3} \end{bmatrix} = \nabla_{a_k}\mathscr{L}(\theta) \cdot \mathbf{h_{k-1}}^T$$

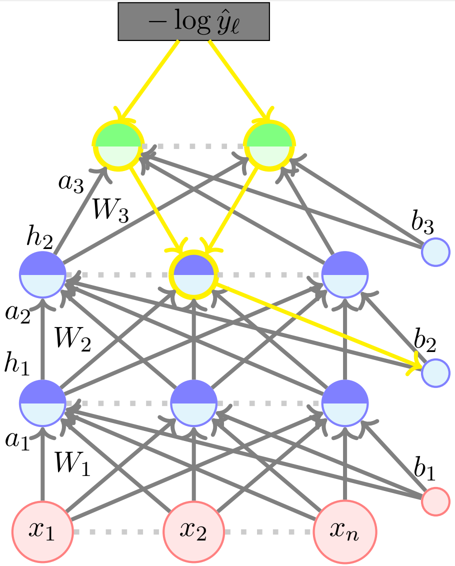
Finally, coming to the biases

$$a_{ki} = b_{ki} + \sum_j W_{kij} h_{k-1,j}$$

$$\frac{\partial \mathscr{L}(\theta)}{\partial b_{ki}} = \frac{\partial \mathscr{L}(\theta)}{\partial a_{ki}} \frac{\partial a_{ki}}{\partial b_{ki}} = \frac{\partial \mathscr{L}(\theta)}{\partial a_{ki}}$$

We can now write the gradient w.r.t. the vector b_k

$$\nabla_{\mathbf{b_k}} \mathscr{L}(\theta) = \begin{bmatrix} \frac{\partial \mathscr{L}(\theta)}{a_{k1}} \\ \frac{\partial \mathscr{L}(\theta)}{a_{k2}} \\ \vdots \\ \frac{\partial \mathscr{L}(\theta)}{a_{kn}} \end{bmatrix} = \nabla_{\mathbf{a_k}} \mathscr{L}(\theta)$$



Module 4.8: Backpropagation: Pseudo code

Finally, we have all the pieces of the puzzle

$$\nabla_{\mathbf{a_L}} \mathscr{L}(\theta) \quad \text{(gradient w.r.t. output layer)}$$

$$\nabla_{\mathbf{h_k}} \mathscr{L}(\theta), \nabla_{\mathbf{a_k}} \mathscr{L}(\theta) \quad \text{(gradient w.r.t. hidden layers, } 1 \leq k < L\text{)}$$

$$\nabla_{W_k} \mathscr{L}(\theta), \nabla_{\mathbf{b_k}} \mathscr{L}(\theta) \quad \text{(gradient w.r.t. weights and biases, } 1 \leq k \leq L\text{)}$$

We can now write the full learning algorithm

Algorithm: gradient_descent()

```
$t \leftarrow 0$;
$max\_iterations \leftarrow 1000$;
$Initialize \quad \theta_0 = [W_1^0, ..., W_L^0, b_1^0, ..., b_L^0]$;
while $t{+}{+} < max\_iterations$ do
    $h_1, h_2, ..., h_{L-1}, a_1, a_2, ..., a_L, \hat{y} = forward\_propagation(\theta_t)$;
    $\nabla\theta_t = backward\_propagation(h_1, h_2, ..., h_{L-1}, a_1, a_2, ..., a_L, y, \hat{y})$;
    $\theta_{t+1} \leftarrow \theta_t - \eta\nabla\theta_t$;
end
```

Algorithm: forward_propagation(θ)

```
for $k = 1$ to $L - 1$ do
    $a_k = b_k + W_k h_{k-1}$;
    $h_k = g(a_k)$;
end
$a_L = b_L + W_L h_{L-1}$;
$\hat{y} = O(a_L)$;
```

Just do a forward propagation and compute all h_i's, a_i's, and $\hat{y}$

Algorithm: back_propagation($h_1, h_2, ..., h_{L-1}, a_1, a_2, ..., a_L, y, \hat{y}$)

```
//Compute output gradient ;
$\nabla_{a_L} \mathscr{L}(\theta) = -(e(y) - \hat{y})$ ;
for $k = L$ to 1 do
    // Compute gradients w.r.t. parameters ;
    $\nabla_{W_k} \mathscr{L}(\theta) = \nabla_{a_k} \mathscr{L}(\theta) h_{k-1}^T$ ;
    $\nabla_{b_k} \mathscr{L}(\theta) = \nabla_{a_k} \mathscr{L}(\theta)$ ;
    // Compute gradients w.r.t. layer below ;
    $\nabla_{h_{k-1}} \mathscr{L}(\theta) = W_k^T (\nabla_{a_k} \mathscr{L}(\theta))$ ;
    // Compute gradients w.r.t. layer below (pre-activation);
    $\nabla_{a_{k-1}} \mathscr{L}(\theta) = \nabla_{h_{k-1}} \mathscr{L}(\theta) \odot [\ldots, g'(a_{k-1,j}), \ldots]$ ;
end
```

Module 4.9: Derivative of the activation function

Now, the only thing we need to figure out is how to compute g'

Logistic function

$$\begin{aligned} g(z) &= \sigma(z) \\ &= \frac{1}{1+e^{-z}} \\ g'(z) &= (-1)\frac{1}{(1+e^{-z})^2}\frac{d}{dz}(1+e^{-z}) \\ &= (-1)\frac{1}{(1+e^{-z})^2}(-e^{-z}) \\ &= \frac{1}{1+e^{-z}}\left(\frac{1+e^{-z}-1}{1+e^{-z}}\right) \\ &= g(z)(1-g(z)) \end{aligned}$$

tanh

$$\begin{aligned} g(z) &= \tanh(z) \\ &= \frac{e^z-e^{-z}}{e^z+e^{-z}} \\ g'(z) &= \frac{\left(\begin{array}{c}(e^z+e^{-z})\frac{d}{dz}(e^z-e^{-z}) \\ -(e^z-e^{-z})\frac{d}{dz}(e^z+e^{-z})\end{array}\right)}{(e^z+e^{-z})^2} \\ &= \frac{(e^z+e^{-z})^2-(e^z-e^{-z})^2}{(e^z+e^{-z})^2} \\ &= 1-\frac{(e^z-e^{-z})^2}{(e^z+e^{-z})^2} \\ &= 1-(g(z))^2 \end{aligned}$$