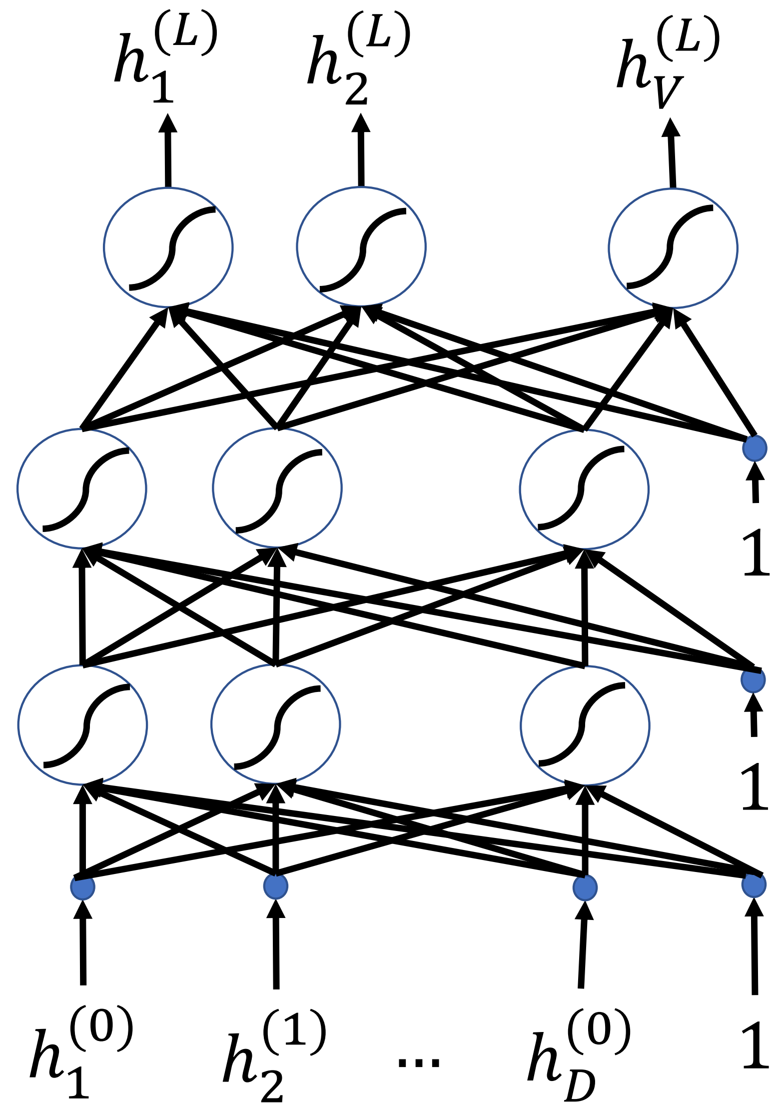
Lecture 10: Back-Propagation

Mark Hasegawa-Johnson

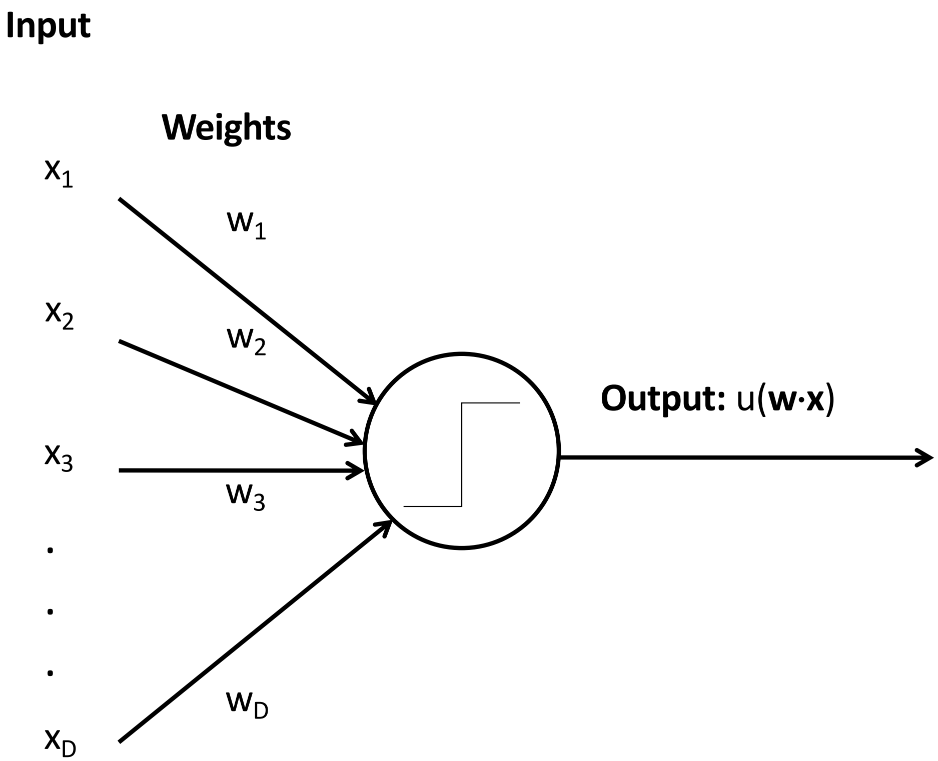
March 1, 2021



Outline

- Breaking the constraints of linearity: multi-layer neural nets
- What’s inside a multi-layer neural net?
- Forward-propagation example
- Gradient descent
- Finding the derivative: back-propagation

Biological Inspiration: McCulloch-Pitts Artificial Neuron, 1943



- In 1943, McCulloch & Pitts proposed that biological neurons have a nonlinear activation function (a step function) whose input is a weighted linear combination of the currents generated by other neurons.
- They showed lots of examples of mathematical and logical functions that could be computed using networks of simple neurons like this.

Biological Inspiration: Neuronal Circuits

- Even the simplest actions involve more than one neuron, acting in sequence in a neuronal circuit.
- One of the simplest neuronal circuits is a reflex arc, which may contain just two neurons:
  - The **sensor neuron** detects a stimulus, and communicates an electrical signal to …
  - The **motor neuron**, which activates the muscle.

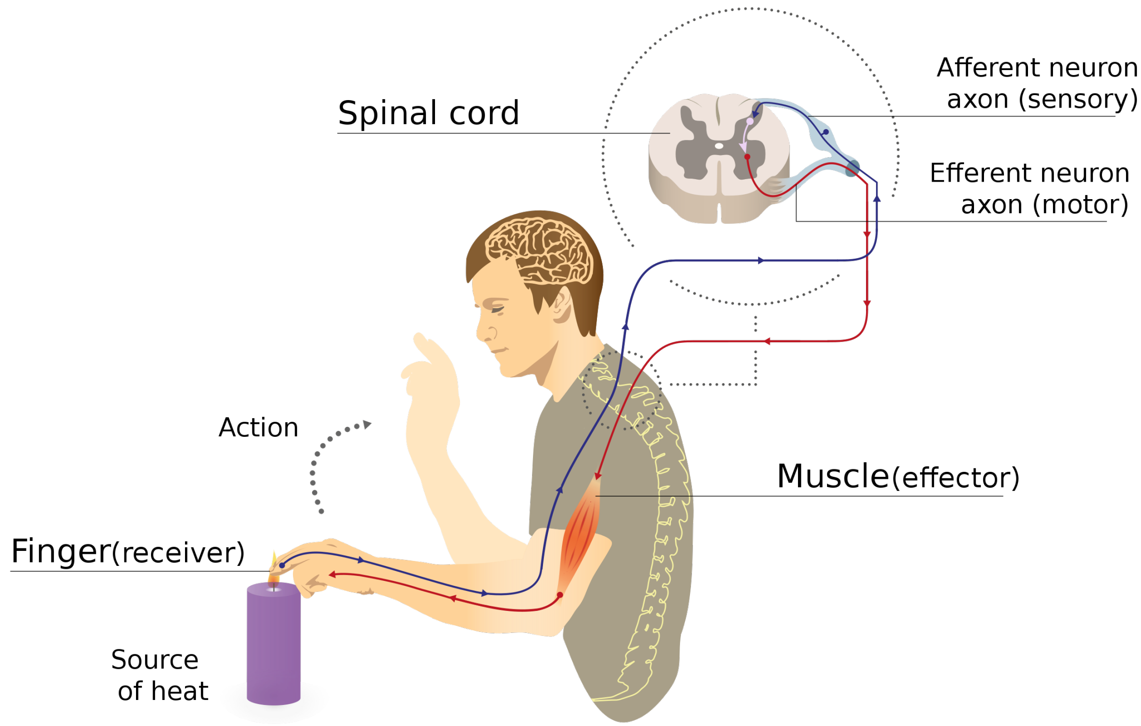


Illustration of a reflex arc: sensor neuron sends a voltage spike to the spinal column, where the resulting current causes a spike in a motor neuron, whose spike activates the muscle.

A McCulloch-Pitts Neuron can compute some logical functions…

When the features are binary ($x_j \in \{0,1\}$), many (but not all!) binary functions can be re-written as linear functions. For example, the function

$$\hat{y} = (x_1 \vee x_2)$$

can be re-written as

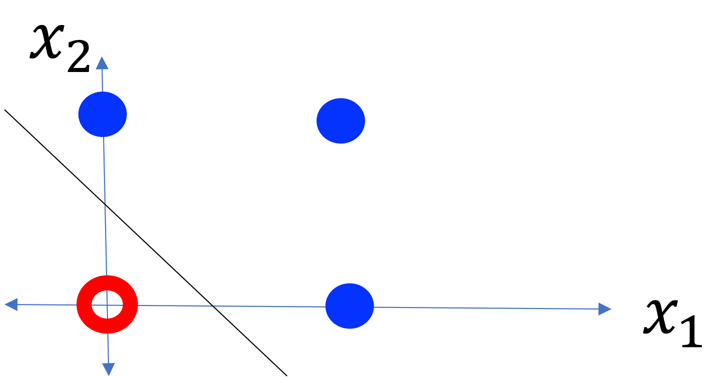
$$\hat{y} = u(x_1 + x_2 - 0.5)$$

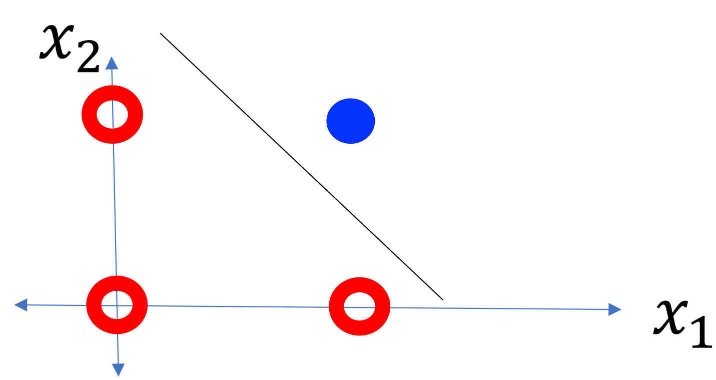
Similarly, the function

$$\hat{y} = (x_1 \wedge x_2)$$

can be re-written as

$$\hat{y} = u(x_1 + x_2 - 1.5)$$

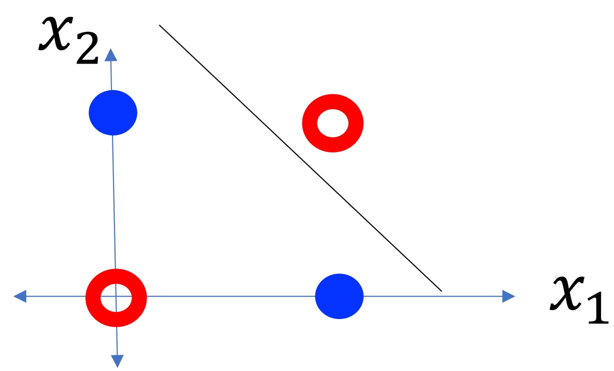




... but not all.

"A linear classifier cannot learn an XOR function."

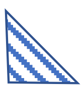
- ...but a ***two-layer neural net*** can compute an XOR function!

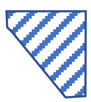


Feature Learning: A way to think about neural nets

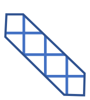
For example, consider the XOR problem.

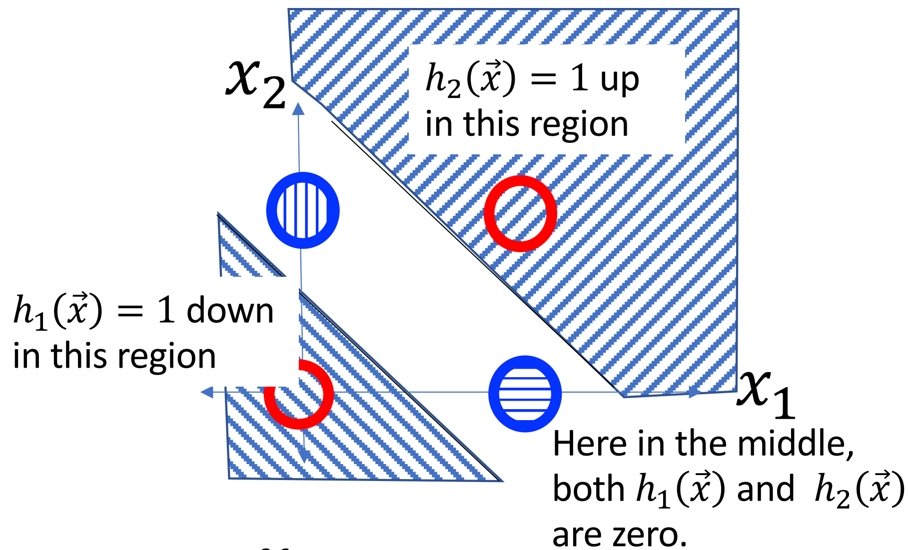
Suppose we create two **hidden nodes**:

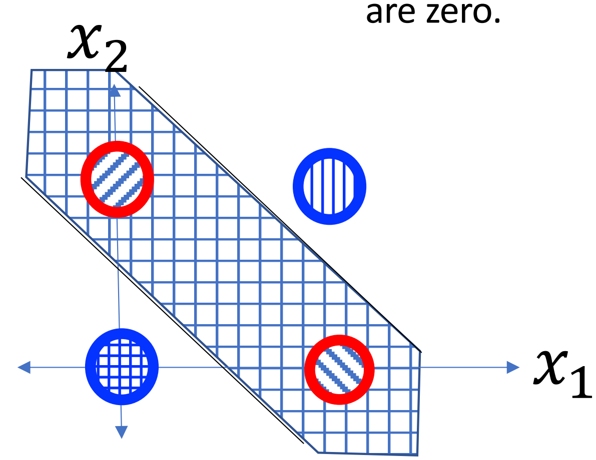
 $h_1(x) = u(0.5 - x_1 - x_2)$

 $h_2(x) = u(x_1 + x_2 - 1.5)$

Then the XOR function $\hat{y} = (x_1 \oplus x_2)$ is given by

 $\hat{y} = u\big(0.5 - h_1(x) - h_2(x)\big)$





Outline

- Breaking the constraints of linearity: multi-layer neural nets
- What's inside a multi-layer neural net?
- Forward-propagation example
- Gradient descent
- Finding the derivative: back-propagation

Multi-layer neural net

- $e_j^{(l)}$ = **excitation** of the j^th neuron (a.k.a. "node") in the l^th layer
  - Computed by adding together inputs from many other neurons, each weighted by a corresponding connection strength or connection weight, $w_{jk}^{(l)}$

- $h_j^{(l)}$ = **activation** of the j^th node in the l^th layer
  - This is computed by just passing the excitation through a scalar nonlinear activation function, thus $h_j^{(l)} = g(e_j^{(l)})$. The activation functions in different layers differ, so to be pedantic, sometimes we'll write $h_j^{(l)} = g^{(l)}\left(e_j^{(l)}\right)$.

Multi-layer neural net

- Given: some training token $x = [x_1, \ldots, x_D, 1]$ and its target label y
- Initialize: $h_k^{(0)} = x_k$
- Forward-propagation: do some magic
- Output: $P(Y = k|x) \; = h_k^{(L)}$

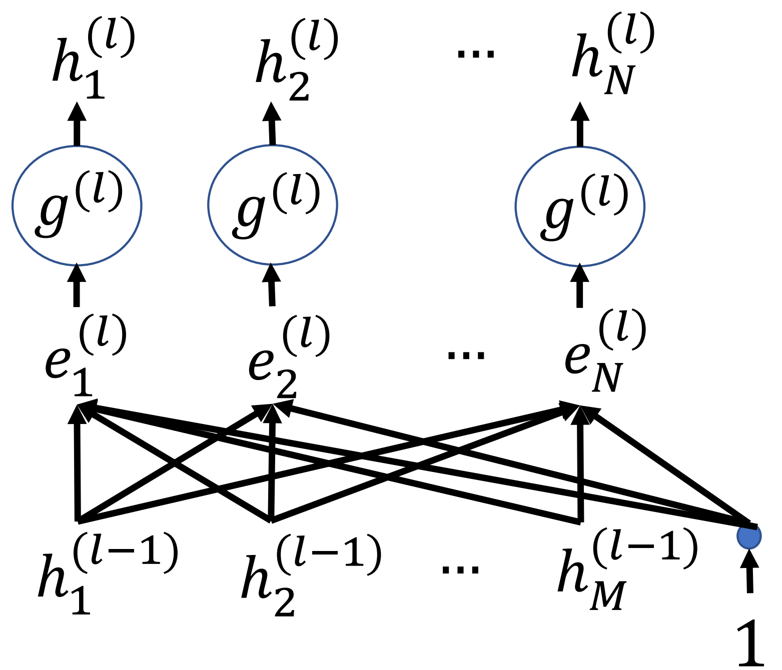
The magical stuff: layers

- From activation to excitation is a matrix multiply:

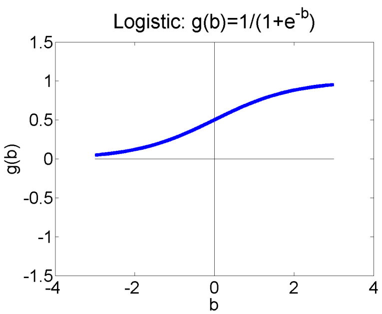
$$e_j^{(l)} = \sum_k w_{jk}^{(l)} h_k^{(l-1)}$$

- From excitation to activation is a scalar nonlinearity:

$$h_j^{(l)} = g^{(l)}\left(e_j^{(l)}\right)$$



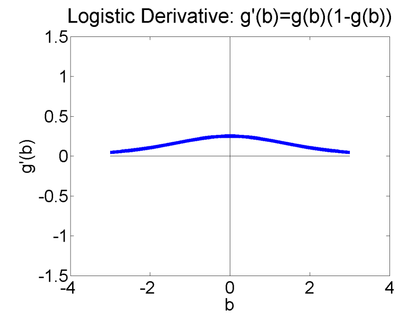
Activation functions

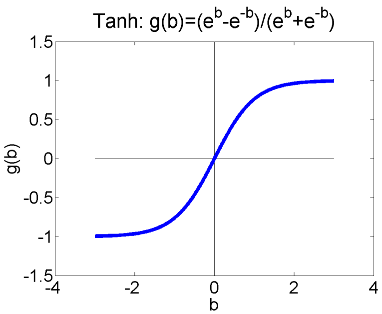


The "activation function," $g^{(l)}(\cdot)$, can be any scalar nonlinearity. For example:

<u>Logistic Sigmoid:</u>

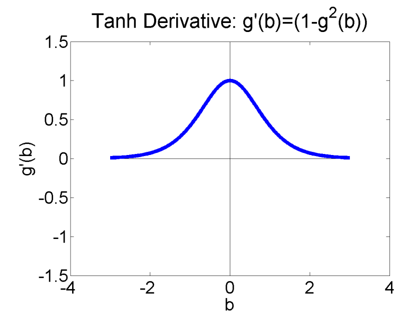
$$\sigma(\beta) = \frac{1}{1+e^{-\beta}}, \qquad \sigma'(\beta) = \sigma(\beta)\big(1-\sigma(\beta)\big)$$

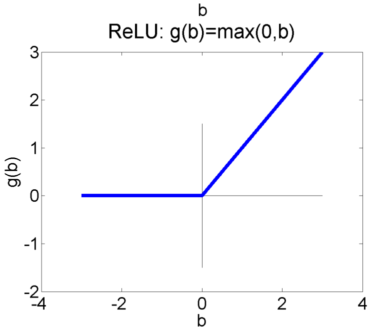




<u>Hyperbolic Tangent (tanh):</u>

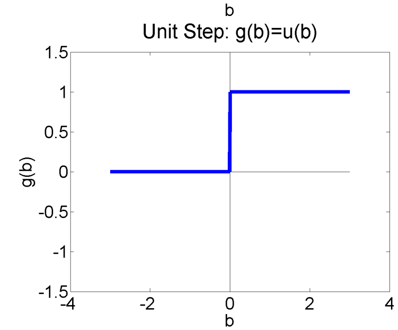
$$\tanh(\beta) = \frac{e^{\beta}-e^{-\beta}}{e^{\beta}+e^{-\beta}}, \tanh'(\beta) = 1-\tanh^2(\beta)$$





<u>Rectified Linear Unit (ReLU):</u>

$$\mathrm{ReLU}(\beta) = \max(0,\beta), \qquad \mathrm{ReLU}'(\beta) = \mathrm{u}(\beta)$$

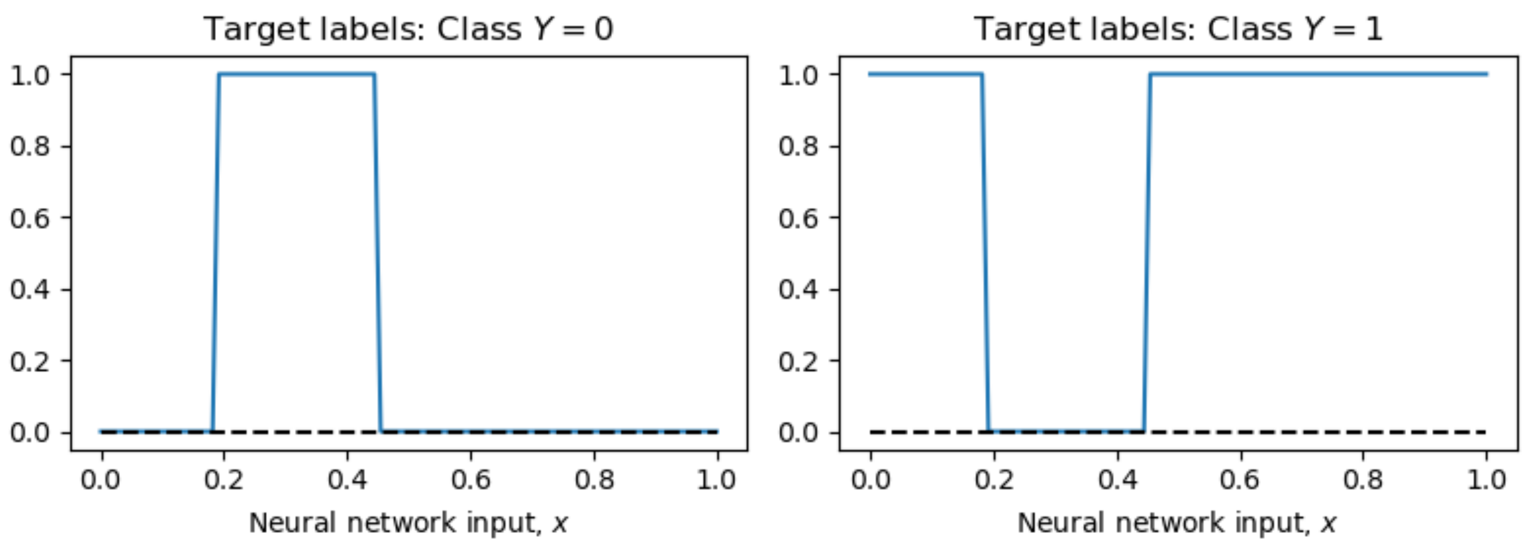


Outline

- Breaking the constraints of linearity: multi-layer neural nets
- What's inside a multi-layer neural net?
- Forward-propagation example
- Gradient descent
- Finding the derivative: back-propagation

Example

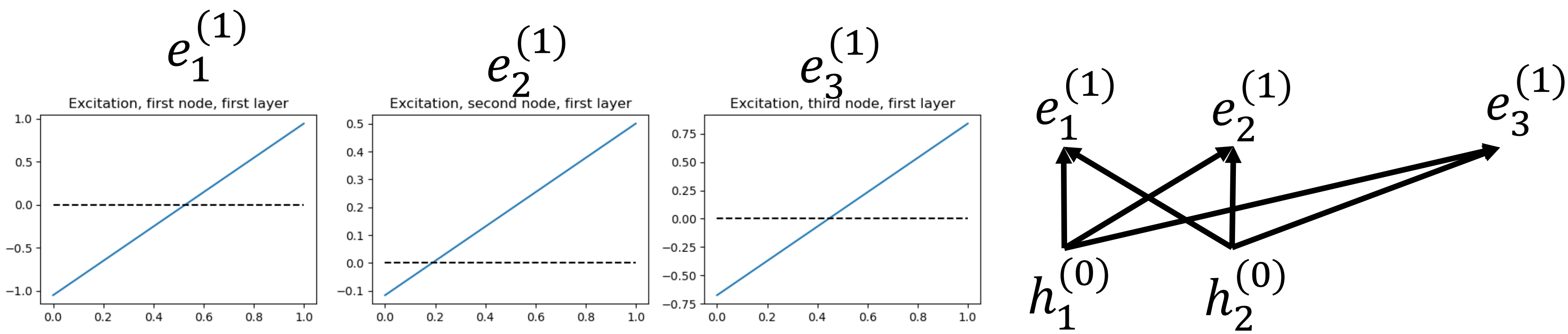
- Suppose x =scalar
- $Y \in \{0,1\}$



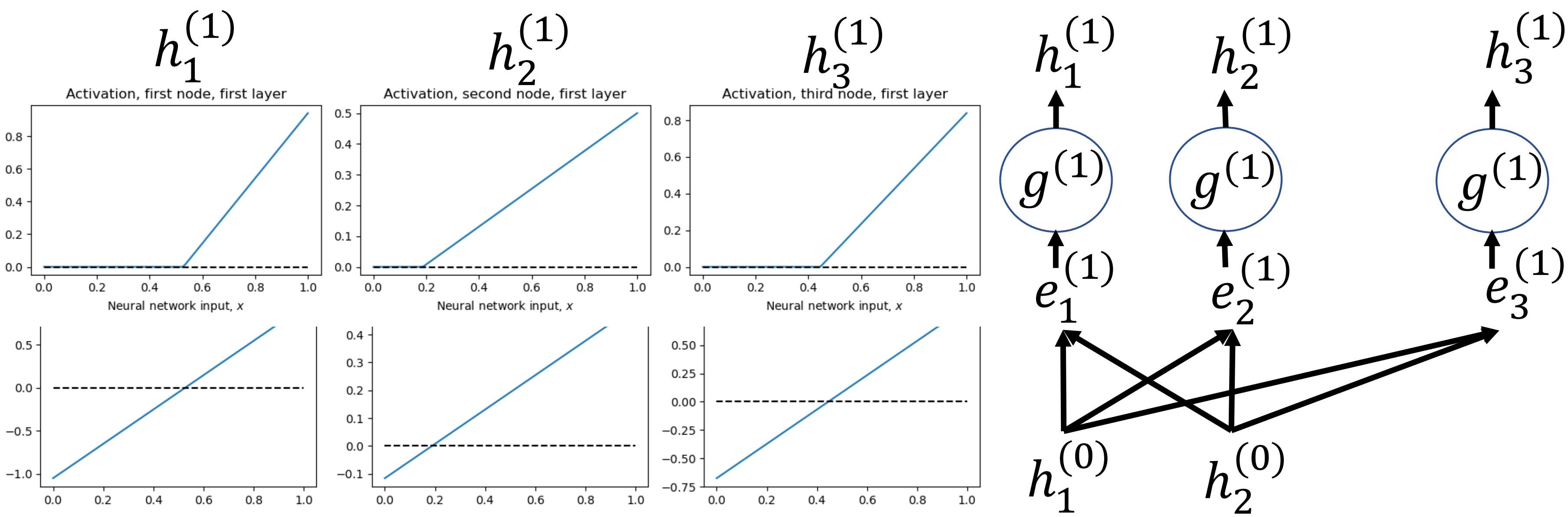
Initialize

$$h_1^{(0)} = x, \qquad h_2^{(0)} = 1$$

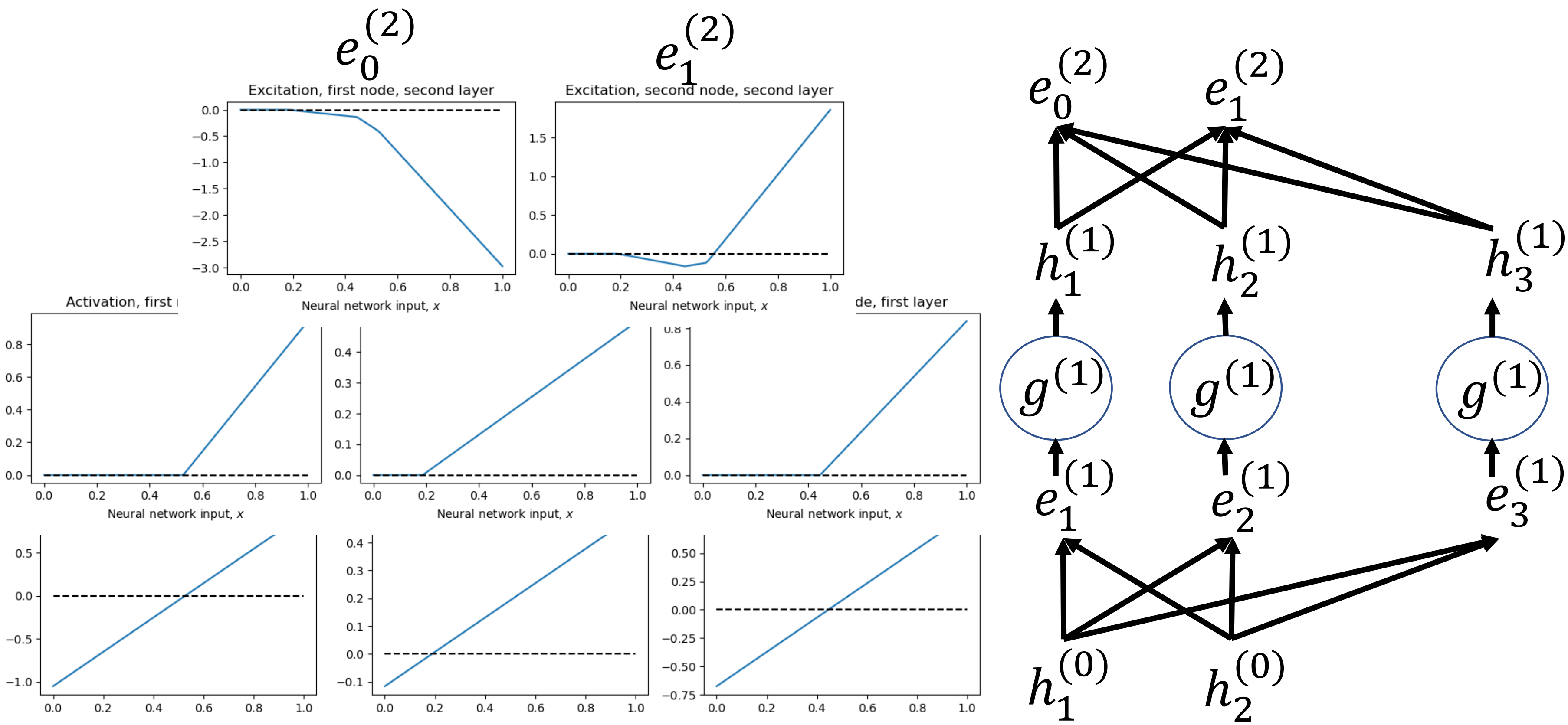
$$e_j^{(1)} = \sum_k w_{jk}^{(1)} h_k^{(0)}$$



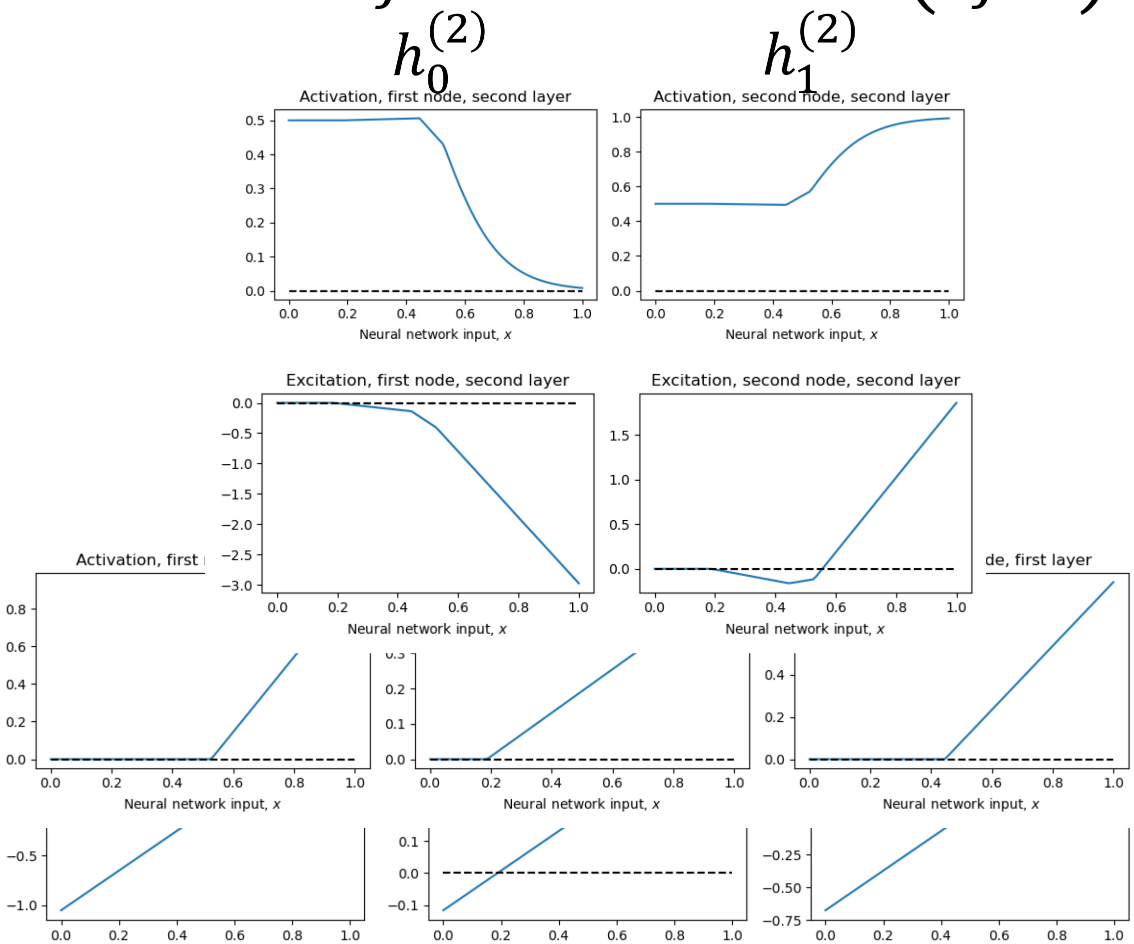
Excitation to Activation: $h_j^{(1)} = \mathrm{ReLU}\left(e_j^{(1)}\right)$

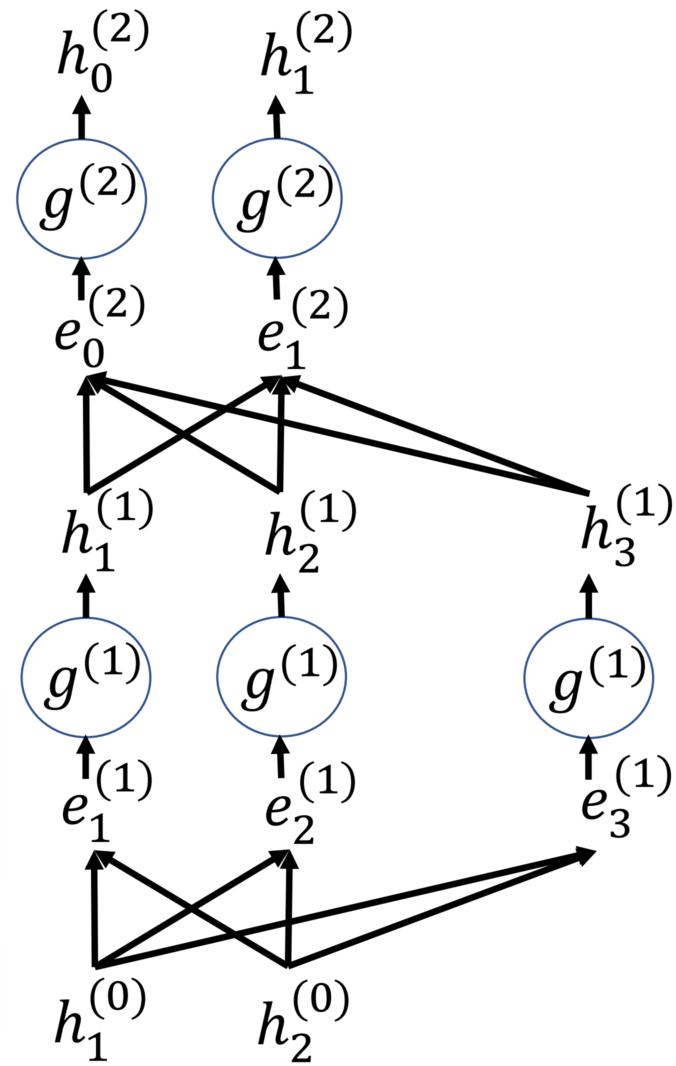


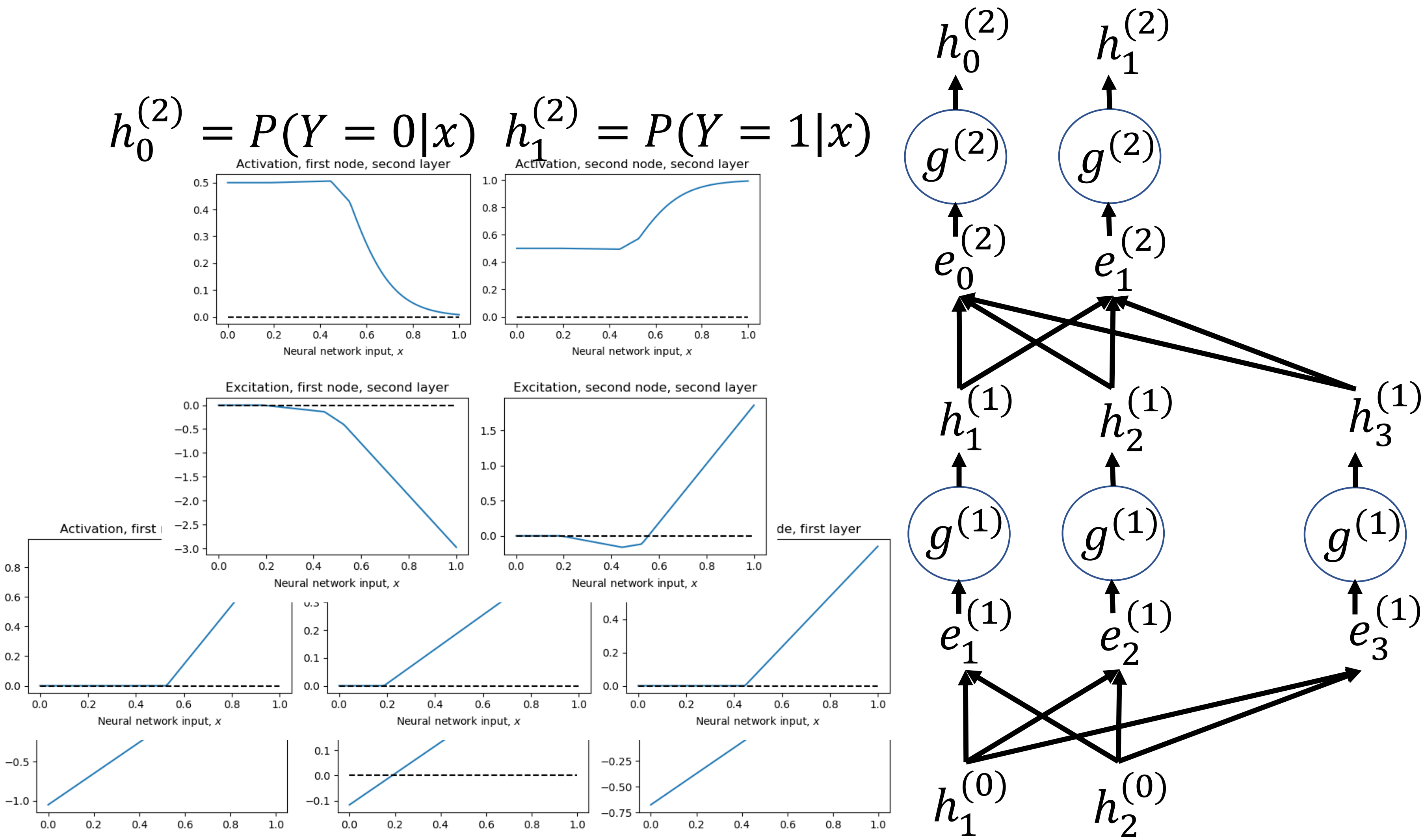
Activation to Excitation: $e_j^{(2)} = \sum_k w_{jk}^{(2)} h_k^{(1)}$



Output: $h_j^{(2)} = \mathrm{softmax}\left(e_j^{(2)}\right)$







$h_0^{(2)} = P(Y = 0|x)$
$h_1^{(2)} = P(Y = 1|x)$
Activation, first node, second layer
Activation, second node, second layer
Excitation, first node, second layer
Excitation, second node, second layer
Activation, first
de, first layer
Neural network input, x
$h_0^{(2)}$ $h_1^{(2)}$
$g^{(2)}$ $g^{(2)}$
$e_0^{(2)}$ $e_1^{(2)}$
$h_1^{(1)}$ $h_2^{(1)}$ $h_3^{(1)}$
$g^{(1)}$ $g^{(1)}$ $g^{(1)}$
$e_1^{(1)}$ $e_2^{(1)}$ $e_3^{(1)}$
$h_1^{(0)}$ $h_2^{(0)}$

Outline

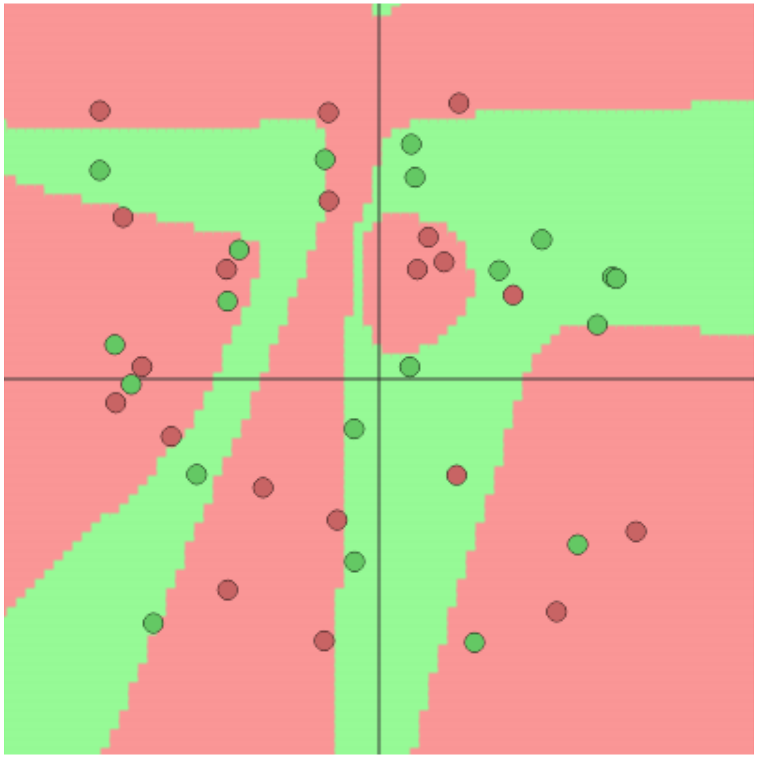
- Breaking the constraints of linearity: multi-layer neural nets
- What’s inside a multi-layer neural net?
- Forward-propagation example
- Gradient descent
- Finding the derivative: back-propagation

Gradient descent: basic idea

- Suppose we have a training token, x.
- Its target label is y.
- The neural net produces output $\hat{y}$, which is not y.
- The difference between y and $\hat{y}$ is summarized by some loss function, $\mathcal{L}(y, \hat{y})$.
- The output of the neural net is determined by some parameters, $w_{jk}^{(l)}$.
- Then we can improve the network by setting:

$$w_{jk}^{(l)} \leftarrow w_{jk}^{(l)} - \eta \frac{d\mathcal{L}}{dw_{jk}^{(l)}}$$

Visualizing gradient descent



https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

Outline

- Breaking the constraints of linearity: multi-layer neural nets
- What's inside a multi-layer neural net?
- Forward-propagation example
- Gradient descent
- Finding the derivative: back-propagation

Finding the derivative

- OK, how do we find $\frac{d\mathcal{L}}{dw_{jk}^{(l)}}$?
- Well, the only way in which $\mathcal{L}$ depends on $w_{jk}^{(l)}$ is by way of $e_j^{(l)}$: $e_j^{(l)} = \sum_k w_{jk}^{(l)} h_k^{(l-1)}$. So we could use the chain rule of calculus:

$$\frac{d\mathcal{L}}{dw_{jk}^{(l)}} = \frac{d\mathcal{L}}{de_j^{(l)}} \times \frac{de_j^{(l)}}{dw_{jk}^{(l)}} = \frac{d\mathcal{L}}{de_j^{(l)}} h_k^{(l-1)}$$

- So we need to forward-propagate from x, to find $h_k^{(l-1)}$
- Then we back-propagate, from y, to find $\frac{d\mathcal{L}}{de_j^{(l)}}$
- Then we multiply those two things.

Finding the derivative

- Well… how do we find $\frac{d\mathcal{L}}{de_j^{(l)}}$?
- Well, the only way in which $\mathcal{L}$ depends on $e_j^{(l)}$ is by way of $h_j^{(l)}$: $h_j^{(l)} = g(e_j^{(l)})$. So we could use the chain rule of calculus:

$$\frac{d\mathcal{L}}{de_j^{(l)}} = \frac{d\mathcal{L}}{dh_j^{(l)}} \times \frac{dh_j^{(l)}}{de_j^{(l)}} = \frac{d\mathcal{L}}{dh_j^{(l)}} g'(e_j^{(l)})$$

- So we need to forward-propagate from x, to find $g(e_j^{(l)})$, and then we look up its derivative in a table, to find $g'(e_j^{(l)})$.
- Then we back-propagate, from y, to find $\frac{d\mathcal{L}}{dh_j^{(l)}}$
- Then we multiply those two things.

Finding the derivative

- OK, great! Then how do we find $\frac{d\mathcal{L}}{dh_j^{(l)}}$?
- Well, the only way in which $\mathcal{L}$ depends on $h_j^{(l)}$ is by way of all of the different nodes in layer l+1: $e_k^{(l+1)} = \sum_j w_{kj}^{(l+1)} h_j^{(l)}$. So we could use the chain rule of calculus:

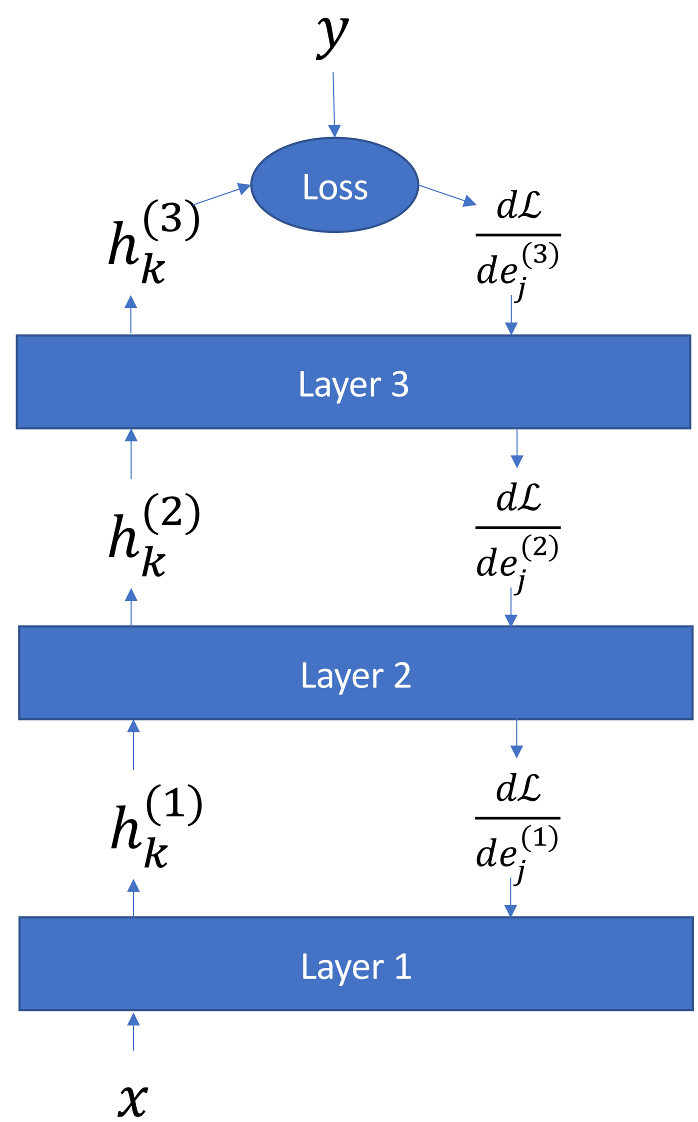
$$\frac{d\mathcal{L}}{dh_j^{(l)}} = \sum_k \frac{d\mathcal{L}}{de_k^{(l+1)}} \times \frac{de_k^{(l+1)}}{dh_j^{(l)}} = \sum_k \frac{d\mathcal{L}}{de_k^{(l+1)}} w_{kj}^{(l+1)}$$

- So we back-propagate, from y, to find $\frac{d\mathcal{L}}{de_k^{(l+1)}}$
- Then we multiply each of those by the corresponding weight, $w_{kj}^{(l+1)}$, and add them up.

Finding the derivative

- Forward propagate, from x, to find $h_k^{(l-1)}$ in each layer
- Back-propagate, from y, to find $\frac{d\mathcal{L}}{de_j^{(l)}}$ in each layer
- Multiply them to get $\frac{d\mathcal{L}}{dw_{jk}^{(l)}}$, then

$$w_{jk}^{(l)} \leftarrow w_{jk}^{(l)} - \eta \frac{d\mathcal{L}}{dw_{jk}^{(l)}}$$



Gradient descent

For example, suppose $\mathcal{L} = -\ln P(Y = y|x) = -\ln h_y^{(L)}$, and the nonlinearity is $h_j^{(L)} = \text{softmax}\left(e_j^{(L)}\right)$. Then we have this derivative, from last time:

$$\frac{d\left(-\ln h_y^{(L)}\right)}{de_j^{(L)}} = \begin{cases} \left(\dfrac{\exp\left(e_j^{(L)}\right)}{\sum_{k=0}^{V-1} \exp\left(e_k^{(L)}\right)} - 1\right) & j = y \\ \left(\dfrac{\exp\left(e_j^{(L)}\right)}{\sum_{k=0}^{V-1} \exp\left(e_k^{(L)}\right)} - 0\right) & j \neq y \end{cases}$$

Back-propagation

- Back-propagating excitation back to activation:

$$\frac{d\mathcal{L}}{dh_k^{(l)}} = \sum_j w_{jk}^{(l+1)} \frac{d\mathcal{L}}{de_j^{(l+1)}}$$

- Back-propagating activation back to excitation:

$$\frac{d\mathcal{L}}{de_k^{(l)}} = \frac{d\mathcal{L}}{dh_k^{(l)}} g^{(l)\prime}\left(e_k^{(l)}\right)$$

Gradient descent to minimize loss

$$w_{jk}^{(l)} \leftarrow w_{jk}^{(l)} - \eta \frac{d\mathcal{L}}{dw_{jk}^{(l)}} = w_{jk}^{(l)} - \eta \frac{d\mathcal{L}}{de_j^{(l)}} h_k^{(l-1)}$$

Outline

- Breaking the constraints of linearity: multi-layer neural nets
- What’s inside a multi-layer neural net?
- Forward-propagation example
- Gradient descent
- Finding the derivative: back-propagation