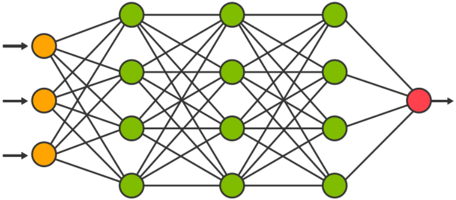
Introduction to Neural Networks

ML Instruction Team, Fall 2022

CE Department
Sharif University of Technology



Biological Analogy

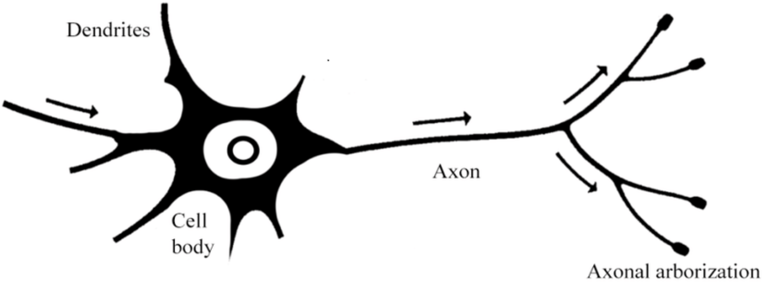


Figure: Anatomy of a biological neuron [1].

Activation Functions

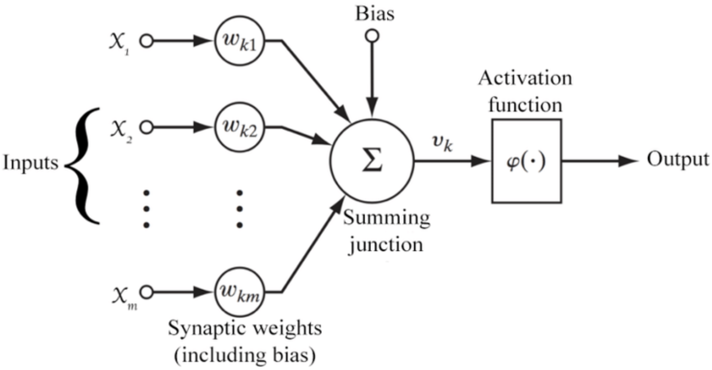


Figure: Neural network neuron [1].

Activation Functions

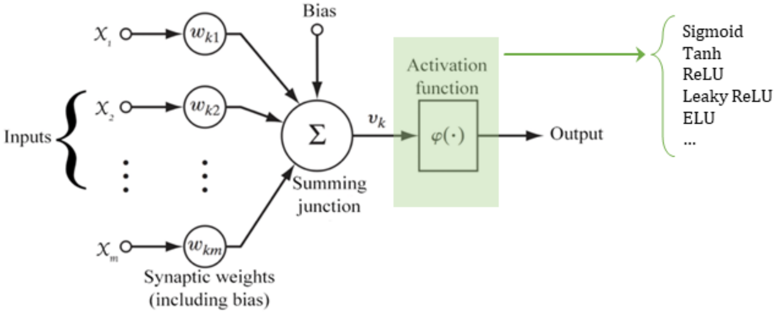
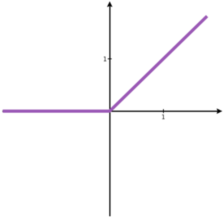
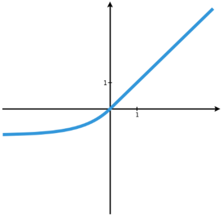
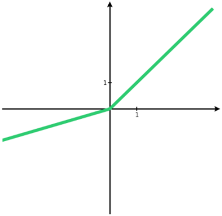
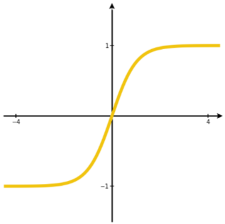
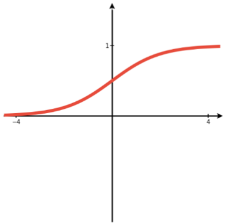
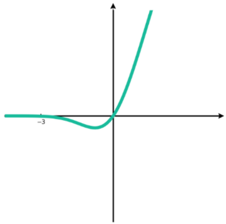


Figure: Activation function

Activation Functions

ReLU	ELU	Leaky ReLU
$f(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases}$	$f(x) = \begin{cases} x & x > 0 \\ \alpha(e^x - 1) & x \leq 0 \end{cases}$	$f(x) = \begin{cases} x & x \geq 0 \\ 0.01x & x < 0 \end{cases}$
		

Activation Functions

Tanh	Sigmoid	GELU
$f(x) = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$	$\sigma(x) = \dfrac{1}{1 + e^{-x}}$	$f(x) = \dfrac{1}{2}x\left(1 + \mathrm{erf}\left(\dfrac{x}{\sqrt{2}}\right)\right)$
		

Softmax

$$f(x) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}} \quad i = 1, \cdots, J$$

Gradient Descent

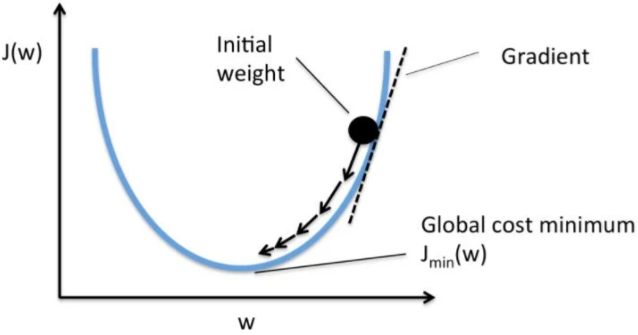


Figure: Gradient descent [3].

Gradient Descent

- Let's define our problem:
  - ▷ We have dataset $\mathcal{D} = \{x^i, y^{(i)}\}_{i=1}^{n}$.
  - ▷ f is a single layer perceptron.
  - ▷ Define $\hat{y}^{(i)} = f(x^{(i)})$.
- We want to minimize following cost function:

$$\mathcal{J}(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2$$

- We are going to use gradient descent algorithm. $\boldsymbol{w}$ will be updated as follows:

$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^{t} - \eta\nabla_{\boldsymbol{w}}\mathcal{J}$$

Gradient Descent

- Let's find $\nabla_{\boldsymbol{w}}\mathcal{J}$:

$$\begin{aligned}\frac{\partial J}{\partial w_j} &= \frac{\partial}{\partial w_j}\frac{1}{2}\sum_i (y^{(i)} - \hat{y}^{(i)})^2 \\ &= \frac{1}{2}\sum_i \frac{\partial}{\partial w_j}(y^{(i)} - \hat{y}^{(i)})^2 \\ &= \frac{1}{2}\sum_i 2(y^{(i)} - \hat{y}^{(i)})\frac{\partial}{\partial w_j}(y^{(i)} - \hat{y}^{(i)}) \\ &= \sum_i (y^{(i)} - \hat{y}^{(i)})\frac{\partial}{\partial w_j}\left(y^{(i)} - \sum_j w_j x_j^{(i)}\right) \\ &= \sum_i (y^{(i)} - \hat{y}^{(i)})(-x_j^{(i)})\end{aligned}$$

$$\Delta w_j = -\eta\frac{\partial J}{\partial w_j} = -\eta\sum_i (y^{(i)} - \hat{y}^{(i)})(-x_j^{(i)}) = \eta\sum_i (y^{(i)} - \hat{y}^{(i)})x_j^{(i)}$$

$$\mathbf{w} := \mathbf{w} + \Delta\mathbf{w}$$

Limitations of SLP

- What are the limitations of SLP?
- Can we learn all functions with SLP?

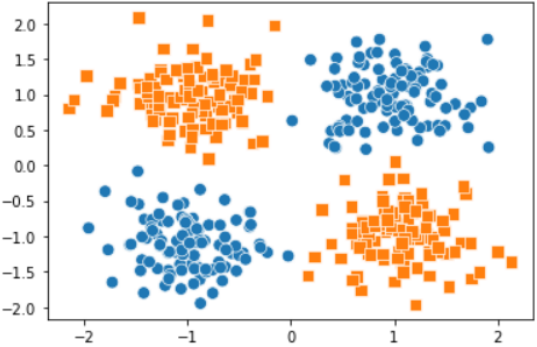
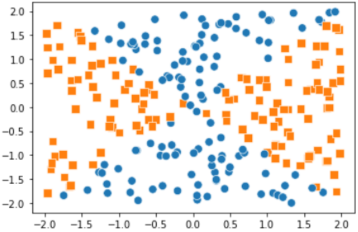


Figure: The XOR function is not linear separable.

Limitations of SLP

- As we saw in the XOR case, nonlinear separable functions can not be learned by SPLs.



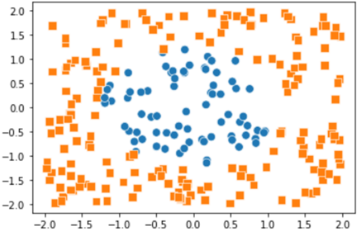


Figure: Examples of nonlinear separable functions.

- How to solve this?

Limitations of SLP

- What if we knew some feature space which our data is linear separable in?!

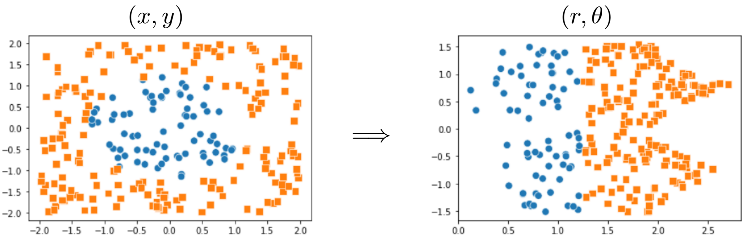


Figure: Data is linear separable after transformation.

Multi-Layer Perceptron

- So if we know some $f_1, \cdots, f_4$ we can use SLP to solve problem.

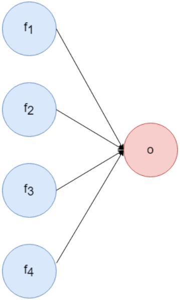


Figure: Using feature space f to solve problem.

- How to learn this f_is? Use SLP!

Multi-Layer Perceptron

- We can use inputs (x_i) to learn features (f_i)

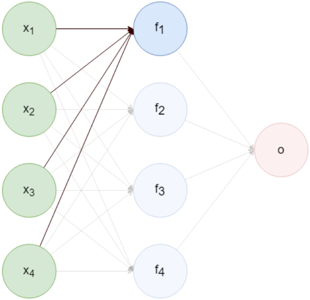


Figure: Using inputs to learn features.

- What if f_is are not sufficient? We can add more layers!

Multi-Layer Perceptron

- Adding more layers we will have a bigger network.

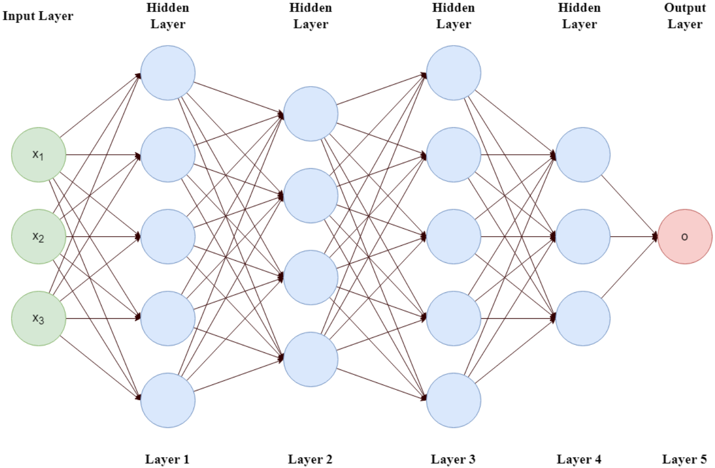


Figure: A 5 layer MLP.

Architecture of MLPs

- Important questions:
  - How many hidden layer should we have?
  - In each hidden layer, how many neuron should we have?

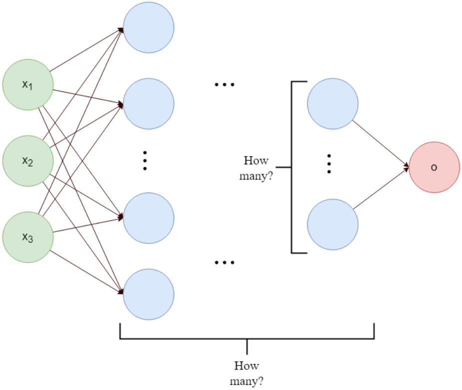


Figure: How many layers and neurons should we have?

Architecture of MLPs

- In practice:
  - You have limited resources
  - There is no universal rule to choose this hyperparameters
  - Need to experiment for different number of layers and neurons in each layer

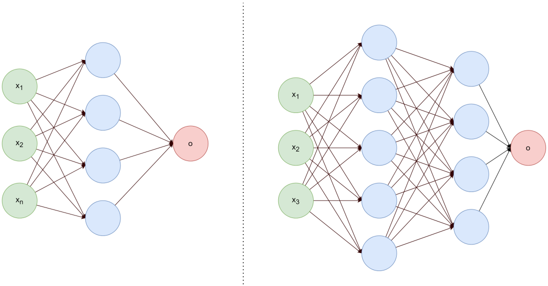


Figure: Experiment for different architecture and choose the best model.

Activation Function of Hidden Layers

- One can use any activation function for each hidden units
- Usually people use the same activation function for all neurons in one layer
- The important point is to use nonlinear activation functions

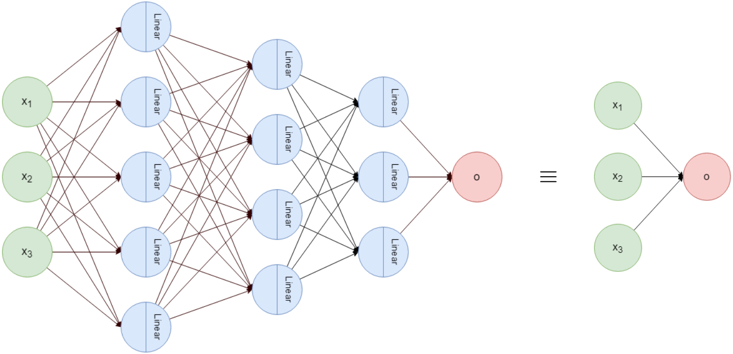


Figure: MLP with linear activation functions is equivalent to simple SLP.

XOR problem

- Now let's solve XOR problem with MLPs.
- We have two binary inputs, build an MLP to calculate their **XOR**.

- First let's build logical **AND** and **OR** functions.

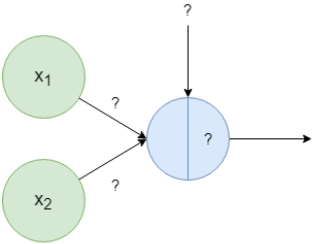
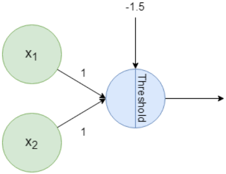
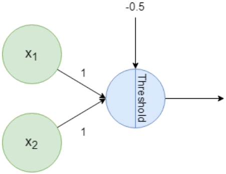


Figure: We need to find weights, biases and activation function.

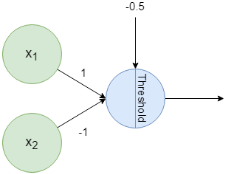
XOR problem



(a) $x_1 \wedge x_2$

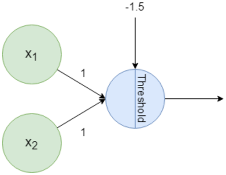


(b) $x_1 \vee x_2$

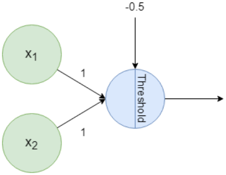


(c) $x_1 \wedge \overline{x_2}$

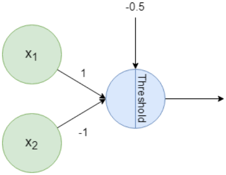
XOR problem



(a) $x_1 \wedge x_2$



(b) $x_1 \vee x_2$



(c) $x_1 \wedge \overline{x_2}$

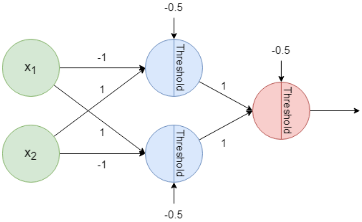
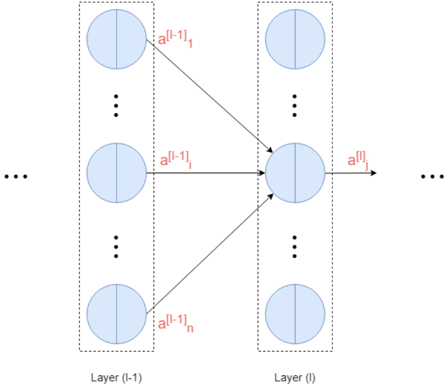


Figure: MLP for XOR function.

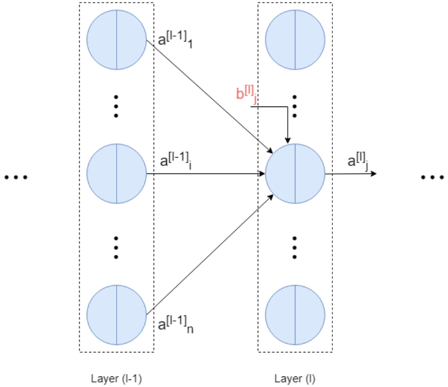
MLP notation

- $a_i^{[l]}$: i-th neuron outpu in layer l
- $\boldsymbol{a}^{[l]}$: layer l output in vector form



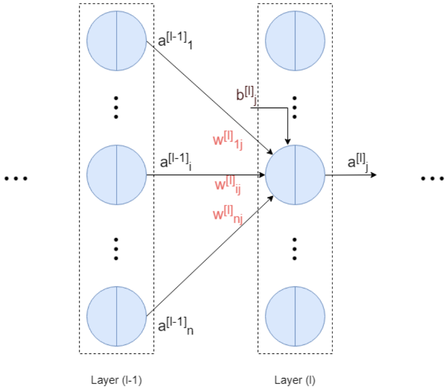
MLP notation

- $b_i^{[l]}$: i-th neuron biase in layer l
- $\boldsymbol{b}^{[l]}$: layer l biases in vector form



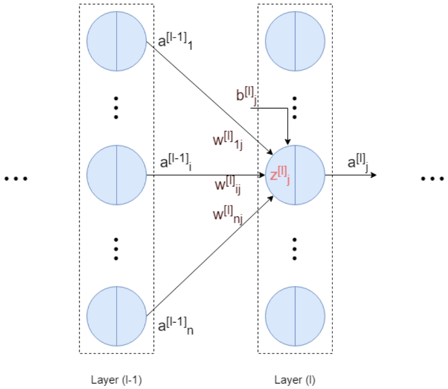
MLP notation

- $W_{ij}^{[l]}$: weight of the edge between i-th nuron in layer $l-1$ and j-th neuron in layer l



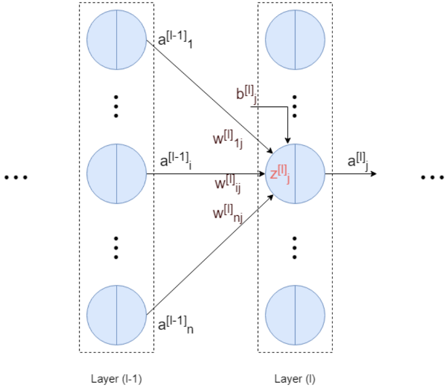
MLP notation

- $z_j^{[l]}$: j-th neuron input in layer l
- $z_j^{[l]} = b_j^{[l]} + \sum_{i=1}^{n} W_{ij}^{[l]} a_i^{[l-1]}$



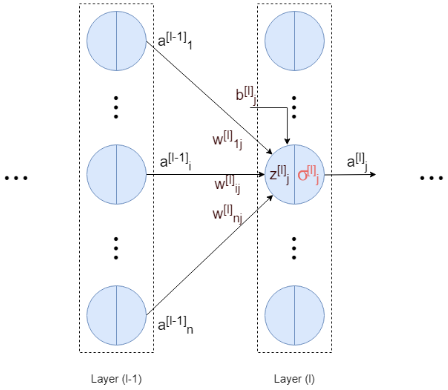
MLP notation

- $\boldsymbol{z}^{[l]}$: input of layer l in vector form
- $\boldsymbol{z}^{[l]} = \boldsymbol{b}^{[l]} + (W^{[l]})^T \boldsymbol{a}^{[l-1]}$



MLP notation

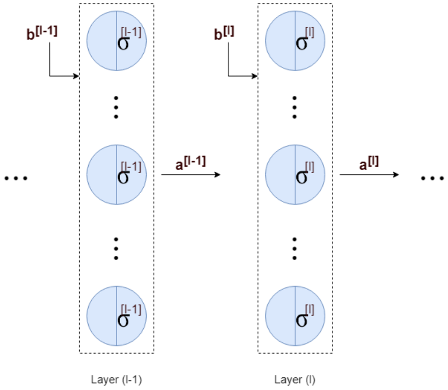
- $\sigma_j^{[l]}$: j-th neuron activation function in layer l



MLP notation

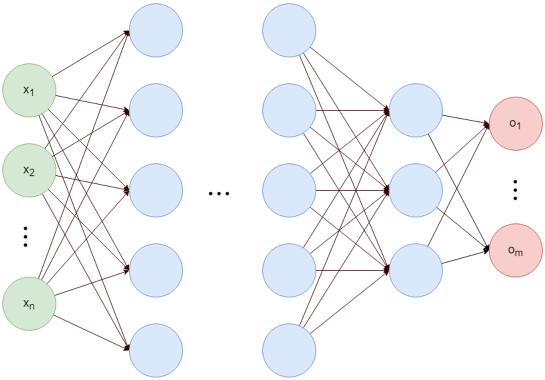
- If we assume all neurons in one layer have the same activation function then:

$$\boldsymbol{a}^{[l]} = \sigma^{[l]}\left(\boldsymbol{b}^{[l]} + (W^{[l]})^T \boldsymbol{a}^{[l-1]}\right)$$



MLP notation

- So for a network with L layer, and $\boldsymbol{x}$ as its input we will have:

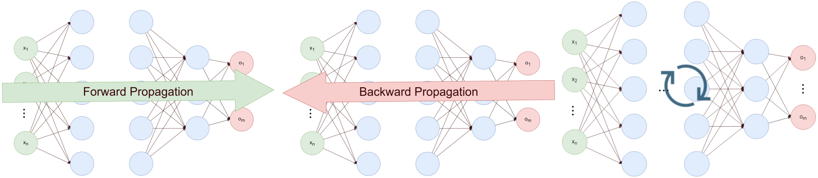


$$\boldsymbol{o} = \boldsymbol{a}^{[L]} = \sigma^{[L]}\left(\boldsymbol{b}^{[L]} + (W^{[L]})^T \sigma^{[L-1]}\left(\cdots \sigma^{[1]}\left(\boldsymbol{b}^{[1]} + (W^{[1]})^T \boldsymbol{x}\right)\cdots\right)\right)$$

Learning MLPs

- Till here we have used networks with predefined weights and biases.
- How to learn these parameters?

- The idea is to use gradient descent



(a) Forward pass (b) Backward pass (c) Update parameters

Learning MLPs

- Let's define the learning problem more formal:
  - $\{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$: dataset
  - f: network
  - W: all weights and biases of the network ($W^{[l]}$ and $\boldsymbol{b}^{[l]}$ for different l)
  - L: loss function

- We want to find W^* which minimizes following cost function:

$$\mathcal{J}(W) = \sum_{i=1}^{n} L\left(f(x^{(i)}; W), y^{(i)}\right)$$

- We are going to use gradient descent, so we need to find $\nabla_W \mathcal{J}$.

Forward Propagation

- First of all we need to find loss value.
- It only requires to know the inputs of each neuron.

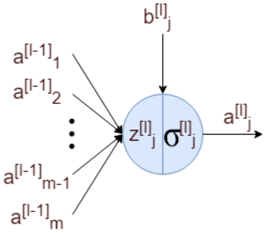


Figure: $a_j^{[l]} = \sum_{i=1}^{m} W_{ij}^{[l]} a_i^{[l-1]} + b_j^{[l]}$

- So we can calculate these outputs layer by layer.

Forward Propagation

- After forward pass we will know:
  - Loss value
  - Network output
  - Middle values

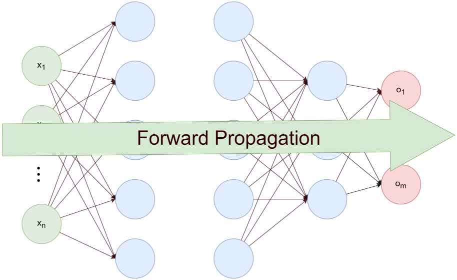


Figure: Forward pass

Backward Propagation

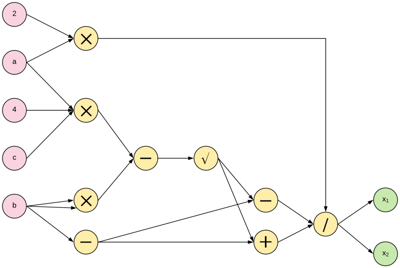
- Now we need to calculate $\nabla_W \mathcal{J}$.
- First idea:
  - Use analytical approach.
  - Write down derivatives on paper.
  - Find the close form of $\nabla_W \mathcal{J}$ (if it is possible to do so).
  - Implement this gradient as a function to work with.

  - Pros:
    - Fast
    - Exact

  - Cons:
    - Need to rewrite calculation for different architectures

Backward Propagation

- Second idea:
  - Using modular approach.
  - Computing the cost function consists of doing many operations.
  - We can build a computation graph for this calculation.
  - Each node will represent a single operation.

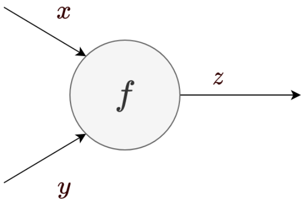


$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Figure: An example of computational graph, Source.

Backward Propagation

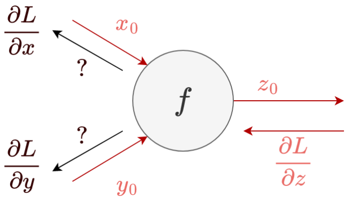
- In this approach if we know how to calculate gradient for single node or module, then we can find gradient with respect to each variables.
- Let's say we have a module as follow:



- It gets x and y as its input and returns $z = f(x, y)$ as its output.
- How to calculate derivative of loss with respect to module inputs?

Backward Propagation

- We know:
  - Module output for x_0 and y_0, let's call it z_0.
  - Gradient of loss with respect to module output at z_0, $\left(\frac{\partial L}{\partial z}\right)$.
- We want:
  - Gradient of loss with respect to module inputs at x_0 and y_0, $\left(\frac{\partial L}{\partial x}, \frac{\partial L}{\partial y}\right)$.

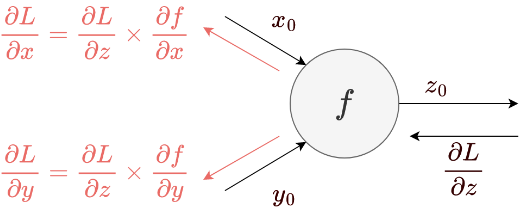


Backward Propagation

- We can use chain rule to do so.

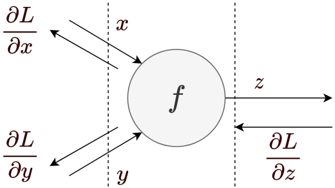
Chain rule:

$$\left.\begin{aligned} z &= f(x, y) \\ L &= L(z) \end{aligned}\right\} \implies \frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \times \frac{\partial z}{\partial x}$$



Backward Propagation

- Backpropagation for single module:



Downstream gradient	=	Local gradient	×	Upstream gradient
$\frac{\partial L}{\partial x}$	$=$	$\frac{\partial f}{\partial x}$	$\times$	$\frac{\partial L}{\partial z}$
$\frac{\partial L}{\partial y}$	$=$	$\frac{\partial f}{\partial y}$	$\times$	$\frac{\partial L}{\partial z}$

Backward Propagation: Example

- Let's solve a simple example using backpropagation.
- We have $f(x, y, z) = \frac{x^2 y}{z}$.
- Find $\frac{\partial f}{\partial x}$, $\frac{\partial f}{\partial y}$ and $\frac{\partial f}{\partial z}$ at $x = 3$, $y = 4$ and $z = 2$.

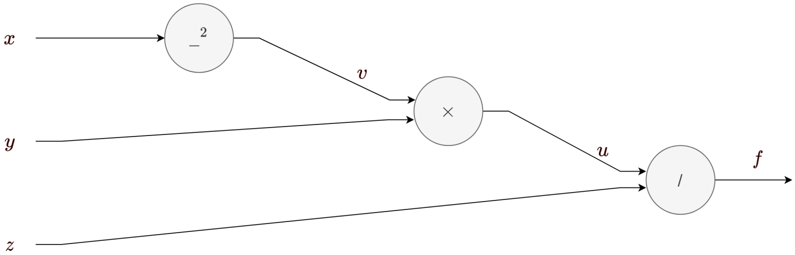


Figure: Computational graph of f.

Backward Propagation: Example

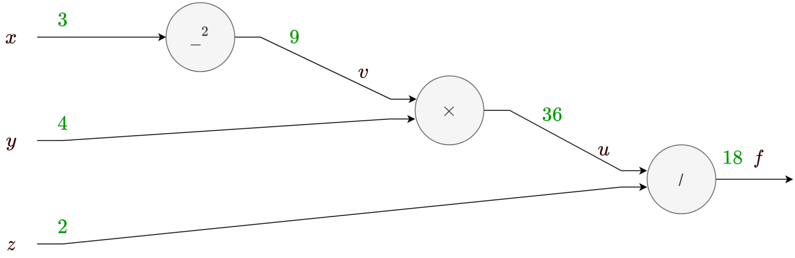
- First let's find gradient analytically.
- We have:

$$\begin{cases} v = x^2 \\ u = vy \\ f = \frac{u}{z} \end{cases} \qquad \begin{cases} \frac{\partial f}{\partial z} = -\frac{u}{z^2} \\ \frac{\partial f}{\partial y} = \frac{\partial u}{\partial y} \times \frac{\partial f}{\partial u} = v \times \frac{1}{z} = \frac{v}{z} \\ \frac{\partial f}{\partial x} = \frac{\partial v}{\partial x} \times \frac{\partial u}{\partial v} \times \frac{\partial f}{\partial u} = 2x \times y \times \frac{1}{z} = \frac{2xy}{z} \end{cases}$$

$$(x = 3, y = 4, z = 2) \implies \begin{cases} v = 9 \\ u = 36 \\ f = 18 \end{cases} \implies \begin{cases} \frac{\partial f}{\partial z} = -9 \\ \frac{\partial f}{\partial y} = 4.5 \\ \frac{\partial f}{\partial x} = 12 \end{cases}$$

Backward Propagation: Example

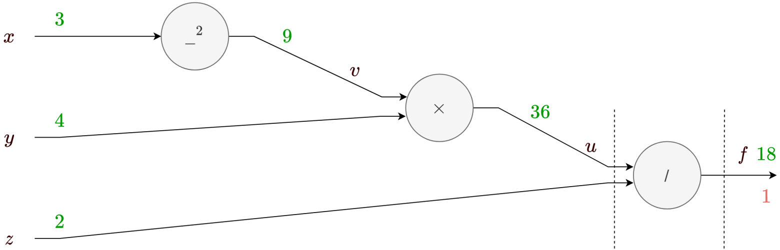
- Now let's use backpropagation.
- First we do forward propagation.



- Second we will do backpropagation for each module.

Backward Propagation: Example

- Backpropagation for / module:



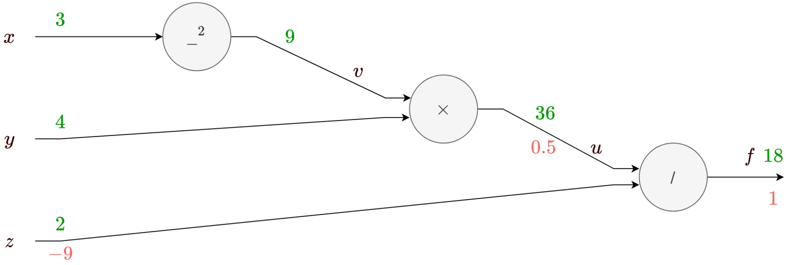
$$\begin{cases} \frac{\partial f}{\partial u} = \frac{1}{z} = 0.5 \\ \frac{\partial f}{\partial z} = -\frac{u}{z^2} = -9 \end{cases} \qquad \begin{cases} \frac{\partial f}{\partial f} = 1 \end{cases}$$

$$\frac{\partial f}{\partial u} = \frac{\partial f}{\partial u} \times \frac{\partial f}{\partial f}$$

$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial z} \times \frac{\partial f}{\partial f}$$

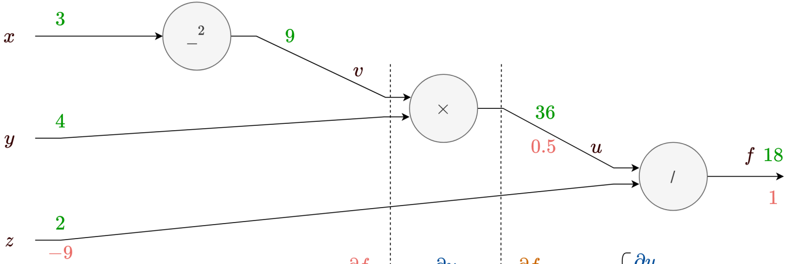
Backward Propagation: Example

- Backpropagation for / module:



Backward Propagation: Example

- Backpropagation for $\times$ module:



$$\left\{ \frac{\partial f}{\partial u} = 0.5 \right.$$

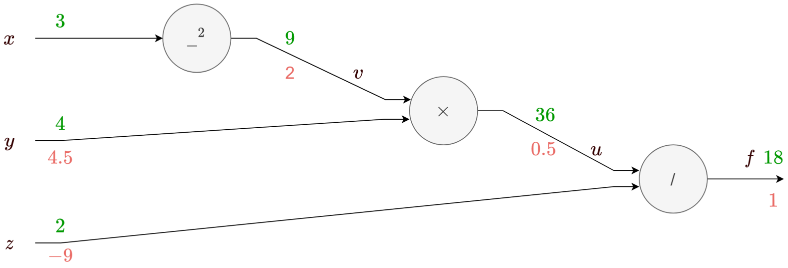
$$\frac{\partial f}{\partial v} = \frac{\partial u}{\partial v} \times \frac{\partial f}{\partial u}$$

$$\frac{\partial f}{\partial y} = \frac{\partial u}{\partial y} \times \frac{\partial f}{\partial u}$$

$$\begin{cases} \dfrac{\partial u}{\partial v} = y = 4 \\ \dfrac{\partial u}{\partial y} = v = 9 \end{cases}$$

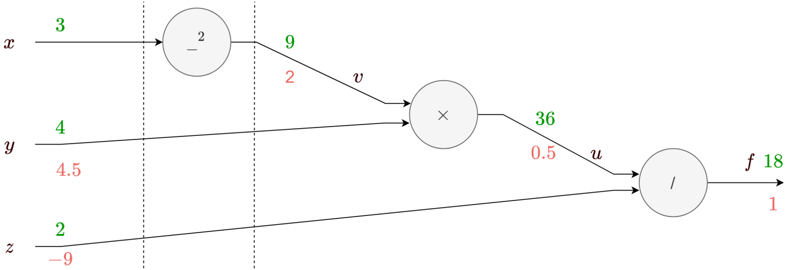
Backward Propagation: Example

- Backpropagation for $\times$ module:



Backward Propagation: Example

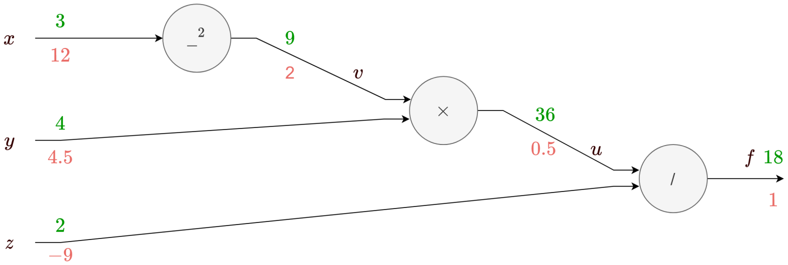
- Backpropagation for $\_^2$ module:



$$\frac{\partial f}{\partial x} = \frac{\partial v}{\partial x} \times \frac{\partial f}{\partial v} \qquad \left\{ \frac{\partial f}{\partial v} = 2 \right. \qquad \left\{ \frac{\partial v}{\partial x} = 2x = 6 \right.$$

Backward Propagation: Example

- Backpropagation for $\_^2$ module:



- Results are the same as analytical results.

Backward Propagation

- So after backward propagation we will have:
  - Gradient of loss with respect to each parameter.
  - We can apply gradient descent to update parameters.

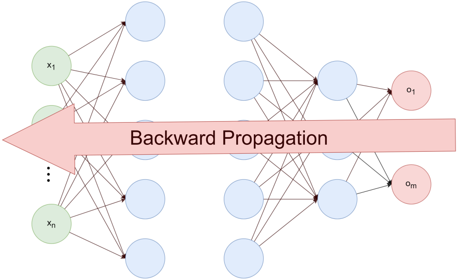


Figure: Backward pass

Thank You!

Any Question?

References

M. D. Ergün Akgün, "Biological and neural network neuron," 2018. https://www.researchgate.net/publication/326417061_Modeling_Course_Achievements_of_Elementary_Education_Teacher_Candidates_with_Artificial_Neural_Networks.

L. Nalborczyk, "A gentle introduction to deep learning in r using keras," 2021. https://www.barelysignificant.com/slides/vendredi_quanti_2021/vendredi_quantis#1.

"Gradient descent." https://subscription.packtpub.com/book/big-data-&-business-intelligence/9781788397872/1/ch01lvl1sec22/gradient-descent.

F.-F. L. . J. J. . S. Yeung, "Training neural networks," 2018. http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture07.pdf.

I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

K. Katanforoosh and D. Kunin, "Initializing neural networks," 2019. https://www.deeplearning.ai/ai-notes/initialization/.