

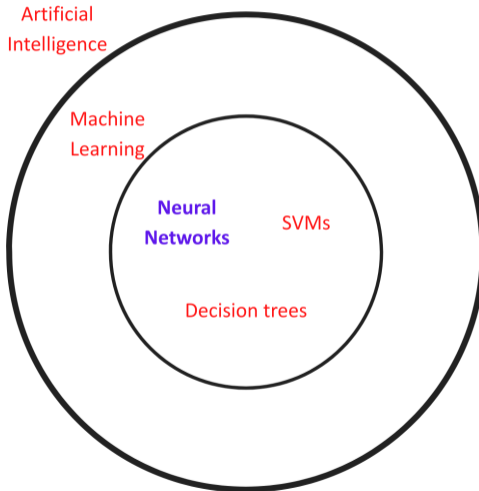
How neural networks work

Hassaan Saleem

Science Ki Duniya

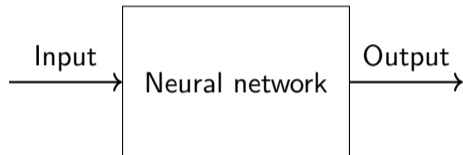


The AI landscape



The main aim

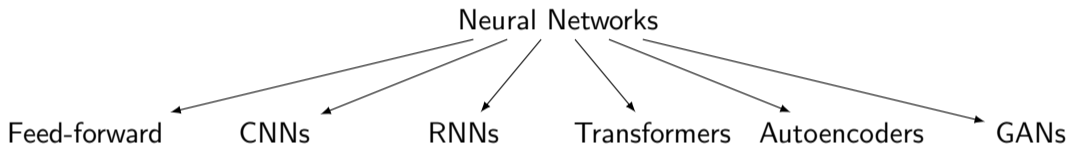
- The main aim of a neural network is to predict the output of an input



- Neural network is a **function**

$$f(\text{input}) = \text{output}$$

- How to get the right function to get a desired output?



Parameters

- How to find the **right** function?

$$f[\underbrace{\ell_1, \dots}_{\text{parameters}}](\text{input}) = \underbrace{\text{output}}_{\text{label}}$$

- Divide the parameters into three categories

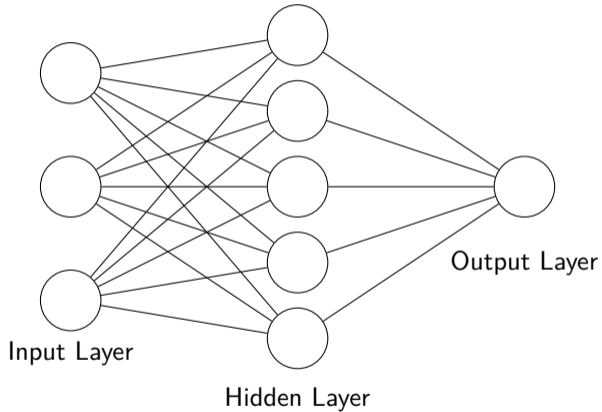
$$f[w_1, \dots; b_1, \dots; h_1, \dots](\text{input}) = \text{output}$$

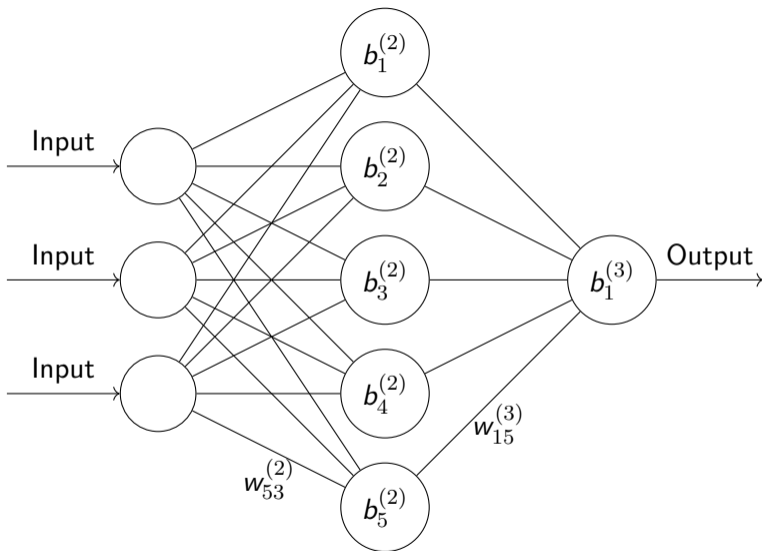
$w_{1, \dots}$ = Weights

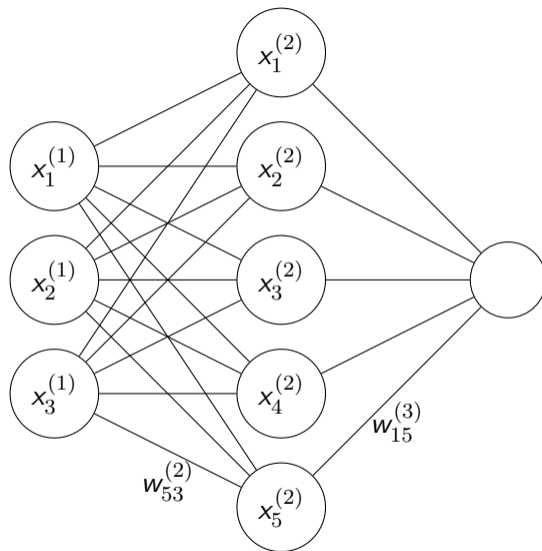
$b_{1, \dots}$ = Biases

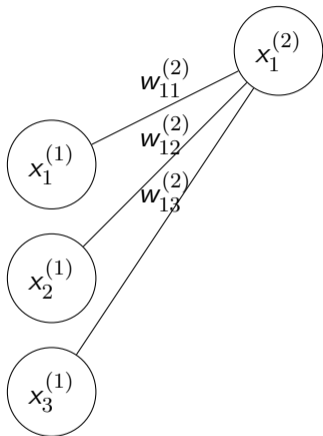
$h_{1, \dots}$ = Hyperparameters

A typical feed-forward neural network

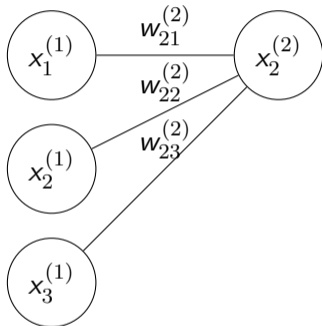




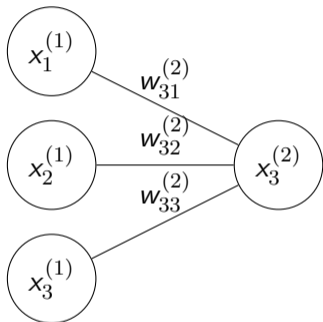




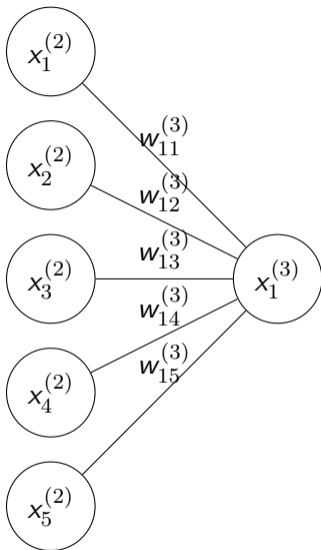
$$\begin{aligned}x_1^{(2)} &= f \left(w_{11}^{(2)} x_1^{(1)} + w_{12}^{(2)} x_2^{(1)} + w_{13}^{(2)} x_3^{(1)} + b_1^{(2)} \right) \\ &= f \left(\sum_{j=1}^3 w_{1j}^{(2)} x_j^{(1)} + b_1^{(2)} \right)\end{aligned}$$



$$\begin{aligned}x_2^{(2)} &= f \left(w_{21}^{(2)} x_1^{(1)} + w_{22}^{(2)} x_2^{(1)} + w_{23}^{(2)} x_3^{(1)} + b_2^{(2)} \right) \\ &= f \left(\sum_{j=1}^3 w_{2j}^{(1)} x_j^{(1)} + b_2^{(2)} \right)\end{aligned}$$



$$\begin{aligned}x_3^{(2)} &= f \left(w_{31}^{(2)} x_1^{(1)} + w_{32}^{(2)} x_2^{(1)} + w_{33}^{(2)} x_3^{(1)} + b_3^{(2)} \right) \\ &= f \left(\sum_{j=1}^3 w_{3j}^{(2)} x_j^{(1)} + b_3^{(2)} \right)\end{aligned}$$



$$\begin{aligned}x_1^{(3)} &= f \left(w_{11}^{(3)} x_1^{(2)} + w_{12}^{(3)} x_2^{(2)} + w_{13}^{(3)} x_3^{(2)} \right. \\ &\quad \left. + w_{14}^{(3)} x_4^{(2)} + w_{15}^{(3)} x_5^{(2)} + b_1^{(3)} \right) \\ &= f \left(\sum_{j=1}^5 w_{1j}^{(3)} x_j^{(2)} + b_1^{(3)} \right)\end{aligned}$$

$$\mathbf{x}^{(i+1)} = f \left(\mathbf{w}^{(i+1)} \cdot \mathbf{x}^{(i)} + \mathbf{b}^{(i+1)} \right)$$

$\mathbf{x}^{(i)}$ = state vector the i -th layer (activation vector)

$\mathbf{b}^{(i)}$ = bias vector the i -th layer

f = Activation function

$\mathbf{w}^{(i+1)}$ = weight matrix for the $(i+1)$ -th layer

$w_{mn}^{(i+1)}$ = from n -th neuron in i -th layer to m -th neuron in $(i+1)$ -th layer

Loss function

- The output in our example is $x_1^{(3)} = y$
- In **supervised learning**, inputs come with **labels** ℓ (correct answers)
- The goal is to minimize the difference between y and ℓ

Loss function

- You have many training examples ($p = 1, \dots, P$). The output and label for p -th training example is y_p and ℓ_p , respectively
- We define the **loss function** as

$$L = \frac{1}{P} \sum_{p=1}^P (y_p - \ell_p)^2$$

- A neural network has to minimize this loss function

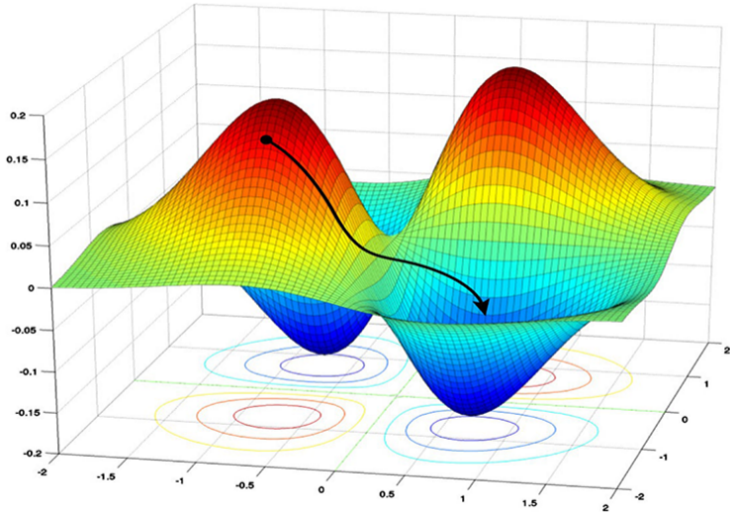
Optimization

- Begin with random weights and biases
- Update the weights and biases to reduce the loss function
- Continue until the loss function is minimum
- This procedure is called **optimization** (many algorithms are available)

- Suppose you have a function $f(x_1, \dots, x_n)$ that you want to minimize (you know that there is only one minimum, and no other extrema)
- You take the derivatives with respect to all x_n and set them to zero

$$\frac{\partial f(x_1, \dots, x_n)}{\partial x_j} = 0 \text{ for } j = 1, \dots, N$$

- Solve these equations to get the values of x_1, \dots, x_N at which $f(x_1, \dots, x_N)$ has a minimum



- We need to find weights and biases that minimize the loss function. So we have

$$\frac{\partial L}{\partial w_{mn}^{(i)}} = 0 \quad \frac{\partial L}{\partial b_m^{(i)}} = 0$$

- The only thing that depends on weights and biases in L is y_p . We need to differentiate y_p .

- In our example, the output y is

$$y = f \left(\mathbf{w}^{(3)} \cdot \mathbf{x}^{(2)} + b^{(3)} \right)$$

- We also have

$$\mathbf{x}^{(2)} = f \left(\mathbf{w}^{(2)} \cdot \mathbf{x}^{(1)} + \mathbf{b}^{(2)} \right)$$

- So we have

$$y = f \left(\mathbf{w}^{(3)} \cdot f \left(\mathbf{w}^{(2)} \cdot \mathbf{x}^{(1)} + \mathbf{b}^{(2)} \right) + b^{(3)} \right)$$

- The easiest derivatives to take are the derivatives with respect to $b^{(3)}$ and $\mathbf{w}^{(3)}$, then $\mathbf{b}^{(2)}$ and $\mathbf{w}^{(2)}$
- This is called **backpropagation** (Geoffrey Hinton got a Nobel Prize for it)

- The weights and biases are changed by **weight update rule**

$$w_{mn}^{(i)}(\text{new}) = w_{mn}^{(i)}(\text{old}) - \alpha \frac{\partial L}{\partial w_{mn}^{(i)}}$$

$$b_m^{(i)}(\text{new}) = b_m^{(i)}(\text{old}) - \alpha \frac{\partial L}{\partial b_m^{(i)}}$$

- α is called the **learning rate** and this method update weights/biases is called **gradient descent**.
- Gradient descent isn't very desirable for many applications. Other methods exist e.g. Stochastic gradient descent (SGD), Adam, and Momentum.

Hyperparameters

- We are optimizing weights and biases only.
- We aren't optimizing
 - the number of layers
 - the learning rate
 - the number of neurons in each layer
- These parameters are called **hyperparameters**. There is no known theory to optimize these hyperparameters.

After training

- After optimization, the neural network is **trained**
- We can test is using some **test data** (in contrast to **training data**)
- If it passes the test, you can use this neural network to perform tasks you trained it for

Miscellaneous points

- There are many issues that I am not mentioning here (underfitting, overfitting, hallucination, vanishing gradient problem)
- Other kinds of neural networks (**Architectures**) are used for other purposes
 - CNNs (for context, e.g., images)
 - RNNs (for tasks that require memory, e.g. language processing)
 - Transformers (RNNs, but better, used to make ChatGPT)
- You can use **Pytorch** library in Python to make and train neural networks