Artificial Neural Networks

John Stachurski

2025

Topics

- Foo
- Bar

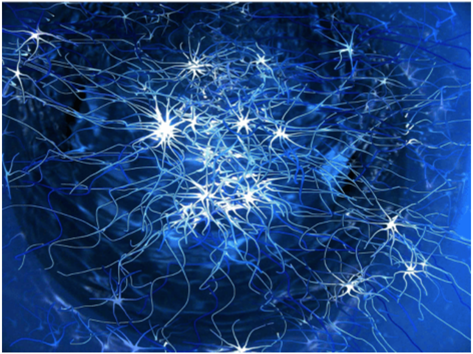
History

- 1940s: McCulloch & Pitts create mathematical model of NN
- 1950s: Rosenblatt develops the perceptron (trainable NN)
- 1960s-70s: Limited progress with single layer perceptrons
- 1980s: Backpropagation algorithm enables training of MLPs
- 1990s: SVMs temporarily overshadow ANNs in popularity
- 2000s: Deep learning finds successes in large problems

Last 10 years: Explosion of progress in deep learning

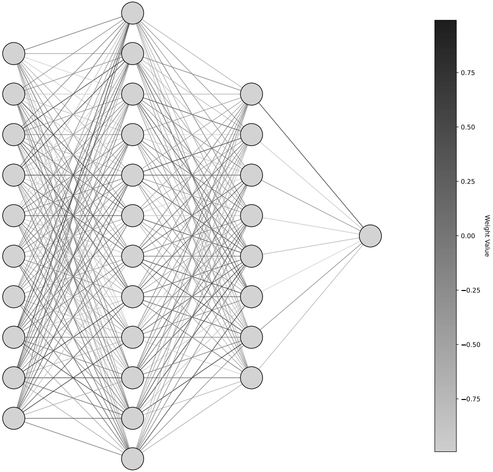
- CNNs, RNNs, LSTMs, transformers, LLMs, etc.

A model of the human brain

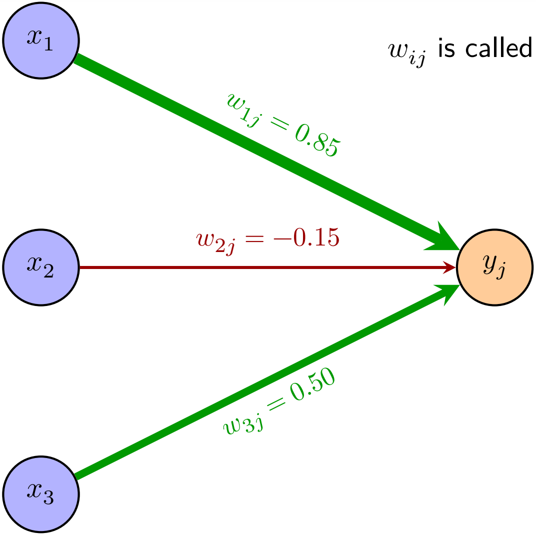


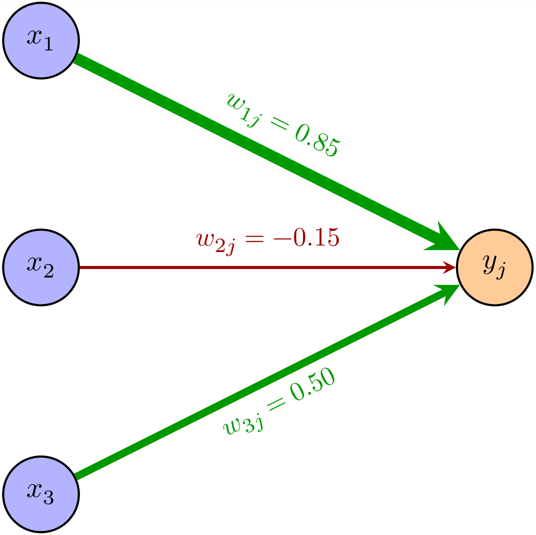
– source: Dartmouth undergraduate journal of science

A mathematical representation: directed acyclic graph

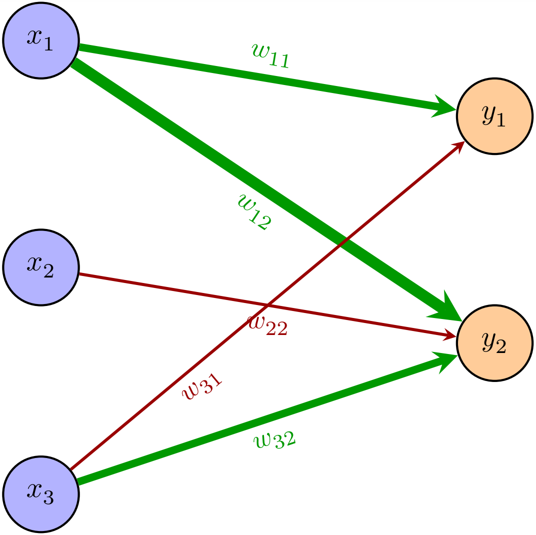


w_{ij} is called a "weight"





$$y_j = \sum_i w_{ij} x_i$$



$$y_1 = \sum_i w_{i1} x_i$$

$$y_2 = \sum_i w_{i2} x_i$$

$$\implies y = xW$$

In fact we add a bias term as well

$$y_j = \sum_i w_{ij} x_i \qquad \to \qquad y_j = \sum_i w_{ij} x_i + b_j$$

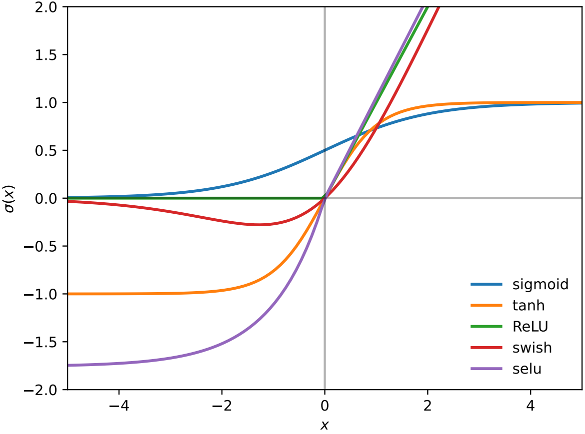
And also a nonlinear "activation function" $\sigma \colon \mathbb{R} \to \mathbb{R}$

$$y_j = \sum_i w_{ij} x_i + b_j \qquad \to \qquad y_j = \sigma\left(\sum_i w_{ij} x_i + b_j\right)$$

Applying σ pointwise, we can write this in vector form as

- $y = \sigma(xW + b)$ or
- $y = \sigma(Ax)$ where $Ax = xW + b$

Common activation functions



Aim: approximate an unknown functional relationship

$$y = f(x) \qquad (x \in \mathbb{R}^k, \ y \in \mathbb{R})$$

Examples.

- $x =$ cross section of returns, $y =$ return on oil futures tomorrow
- $x =$ weather sensor data, $y =$ max temp tomorrow

Problem:

- observe $(x_i, y_i)_{i=1}^n$ and seek f such that $y_{n+1} \approx f(x_{n+1})$

Nonlinear regression: choose model $\{f_\theta\}_{\theta\in\Theta}$ and minimize the empirical loss

$$\ell(\theta) := \frac{1}{n}\sum_{i=1}^{n}(y_i - f_\theta(x_i))^2 \quad \text{s.t.} \quad \theta \in \Theta$$

In the case of ANNs, we consider all f_θ having the form

$$f_\theta = \sigma \circ A_m \circ \cdots \circ \sigma \circ A_2 \circ \sigma \circ A_1$$

where

- $A_\ell x = xW_\ell + b_\ell$ is an affine map
- σ is a nonlinear "activation" function

Nonlinear regression: choose model $\{f_\theta\}_{\theta\in\Theta}$ and minimize the empirical loss

$$\ell(\theta) := \frac{1}{n}\sum_{i=1}^{n}(y_i - f_\theta(x_i))^2 \quad \text{s.t.} \quad \theta \in \Theta$$

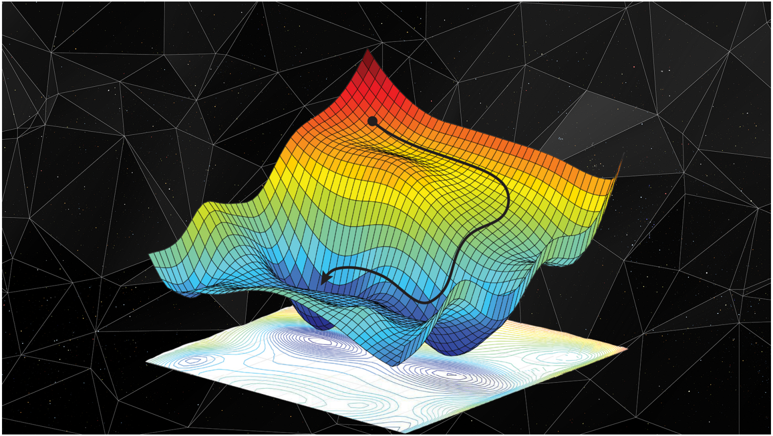
In the case of ANNs, we consider all f_θ having the form

$$f_\theta = \sigma \circ A_m \circ \cdots \circ \sigma \circ A_2 \circ \sigma \circ A_1$$

where

- $A_\ell x = xW_\ell + b_\ell$ is an affine map
- σ is a nonlinear "activation" function

Minimizing the loss functions



Source: https://danielkhv.com/

Gradient descent

Algorithm:

$$\theta_{\text{next}} = \theta - \lambda \nabla_\theta \ell(\theta, x, y)$$

- take a step in the opposite direction to the grad vector
- λ is the **learning rate** – often changes at each step
- iterate until hit a stopping condition
- in practice replace $\ell(\theta)$ with batched loss

$$\frac{1}{|B|} \sum_{i \in B} (y_i - f_\theta(x_i))^2$$

Using batches $\rightarrow$ **stochastic gradient descent**

Extensions

Different loss functions, different architectures

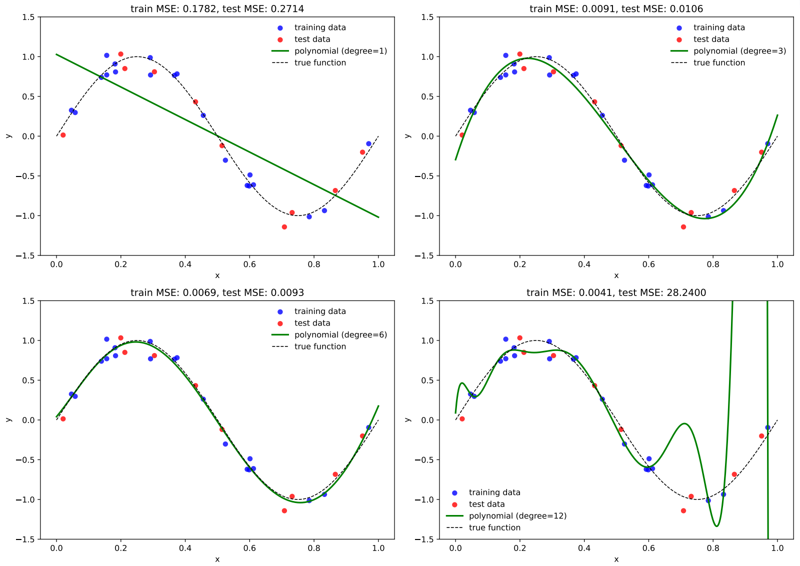
- Loss functions with regularization
- – insert other loss funcs –
- Convolutional neural networks
- Recurrent neural networks
- Transformers, etc.

Why is deep learning so successful for some problems?

Story 1 – human brain, universal function approx, can break curse of dim

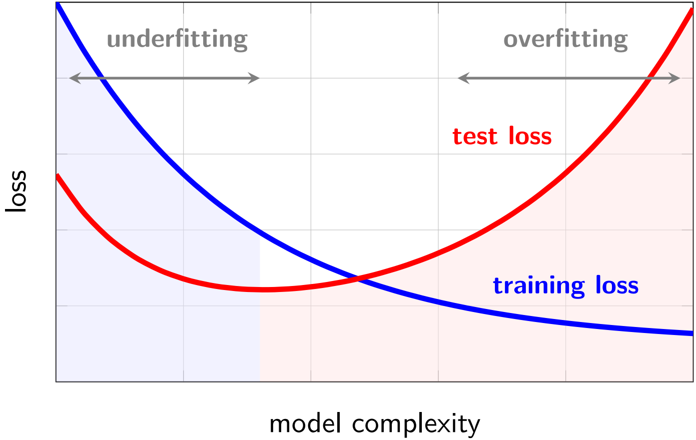
Story 2 – a flexible function fitting technique that extends well to high dimensions and has received massive investment from the CS community

What about overfitting?



train MSE: 0.1782, test MSE: 0.2714
training data
test data
polynomial (degree=1)
true function
train MSE: 0.0091, test MSE: 0.0106
training data
test data
polynomial (degree=3)
true function
train MSE: 0.0069, test MSE: 0.0093
training data
test data
polynomial (degree=6)
true function
train MSE: 0.0041, test MSE: 28.2400
training data
test data
polynomial (degree=12)
true function
x
y

Overfitting and underfitting



If production-level deep learning models are so large, why don't they overfit data?

Some solutions don't use the full complexity of the model

- Early stopping halts training when validation performance begins to decline
- Many architectures include random drop out – randomly shut down neurons during training

The insight behind dropout - introducing randomness can prevent overfitting - has influenced other regularization techniques in deep learning, such as DropConnect and Stochastic Depth

Also, modern architectures have inductive biases that guide learning toward useful patterns

- translation invariance in CNNs (same pattern can be recognized anywhere in an image)
- parameter sharing in CNNs – using the same weights across different parts of the image – enforces learning of features that are useful everywhere
- parameter sharing in RNNs – similarity of transformations across time
- Markovian assumption in RNNs
- Layer normalization and residual connections in transformers - create a bias toward stable training dynamics and information preservation across layers.

Finally, there is some evidence of "double descent" – test error starts to fall again when the number of parameters is very high

Double descent phenomenon