Introduction to Data Science

Lesson 9
Deep Learning

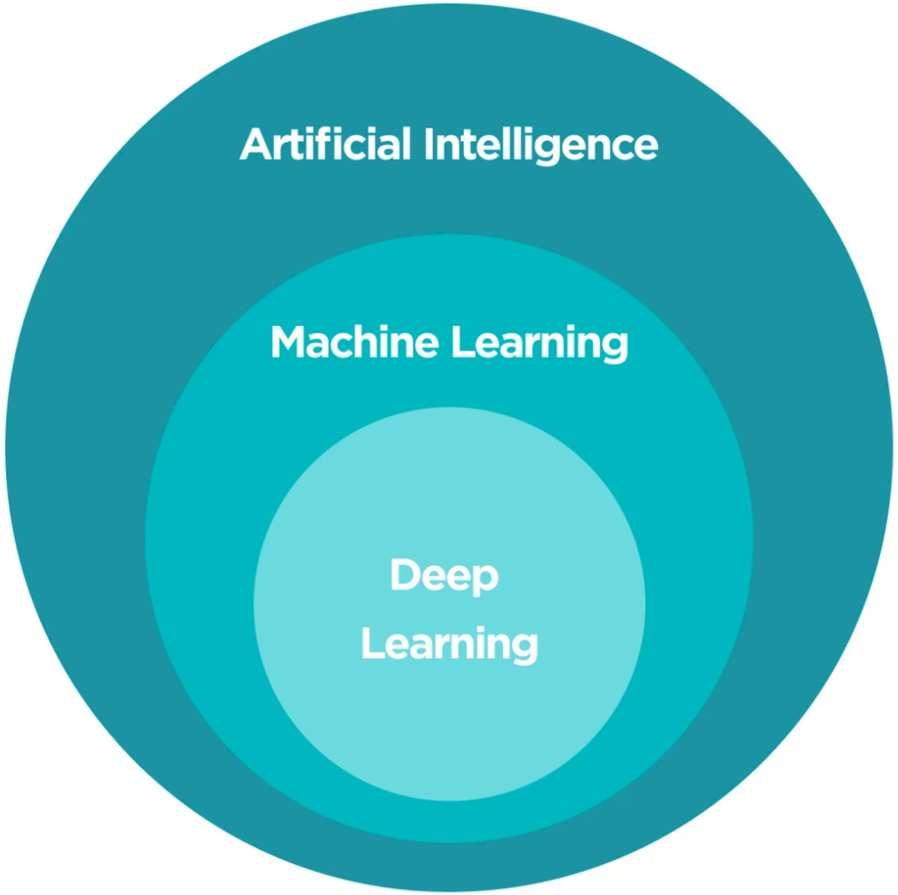
Marija Stankova Medarovska, PhD
marija.s.medarovska@uacs.edu.mk

Deep Learning

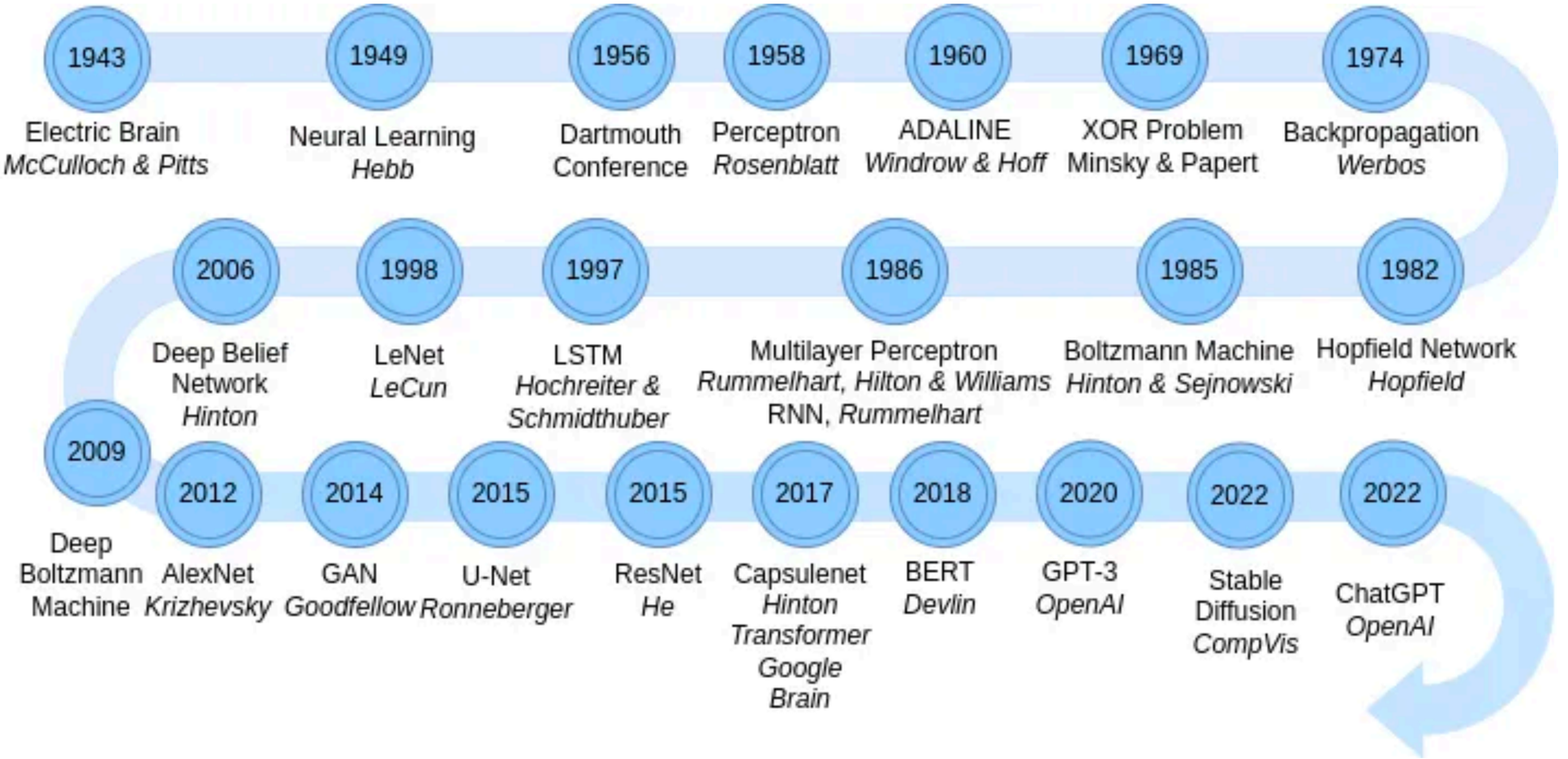
Deep Learning

Deep learning has its origins in early work that tried to model networks of neurons in the brain (McCulloch and Pitts, 1943) with computational circuits. For this reason, the networks trained by deep learning methods are often called neural networks. These neural networks are composed of many layers of interconnected nodes, which process and transform data.

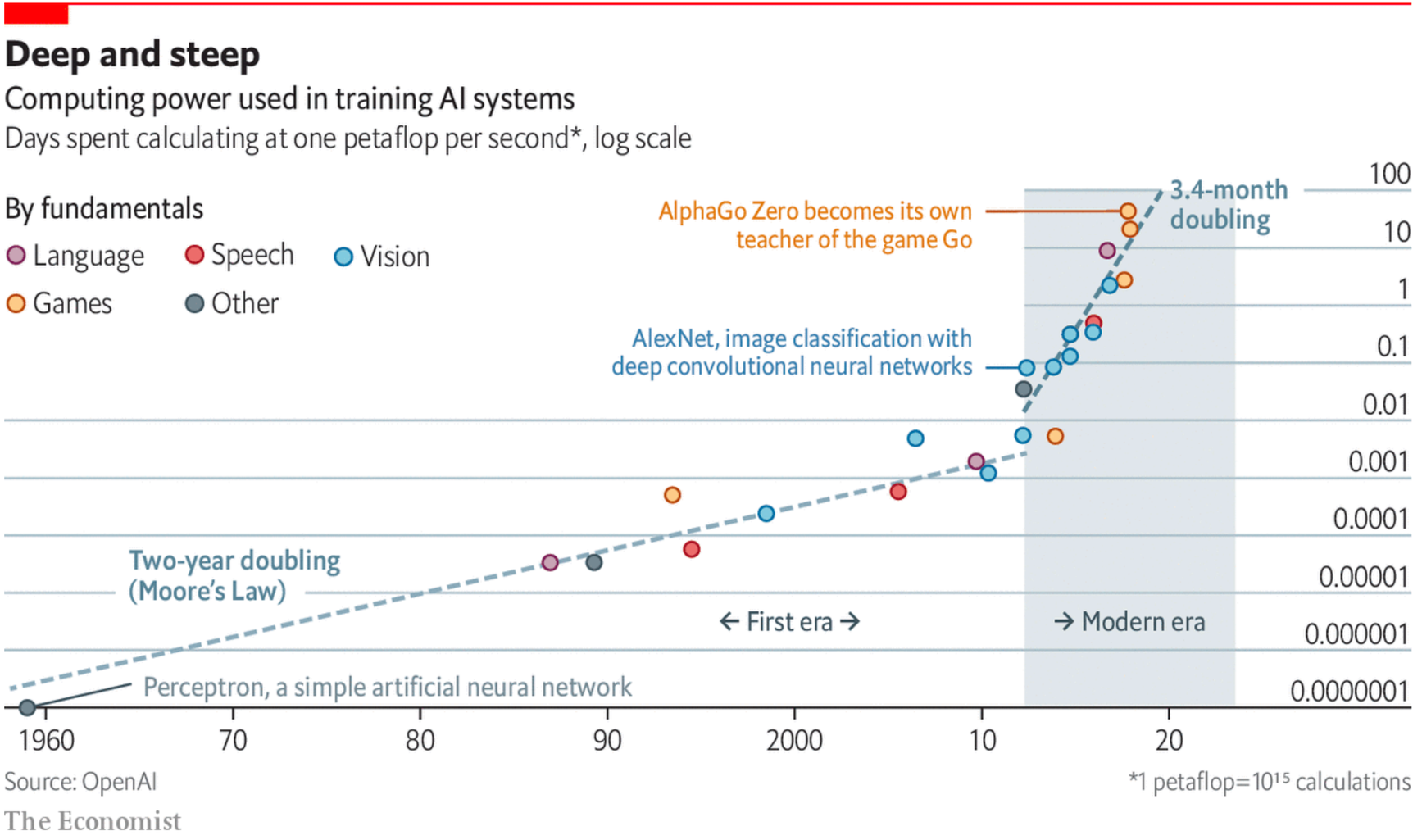
The word “deep” refers to the fact that the circuits are typically organized into many layers, which means that computation paths from inputs to outputs have many steps.



Historical Development



Historical Development



Why now?

Big Data

- Larger Datasets
- Easer collection and storage

IMAGENET kaggle

Google Dataset Search

Papers With Code

Hardware

- GPUs (Graphics Processing Units) and specialized AI chips
- Massively Parallelizable
- Cloud Platforms

aws Azure Google Cloud

Software

- Improved algorithms
- Availability of Open Source Tools and Platforms

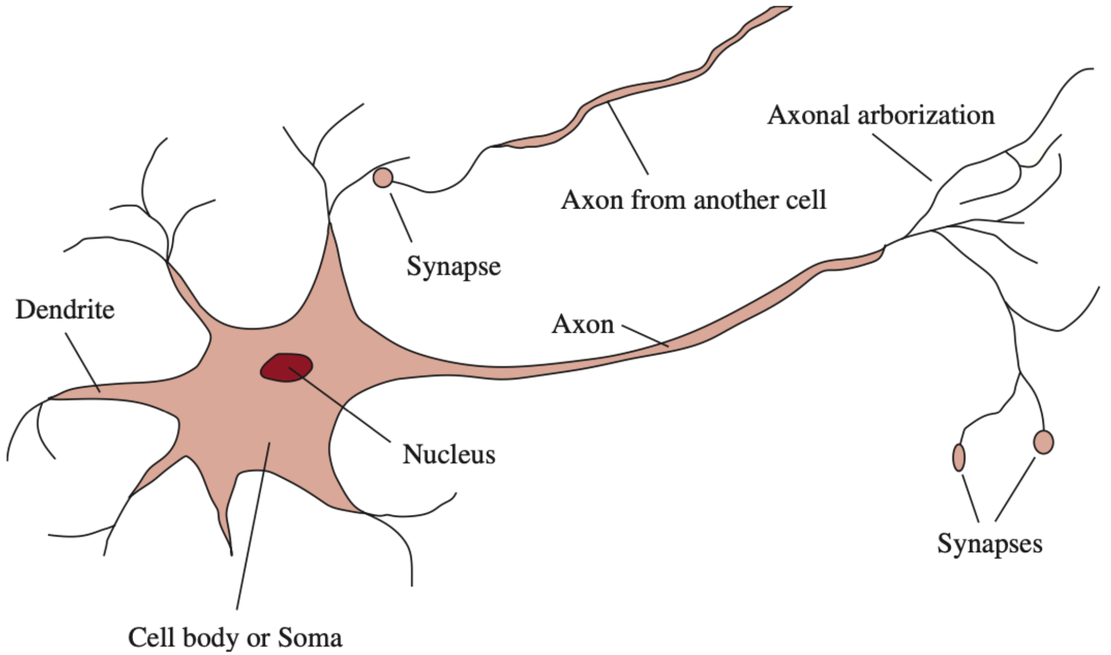
TensorFlow PyTorch

Google Colaboratory Hugging Face

jupyter scikit learn

Perceptron

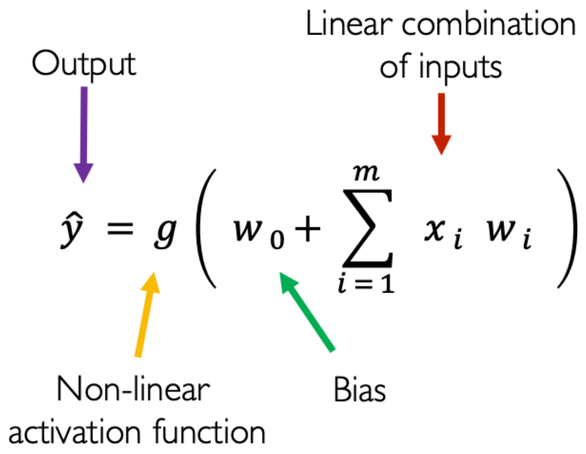
Biological Brains as Inspiration

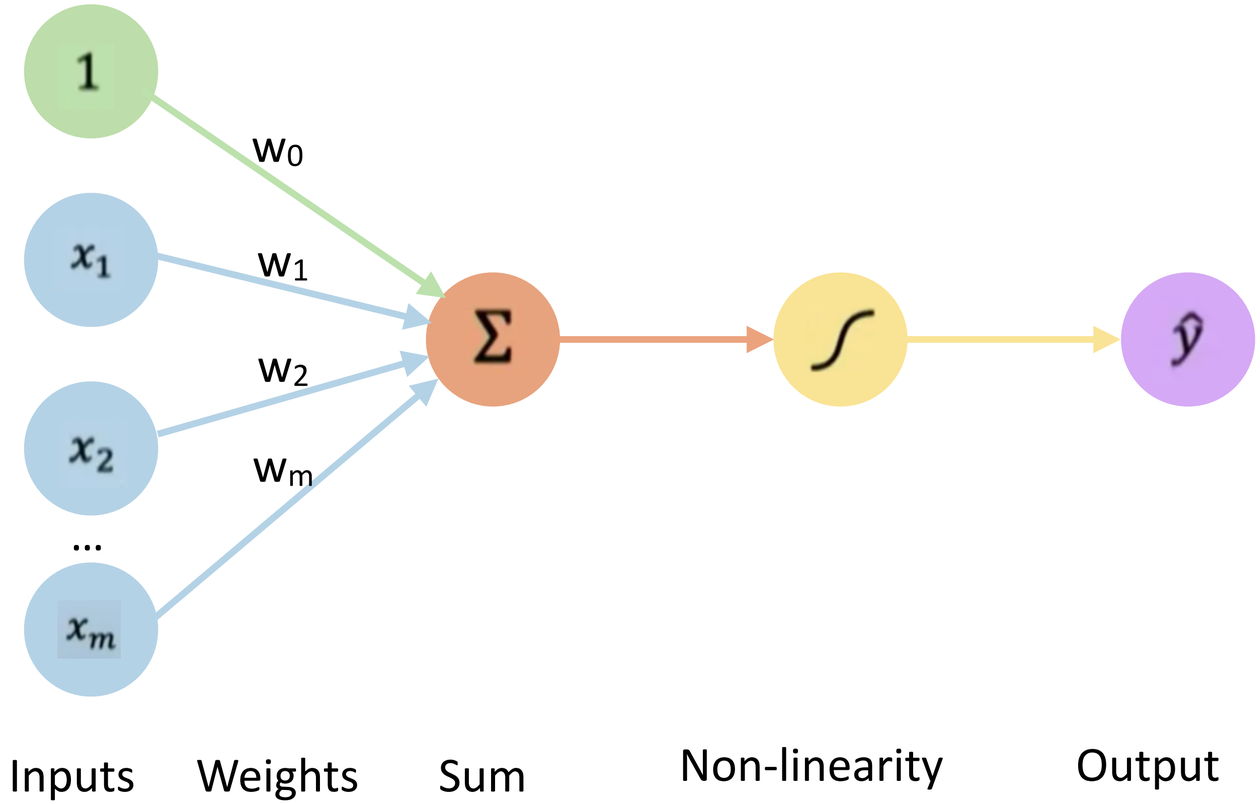


The parts of a nerve cell or neuron. Each neuron consists of a cell body, or soma, that contains a cell nucleus. Branching out from the cell body are a number of fibers called dendrites and a single long fiber called the axon. The axon stretches out for a long distance, much longer than the scale in this diagram indicates. Typically, an axon is 1 cm long (100 times the diameter of the cell body), but can reach up to 1 meter. A neuron makes connections with 10 to 100,000 other neurons at junctions called synapses. Signals are propagated from neuron to neuron by a complicated electrochemical reaction. The signals control brain activity in the short term and also enable long-term changes in the connectivity of neurons. These mechanisms are thought to form the basis for learning in the brain. Most information processing goes on in the cerebral cortex, the outer layer of the brain. The basic organizational unit appears to be a column of tissue about 0.5 mm in diameter, containing about 20,000 neurons and extending the full depth of the cortex (about 4 mm in humans). There are approximately 86 billion neurons in the human brain.

Perceptron

Each node within a network is called a perceptron (or unit). Traditionally, following the design proposed by McCulloch and Pitts, a unit calculates the weighted sum of the inputs from predecessor nodes and then applies a non linear function to produce an output.





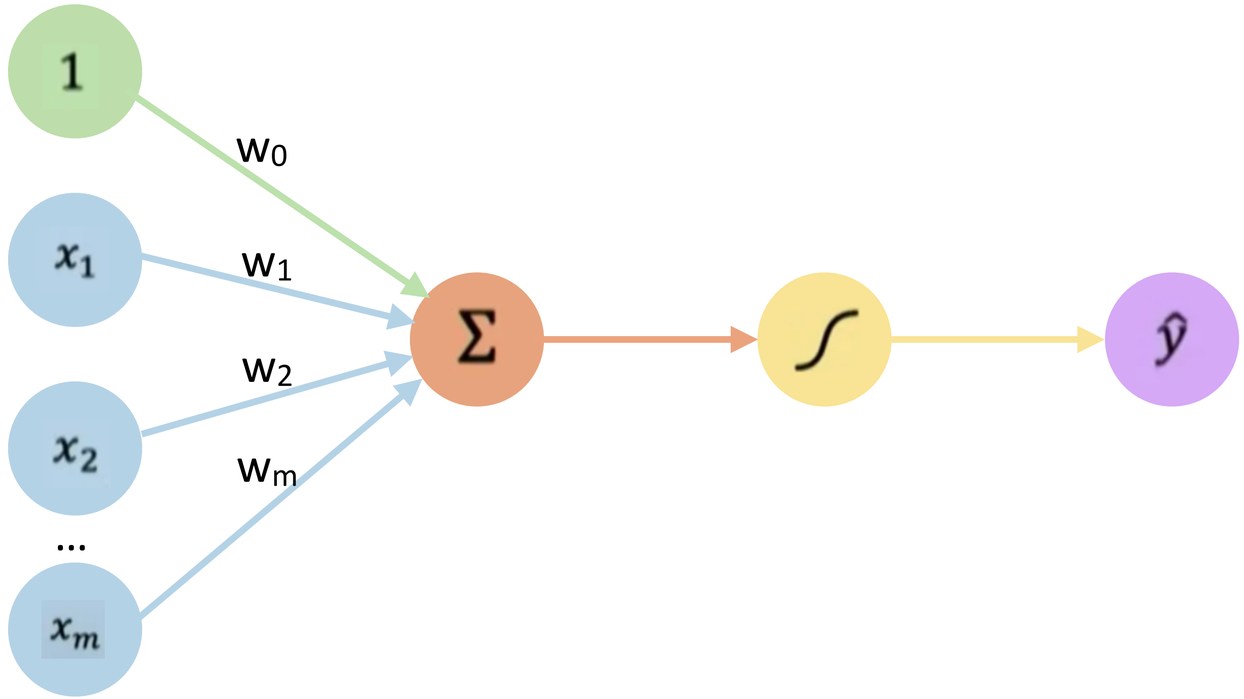
Example: Single perceptron

Perceptron

Each node within a network is called a perceptron (or unit). Traditionally, following the design proposed by McCulloch and Pitts, a unit calculates the weighted sum of the inputs from predecessor nodes and then applies a non linear function to produce an output.

$$\hat{y} = g(w_0 + \boldsymbol{X}^T \boldsymbol{W})$$

where: $\boldsymbol{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$ and $\boldsymbol{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$



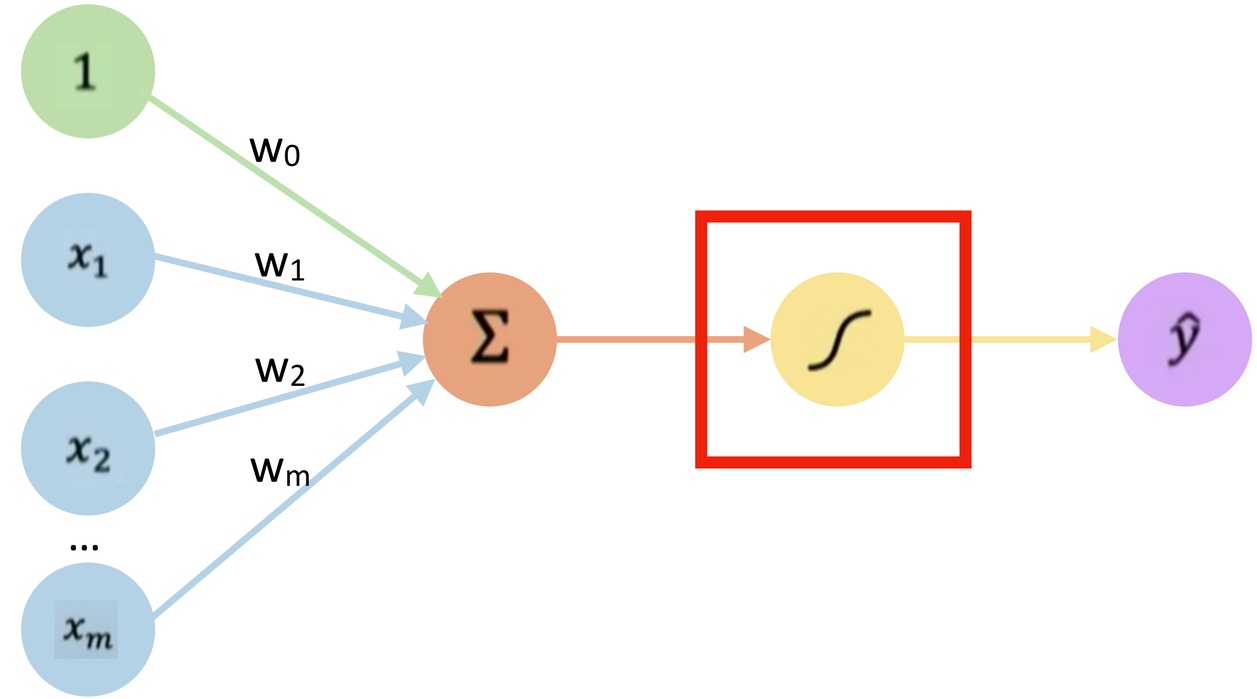
Example: Single perceptron

Activation function

The purpose of the activation function is to introduce non-linearities into the network.

The fact that the activation function is nonlinear is important because if it were not, any composition of units would still represent a linear function. The nonlinearity is what allows sufficiently large networks of units to represent arbitrary functions.

The universal approximation theorem states that a network with just two layers of computational units, the first nonlinear and the second linear, can approximate any continuous function to an arbitrary degree of accuracy



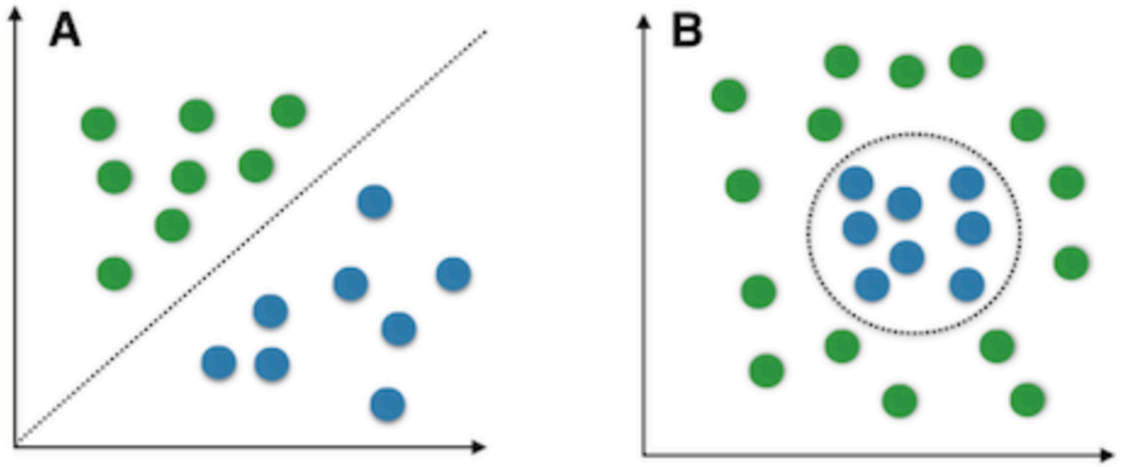
Example: Single perceptron

Activation function

The purpose of the activation function is to introduce non-linearities into the network.

The fact that the activation function is nonlinear is important because if it were not, any composition of units would still represent a linear function. The nonlinearity is what allows sufficiently large networks of units to represent arbitrary functions.

The universal approximation theorem states that a network with just two layers of computational units, the first nonlinear and the second linear, can approximate any continuous function to an arbitrary degree of accuracy



Example: (a) Linearly vs (b) non-linearly separable data

Common Activation Functions

Common activation functions:

- Logistic (sigmoid) function, also used in logistic regression

$$\sigma(x) = 1/(1+e^{-x}).$$

- ReLU (rectified linear unit) function

$$\mathrm{ReLU}(x) = \max(0,x)$$

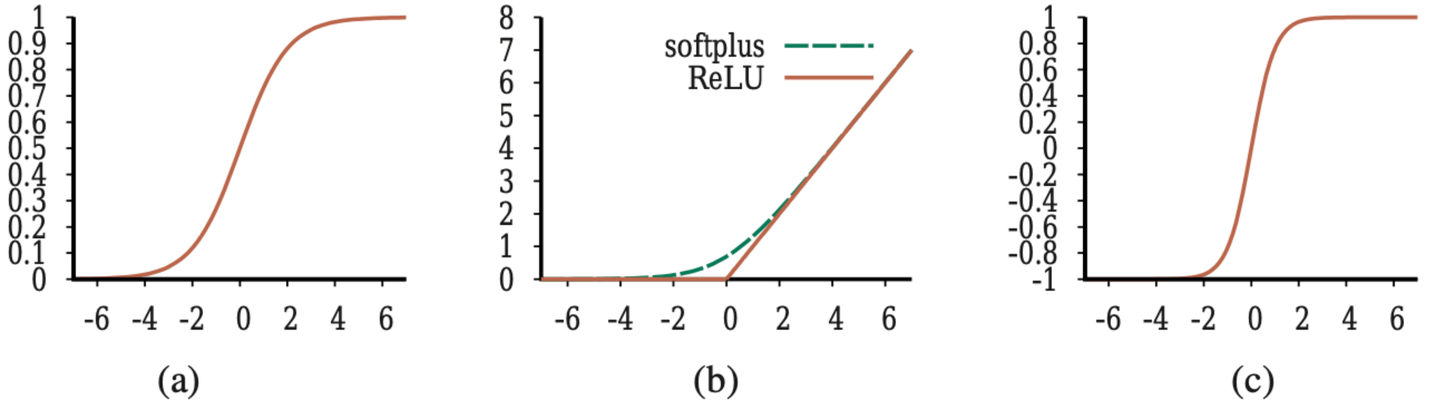
- Softplus function (smooth version of the ReLU function)

$$\mathrm{softplus}(x) = \log(1+e^{x}).$$

- The tanh function

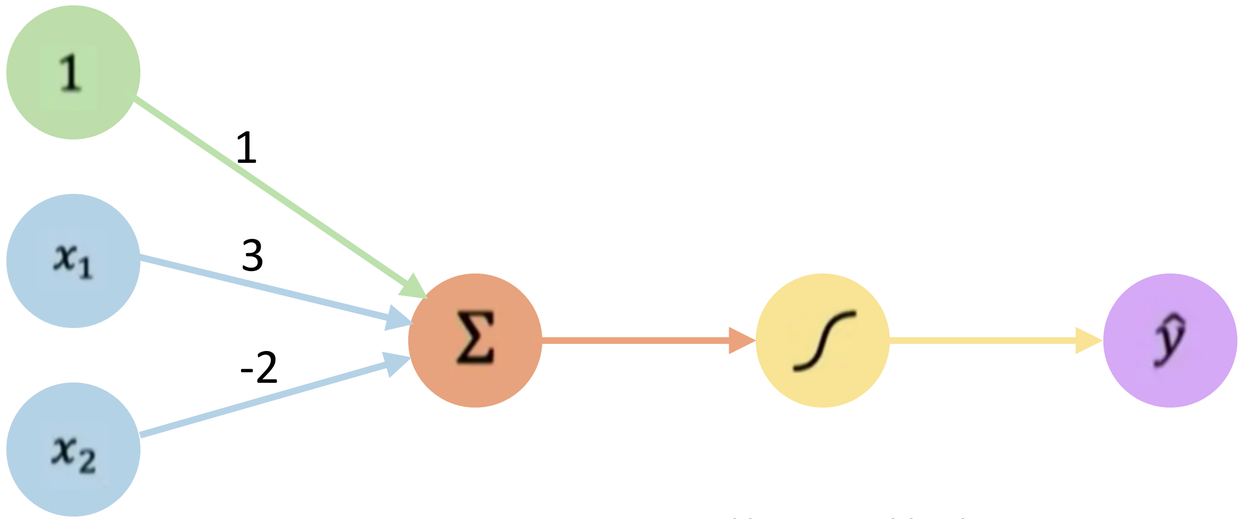
$$\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}.$$

For the first 25 years of research with multilayer networks (roughly 1985–2010), internal nodes used sigmoid and tanh activation functions almost exclusively. From around 2010 onwards, the ReLU and softplus become more popular.



Activation functions commonly used in deep learning systems: (a) the logistic or sigmoid function; (b) the ReLU function and the softplus function; (c) the tanh function.

The Perceptron: Example



Output

Linear combination of inputs

$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i\, w_i\right)$$

Non-linear activation function

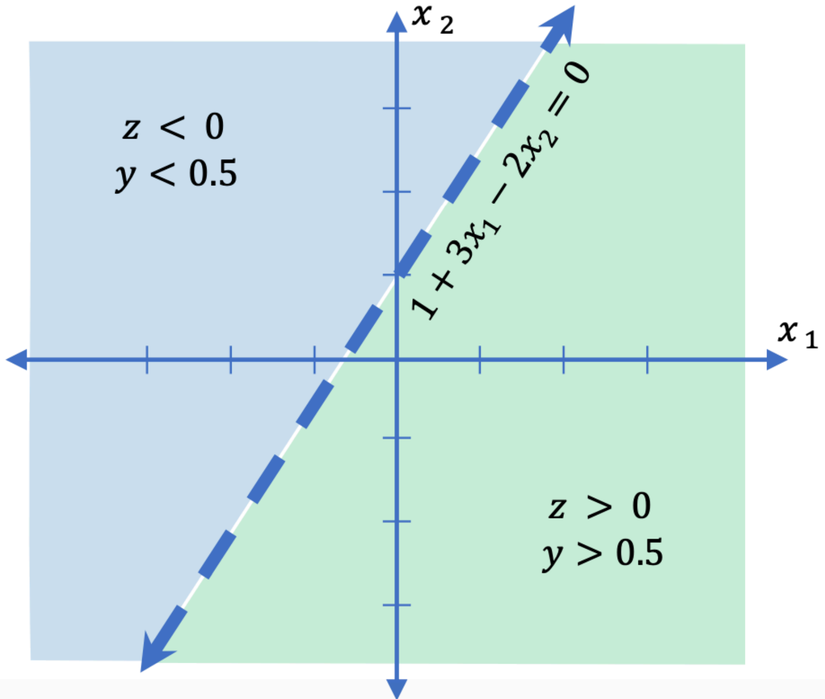
Bias

We have: $w_0 = 1$ and $\boldsymbol{W} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$

$$\hat{y} = g(w_0 + \boldsymbol{X}^T \boldsymbol{W})$$

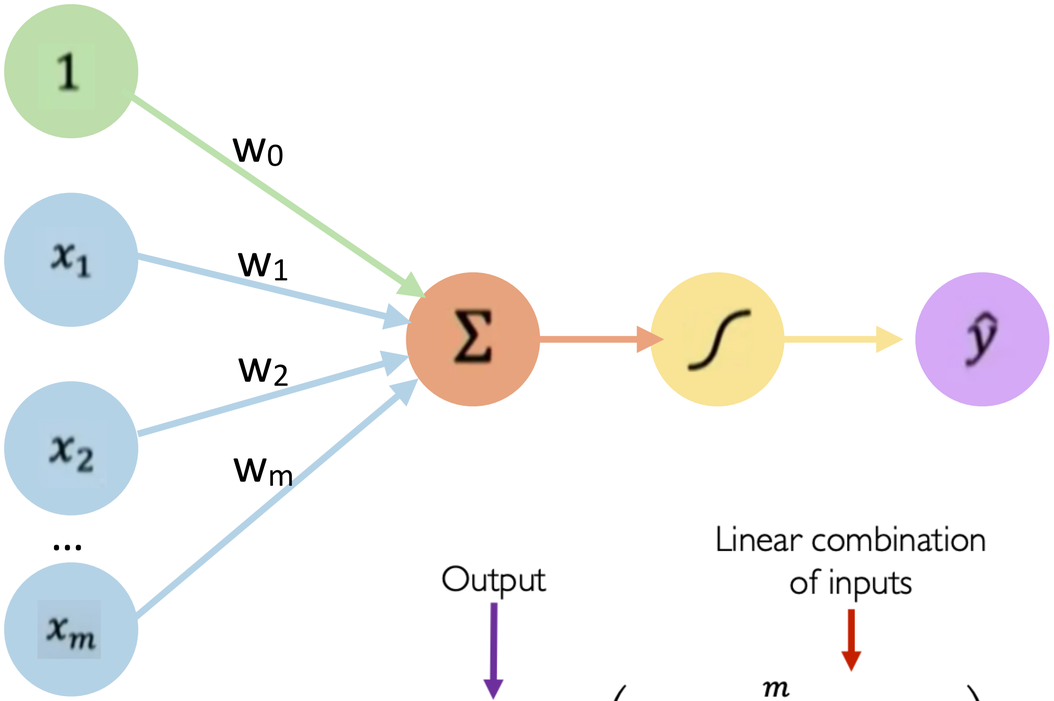
$$= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right)$$

$$\hat{y} = g(\underbrace{1 + 3x_1 - 2x_2}_{})$$



Neural Networks

Simplified Perceptron



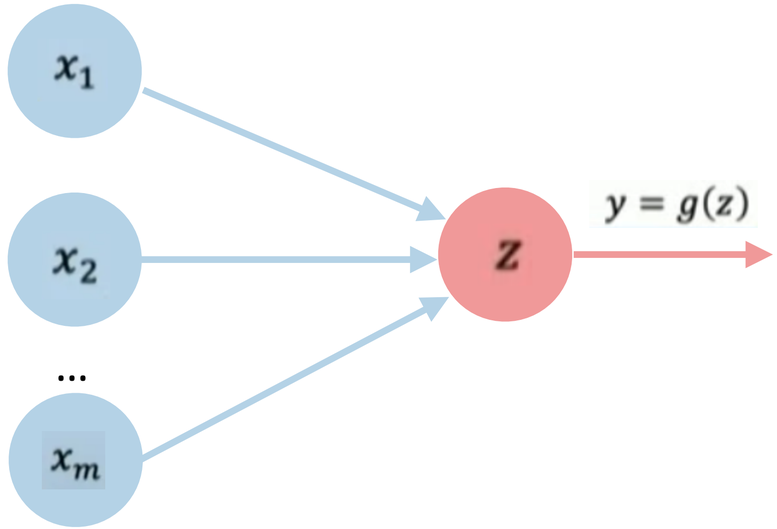
Output

Linear combination of inputs

$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i \, w_i\right)$$

Non-linear activation function

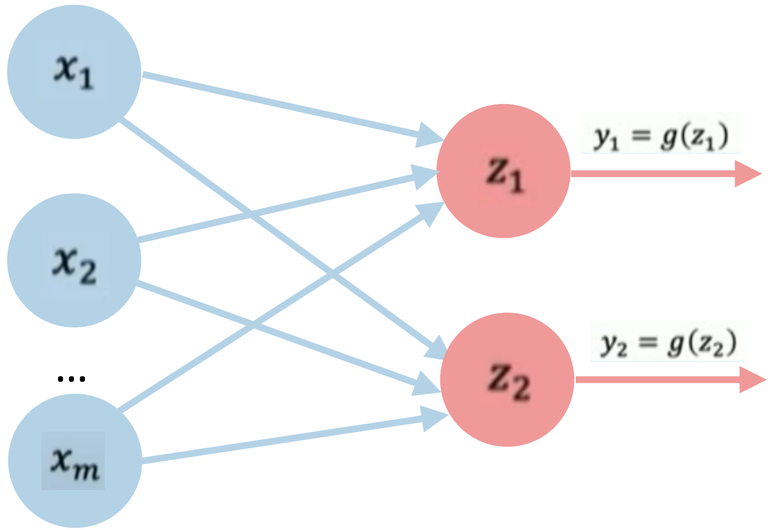
Bias



$$z = w_0 + \sum_{j=1}^{m} x_j \, w_j$$

Coupling Perceptrons

Coupling multiple perceptrons together into a neural network creates a complex function that is a composition of the algebraic expressions represented by the individual units.

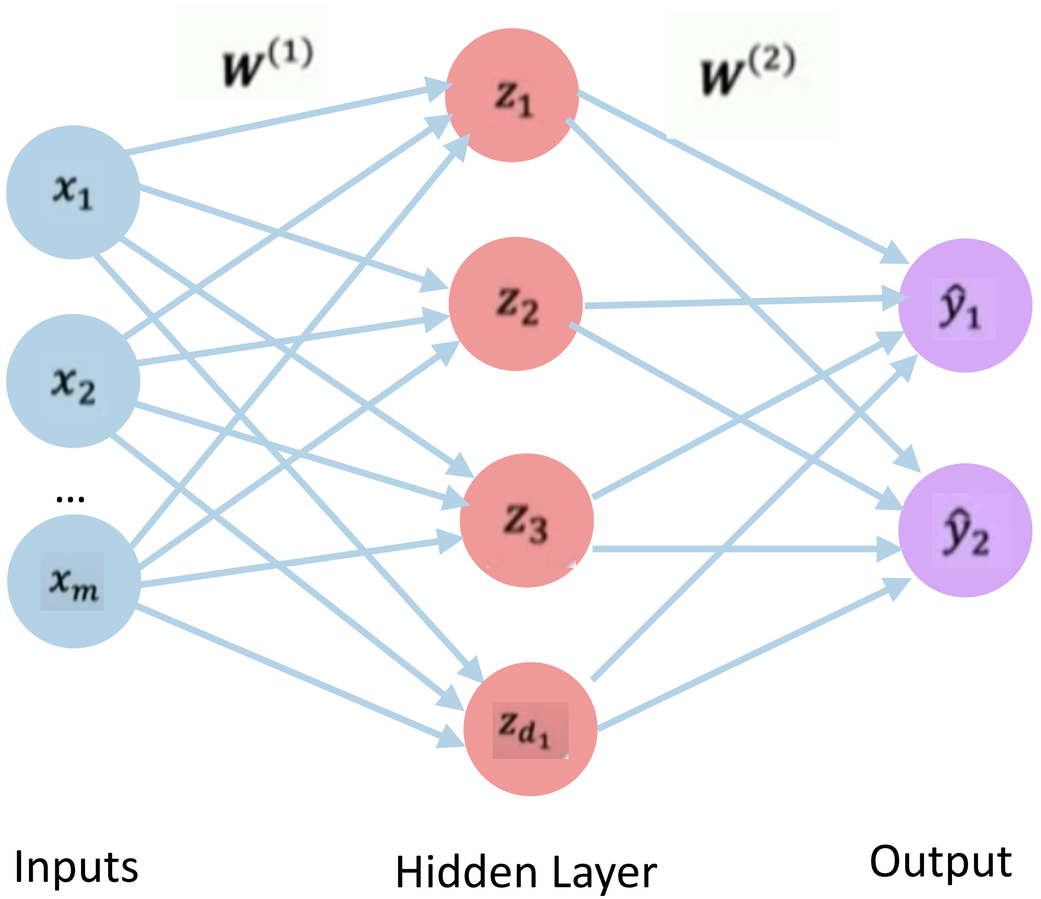


$$z_i = w_{0,i} + \sum_{j=1}^{m} x_j \, w_{j,i}$$

Simple Feedforward Networks

A feedforward network has connections only in one direction. Each node computes a function of its inputs and passes the result to its successors in the network. Information flows through the network from the input nodes to the output nodes, and there are no loops.

The input and output nodes of a computational graph are the ones that connect directly to the input data x and the output data y.

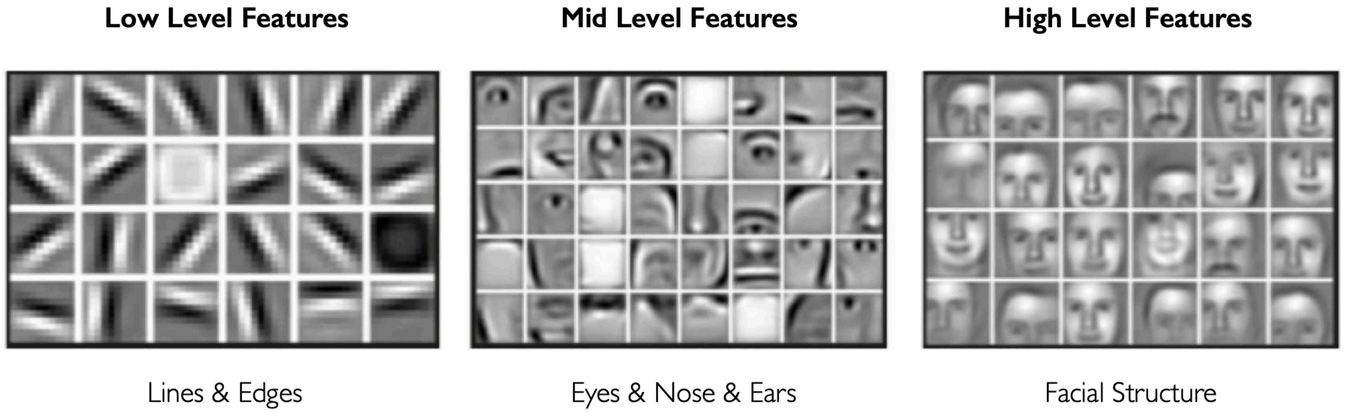


Hidden Layers

Hidden layer(s) come between the input layer and the output layer. These layers are called "hidden" because they are not directly observable from the input or output of the network; their activations are intermediate representations that the network learns from the input data.

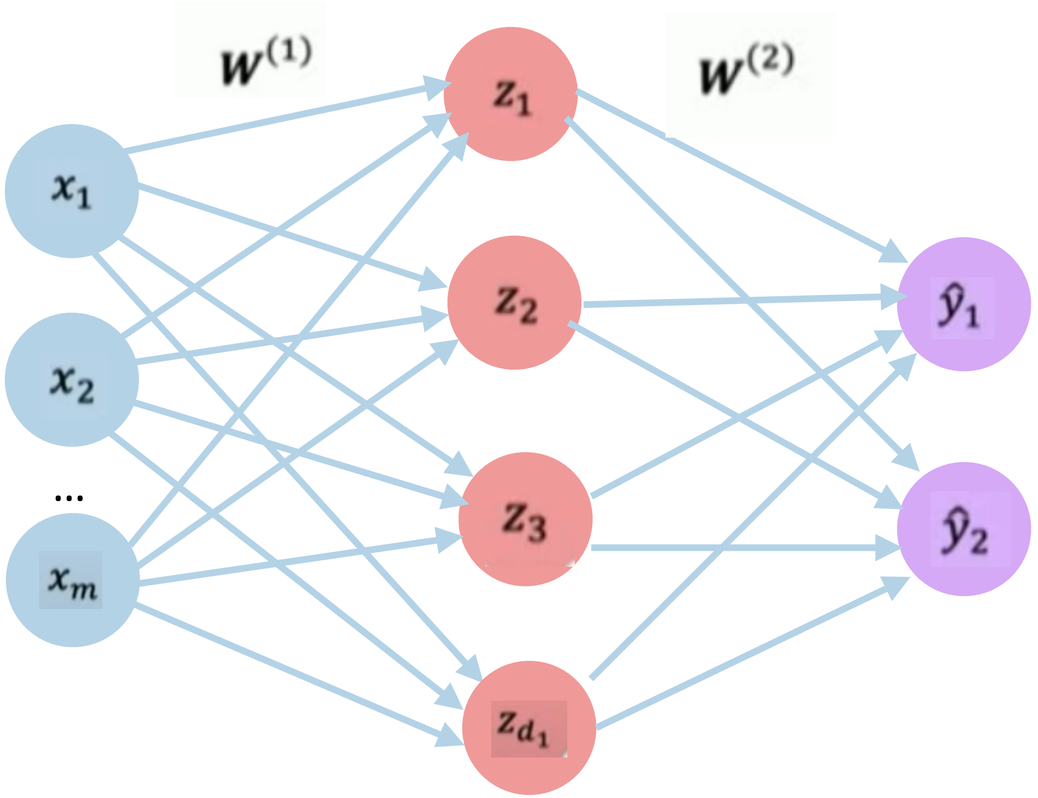
Each layer transforms the representation produced by the preceding layer to produce a new representation.

In the process of forming all these internal transformations, neural networks often discover meaningful intermediate representations of the data



Example: A neural network trained to recognize complex objects in images may form internal layers that detect useful subunits: edges, corners, ellipses, eyes, faces. Or it may not—deep networks may form internal layers whose meaning is opaque to humans, even though the output is still correct.

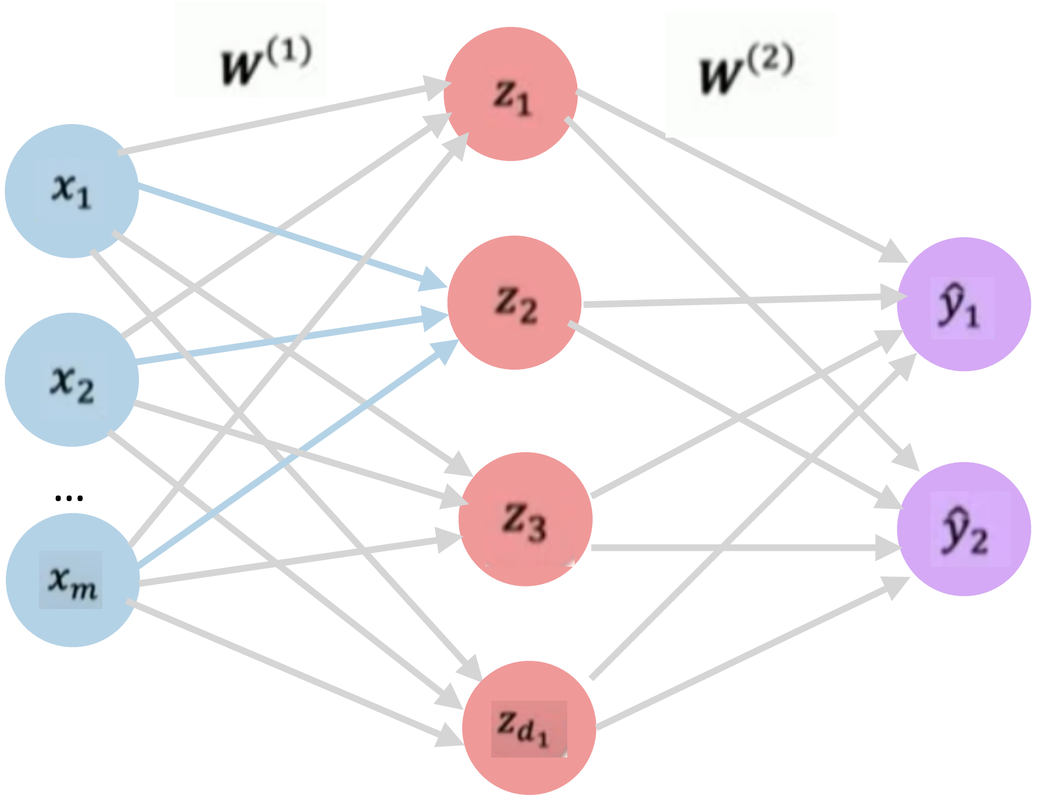
Simple Feedforward Networks



Inputs Hidden Layer Output

$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,i}^{(1)} \qquad \hat{y}_i = g\left(w_{0,i}^{(2)} + \sum_{i=1}^{d_1} z_j \, w_{j,i}^{(2)}\right)$$

Simple Feedforward Networks



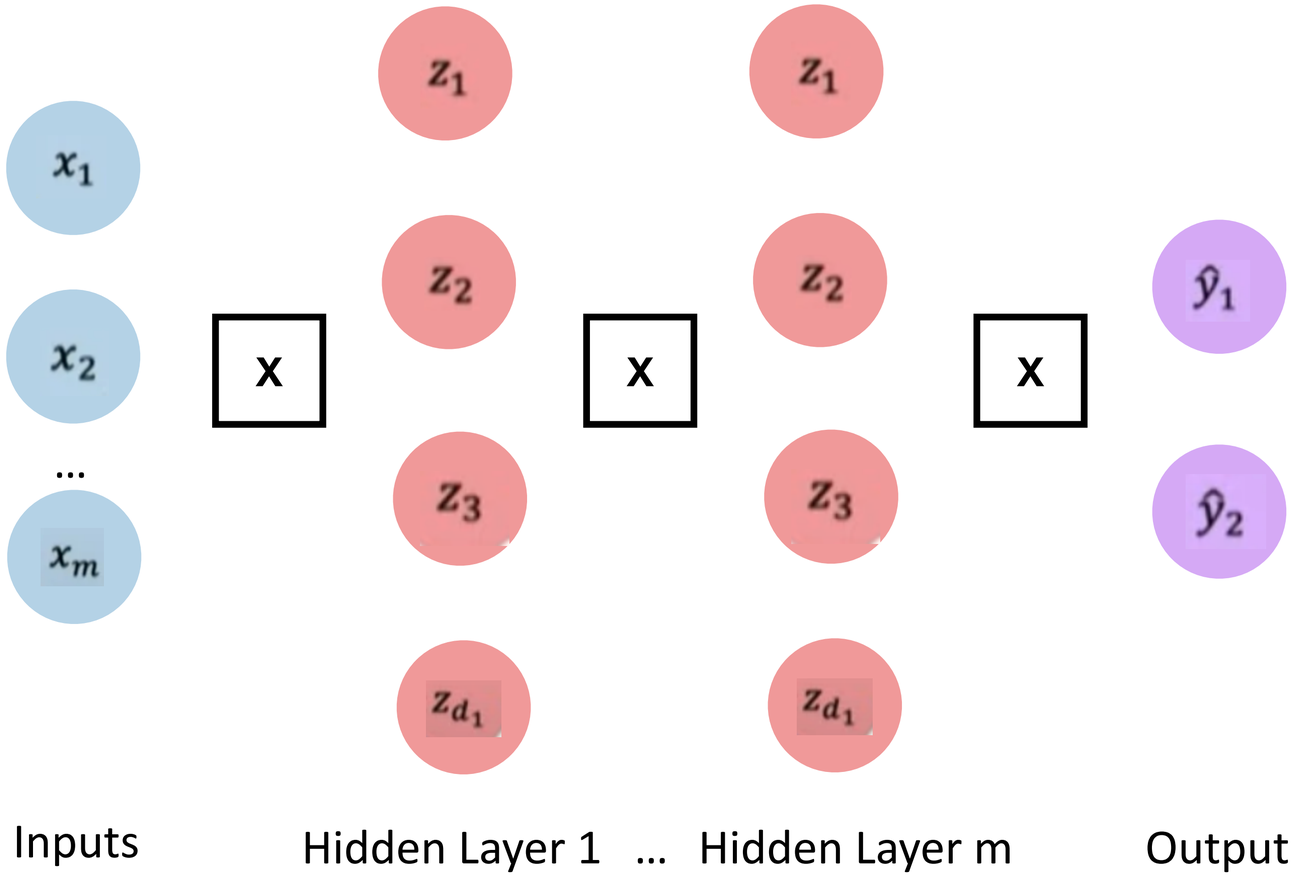
Inputs Hidden Layer Output

$$z_2 = w_{0,2}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,2}^{(1)}$$

$$= w_{0,2}^{(1)} + x_1 \, w_{1,2}^{(1)} + x_2 \, w_{2,2}^{(1)} + x_m \, w_{m,2}^{(1)}$$

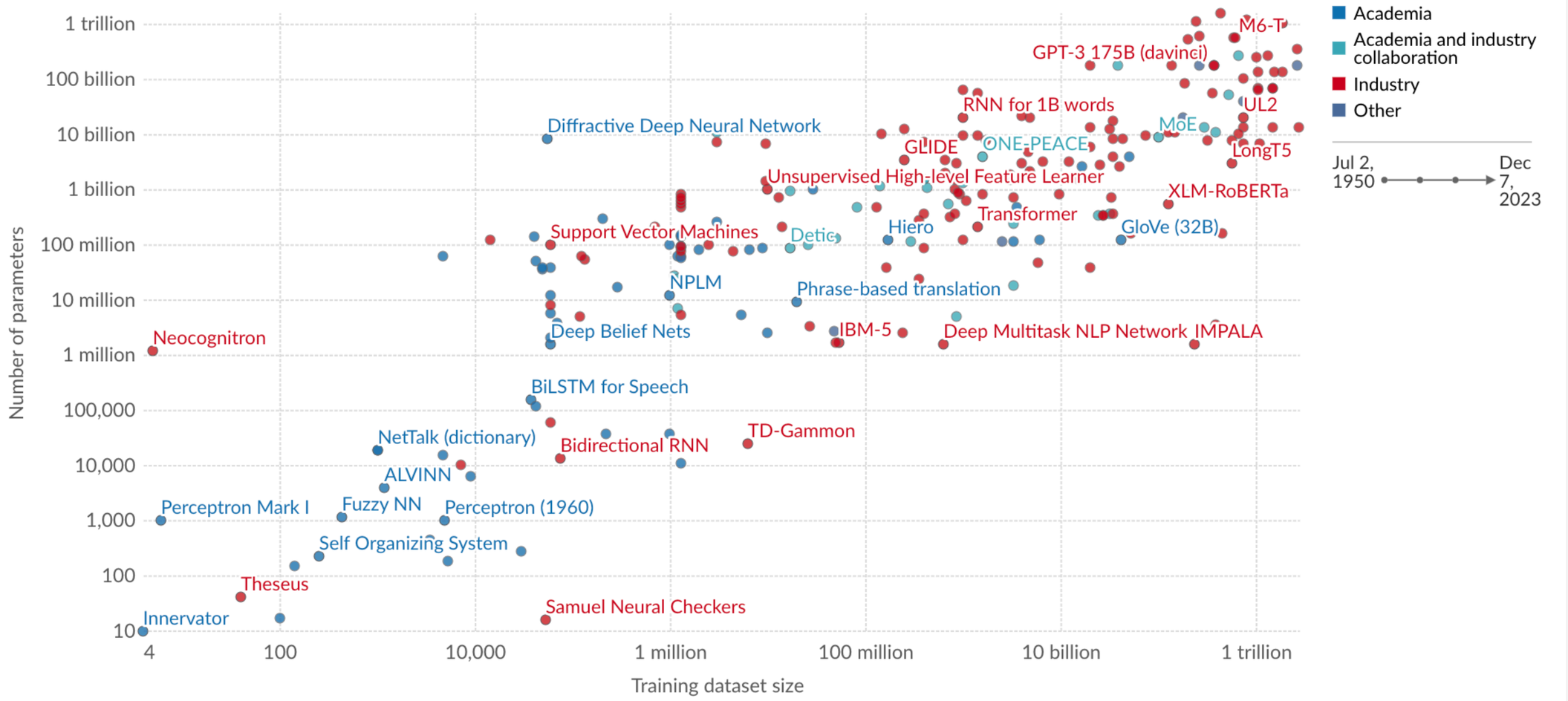
Deep Neural Networks

Neural networks with multiple hidden layers are referred to as Deep Neural Networks (DNNs)



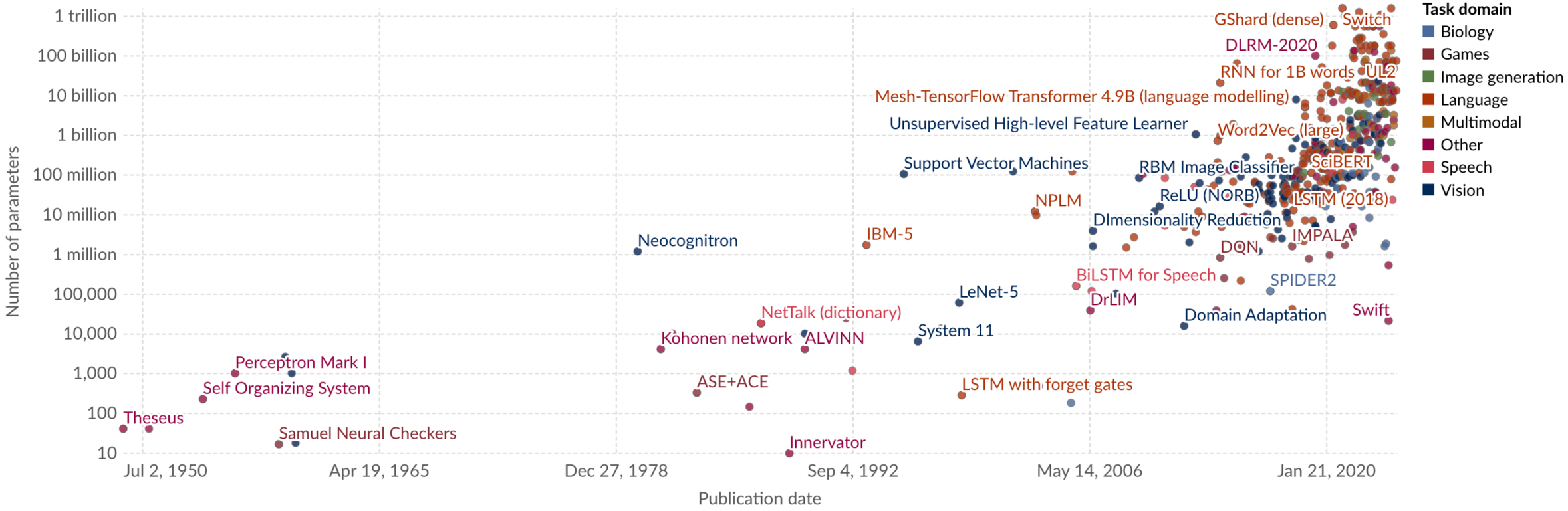
$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{d_{k-1}} g(z_{k-1,j})\, w_{j,i}^{(k)}$$

Parameters vs. training dataset size in notable AI systems



[1] https://ourworldindata.org/grapher/parameters-vs-training-dataset-size-in-notable-ai-systems-by-researcher-affiliation?time=earliest..2023-12-07

Parameters over time in notable AI systems



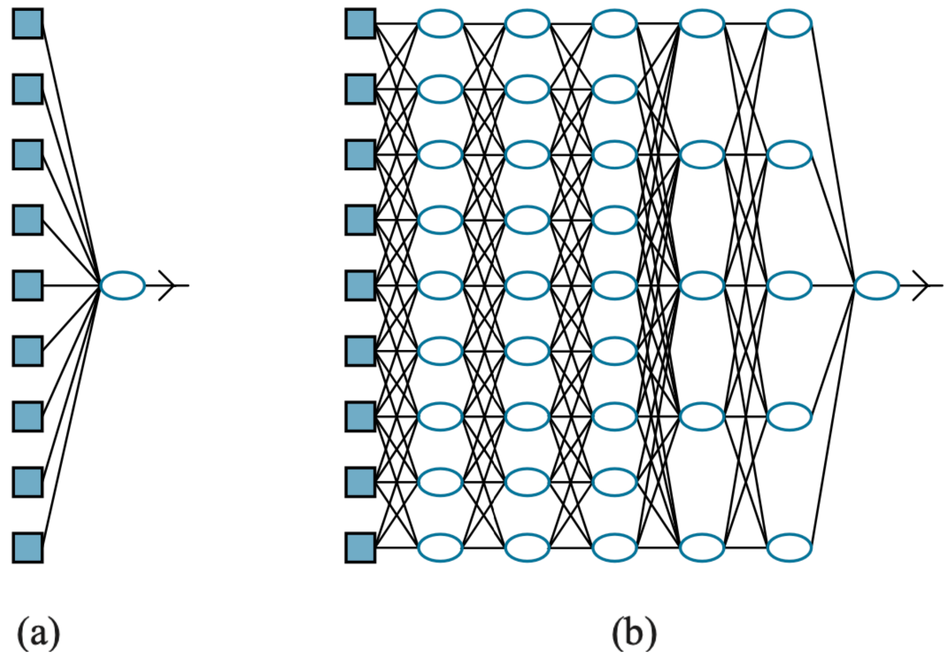
[1] https://ourworldindata.org/grapher/artificial-intelligence-parameter-count?time=earliest..2024-02-12

Difference with Linear Models

In methods such as linear and logistic regression the computation path from each input to the output is very short: multiplication by a single weight, then adding into the aggregate output.

Moreover, the different input variables contribute independently to the output, without interacting with each other.

This significantly limits the expressive power of such models. They can represent only linear functions and boundaries in the input space, whereas most real-world concepts are far more complex.



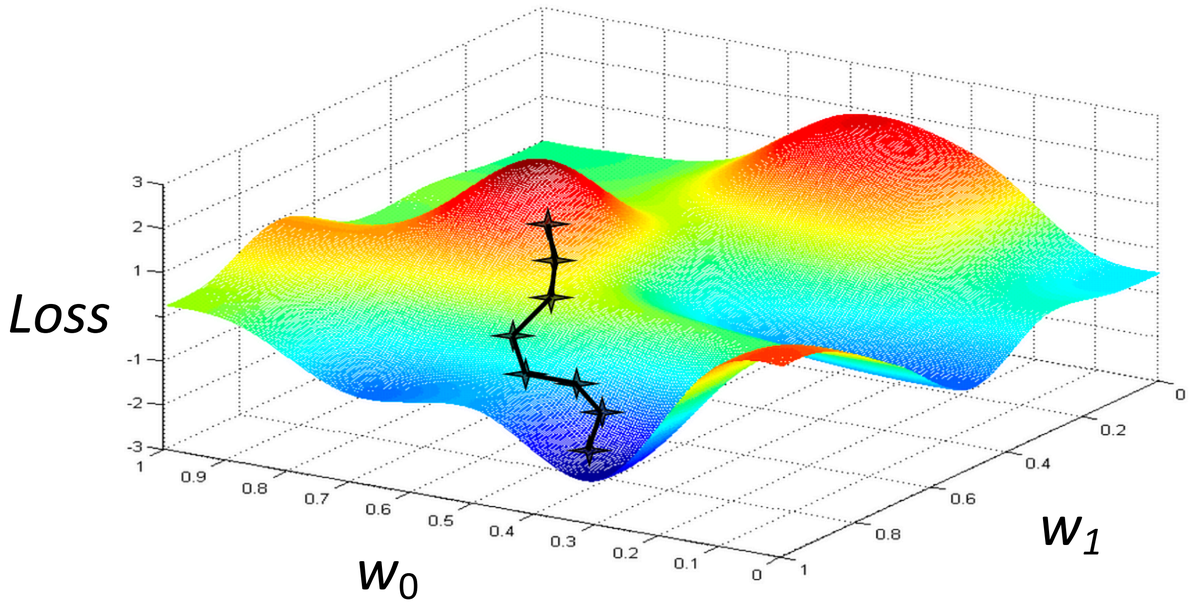
(a) A shallow model, such as linear regression, has short computation paths between inputs and output. (b) A deep learning network has longer computation paths, allowing each variable to interact with all the others.

Training Neural Networks

The network learns by adjusting the values of the parameters (weights and biases) so that the network as a whole fits the training data.

We want to find the network parameters that minimise the loss function, which measures the difference between the network's predicted output and the true target value.

Gradient Descent can be used for this:
calculate the gradient of the loss function with respect to the weights, and adjust the weights along the gradient direction to reduce the loss.



Gradient descent illustration

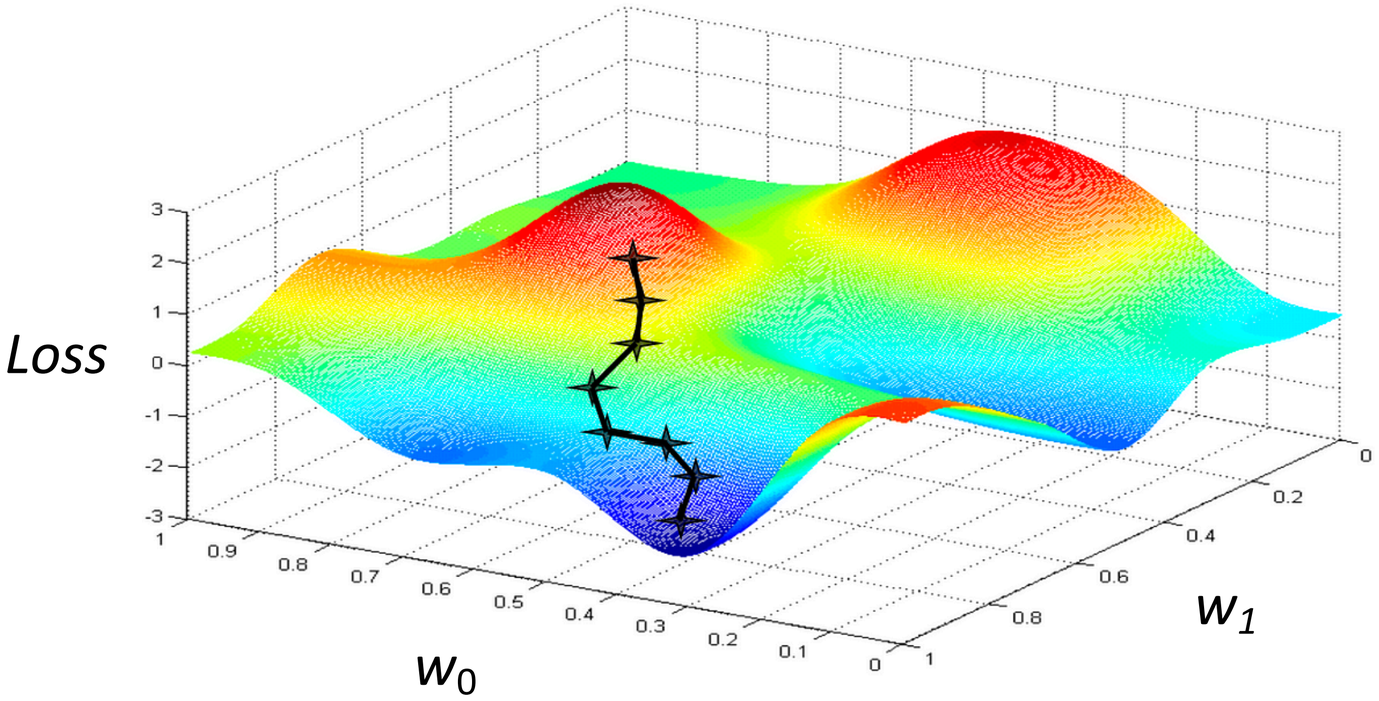
Gradient Descent

With gradient descent we can search through a continuous weight space by incrementally modifying the parameters in order to minimise the loss.

$\mathbf{w} \leftarrow$ any point in the parameter space
while not converged **do**
  for each w_i **in** $\mathbf{w}$ **do**

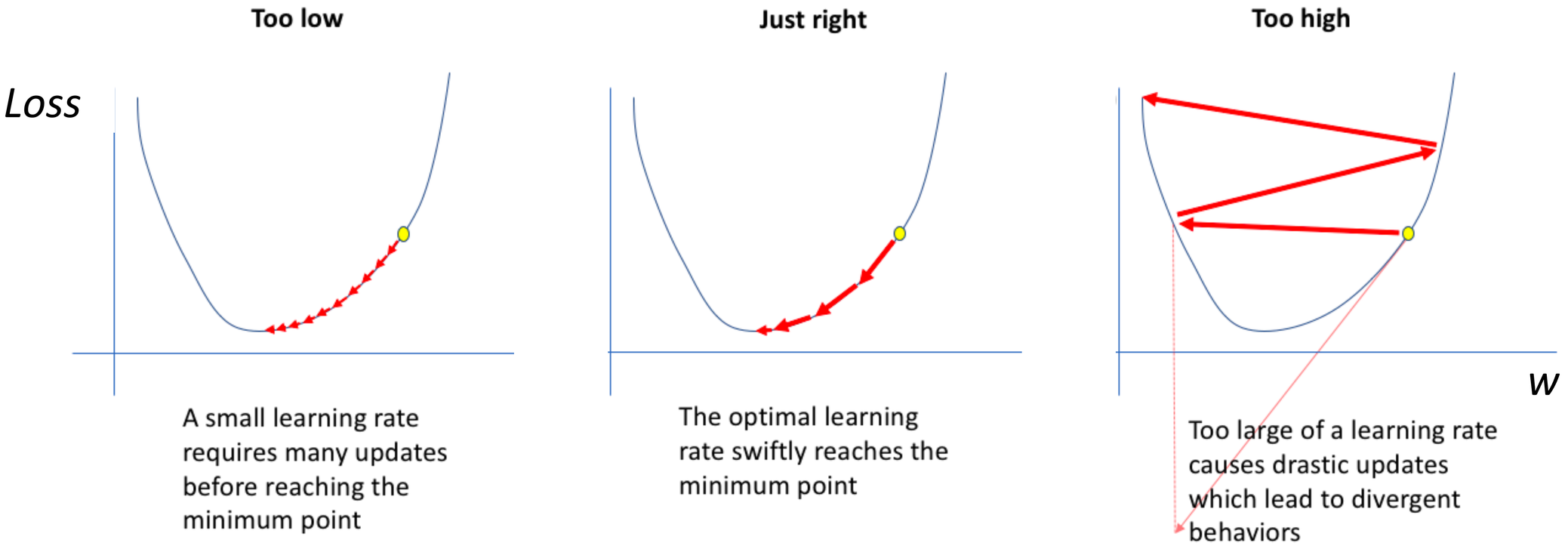
$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(\mathbf{w})$$

α is a parameter called a learning rate that can be a fixed constant or decay over time



Example: Gradient descent illustration

Gradient Descent



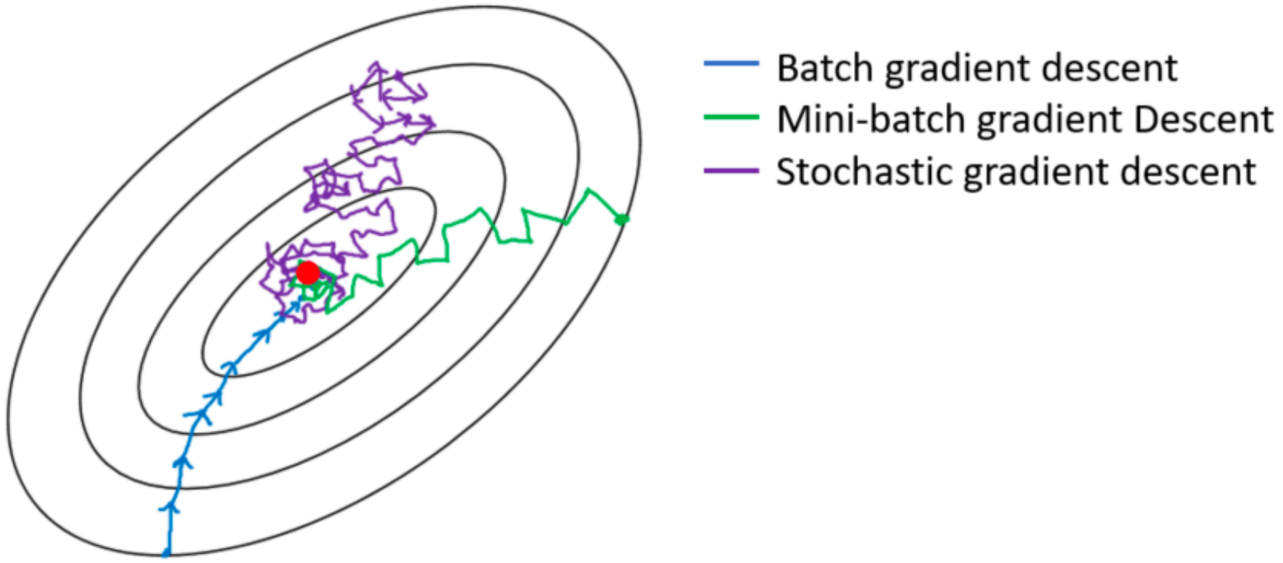
Example: Different values of the learning rate α

Stochastic Gradient Descent

Batch gradient descent (also called deterministic gradient descent) takes into account all *N* training examples for every step, thus it could be stow. A step that covers all the training examples is called an epoch.

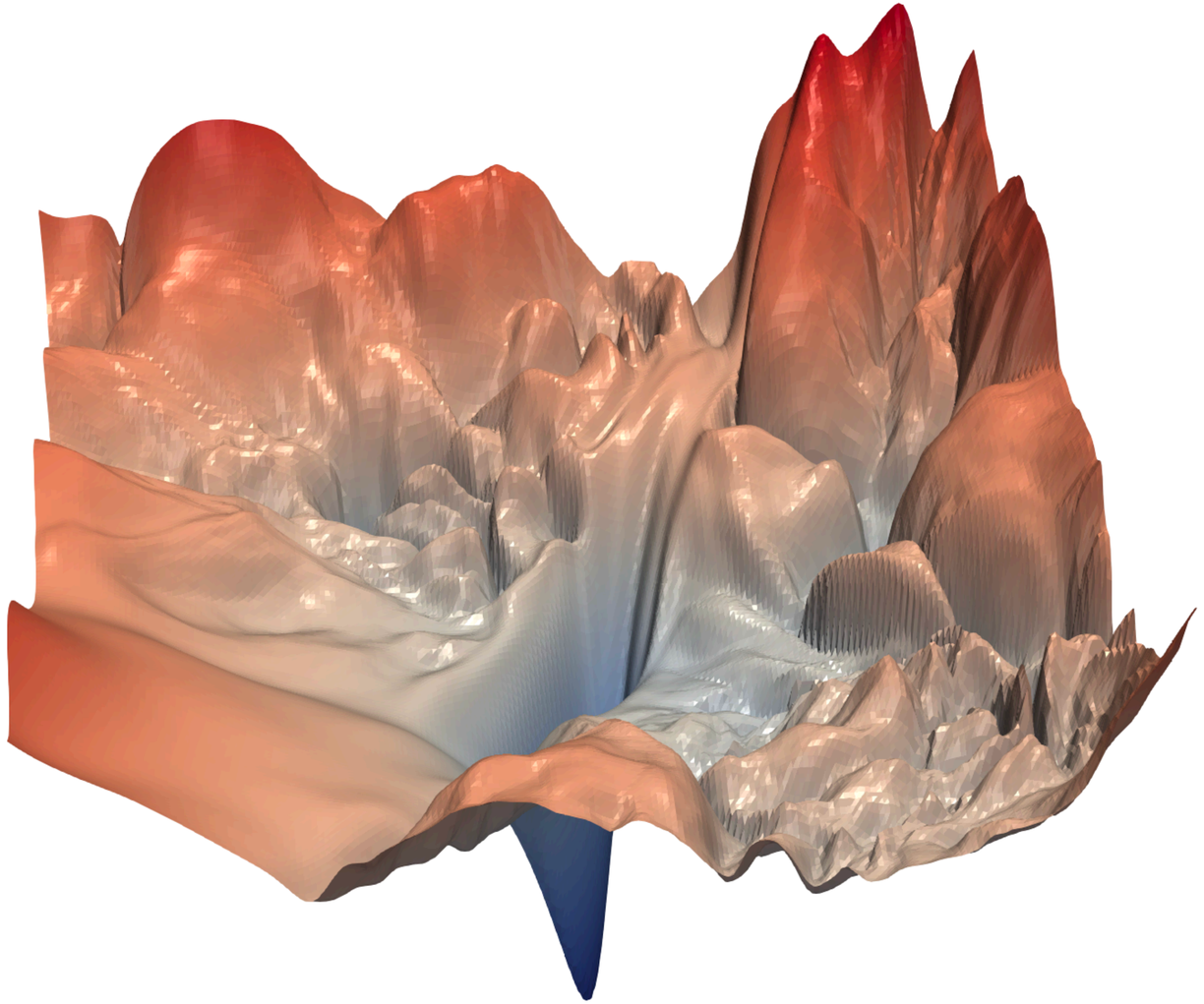
A faster variant is called stochastic gradient descent (SGD): it randomly selects a small number of training examples at each step.

Original SGD selects only one training example at each step, mini-batch selects *m* out of *N* training examples.



Example: Convergence of mini-batch SGD is not strictly guaranteed; it can oscillate around the minimum without settling down.

Optimization of the Loss Function in Reality



Optimisation landscape for an example deep neural network[1]

[1] Li, H., Xu, Z., Taylor, G., Studer, C. and Goldstein, T., 2018. Visualizing the loss landscape of neural nets. *Advances in neural information processing systems*, *31*.

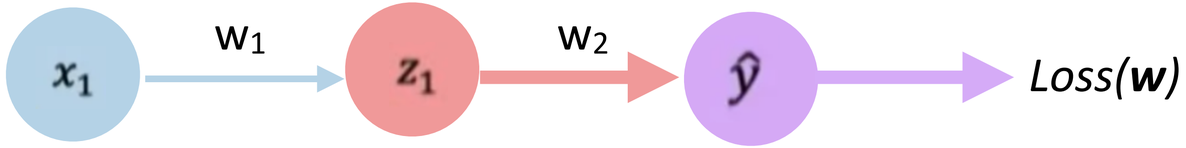
Computing Gradients: Backpropagation

Backpropagation (backward propagation of error) is a training algorithm for computing the gradients, by adjusting the network parameters in order to minimize the loss function

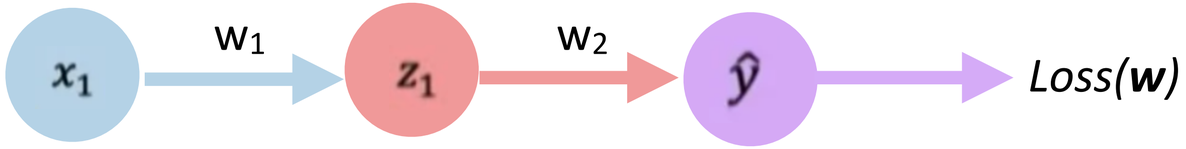
Gradient descent pseudocode:

$\mathbf{w} \leftarrow$ any point in the parameter space
while not converged **do**
  for each w_i **in** $\mathbf{w}$ **do**

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(\mathbf{w})$$



$$\frac{\partial J(\boldsymbol{W})}{\partial w_2} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_2}$$



$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

Chain rule: If $y = u(v(x))$, then $\frac{dy}{dx} = \frac{du}{dv} \cdot \frac{dv}{dx}$

*Loss(**w**) = J(**W**)

Computing Gradients: Backpropagation

Algorithm:

1. Forward Pass:

The input data is passed through the network layer by layer.

Each layer performs a weighted sum and applies an activation function to produce the output.

A loss function is used to measure the discrepancy between the predicted and actual outputs.

2. Backward Pass:

Starting from the output layer, the gradient of the loss function with respect to the output is computed.

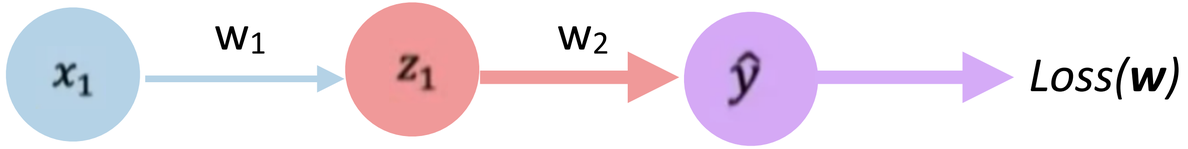
The chain rule is applied iteratively to compute the gradient of the loss function with respect to each layer's parameters and inputs.

Gradients are propagated backward through the network.

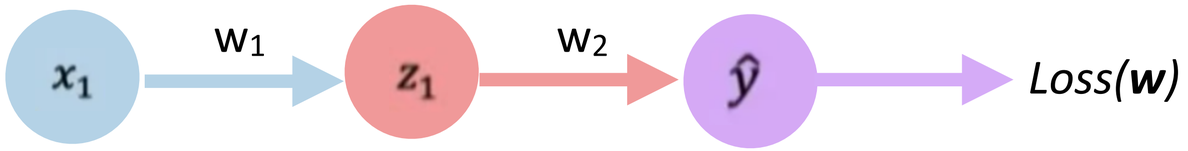
3. Parameter Update:

The computed gradients are used to update the network's parameters through optimization algorithms like gradient descent.

This process is repeated for a predefined number of iterations or until convergence.



$$\frac{\partial J(\boldsymbol{W})}{\partial w_2} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_2}$$



$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

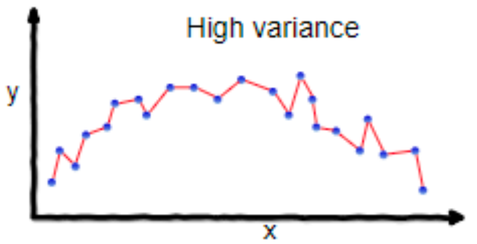
*Loss(**w**) = J(**W**)

Generalization

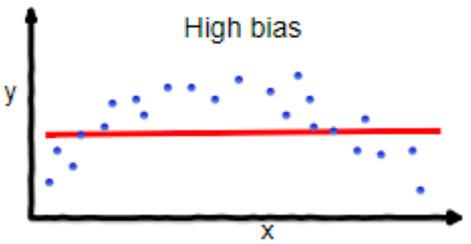
Goal is to fit a neural network to a training set, that is able to generalize well to new data that have not been seen previously, as measured by performance on a test set.

Several approaches to improving generalization performance:

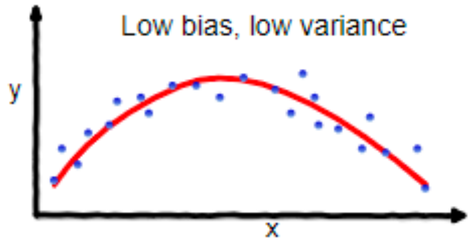
- choosing the right network architecture
- dropout
- early stopping



overfitting



underfitting



Good balance

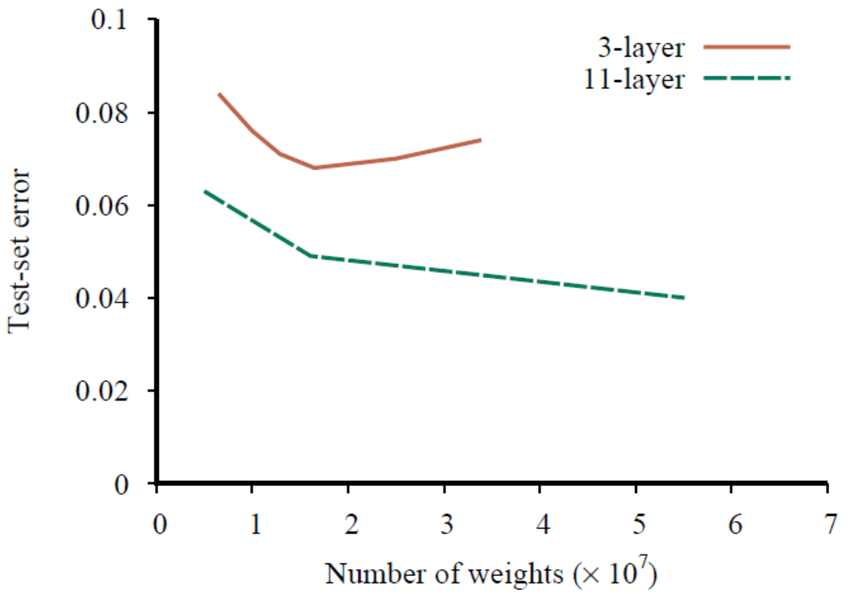
Choosing a network architecture

The generalization performance is dependant on the type of network architecture and the varying number of layers, their connectivity, and the types of node in each layer.

Some neural network architectures are explicitly designed to generalize well on particular types of data

-ex. Convolutional Neural Networks (CNN) for images, Recurrent Neural Networks (RNN) for text and audio signals

When comparing two networks with similar numbers of weights, the deeper network usually gives better generalization performance.



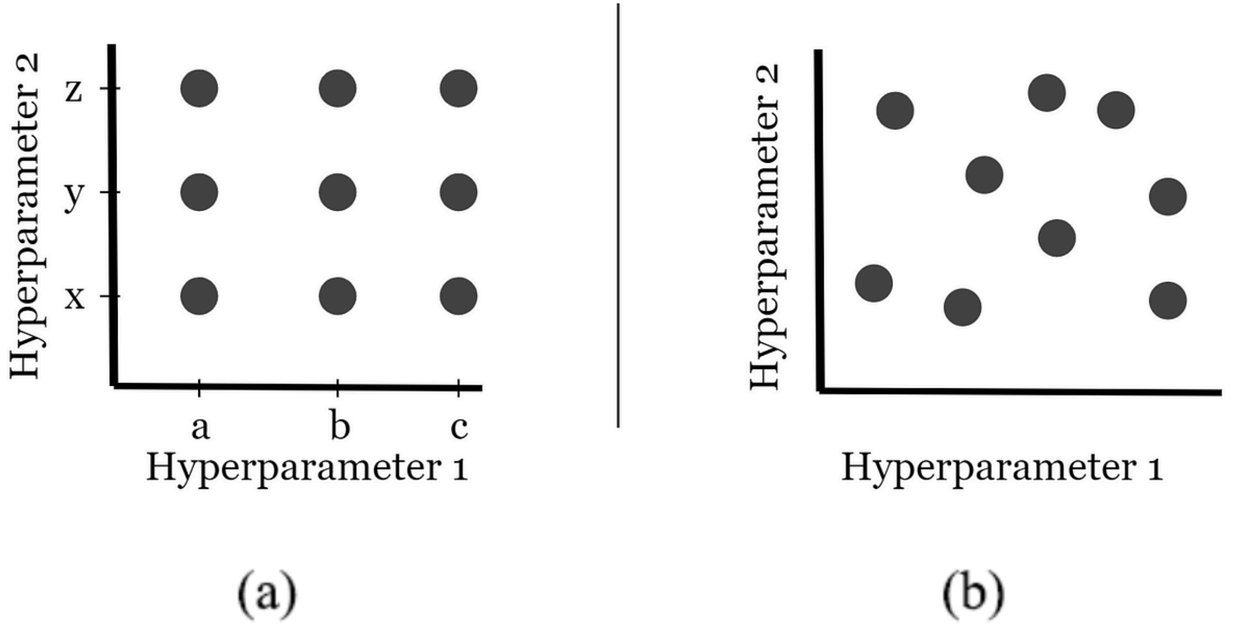
Test-set error as a function of layer width (as measured by total number of weights) for three-layer and eleven-layer convolutional networks. The data come from early versions of Google's system for transcribing addresses in photos taken by Street View cars (Goodfellow et al., 2014).

Choosing a network architecture

The process of architecture selection can be seen as a case of hyperparameter tuning, where the hyperparameters determine the depth, width, connectivity, and other attributes of the networ

Ideas for speed up this estimation process by eliminating or at least reducing the expensive training process.:

- train on a smaller data set
- train for a small number of batches and predict how the network would improve with more batches
- use a reduced version of the network architecture retains the properties of the full version



Example: Hyperparameter tuning using (a) Grid Search and (b) Random Search

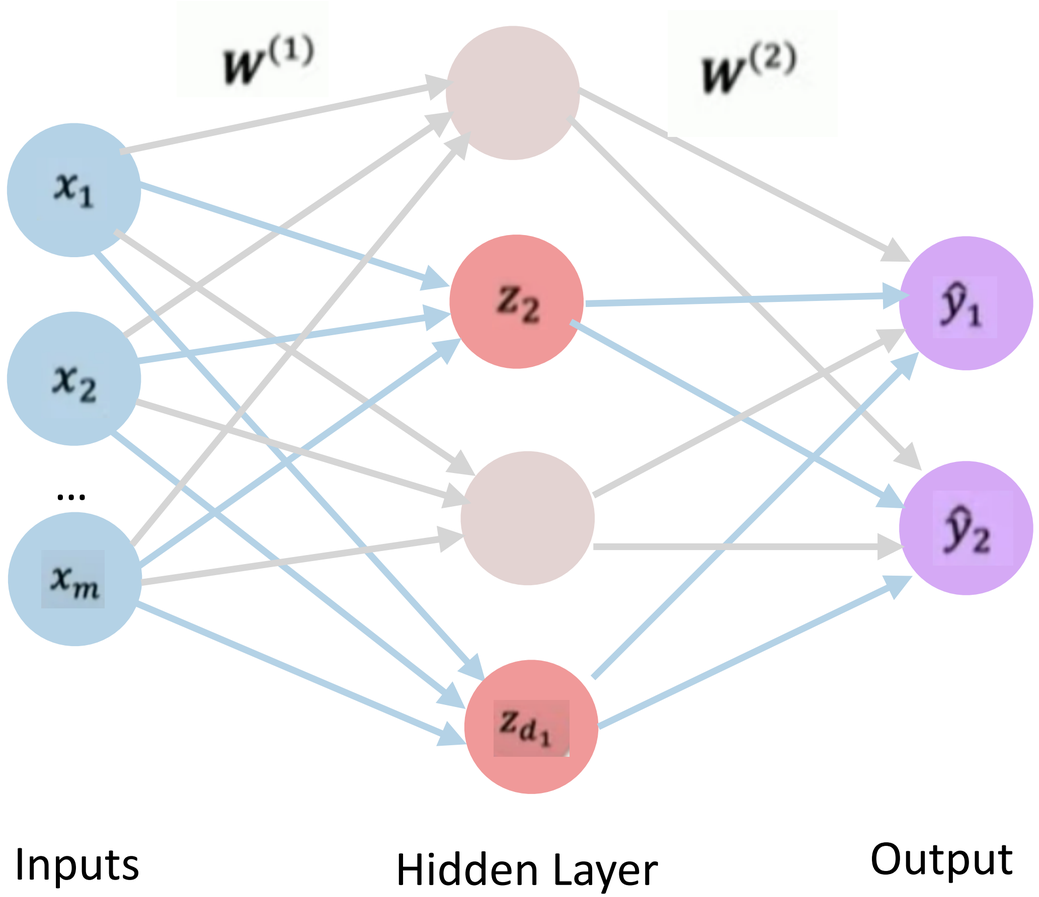
Dropout

At each step of training, dropout applies one step of back-propagation learning to a new version of the network that is created by deactivating a randomly chosen subset of the units.

Dropout approximates the creation of a large ensemble of different networks.

Intervene to reduce the test-set error of a network—at the cost of making it harder to fit the training set.

Dropout forces the model to learn multiple, robust explanations for each input



Dropout

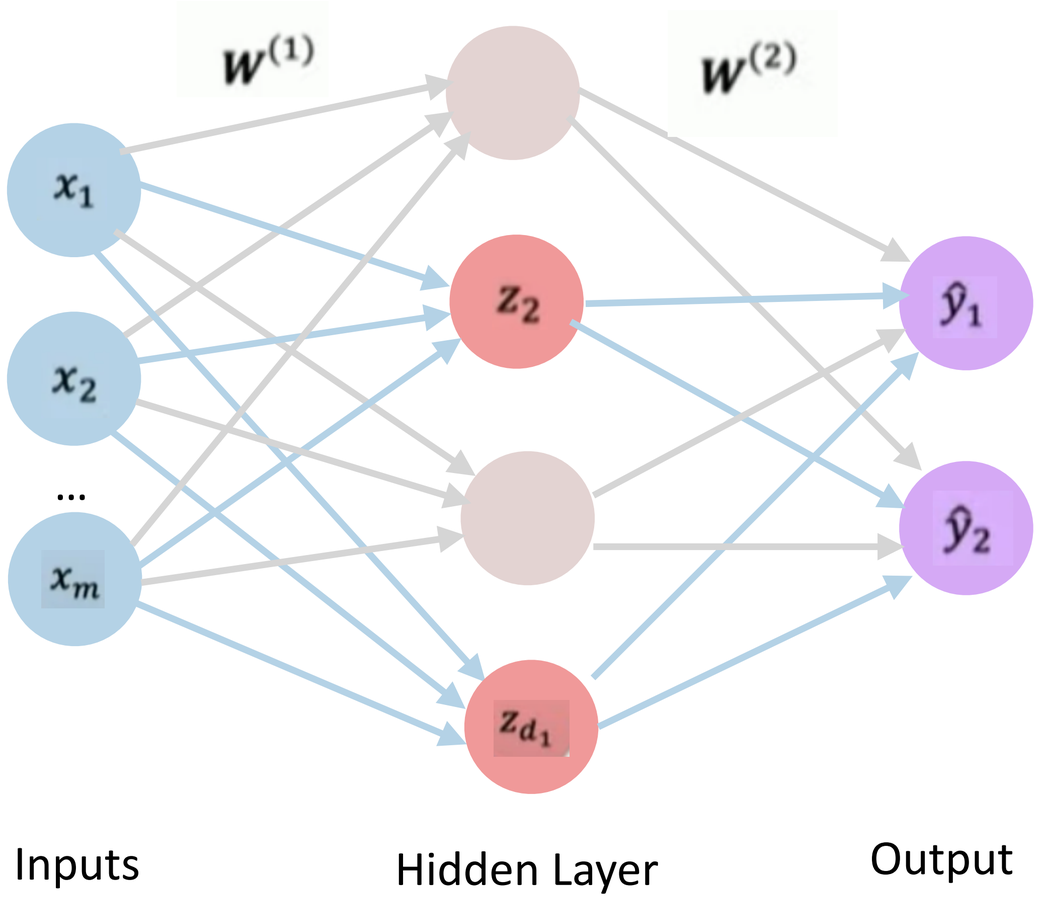
Algorithm:

For each minibatch, with probability *p*, the unit output is multiplied by a factor of 1/*p*; otherwise, the unit output is fixed at zero.

This process produces a thinned network with less units then the original, to which back-propagation is applied with the minibatch of *m* training examples.

The process repeats in the usual way until training is complete.

At test time, the model is run with no dropout.



Early Stopping

Early stopping addresses this issue of overfitting by monitoring the model's performance on a validation set during training and stopping the training process when the performance starts to degrade, indicating that the model is beginning to overfit.

Appendix