CS286: AI for Science and Engineering

Lecture 3: Artificial Neural Network and Deep Learning

Jie Zheng (郑杰)

PhD, Associate Professor

School of Information Science and Technology (SIST), ShanghaiTech University

Fall, 2023

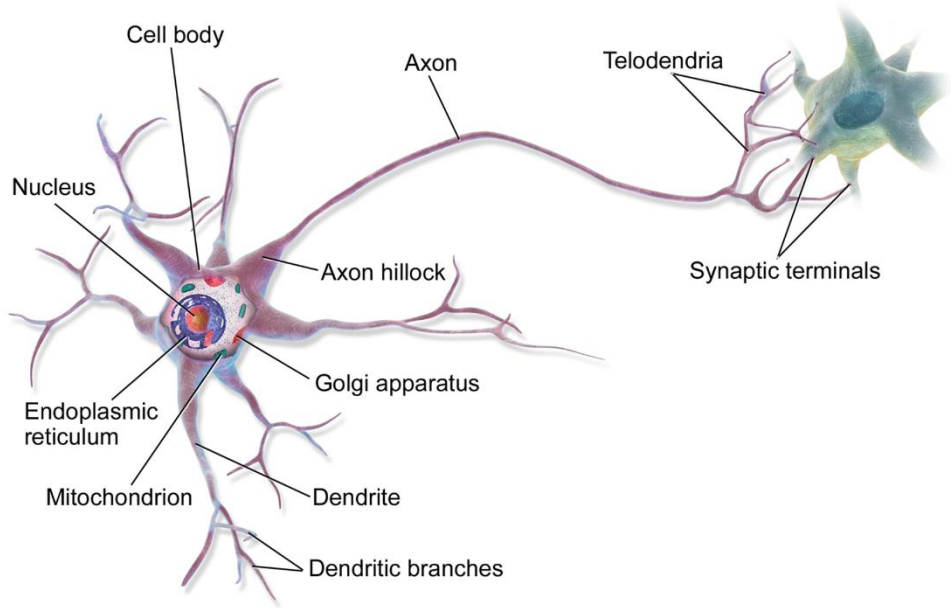
Learning objectives

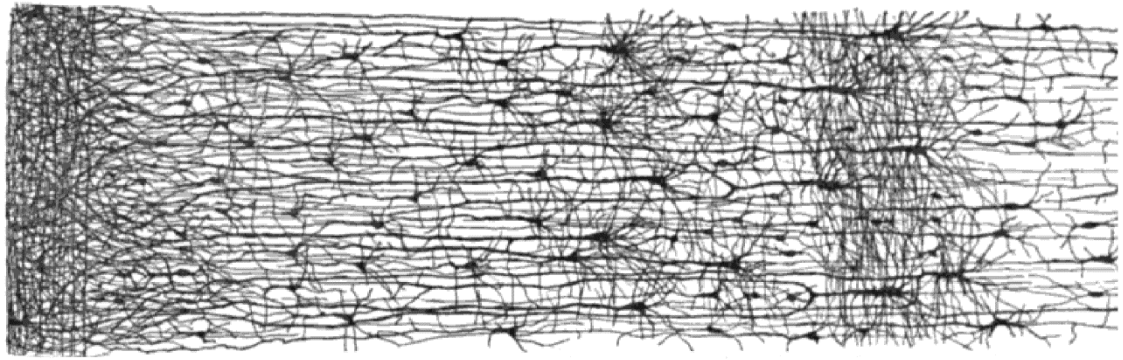
- After taking this lecture, you should be able to:
  - Describe the biological origin and history of artificial neural network
  - Describe challenges in training a deep neural network (e.g. vanishing / exploding gradients, overfitting)
  - Explain how to avoid overfitting through dropout

From Biological to Artificial Neurons

Biological neurons

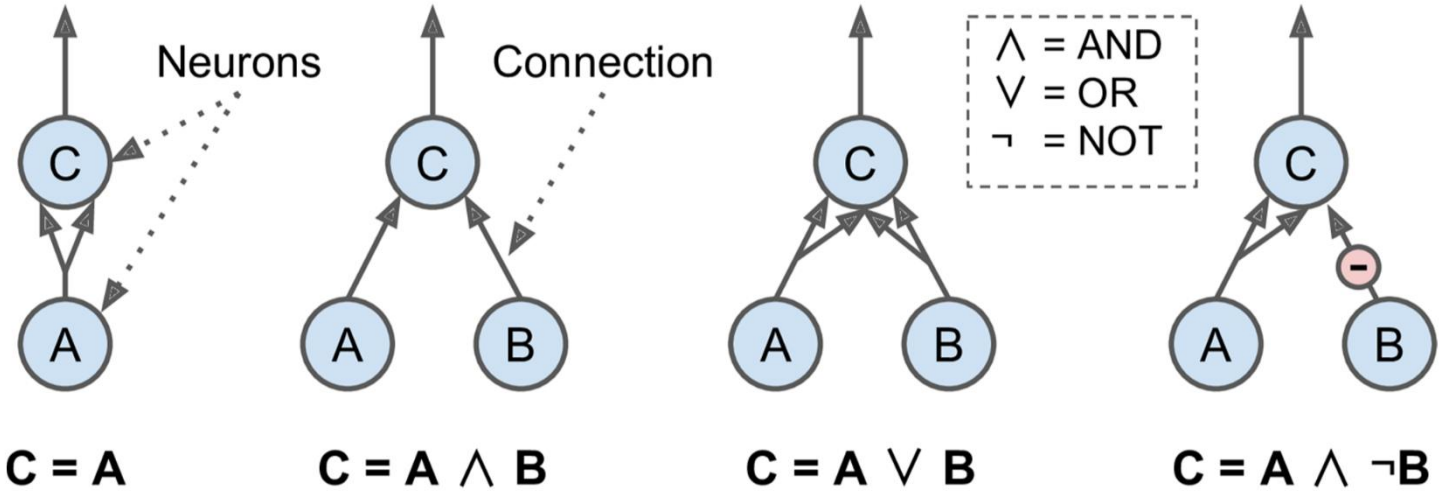
- Components in a biological neuron
  - Cell body
  - Dendrite
  - Axon
  - Synapses
- Communications among neurons
  - Neurons receive signals (i.e. short electrical impulses) from other neurons via synapses
  - When a neuron receives an enough number of signals, it fires its own signals
- Biological neural networks:
  - Neurons are organized in consecutive layers





Logical computations with neurons

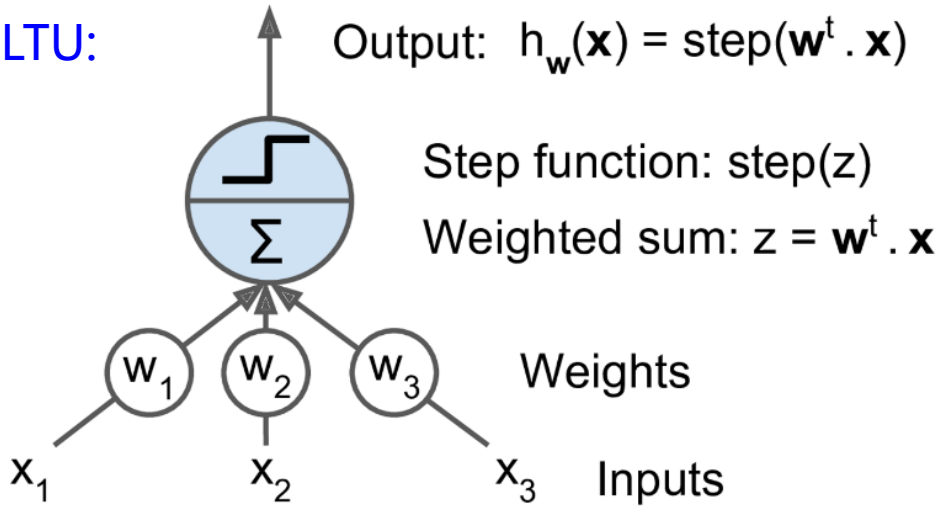
- **Artificial neural network** (McCulloch and Pitts, 1943): A simplified computational model of how biological neurons might work together in animal brains to perform computations using propositional logic
- An **artificial neuron**:
  - has one or more binary (on/off) inputs and one binary output
  - activates its output when at least a certain number (e.g. two in the example below) of its inputs are active



Linear threshold unit (LTU)

- A **linear threshold unit (LTU)** is a type of artificial neuron, where the inputs and output are numbers, instead of binary on/off values
  - Compute a **weighted sum** of inputs
  - Apply a **step function** to the sum to get the output

LTU:



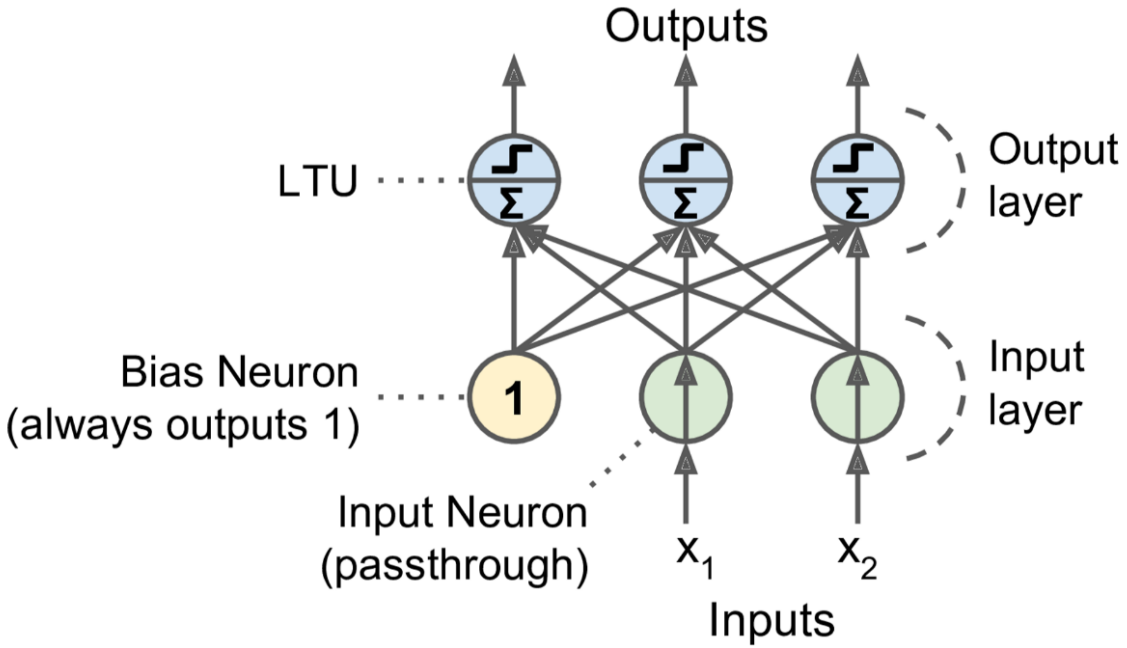
Common step functions:

$$\text{heaviside}(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases} \qquad \text{sgn}(z) = \begin{cases} -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ +1 & \text{if } z > 0 \end{cases}$$

Perceptron

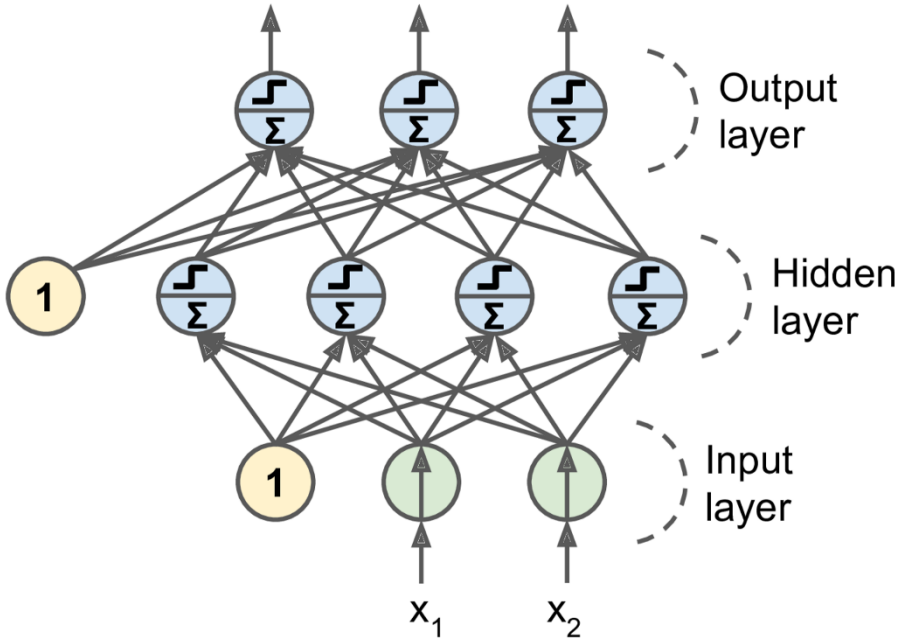
- A **perceptron** (invented by Frank Rosenblatt in 1957) is a simple ANN composed of a single layer of LTUs
  - Each neuron is connected to all the inputs
  - **Input neurons** output whatever input they are fed
  - **Bias neuron**: output 1 all the time
- A Perceptron is trained using **Hebb's rule**:
  - Between two neurons having the same output, increase the connection weight
  - Not to reinforce connections that lead to wrong output

Perceptron diagram:



Multi-Layer Perceptron (MLP)

- Perceptrons are unable to solve some trivial problems (e.g. the XOR classification problem)
- But some limitations of Perceptrons can be overcome by stacking multiple Perceptrons, resulting in the **Multi-Layer Perceptron (MLP)**
- An MLP is composed of
  - One **input layer**
  - One or more **hidden layers** (of LTUs)
  - One final **output layer** (of LTUs)



A Multi-Layer Perceptron

Deep neural network (DNN)

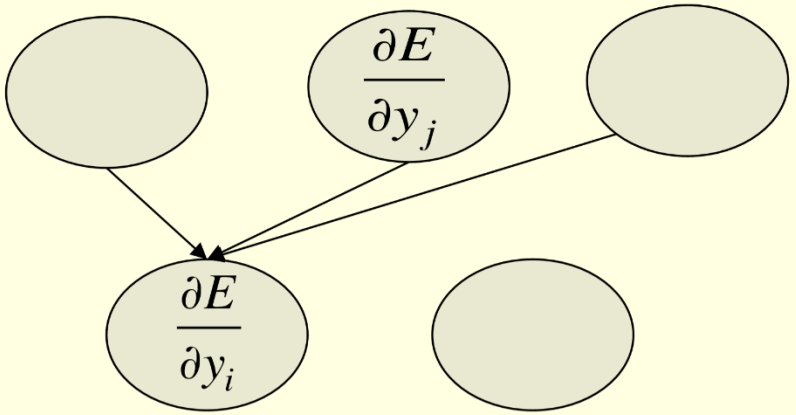
- When an ANN has two or more hidden layers, it is called a **deep neural network (DNN)**
- However, it was difficult to train MLPs for many years, until the introduction of the **backpropagation** algorithm in 1986

Backpropagation

- For each training instance, the backpropagation algorithm goes through 3 steps:
  - **Forward pass**: feed the training instance to the network, and compute the output of every neuron in each consecutive layer, i.e. to make a prediction
  - **Reverse pass**: measure the error (i.e. the difference between the desired output and the actual output), and go through each layer **in reverse** to measure the error contribution from each neuron in the previous hidden layer, until reaching the input layer
  - **Gradient descent**: slightly tweak the connection weights to reduce the error

$$E = \frac{1}{2} \sum_{j \in output} (t_j - y_j)^2$$

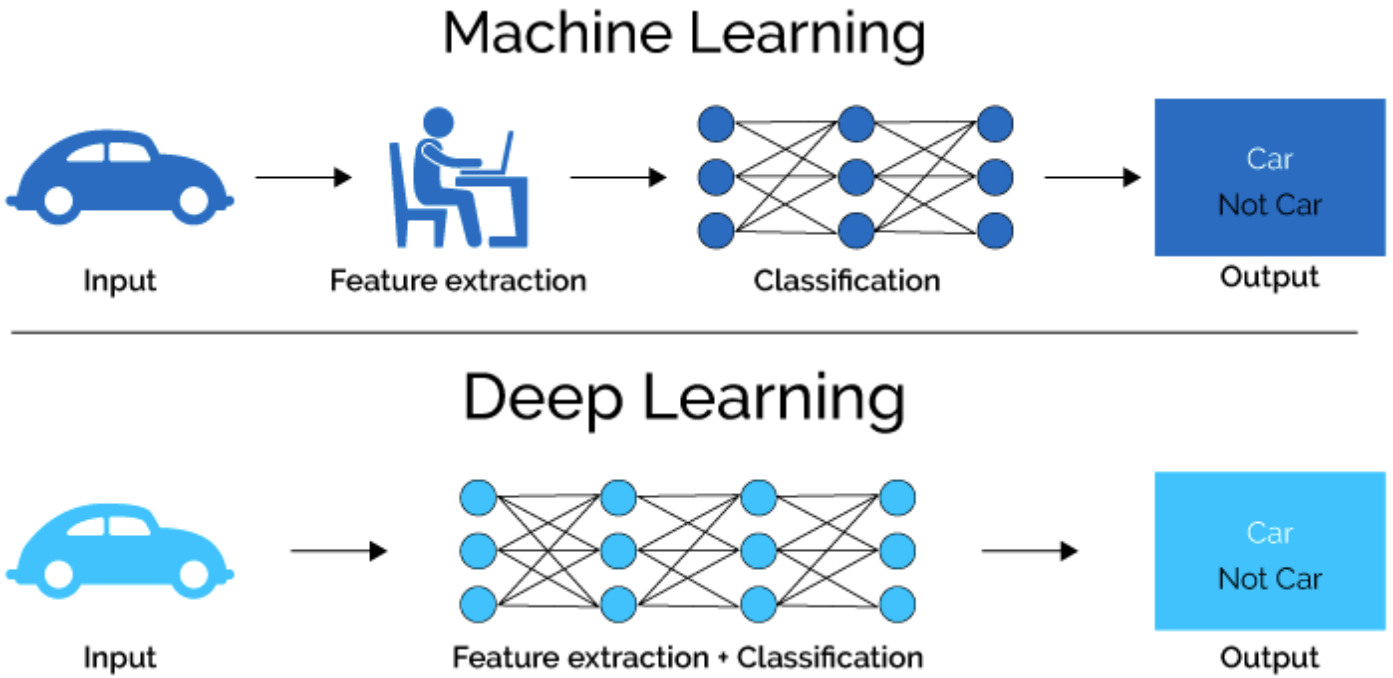
$$\frac{\partial E}{\partial y_j} = -(t_j - y_j)$$



Error derivative in each hidden layer can be computed from the error derivatives in the layer above (i.e. backpropagating dE/dy)

Training Deep Neural Networks

Deep learning vs. traditional machine learning



- **Feature engineering**
  - ML often requires complex feature engineering by human experts
  - DL eliminates or reduces the need of feature extraction, but more costly in hardware and data
- **Interpretability**
  - An ML model is easier to interpret, i.e. understand how and why it works
  - A DL model is often seen as a "**black box**"; intensive ongoing research to open it

Deep learning vs. traditional machine learning

上海科技大学 ShanghaiTech University

Traditional machine learning

Input → **Feature extractor** → **Machine learning model** → **Output**

(e.g., )

(e.g., color and texture histograms)

(e.g., support vector machines and random forests)

(e.g., presence or not or disease)

Deep learning

Directly learns what are useful (and better!) features from the training data

Input → **Deep Learning Model** → **Output**

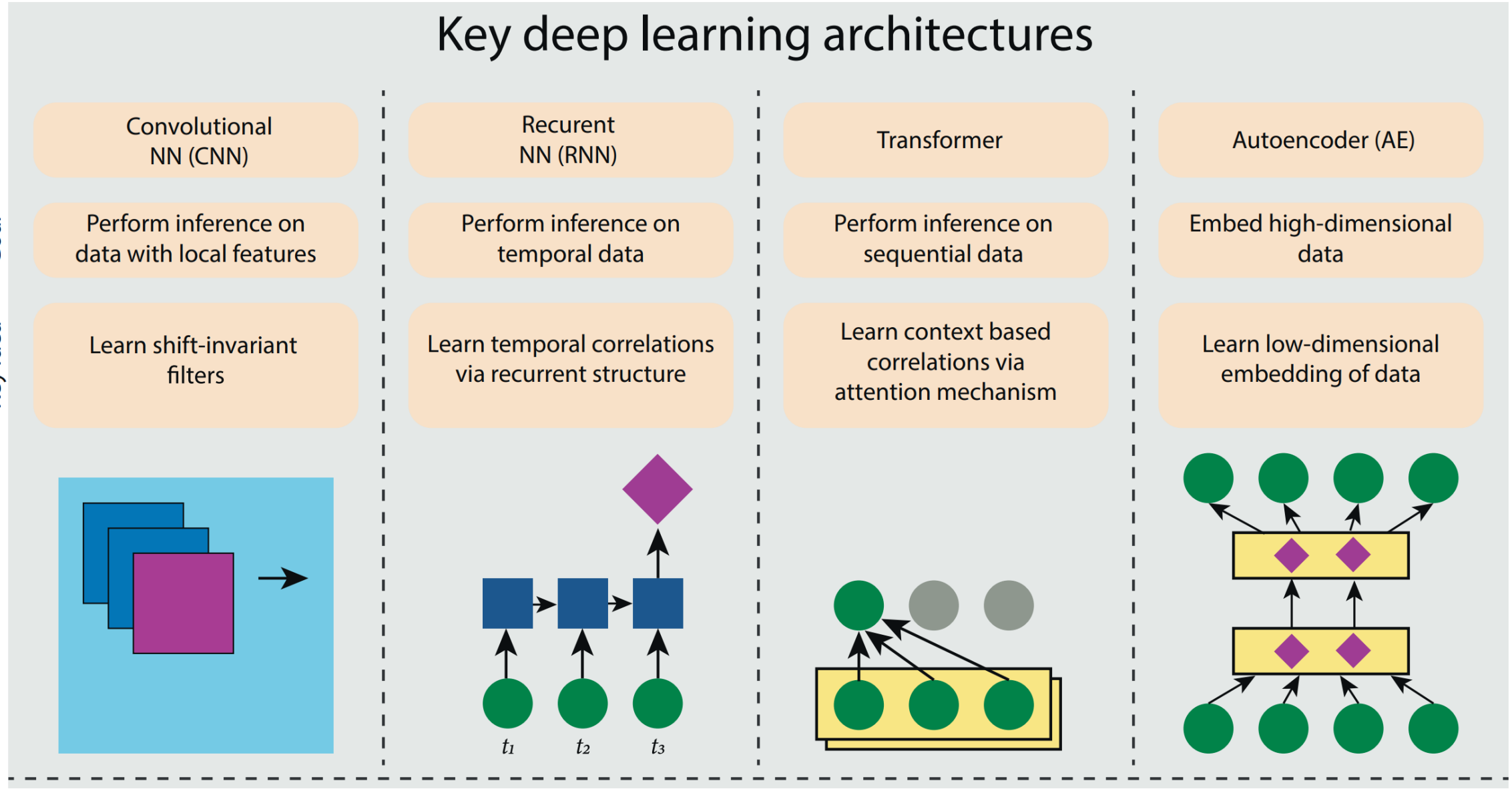
(e.g., )

(e.g., convolutional and recurrent neural networks)

(e.g., presence or not or disease)

Slide from Serena Yeung, BIODS220, Stanford

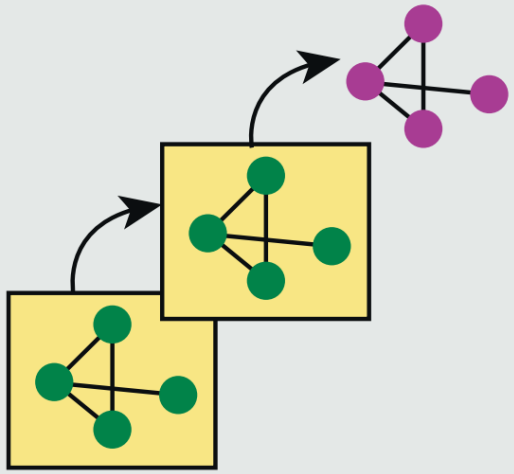
Popular deep learning architectures

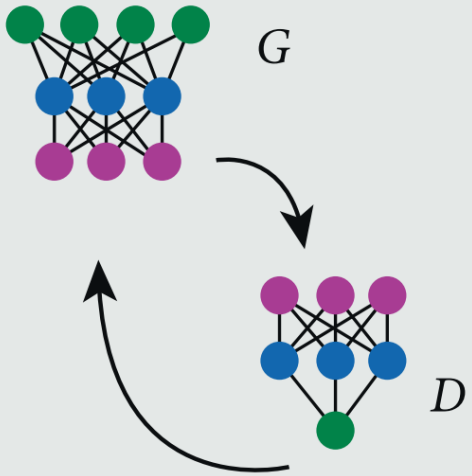


Sapoval et al. Nat. Commun. 2022

Popular deep learning architectures

	Graph NN (GNN)	Generative Adversarial Network (GAN)
Goal	Capture graph based dependencies in the data	Generate samples from data distribution
Key idea	Perform message passing between nodes in a layer	Simultanoeusly train generator and discriminator





Denoising autoencoders (DAE) are autoencoder models that learn low dimensional embeddings of noisy high dimensional data, i.e. inputs that differ by a small amount of noise give rise to a similar embedding vector.

Attention mechanism mimics cognitive attention by learning importance weights for the inputs based on the whole input context (e.g. in a task of translating codons to amino acids attention mechanism will learn to give higher weight to the first two nucleid acids). Attention is the key part of transformer models, but can also be applied in conjunction with other layer types.

Convolutional layers have dimension which indicates the dimension of learned filters. Thus, we can have a 1-dimensional convolutional layer for sequences, 2-dimensional layer for matrices, and so on.

Graph convolutional network (GCN) is a graph neural network with convolutional layers defined by the topology of the graph. Thus instead of passing neighboring sequence or matrix entries through a filter, graph defined neighborhoods are used.

Sapoval et al. Nat. Commun. 2022

Challenges of training deep neural networks

- The **vanishing / exploding gradients** problems which make lower layers very hard to train
- There are **not enough training data**, or it is too costly to label the data
- Training may be extremely **slow**
- The risk of **overfitting** the training set, when
  - the model has many parameters,
  - there are not enough training instances, or
  - the data are too noisy

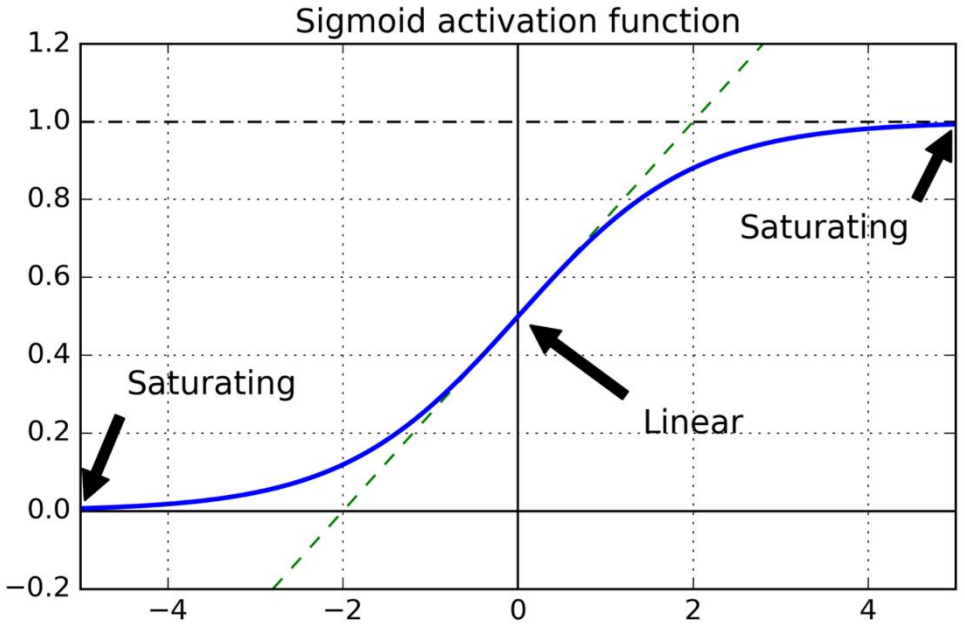
Vanishing/exploding gradients problems

- **Vanishing gradients** problem: In the backpropagation to lower layers, gradients often get smaller and smaller
  - Gradient Descent update leaves the lower-layer weights virtually unchanged, and training cannot converge to a good solution

- **Exploding gradients** problem: The gradients grow bigger and bigger
  - Many layers get large weight updates
  - The algorithm diverges
  - Mostly happens in recurrent neural networks

Vanishing/exploding gradients problems

- Reasons found by Xavier Glorot and Yoshua Bengio, around 2010:
  - With the combination of random initialization with a normal distribution and logistic sigmoid activation function, the variance of outputs of each layer is much bigger than the variance of its inputs
  - The activation function tends to saturate at the top layers



Saturation of logistic activation function

Vanishing/exploding gradients problems

- To alleviate the vanishing/exploding gradients problems, we need the signals to flow properly in both directions of backpropagation:
  - The variance of the outputs of each layer should be equal to the variance of its inputs
  - The gradients should have equal variance before and after flowing through a layer in the reverse direction

Glorot and He initialization

- **Glorot initialization** (or **Xavier initialization**)
  - For each layer, the numbers of input and output neurons are called $\mathbf{fan_{in}}$ and $\mathbf{fan_{out}}$ of the layer
  - The connection weights of each layer must be initialized randomly as described in the equations on the right side, where

$$fan_{avg} = (fan_{in} + fan_{out})/2$$

Glorot initialization with logistic activation function

Normal distribution with mean 0 and variance $\sigma^2 = \frac{1}{fan_{avg}}$

Or a uniform distribution between $-r$ and $+r$, with $r = \sqrt{\frac{3}{fan_{avg}}}$

- **LeCun initialization**: Replace fan_{avg} with fan_{in} in the equations of Glorot initialization
- **He initialization**: For RELU and its variants

Initialization for different activation functions

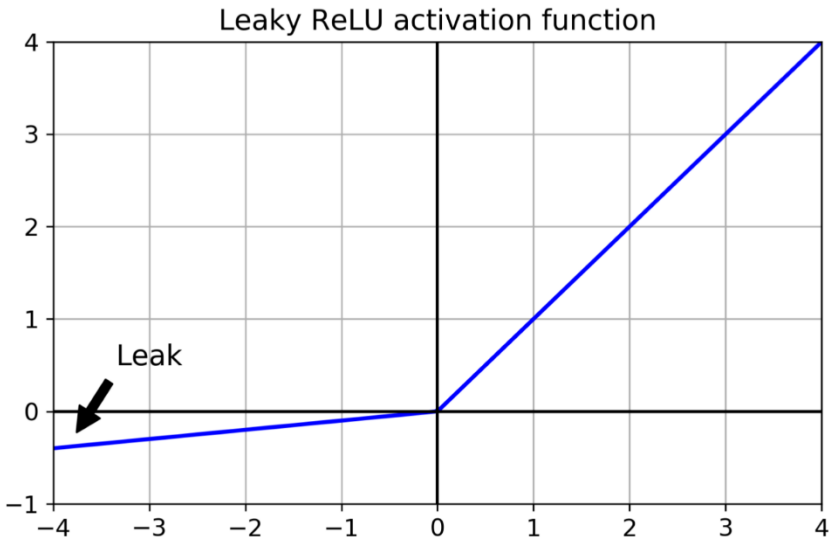
Initialization	Activation functions	σ^2 (Normal)
Glorot	None, Tanh, Logistic, Softmax	$1 / fan_{avg}$
He	ReLU & variants	$2 / fan_{in}$
LeCun	SELU	$1 / fan_{in}$

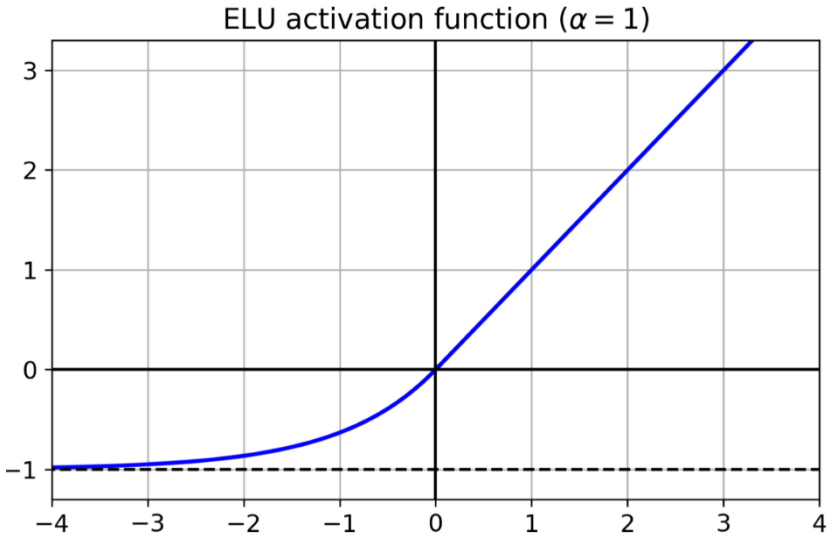
Nonsaturating activation function

- The **ReLU (Rectified linear unit)** activation function behaves better than sigmoid functions, because:
  - It does not saturate for positive values
  - It is fast to compute
- But ReLU suffers the problem of **dying ReLUs**, i.e. during training some neurons die, outputting only 0
- Variants of ReLU:
  - **Leaky ReLU**: $LeakyReLU_\alpha(z) = \max(\alpha z, z)$, where α is a hyperparameter that defines how much it "leaks", and α is usually set to 0.01
  - **RReLU (randomized leaky ReLU)**: α is picked randomly in a given range during training, and fixed during testing
  - **PReLU (parametric leaky ReLU)**: α is to be learned during training as a parameter (modified by backpropagation)
- **ELU (exponential linear unit)** activation function:

$$\mathrm{ELU}_\alpha(z) = \begin{cases} \alpha(\exp(z) - 1) & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases}$$

- **SELU (scaled ELU)**: make the network self-normalize





Batch Normalization (BN)

- How to reduce vanishing/exploding gradients problems during training?
- **Batch Normalization (BN)** is a technique (proposed in 2015):
  - Add an operation in the model just before or after the activation function of each hidden layer
  - Zero-center and normalize each input
  - Scale and shift the result using two parameter vectors per layer: one for scaling, the other for shifting
  - Evaluate the mean and standard deviation of each input over the current mini-batch

1. $\mu_B = \frac{1}{m_B}\sum_{i=1}^{m_B} \mathbf{x}^{(i)}$

2. $\sigma_B^2 = \frac{1}{m_B}\sum_{i=1}^{m_B} \left(\mathbf{x}^{(i)} - \mu_B\right)^2$

3. $\hat{\mathbf{x}}^{(i)} = \frac{\mathbf{x}^{(i)} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$

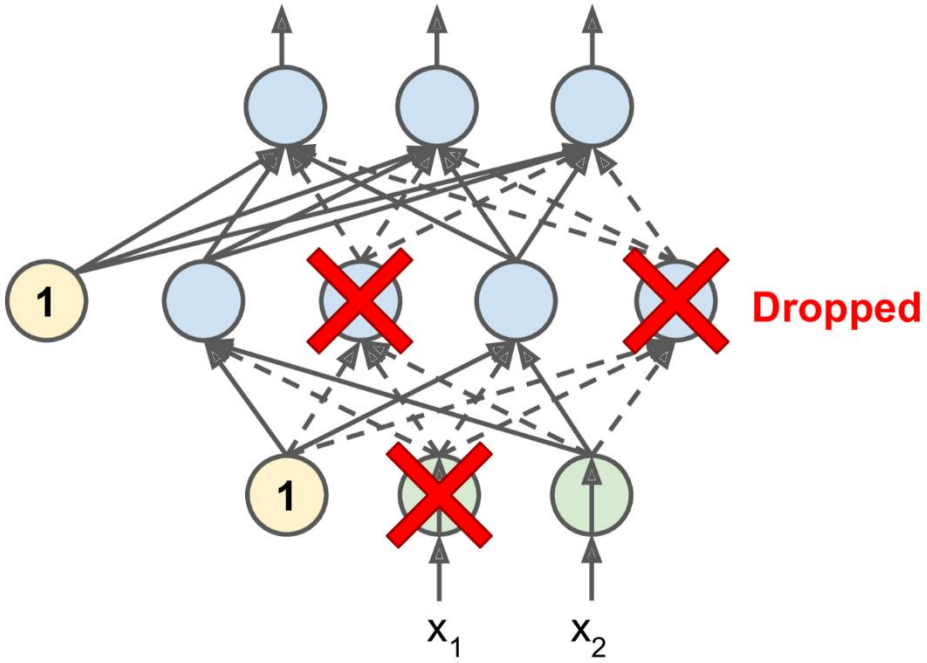
4. $\mathbf{z}^{(i)} = \gamma \otimes \hat{\mathbf{x}}^{(i)} + \beta$

Batch Normalization algorithm

Avoiding Overfitting through Regularization

Dropout

- The **dropout algorithm**:
  - At every training step, every neuron (except for output neurons) has a probability p of being temporarily "dropped out"
  - The hyperparameter p is called the **dropout rate**, and is typically set to 50%
  - After training, neurons don't get dropped any more



Why dropout works?

- Neurons trained with dropout cannot co-adapt with their neighboring neurons
- Neurons are forced to pay attention to **all** input neurons, and thus be less sensitive to small changes in the inputs

Other methods of regularization

- **L_1 and L_2 regularization**: Add L_1 and L_2 regularization loss to the final loss, to constrain the connection weights of a neural network

- **Early stopping**: stop training as soon as the validation error reaches a minimum

- **Max-Norm regularization**: Constrains the weights $\mathbf{w}$ of the incoming connections such that $\|\mathbf{w}\|_2 \leq r$, where r is the max-norm hyperparameter

- **Batch Normalization**

Recap

- In this lecture we have learned
  - Analogy between biological neurons and artificial neurons
  - Challenges of training deep neural networks
  - The vanishing / exploding gradients problems and techniques to address them
  - How to avoid overfitting by regularization techniques (e.g. dropout)

- For details, read Chapter 10 and Chapter 11 of Aurélien Géron’ s book “Hands-On Machine Learning with SciKit-Learn & TensorFlow”