

Lecture Title and Date

Deep Learning Fundamentals I: Artificial Neural Network, 2/24

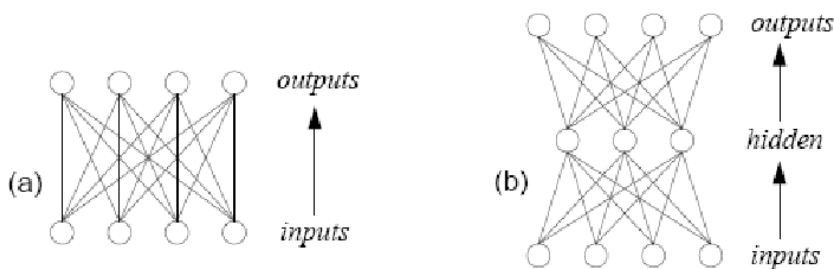
Objectives of the Lecture

- Understand what is “deep learning” and what is an artificial neural network (NNs)
 - Know applications of deep learning
- Describe the structure of a deep neural network
 - Explain how an output is generated from a neural network
- Understand how MLPs and NNs are trained through backward propagation
- Describe advantages, limitations, and key considerations (i.e. how many hidden layers to use) in designing and using deep NNs.

Key Concepts and Definitions

- **Neural network** - refers to a machine learning model designed to mimic artificial neurons that are interconnected and process information to arrive at some conclusion. Every neural network consists of layers of nodes, or artificial neurons—an input layer, one or more hidden layers, and an output layer. Each node connects to others, and has its own associated weight and threshold.
- **Multilayer perceptron** - a type of neural network with multiple layers of neurons, including an input layer, one or more hidden layers, and an output layer, allowing it to learn complex non-linear relationships, while a "single layer perceptron" only has one layer of neurons, limiting its ability to learn only simple linear patterns; essentially, an MLP is a more powerful model due to its additional hidden layers.

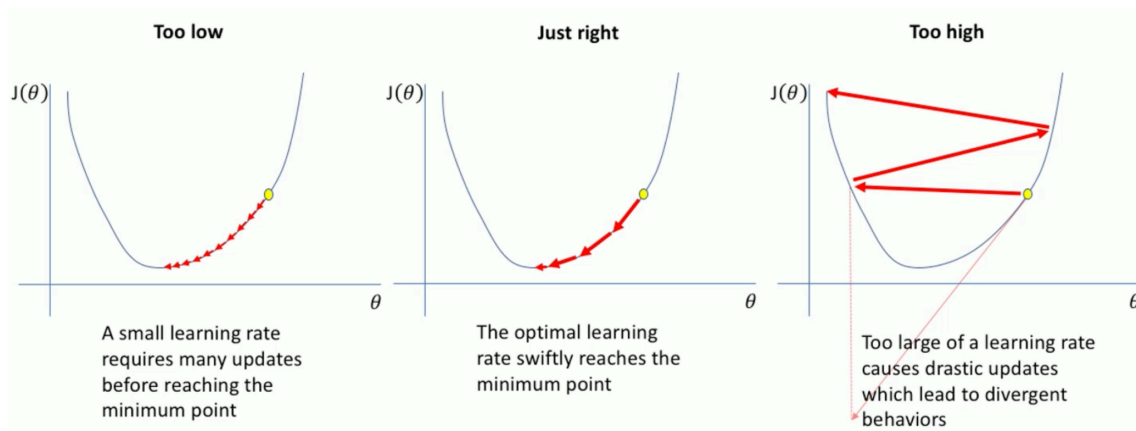
(a) = single-layer perceptron (b) multi-layer perceptron



- **Bias** - measure of ease in firing the perceptron. Correspond to the intercept in a linear model of the weighted inputs.
- **Gradient descent** - standard optimization algorithm that facilitates the search of parameters values that minimize the cost function towards a local minimum or optimal accuracy by calculating the gradient of a differentiable function and moving in the opposite direction of the gradient (i.e, backpropagation).
- **Backpropagation** - mechanism by which components that influence the output of a neuron (bias, weights, activations) are iteratively adjusted to reduce the cost function.

The backpropagation's operations calculate the partial derivative of the cost function with respect to the weights, biases, and previous layer activations to identify which values affect the gradient of the cost function.

- **Learning rate** - a hyperparameter chosen during model training that governs how much a machine learning model adjusts its parameters at each step of its optimization algorithm. It helps ensure that a model learns enough from training to make meaningful adjustments to its parameters while also not overcorrecting.



- **Dropout** - regularization technique designed to reduce overfitting in neural networks by randomly "dropping" a fraction of neurons during each training iteration. This is supposed to force the network to learn more robust features and not be overly reliant on any single neuron.
- **Convolutional neural network (CNN)** - a specific type of neural network that is particularly well-suited for analyzing visual data like images, where it leverages a process called "convolution" to extract features from the input data by focusing on a small region of the input, being the receptive field.

Main Content/Topics

A brief history on deep learning:

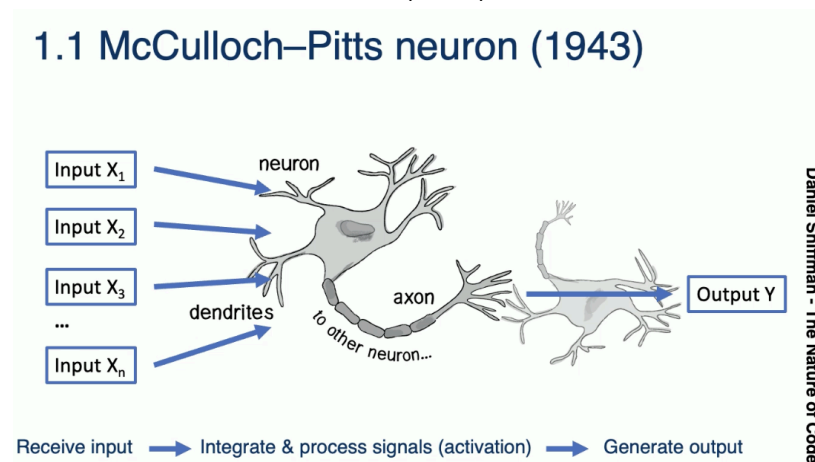
- (1997) **Deep Blue** - "artificial intelligence," 1997 IBM supercomputer that defeated chess world champion Garry Kasparov. The key difference between Deep Blue and modern artificial intelligence is that Deep Blue's parameters were all pre-defined by humans. Thus, Deep Blue just memorized millions of openings and end games, stored in a huge database, and computed positional combinations to beat Kasparov.
- (2016) **AlphaGo** - modern artificial intelligence computer that defeated Weiqi (Go) world champions Lee Sedol and Ke Jie. Rather than memorize positions and endgames, AlphaGo uses deep neural networks, reinforcement learning, plus a decision search tree to actually learn how to play Go. AlphaZero - iteration of AlphaGo that doesn't require human knowledge; Took just 3 days to train and only 3 weeks to beat the Go world champion. AlphaStar - deep learning model that can play StarCraft

- In the biomedical field: **AlphaFold**. The idea underlying AlphaFold was postulated first by Anfinsen (1972): a protein's amino acid sequence should fully determine its structure. However, there was a problem with us, best described by **Levinthal's paradox**: "It would take longer than the age of the known universe to enumerate all possible configurations of a typical protein by brute force. Yet in nature, proteins fold spontaneously and some within milliseconds" In other words, computing all possible configurations of an average protein would require huge computational resources & time. Using deep learning and improving from AlphaFold2, AlphaFold3 demonstrated more accurate single protein structures, protein complexes, and the effect of covalent modifications on protein structure. Note that complexes include protein-RNA interactions and protein-DNA interactions and covalent modifications include bonded ligands, glycosylation, modified protein residues, nucleic acid bases.
 - The AlphaFold3 architecture comprises:
 - Core component: Pairformer (**Self-attention & Transformer**)
 - **Diffusion-based (generative modeling)** approach
 - Replaces the **ReLU activation function** in the 48 transition blocks with **SwiGLU**

1. KEY Q: What is an artificial neural network?

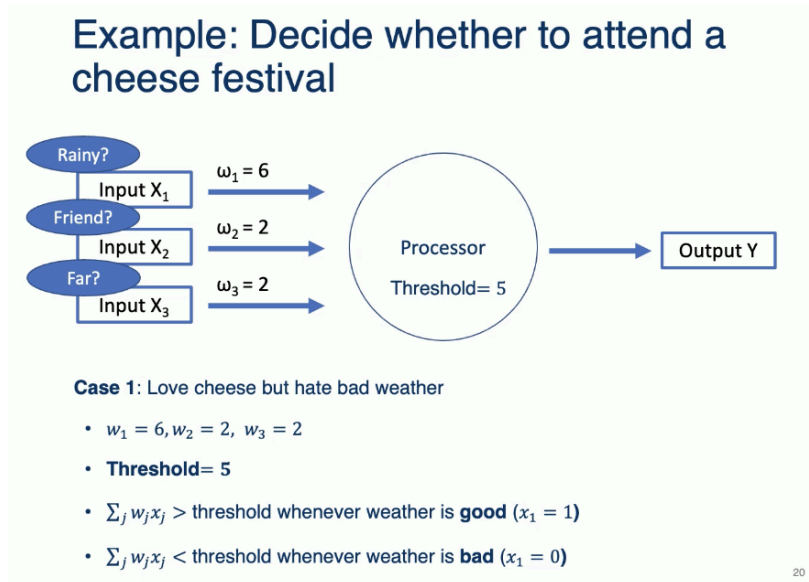
Conceptual background:

1.1. McCulloch-Pitts neuron (1943)



- Briefly, the structure and signaling process: dendrites receive input > soma integrates and processes signal (decides whether to activate) > axon transmits and outputs the signal > downstream neurons. A mathematical representation of these components looks like: \mathbf{x} = input matrix; \mathbf{w} = weight matrix, $y = \mathbf{o}$ (omega) = sum of the weighted inputs
- How is an output generated?
 - (1) Multiply each binary input by its weight
 - (2) Sum all weighted inputs
 - (3) Compare sum with the **threshold**
 - (4) Output binary (1 or 0) representing whether the neuron is **fired**

- Example: Deciding whether to attend a cheese festival



Note: Since we hate bad weather, we place higher weight on the “rainy” binary input. In this case, if the weather is good (i.e. $x_1 = 1$), then the sum of weighted x ’s ($x_1 \cdot w_1 + x_2 \cdot w_2 + x_3 \cdot w_3$) will be greater than the threshold value even if we don’t have a friend to go with ($x_2 = 0$) and the festival is far away ($x_3 = 0$).

- Math: $x_1 \cdot w_1 + x_2 \cdot w_2 + x_3 \cdot w_3 = 6 + 0 + 0 = 6$. $6 > \text{threshold of } 5 \rightarrow$ “neuron fires”, i.e. we go to the cheese festival

If the weather is bad (i.e. $x_1=0$) even if we have a friend to go with ($x_2=1$) and the festival is close ($x_3=1$), the weighted sum will not be $>$ threshold of 5, so we will not go to the festival.

Example #2: Threshold=3. In this case, even if the weather is not good ($x_1=0$), we still have a chance of exceeding the threshold and going to the festival if $x_2=1$ and $x_3=1$.

1.2 Perceptron (Rosenblatt, 1958)

The perceptron was a new version of the artificial neuron that replaces threshold with a perceptron **bias $b = -\text{threshold}$** .

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

The perception thus has a new training method, which is very similar to that of a modern NN.

- (1) **Initialization** - start with random weights and bias
- (2) **Prediction** - compute the perceptron output for each input

- (3) **Update rule** - based on loss function and learning rate

3. Update Rule:

If the prediction is incorrect ($y \neq \hat{y}$), update the weights and bias:

$$\omega_i \leftarrow \omega_i + \eta(y - \hat{y})x_i$$

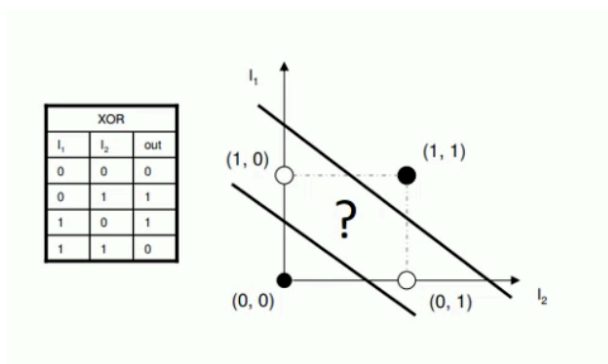
$$b \leftarrow b + \eta(y - \hat{y})$$

- Where y - true label; \hat{y} - predicted label; η - learning rate

- (4) **Repeat** - iterate over the dataset until the perceptron converges or maximum number of epochs reached.

1.3 The XOR Problem

The main issue with single-layer perceptions is that they are only capable of learning **linearly separable patterns** since they compute a **linear decision boundary**. For example, **XOR** (“exclusive or”) is a logical operator that compares 2 inputs and returns true if only one of them is true. However, XOR’s True/False outputs can’t be separated by a single, linear decision boundary, as shown in the figure below.



The solution? Add **nonlinearity** and **more layers**.

2. Multi-Layer Perceptron (MLP):

2.1 Modern MLP Structure

- Input layer takes raw data as input and has the same number of neurons as the number of input features. The second component are the Hidden layers (potentially many). Hidden layers extract features using weighted connections and apply activation functions (e.g. sigmoid, ReLU) to the weighted inputs, which adds nonlinearity to the output. Finally, the output layer produces predictions (e.g. 1 for regression, n for classification). The number of neurons is defined as the number of output classes.

KEY Q: How do we train a deep NN?

2.2 Backward propagation

- The purpose of backward propagation is to minimize prediction error by adjusting weights. To do so, we efficiently compute gradients using the **chain rule**.
- The Process:
 - (1) **Forward propagation** - compute predictions starting from input
 - (2) **Error/loss calculation** - compare predictions to the true labels (e.g. error = difference between predictions and actual values)
 - Ex: mean squared error = $\frac{1}{2}(\text{prediction} - \text{actual})^2$
 - (3) **Backward propagation** - calculate **gradients** of the loss function wrt weights
 - **Gradient descent**: for each **step**, calculate the gradient at that point
 - As go further back, need to take partial derivatives and use the chain rule
 - Step size determined by **learning rate**

$$\omega_i = \omega_i - \eta \left(\frac{\partial \text{Error}}{\partial \omega_i} \right)$$

- (4) **Weight update** - adjust weights using gradient descent

There are millions of weights (i.e. parameters) that are trained in modern NNs.

- Note: No nonlinearity or activation applied yet
- See reference “Neural Networks Pt 2: Backpropagation Main Ideas” for more details.

2.3 Activation function

- **Activation function** - analogous to the rate of action potential firing in a neuron. Applies a nonlinearity to the weighted inputs, allowing the model to use nonlinear decision boundaries for regression / classification. The simplest activation function is **binary** (i.e. step function). Modern activation functions introduce a **region of uncertainty** and are **differentiable** (important for using the chain rule in backpropagation). For example, the **Sigmoid** function is great because it's derivative is easy to calculate & thus makes the chain rule easy to compute:

Example: Backpropagation with (Sigmoid) Nonlinear Activation

$$z = \mathbf{W}^T \mathbf{x} + b$$

$$a = \sigma(z)$$

If $L = \frac{1}{2}(a - y)^2$:

1. Compute:

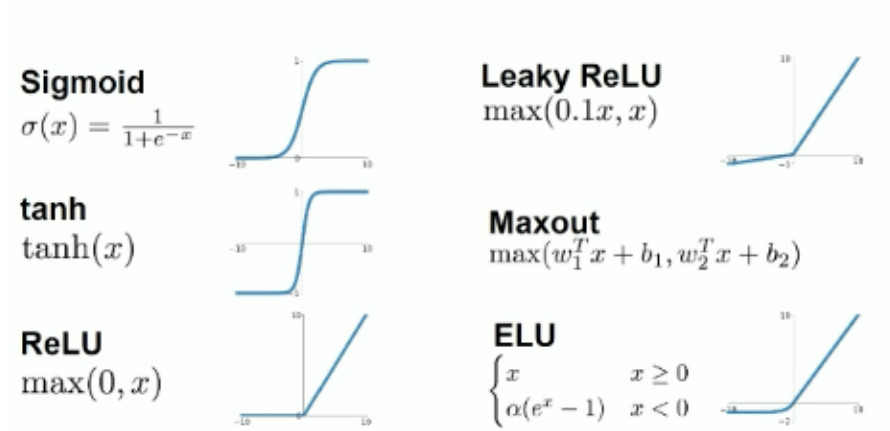
$$\frac{\partial L}{\partial a} = a - y$$
2. Chain rule:

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial a} \cdot \sigma'(z) = (a - y) \cdot \sigma(z)(1 - \sigma(z))$$
3. Gradients:

$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial z} \cdot \mathbf{x}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial z}$$

Common activation functions enable efficient optimization of gradient-based methods:



2.4 Key Concepts

- **Learning rate:** When the learning rate (which defines step size in gradient descent algorithm) is **too low**, gradient descent requires too many updates / time before reaching the minimum (of the loss function). On the other hand, if the learning rate is **too high**, we risk the model jumping around the minimum and possibly never converging. Ideally, the learning rate allows the model to swiftly reach the minimum point.
- **Dropout** is a regularization technique designed to reduce **overfitting** in neural networks. How it works is that for each training iteration (“mini-batch”), we randomly “drop” (set to zero) a fraction of the neurons. This forces the network to NOT rely on a few or a single neuron, but rather, to actually learn underlying features of the inputs.
- **Convolutional neural networks (CNN)** use a convolutional layer to automatically learn and extract features from input data. CNNs are commonly used for image data, which are represented by a large grid of pixels, with each pixel grayscale value representing an input with range (0,255). CNNs focus on the **receptive field**, a small region of the input. The convolutional layer gradually “slides” through each receptive field -> represents in its own pixel / input neuron -> generate features

2.5 KEY Q: What is deep learning?

- TL;DR: A rebranding of NNs
- “Deep” indicates a multi-layer structure (i.e. having 2+ hidden layers), and modern networks commonly have > 10 hidden layers. Deep learning models use learning algorithms to automatically learn weights and biases, and are trained via Stochastic gradient descent (SGD) and backpropagation. Deep learning models have been shown to perform better for many problems than single-hidden layer (“shallow”) NNs.

2.6 Caveats

There are important limitations or disadvantages to deep NNs. First, they require **lots of training data**. Second, deep networks contain **lot of parameters**, making design and optimization difficult. Because the NN takes **blackbox** approach to training, hence the “hidden” aspect of “hidden layers”, it can be **hard to interpret** the output and/or how the NN arrived at that output. Finally, there are important considerations needed for choosing the number of hidden layers and the number of neurons in each hidden layer. The general consensus is that **one hidden layer is sufficient** for most problems, as the performance improvement from adding > 2 hidden layers is very small. As for the number of neurons, there is general consensus on the optimal size of hidden layer being typically between the size of input and size of output layers. Indeed, “Intro to Neural Networks in Java” author Jeff Heaton suggests using the mean of the number of neurons in input and output layers as the number of neurons in the hidden layer(s).

Discussion/Comments

- Artificial Neural Networks (ANNs) are machine learning models inspired by the structure and function of biological neural networks.
- Their ability to learn complex patterns from large datasets without explicit programming allows them to solve problems that traditional algorithms struggle with
- The main initial innovation is their capacity to detect nonlinear relationships in high-dimensional and complex data.
- They haven used to analyze high-dimensional biological data, predict protein structures, classify tumors from histopathological images, and identify potential drug targets by modeling molecular interactions.

List all suggested reading here and please answer:

Are the readings for the class useful? If so, are the specific subsections useful or would change. If not, are there other references you could suggest? Please suggest one.


- Free online books
 - The Nature of Code by Daniel Shiffman - 10. Neural Networks (easy to read): <http://natureofcode.com/book/chapter-10-neural-networks/>
 - Neural Networks and Deep Learning by Michael Nielsen: <http://neuralnetworksanddeeplearning.com/index.html>
- Review articles
 - Yann LeCun, Yoshua Bengio, Geoffrey Hinton. Deep learning. Nature 521, 436–444 (28 May 2015)


References ISL/ESL (if any)

- An Introduction to Statistical Learning, Chapter 10, Deep learning: <https://www.statlearning.com/>

The ISL reference on deep learning is especially useful, as it breaks down the architecture of NNs starting with single-layer, then multi-layer networks, then CNNs. The text also gives students a more in-depth mathematical background on how output is generated in MLPs and goes beyond the lecture content starting from approx. section 10.4.

Other Suggest references for many of the key concepts

 [The Essential Main Ideas of Neural Networks](#)

 [Neural Networks Pt. 2: Backpropagation Main Ideas](#)