CSCE566-DATA MINING
WEEK 5

Feedforward Neural Networks

Min Shi
min.shi@louisiana.edu

Sep 23, 2024

Outline

- Introduction & Recap
- Perceptron (The Simplest Feedforward Neural Network)
- Multilayer Perceptron (MLP)
- Overfitting Problem
- Regularization Technique

Classical Data Mining (DM) and Machine Learning (ML) Algorithms

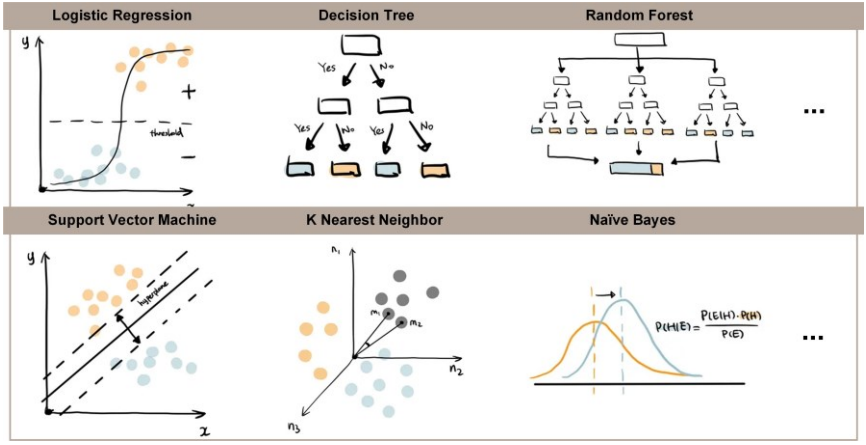


Figure: Machine learning for classification and regression problems.

Five Core Steps for ML

Step 1: **Define Problem**

- Medical image analysis: classification, segmentation and detection, etc.
- Electronic health record analysis: outcome prediction, treatment recommendation.

Step 2: **Prepare and Process Data**

- Data collection, addressing missing features, formatting, etc.

Step 3: **Select and Build ML Model**

- Supervised or unsupervised - classification or regression.

Step 4: **Train and Evaluate ML Model**

- Training/testing data, optimization objective, evaluation metrics, etc.

Step 5: **Improve Model Performance**

- Reconsider and readjust the pipeline, finetuning model parameters, etc.

ML Paradigms

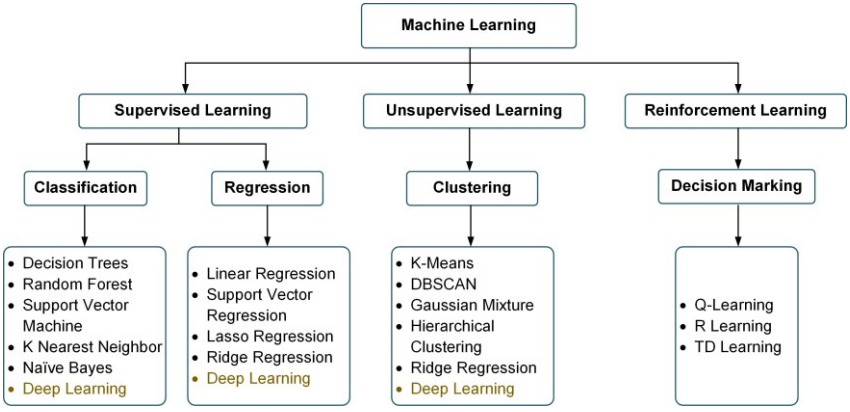


Figure: Different types of machine learning.
https://www.analyticsvidhya.com/blog/2021/03/everything-you-need-to-know-about-machine-learning/

Pros and Cons of Classical ML

Advantages

- Suitable for small data (e.g., linear regression).
- Easy to interpret outcomes (e.g., rule-based decision tree).
- Does not need large computational power (e.g., run on low-end machines)
- Easy to implement.

Disadvantages

- Difficult to learn large datasets (i.e., limited learning capability).
- Require feature engineering (i.e., handcrafted features).
- Difficult to perform complex tasks (e.g., natural language processing, image segmentation)

Pros and Cons of Classical ML

Advantages

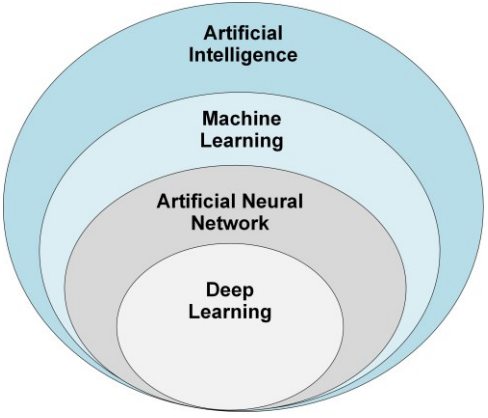
- Suitable for small data (e.g., linear regression).
- Easy to interpret outcomes (e.g., rule-based decision tree).
- Does not need large computational power (e.g., run on low-end machines)
- Easy to implement.

Disadvantages

- Difficult to learn large datasets (i.e., limited learning capability).
- Require feature engineering (i.e., handcrafted features).
- Difficult to perform complex tasks (e.g., natural language processing, image segmentation)

Artificial neural networks and deep learning can resolve the limitations.

Deep Learning vs. Machine Learning



Artificial Intelligence:
A science developed to making machines think and act like human

Machine Learning:
A subset of AI. Focuses on enabling machines to perform tasks without explicit programming.

Artificial Neural Network:
A subset of machine learning. The structure is inspired by the human brain, mimicking the way that biological neurons signal from one to another.

Deep Learning:
A subset of neural networks in which multilayered neural networks learn from vast amount data

Figure: Relations among artificial intelligence, machine learning, neural network and deep learning.

Deep Learning Generated Image

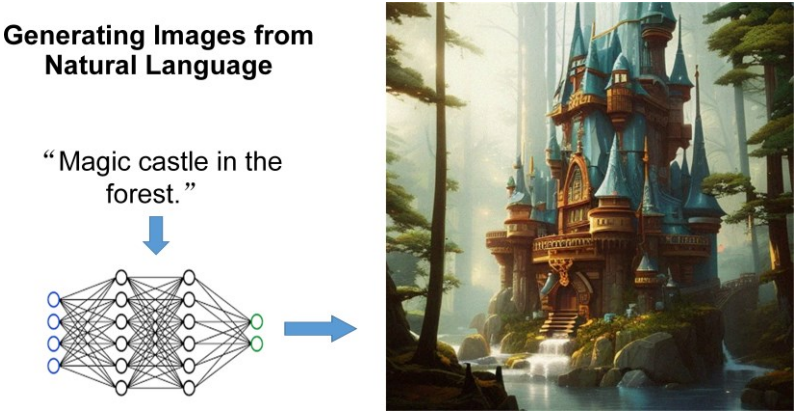
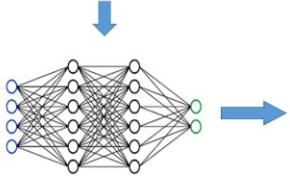


Figure: Free AI art generator at https://neural.love/ai-art-generator.

Deep Learning Generated Code

Generating Code from Natural Language

"Write code in Tensorflow to define a 3-layer neural network."



Here's an example code in TensorFlow to define a 3-layer neural network using the Keras API:

python Copy code

```
import tensorflow as tf
from tensorflow.keras import layers

# Define the neural network architecture
model = tf.keras.Sequential([
    layers.Flatten(input_shape=(28, 28)),
    layers.Dense(256, activation='relu'),
    layers.Dense(128, activation='relu'),
    layers.Dense(10)
])
```

In this code, we define a 3-layer neural network using the Keras API. The first layer is a `Flatten` layer that reshapes the input data into a 1D array. The next two layers are dense layers with 256 and 128 units, respectively, and use the ReLU activation function. The final layer has 10 units, which corresponds to the number of classes in the classification problem. Note that this is just an example architecture and can be modified according to the problem at hand.

Regenerate response

Figure: Try ChatGPT at https://chat.openai.com/.

A Brief History of Neural Networks

1943 Warren McCulloch & Walter Pitts modeled a simple neural network with electrical circuits.

1956 The term of Artificial Intelligence was coined at Dartmouth conference.

1958 Frank Rosenblatt developed the perceptron (single-layer neural network as a linear binary classifier) inspired by the way neurons work in the brain.

1974 Paul Werbos describes the algorithm of training a Neural Net through Backpropagation.

1986 Rummelhart, Hilton and Williams proposed the Multilayer Perceptron - a nonlinear multilayer neural network trained with backpropagation.

1998 Yann LeCun developed a Convolutional Neural Network for processing images.

A Brief History of Neural Networks

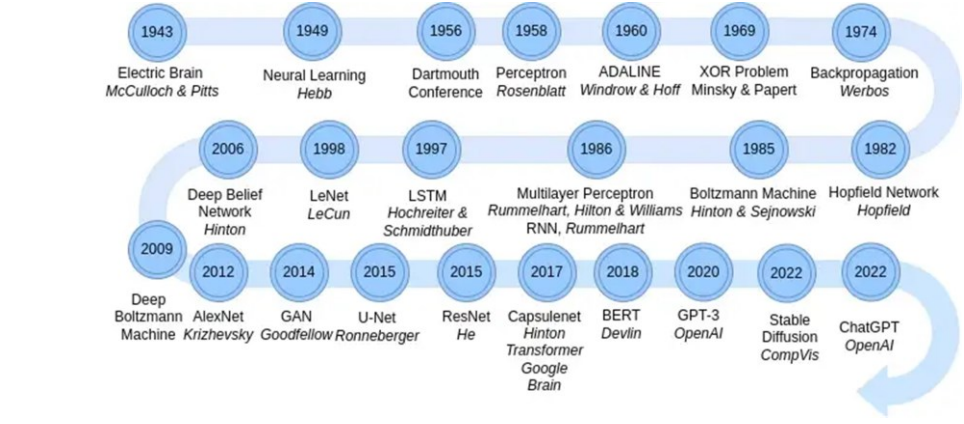


Figure: Important history events of neural networks.
https://pub.towardsai.net/a-brief-history-of-neural-nets-472107bc2c9c

Rises and Falls of Neural Network Popularity

1st winter (1974 – 1980) Limited applicability of AI leads to funding pullback in the U.S. and abroad.

2nd winter (1987 – 1994) Limitations of if-then reasoning. Expert systems, an attempt to replicate human reasoning through a series of if-then rules, failed.

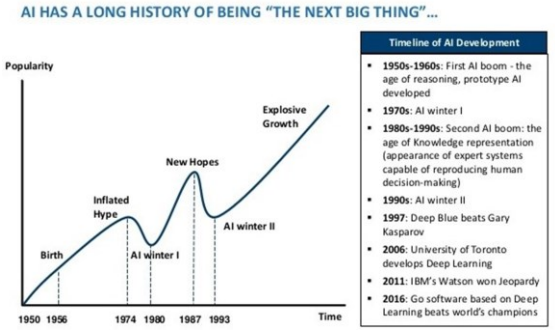


Figure: https://www.actuaries.digital/2018/09/05/history-of-ai-winters.

Why Deep Learning Now?

Figure: Some reasons for the resurgence of deep learning.

The Perceptron - The Structural Building Block of Deep Learning

Inspired by animal's computing machinery: 1) The dendrites receive information; 2) The cell body processes and integrates that information; 3) The axon carries the information along long distances from one part of the neuron to another; and 4) the axon terminal transmits the information to the next cell in the chain.

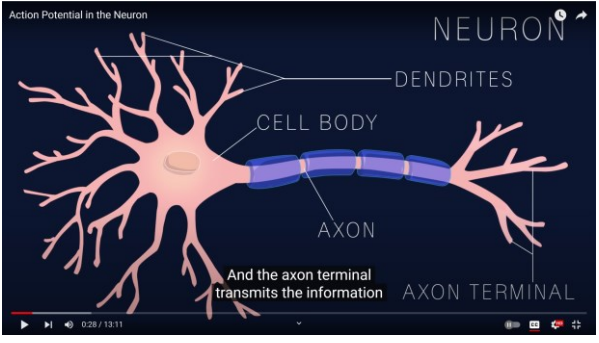


Figure: Full video at https://www.youtube.com/watch?v=oa6rvUJlg7o.

Animal's Computing Machinery

When the input signals exceed a certain threshold within a short period of time, a neuron "fires" (i.e., an electrical impulse) - generating an output signal.

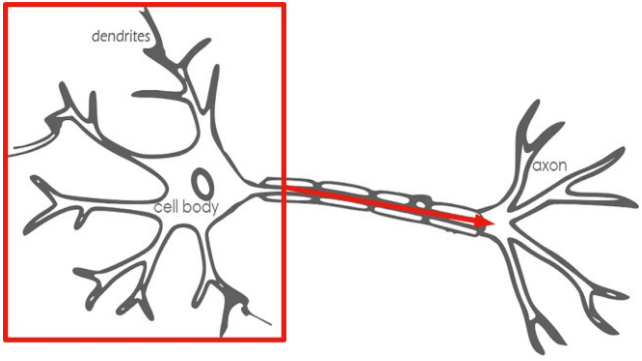
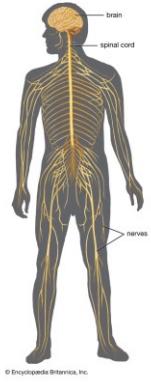


Figure: https://becominghuman.ai/introduction-to-neural-networks-bd042ebf2653.

Animal's Computing Machinery

https://en.wikipedia.org/wiki/Nematode#/media/File:CelegansGoldsteinLabUNC.jpg

Nematode worm: 302 neurons



https://www.britannica.com/science/human-nervous-system

Human: ~100,000,000,000 neurons

The Architecture of Perceptron

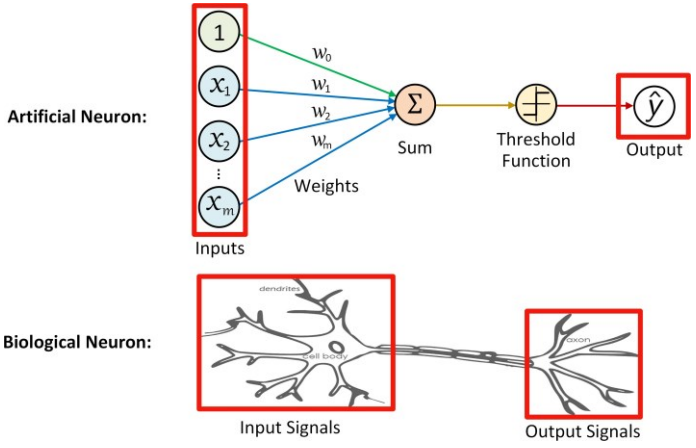


Figure: Artificial neuron vs. biological neuron.

The Architecture of Perceptron

The formulation of perceptron:

Inputs: $\mathbf{x} = [x_1, x_2, \ldots, x_m]$

Weights: $\mathbf{w} = [w_1, w_2, \ldots, w_m]^T$

$$z = w_0 + w_1x_1 + w_2x_2 + \cdots + w_mx_m = w_0 + \mathbf{w}^T\mathbf{x} \quad (1)$$

$$\hat{y} = \varphi(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{otherwise} \end{cases} \quad (2)$$

where w_0 is a bias (e.g., intercept). The threshold function φ is also called the non-linear Activation Function.

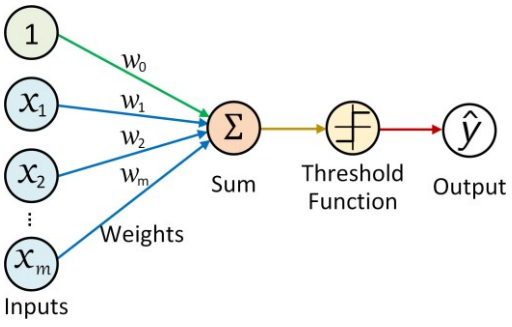


Figure: A single-layer single-output perceptron.

The Architecture of Perceptron

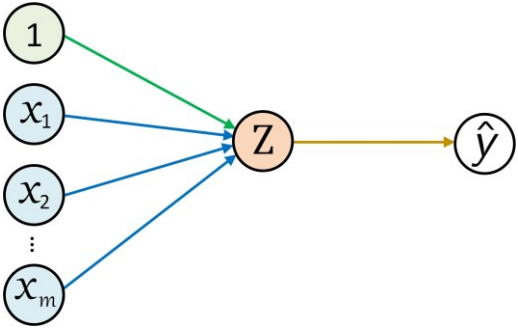


Figure: A single-layer perceptron with one output.

$$\hat{y} = \varphi(z) = \varphi(w_0 + \mathbf{w}^T\mathbf{x})$$

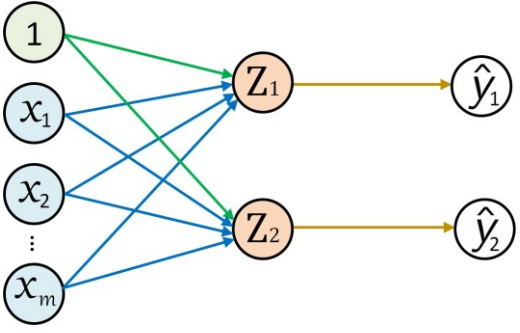


Figure: A single-layer perceptron with two outputs.

$$\hat{y_i} = \varphi(z_i) = \varphi(w_{0,i} + \mathbf{w}_i^T\mathbf{x}),\ i \in [1, 2]$$

Implement a Perceptron in Tensorflow from Scratch

Sure, here's an example of how to implement a Perceptron from scratch in TensorFlow:

python Copy code

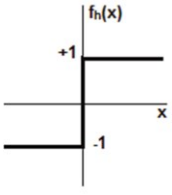
```python
import tensorflow as tf

class Perceptron(tf.Module):
    def __init__(self, num_inputs, num_outputs):
        super().__init__()
        self.W = tf.Variable(tf.random.normal(shape=(num_inputs, num_outputs)))
        self.b = tf.Variable(tf.zeros(shape=(num_outputs,)))

    def __call__(self, x):
        z = tf.matmul(x, self.W) + self.b
        return tf.nn.sigmoid(z)

model = Perceptron(num_inputs=2, num_outputs=1)
```

Figure: Try ChatGPT at https://chat.openai.com/chat/.

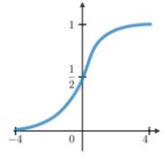
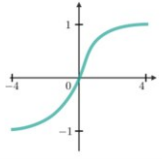
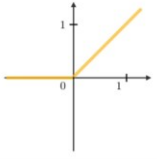
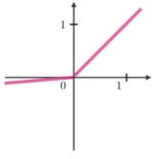
Activation Functions φ

The activation function decides whether a neuron should be activated or not by calculating the weighted sum and further adding bias to it.

There are different activation functions (Binary Step, Linear, Sigmoid, Tanh, ReLU, Leaky ReLU, Parameterised ReLU, Exponential Linear Unit, Swish, Softmax) used in different applications. The common ones are:



hard limiter

Sigmoid	Tanh	ReLU	Leaky ReLU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$
			

Which Activation Function Should be Used?

Activation function for the output layer:

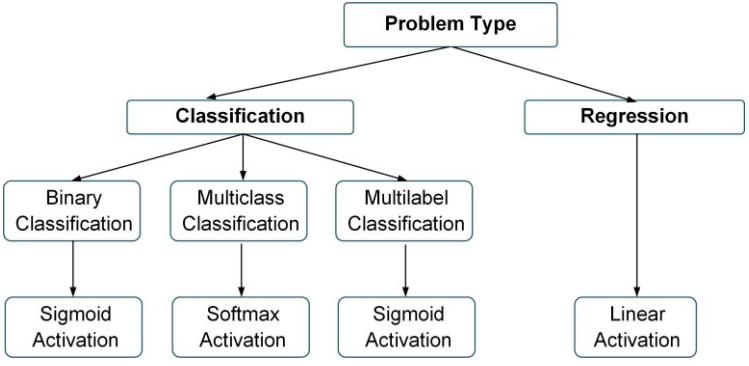


Figure: https://www.kaggle.com/general/212325.

Perceptron for Classification

- A perceptron is used for
  - binary classification.
- Given training samples of two classes: C_1 and C_2
  - Train the perceptron in such a way that it correctly classifies any input sample:
    - *if the output of the perceptron is +1*
      - then the input is assigned to class C_1
    - *if the output of the perceptron is -1 (sometimes we use 0)*
      - then the input is assigned to class C_2

Training a Perceptron for Classification

- Find suitable values for the weights, so that the training examples are correctly classified.
- Geometrically, we try to find a hyper-plane that separates the examples of the two classes.

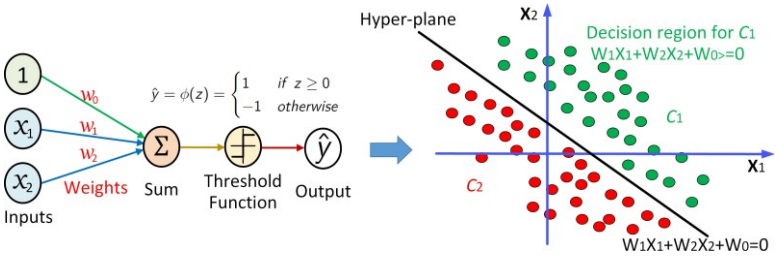


Figure: The perceptron learns a hyper-plane to separate the samples in two classes.

Perceptron Training

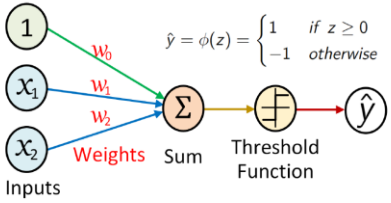
1. Initialize weights to 0 or small random numbers;
2. For each training sample (i.e., i^{th} sample):
   - Compute output value: $\hat{y}_i = \phi(w_0 + \mathbf{w}^T\mathbf{x}) = \phi(w_0 + \sum_{j=1}^{m} w_j x_j)$
   - Update weights with the following equation: $w_j := w_j + \triangle w_j$

$$\triangle w_j = \eta(y_i - \hat{y}_i)x_j$$

   η is the learning rate. y_i is the true class label for i^{th} sample and $\hat{y}_i$ is the predicted class label (i.e., perceptron output).
3. Repeat step 2 until stopping criteria are satisfied (e.g., the number of epochs is reached or the prediction accuracy is good enough).

Perceptron Training (e.g., 2-Dimensional Dataset)

1. Initialize weights to 0 or small random numbers;
2. For each training sample (i.e., i^{th} sample):
   - Compute output value: $\hat{y}_i = \phi(w_0 + \mathbf{w}^T\mathbf{x}) = \phi(w_0 + \sum_{j=1}^{2} w_j x_j)$
   - Update weights with the following definition: $w_j := w_j + \triangle w_j$



$$\triangle w_0 = \eta(y_i - \hat{y}_i)$$
$$\triangle w_1 = \eta(y_i - \hat{y}_i)x_1$$
$$\triangle w_2 = \eta(y_i - \hat{y}_i)x_2$$

All weights updated simultaneously

Perceptron Training

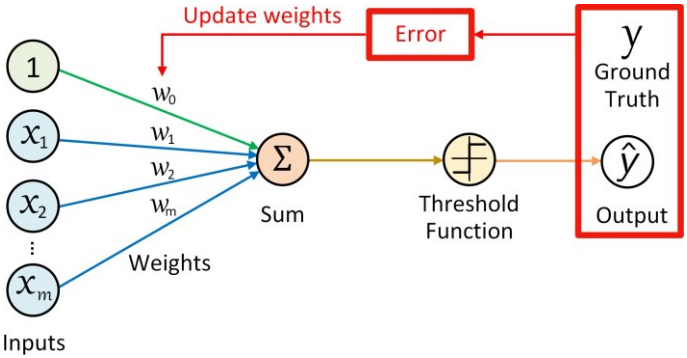


Figure: The overall training principle for perceptron.

Perceptron Training Example

True Model: Y is 1 if at least two of the three values are equal to 1, or -1 otherwise.

ID	X_1	X_2	X_3	Y
1	1	0	0	?
2	1	0	1	
3	1	1	0	
4	1	1	1	
5	0	0	1	
6	0	1	0	
7	0	1	1	
8	0	0	0	

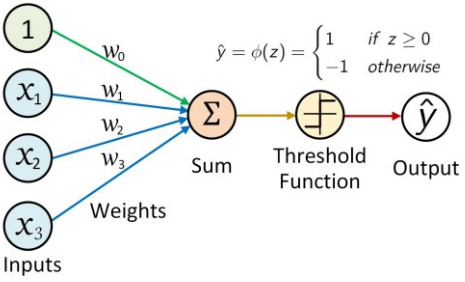


Figure: Training a perceptron model to predict Y from X_1, X_2 and X_3.

Perceptron Training Example

True Model: Y is 1 if at least two of the three values are equal to 1, or -1 otherwise.

Calculate the true class labels:

ID	X_1	X_2	X_3	Y
1	1	0	0	-1
2	1	0	1	
3	1	1	0	
4	1	1	1	
5	0	0	1	?
6	0	1	0	
7	0	1	1	
8	0	0	0	

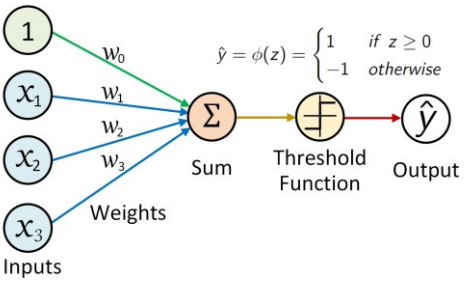


Figure: Training a perceptron model to predict Y from X_1, X_2 and X_3.

Perceptron Training Example

True Model: Y is 1 if at least two of the three values are equal to 1, or -1 otherwise.

Calculate the true class labels:

ID	X_1	X_2	X_3	Y
1	1	0	0	-1
2	1	0	1	1
3	1	1	0	?
4	1	1	1	
5	0	0	1	
6	0	1	0	
7	0	1	1	
8	0	0	0	

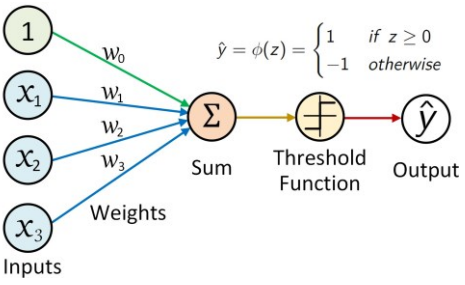


Figure: Training a perceptron model to predict Y from X_1, X_2 and X_3.

Perceptron Training Example

True Model: Y is 1 if at least two of the three values are equal to 1, or -1 otherwise.

Calculate the true class labels:

ID	X_1	X_2	X_3	Y
1	1	0	0	-1
2	1	0	1	1
3	1	1	0	1
4	1	1	1	1
5	0	0	1	-1
6	0	1	0	-1
7	0	1	1	1
8	0	0	0	-1

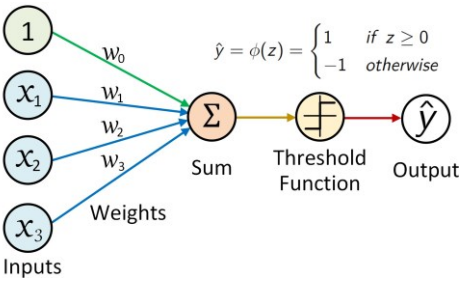


Figure: Training a perceptron model to predict Y from X_1, X_2 and X_3.

Perceptron Training Example

Training the first sample

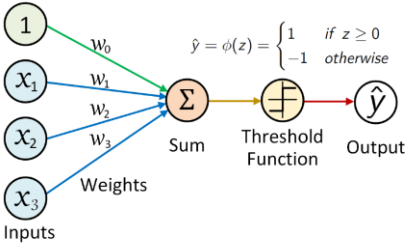
- Compute output value: $z = w_0 + \sum_{j=1}^{3} w_j x_j$; $\hat{y} = 1$ if $z \geq 0$, $\hat{y} = -1$ otherwise.

Weights are initialized as zeros:

ID	X_1	X_2	X_3	Y
1	1	0	0	-1

Output
?

W_0	W_1	W_2	W_3
0	0	0	0



Perceptron Training Example

Training the first sample

- Compute output value: $z = w_0 + \sum_{j=1}^{3} w_j x_j$; $\hat{y} = 1$ if $z \geq 0$, $\hat{y} = -1$ otherwise.

$z = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 = 0$

$$\hat{y} = \phi(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

Threshold Function

ID	X_1	X_2	X_3	Y
1	1	0	0	-1

Output
1

W_0	W_1	W_2	W_3
0	0	0	0

Perceptron Training Example

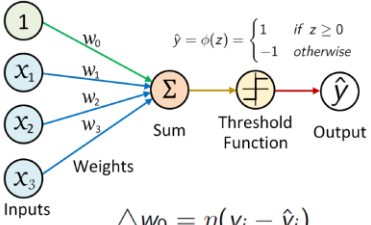
Training the first sample

- Update weights: $w_j = w_j + \triangle w_j$, $\triangle w_j = \eta(y_i - \hat{y}_i)x_j$, learning rate $= 0.1$.

ID	X_1	X_2	X_3	Y
1	1	0	0	-1

Output
1

W_0	W_1	W_2	W_3
0	0	0	0
?	?	?	?



$$\triangle w_0 = \eta(y_i - \hat{y}_i)$$
$$\triangle w_1 = \eta(y_i - \hat{y}_i)x_1$$
$$\triangle w_2 = \eta(y_i - \hat{y}_i)x_2$$
$$\triangle w_3 = \eta(y_i - \hat{y}_i)x_3$$

Perceptron Training Example

Training the first sample

item Update weights: $w_j = w_j + \triangle w_j$, $\triangle w_j = \eta(y_i - \hat{y}_i)x_j$, learning rate $= 0.1$.

ID	X_1	X_2	X_3	Y
1	1	0	0	-1

Output
1

W_0	W_1	W_2	W_3
0	0	0	0
?	?	?	?

$$\triangle w_0 = 0.1(-1-1) = -0.2$$
$$\triangle w_1 = 0.1(-1-1) * 1 = -0.2$$
$$\triangle w_2 = 0.1(-1-1) * 0 = 0$$
$$\triangle w_3 = 0.1(-1-1) * 0 = 0$$

Perceptron Training Example

Training the second sample

- Compute output value: $z = w_0 + \sum_{j=1}^{3} w_j x_j$; $\hat{y} = 1$ if $z \geq 0$, $\hat{y} = -1$ otherwise.

Weights are up-to-date:

ID	X_1	X_2	X_3	Y
1	1	0	0	-1
2	1	0	1	1

Output
1
?

W_0	W_1	W_2	W_3
0	0	0	0
-0.2	-0.2	0	0

Perceptron Training Example

Training the second sample

- Compute output value: $z = w_0 + \sum_{j=1}^{3} w_j x_j$; $\hat{y} = 1$ if $z \geq 0$, $\hat{y} = -1$ otherwise.

$z = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 = -0.4$

ID	X_1	X_2	X_3	Y
1	1	0	0	-1
2	1	0	1	1

Output
1
-1

W_0	W_1	W_2	W_3
0	0	0	0
-0.2	-0.2	0	0

Perceptron Training Example

Training the second sample

- Update weights: $w_j = w_j + \triangle w_j$, $\triangle w_j = \eta(y_i - \hat{y}_i)x_j$, learning rate $= 0.1$.

ID	X_1	X_2	X_3	Y
1	1	0	0	-1
2	1	0	1	1

Output
1
-1

W_0	W_1	W_2	W_3
0	0	0	0
-0.2	-0.2	0	0
?	?	?	?

$$\triangle w_0 = \eta(y_i - \hat{y}_i)$$
$$\triangle w_1 = \eta(y_i - \hat{y}_i)x_1$$
$$\triangle w_2 = \eta(y_i - \hat{y}_i)x_2$$
$$\triangle w_3 = \eta(y_i - \hat{y}_i)x_3$$

Perceptron Training Example

Training the second sample

- Update weights: $w_j = w_j + \triangle w_j$, $\triangle w_j = \eta(y_i - \hat{y}_i)x_j$, learning rate $= 0.1$.

ID	X_1	X_2	X_3	Y
1	1	0	0	-1
2	1	0	1	1

Output
1
-1

W_0	W_1	W_2	W_3
0	0	0	0
-0.2	-0.2	0	0
?	?	?	?

$$\triangle w_0 = 0.1(1 - -1) = 0.2$$
$$\triangle w_1 = 0.1(1 - -1) * 1 = 0.2$$
$$\triangle w_2 = 0.1(1 - -1) * 0 = 0$$
$$\triangle w_3 = 0.1(1 - -1) * 1 = 0.2$$

Perceptron Training Example

Training the second sample

- Update weights: $w_j = w_j + \triangle w_j$, $\triangle w_j = \eta(y_i - \hat{y}_i)x_j$, learning rate $= 0.1$.

ID	X_1	X_2	X_3	Y
1	1	0	0	-1
2	1	0	1	1

Output
1
-1

W_0	W_1	W_2	W_3
0	0	0	0
-0.2	-0.2	0	0
0	0	0	-0.2

$$\triangle w_0 = 0.1(1 - -1) = 0.2$$
$$\triangle w_1 = 0.1(1 - -1) * 1 = 0.2$$
$$\triangle w_2 = 0.1(1 - -1) * 0 = 0$$
$$\triangle w_3 = 0.1(1 - -1) * 1 = 0.2$$

Perceptron Training Example

Training one epoch for all samples

- Update weights: $w_j = w_j + \triangle w_j$, $\triangle w_j = \eta(y_i - \hat{y}_i)x_j$, learning rate $= 0.1$.

ID	X_1	X_2	X_3	Y
1	1	0	0	-1
2	1	0	1	1
3	1	1	0	1
4	1	1	1	1
5	0	0	1	-1
6	0	1	0	-1
7	0	1	1	1
8	0	0	0	-1

Output
1
-1
1
1
1
-1
-1
1

	W_0	W_1	W_2	W_3
0	0	0	0	0
1	-0.2	-0.2	0	0
2	0	0	0	0.2
3	0	0	0	0.2
4	0	0	0	0.2
5	-0.2	0	0	0
6	-0.2	0	0	0
7	0	0	0.2	0.2
8	-0.2	0	0.2	0.2

Perceptron Training Example

Training one epoch for all samples

- Update weights: $w_j = w_j + \triangle w_j$, $\triangle w_j = \eta(y_i - \hat{y}_i)x_j$, learning rate $= 0.1$.
- Each sample has been trained once.

ID	X_1	X_2	X_3	Y
1	1	0	0	-1
2	1	0	1	1
3	1	1	0	1
4	1	1	1	1
5	0	0	1	-1
6	0	1	0	-1
7	0	1	1	1
8	0	0	0	-1

Output
1
-1
1
1
1
-1
-1
1

Epoch	W_0	W_1	W_2	W_3
1	-0.2	0	0.2	0.2

Perceptron Training Example

Training second epoch for all samples

- Update weights: $w_j = w_j + \triangle w_j$, $\triangle w_j = \eta(y_i - \hat{y}_i)x_j$, learning rate $= 0.1$.
- Each sample has been trained twice.

ID	X_1	X_2	X_3	Y
1	1	0	0	-1
2	1	0	1	1
3	1	1	0	1
4	1	1	1	1
5	0	0	1	-1
6	0	1	0	-1
7	0	1	1	1
8	0	0	0	-1

Output
-1
1
1
1
1
-1
-1
-1

Epoch	W_0	W_1	W_2	W_3
1	-0.2	0	0.2	0.2
2	-0.2	0	0.4	0.2

Perceptron Training Example

Training six epochs for all samples

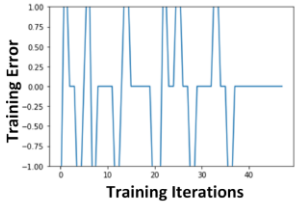
- Update weights: $w_j = w_j + \triangle w_j$, $\triangle w_j = \eta(y_i - \hat{y}_i)x_j$, learning rate $= 0.1$.
- Each of the 8 samples has been trained 6 times - 48 training iterations in total.

ID	X_1	X_2	X_3	Y
1	1	0	0	-1
2	1	0	1	1
3	1	1	0	1
4	1	1	1	1
5	0	0	1	-1
6	0	1	0	-1
7	0	1	1	1
8	0	0	0	-1

Output
-1
1
1
1
-1
-1
1
-1

Epoch	W_0	W_1	W_2	W_3
1	-0.2	0	0.2	0.2
2	-0.2	0	0.4	0.2
3	-0.4	0	0.4	0.2
4	-0.4	0.2	0.4	0.2
5	-0.4	0.4	0.4	0.2
6	-0.4	0.4	0.4	0.2

Final Model

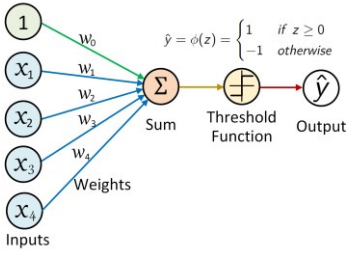


Implementation code: https://colab.research.google.com/drive/12vNfZ9VRellcTFpwKS42ndyIOyIqpSmk?usp=sharing

Homework

True Model: Y is 1 if at least **three** of the four values are equal to 1.

ID	X_1	X_2	X_3	X_4	Y	Epoch 2 Output
1	1	1	0	0	?	?
2	0	1	1	1		
3	1	1	0	0		
4	1	1	1	0		
5	0	0	1	1		
6	0	1	0	1		
7	0	1	1	1		
8	0	0	0	1		



Learning rate=0.1

Epoch	W_0	W_1	W_2	W_3	W_4
0	0	0	0	0	0
1	?	?	?	?	?
2	?	?	?	?	?

Task: Calculate the updated weights and outputs in the first two epochs by hand. Then, modify the **Code** to implement the perceptron.

Interactive Perceptron Training

Interactive visualization of neural networks:
https://pages.nist.gov/nn-calculator

Perceptron Training Hyper-Parameters

We normally need to try back and forth to identify good hyper-parameters:

- Learning rate
- Number of epochs

Common ways to find suitable hyper-parameters:

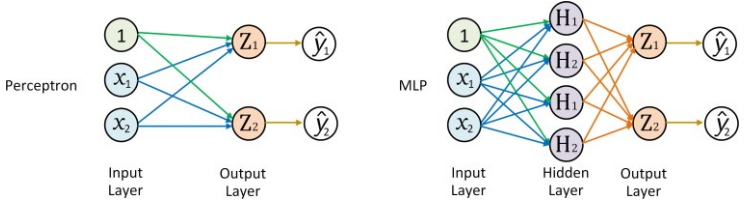
- Grid search
- Random search
- Use a validation dataset

Deep Learning Tuning Playbook at https://github.com/google-research/tuning_playbook

Tuning with PyTorch at https://docs.ray.io/en/releases-1.11.0/tune/tutorials/tune-pytorch-cifar.html

Multilayer Perceptron (MLP)

- The multilayer neural networks contain at least three layers.
- Between the input and output layers there are hidden layers, as illustrated below.
  - Hidden nodes do not directly send outputs to the external environment.
- MLP overcomes the limitation of a single-layer perceptron:
  - Handle non-linearly separable learning tasks.
  - Handle arbitrarily shaped decision boundaries.



Different Non-Linearly Separable Problems

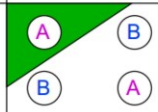
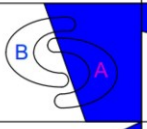
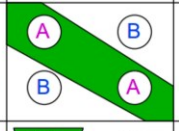
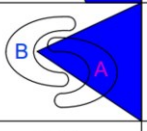
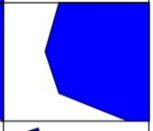
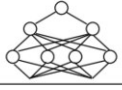
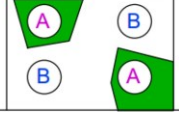
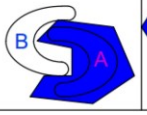
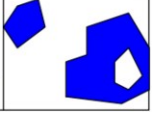
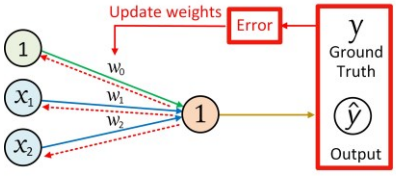
Structure	Types of Decision Regions	Exclusive-OR Problem	Class Separation	Most General Region Shapes
Single-Layer 	Half Plane Bounded By Hyperplane			
Two-Layer 	Convex Open Or Closed Regions			
Three-Layer 	Arbitrary (Complexity Limited by No. of Nodes)			

Figure: https://www.slideserve.com/frisco/various-neural-networks.

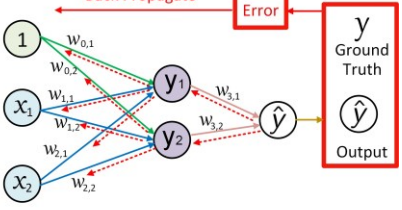
Training MLP with the Back-Propagation Algorithm

$$w_j = w_j + \eta(y_i - \hat{y}_i)x_{j,}$$



Update weights for perceptron

$$w_j \leftarrow w_j - \eta\frac{\partial J}{\partial w_j} \qquad J = (\hat{y} - y)^2$$



Update weights for MLP

$$\frac{\partial J}{\partial w_{3,1}}, \frac{\partial J}{\partial w_{3,2}}, \boxed{\frac{\partial J}{\partial w_{0,1}}}, \frac{\partial J}{\partial w_{0,2}}, \frac{\partial J}{\partial w_{1,1}}, \frac{\partial J}{\partial w_{1,2}}, \frac{\partial J}{\partial w_{2,1}}, \frac{\partial J}{\partial w_{2,2}}$$

$$\frac{\partial J}{\partial w_{0,1}} = \frac{\partial J}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_{3,1}} * \frac{\partial w_{3,1}}{\partial y_1} * \frac{\partial y_1}{\partial w_{0,1}}$$ Chain rule

Back-Propagation Algorithm

- Search for weight values that minimize the total error of the network over the set of training samples.
- Repeat procedures of the following two passes:

  - **Forward pass:** Compute the outputs of all neurons in the network, and the error (i.e., loss) of the output layers.
  - **Backward pass:** The network error is used for updating the weights.
    - Starting at the output layer
      - The error is propagated backward through the network, layer by layer.
      - Recursively computing the local gradient of each neuron.

Backpropagation demo: https://remykarem.github.io/backpropagation-demo/

The Essence of MLP Training - Gradient Descent

The Optimization Algorithm:

1. Initialize weights randomly $W \sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J}{\partial W} = \frac{1}{m} \sum_j^m \frac{\partial J_j}{\partial W_j}$
4. Update weights, $W \leftarrow W - \eta \frac{\partial J}{\partial W}$
5. Return weights

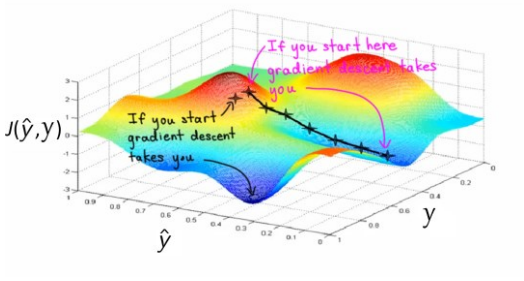


Figure: https://regenerativetoday.com/machine-learning-gradient-descent-concept/.

Gradient Descent

The Optimization Algorithm:

1. Initialize weights randomly $W \sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J}{\partial W} = \frac{1}{m} \sum_j^m \frac{\partial J_j}{\partial W_j}$
4. Update weights, $W \leftarrow W - \eta \frac{\partial J}{\partial W}$
5. Return weights

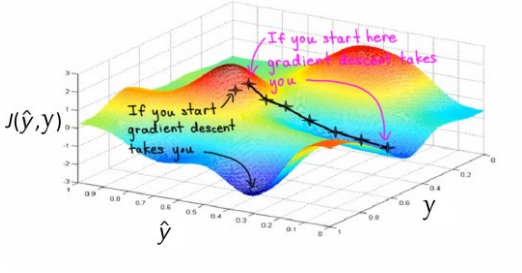


Figure: https://regenerativetoday.com/machine-learning-gradient-descent-concept/.

However, computing the gradients for all samples once can be very computationally intensive!

Stochastic Gradient Descent

The Optimization Algorithm:

1. Initialize weights randomly $W \sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Feed a single sample i each time
4. Compute gradient, $\frac{\partial J_i}{\partial W}$
5. Update weights, $W \leftarrow W - \eta \frac{\partial J}{\partial W}$
6. Return weights

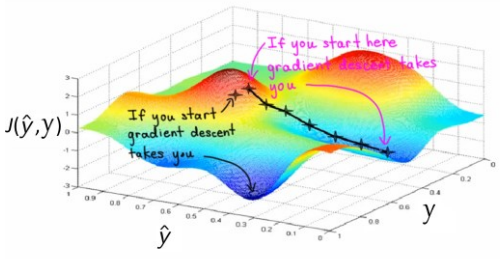


Figure: https://regenerativetoday.com/machine-learning-gradient-descent-concept/.

Although it is easy to compute, the updating of weights can be very stochastic (i.e., unstable)!

Stochastic Gradient Descent - Training in Mini-batches

The Optimization Algorithm:

1. Initialize weights randomly $W \sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Feed n ($n << m$) samples each time.
4. Compute gradient, $\frac{\partial J}{\partial W} = \frac{1}{n} \sum_j^n \frac{\partial J_j}{\partial W_j}$
5. Update weights, $W \leftarrow W - \eta \frac{\partial J}{\partial W}$
6. Return weights

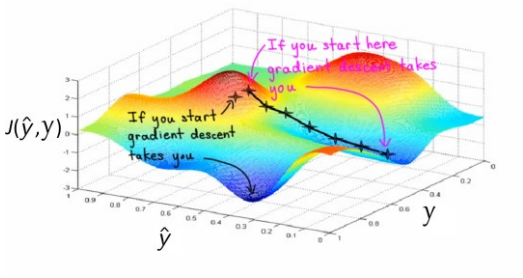


Figure: https://regenerativetoday.com/machine-learning-gradient-descent-concept/.

This is a compromised solution with easy computation and stable optimization!

Underfitting and Overfitting Problems

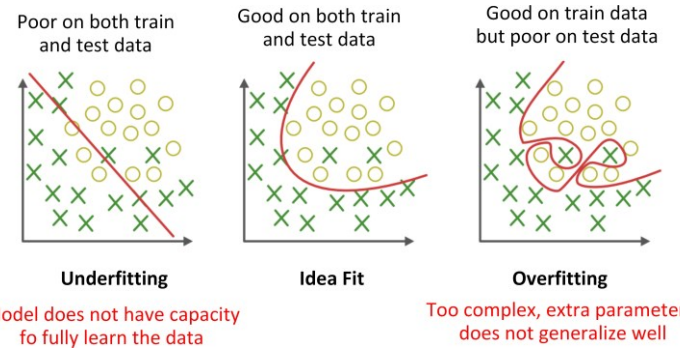


Figure: https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/.

Interactive tool: https://pages.nist.gov/nn-calculator

Ways to Mitigate Underfitting and Overfitting

Ways to mitigate underfitting:

1. Increase model complexity (e.g., more neurons and hidden layers).
2. Reduce noise in data.
3. Increase the duration of training

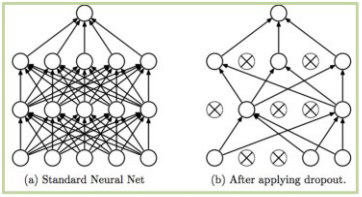
Ways to mitigate overfitting:

1. Train the model with sufficient data.
2. Use K-fold cross-validation.
3. Adopt ensembling techniques (Train multiple models on different random subsets of the training set).
4. Adopt regularization techniques.

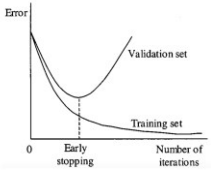
Regularization Techniques

Regularization aims to add constrains to the optimization to discourage complex models. The goal is to improve generalization of the trained model on unseen data.

- **Dropout:** Randomly ignore neurons during the training phase. These neurons are not considered during a particular forward or backward pass
- **Early Stopping:** Stop training when weight parameter updates no longer begin to yield improves (after a set number of iterations) on a validation set.



Dropout neurons to prevent overfitting the training data



Early stop the training

Figure: https://miro.medium.com/v2/resize:fit:1400/format:webp/1*iWQzxhVlvadk6VAJjsgXgg.png; https://paperswithcode.com/method/early-stopping.

Quesions

You can contact Dr. Min Shi via email for further questions:
min.shi@louisiana.edu