System Support for ML

Instructor: Yang Wang

Self Introduction

- Yang Wang
- Associate Professor in CSE
- Email: wang.7564@osu.edu
- Office: DL 689
- Research areas: Distributed and database systems
- https://yangwang83.github.io/

How about you?

- Name?
- PhD or master?
- Research area and advisor, if any?
- What do you expect from this course?

Course info

- Schedule: ……

- Paper reading (30 points)
- Paper presentation (30 points)
- Project (40 points)

- No exams

Paper reading (30 points)

- You are expected to read each paper before class
- You are expected to submit critiques for 30 papers

- Critique: One page; like a paper review
  - One paragraph to summarize the paper
  - Strengths of the paper
  - Weaknesses of the paper
  - Any potential future work

Presentation (30 points)

- You are expected to present 2-3 papers
  - Two are required
  - If you choose three, you get an extra 10 points
- 30 mins presentation + 10 mins Q/A
- If a paper heavily relies on a related work that we have not discussed, then the presenter is responsible for studying and presenting the related work

Project (40 points)

- A group of 1-3 students: Project effort should scale with #students
- Topic: You can choose any topic related to ML. If you don’t have an idea, I can provide some options
- Timeline:
  - Submit a proposal by the end of Jan. Describe your goal, your plan, and your expected output. Discuss with me if you are not sure.
  - Submit the report by the end of the semester.
  - Make a presentation at the end of the semester

Project (40 points)

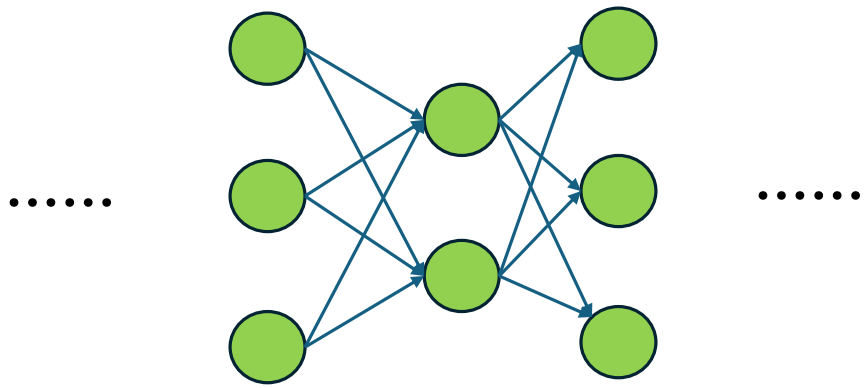
- You are expected to work on a research project.

- Due to the nature of research, your project may not achieve your expected output, which is fine
  - In this case, please analyze and report the reasons why it does not achieve your expectation
  - “My project does not work and I don’t know why” is bad.

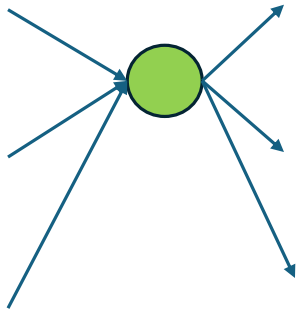
Topics of this course

- This course is more on the system side and less on the math/model part
- Included:
  - Training: Parallelism, compilation, memory, cluster management, …
  - Inference
- Not included:
  - Model architecture design: CNN, RNN, Transformer, ……
  - Hyperparameter tuning: Activation function, learning rate, ……
- Example: We will learn how to train a transformer efficiently, but not how to change its architecture to make it more effective

NN Training – A system perspective



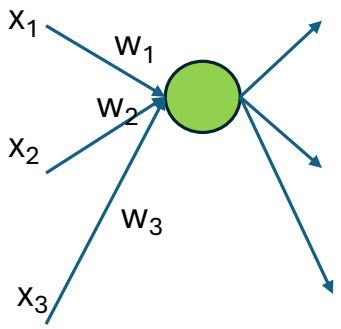
One neuron first



Outputs to the next layer are the same, so one neuron has only one output

One neuron first

It has a weight for each input

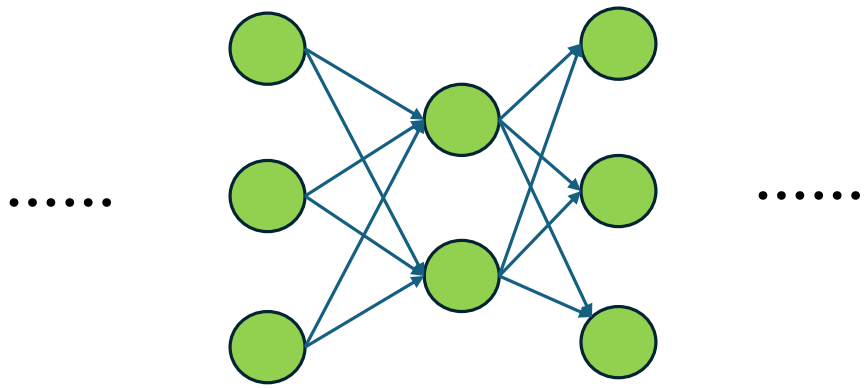


First do a weighted sum $z = w_1x_1 + w_2x_2 + w_3x_3 + b$

Then do a nonlinear transform called activation output = $\sigma(z)$

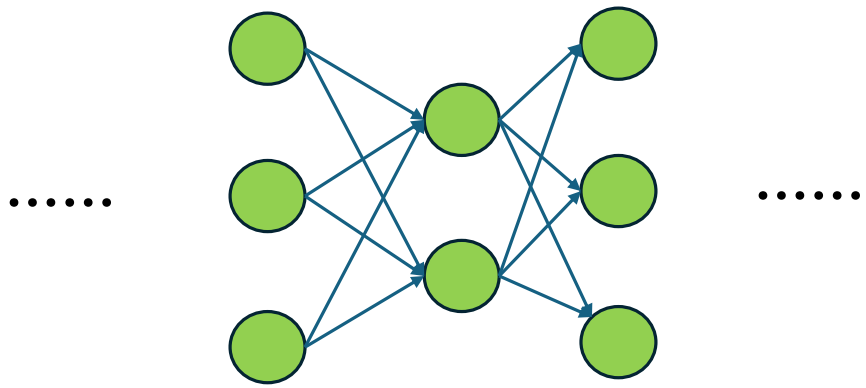
Examples of activation functions: ReLU, Sigmoid, etc

Go back to the full NN



Given inputs at the first (leftmost) layer, we perform the above computations layer by layer and finally get outputs at the last (rightmost) layer
This is pretty much what inference does.

NN Training – A system perspective

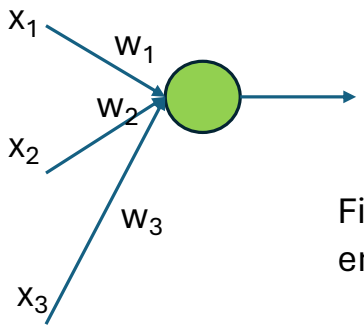


Goal of training: Find the appropriate weights for each neuron to minimize inference errors (or loss)

NN Training – A system perspective

- Three major steps
- Forward propagation: Same as above
- Backward propagation: Compute the loss and gradient
  - Gradient: How does changing a weight affect the loss
- Optimizer: Adjust the weight based on gradients

Backward Propagation – One Neuron

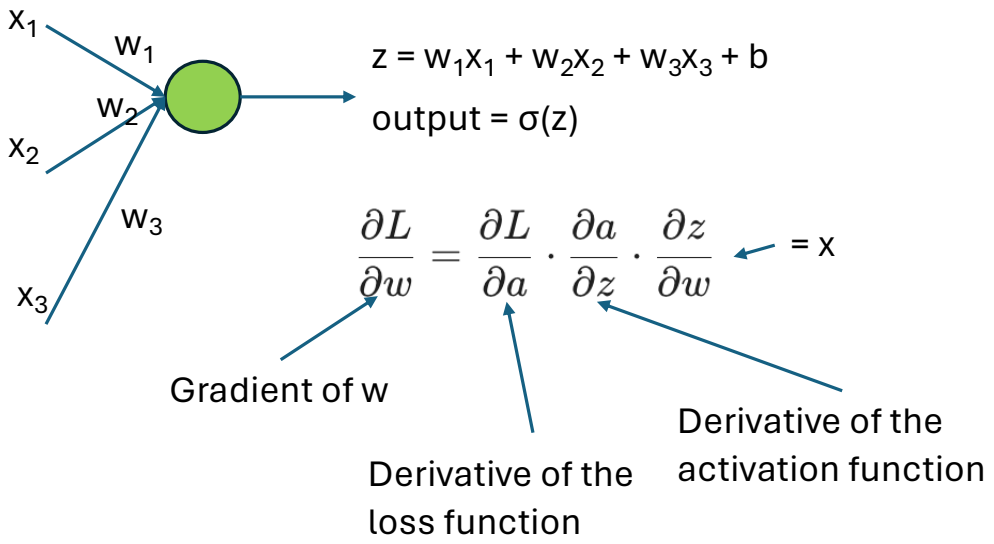


$z = w_1x_1 + w_2x_2 + w_3x_3 + b$

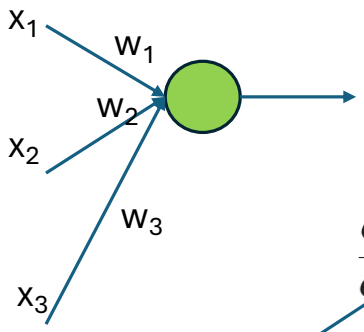
output = $\sigma(z)$

First, define a loss function L to compute the error/loss L(output, expected) (e.g., squared root)

Backward Propagation – One Neuron



Backward Propagation – One Neuron



$z = w_1x_1 + w_2x_2 + w_3x_3 + b$

output = σ(z)

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial w}$$

= x

Gradient of w

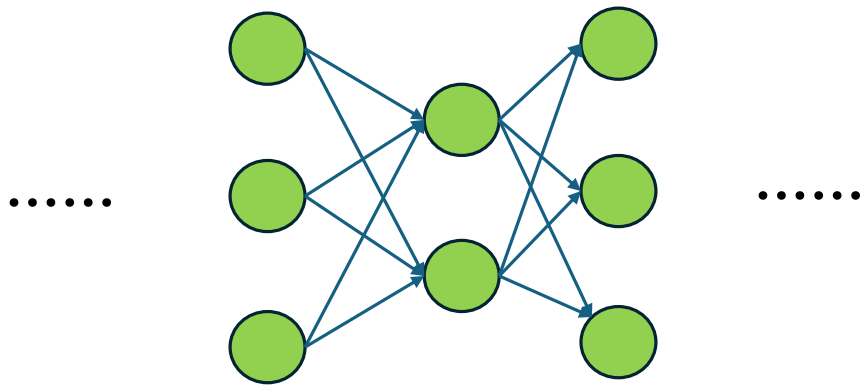
Derivative of the loss function

Derivative of the activation function

To do backward propagation, a neuron needs to store states during forward propagation

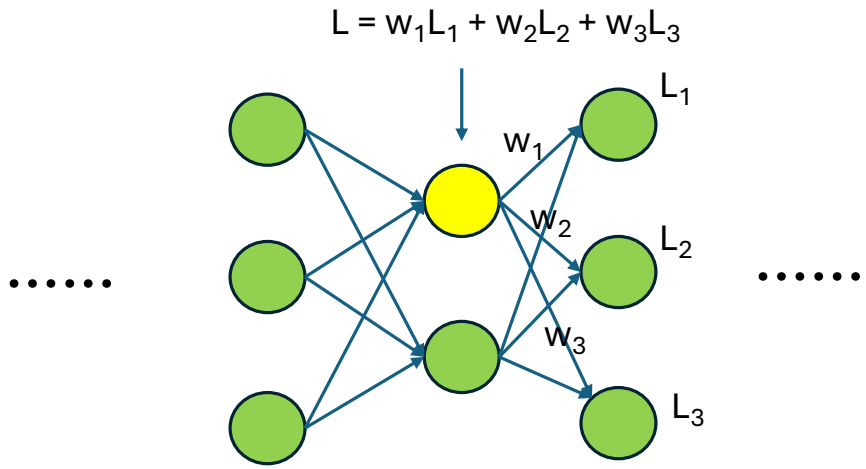
- At least x. Maybe intermediate results as well.
- This is called "activation state".

Backward Propagation - Full NN



Loss at the last layer is obvious. How about loss in other layers?
Answer: Propagation loss from the last layer to the first layer

Backward Propagation - Full NN



Loss at the last layer is obvious. How about loss in other layers?
Answer: Propagation loss from the last layer to the first layer
Then we can compute gradients at each neuron

Optimizer

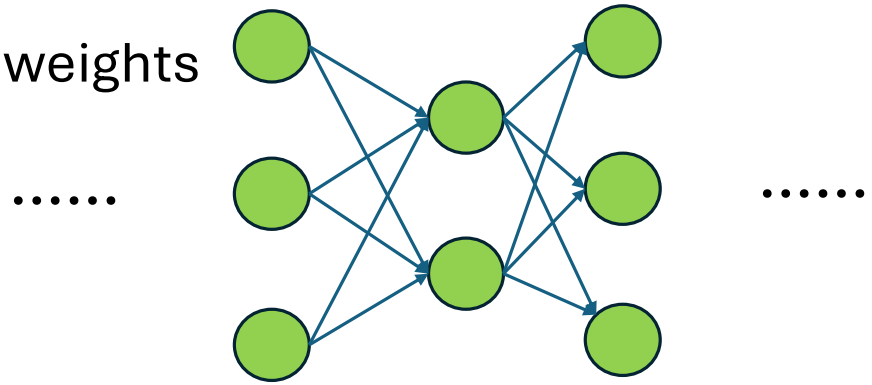
- Goal: Apply gradients to update weights
  - Computation is local to each neuron

- Simple version (SGD): $w_{new} = w_{old} - \eta * \text{gradient}$

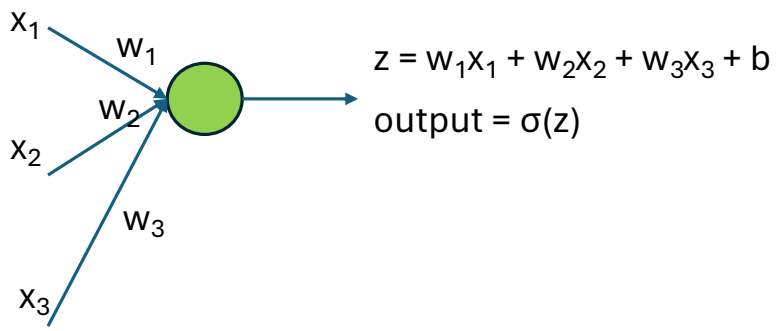
- Modern versions: Adam, etc
  - Also leverage historical information
  - Require each neuron to store such information called “optimizer state”

Summary so far

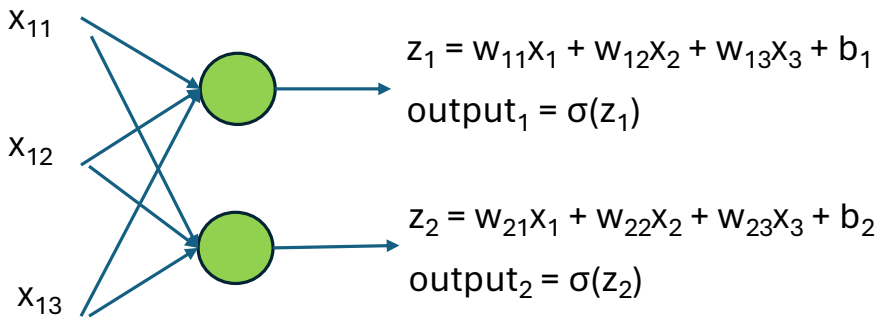
- Forward propagation: Use input from left to compute output on the right (weighted sum and activation)
- Backward propagation: Compute loss/error, propagate loss from right to left, and compute gradients
- Optimizer: Apply gradients to update weights



System perspective: Matrix Representation



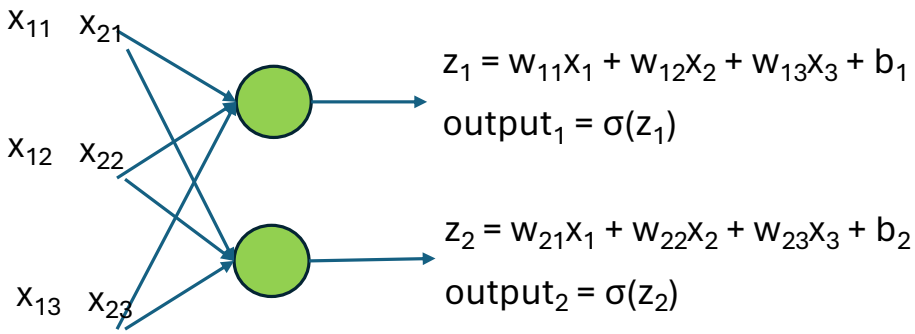
System perspective: Matrix Representation



$$[z_1 \quad z_2] = \begin{bmatrix} x_1 & x_2 & x_3 \\ x_1 & x_2 & x_3 \end{bmatrix} \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \end{bmatrix} + [b_1 \quad b_2]$$

System perspective: Matrix Representation

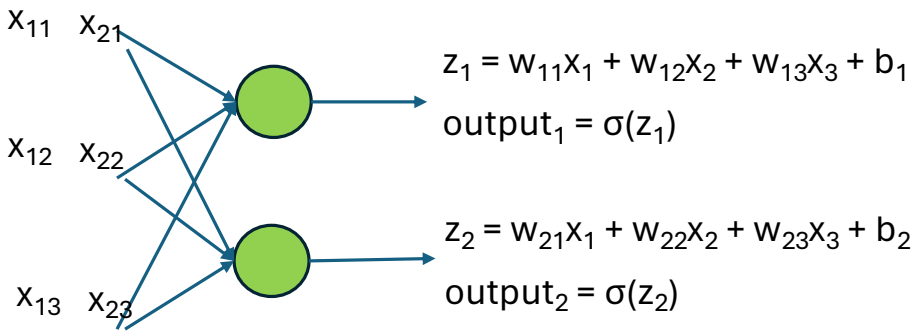
May train multiple inputs (batch) in one iteration



$$\begin{bmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \end{bmatrix} \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \end{bmatrix} + \begin{bmatrix} b_1 & b_2 \\ b_1 & b_2 \end{bmatrix}$$

System perspective: Matrix Representation

May train multiple inputs (batch) in one iteration



Accelerators like GPU are very good at matrix computation
Activation is a vector operation and is also highly parallelizable

$$\begin{bmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \end{bmatrix} \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \end{bmatrix} + \begin{bmatrix} b_1 & b_2 \\ b_1 & b_2 \end{bmatrix}$$

System perspective: Memory

- Weights
- Activation state: State for computing gradients
- Optimizer state: State for updating weights
- Activation and optimizer states can be much larger than weights without further optimization

System perspective: Challenges

- How to train a very large model:
  - A large amount of input
  - Using one GPU will take too long
  - States do not fit into the memory of one GPU
  - ……