Large-Scale Machine Learning: Neural Nets

CS246: Mining Massive Datasets
Jure Leskovec, Stanford University
http://cs246.stanford.edu
**Task:** Given data \((X,Y)\) build a model \(f\) to predict \(Y'\) based on \(X'\)

**Strategy:** Estimate \(y = f(x)\) on \((X,Y)\).

Hope that the same \(f(x)\) also works to predict unknown \(Y'\)

- The “hope” is called **generalization**
  - **Overfitting:** If \(f(x)\) predicts \(Y\) well but is unable to predict \(Y'\)
- **We want to build a model that generalizes well to unseen data**
1) Training data is drawn independently at random according to unknown probability distribution \( P(x, y) \)

2) The learning algorithm analyzes the examples and produces a classifier \( f \)

Given new data \((x, y)\) drawn from \( P \), the classifier is given \( x \) and predicts \( \hat{y} = f(x) \)

The loss \( L(\hat{y}, y) \) is then measured.

**Goal of the learning algorithm:**
Find \( f \) that minimizes expected loss \( E_P[\mathcal{L}] \)
Why is it hard?
We estimate $f$ on training data but want the $f$ to work well on unseen future (i.e., test) data.
Minimizing the Loss

- **Goal:** Minimize the expected loss
  \[
  \min_f \mathbb{E}_P[\mathcal{L}]
  \]
- But we don’t have access to \( P \) -- we only know the training data \( D \):
  \[
  \min_f \mathbb{E}_D[\mathcal{L}]
  \]
- So, we minimize the average loss on the training data:
  \[
  \min_f J(f) = \min_f \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f(x_i), y_i)
  \]

**Problem:** Just memorizing the training data gives us a perfect model (with zero loss)
Given:

- A set of \( N \) training examples
  - \( \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \)
- A loss function \( \mathcal{L} \)

Choose the model: \( f_\theta(x) \)

Find:

- The parameter \( \theta \) that minimizes the expected loss on the training data

\[
\min_{f} J(f) = \min_{f} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f_\theta(x_i), y_i)
\]
The Key Questions of ML

- What is the model $f_\theta(x)$?
- What is the loss $\mathcal{L}$?
- How do we optimize the loss?


What is the model \( f_\theta(x) \)?

- \( \theta \) refers to **all the parameters of a model** that we optimize

**Examples:**

- A “Linear layer” \( f_\theta(x) = Wx + b \)
  - Here \( \theta = \{W, b\} \)

- A Multi-layer Perceptron (MLP)
  \[ f_\theta(x) = W_2\sigma(W_1x + b_1) + b_2 \]
  - Here \( \theta = \{W_1, W_2, b_1, b_2\} \)
  - \( \sigma \) is an **activation function** (we will come to this later)
What is the loss $\mathcal{L}$?

- $\mathcal{L}$: loss function. Example: L2 loss
  \[
  \mathcal{L}(y, f(x)) = \|y - f(x)\|_2
  \]

  - Common loss functions for regression:
    - L2 loss, L1 loss, huber loss, ...
  
  - Common loss functions for classification:
    - Cross entropy, max margin (hinge loss), ...

- Example
One common loss for classification: **cross entropy (CE)**

- Label $y$ is a categorical vector ([one-hot encoding](#))
  - e.g. $y = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \end{bmatrix}^T$  
    - $y$ is of class “3”

- $f(x) = \text{Softmax}(g(x))$
  
  - $f(x)_i = \frac{e^{g(x)_i}}{\sum_{j=1}^{C} e^{g(x)_j}}$
  
  - Softmax normalizes a vector into a probability distribution that sums to 1
  
  - e.g. $f(x) = \begin{bmatrix} 0.1 & 0.3 & 0.4 & 0.1 & 0.1 \end{bmatrix}^T$

- $\text{CE}(y, f(x)) = -\sum_{i=1}^{C} (y_i \log f(x)_i)$
  
  - $= -\log(f(x)_{\text{correct class}})$

  - $y_i, f(x)_i$ are the **actual** and **predicted** value of the $i$-th class.

- **Intuition:** the lower the loss, the closer the prediction is to one-hot $y$
How to optimize the **objective function**?

**Gradient vector**: Direction and rate of fastest increase

\[ \nabla_{\Theta} \mathcal{L} = \left( \frac{\partial \mathcal{L}}{\partial \Theta_1}, \frac{\partial \mathcal{L}}{\partial \Theta_2}, \ldots \right) \]

- \( \Theta_1, \Theta_2 \ldots \): components of \( \Theta \)
- Recall **directional derivative** of a multi-variable function (e.g. \( \mathcal{L} \)) along a given vector represents the instantaneous rate of change of the function along the vector.
- **Gradient is the directional derivative in the direction of largest increase**

Gradient Descent

- **Iterative algorithm:** repeatedly update weights in the (opposite) direction of gradients until convergence

\[ \Theta \leftarrow \Theta - \eta \nabla_{\Theta} L \]

- **Training:** Optimize \( \Theta \) iteratively
  - **Iteration:** 1 step of gradient descent

- **Learning rate (LR) \( \eta \):**
  - Hyperparameter that controls the size of gradient step
  - Can vary over the course of training (LR scheduling)

- **Ideal termination condition:** \( 0 \) gradient
  - In practice, we stop training if it no longer improves performance on the **validation set** (part of dataset we hold out from training)
Problem with gradient descent:

- Exact gradient requires computing $\nabla_\theta \mathcal{L}(y, f(x))$, where $x$ is the entire dataset!
  - This means summing gradient contributions over all the points in the dataset
  - Modern datasets often contain billions of data points
  - Extremely expensive for every gradient descent step

Solution: Stochastic gradient descent (SGD)

- At every step, pick a different minibatch $\mathcal{B}$ containing a subset of the dataset, use it as input $x$
Minibatch SGD

- Concepts:
  - **Batch size**: the number of data points in a minibatch
    - E.g. number of nodes for node classification task
  - **Iteration**: 1 step of SGD on a minibatch
  - **Epoch**: one full pass over the dataset (# iterations is equal to ratio of dataset size and batch size)

- SGD is unbiased estimator of full gradient:
  - But there is no guarantee on the rate of convergence
  - In practice often requires tuning of learning rate
  - Common optimizer that improves over SGD:
    - Adam, Adagrad, Adadelta, RMSprop ...
Objective: \( \min_{\Theta} \mathcal{L}(y, f(x)) \)

In deep learning, the function \( f \) can be very complex.

To start simple, consider linear function

\[
f(x) = W \cdot x, \quad \Theta = \{W\}
\]

If \( f \) returns a scalar, then \( W \) is a learnable vector

\[
\nabla_W f = \left( \frac{\partial f}{\partial w_1}, \frac{\partial f}{\partial w_2}, \frac{\partial f}{\partial w_3}, \ldots \right)
\]

If \( f \) returns a vector, then \( W \) is the weight matrix

\[
\nabla_W f = W^T
\]

Jacobian matrix of \( f \)
How about a more complex function:

\[ f(x) = W_2(W_1x), \quad \Theta = \{W_1, W_2\} \]

Recall chain rule:

\[ \frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx} \]

E.g. \( \nabla_x f = \frac{\partial f}{\partial (W_1x)} \cdot \frac{\partial (W_1x)}{\partial x} \)

Back-propagation: Use of chain rule to propagate gradients of intermediate steps, and finally obtain gradient of \( \mathcal{L} \) w.r.t. \( \Theta \)
Back-propagation: General Setting

Suppose we can write out the local gradients:

\[ z = f(x, y) \]

\[ \frac{\partial z}{\partial x}, \quad \frac{\partial z}{\partial y} \]
And we further have $L = g(z)$, with the upstream gradient $\frac{\partial L}{\partial z}$.

How do we compute $\frac{\partial L}{\partial x}$ and $\frac{\partial L}{\partial y}$? Back prop!
Back-propagation: General Setting

Back prop: Chain rule

Chain rule:
\[
\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}
\]

Credit: Stanford CS231n
Back-propagation: General Setting

Credit: Stanford CS231n

Back prop: Chain rule

Chain rule:
\[
\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}
\]

“Downstream gradients”

“Upstream gradient”

“Local gradient”
And we will recursively apply back propagation throughout the computation graph.
Back-propagation Example

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]
Back-propagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

Credit: Stanford CS231n
Back-propagation Example

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

Gradient computation for function \( q \)

Credit: Stanford CS231n
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

Gradient computation for function \( q \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

Gradient computation for function \( f \)

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Credit: Stanford CS231n
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

\[ \text{e.g. } x = -2, y = 5, z = -4 \]

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Gradient computation for function \( q \)

Gradient computation for function \( f \)

Want: \[ \frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z} \]

We want to compute gradients with respect to the input \( x, y, z \)

Credit: Stanford CS231n
Back-propagation Example

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)

Let's start back prop!

Credit: Stanford CS231n
Backpropagation Example

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)

The gradient value

Let's start back prop!

Credit: Stanford CS231n
Backpropagation Example

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)

Credit: Stanford CS231n
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, \ y = 5, \ z = -4 \)

\[
q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1
\]

\[
f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q
\]

Want: \( \frac{\partial f}{\partial x}, \ \frac{\partial f}{\partial y}, \ \frac{\partial f}{\partial z} \)

The gradient value

The gradient function that we need here

Back prop to \( z \)

Credit: Stanford CS231n
Back-propagation Example

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)

Credit: Stanford CS231n
Back-propagation Example

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

\[ \text{e.g. } x = -2, y = 5, z = -4 \]

\[ q = x + y \]
\[ \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \]
\[ \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \[ \frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z} \]

Credit: Stanford CS231n
Back-propagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want:

\[ \frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z} \]

Chain rule:

\[ \frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y} \]

Back prop to \( y \)

Upstream gradient

This is -4

Local gradient

This is 1

Credit: Stanford CS231n
Back-propagation Example

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)

The gradient value

Chain rule:

\[ \frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y} \]

Credit: Stanford CS231n
Back-propagation Example

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, \ y = 5, \ z = -4 \)

\[
q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1
\]

\[
f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q
\]

Want: \( \frac{\partial f}{\partial x}, \ \frac{\partial f}{\partial y}, \ \frac{\partial f}{\partial z} \)

Chain rule:

\[
\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \cdot \frac{\partial q}{\partial x} + \frac{\partial f}{\partial z} \cdot \frac{\partial z}{\partial x}
\]

Back prop to \( x \)

Credit: Stanford CS231n
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, \ y = 5, \ z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \ \frac{\partial f}{\partial y}, \ \frac{\partial f}{\partial z} \)

The gradient value

Chain rule:

\[ \frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x} \]

Back prop to \( x \)

Upstream gradient

This is -4

Local gradient

This is 1

Credit: Stanford CS231n
Non-linearity

- Note that in $f(x) = W_2(W_1x)$, $W_2W_1$ is another matrix (vector, if we do binary classification)
- Hence $f(x)$ is still linear w.r.t. $x$ no matter how many weight matrices we compose
- Introduce non-linearity:
  - Rectified linear unit (ReLU)
    $$ReLU(x) = \max(x, 0)$$
  - Sigmoid
    $$\sigma(x) = \frac{1}{1 + e^{-x}}$$
Each layer of MLP combines linear transformation and non-linearity:

\[ x^{(l+1)} = \sigma(W_l x^{(l)} + b^l) \]

- where \( W_l \) is weight matrix that transforms hidden representation at layer \( l \) to layer \( l + 1 \)
- \( b^l \) is bias at layer \( l \), and is added to the linear transformation of \( x \)
- \( \sigma \) is non-linearity function (e.g., sigmoid)

Suppose \( x \) is 2-dimensional, with entries \( x_1 \) and \( x_2 \)
Objective function:
\[
\min_{\Theta} \mathcal{L}(y, f(x))
\]

- \(f\) can be a simple linear layer, an MLP, or other neural networks (e.g., a GNN later)
- Sample a minibatch of input \(x\)
- **Forward propagation:** Compute \(\mathcal{L}\) given \(x\)
- **Back-propagation:** Obtain gradient \(\nabla_{\Theta} \mathcal{L}\) using the chain rule
- Use **stochastic gradient descent (SGD)** to optimize for \(\Theta\) over many iterations
Convolutional Neural Networks
CNN on an image:

Convolutional Networks

[Diagram showing the process of a CNN on an image, including convolutional and subsampling layers.]

Input → Feature maps → Convolutions → Subsampling → Convolutions → Subsampling → Fully connected → Output
**A Naïve Approach**

- Flatten an image into a vector
- Then apply a Linear layer \( f(x) = Wx \)

**Issues:**
- Cannot work with images with different sizes
- Need \( O(N^2D) \) parameters
  - \( N \) is the width/length of image, \( D \) is the number of channels

---

[Credit: Stanford CS231n]
Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Credit: Stanford CS231n
**Convolution Layer**

32*32*3 image

5*5*3 filter

The result of taking the dot product between the filter and a small 5*5*3 chunk of the image.

(i.e. 5*5*3 = 75-dimensional dot product + bias)

Credit: Stanford CS231n
Convolve Layer

Convolve (slide) over all spatial locations

28*28*1 output
- **Solve the previous issues:**
  - The same Conv layer can work with images with different sizes
  - Only Need $O(K^2D)$ parameters rather than $O(N^2D)$
    - 5*5*3 parameters vs 32*32*3 parameters
- **Solve the previous issues:**
  - The same Conv layer can work with images with different sizes
  - Only Need $O(K^2D)$ parameters rather than $O(N^2D)$
    - 5*5*3 parameters vs 32*32*3 parameters

$K$: size of kernel

$N$: size of image
Graph Neural Networks
Assume we have a graph $G$:

- $V$ is the vertex set
- $A$ is the adjacency matrix (assume binary)
- $X \in \mathbb{R}^{m \times |V|}$ is a matrix of node features
- $v$: a node in $V$; $N(v)$: the set of neighbors of $v$

**Node features:**

- Social networks: User profile, User image
- Biological networks: Gene expression profiles, gene functional information
- When there is no node feature in the graph dataset:
  - Indicator vectors (one-hot encoding of a node)
  - Vector of constant 1: $[1, 1, \ldots, 1]$
Join adjacency matrix and features
Feed them into a deep neural net:

Issues with this idea:
- $O(|V|)$ parameters
- Not applicable to graphs of different sizes
- Sensitive to node ordering
But our graphs look like this:

- There is no fixed notion of locality or sliding window on the graph
- Graph is permutation invariant

Credit: Stanford CS224W
Single Convolutional neural network (CNN) layer with 3x3 filter:

**Idea:** transform information at the neighbors and combine it:

- Transform “messages” $h_i$ from neighbors: $W_i h_i$
- Add them up: $\sum_i W_i h_i$

Credit: Stanford CS224W
Idea: Node’s neighborhood defines a computation graph

Learn how to propagate information across the graph to compute node features

Credit: Stanford CS224W

Determine node computation graph

Propagate and transform information
**Key idea:** Generate node embeddings based on local network neighborhoods

Credit: Stanford CS224W
**Intuition:** Nodes aggregate information from their neighbors using neural networks.
**Intuition:** Network neighborhood defines a computation graph

Every node defines a computation graph based on its neighborhood!

Credit: Stanford CS224W
Model can be of arbitrary depth:

- Nodes have embeddings at each layer
- Layer-0 embedding of node $u$ is its input feature, $x_u$
- Layer-$k$ embedding gets information from nodes that are $K$ hops away
Neighborhood aggregation: Key distinctions are in how different approaches aggregate information across the layers.
Basic approach: Average information from neighbors and apply a neural network

(1) average messages from neighbors

(2) apply neural network

Credit: Stanford CS224W
**Basic approach:** Average neighbor messages and apply a neural network

Initial 0-th layer embeddings are equal to node features:

$$ h_v^0 = x_v $$

Embedding after $L$ layers of neighborhood aggregation:

$$ z_v = h_v^{(L)} $$

Average of neighbor’s previous layer embeddings:

$$ h_v^{(l+1)} = \sigma(W_l \sum_{u \in N(v)} \frac{h_u^{(l)}}{|N(v)|} + B_l h_v^{(l)}), \forall l \in \{0, \ldots, L - 1\} $$

Non-linearity (e.g., ReLU)

Credit: Stanford CS224W
How do we train the model to generate embeddings?

Need to define a loss function on the embeddings

Credit: Stanford CS224W
We can feed these embeddings into any loss function and run SGD to train the weight parameters.

$\mathbf{h}_v^{(0)} = \mathbf{x}_v$

$\mathbf{h}_v^{(l+1)} = \sigma(\mathbf{W}_l \sum_{u \in \mathbf{N}(v)} \frac{\mathbf{h}_u^{(l)}}{|\mathbf{N}(v)|} + \mathbf{B}_l \mathbf{h}_v^{(l)}), \forall l \in \{0, ..., L - 1\}$

$\mathbf{z}_v = \mathbf{h}_v^{(L)}$

$\mathbf{h}_v^l$: the hidden representation of node $v$ at layer $l$

- $\mathbf{W}_k$: weight matrix for neighborhood aggregation
- $\mathbf{B}_k$: weight matrix for transforming hidden vector of self

Credit: Stanford CS224W
Node embedding $z_v$ is a function of input graph.

**Supervised setting**: we want to minimize the loss $\mathcal{L}$ (see also slide 15):

$$\min_{\Theta} \mathcal{L}(y, f(z_v))$$

- $y$: node label
- $\mathcal{L}$ could be L2 if $y$ is real number, or cross entropy if $y$ is categorical.
Directly train the model for a supervised task (e.g., node classification)

Safe or toxic drug?

E.g., a drug-drug interaction network
Directly train the model for a supervised task (e.g., node classification)

- Use cross entropy loss (slide 16)

\[
\mathcal{L} = \sum_{v \in V} y_v \log(\sigma(z_v^T \theta)) + (1 - y_v) \log(1 - \sigma(z_v^T \theta))
\]

Encoder output: node embedding
Classification weights
Node class label
Safe or toxic drug?

Credit: Stanford CS224W
Model Design: Overview

(1) Define a neighborhood aggregation function

(2) Define a loss function on the embeddings

Credit: Stanford CS224W
(3) Train on a set of nodes, i.e., a batch of compute graphs
(4) Generate embeddings for nodes as needed

Even for nodes we never trained on!
Outline of Today's Lecture

- **The key considerations for ML:**
  - The model $f_\theta(x)$: Linear model, neural networks, ...
  - The loss $\mathcal{L}$: Classification & regression losses
  - Optimize the loss: Backpropagation
- **Two widely used Deep Learning models**
  - Convolutional Neural Networks
  - Graph Neural Networks