291K
Deep Learning for Machine Translation
Basic Neural Networks

Lei Li
UCSB

10/4/2021

Outline

- MT as a ML problem
- Basic Neural Net Layers
  - Single artificial neuron, Word Embedding, Feed-forward, Softmax, Positional Embedding
  - Universal approximation
- Model Training
  - Risk Minimization and Maximum Likelihood Estimation
- Stochastic Optimization methods
  - SGD and Backpropogation
  - Adaptive gradient methods: Adagrad, Adam

What is Machine Learning?

- A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E”
  - [Tom Mitchell, Machine Learning, 1997]

Task T

- To find a function f: x -> y
  - Classification: label y is categorical
  - Regression: label y is continuous numerical
- Example:
  - Image classification
    - Input space: x in $R^{h \times h \times 3}$ is h x h pixels (rgb), so it is a tensor of h x h x 3.
    - Output space: y is {1..10} in Cifar-10, or {1..1000} in ImageNet.
  - Text-to-Image generation
    - Input: x is a sentence in V^L, V is vocabulary, L is length
    - Output: y is $R^{h \times h \times 3}$

Formulation of NLP Tasks

- Text classification: sentence (or document) => label
  - Sentiment prediction
  - Intent classification
  - NLI: natural language inference, logical relation of two sentences
- Sequence Generation/Structured Prediction: Given an input, to predict a sequence of labels
  - Machine Translation
  - Dialog response generation
  - Named entity recognition
- Sentence Retrieval/Matching
  - Comparing similarity of two sequences

Experience E

- Supervised Learning: if pairs of (x, y) are given
- Unsupervised Learning: if only x are given, but not y
- Semi-supervised Learning: both paired data and raw data
- Self-supervised Learning:
  - use raw data but construct supervision signals from the data itself
  - e.g. to predict neighboring pixel values for an image
  - e.g. to predict neighboring words for a sentence

How Experience is Collected?

- Offline/batch Learning:
  - All data are available at training time
  - At inference time: fix the model and predict
- Online Learning:
  - Experience data is collected one (or one mini-batch) at a time (can be either labeled or unlabeled)
  - Incrementally train and update the model, and make predictions on the fly with current and changing model
  - e.g. predicting ads click on search engine
- Reinforcement Learning:
  - A system (agent) is interacting with an environment (or other agents) by making an action
  - Experience data (reward) is collected from environment.
  - The system learns to maximize the total accumulative rewards.
  - e.g. Train a system to play chess

Learning w/ various Number of Tasks

- Multi-task learning
  - one system/model to learn multiple tasks simultaneously, with shared or separate Experience, with different performance measures
  - e.g. training a model that can detect human face and cat face at the same time
- Pre-training & Fine-tuning
  - Pre-training stage: A system is trained with one task, usually with very large easily available data
  - Fine-tuning stage: it is trained on another task of interest, with different (often smaller) data
  - e.g. training an image classification model on ImageNet, then finetune on object detection dataset.

Machine Translation as a Machine Learning Task

- Input (Source)
  - – discrete sequence in source language, V_s
- Output (Target)
  - – discrete sequence in target langauge, V_t
- Experience E
  - – Supervised: parallel corpus, e.g. English-Chinese parallel pairs
  - – Unsupervised: monolingual corpus, e.g. to learn MT with only Tamil text and English text, but no Eng-Tamil pairs
  - – Semi-supervised: both
- Number of languages involved
  - – Bilingual versus Multilingual MT
  - – Notice: it can be multilingual parallel data, or multilingual monolingual data
- Measure P
  - – Human evaluation metric, or Automatic Metric (e.g. BLEU), see previous lecture

What is Deep Learning

- Deep learning is a particular kind of machine learning
- that achieves great power and flexibility by representing the world as a nested hierarchy of concepts,
- with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones.

Ian Goodfellow and Yoshua Bengio and Aaron Courville.
Deep Learning, 2016

Neural Networks

- Given a labeled dataset $\{(x_n, y_n)\}$, how to train a model that maps from x —> y
- Idea: develop a complex model using massive basic simple units

Inspired by a biological neuron

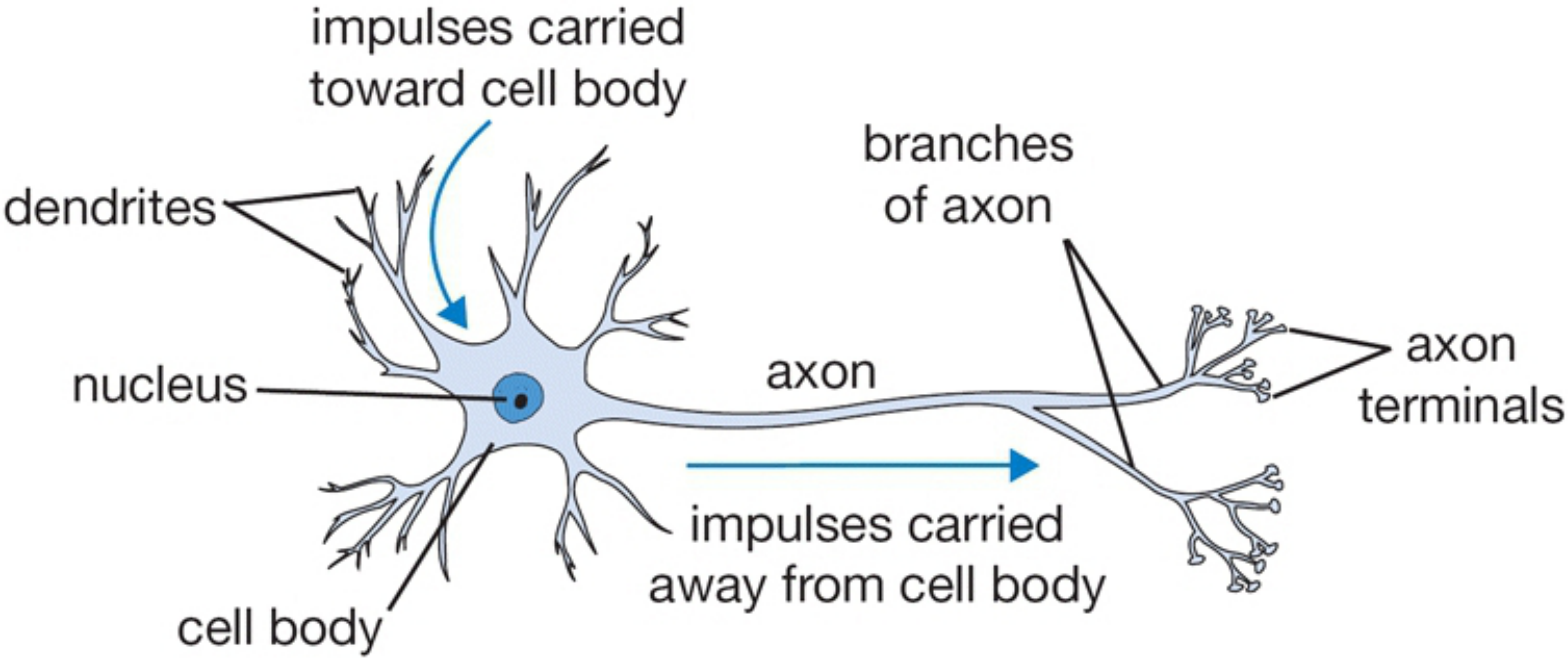
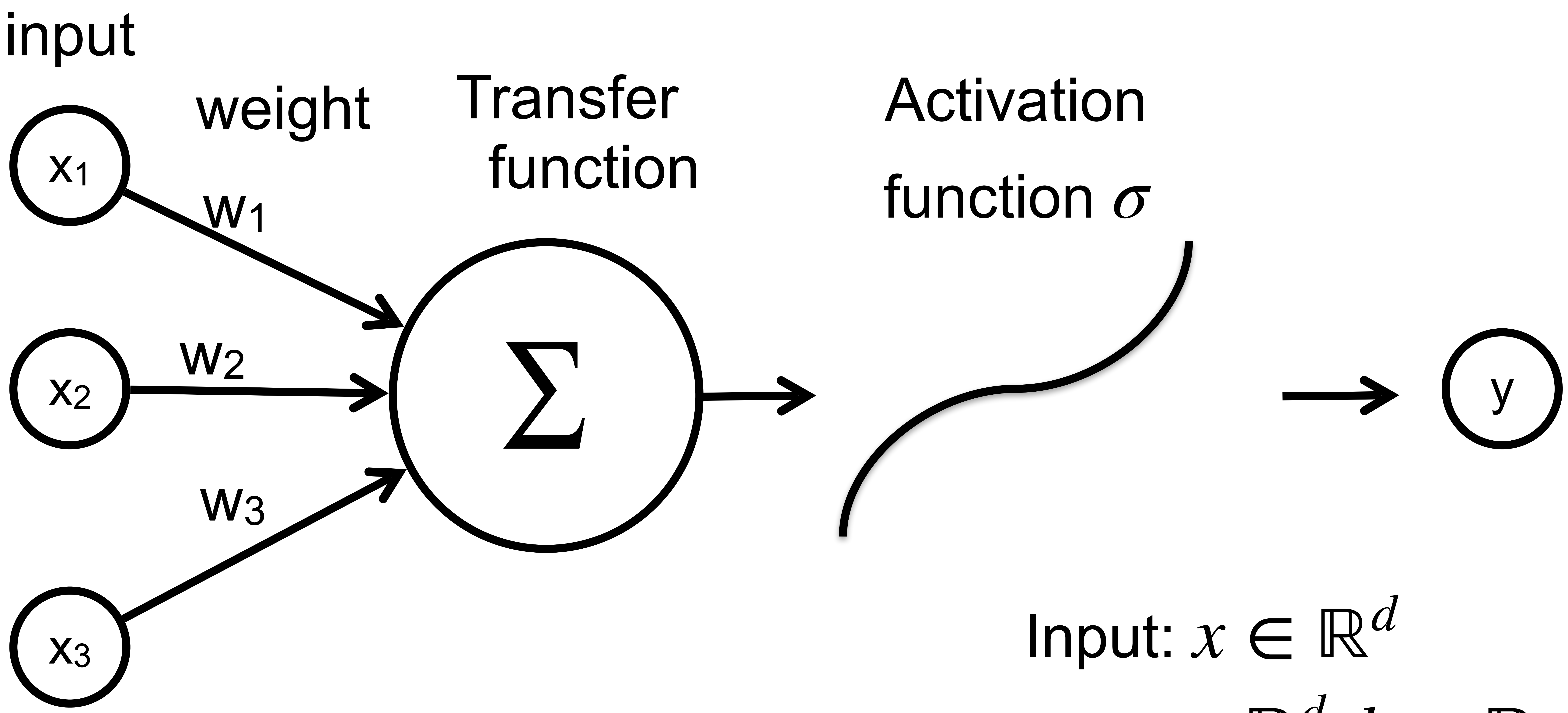


Image credit:

http://cs231n.github.io/neural-networks-1/

A single Artificial Neuron



Input: $x \in \mathbb{R}^d$

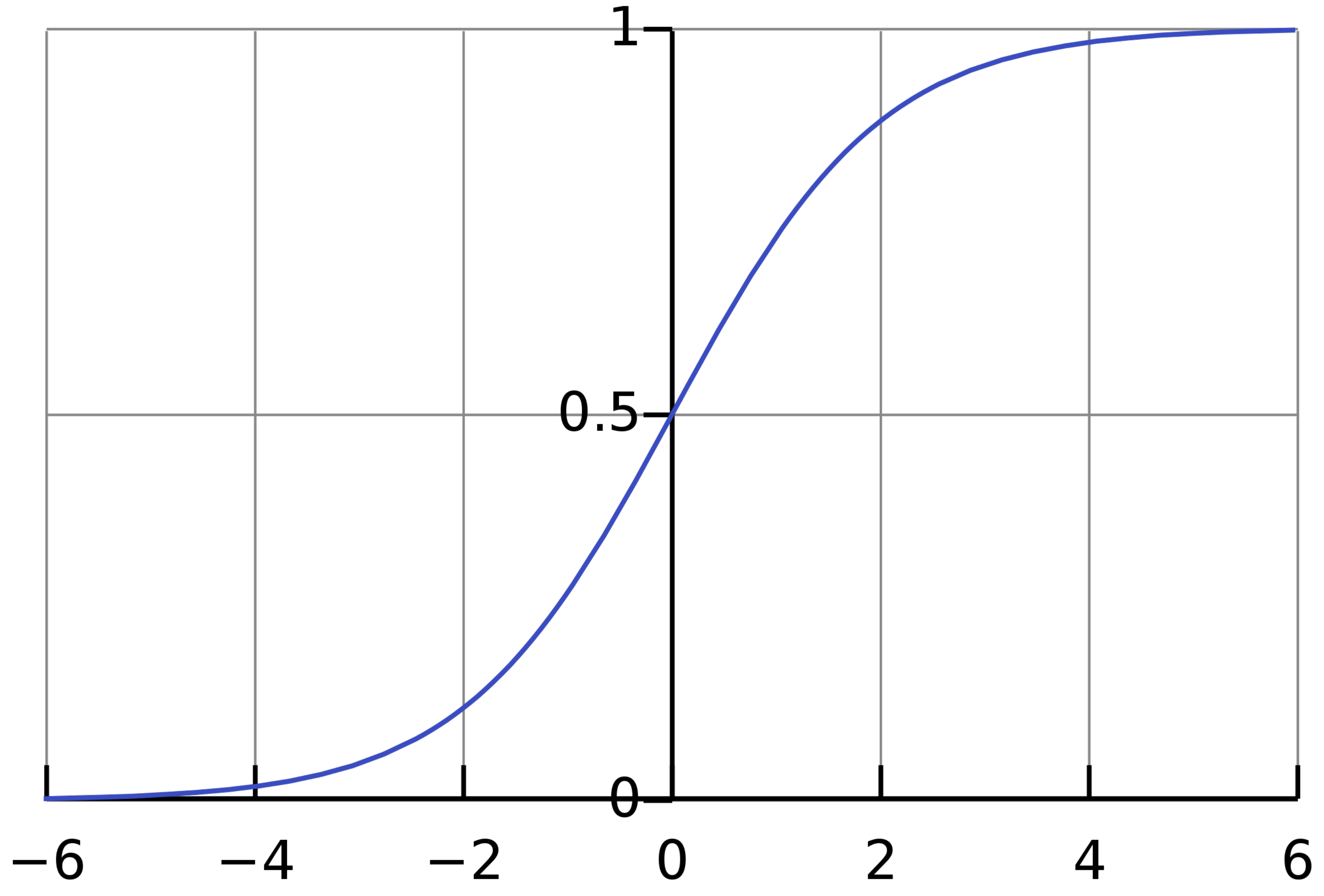
Weight: $w \in \mathbb{R}^d, b \in \mathbb{R}$

Output: $y = \sigma(w \cdot x + b)$

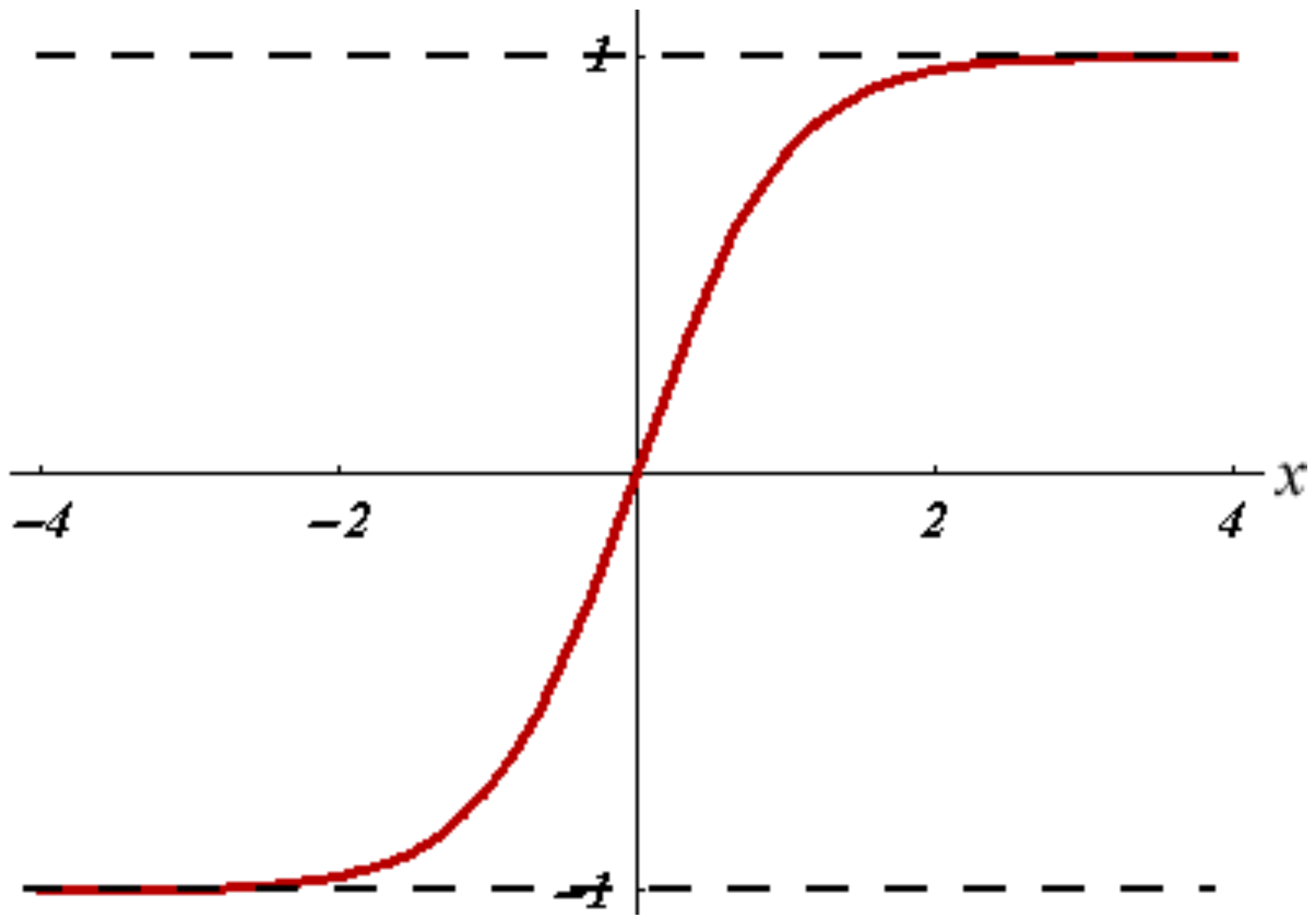
Activation functions

Activation function is nonlinear

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$



$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

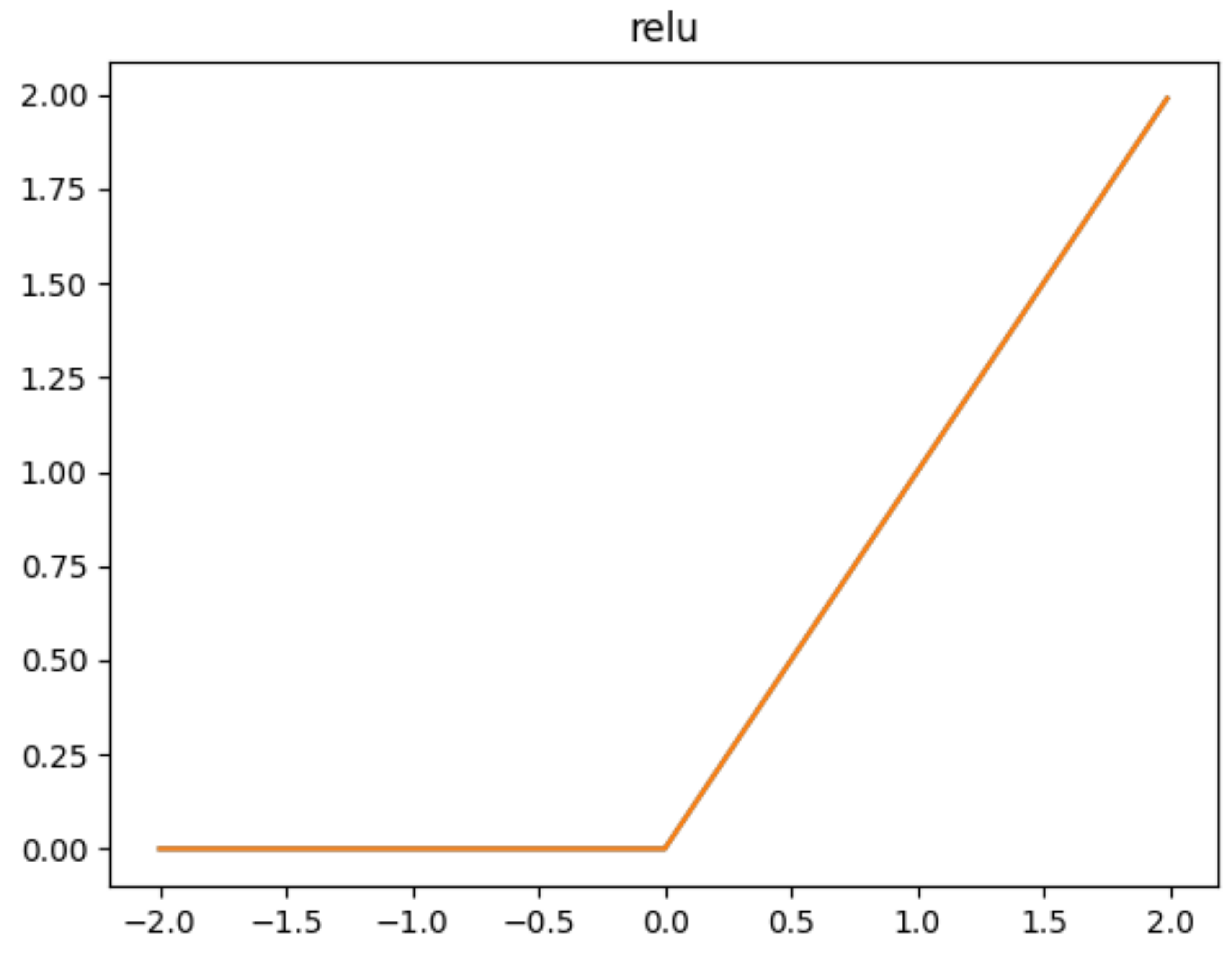


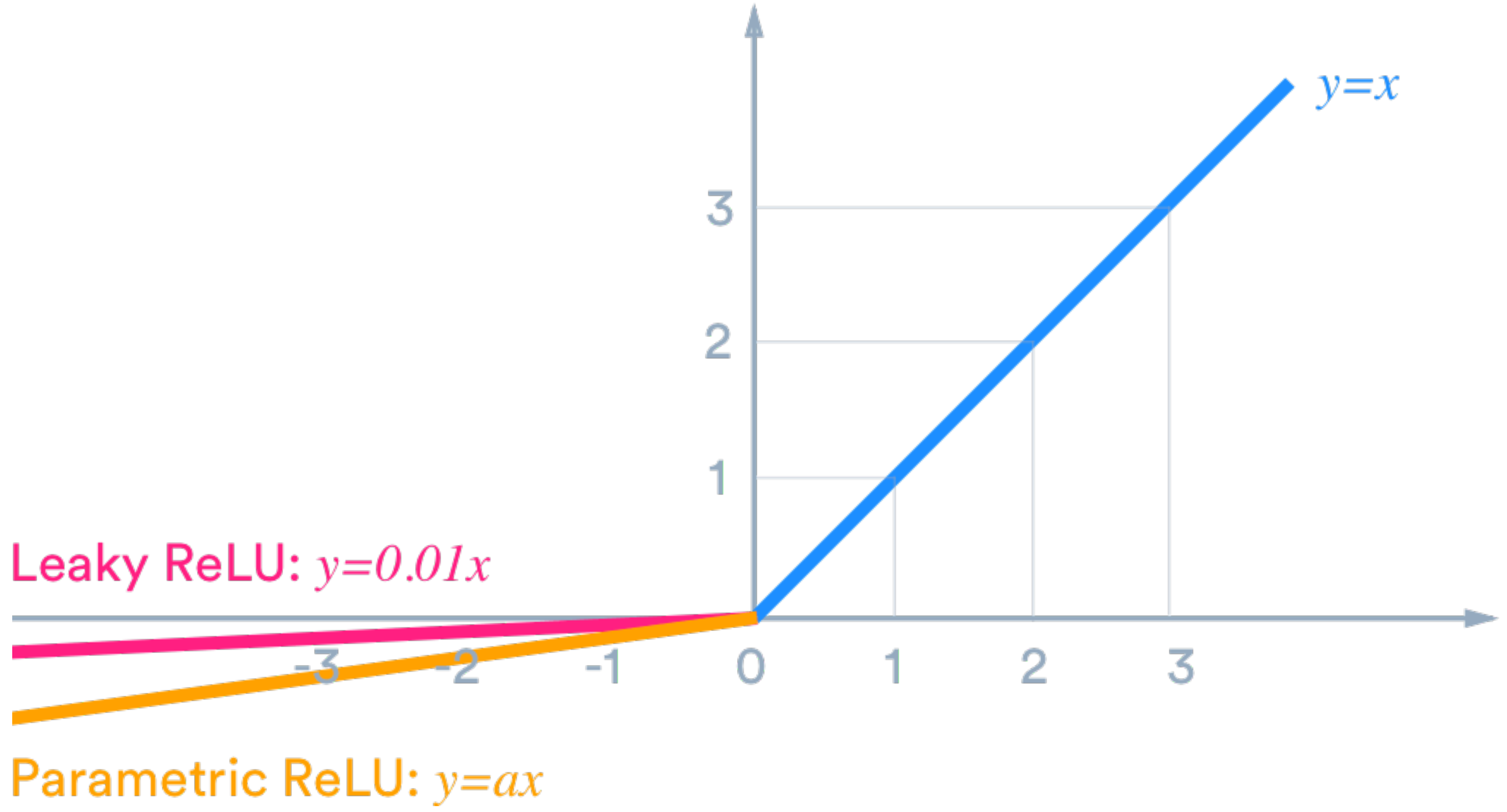
Activation functions

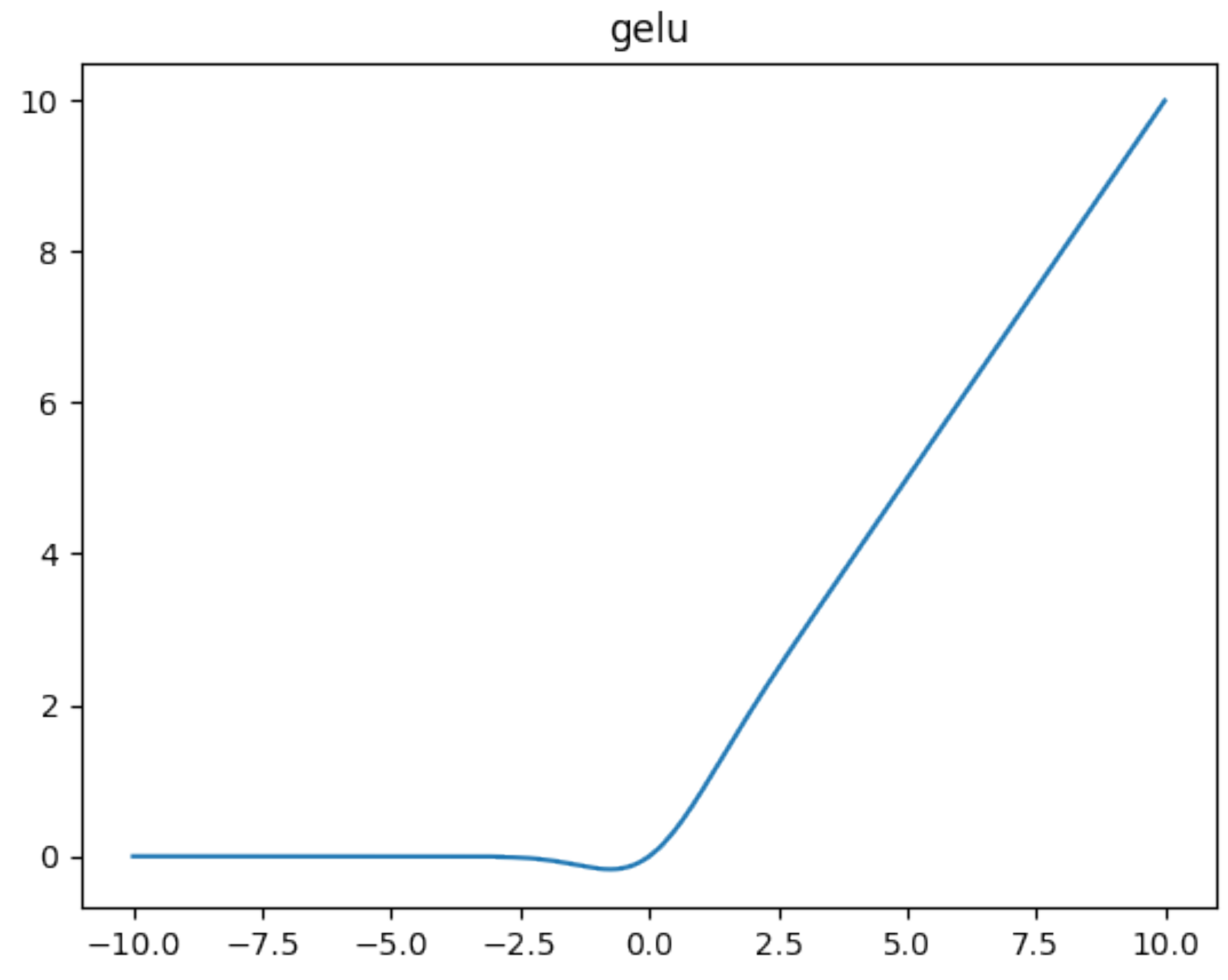
$$\text{GELU}(x) = 0.5x\left(1 + \tanh\left(\sqrt{2/\pi}(x + 0.044715x^3)\right)\right)$$

$$\text{relu}(x) = \max(0,x)$$

Leaky Relu







Softmax

$$\mathrm{softmax}(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

Useful for modeling probability

(in classification task)

Running Example: Predicting Sentiment

Given a sentence, to predict sentiment label:

positive, neural, negative

This
movie
is
great

$\xrightarrow{?}$

0
1
2

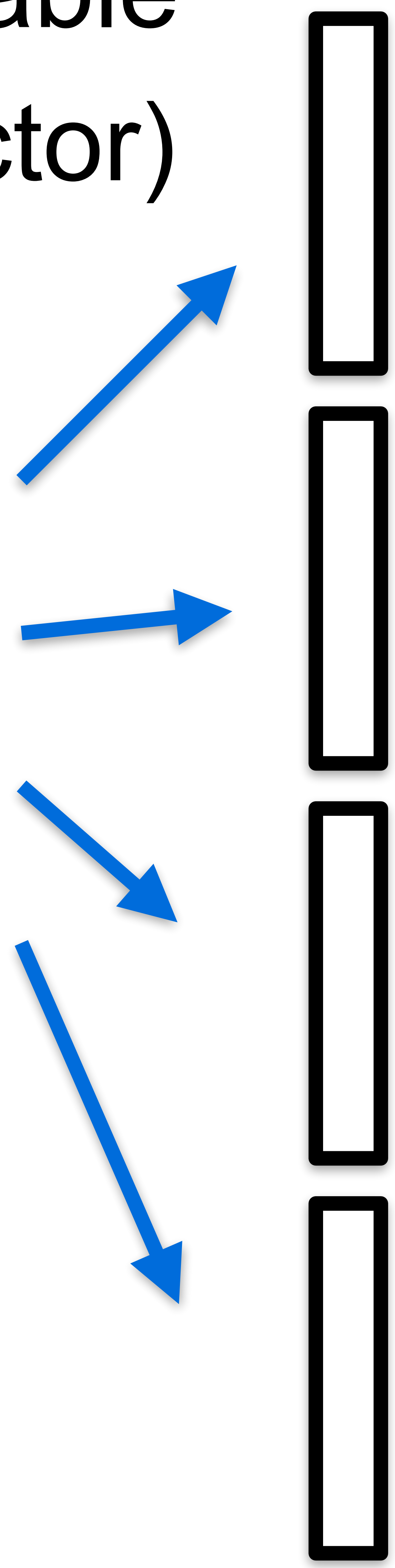
Word Embedding: Discrete Input to Continuous Representation

This
movie
is
great

Vocabulary
a -> 1
is -> 8
this -> 25
that -> 26
great -> 532
movie -> 876
…

Lookup table
(id-to-vector)

25
876
8
532



how large is the lookup table?

V·d

Typical:

V=30k

d=100

Single-Layer Neural Net

For simplicity: start from single word input

Input: $x \in \mathbb{R}^d$

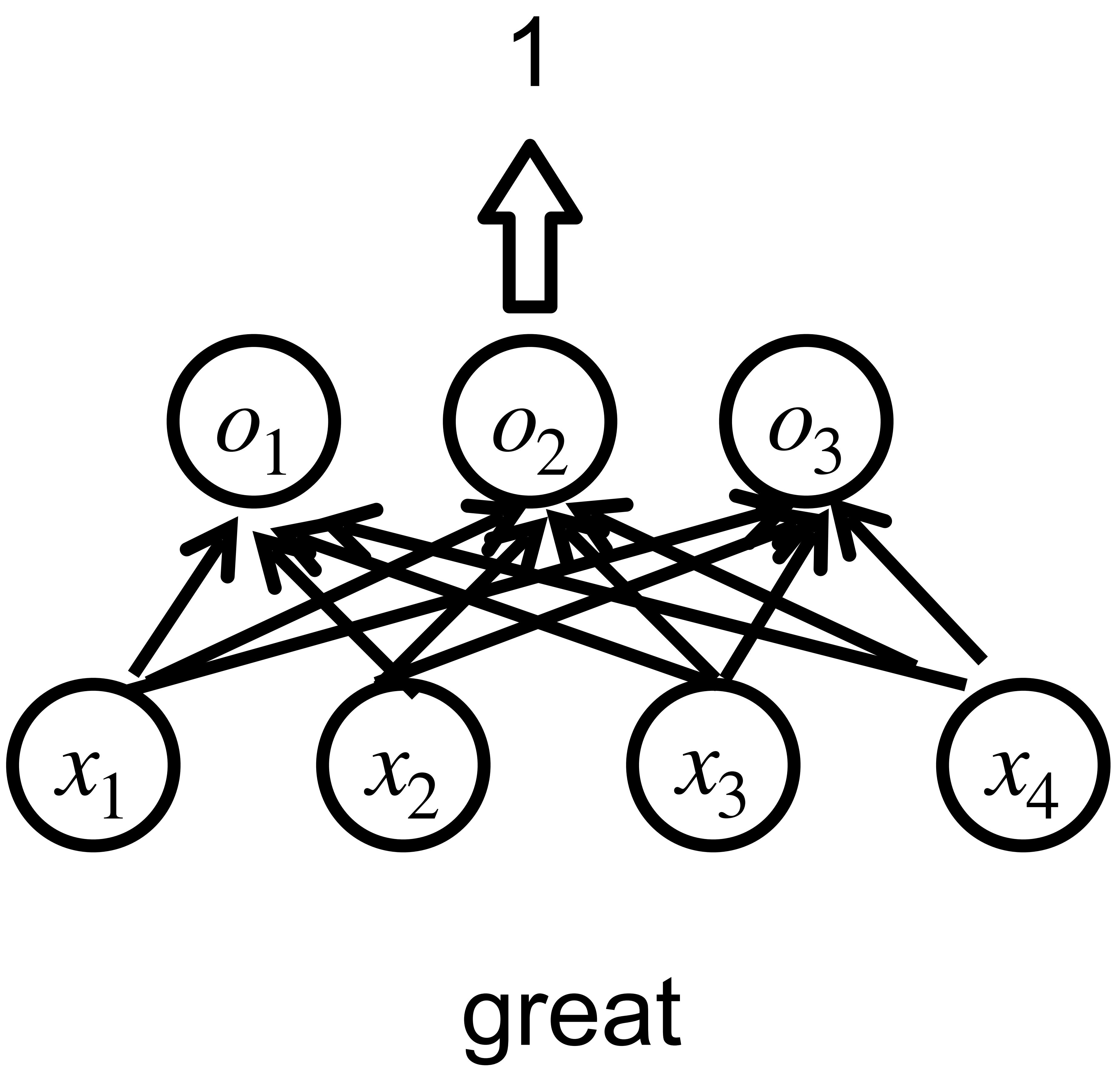
Weight: $w \in \mathbb{R}^d, b \in \mathbb{R}$

Output: $o = \mathrm{Softmax}(w \cdot x + b) \in \mathbb{R}^3$

o_1, o_2, o_3 representing probabilities of positive, neutral, and negative labels

The prediction is chosen by

$$y = \operatorname*{argmax}_i o_i$$



Multi-layer Feed-forward Neural Net

- also known as multilayer perceptron

$$x \in \mathbb{R}^d$$

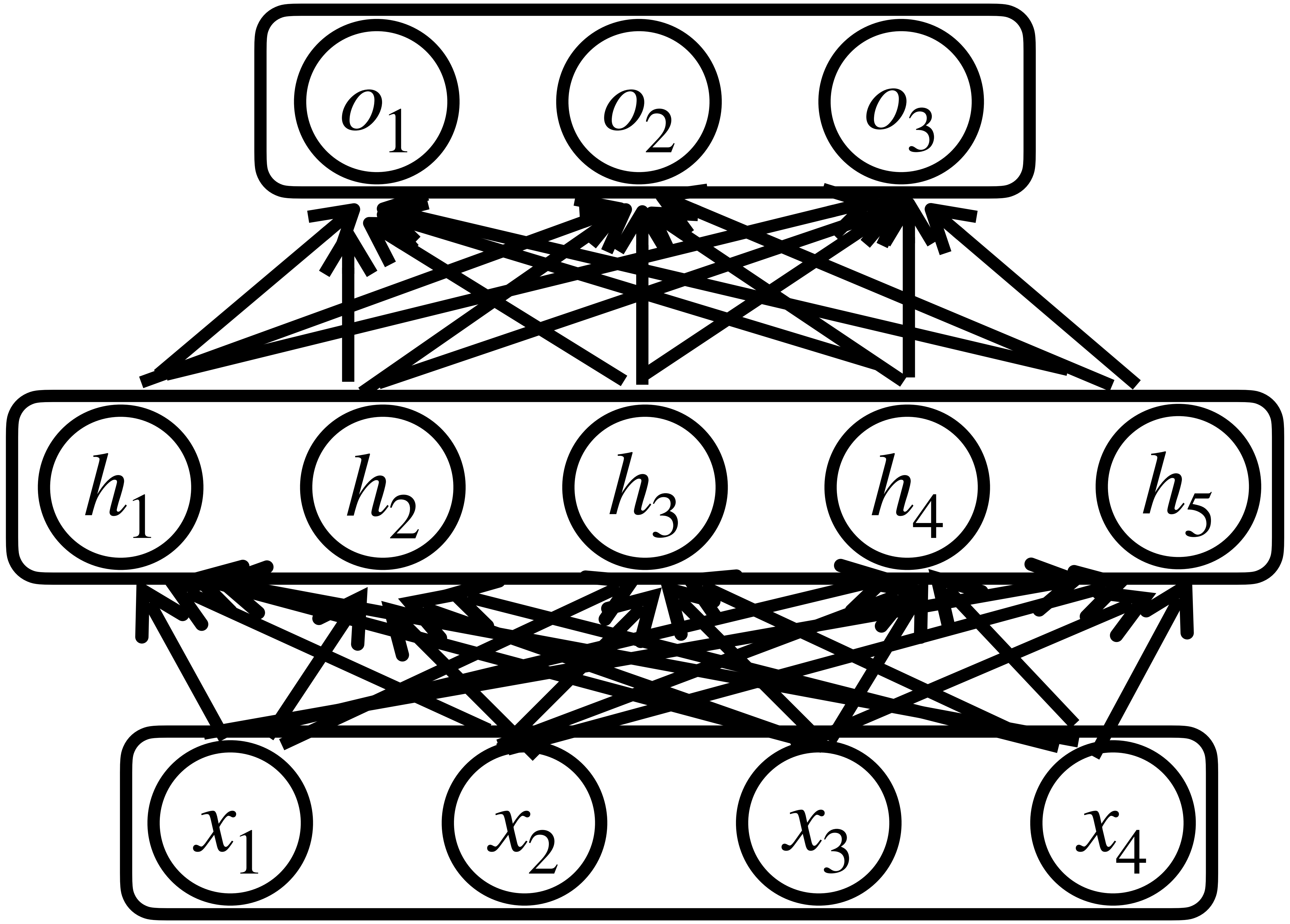
$$h_1 = \sigma(w_1 \cdot x + b_1) \in \mathbb{R}^{d_1}$$

$$h_2 = \sigma(w_2 \cdot h_1 + b_2) \in \mathbb{R}^{d_2}$$

$$o = \mathrm{Softmax}(w_3 \cdot h_2 + b_3) \in \mathbb{R}^3$$

Parameters

$$\theta = \{w_1, b_1, w_2, b_2, w_3, b_3\}$$



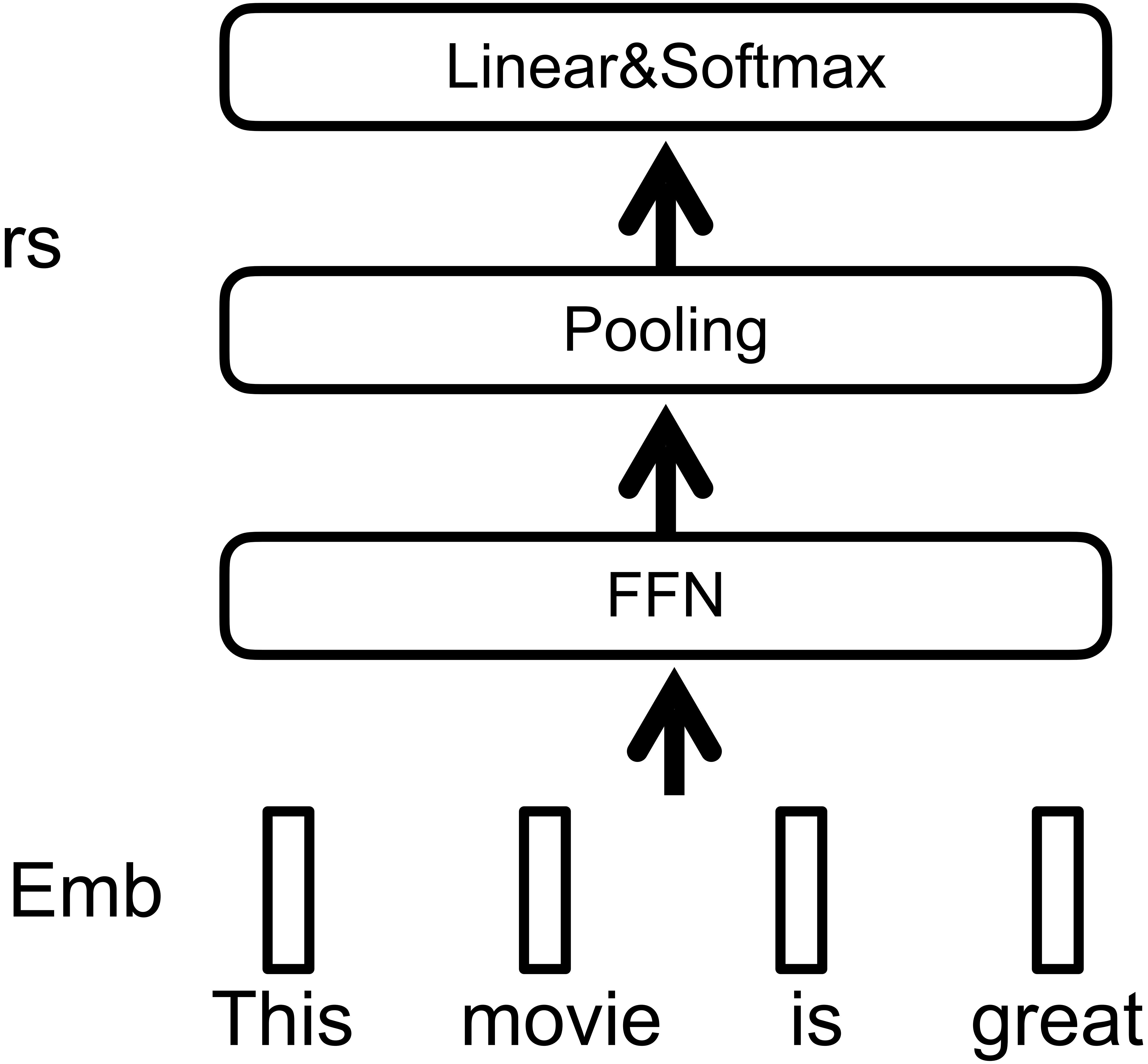
Sentence with Variable Length

- Pooling Layer
  - Element-wise operation to cmpress variable length vectors into a fixed-size vector
  - Average pooling

$$h^{next} = \frac{1}{L}\sum_i h_i$$

  - Max pooling

$$h_j^{next} = \max_i h_{i,j}$$

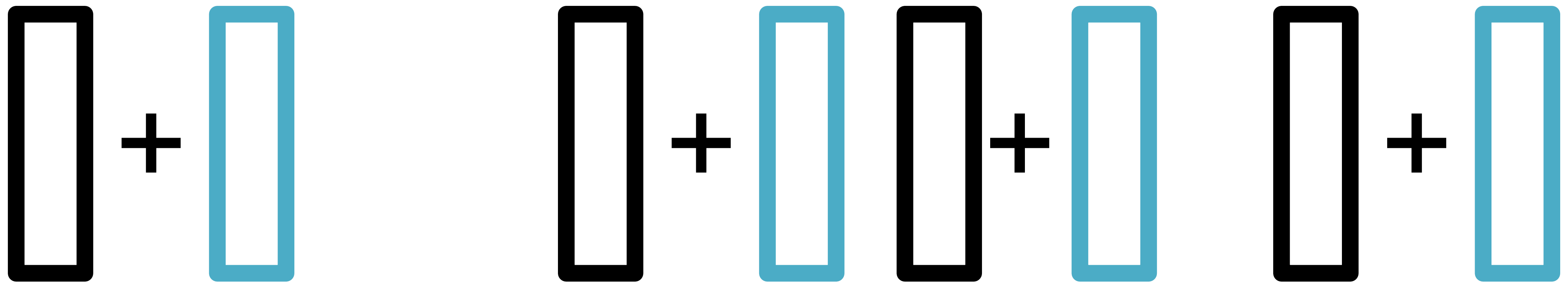


Order Matters — Positional Embedding

- The same word appearing at different position in a sentence may have different function/semantics
- The movie is great <—> movie is the great <—> great the is movie ?
- Map position labels to embedding

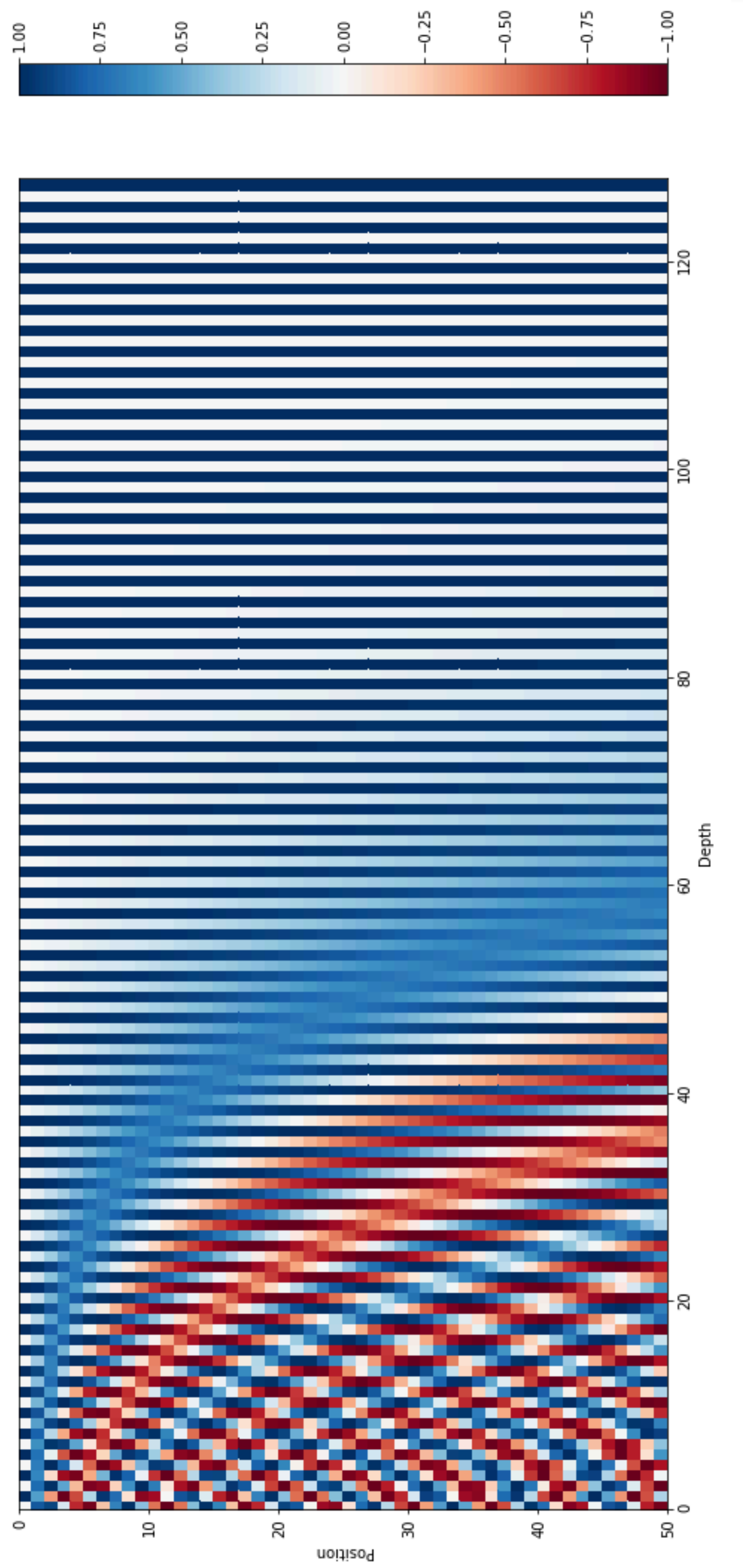
$$PE_{pos,2i} = \sin(\frac{pos}{1000^{2i/d}})$$

$$PE_{pos,2i+1} = \cos(\frac{pos}{1000^{2i/d}})$$



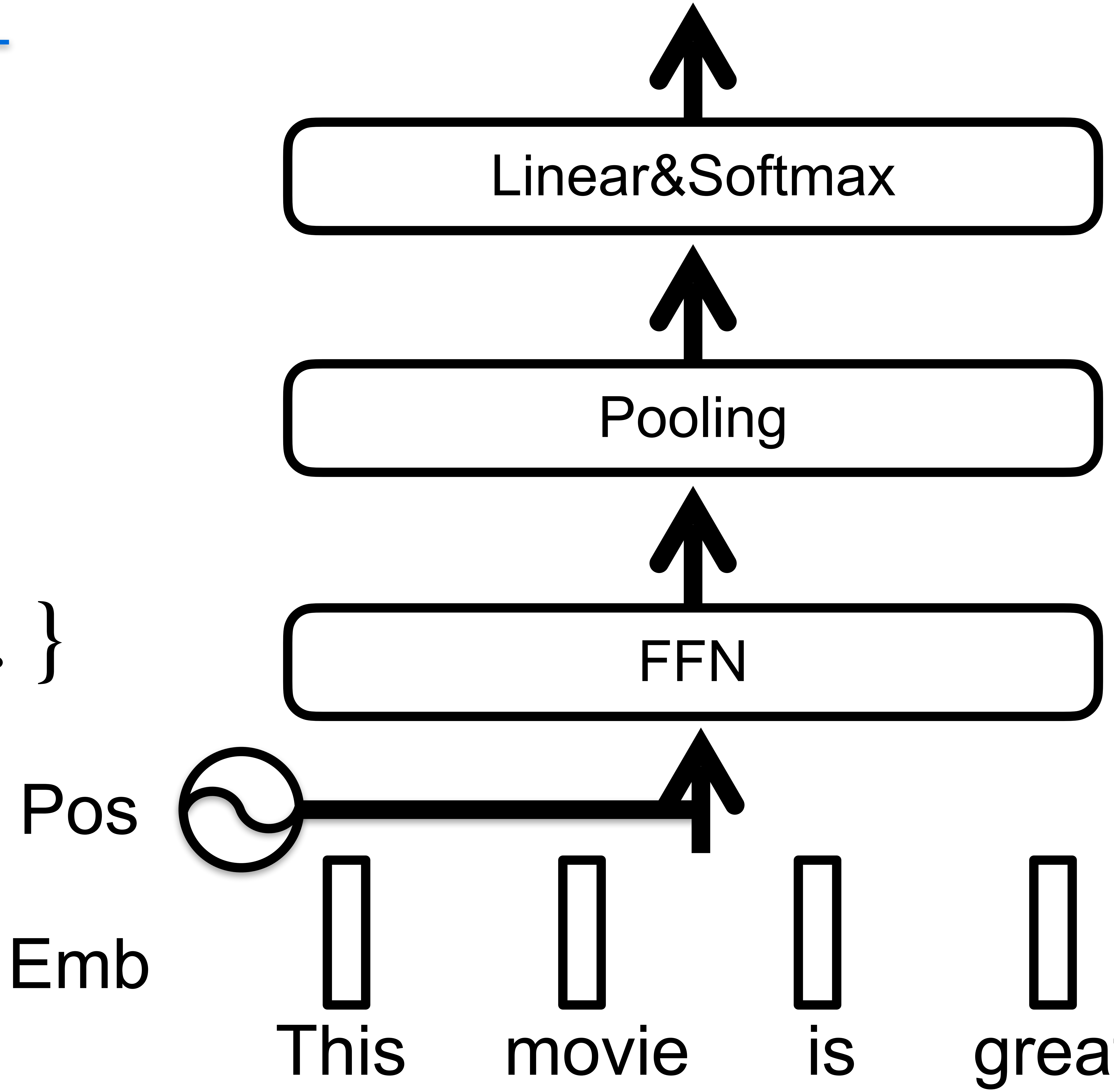
This movie is great

1 2 3 4



Full Model

- The whole network represents a function
  $f(x;\theta) : V^* \rightarrow \mathbb{R}^3$
- The parameter set
  $\theta = \{emb, w_1, w_2, \ldots\}$



Universal Approximation

- What is the representation power of NN?
- Theorem: Feedforward neural network with at least one hidden layer (with many units) can approximate any Borel measurable function to arbitrary accuracy. [Hornik et al 1989]
- But not without hidden layer!

Training a Model

- Given data $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$
- A function f as defined by a neural network (can be generalized to other model)
- Find the best parameter θ to fit the data
- How to define best fit?
  - Several principled approaches

Empirical Risk Minimization

- For a function $f(x;\theta)$, and a data distribution $(x, y) \sim P$
- Define (expected) risk function

$$R(\theta) = \int \ell(f(x;\theta), y) dP$$

$\ell(\hat{y}, y)$ is the loss function/distance defined on predicted and actual outcomes

- Empirical risk:

$$R_e(\theta) = \frac{1}{N} \sum_n \ell(f(x_n;\theta), y_n)$$

i.e. expected risk under empirical distribution that puts 1/N probability mass on each data sample

- Under ERM framework, $\hat{\theta} \leftarrow \operatorname*{argmin}_{\theta} R_e(\theta)$

Empirical Risk Minimization

- ERM provides a very generic way to define and find best-fit parameters
- $$R_e(\theta) = \frac{1}{N}\sum_n \ell(f(x_n;\theta), y_n)$$
- Many ways to define loss function $\ell(f, y)$
- Commonly used:
  - Cross-entropy for classification: $\ell(f, y) = -\sum_j y_j \log f_j$, y is one-hot vector
  - Square loss for regression: $\ell(f, y) = \frac{1}{2}|f - y|_2^2$

Cross Entropy (CE)

- Cross-entropy $H(p,q) = -\sum_k p_k \log q_k$
- Average number of bits needed to represent message in q, while the actual message is distributed in p
- OR. roughly the information gap between p and q + (some const)
- Minimizing cross-entropy == diminishing the information gap
- $H\left(y_i, f(x_i)\right) = -\sum_k y_{i,k} \log f(x_i)_k = -\log f(x_i)_{y_i}$
- Ideal case $f(x_i)_{y_i}$ ==> 1.0

$$f(x_n;\theta) = \begin{bmatrix} 0.2 \\ 0.3 \\ 0.5 \end{bmatrix} \qquad y_n = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

Minimizing cross-entropy

- The whole network represents a function

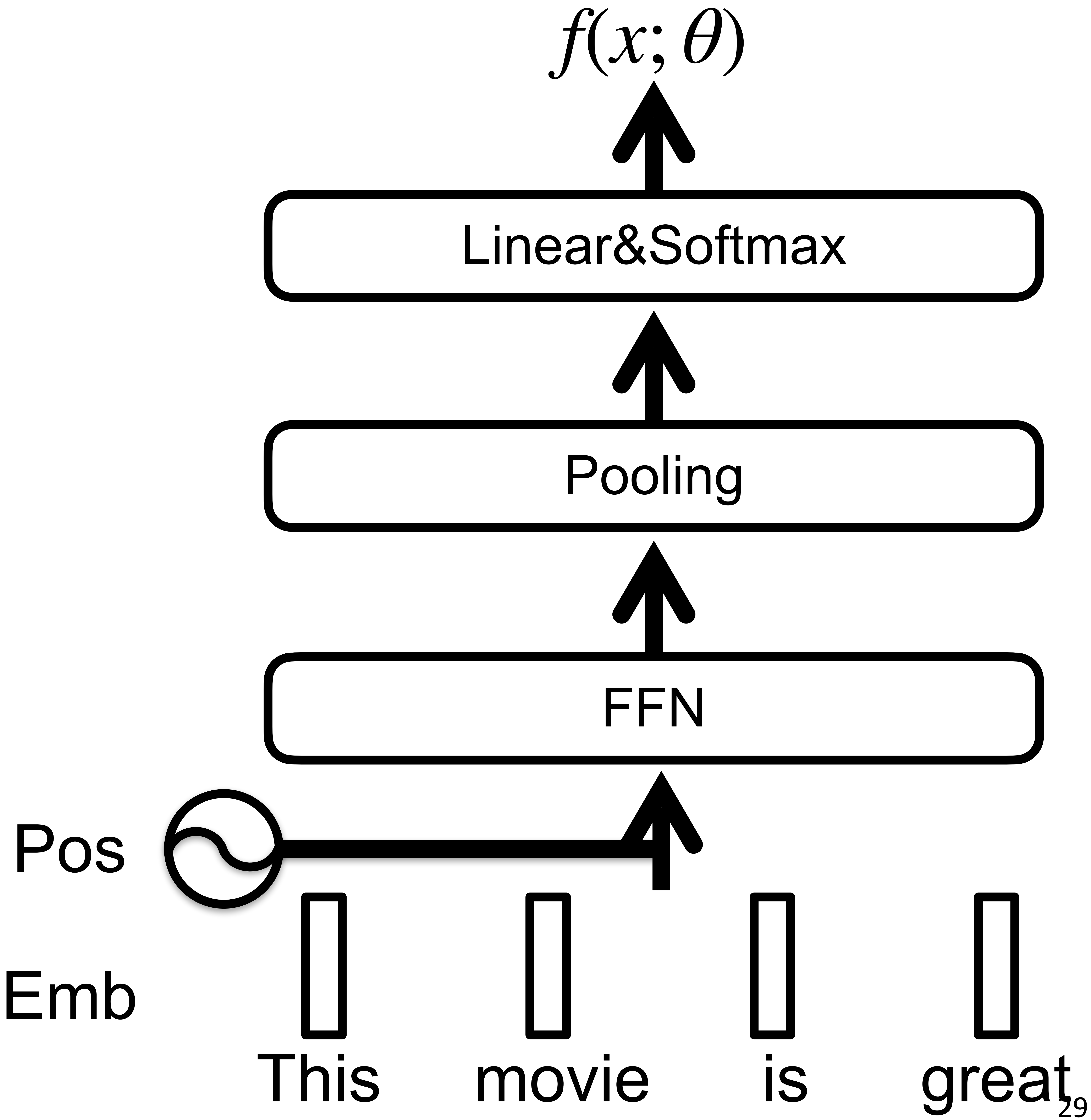
$$f(x;\theta) : V^* \rightarrow \mathbb{R}^3$$

- The parameter set

$$\theta = \{emb, w_1, w_2, \ldots\}$$

$$\theta \leftarrow \operatorname*{argmin}_{\theta} R_e(\theta)$$

$$= -\frac{1}{N}\sum_{n}\sum_{j} y_{n,j} \log f(x_n;\theta)$$



Alternatively: Maximum Likelihood Estimation

- Consider f as a conditional distribution of y given x
- Given $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$
- To find a θ that best describe data, i.e. θ defines a conditional distribution under which the data is most probable

$$\hat{\theta} \leftarrow \operatorname{argmax} \log L(\theta)$$

$$L(\theta) = \prod_{n} P(f(x_n; \theta) = y_n \mid x_n)$$

MLE Example

- For the simple neural model

$$\hat{\theta} \leftarrow \operatorname{argmax} \log L(\theta)$$

- $$L(\theta) = \prod_n P(f(x_n;\theta) = y_n \mid x_n) = \prod_n \prod_j f(x_n;\theta)_j^{y_{n,j}}$$

$$f(x_n;\theta) = \boxed{\begin{matrix} 0.2 \\ 0.3 \\ 0.5 \end{matrix}} \qquad y_n = \boxed{\begin{matrix} 0 \\ 1 \\ 0 \end{matrix}}$$

Risk minimization and MLE

- Discussion: Is minimizing cross-entropy equivalent to maximizing likelihood?
  - Under what condition?

Learning the Model

- Given a risk function, how to estimate the optimal parameter for a model?

$$\theta^* = \operatorname{argmin} \frac{1}{N} \sum_{n=1}^{N} \ell(f(x_n; \theta), y_n)$$

- Stochastic optimization algorithms
  - for large-scale data

Optimization

- Consider a generic function minimization problem

$$\min_x f(x) \text{ where } f : \mathbb{R}^d \rightarrow \mathbb{R}$$

- Optimal condition: $\nabla f|_x = 0$, where i-th element of $\nabla f|_x$ is $\dfrac{\partial f}{\partial x_i}$
- In general, no closed-form solution for the equation.
- Iterative update algorithm

$$x_{t+1} \leftarrow x_t + \Delta$$

- so that $f(x_{t+1}) \ll f(x_t)$
- How to find Δ

Taylor approximation

- $f(x+\Delta x) = f(x) + \Delta x^T \nabla f|_x + \frac{1}{2}\Delta x^T \nabla^2 f|_x \Delta x + \cdots$
- Theorem: if f is twice-differentiable and has continuous derivatives around x, for any small-enough Δx, there is $f(x+\Delta x) = f(x) + \Delta x^T \nabla f|_x + \frac{1}{2}\Delta x^T \nabla^2 f|_z \Delta x$, where $\nabla^2 f|_z$ is the Hessian at z which lies on the line connecting x and $x + \Delta x$
- First-order and second-order Taylor approximation result in gradient descent and Newton's method

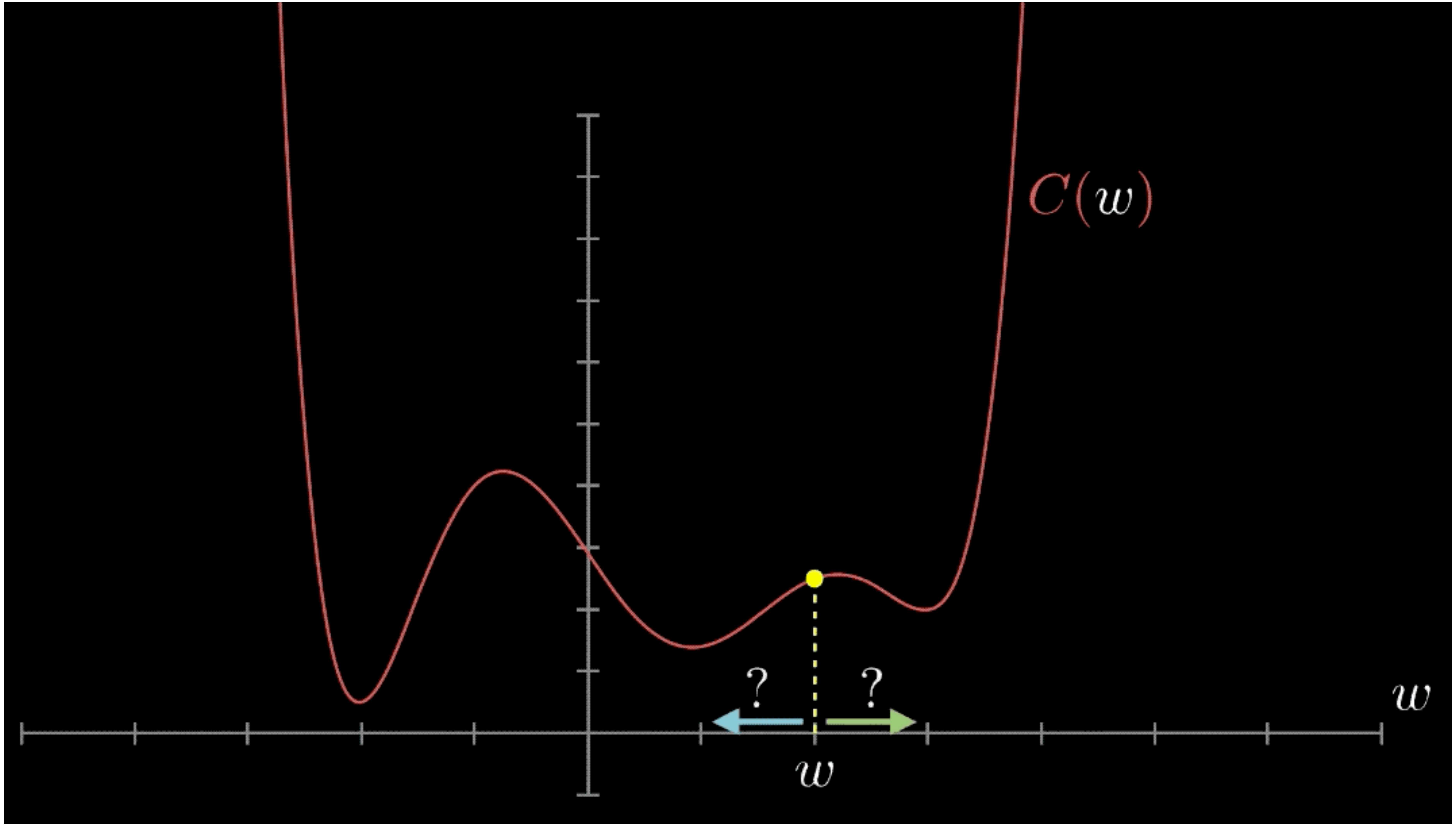
Gradient Descent

- $f(x_t + \Delta x) \approx f(x_t) + \Delta x^T \nabla f|_{x_t}$
- To make $\Delta x^T \nabla f|_{x_t}$ smallest
- $\Rightarrow \Delta x$ in the opposite direction of $\nabla f|_{x_t}$ i.e. $\Delta x = -\nabla f|_{x_t}$
- Update rule: $x_{t+1} = x_t - \eta \nabla f|_{x_t}$
- η is a hyper-parameter to control the learning rate

Stochastic Gradient Descent

- Gradient descent requires calculating over full data.
- $$\theta_{t+1} = \theta_t - \frac{\eta}{N}\sum_{n=1}^{N} \nabla_\theta \ell(f(x_n;\theta_t), y_n)$$
- Instead of full gradient, evaluate and update on random minibatch of data samples B_t
- $$\theta_{t+1} = \theta_t - \frac{\eta}{|B_t|}\sum_{n\in B_t} \nabla_\theta \ell(f(x_n;\theta_t), y_n)$$

SGD: Illustration



[credit: gif from 3blue1brown]

Newton's Method

- $f(x_t + \Delta x) \approx f(x_t) + \Delta x^T \nabla f|_{x_t} + \frac{1}{2}\Delta x^T \nabla^2 f|_{x_t} \Delta x$
- Let gradient $g_t = \nabla f|_{x_t}$, Hessian $H_t = \nabla^2 f|_{x_t}$
- Let $\frac{\partial f(x_t + \Delta x)}{\partial \Delta x} = 0$

$$x_{t+1} = x_t - \eta \cdot H_t^{-1} \cdot g_t$$

- updated on stochastic minibatch for large data

Convergence Rate versus Computation Cost

- Under some condition (Lipschitz continuous), GD converges with $O(\frac{1}{T})$, or $O(\frac{1}{\epsilon})$ to achieve error within ϵ
- SGD converges with $O(\frac{1}{\sqrt{T}})$
- Newton's method has better convergence, but higher per-iteration computation cost.

Computing Gradient for Neural Net

- Forward and back-propagation
- Suppose y=f(x), z=g(y), therefore z=g(f(x))
- Use the chain rule, $\nabla g(f(x))\,|_x = (\nabla f\,|_x)^T \cdot \nabla g\,|_y$
- For a neural net and its loss $\ell(\theta)$
- First compute gradient with respect to last layer
- then using chain-rule to back propagate to second last, and so on

Accelerate SGD

- $\theta_{t+1} = \theta_t - \eta \cdot g_t$, where g_t is the gradient
- Adaptive step-size η for each dimension of parameters
- Adaptive gradients
  - AdaGrad: $\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t}} \odot g_t$, where $v_t = \sum_{j=1}^{t} g_j^2$ accumulative second moments
  - Adam: $\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t}} \odot m_t$,

  where momentum $m_t = \beta \cdot m_{t-1} + (1-\beta) \cdot g_t$

$$v_t = \gamma v_{t-1} + (1-\gamma) g_t^2$$

Neural Network Framework

- Pytorch
- Tensorflow
- PaddlePaddle
- Define the computation graph of a model
  - Already provide a library of basic layers
  - along with automatic gradient calculation
  - with many loss functions

Simple Text Classification in Pytorch

```
from torch import nn

class TextClassificationModel(nn.Module):

    def __init__(self, vocab_size, embed_dim, num_class):
        super(TextClassificationModel, self).__init__()
        self.embedding = nn.EmbeddingBag(vocab_size, embed_dim, sparse=True)
        self.fc = nn.Linear(embed_dim, num_class)
        self.init_weights()

    def init_weights(self):
        initrange = 0.5
        self.embedding.weight.data.uniform_(-initrange, initrange)
        self.fc.weight.data.uniform_(-initrange, initrange)
        self.fc.bias.data.zero_()

    def forward(self, text, offsets):
        embedded = self.embedding(text, offsets)
        return self.fc(embedded)
```

https://github.com/pytorch/tutorials/blob/master/beginner_source/text_sentiment_ngrams_tutorial.py

Practical Trick

- Gradient clipping
  - avoid explode/overflow

```
torch.nn.utils.clip_grad_norm_(model.parameters(), 0.1)
```

Reading

- Chap 6 of DL book.