Deep Neural Networks for Natural Language Processing (AI6127)

JUNG-JAE KIM

WORD WINDOW CLASSIFICATION

Contents

- NLP application: Named entity recognition (NER)
  - Before deep neural network: HMM, CRF
- Word-window classification
  - Softmax classifier, Multilayer Perceptron: Gradient, Backpropagation

Named entity recognition (NER)

- The task: Find and classify names in text
- Possible applications
  - Tracking mentions of particular entities in documents
  - For question answering, answers are usually named entities
  - A lot of wanted information is associations between named entities
- Often followed by named entity linking/canonicalization into knowledge base

In 1917, Einstein applied the general theory of relativity to model the large-scale structure of the universe. He was visiting the United States when Adolf Hitler came to power in 1933 and did not go back to Germany, where he had been a professor at the Berlin Academy of Sciences. He settled in the U.S., becoming an American citizen in 1940. On the eve of World War II, he endorsed a letter to President Franklin D. Roosevelt alerting him to the potential development of "extremely powerful bombs of a new type" and recommending that the U.S. begin similar research. This eventually led to what would become the Manhattan Project. Einstein supported defending the Allied forces, but largely denounced using the new discovery of nuclear fission as a weapon. Later, with the British philosopher Bertrand Russell, Einstein signed the Russell-Einstein Manifesto, which highlighted the danger of nuclear weapons. Einstein was affiliated with the Institute for Advanced Study in Princeton, New Jersey, until his death in 1955.

Tag colours:

LOCATION TIME PERSON ORGANIZATION MONEY PERCENT DATE

http://www.europeana-newspapers.eu/named-entity-recognition-for-digitised-newspapers/

Source: CS224n

NER as Sequence labelling

CoNNL format

All data files contain one word per line with empty lines representing sentence boundaries. At the end of each line there is a tag which states whether the current word is inside a named entity or not. The tag also encodes the type of named entity. Here is an example sentence:

U.N.	NNP	I-NP	I-ORG
official	NN	I-NP	O
Ekeus	NNP	I-NP	I-PER
heads	VBZ	I-VP	O
for	IN	I-PP	O
Baghdad	NNP	I-NP	I-LOC
.	.	O	O

* **CoNLL**: Conference on Computational Natural Language Learning

BIO schema

John	B-PER
Smith	I-PER
lives	O
in	O
New	B-LOC
York	I-LOC

John Smith ⇒ PERSON
New York ⇒ LOCATION

John Snow LABS

https://towardsdatascience.com/named-entity-recognition-ner-with-bert-in-spark-nlp-874df20d1d77

Why might NER be hard?

- Hard to work out boundaries of entity


First National Bank Donates 2 Vans To Future School Of Fort Smith

POSTED 3:43 PM, JANUARY 11, 2019, BY 5NEWS WEB STAFF

  - Is the first entity "First National Bank" or "National Bank"

- Hard to know if something is an entity
  - Is there a school called "Future School" or is it a future school?

- Hard to know class of unknown/novel entity:
  - What class is "Zig Ziglar"? (A person.)

- Entity class is ambiguous and depends on context
  - "Charles Schwab" is PER not ORG here!


where Larry Ellison and Charles Schwab can live discreetly amongst wooded estates. And

Source: CS224n

Contents

- NLP application: Named entity recognition (NER)
  - Before deep neural network: HMM, CRF
- Word-window classification
  - Softmax classifier, Multilayer Perceptron: Gradient, Backpropagation

Sequence labelling/classification: Hidden Markov Model (HMM)

- Input: Sequence of observations
  - e.g. sentence – list of tokens (w_1, …, w_n)
- Output: What is the best sequence of tags that corresponds to the input sequence of observations? (t_1, …, t_n)
- Hidden Markov Model (HMM)
  - Consider all possible sequences of tags
  - Select the tag sequence with the highest probability
  - $\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}}\, p(t_1^n|w_1^n) \Rightarrow \hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}}\, \underbrace{p(w_1^n|t_1^n)}_{\text{likelihood}} \underbrace{p(t_1^n)}_{\text{prior}}$
    - by using Bayes rule

Source: NLP

HMM: Two kinds of probabilities

$$p(w_1^n|t_1^n)p(t_1^n) \approx \prod_{i=1}^{n} p(w_i|t_i)p(t_i|t_{i-1})$$

- Tag transition probability $p(t_i|t_{i-1})$
- Word likelihood probability $p(w_i|t_i)$
- Computed by counting in a labelled corpus

$$p(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})} \qquad p(w_i|t_i) = \frac{C(t_i, w_i)}{C(t_i)}$$

Source: NLP

HMM: Calculating estimated probability

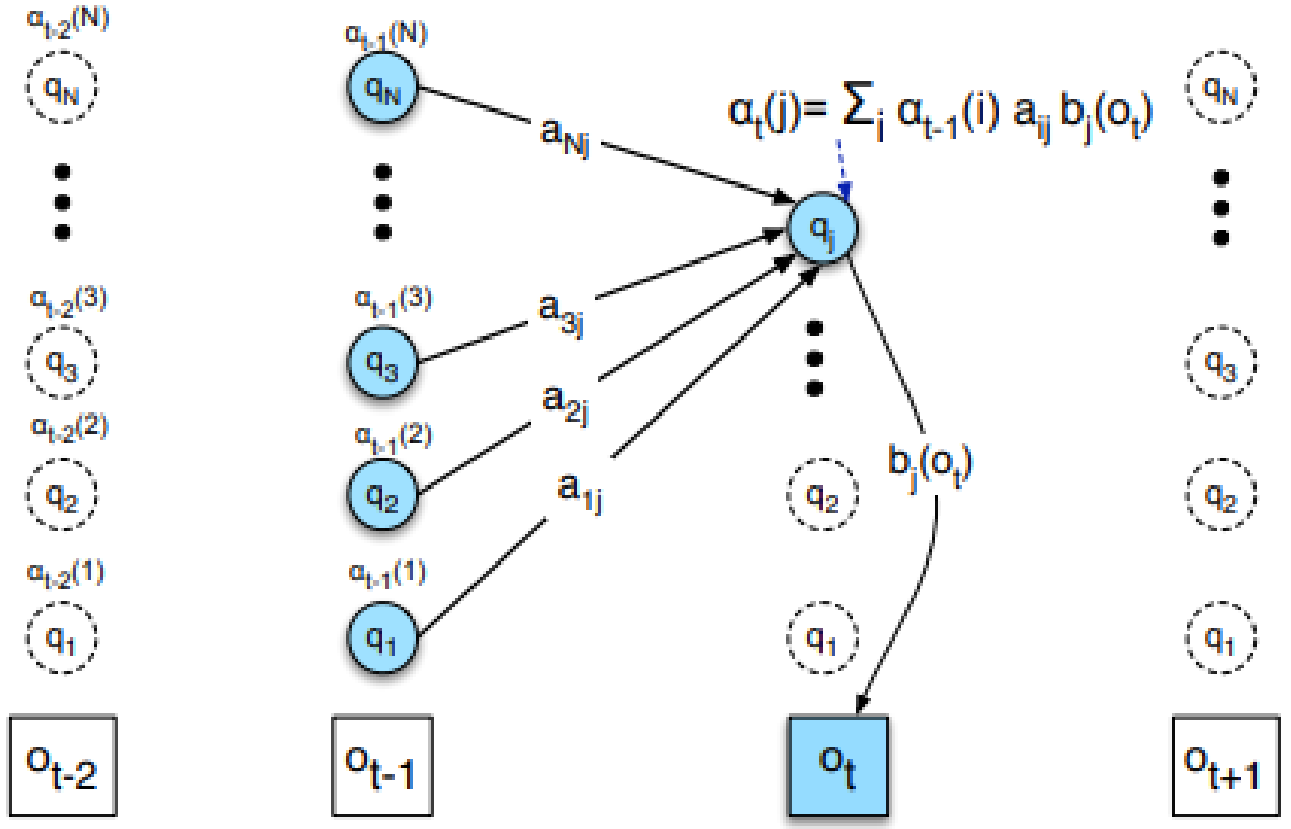
- Examples
  - John/B-PER Smith/I-PER lives/O in/O New/B-LOC York/I-LOC
  - John/B-PER Smith/I-PER lives/O in/O New/O York/B-LOC
- Probabilities
  - p(John|B-PER) x p(B-PER|START) x p(Smith|I-PER) x p(I-PER|B-PER) x p(lives|O) x p(O|I-PER) x p(in|O) x p(O|O) x p(New|B-LOC) x p(B-LOC|O) x p(York|I-LOC) x p(I-LOC|B-LOC)
  - p(John|B-PER) x p(B-PER|START) x p(Smith|I-PER) x p(I-PER|B-PER) x p(lives|O) x p(O|I-PER) x p(in|O) x p(O|O) x p(New|O) x p(O|O) x p(York|B-LOC) x p(B-LOC|O)

$$p(w_1^n|t_1^n)p(t_1^n) \approx \prod_{i=1}^{n} p(w_i|t_i)p(t_i|t_{i-1})$$

Source: NLP

HMM: Viterbi decoding

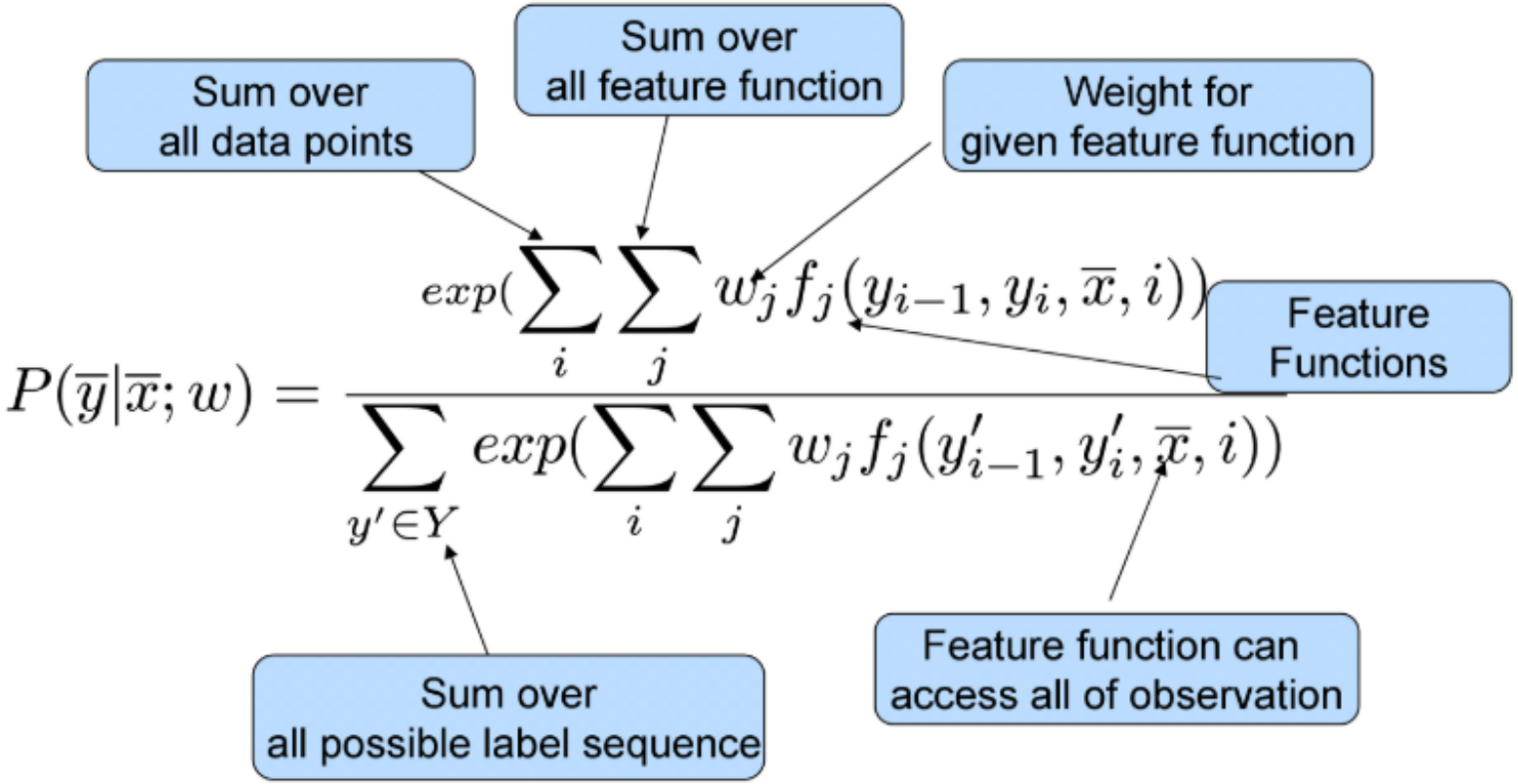
- Viterbi decoding: based on dynamic programming using table
  - See the link below for details



https://web.stanford.edu/~jurafsky/slp3/A.pdf

Source: NLP

Conditional Random Field (CRF)



$$P(\overline{y}|\overline{x};w) = \frac{exp(\sum_i \sum_j w_j f_j(y_{i-1}, y_i, \overline{x}, i))}{\sum_{y' \in Y} exp(\sum_i \sum_j w_j f_j(y'_{i-1}, y'_i, \overline{x}, i))}$$

- See the link below for details

http://www.cs.columbia.edu/~smaskey/CS6998/slides/statnlp_week10.pdf

HMM vs. CRF

- Both have efficient inference algorithms to find the best sequence

- **HMM**
- Maximize p(x|y)p(x)
  - Bayes rule
- Limited on types of features that can be used
  - E.g. two types of probabilities
- Per state normalization
  - Viterbi decoding

- **CRF**
- Maximize p(y|x)
- Allows much more set of features to be used
- Normalization over the whole sequence

http://www.cs.columbia.edu/~smaskey/CS6998/slides/statnlp_week10.pdf

Sequence labelling applications

- Part-of-speech (POS) tagging
  - E.g. Pierre/NNP Vinken/NNP ,/, 61/CD years/NNS old/JJ ,/, will/MD join/VB the/DT board/NN as/IN a/DT nonexecutive/JJ director/NN Nov./NNP 29/CD ./.
- Semantic role labelling (SRL)
  - E.g. [agent John] [predicate broke] [theme the window] [instrument with a rock]
  - Per predicate
- Word segmentation
  - E.g. 上海浦东开发与建设同步 → 上/B海/I浦/B东/I开/B发/I与/B建/B设/I同/B步/I

Hands-on: Named entity recognition by using CRF

- Download and preprocess NER corpus (CONLL 2002)
- Prepare CRF model for NER
- Run CRF for training and evaluation

https://drive.google.com/file/d/1Neuy76b_R6-OYevNpKAGumA8rJR7Udnj/view?usp=sharing

Contents

- NLP application: Named entity recognition (NER)
  - Before deep neural network: HMM, CRF
- Word-window classification
  - Softmax classifier, Multilayer Perceptron: Gradient, Backpropagation

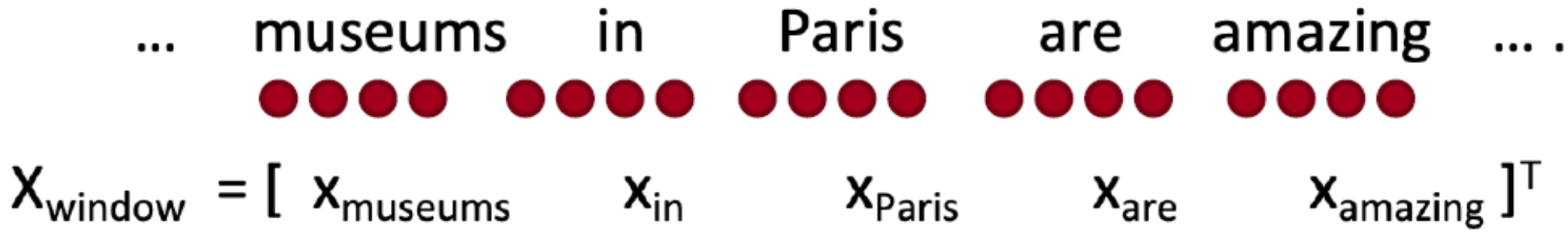
Word-Window classification

- Idea: classify a word in its context window of neighboring words
  - … [museums in Paris are amazing] … (classify 'Paris' with window length 2)
- For example, Named Entity Classification of a word in context:
  - Person, Location, Organization, None
- A simple way to classify a word in context might be to average the word vectors in a window and to classify the average vector
  - Problem: that would lose position information

Source: CS224n

Word-Window classification: Softmax

- Train a softmax classifier to classify a center word by taking concatenation of word vectors surrounding it in a window
- Example: Classify ‘Paris’ in the context of this sentence with window length 2:



- Resulting vector $x_{window} = x \in \mathbb{R}^{5d}$

Source: CS224n

Softmax classifier

$$p(y|x) = \frac{\exp(W_y \cdot x)}{\sum_{c=1}^{C} \exp(W_c \cdot x)}$$

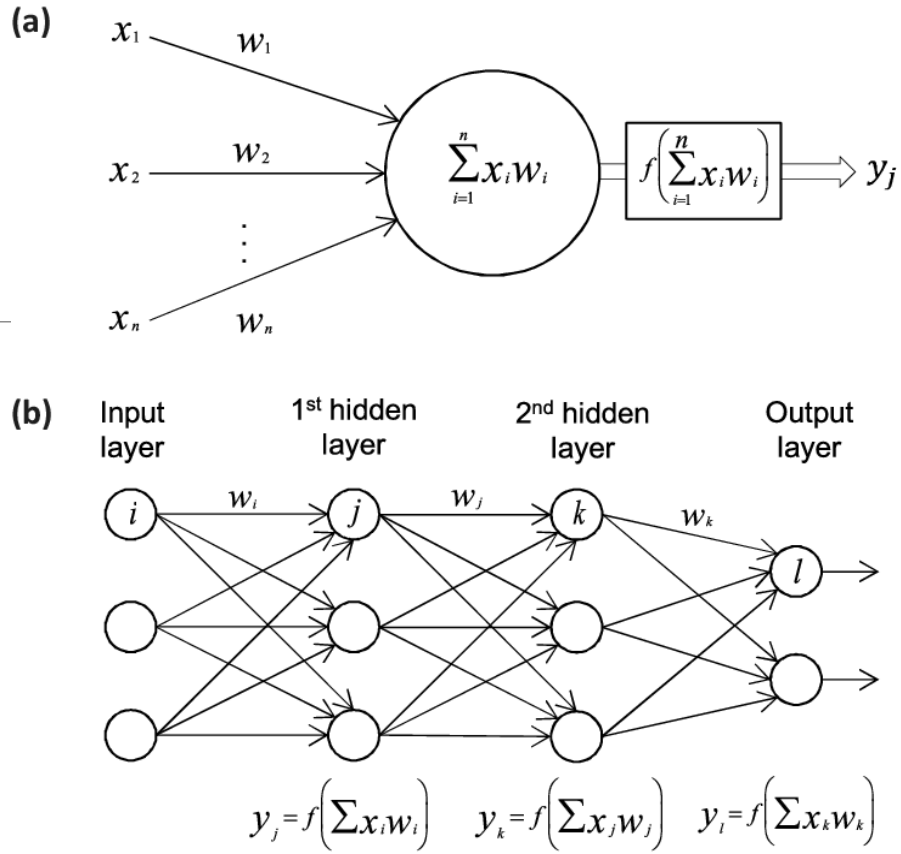
- We can tease apart the prediction function into two steps:

1. Take the y^{th} row of *W* and multiply that row with *x*:
   Compute all f_c for *c*=1,…,*C*

$$W_y \cdot x = \sum_{i=1}^{d} W_{y_i} x_i = f_y$$

2. Apply softmax function to get normalized probability:

$$p(y|x) = \frac{\exp(f_y)}{\sum_{c=1}^{C} \exp(f_c)} = \text{softmax}(f_y)$$



Source: CS224n

Training with softmax and cross-entropy loss

- For each training example (*x*,*y*), our objective is to maximize the probability of the correct class *y*
- This is equivalent to minimizing the negative log probability of that class:

$$-\log p(y|x) = -\log\left(\frac{\exp(f_y)}{\sum_{c=1}^{C}\exp(f_c)}\right)$$

- Using log probability converts our objective function to sums, which is easier to work with on paper and in implementation
- Cross entropy loss function over full dataset $\{x_i, y_i\}_{i=1}^{N}$

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} -\log\left(\frac{e^{f_{y_i}}}{\sum_{c=1}^{C} e^{f_c}}\right)$$

Source: CS224n

Background: What is "cross entropy" loss/error?

- Concept of "cross entropy" is from information theory
  - Let the true probability distribution be *p*
  - Let our computed model probability be *q*
- The cross entropy is: $$H(p,q) = -\sum_{c=1}^{C} p(c) \log q(c)$$
- Assuming a ground truth (or true or gold or target) probability distribution that is 1 at the right class and 0 everywhere else:
  - E.g. *p* = [0,…,0,1,0,…0]
- Because of one-hot *p*, the only term left is the negative log probability of the true class

Source: CS224n

Traditional machine learning optimization

- For general machine learning θ usually only consists of columns of W:

$$\theta = \begin{bmatrix} W_{1\cdot} \\ \vdots \\ W_{C\cdot} \end{bmatrix} = W \in \mathbb{R}^{Cd}$$

- So we only update the decision boundary via

$$\nabla J(\theta) = \begin{bmatrix} \nabla W_{1\cdot} \\ \vdots \\ \nabla W_{C\cdot} \end{bmatrix} \in \mathbb{R}^{Cd}$$

Source: CS224n

Classification difference with word vectors

- Commonly in NLP deep learning:
- We learn both W and word vectors x
- We learn both conventional parameters and (distributed) representations
- The word vectors re-represent one-hot vectors, conceptually via an embedding layer: *x = Le*

$$\nabla_\theta J(\theta) = \begin{bmatrix} \nabla_{W_{\cdot 1}} \\ \vdots \\ \nabla_{W_{\cdot d}} \\ \nabla_{x_{aardvark}} \\ \vdots \\ \nabla_{x_{zebra}} \end{bmatrix}$$

$$\in \mathbb{R}^{C_d + \boxed{V_d}}$$

→ Very large number of parameters

Source: CS224n

Hands-on: Word-window classification using SoftmaxClassifier

- Data preparation
- Build model
- Train model
- Predict with model

http://web.stanford.edu/class/cs224n/materials/ww_classifier.ipynb

Contents

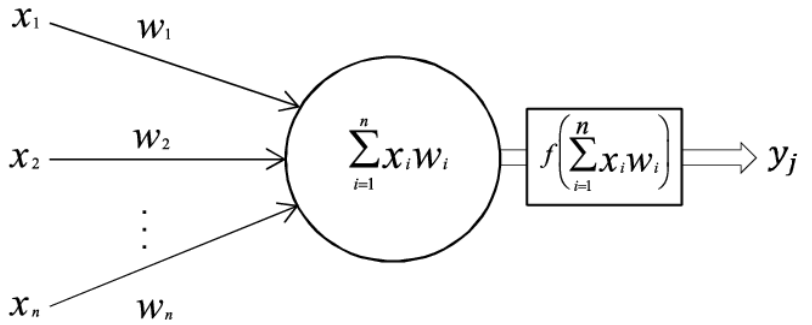
- NLP application: Named entity recognition (NER)
  - Before deep neural network: HMM, CRF
- Word-window classification
  - Softmax classifier, Multilayer Perceptron: Gradient, Backpropagation

Multilayer Perceptron (MLP)

- Introduce an additional layer in our softmax classifier with a non-linearity
- MLPs are fundamental building blocks of more complex neural systems!
- Assume we want to classify whether the center word is a Location
- Like word2vec, we will go over all positions in a corpus. But this time, it will be supervised s.t. positions that are true NER Locations should assign high probability to that class, and others should assign low probability.

Source: CS224n

Neural Network Feed-forward Computation

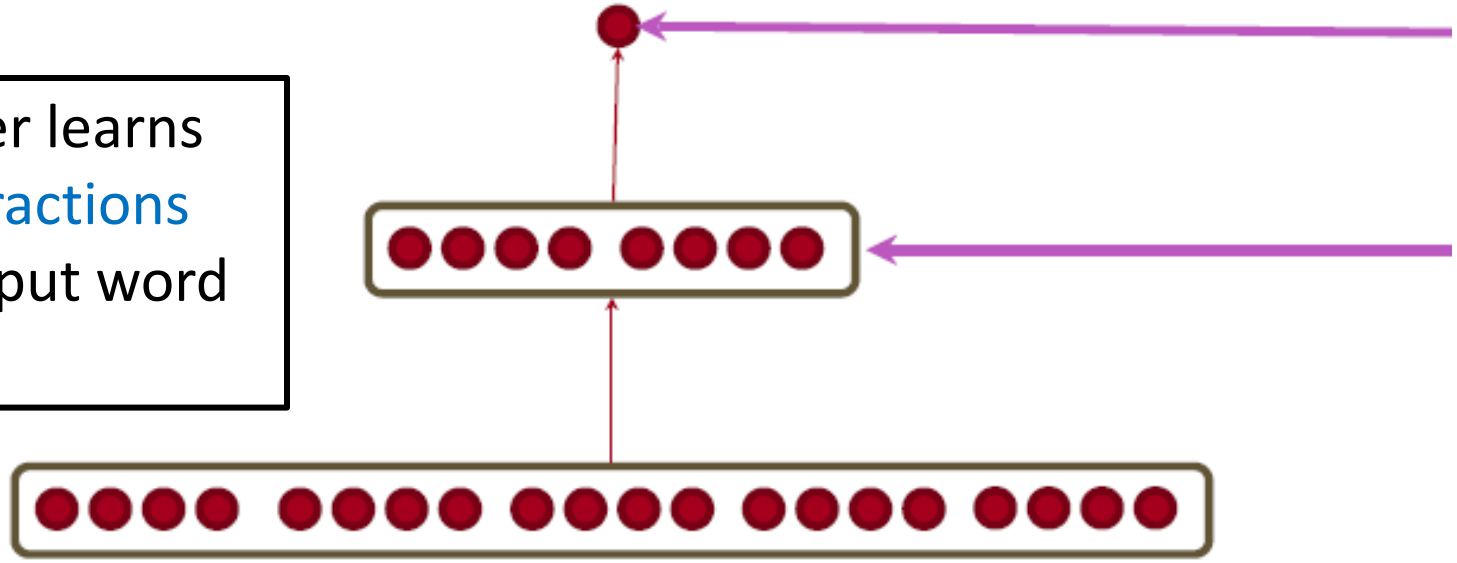


$$score(x) = U^T a \in \mathbb{R}$$

- We compute a window's score with a 3-layer neural net:
  - *s* = *score*("museums in Paris are amazing")

$$s = U^T f(Wx + b) \quad x \in \mathbb{R}^{20\times 1}, W \in \mathbb{R}^{8\times 20}, U \in \mathbb{R}^{8\times 2}$$

The middle layer learns non-linear interactions between the input word vectors.



Source: CS224n

Recall

Word2vec: Stochastic Gradient Descent

- Problem: $J(\theta)$ is a function of all windows in the corpus (potentially billions!)
  - So $\nabla_\theta J(\theta)$ is very expensive to compute
- You would wait a very long time before making a single update!
- Solution: Stochastic gradient descent (SGD)
  - Repeatedly sample windows, and update after each one
- Algorithm:
  - alpha: step size or learning rate

```
while True:
    window = sample_window(corpus)
    theta_grad = evaluate_gradient(J,window,theta)
    theta = theta - alpha * theta_grad
```


Source: CS224n

Stochastic Gradient Descent

- Update equation $\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$
- How can we compute $\nabla_\theta J(\theta)$?
  - Matrix calculus for computing gradients
  - Backpropagation algorithm

Source: CS224n

Contents

- NLP application: Named entity recognition (NER)
  - Before deep neural network: HMM, CRF
- Word-window classification
  - Softmax classifier, Multilayer Perceptron: Gradient, Backpropagation

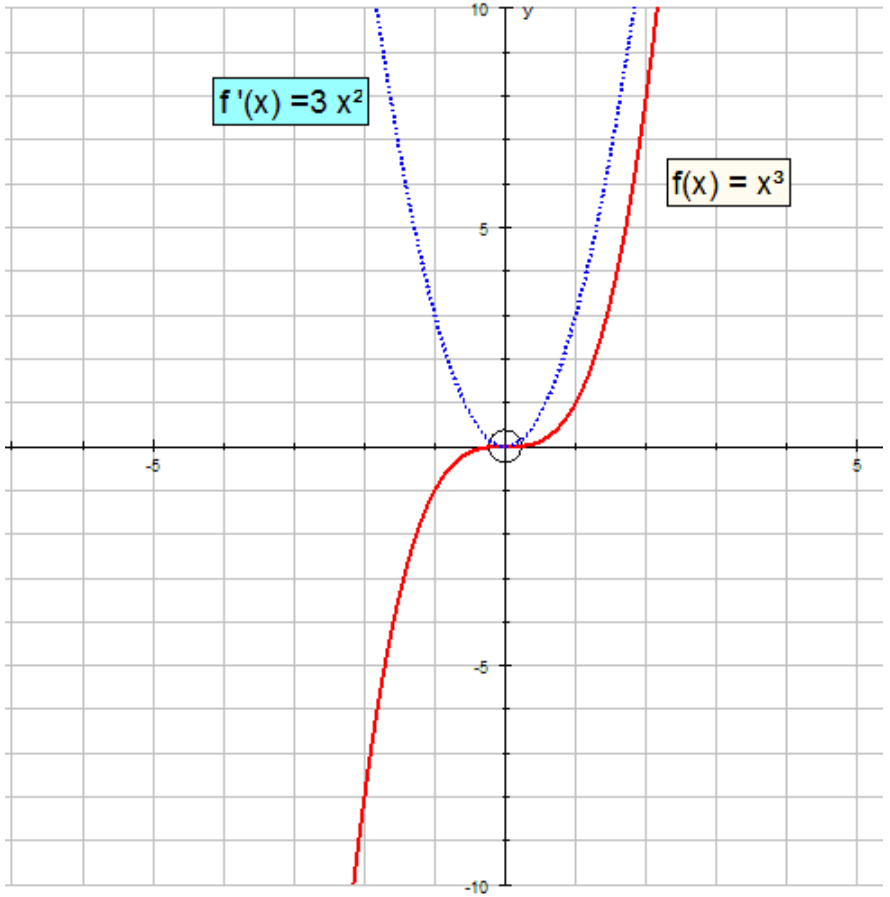
Gradient example

- Given a function with 1 output and 1 input

$$f(x) = x^3$$

- Its gradient (slope): How much will the output change if we change the input a bit?

$$\frac{df}{dx} = 3x^2$$



Source: CS224n

Gradient example

- Given a function with 1 output and n inputs

$$f(\boldsymbol{x}) = f(x_1, x_2, \cdots x_n)$$

- Its gradient is a vector of partial derivatives with respect to each input

$$\frac{\partial f}{\partial \boldsymbol{x}} = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots, \frac{\partial f}{\partial x_n}\right]$$

Source: CS224n

Jacobian Matrix: Generalization of the Gradient

- Given a function with m outputs and n inputs

$$\boldsymbol{f}(\boldsymbol{x}) = [f_1(x_1, x_2, \cdots\ x_n), \cdots, f_m(x_1, x_2, \cdots\ x_n)]$$

- Its gradient is an m x n matrix of partial derivatives

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}, \cdots, \frac{\partial f_1}{\partial x_n} \\ \vdots \quad \ddots \quad \vdots \\ \frac{\partial f_m}{\partial x_1}, \cdots, \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

$$\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right)_{ij} = \frac{\partial f_i}{\partial x_j}$$

Source: CS224n

Chain Rule

- For one-variable functions: **multiply derivatives**

$$z = 3y \qquad y = x^2 \qquad \frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} = (3)(2x) = 6x$$

- For multiple variables at once: **multiply Jacobians**

$$\boldsymbol{h} = f(\boldsymbol{z}) \qquad \boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} \qquad \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = \cdots$$

Source: CS224n

Elementwise activation Function

$$\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} = \begin{pmatrix} f'(z_1) & & 0 \\ & \ddots & \\ 0 & & f'(z_n) \end{pmatrix} = \text{diag}(\boldsymbol{f}'(\boldsymbol{z}))$$

$$\frac{\partial}{\partial \boldsymbol{x}}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) = \boldsymbol{W}$$

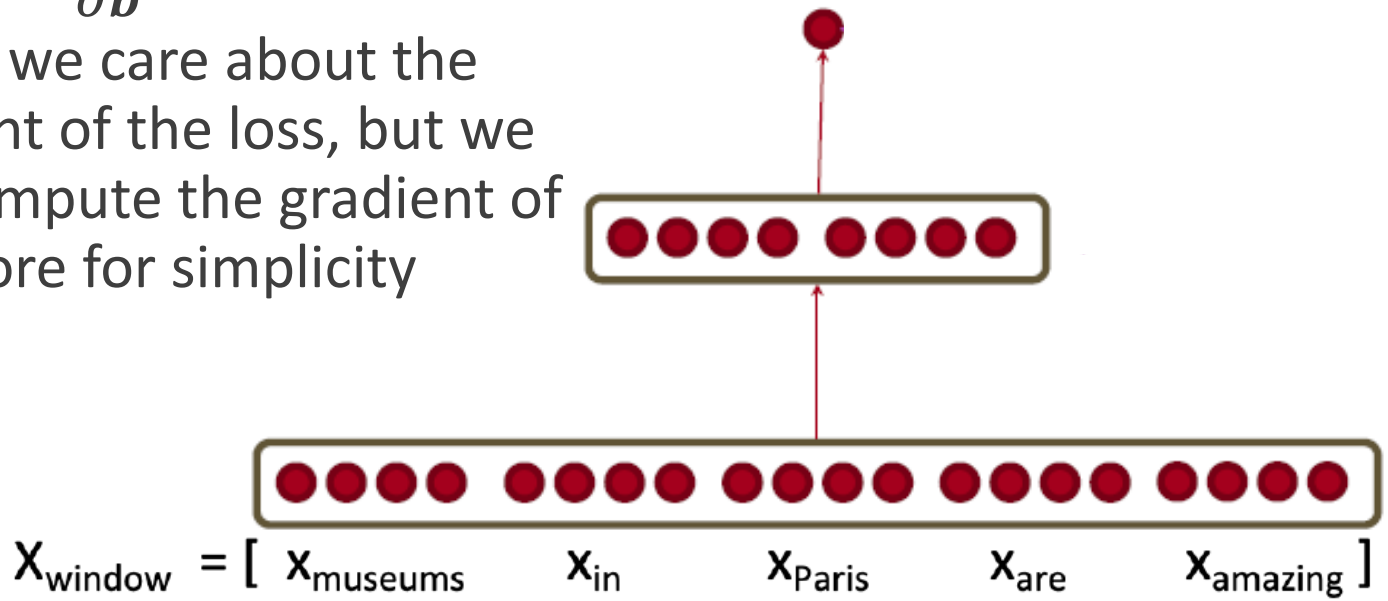
$$\frac{\partial}{\partial \boldsymbol{b}}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) = \boldsymbol{I}$$ (Identity matrix)

$$\frac{\partial}{\partial \boldsymbol{u}}(\boldsymbol{u}^T\boldsymbol{h}) = \boldsymbol{h}^T$$

Source: CS224n

Back to our Neural Net!

- Let's find $\frac{\partial s}{\partial \boldsymbol{b}}$
  - Really, we care about the gradient of the loss, but we will compute the gradient of the score for simplicity



$$s = \boldsymbol{u}^T \boldsymbol{h}$$

$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$$

$\boldsymbol{x}$ (input)

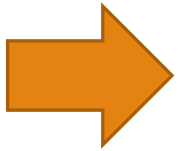
Source: CS224n

1. Break up equations into simple pieces

$s = \boldsymbol{u}^T \boldsymbol{h}$

$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$

$\boldsymbol{x}$ (input)



$s = \boldsymbol{u}^T \boldsymbol{h}$

$\boldsymbol{h} = f(\boldsymbol{z})$

$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$

$\boldsymbol{x}$ (input)

Source: CS224n

2. Apply the chain rule

$$s = \boldsymbol{u}^T \boldsymbol{h}$$

$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$$

$\boldsymbol{x}$ (input)

$$s = \boldsymbol{u}^T \boldsymbol{h}$$

$$\boldsymbol{h} = f(\boldsymbol{z})$$

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

$\boldsymbol{x}$ (input)

$$\frac{\partial s}{\partial \boldsymbol{b}} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}$$

$$\boldsymbol{u}^T \quad \text{diag}(f'(\boldsymbol{z})) \quad \boldsymbol{I}$$

$$= \boldsymbol{u}^T \circ f'(\boldsymbol{z})$$

Hadamard Product
(Element-wise Multiplication)

Source: CS224n

How to compute $\frac{\partial s}{\partial \boldsymbol{W}}$?

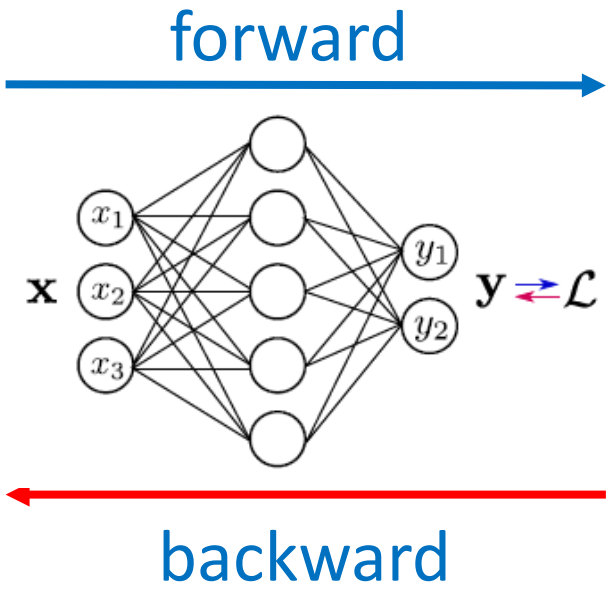
- See the link below for details

Contents

- NLP application: Named entity recognition (NER)
  - Before deep neural network: HMM, CRF
- Word-window classification
  - Softmax classifier, Multilayer Perceptron: Gradient, Backpropagation

Forward & backward propagation

- Forward propagation
  - Neural network accepts an input **x** and produces an output **y**
  - During training, it continues onward until it produces a scalar cost **L**
- Backward propagation
  - Information (**gradients** with respect to the parameters) from the cost flows backward through the network
  - It takes derivatives and uses the (generalized, multivariate, or matrix) chain rule
  - We re-use derivatives computed for higher layers in computing derivatives for lower layers to minimize computation



Source: NTUNLP

Computation graphs

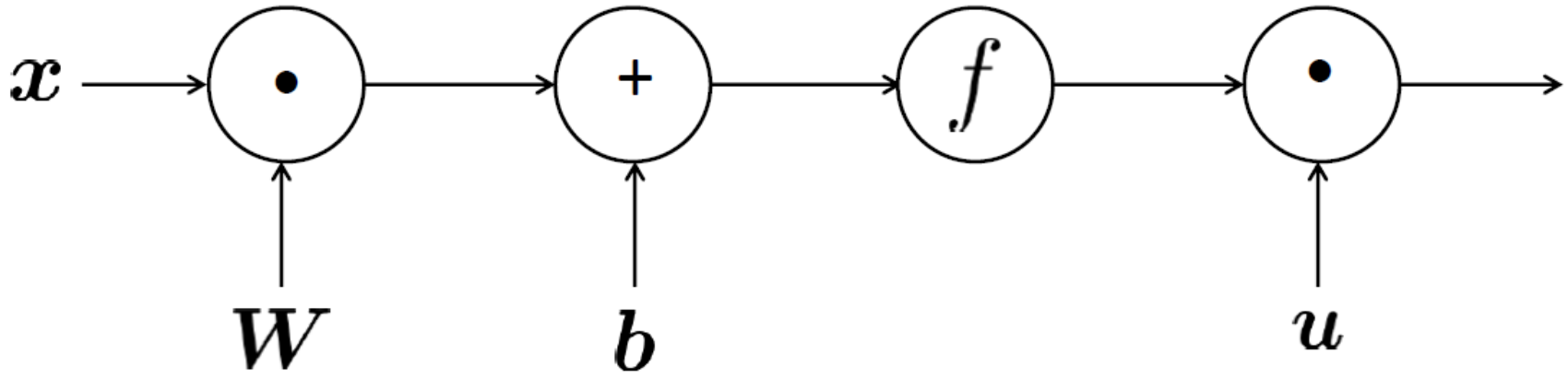
- We represent our neural network equations as a graph
  - Source nodes: inputs
  - Interior nodes: operations

$$s = \boldsymbol{u}^T \boldsymbol{h}$$

$$\boldsymbol{h} = f(\boldsymbol{z})$$

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

$\boldsymbol{x}$ (input)



Source: CS224n

Computation graphs: Forward propagation

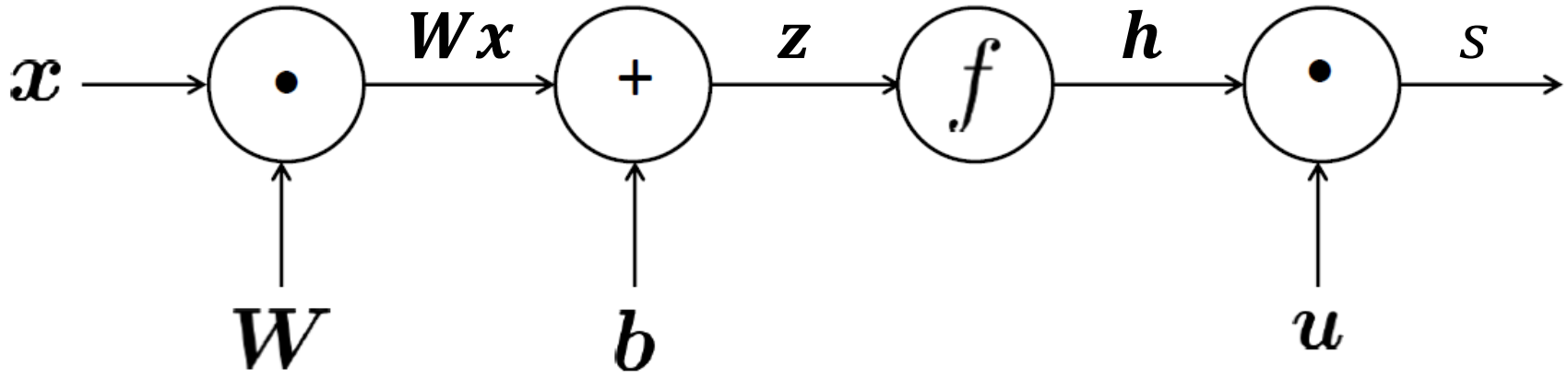
- We represent our neural network equations as a graph
  - Source nodes: inputs
  - Interior nodes: operations
  - Edges pass along result of the operation

$$s = \boldsymbol{u}^T \boldsymbol{h}$$

$$\boldsymbol{h} = f(\boldsymbol{z})$$

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

$\boldsymbol{x}$ (input)



Source: CS224n

Computation graphs: Backpropagation

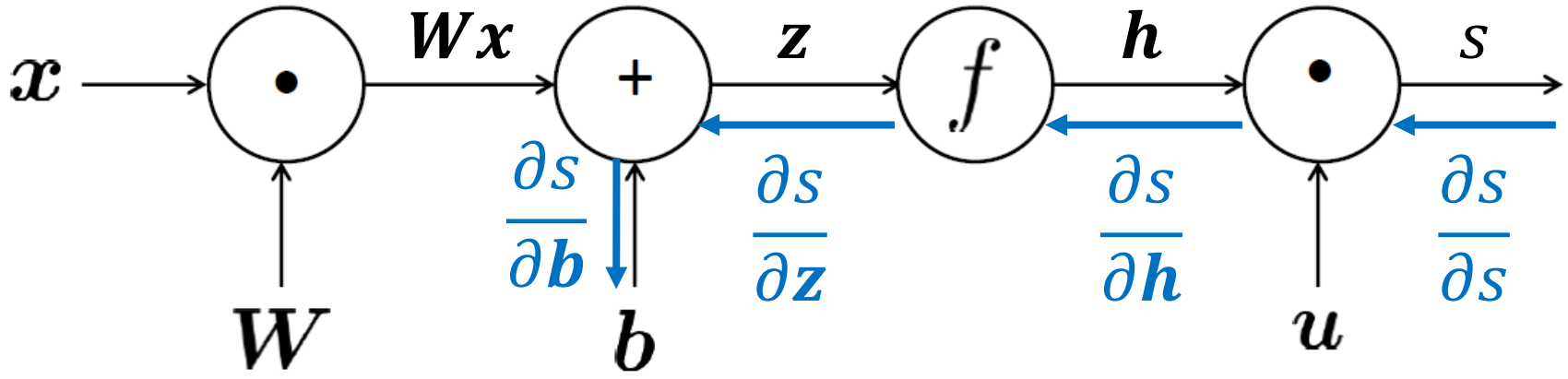
- Go backwards along edges
  - Pass along gradients
- See the link below for details

$$s = \boldsymbol{u}^T \boldsymbol{h}$$

$$\boldsymbol{h} = f(\boldsymbol{z})$$

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

$\boldsymbol{x}$ (input)



Automatic Differentiation

- The gradient computation can be automatically inferred from the symbolic expression of the forward propagation
- Each node type (e.g. +, *, max) needs to know how to compute its output and how to compute the gradient wrt its inputs given the gradient wrt its output
- Modern DL frameworks (e.g. tensorflow, PyTorch) do backpropagation for you but mainly leave layer/node writer to hand-calculate the local derivative

Source: CS224n

Why learn details about gradients?

- Modern deep learning frameworks compute gradients for you!
- But backpropagation doesn't always work perfectly
  - Understanding why is crucial for debugging and improving models
    - See Karpathy article linked below
  - Example in future lecture: exploding and vanishing gradients

https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b

Source: CS224n