Foundations of NLP

CS3126

Lecture-5

Probabilistic (N-grams) and Neural Language models

Mahindra™ University

Global Thinkers. Engaged Leaders.

Recap

- NLP
- Applications
- Regular expressions
- Tokenization
- Stemming
    - Porter Stemmer
- Lemmatization
- Normalization
- Stopwords
- Bag-of-Words
- TF-IDF
- NER
- POS tagging
- Semantics, Distributional semantics, Word2vec

Minor-I, Do's and Don'ts

- How to prepare for exam?
  - Bad strategy
    - preparing last minute
    - Searching for shortcuts
    - What will come in exam, DMing ("Mam will porter stemmer will be given"?)
  - Good strategy
    - Make use of Office Hours for Doubts
    - Active class participation
    - Use #helpdoubt channel for clarifications.
    - Last minute panic would not help, plan your obstacles well (submission format, links, etc.)!
    - Keep a window for doubts clarifications
    - Have to talk to others, have to check the communication platform (Slack/email)
    - Don't work in isolation, closing eyes, learn about any updates from Slack!

This will help you maintain reasonable grades throughout (irrespective of difficulty level)

Project Groups (First evaluation)

1. Prepare 7-8 slides and upload on the link (to be shared on slack)
    - Motivation
    - Problem Statement
    - Proposed pipeline
    - Timeline
    - Expected Outcome/Application

2. Deadline- 28th September 2024

Language Model

- **Classic definition-** Probability distribution over sequence of tokens
- Vocabulary V – a set of tokens!

- A language model **p** assigns each sequence of tokens $(x_1, \ldots\ldots\ldots x_L) \in V$, a probability (a number between 0 and 1),

$$P\,(x_1,\, x_2,\, x_3,\, \ldots\ldots\ldots\ldots,\, x_L)$$

- The probability intuitively tells us how "good" a sequence of tokens is.

Language Model

- Consider the probability of four strings in English.

- Here Vocabulary,

$$V = \{ate,\ ball,\ cheese,\ mouse,\ the\}$$

- $p(the,\ mouse,\ ate,\ the,\ cheese)=0.02$ ------> P_1
- $p(the,\ cheese,\ ate,\ the,\ mouse)=0.01$ -------> P_2
- $p(mouse\ ,the,\ the,\ cheese,\ ate)=0.0001$ -----> P_3

Clearly, $P1 > P2 > P3$

Language Model

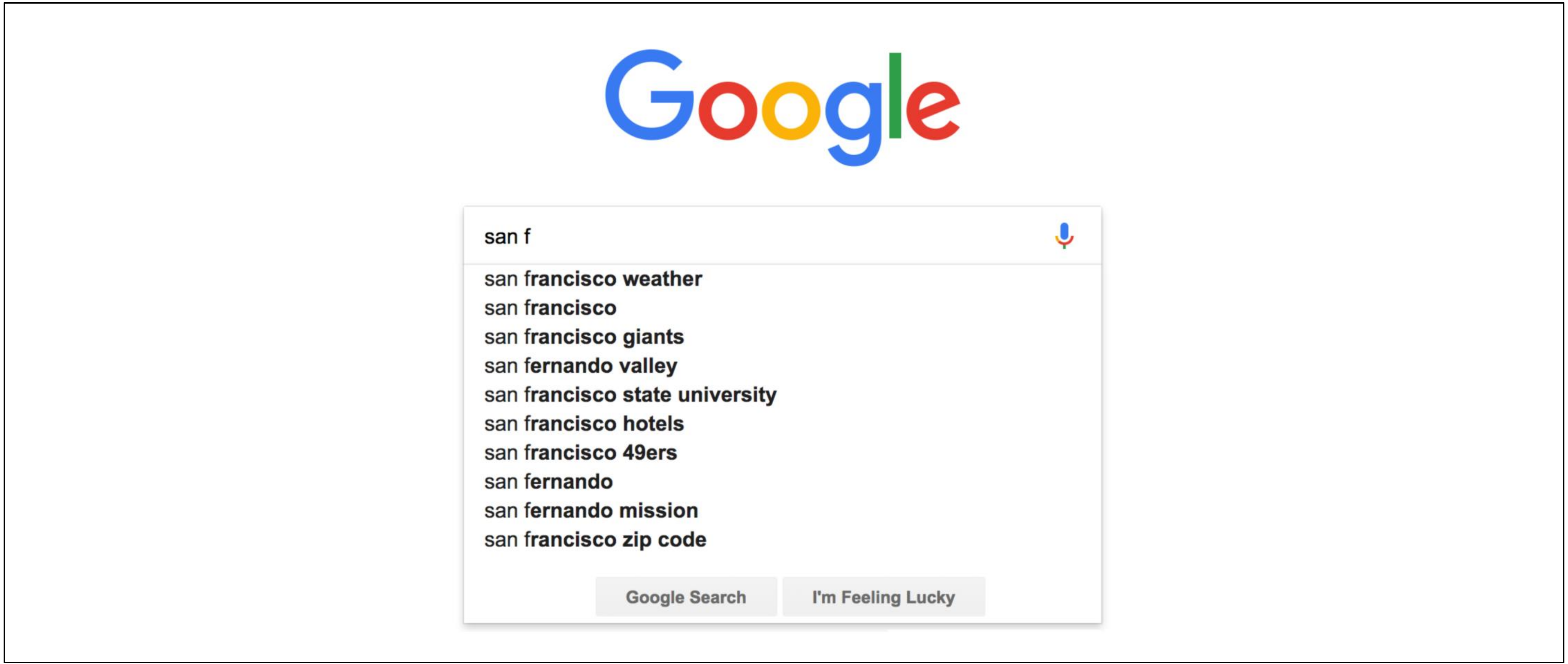
- LM assigned, "*mouse the the cheese ate*" a very low probability implicitly because it's ungrammatical (syntactic knowledge).

- The LM should assign ***the mouse ate the cheese*** higher probability than ***the cheese ate the mouse*** implicitly because of,

  - world knowledge: both sentences are the same syntactically,

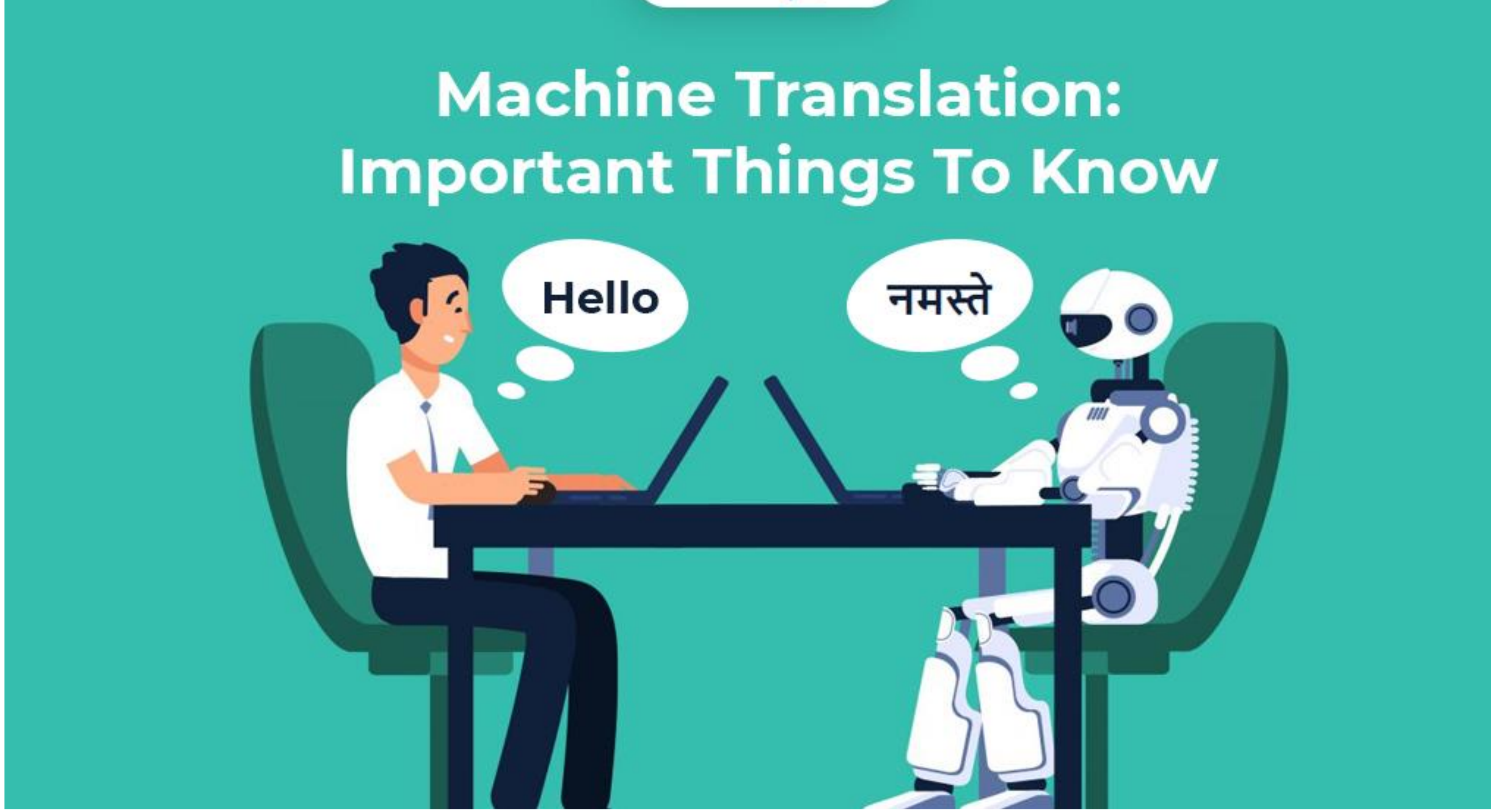
  - they differ in semantic plausibility

Where do you see language models? Google Search system



Spell correction

- The office is about fifteen minuets from my house
  - P(about fifteen minutes from) > P(about fifteen minuets from)



P(high winds tonite) > P(large winds tonite)

Speech recognition

Here, *P(I saw a van) >> P(eyes awe of an)*

Types of Language Models

- Probabilistic language models (PLMs)
- Neural language models (NLMs)

Probabilistic Language Modeling

- **Goal**: compute the probability of a *sentence* or *sequence of words*:

$$P(W) = P(w_1,w_2,w_3,w_4,w_5,\ldots\ldots w_n)$$

- Related task: probability of an upcoming word:

$$P(w_5 | w_1,w_2,w_3,w_4)$$

- A model that computes either of these:

  $P(W)$ or $P(w_n | w_1,w_2\ldots\ldots w_{n-1})$ is called a language model.

Credits: Slide adapted from [2]

How to compute P(W)

- How to compute this joint probability:
  - $P(its,\ water,\ is,\ so,\ transparent,\ that)$
- Intuition: let's rely on the Chain Rule of Probability

Credits: Slide adapted from [2]

Reminder: The Chain Rule

- Recall the definition of conditional probabilities

$$P(B \mid A) = P(A,B) \,/\, P(A)$$

Rewriting: $P(A,B) = P(A)P(B \mid A)$

- More variables:

$$P(A,B,C,D) = P(A)P(B \mid A)P(C \mid A,B)P(D \mid A,B,C)$$

- The Chain Rule in General

$$P(x_1,x_2,x_3,\ldots\ldots\ldots\ldots,x_n) = P(x_1)P(x_2 \mid x_1)P(x_3 \mid x_1,x_2)\ldots P(x_n \mid x_1,\ldots,x_{n-1})$$

Credits: Slide adapted from [2]

Chain Rule

To Compute joint probability of words in a sentence

"its water is so transparent"

$$P(w_1 w_2 \ldots w_n) = \prod_i P(w_i \mid w_1 w_2 \ldots w_{i-1})$$

P("its water is so transparent") =

P(its) × P(water|its) × P(is|its water)

× P(so|its water is) × P(transparent|its water is so)

Credits: Slide adapted from [2]

How to estimate these probabilities

Could we just count and divide?

$$P(\textit{the} \mid \textit{its water is so transparent that}) = \frac{Count(\textit{its water is so transparent that the})}{Count(\textit{its water is so transparent that})}$$

Why Not?

No!!!! Too many possible sentences!

We will never see enough data to estimate these

Credits: Slide adapted from [2]

Markov Assumption :

- Simplifying assumption:

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{that})$$

- Or maybe

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{transparent that})$$

Credits: Slide adapted from [2]

Markov Assumption

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i \mid w_{i-k} \ldots w_{i-1})$$

- In other words, we approximate each component in the product

$$P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-k} \ldots w_{i-1})$$

Credits: Slide adapted from [2]

Simplest case: Unigram model

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i)$$

Some automatically generated sentences from a unigram model

```
fifth, an, of, futures, the, an, incorporated, a,
a, the, inflation, most, dollars, quarter, in, is,
mass

thrift, did, eighty, said, hard, 'm, july, bullish

that, or, limited, the
```

Credits: Slide adapted from [2]

Bigram model

- Condition on the previous word:

$$P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-1})$$

```
texaco, rose, one, in, this, issue, is, pursuing, growth, in,
a, boiler, house, said, mr., gurria, mexico, 's, motion,
control, proposal, without, permission, from, five, hundred,
fifty, five, yen

outside, new, car, parking, lot, of, the, agreement, reached

this, would, be, a, record, november
```

Credits: Slide adapted from [2]

Estimating bigram probabilities

The Maximum Likelihood Estimate

$$P(w_i \mid w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

Credits: Slide adapted from [2]

An example

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

<s> I am Sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and ham </s>

Credits: Slide adapted from [2]

Solution?

Solution

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

<s> I am Sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and ham </s>

$$P(\texttt{I} \mid \texttt{<s>}) = \tfrac{2}{3} = .67 \qquad P(\texttt{Sam} \mid \texttt{<s>}) = \tfrac{1}{3} = .33 \qquad P(\texttt{am} \mid \texttt{I}) = \tfrac{2}{3} = .67$$

$$P(\texttt{</s>} \mid \texttt{Sam}) = \tfrac{1}{2} = 0.5 \qquad P(\texttt{Sam} \mid \texttt{am}) = \tfrac{1}{2} = .5 \qquad P(\texttt{do} \mid \texttt{I}) = \tfrac{1}{3} = .33$$

Credits: Slide adapted from [2]

Raw bigram counts

- Total- 9333 sentences

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Normalize by unigrams:

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

Result:

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Credits: Slide adapted from [2]

Bigram estimates of sentence probabilities

$$
\begin{aligned}
&P(\langle s\rangle \text{ I want english food } \langle /s\rangle) = \\
&\quad P(\text{I} \mid \langle s\rangle) \\
&\quad \times\ P(\text{want} \mid \text{I}) \\
&\quad \times\ P(\text{english} \mid \text{want}) \\
&\quad \times\ P(\text{food} \mid \text{english}) \\
&\quad \times\ P(\langle /s\rangle \mid \text{food}) \\
&\quad = .000031
\end{aligned}
$$

Credits: Slide adapted from [2]

What kinds of knowledge?

- P(english|want) = .0011
- P(chinese|want) = .0065
- P(to|want) = .66
- P(eat | to) = .28
- P(food | to) = 0
- P(want | spend) = 0
- P (i | <s>) = .25

Practical Issues

- We do everything in log space
  - Avoid underflow
  - (also adding is faster than multiplying)

$$\log(p_1 \times p_2 \times p_3 \times p_4) = \log p_1 + \log p_2 + \log p_3 + \log p_4$$

Summary- Language Modeling Task

Language Modeling is the task of predicting what word comes next

the students opened their ______

books

minds

laptop
s

exams

- More formally: given a sequence of words , compute the probability distribution of the next word
- A system that does this is called a Language Model

Credits: Slide adapted from [2]

Summary: N-gram Language Models

the students opened their ______

- Question: How to learn a Language Model?
- Answer (pre- Deep Learning): learn an n-gram Language Model!
    - Definition: An n-gram is a chunk of n consecutive words.
    - ***unigrams***: "the", "students", "opened", "their"
    - ***bigrams***: "the students", "students opened", "opened their"
    - ***trigrams***: "the students opened", "students opened their"
    - ***four-grams:*** "the students opened their"
- Idea: *Collect statistics about how frequent different n-grams are and use these to predict next word.*

Credits: Slide adapted from [2]

N-gram models

- Extend to trigrams, 4-grams, 5-grams
- In general, this is an insufficient model of language

  because language has long-distance dependencies:

"The computer which I had just put into the machine room on the fifth floor crashed."

- But we can often get away with N-gram models

Smoothing in N-grams

- Like many statistical models, the N-gram probabilistic language model is dependent on the training corpus.
- One practical issue with this is that some word sequences and phrases appear in practice (or in the test set), may not also occur in the training set.
- SPARSITY AND STORAGE PROBLEMS
- Important to train robust models that generalize well to handle the unseen words and zero probabilities

Credits: Slide adapted from [2]

Add one Smoothing

- Also called Laplace smoothing
- Pretend we saw each word one more time than we did
- Just add one to all the counts!

MLE estimate:

$$P_{MLE}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

Add-1 estimate:

$$P_{Add-1}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$$

Credits: Slide adapted from [2]

Recall the Language modeling task

Input: sequence of words, $x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}, \ldots\ldots\ldots\ldots x^{(t)}$

Output: probability distribution of the next word: $P(x^{(t+1)} | x^{(t)}, \ldots\ldots\ldots x^{(1)})$

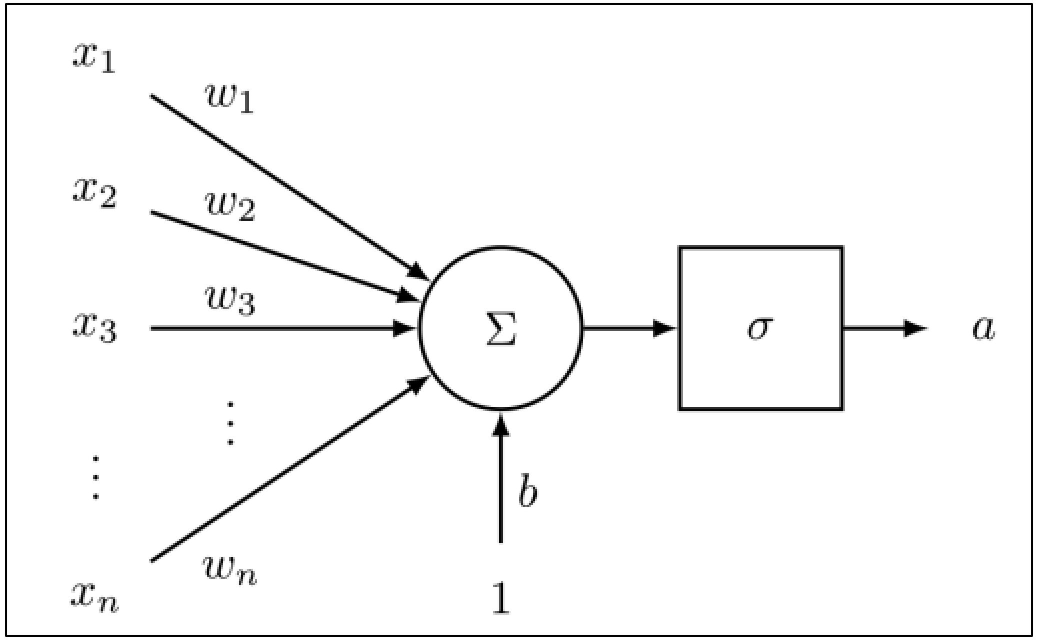
How about a window-based neural model?

Before building Neural language models

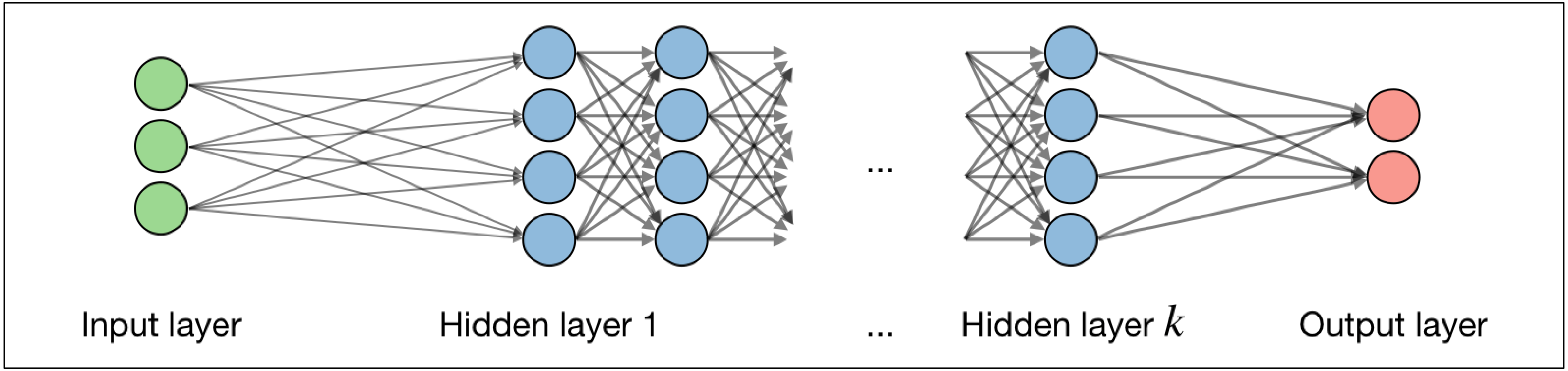
Let's first look into basics of Neural Network.........

Neurons

A neuron is the fundamental building block of neural networks



Neural Networks/Feed- Forward Neural Network



$$z_j^{[i]} = {w_j^{[i]}}^T x + b_j^{[i]}$$

where we denote, *w, b, z* as the weight, bias and output respectively.

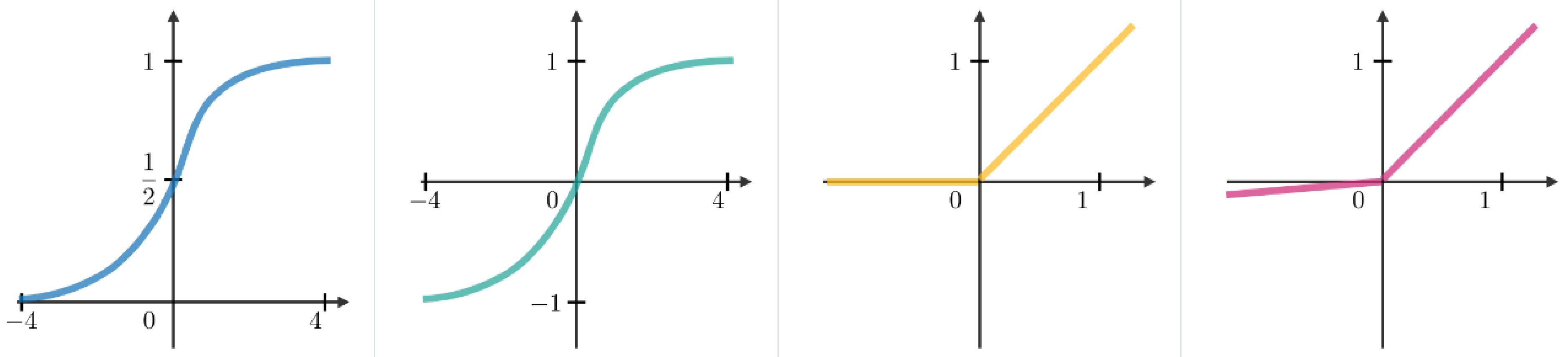
https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-deep-learning#Neural%20Networks

Activation functions

Activation functions are used at the end of a hidden unit to introduce non-linear complexities to the model.

Here are the most common ones:

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



Training Neural Networks

In a neural network, weights are updated as follows:

Step 1: Take a batch of training data.

Step 2: Perform forward propagation to obtain the corresponding loss.

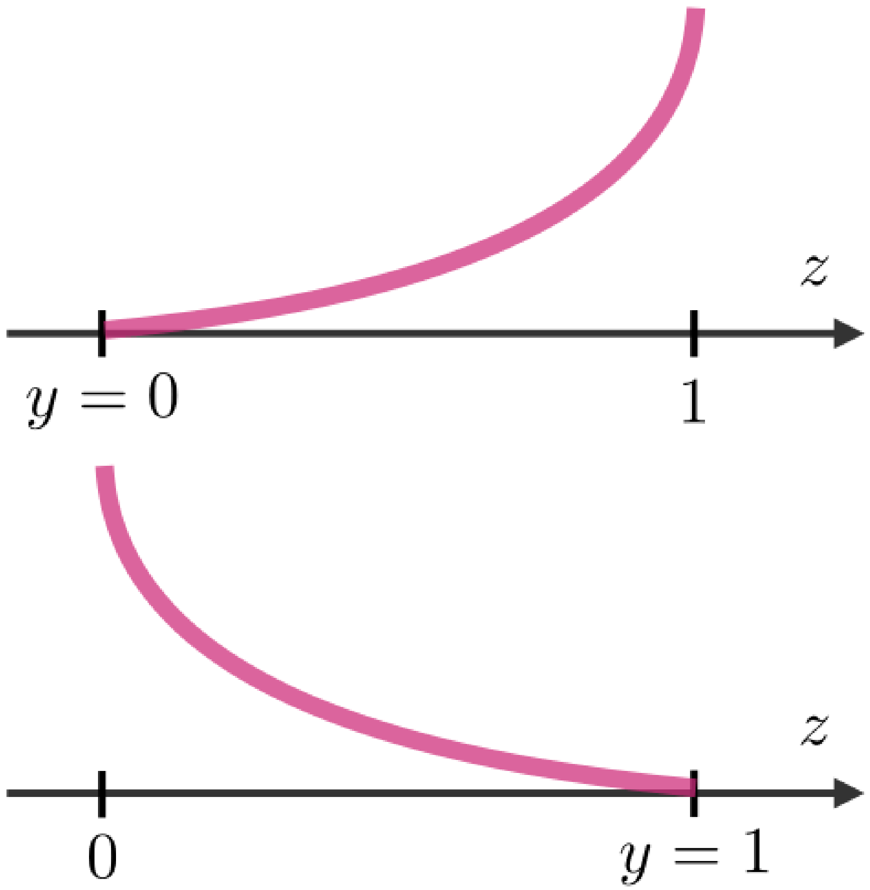
Step 3: Backpropagate the loss to get the gradients.

Step 4: Use the gradients to update the weights of the network.

https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-deep-learning#Neural%20Networks

Neural Network Loss function (Cross-entropy loss)

$$-\left[y\log(z) + (1-y)\log(1-z)\right]$$



Learning rate

- **Learning rate**: α or sometimes η, indicates at which pace the weights get updated.

- This can be fixed or adaptively changed.

- The current most popular method is called Adam, which is a method that adapts the learning rate.

https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-deep-learning#Neural%20Networks

Backpropagation

- Backpropagation is a method to update the weights in the neural network by taking into account the actual output and the desired output.
- The derivative with respect to weight w is computed using **chain rule** and is of the following form:

$$\boxed{\frac{\partial L(z,y)}{\partial w} = \frac{\partial L(z,y)}{\partial a} \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial w}}$$

- Weight is updated as follows:

$$\boxed{w \longleftarrow w - \alpha \frac{\partial L(z,y)}{\partial w}}$$

Backpropagation Visualization

Chain Rule!!!!

https://docs.google.com/presentation/d/1pmstGvQCoIwIDP9fJkVLDlG4BNRIfJu8cJTDcokUjrM/edit#slide=id.g27be483e10_0_0

Output of Neural Networks

- *For example: Classification task/Text classification*

- *Neural networks* are capable of producing raw output scores for each of the classes.

- How do we convert output scores into probabilities?

Interpreting logits: Sigmoid

Sigmoid function: σ: $\mathrm{R} \rightarrow [0,1]$

$$\sigma(z) = e^{z} / 1+e^{z} = 1/(1+e^{-z})$$

- **Class A** (also called the positive class)
- **Not Class A** (complement of Class A or also called the negative class)

https://web.stanford.edu/~nanbhas/blog/sigmoid-softmax/#binary-classification

Interpreting logits: SoftMax

Presenting the **softmax** function $S : \mathbf{R}^C \to [0,1]^C$

$$S(\mathbf{z})_i = \frac{e^{\mathbf{z}_i}}{\sum_{j=1}^{C} e^{\mathbf{z}_j}} = \frac{e^{\mathbf{z}_i}}{e^{\mathbf{z}_1}+\ldots+e^{\mathbf{z}_j}+\ldots+e^{\mathbf{z}_C}}$$

This function takes a vector of real-values and converts each of them into corresponding probabilities. In a C-class classification where $k \in \{1,2,\ldots,C\}$, it naturally lends the interpretation

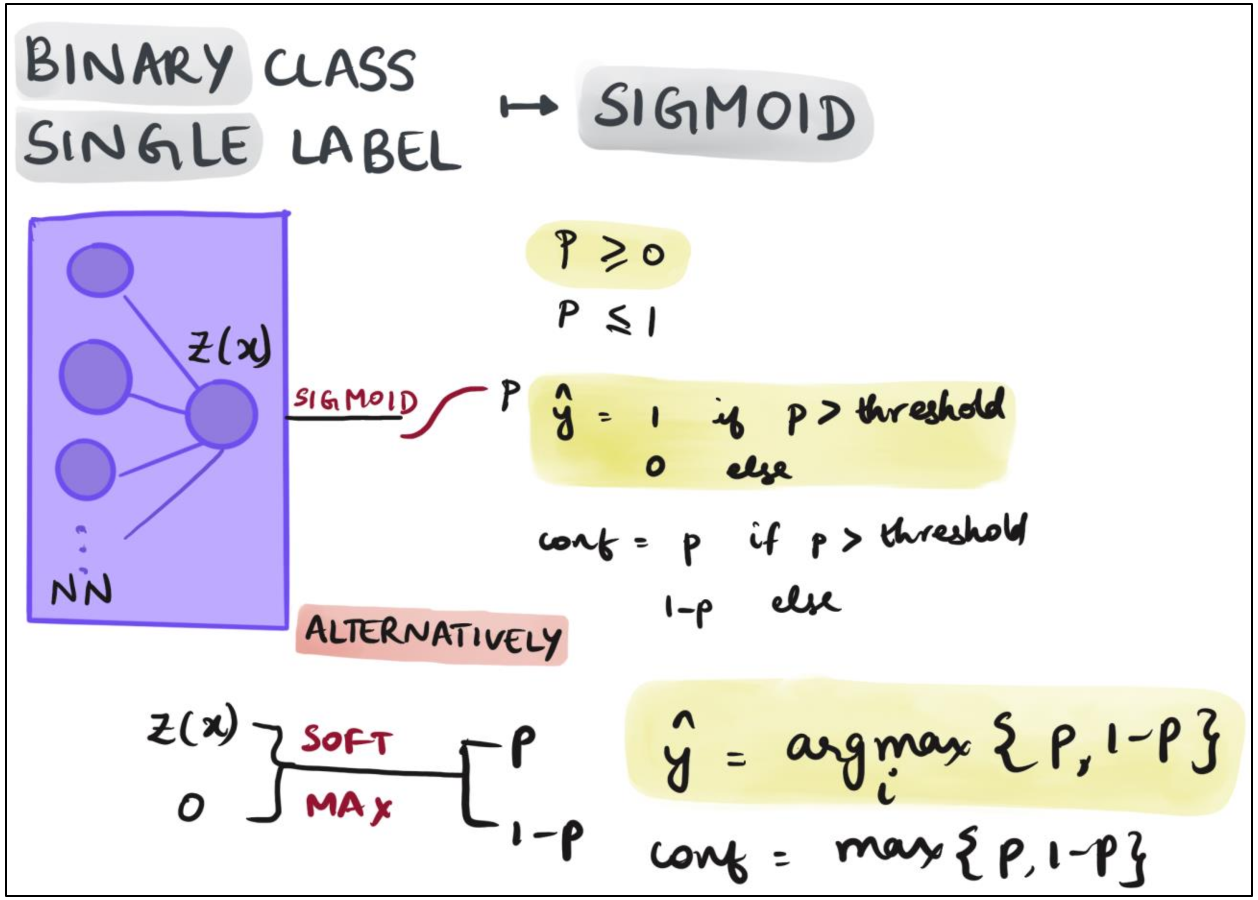
$$\mathbf{prob}(y = k|\mathbf{x}) = \frac{e^{\mathbf{z}(\mathbf{x})_k}}{\sum_{j=1}^{C} e^{\mathbf{z}(\mathbf{x})_j}}$$

https://web.stanford.edu/~nanbhas/blog/sigmoid-softmax/#binary-classification

SoftMax Classifier Implementation

https://docs.google.com/presentation/d/1dFbY-Buku18xhHHbFVNPNuc6vsPeR-LbE8KQXzvuzZg/edit#slide=id.g27be483e10_0_0

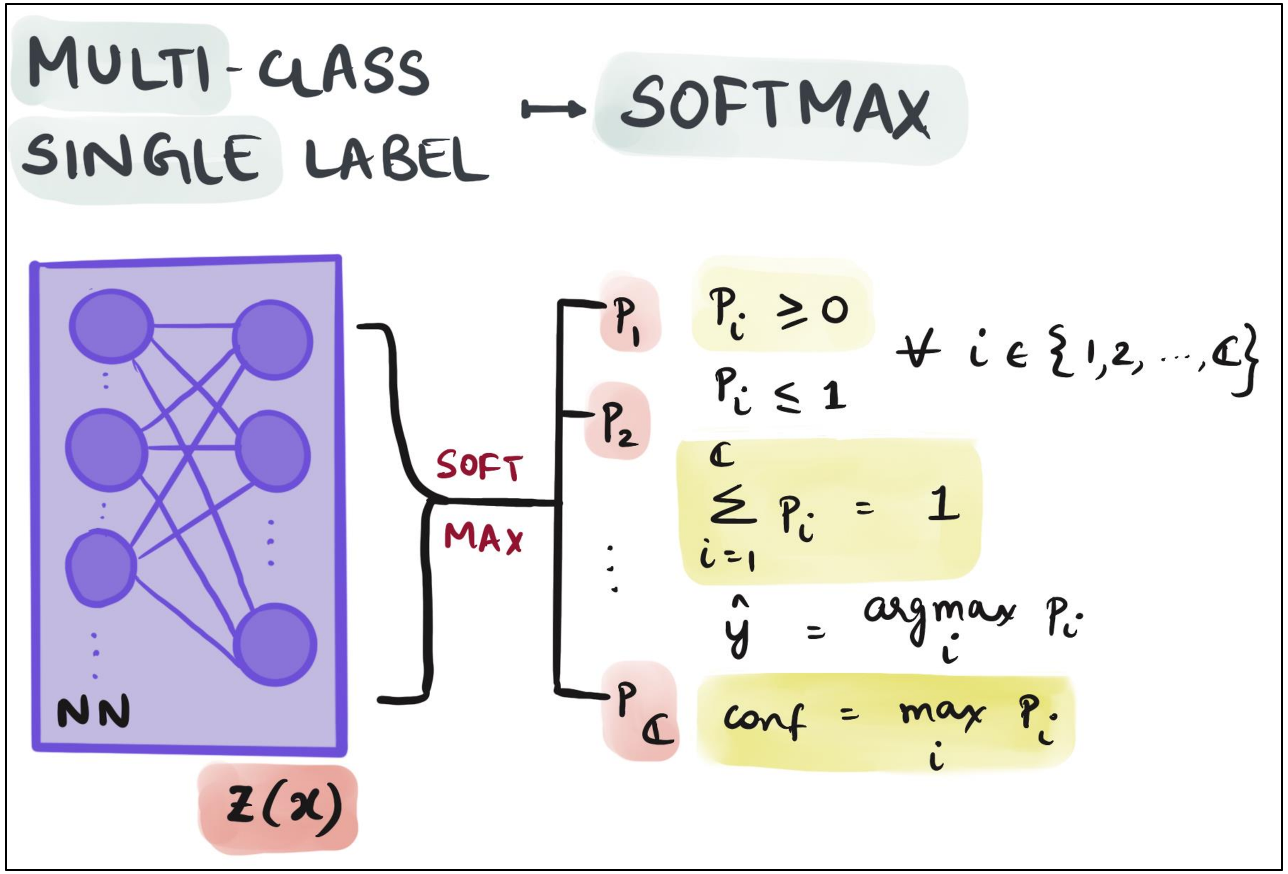
Binary Classification/Single Label



Mutually exclusive and exhaustive, i.e., an input instance can belong to either class, but not both and their **probabilities sum to 1.**

https://web.stanford.edu/~nanbhas/blog/sigmoid-softmax/#binary-classification

Multi-class Classification/Single Label

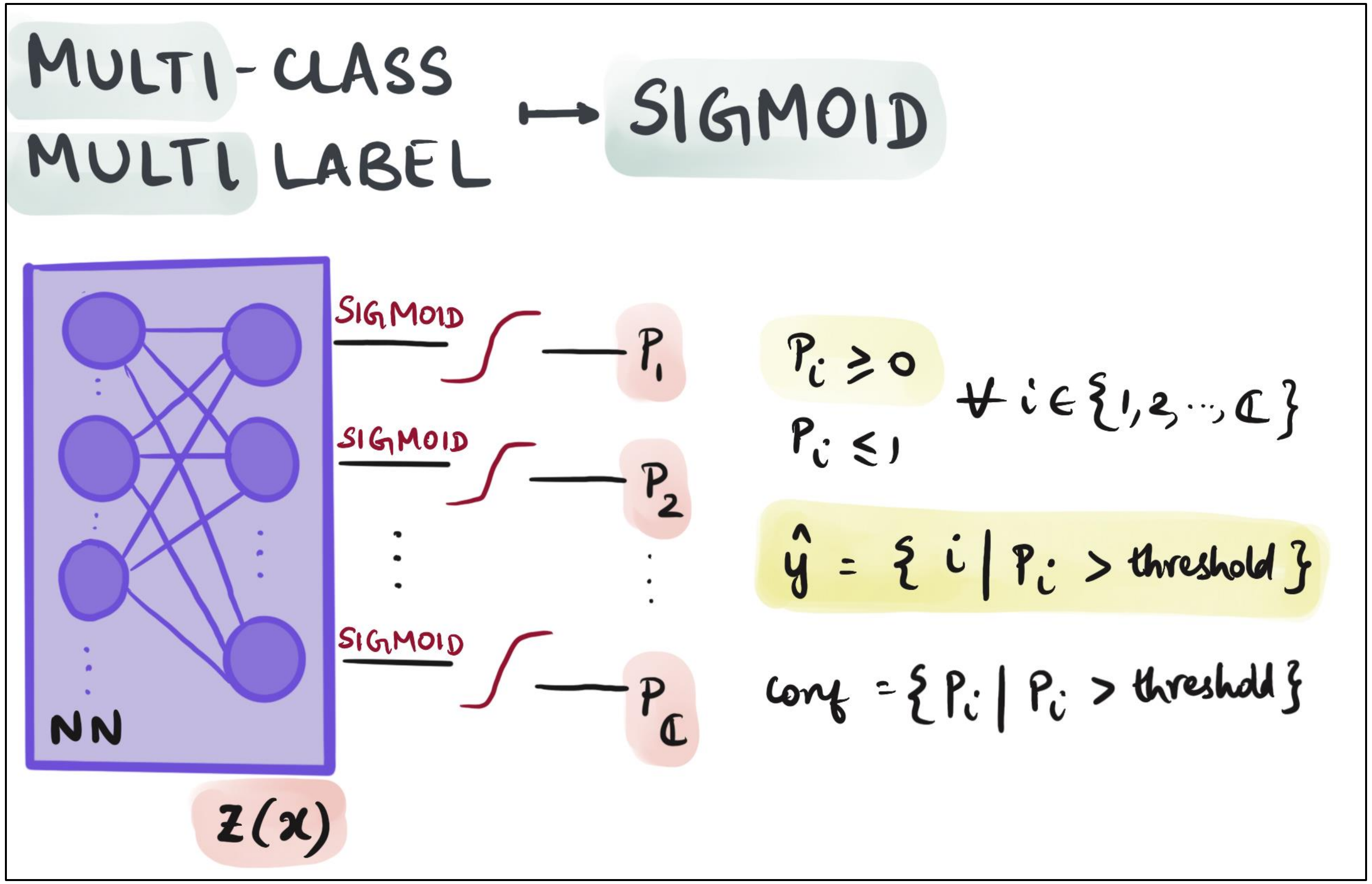


Mutually exclusive and exhaustive, i.e., an input instance can belong to only one of these classes, not more and **their probabilities sum to 1.**

https://web.stanford.edu/~nanbhas/blog/sigmoid-softmax/#binary-classification

Multi-class Classification/Multi Label

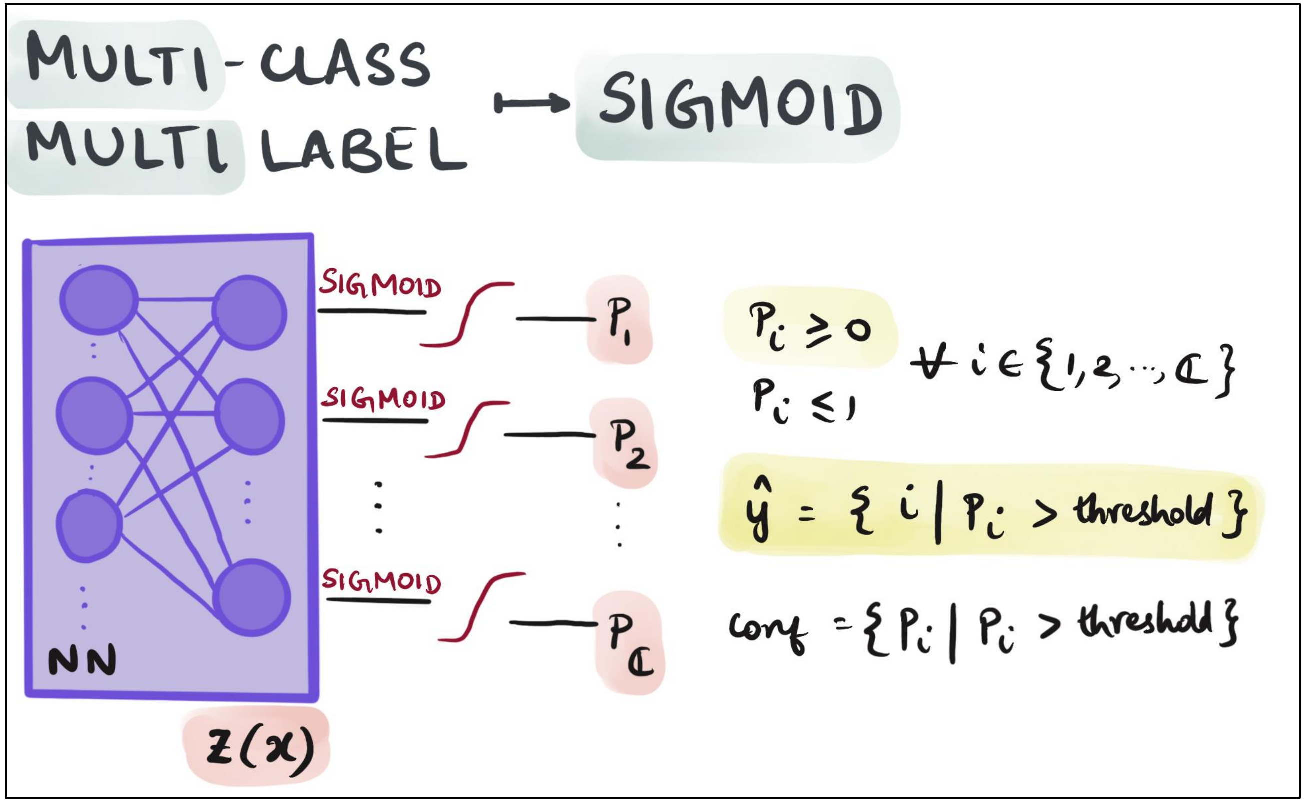
If input data can belong to more than one class in a multi-class classification problem?



For instance, Genre **classification of movies** **(a movie can fall into multiple genres)** **or** **classification of chest x-rays** **(a given chest x-ray can have more than one disease).**

https://web.stanford.edu/~nanbhas/blog/sigmoid-softmax/#binary-classification

Multi-class Classification/Multi Label



Here Classes are NOT mutually exclusive. **i.e., train a binary classifier independently for each class.** This can be done easily by just applying **sigmoid function** to each of raw scores. **Note that the output probabilities will NOT sum to 1.**

https://web.stanford.edu/~nanbhas/blog/sigmoid-softmax/#binary-classification

Implementation of SoftMax/Sigmoid

```
import torch

def getSoftmaxScores(inputs, dimen):
        ''' Get the softmax scores '''
        print('---Softmax---')
        print('---Dim = ' + str(dimen) + '---')
        softmaxFunc = torch.nn.Softmax(dim = dimen)
        softmaxScores = softmaxFunc(inputs)
        print('Softmax Scores: \n', softmaxScores)
        sums_0 = torch.sum(softmaxScores, dim=0)
        sums_1 = torch.sum(softmaxScores, dim=1)
        print('Sum over dimension 0: \n', sums_0)
        print('Sum over dimension 1: \n', sums_1)

def getSigmoidScores(inputs):
        ''' Get the sigmoid scores: they are element-wise '''
        print('---Sigmoid---')
        sigmoidScores = torch.sigmoid(inputs)
        print('Sigmoid Scores: \n', sigmoidScores)

logits = torch.randn(2, 3)*10 - 5
print('Logits: ', logits)
```

https://web.stanford.edu/~nanbhas/blog/sigmoid-softmax/#binary-classification

A fixed-window Neural Language model

output distribution

$$\hat{\boldsymbol{y}} = \mathrm{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2) \in \mathbb{R}^{|V|}$$

hidden layer

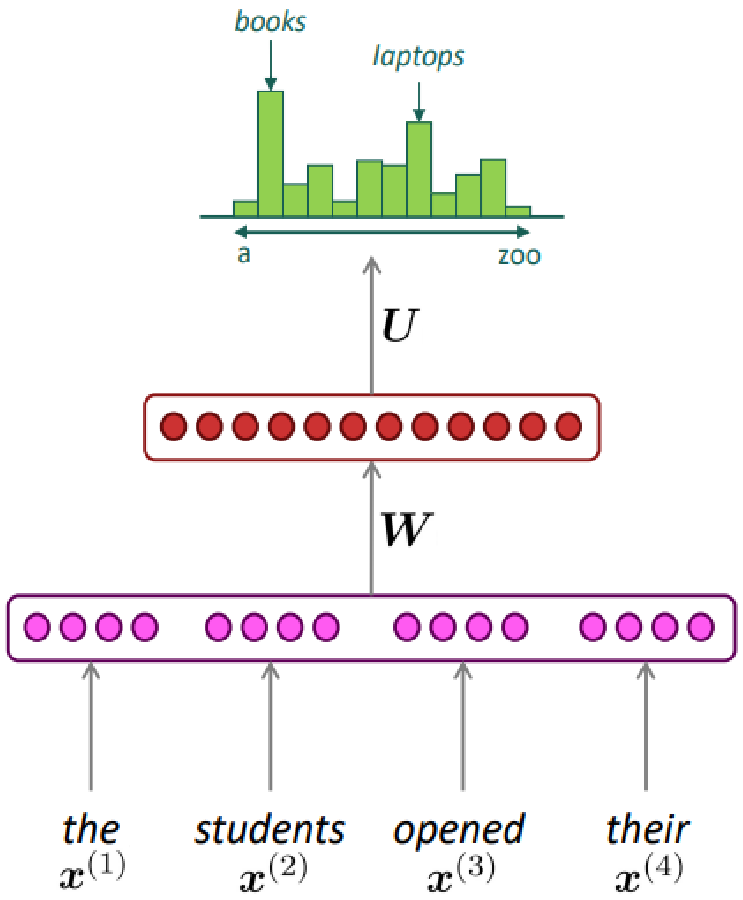
$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

concatenated word embeddings

$$\boldsymbol{e} = [\boldsymbol{e}^{(1)}; \boldsymbol{e}^{(2)}; \boldsymbol{e}^{(3)}; \boldsymbol{e}^{(4)}]$$

words / one-hot vectors

$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}$$



Credits: Slide adapted from [1]

A fixed-window Neural Language Model

Approximately: Y. Bengio, et al. (2000/2003): A Neural Probabilistic Language Model

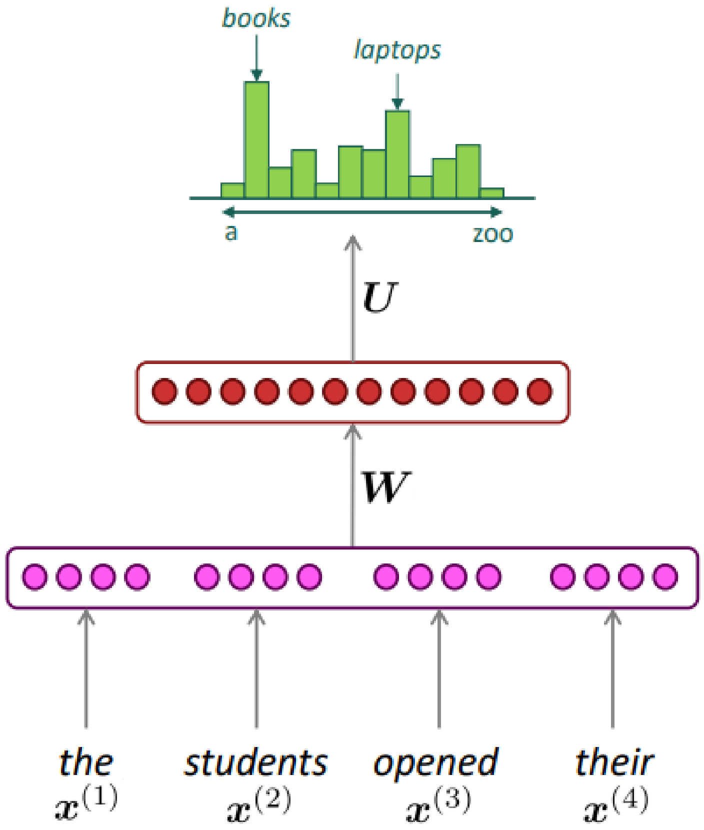
Improvements over *n*-gram LM:

- No sparsity problem
- Don't need to store all observed *n*-grams

Remaining **problems**:

- Fixed window is too small
- Enlarging window enlarges W
- Window can never be large enough!
- $x^{(1)}$ and $x^{(2)}$ are multiplied by completely different weights in W. No symmetry in how the inputs are processed.

We need a neural architecture that can process *any length input*



Credits: Slide adapted from [1]

References

[1] https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1234/slides/cs224n-2023-lecture05-rnnlm.pdf

[2] https://web.stanford.edu/~jurafsky/slp3/slides/LM_4.pdf

[3] https://stanford-cs324.github.io/winter2022/lectures/introduction/

Acknowledgments

- These slides were adapted from the book SPEECH and LANGUAGE PROCESSING: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition
- Practical Natural Language Processing (A Comprehensive Guide to Building Real-World NLP Systems) O'reilly and some modifications from presentations and resources found in the WEB by several scholars.