Natural Language Processing

AIGC 5501

Neural Networks
Deep Learning Intro

Instructor: Ritwick Dutta
Email: ritwick.dutta@humber.ca

This Week

Neural Networks
Deep Learning Intro
Word2Vec
GloVe

A Brief History of Neural Nets

Increase of data transmission speed

1940 ~ 1970: The 1st AI Boom | 1980 ~ 1990: The 2nd Boom | 2006 ~: The 3rd Boom

The advent of the idea of AI

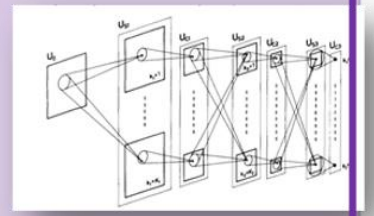
1958: Perceptron

1968: "2001 : A Space Odyssey"

1st AI Winter

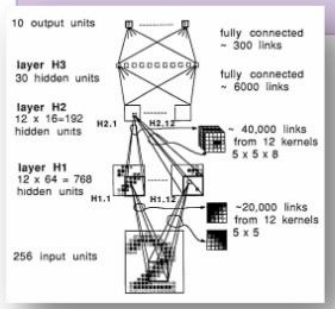
- Limitation of hardware
- Lack of computation algorithms
- Hardships in linearly inseparable data problems

1980: Neocognitron



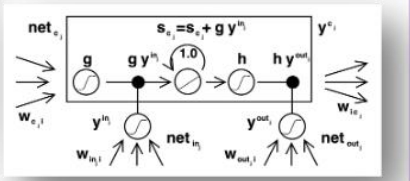
1986: Backpropagation

1989: The first practical CNN



2001: Release of Xbox

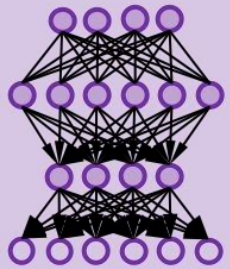
1997: LSTM



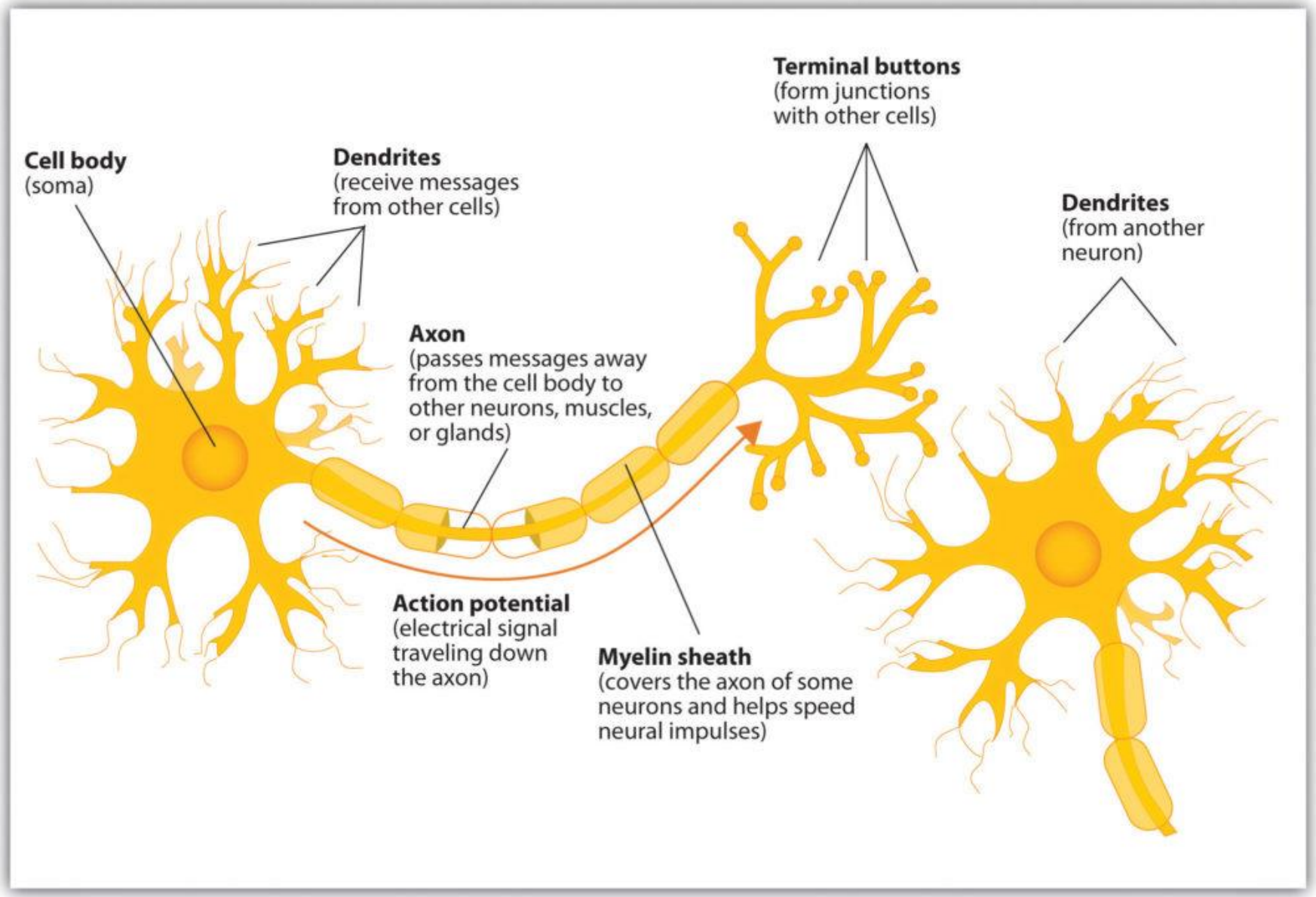
2nd AI Winter

- Limitation of hardware
- Shortage of data sources
- Lack of theories for hyper parameters
- **Vanishing / exploding gradient problem**

2006: Pretraining of deep belief net

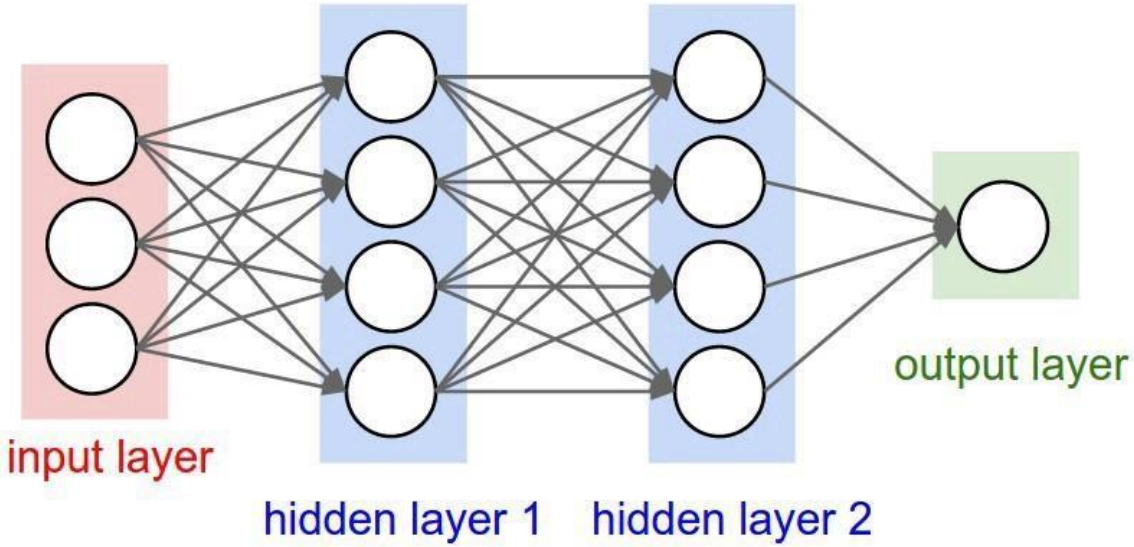


Neural networks ~ Human Brain

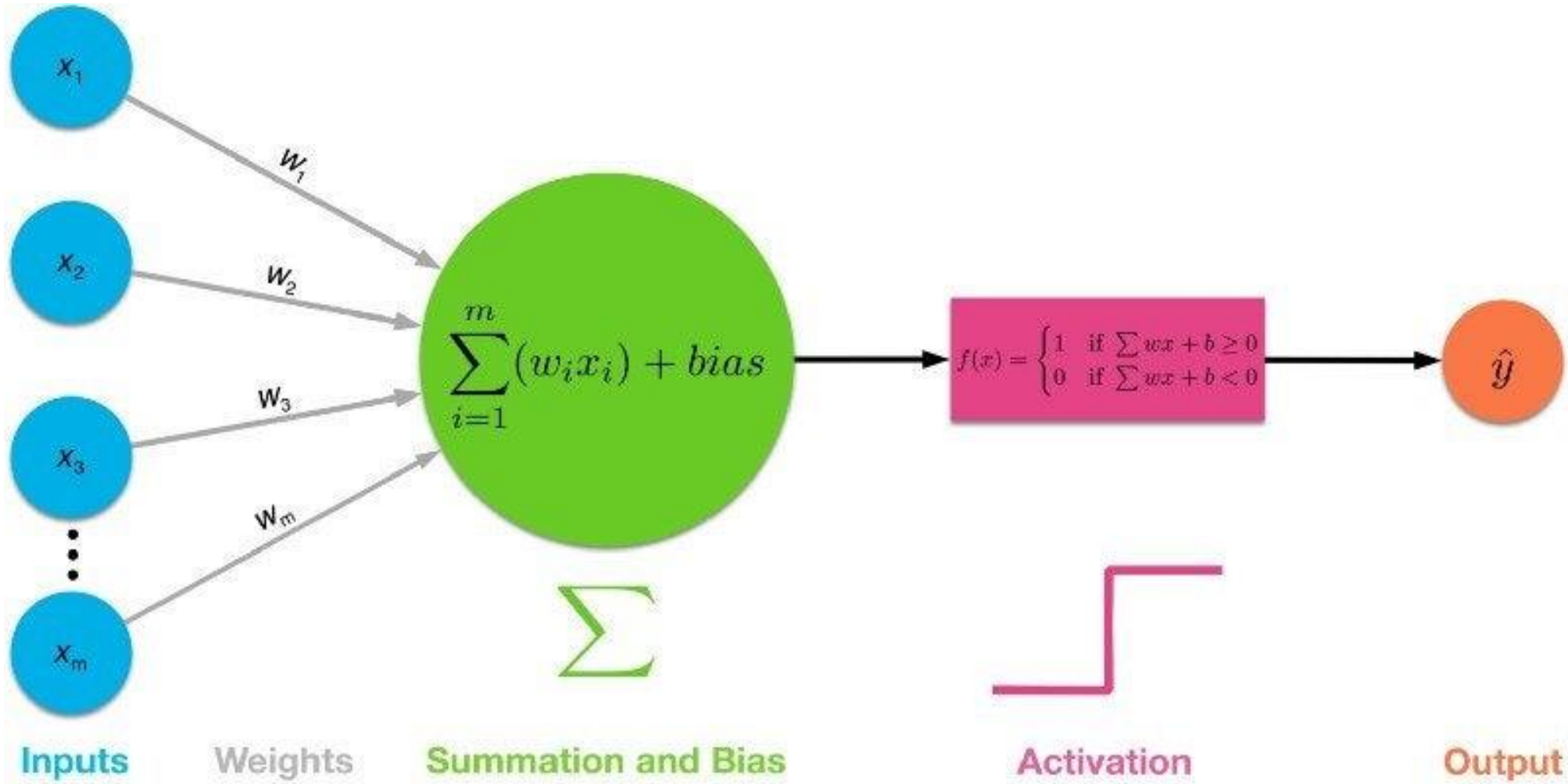


Neural networks

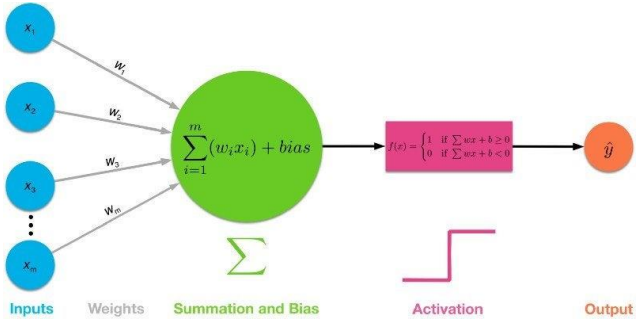
- A neural network algorithm will try to create a function to map your input to your desired output.
- As an example, you want the program output “cat” as an output, given an image of a cat.
- Take a look at the image. The cat image is the input in the input layer, while the “cat” will be on the outer layer. The hidden layers are the function that will map the image correct category.

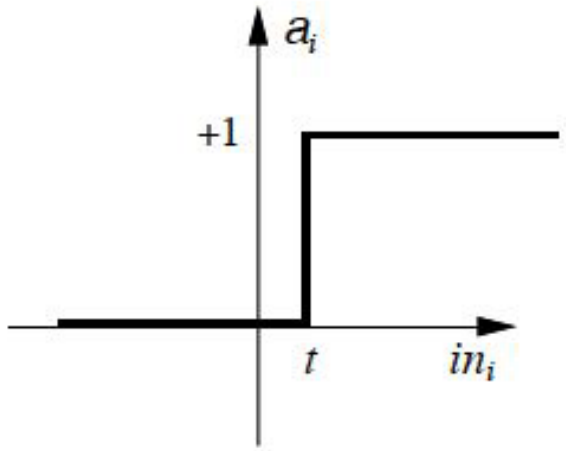


Neural networks

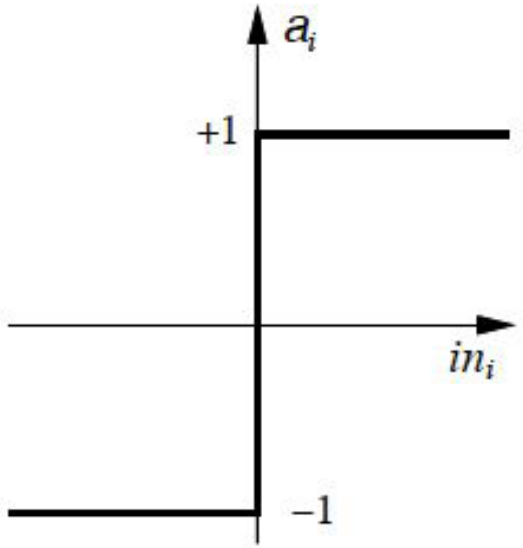


Activation functions

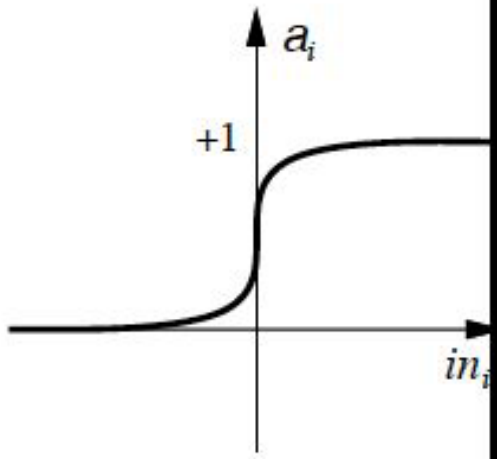




(a) Step function



(b) Sign function

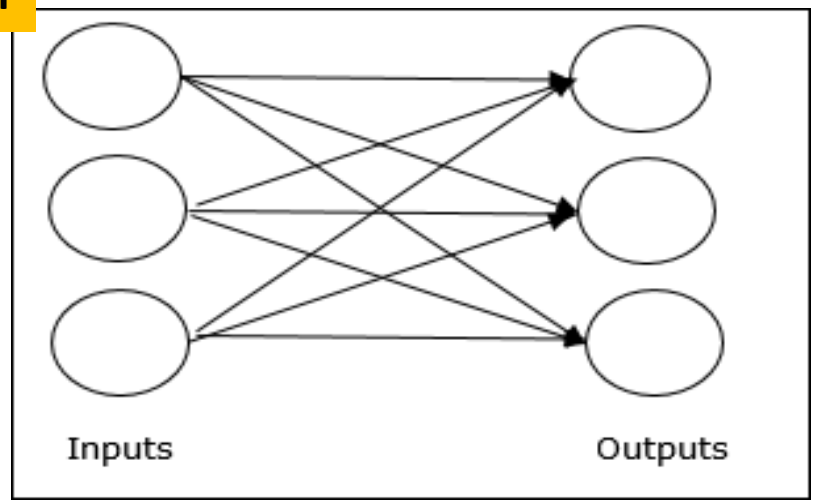


(c) Sigmoid function

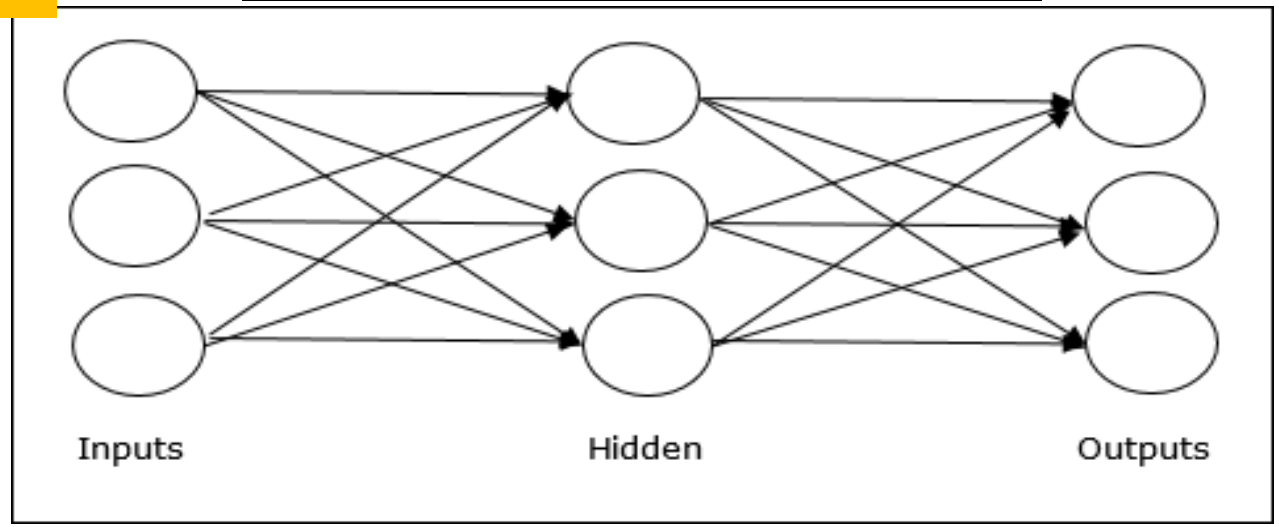
Perceptron units

NN Designs

Single layer



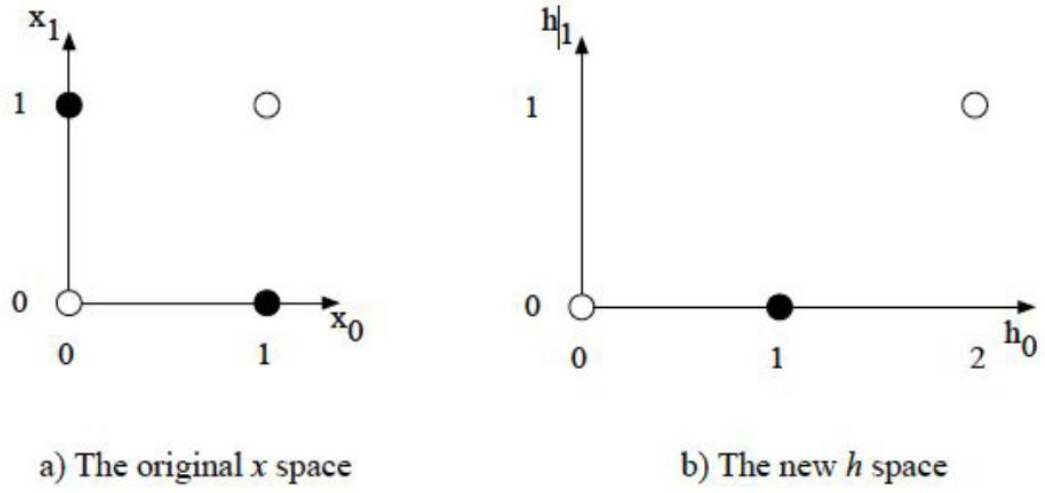
Multilayer



Representation learning in MLPs

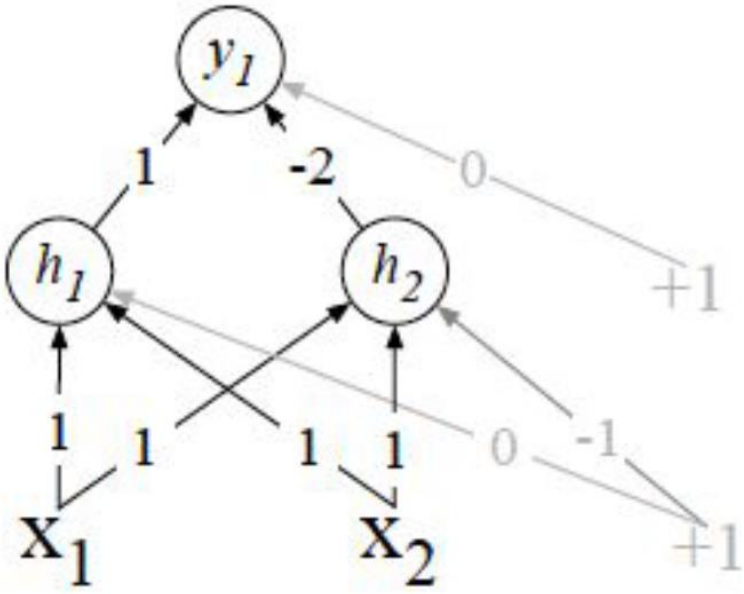
Representation Learning

- The hidden layers can learn to form useful representations of the input



a) The original x space

b) The new h space



Neural net classifiers

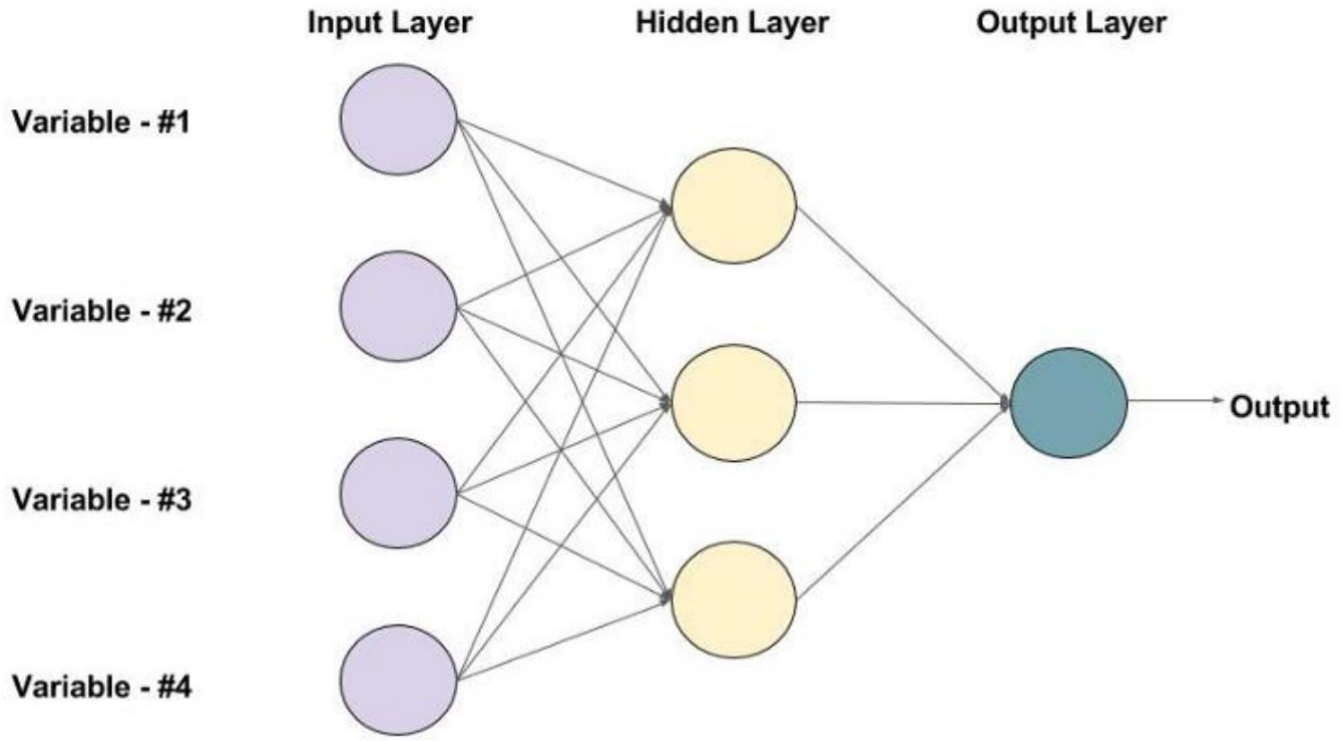
- Trained properly, more powerful classifier than naïve Bayes, logistic regression, decision trees
    - a neural network with one "hidden" layer can be shown to learn any function

- Not dependent on the availability of many kinds of features based on domain knowledge
    - Can take raw words (word embeddings) as inputs
    - Induce features as part of the process of learning to classify

- Need sufficient training data to learn latent features automatically

Feedforward network

- Computation proceeds iteratively from one layer of units to the next.
- A feedforward network is a **multilayer** feedforward network in which the units are connected with **no cycles**.
- I.e., the outputs from units in each layer are passed to units in the next higher layer, and no outputs are passed back to lower layers.
- Sometimes called **multi-layer perceptrons (MLPs)**; however, units in modern multilayer networks aren't perceptrons (linear). They are units with non-linear activation functions (e.g. tanh).



An example of a Feed-forward Neural Network with one hidden layer (with 3 neurons)

Simple FFNNs have three types of nodes:

- Input units,
- Hidden units, and
- Output units

Feedforward network

Fully-connected FFNN

Each unit in each layer takes as input the outputs from all the units in the previous layer, and there is a link between every pair of units from two adjacent layers.

Hidden Layers (h)

Consists of hidden units where we take **weighted sum of its inputs** and then **apply a non-linearity**.

Take a weighted sum of its inputs and then applying a non-linearity.

$$h = \sigma(Wx + b)$$

$x \in \mathbb{R}^{n_0}$

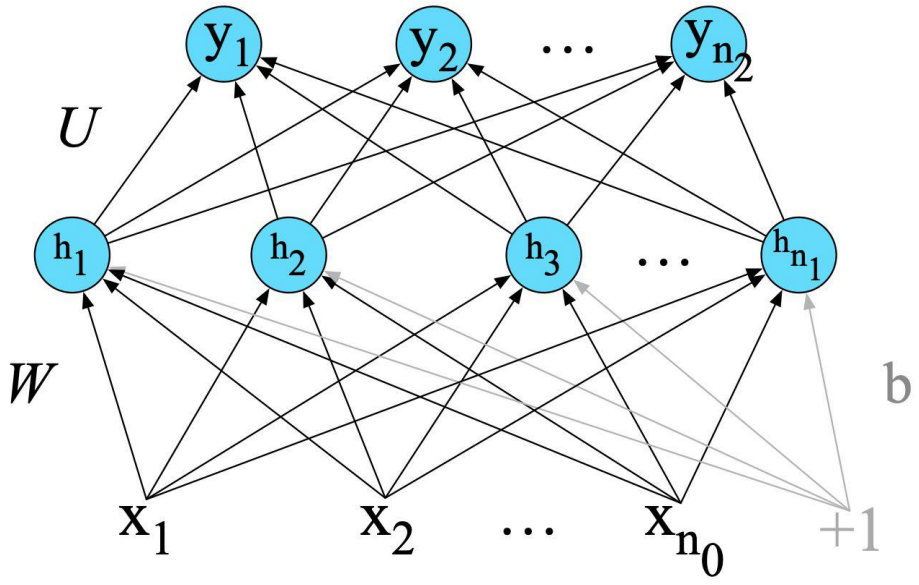
$h \in \mathbb{R}^{n_1}$

$b \in \mathbb{R}^{n_1}$

$$h_j = \sigma\left(\sum_{i=1}^{n_0} W_{ji} x_i + b_j\right)$$

So, $W \in \mathbb{R}^{n_1 \times n_0}$

W_{ji} represents weight to h_j from x_i.



Output nodes

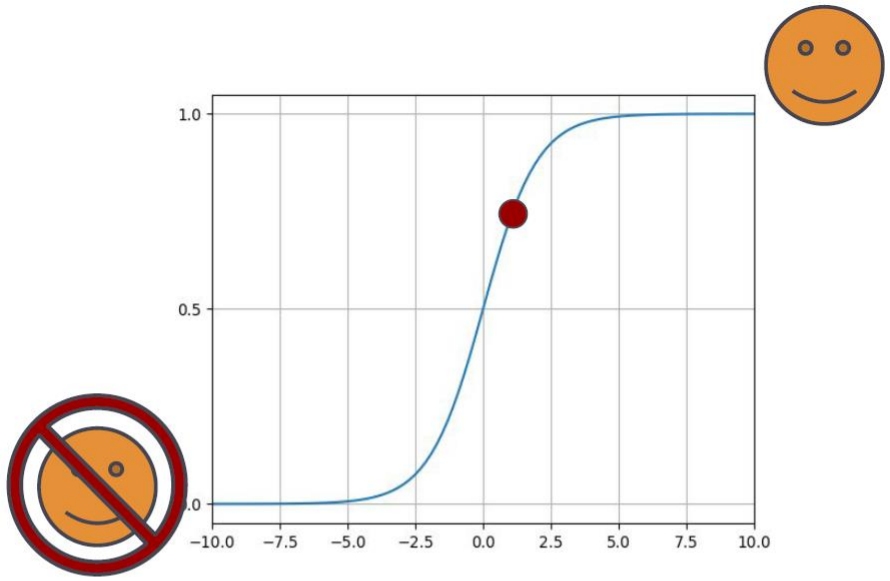
- Could have a single output node, e.g. binary classification (e.g. sentiment analysis).

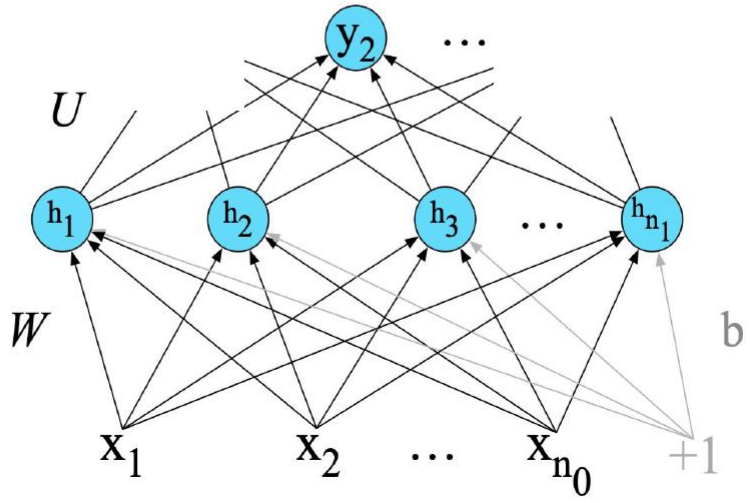
- Could use one output node per class, e.g. multinomial classification (e.g. POS-tagging).

- In case of classification, output layer should produce a probability distribution across the output nodes.

Output nodes

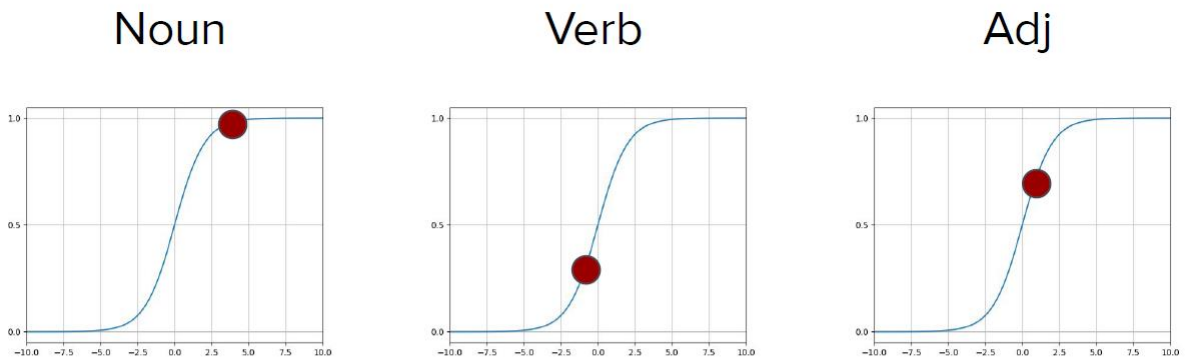
- Could have a single output node, e.g. binary classification (e.g. sentiment analysis).

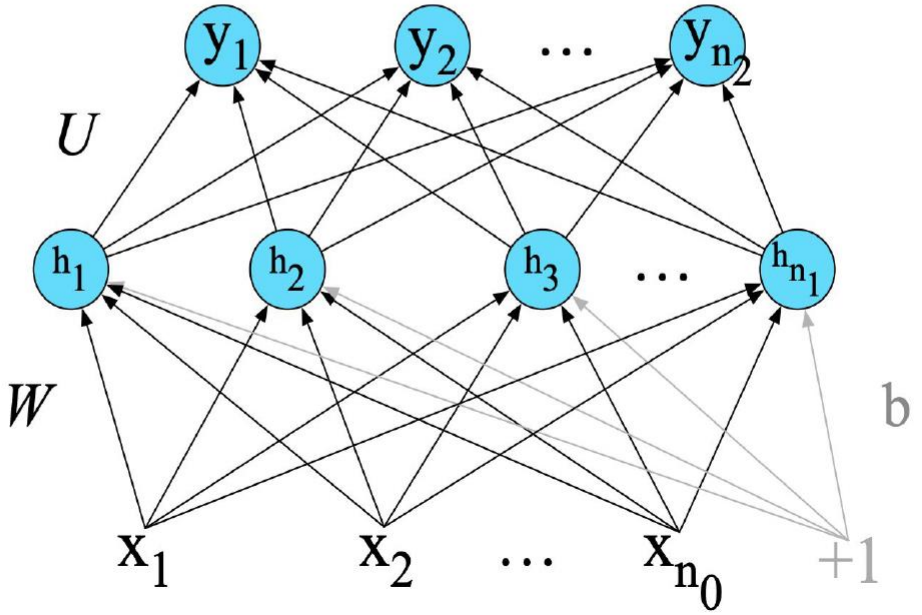




Output nodes

- Could use one output node per class, e.g. multinomial classification (e.g. POS-tagging).

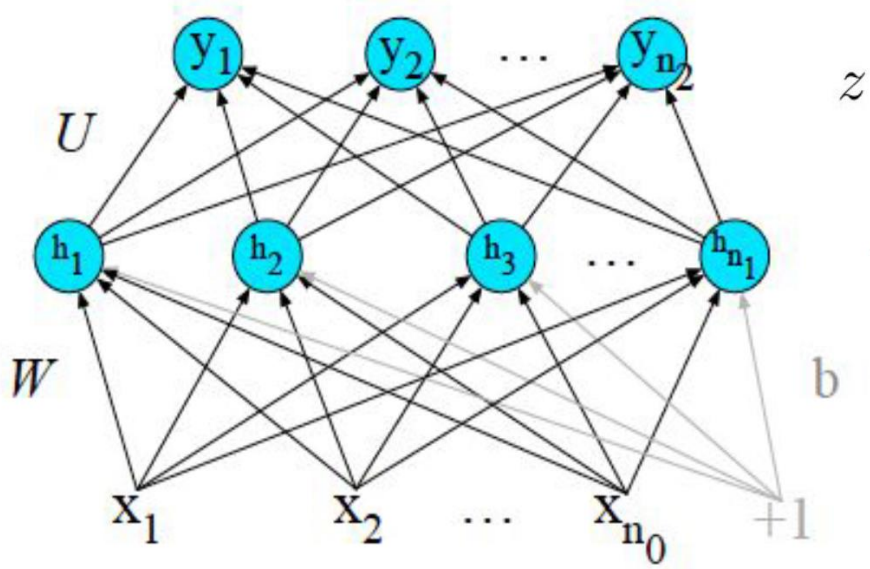




Computing probability distribution

$z = Uh$

$h = \sigma(Wx + b)$



$z \in \mathbb{R}^{n_2} \quad U \in \mathbb{R}^{n_2 \times n_1}$

z is a vector of real-valued numbers.

We need a vector of probabilities!

Softmax Normalizes the vector of reals (values between 0-1 and sums up to 1).

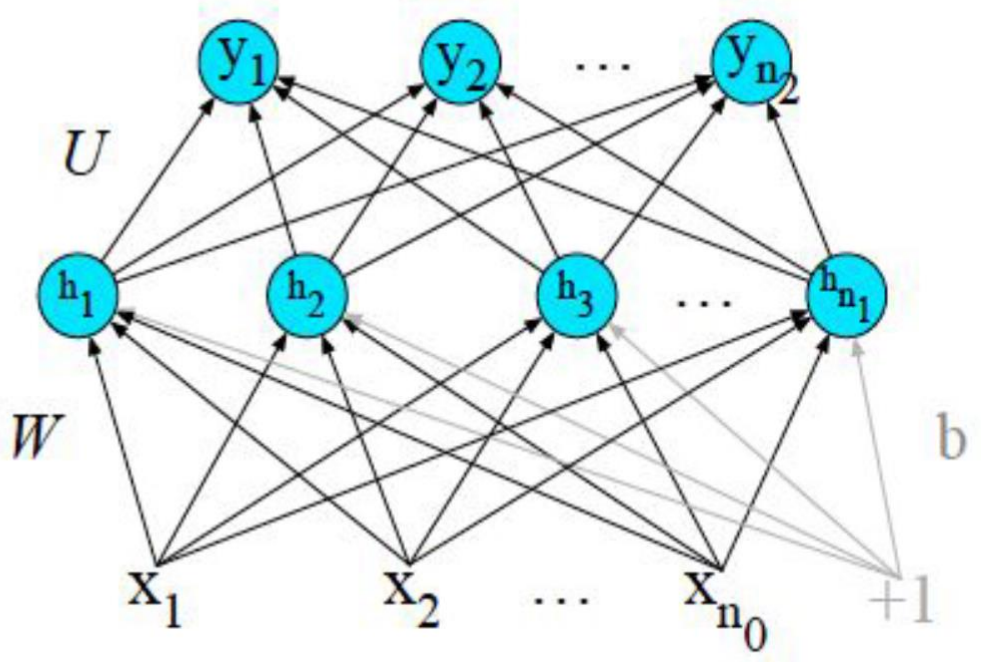
The softmax of an input vector $z = [z_1, z_2, ..., z_k]$ is thus a vector itself:

$$\text{softmax}(z) = \left[\frac{e^{z_1}}{\sum_{i=1}^{k} e^{z_i}}, \frac{e^{z_2}}{\sum_{i=1}^{k} e^{z_i}}, ..., \frac{e^{z_k}}{\sum_{i=1}^{k} e^{z_i}} \right]$$

- ***z*** = [0.6 1.1 -1.5 1.2 3.2 -1.1]
- softmax(***z***) = [0.055 0.090 0.0067 0.10 0.74 0.010]

Computing probability distribution

$$h = \sigma(Wx + b)$$
$$z = Uh$$
$$y = \text{softmax}(z)$$



Training Neural Nets

- Determines the weights and biases, *W* and *b*, for each layer
- Supervised learning
- Use optimization methods like (**stochastic) gradient descent**

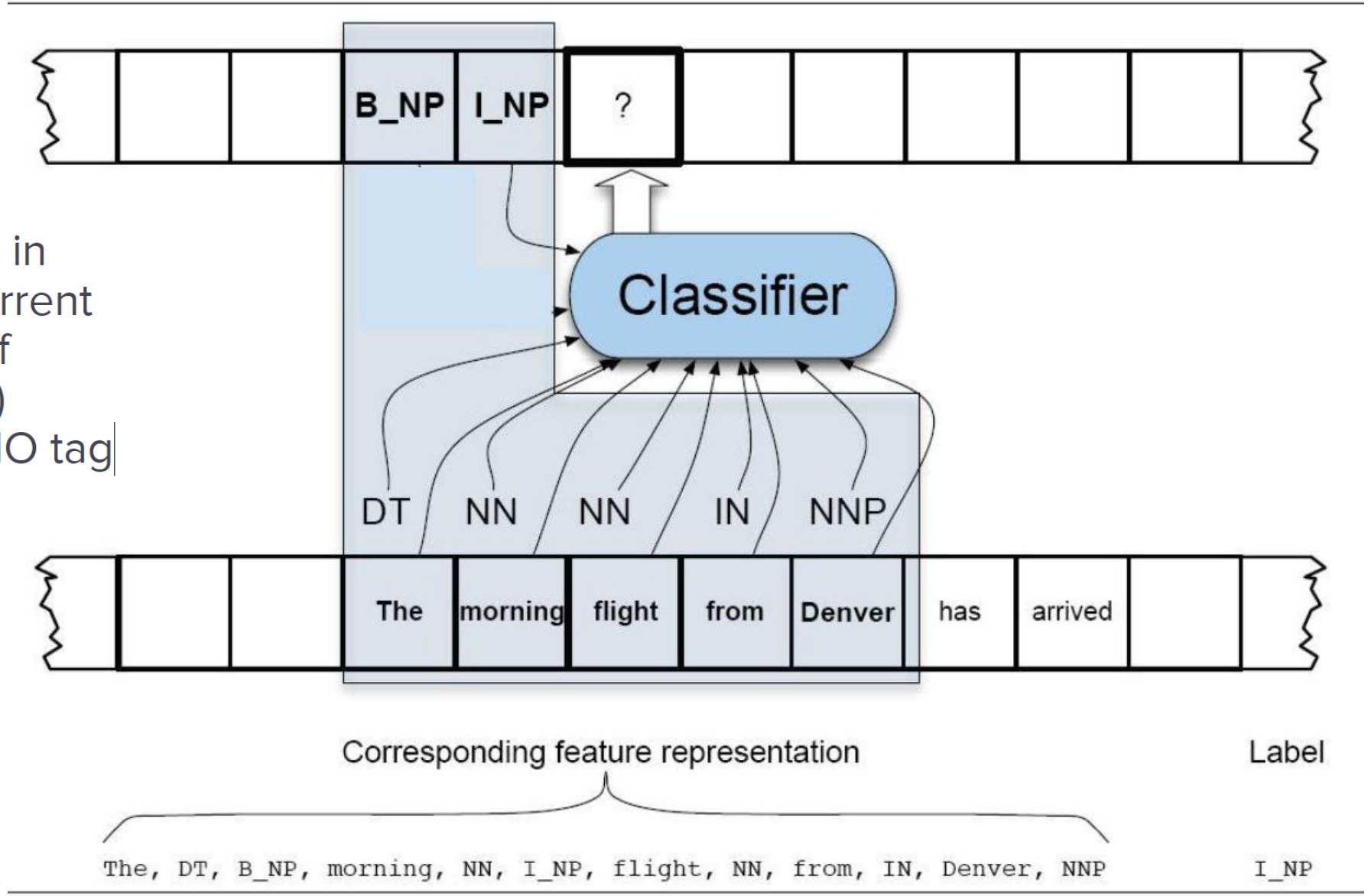
Goal:

The output (y) from the network for each training instance should be as close as possible to the gold (correct) label (y)

Want to modify the weights to minimize this distance

Training Neural Nets

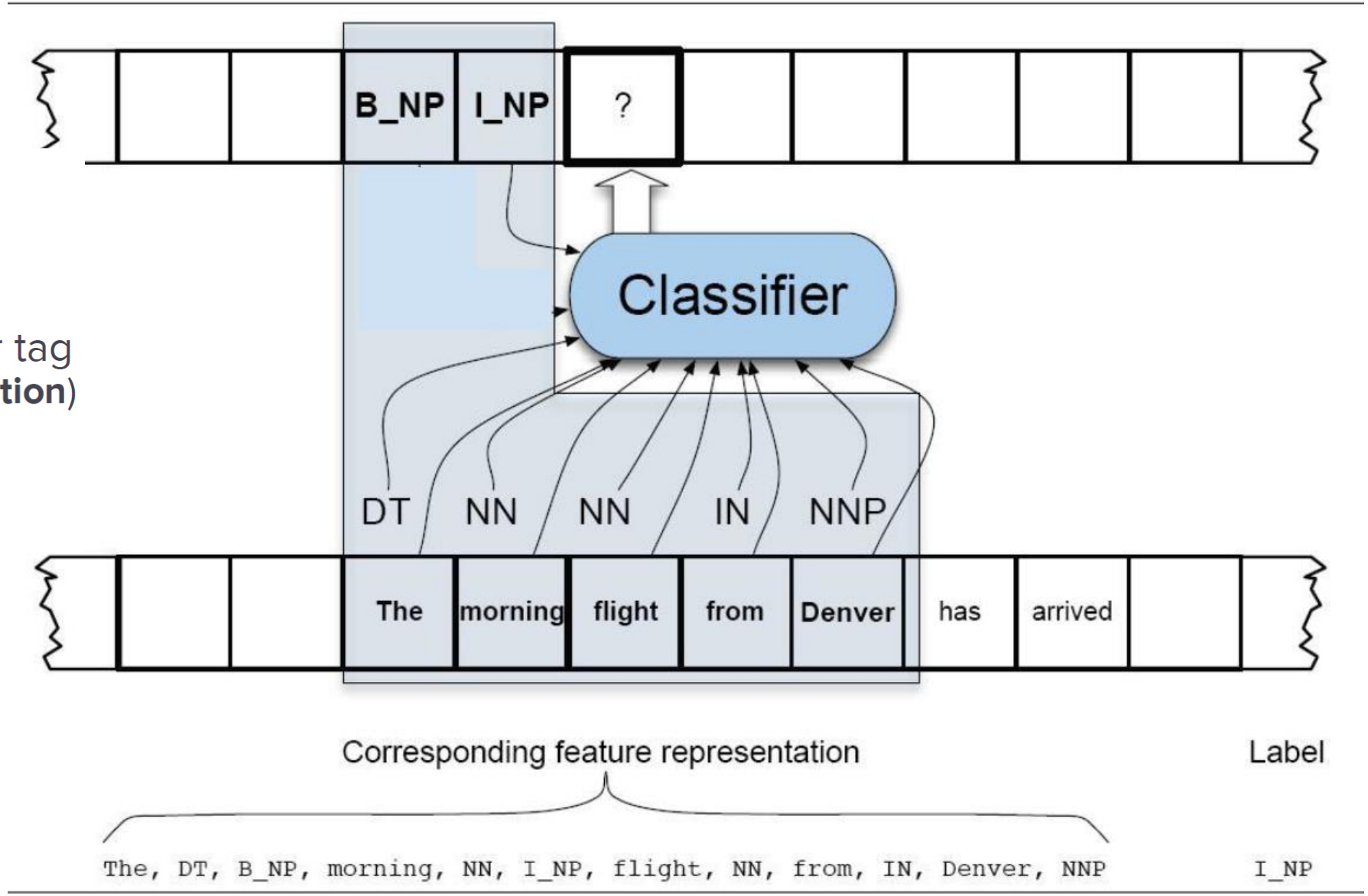
- We are interested in **classifying** the current token (at center of moving "window") according to its BIO tag



Training Neural Nets

Output Layer?

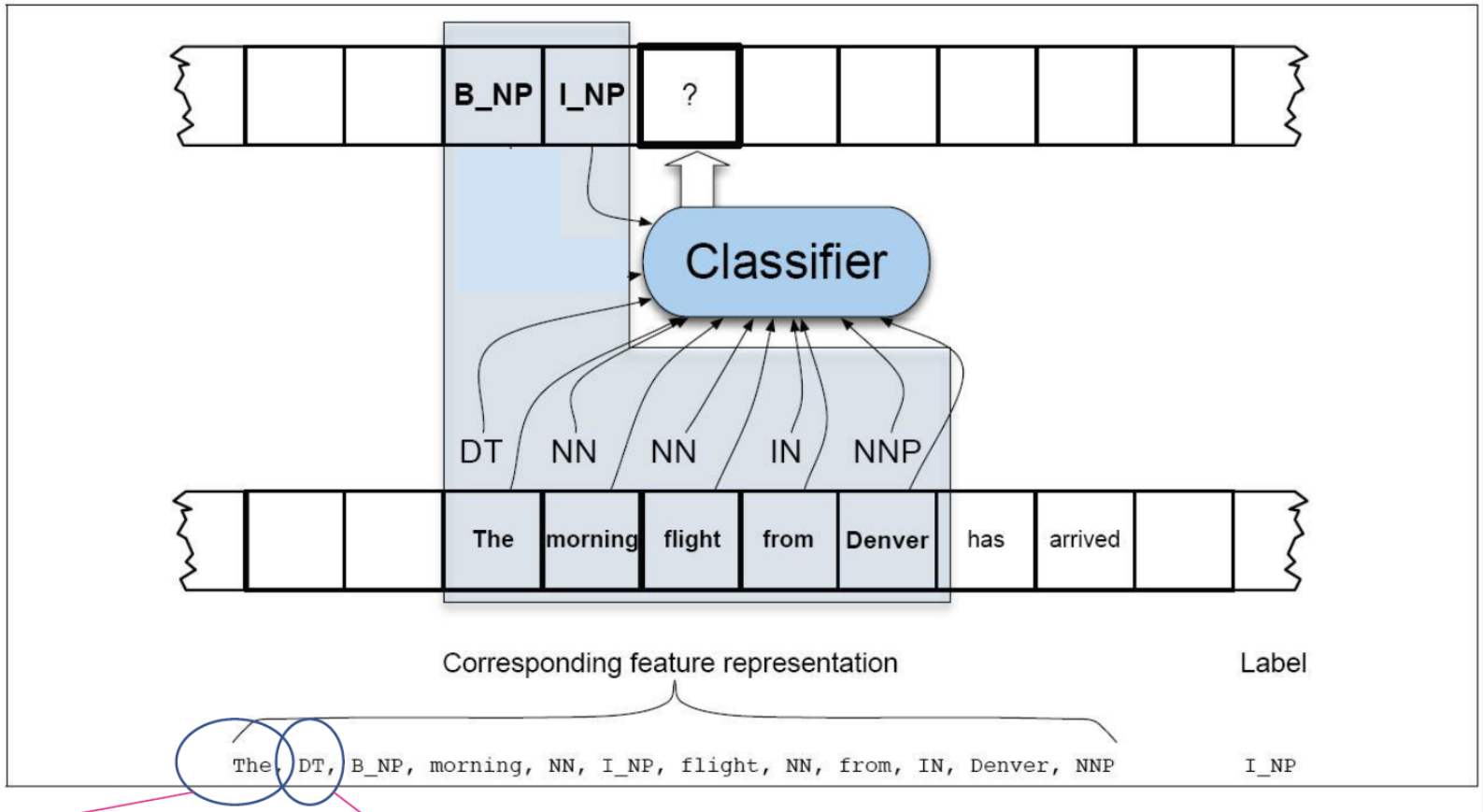
- One output node per tag (**multinomial classification**)
- A single output node (**binary classification**)



Designing Neural Nets

Converting features to integer/real values

- Binarize each feature
- And then concatenate



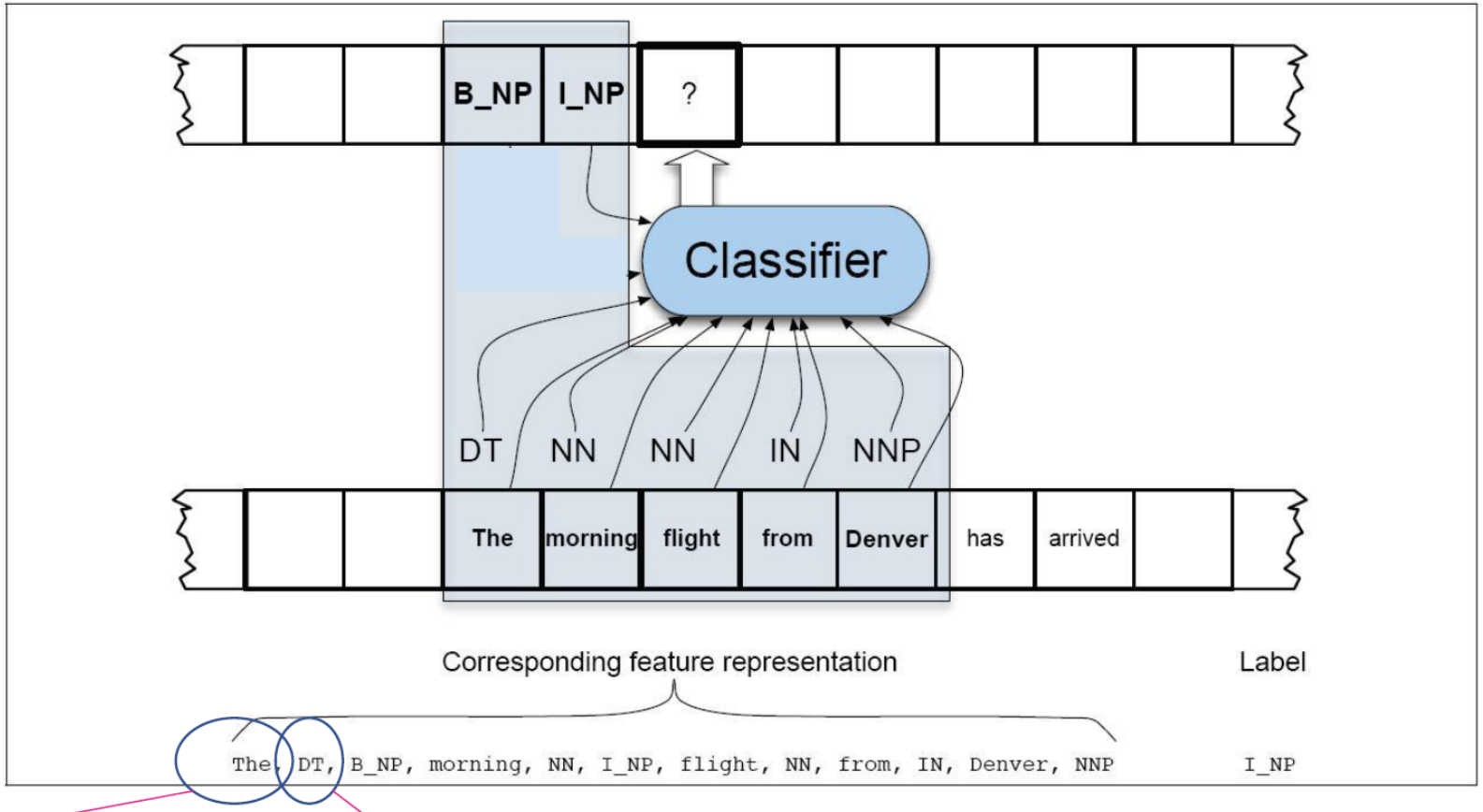
[a? apple? … dog? eat? …. the? … zipper?]
[0 0 … 0 0 … 1 … 0]

[NN? NNS? … DT? IN? …. VB? … VBN?]
[0 0 … 1 0 … 0 … 0]

Designing Neural Nets

Converting features to integer/real values

- Binarize each feature
  - Or convert to dense embeddings
- And then concatenate



E.g. Word2vec representation

[0.22 -0.8 … 0.55 … -0.35]

[NN? NNS? … DT? IN? …. VB? … VBN?]

[0 0 … 1 0 … 0 … 0]

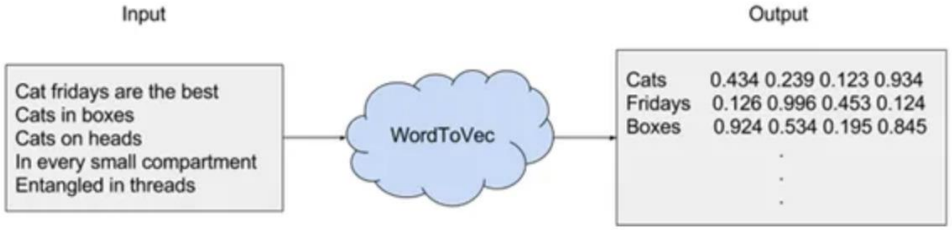
The move to non-linear classifiers

- E.g. **Deep Learning** (aka neural networks)
  - a family of nonlinear methods
  - learn complex functions of the input through multiple layers of computation
- Facilitates the use of **word embeddings**
- Remove the need for rich hand-derived features
    - Learns useful representations of the input **automatically**

Advances in GPU (graphics processing unit) development were also key

- Make deep learning (almost) practical

Word2Vec



In word embeddings, each word is represented by a vector, usually with a size of 100 to 300. **Word2Vec** is a popular method to create embeddings.

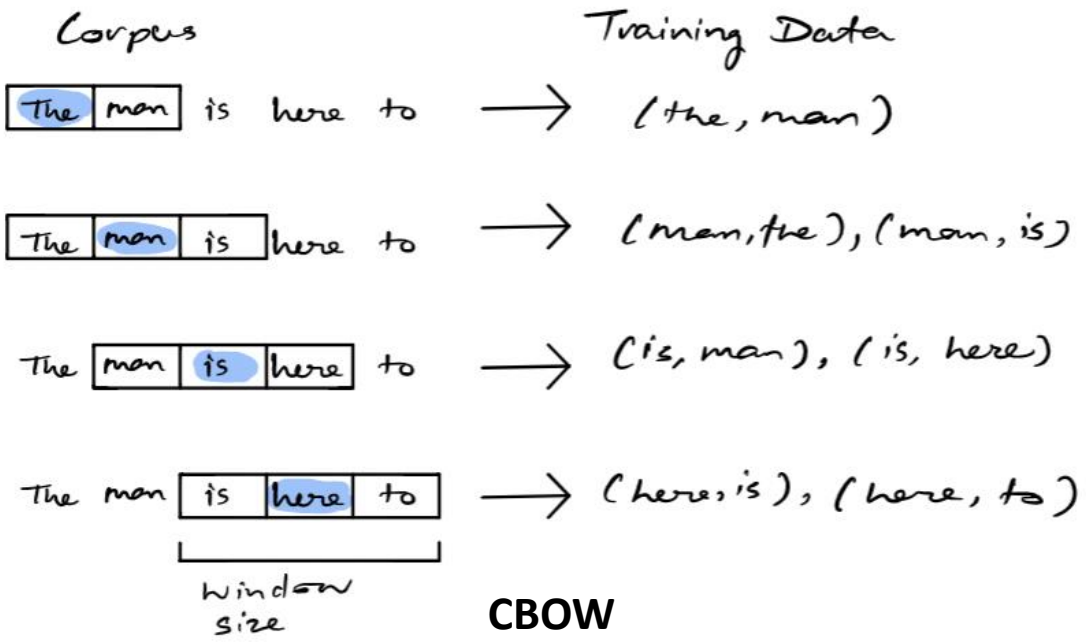
The basic intuition behind **Word2Vec** is this: We can get useful information about a word by observing its context/neighbours.

In Word2Vec, there are two architectures or learning algorithms we can use to obtain vector representation (just another word for embedding) of words: **Continuous Bag of Words** (a.k.a. CBOW) and **Skip-gram**.

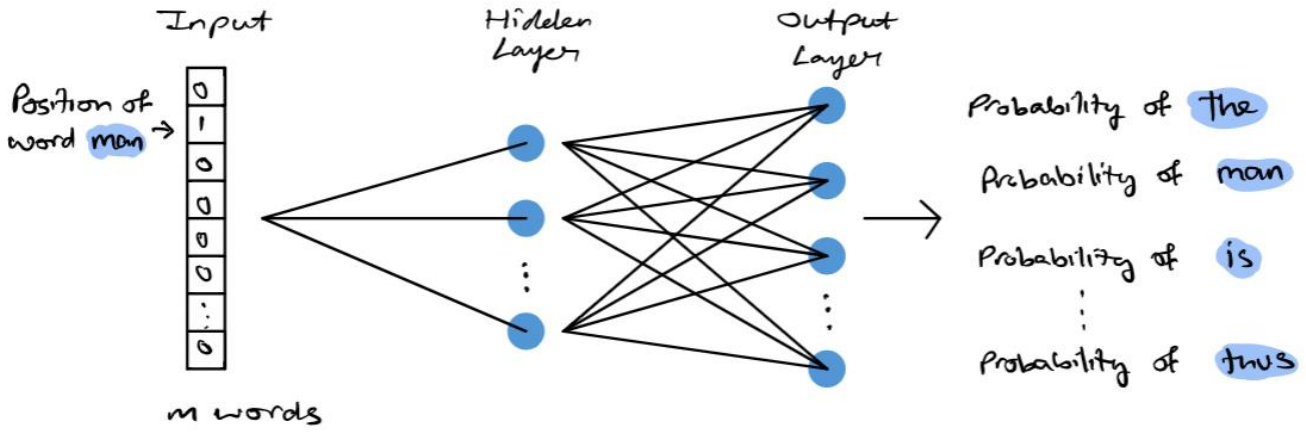
- CBOW: Predict focus word given surrounding context words
- Skip-gram: Predict context words given focus word (the focus of this post)

CBOW & Skip-Gram

Given a set of sentences (also called corpus) the model loops on the words of each sentence and either tries to use the current word *w* in order to predict its neighbors (i.e., its context), this approach is called "**Skip-Gram**", or it uses each of these contexts to predict the current word *w*, in that case the method is called "**Continuous Bag Of Words**" (CBOW). In CBOW To limit the number of words in each context, a parameter called "window size" is used.



CBOW



Skip-Gram

Gensim Word2Vec Pre-trained Models

Name	Num vectors	File size	Base dataset	Description	Parameters	Preprocessing
conceptnet-numberbatch-17-06-300	1917247	1168 MB	ConceptNet, word2vec, GloVe, and OpenSubtitles 2016	ConceptNet Numberbatch consists of state-of-the-art semantic vectors (also known as word embeddings) that can be used directly as a representation of word meanings or as a starting point for further machine learning. ConceptNet Numberbatch is part of the ConceptNet open data project. ConceptNet provides lots of ways to compute with word meanings, one of which is word embeddings. ConceptNet Numberbatch is a snapshot of just the word embeddings. It is built using an ensemble that combines data from ConceptNet, word2vec, GloVe, and OpenSubtitles 2016, using a variation on retrofitting.	•dimension - 300	-
fasttext-wiki-news-subwords-300	999999	958 MB	Wikipedia 2017, UMBC webbase corpus and statmt.org news dataset (16B tokens)	1 million word vectors trained on Wikipedia 2017, UMBC webbase corpus and statmt.org news dataset (16B tokens).	•dimension - 300	-
glove-twitter-100	1193514	387 MB	Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased)	Pre-trained vectors based on 2B tweets, 27B tokens, 1.2M vocab, uncased (https://nlp.stanford.edu/projects/glove/)	•dimension - 100	Converted to w2v format with python -m gensim.scripts.glove2word2vec -i <fname> -o glove-twitter-100.txt.
glove-twitter-200	1193514	758 MB	Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased)	Pre-trained vectors based on 2B tweets, 27B tokens, 1.2M vocab, uncased (https://nlp.stanford.edu/projects/glove/).	•dimension - 200	Converted to w2v format with python -m gensim.scripts.glove2word2vec -i <fname> -o glove-twitter-200.txt.
glove-twitter-25	1193514	104 MB	Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased)	Pre-trained vectors based on 2B tweets, 27B tokens, 1.2M vocab, uncased (https://nlp.stanford.edu/projects/glove/).	•dimension - 25	Converted to w2v format with python -m gensim.scripts.glove2word2vec -i <fname> -o glove-twitter-25.txt.
glove-twitter-50	1193514	199 MB	Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased)	Pre-trained vectors based on 2B tweets, 27B tokens, 1.2M vocab, uncased (https://nlp.stanford.edu/projects/glove/)	•dimension - 50	Converted to w2v format with python -m gensim.scripts.glove2word2vec -i <fname> -o glove-twitter-50.txt.
glove-wiki-gigaword-100	400000	128 MB	Wikipedia 2014 + Gigaword 5 (6B tokens, uncased)	Pre-trained vectors based on Wikipedia 2014 + Gigaword 5.6B tokens, 400K vocab, uncased (https://nlp.stanford.edu/projects/glove/).	•dimension - 100	Converted to w2v format with python -m gensim.scripts.glove2word2vec -i <fname> -o glove-wiki-gigaword-100.txt.
glove-wiki-gigaword-200	400000	252 MB	Wikipedia 2014 + Gigaword 5 (6B tokens, uncased)	Pre-trained vectors based on Wikipedia 2014 + Gigaword, 5.6B tokens, 400K vocab, uncased (https://nlp.stanford.edu/projects/glove/).	•dimension - 200	Converted to w2v format with python -m gensim.scripts.glove2word2vec -i <fname> -o glove-wiki-gigaword-200.txt.
glove-wiki-gigaword-300	400000	376 MB	Wikipedia 2014 + Gigaword 5 (6B tokens, uncased)	Pre-trained vectors based on Wikipedia 2014 + Gigaword, 5.6B tokens, 400K vocab, uncased (https://nlp.stanford.edu/projects/glove/).	•dimension - 300	Converted to w2v format with python -m gensim.scripts.glove2word2vec -i <fname> -o glove-wiki-gigaword-300.txt.
glove-wiki-gigaword-50	400000	65 MB	Wikipedia 2014 + Gigaword 5 (6B tokens, uncased)	Pre-trained vectors based on Wikipedia 2014 + Gigaword, 5.6B tokens, 400K vocab, uncased (https://nlp.stanford.edu/projects/glove/).	•dimension - 50	Converted to w2v format with python -m gensim.scripts.glove2word2vec -i <fname> -o glove-wiki-gigaword-50.txt.
word2vec-google-news-300	3000000	1662 MB	Google News (about 100 billion words)	Pre-trained vectors trained on a part of the Google News dataset (about 100 billion words). The model contains 300-dimensional vectors for 3 million words and phrases. The phrases were obtained using a simple data-driven approach described in 'Distributed Representations of Words and Phrases and their Compositionality' (https://code.google.com/archive/p/word2vec/).	•dimension - 300	-
word2vec-ruscorpora-300	184973	198 MB	Russian National Corpus (about 250M words)	Word2vec Continuous Skipgram vectors trained on full Russian National Corpus (about 250M words). The model contains 185K words.	•dimension - 300 •window_size - 10	The corpus was lemmatized and tagged with Universal PoS

Gensim Word2Vec Manual & Pre-trained Models

```
>>> model = Word2Vec(sent, min_count=1,size= 50,workers=3, window =3,
sg = 1)
```

Let's try to understand the hyperparameters of this model.

size: The number of dimensions of the embeddings and the default is 100.

window: The maximum distance between a target word and words around the target word. The default window is 5.

min_count: The minimum count of words to consider when training the model; words with occurrence less than this count will be ignored. The default for min_count is 5.

workers: The number of partitions during training and the default workers is 3.

sg: The training algorithm, either CBOW(0) or skip gram(1). The default training algorithm is CBOW.

```
# Download the "word2vec-google-news-300" embeddings
w2v_vectors = gensim.downloader.load('word2vec-google-news-300')
```

```
[==================================================] 100.0% 1662.8/1662.8MB downloaded
```

```
# calculate: (king - man) + woman = ?
result = w2v_vectors.most_similar(positive=['woman', 'king'], negative=['man'], topn=1)
print(result)
```

```
[('queen', 0.7118193507194519)]
```

GloVe

To generate glove embeddings using the GloVe algorithm, you will need to follow these steps:

Step - 1: Collect a large dataset of words and their co-occurrences. This can be a text corpus, such as documents or web pages.
Step - 2: Preprocess the dataset by tokenizing the text into individual words and filtering out rare or irrelevant words.
Step - 3: Construct a co-occurrence matrix that counts the number of times each word appears in the same context as every other word.
Step - 4: Use the co-occurrence matrix to compute the word embeddings using the GloVe algorithm. This involves training a model to minimize the error between the word vectors' dot product and the co-occurrence counts' logarithm.
Step - 5: Save the resulting word embeddings to a file or use them directly in your model.

The links below contain word vectors obtained from the respective corpora. If you want word vectors trained on massive web datasets, you need only download one of these text files! Pre-trained word vectors are made available under the Public Domain Dedication and License.

•Common Crawl (42B tokens, 1.9M vocab, uncased, 300d vectors, 1.75 GB download): glove.42B.300d.zip [mirror]
•Common Crawl (840B tokens, 2.2M vocab, cased, 300d vectors, 2.03 GB download): glove.840B.300d.zip [mirror]
•Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab, uncased, 300d vectors, 822 MB download): glove.6B.zip [mirror]
•Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased, 200d vectors, 1.42 GB download): glove.twitter.27B.zip [mirror]

Pros & Cons

Both Glove and Word2vec are word-based models - that is the models take as input words and output word embeddings.

Representing input as subwords as opposed to words has become the latest trend because it strikes a balance between character based and word-based representations - *the most important benefit being avoidance of OOV (out of vocabulary) cases which the other two models (Glove, Word2vec) mentioned in the question suffer from.*

Model Name	Context Sensitive embeddings	Learnt representations
Word2vec	No	Words
Glove	No	Words
ELMo	Yes	Words
BERT	Yes	Subwords

Lab -5

Exercise 1: Reading with Code
https://towardsdatascience.com/first-neural-network-for-beginners-explained-with-code-4cfd37e06eaf

Exercise 2: https://towardsdatascience.com/introduction-to-word2vec-skip-gram-cb3e9533bcf1

Exercise 3: https://www.scaler.com/topics/nlp/glove-embeddings/

Mid - Term

Date: Feb 20 & Feb 22
Conducted – Virtual, from 12PM to 11.59PM
Individual Submissions
Coding Exercise
Topic Focus: Hidden Markov Models, NER, Part-of-speech tagging, Word2Vec and GloVe