Introduction to Language Model in NLP

Language Model

- A language model is a probabilistic statistical model that estimates the probability that a given word sequence will occur in a sentence based on words that have come before it.
- It is a probability distribution over sequence of words.
- A language model learns to predict the probability of a sequence of words.
- Word ordering
  - P(The cat is sat on the mat) > P(mat on is the sat cat the)
- This ability to model the rules of a language as a probability gives great power for NLP related tasks.

Types of Language Model

- Statistical Language Models:
  - Use traditional statistical techniques like N-grams, Hidden Markov Models (HMM) and certain linguistic rules to learn the probability distribution of words

- **Neural Language Models:**
  - **Use different kinds of Neural Networks to model language**

Application of Language Model

- Speech recognition :
  - “I ate a cherry” is a more likely sentence than “Eye eight uh Jerry”
- Machine translation:
  - More likely sentences are probably better translations.
- Spelling Correction:
  - Their are problems wit this sentence.”
- Optical Character Recognition:
  - More probable sentences are more likely correct readings
- Handwriting recognition:
  - More probable sentences are more likely correct readings
- Information retrieval:
  - More likely words are more likely retrieved.
- Completion Prediction:
  - Predictive text input systems can guess what you are typing and give choices on how to complete it.

N-Gram Language Model

- Unigram, Bigram Trigram etc.
- An N-gram is a sequence of N tokens (or words).
- Example:
- “I love reading books about Natural language processing”
- Unigram:
  - It is a one-word sequence.
  - “I”, “love”, “reading”, “books”, “about” ,“Natural”, “language” ,“processing”
- Bigram:
  - Two word sequence
  - “ I Love” “love reading” , “reading books”
- Trigram:
  - Three word sequence:
  - “I love reading”, “love reading books”, “reading books about”

N-gram Model

- An N-gram language model predicts the probability of a given N-gram within any sequence of words in the language.
- Predict $P(w \mid h)$
  - What is the probability of seeing the word w given a history of previous words h where the history contains n-1 words.
  - Estimate this probability to construct an N-gram model

- Steps to Compute P(w\h):
- Apply the chain rule of probability
  - It tells how to compute the joint probability of a sequence by using the conditional probability of a word given previous words.
  - p(w1...ws) = p(w1) . p(w2 | w1) . p(w3 | w1 w2) . p(w4 | w1 w2 w3) ..... p(wn | w1...wn-1)
- We then apply a very strong simplification assumption to allow us to compute p(w1…ws) in an easy manner
  - Here , we can assume for all conditions, that:
  - p(wk | w1...wk-1) = p(wk | wk-1)
  - we approximate the history (the context) of the word wk by looking only at the last word of the context.
  - This assumption is called the Markov assumption and it is same as bigram model.
  - The Markov assumption is the presumption that the future behavior of a dynamical system only depends on its recent history

Generally in practical applications, Bi-gram(previous one word), Tri-gram(previous two word, Four-gram (previous three word) are used.

Unigram(1-gram): No history is used.

"about five minutes from....."

Assume in corpus dinner word is present with highest probability.

Unigram doesn't take into account probabilities with previous words like from, minutes.

Unigram will predict dinner.

"about five minutes from **dinner**"

- N-gram conditional probabilities can be estimated from raw text based on the ***relative frequency*** of word sequences.

Bigram: $$P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

N-gram: $$P(w_n \mid w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}w_n)}{C(w_{n-N+1}^{n-1})}$$

- To have a consistent probabilistic model, append a unique start (<s>) and end (</s>) symbol to every sentence and treat these as additional words.

Bi-gram(2-gram): **One word history**

$$P(w_1, w_2) = \prod_{i=2} P(w_2 \mid w_1) \qquad P(w_i \mid w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

"about five minutes from....."

Assumption: Next word may be college, class

$$P(college \mid about\ \ five\ \ minutes\ \ from) = \frac{count(about\ \ five\ \ minutes\ \ from\ \ college)}{count(about\ \ five\ \ minutes\ \ from)}$$

$$P(class \mid about\ \ five\ \ minutes\ \ from) = \frac{count(about\ \ five\ \ minutes\ \ from\ \ class)}{count(about\ \ five\ \ minutes\ \ from)}$$

"about five minutes from....."

Count(about five minutes from)= P(about|<S>) × P(five| about) × P(minutes | five) × P(from| minutes)

Count(about five minutes from **college**)= P(about|<S>) × P(five| about) × P(minutes | five) × P(from| minutes) × P(college| from)

Count(about five minutes from **class**)= P(about|<S>) × P(five| about) × P(minutes | five) × P(from| minutes) × P(class| from)

$$P(\text{college} \mid \text{about five minutes from}) = \frac{\text{count(about five minutes from college)}}{\text{count(about five minutes from)}}$$

=P(college| from)

$$P(\text{class} \mid \text{about five minutes from}) = \frac{\text{count(about five minutes from class)}}{\text{count(about five minutes from)}}$$

=P(class| from)

Tri-gram(2-gram): **Two words history**

$$P(w_1, w_2, w_3) = \prod_{i=3} P(w_3 \mid w_1, w_2) \qquad P(w_i \mid w_{i-1}, w_{i-2}) = \frac{\text{count}(w_{i-2}, w_{i-1}, w_i)}{\text{count}(w_{i-2}, w_{i-1})}$$

Count(about five minutes from)= P(five|<S>, about) × P(minutes| about, five) × P(from| five , minutes)

Count(about five minutes from **college**)= P(five|<S>, about) × P(minutes| about, five) × P(from| five , minutes) × P(college| minutes from)

Count(about five minutes from **class**)= P(five|<S>, about) × P(minutes| about, five) × P(from| five , minutes) × P(class| minutes from)

$$P(\text{college} \mid \text{about five minutes from}) = \frac{\text{count(about five minutes from college)}}{\text{count(about five minutes from)}}$$

=P(college| minutes from)

$$P(\text{class} \mid \text{about five minutes from}) = \frac{\text{count(about five minutes from class)}}{\text{count(about five minutes from)}}$$

=P(class| minutes from)

Four-gram(4-gram): **Three words history**

$$P(w_1, w_2, w_3, w_4) = \prod_{i=4} P(w_4 \mid w_1, w_2, w_3)$$

$$P(w_i \mid w_{i-1}, w_{i-2}, w_{i-3}) = \frac{\text{count}(w_{i-3}, w_{i-2}, w_{i-1}, w_i)}{\text{count}(w_{i-3}, w_{i-2}, w_{i-1})}$$

Count(about five minutes from)= P(minutes | <S>, about, five) × P(from | about, five, minutes)

Count(about five minutes from **college**)= P(minutes | <S>, about, five) ×
P(from | about, five, minutes) ×
P(college | five , minutes, from)

Count(about five minutes from **class**)= P(minutes | <S>, about, five) ×
P(from | about, five, minutes) ×
P(class | five , minutes, from)

$$P(\text{college} \mid \text{about five minutes from}) = \frac{\text{count(about five minutes from college)}}{\text{count(about five minutes from)}}$$

=P(college | five minutes from)

$$P(\text{class} \mid \text{about five minutes from}) = \frac{\text{count(about five minutes from class)}}{\text{count(about five minutes from)}}$$

=P(college | five minutes from)

As no. of previous state (history) increases, it is very difficult to match that set of words in corpus.

Probabilities of **larger collection of word is minimum.** To overcome this problem, Bi-gram model is used

Example

Exercise 1: Estimating Bi-gram probabilities

What is the most probable next word predicted by the model for the following word sequence?

Given Corpus

<S> I am Henry </S>

<S> I like college </S>

<S> Do Henry like college </S>

<S> Henry I am </S>

<S> Do I like Henry </S>

<S> Do I like college </S>

<S> I do like Henry </S>

Word	Frequency
<S>	7
</S>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

1) <S> Do ?

<S> I am Henry </S>
<S> I like college </S>
<S> Do Henry like college </S>
<S> Henry I am </S>
<S> Do I like Henry </S>
<S> Do I like college </S>
<S> I do like Henry </S>

Word	Frequency
<S>	7
</S>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

Next word prediction probability W_{i-1}=do

Next word	Probability Next Word= $\frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$
P(</S> | do)	0/4
P(<I> | do)	**2/4**
P(<am>| do)	0/4
P(<Henry>| do)	1/4
P(<like | do)	1/4
P(<college | do)	0/4
P(do | do)	0/4

I is more probable

2) <S> I like Henry ?

<S> I am Henry </S>
<S> I like college </S>
<S> Do Henry like college </S>
<S> Henry I am </S>
<S> Do I like Henry </S>
<S> Do I like college </S>
<S> I do like Henry </S>

Word	Frequency
<S>	7
</S>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

Next word prediction probability W_{i-1}=Henry

Next word	Probability Next Word= $\frac{N}{D} = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$
P(</S> | Henry)	**3/5**
P(<I> | Henry)	1/5
P(<am>| Henry)	0
P(<Henry>| Henry)	0
P(<like | Henry)	1/5
P(<college | Henry)	0
P(do | Henry)	0

</S> is more probable

3) <S> Do I like ?

Use Tri-gram

P<I like>=3

<S> I am Henry </S>
<S> I like college </S>
<S> Do Henry like college </S>
<S> Henry I am </S>
<S> Do I like Henry </S>
<S> Do I like college </S>
<S> I do like Henry </S>

Next word prediction probability

W_{i-2}=I and W_{i-1}=like

Next word	Probability Next Word= $\frac{\text{count}(w_{i-2}, w_{i-1}, w_i)}{\text{count}(w_{i-2}, w_{i-1})}$
P(</S> | I like)	0/3
P(<I> | I like)	0/3
P(<am>| I like)	0/3
P(<Henry>| I like)	1/3
P(<like | I like)	0/3
P(<college | I like)	2/3
P(do | I like)	0/3

College is probable

4) <S> Do I like college ?

Use Four-gram

<S> I am Henry </S>
<S> I like college </S>
<S> Do Henry like college </S>
<S> Henry I am </S>
<S> Do I like Henry </S>
<S> Do I like college </S>
<S> I do like Henry </S>

Next word prediction probability

W_{i-3}=I, W_{i-2}=like W_{i-1}=college

Next word	Probability Next Word= $\frac{\text{count}(w_{i-3}, w_{i-2}, w_{i-1}, w_i)}{\text{count}(w_{i-3}, w_{i-2}, w_{i-1})}$
P(</S> | I like college)	2/2
P(<I> | I like college)	0/2
P(<am>| I like college)	0/2
P(<Henry>| I like college)	0/2
P(<like | I like college)	0/2
P(<college | I like college)	0/2
P(do | I like college)	0/2

</S> is more probable

Which of the following sentence is better. i.e. Gets a higher probability with this model. Use Bi-gram

<S> I am Henry </S>
<S> I like college </S>
<S> Do Henry like college </S>
<S> Henry I am </S>
<S> Do I like Henry </S>
<S> Do I like college </S>
<S> I do like Henry </S>

Word	Frequency
<S>	7
</S>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

1. <S> I like college </S>

<S> like college </S>=?

=P(I| <S>) × P(like | I) × P(college| like) × P(</S> | college)

=3/7 × 3/6 × 3/5 ×3/3 = 9/70=0.13

2. <S> Do I like Henry </S>

=P(do | <S>) × P(I | do) × P(like | I) × P(Henry | like) × P(</S> | Henry)

=3/7 × 2/4 × 3/6 ×2/5 ×3/5 = 9/350=0.0257

ANS: First statement is more probable

Evaluation of Language Models

- Extrinsic Evaluation:
  - Evaluating systems outputs based on their impact on the performance of other NLP systems.
  - Evaluate use of model in end application
  - Can be slow to compute performance
  - Unclear if subsystem is the problem, other subsystems, or internal interactions
  - If replacing subsystem improves performance, the change is likely good
  - Choose extrinsic evaluation when you need to measure the model's effectiveness in real-world applications or when assessing its contribution to achieving broader goals.

Evaluation of Language Models…

- Intrinsic Evaluation:
  - Quality of NLP systems outputs is evaluated against pre-determined ground truth (reference text)
  - Evaluate on ability to model test corpus
  - Evaluate the performance of model independent of any application.
  - Fast to compute performance
  - Needs test set and training set.
  - Needs positive correlation with real
  - task to determine usefulness
  - Use intrinsic evaluation when you want to fine-tune and assess the performance of individual NLP components or when benchmarking against specific tasks.

Perplexity

- Perplexity is the standard metric for measuring quality of a language model.
- It is the probability of test set normalized by the number of words.
- Perplexity of language model on test set is the inverse probability of test set normalized by the number of words.
- Higher the conditional probability of word sequence, lower the perplexity.
- The perplexity of two language model is only comparable if they uses identical vocabularies.
- Lower perplexity of language model is better of the word in test set.

Multinomial Naive Bayes to NLP Problems

- It is a popular machine learning algorithm for text classification problems in Natural Language Processing (NLP).
- It is particularly useful for problems that involve text data with discrete features such as word frequency counts.
- It works on the principle of Bayes theorem and assumes that the features are conditionally independent given the class variable

- Steps for applying Multinomial Naive Bayes to NLP problems:
- Preprocessing the text data:
  - This steps involves steps such as tokenization, stop-word removal, stemming, and lemmatization.
- Feature extraction:
  - The text data needs to be converted into a feature vector format that can be used as input to the MNB algorithm.
- Splitting the data:
  - The data needs to be split into training and testing sets.
  - The training set is used to train the MNB model, while the testing set is used to evaluate its performance.

- Training the MNB model:
  - The MNB model is trained on the training set by estimating the probabilities of each feature given each class.
  - This involves calculating the prior probabilities of each class and the likelihood of each feature given each class.
- Using the model to make predictions:
  - Once the model is trained, it can be used to make predictions on new text data.
  - The text data is preprocessed and transformed into the feature vector format, which is then input to the trained model to obtain the predicted class label.
- Evaluating the performance of the model:
  - The performance of the model is evaluated using metrics such as accuracy, precision, recall, and F1-score on the testing set.

- Naive Bayes Classifier :
- Bayes theorem calculates probability P(c|x) where c is the class of the possible outcomes and x is the given instance which has to be classified, representing some certain features.
- P(c|x) = P(x|c) * P(c) / P(x)
- Naive Bayes are mostly used in natural language processing (NLP) problems.
- Naive Bayes predict the tag of a text. They calculate the probability of each tag for a given text and then output the tag with the highest one.

Example

Text	Reviews
“I liked the movie”	positive
“It’s a good movie. Nice story”	positive
“Nice songs. But sadly boring ending. ”	negative
“Hero’s acting is bad but heroine looks good. Overall nice movie”	positive
“Sad, boring movie”	negative

We classify whether the text “overall liked the movie” has a positive review or a negative review.

- P(positive | overall liked the movie) — the probability that the tag of a sentence is positive given that the sentence is “overall liked the movie”.
- P(negative | overall liked the movie) — the probability that the tag of a sentence is negative given that the sentence is “overall liked the movie”.

- apply Removing Stopwords and Stemming in the text.
- Removing Stopwords: These are common words that don’t really add anything to the classification, such as an able, either, else, ever and so on.
- Stemming: Stemming to take out the root of the word.
- Now After applying these two techniques, our text becomes

Text	Reviews
“ilikedthemovi”	positive
“itsagoodmovienicestori”	positive
“nicesongsbutsadlyboringend”	negative
“herosactingisbadbutheroinelooksgoodoverallnicemovi”	positive
“sadboringmovi”	negative

- P(positive | overall liked the movie) = P(overall liked the movie | positive) * P(positive) / P(overall liked the movie)
- Since for our classifier we have to find out which tag has a bigger probability, we can discard the divisor which is the same for both tags.
- There’s a problem though: “overall liked the movie” doesn’t appear in our training dataset, so the probability is zero. Here, we assume the ‘naive’ condition that every word in a sentence is independent of the other ones.
- P(overall liked the movie) = P(overall) * P(liked) * P(the) * P(movie)

- P(overall liked the movie| positive) = P(overall | positive) * P(liked | positive) * P(the | positive) * P(movie | positive)
- P(positive) is 3/5
- P(negative) is 2/5.
- P(overall | positive) means counting how many times the word “overall” appears in positive texts (1) divided by the total number of words in positive (17).
- P(overall | positive) = 1/17
- P(liked/positive) = 1/17,
- P(the/positive) = 2/17,
- P(movie/positive) = 3/17.

- If probability comes out to be zero then By using Laplace smoothing: we add 1 to every count so it’s never zero.
- To balance this, we add the number of possible words to the divisor, so the division will never be greater than 1.
- In our case, the total possible words count are 21.

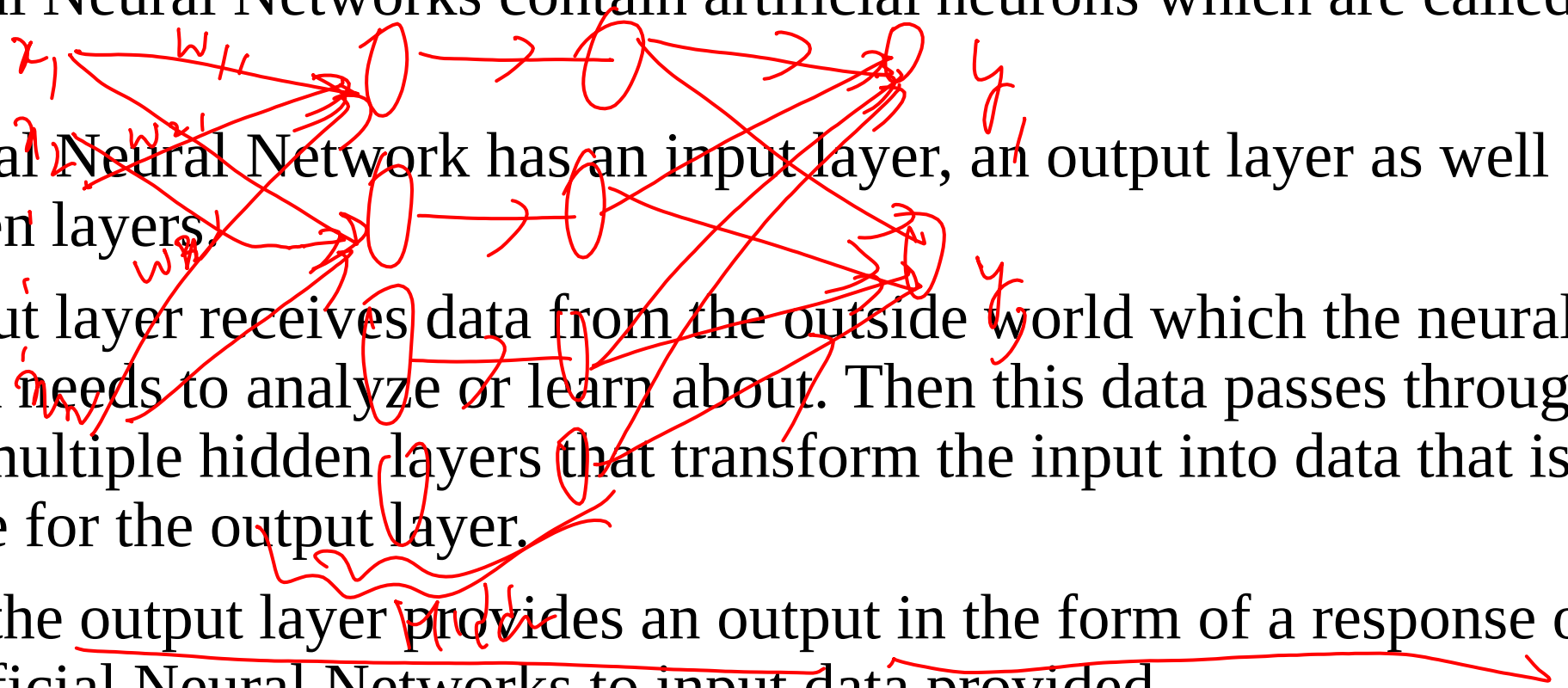
Word	P(word | positive)	P(word | negative)
overall	(1 + 1)/(17 + 21)	(0 + 1)/(7 + 21)
liked	(1 + 1)/(17 + 21)	(0 + 1)/(7 + 21)
the	(2 + 1)/(17 + 21)	(0 + 1)/(7 + 21)
movie	(3 + 1)/(17 + 21)	(1 + 1)/(7 + 21)

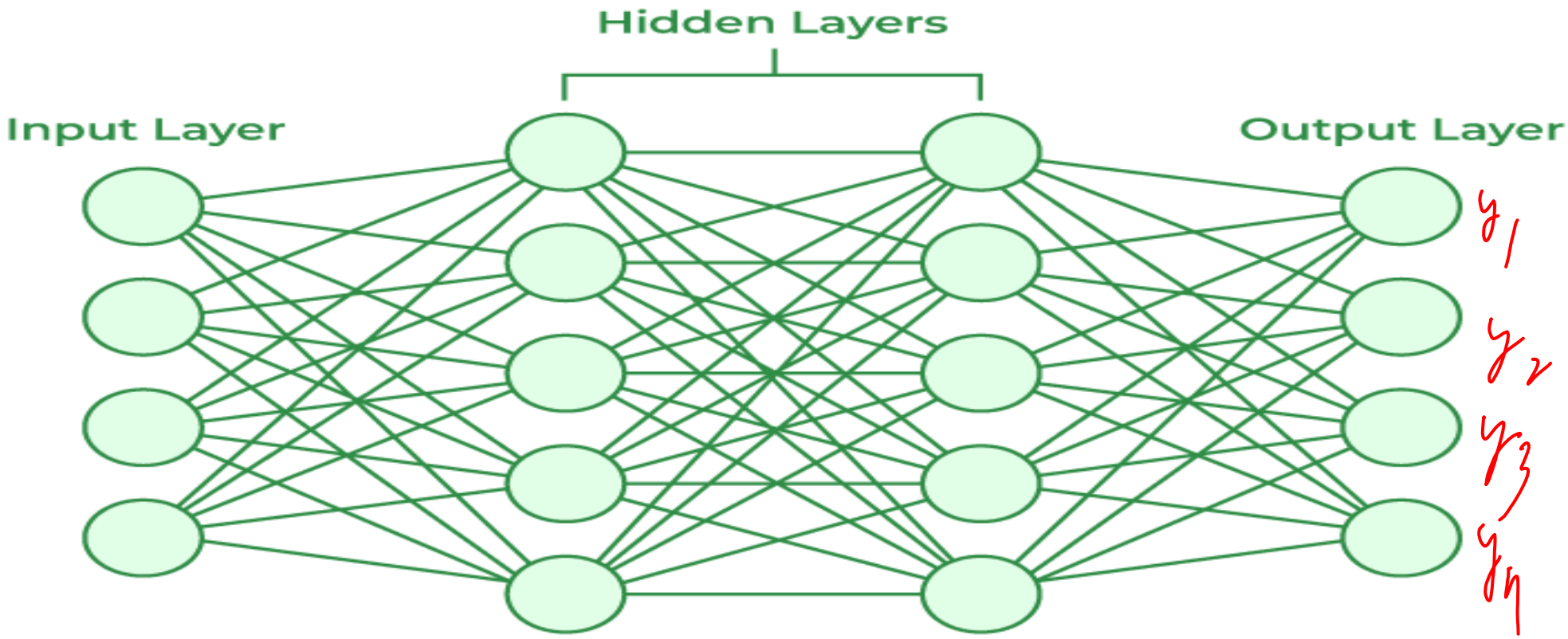
P(overall | positive) * P(liked | positive) * P(the | positive) * P(movie | positive) * P(positive)
= 1.38 * 10^{-5} = 0.0000138
P(overall | negative) * P(liked | negative) * P(the | negative) * P(movie | negative) * P(negative)
= 0.13 * 10^{-5} = 0.0000013

Results: classifier gives “overall liked the movie” the positive tag.

Artificial Neural Networks

- Artificial Neural Networks contain artificial neurons which are called units.
- Artificial Neural Network has an input layer, an output layer as well as hidden layers.
- The input layer receives data from the outside world which the neural network needs to analyze or learn about. Then this data passes through one or multiple hidden layers that transform the input into data that is valuable for the output layer.
- Finally,the output layer provides an output in the form of a response of the Artificial Neural Networks to input data provided.

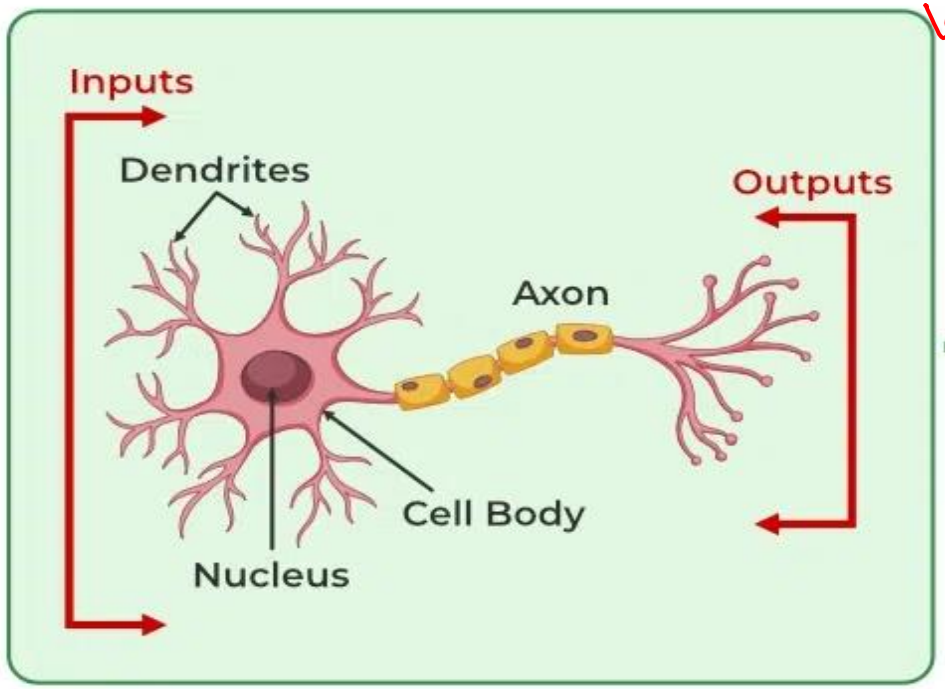



Hidden Layers
Input Layer
Output Layer
y_1
y_2
y_3
y_4

$$f\left((x_1w_1 + x_2w_2 + x_3w_3 + \cdots x_nw_n) + b\right)$$

bias

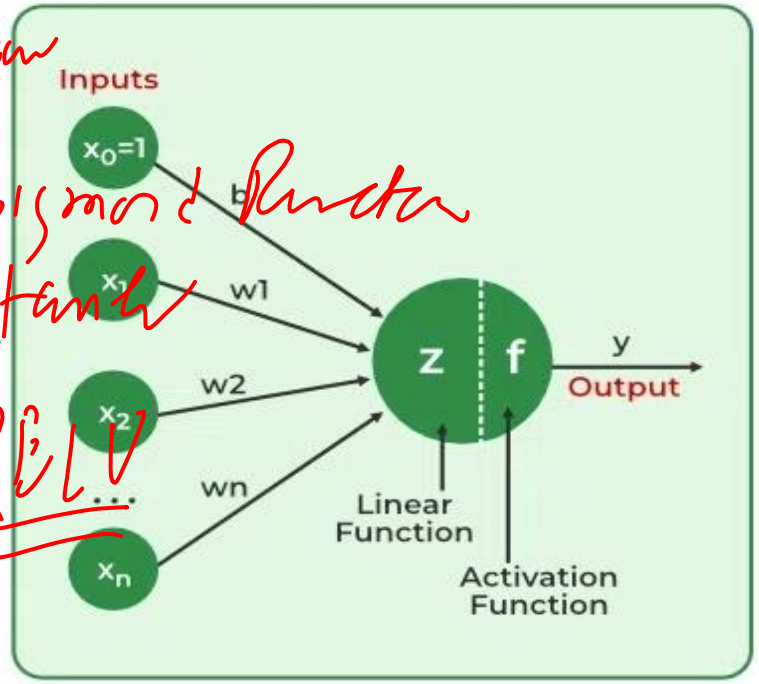
Activation



⇒ Sigmoid Function

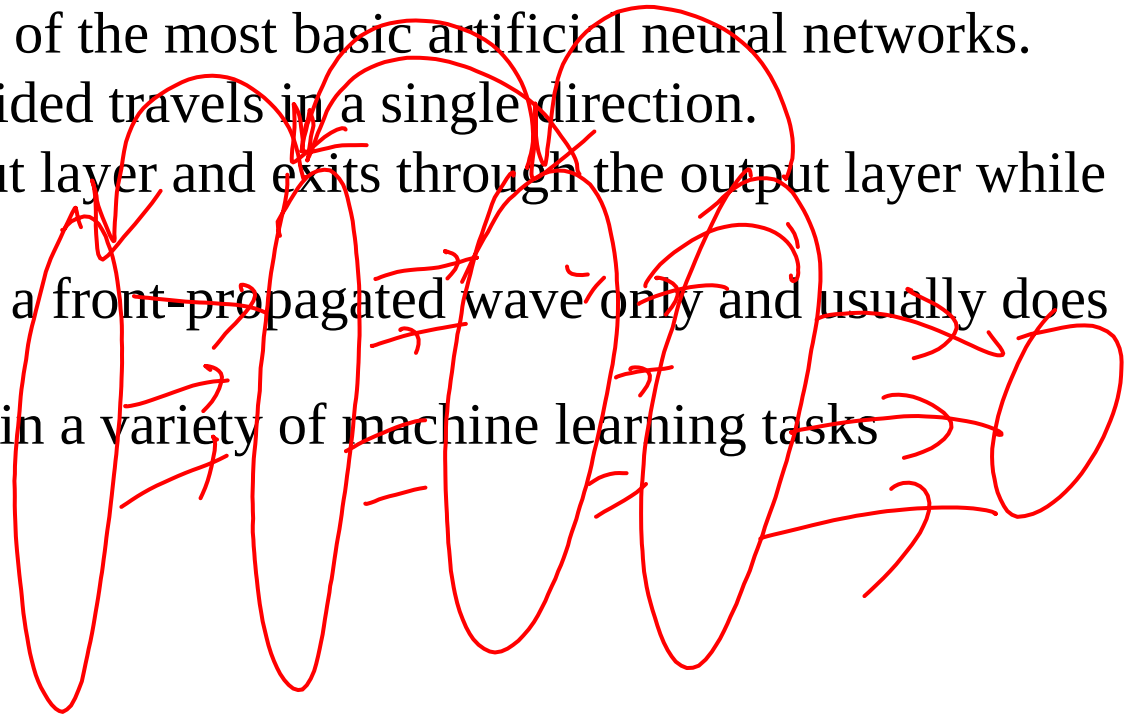
⇒ tanh

⇒ ReLU

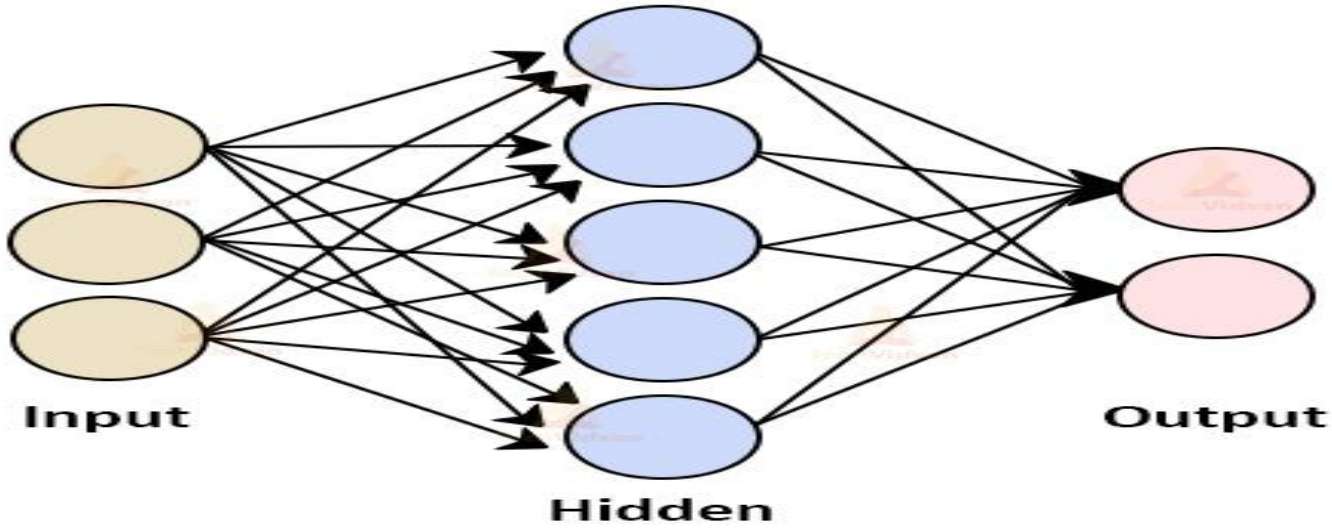


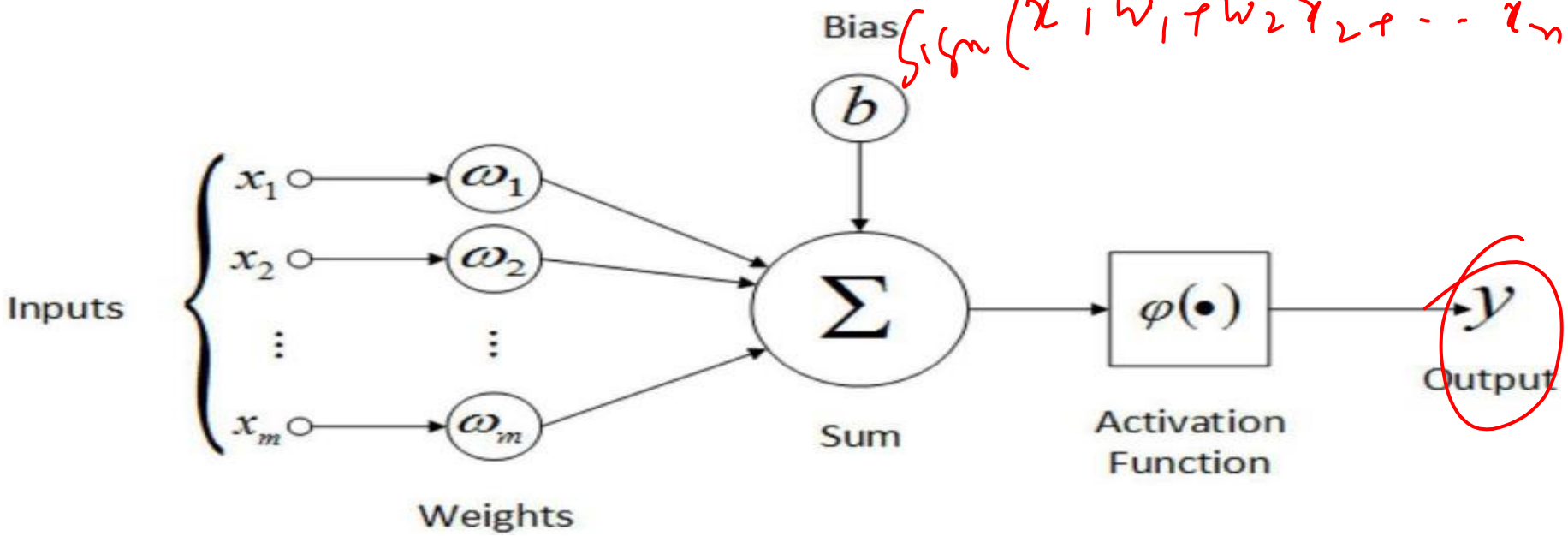
Types of Artificial Neural Networks?

- Feedforward Neural Network:
  - The feedforward neural network is one of the most basic artificial neural networks.
  - In this ANN, the data or the input provided travels in a single direction.
  - It enters into the ANN through the input layer and exits through the output layer while hidden layers may or may not exist.
  - So the feedforward neural network has a front-propagated wave only and usually does not have backpropagation.
  - Feedforward neural networks are used in a variety of machine learning tasks including:
    - Pattern recognition
    - Classification tasks
    - Regression analysis
    - Image recognition
    - Time series prediction



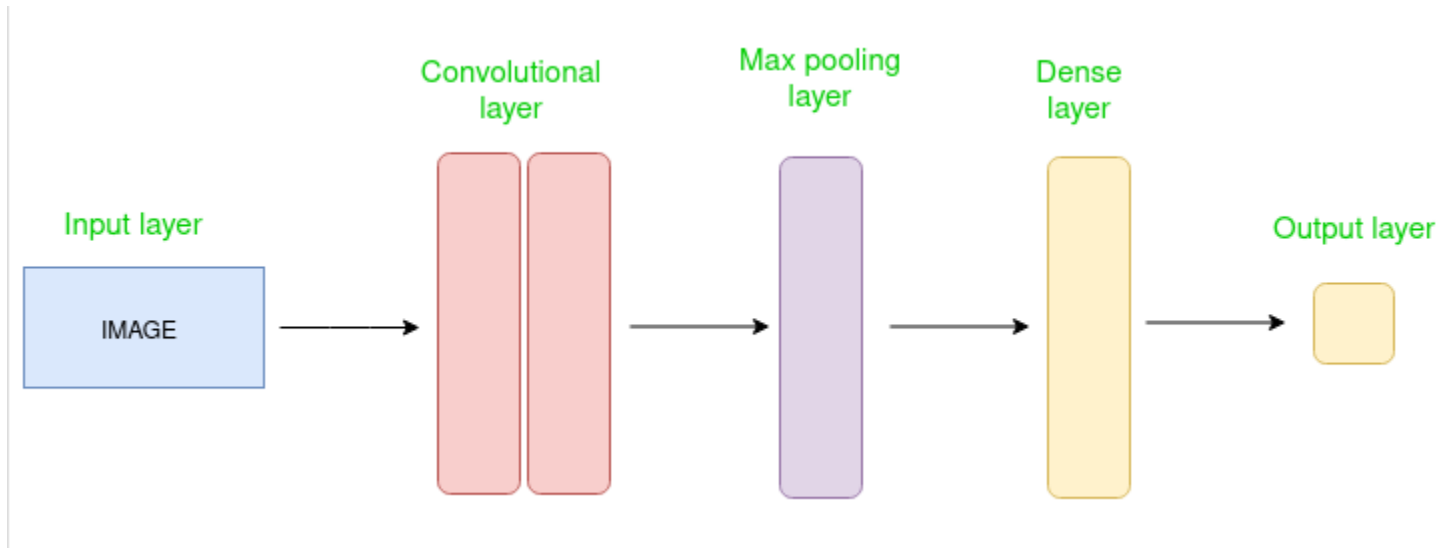
Architecture of Artificial Neural Network



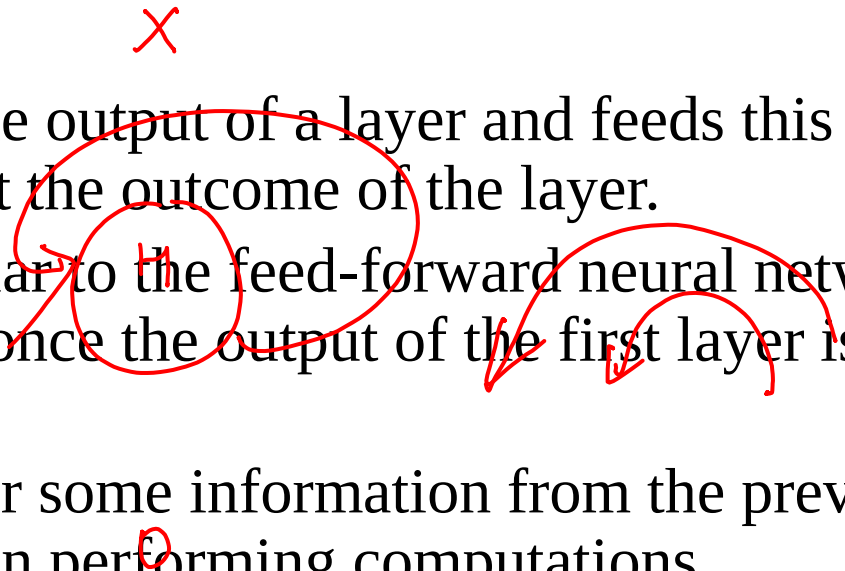


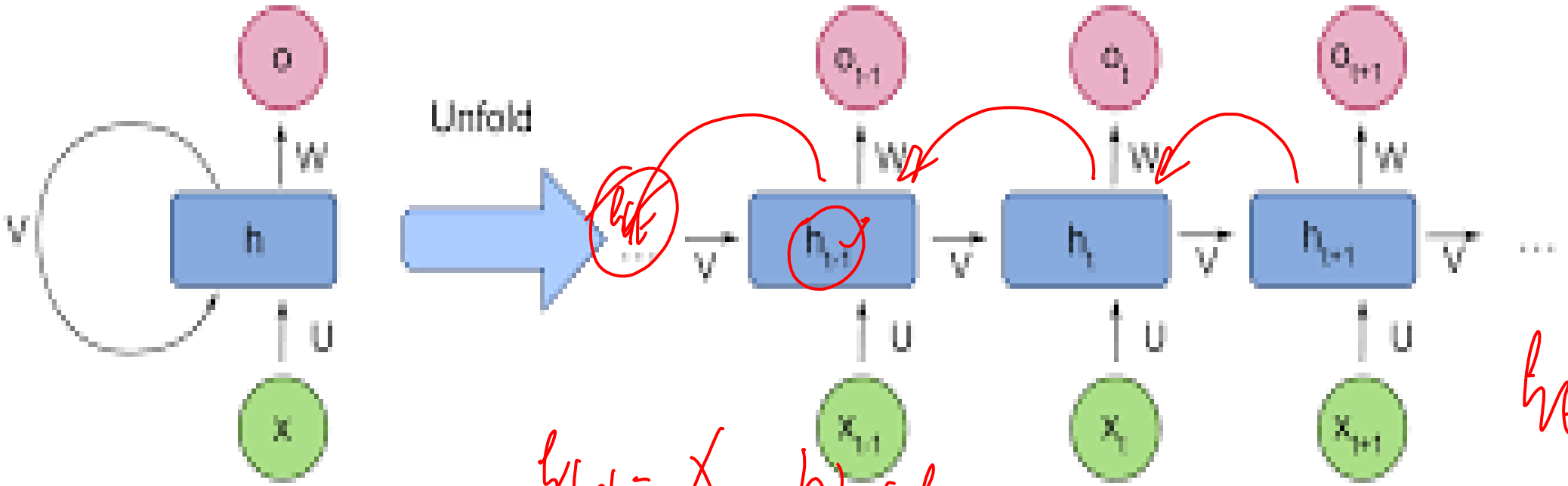
$\text{sigm}(x_1 w_1 + w_2 x_2 + \cdots x_n w_n) + b$

- Convolutional Neural Network:
  - A Convolutional neural network has some similarities to the feed-forward neural network, where the connections between units have weights that determine the influence of one unit on another unit.
  - But a CNN has one or more than one convolutional layer that uses a convolution operation on the input and then passes the result obtained in the form of output to the next layer.
  - CNN has applications in speech and image processing which is particularly useful in computer vision.


Input layer
IMAGE
Convolutional layer
Max pooling layer
Dense layer
Output layer

- Recurrent Neural Network:
  - The Recurrent Neural Network saves the output of a layer and feeds this output back to the input to better predict the outcome of the layer.
  - The first layer in the RNN is quite similar to the feed-forward neural network and the recurrent neural network starts once the output of the first layer is computed.
  - After this layer, each unit will remember some information from the previous step so that it can act as a memory cell in performing computations.





$h_{t+1} = f(x_{t+1} W_{inf} + h_t)$

$o_{t+1} = h_{t+1} W_o$

h_{t+2}

$= x_2 W_{in} + h_{t+1}$

Generative Adversarial Network

- powerful class of neural networks that are used for an unsupervised learning.
- GANs are made up of two neural networks, a discriminator and a generator.
- They use adversarial training to produce artificial data that is identical to actual data.

- Generative Adversarial Networks (GANs) can be broken down into three parts:
  - Generative: To learn a generative model, which describes how data is generated in terms of a probabilistic model.
  - Adversarial: The word adversarial refers to setting one thing up against another. This means that, in the context of GANs, the generative result is compared with the actual images in the data set. A mechanism known as a discriminator is used to apply a model that attempts to distinguish between real and fake images.
  - Networks: Use deep neural networks as artificial intelligence (AI) algorithms for training purposes.

Real images
Sample
Generator
Sample
Discriminator
Discriminator loss
Generator loss