Modern Information Retrieval

Neural information retrieval

Hamid Beigy

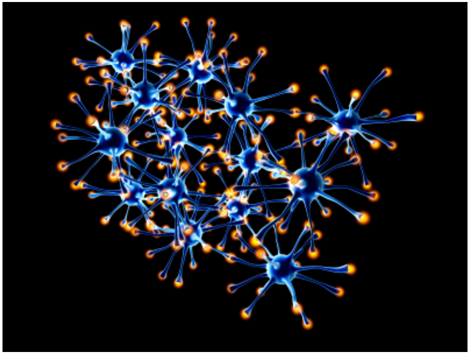
Sharif university of technology

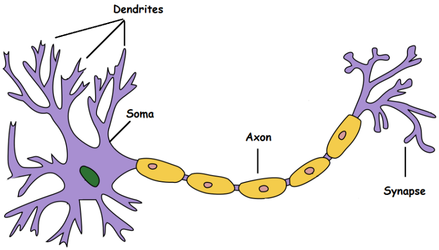
May 25, 2025

Table of contents

Introduction

1. Brain is a network of simple elements called neuron.

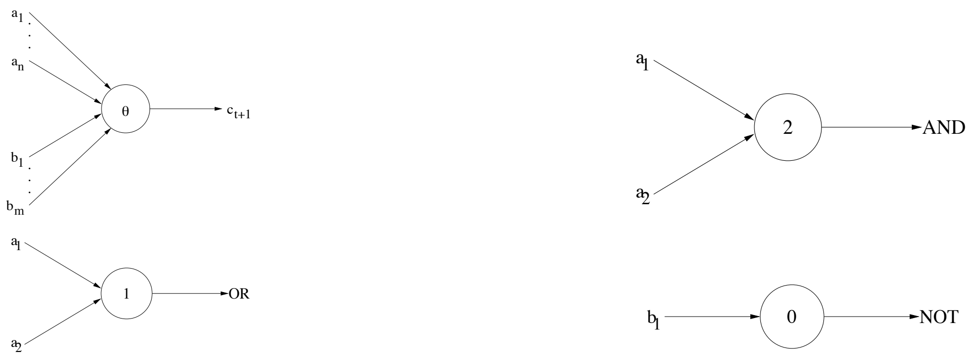




2. The idea of neural networks began as a model of how neurons in the brain function and used connected circuits to simulate intelligent behavior.

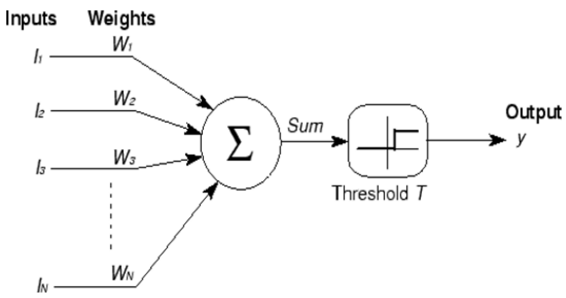
McCulloch and Pitts network (1943)

1. The first model of a neuron was invented by McCulloch (physiologists) and Pitts (logician).
2. This neuron has two types of binary inputs:
   - excitatory inputs (shown by *a*)
   - Inhibitory inputs(shown by *b*)
3. The output is binary: fires (1) and not fires (0).
4. Until sum of inputs is less than a certain threshold level, output remains zero.



Perceptron (Frank Rosenblat (1958))

1. McCulloch and Pitts neurons can not learn.



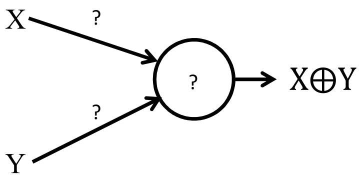
2. Let y be the correct output, and $f(x)$ the output function of the network. Perceptron updates weights.

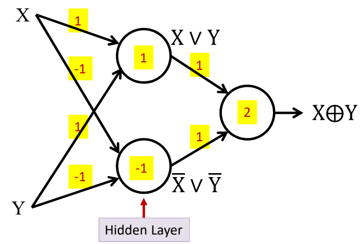
$$w_j^{(t)} \leftarrow w_j^{(t)} + \alpha x_j (y - f(x))$$

3. McCulloch and Pitts' neuron is a better model for the electrochemical process inside the neuron than the Perceptron.

4. But Perceptron is the basis and building block for the modern neural networks.

1. Minsky and Papert published their book Perceptron.
2. The book shows that Perceptron could only solve linearly separable problems.
3. They showed that it is not possible for Perceptron to learn an XOR function.

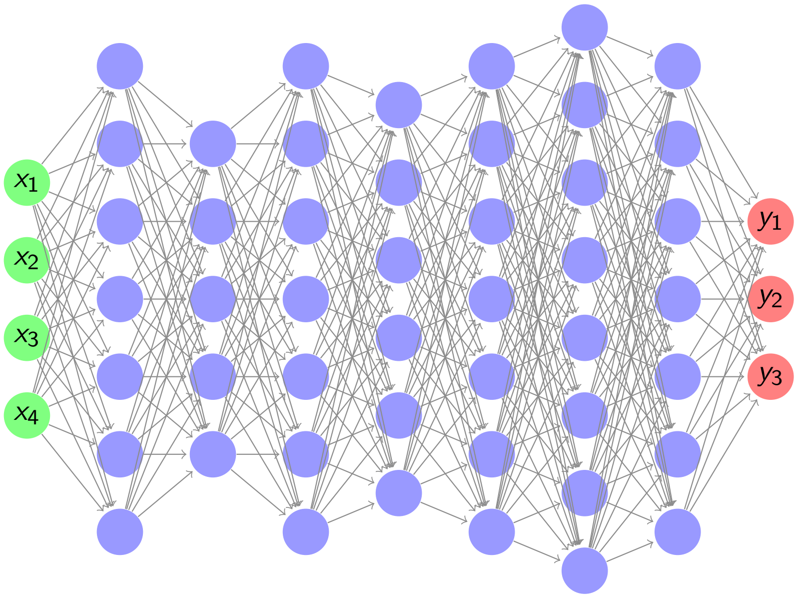




4. The right network is called multilayer Perceptrons (MLP) network.
5. How do you learn a MLP network?
6. How many hidden layers do use in a MLP network?
7. How many hidden units in each hidden layer do use in a MLP network?

Gradient based learning

1. How do you learn the following Multilayer Perceptrons?

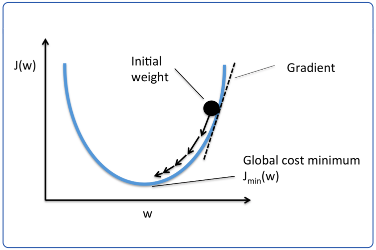


2. What are deep networks and shallow networks?

1. The goal of machine learning algorithms is to construct a model that can be used to estimate y based on x.
2. Let the model be in form of
$$h(x) = w_0 + w_1 x$$
3. The goal of creating a model is to choose parameters so that $h(x)$ is close to y for the training data, x and y.
4. We need a function that will minimize the parameters over our dataset. A function that is often used is mean squared error,
$$J(w) = \frac{1}{2m} \sum_{i=1}^{m} (h(x_i) - y_i)^2$$
5. How do we find the minimum value of cost function?

Gradient descent

1. Gradient descent is by far the most popular optimization strategy, used in machine learning and deep learning at the moment.
2. Cost (error) is a function of the weights (parameters).
3. We want to reduce/minimize the error.
4. Gradient descent: move towards the error minimum.
5. Compute gradient, which implies get direction to the error minimum.
6. Adjust weights towards direction of lower error.



Gradient descent (Linear Regression)

1. We have the following hypothesis and we need fit to the training data

$$h(x) = w_0 + w_1 x$$

2. We use a cost function such Mean Squared Error

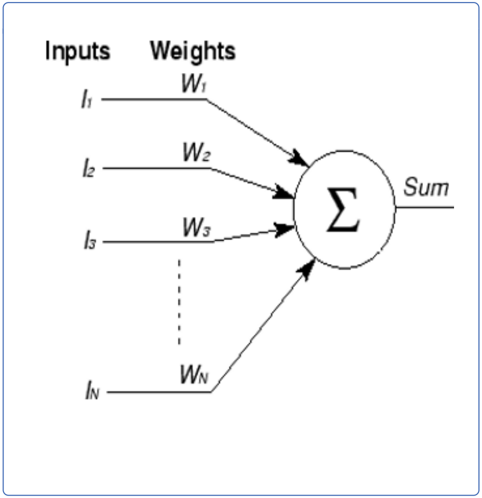
$$J(w) = \frac{1}{2m} \sum_{i=1}^{m} (h(x_i) - y_i)^2$$

3. This cost function can be minimized using gradient descent.

$$w_0^{(t+1)} = w_0^{(t)} - \alpha \frac{\partial J(w^{(t)})}{\partial w_0}$$
$$w_1^{(t+1)} = w_1^{(t)} - \alpha \frac{\partial J(w^{(t)})}{\partial w_1}$$

α is step (learning) rate.

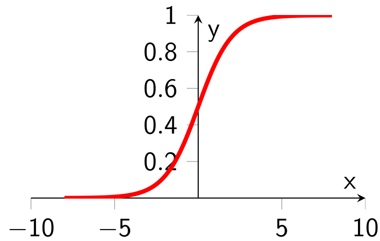
1. Considering the following single neuron



1. We want to train this neuron to minimize the following cost function

$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}(h(x^i) - y^i)^2$$

2. Considering the sigmoid activation function $f(z) = \frac{1}{1+e^{-z}}$



3. We want to calculate $\frac{\partial J(w)}{\partial w_i}$

Training neuron with sigmoid activation(regression)

1. We want to calculate $\frac{\partial J(w)}{\partial w_i}$
2. By using the chain rule, we obtain

$$\frac{\partial J(w)}{\partial w_j} = \frac{\partial J(w)}{\partial f(z)} \times \frac{\partial f(z)}{\partial z} \times \frac{\partial z}{\partial w_j}$$

$$\frac{\partial J(w)}{\partial f(z^i)} = \frac{1}{m}\sum_{i=1}^{m}(f(z^i) - y^i)$$

$$\frac{\partial f(z)}{\partial z} = \frac{e^{-z}}{(1+e^{-z})^2} = f(z)(1-f(z))$$

$$\frac{\partial z}{\partial w_j} = x^j$$

$$w_j^{(t+1)} = w_j^{(t)} - \alpha \frac{\partial J(w)}{\partial w_j}$$

α is the learning rate.

Training neuron with sigmoid activation(classification)

1. We want to train this neuron to minimize the following cost function

$$J(w) = \sum_{i=1}^{m} \left[-y^i \ln h(x^i) - (1 - y^i) \ln(1 - h(x^i)) \right]$$

2. Computing the gradients of $J(w)$ with respect to w, we obtain

$$\nabla J(w) = \sum_{i=1}^{m} y^i x^i (h(x^i) - y^i)$$

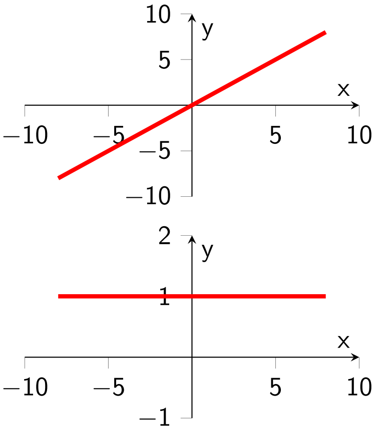
3. Updating the weight vector using the gradient descent rule will result in

$$w^{(t+1)} = w^{(t)} - \alpha \sum_{i=1}^{m} y^i x^i (h(x^i) - y^i)$$

α is the learning rate.

Activation function

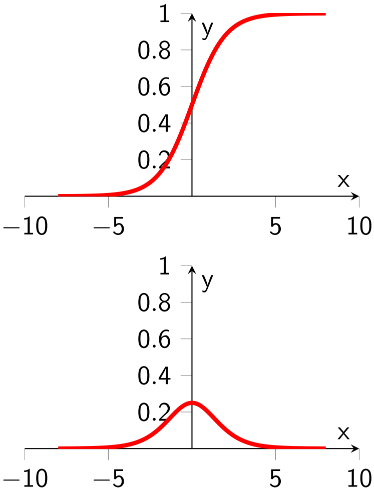
Identity activation function



Properties of identity activation function

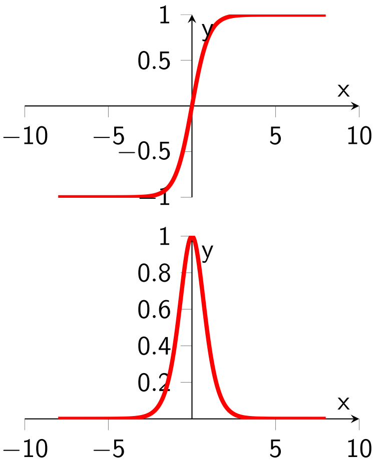
1. Output of this functions will not be confined between any range.
2. It doesn't help with the complexity or various parameters of usual data that is fed to the neural networks.
3. It doesn't increase the complexity of hypothesis space of neural network

Sigmoid activation function



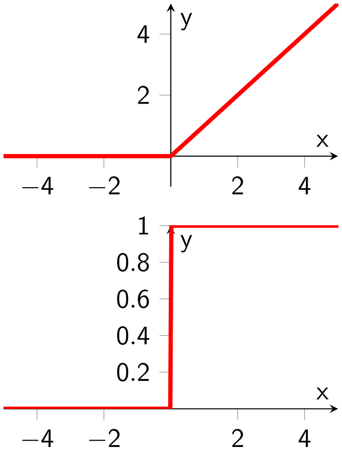
Properties of sigmoid activation function

1. The sigmoid function is in interval $(0, 1)$.
2. It is used to predict the probability as an output.
3. The function is differentiable.
4. The function is monotonic but its derivative is not.
5. This function can cause a neural network to get stuck at the training time.



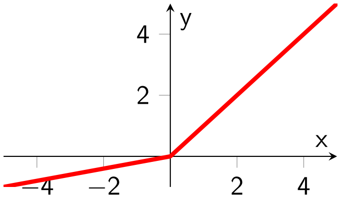
Properties Hyperbolic tangent activation function

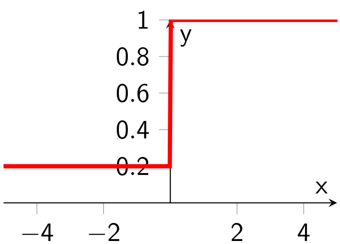
1. The Tanh function is in interval $(-1, 1)$.
2. It is used for classification of two classes.
3. The function is differentiable.
4. The function is monotonic but its derivative is not.
5. This function can cause a neural network to get stuck at the training time.
6. Both tanh and logistic sigmoid activation functions are used in feed-forward nets



Properties Rectified linear unit (ReLU)

1. The ReLU is the most used activation function in the world right now.
2. The function is differentiable except at the origin.
3. The function and its derivative both are monotonic
4. All the negative values become zero immediately which decreases the ability of the model to train from the data properly.





Properties Leaky

1. The leaky ReLU helps to increase the range of the ReLU function.
2. Usually, the value of a is 0.01. a is the slope of negative part.
3. When $a \neq 0.01$, then it is called Randomized ReLU.
4. Both Leaky and Randomized ReLU functions are monotonic in nature. Also, their derivatives monotonic in nature.

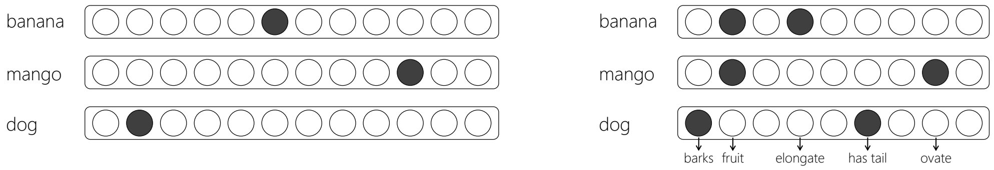
Word embedding

1. How do you represent a word?
   - Represent words as atomic symbols such as talk, university, building.
   - Represent word as a one-hot vector such as

$$\textit{university} = (\underbrace{0}_{egg}, \underbrace{0}_{student}, \underbrace{0}_{talk}, \underbrace{1}_{university}, \underbrace{0}_{building}, \ldots, \underbrace{0}_{buy})$$

2. Issues with on-hot representation
   - How large is this vector? dimensionality is large; vector is sparse
   - Representing new words (any idea?).
   - How measure word similarity?

1. Local versus distributional representation



(a) Local representation **(b)** Distributed representation

2. Linguistic items with similar distributions have similar meanings (words occur in the same contexts probably have similar meaning).

$$\textit{university} = (\underset{\textit{egg}}{0.2}, \underset{\textit{student}}{0.1}, \underset{\textit{talk}}{0.12}, \underset{\textit{university}}{0.38}, \underset{\textit{building}}{0.2}, \ldots, \underset{\textit{buy}}{0.12})$$

3. Word meanings are vector of basic concept.
4. What are basic concept?
5. How to assign weights?
6. How to define the similarity/distance?

1. Distance/similarity

Cosine similarity Word vector are normalized by length

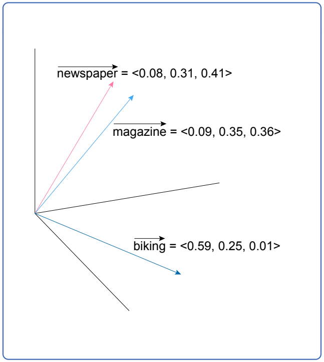
$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\langle \mathbf{u}, \mathbf{v} \rangle}{\|\mathbf{u}\| \|\mathbf{v}\|}$$

Euclidean distance

$$d(\mathbf{u}, \mathbf{v}) = \|\mathbf{u} - \mathbf{v}\|^2$$

Inner product This is same as cosine similarity if vectors are normalized

$$d(\mathbf{u}, \mathbf{v}) = \langle \mathbf{u}, \mathbf{v} \rangle$$



2. Choosing the right similarity metric is important.

How to learn word vectors?

1. What are basic concept?
   - We want that the number of basic concepts to be small and
   - Basis be orthogonal
2. How to assign weights?
3. How to define the similarity/distance such as cosine similarity?

Term-document incidence matrix

Example

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
Anthony	1	1	0	0	0	1	
Brutus	1	1	0	1	0	0	
Caesar	1	1	0	1	1	1	
Calpurnia	0	1	0	0	0	0	
Cleopatra	1	0	0	0	0	0	
mercy	1	0	1	1	1	1	
worser	1	0	1	1	1	0	
...							

Entry is 1 if term occurs. Example: Calpurnia occurs in *Julius Caesar*.

Entry is 0 if term doesn't occur. Example: Calpurnia doesn't occur in *Tempest*.

Each term is represented as a vector of bits.

1. Evaluation of how important a term is with respect to a document.
2. First idea: the more important a term is, the more often it appears: *term frequency*

$$tf_{t,d} = \sum_{x \in d} f_t(x) \text{ where } f_t(x) = \begin{cases} 1 & \text{if } x = t \\ 0 & \text{otherwise} \end{cases}$$

3. The order of terms within a doc is ignored

1. *Inverse document frequency* of a term t:

$$idf_t = log \frac{N}{df_t} \qquad \text{with } N = \text{collection size}$$

2. Rare terms have high *idf*, contrary to frequent terms
3. Example (Reuters collection):

Term t	df_t	idf_t
car	18165	1.65
auto	6723	2.08
insurance	19241	1.62
best	25235	1.5

4. In tf-idf weighting, the weight of a term is computed using both *tf* and *idf*:

$$w(t,d) = tf_{t,d} \times idf_t \qquad \text{called } tf - idf_{t,d}$$

1. we don't need all of the dimensions that represent a word, only the most important ones.
2. There are several techniques such as
   - Principle Component Analysis (PCA): The most important dimensions contain the most variance
   - Latent Semantic Analysis (LSA): Project terms and documents into a topic space using SVD on term-document (co-occurrence) matrix.
   - Low-rank Approximation
3. Can we learn the dimensionality reduction from texts?

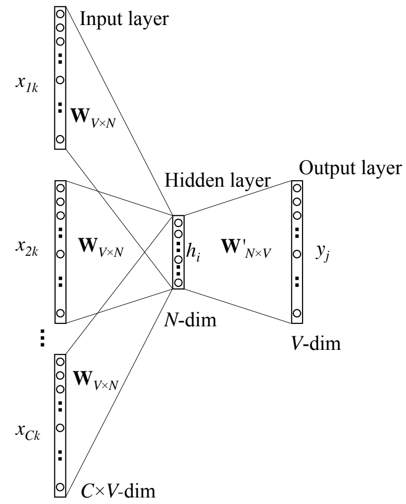
Word2vec algorithm

Word2vec algorithm

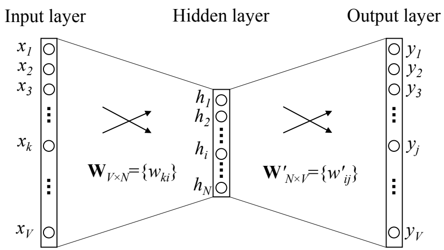
1. Proposed by Mikolov et. al. and widely used for many NLP applications (Mikolov, Chen, et al. 2013; Mikolov, Sutskever, et al. 2013).
2. Key features
   - Uses neural networks to train word / context classifiers (feed-forward neural net)
   - Uses local context windows (environment around any word in a corpus) as inputs to the NN
   - Removed hidden layer.
   - Use of additional context for training LM's.
   - Introduced newer training strategies using huge database of words efficiently.
3. In (Mikolov, Chen, et al. 2013), they proposed two architectures for learning word embeddings that are computationally less expensive than previous models.
4. In (Mikolov, Sutskever, et al. 2013), they improved upon these models by employing additional strategies to enhance training speed and accuracy.

1. Mikolov et al. thus used both the n words before and after the target word w_t to predict it.
2. They called this continuous bag-of-words (CBOW), as it uses continuous representations whose order is of no importance.
3. The objective function of CBOW in turn is

$$J(\theta) = \sum_{t \in Text} \log P(w_t | w_{t-n}, \cdots, w_{t-1}, w_{t+1}, \cdots, w_{t+n})$$



1. Let vocabulary size be V and hidden layer size be N.
2. The input is a one-hot encoded vector $\mathbf{x} = (x_1, \ldots, x_V)$.



3. Weights between input layer and hidden layer is represented by $V \times N$ matrix $\mathbf{W}$.
4. Each row of $\mathbf{W}$ is the N-dimension vector representation $\mathbf{v}_{w_I}$ of the input word.
5. Let $x_k = 1$ and $x_j = 0$ for $j \neq k$, then

$$\mathbf{h} = \mathbf{W}^\top \mathbf{x} = \mathbf{v}_{w_I}^\top$$

6. This implies that activation function of the hidden layer units is simply linear.
7. Weights between hidden layer and output layer is represented by $N \times V$ matrix $\mathbf{W}'$.

1. Using these weights, we can compute a score $\mathbf{u}_j$ for each word in the vocabulary.

$$\mathbf{u}_j = \mathbf{v}'^{\top}_{w_j}\mathbf{h}$$

   where $\mathbf{v}'_{w_j}$ is the j-th column of the matrix $\mathbf{W}'$

2. Then, softmax is used to obtain the posterior distribution of words.

$$p(w_j|w_I) = y_j = \frac{\exp(\mathbf{u}_j)}{\sum_{k=1}^{V}\exp(\mathbf{u}_k)}$$

   where y_j is the output of the j-the unit in the output layer.

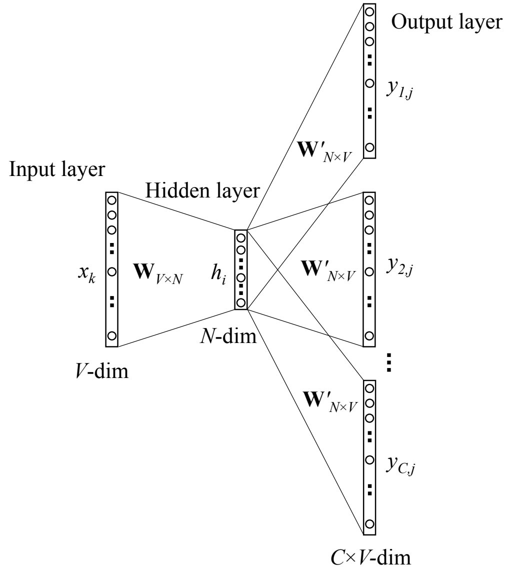
3. By replacing the above two equation, we obtain

$$p(w_j|w_I) = \frac{\exp(\mathbf{v}'^{\top}_{w_j}\mathbf{v}_{w_I})}{\sum_{k=1}^{V}\exp(\mathbf{v}'^{\top}_{w_k}\mathbf{v}_{w_I})}$$

4. Note that $\mathbf{v}_w$ and $\mathbf{v}'_w$ are two representations of the word w.
5. They are called input vector, and output vector of the word w.

1. Instead of using the surrounding words to predict the center word as with CBOW, skip-gram uses the center word to predict the surrounding words.
2. The skip-gram objective thus sums the log probabilities of the surrounding n words to the left and to the right of the target word w_t to produce the following objective function.

$$l(\theta) = \sum_{t \in Text} \sum_{-n \leq j \leq n,\ j \neq 0} \log\ P(w_{t+j} | w_t)$$



Skip-gram (calculating the output)

1. Let vocabulary size be V and hidden layer size be N.
2. The input is a one-hot encoded vector $\mathbf{x} = (x_1, \ldots, x_V)$.
3. Weights between input layer and hidden layer is represented by $V \times N$ matrix $\mathbf{W}$.
4. Each row of $\mathbf{W}$ is the N-dimension vector representation $\mathbf{v}_{w_I}$ of the input word.

$$\mathbf{h} = \mathbf{W}^\top \mathbf{x} = \mathbf{v}_{w_I}^\top$$

5. On the output layer, instead of outputting one multinomial distribution, C-multinomial distributions are output.
6. Each output is computed using the same hidden-output

$$p(w_{c,j} = w_{o,c} | w_I) = y_{c,j} = \frac{\exp(\mathbf{u}_{c,j})}{\sum_{k=1}^{V} \exp(\mathbf{u}_k)}$$

where $w_{c,j}$ is the j-th word on the c-th panel of output layer and $w_{o,c}$ is the actual c-th word in the output context words.

Global Vectors for Word Representation

1. Skip-gram doesn't utilize the statistics of corpus since they train on separate local context windows instead of on global co-occurrence counts.
2. The statistics of word occurrences in a corpus is the primary source of information available to all unsupervised methods for learning word representations.
3. Glove model aims to combine the count-based matrix factorization and the context-based skip-gram model together (Pennington, Socher, and Manning 2014).
4. Let X_{ij} be the number of times word j occurs in the context of word i.
5. Let $X_i = \sum_k X_{ik}$ be the number of times any word appears in context of word i.
6. Let $P_{ij} = P(j|i) = \frac{X_{ij}}{X_i}$ be the probability that word j appear in context of word i.

1. Co-occurrence probabilities for target words ice and steam with selected context words from a 6 billion token corpus.

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

2. Considering two words $i = ice$ and $j = steam$ and study their relationship using various probe words, k.
3. For words k related to ice but not steam, say $k = solid$, we expect the ratio $\frac{P_{ik}}{P_{jk}}$ will be large.
4. For words k related to steam but not ice, say $k = gas$, we expect the ratio $\frac{P_{ik}}{P_{jk}}$ will be small.
5. For words k like water or fashion, that are either related to both ice and steam, or to neither, the ratio should be close to one.
6. Glove uses these global information to learn word representation.

Term embeddings for IR

1. Traditional IR models use local representations of terms for query-document matching.
2. The most straight-forward use case for term embeddings in IR is to enable inexact matching in the embedding space.
3. These approaches can be broadly categorized as (Mitra and Craswell 2018)
   - Methods comparing the query with the document directly in the embedding space,
   - Methods using embeddings to generate suitable query expansion candidates from a global vocabulary and then perform retrieval based on the expanded query.

Query-document matching

1. These methods

   Query embedding finds the term embedding for each query term, and then aggregate these embedding using average word/ term embeddings.

   Document embedding finds the term embedding for each document term, and then aggregate these embedding using average word/ term embeddings.

   Query-document mathcing The query and the document embeddings can be compared using a variety of similarity metrics, such as cosine similarity or dot-product.

2. The term embeddings must be appropriate for the retrieval scenario.
3. The following embeddings are common.
   - LSA
   - word2vec
   - GloVe

1. Instead of comparing the query and the document directly in the embedding space, an alternative approach is to use term embeddings to find good expansion candidates from a global vocabulary, and then retrieving documents using the expanded query.
2. Different functions have been proposed for estimating the relevance of candidate terms to the query.
3. This approach involves comparing the candidate term individually to every query term using their vector representations, and then aggregating the scores.
4. For example,

$$score(t_c, q) = \frac{1}{|q|} \sum_{t_q \in q} \cos(v_{t_c}, v_{t_q})$$

5. Term embedding based query expansion performs worse than pseudo-relevance feedback.
6. But it has better performances when used in combination with pseudo-relevance feedback.

Transformers model

1. The attention make it possible to do sequence to sequence modeling without recurrent network units (Vaswani et al. 2017).
2. The transformer model is entirely built on the self-attention mechanisms without using sequence-aligned recurrent architecture.

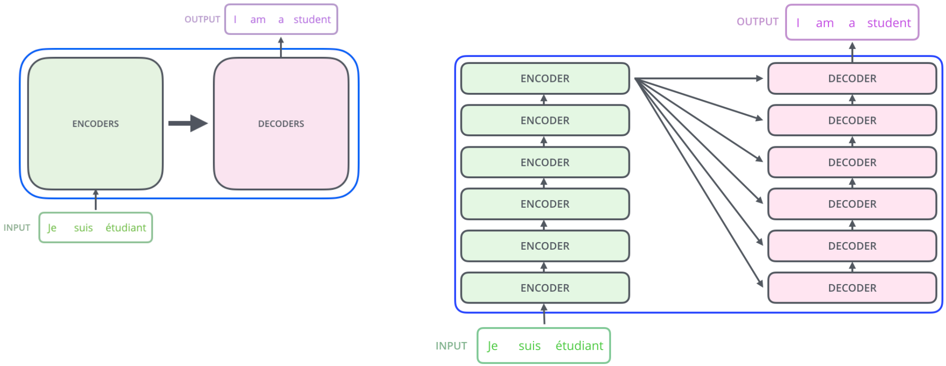


Figure: Jay Alammar

3. The encoding component is a stack of six encoders.
4. The decoding component is a stack of decoders of the same number.

1. The Transformers works slightly differently during training and inference.
2. Input sequence: You are welcome in English.
3. Target sequence: De nada in Spanish

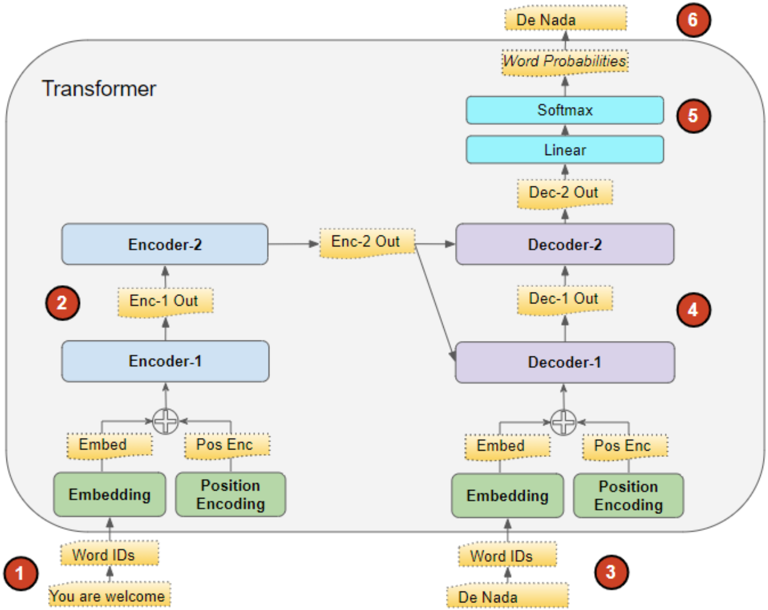


Figure:Ketan Doshi

1. During Inference, we have only the input sequence and don't have the target sequence to pass as input to the Decoder.
2. The goal is to produce the target sequence from the input sequence alone.

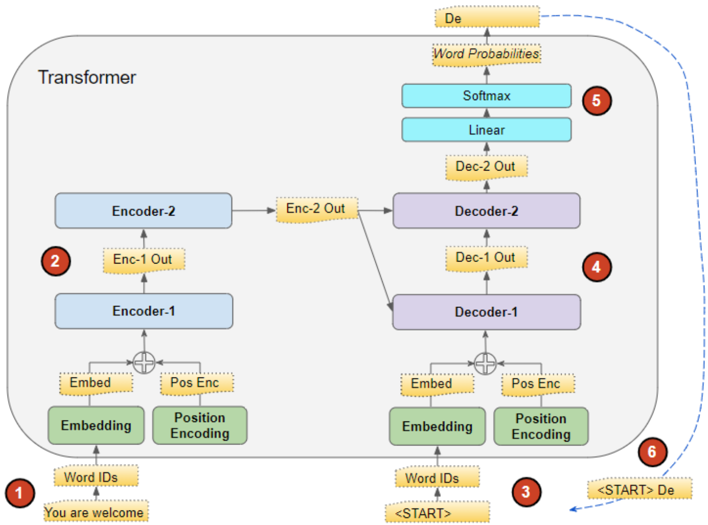
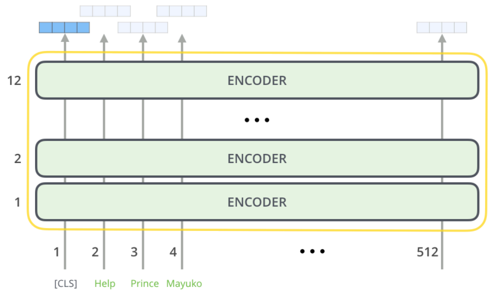


Figure:Ketan Doshi

BERT model

1. BERT (Pre-training of Deep Bidirectional Transformers for Language Understanding) is basically a trained Transformers Encoder stack (**Devlin19**).
2. Each position outputs a vector. For the sentence classification, we focus on the output of only the first position ([CLS]).
3. That vector can now be used as the input for a classifier. The paper achieves great results by just using a single-layer neural network as the classifier.



4. BERT is trained with two tasks instead of the basic language task: **masked language model** and **next sentence prediction**.

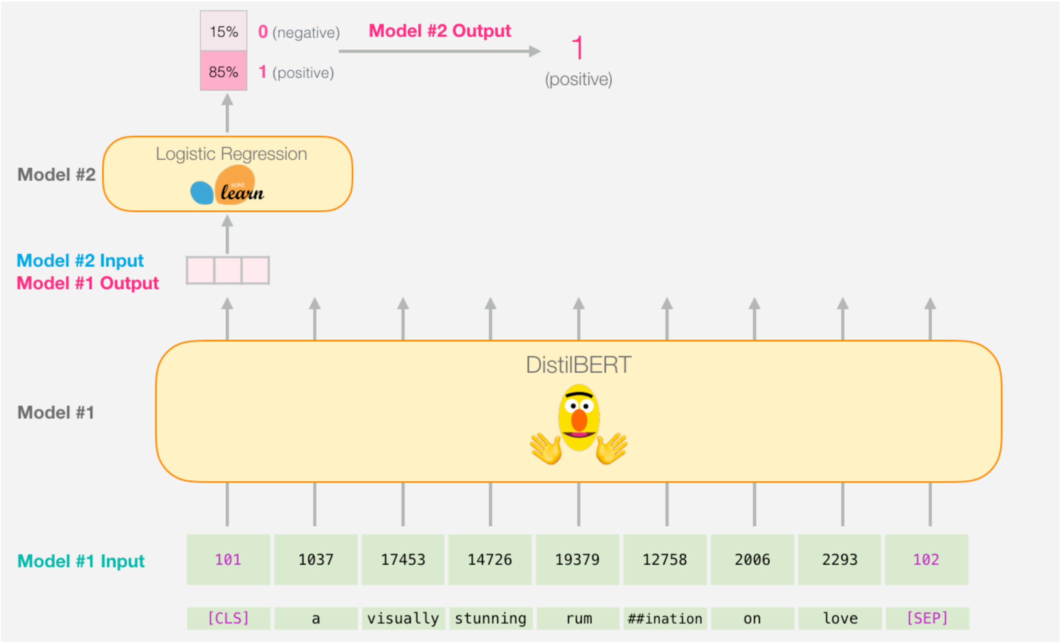


Figure: Jay Alammar

GPT model

1. The GPT (Generative Pre-training Transformer) is built using transformer decoder blocks (Radford et al. 2019).
2. BERT uses transformer encoder blocks.
3. A key difference between the two is that GPT2 outputs one token at a time.

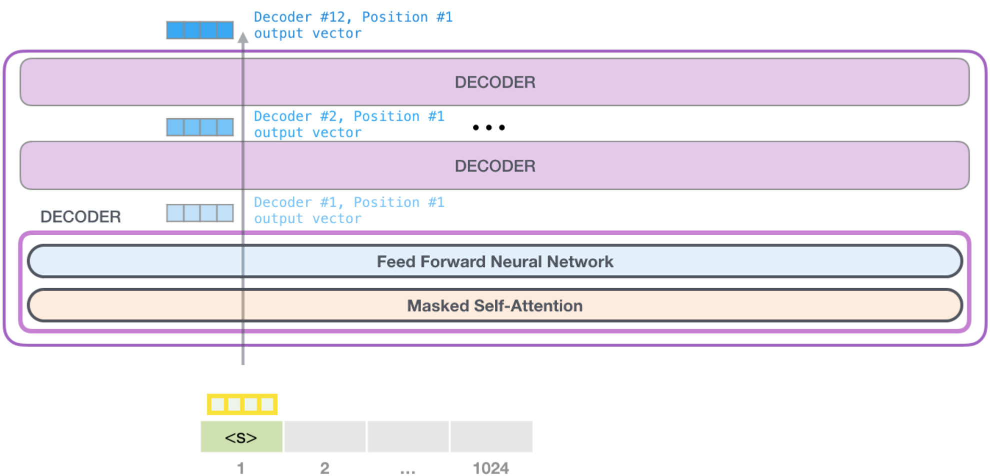
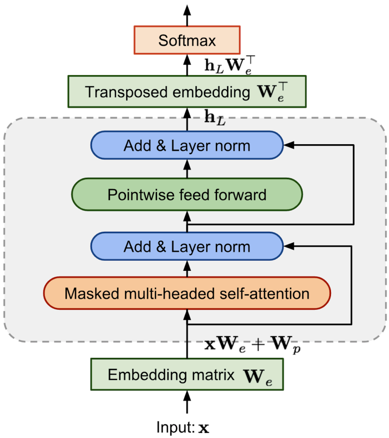


Figure: Jay Alammar

1. GPT applies multiple transformer decoder-blocks over the embeddings of input sequences.
2. Each block contains a masked multi-headed self-attention layer and a pointwise feed-forward layer.
3. The final output produces a distribution over target tokens after softmax normalization .



Credit: Lilian Weng

4. GPT is called **unidirectional** while BERT is called **Bi-directional**.

GPT-2 pre-trained architecture

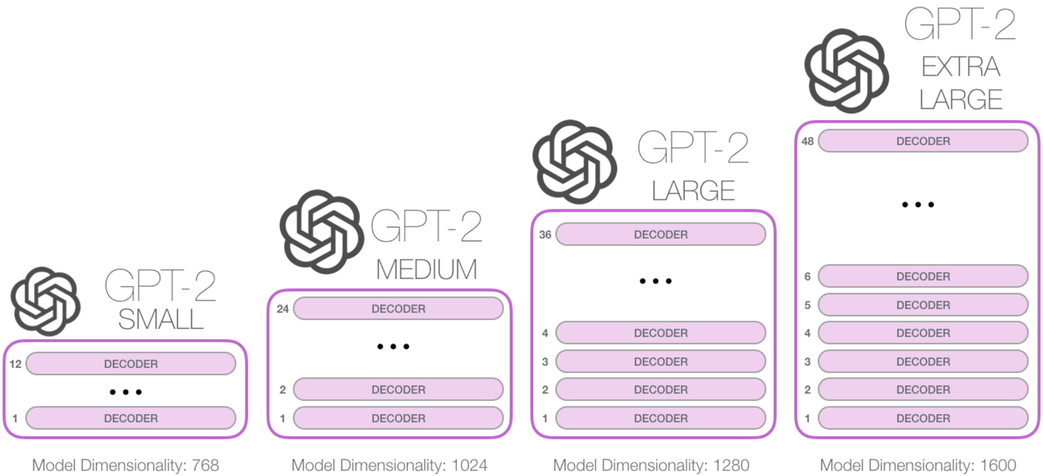


Figure: Jay Alammar

Summary

1. In information retrieval, deep learning can be used for
   - distributed representations of documents and queries,
   - learn to match models by/ without using relevance feedback,
   - learn to rank,
   - entity linking/resolution,
   - recommender systems,
   - sentiment analysis,
   - expertise retrieval,

References

References

- Mikolov, Tomas, Kai Chen, et al. (2013). "Efficient Estimation of Word Representations in Vector Space". In: *Proc. of International Conference on Learning Representations (ICLR).*
- Mikolov, Tomas, Ilya Sutskever, et al. (2013). "Distributed Representations of Words and Phrases and their Compositionality". In: *Proc. of Advances in Neural Information Processing Systems*, pp. 3111–3119.
- Mitra, Bhaskar and Nick Craswell (2018). "An Introduction to Neural Information Retrieval". In: *Foundations and Trends in Information Retrieval* 13.1, pp. 1–126.
- Pennington, Jeffrey, Richard Socher, and Christopher D. Manning (2014). "Glove: Global Vectors for Word Representation". In: *Proc. of Advances in Neural Information Processing Systems*, pp. 1532–1543.
- Radford, Alec et al. (2019). *Language Models are Unsupervised Multitask Learners*. Technical report, OpenAi.
- Vaswani, Ashish et al. (2017). "Attention is All you Need". In: *Advances in Neural Information Processing Systems*, pp. 5998–6008.

Questions?