Information Retrieval

Fundamentals of AI for IR

Up until now

- You have learned how to index and search unstructured information sources by exploring exact match signals (e.g. BM25).

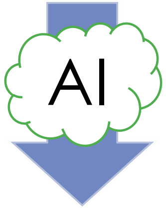
- Problems ?
  - Vocabulary mismatch
  - Lack of semantic understanding
  - Search is contextless

Query: What medications help with heart problems?

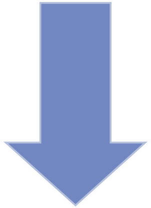
Document: Cardiovascular disease patients often benefit from taking beta blockers or ACE inhibitors. These pharmaceuticals can significantly reduce complications and improve cardiac function. Your physician may prescribe anticoagulants to prevent blood clots.

How can we do better?

- We need to build intelligent retrievals systems that are capable to tackle the previous problems.



- Use data to teach an AI model how to search for information!
  - The aim is to let the AI model learn an internal notion of relevance without human intervention



Neural Information Retrieval

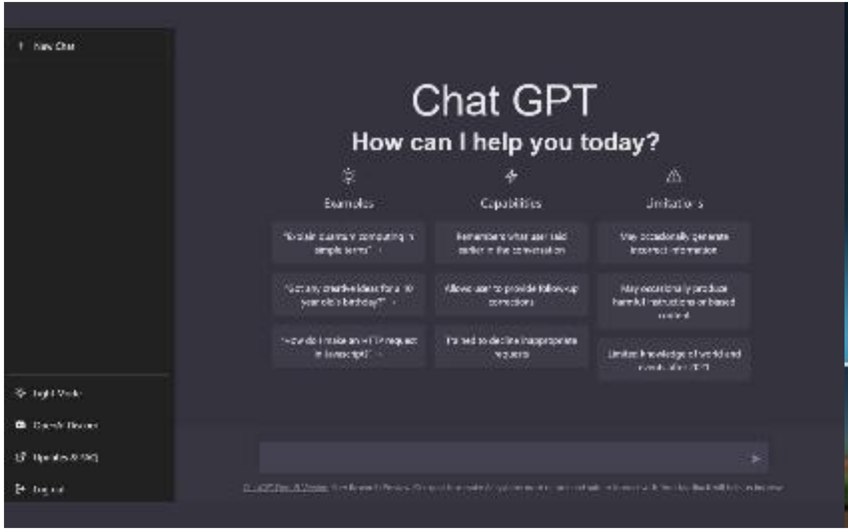
This lecture

- State of AI in 2024
  - Framing as a field of study
  - Examples
- Principles
  - Learning Tasks
  - Neural Networks as the universal approximator
  - How NN learns?
- Textual representations
  - Local vs Distributed representations
  - How to learn distributed representations?
  - Limitations and Solutions

Gentle introduction to AI

State of AI in 2024

- The term “AI” has nowadays become mainstream through powerful applications like ChatGPT.



- However, it wasn't always like this.

Storytime…

- In 2012, a small team led by a “guy” named ***Geoffrey Hinton*** entered in a competition about automatic drug discovery.
- Surprisingly with only **two weeks** and without any background in chemistry, biology or life science, they **WON!!!**

How did they do this?

Source: https://www.youtube.com/watch?v=t4kyRyKyOpo&t=184s&ab_channel=TED

Storytime…

- In 2012, a small team led by a "guy" named ***Geoffrey Hinton*** entered in a competition about automatic drug discovery.
- Surprisingly with only **two weeks** and without any background in chemistry, biology or life science, they **WON!!!**

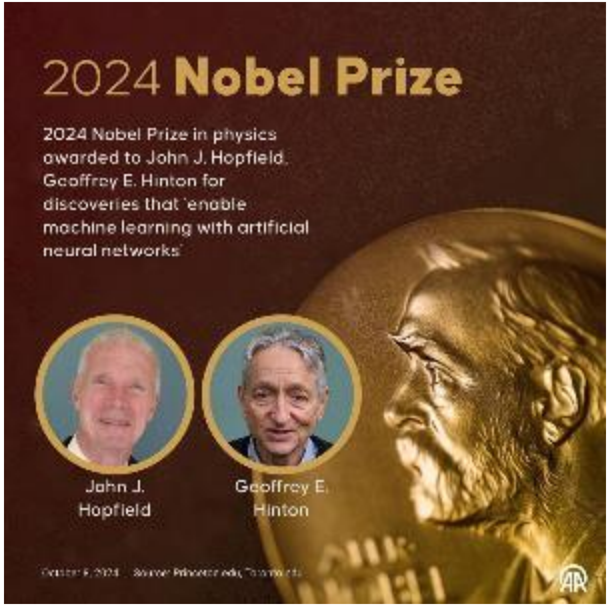
How did they do this?

They used an incredible algorithm called **deep learning**.

Source: https://www.youtube.com/watch?v=t4kyRyKyOpo&t=184s&ab_channel=TED

Geoffrey Hinton - The Godfather of AI

2018 – Turing Award

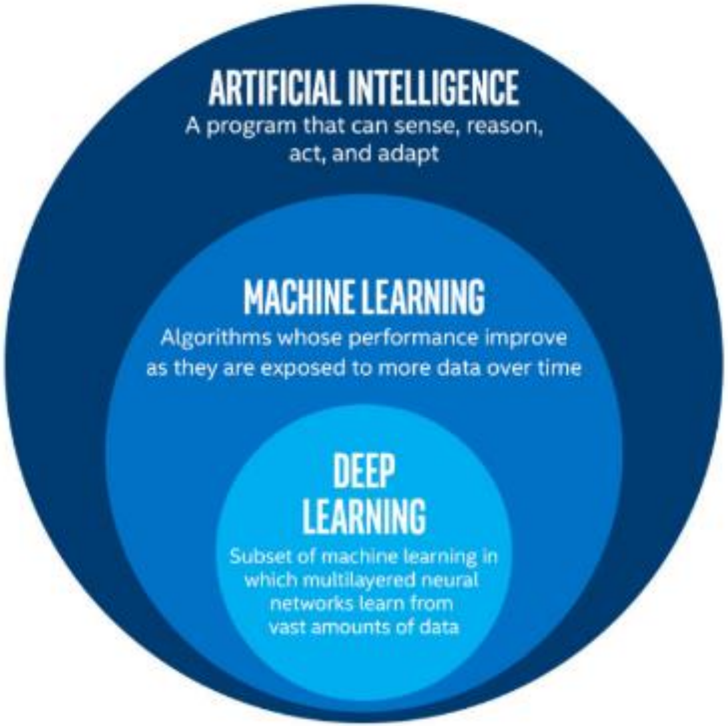


2024 – Nobel Prize in physics

Deep Learning as a field of study

- Nowadays most of the buzz about AI is mostly referring to deep learning (DL) algorithms.

- Deep Learning explores algorithms inspired by the human brain, called neural networks, which are capable of learning tasks directly from data.



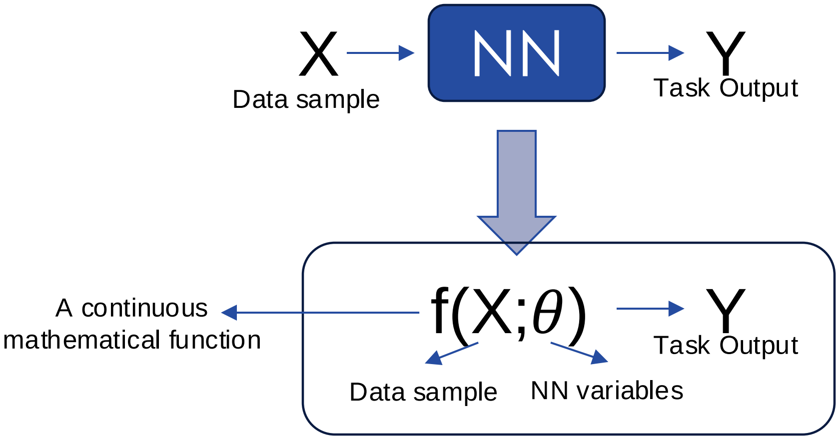
https://towardsdatascience.com/cousins-of-artificial-intelligence-dda4edc27b55

Principles in DL – Neural Network

- Neural Networks are mathematical models that try to mimic the way the human brain works.
  - They contain a large number of trainable variables (NN variables).
  - Their internal computations consist of additions and multiplications between the NN variables and the input values.

Principles in DL – Neural Network

- Neural Networks are mathematical models that try to mimic the way the human brain works.
  - They contain a large number of trainable variables (NN variables).
  - Their internal computations consist of additions and multiplications between the NN variables and the input values.



Correct way to think and see neural networks.

Principles in DL – Example

$f(X;\theta) \longrightarrow Y$

Task Output

Data sample

NN variables

- Given its super generic definition NN can be used in a large variety of tasks, e.g.:

Hotdog vs Not hotdog app from silicon valley tv show.

- Goal:

f( $;\theta) \longrightarrow$ 1(hotdog)

f( $;\theta) \longrightarrow$ 0(not hotdog)

Principles in DL – Example

$f(X;\theta) \rightarrow Y$

Data sample   NN variables   Task Output

- Given its super generic definition NN can be used in a large variety of tasks, e.g.:

Hotdog vs Not hotdog app from silicon valley tv show.

- Goal:

f( ; θ) → 1(hotdog)

f( ; θ) → 0(not hotdog)

Next question, how a NN can automatically learn this mapping?

Principles in DL – The learning process

$f(X;\theta) \longrightarrow Y$

Data sample — NN variables — Task Output

- To learn any data mapping a NN can be trained with different types of supervision.
  - Supervised learning -> Your data has labels.
  - Unsupervised learning and self-supervised -> Your data does not have labels.
  - Semi-supervised learning -> Only small set of the data has labels.
  - Reinforcement learning -> Interactive environment.

- The idea of "training" or "learning" in the context of a NN means that we want to find the NN variables (θ) that allows our function (f) to return the correct predictions for a given dataset!

Find this

$$f(X;\boxed{\theta}) \longrightarrow Y$$

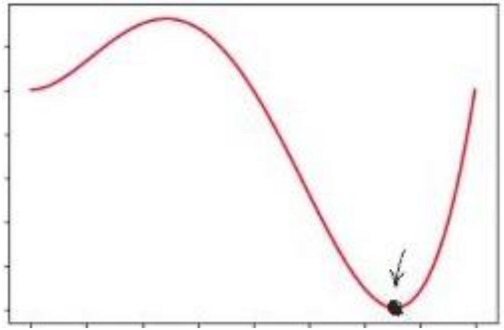
Principles in DL – The learning process

$f(X;\theta) \rightarrow Y$

Data sample — NN variables — Task Output

- If we are able to quantify the error of our NN, then we can optimize our NN variables such that it decreases our NN error value.
  - This notion of error is called a loss function, which is a continuous function that, given our neural network outputs, tells us how wrong our prediction was.

- So, in context of NN "learning" is in reality an "optimization" process.
  - Specially in the optimization of this loss function.



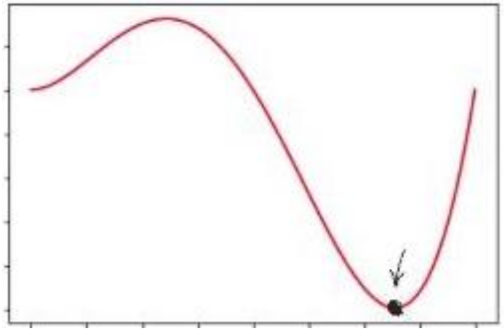
Principles in DL – The learning process

$f(X;\theta) \rightarrow Y$

Data sample NN variables Task Output

- If we are able to quantify the error of our NN, then we can optimize our NN variables such that it decreases our NN error value.
  - This notion of error is called a loss function, which is a continuous function that, given our neural network outputs, tells us how wrong our prediction was.

- So, in context of NN "learning" is in reality an "optimization" process.
  - Specially in the optimization of this loss function.



Problem:

- NN can be very large (millions and billions of variables).
- How can we know which variable we should change to decrease our error value?

Principles in DL – The learning process

$f(X;\theta) \rightarrow Y$

Data sample — NN variables — Task Output

- How can we know which variable we should change to decrease our error value?

Principles in DL – The learning process

$f(X;\theta) \rightarrow Y$

Data sample · NN variables · Task Output

- How can we know which variable we should change to decrease our error value?

- **Use derivatives!**

Yes, the most powerful tool of the last decade is supported on math that you learned in high school!!

- A derivative will tell you the “rate of change” of one variable with respect to another! E.g.: $\frac{\partial L}{\partial \theta_1}$
  - If positive: θ_1 increases -> L increases **Goal: decrease $\boldsymbol{\theta_1}$**
  - If negative: θ_1 increases -> L decreases **Goal: increase $\boldsymbol{\theta_1}$**

- **Backpropagation** is the algorithm that describes how you can compute the partial derivative of the loss (L) with respect to any NN variable

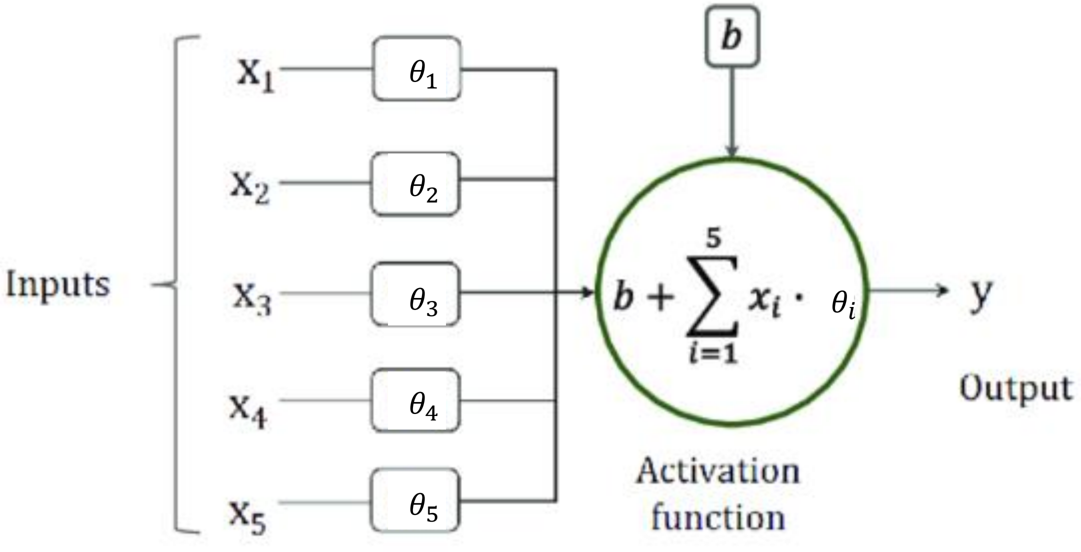
Note: That in reality what is used are “partial derivatives”, because we need to know the partial contribution of each variable individually. These calculations are similar to regular derivatives in many ways.

Principles in DL – Peaking inside of a NN

$f(X;\theta) \rightarrow Y$

Data sample NN variables Task Output

- As the simple neural unit can be represented as:

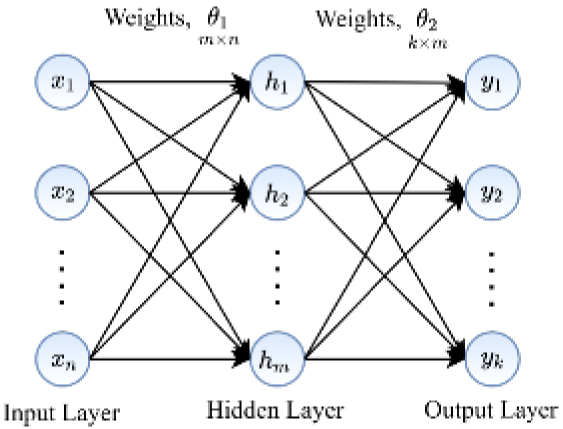


Principles in DL – NN Architectures

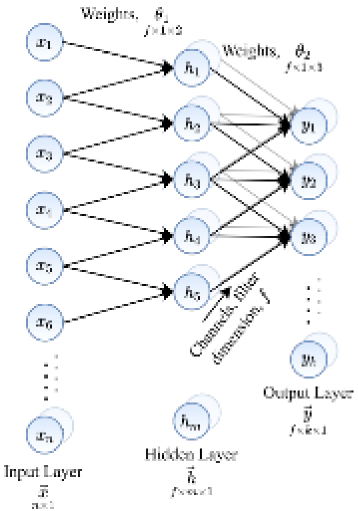
$f(X;\theta) \rightarrow Y$

Data sample | NN variables | Task Output

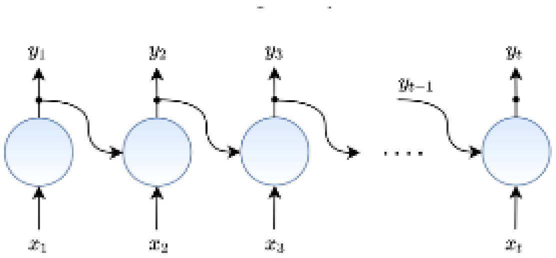
- Usually, a neural unit is organized in architectures:



Fully connected



Convolutional
- Good for spatial data like images



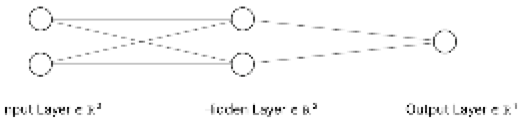
Recurrent
- Good for temporal data

Principles in DL – Practical

$f(X;\theta) \rightarrow Y$

Data sample NN variables Task Output

- What do you need to implement DL models?
  - Since NN are just math the only thing you would need is a good mathematical library.



Code in python for this NN

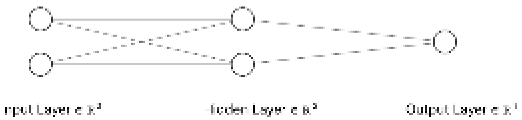
```python
def neuralnetwork(x, variables):
    # First layer
    hidden = [0] * len(variables['w1'][0])
    for i in range(len(hidden)):
        for j in range(len(x)):
            hidden[i] += x[j] * variables['w1'][j][i]

    # Second layer
    out = [0] * len(variables['w2'][0])
    for i in range(len(out)):
        for j in range(len(hidden)):
            out[i] += hidden[j] * variables['w2'][j][i]

    return out

# Example:
variables = {
    'w1': [[1, 2],      # 2x2 matrix for first layer
           [3, 4]],
    'w2': [[1],         # 2x1 matrix for second layer
           [2]]
}

x = [0.5, 0.8]        # Input vector
result = neuralnetwork(x, variables)
print(result)         # Will output: [5.9]
```

Principles in DL – Practical

$f(X;\theta) \rightarrow Y$

Data sample · NN variables · Task Output

- What do you need to implement DL models?
  - Since NN are just math the only thing you would need is a good mathematical library.



Code in python for this NN

```python
def neuralnetwork(x, variables):
    # First layer
    hidden = [0] * len(variables['w1'][0])
    for i in range(len(hidden)):
        for j in range(len(x)):
            hidden[i] += x[j] * variables['w1'][j][i]

    # Second layer
    out = [0] * len(variables['w2'][0])
    for i in range(len(out)):
        for j in range(len(hidden)):
            out[i] += hidden[j] * variables['w2'][j][i]

    return out

# Example:
variables = {
    'w1': [[1, 2],     # 2x2 matrix for first layer
           [3, 4]],
    'w2': [[1],        # 2x1 matrix for second layer
           [2]]
}

x = [0.5, 0.8]        # Input vector
result = neuralnetwork(x, variables)
print(result)         # Will output: [5.9]
```

Problem:

- Slow
- Where is the training?
- Where are the partial derivatives (gradients)?

Principles in DL – Practical

$f(X;\theta) \rightarrow Y$

Data sample NN variables Task Output

- What do you need to implement DL models?
  - Since NN are just math the only thing you would need is a good mathematical library.
- Bonus to have:
  - Automatic differentiation.
  - CUDA acceleration.
  - High-level training framework.

PyTorch

Recommended

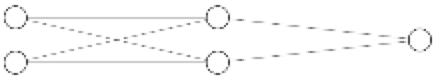
TensorFlow

Principles in DL – Practical

$f(X;\theta) \rightarrow Y$

Data sample — NN variables — Task Output

PyTorch



Code in python with torch for this NN

```python
import torch
import torch.nn as nn

class SimpleNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.w1 = nn.Linear(2, 2, bias=False)
        self.w2 = nn.Linear(2, 1, bias=False)

    def forward(self, x):
        hidden = self.w1(x)
        out = self.w2(hidden)
        return out

model = SimpleNN()
x = torch.tensor([0.5, 0.8])
result = model(x)
print(result)
```

Textual Representation – Moving to Text

- NN are mathematical models!
- How can we feed them textual words?

Textual Representation – Moving to Text

- NN are mathematical models!
- How can we feed them textual words?

Word -> number?

Dog -> 0
Apple -> 1
Cat -> 2

Textual Representation – Moving to Text

- NN are mathematical models!
- How can we feed them textual words?

Word -> number?

Dog -> 0
Apple -> 1
Cat -> 2

Problem:

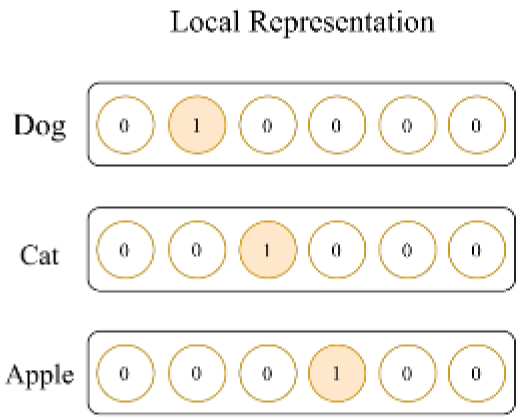
- Numerical order encodes information, which is incorrect in the perspective of the words.
- Arithmetic does not hold.
  - E.g.: if we try to do Cat-Dog = 2-0 = 2. So, Cat-Dog = Cat? It does not make sense.
- Relationships between words are also lost.

Let's do better!

Textual Representation – Moving to Text

- NN are mathematical models!
- How can we feed them textual words?

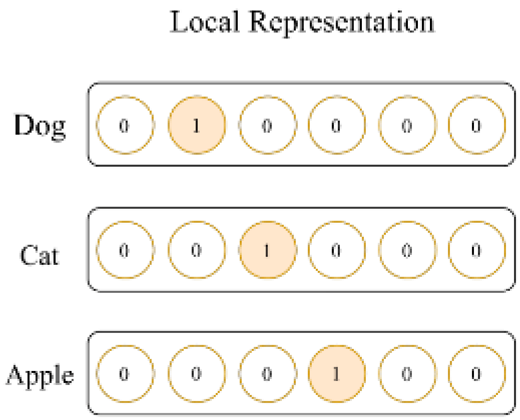
Word -> local representation (one-hot-encoding)



Textual Representation – Moving to Text

- NN are mathematical models!
- How can we feed them textual words?

Word -> local representation (one-hot-encoding)



Problem:

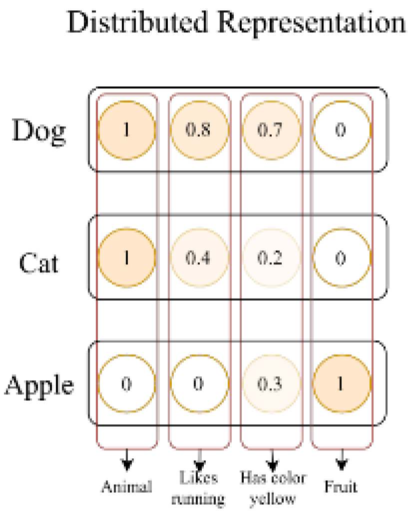
- It creates enormous vectors.
- Relationships between words are still lost because the vectors are orthogonal.

Can you do better?

Textual Representation – Moving to Text

- NN are mathematical models!
- How can we feed them textual words?

Word -> distributed representation (embedding)



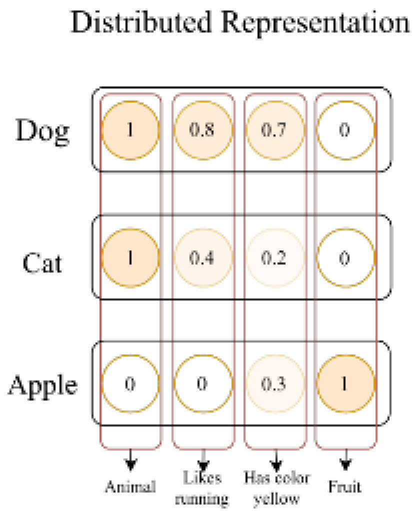
Idea:

- Decomposing words into sets of fixed meaningful dimensions, keeping the relationship between words.

Textual Representation – Moving to Text

- NN are mathematical models!
- How can we feed them textual words?

Word -> distributed representation (embedding)



Idea:

- Decomposing words into sets of fixed meaningful dimensions, keeping the relationship between words.

Next question, how can we create these representations automatically?

Textual Representation – Learn distributed representations

- Let's use NN, since they excel in learning patterns from data!

- Let's follow the *distributional hypothesis*[1] as our training objective.

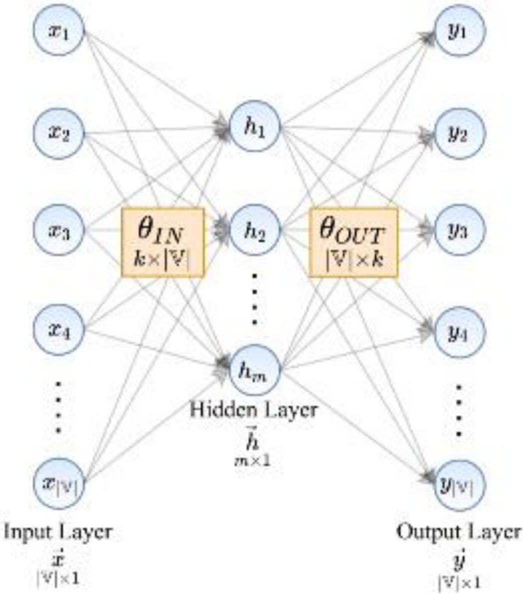
> words occurring in similar contexts tend to have similar meanings

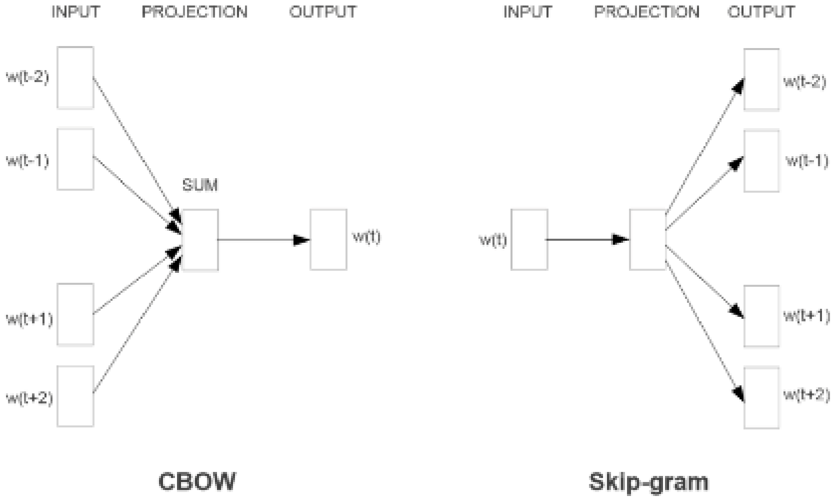
- I.e. the main idea is to represent words that appear in similar contexts with similar vectors since these words will have similar meanings.

[1]Harris, Zellig S. (1954). "Distributional Structure." In: WORD 10.2-3, pp. 146–162. doi: 10. 1080/00437956.1954.11659520

Textual Representation – Word2Vec

- Word2Vec is an algorithm that follows *distributional* hypothesis train NN to learn distributed representations over large text collections.

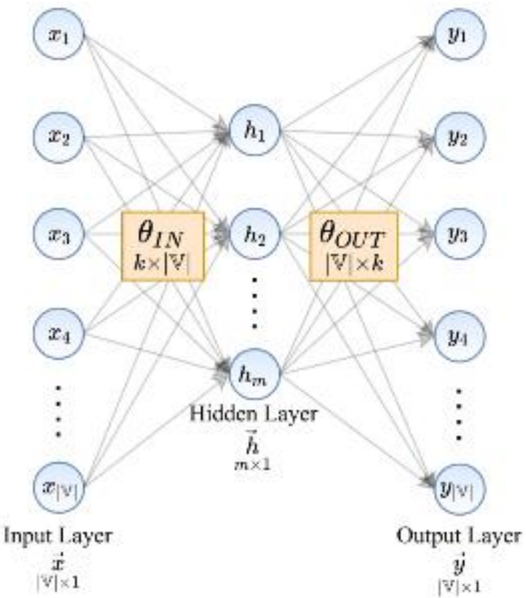


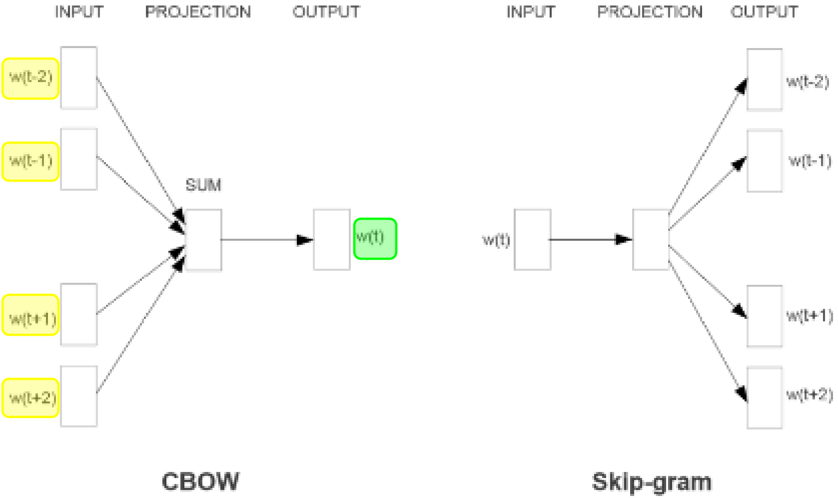


Mikolov, Tomás, Kai Chen, Greg Corrado, and Jeffrey Dean (2013b). "Efficient Estimation of Word Representations in Vector Space." In: 1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings. Ed. by Yoshua Bengio and Yann LeCun. url: http://arxiv.org/abs/1301.3781

Textual Representation – Word2Vec

- Word2Vec is an algorithm that follows *distributional* hypothesis train NN to learn distributed representations over large text collections.

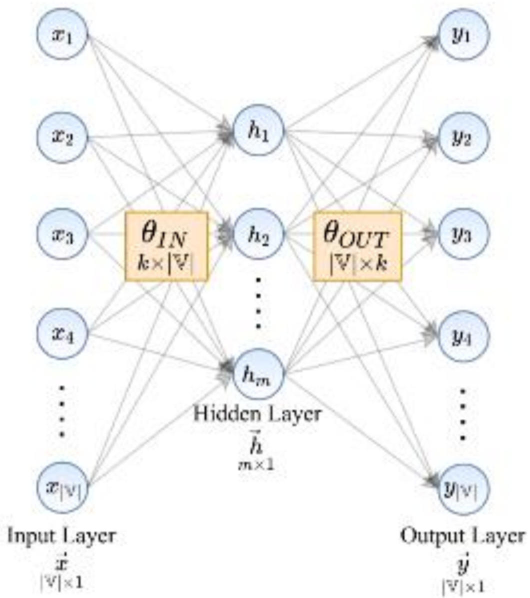


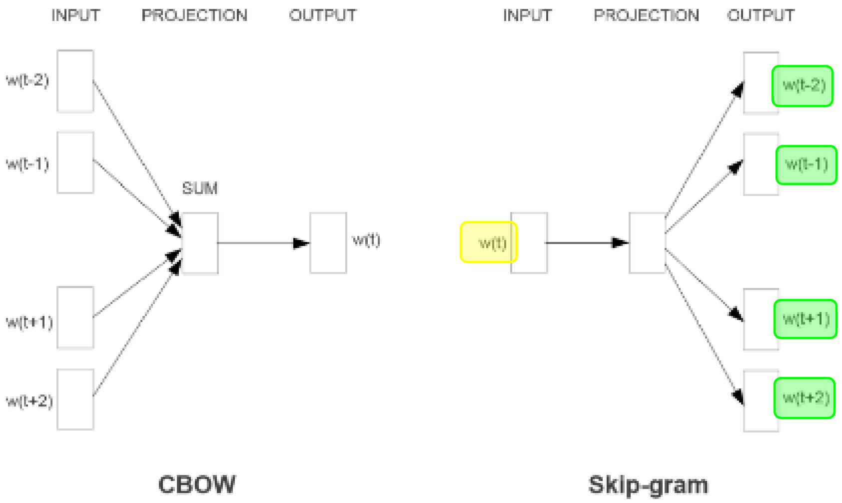


Mikolov, Tomás, Kai Chen, Greg Corrado, and Jeffrey Dean (2013b). "Efficient Estimation of Word Representations in Vector Space." In: 1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings. Ed. by Yoshua Bengio and Yann LeCun. url: http://arxiv.org/abs/1301.3781

Textual Representation – Word2Vec

- Word2Vec is an algorithm that follows *distributional* hypothesis train NN to learn distributed representations over large text collections.

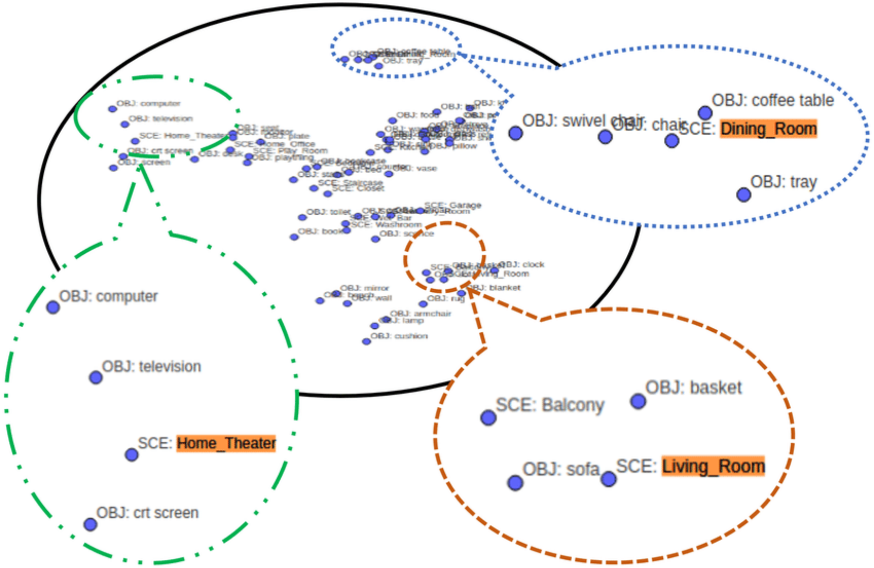




Mikolov, Tomás, Kai Chen, Greg Corrado, and Jeffrey Dean (2013b). "Efficient Estimation of Word Representations in Vector Space." In: 1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings. Ed. by Yoshua Bengio and Yann LeCun. url: http://arxiv.org/abs/1301.3781

Textual Representation – Word2Vec properties

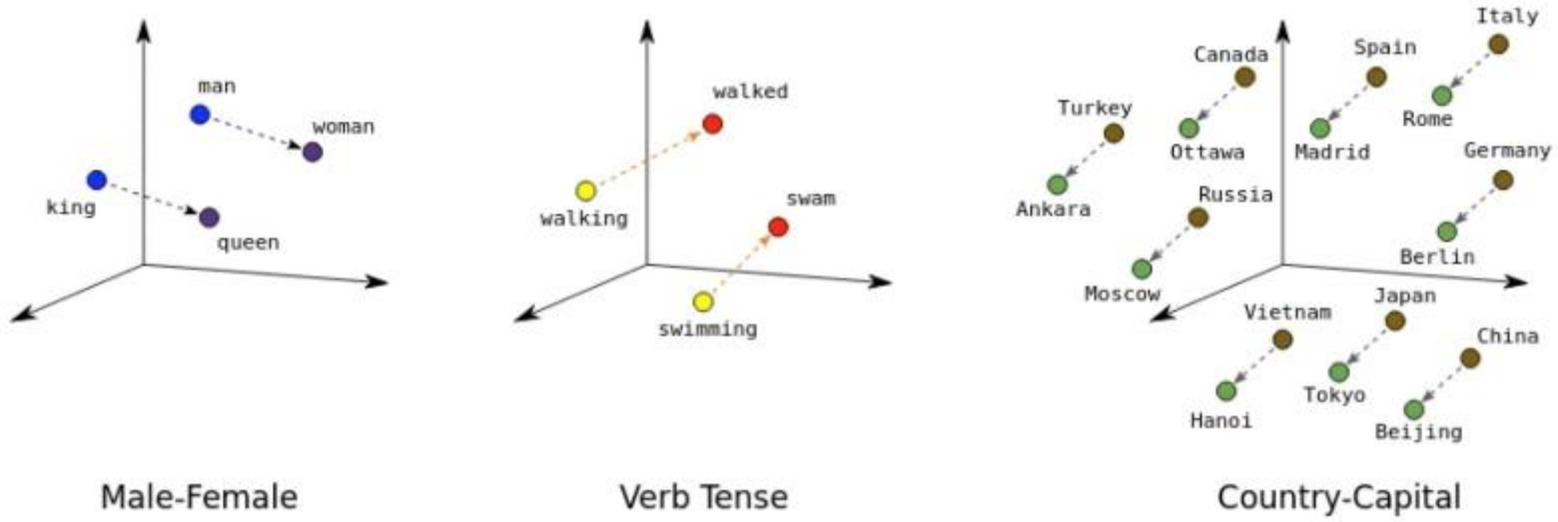
Similar words have similar representations



https://towardsdatascience.com/pre-trained-word-embedding-for-text-classification-end2end-approach-5fbf5cd8aead

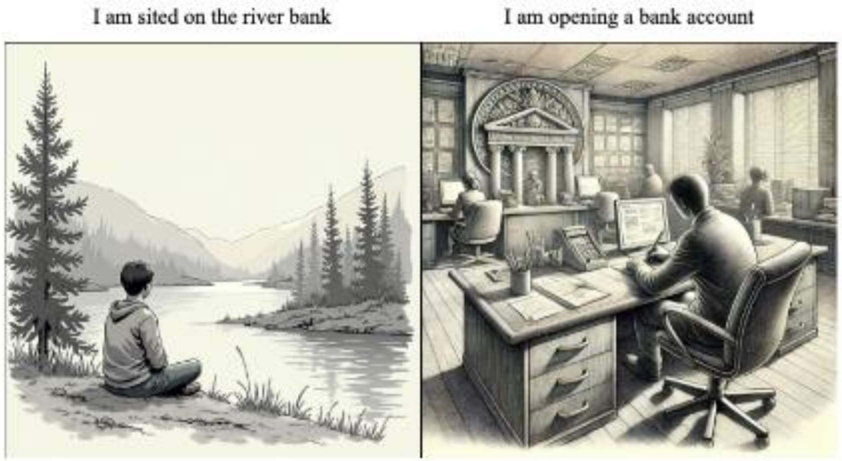
Textual Representation – Word2Vec properties

Arithmetic Holds



Textual Representation – Word2Vec Limitations

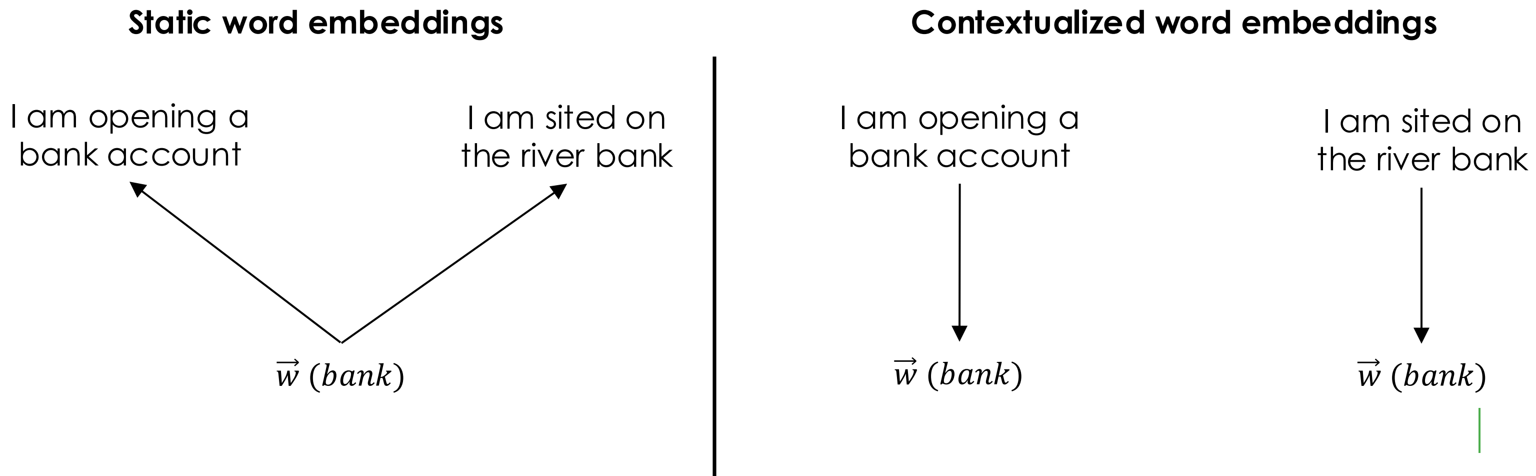
- The word representations are static.
  - Each word has a unique fixed representation regardless of its context.
- What happened with polysemous words?
  - Consider the word "bank" in both contexts:



Textual Representation – Contextualized representations

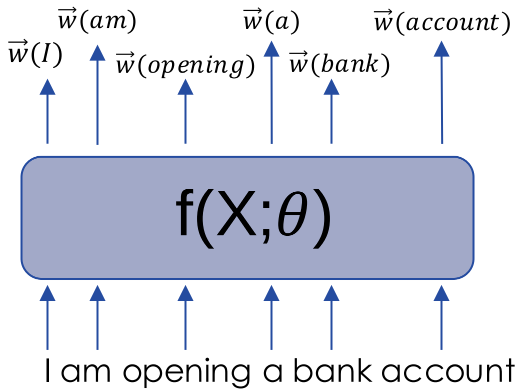
- To address this issue, the distributed word representations need to be contextualized by their surrounding words.

- Creating contextualized distributed representations that are dynamic and change depending on the context.



Textual Representation – Contextualized representations

- Main difference for the word2vec is that now we use large NN to dynamically predict for each word its representation given a context sentence.



Notebooks Resources

- Intro to Pytorch: https://colab.research.google.com/drive/1KX9HUd--QupV_GdROvm3XA28ZyFCuscF?usp=sharing
- Embeddings: https://colab.research.google.com/drive/1OtLgsQD6wPNUcrRixOKAo_eXJL7rD2Px?usp=sharing

Next lecture

- Neural Information Retrieval (NIR)
  - NIR Architectures
  - Shallow Models
  - Deep Models
- Beyond Retrieval, let's generate an answer
  - Conditional Answer Generation
  - Retrieval Augmented Generation (RAG)