AI/ML Applications on Gadi

- Natural Language Processing

NCI Training

Zhuochen Wu

Outline

Lectures

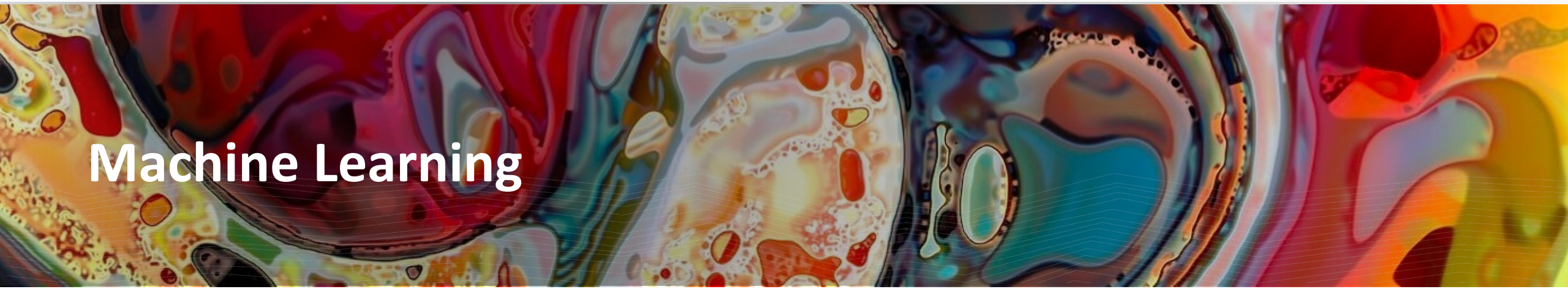
- Introduction to Machine Learning and Text Processing
- Deep Learning and Recurrent Neural Networks
- Transformers
- Topic Modeling

Notebooks

1. Text processing
2. RNN
3. Q&A
4. Topic Modeling

NLP – Natural Language Processing

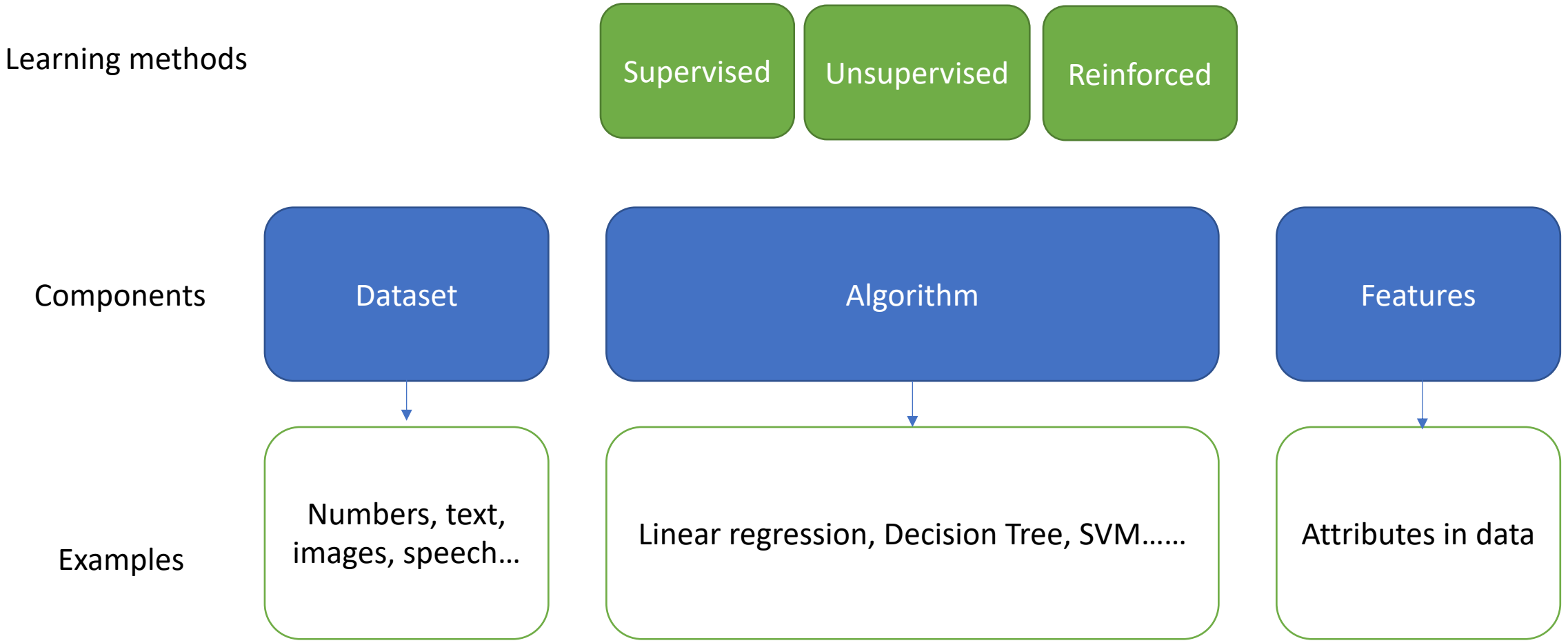
- The techniques for computer software to classify, understand and generate human language.
- Applications:
  - Machine translation(Google Translate)
  - Natural language generation
  - Information retrieval
  - Spam filters
  - Sentiment Analysis
  - Chatbots
  - Linguistic analysation
  - Social science analysation



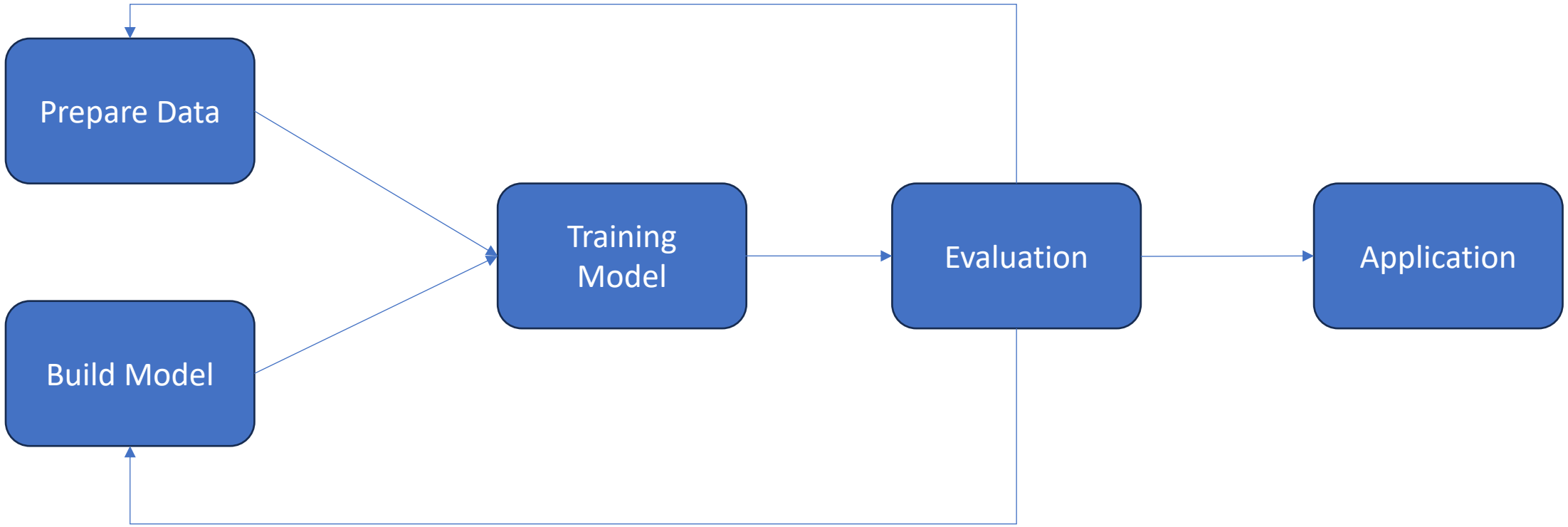
Machine Learning

- Machine learning

Machine learning

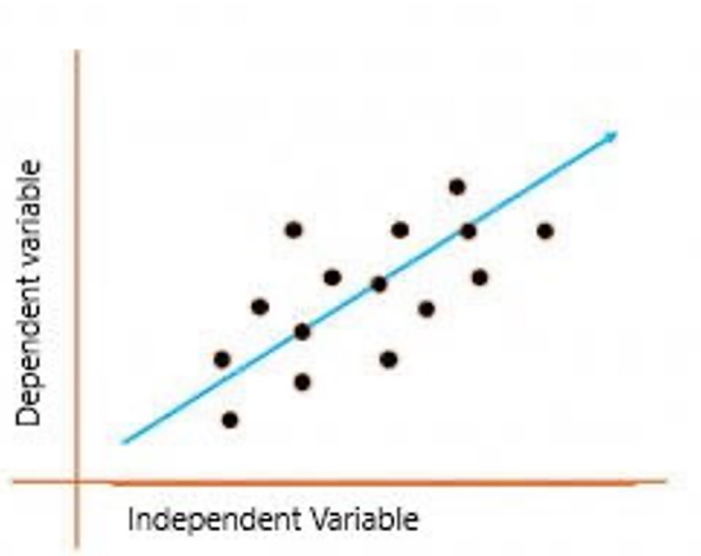


Machine Learning Cycle



Supervised - Linear Regression

- The output variable to be predicted is *continuous* in nature, e.g. scores of a student, diamond prices, etc.
- In a simple linear regression, there is one independent variable and one dependent variable. The model estimates the slope and intercept of the line of best fit, which represents the relationship between the variables. $Y = B_0 + B_1 X$

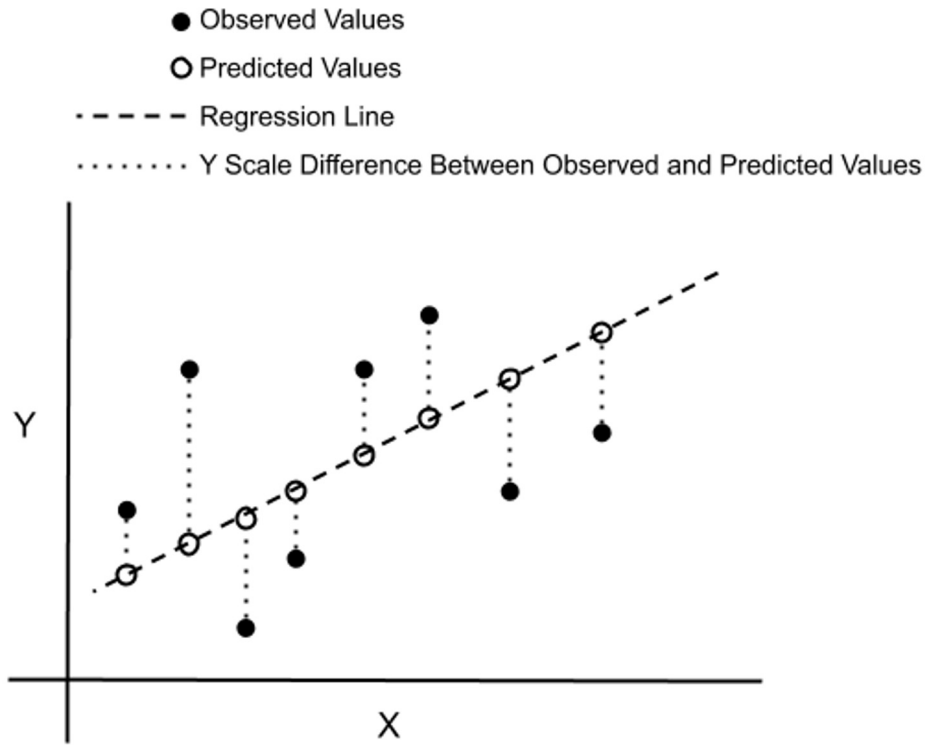


Supervised - Linear Regression

$Y = B_0 + B_1 X$

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

The goal of the linear regression algorithm is to get the **best values for B_0 and B_1** to find the best fit line. The best fit line is a line that has the least error which means the error between predicted values and actual values should be minimum.

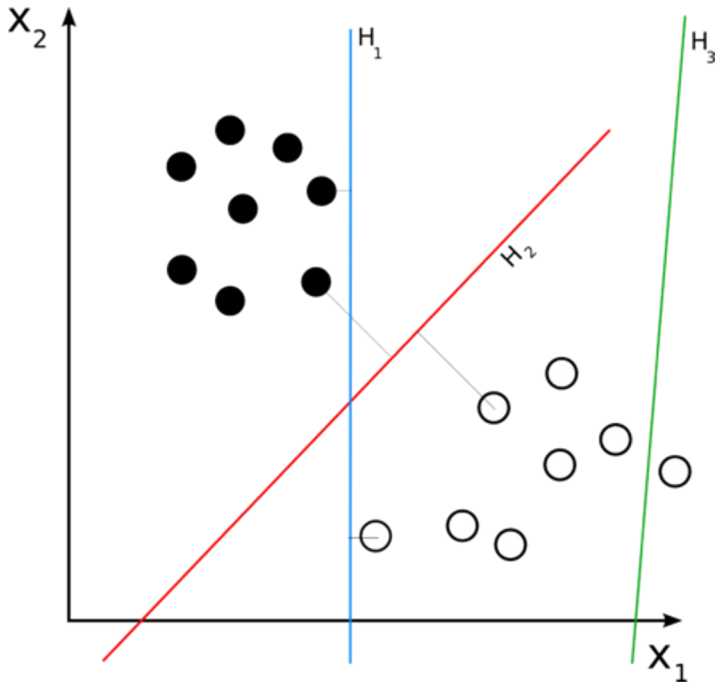


Supervised - Classification

- The output variable to be predicted is categorical nature.

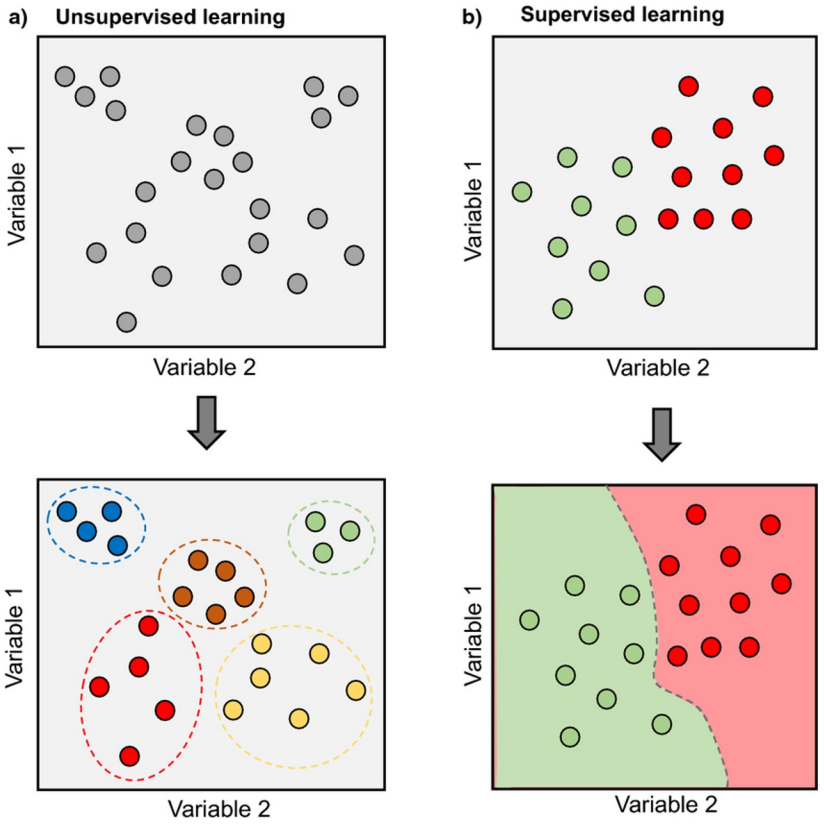
Suppose some given data points each belong to one of two classes, and the goal is to decide which class a *new* data point will be in.

e.g.classifying incoming emails as spam or ham, Yes or No, True or False, 0 or 1.



Unsupervised - Clustering

- It contains no predefined labels assigned to the past data.



Centroid-based

- **K-means**: assigns data to the nearest cluster center (of k clusters), such that the squared distances from the center are minimised

Connectivity-based

- **Hierarchical clustering**: data belong to a child cluster also belong to the parent cluster

Density-based

– **DBSCAN**: Clusters data that satisfy a density criterion

Distribution-based

– **Gaussian mixture models**: Models (iteratively optimized) data with a fixed number of Gaussian distributions

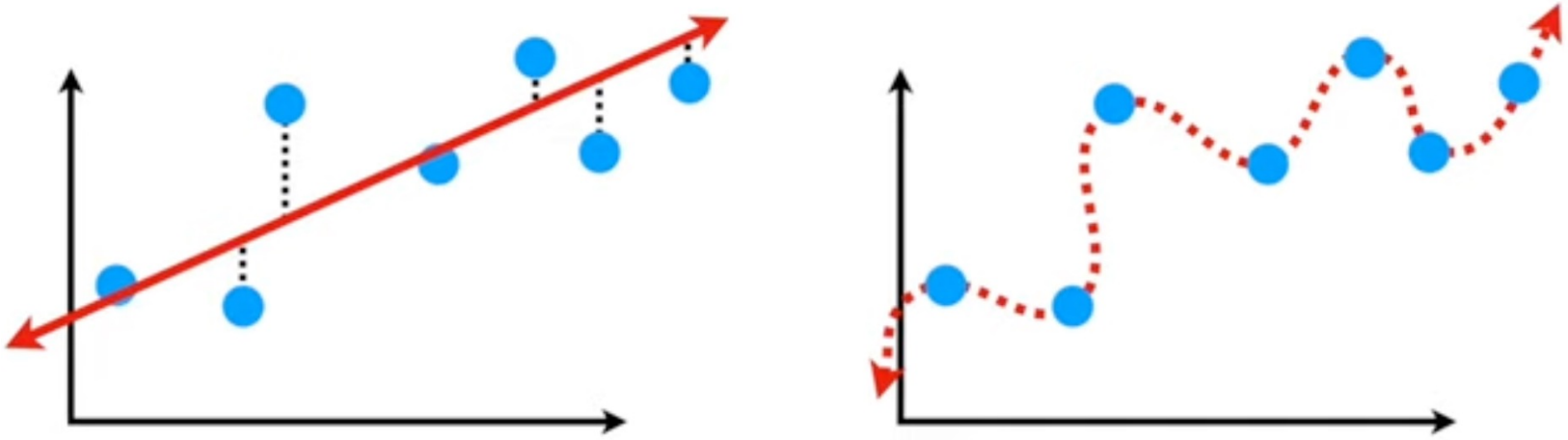
Evaluation – The confusion matrix

	Actual Positive	Actual Negative
Predicted Positive	True positive	False positive
Predicted Negative	False negative	True negative

- This is a binary example, the size of the matrix is determined by things we want to predict

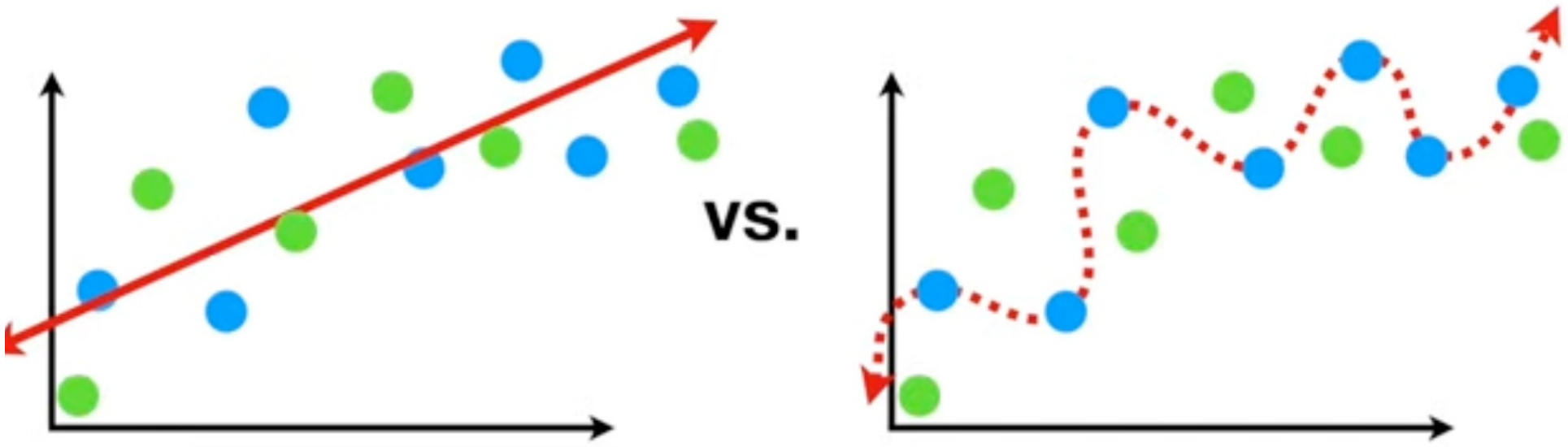
Evaluation - Bias

- When a ML model couldn’t capture the true relationship between data, it has high bias.



Evaluation - Variance

- The performance difference between different datasets



Overfitting

When the machine learning model gives accurate predictions for training data but not for new data.

- The training data size is too small and does not contain enough data samples to accurately represent all possible input data values.
- The training data contains large amounts of irrelevant information, called noisy data.
- The model trains for too long on a single sample set of data.
- The model complexity is high, so it learns the noise within the training data.

Prevent Overfitting

- Early stopping
- Prunning
- Regularization
- Data augmentation

Underfitting vs. overfitting

Underfit models experience high bias—they give inaccurate results for both the training data and test set. On the other hand, overfit models experience high variance—they give accurate results for the training set but not for the test set. More model training results in less bias but variance can increase.

PyTorch

- A framework for building deep learning models, commonly used in applications like image recognition and language processing.
- Relatively easy for most machine learning developers to learn and use.
- Supports for GPUs and its use of reverse-mode auto-differentiation.

PyTorch - Tensors

- Tensors are a specialized data structure that are very similar to arrays and matrices. In PyTorch, we use tensors to encode the inputs and outputs of a model, as well as the model's parameters.
- Tensors are similar to NumPy's ndarrays, except that tensors can run on GPUs or other hardware accelerators.

Create from data

```
data = [[1, 2],[3, 4]]
x_data = torch.tensor(data)
```

Create from numpy array

```
np_array = np.array(data)
x_np = torch.from_numpy(np_array)
```

Create from other tensors

```
# retains the properties of x_data
x_ones = torch.ones_like(x_data)
# overrides the datatype of x_data
x_rand = torch.rand_like(x_data, dtype=torch.float)
```

PyTorch – Tensor Attributes

```
tensor = torch.rand(3,4)
```

Shape of tensor:

```
>>> tensor.shape
Out: torch.Size([3, 4])
```

Datatype of tensor:

```
>>> tensor.dtype
Out: torch.float32
```

Device tensor is stored on:

```
>>> tensor.device
Out: cpu
```

```
shape = (2,3,)
rand_tensor = torch.rand(shape)
ones_tensor = torch.ones(shape)
zeros_tensor = torch.zeros(shape)
```

Random Tensor:

```
 tensor([[0.3904, 0.6009, 0.2566],
        [0.7936, 0.9408, 0.1332]])
```

Ones Tensor:

```
 tensor([[1., 1., 1.],
        [1., 1., 1.]])
```

Zeros Tensor:

```
 tensor([[0., 0., 0.],
        [0., 0., 0.]])
```

PyTorch – Tensor Operations

Tensor to Numpy

```
t = torch.ones(5)        t: tensor([1., 1., 1., 1., 1.])
n = t.numpy()            n: [1. 1. 1. 1. 1.]
```

We move our tensor to the GPU if available

```
if torch.cuda.is_available():
    tensor = tensor.to("cuda")
```

Joining Tensors

```
t1 = torch.cat([tensor, tensor, tensor], dim=1)
tensor([[1., 0., 1., 1., 1., 0., 1., 1., 1., 0., 1., 1.], [1.,
0., 1., 1., 1., 0., 1., 1., 1., 0., 1., 1.], [1., 0., 1., 1., 1.,
0., 1., 1., 1., 0., 1., 1.], [1., 0., 1., 1., 1., 0., 1., 1., 1.,
0., 1., 1.]])
```

Standard numpy like indexing and slicing

```
tensor = torch.ones(4, 4)
First row: tensor[0]
First column: tensor[:, 0]
Last column: tensor[..., -1]
tensor[:,1] = 0

First row: tensor([1., 1., 1., 1.])
First column: tensor([1., 1., 1., 1.])
Last column: tensor([1., 1., 1., 1.])
tensor([[1., 0., 1., 1.],
        [1., 0., 1., 1.],
        [1., 0., 1., 1.],
        [1., 0., 1., 1.]])
```

PyTorch – Tensor Operations

- Matrix multiplication

tensor3 = tensor1 @ tensor2

tensor3 = tensor1.matmul(tensor2)

torch.matmul(tensor1, tensor2, out=tensor3)

- Element-wise product

```
tensor * tensor
tensor.mul(tensor)
torch.mul(tensor, tensor, out=z3)
```

- Transpose

```
tensor.T
```

- Aggregation

tensor.sum()

- Extract to a Python numerical value

tensor.item()

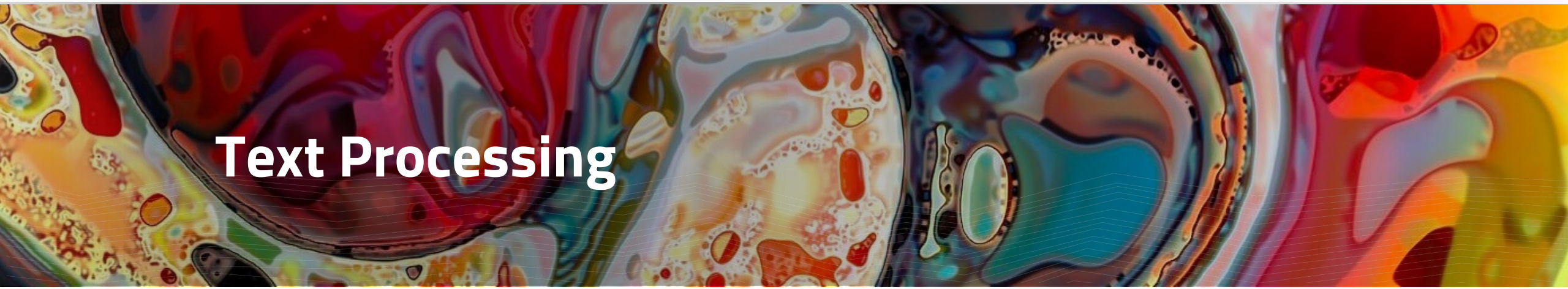
PyTorch – Loading Data

loading data

```
from torch.utils.data import DataLoader


train_dataloader = DataLoader(training_data, batch_size=64, shuffle=True)
test_dataloader = DataLoader(test_data, batch_size=64, shuffle=True)
```

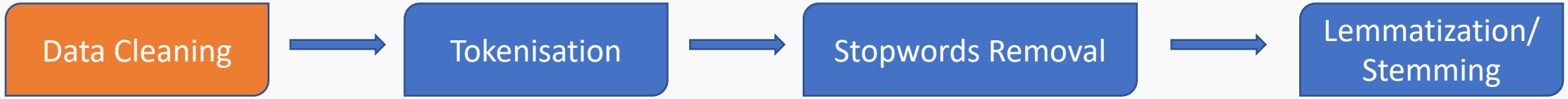
iterate through data

```
for data in data_loader:
        # do somenthing
```



Text Processing

- Text cleaning
- Co-occurance
- Word2vec
  - CBOW

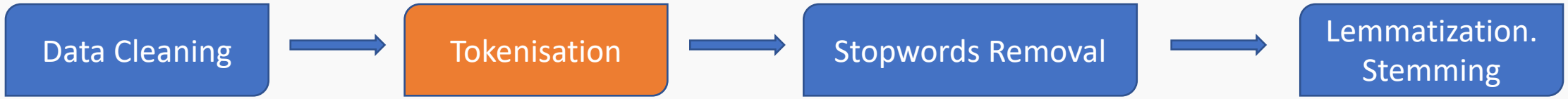


- Data structure
- Data source
- Common dirty strings:
  - HTML tags
  - Human typos
  - Data encoding
  - Punctuations

➢"A touching movie!!\n It is full of emotions and wonderful acting\n\n\n.
 I could have sat through it a second time."

Python string functions
Regular expression – Regex

➢"A touching movie It is full of emotions and wonderful acting I could have sat through it a second time"



- "A touching movie It is full of emotions and wonderful acting I could have sat through it a second time"

Python string functions
Libraries: Nltk, WordNet, Spacy …

- ["a", "touching", "movie", "it", "is", "full", "of", "emotions", "and", "wonderful", "acting", "I", "could", "have", "sat", "through", "it", "a", "second", "time"]

- [“a”, “touching”, “movie”, “it”, "is”, “full”, “of”, “emotions”, “and”, “wonderful”, “acting”, “I”, “could”, “have”, “sat”, “through”, “it”, “a”, “second”, “time”]

Python string functions (customization)
Libraries: Nltk, WordNet, Spacy …

- [“touching”, “movie”, “full”, “emotions”, “wonderful”, “acting”, “have”, “sat”, “second”, “time”]

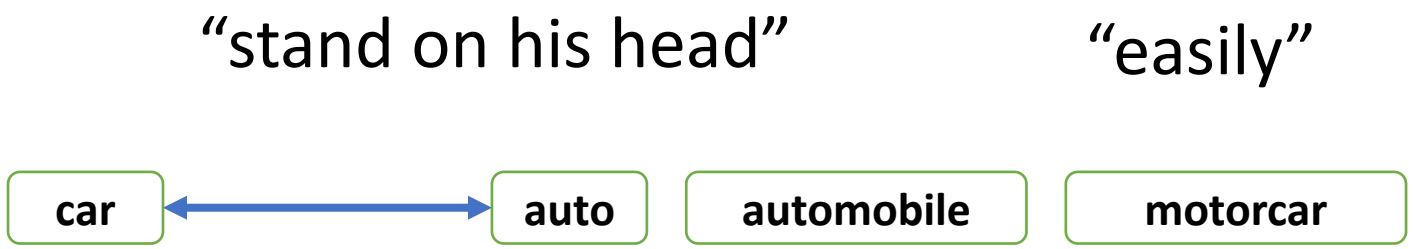
- [“touching”, “movie”, “full”, “emotions”, “wonderful”, “acting”, “have”, “sat”, “second”, “time”]

Libraries: Nltk, WordNet, Spacy …

- [“touching”, “movie”, “full”, “emotions”, “wonderful”, “acting”, “have”, “sait”, “second”, “time”]
- [“touching”, “movie”, “full”, “emotions”, “wonderful”, “acting”, “have”, “sait”, “second”, “time”]

How to represent words so that computer can understand?

- Thesaurus
- Co-occurrence
- **word2vec**
- **CBOW**



Co-occurrence

- Represent the words
- Keep the context
  - Window size = 1

Window size

➢ [“I”, “eat”, ‘burger”, “and”, “you”, “eat”, “salad”]

Corpus

word_to_id

id_to_word

➢ [1, 2, 3, 4, 5, 2, 6]

Vocabulary

➢ {“I”:1, “eat”:2, “burger”:3, “and”:4, “you”:5, “salad”: 6}

Co-occurrence Matrix

	I	eat	burger	and	you	salad
burger	0	1	0	1	0	0

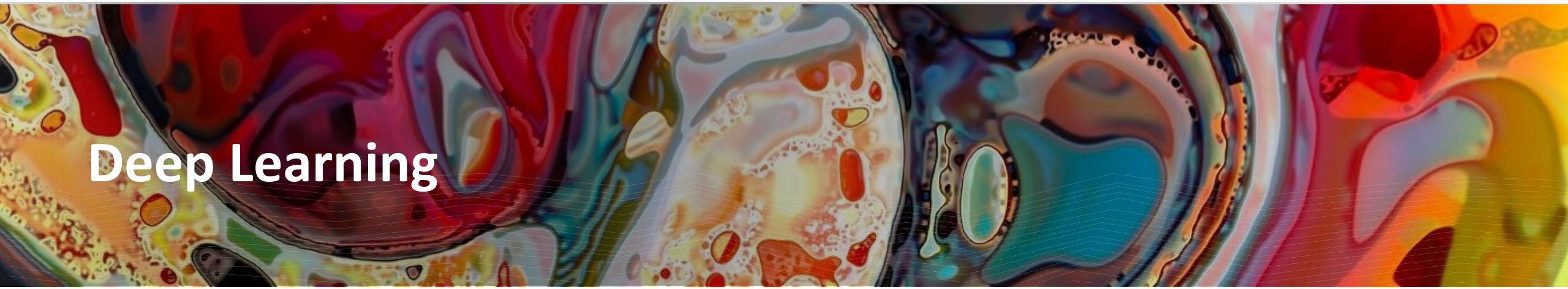
← Unique words

← Co-occurrence count

Vector for “burger” = [0, 1, 0, 1, 0, 0, 0]

Repeat for all words in our sentence …

I	[0, 1, 0, 0, 0, 0]
Eat	[1, 0, 1, 0, 1, 1]
Burger	[0, 1, 0, 1, 0, 0]
And	[0, 0, 1, 0, 1, 0]
You	[0, 1, 0, 1, 0, 0]
Salad	[0, 1, 0, 0, 0, 0]

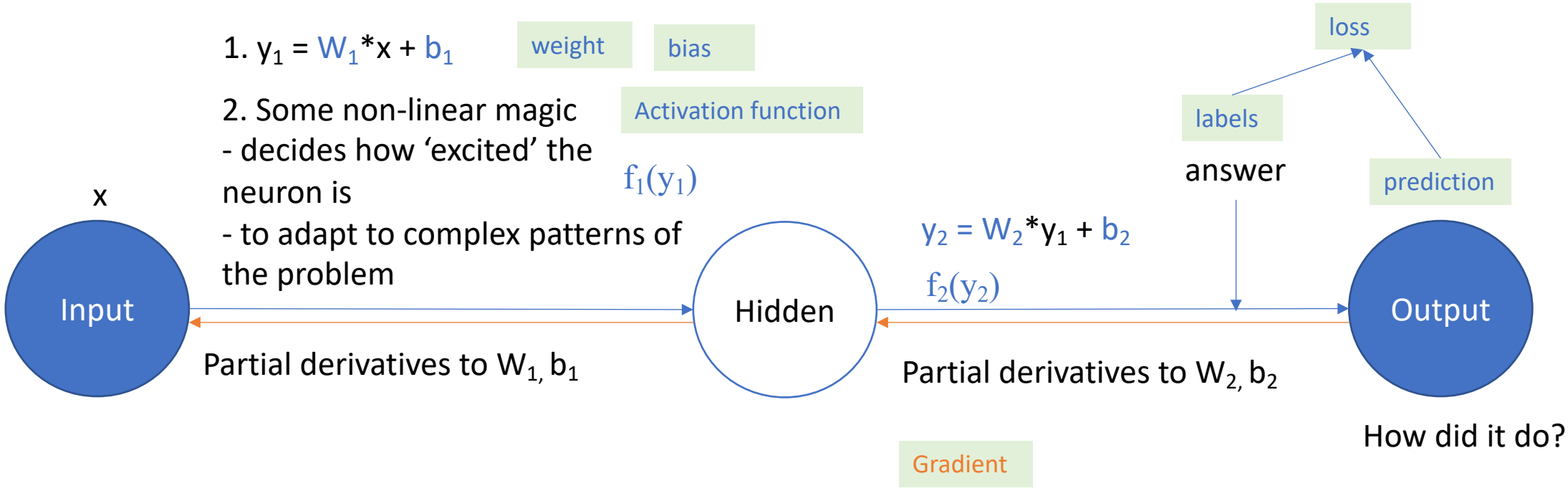


Deep Learning

- Deep learning
  - Neural networks
  - Matrix
  - Loss function
  - Gradient and backpropagation
  - SGD and learning rate
  - Example code

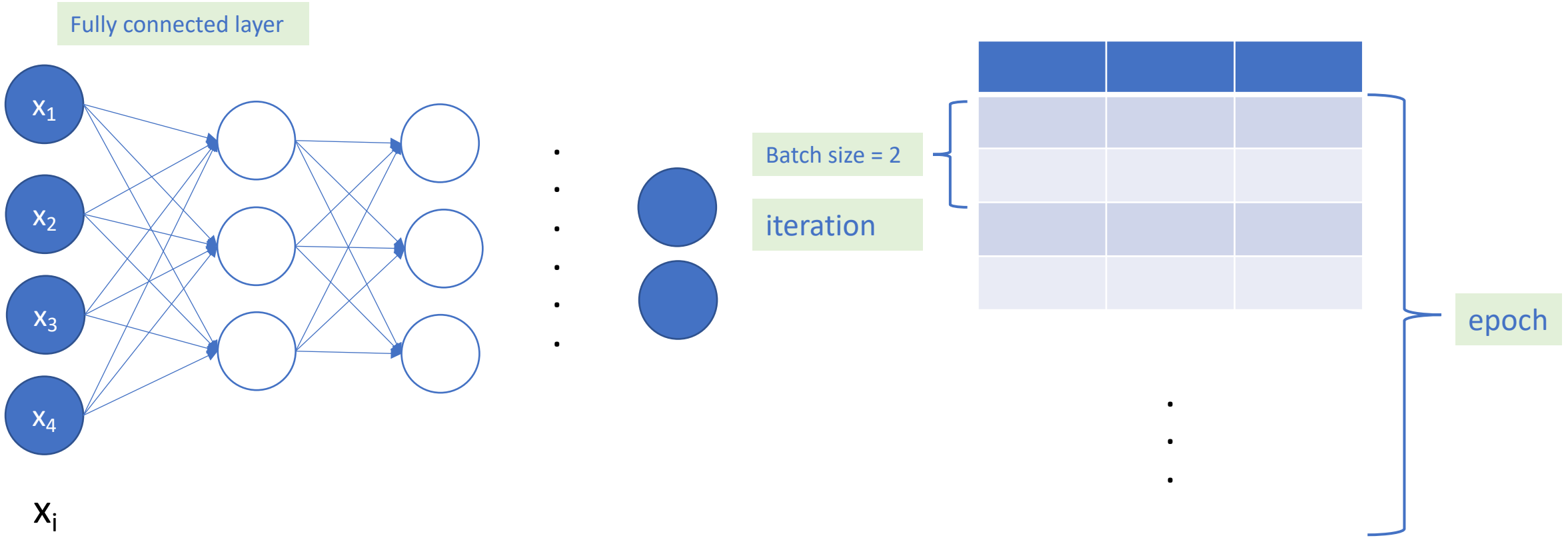
Deep learning – Neural Networks

Given input x, to predict the answer



Now repeat it…

And more data!



Matrix

- Element-wise multiplication

```
[1, 2]     [5, 6]     [5, 12]
[3, 4]     [7, 8]  =  [21, 32]
```

y = tensor * tensor

- Inner product

1x5 +2x7 = 19

```
[1, 2]     [5, 6]     [19, 22]
[3, 4]     [7, 8]  =  [43, 50]
```

tensor2 = tensor.matmul(tensor1)

- Matrix shape

$$A_{nxm} \quad B_{mxh} = \quad C_{nxh}$$
$$C_{nxh} \rightarrow \quad C^{T}{}_{hxn}$$

x $[x_{11}, x_{12}, x_{13}, \ldots x_{1n}]$
$[x_{21}, x_{22}, x_{23}, \ldots x_{2n}]$
.....
$[x_{i1}, x_{i2}, x_{i3}, \ldots x_{in}]$

Input feature dimension = n

w $[w_{11}, w_{12}, w_{13}, \ldots w_{1i}]$
$[w_{21}, w_{22}, w_{23}, \ldots w_{2i}]$
.....
$[w_{m1}, w_{n2}, w_{n3}, \ldots w_{ni}]$

Output number = m

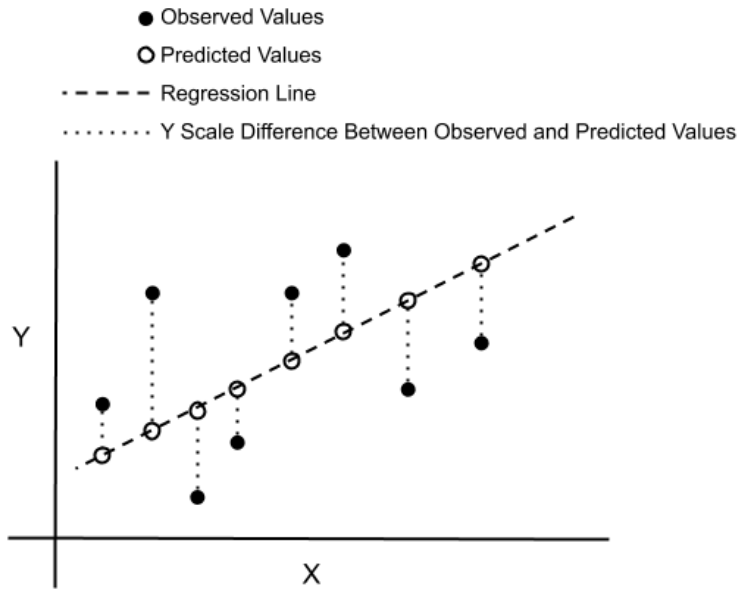
b $[b_1, b_2, b_3, \ldots b_n]$

Broadcasting to mxn

Loss function

- Example: Mean Square Error

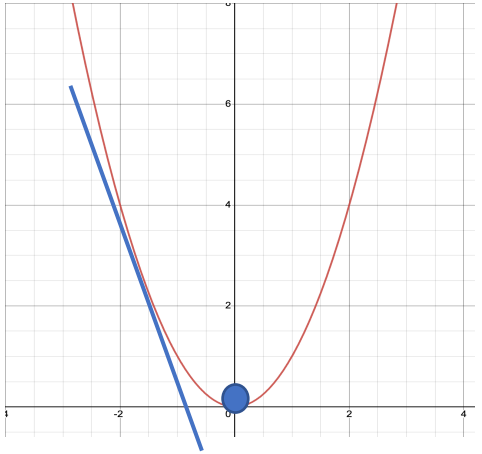
$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$



Predicted value

$$L(\mathbf{w}, b) = \frac{1}{n}\sum_{i=1}^{n} l^{(i)}(\mathbf{w}, b) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{2}\left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)}\right)^2$$

Training goal: $\mathbf{w}^*, b^* = \underset{\mathbf{w},b}{\operatorname{argmin}}\ L(\mathbf{w}, b)$



$Y = x^2$

$Y' = 2x$

Derivative -> value change direction

Gradient and Backpropagation

Partial derivatives $\frac{\partial L(w,b)}{\partial w}, \quad \frac{\partial L(w,b)}{\partial b}$

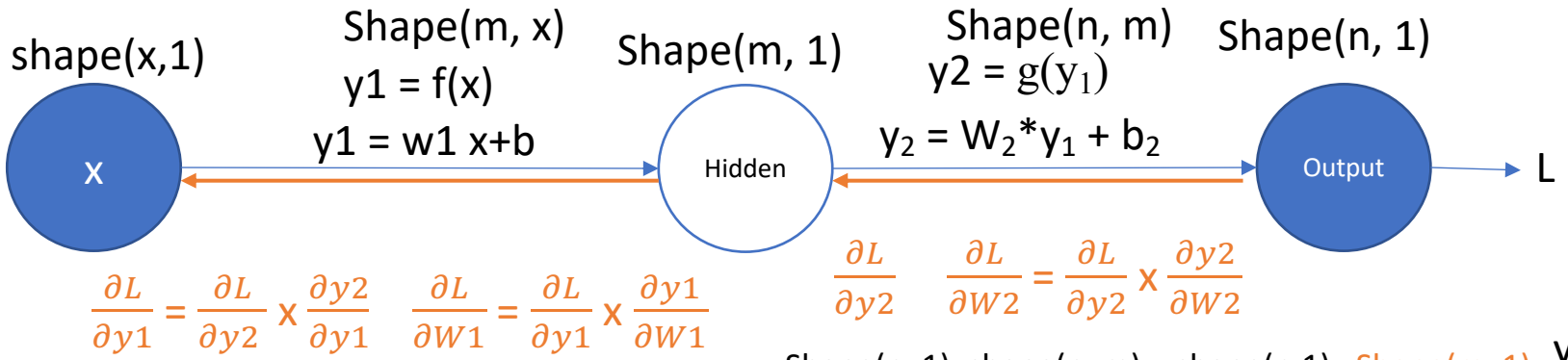
Shape: $\text{shape}(\frac{\partial z}{\partial x}) = \text{shape}(x)$

Chain rule: $y = f(x), z = g(y)$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x}$$

Back propagation:

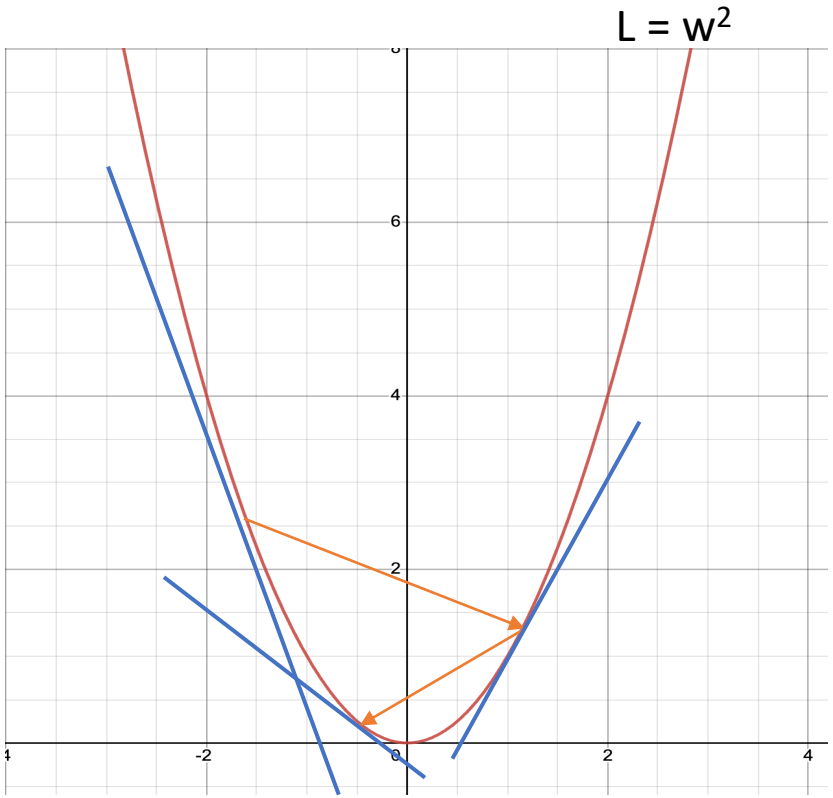
- From loss function to input direction.
- Store value for each step for quick read;
- Using the parameters from forwarding



Shape(n, 1), shape(n, m) = shape(n,1) . Shape(m, 1) y_1^T

Shape(m, 1) = shape(n, 1) x shape(n, m), shape(m, x) = shape(m, 1) x shape(x, 1)

Stochastic Gradient Descent and Learning Rate



gradient

$$(\mathbf{w}, b) \leftarrow (\mathbf{w}, b) - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \partial_{(\mathbf{w},b)} l^{(i)}(\mathbf{w}, b).$$

Optimizer

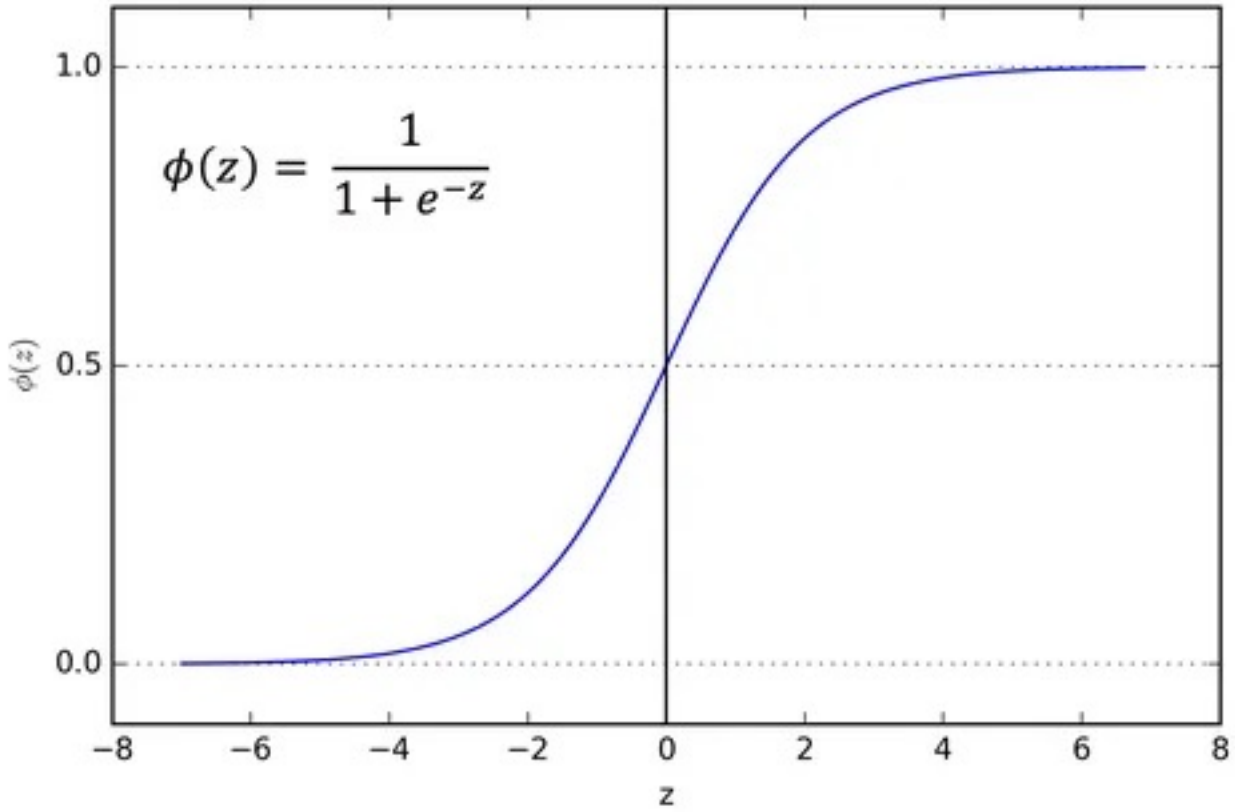
η learning rate

B mini-batch size

$$\mathbf{w} \leftarrow \mathbf{w} - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \partial_{\mathbf{w}} l^{(i)}(\mathbf{w}, b) = \mathbf{w} - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \mathbf{x}^{(i)} \left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)} \right),$$

$$b \leftarrow b - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \partial_b l^{(i)}(\mathbf{w}, b) = b - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)} \right).$$

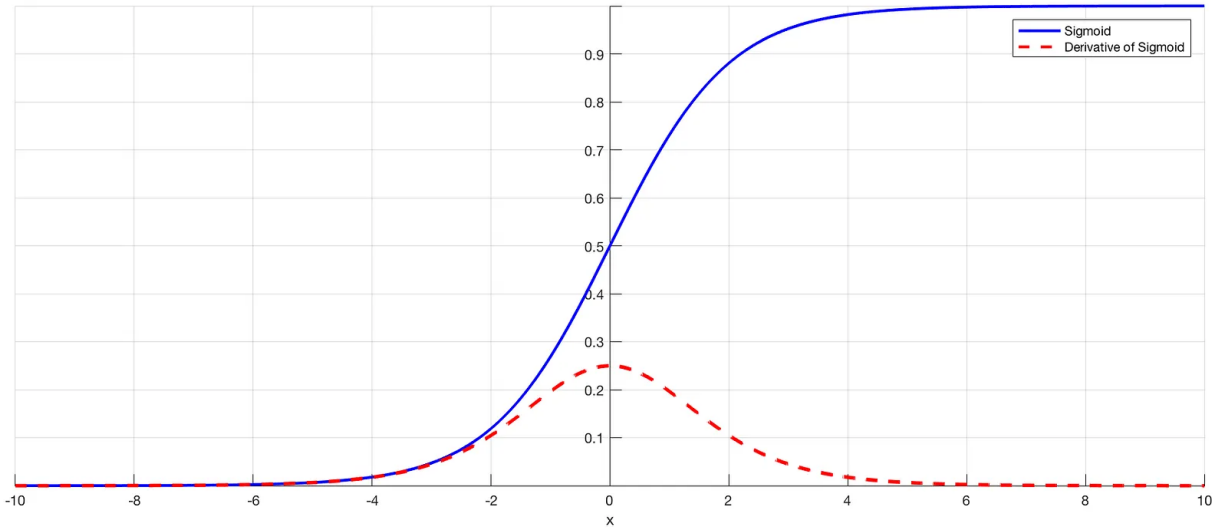
Activation Function - Sigmoid



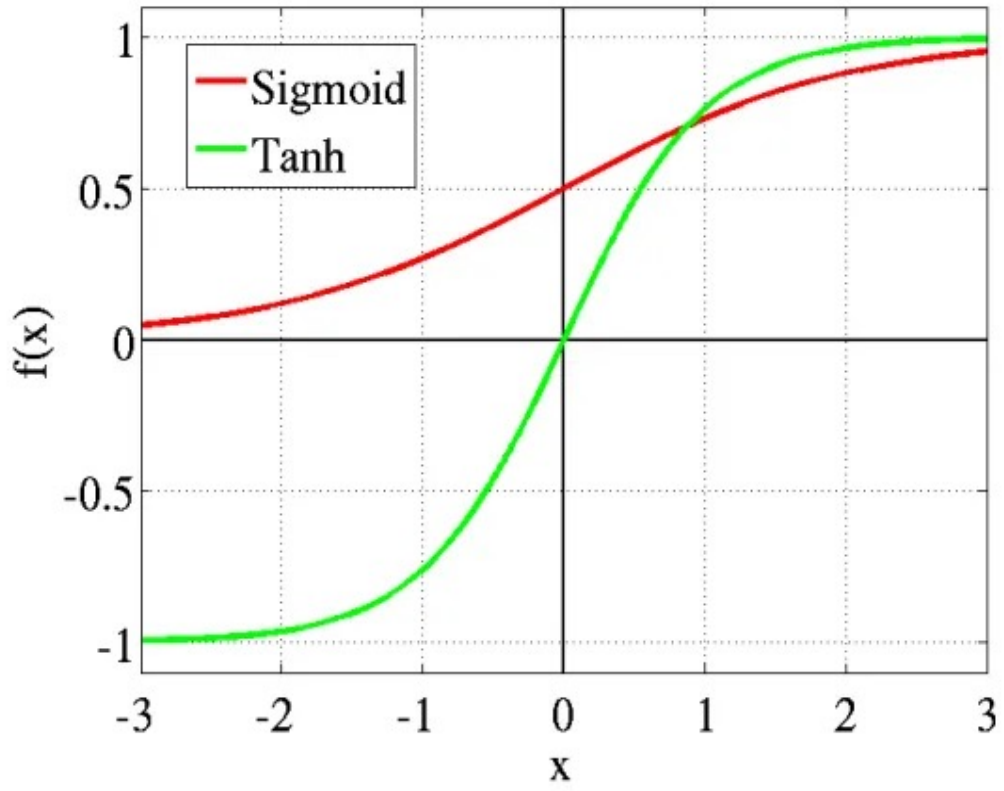
Since probability of anything exists only between the range of **0 and 1**, it is especially used for models where we have to **predict the probability** as an output.

Gradient Vanishing

- As more layers using certain activation functions are added to neural networks, the gradients of the loss function approaches zero, making the network hard to train.

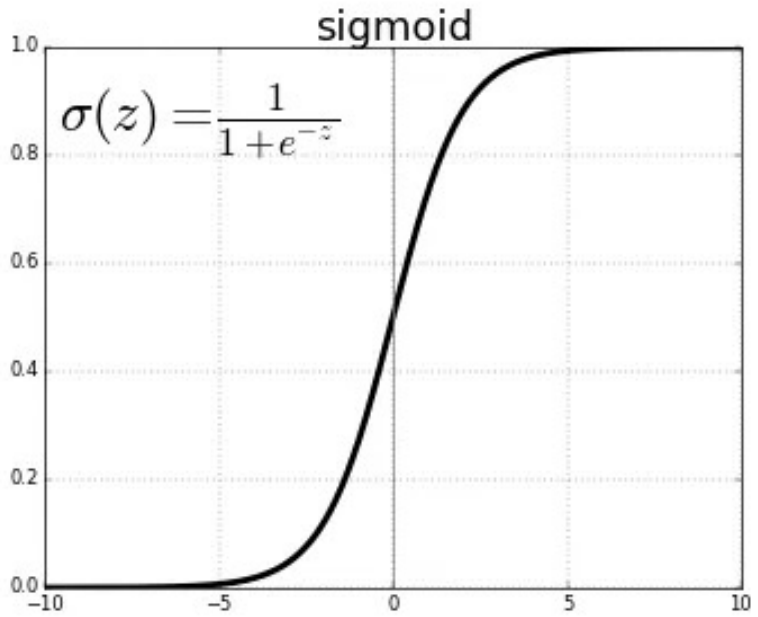


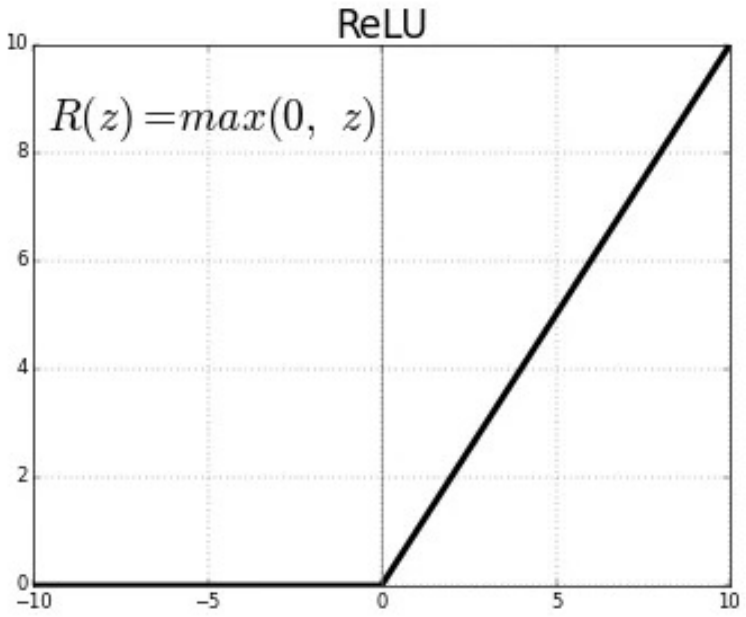
Activation Function - Tanh



The advantage is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the tanh graph.

Activation Function – ReLU (Rectified Linear Unit)

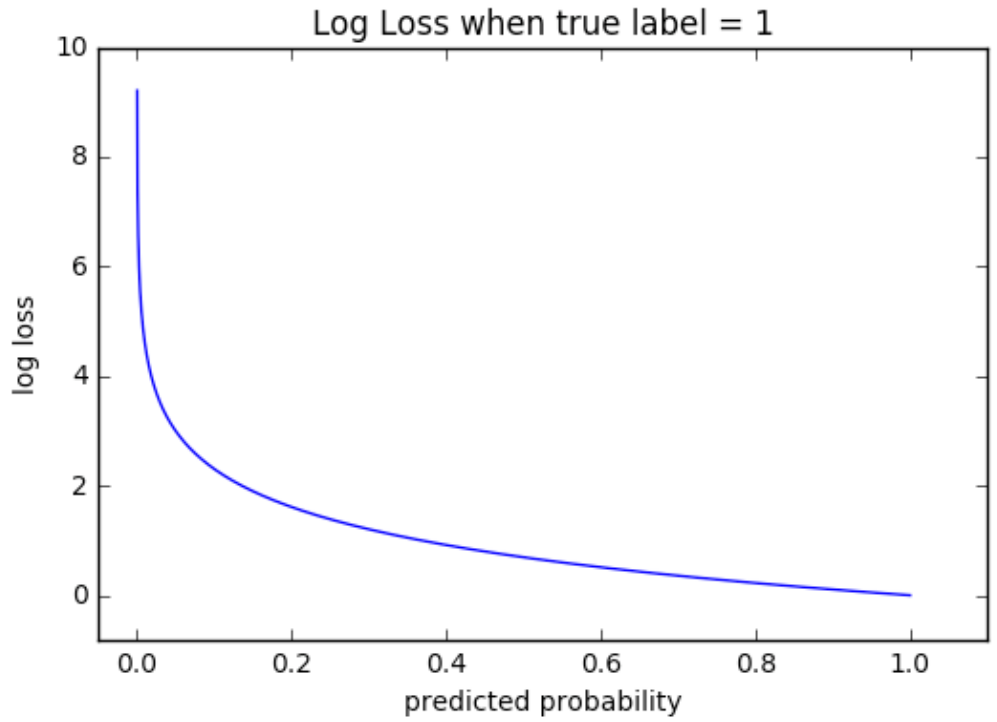




Other Loss Functions – Cross Entropy/Negative Log Likelihood

$$CrossEntropyLoss = -(y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i))$$

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$



Training Process – from scratch

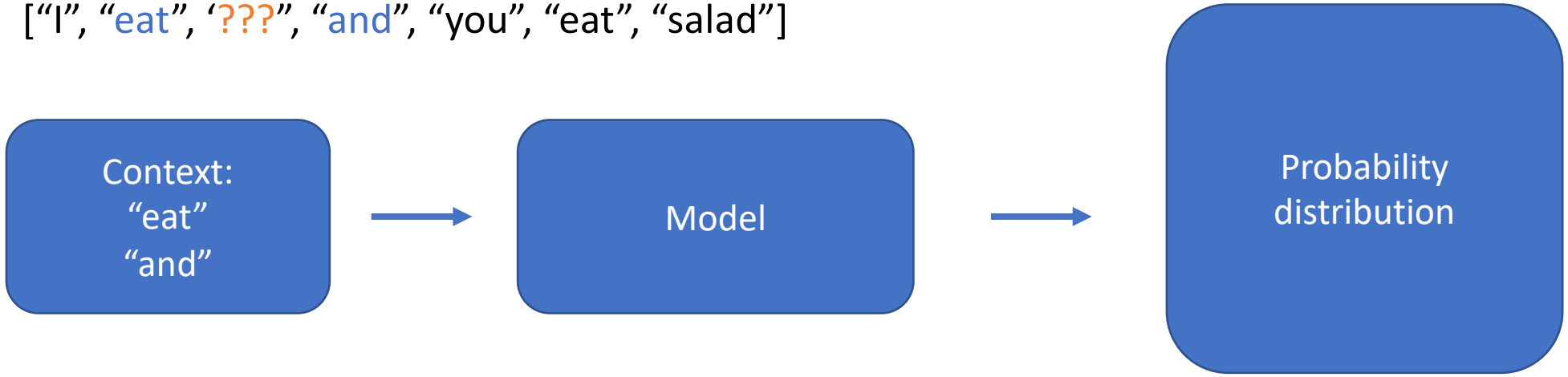
```
# Initialize parameters
w = torch.normal(0, 0.01, size=(2,1), requires_grad=True)
b = torch.zeros(1, requires_grad=True)

# Define the model:
def linreg(X, w, b):
        return torch.matmul(X, w) + b

# Define loss function:
def squared_loss(y_hat, y):
        return (y_hat - y.reshape(y_hat.shape)) ** 2 / 2

# Define optimizer
def sgd(params, lr, batch_size):
        with torch.no_grad():
                for param in params:
                    param -= lr * param.grad / batch_size
                    param.grad.zero_()
```

Training Process (Cont.)

```
# hyperparameters
lr = 0.03
num_epochs = 3
net = linreg
loss = squared_loss
# train the model
for epoch in range(num_epochs):
        for X, y in data_iter(batch_size, features, labels):
                l = loss(net(X, w, b), y)
                l.sum().backward()
                sgd([w, b], lr, batch_size)
        with torch.no_grad():
                train_l = loss(net(features, w, b), labels)
```

Word2vec – a prediction problem

["I", "eat", '???', "and", "you", "eat", "salad"]

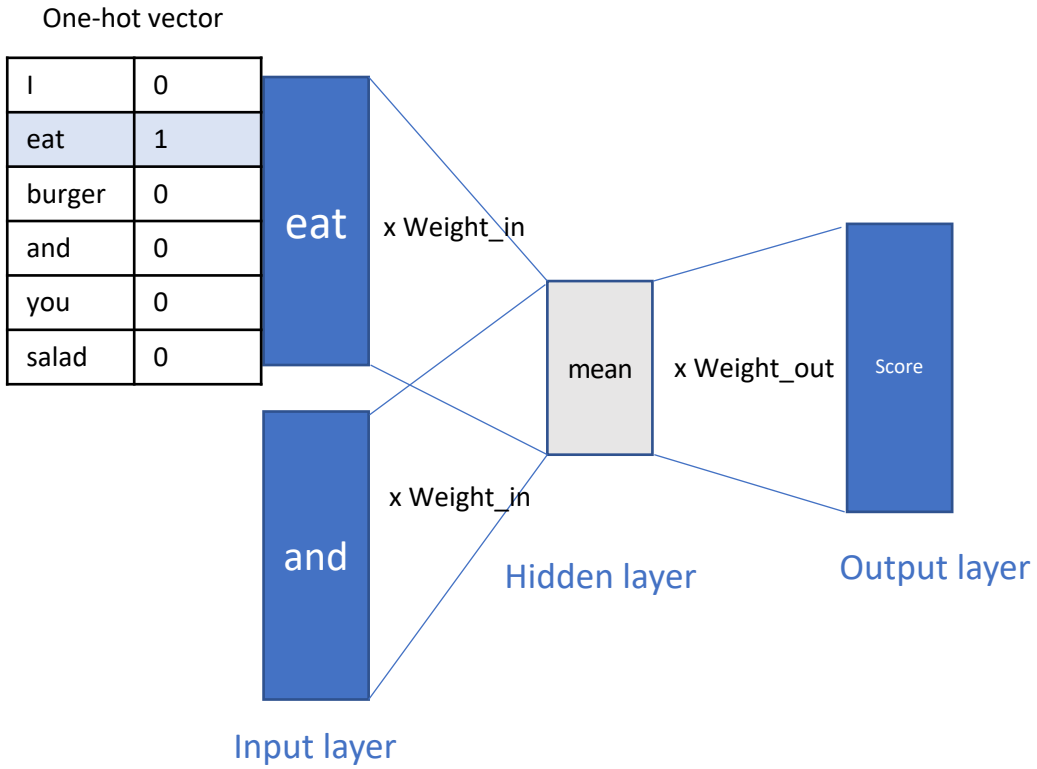


- One-hot vector:

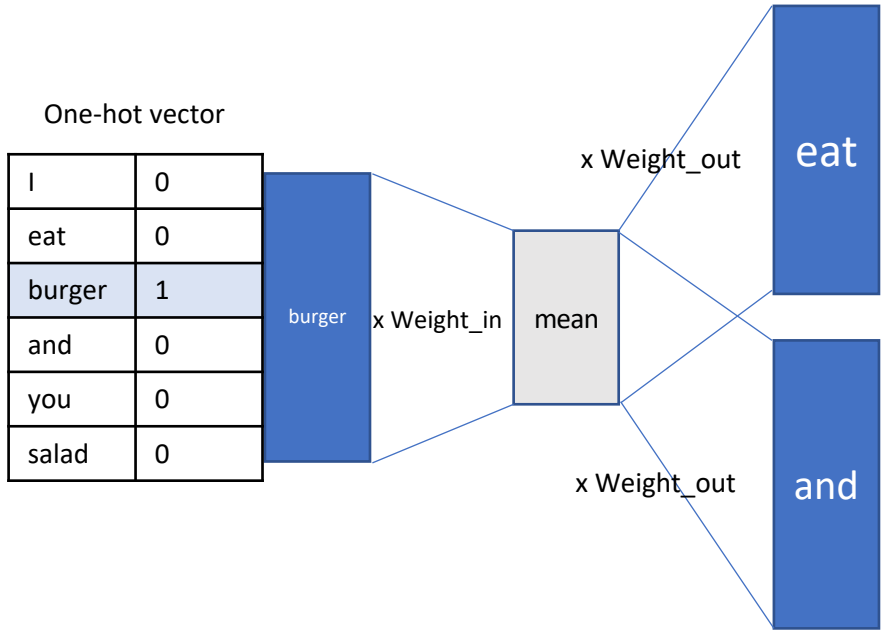
ID	Word	I	eat	burger	and	you	salad
2	eat	0	1	0	0	0	0

Word2vec

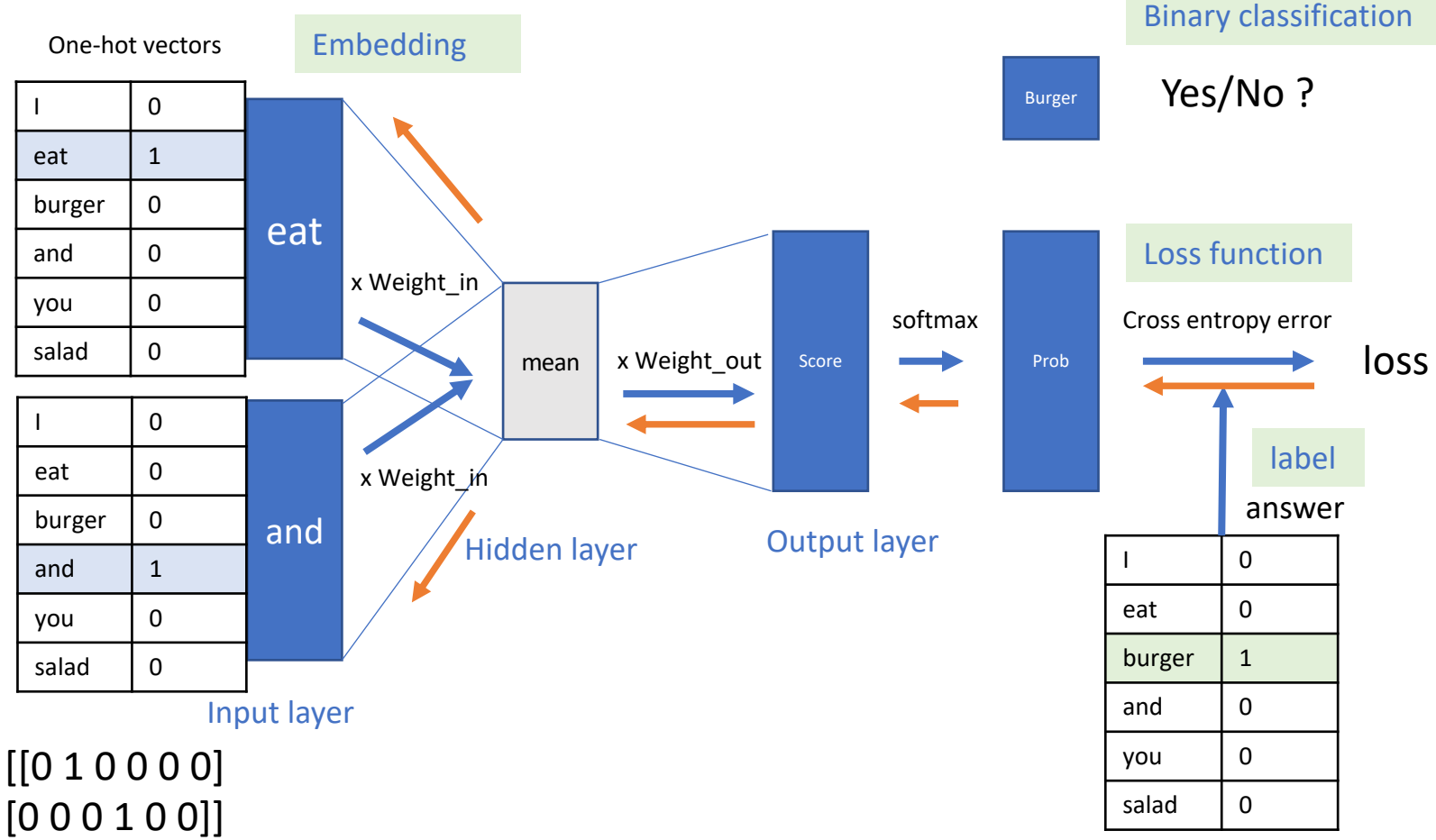
CBOW



Skip-gram



CBOW Learning Process



Softmax – get probability

$$\mathrm{softmax}(\mathbf{X})_{ij} = \frac{\exp(\mathbf{X}_{ij})}{\sum_k \exp(\mathbf{X}_{ik})}.$$

Cross-entropy loss

$$l(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{j=1}^{q} y_j \log \hat{y}_j.$$

CBOW Result – Distributional Representation

- Weight_in matrix to represent words meaning
  - Syntax – plurals, past tenses…
  - Semantics
    - “king – men + women = queen”

```
[analogy] king:man = queen:?
 woman: 5.161407947540283
 veto: 4.928170680999756
 ounce: 4.689689636230469
 earthquake: 4.633471488952637
 successor: 4.6089653968811035
```

```
[analogy] take:took = go:?
 went: 4.548568248748779
 points: 4.248863220214844
 began: 4.090967178344727
 comes: 3.9805688858032227
 oct.: 3.9044761657714844
```

```
[analogy] car:cars = child:?
 children: 5.217921257019043
 average: 4.725458145141602
 yield: 4.208011627197266
 cattle: 4.18687629699707
 priced: 4.178797245025635
```

Language Model

- Language model: the probability of a sequence of words.

$$P(w_1, \cdots, w_m) = P(w_m \,|\, w_1, \cdots, w_{m-1}) P(w_{m-1} \,|\, w_1, \cdots, w_{m-2})$$

$$\cdots P(w_3 \,|\, w_1, w_2) P(w_2 \,|\, w_1) P(w_1)$$

$$= \prod_{t=1}^{m} P(w_t \,|\, w_1, \cdots, w_{t-1}) \quad ①$$

$$P(A,B) = P(A|B)P(B)$$

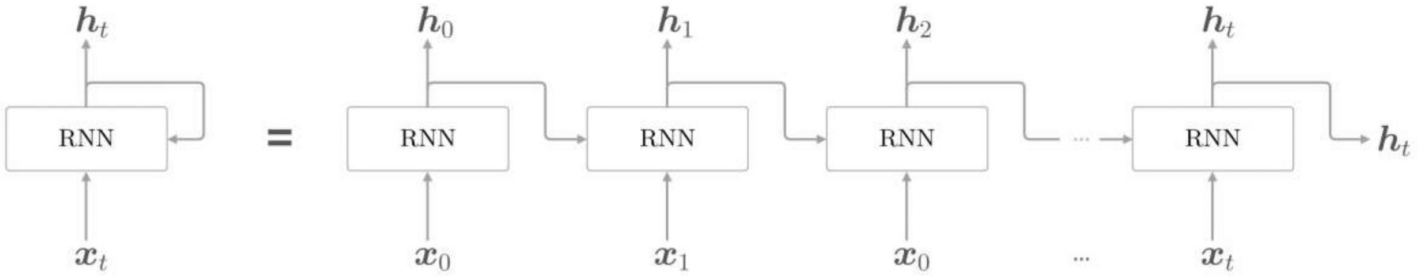
Recurrent Neural Networks

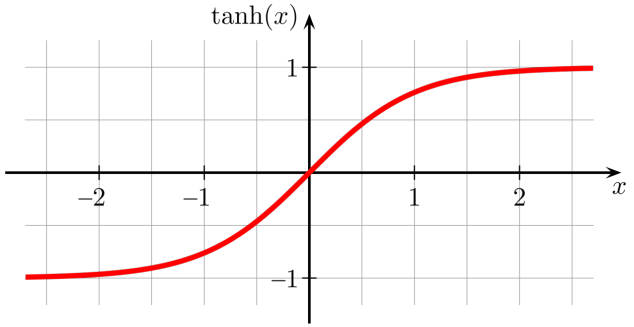
- Simple RNN
- LSTM
- Seq2seq
- Attention
- Transformer

Simple RNN

Hidden state

Truncated BPTT – Truncated Backpropagation Through Time

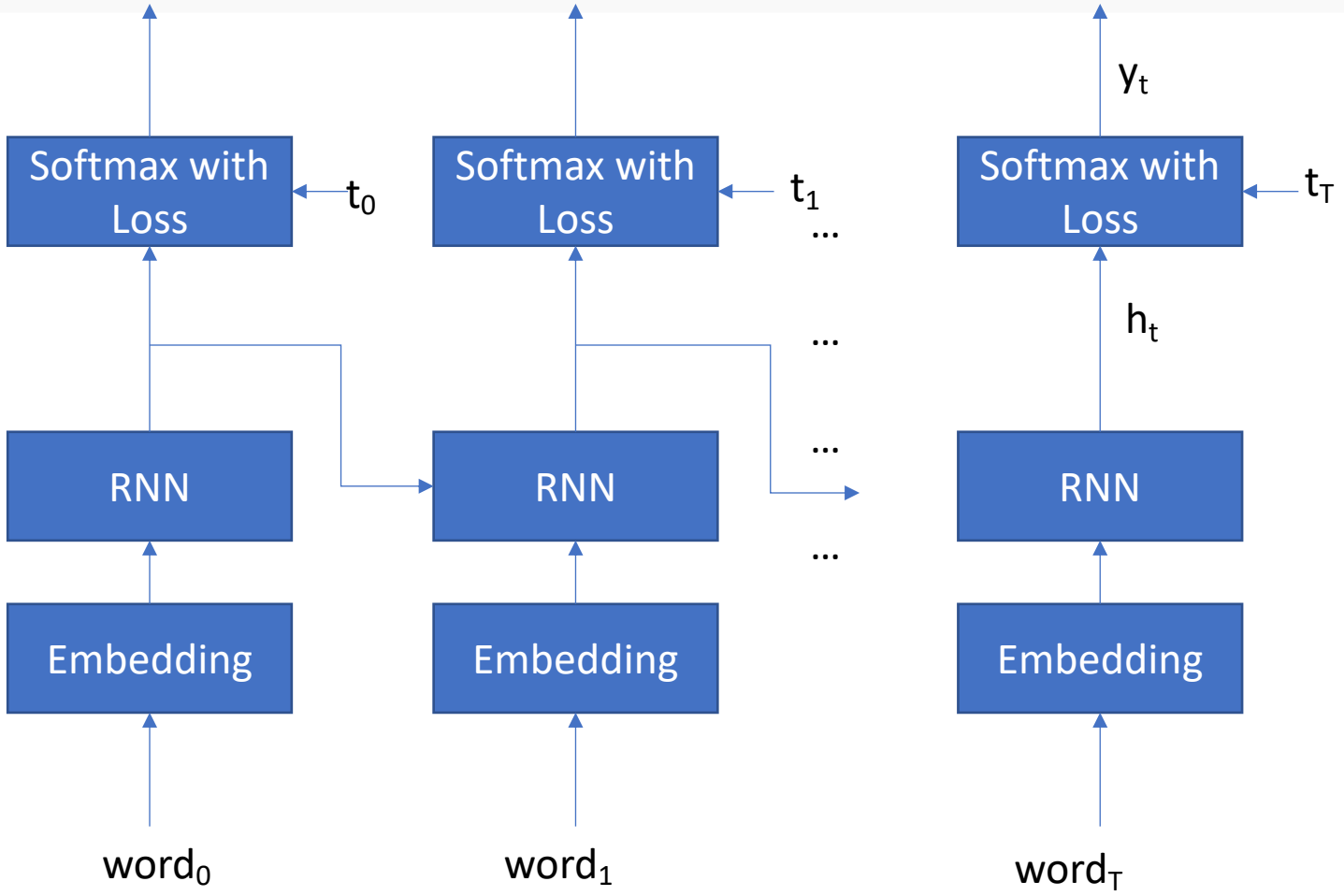




$$h_t = tanh(h_{t-1}W_h + x_tW_x + b)$$

Output = $W_h * h_t$

Simple RNN

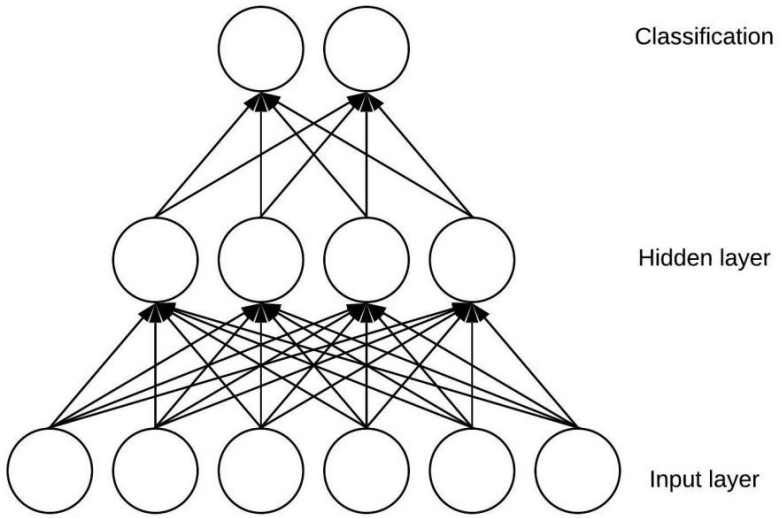


$$L = -\frac{1}{N}\sum_{n}\sum_{k} t_{nk} \log y_{nk}$$

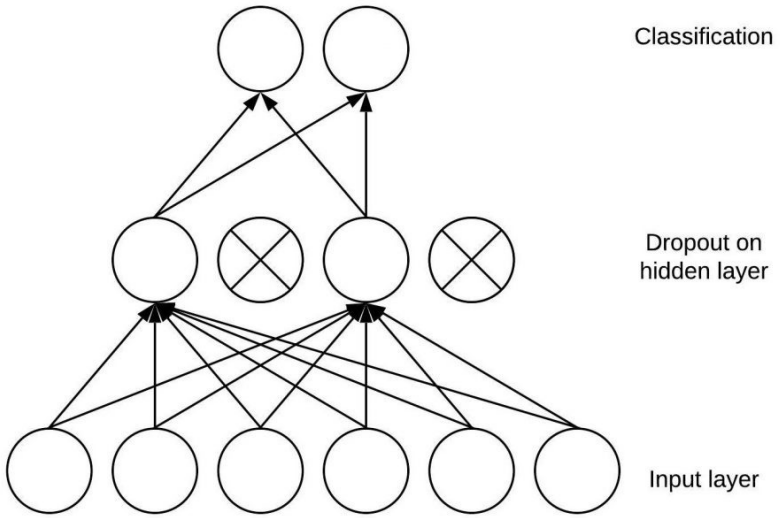
$$\text{Perplexity} = e^{L}$$

Perplexity: the number of possible candidates

Additional Layers - Dropout



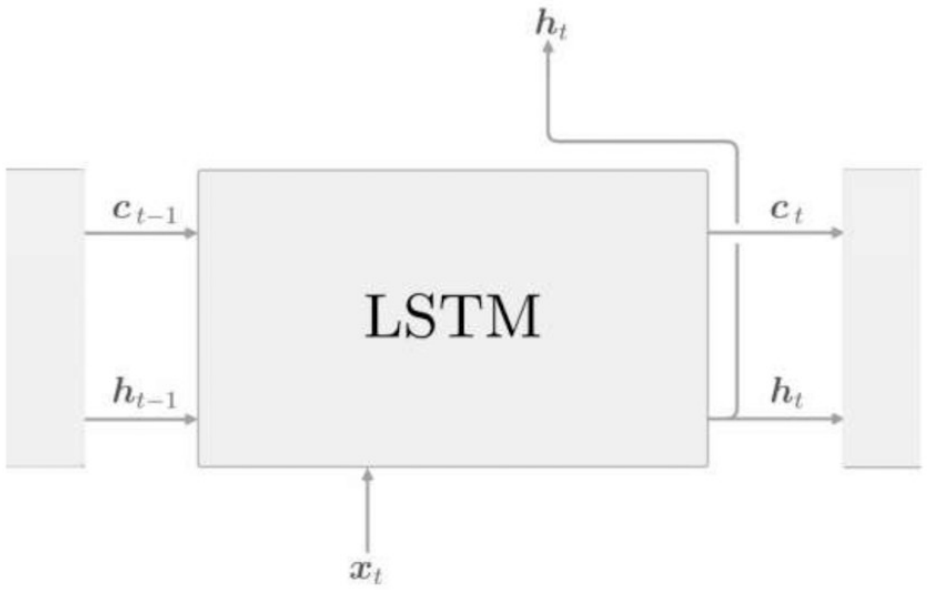
Without Dropout

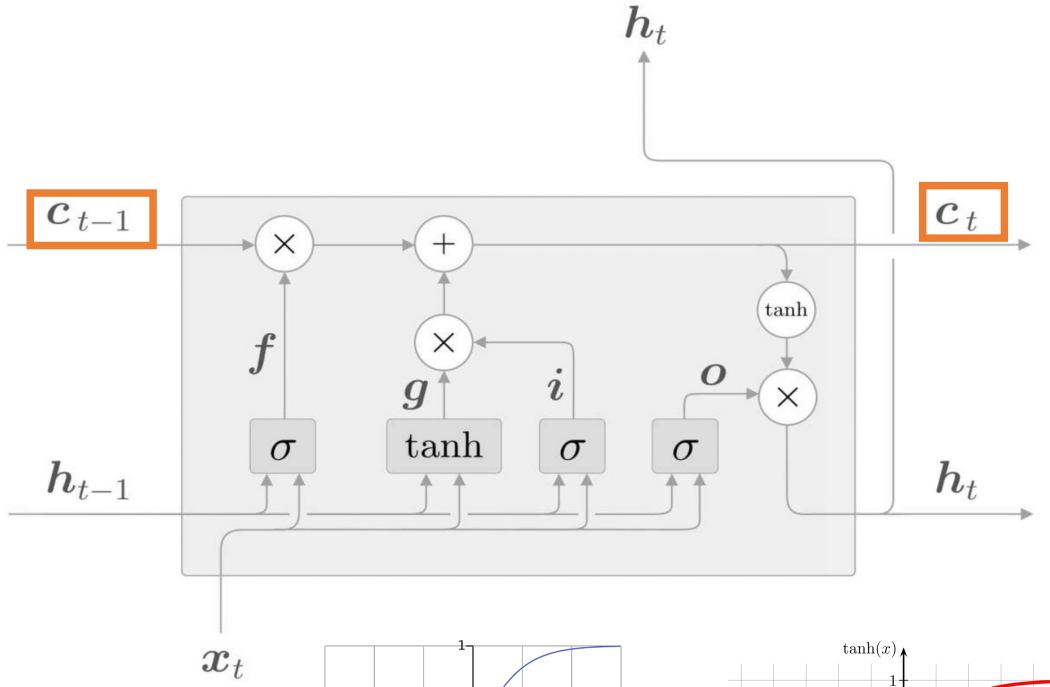


With Dropout

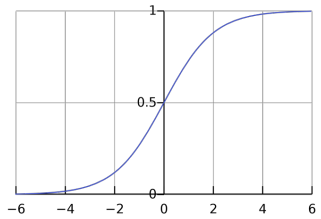
Prevent model Overfitting

LSTM – Long Short-Term Memory (Gated RNN)

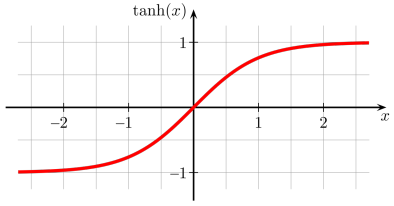




Controls gates



Sigmoid(x)



Tanh(x)

LSTM – Long Short-Term Memory (Gated RNN)

- Output gate

$$\boldsymbol{o} = \sigma(\boldsymbol{x}_t \boldsymbol{W}_x^{(\mathrm{o})} + \boldsymbol{h}_{t-1} \boldsymbol{W}_h^{(\mathrm{o})} + \boldsymbol{b}^{(\mathrm{o})})$$

- Forget gate

$$\boldsymbol{f} = \sigma(\boldsymbol{x}_t \boldsymbol{W}_x^{(\mathrm{f})} + \boldsymbol{h}_{t-1} \boldsymbol{W}_h^{(\mathrm{f})} + \boldsymbol{b}^{(\mathrm{f})})$$

- Memory gain

$$\boldsymbol{g} = \tanh(\boldsymbol{x}_t \boldsymbol{W}_x^{(\mathrm{g})} + \boldsymbol{h}_{t-1} \boldsymbol{W}_h^{(\mathrm{g})} + \boldsymbol{b}^{(\mathrm{g})})$$

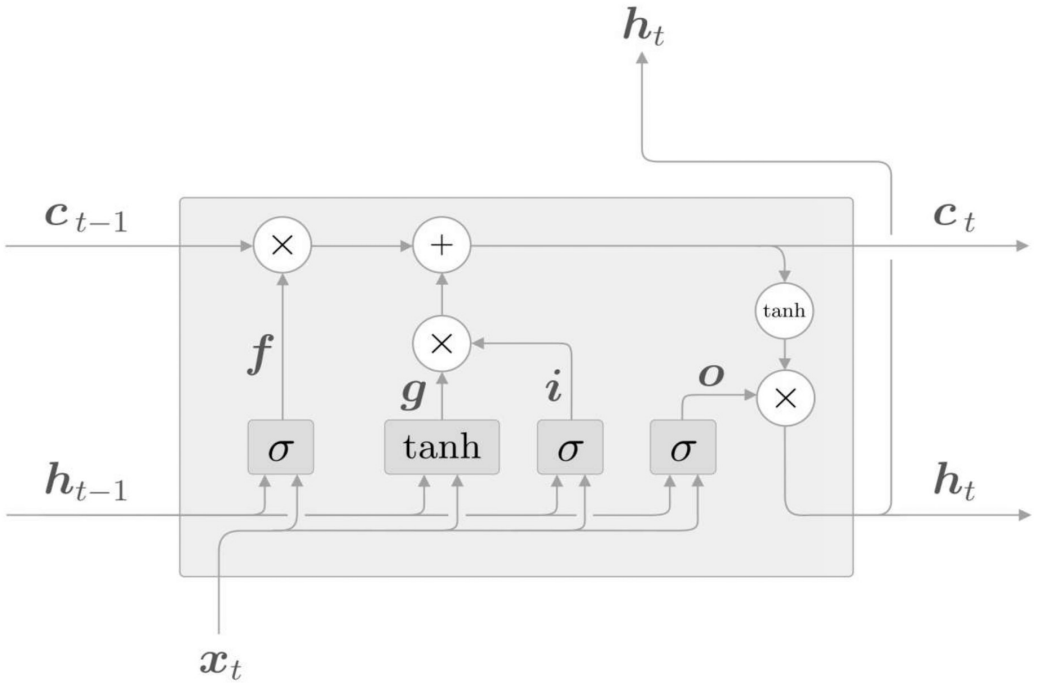
- Input gate

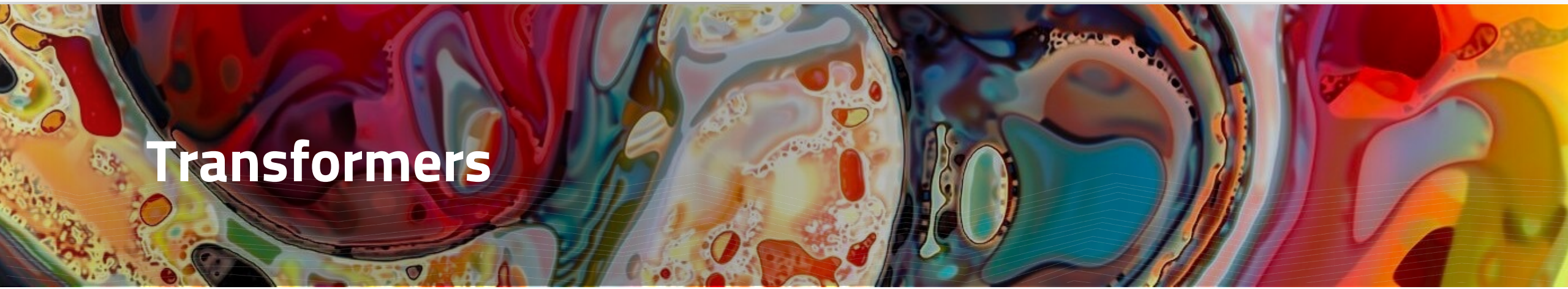
$$\boldsymbol{i} = \sigma(\boldsymbol{x}_t \boldsymbol{W}_x^{(\mathrm{i})} + \boldsymbol{h}_{t-1} \boldsymbol{W}_h^{(\mathrm{i})} + \boldsymbol{b}^{(\mathrm{i})})$$

Element-wise multiplication

$$\boldsymbol{h}_t = \boldsymbol{o} \odot \tanh(\boldsymbol{c}_t)$$

$$\boldsymbol{c}_t = \boldsymbol{f} \odot \boldsymbol{c}_{t-1} + \boldsymbol{g} \odot \boldsymbol{i}$$





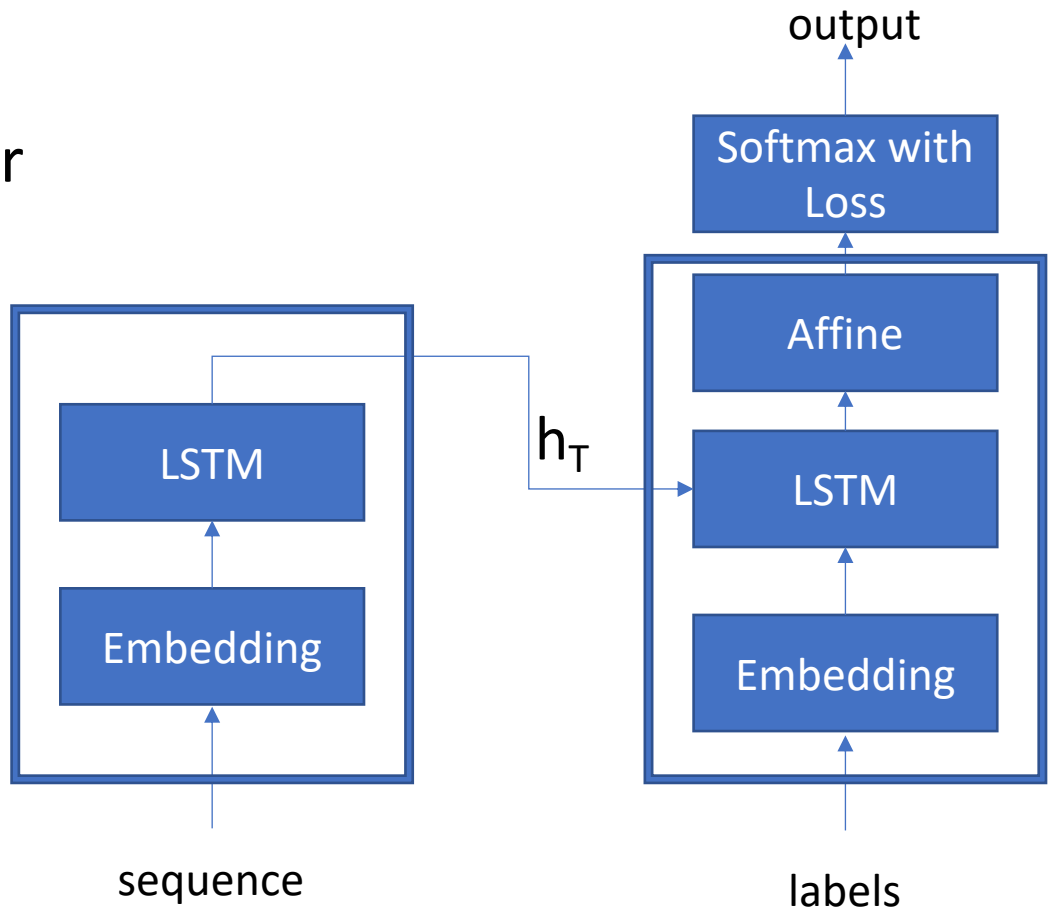
Transformers

- Transformer
- GPT
- Bert

Seq2Seq Model

Encoder

Decoder



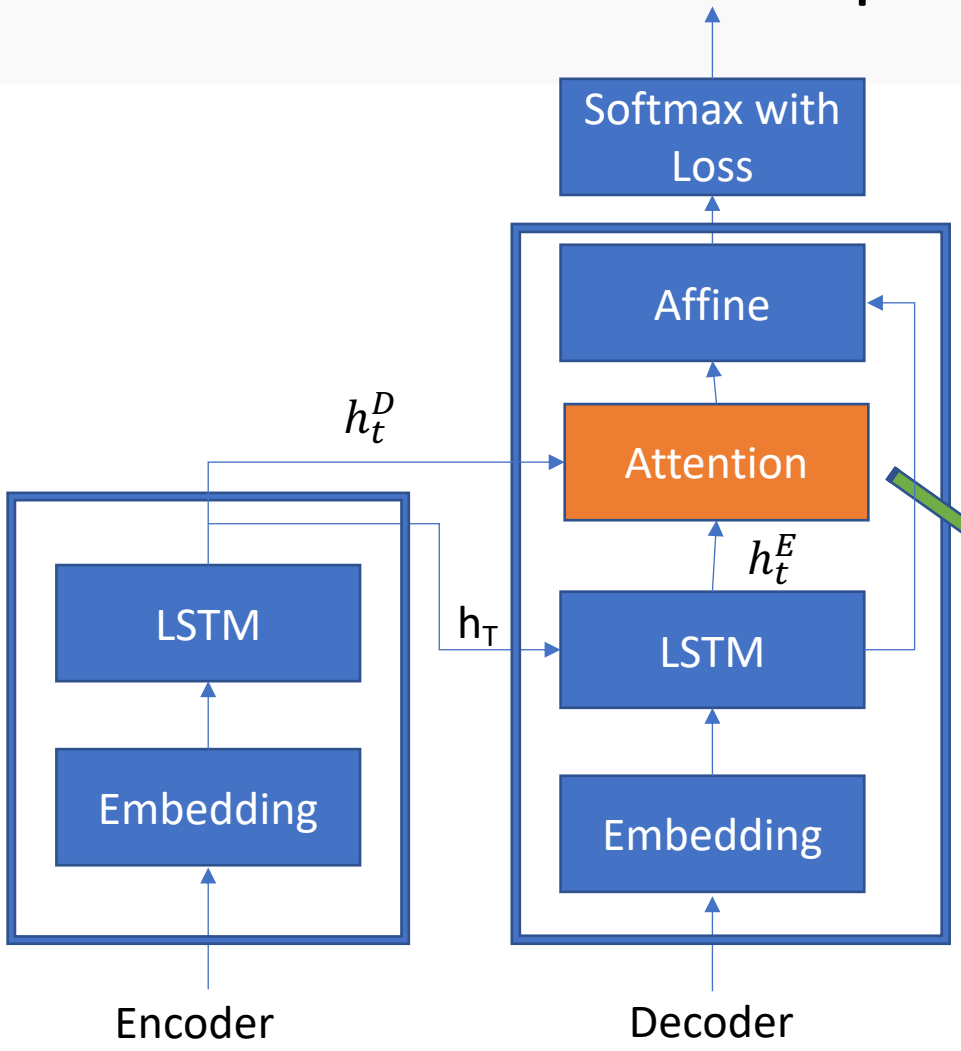
Applications:

- Translation
- Chat bot
- Summarization
- Image to caption

h_T - the last row of hidden state

Time direction repetition

Attention in Seq2Seq



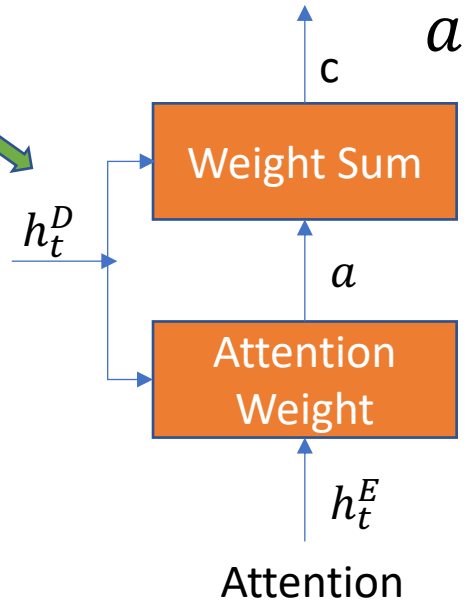
Weight sum

$sum(h_t^D \odot a) = c$ Context vector

a - probability distribution of each word's importance

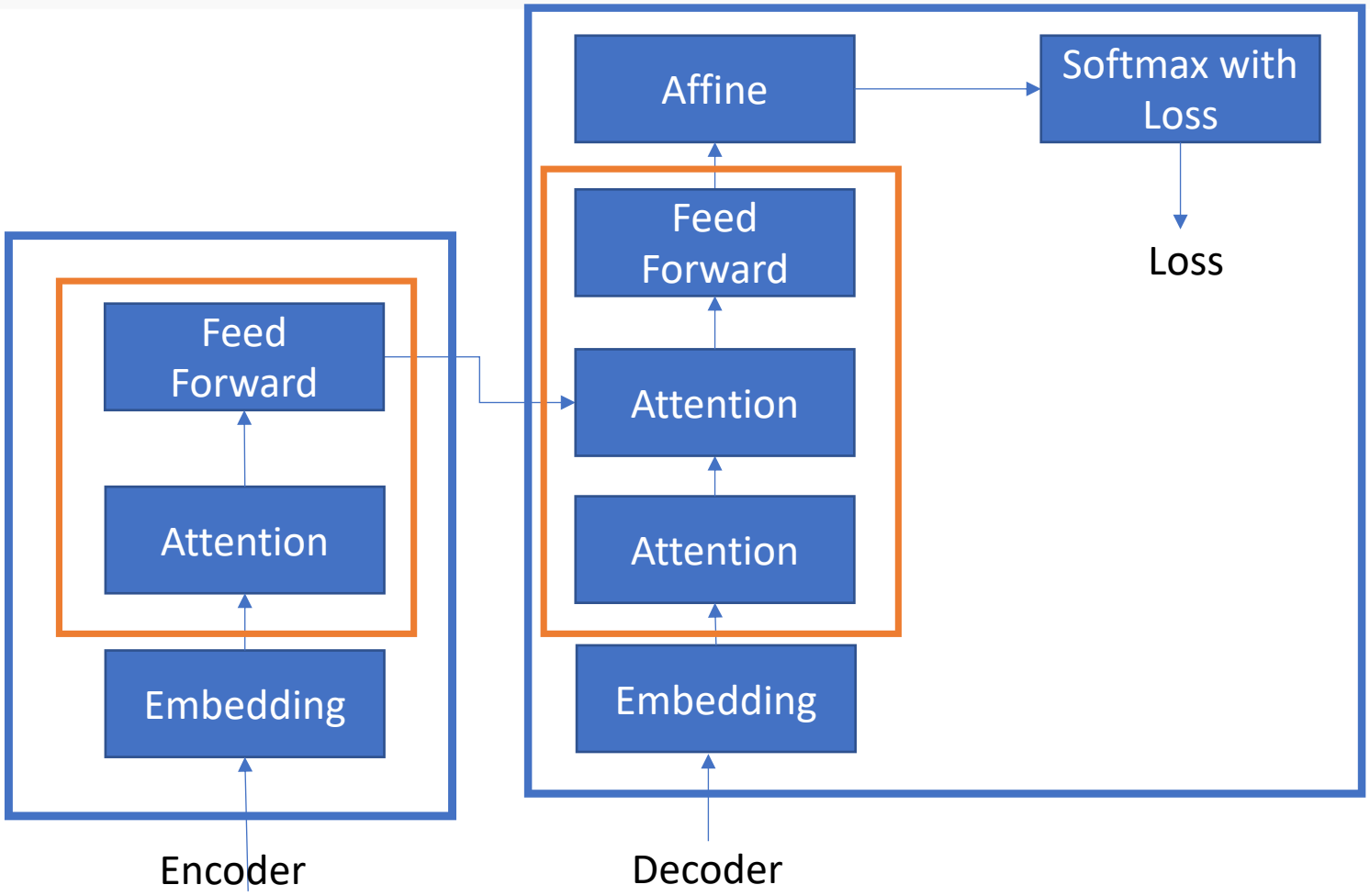
Attention weight

$$a = \text{softmax}\left(sum(h_t^D \odot h_t^E)\right)$$



e.g. vector similarity, here we use dot product

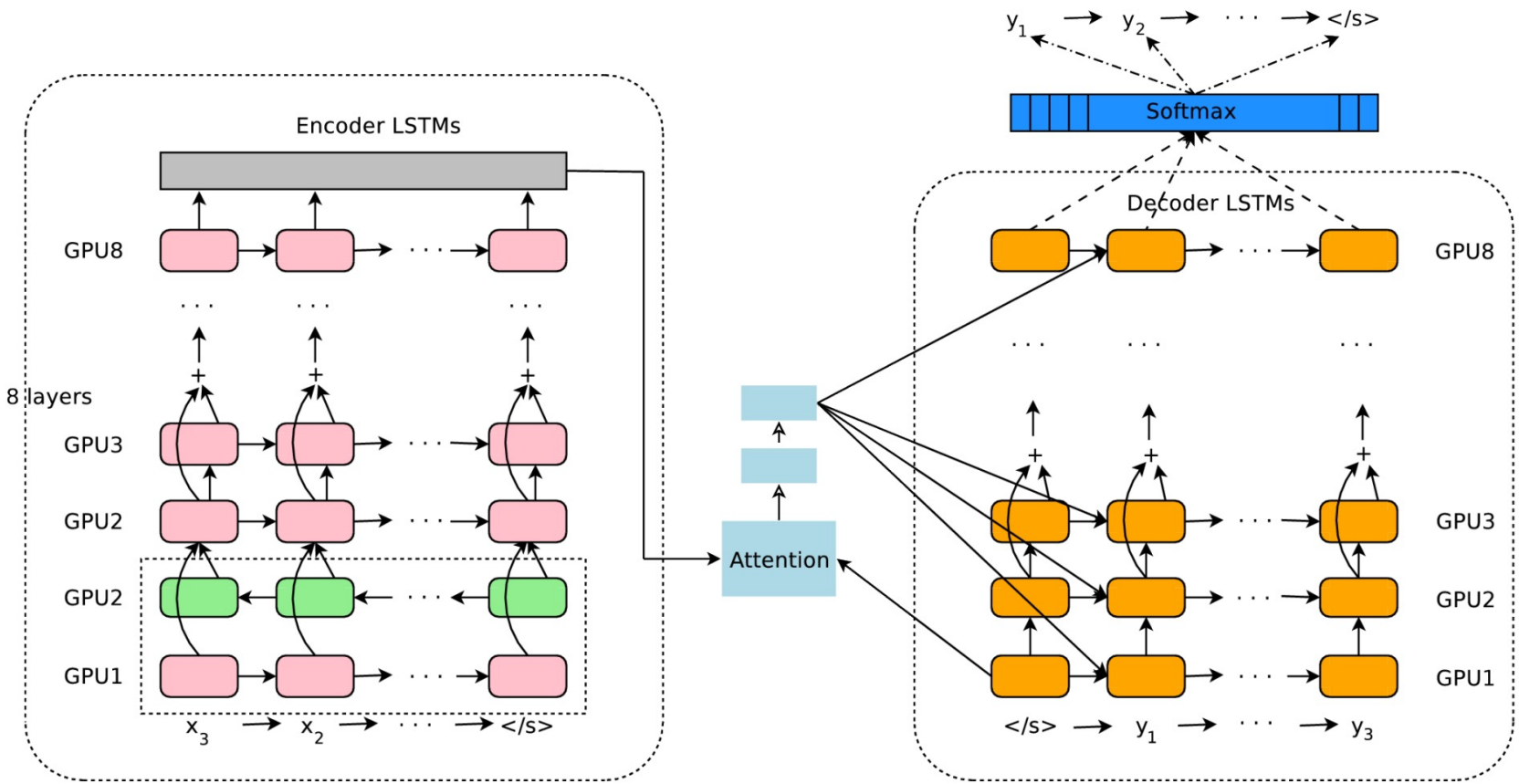
Transformer – Attention without RNN



- Self-attention instead of LSTM layer
- Multi-head attention

Depth repetition

Example. Google Neural Machine Translation



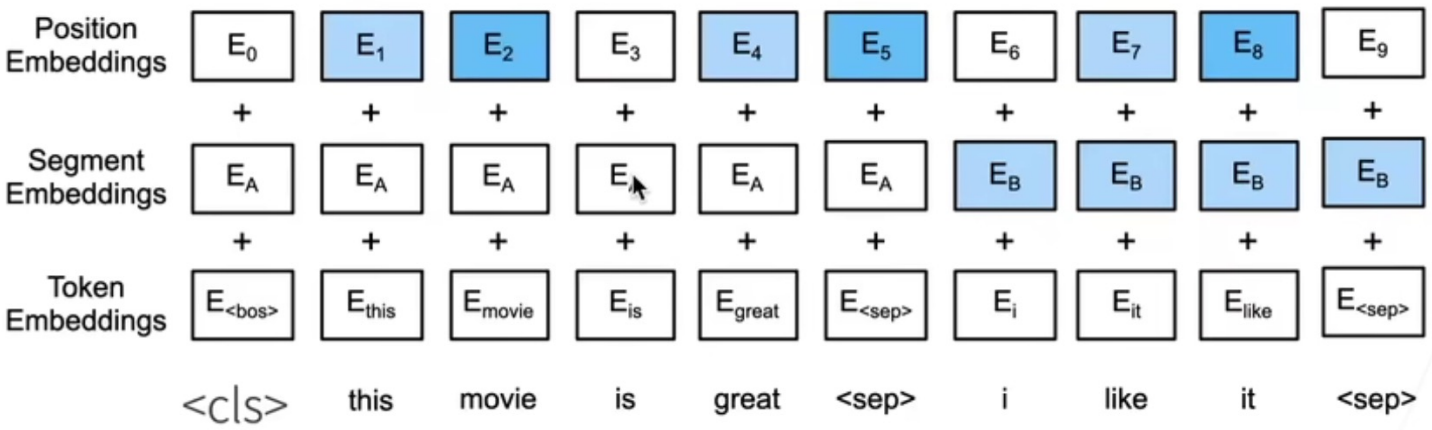
Wu, Yonghui, et al. Google's neural machine translation system:Bridging the gap between human and machine translation[J]. arXiv preprint arXiv:1609.08144, 2016.

Bert – NLP Model Based on Fine Tuning!

Bert – transformer without decoder

- Pre-trained model has extracted enough features
- Only need to replace output layer for a new task
- Original models in the paper:
  - Base: 12 (transformer encoder) blocks, hidden size = 768, 12 heads, 110M parameters
  - Large: 24 blocks, hidden size = 1024, 16 heads, 340M parameters
  - Trained on more than 3B words (whole Wikipedia and some books)

Bert

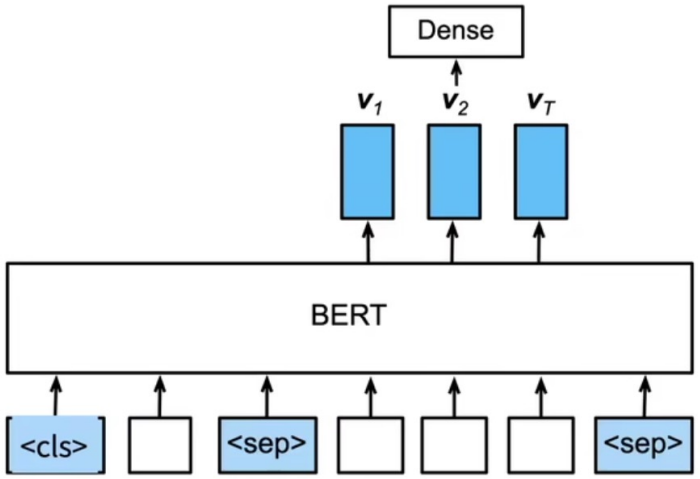


Pretraining:

- Masked language model - randomly mask some words in the sentence to predict
- Predict next sentence – 50% chance to select adjacent sentences as paired input, 50% chance to select random sentence pairs. Predict <cls> in output

1. Paired sentence input
2. Additional special tokens
3. Trainable position embedding

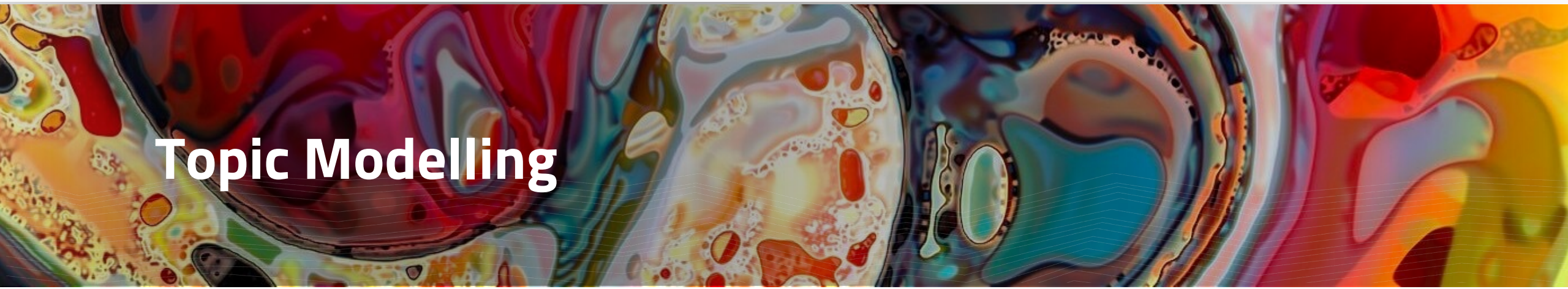
Bert – Q&A



Output:

- Token is the start of the answer
- Token is the end of the answer
- Neither

Fine tuning:
Increase the learning rate for output layer
Set some of the base layer parameters so finish training faster

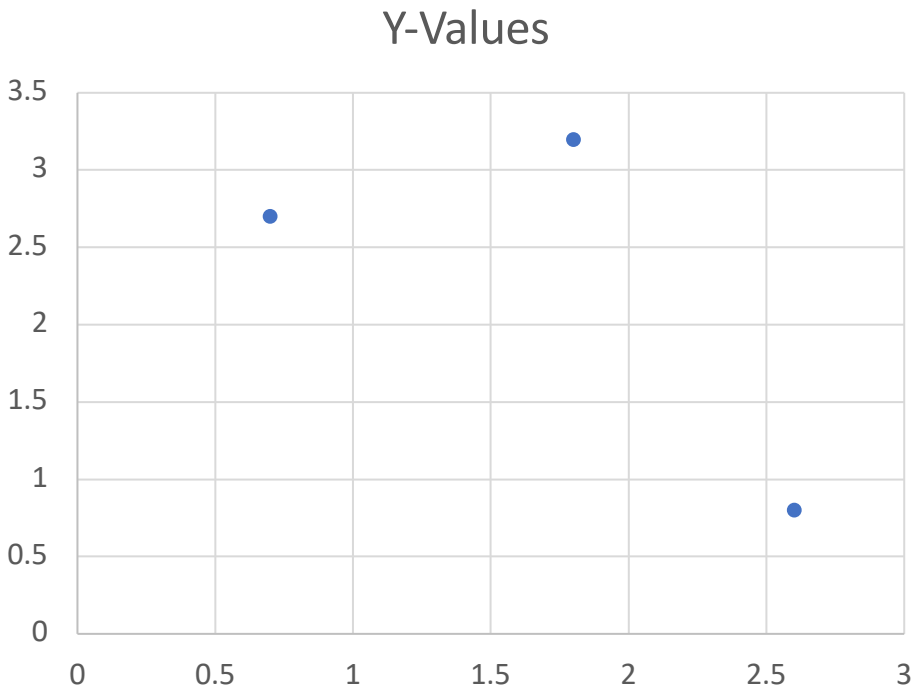


Topic Modelling

K-means Clustering

- One document → one topic

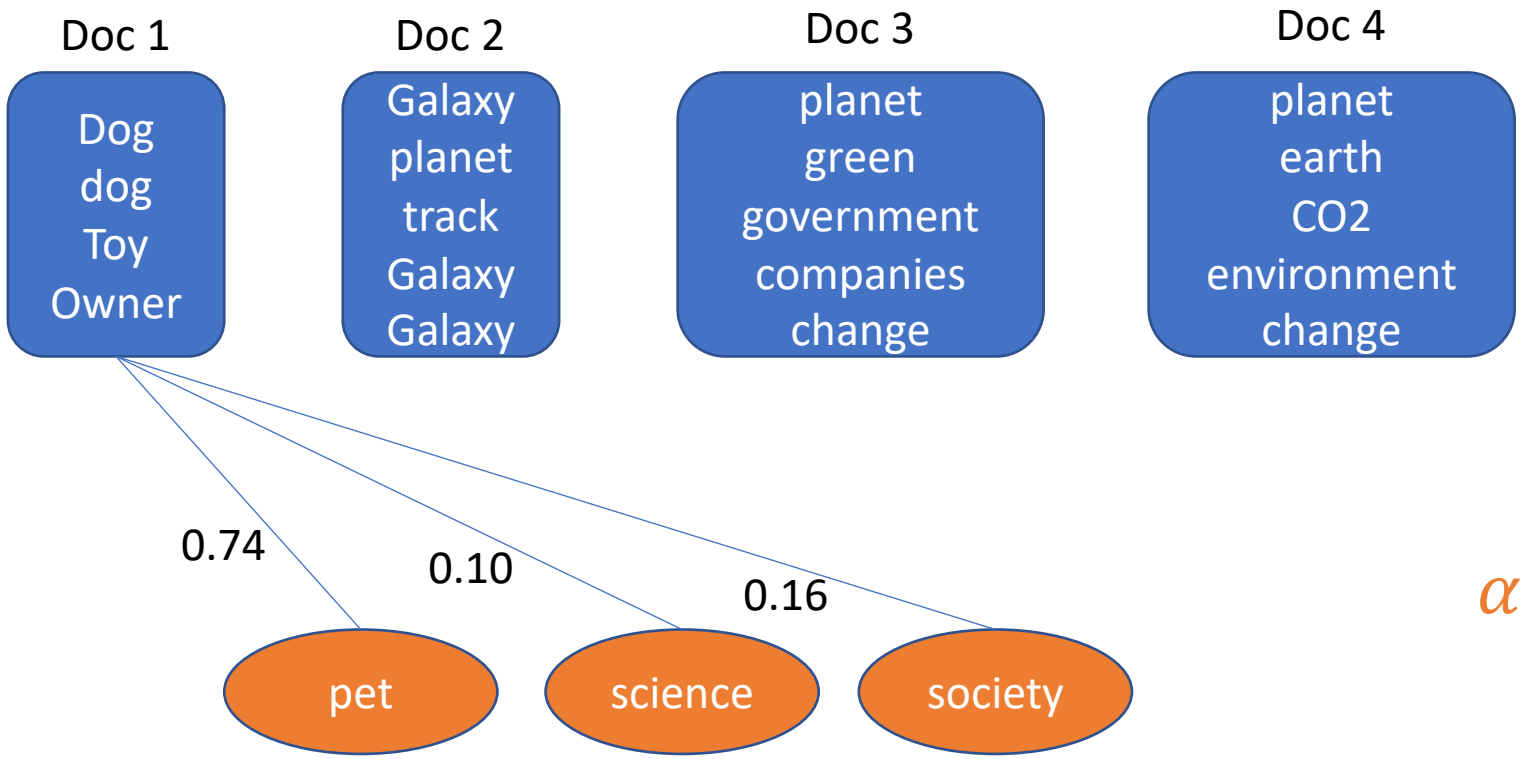
1. Set K – number of clusters
2. Randomly assign k points as the centroid of the clusers
3. Measure distance between point a and the k clusters
4. Assign point a to the cluster with the minimum distance
5. Repeat 3-4 for all data points
6. Recalculate the cluster centroid
7. Repeat 5-6 until the clusters don't change anymore
8. Calculate total clusters variance
9. Repeat 2-8 N times, result is the clustering with the minimum variance



Similarities in Text Vectors

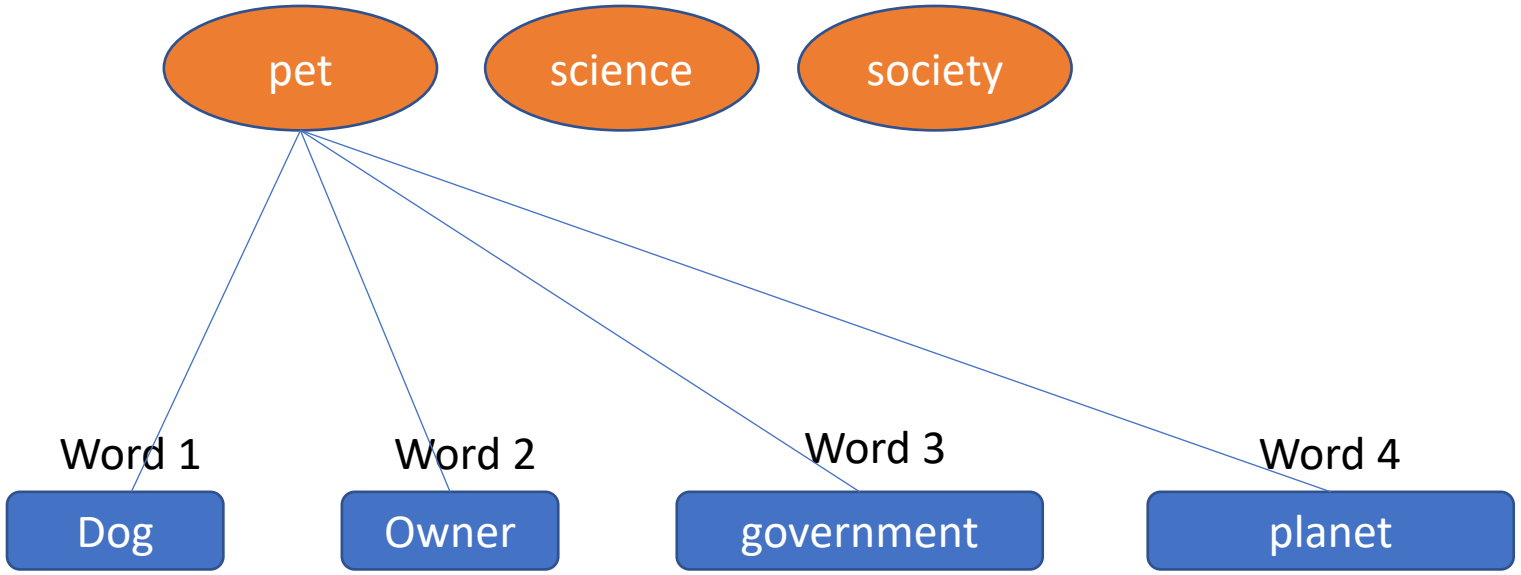
$$\text{cosine similarity} = S_C(A, B) := \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}},$$

LDA – Latent Dirichlet Allocation



α Documents-Topics Distribution

LDA – Latent Dirichlet Allocation

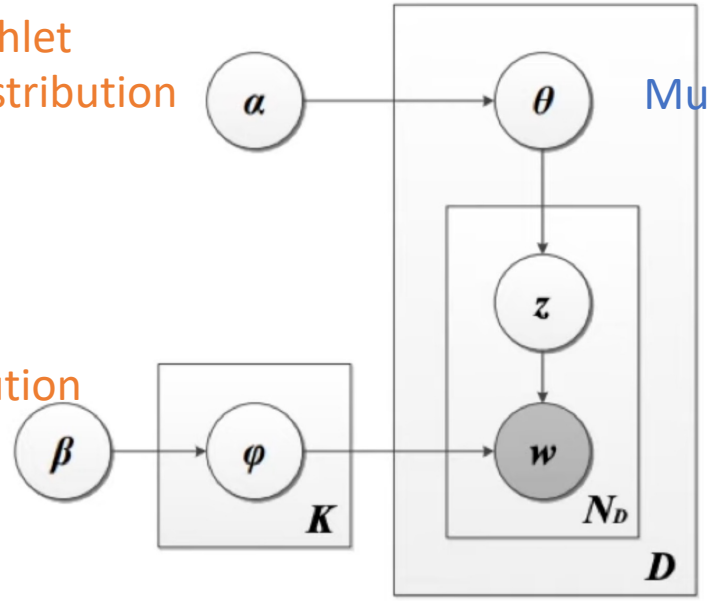


β Topics-Words Distribution

Goal:
1. each document has a main topic
2. Each word has a main topic

LDA – Latent Dirichlet Allocation

Dirichlet
Docs-topics distribution



Multinomial distribution – selecting a word from a document and it has the topic t

Chosen topics

Dirichlet
Topics-words distribution

Chosen words

Multinomial distribution –
Selecting a word from a topic and it is the word w

Goal:
1. each document has a main topic
2. Each word has a main topic
→ create a document as close to input document as possible

https://www.youtube.com/watch?v=T05t-SqKArY&list=TLPQMjQxMDIwMjLirqQmTCjo-w&index=1

tf-idf – Term Frequency-Inversed Document Frequency

- Term frequency – how often a word occurs in a document
- Document frequency – how often a word occurs in the corpus

$$\frac{Term\ Frequency}{Document\ Frequency}$$

Relevance/importance/uniqueness of a word to a document

Assumes a document as a “bag of words”

Dimensionality Reduction

- Plot performances for different number of topics and find the ‘elbow point’
- t-SNE – projecting highD data to lower dimension with cluster information
- PCA

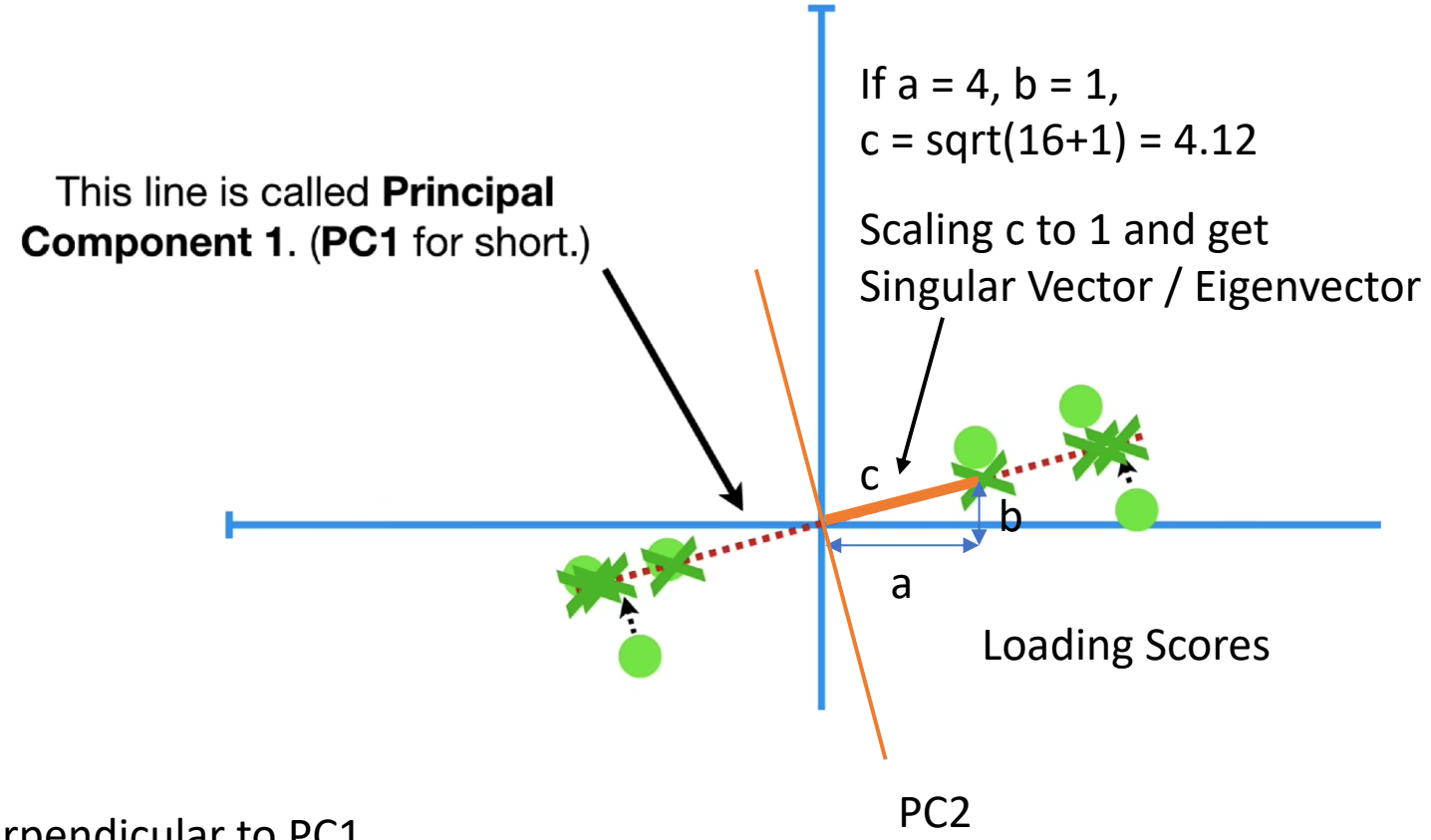
Principle Component Analysis - PCA

If you have data with more than 3 attributes in each record and you want to cluster the data using all attributes…
And you also want to plot the clustering result

- Using Singular Value Decomposition – SVD
- Transform 4+ Dimension data to 2D plot
- Telling us which dimension/feature is most important for clustering
- Telling us the accuracy of the 2D plot

PCA

1. Find the line that fits the data points and goes through origin by
    1. finding the maximum distance from origin to projected points on the line
    2. Sum of square of those distances (Eigenvalue for PC 1, actually variation)
    3. The line has slope of k, in this case, our data points are mostly spread out along x axis. Now we have a linear combination of x and y dimensions.

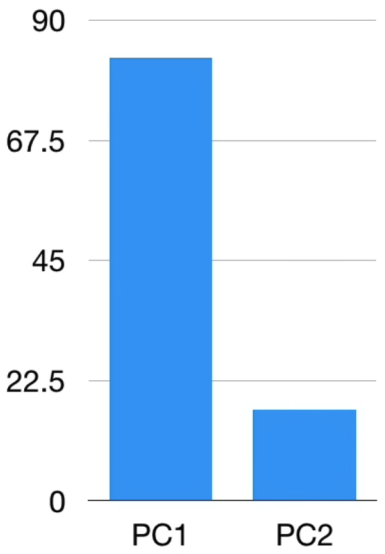


2. Find PC2 that goes through the origin and perpendicular to PC1

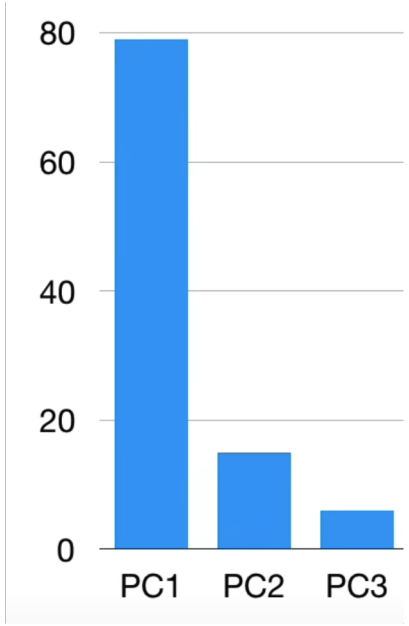
(3. Find PC3 that goes through origin and perpendicular to PC1 and PC2…)

PCA

Calculate Eigenvalues (sum of square of projection point distances) of each PC -> contribution of variation from each PC



Scree plot



Then we can choose to plot only using PC1 and PC2 for a 2D representation of the data.

Note:
Scale your data - dividing each variable by standard deviation.

- If your variables have very different scales, the result will be biased

References

- *Deep Learning from Scratch 2 © 2018 Koki Saitoh, O' Reilly Japan, Inc.*
- *Wu, Yonghui, et al. Google's neural machine translation system:Bridging the gap between human and machine translation[J]. arXiv preprint arXiv:1609.08144, 2016.*
- *https://www.youtube.com/watch?v=T05t-SqKArY&list=TLPQMjQxMDIwMjLirqQmTCjo-w&index=1*
- *https://www.youtube.com/watch?v=6ArSys5qHAU*