NLP Recap

Machine Learning

Overview: Machine Learning

Supervised Learning:

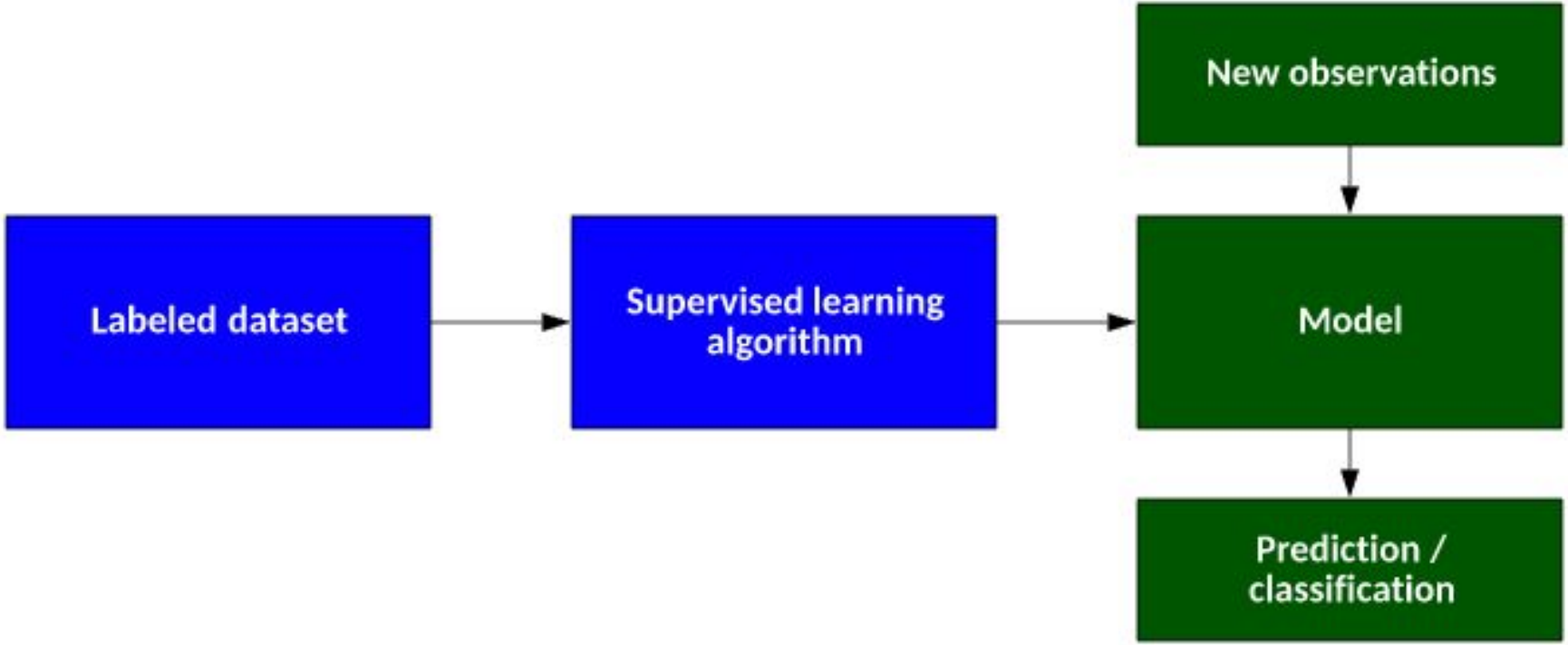
Classification, Regression

Unsupervised learning: k-means

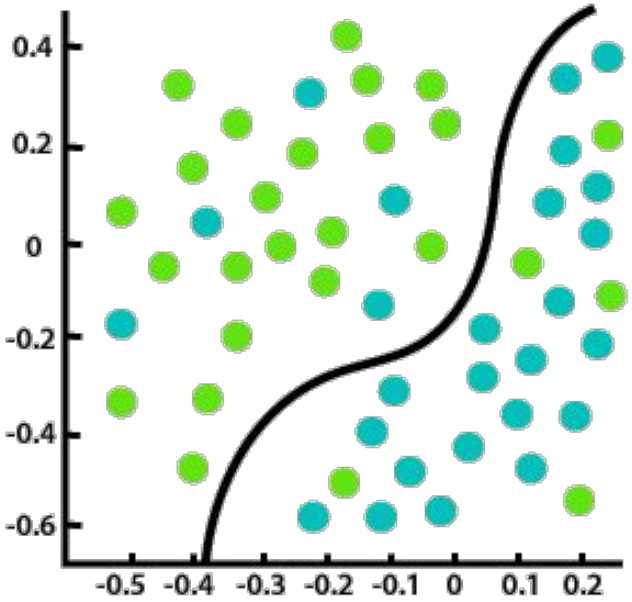
Programming Example and Assignment

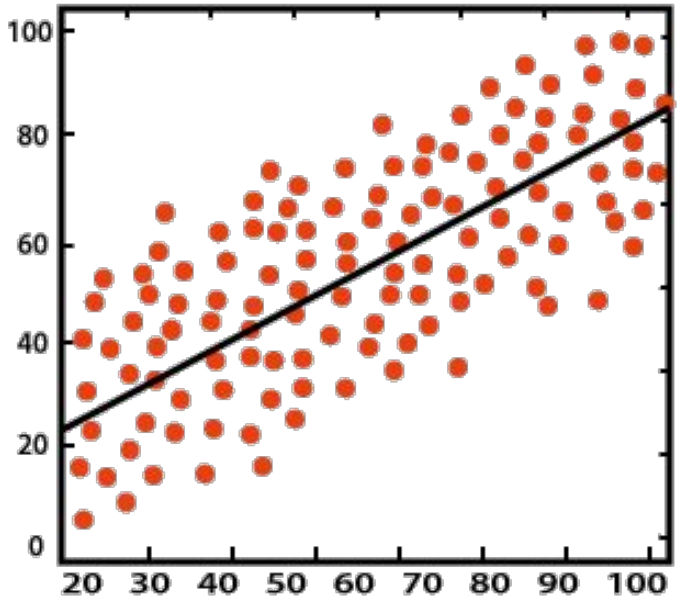
Overfitting, Training Strategy

Supervised Learning: Work with Labeled data



Supervised Learning tasks





Classification

Regression

Regression

White Board

Use Linear Regression -> Logistic Regression (to solve classification)

Computer Solves the parameters

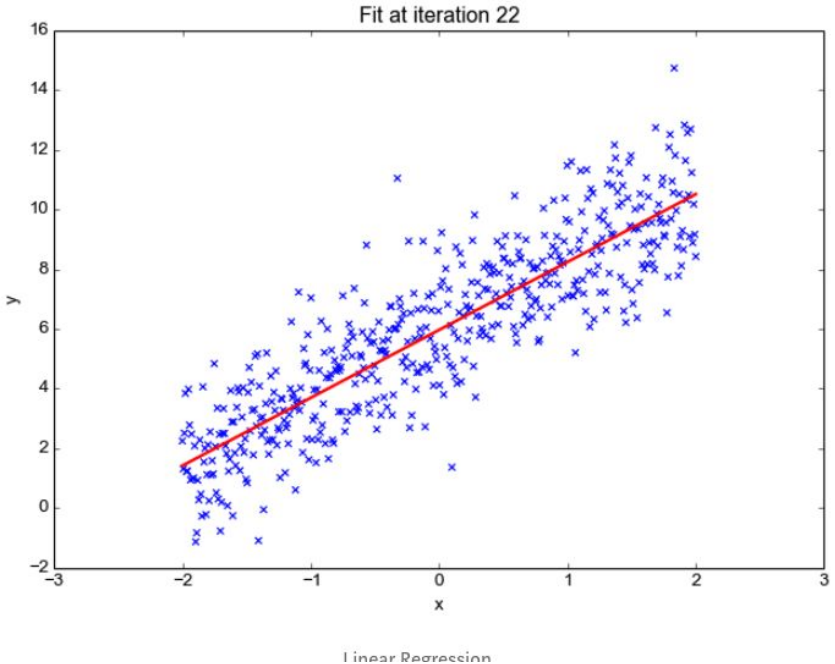
----Machine is “learning”

A toy example: linear regression

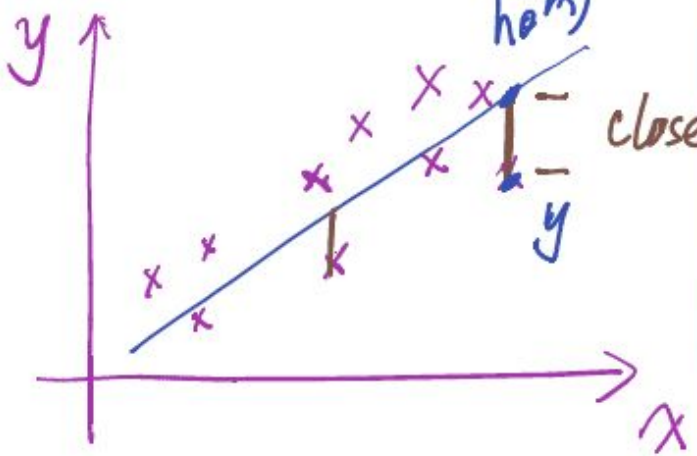
(x,y)

Y = k x

White Board



Linear Regression



$\hat{y} = wx + b$ $h_\theta(x)$ y

loss function?

Mean Square Error

$$L = \frac{1}{n} \sum_{i} \left| y_i - h_\theta(x_i) \right|^2$$

$$\min_{\theta} L$$

Optimization.

"Gradient Descent"

Cost Function

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 +$$

Metrics - Regression

Mean Absolute Errors & **M. Square E.**

$$MAE(y, \hat{y}) = \frac{1}{n_samples} \sum |y_i - \hat{y}_i|$$
$$MSE(y, \hat{y}) = \frac{1}{n_samples} \sum |y_i - \hat{y}_i|^2$$

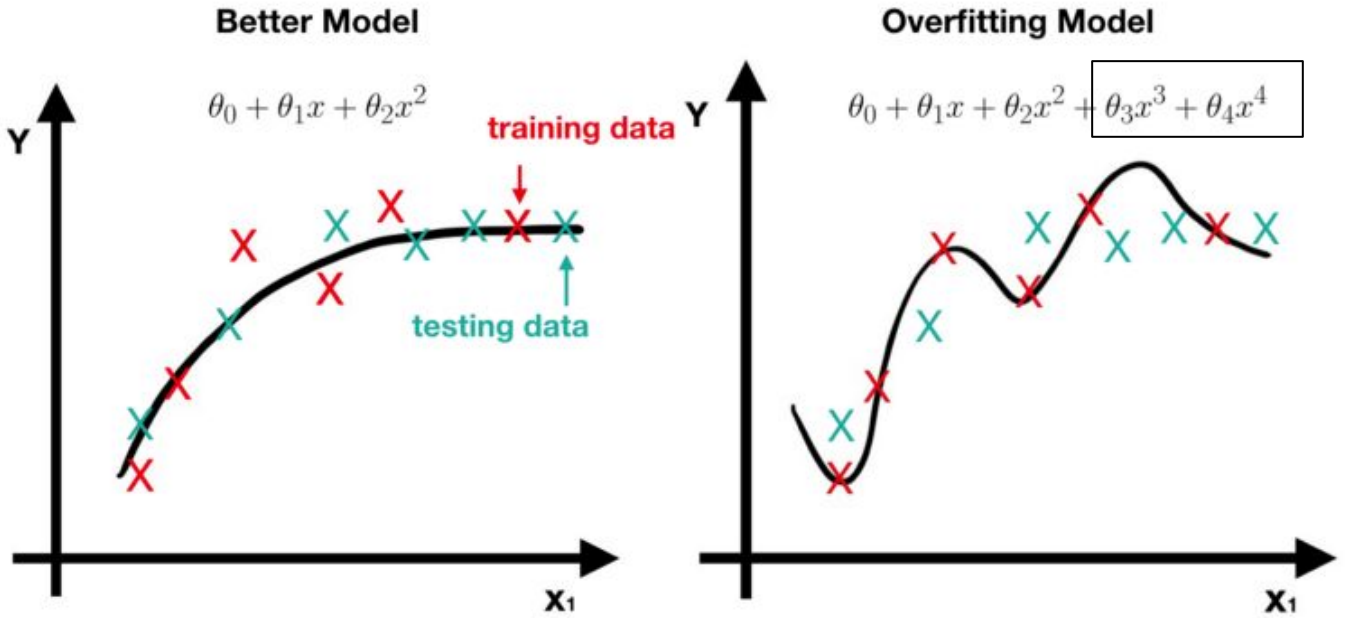
R-squared error (coefficient of determination):

$$R^2(y, \hat{y}) = 1 - \frac{\sum (y_i - \hat{y}_i)^2}{\sum (y_i - \bar{y}_i)^2}$$

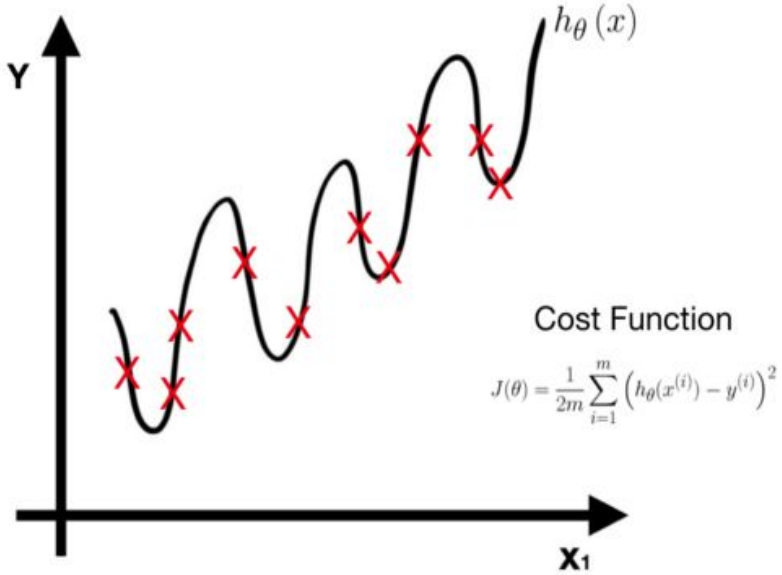
A higher value means a good model.

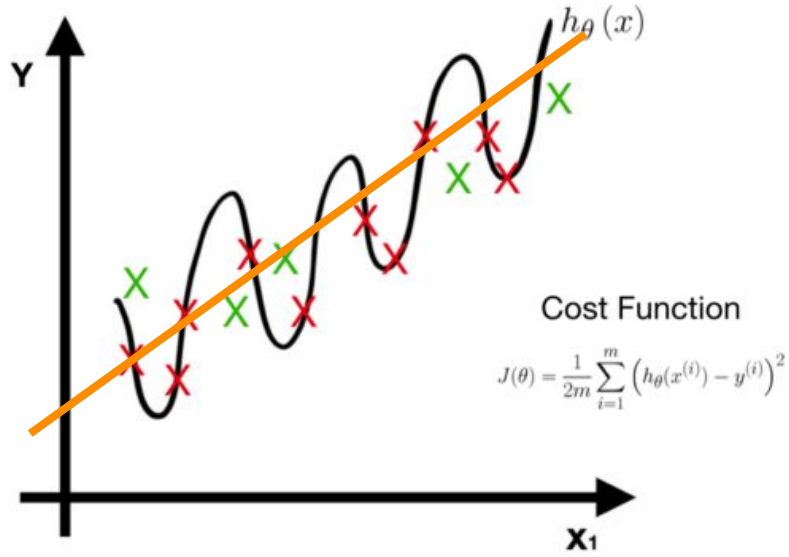
Problem of linear regression

Training error is small, but testing error is large: **overfitting**

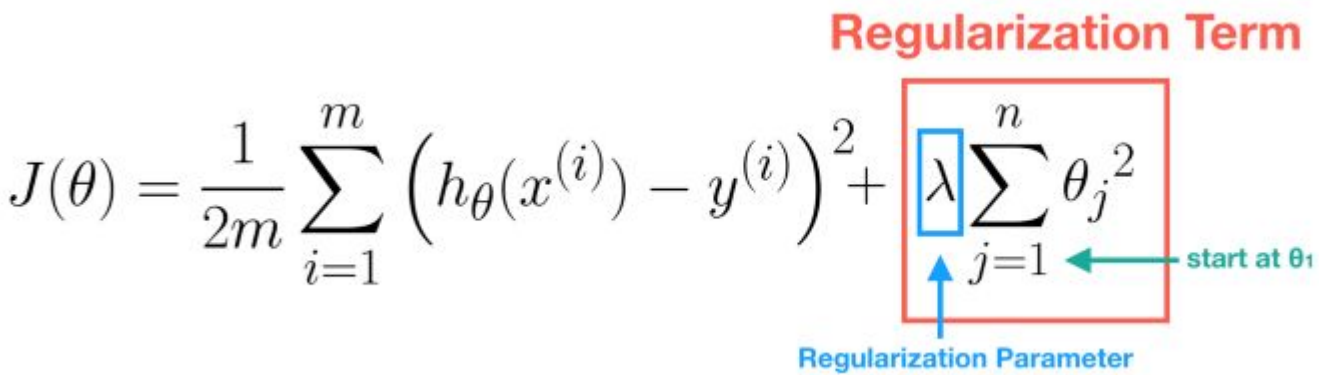


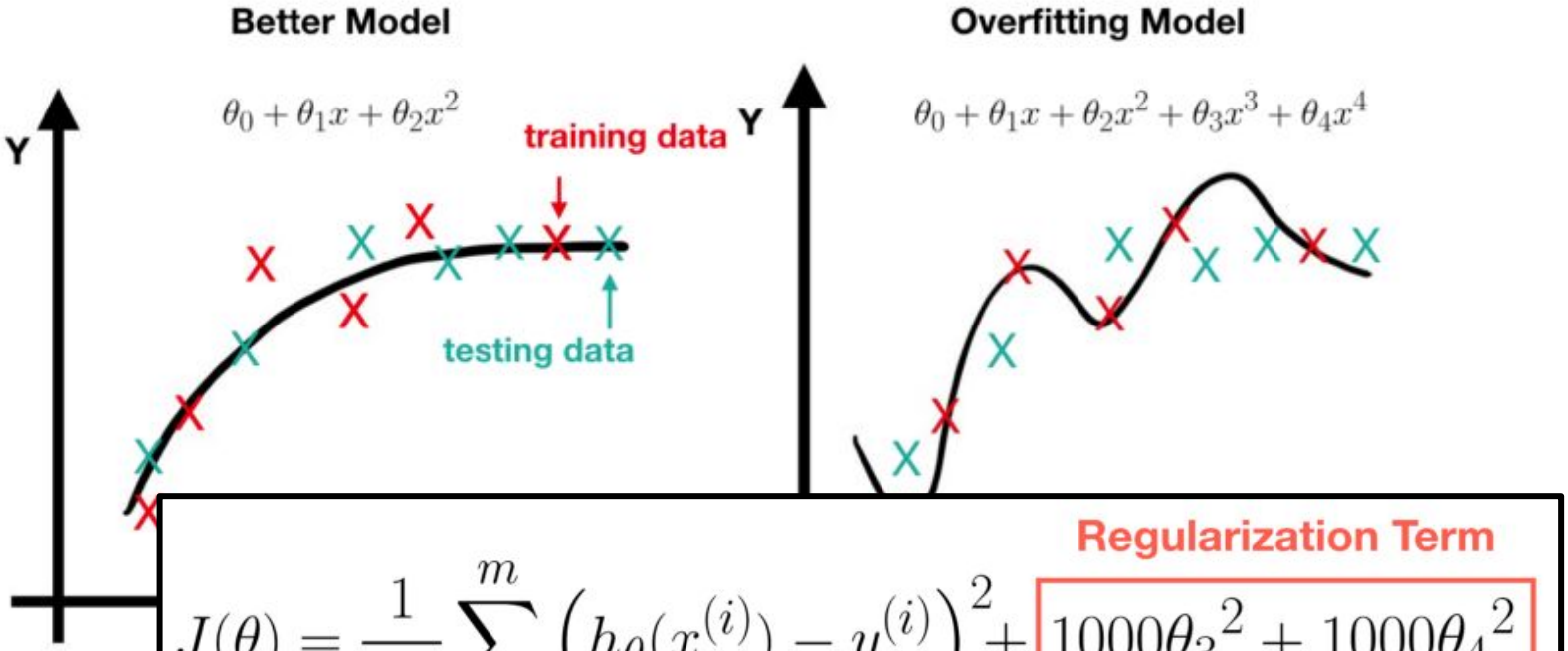
Overfitting





Overfitting: regularization





Regularization Term

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 + \boxed{1000\theta_3{}^2 + 1000\theta_4{}^2}$$

$$\min_{\theta} J(\theta), \text{ getting } \theta_3 \approx 0, \theta_4 \approx 0$$

Sklearn: basic usage

Coding time

Assignment: read this page

Classification

Supervised Learning: Classification

Binary:

- Emails: Spam / Not Spam?
- Cancer: Positive/ Negative?
- Tumor: Malignant / Benign?

$$y \in \{0, 1\}$$

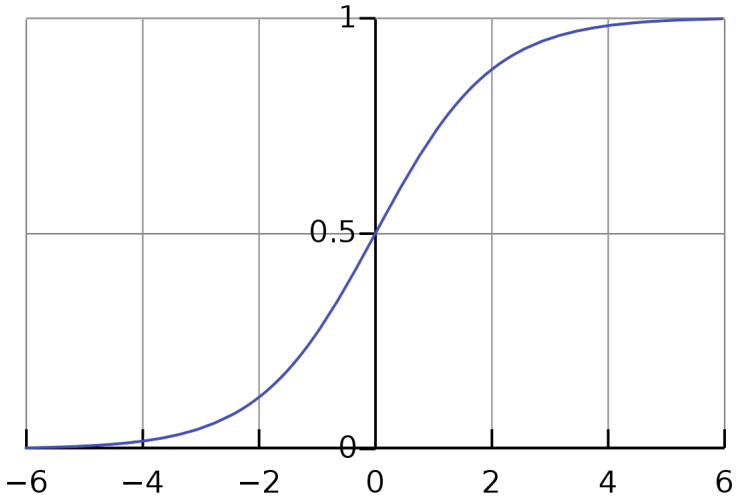
Multiple:

- Classify an image into the correct category
- Classify the news into its category (business, tech, …)

$$y \in \{0, 1, 2, 3..\}$$

From Regression to Classification

Logistic/Sigmoid Function



Maps any R -> (0,1)

(0,1) ? Probabilities!

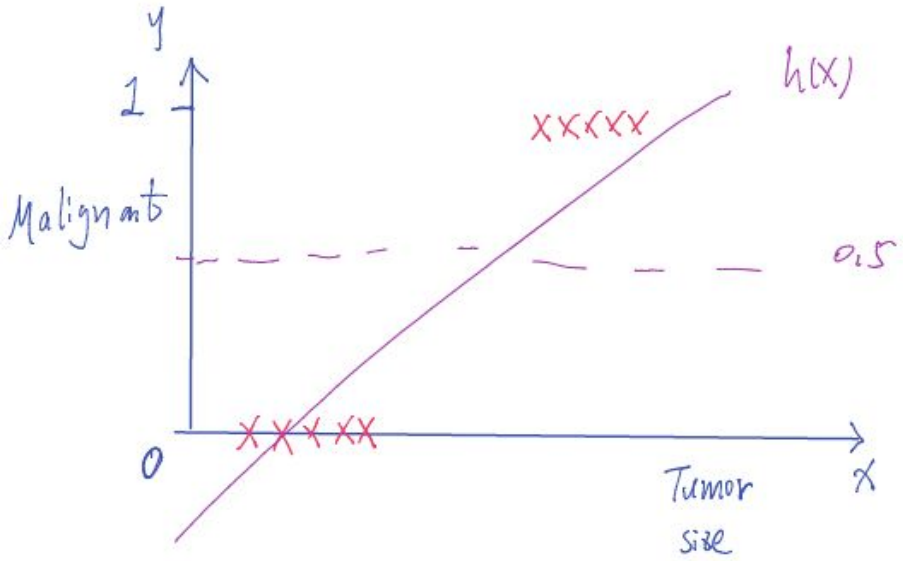
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

```
def logistic(z):
    """Logistic function."""
    return 1 / (1 + np.exp(-z))
```

Supervised Learning: Classification

Whiteboard Time!

The key thing to note is the cost function penalizes confident and wrong predictions more than it rewards confident and right predictions!



$h(x) = \theta^T x$ Linear regression.

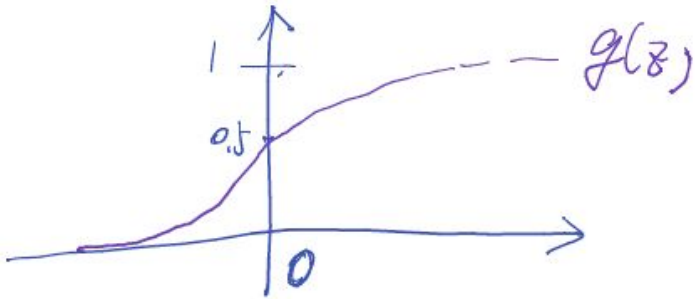
if $h(x) \geq 0.5$ $y = 1$

if $h(x) < 0.5$ $y = 0$

Logistic regression

$$g(z) = \frac{1}{1+e^{-z}}$$

"Sigmoid / Logistic Function"



$h_\theta(x) = 0.7$

$h_\theta(x) = P(y=1 \mid x; \theta)$ "prob. $y=1$, given x, parameterized by θ."

Training Set $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots (x^{(m)}, y^{(m)})\}$

m examples $x \in \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}$ $y \in \{0, 1\}$

θ?

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

$$\text{Cost}\left(h_\theta(x^{(i)}), y \right)$$

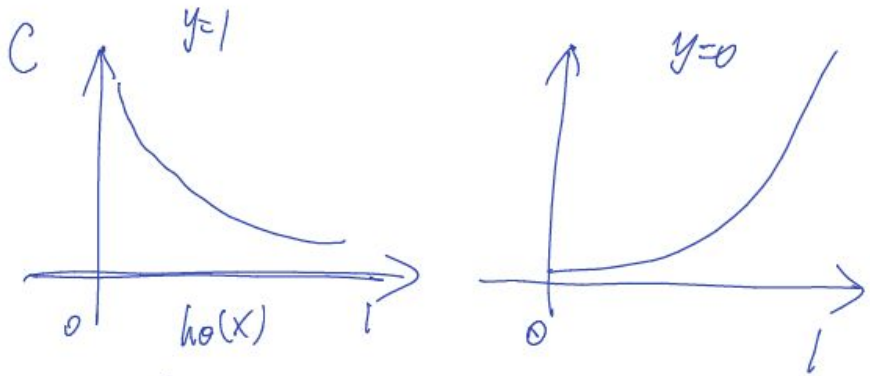
Loss function: Cross-entropy

Cross-entropy

"ground truth"

$$\text{cost}(h_\theta(x), y) = -\log(h_\theta(x)) \quad y=1$$

$$= -\log(1-h_\theta(x)) \quad y=0$$



$$J(\theta) = -\frac{1}{m}\sum_{t}^{m}\left[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right]$$

Metrics - Classification

Accuracy

Precision & Recall (generally binary classification)

$$P = \frac{TP}{TP + FP}$$

TP FP

$$R = \frac{TP}{TP + FN}$$

FN TN

F1 score

P & R are high, F1 is high.

$$F_1 = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

“Confusion Matrix”

True Positive (TP): • Reality: Malignant • ML model predicted: Malignant • **Number of TP results: 1**	**False Positive (FP):** • Reality: Benign • ML model predicted: Malignant • **Number of FP results: 1**
False Negative (FN): • Reality: Malignant • ML model predicted: Benign • **Number of FN results: 8**	**True Negative (TN):** • Reality: Benign • ML model predicted: Benign • **Number of TN results: 90**

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{1 + 90}{1 + 90 + 1 + 8} = 0.91$$

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

Precision and Recall

True Positive (TP):

- Reality: Malignant
- ML model predicted: Malignant
- **Number of TP results: 1**

False Positive (FP):

- Reality: Benign
- ML model predicted: Malignant
- **Number of FP results: 1**

False Negative (FN):

- Reality: Malignant
- ML model predicted: Benign
- **Number of FN results: 8**

True Negative (TN):

- Reality: Benign
- ML model predicted: Benign
- **Number of TN results: 90**

$$P = \frac{TP}{TP + FP}$$

$$R = \frac{TP}{TP + FN}$$

Precision = 1/(1+1) = 0.5 Recall = 1/(1+8) = 0.11

Range [0,1]
Upper bound 1

F1 Score

$$F_1 = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

F1 Score considers both Precision and Recall.

If P and R are hight, then F1 is high.

In papers, for classification, we always report Accuracy, F1, sometimes with Precision and Recall.

There is also micro-F1 and macro-F1.

Micro-F1 and Macro-F1

Useful for multi-classification (not binary)

micro-averaging (biased by class frequency) or
macro-averaging (taking all classes as equally important)

(Optional) reading link

A toy example

$$P = \frac{TP}{TP + FP}$$

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

$$R = \frac{TP}{TP + FN}$$

		True/Actual		
		Cat (🐱)	Fish (🐟)	Hen (🐔)
Predicted	Cat (🐱)	4	6	3
	Fish (🐟)	1	2	0
	Hen (🐔)	1	2	6

F1 for each class..

F1-score(Cat) = 2 × (30.8% × 66.7%) / (30.8% + 66.7%) = 42.1%

Class	Precision	Recall	F1-score
Cat	30.8%	66.7%	42.1%
Fish	66.7%	20.0%	30.8%
Hen	66.7%	66.7%	66.7%

Macro-F1 = (42.1% + 30.8% + 66.7%) / 3 = 46.5%

Micro-F1 = (6 × 42.1% + 10 × 30.8% + 9 × 66.7%) / 25 = 46.4%

In Sklearn

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html#sklearn.metrics.accuracy_score

Examples

```
>>> from sklearn.metrics import accuracy_score
>>> y_pred = [0, 2, 1, 3]
>>> y_true = [0, 1, 2, 3]
>>> accuracy_score(y_true, y_pred)
0.5
```

Use Logistic Regression to classify MNIST data

Code

Assignment (Optional):

try other classification methods and compare the accuracy:

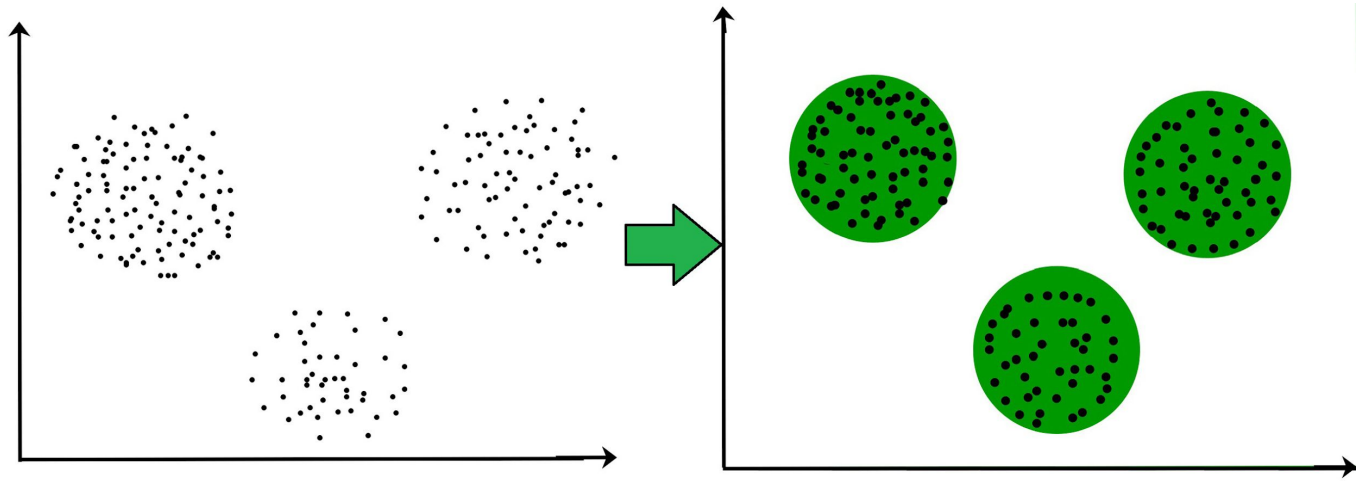
SVM(reading link), **Random Forest** (link) and Naive Bayes, etc

Clustering

K-Means

Unsupervised Learning

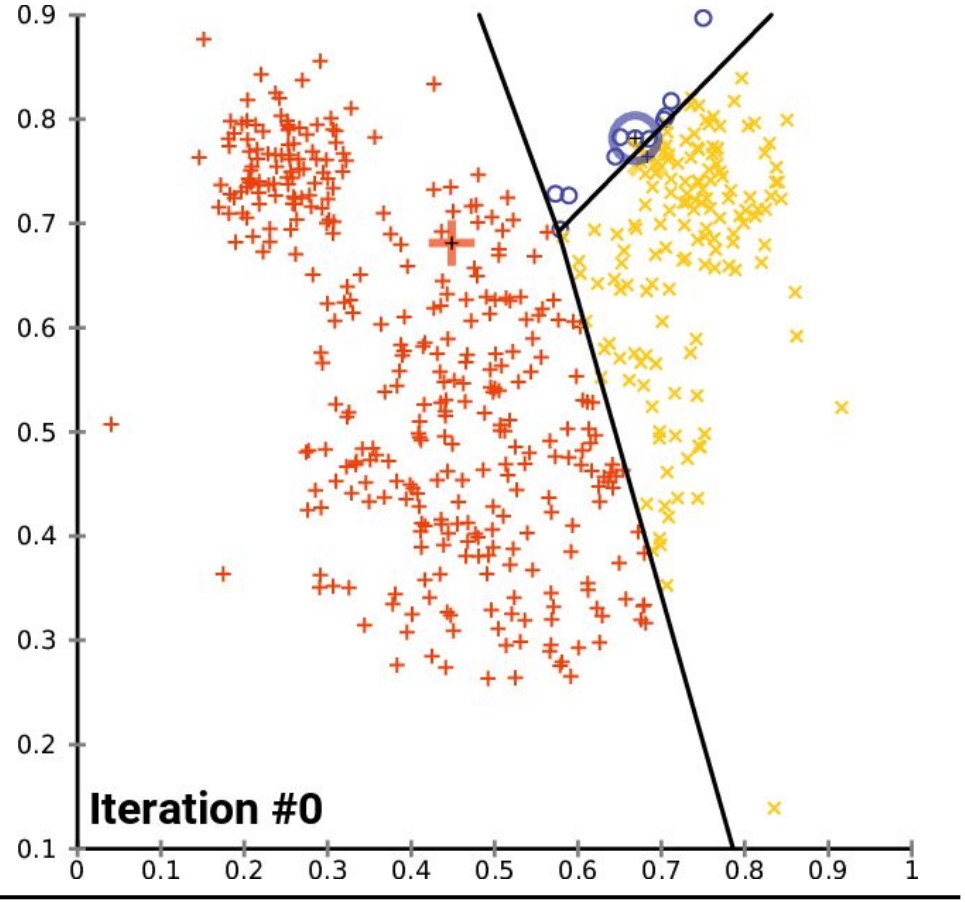
Clustering



Unsupervised Learning

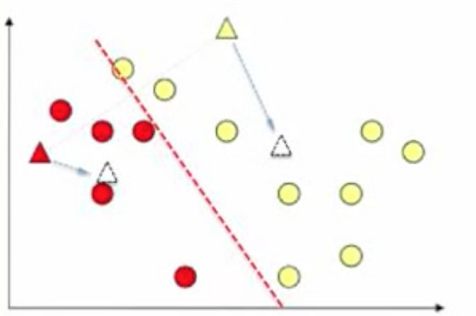
Clustering: k-means

2D

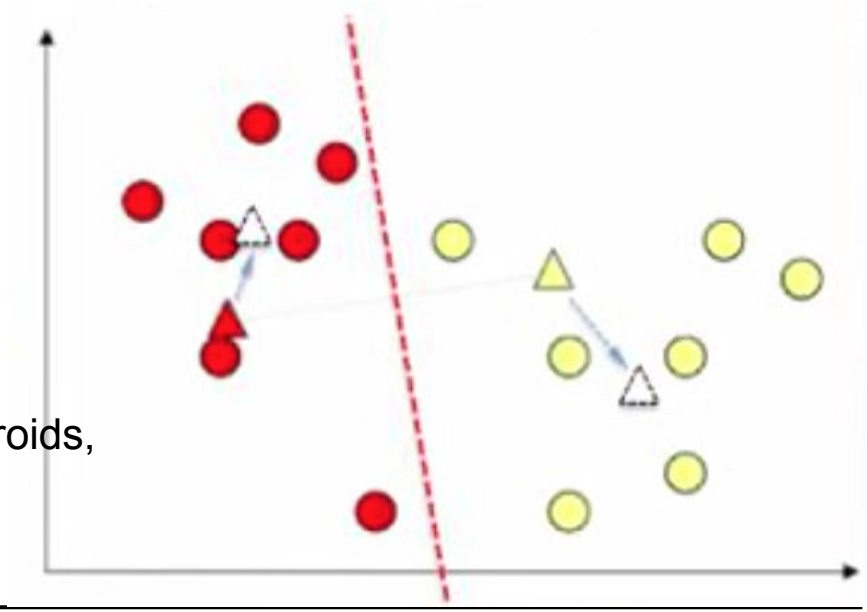


K-means: a demo

Two clusters, squares are the **centroids**.



Calculate new centroids,
finish one iteration.



Prototype-based Clustering: k-means

Dataset D, Clusters C

$$D=\{x_1,x_2,..,x_m\}$$
$$C=\{C_1,C_2,..,C_k\}$$

Error (each one in each cluster) whiteboard

$$E=\sum_{i=1}^{k}\sum_{x\in C_i}||x-\mu_i||^2$$

$$\mu_i=\frac{1}{|C_i|}\sum_{x\in C_i}x$$

K-means: algorithm

O(#iter * #clusters * #instances * #dimensions)

(Input: whole dataset points x1,...xn)

Initialization: Randomly place centroids c1..ck

Repeat until convergence (stop when no points changing):

- for each data point xi:

  argmin D (xi,cj) for all cj

  find nearest **centroid**, set the point to the centroid cluster

- for each cluster:

  calculate new centroid (means of all new points)

$$\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$$

K-means

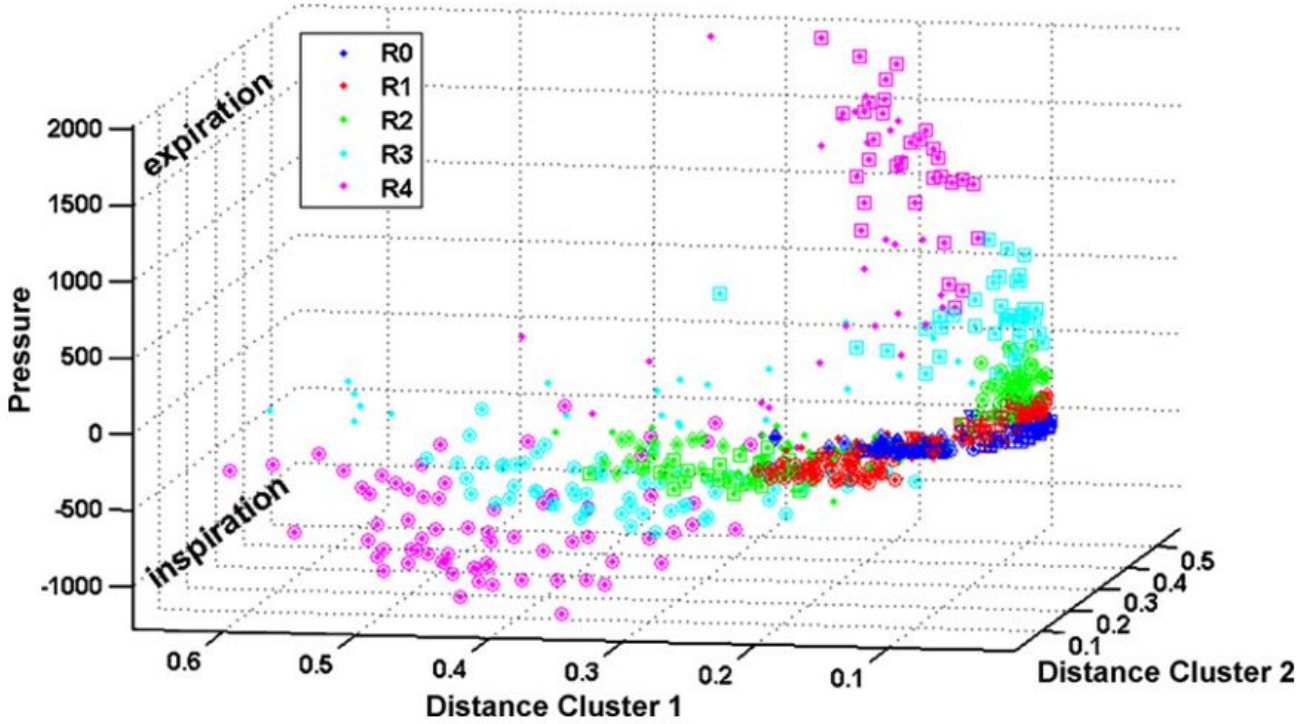
- Strength
  - *Relatively efficient*: *O*(*tkn*), where *n*, k, and t are the number of objects, number of clusters, and number iterations respectively. Normally, *k*, *t* << *n*.
  - Often terminates at a *local optimum*
  - The *global optimum* may be found using techniques such as: *deterministic annealing* and *genetic algorithms*
- Weakness
  - Applicable only when *mean* is defined, then what about categorical data?
  - Need to specify *k*, the *number* of clusters, in advance
  - Unable to handle noisy data and *outliers*

High-dim...

Clustering

In

High-dimensional



Other Concepts

Cross validation (1)

A standard way:

Split your dataset into:

Use training set to train, dev to adjust parameters, and test set to test and report the scores (accuracy, etc)...

Some times, we can only have **Training** and **Testing (no dev)**

75%/25% or 80%/20%

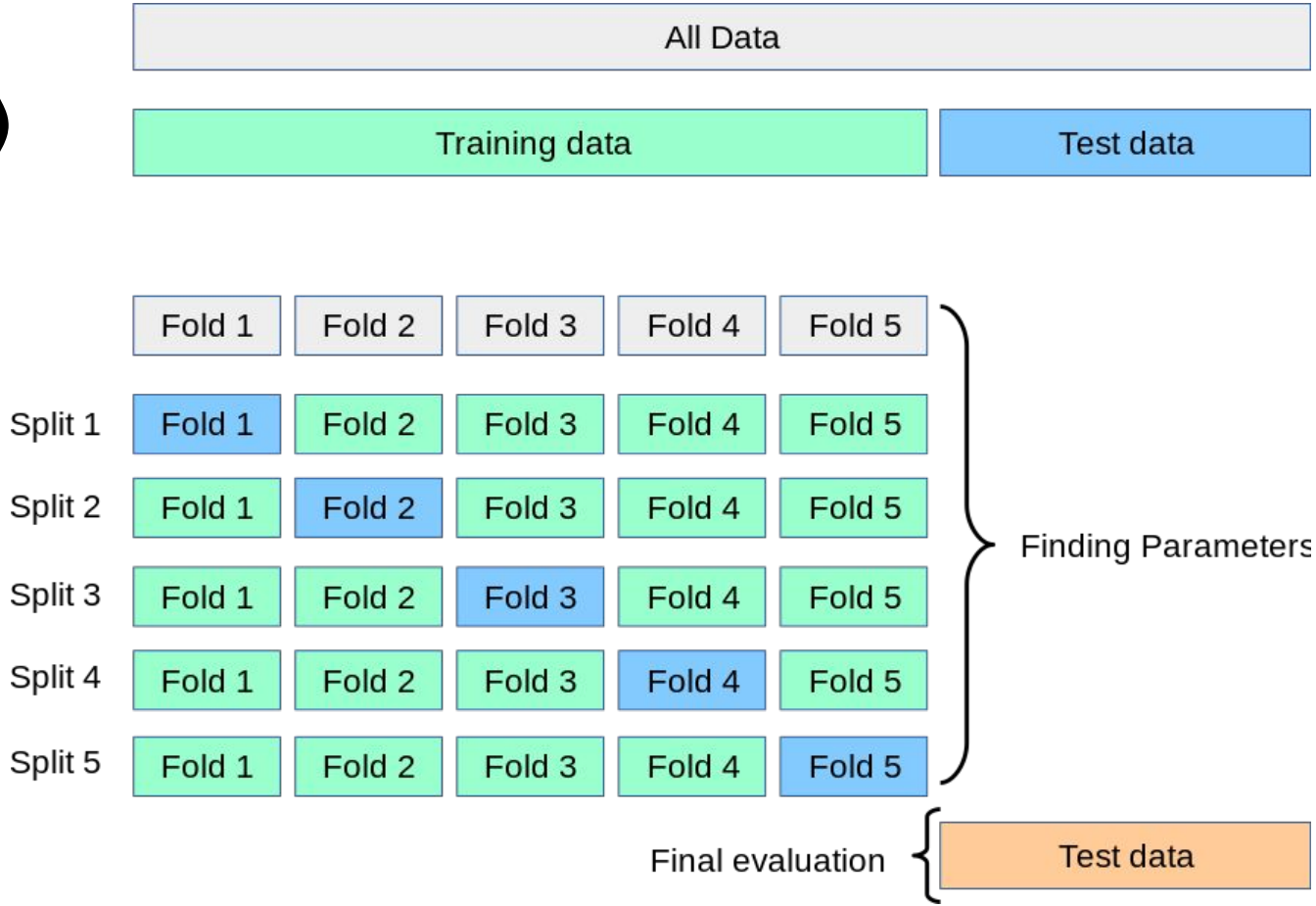
Cross validation (2)

5-fold Cross validation

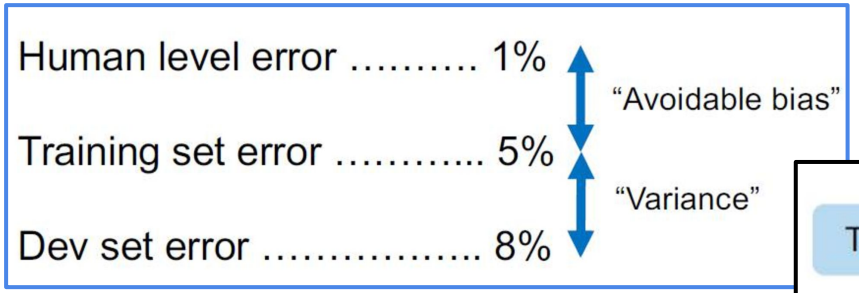
In a fair way to find parameters

K-fold

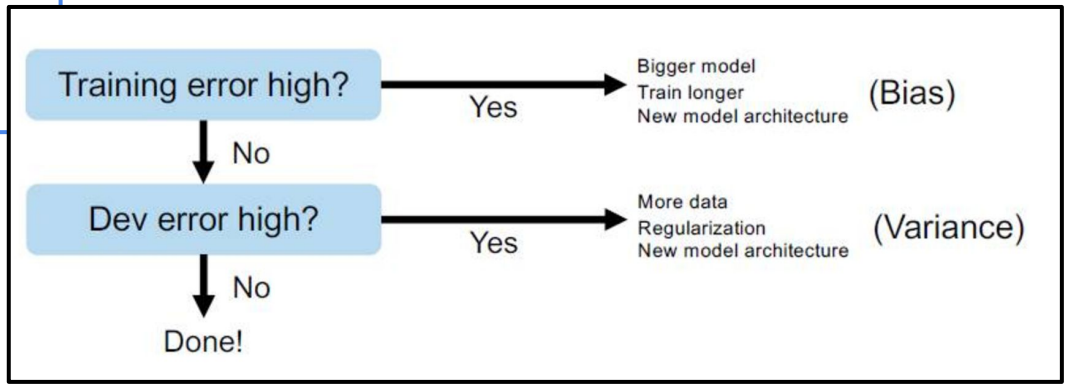
[Sklearn code]



Dataset ML Strategy



Bias-variance Trade-off =
Avoidable Bias + Variance



More Data; Regularization; New Model

More Algorithms...

Supervised Learning:
Linear regression,Logistic regression
Linear Discriminant Analysis
Principal Component Analysis
Neural network
Support vector machines
K-nearest neighbor
Gradient Boosting Decision Tree
Decision trees(C4.5,ID3,CART), Random Forests (read)
Kernels, Kernel-based PCA

Optimization Methods

Natural Language Processing

Overview: NLP

Basics: TF-IDF, Language Models, Activation Functions

Models: RNNs, LSTMs, Seq2seq, Attention, Transformers, BERT

Machine Translation: BLUE, Meteor, ROUGE, Perplexity

BERT for classification, machine translation

Text Summarization: Textrank, evaluation

Advanced models: Graph Convolutional Networks (GCN), GAT

TF-IDF

Term Frequency-Inverse Document Frequency

Doc 1 It is going to rain today.

Doc 2 Today I am not going outside.

Doc 3 I am going to watch the season premiere.

Word	Count
going	3
to	2
today	2
i	2
am	2
it	1
is	1
rain	1

Missing References

IDF: Inverse Document Frequency

IDF =Log[(Number of documents) / (Number of documents containing the word)]

Word	Count
going	3
to	2
today	2
i	2
am	2
it	1
is	1
rain	1

Words	IDF Value
going	log(3/3)
to	log(3/2)
today	log(3/2)
i	log(3/2)
am	log(3/2)
It	log(3/1)
is	log(3/1)
rain	log(3/1)

Term Frequency-Inverse Document Frequency

TF = (Number of repetitions of word in a document) / (# of words in a document)

Doc 1 It is going to rain today.

“Going” = ⅙ = 0.16
“Am” = 0/6 = 0

Find TF

(Term Frequency)

Words/ Documents	Document 1
going	0.16
to	0.16
today	0.16
i	0
am	0
it	0.16
is	0.16
rain	0.16

TF for all documents

Words/ Documents	Document 1	Document 2	Document 3
going	0.16	0.16	0.12
to	0.16	0	0.12
today	0.16	0.16	0
i	0	0.16	0.12
am	0	0.16	0.12
it	0.16	0	0
is	0.16	0	0
rain	0.16	0	0

TF-IDF = TF * IDF

Words	IDF Value
going	0
to	0.41
today	0.41
i	0.41
am	0.41
It	1.09
is	1.09
rain	1.09

Words/ Documents	Document 1	Document 2	Document 3
going	0.16	0.16	0.12
to	0.16	0	0.12
today	0.16	0.16	0
i	0	0.16	0.12
am	0	0.16	0.12
it	0.16	0	0
is	0.16	0	0
rain	0.16	0	0

TF-IDF

Each row represents a document, in a vector!

Doc1: [0,0.07,0.07,0,0,0.17,0.17,0.17]

Doc2:[0,0,0.07,0.07,0.07,0,0,0]

Doc3:[0,0.05,0,0.05,0.05,0,0,0]

Words/ Documents	going	to	today	i	am	it	is	rain
Document 1	0	0.07	0.07	0	0	0.17	0.17	0.17
Document 2	0	0	0.07	0.07	0.07	0	0	0
Document 3	0	0.05	0	0.05	0.05	0	0	0

Similarity of documents

Doc1: [0,0.07,0.07,0,0,0.17,0.17,0.17]

Doc2:[0,0,0.07,0.07,0.07,0,0,0]

Doc3:[0,0.05,0,0.05,0.05,0,0,0]

cos(doc1,doc2)= 0.130

cos(doc3,doc2)= 0.667

Doc 1 It is going to rain today.

Doc 2 **Today I am not going outside.**

Doc 3 **I am going to watch the season premiere.**

Practice (1/3)

Calculate the TF*IDF for the terms listed below for documents 1 to 4. There are 10,000 documents in a collection. The number of times each of these terms occur in documents 1 to 4 as well as the number of documents in the collections are listed below. Use this information to fill in the TF*IDF scores in the table below.

Number of Documents Containing Terms:

- *reverse cascade*: 3
- *full shower*: 50
- *half bath*: 10
- *multiplex*: 3

Practice (2/3)

Term Frequencies				
	Documents			
	Doc 1	Doc 2	Doc 3	Doc 4
reverse cascade	8	10	0	0
full shower	3	1	2	2
half bath	0	0	8	7
multiplex	2	2	2	9

First calculate the IDF of the mentioned terms; then fill the following table:

Practice (3/3)

TFIDF for terms in documents				
	Documents			
	Doc 1	Doc 2	Doc 3	Doc 4
reverse cascade				
full shower				
half bath				
multiplex				

Solution.. TF-IDF

- **reverse cascade**: 3 IDF = log(10000/3) ≈ 8.11
- **full shower**: 50 IDF = log(10000/50) ≈ 5.30
- **half bath**: 10 IDF = log(10000/10) ≈ 6.91
- **multiplex**: 3 IDF = log(10000/3) ≈ 8.11

Term Frequencies				
	Documents			
	Doc 1	Doc 2	Doc 3	Doc 4
reverse cascade	8	10	0	0
full shower	3	1	2	2
half bath	0	0	8	7
multiplex	2	2	2	9

TFIDF for terms in documents				
	Documents			
	Doc 1	Doc 2	Doc 3	Doc 4
reverse cascade	8.11 * 8 = 64.88	8.11 * 10 = 81.10	0	0
full shower	5.30 * 3 + 15.90	5.30 * 1 = 5.30	5.30 * 2 = 10.60	5.30 * 2 = 10.60
half bath	0	0	6.91 * 8 = 55.28	6.91 * 7 = 48.37
multiplex	8.11 * 2 = 16.22	8.11 * 2 = 16.22	8.11 * 2 = 16.22	8.11 * 9 = 72.99

Language Modeling

N-gram: n tokens tuple

OK (is a uni-gram)

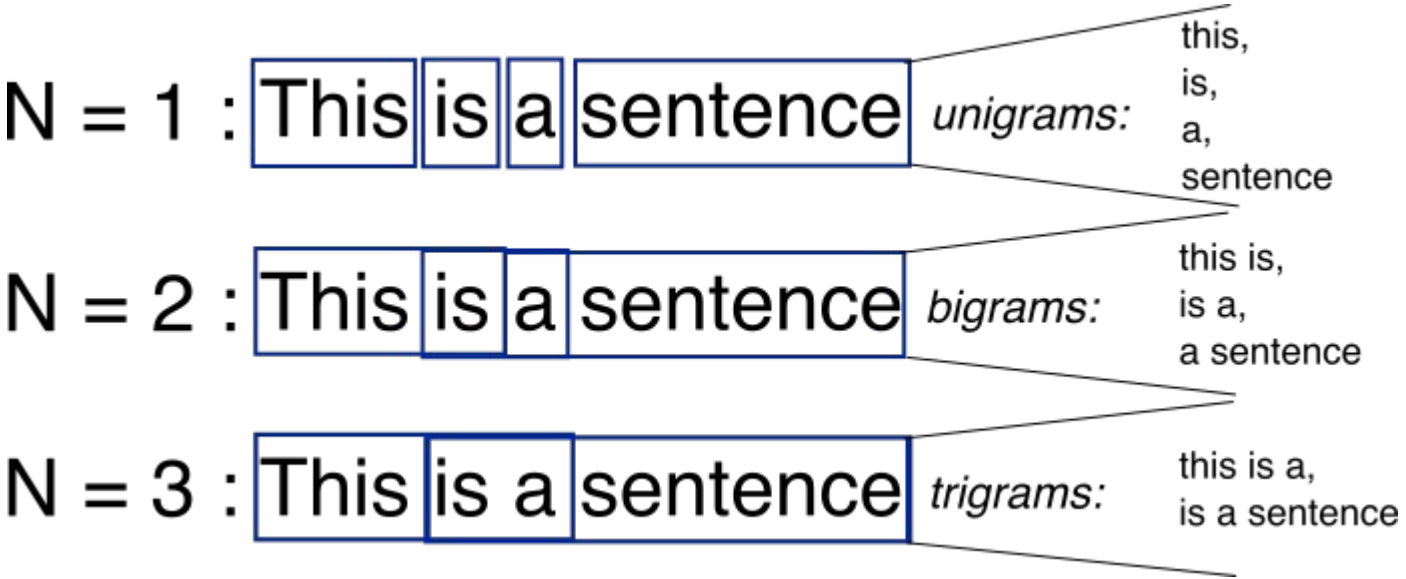
San Francisco (is a bi-gram)

The Three Musketeers (is a tri-gram)

She stood up slowly (is a 4-gram)

...

In a sentence...



Preliminary

Conditional Probability

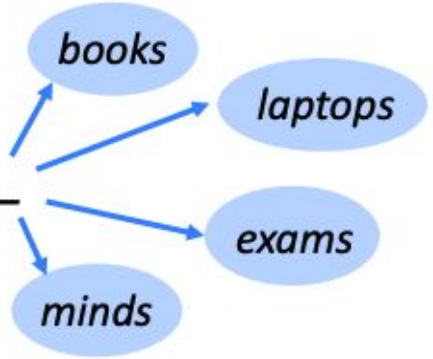
$$P(A|B) = \frac{P(A,B)}{P(B)}$$

Chain Rule

$$P(x_1, x_2, \ldots, x_n) = P(x_1)P(x_2|x_1)\ldots P(x_n|x_1, \ldots, x_{n-1})$$

- **Language Modeling** is the task of predicting what word comes next.

the students opened their _____



- More formally: given a sequence of words $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$, compute the probability distribution of the next word $x^{(t+1)}$:

$$P(\boldsymbol{x}^{(t+1)} | \ \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$$

where $x^{(t+1)}$ can be any word in the vocabulary $V = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{|V|}\}$

- A system that does this is called a **Language Model**.

Source from Stanford University

You can also think of a Language Model as a system that assigns probability to a piece of text.

For example, if we have some text $x^{(1)}, \ldots, x^{(T)}$, then the probability of this text (according to the Language Model) is:

$$P(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}) = P(\boldsymbol{x}^{(1)}) \times P(\boldsymbol{x}^{(2)} | \, \boldsymbol{x}^{(1)}) \times \cdots \times P(\boldsymbol{x}^{(T)} | \, \boldsymbol{x}^{(T-1)}, \ldots, \boldsymbol{x}^{(1)})$$

$$= \prod_{t=1}^{T} P(\boldsymbol{x}^{(t)} | \, \boldsymbol{x}^{(t-1)}, \ldots, \boldsymbol{x}^{(1)})$$

This is what our LM provides

Preliminary

Chain Rule on a sequence

$$
\begin{aligned}
P(\text{" its water is so transparent "}) = \\
P(its) * \\
P(water|its) * \\
P(is|its\ water) * \\
P(so|its\ water\ is) * \\
P(transparent|its\ water\ is\ so)
\end{aligned}
$$

We can, but it is crazy!

$$
P(transparent|its\ water\ is\ so) = \frac{count(its\ water\ is\ so\ transparent)}{count(its\ water\ is\ so)}
$$

Markov Assumption

"The current state is only dependent on the previous state".

$$P(transparent|its\ water\ is\ so) \approx P(transparent|so)$$

$$P(transparent|its\ water\ is\ so) \approx P(transparent|is\ so)$$

Look at k-gram:

$$P(w_i|w_1 w_2 \ldots w_{i-1}) \approx P(w_i|w_{i-k} \ldots w_{i-1})$$

Markov Assumption with k-gram

K = 1

Unigram

$$P(w_1 w_2 \dots w_n) \approx \prod_i P(w_1)$$

K = 2

Bigram

$$P(w_i | w_1 w_2 \dots w_i - 1) \approx P(w_i | w_{i-1})$$

Markov Assumption with k-gram

Used to be:

$$P(x_1, x_2, \ldots, x_n) = P(x_1)P(x_2|x_1)\ldots P(x_n|x_1, \ldots, x_{n-1})$$

In bi-gram:

$$\boxed{P(x_1, x_2, \ldots, x_n) = P(x_1)P(x_2|x_1)P(x_3|x_2)\ldots P(x_n|x_{n-1})}$$

Tri-gram:

$$\boxed{P(x_1, x_2, \ldots, x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2)\ldots P(x_n|x_{n-2}, x_{n-1})}$$

Maximum Likelihood

Estimate probabilities in bigram case:

$$P(w_i|w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

An example in bigram case...

<s I am Sam /s>

<s Sam I am /s>

<s I do not like green eggs and ham /s>

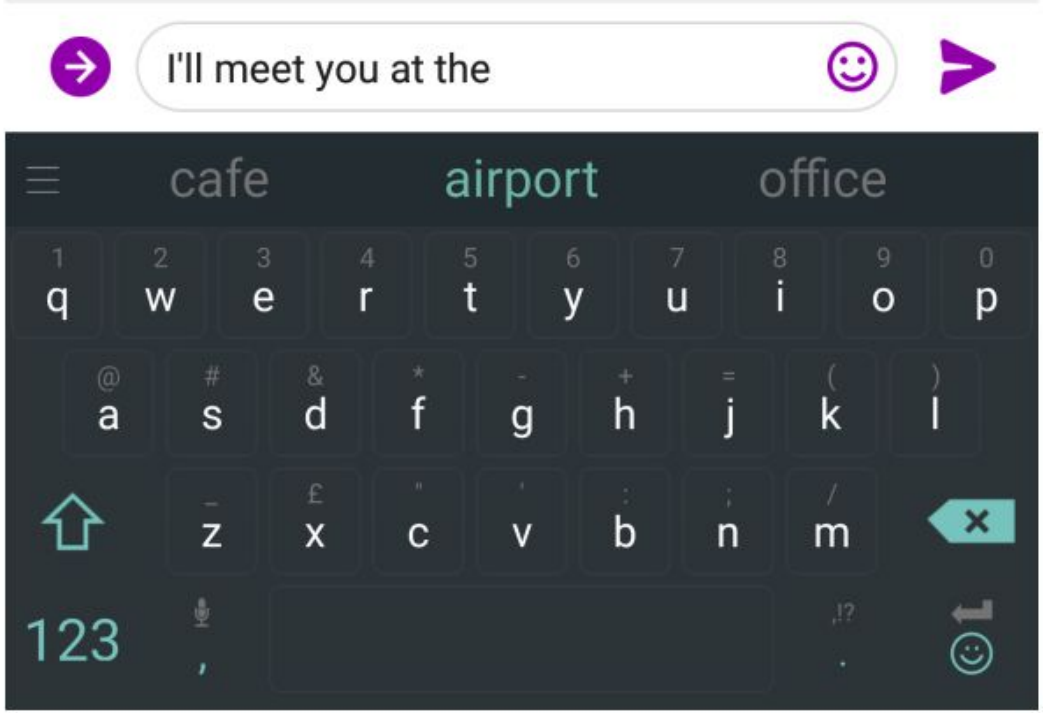
$$P(I|<s) = \frac{count(<s, I)}{count(<s)} = \frac{2}{3}$$

P(am| I) = ?

P(<s I am ham /s>) = ?

“<s”: start; “/s>”: end

Applications of LM



what is the

what is the **weather**
what is the **meaning of life**
what is the **dark web**
what is the **xfl**
what is the **doomsday clock**
what is the **weather today**
what is the **keto diet**
what is the **american dream**
what is the **speed of light**
what is the **bill of rights**

Google Search I'm Feeling Lucky

N-gram Language Models

the students opened their _____

Question: How to learn a Language Model?

Deep Learning model: RNNs/LSTMs

Neural LM: based on contents

output distribution

$$\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2) \in \mathbb{R}^{|V|}$$

hidden layer

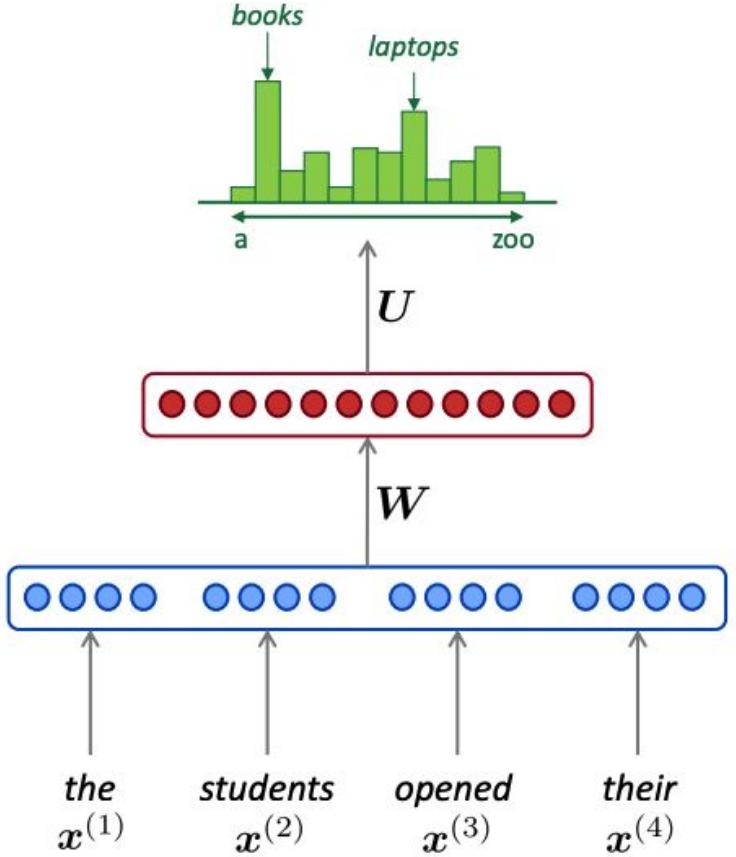
$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

concatenated word embeddings

$$\boldsymbol{e} = [\boldsymbol{e}^{(1)}; \boldsymbol{e}^{(2)}; \boldsymbol{e}^{(3)}; \boldsymbol{e}^{(4)}]$$

words / one-hot vectors

$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}$$



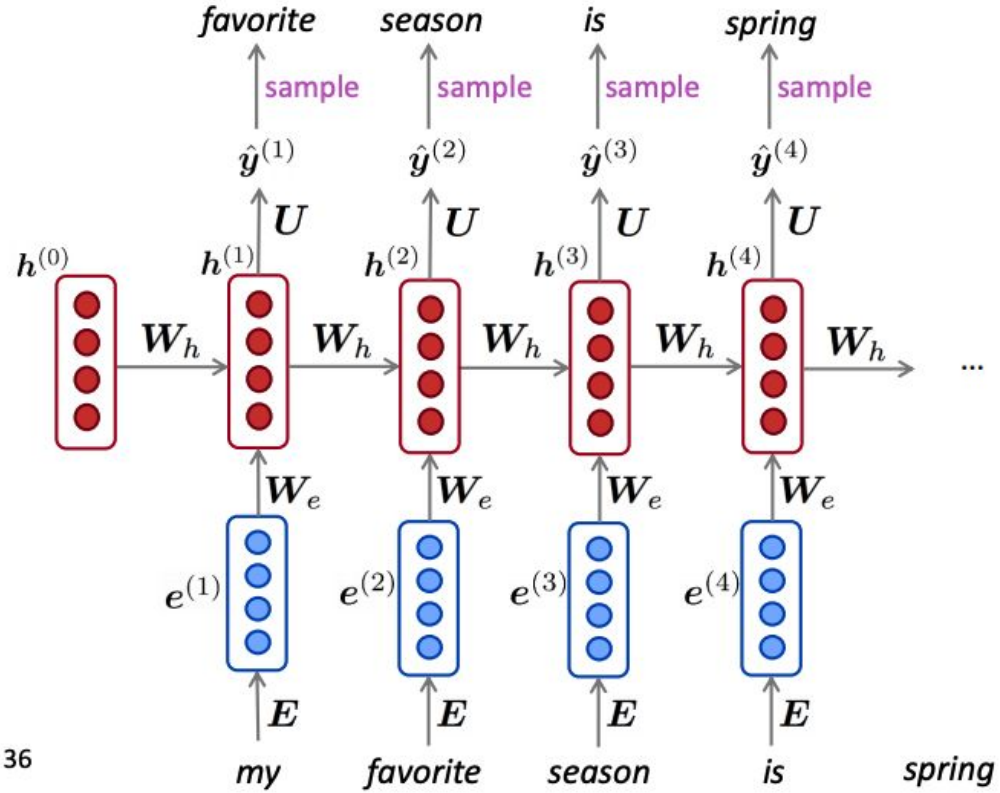
RNNs

"Unsupervised Training"

Inputs are sentences, outputs are themselves!

Generating text with a RNN Language Model

Just like a n-gram Language Model, you can use a RNN Language Model to generate text by repeated sampling. Sampled output is next step's input.



Apps?

Trained on book I-IV.
Let the RNN to write a new Harry Potter book!

[Link]

Generating text with a RNN Language Model

- Let's have some fun!
- You can train a RNN-LM on any kind of text, then generate text in that style.
- RNN-LM trained on *Harry Potter*:

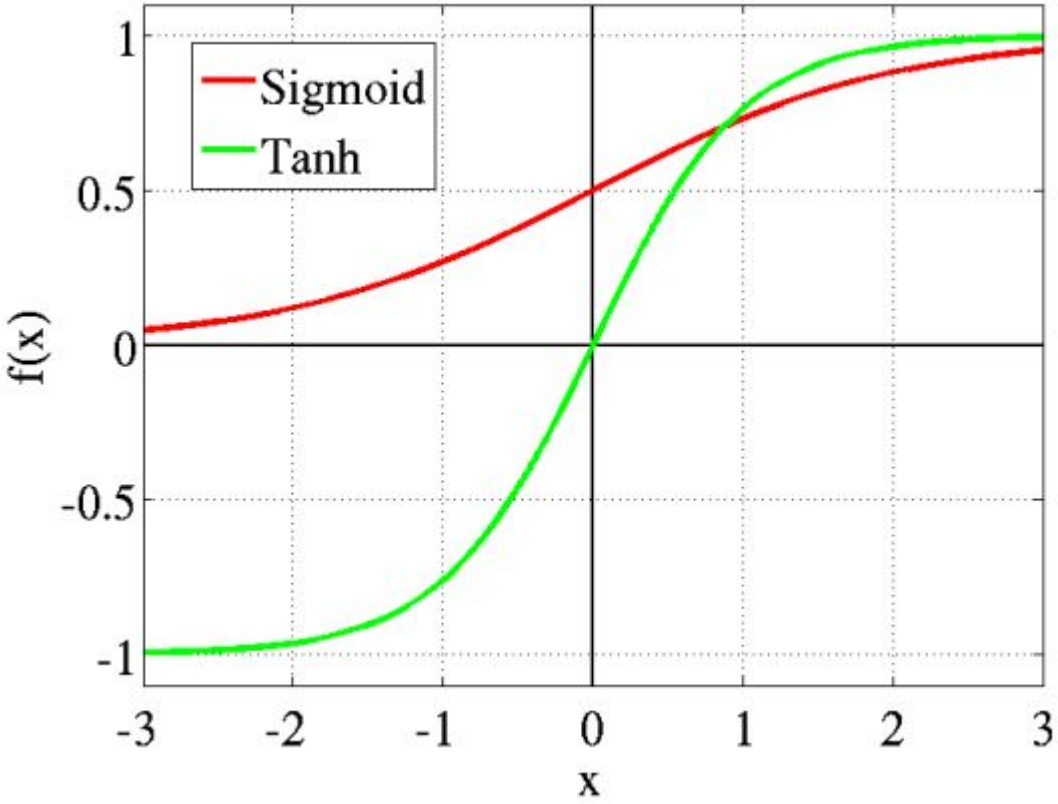
"Sorry," Harry shouted, panicking—"I'll leave those brooms in London, are they?"

"No idea," said Nearly Headless Nick, casting low close by Cedric, carrying the last bit of treacle Charms, from Harry's shoulder, and to answer him the common room perched upon it, four arms held a shining knob from when the spider hadn't felt it seemed. He reached the teams too.

Activation Functions

Tangent (Tanh)

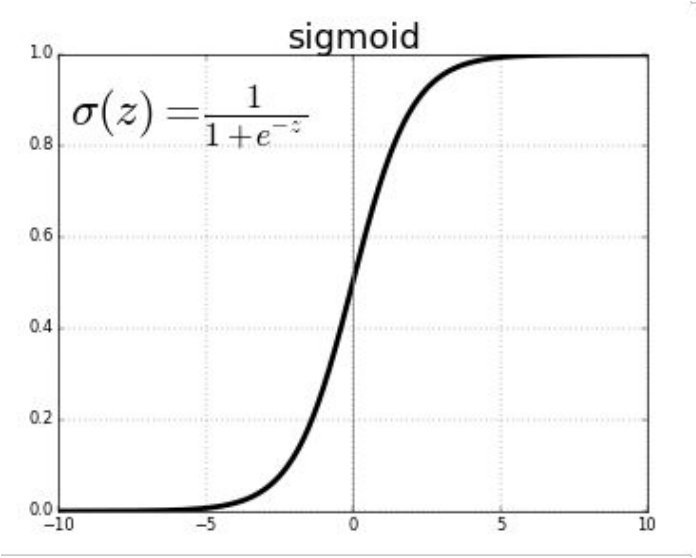
Tanh range(-1,1)

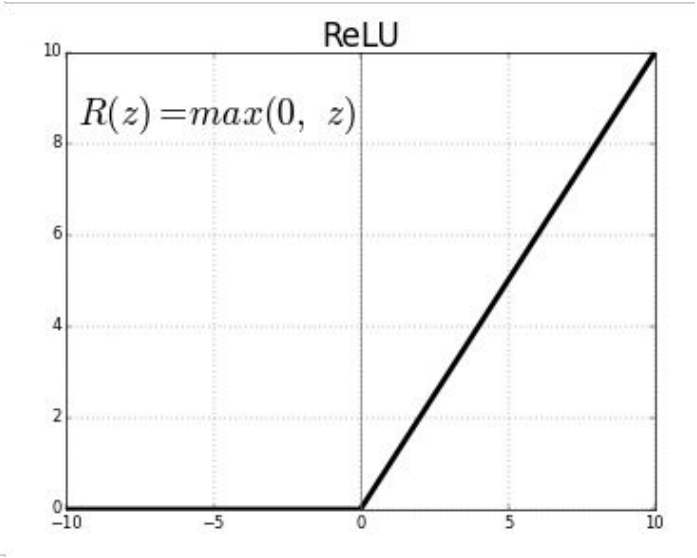


ReLU

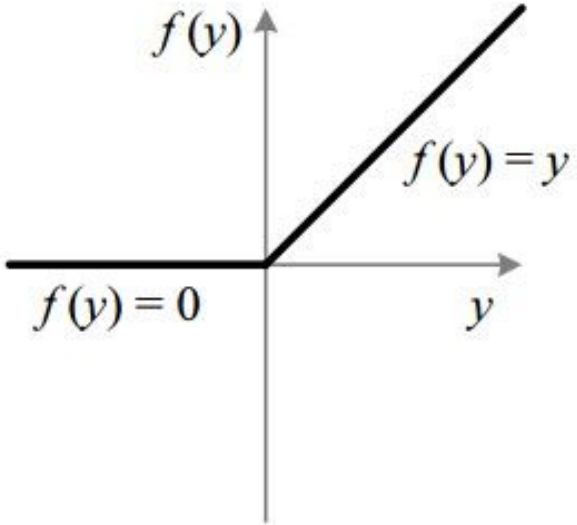
max(0,z)

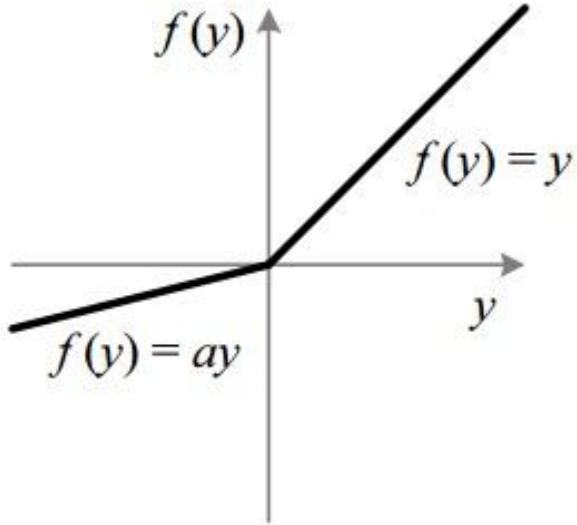
Range: [0 to infinity)





Leaky ReLU

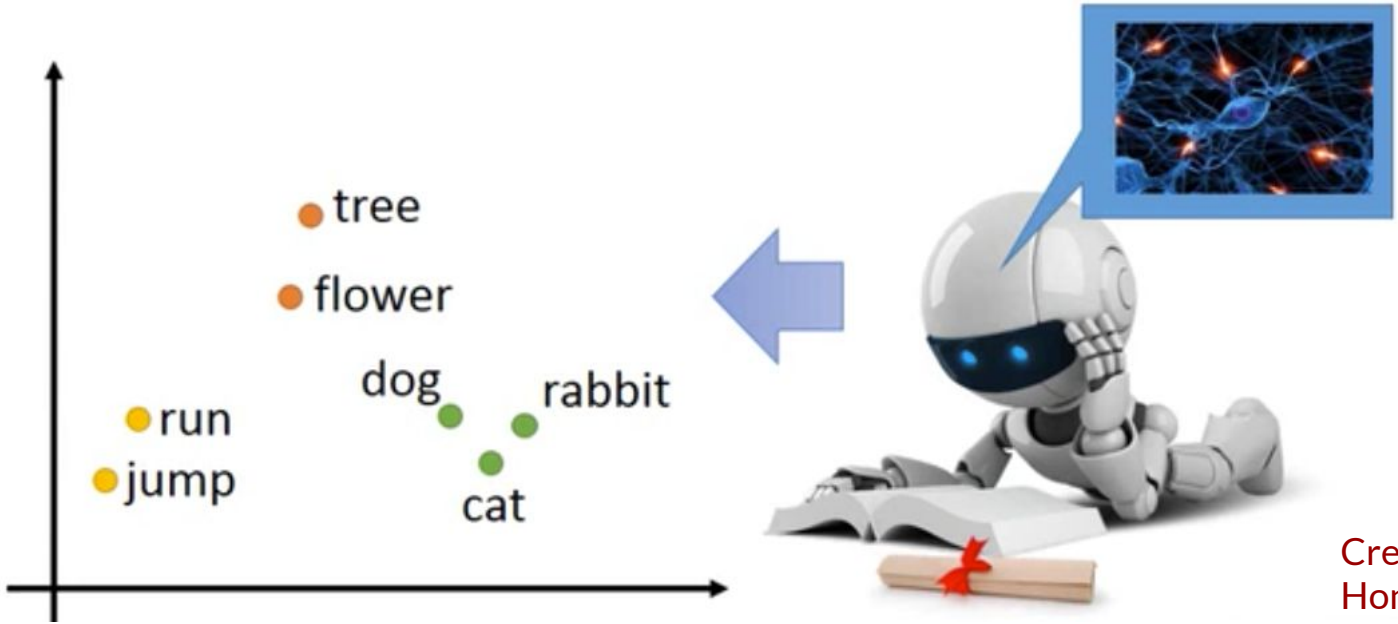




Word2vec

Word Embedding

- Machine learns the meaning of words from reading a lot of documents without supervision



Credit goes to Prof. Hong-Yee Lee

1-of-N Encoding

apple = [1 0 0 0 0]

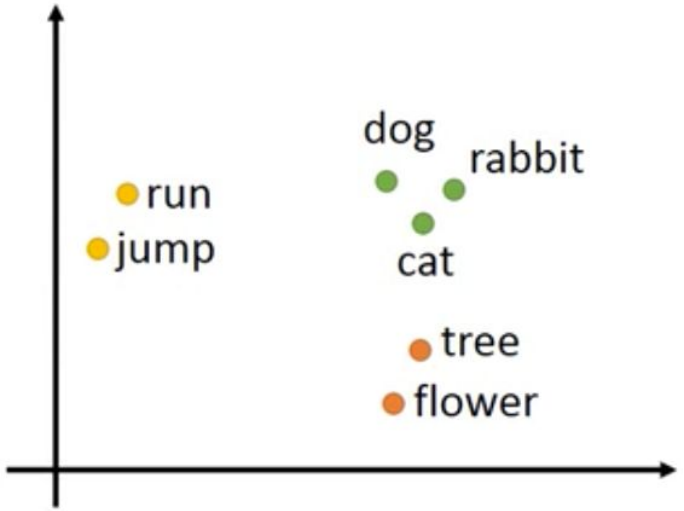
bag = [0 1 0 0 0]

cat = [0 0 1 0 0]

dog = [0 0 0 1 0]

elephant = [0 0 0 0 1]

Word Embedding



Word Class

Word meanings are defined by its company

- **Count based**
  - If two words w_i and w_j frequently co-occur, $V(w_i)$ and $V(w_j)$ would be close to each other

$$V(w_i) \cdot V(w_j)$$

Inner product

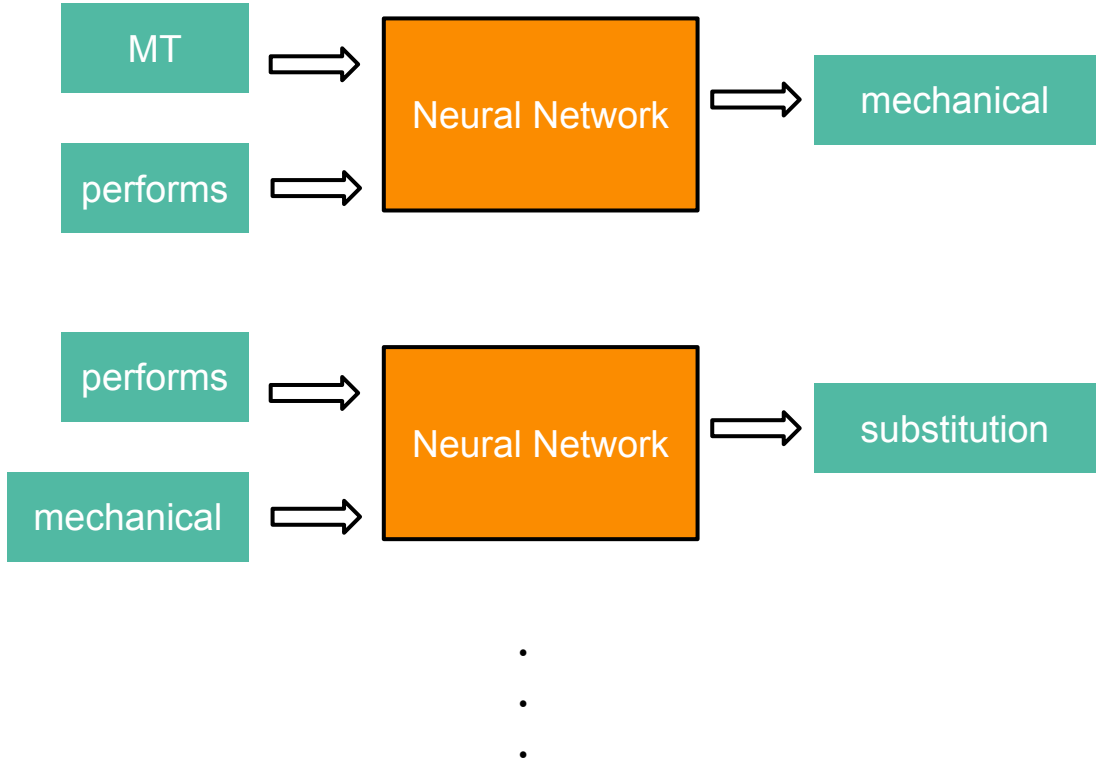
$$N_{i,j}$$

Number of times w_i and w_j in the same document

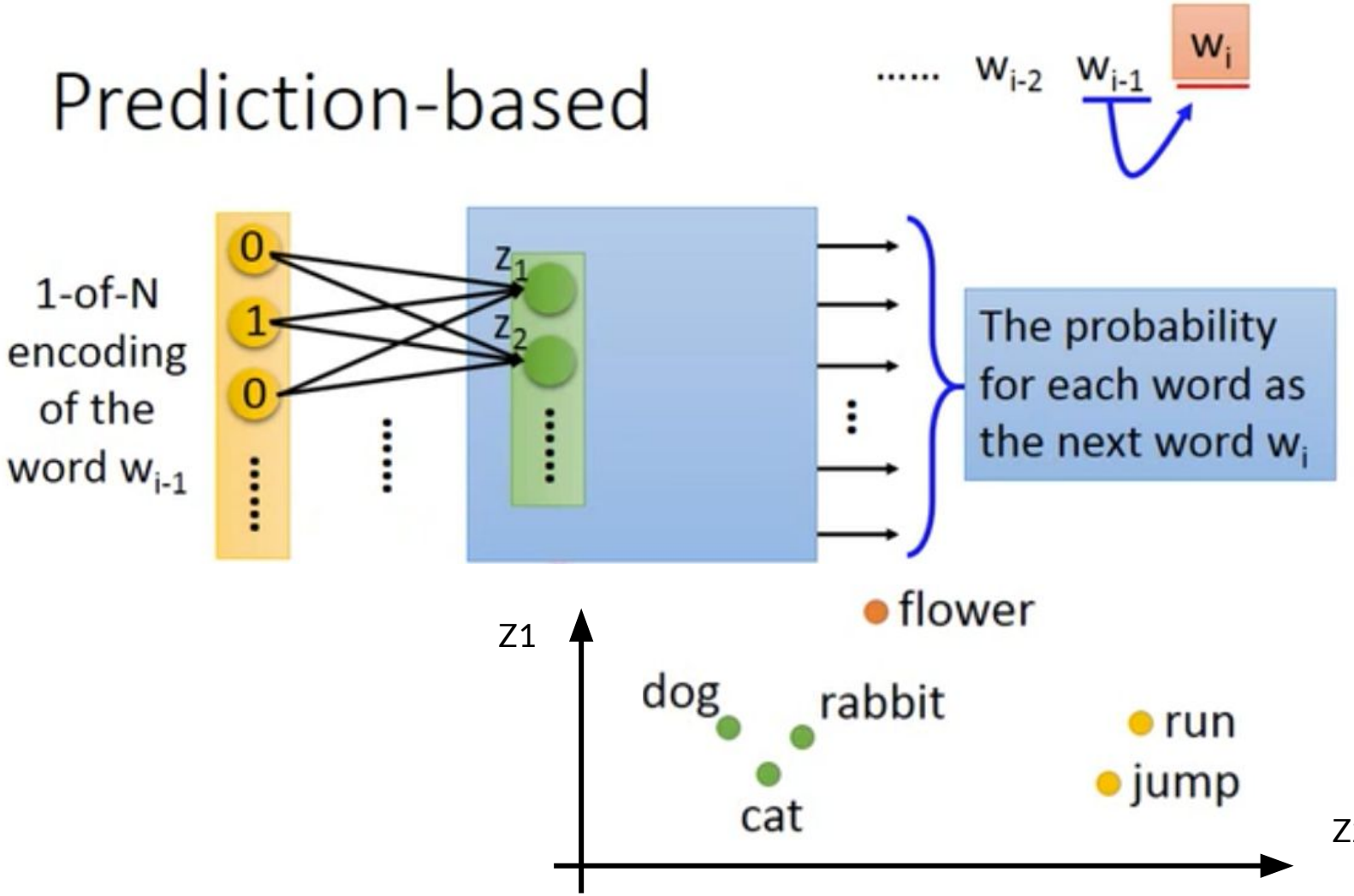
Prediction-based Training

Data:
"MT performs mechanical substitution of words in one language for words …"

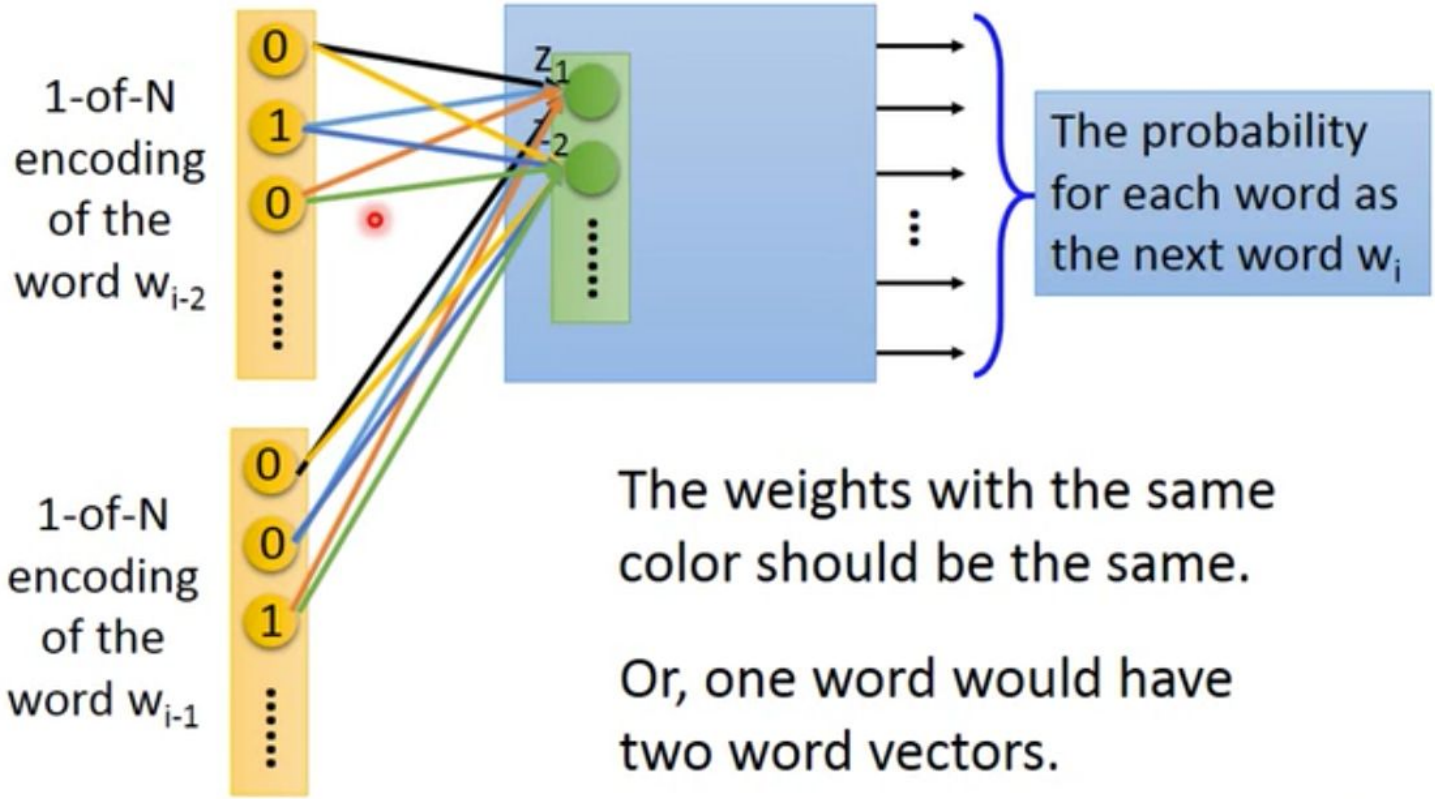
Minimizing cross entropy



Prediction-based



Prediction-based – Sharing Parameters



The weights with the same color should be the same.

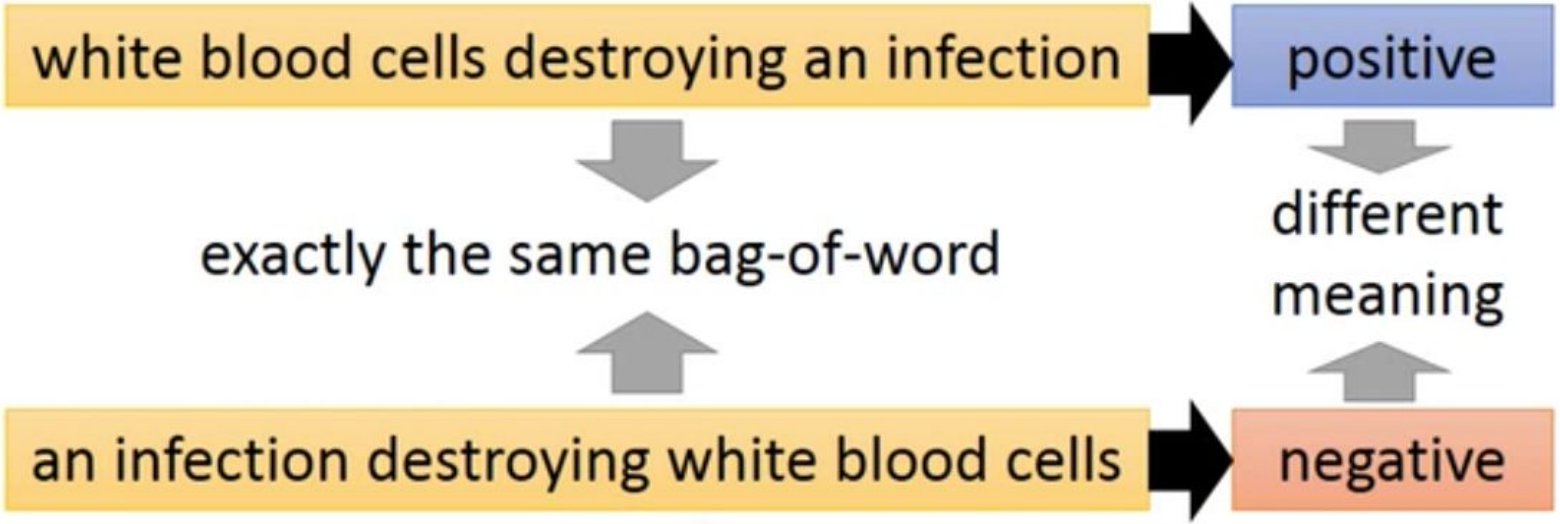
Or, one word would have two word vectors.

Document embeddings

Bag-of-words
Average...

Beyond Bag of Word

- To understand the meaning of a word sequence, the order of the words can not be ignored.



We then need RNNs, LSTMs to achieve embeddings for sentences or documents...

RNNs LSTMs

Sequential Data

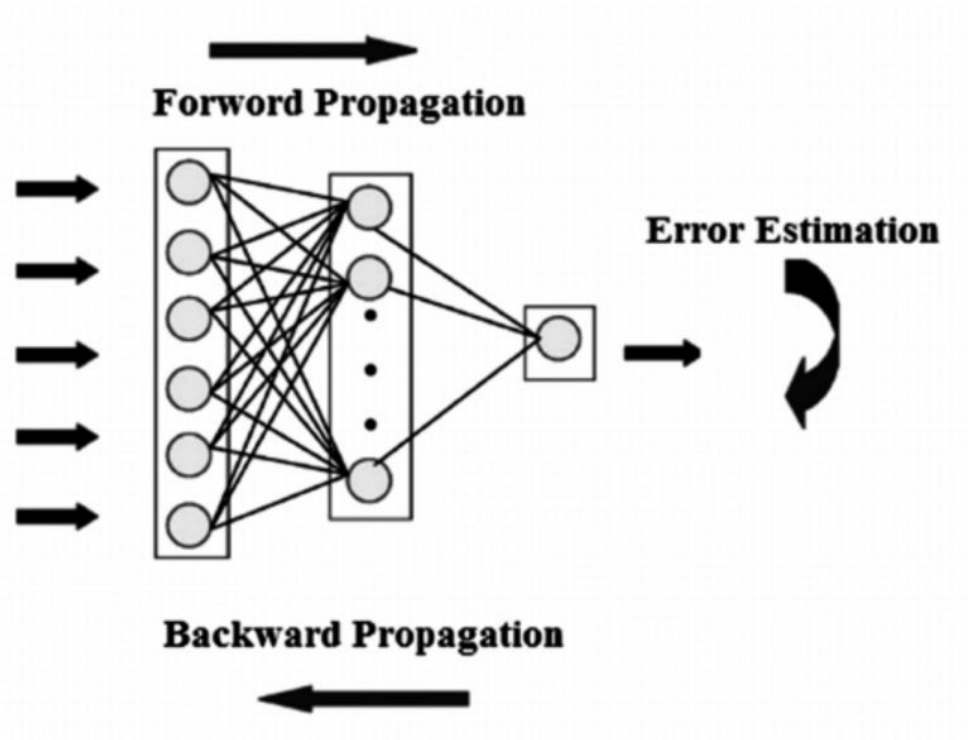
Sentences:

the cat was found under the bed; New York Times

Stock prices:

The current value depends on the history!

From neural networks to RNNs

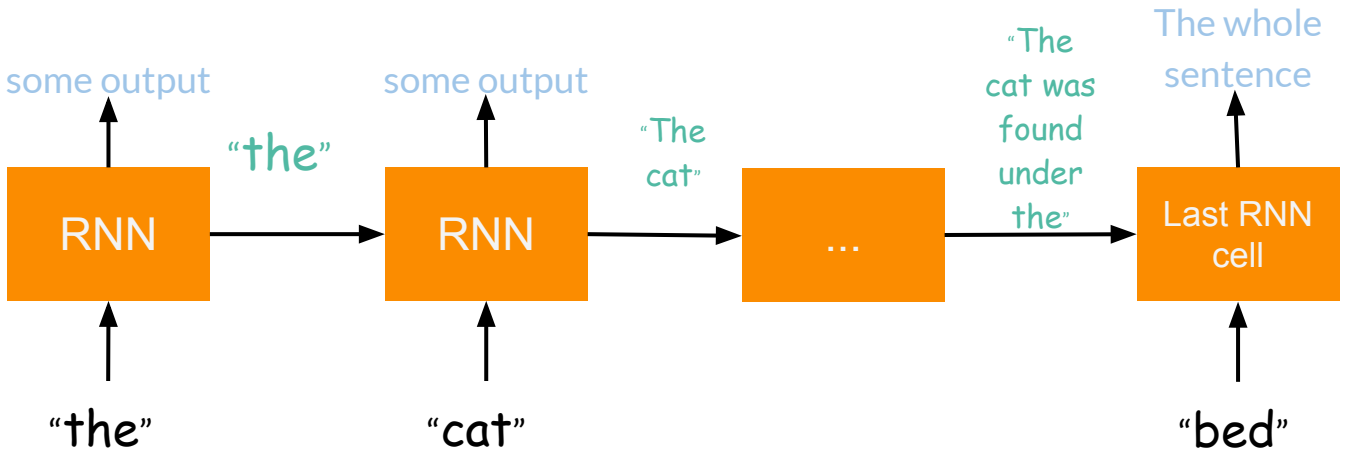


Input → Hidden → Output

Add the History...

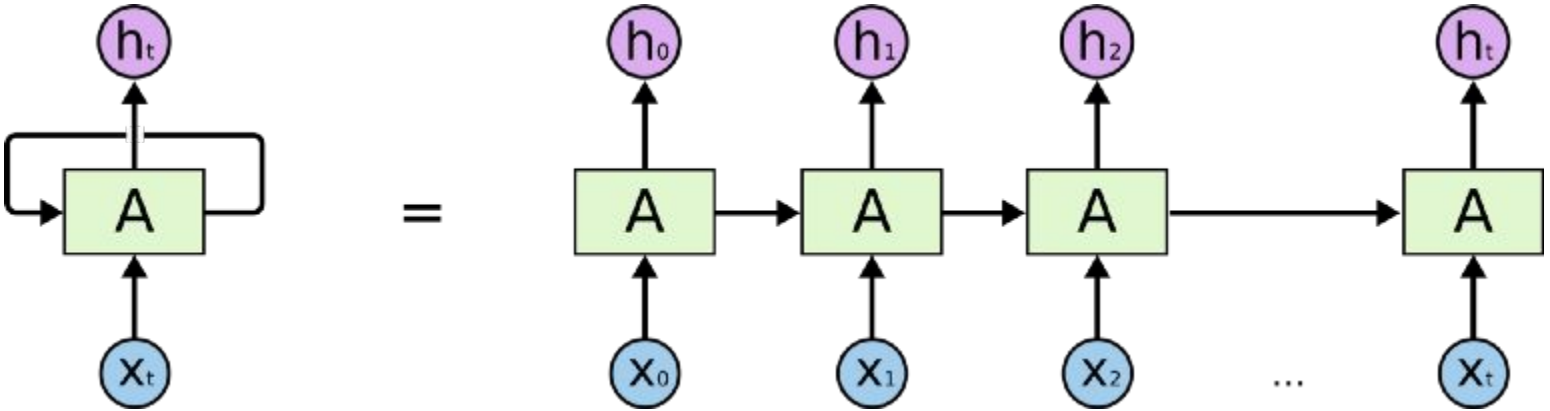
*Input + **Previous Hidden** → Hidden → Output*

"the cat was found under the bed"



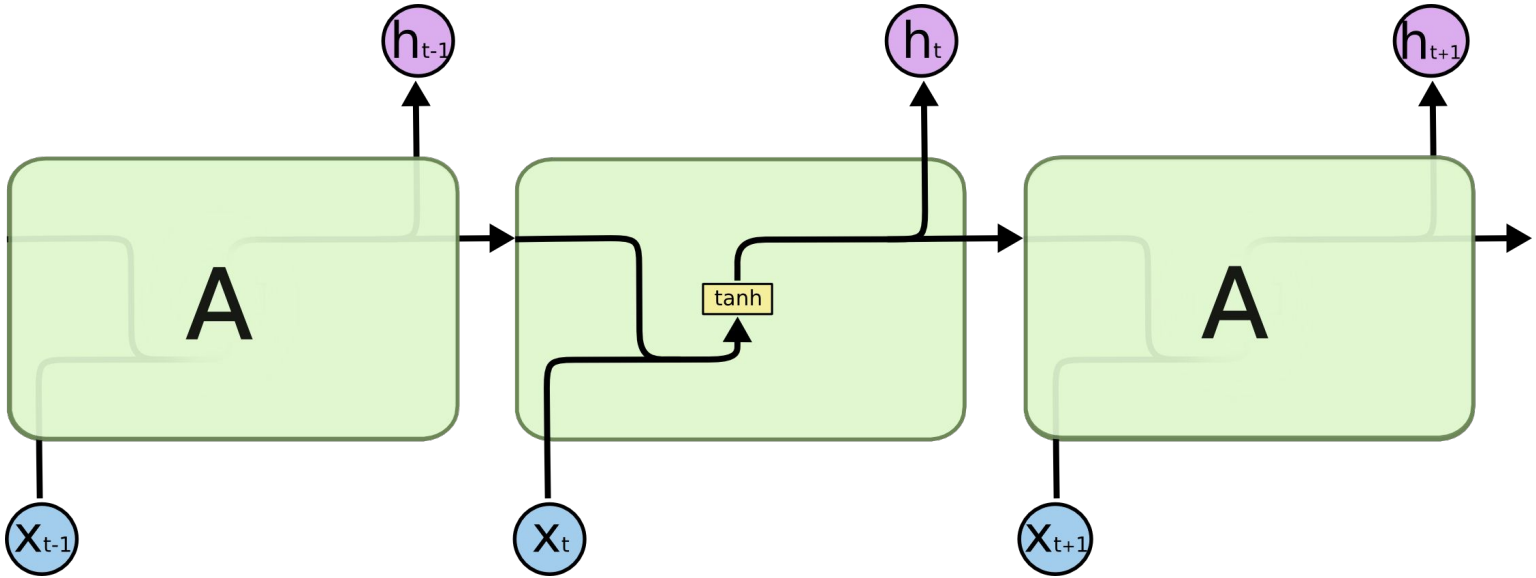
Add the History...

*Input + **Previous Hidden** → Hidden → Output*



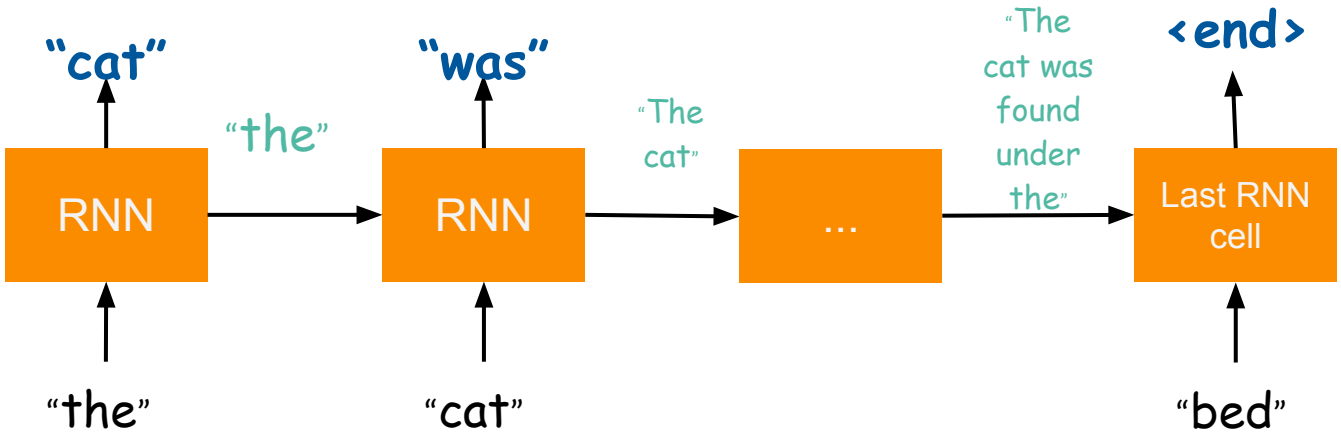
Recurrent Neural Nets

$$h_t = Tanh(W[h_{t-1}, x_t] + b)$$

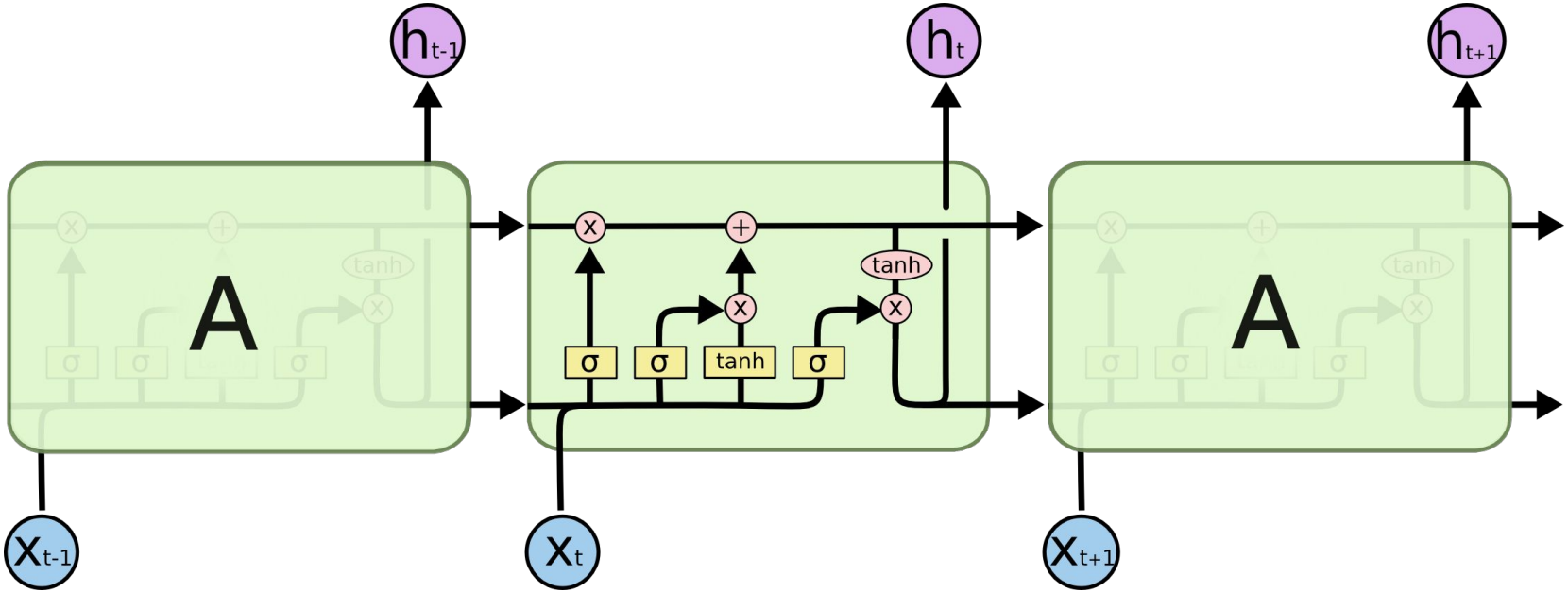


The outputs of RNNs

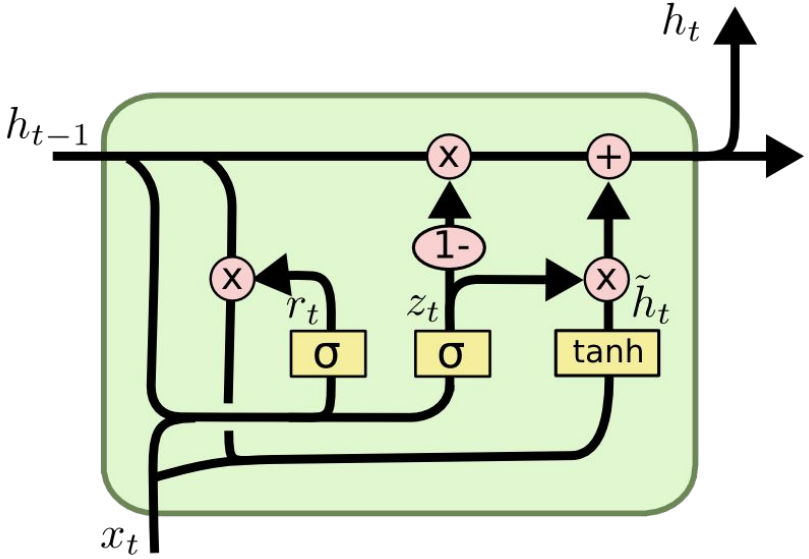
"the cat was found under the bed"



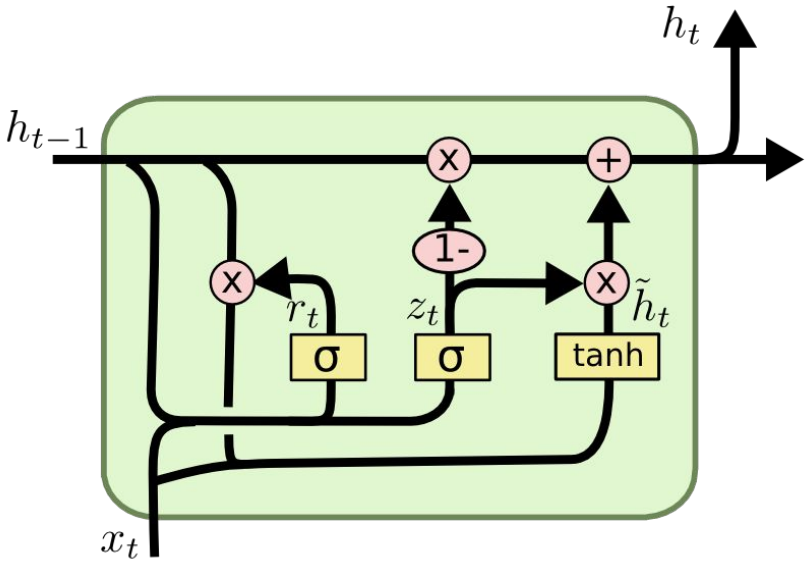
LSTMs



Simple Math



Simple Math



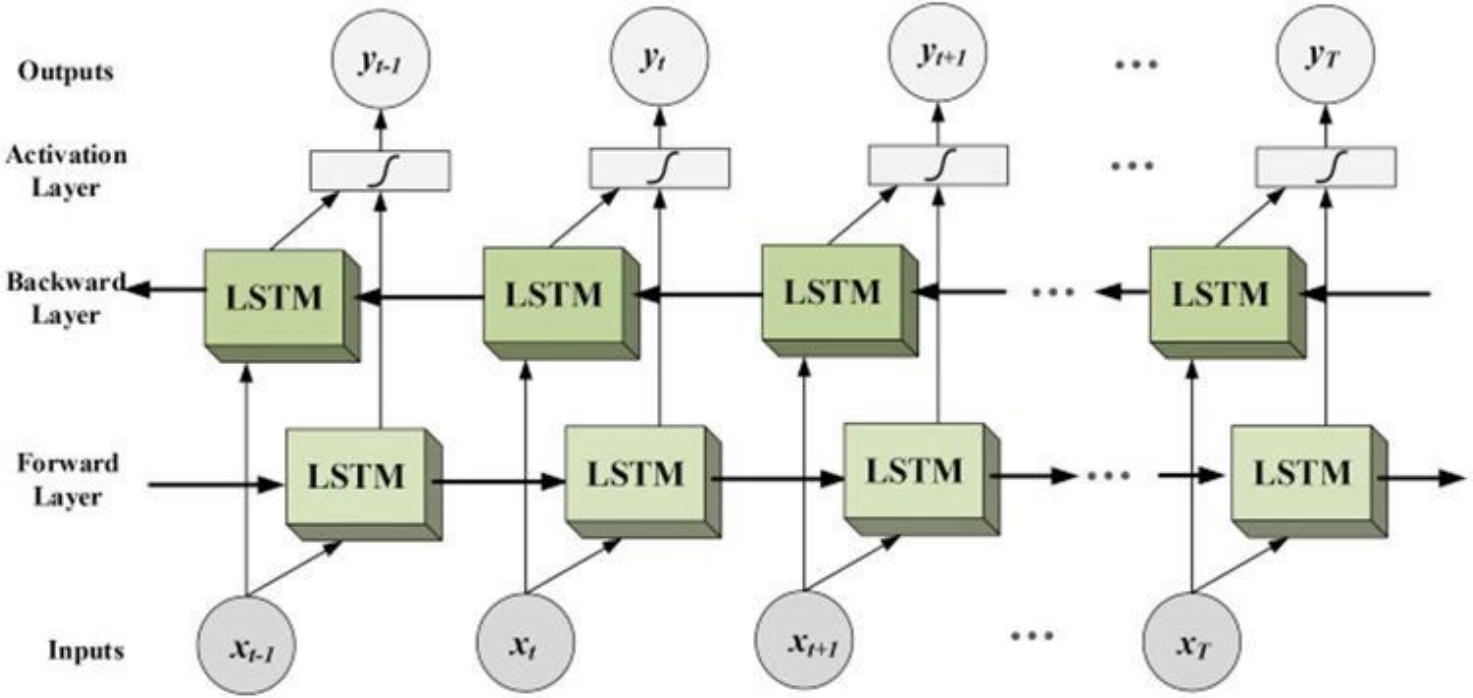
$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

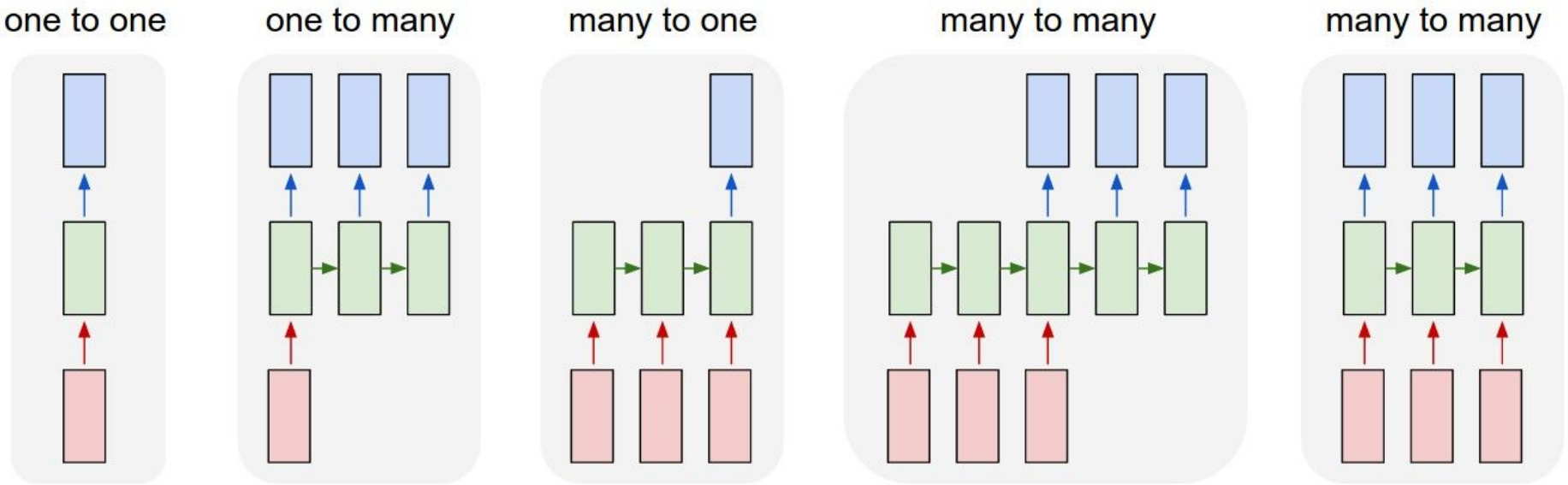
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

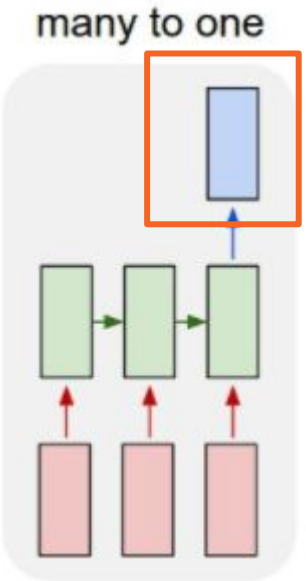
Bi-directional LSTMs



Types



RNN/LSTMs for text classification

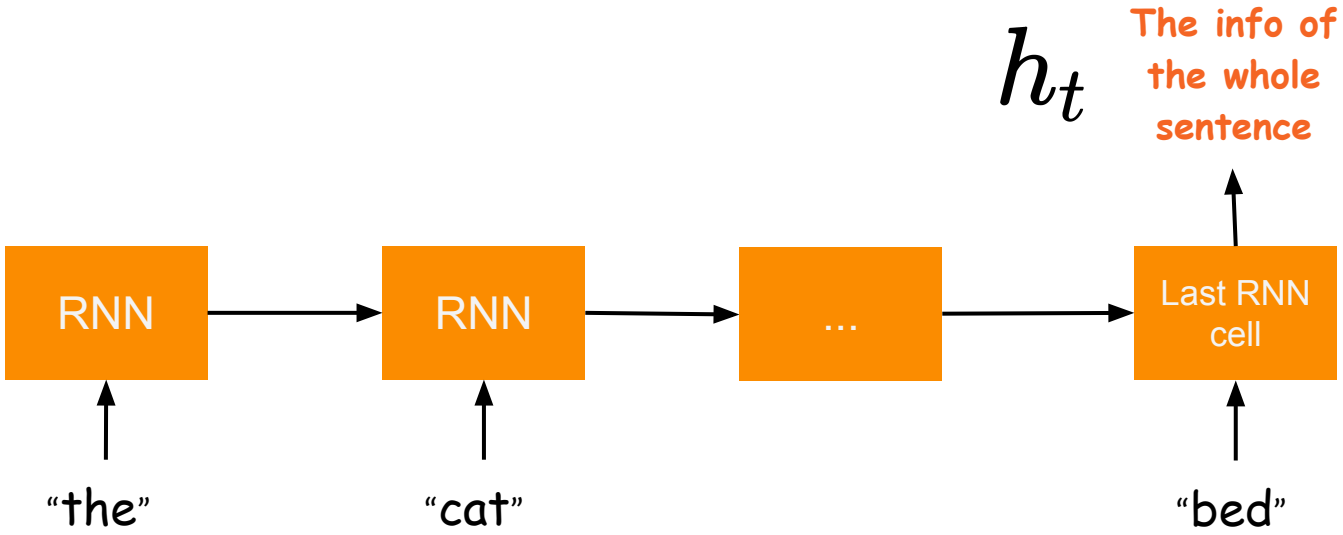


A vector representing the sentence

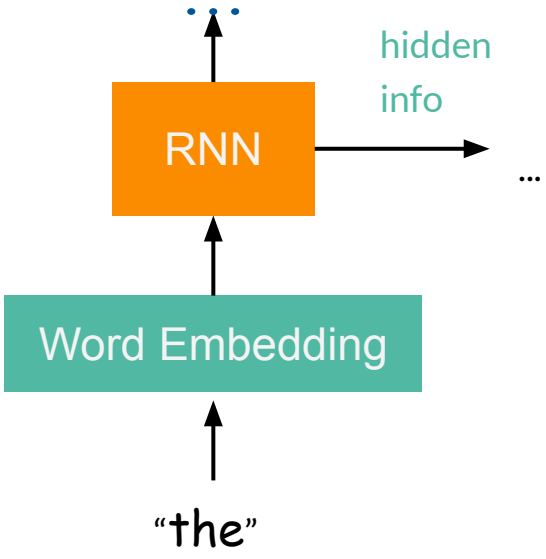
Sentence as input: “I like it”

RNN/LSTMs for text classification

"the cat was found under the bed"



RNN/LSTMs for text classification



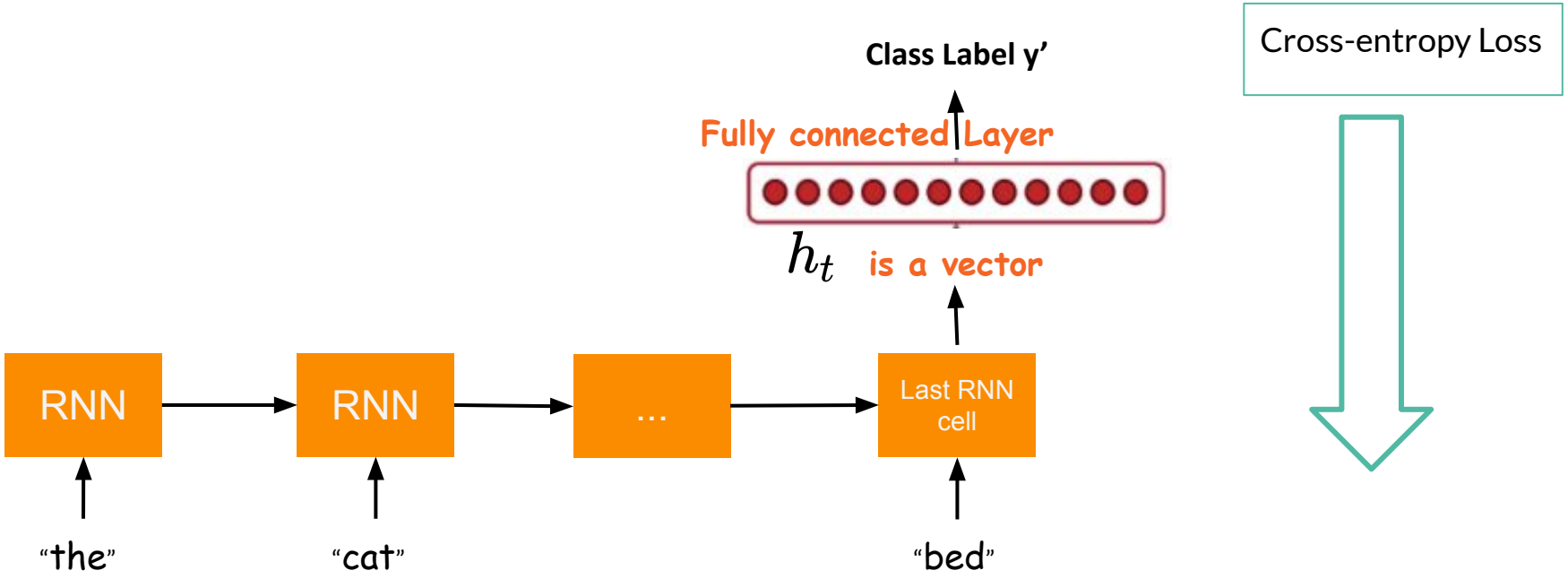
Input X: sentence

Output Y: classification tags {0,1}.

Simplified case: we use the last hidden output to predict the labels, because we think it..

The info of the whole sentence

Backpropagation in RNNs



In Pytorch (Next week)

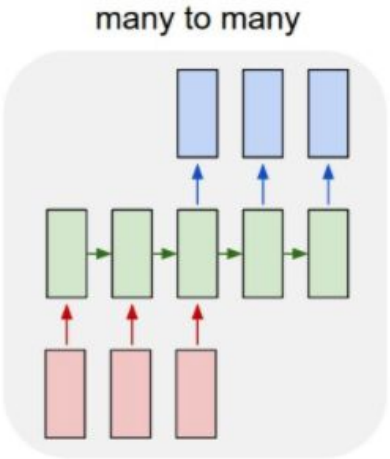
Then we add a linear layer to do classification…

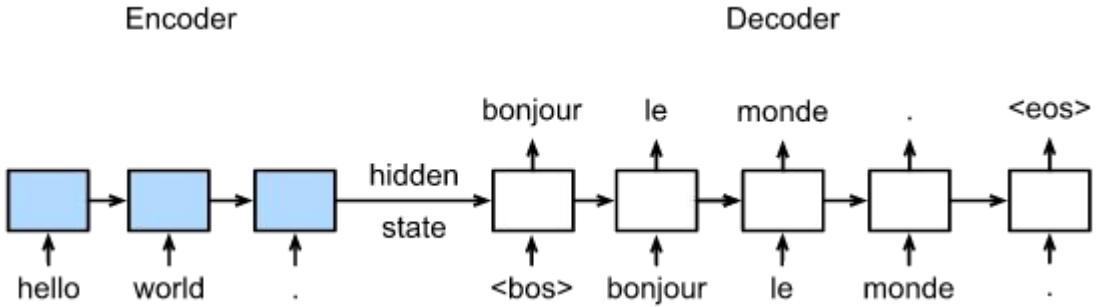
```
self.embedding = nn.Embedding(input_dim, embedding_dim)
self.rnn = nn.RNN(embedding_dim, hidden_dim)
self.fc = nn.Linear(hidden_dim, output_dim)
```

Code and Assignment (finish the LSTM part)

Reading: link

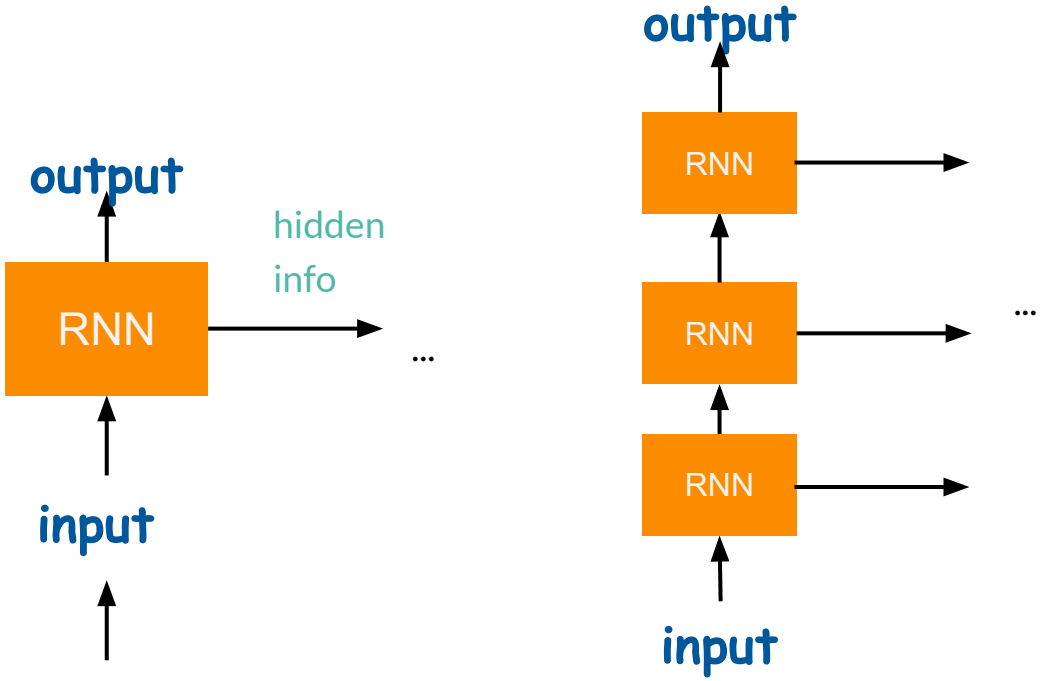
Many-to-many: seq2seq model





Deep LSTMs

More than one layer, the net is getting deeper.

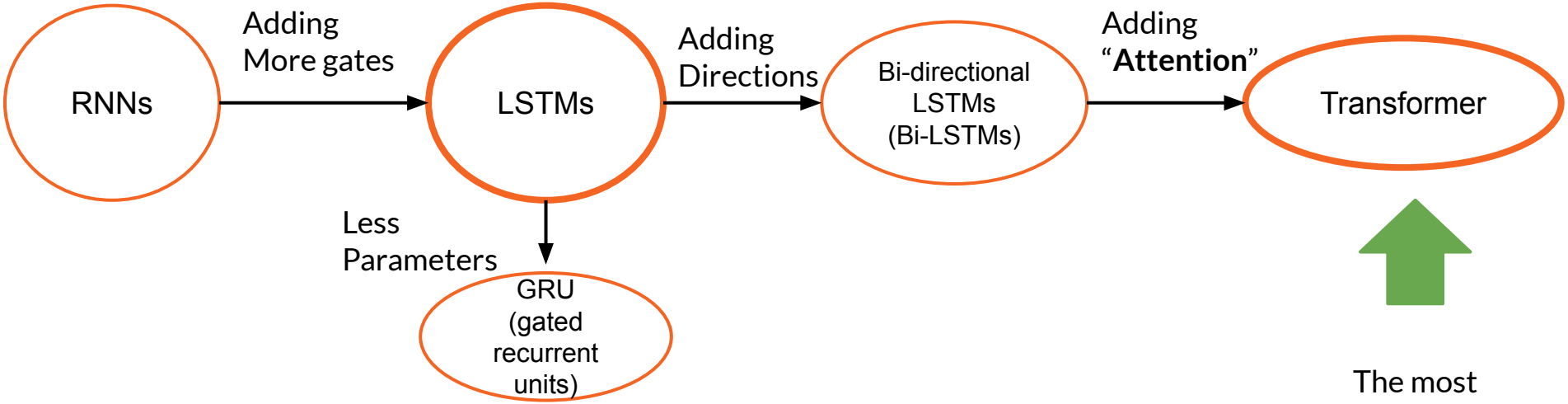


Machine Translation Seq2seq

RNNs

A model used for Neural Machine Translation.

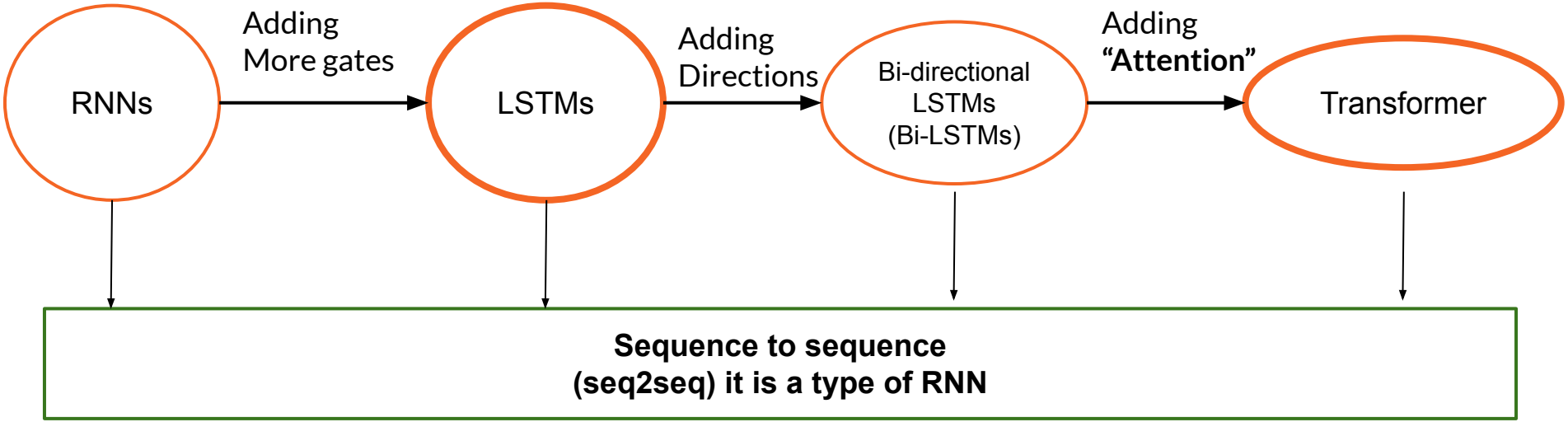
RNNs and Variations



We will focus on applying LSTMs (this week) and Transformer (next week).

The most advanced RNN model.

RNNs and Variations in Application



Machine Translation

Statistical MT

Based on probabilities.

P(I| 私は) = ?
P(you| 私は) = ?
P(eat| 私は) = ?

Neural MT

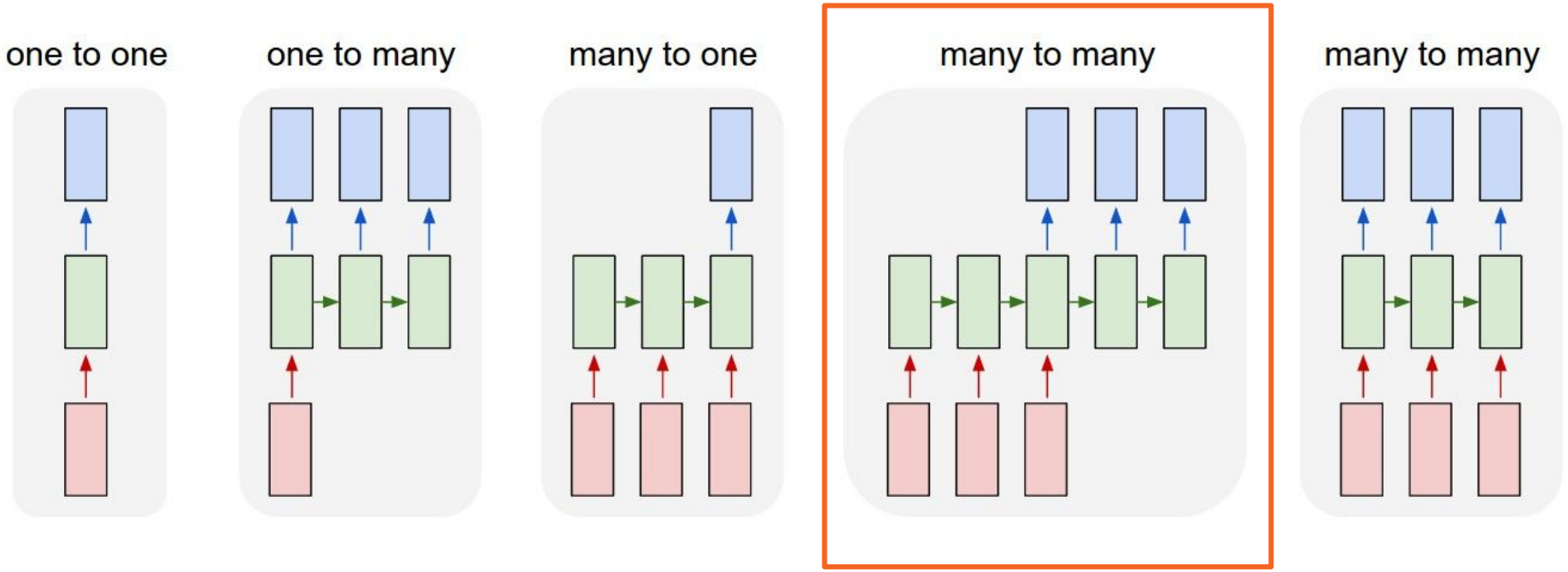
Mainly we use RNNs

Learn parameters...

Seq2seq

Neural Machine Translation

Seq2seq: Advanced RNN



Motivation, applications

Input is a **sequence**, output is also a **sequence**. “generative”

Machine Translation:

English text => French text

Summarization:

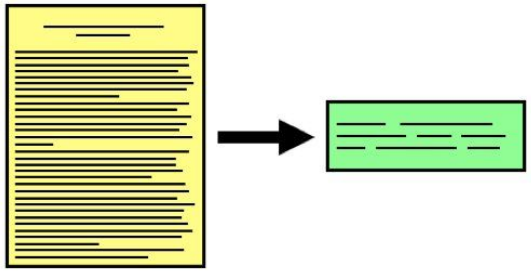
Full document => Shot summary

Chatbot:

Question => Answer

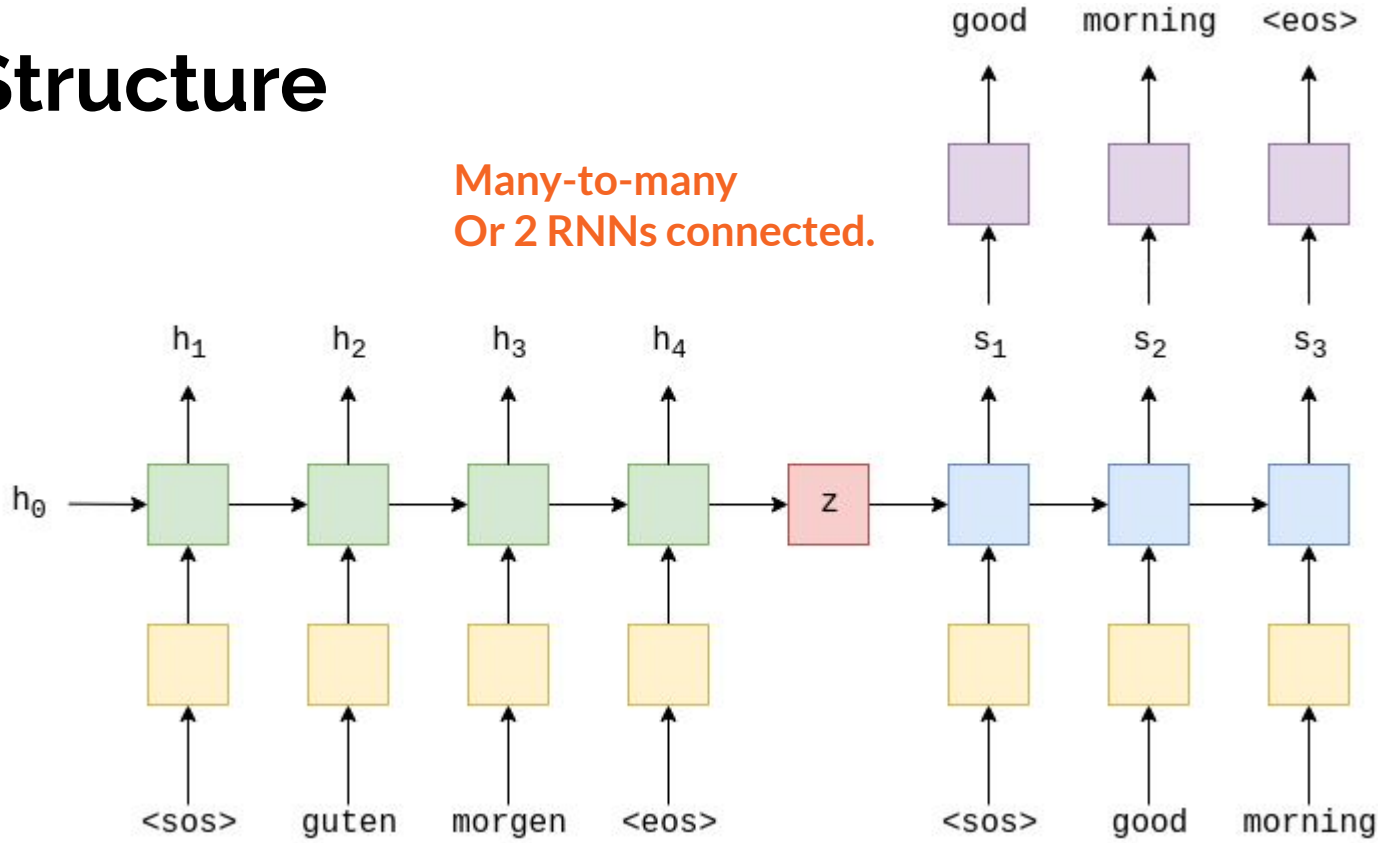
Speech Recognition:

Audio speech => text

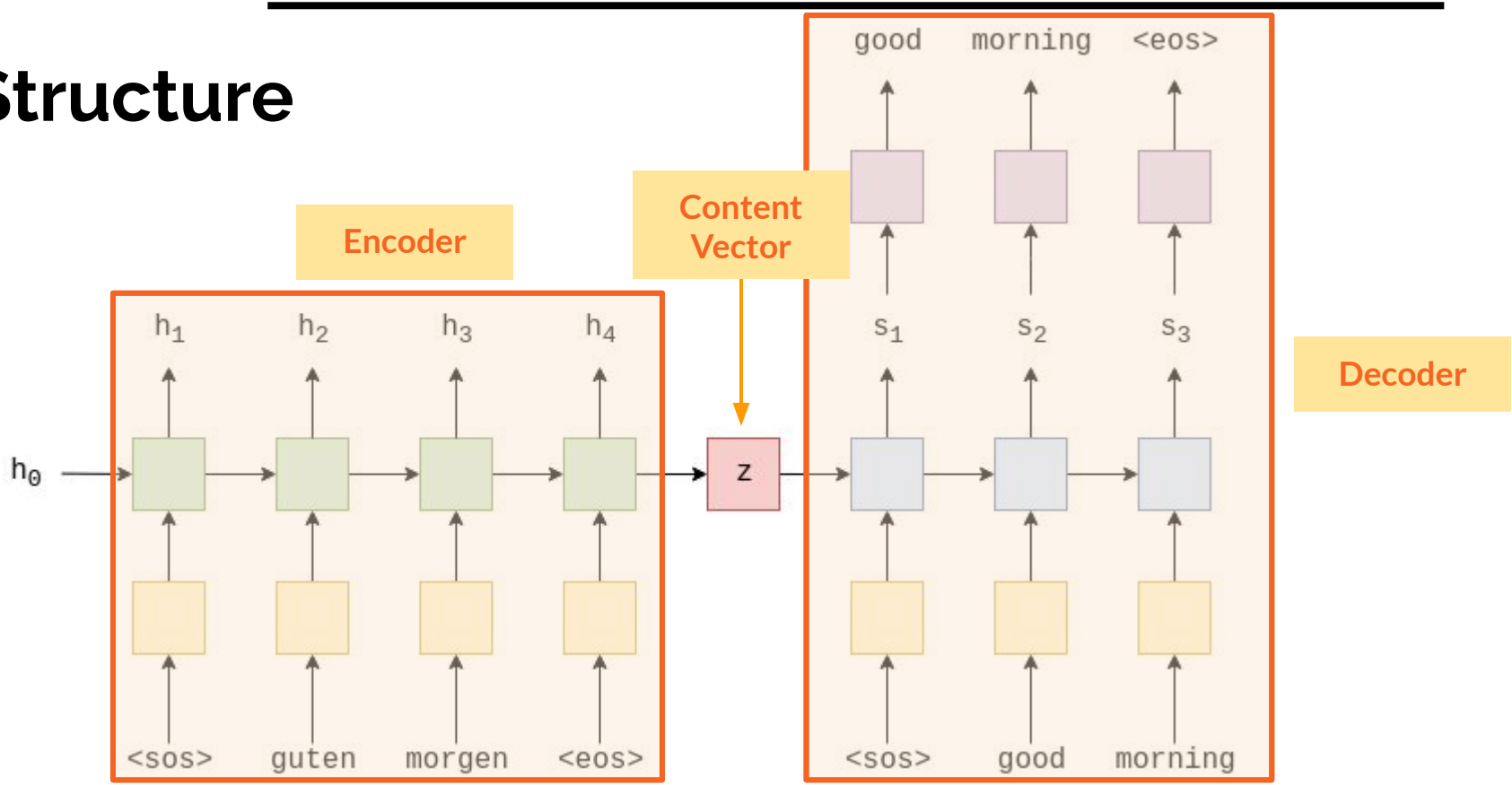


Structure

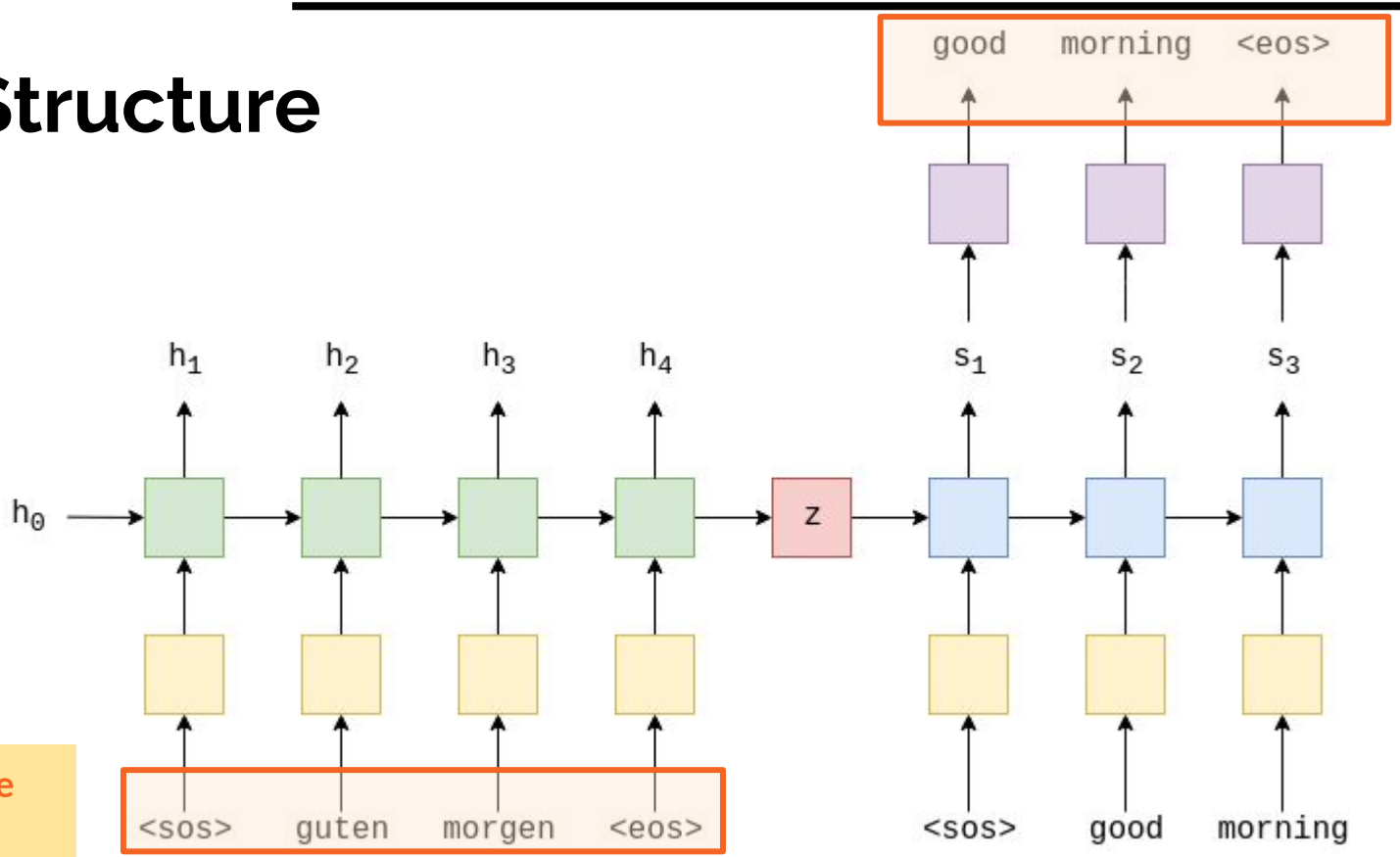
Many-to-many
Or 2 RNNs connected.



Structure

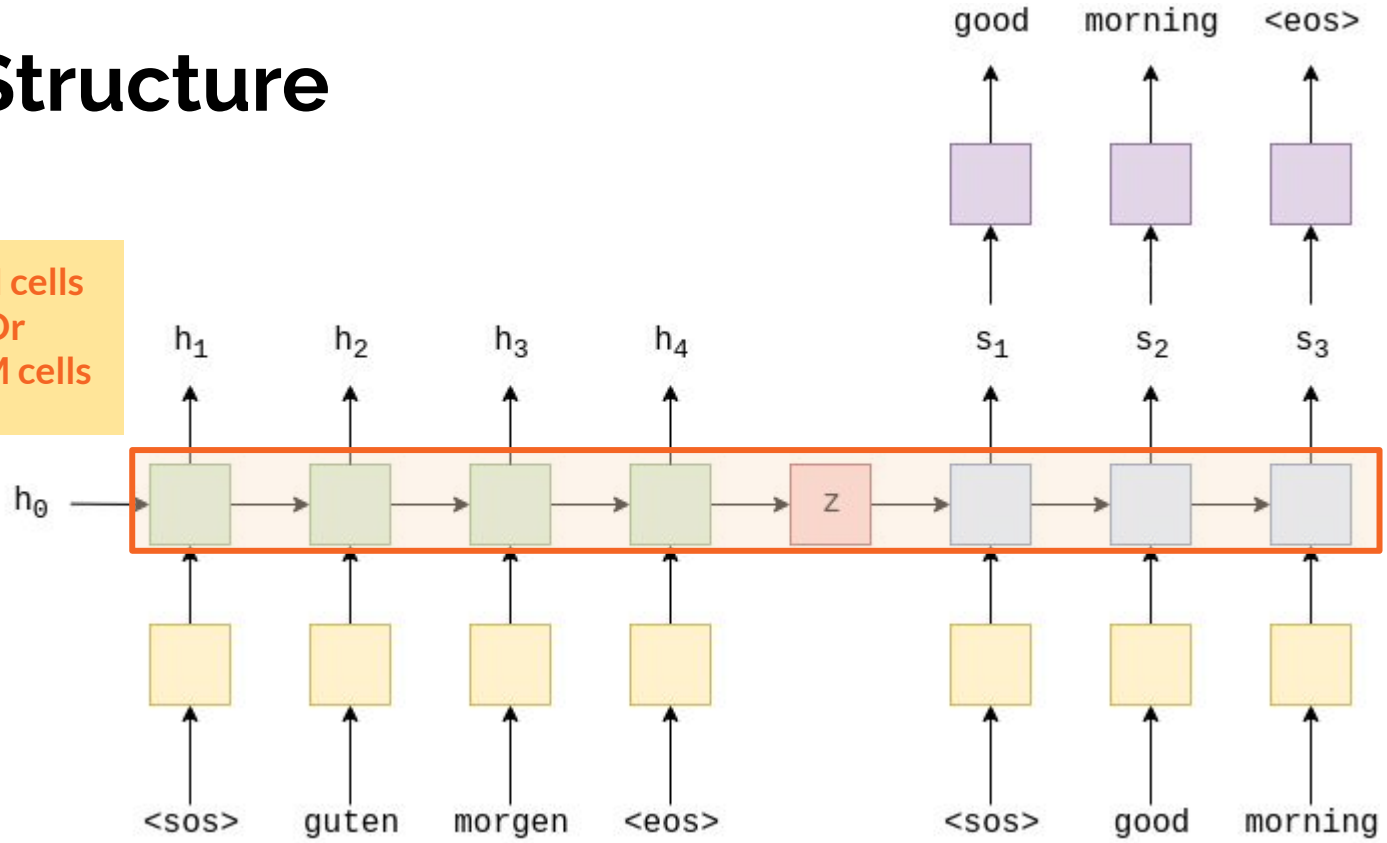

Encoder
Content Vector
Decoder
h0
h1
h2
h3
h4
z
s1
s2
s3
<sos>
guten
morgen
<eos>
<sos>
good
morning
good
morning
<eos>

Structure

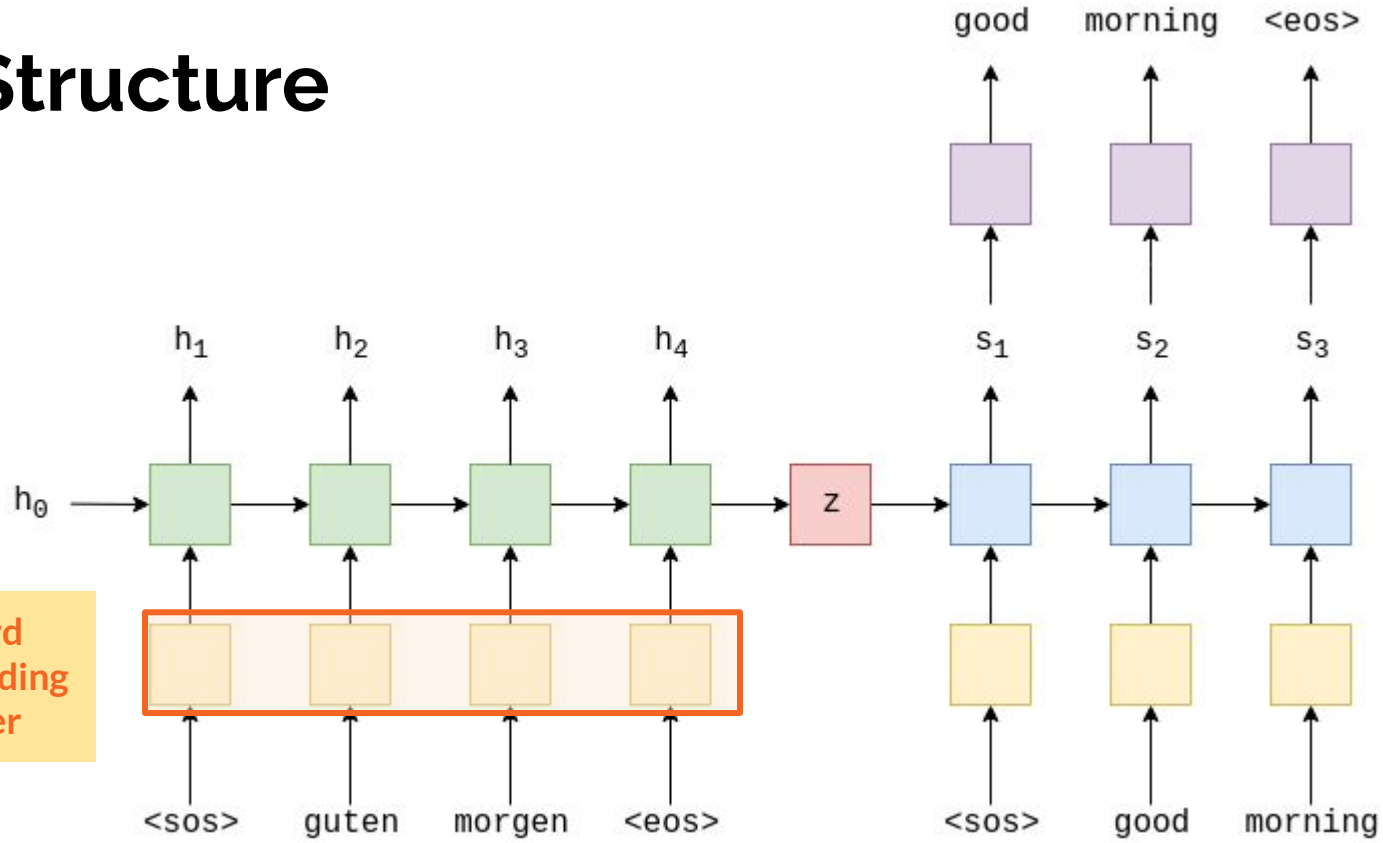

good morning <eos>
Target output
h1 h2 h3 h4
s1 s2 s3
h0
z
<sos> guten morgen <eos>
<sos> good morning
Source input

Structure

RNN cells
Or
LSTM cells

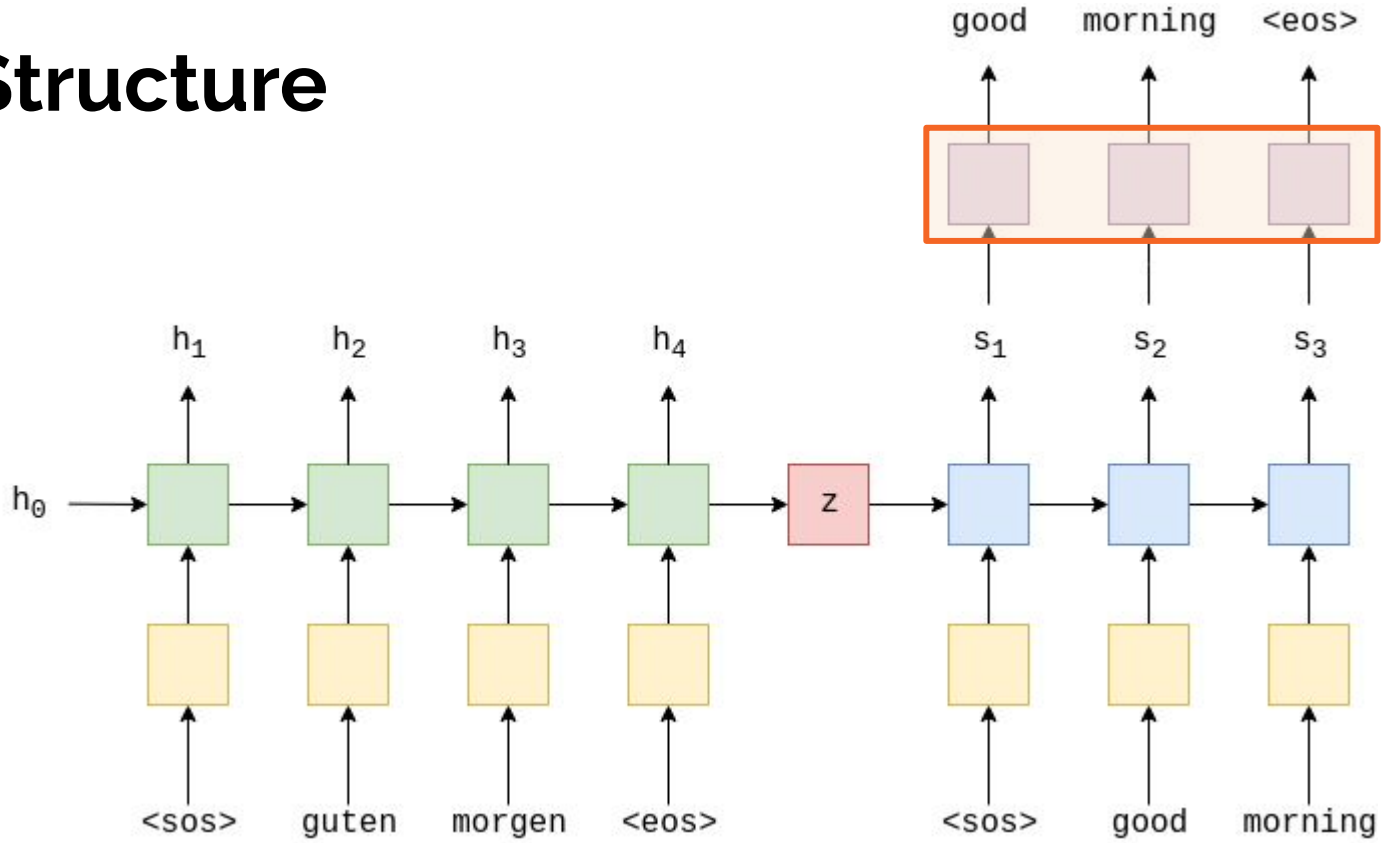


Structure

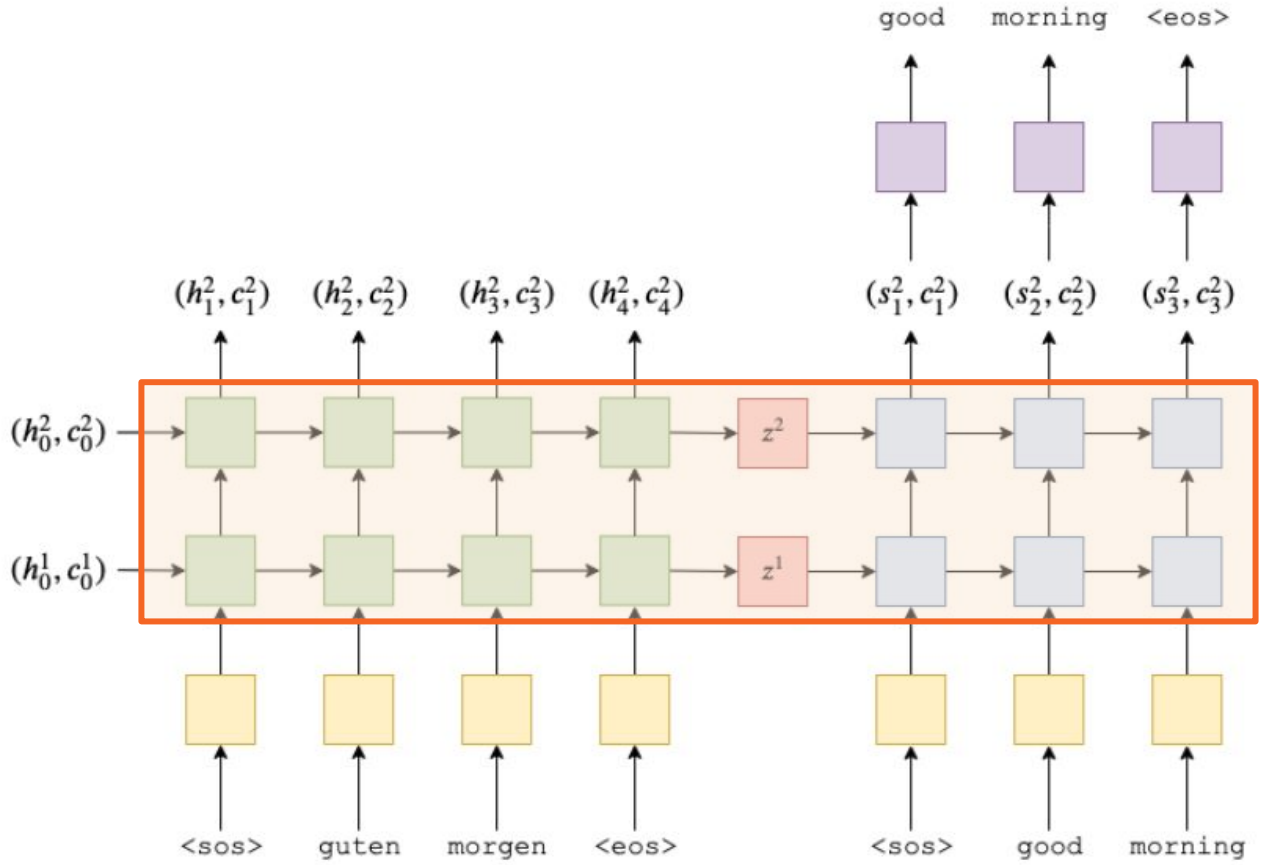


Word Embedding Layer

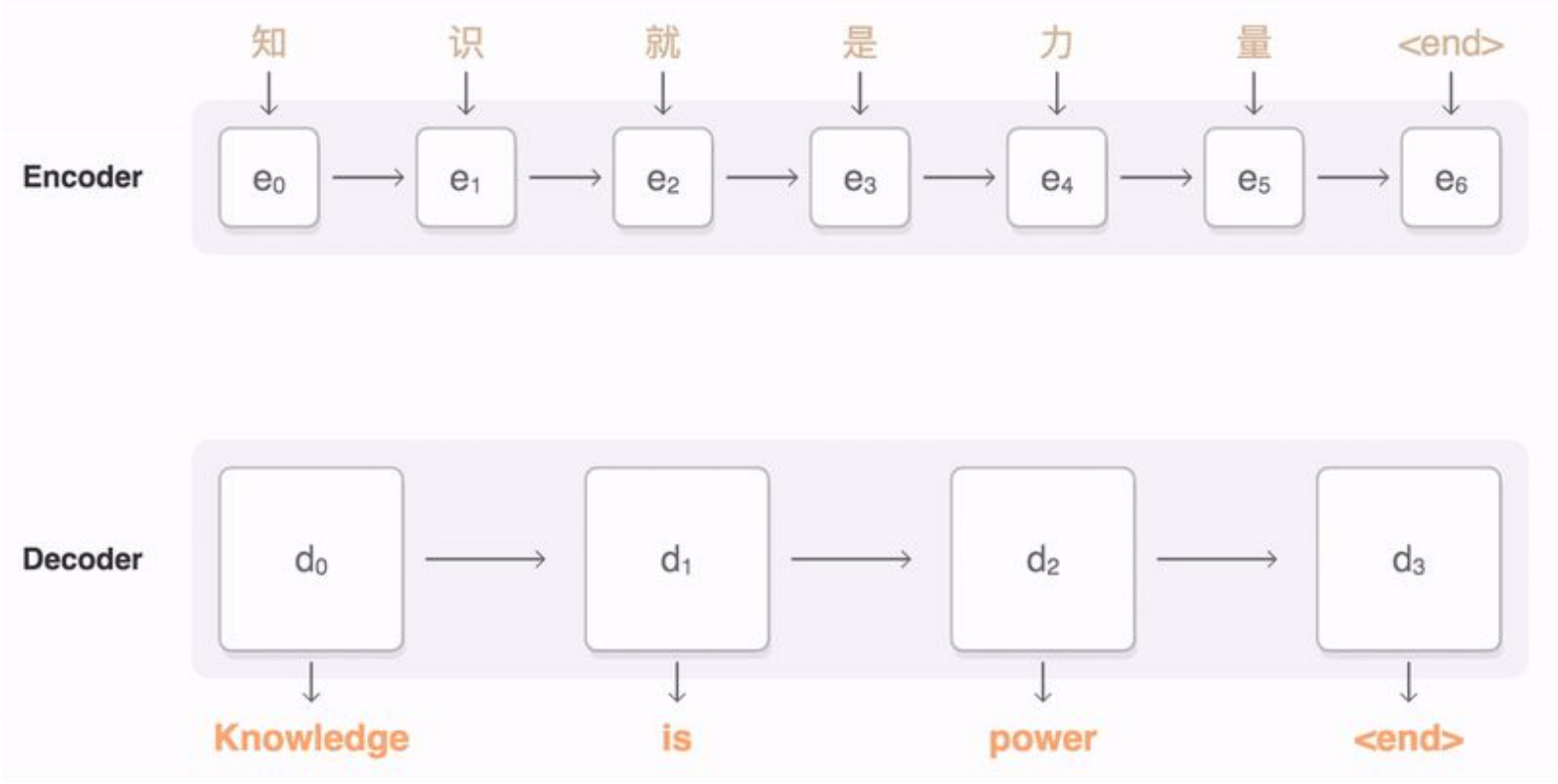
Structure


good
morning
<eos>
Linear Layer (predict words)
h1
h2
h3
h4
s1
s2
s3
h0
z
<sos>
guten
morgen
<eos>
<sos>
good
morning

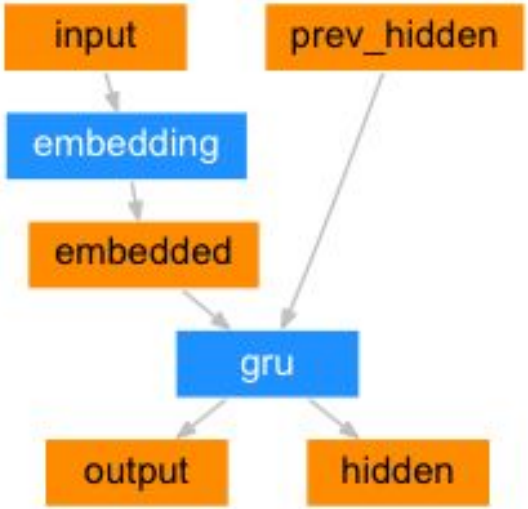
The RNN/LSTMs can be stacked



A gif to show encoder-decoder



Programming: Encoder with PyTorch



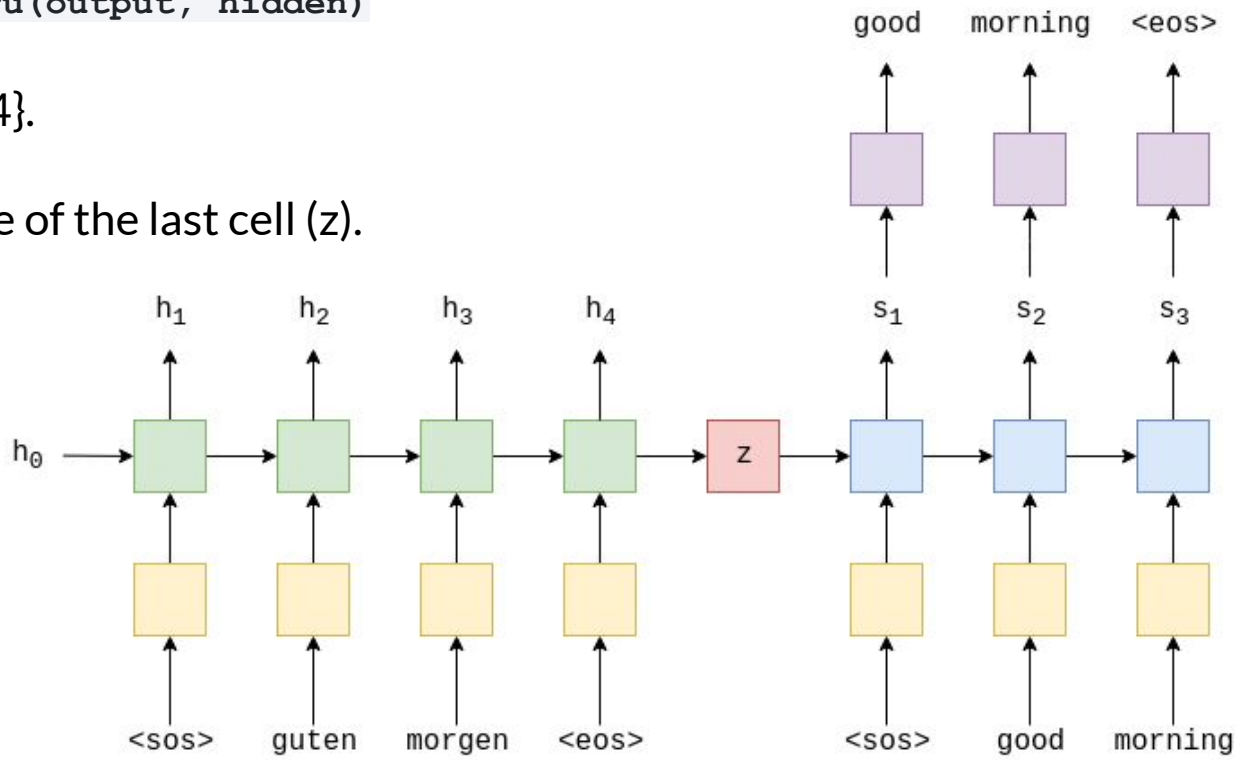

```
class EncoderRNN(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(EncoderRNN, self).__init__()
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding(input_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size)

    def forward(self, input, hidden):
        embedded = self.embedding(input).view(1, 1, -1)
        output = embedded
        output, hidden = self.gru(output, hidden)
        return output, hidden

    def initHidden(self):
        return torch.zeros(1, 1, self.hidden_size, device=device)
```

Programming: Encoder with PyTorch

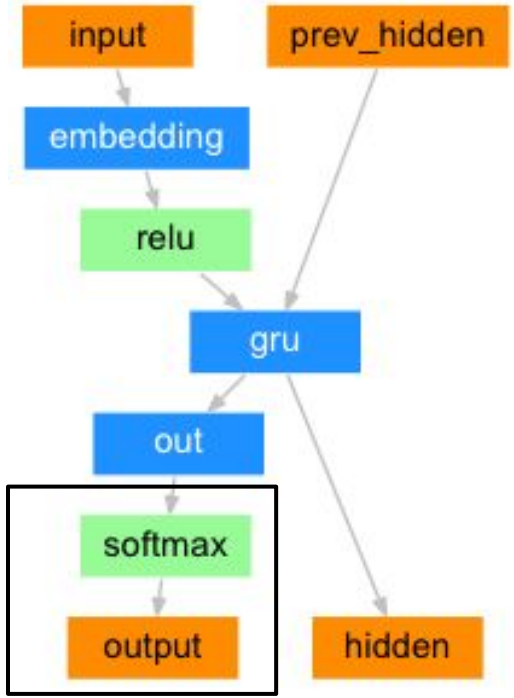
```
output, hidden = self.gru(output, hidden)
```

‘Output’ is H = {h1, h2…, h4}.

‘Hidden’ is the hidden state of the last cell (z).



Programming: Decoder with PyTorch




```
class DecoderRNN(nn.Module):
    def __init__(self, hidden_size, output_size):
        super(DecoderRNN, self).__init__()
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding(output_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size)
        self.out = nn.Linear(hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, input, hidden):
        output = self.embedding(input).view(1, 1, -1)
        output = F.relu(output)
        output, hidden = self.gru(output, hidden)
        output = self.softmax(self.out(output[0]))
        return output, hidden

    def initHidden(self):
        return torch.zeros(1, 1, self.hidden_size, device=device)
```

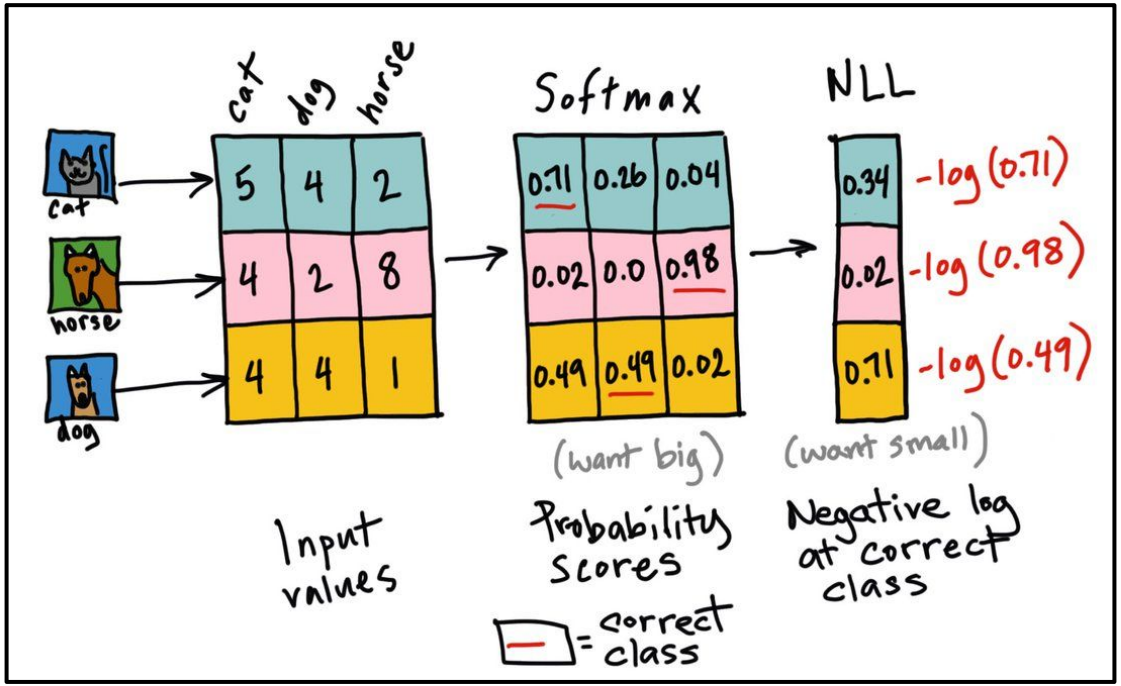
Predicting Words

Loss Function: torch.nn.NLLLoss()

Negative log likelihood loss

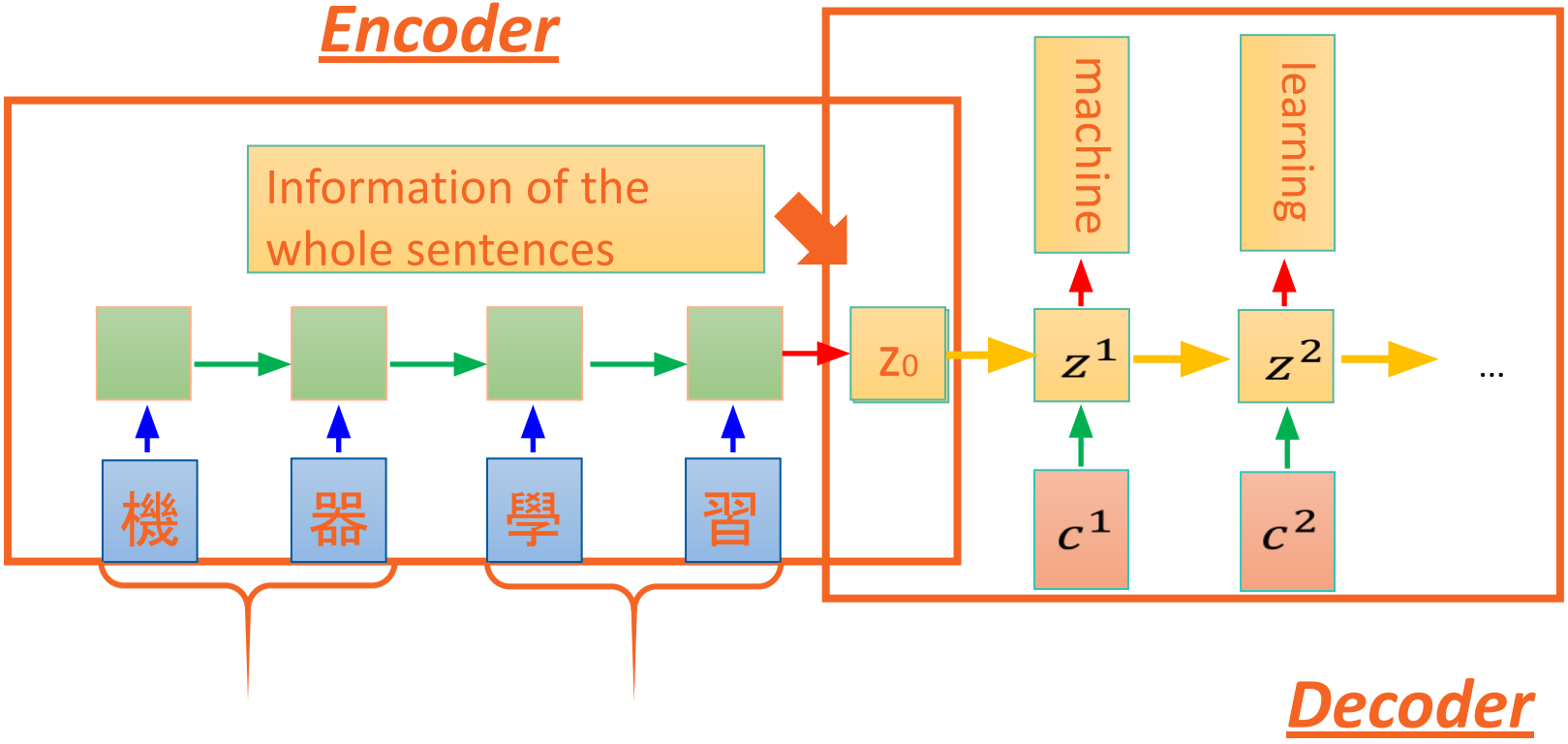
A classification loss: we are predicting the words!

$$L(\mathbf{y}) = -\log(\mathbf{y})$$

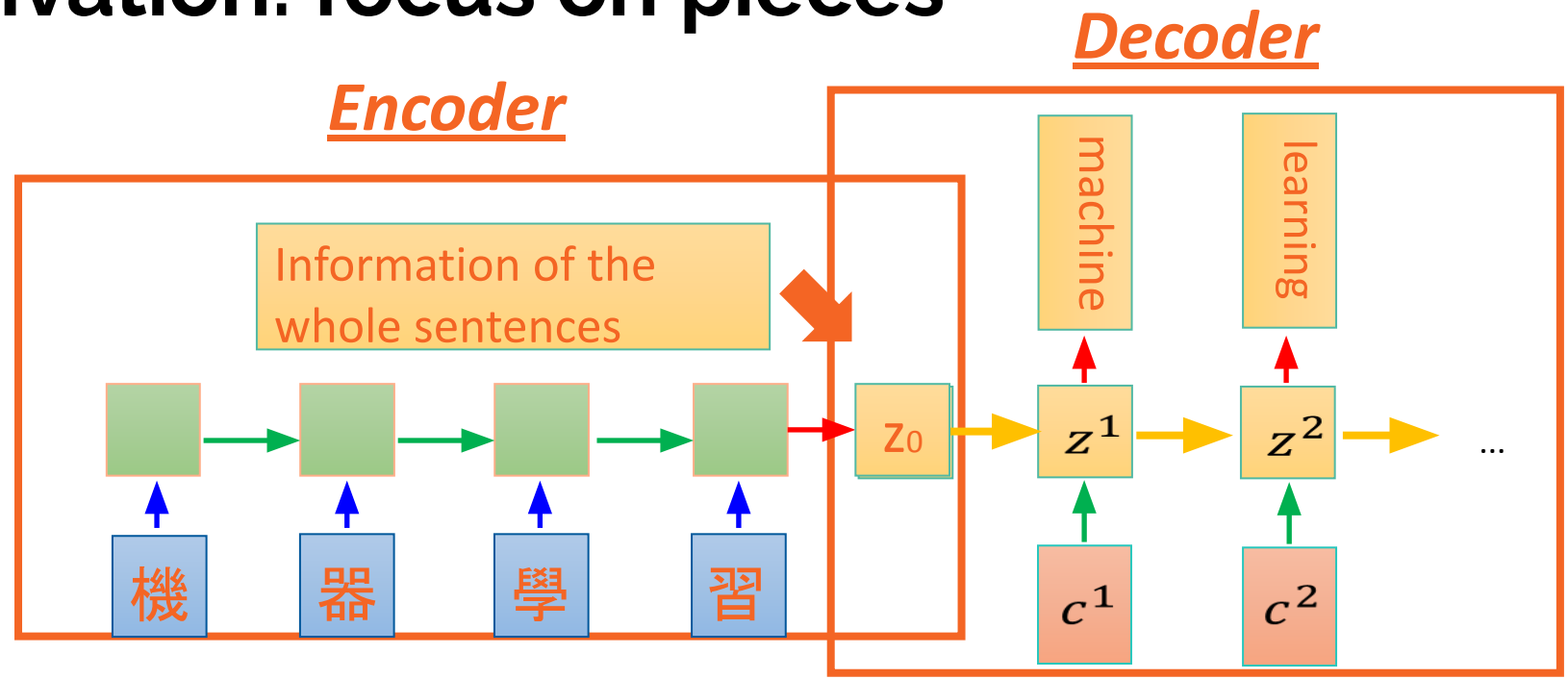


Attention

Motivation: focus on pieces



Motivation: focus on pieces



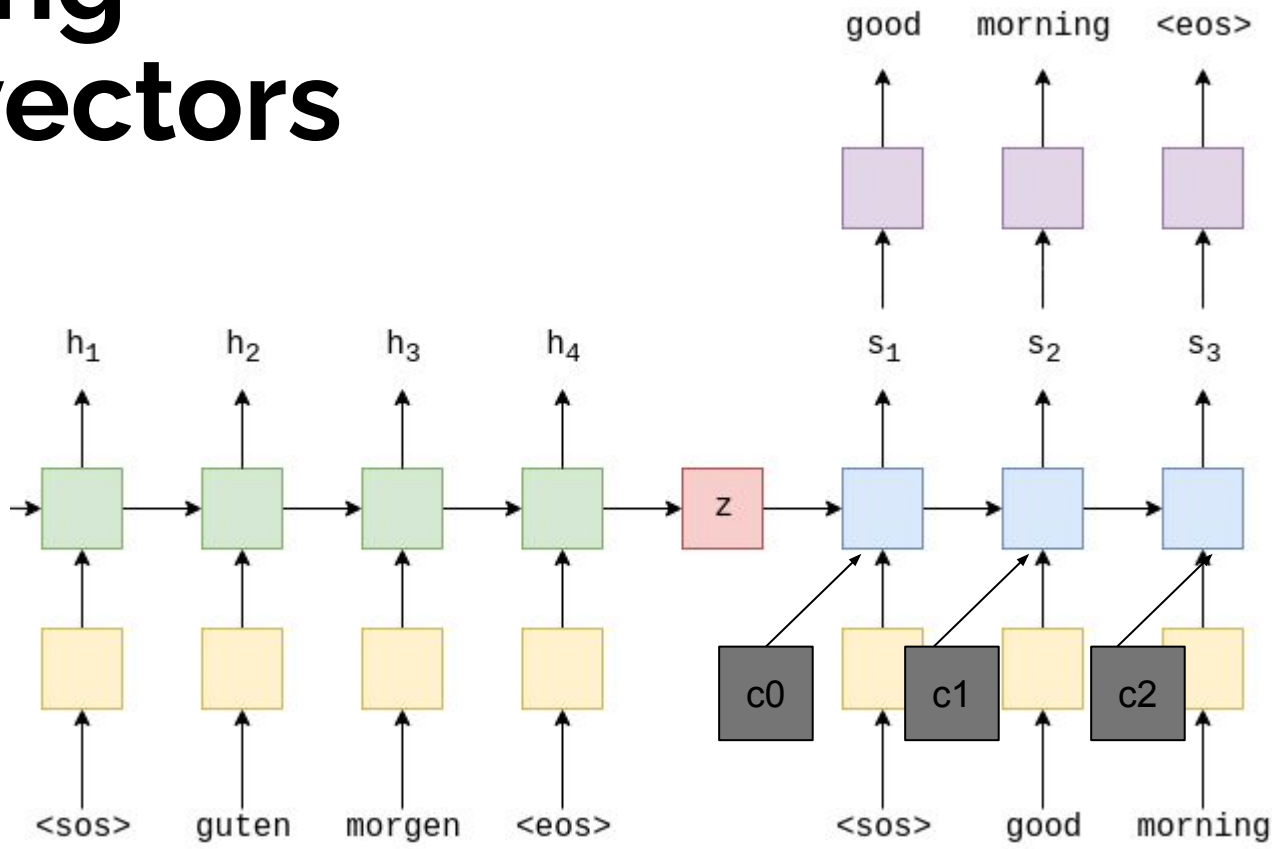
Ignore decoder input for now.

Introducing a new “cell”
Content Vector c

Introducing content vectors

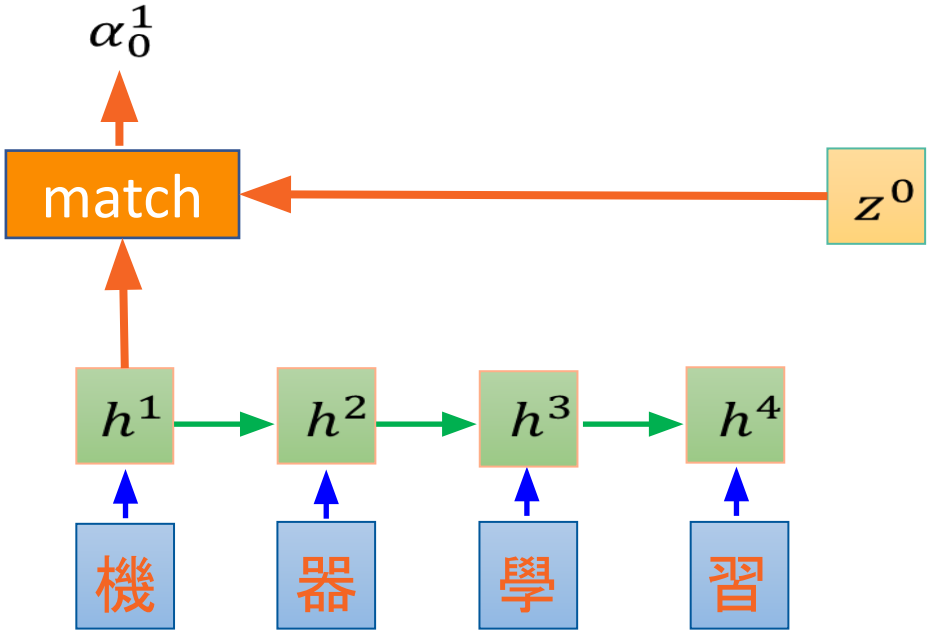
Ignore the decoder input words.

Introducing a new “cell” Content Vector c

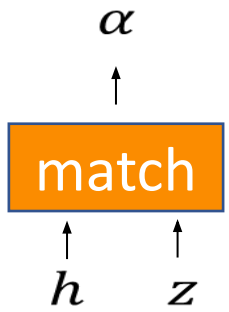


Seq2seq with attention

Attention-based model



Jointly learned with other part of the network



What is match ?

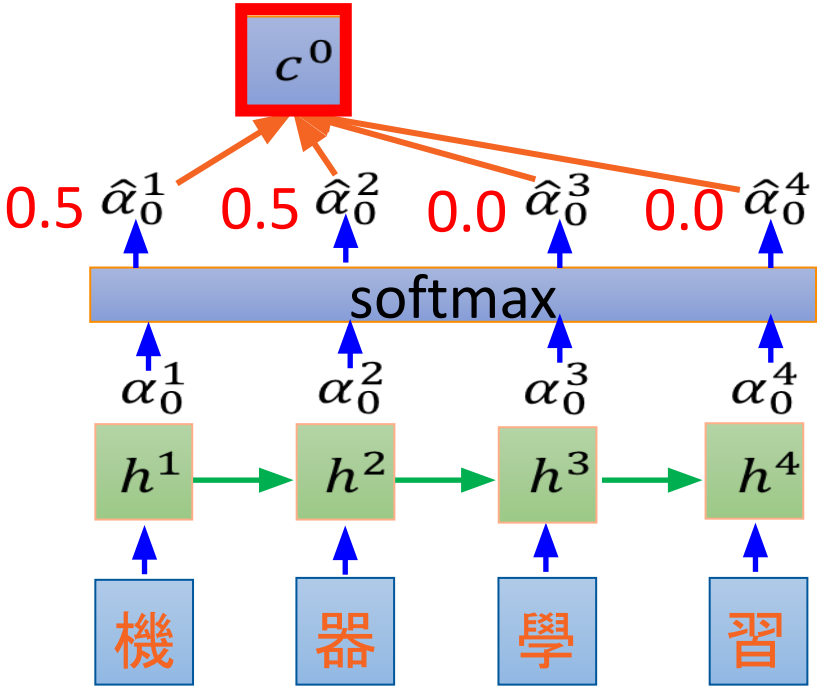
Design by yourself

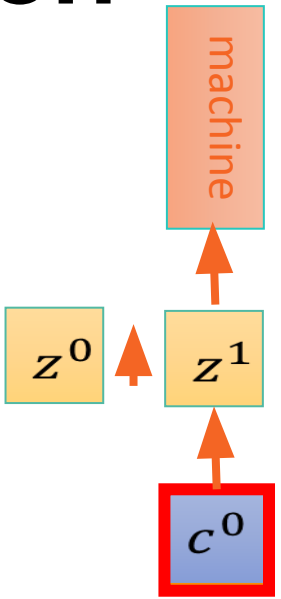
- Cosine similarity of z and h $\alpha = cos(z, h)$
- Small NN whose input is z and h, output a scalar

$$\alpha = h^T W z$$

Seq2seq with attention

Attention-based model





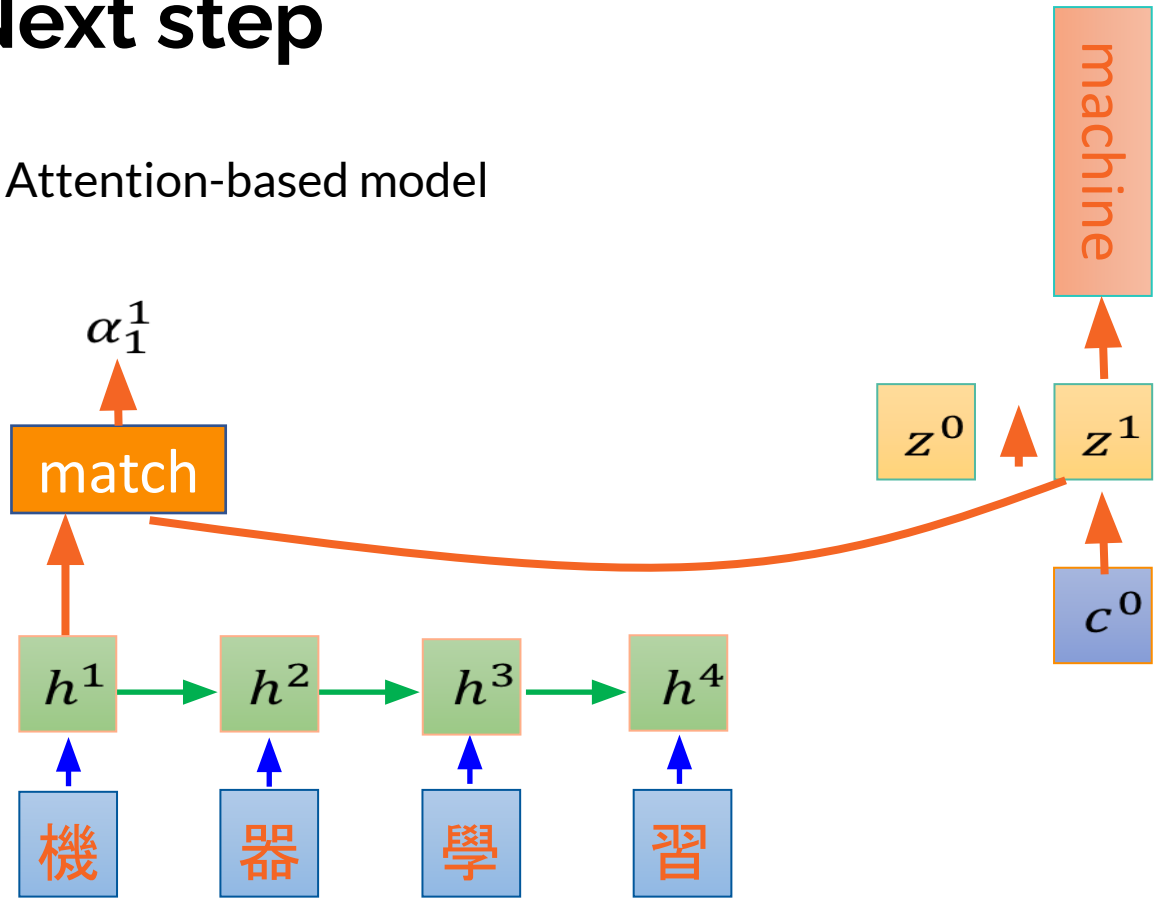
Decoder input

$$c^0 = \sum \hat{\alpha}_0^i h^i$$

$$= 0.5h^1 + 0.5h^2$$

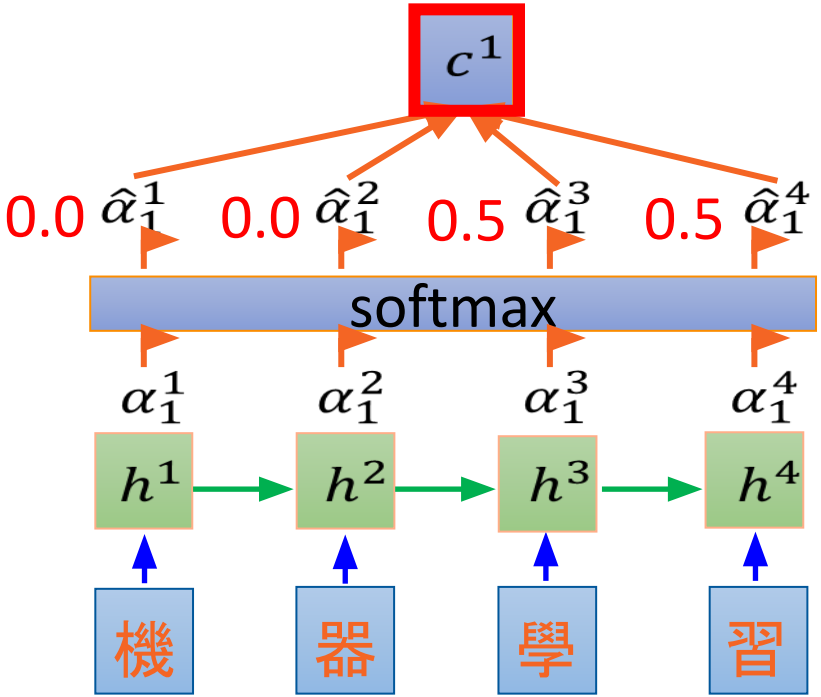
Next step

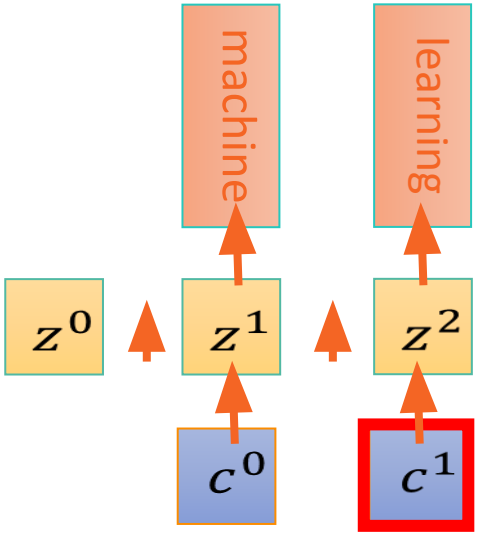
Attention-based model



Machine Translation

Attention-based model



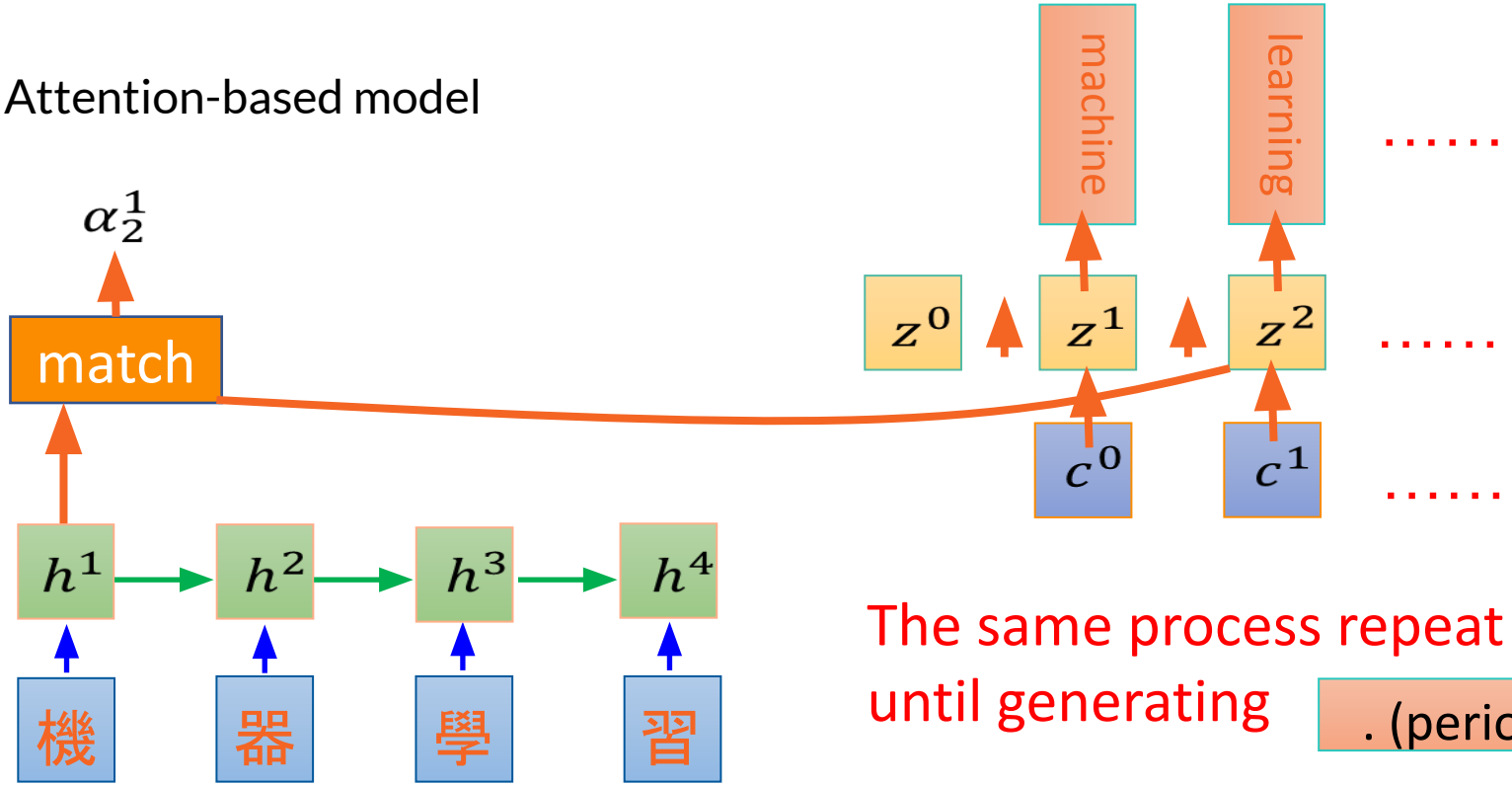


$$c^1 = \sum \hat{\alpha}_1^i h^i$$

$$= 0.5h^3 + 0.5h^4$$

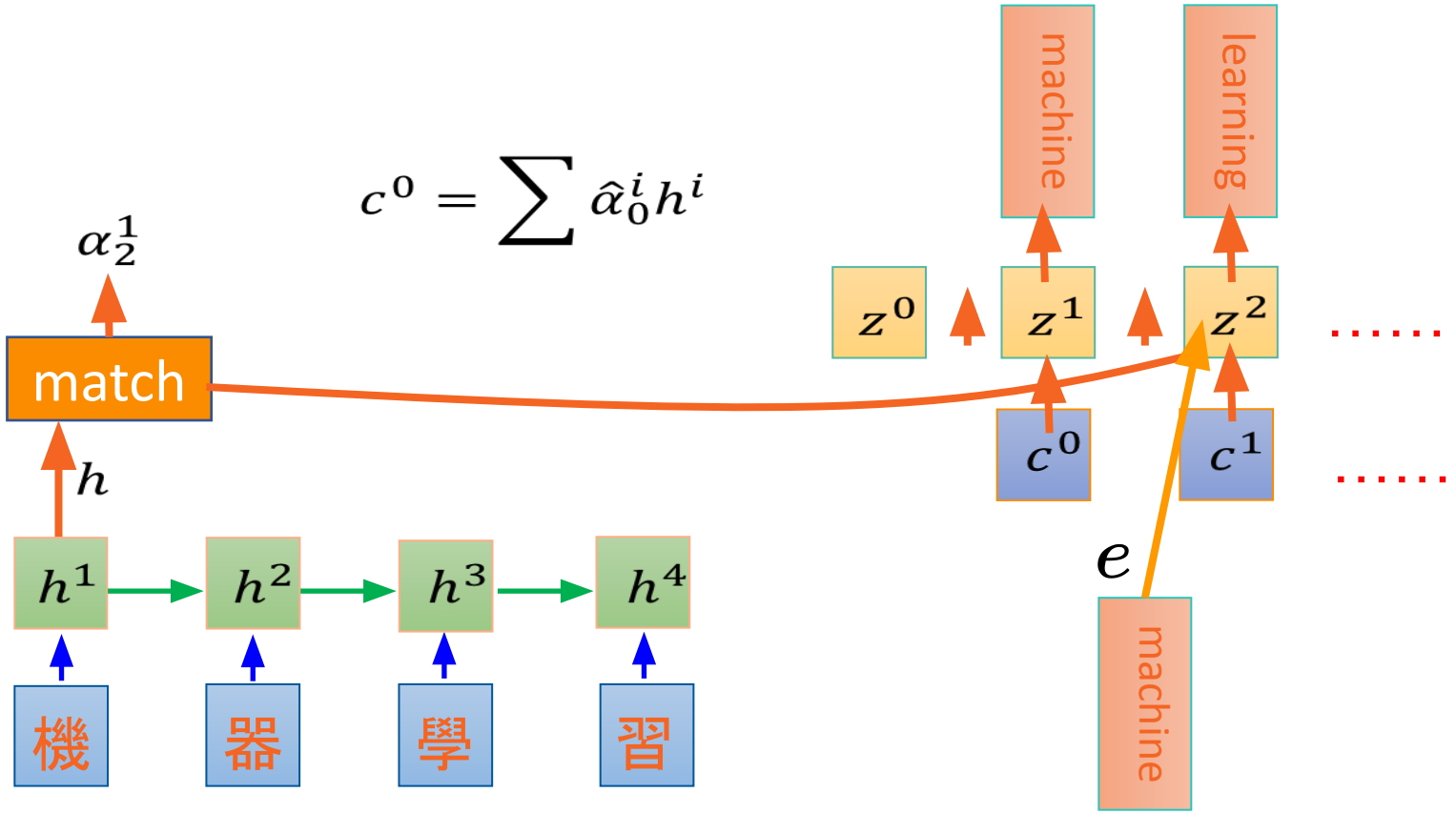
Machine Translation

- Attention-based model



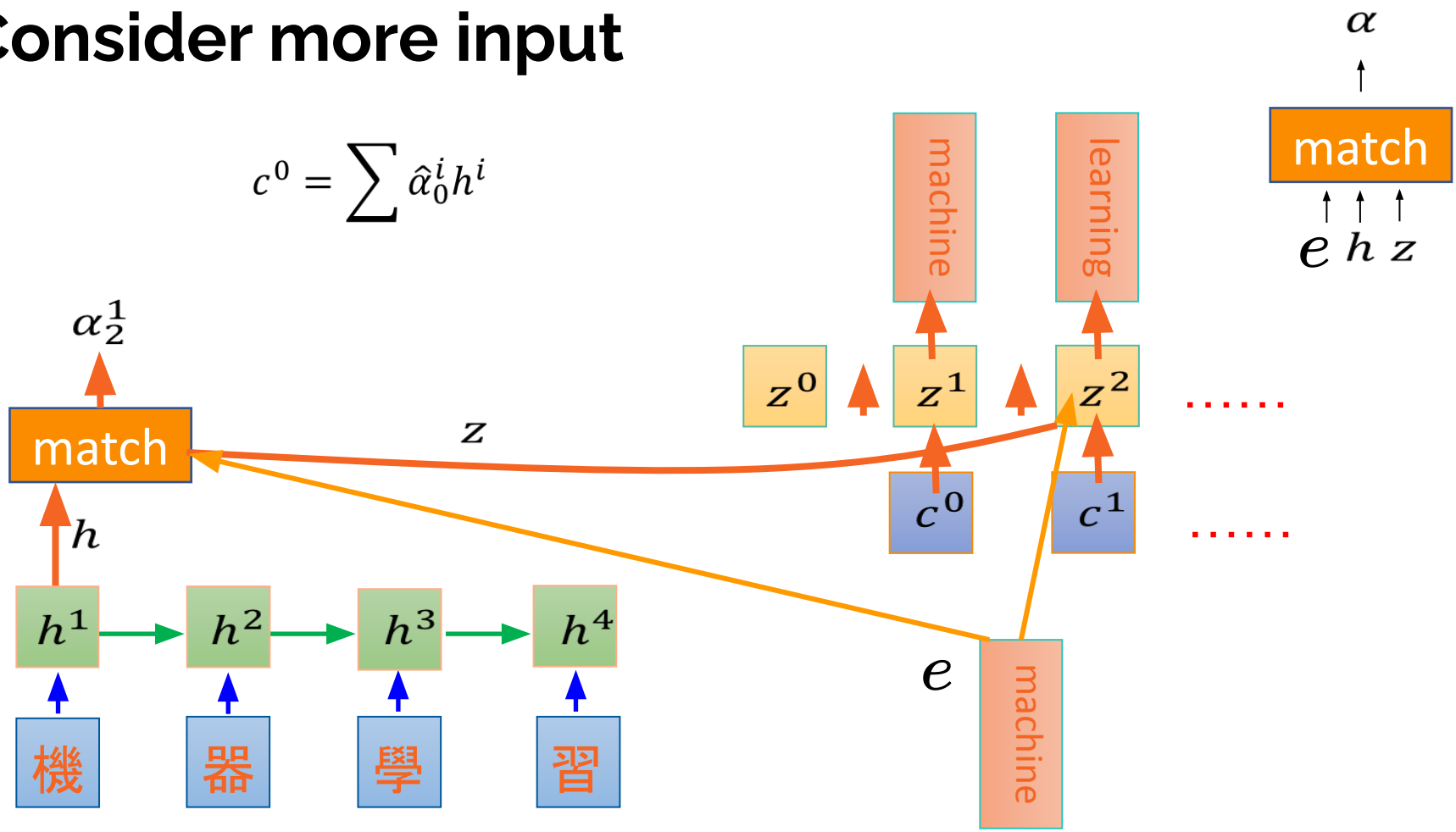
The same process repeat until generating . (period)

Consider more input



Consider more input

$$c^0 = \sum \hat{\alpha}_0^i h^i$$



Updated Attention

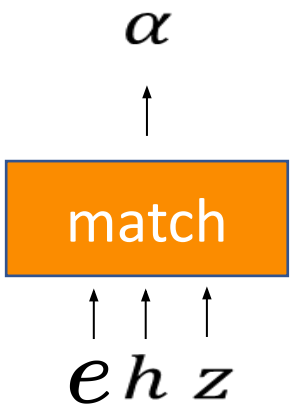
Concatenate (e,h)

$$attn = concate[e, h]$$

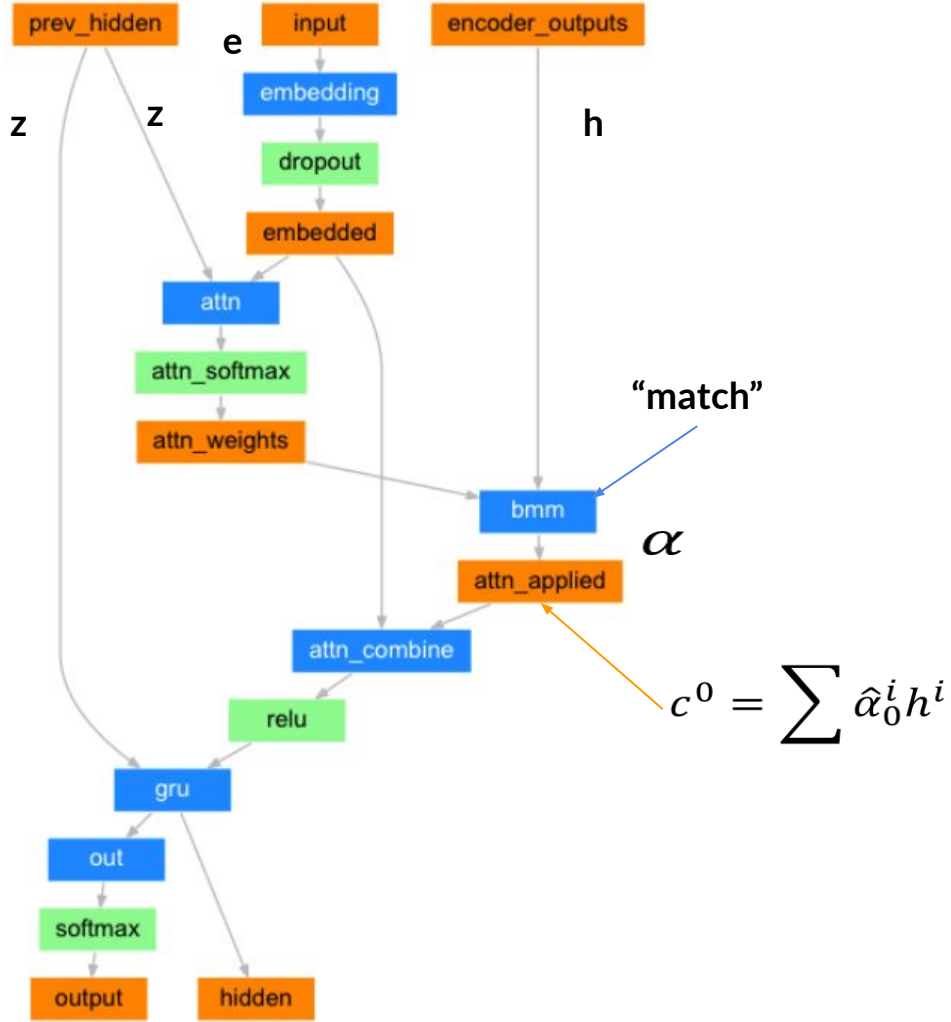
$$\alpha = cos(attn, z)$$

same

$$\alpha = matrix\_mul(attn, z)$$



Attention Decoder



Machine Translation Evaluation

Automatic Evaluation (BLUE, Meteor, ROUGE) & Human Evaluation

Case study

Automatic Method: ROUGE

Recall-**O**riented **U**nderstudy for **G**isting **E**valuation

Motivation: how many are overlapped?

System Summary (output of our model): the cat was found under the bed

Reference Summary (ground truth): the cat was under the bed

Compute the precision and recall using the overlap.

System Summary: the cat was found under the bed

Reference Summary: the cat was under the bed

Recall in the context of ROUGE means how much of the reference summary is the system summary recovering or capturing?

$$\frac{number\_of\_overlapping\_words}{total\_words\_in\_reference\_summary}$$

$$Recall = \frac{6}{6} = 1.0$$

Calculate Precision

System Summary: the cat was found under the bed

Reference Summary: the cat was under the bed

$$\frac{number\_of\_overlapping\_words}{total\_words\_in\_system\_summary}$$

$$Precision = \frac{6}{7} = 0.86$$

ROUGE: what to report

Precision

Recall

And **F1 Score** = 2*(Recall * Precision) / (Recall + Precision)

Another system summary

System Summary: the tiny little cat was found under the big funny bed

Reference Summary: the cat was under the bed

$$Recall = \frac{6}{6} = 1.0$$

$$Precision = \frac{6}{11} = 0.55$$

Rouge-N

ROUGE-1: unigram

ROUGE-2: bigram

ROUGE-3: trigram

…

Rouge-2

System Summary: the cat was found under the bed

Reference Summary: the cat was under the bed

System bigrams:

the cat, cat was, was found, found under, under the, the bed

Reference Summary:

the cat, cat was, was under, under the, the bed

Rouge-2 P,R, and F1

System bigrams:

the cat, cat was, was found, found under, under the, the bed

Reference Summary:

the cat, cat was, was under, under the, the bed

$$ROUGE2_{Precision} = \frac{4}{6} = 0.67 \qquad ROUGE2_{Recall} = \frac{4}{5} = 0.8$$

Python with Rouge

https://pypi.org/project/pyrouge/

Sample outputs:

---

1 ROUGE-1 Average_R: 0.78378 (95%-conf.int. 0.78378 - 0.78378)
1 ROUGE-1 Average_P: 0.80556 (95%-conf.int. 0.80556 - 0.80556)
1 ROUGE-1 Average_F: 0.79452 (95%-conf.int. 0.79452 - 0.79452)

---

1 ROUGE-2 Average_R: 0.69444 (95%-conf.int. 0.69444 - 0.69444)
1 ROUGE-2 Average_P: 0.71429 (95%-conf.int. 0.71429 - 0.71429)
1 ROUGE-2 Average_F: 0.70423 (95%-conf.int. 0.70423 - 0.70423)

Assignment: install pyrouge and have a try!

Is ROUGE always a smart way?

System Summary 1: the kitty was seen under the bed

System Summary 2: the cat was found under the bed

Reference Summary: the cat was seen under the bed

BLEU: Bilingual Evaluation Understudy

Chinese: 猫坐在垫子上

System1: the cat the cat on the mat
System2: the the the the the the the

Reference: the cat is on the mat

BLEU: Bilingual Evaluation Understudy

Unigram Precision (candidate) is 7/7 = 1.0

Unique Unigram	Count
the	3
cat	2
on	1
mat	1

$$\frac{number\_of\_overlapping\_words}{total\_words\_in\_system\_summary}$$

BLEU: Bilingual Evaluation Understudy

Bigram Precision (candidate) is 5/6 = 0.833

Unique Bigram	Count
the cat	2
cat the	0
cat on	1
on the	1
the mat	1

$$\frac{number\_of\_overlapping\_words}{total\_words\_in\_system\_summary}$$

Modified Precision

For each unique ngram, we count its maximum frequency in each of the reference sentences.

Candidate 7

5/7 = 0.714

System1: the cat the cat on the mat
System2: the the the the the the the

Reference: the cat is on the mat

Unique Unigram	Count	Clipped Count
the	3	2
cat	2	1
on	1	1
mat	1	1

Unique Bigram	Count	Clipped Count
the cat	2	1
cat the	0	0
cat on	1	1
on the	1	1
the mat	1	1

Modified Precision

Bigram: 4/6 = 0.667

BLEU

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

Pn: modified precision for n-gram

Wn: weight

BP: brevity penalty

BP

$$\mathrm{BP} = \begin{cases} 1 & \text{if } c > r \\ \exp\left(1 - \frac{r}{c}\right) & \text{if } c \leq r \end{cases}$$

The best match length is the closest reference sentence length to the candidate sentences.

3 references lengths 12, 14, and 17 words

Candidate translation 13 words

Ideally the best match length could be either 12 or 14, but we arbitrary choose the shorter one which is 12.

c is the number of unigrams (length) in all the candidate sentences, and r is the best match lengths for each candidate sentence in the corpus.

BLEU for system 1

$$p_1 = \frac{5}{7}$$

$$p_2 = \frac{4}{6}$$

$$p_3 = \frac{2}{5}$$

$$p_4 = \frac{1}{4}$$

$$w_1 = w_2 = w_3 = w_4 = \frac{1}{4}$$

c=7 and r=7

BP is 1

BLEU=0.467

NLTK to compute

```
import nltk
reference_1 = "the cat is on the mat".split()
reference_2 = "there is a cat on the mat".split()
candidate = "the cat the cat on the mat".split()

bleu =
nltk.translate.bleu_score.sentence_bleu(references=[reference_1,
reference_2], hypothesis=candidate, weights=(0.25,0.25,0.25,0.25))

print(bleu)
```

>> 0.4671379777282001

Human evaluation

Evaluation protocols

- **semantics**
- **consistency (factual alignment between the summary and the source)**
- **fluency (quality of individual sentences)**
- **coherence (collective quality of all sentences)**
- **...**

Human evaluation

Randomly choose 100-200 samples (a reasonable number)

2-5 human judges (only 1 is not enough)

You can decide how to set the points (0/1 or 0-5)

Report in a table.

Perplexity

Language Model Evaluation

Language Models

How do we know if a classification model A is better than a model B?

Classification => predict labels => accuracy, etc. Higher accuracy

How do we know if a language model A is better than a model B?

Language Models => predict the next word

How do we judge if the next word is correct? We look at the “test set.”

Use **Perplexity**

Perplexity: motivation

Given a sentence from the test set: [Unseen sentences]

S = "Python is a widely used general-purpose, high-level programming language. "

Model A: P(S) = 0.888

Model B: P(S) = 0.111

Model A is better than model B,
Because it assigns a higher probability to a piece of text in the test set.

Perplexity: definition

Perplexity is the probability of sentences in test set, normalized by the number of words N.

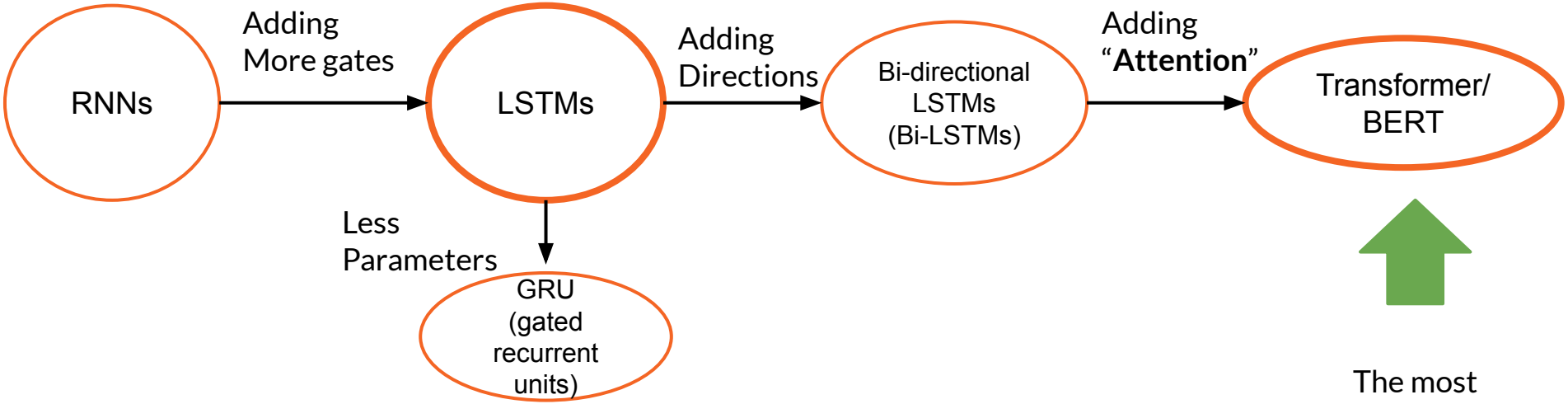
$$PP(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}$$

Lower perplexity means a better LM performance.

BERT

RNNs and Variations



We will focus on Transformer (this week).

Transformer

With Self-Attention

Self-attention

https://arxiv.org/abs/1706.03762

q: query (to match others)

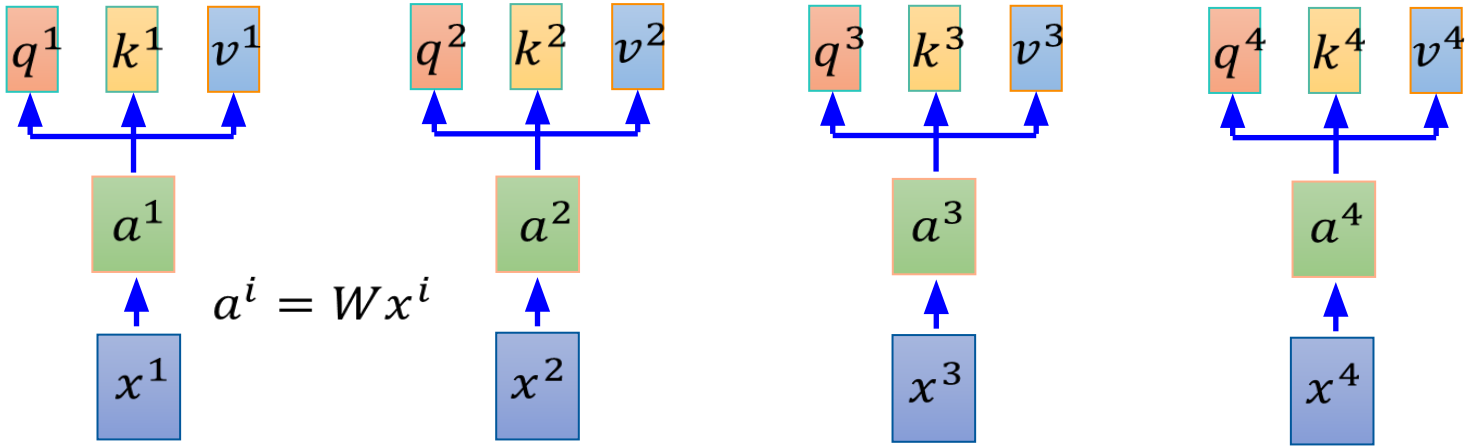
$$q^i = W^q a^i$$

k: key (to be matched)

$$k^i = W^k a^i$$

v: information to be extracted

$$v^i = W^v a^i$$



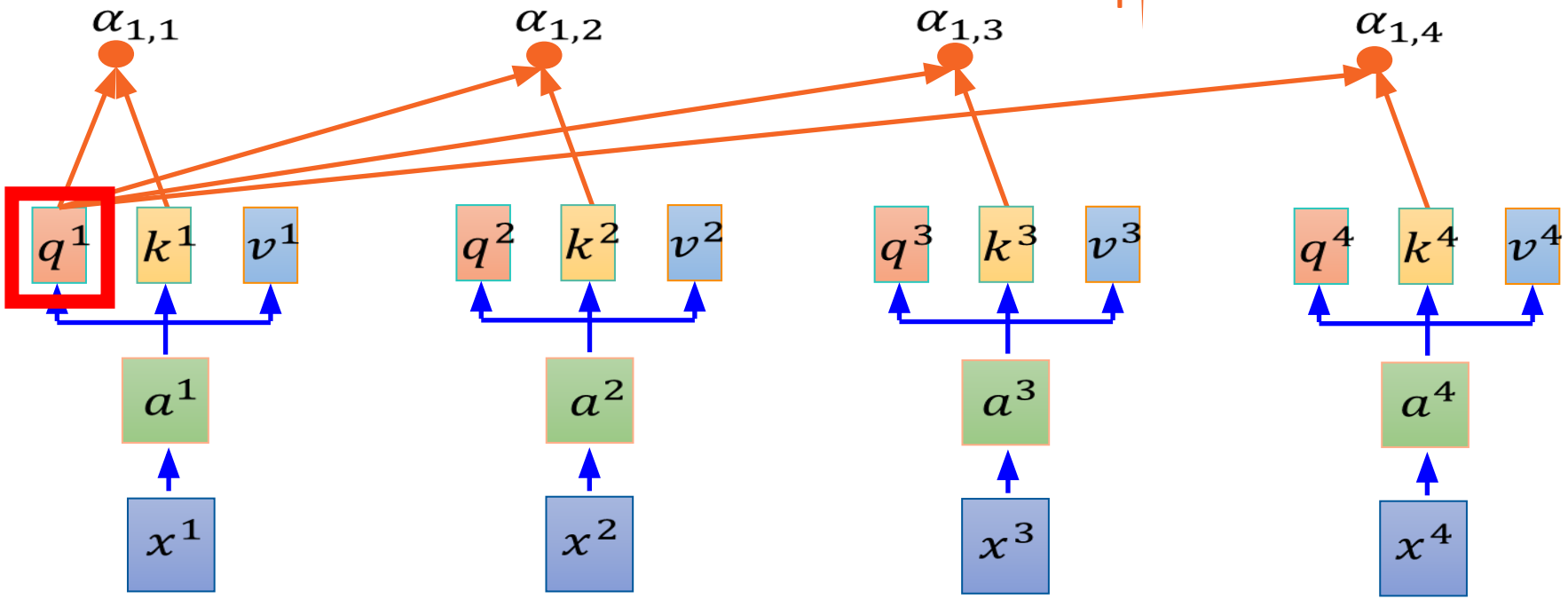
Self-attention

Match each query q with key k ->attention

d is the dim of q and k

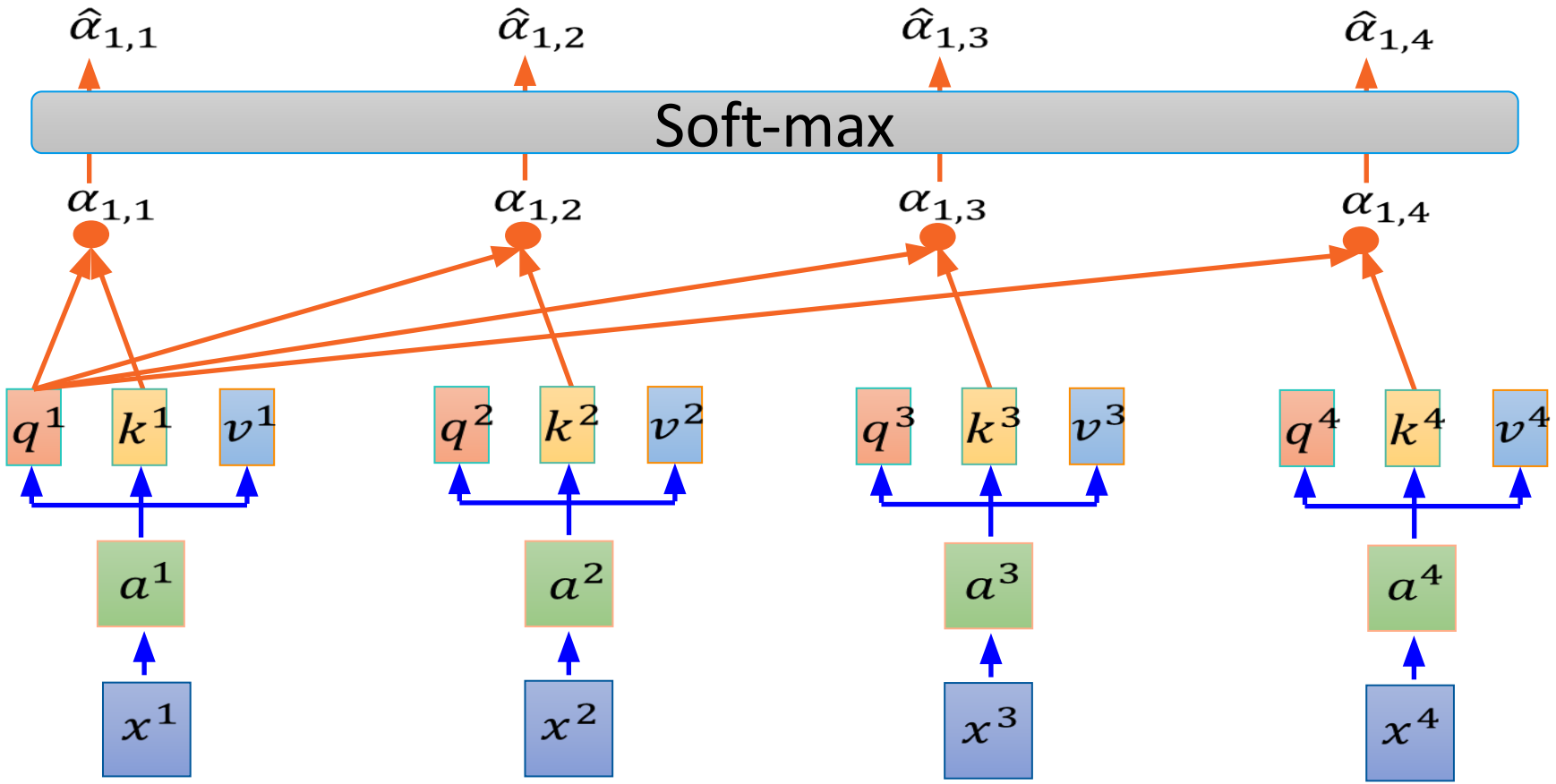
Scaled Dot-Product Attention: $\alpha_{1,i} = q^1 \cdot k^i / \sqrt{d}$

dot product



Self-attention

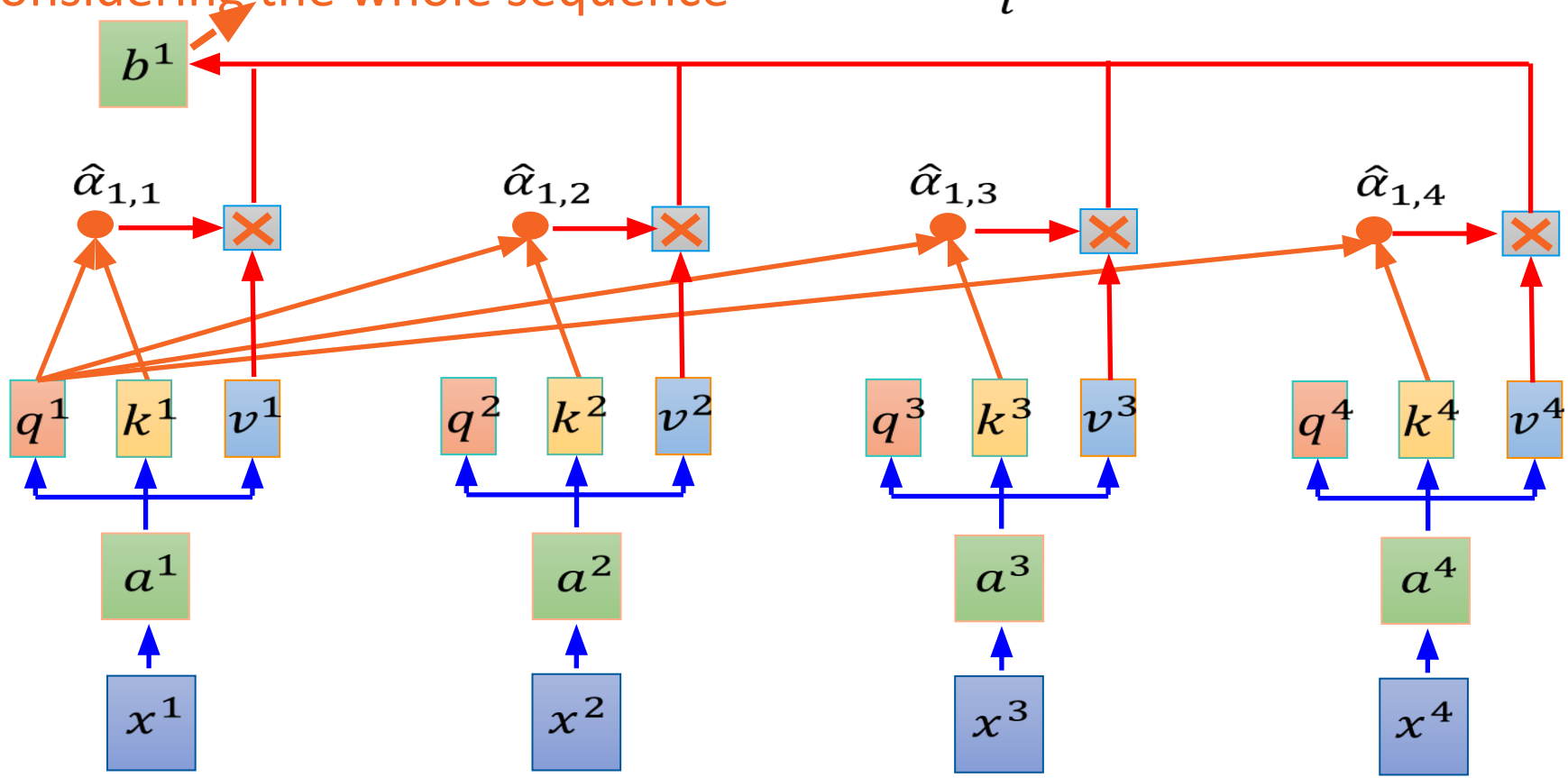
$$\hat{\alpha}_{1,i} = exp(\alpha_{1,i}) / \sum_j exp(\alpha_{1,j})$$



Self-attention

Considering the whole sequence

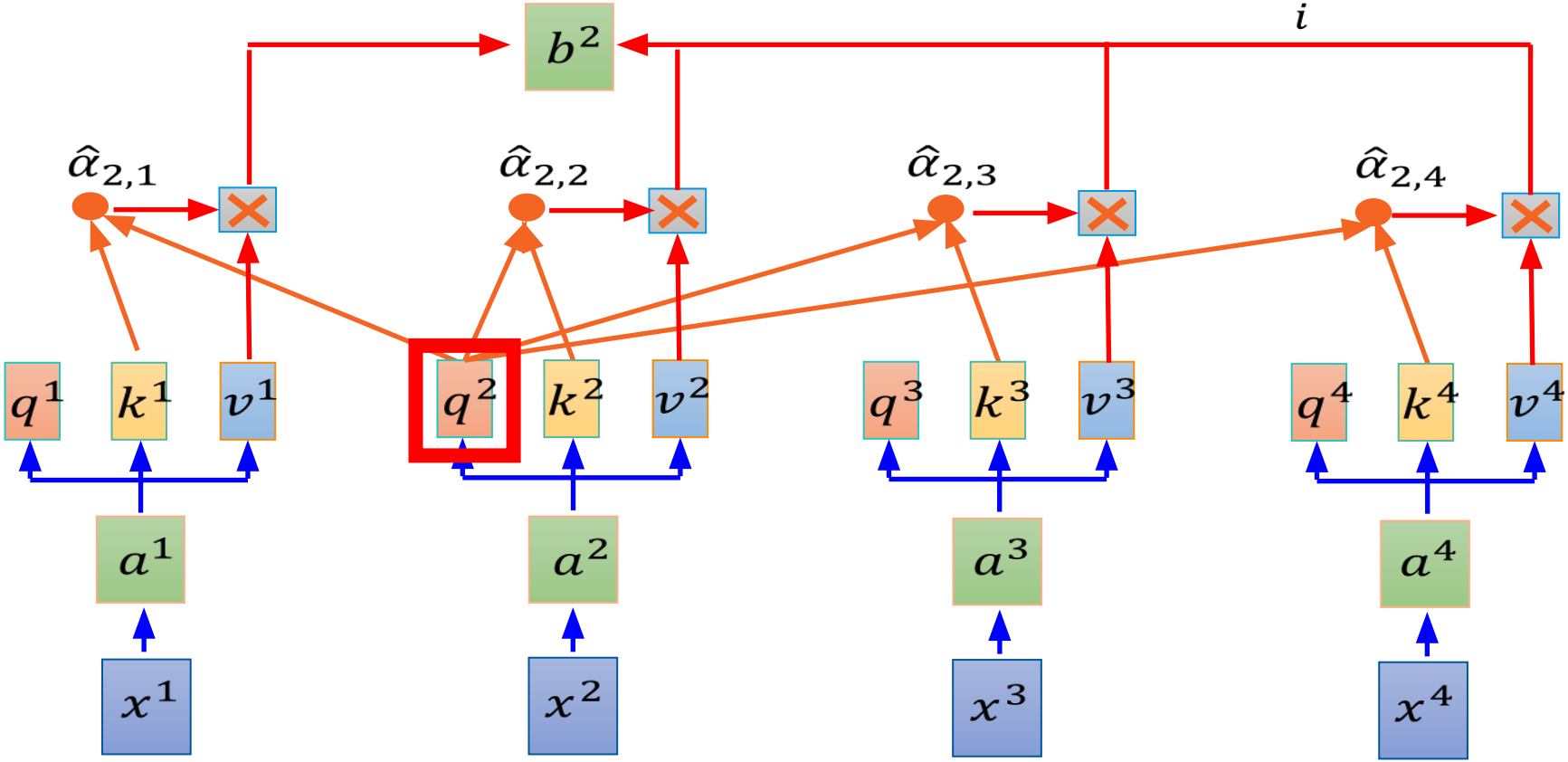
$$b^1 = \sum_i \hat{\alpha}_{1,i} v^i$$



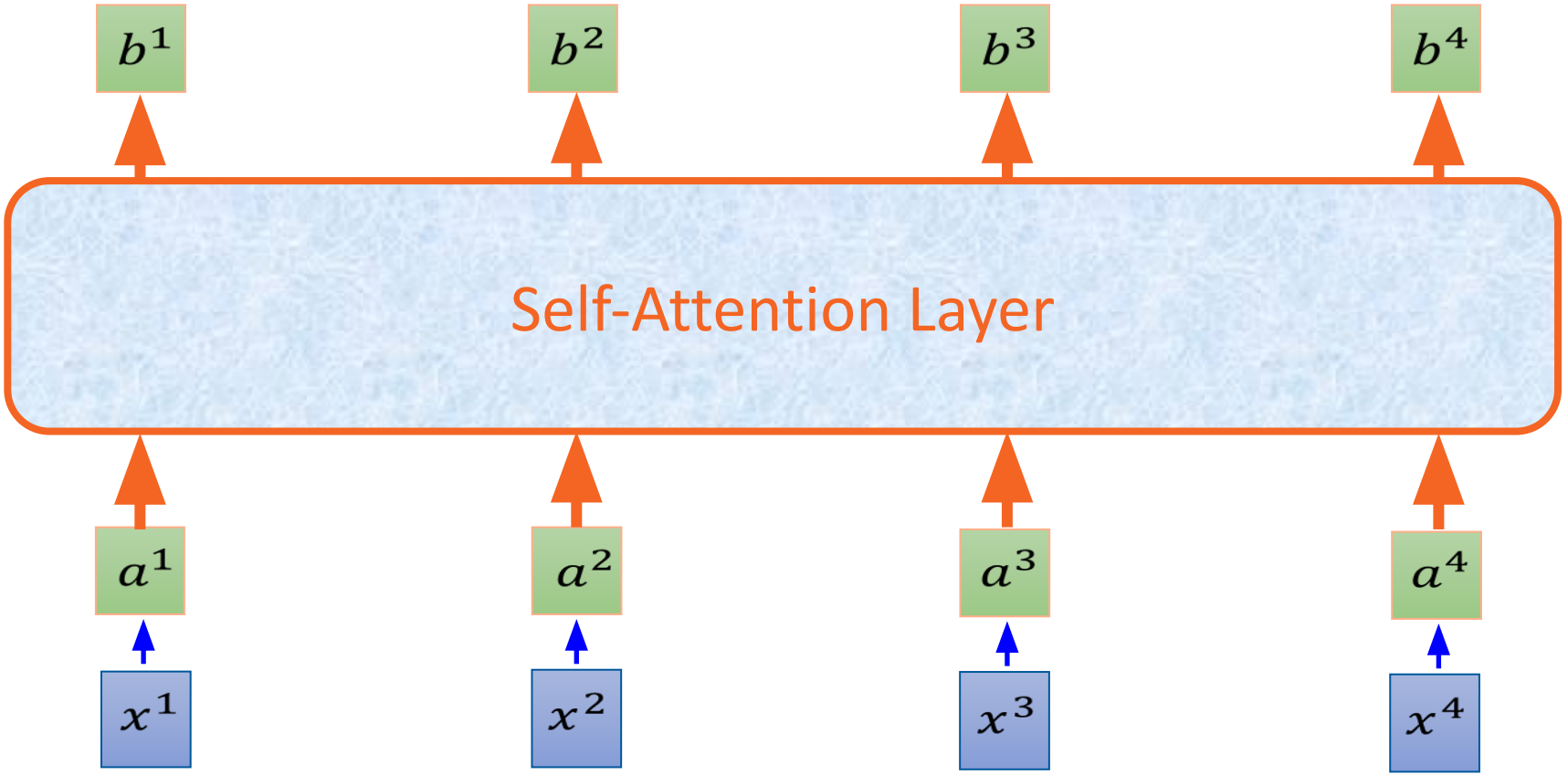
Self-attention

Match each query q with key k ->attention

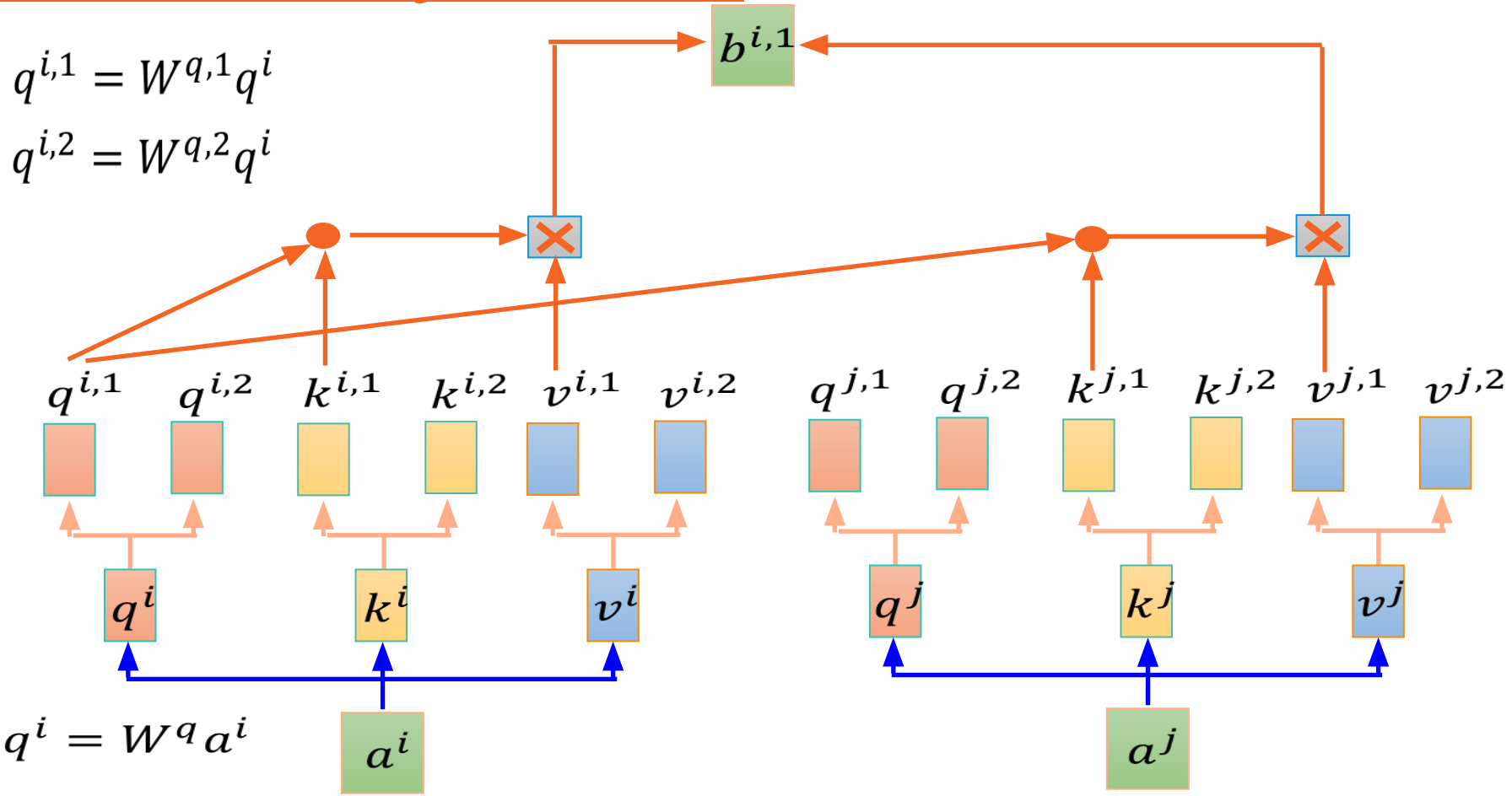
$$b^2 = \sum_i \hat{\alpha}_{2,i} v^i$$



Self-attention



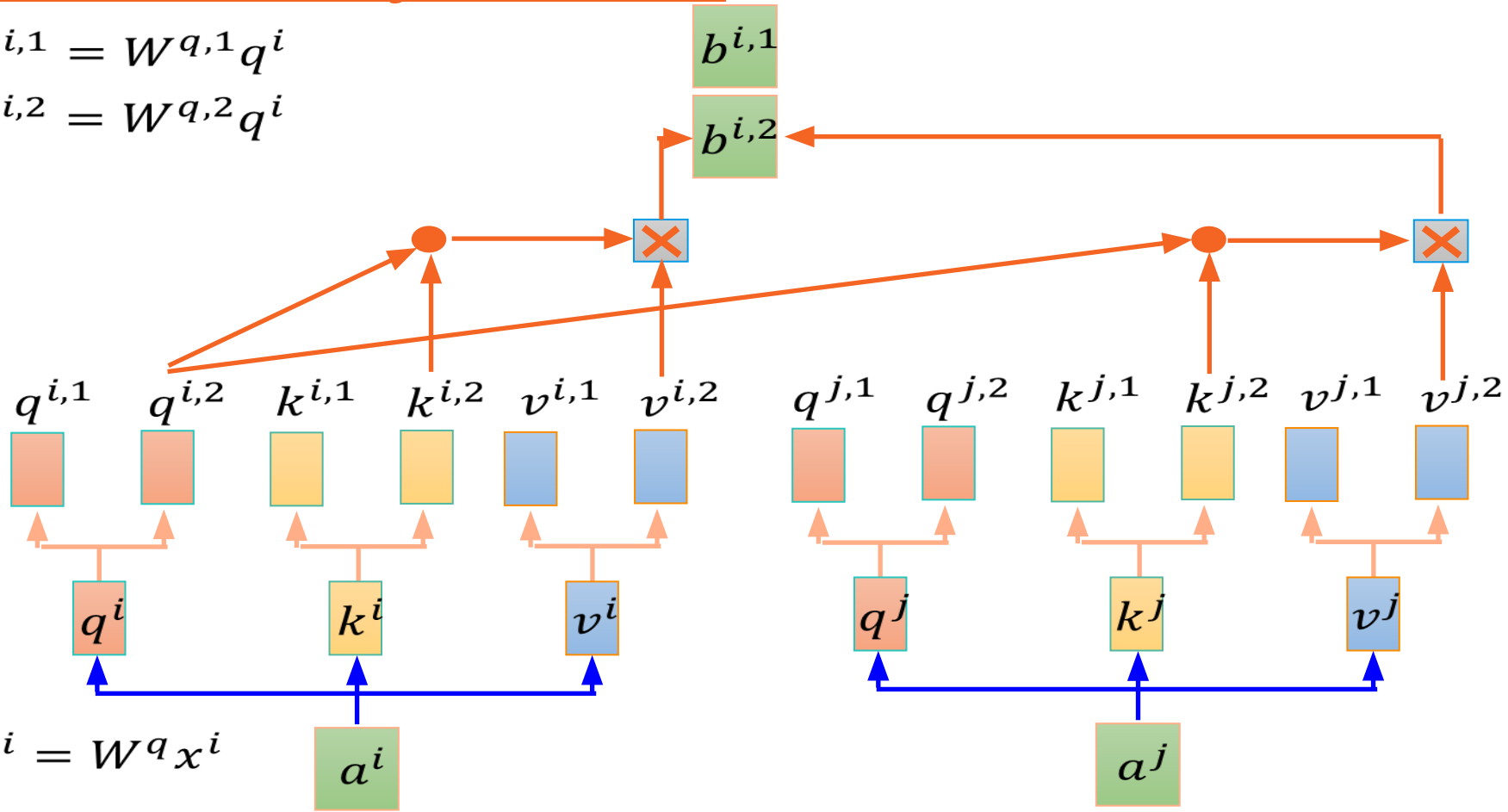
Multi-head Self-attention (2 heads as example)



Multi-head Self-attention (2 heads as example)

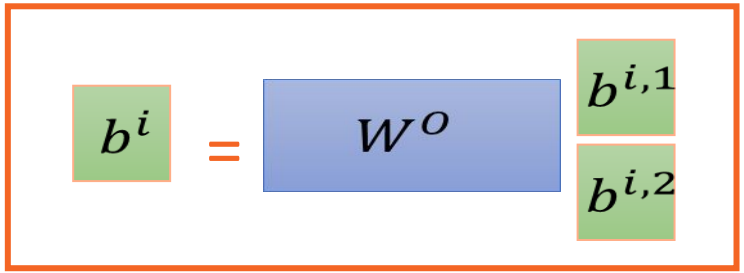
$q^{i,1} = W^{q,1} q^i$

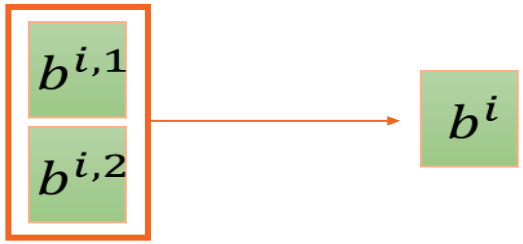
$q^{i,2} = W^{q,2} q^i$

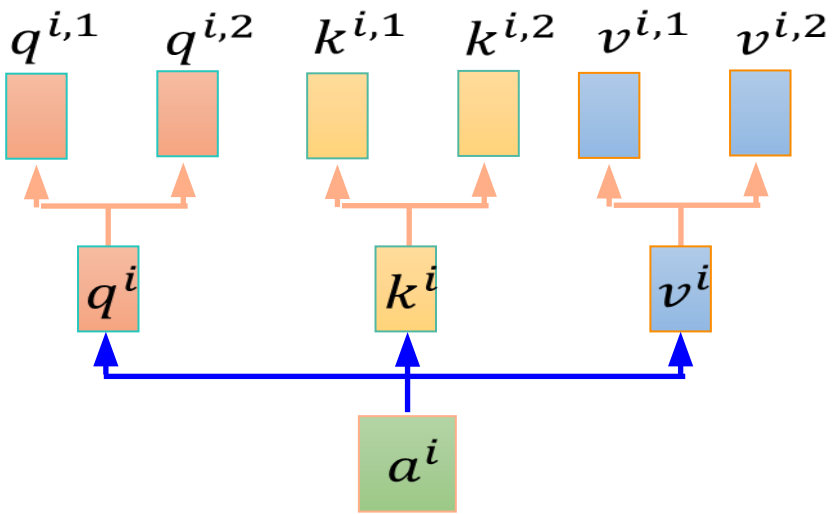


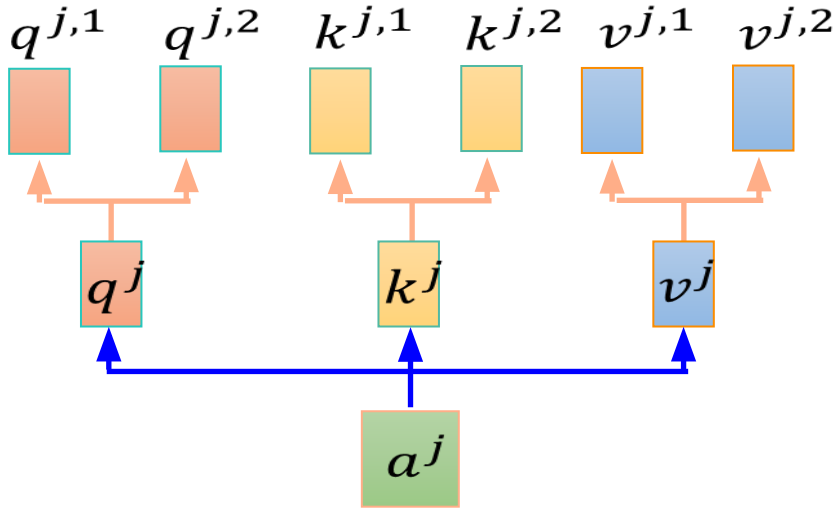
$q^i = W^q x^i$

Multi-head Self-attention (2 heads as example)

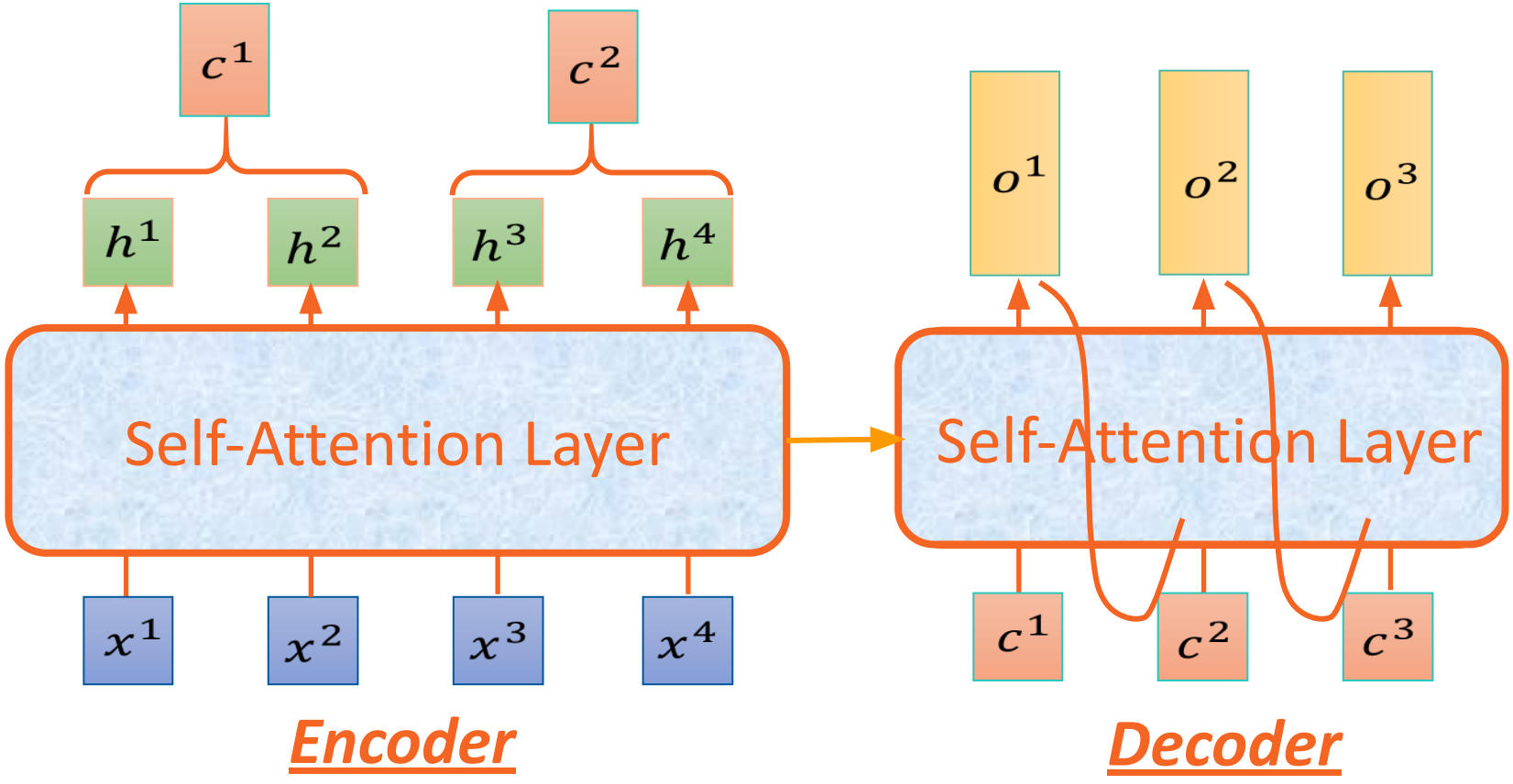






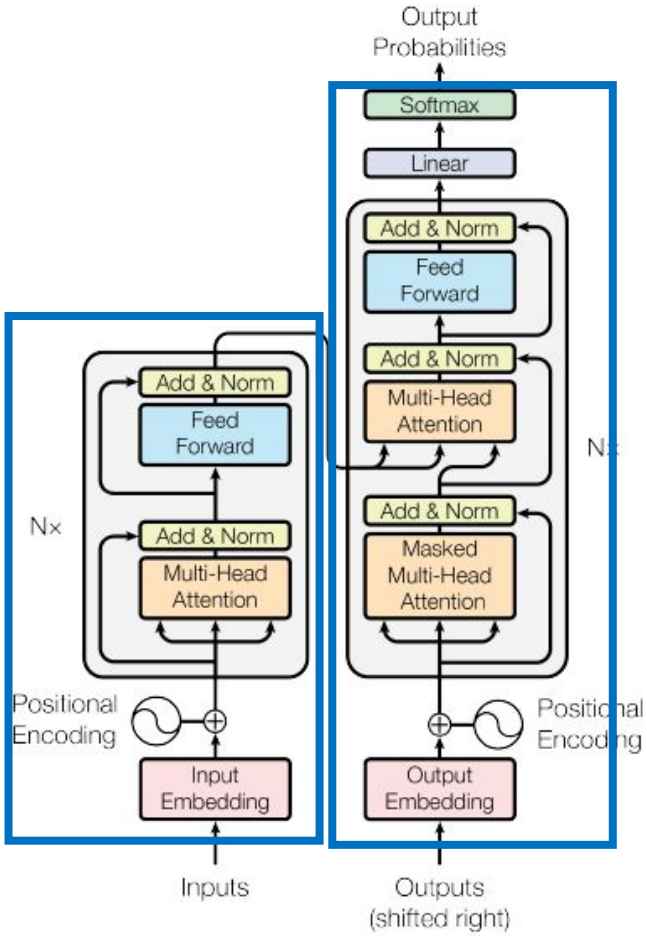


Seq2seq with Attention



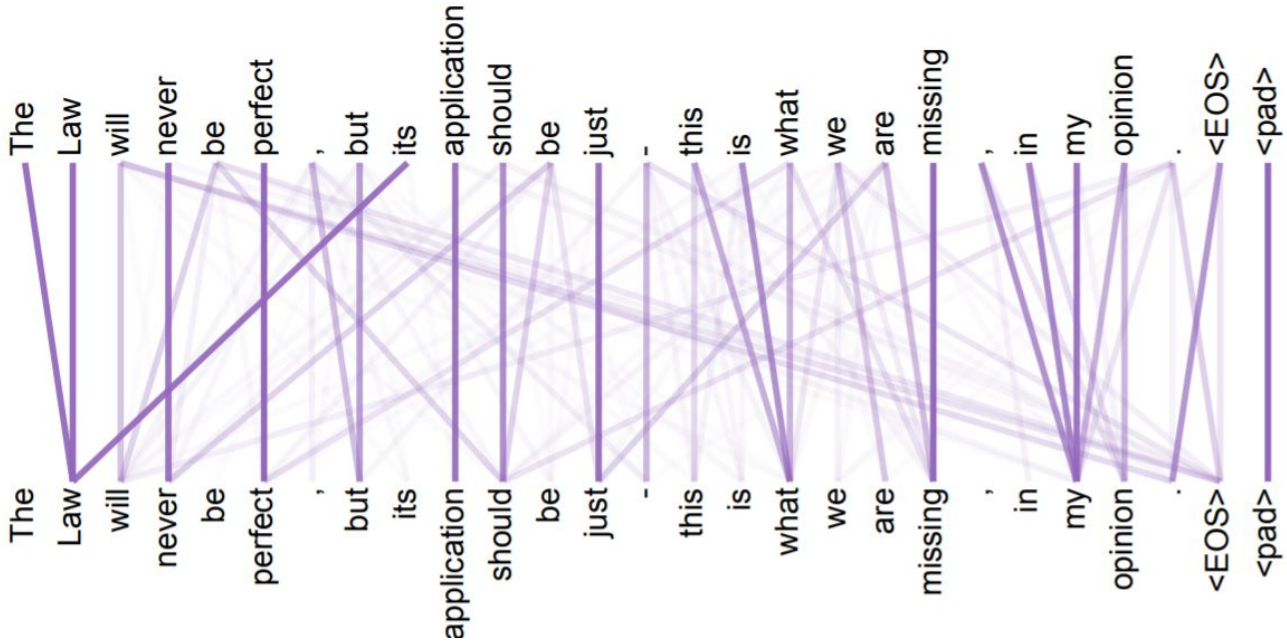
Transformer

Encoder



Decoder

Attention Visualization



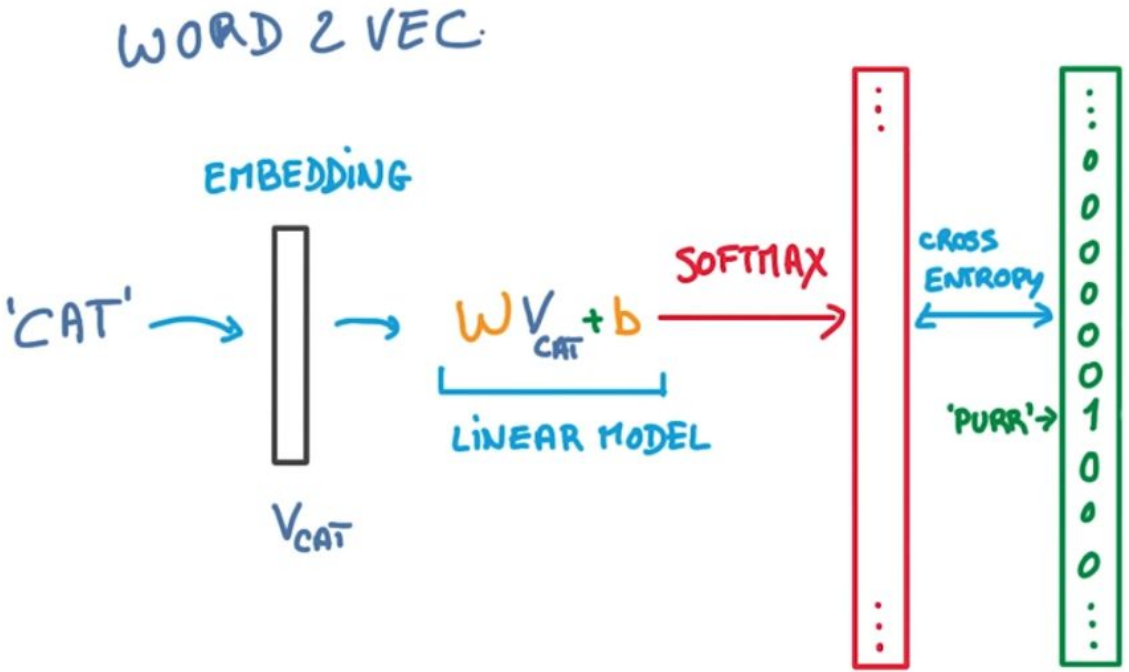
https://arxiv.org/abs/1706.03762

Apply Transformers

Motivation: word embeddings

Word embedding matrix
→ search with ID.

Context-free models.



Motivation: look at content!

“bank”

in

“River bank”

“bank deposit”

Same word vector for **bank**.

Context-free models!

In Word2vec:

a single word embedding representation for each word.

Contextual models

“bank”

in

“River bank”

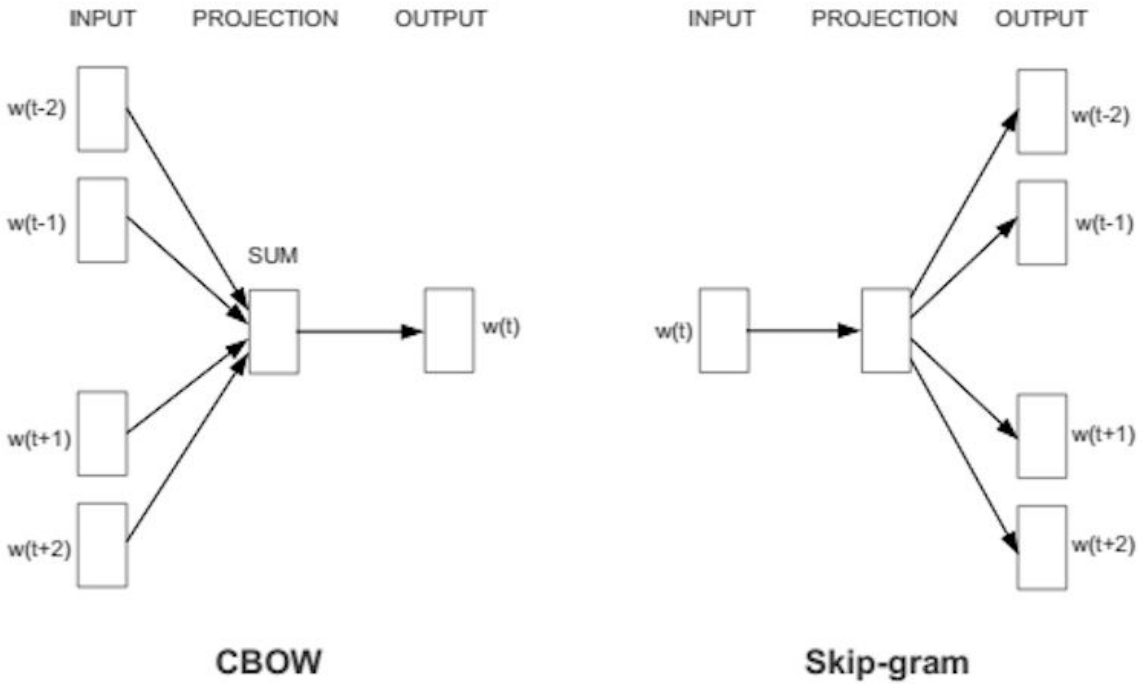
“bank deposit”

Different word vector for **bank**.

Contextual models instead generate a representation of each word that is based on the other words in the sentence.

=> “bank” has more than one word vectors, depending on its content!

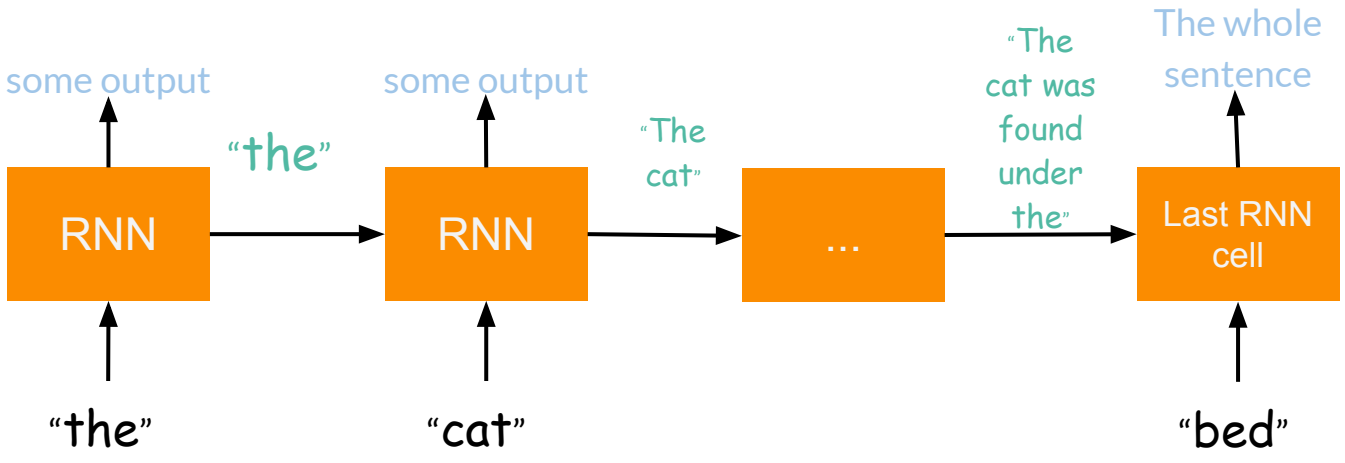
Word2vec vs LSTMs



RNNs

*Input + **Previous Hidden** → Hidden → Output*

"the cat was found under the bed"

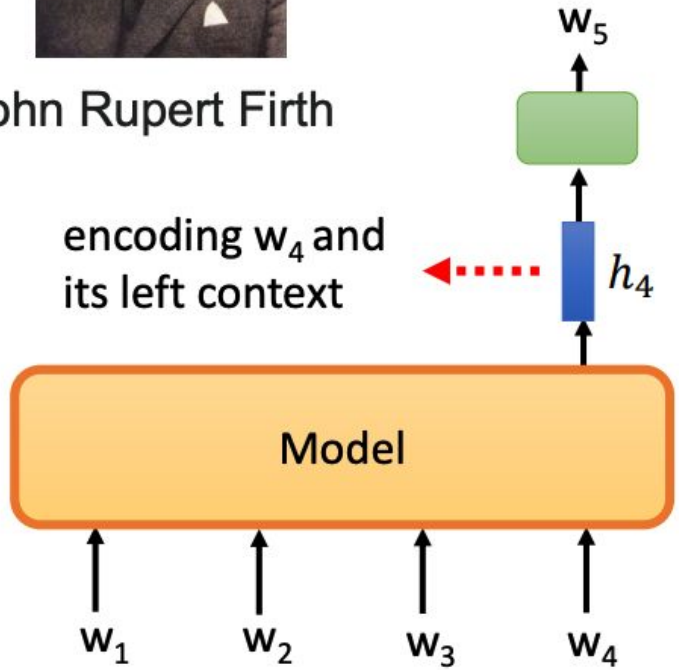


One Direction

" You shall know a word by the company it keeps! "
-- (1957)

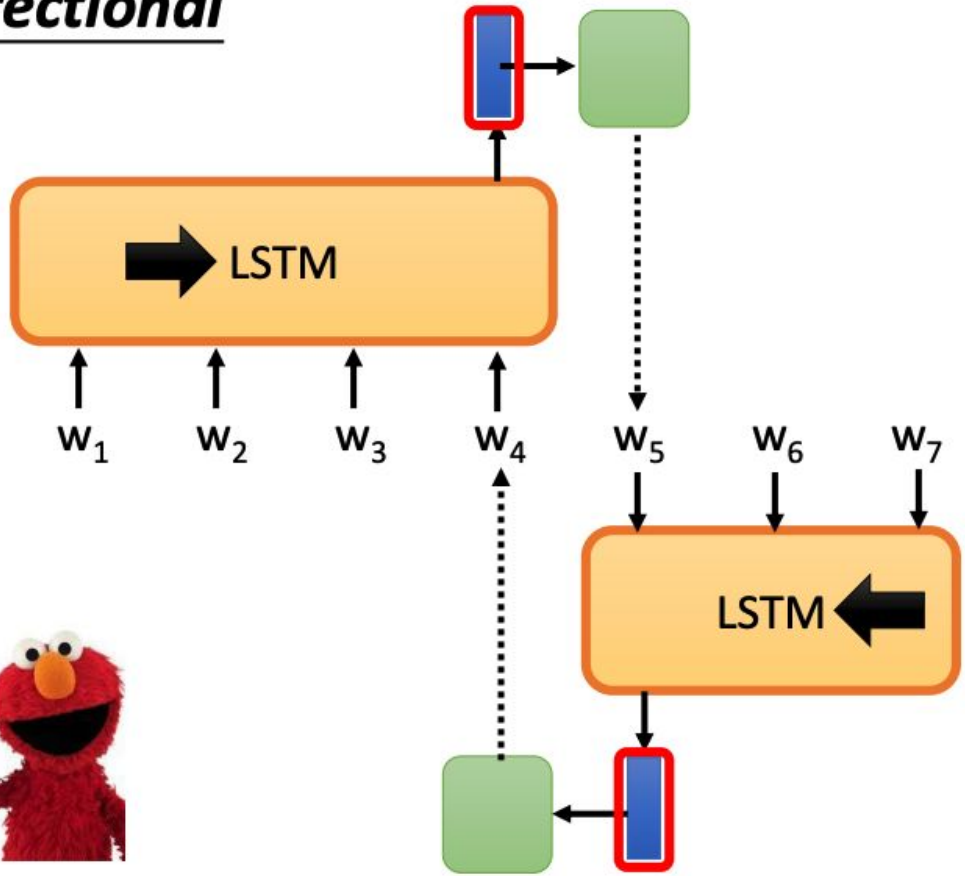
You shall know a word by the company it keeps

John Rupert Firth



How about the right context!?

Predict Next Token - Bidirectional



ELMO

Combine the content from the left and from right

~8 layers LSTM

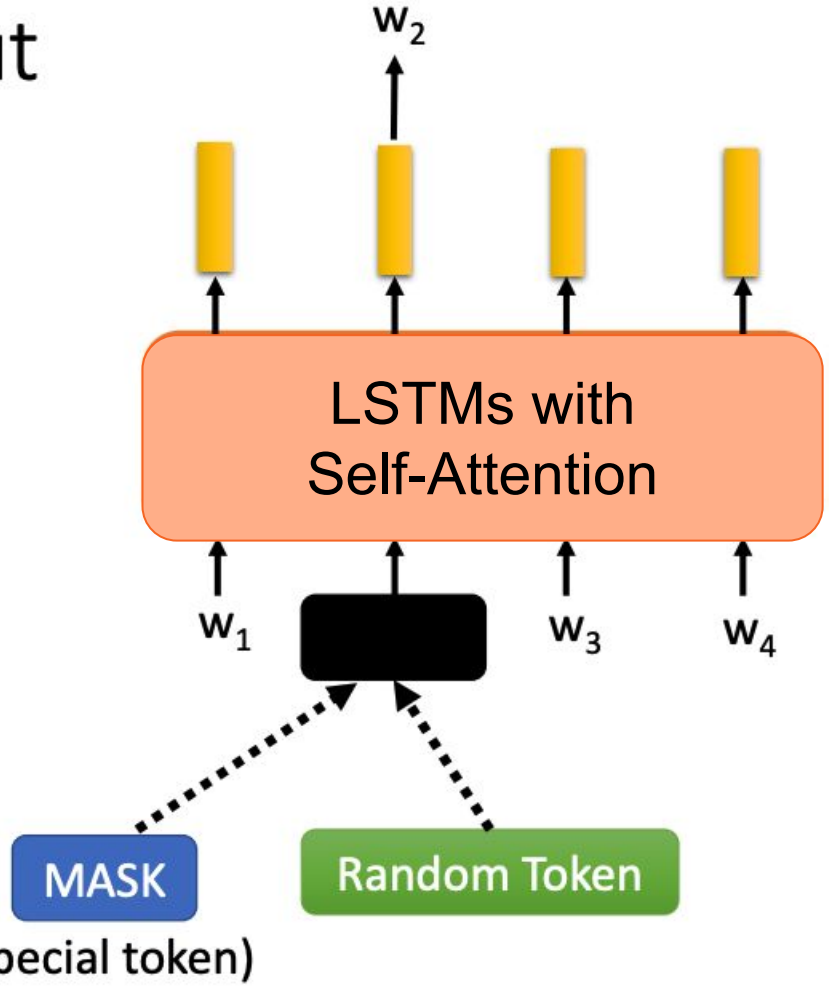
Bi-directional LSTM

Masking Input

BERT

[Devlin, et al., NAACL'19]

Transformer
(no limitation on self-attention)



12-24 layers LSTM

Bi-directional LSTM

Attention inside..

BERT

BERT: on New York Times

Finally, a Machine That Can Finish Your Sentence

Completing someone else's thought is not an easy trick for A.I. But new systems are starting to crack the code of natural language.

https://www.nytimes.com/2018/11/18/technology/artificial-intelligence-language.html

Pre-trained Models

Word2vec:

Word embeddings

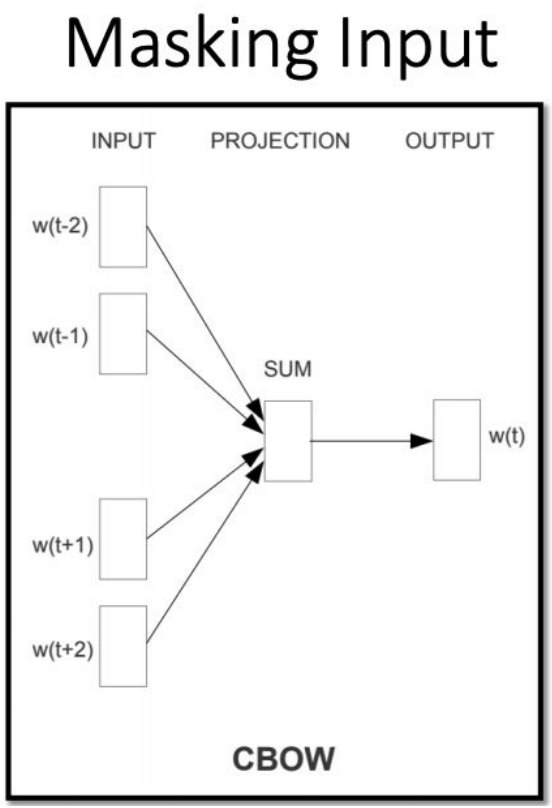
BERT:

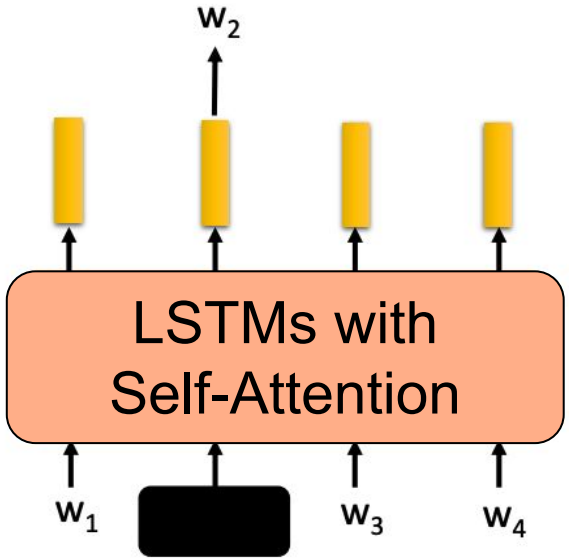
Word embeddings and the model

Illustration of BERT

https://github.com/cedrickchee/awesome-bert-nlp

Training BERT: task 1

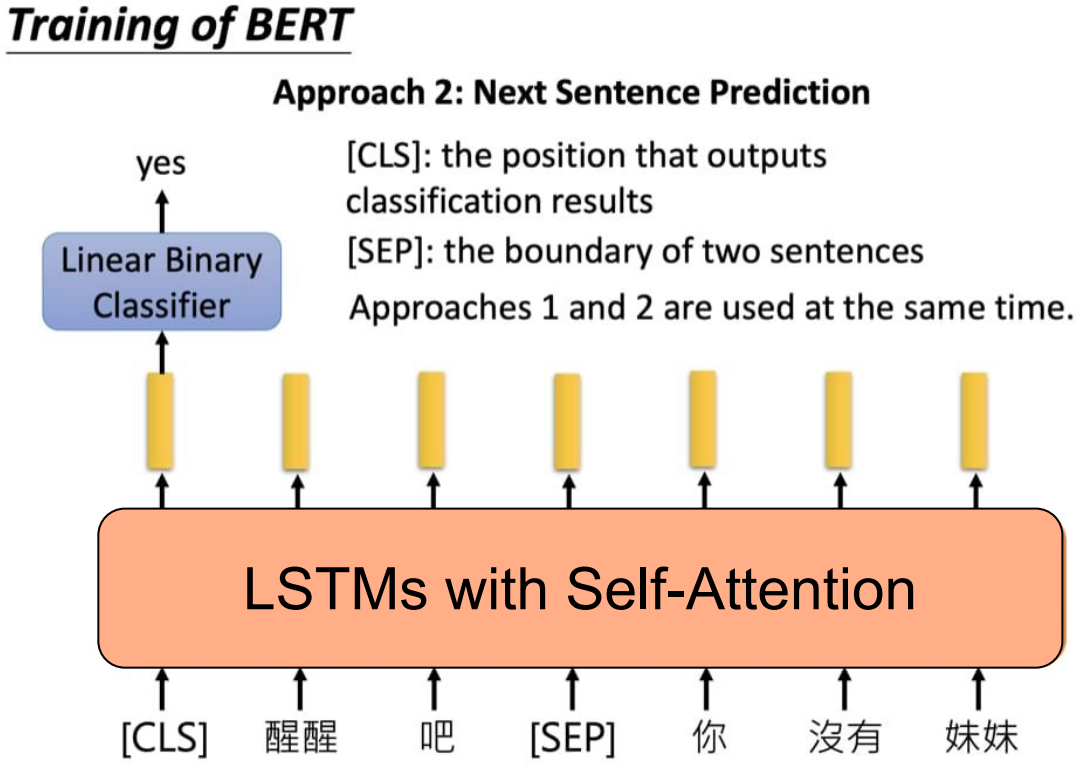




Using context to predict the missing token

Use Wiki Articles
Free-texts..
Millions of articles.
~200 languages

Training BERT: task 2



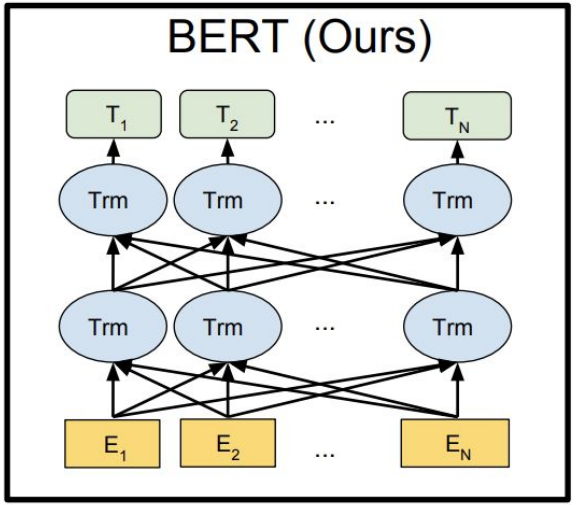
"How are you"
"I am fine"

→ Yes

"How are you?"
"My book is here."

→ No

BERT Features...



Bi-directional Transformer seq2seq model

Much stronger encoder and decoder.

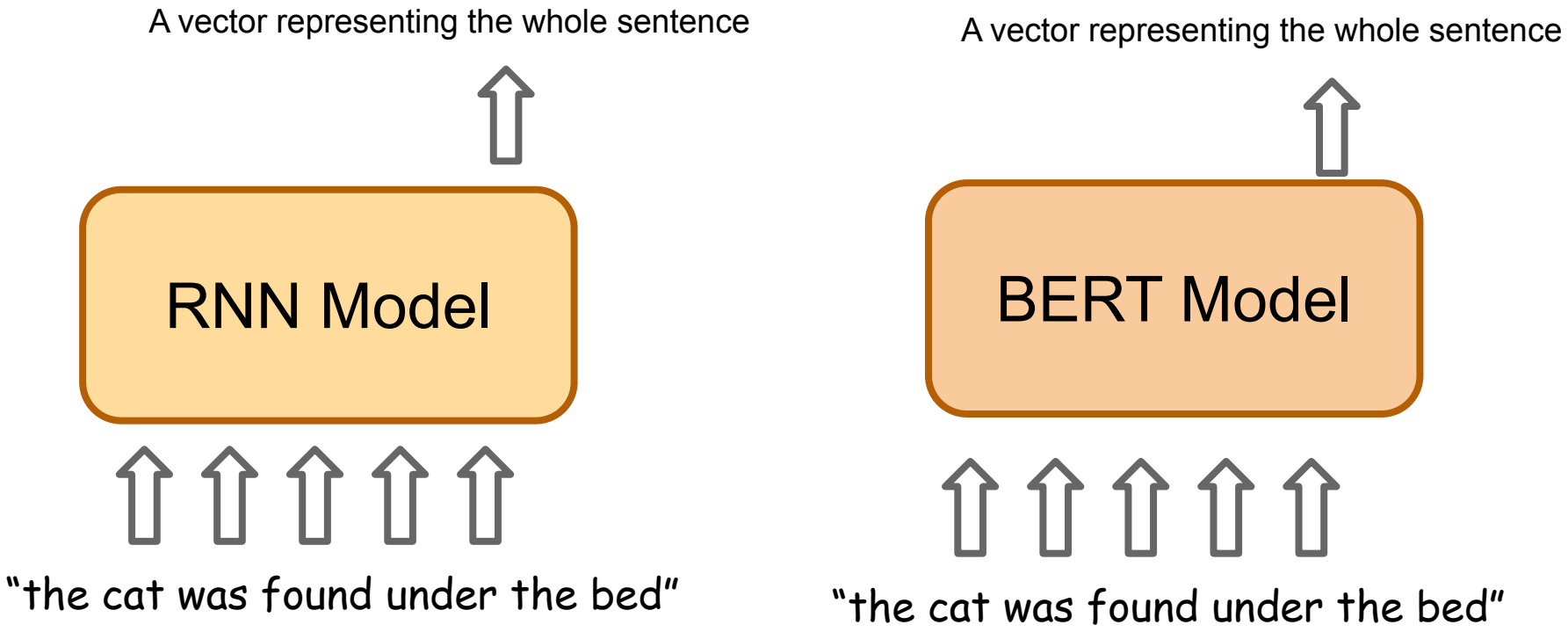
Task 1: Masked Language Modeling

Replace words randomly: “my dog is cute” -> “my dog is [MASK]”

Task 2: Next Sentence Prediction (QA, NLI)

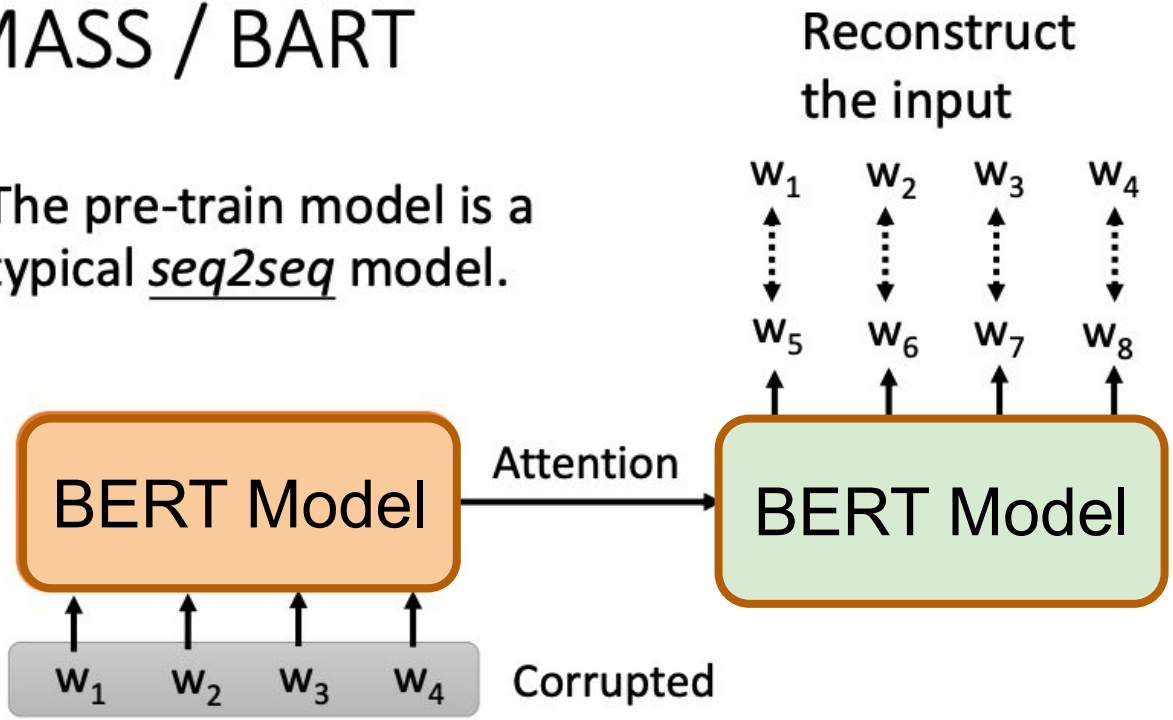
Binary classification: (sentence1, sentence2) -> IsNext or NotNext ?

BERT as an **encoder**



MASS / BART

- The pre-train model is a typical *seq2seq* model.



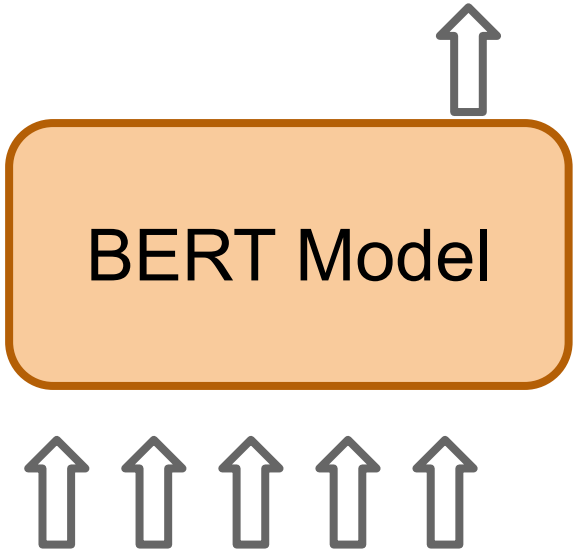
MAsked Sequence to Sequence pre-training (MASS) [Song, et al., ICML'19]

Bidirectional and Auto-Regressive Transformers (BART) [Lewis, et al., arXiv'19]

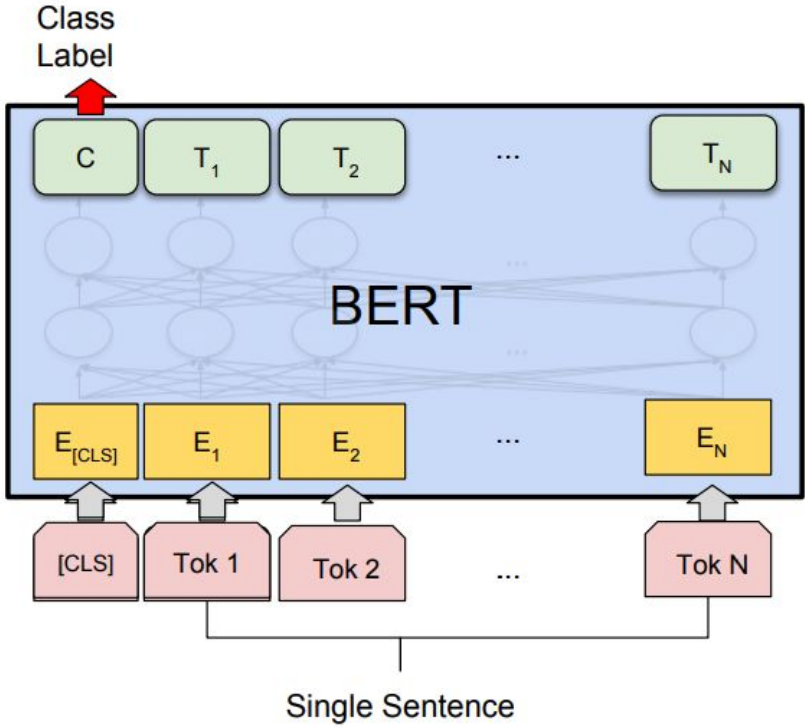
BERT for Classification

BERT as an **encoder**

A vector representing the whole sentence



"the cat was found under the bed"



Special Token [CLS]

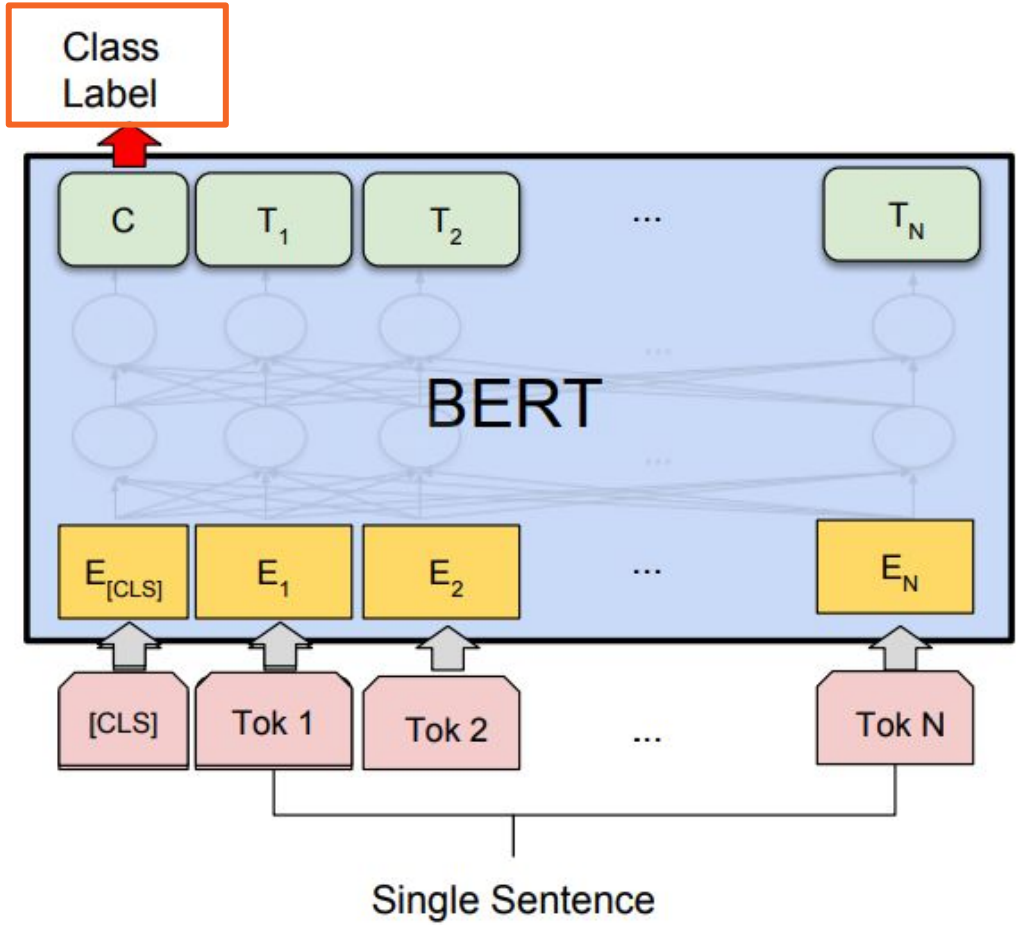
Add a special token [CLS] to the beginning of each sentence.

So the input of BERT becomes:

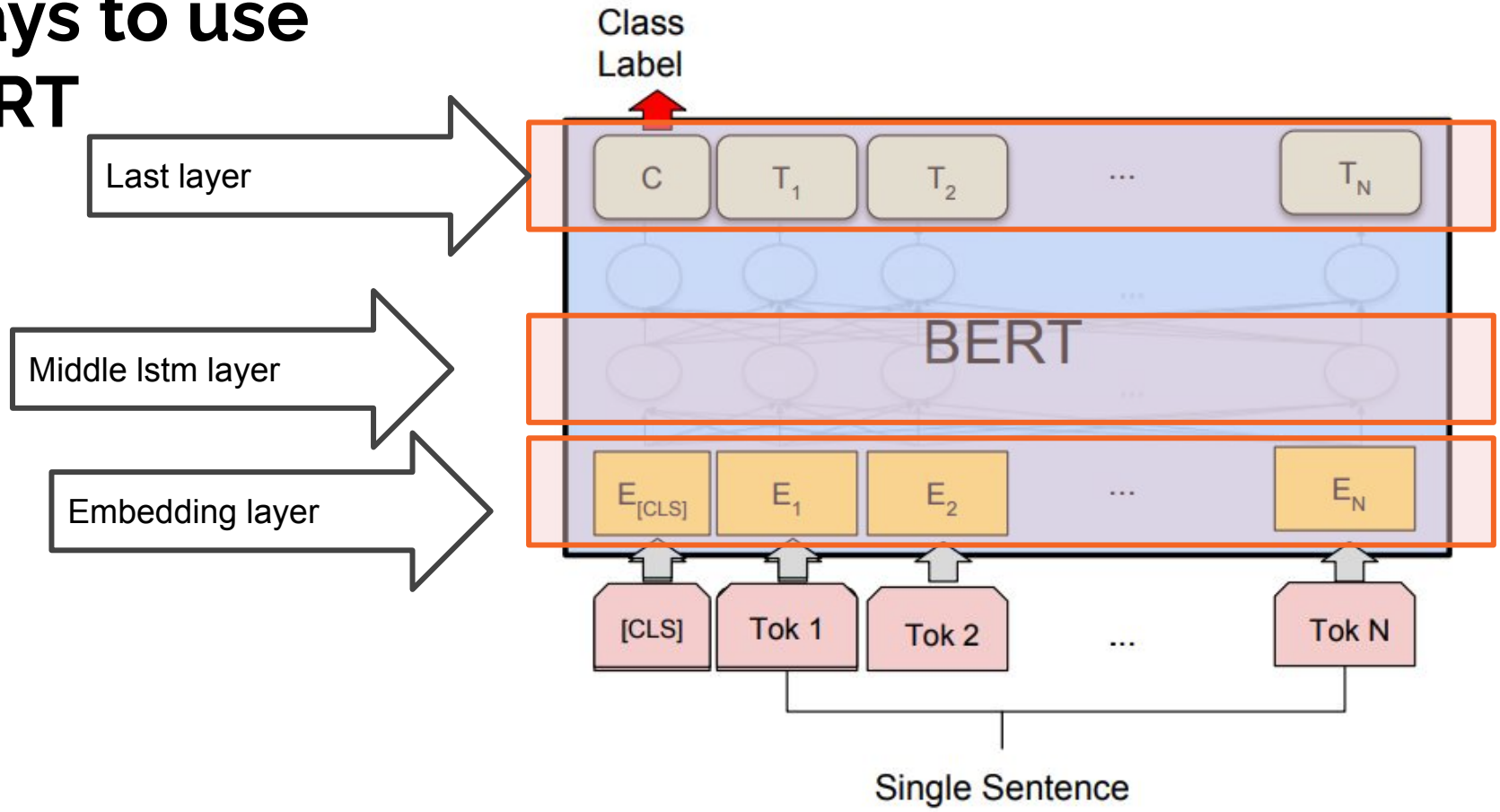
We then take [CLS] Embedding

as

“sentence information”



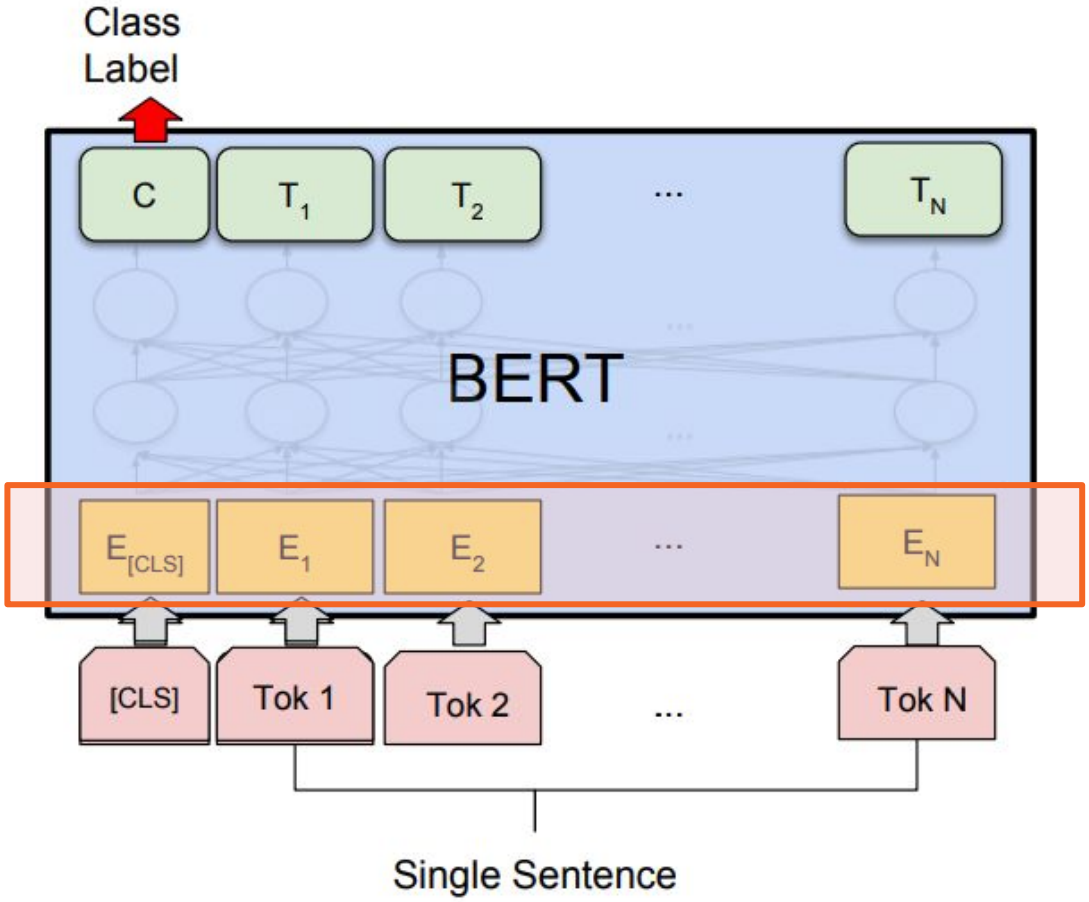
Ways to use BERT



Code to read

BERT for text classification [link]

[Transformers] library



BERT for Machine Translation

Facebook Fairseq

Fairseq PyTorch is an open-source machine learning library based on a sequence modeling toolkit.

It allows the researchers to train custom models for fairseq summarization transformer, language, translation, and other generation tasks.

Pre-trained Seq2seq model.

Applying to our data (EN-DE)

[Demo Link]

Installation:

https://github.com/pytorch/fairseq

1. Install fairseq locally, run the demo.
2. Run fairseq on our data, generate some sample outputs. Save them in a txt file.

```
import torch

# List available models
torch.hub.list('pytorch/fairseq')  # [..., 'transformer.wmt16.en-de', ... ]

# Load a transformer trained on WMT'16 En-De
# Note: WMT'19 models use fastBPE instead of subword_nmt, see instructions below
en2de = torch.hub.load('pytorch/fairseq', 'transformer.wmt16.en-de',
                       tokenizer='moses', bpe='subword_nmt')
en2de.eval()  # disable dropout

# The underlying model is available under the *models* attribute
assert isinstance(en2de.models[0], fairseq.models.transformer.TransformerModel)

# Move model to GPU for faster translation
en2de.cuda()

# Translate a sentence
en2de.translate('Hello world!')
# 'Hallo Welt!'

# Batched translation
en2de.translate(['Hello world!', 'The cat sat on the mat.'])
# ['Hallo Welt!', 'Die Katze saß auf der Matte.']
```

Text Summarization

Extractive summarization & Abstractive summarization

- Extractive summarization:
  - create the summary from phrases or sentences in the source document(s)
- Abstractive summarization:
  - express the ideas in the source documents using (at least in part) different words

Extractive Summarization: Directly selecting existing sentences from input document instead of rewriting them

Abstractive Summarization: novel words maybe used, re-phrasing contents.

(a) Extractive Summarization

Source Text: Peter and Elizabeth took a taxi to attend the night party in the city.

While in the party, Elizabeth collapsed and was rushed to the hospital.

Summary: Peter and Elizabeth attend party city. Elizabeth rushed hospital.

(b) Abstractive Summarization

Source Text: Peter and Elizabeth took a taxi to attend the night party in the city.

While in the party, Elizabeth collapsed and was rushed to the hospital.

Summary: Elizabeth was hospitalized after attending a party with Peter.

RNNs?
Generations?

History of Summarization

Since 1950s:

- Concept Weight (Luhn, 1958), Centroid (Radev et al., 2004), LexRank (Erkan and Radev, 2004), TextRank (Mihalcea and Tarau, 2004), Sparse Coding (He et al., 2012; Li et al., 2015)
- Feature+Regression (Min et al., 2012; Wang et al., 2013)

Most of the summarization methods are extractive.

Abstractive summarization is full of challenges.

- Some indirect methods employ sentence fusing (Barzilay and McKeown, 2005) or phrase merging (Bing et al., 2015).

The indirect strategies will do harm to the linguistic quality of the constructed sentences.

Head-N: take the first N sentences

Die Brücke

From Wikipedia, the free encyclopedia

For other uses, see Die Brücke (disambiguation).

Die Brücke (The Bridge) was a group of German expressionist artists formed in Dresden in 1905, after which the **Brücke Museum in Berlin** was named. Founding members were Fritz Bleyl, Erich Heckel, Ernst Ludwig Kirchner and Karl Schmidt-Rottluff. Later members were Emil Nolde, Max Pechstein and Otto Mueller. The seminal group had a major impact on the evolution of modern art in the 20th century and the creation of expressionism.[1]

Die Brücke is sometimes compared to the Fauves. Both movements shared interests in primitivist art. Both

N= 1,2 or 3

With Deep Learning: abstractive

Neural Abstractive Text Summarization

- Input: Congratulations to Australia for seeing sense and dropping the ridiculous policy of not selecting their best players if they are playing overseas.

- Summary: Australia have seen sense by revamping their overseas selection policy.

Classic Methods

Baseline: Classic Methods

Graph-based: TextRank and LexRank

Prerequisite: PageRank

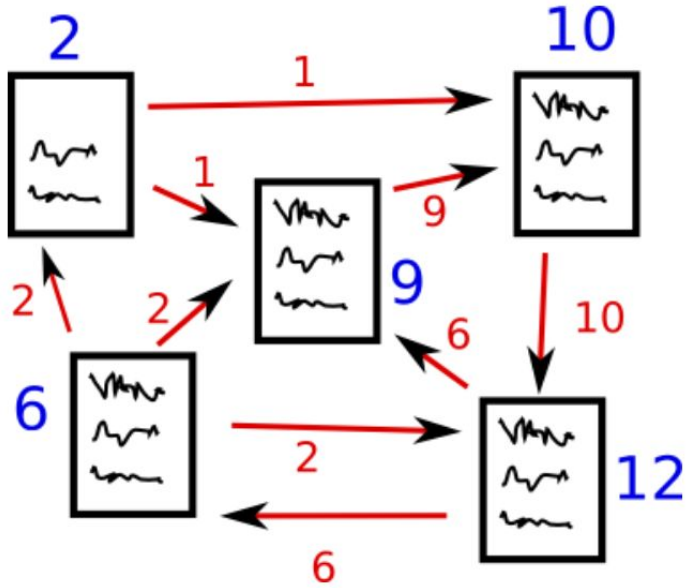
Graph-based Ranking Algorithms

TextRank: Bringing Order into Texts (motivation)

An extractive method.

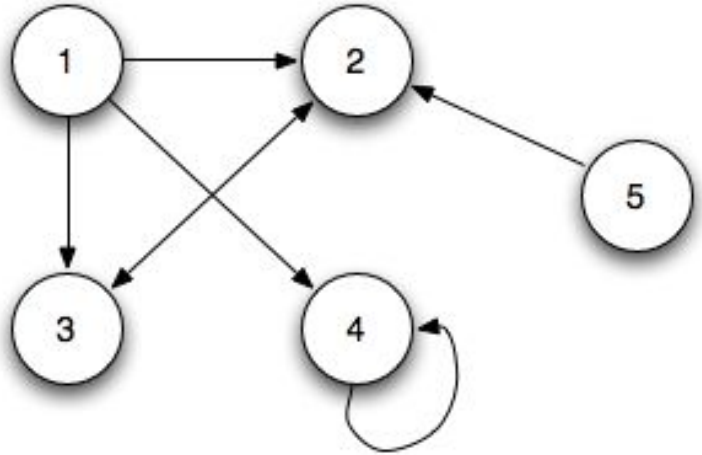
-> Select the most important/top pieces.

Work with Graphs.



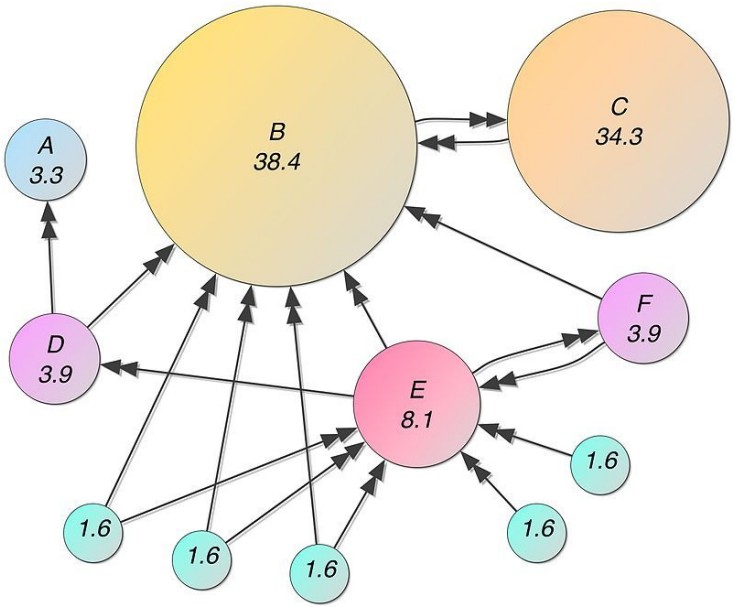
Directed graph

Adjacency matrix is asymmetric for directed graphs.



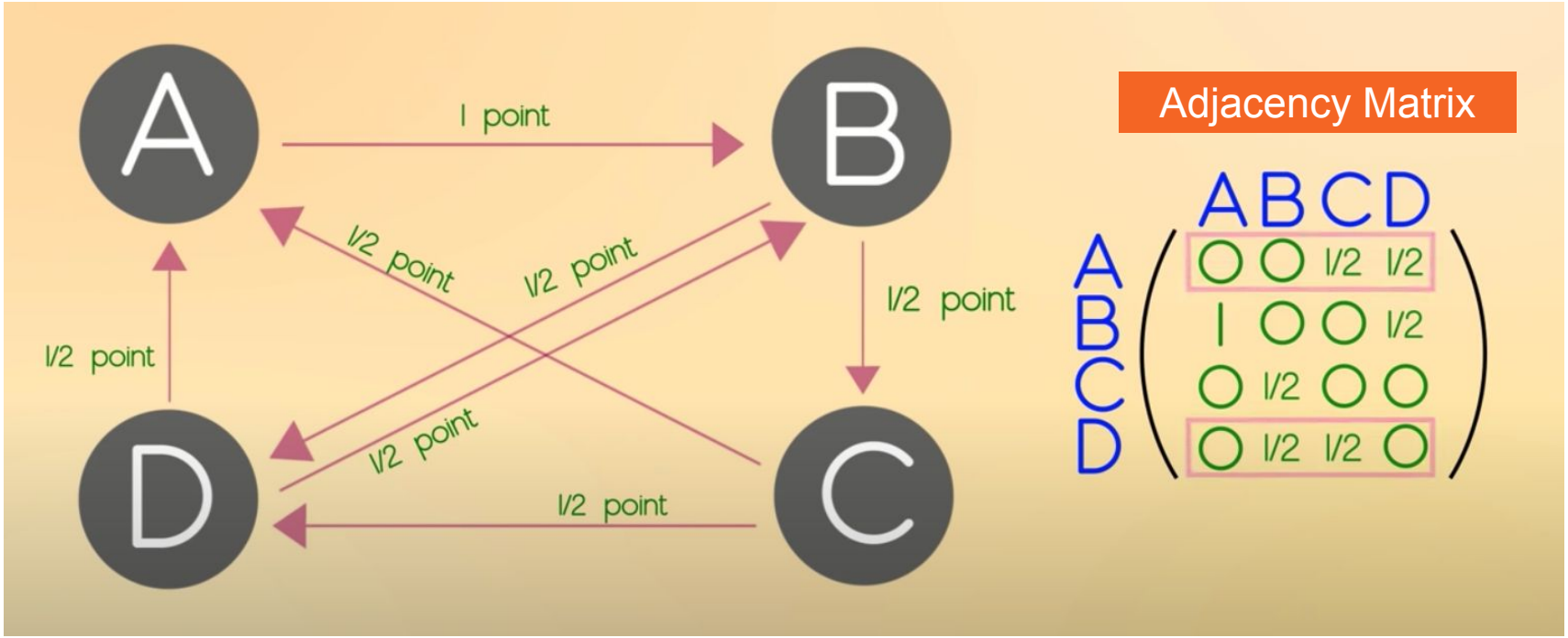
	1	2	3	4	5
1	0	1	1	1	0
2	0	0	1	0	0
3	0	1	0	0	0
4	0	0	0	1	0
5	0	1	0	0	0

Pagerank



PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites.

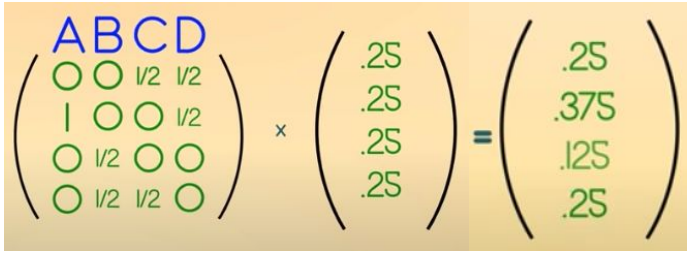
Pagerank: illustration



Pagerank: illustration

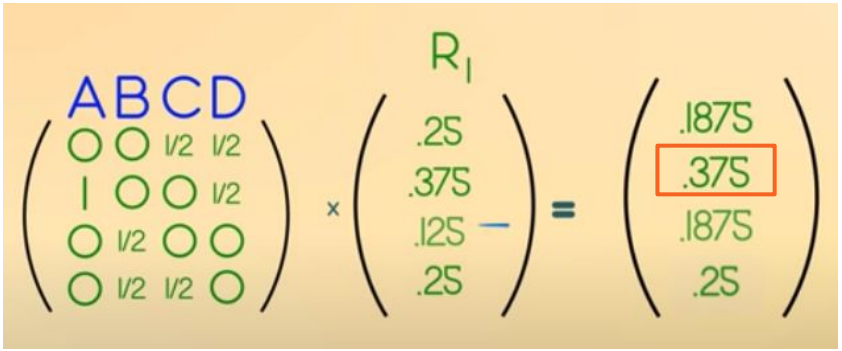
Adjacency Matrix * Initial probabilities (R) = updated probabilities (R1)

Then we go many iterations:



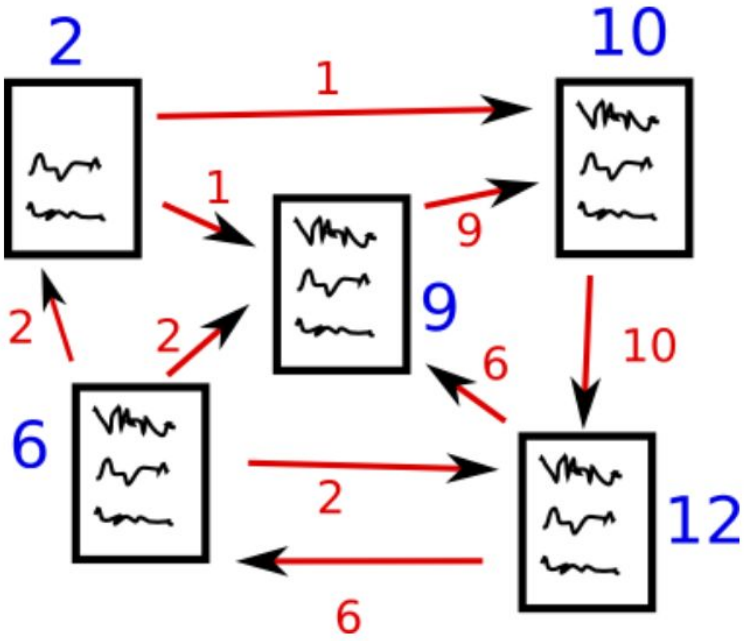
If 2 iterations:

Top node is B

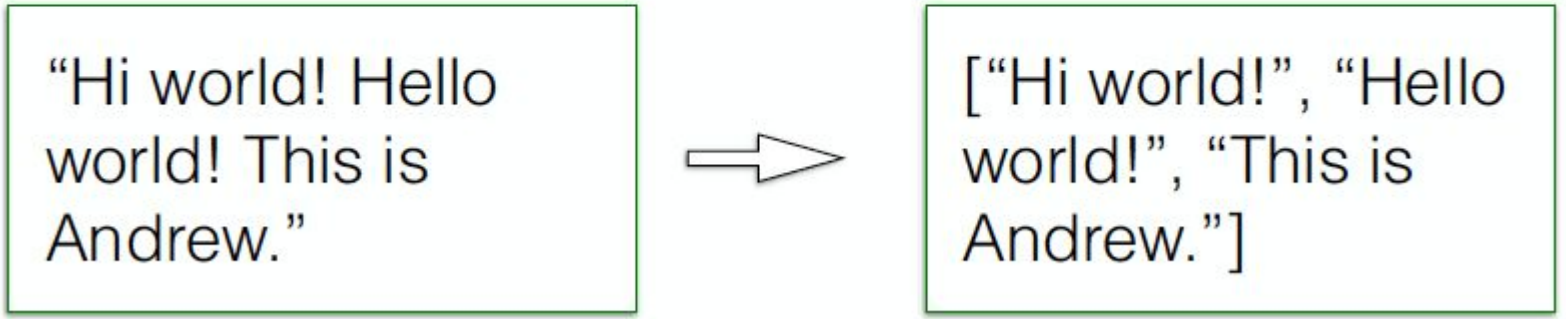


Textrank

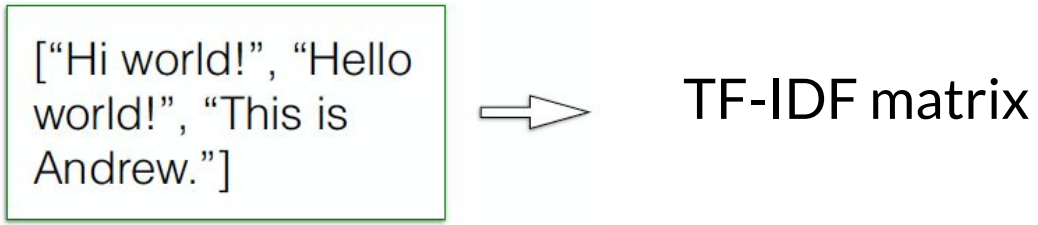
1. Separate the text into sentences
2. Build a sparse matrix of words and the count it appears in each sentence to calculate tf-idf matrix.
3. Construct the similarity matrix between sentences
4. Use Pagerank to score the sentences in graph



1. Separate the Text into Sentences



2. Build a sparse matrix of words and the count it appears in each sentence, get tf-idf



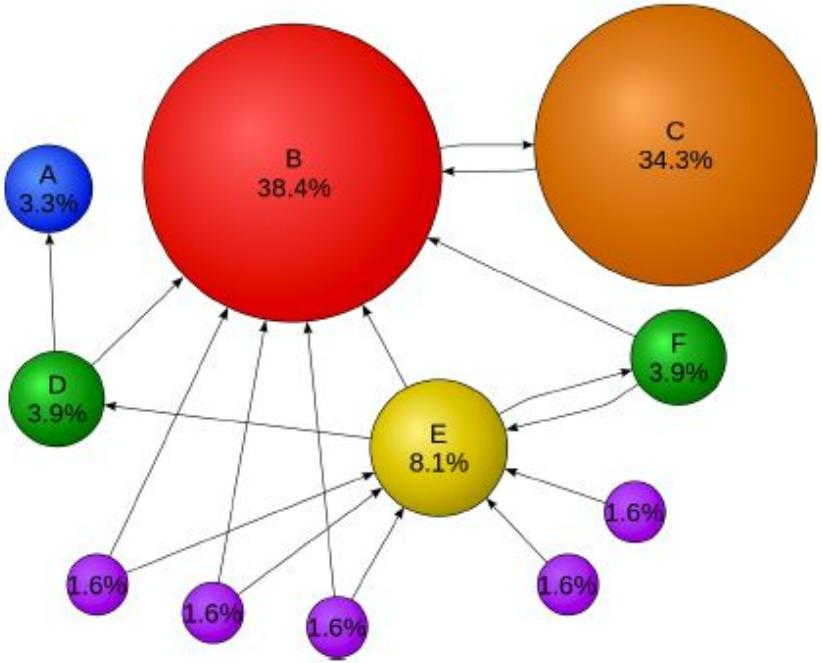
3. Construct the similarity matrix between sentences

TF-IDF matrix

1	0.366	0
0.366	1	0
0	0	1

similarity matrix

4. Use Pagerank to score the sentences in graph



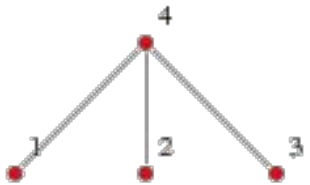
- Rank the sentences with underlying assumption that “summary sentences” are similar to most other sentences

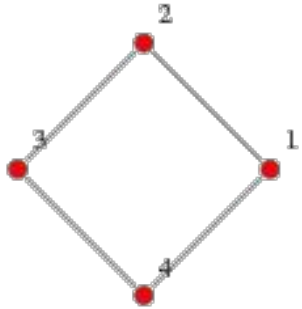
Graph: definition

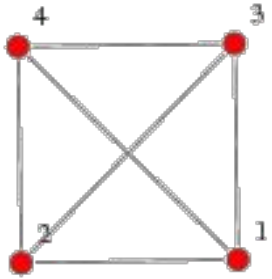
G = (V,E)

Adjacency Matrix

Symmetric







$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Differences between Lexrank and Textrank

TextRank was applied to summarization exactly as described here, while LexRank combines the LexRank score with other features like sentence position and length.

TextRank was used for single document summarization, while LexRank has been applied to multi-document summarization.

How to implement Textrank?

No official implement, but there are some…[**Extractive**]

https://pypi.org/project/pytextrank/

https://pypi.org/project/textrank/

https://pypi.org/project/lexrank/ (Lexrank)

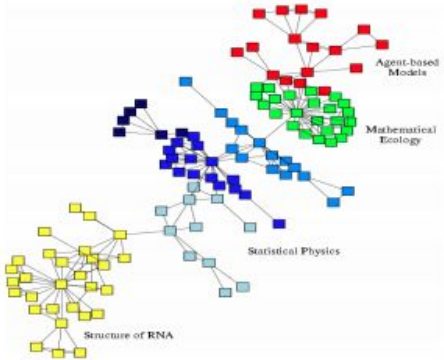
Assignment: Choose one to generate sample summaries using MIMIC NOTEEVENTS data.

Graph Neural Networks

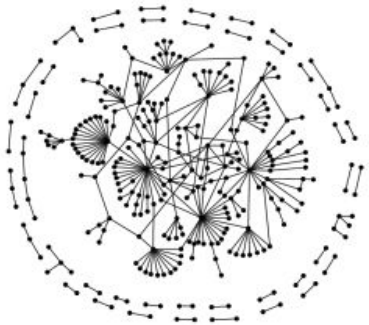
Introduction: Graphs

Source

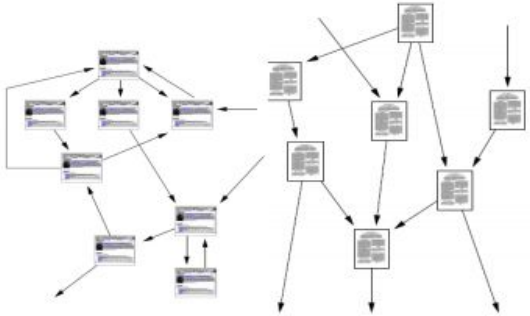
Social networks



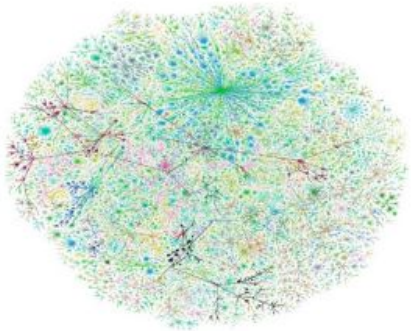
Economic networks



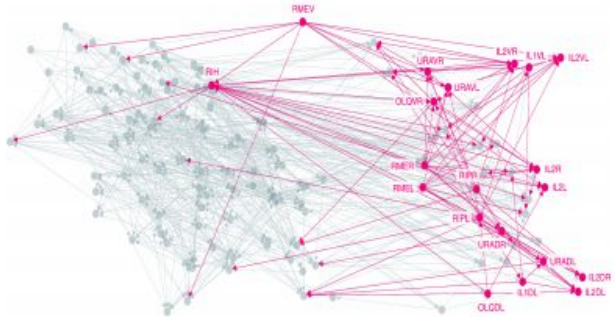
Biomedical networks



Information networks:
Web & citations

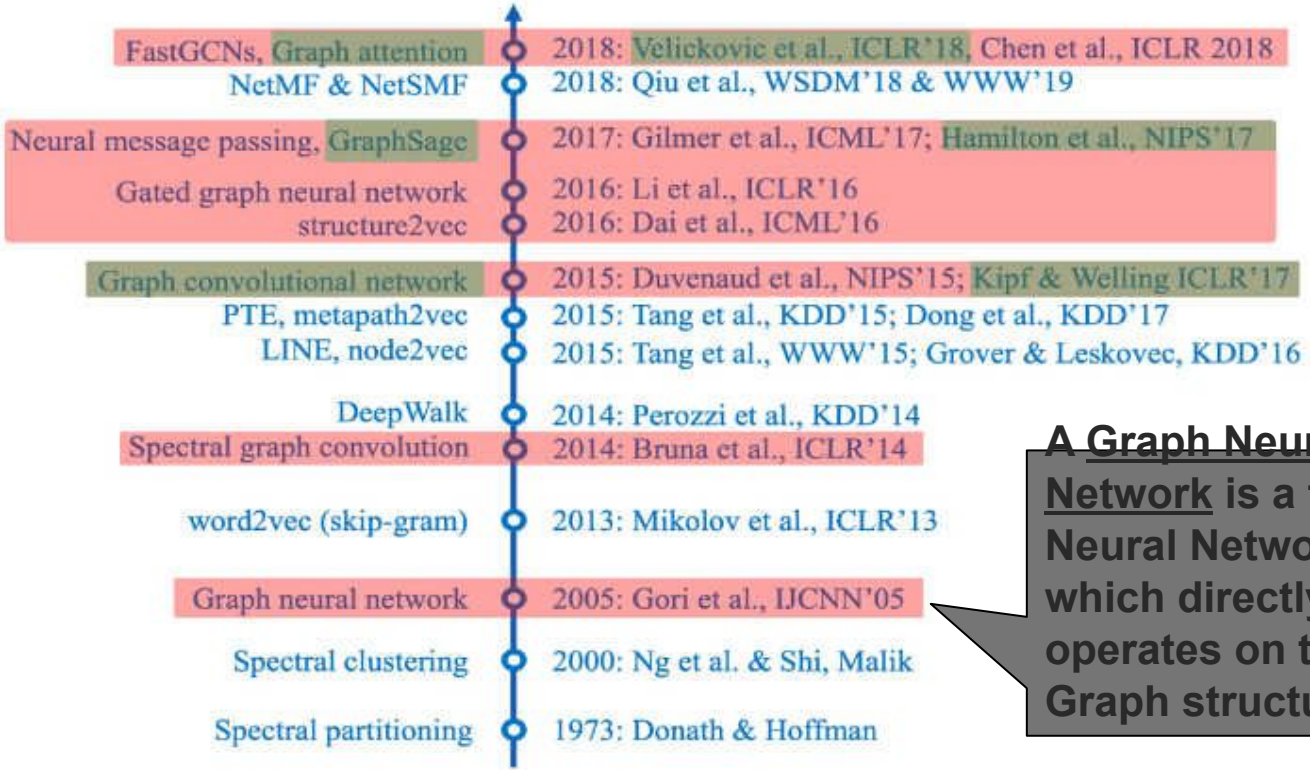


Internet



Networks of neurons

A brief history...

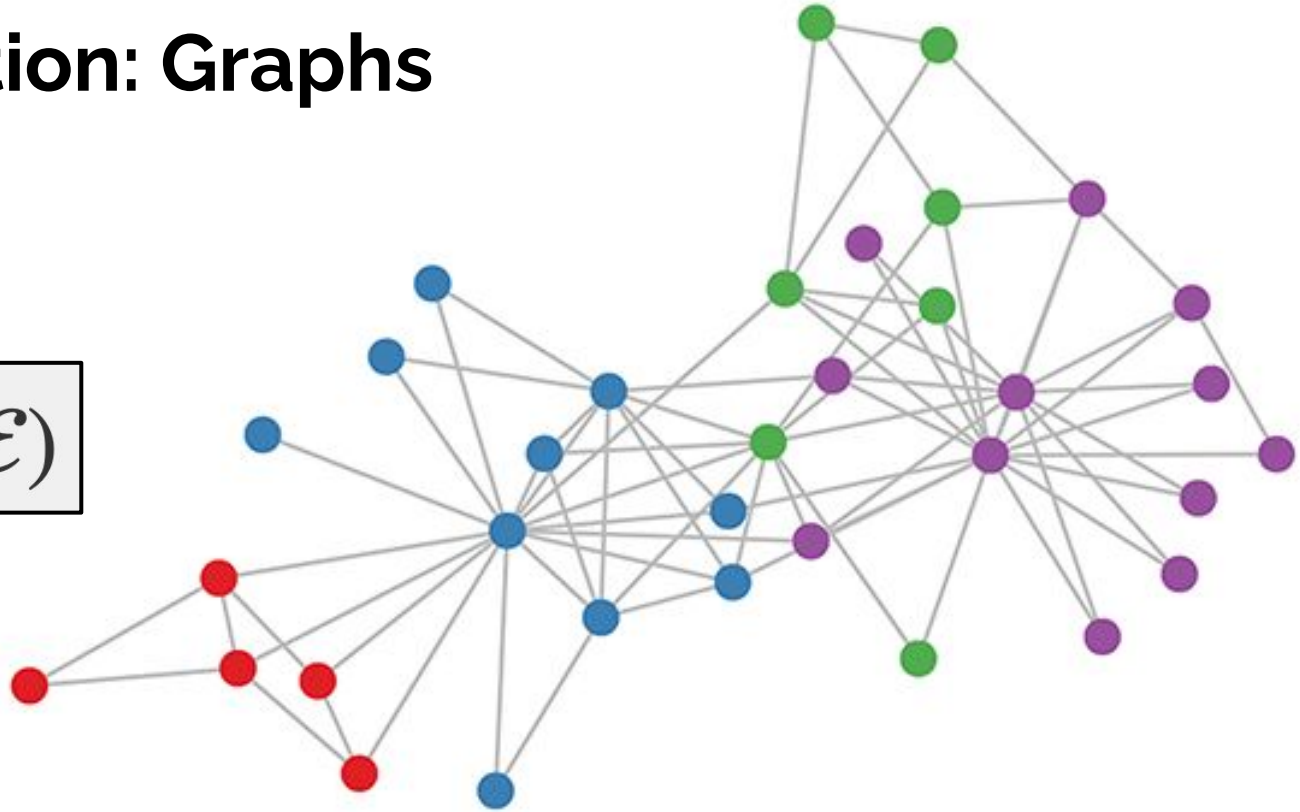


A Graph Neural Network is a type of Neural Network which directly operates on the Graph structure.

Introduction: Graphs

Vertices V
Edges E

$$\mathcal{G} = (\mathcal{V}, \mathcal{E})$$



https://images.app.goo.gl/mnaK9vc1547Sd5UU8 Karate Network

Early Research: DeepWalk (2014)

Goal: Learning latent representations of vertices in a network.

Idea: Skip-Gram Model (Word2vec)

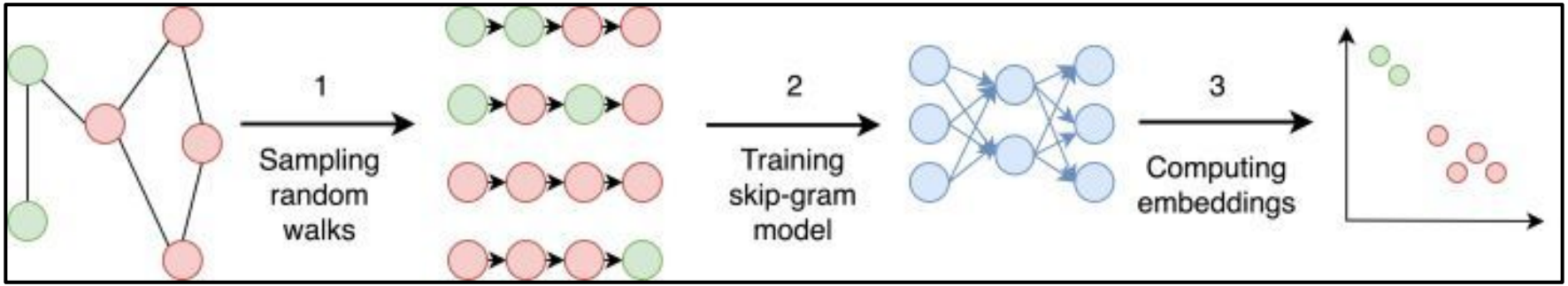
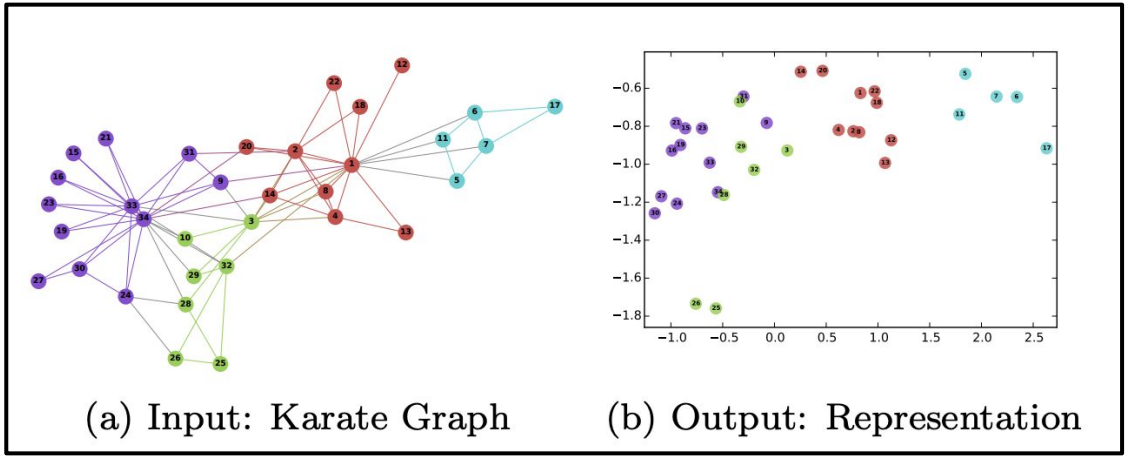
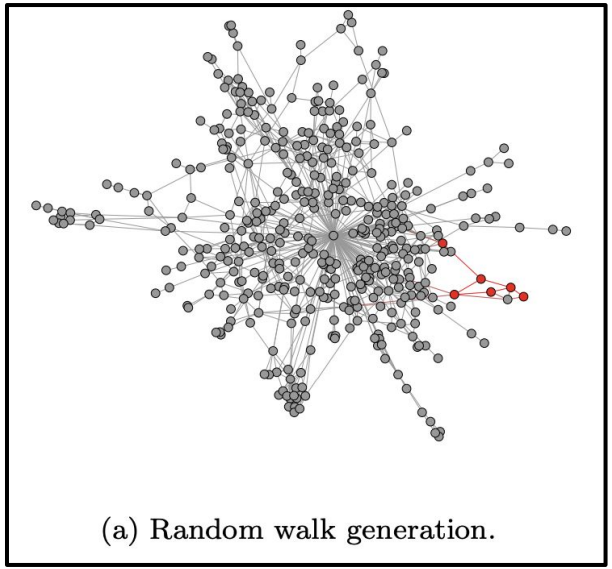


Image source

Node representation



(a) Input: Karate Graph (b) Output: Representation

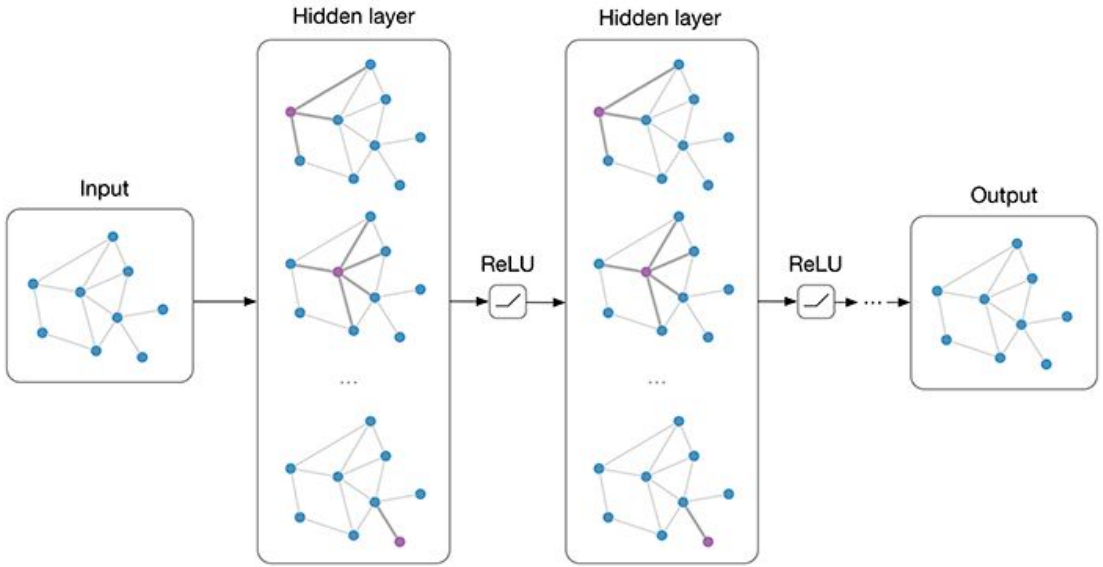


(a) Random walk generation.

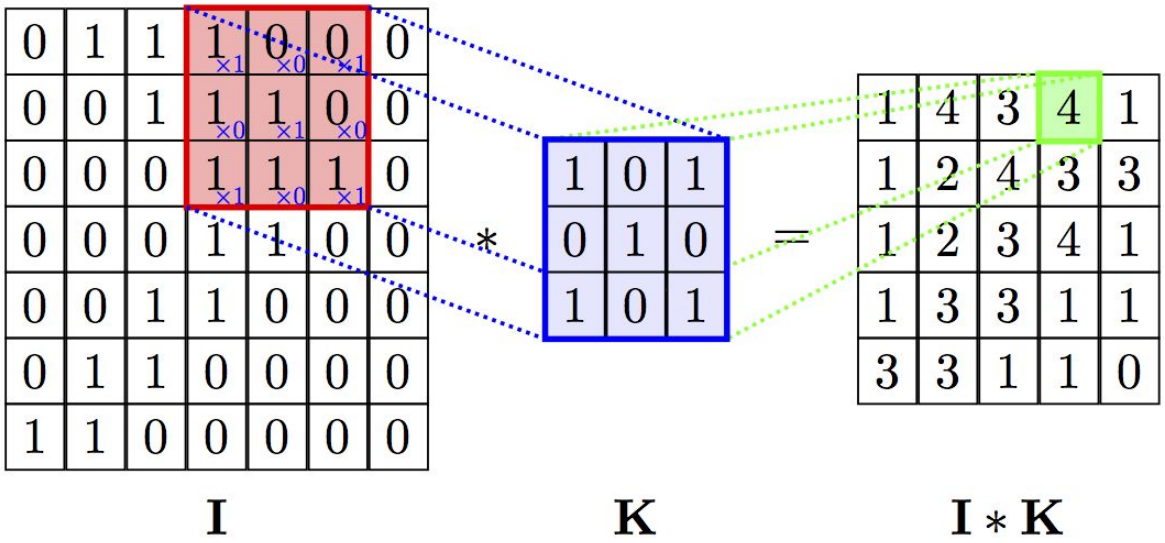
GCN: Graph Convolutional Network (ICLR, 2017)

Focus on Graph structure data & convolution.

How to understand traditional CNN?



From CNN to GCN (1)

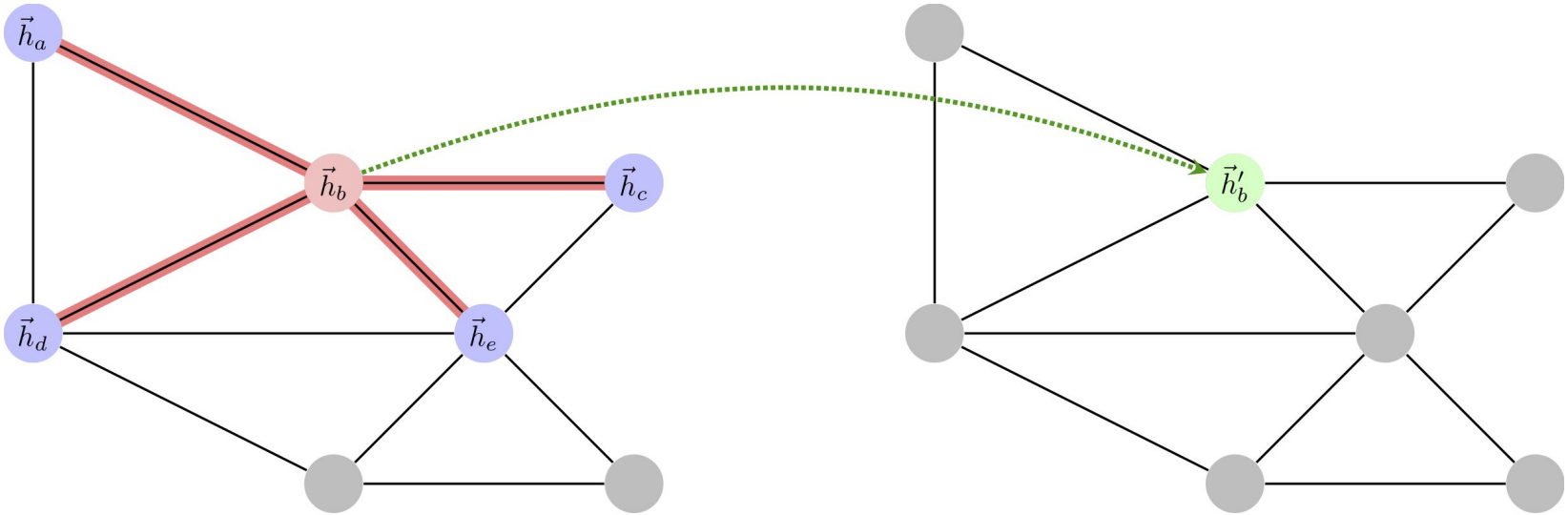


Filter: shared with all locations

http://petar-v.com/GAT/

From CNN to GCN (2)

to go to the next layer using “graph convolution”



http://petar-v.com/GAT/

GCN: formulation

$$\mathcal{G} = (\mathcal{V}, \mathcal{E})$$

Input:

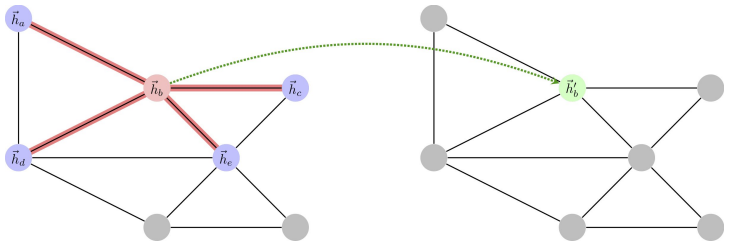
A is adjacency matrix

X is the feature matrix, shape N×D, can be one-hot

H is the outputs of the current layer l

Then choosing **f**.

Every neural network layer:



$$H^{(l+1)} = f(H^{(l)}, A)$$

GCN: formulation

Simple version of f: a single layer NN, with a ReLU as **f**.
W is the parameter matrix.

$$H^{(l+1)} = f(H^{(l)}, A)$$

$$H^{(l+1)} = \sigma(AH^{(l)}W^{(l)})$$

$$H^{(0)} = X$$

GCN: formulation

$$H^{(l+1)} = \sigma\left(AH^{(l)}W^{(l)}\right)$$

Two tricks:

1) Add information of the node from itself: A = A + I
2) Normalize the adjacency matrix A

   Multiplication with A will completely change the scale of the feature vectors (especially in neural networks)

   Degree matrix D

$$D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$$

GCN: formulation

Definition 1 *The* **normalized adjacency matrix** *is*

$$\mathscr{A} \equiv D^{-1/2} A D^{-1/2},$$

where A is the adjacency matrix of G and $D = diag(d)$ for $d(i)$ the degree of node i.

For a graph G (with no isolated vertices), we can see that

$$D^{-1/2} = \begin{pmatrix} \frac{1}{\sqrt{d(1)}} & 0 & \cdots & 0 \\ 0 & \frac{1}{\sqrt{d(2)}} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{1}{\sqrt{d(n)}} \end{pmatrix}.$$

$$\hat{A} = A + I$$

$$f(H^{(l)}, A) = \sigma\left(\hat{D}^{-\frac{1}{2}} \hat{A} \hat{D}^{-\frac{1}{2}} H^{(l)} W^{(l)}\right)$$

GCN: formulation

$$H^{(l+1)} = f(H^{(l)}, A)$$

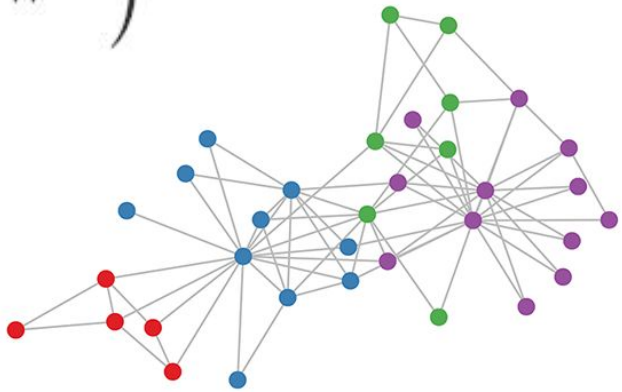
$$H^{(l+1)} = \sigma(AH^{(l)}W^{(l)})$$

$$f(H^{(l)}, A) = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

Node classification: an MLP Layer at the end.

PyTorch: https://github.com/tkipf/pygcn

http://opennmt.net/OpenNMT-py/main.html

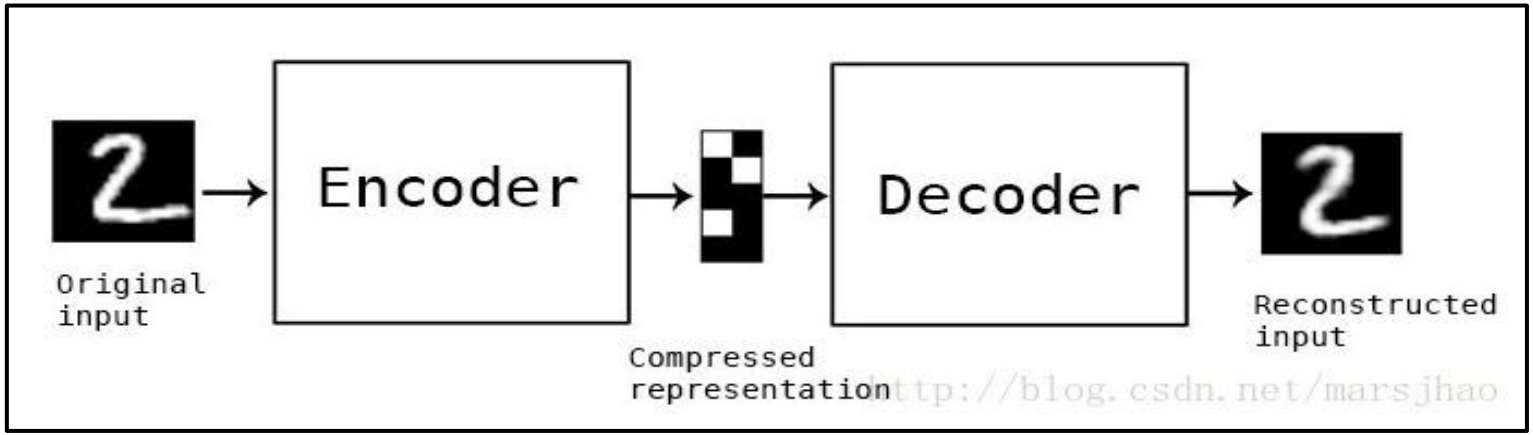


Results: [supervised learning] classify the nodes in citation network

Table 2: Summary of results in terms of classification accuracy (in percent).

Method	Citeseer	Cora	Pubmed	NELL
ManiReg [3]	60.1	59.5	70.7	21.8
SemiEmb [28]	59.6	59.0	71.1	26.7
LP [32]	45.3	68.0	63.0	26.5
DeepWalk [22]	43.2	67.2	65.3	58.1
ICA [18]	69.1	75.1	73.9	23.1
Planetoid* [29]	64.7 (26s)	75.7 (13s)	77.2 (25s)	61.9 (185s)
GCN (this paper)	**70.3** (7s)	**81.5** (4s)	**79.0** (38s)	**66.0** (48s)

Variational Graph Auto-Encoders for link prediction



Similar models on texts, images, ...

Can we do the same thing on graph-structure data?

Details of VGAE

Two layers of GCN as the **encoder**.

$$\mathrm{GCN}(\mathbf{X}, \mathbf{A}) = \tilde{\mathbf{A}}\,\mathrm{ReLU}\big(\tilde{\mathbf{A}}\mathbf{X}\mathbf{W}_0\big)\mathbf{W}_1$$

$$\boldsymbol{\mu} = \mathrm{GCN}_{\boldsymbol{\mu}}(\mathbf{X}, \mathbf{A}) \qquad \log \boldsymbol{\sigma} = \mathrm{GCN}_{\boldsymbol{\sigma}}(\mathbf{X}, \mathbf{A})$$

An inner product as the **Decoder**

$$q(\mathbf{Z} \,|\, \mathbf{X}, \mathbf{A}) = \prod_{i=1}^{N} q(\mathbf{z}_i \,|\, \mathbf{X}, \mathbf{A}), \quad \text{with} \quad q(\mathbf{z}_i \,|\, \mathbf{X}, \mathbf{A}) = \mathcal{N}(\mathbf{z}_i \,|\, \boldsymbol{\mu}_i, \mathrm{diag}(\boldsymbol{\sigma}_i^2))$$

$$p\left(\mathbf{A} \,|\, \mathbf{Z}\right) = \prod_{i=1}^{N} \prod_{j=1}^{N} p\left(A_{ij} \,|\, \mathbf{z}_i, \mathbf{z}_j\right), \quad \text{with} \quad p\left(A_{ij} = 1 \,|\, \mathbf{z}_i, \mathbf{z}_j\right) = \sigma(\mathbf{z}_i^\top \mathbf{z}_j)$$

$$\mathcal{L} = \mathbb{E}_{q(\mathbf{Z}|\mathbf{X},\mathbf{A})}\big[\log p\left(\mathbf{A} \,|\, \mathbf{Z}\right)\big] - \mathrm{KL}\big[q(\mathbf{Z} \,|\, \mathbf{X}, \mathbf{A}) \,||\, p(\mathbf{Z})\big]$$

A simple version (without *variation*) ...

Graph Auto-Encoders (GAE)

For a simple GAE, we will get rid of the distribution restrictions, simply take a GCN as the encoder and an inner product function as the decoder:

$$\hat{\mathbf{A}} = \sigma\left(\mathbf{Z}\mathbf{Z}^{\top}\right)$$

$$\mathbf{Z} = GCN(\mathbf{X}, \mathbf{A})$$

Results: [Semi-supervised] Classify edges/non-edges

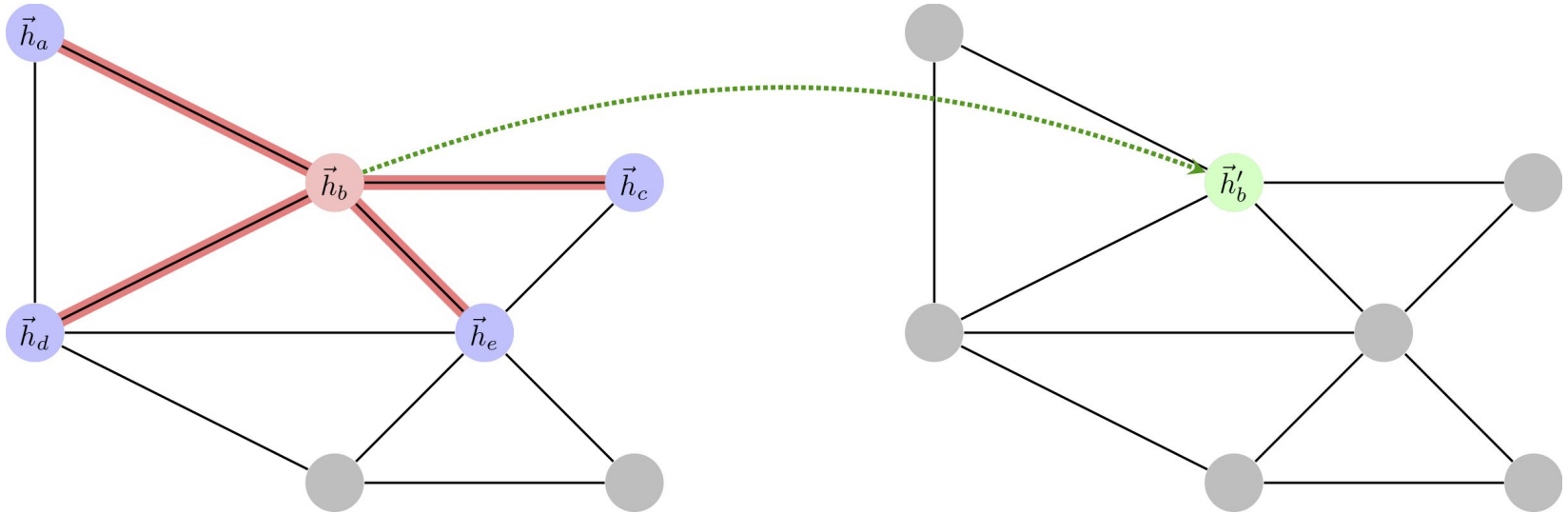
The models are trained on an incomplete version of these datasets where parts of the citation links (edges) have been removed, while all node features are kept. Complete X; incomplete A → A

Table 1: Link prediction task in citation networks. See [1] for dataset details.

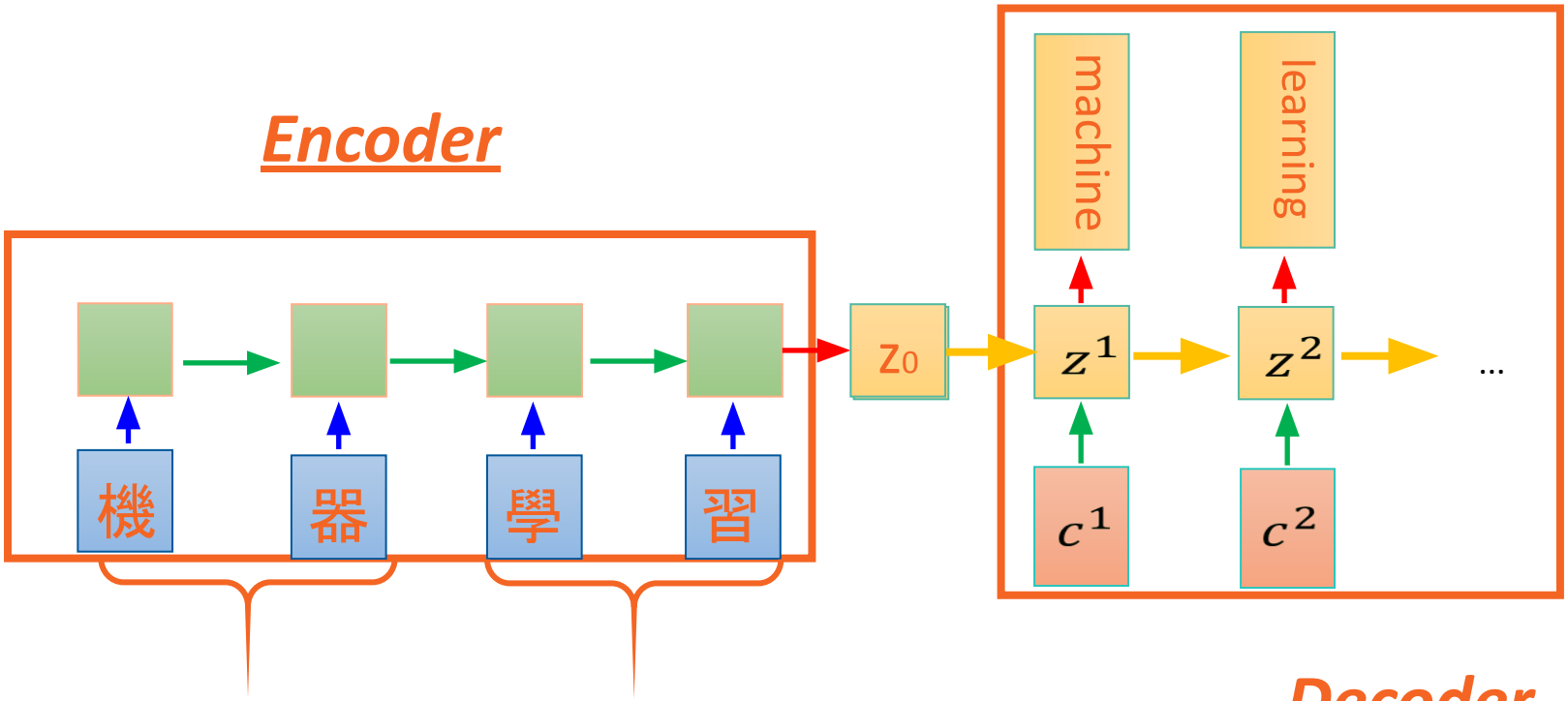
Method	Cora		Citeseer		Pubmed	
	AUC	AP	AUC	AP	AUC	AP
SC [5]	84.6 ± 0.01	88.5 ± 0.00	80.5 ± 0.01	85.0 ± 0.01	84.2 ± 0.02	87.8 ± 0.01
DW [6]	83.1 ± 0.01	85.0 ± 0.00	80.5 ± 0.02	83.6 ± 0.01	84.4 ± 0.00	84.1 ± 0.00
GAE*	84.3 ± 0.02	88.1 ± 0.01	78.7 ± 0.02	84.1 ± 0.02	82.2 ± 0.01	87.4 ± 0.00
VGAE*	84.0 ± 0.02	87.7 ± 0.01	78.9 ± 0.03	84.1 ± 0.02	82.7 ± 0.01	87.5 ± 0.01
GAE	91.0 ± 0.02	92.0 ± 0.03	89.5 ± 0.04	89.9 ± 0.05	$\mathbf{96.4} \pm 0.00$	$\mathbf{96.5} \pm 0.00$
VGAE	$\mathbf{91.4} \pm 0.01$	$\mathbf{92.6} \pm 0.01$	$\mathbf{90.8} \pm 0.02$	$\mathbf{92.0} \pm 0.02$	94.4 ± 0.02	94.7 ± 0.02

Question: Better ways to “move” to the next layer?

Attention!

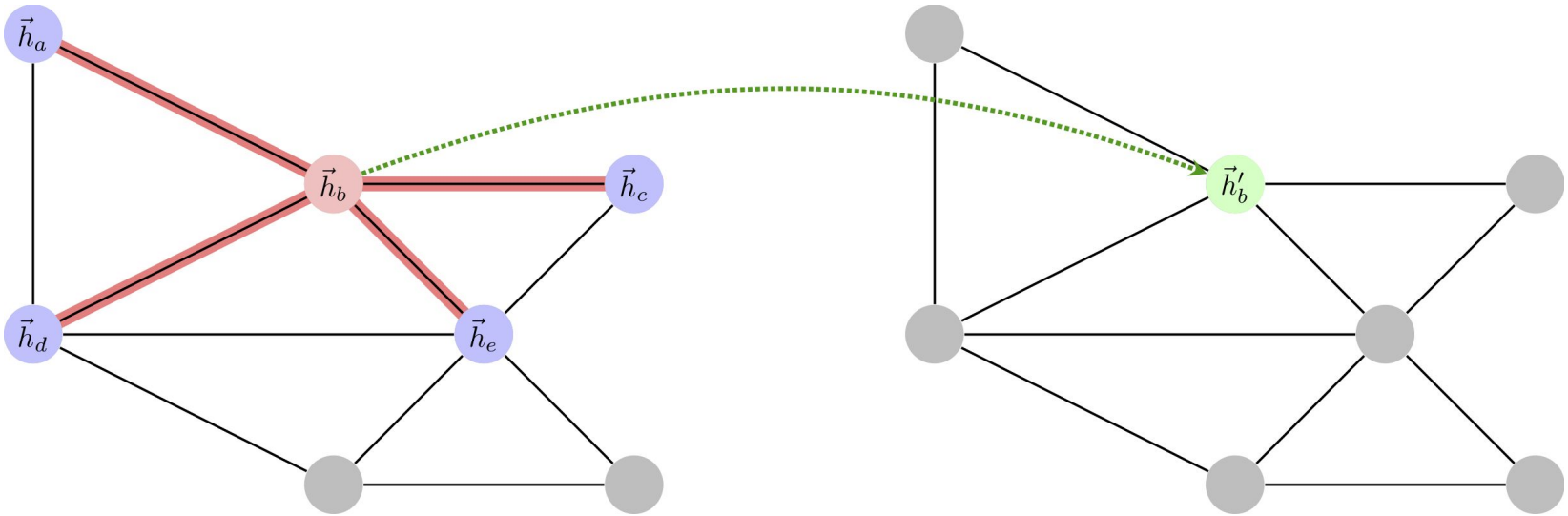


Recall Attention in seq2seq



Source image: https://speech.ee.ntu.edu.tw/~tlkagk/

Attention in graph: looking at the neighbors



http://petar-v.com/GAT/

Graph Attention Networks (ICLR, 2019)

Self-attention: by looking at the **neighbors** with different weights.

Which neighbor has a larger impact/similar to it?
Calculate a node pair (i,j):

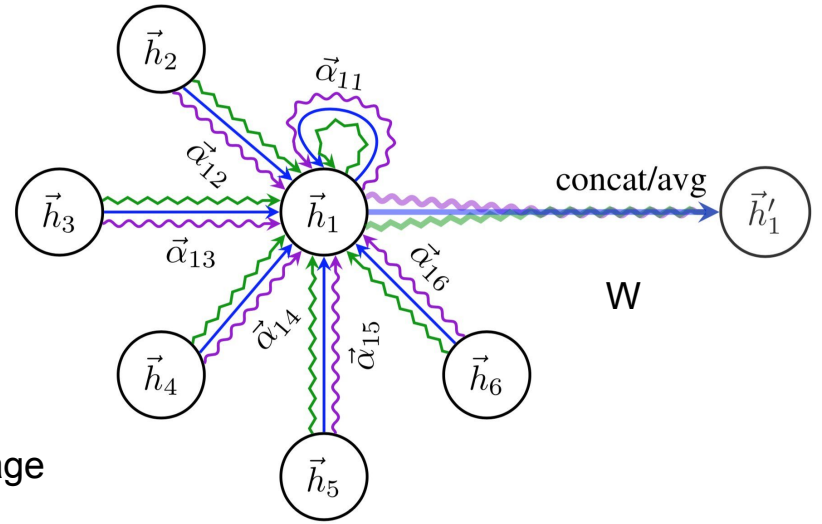
$$e_{ij} = a(\vec{h}_i, \vec{h}_j)$$

Normalize over all the neighbors:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})}$$

Use neighbors to represent the current node: weighted average

$$\vec{h}_i' = \Big\|_{k=1}^{K} \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k \vec{h}_j\right)$$



Multi-head

Results

Table 2: Summary of results in terms of classification accuracies, for Cora, Citeseer and Pubmed. GCN-64* corresponds to the best GCN result computing 64 hidden features (using ReLU or ELU).

Transductive

Method	**Cora**	**Citeseer**	**Pubmed**
MLP	55.1%	46.5%	71.4%
ManiReg (Belkin et al., 2006)	59.5%	60.1%	70.7%
SemiEmb (Weston et al., 2012)	59.0%	59.6%	71.7%
LP (Zhu et al., 2003)	68.0%	45.3%	63.0%
DeepWalk (Perozzi et al., 2014)	67.2%	43.2%	65.3%
ICA (Lu & Getoor, 2003)	75.1%	69.1%	73.9%
Planetoid (Yang et al., 2016)	75.7%	64.7%	77.2%
Chebyshev (Defferrard et al., 2016)	81.2%	69.8%	74.4%
GCN (Kipf & Welling, 2017)	81.5%	70.3%	**79.0%**
MoNet (Monti et al., 2016)	$81.7 \pm 0.5\%$	—	$78.8 \pm 0.3\%$
GCN-64*	$81.4 \pm 0.5\%$	$70.9 \pm 0.5\%$	$\mathbf{79.0} \pm 0.3\%$
GAT (ours)	$\mathbf{83.0} \pm 0.7\%$	$\mathbf{72.5} \pm 0.7\%$	$\mathbf{79.0} \pm 0.3\%$

About GATs

Better representation ability than GCN with attention mechanism.
Shared attention module: locally, efficient.

More: http://petar-v.com/GAT/

4 things you must know about GNNs

1. Graph-structural data?
2. Adjacency matrix A?
3. Node representation X?
4. Propagation Rule f?

Fit in a proper scenario!

How could we utilize GNNs in NLP?

Application: Graph Convolutional Networks for Text Classification (AAAI, 2019)

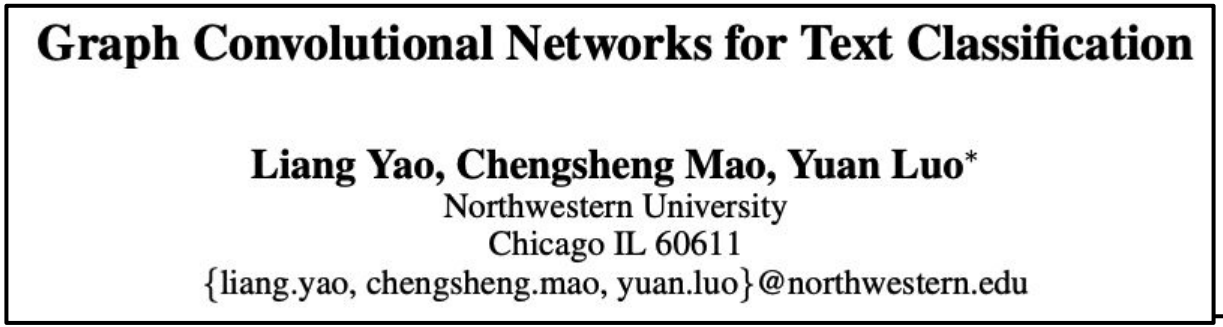
Highlights:

Text classification: normally treated as sequences;

Investigate solving this task using GCN-based models;

No complicated embeddings are needed to initialize the nodes: 1-hot vectors. Then it jointly learns the embeddings for both words and documents;

Efficiency and robustness.



Graph Convolutional Networks for Text Classification

Liang Yao, Chengsheng Mao, Yuan Luo*
Northwestern University
Chicago IL 60611
{liang.yao, chengsheng.mao, yuan.luo}@northwestern.edu

Background: document classification

Representation learning methods: TF-IDF, Bag-of-word, word2vec, contextualized word embeddings...

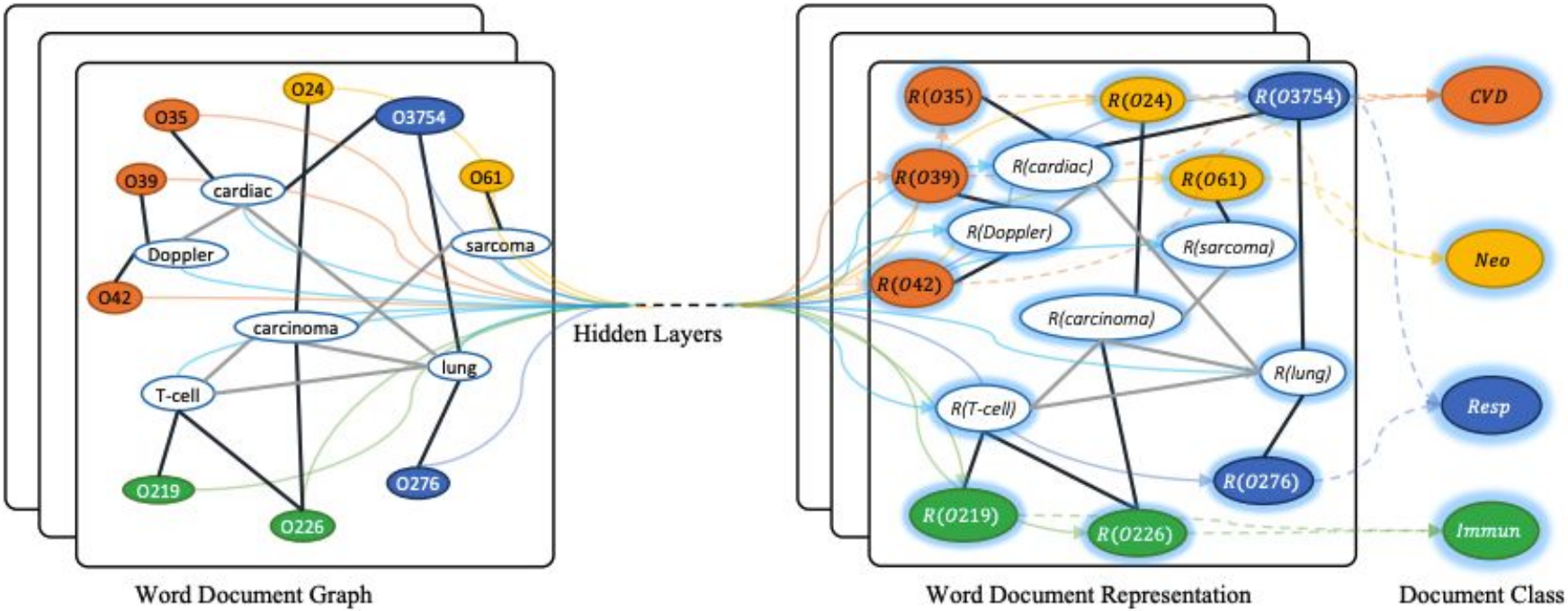
Recent methods for classification: CNN, LSTM, BERT

Treat each individual sentence/document as sequences;
To some extent, each training sample is independent.

GCNs: Graph neural networks have been effective at tasks thought to have rich relational structure and can preserve global structure information of a graph in graph embeddings.

Utilizing relations BETWEEN the corpus?

Building a graph for the corpus



Adjacency Matrix A

Word node and document node.
Pointwise mutual information (PMI)

$$A_{ij} = \begin{cases} \text{PMI}(i,j) & i,j \text{ are words, } \text{PMI}(i,j) > 0 \\ \text{TF-IDF}_{ij} & i \text{ is document, } j \text{ is word} \\ 1 & i = j \\ 0 & \text{otherwise} \end{cases}$$

$$\text{PMI}(i,j) = \log \frac{p(i,j)}{p(i)p(j)}$$

$$p(i,j) = \frac{\#W(i,j)}{\#W}$$

$$p(i) = \frac{\#W(i)}{\#W}$$

Node representation X

1-hot vector to initialize each single node: tokens and documents

Propagation rule

Two GCN lavers: A two-laver GCN can allow message passing among nodes that are at $Z = \mathrm{softmax}(\tilde{A}\,\mathrm{ReLU}(\tilde{A}XW_0)W_1)$

$$\mathcal{L} = -\sum_{d \in \mathcal{Y}_D} \sum_{f=1}^{F} Y_{df} \ln Z_{df}$$

Experiments: public datasets

Table 1: Summary statistics of datasets.

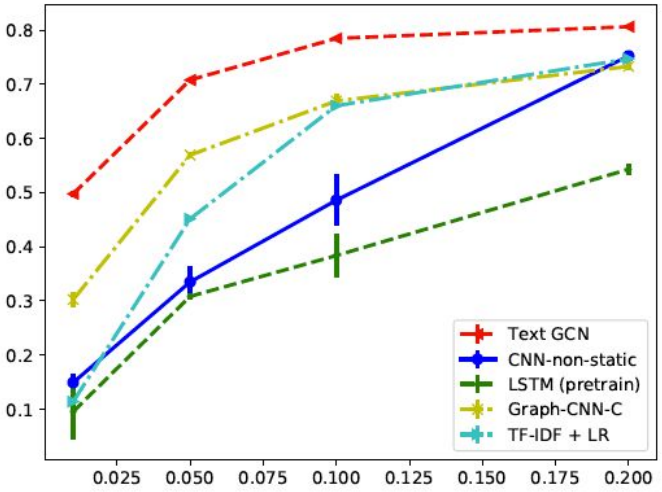
Dataset	# Docs	# Training	# Test	# Words	# Nodes	# Classes	Average Length
20NG	18,846	11,314	7,532	42,757	61,603	20	221.26
R8	7,674	5,485	2,189	7,688	15,362	8	65.72
R52	9,100	6,532	2,568	8,892	17,992	52	69.82
Ohsumed	7,400	3,357	4,043	14,157	21,557	23	135.82
MR	10,662	7,108	3,554	18,764	29,426	2	20.39

Experimental results

Model	20NG	R8	R52	Ohsumed	MR
TF-IDF + LR	0.8319 ± 0.0000	0.9374 ± 0.0000	0.8695 ± 0.0000	0.5466 ± 0.0000	0.7459 ± 0.0000
CNN-rand	0.7693 ± 0.0061	0.9402 ± 0.0057	0.8537 ± 0.0047	0.4387 ± 0.0100	0.7498 ± 0.0070
CNN-non-static	0.8215 ± 0.0052	0.9571 ± 0.0052	0.8759 ± 0.0048	0.5844 ± 0.0106	**0.7775 ± 0.0072**
LSTM	0.6571 ± 0.0152	0.9368 ± 0.0082	0.8554 ± 0.0113	0.4113 ± 0.0117	0.7506 ± 0.0044
LSTM (pretrain)	0.7543 ± 0.0172	0.9609 ± 0.0019	0.9048 ± 0.0086	0.5110 ± 0.0150	0.7733 ± 0.0089
Bi-LSTM	0.7318 ± 0.0185	0.9631 ± 0.0033	0.9054 ± 0.0091	0.4927 ± 0.0107	0.7768 ± 0.0086
PV-DBOW	0.7436 ± 0.0018	0.8587 ± 0.0010	0.7829 ± 0.0011	0.4665 ± 0.0019	0.6109 ± 0.0010
PV-DM	0.5114 ± 0.0022	0.5207 ± 0.0004	0.4492 ± 0.0005	0.2950 ± 0.0007	0.5947 ± 0.0038
PTE	0.7674 ± 0.0029	0.9669 ± 0.0013	0.9071 ± 0.0014	0.5358 ± 0.0029	0.7023 ± 0.0036
fastText	0.7938 ± 0.0030	0.9613 ± 0.0021	0.9281 ± 0.0009	0.5770 ± 0.0049	0.7514 ± 0.0020
fastText (bigrams)	0.7967 ± 0.0029	0.9474 ± 0.0011	0.9099 ± 0.0005	0.5569 ± 0.0039	0.7624 ± 0.0012
SWEM	0.8516 ± 0.0029	0.9532 ± 0.0026	0.9294 ± 0.0024	0.6312 ± 0.0055	0.7665 ± 0.0063
LEAM	0.8191 ± 0.0024	0.9331 ± 0.0024	0.9184 ± 0.0023	0.5858 ± 0.0079	0.7695 ± 0.0045
Graph-CNN-C	0.8142 ± 0.0032	0.9699 ± 0.0012	0.9275 ± 0.0022	0.6386 ± 0.0053	0.7722 ± 0.0027
Graph-CNN-S	–	0.9680 ± 0.0020	0.9274 ± 0.0024	0.6282 ± 0.0037	0.7699 ± 0.0014
Graph-CNN-F	–	0.9689 ± 0.0006	0.9320 ± 0.0004	0.6304 ± 0.0077	0.7674 ± 0.0021
Text GCN	**0.8634 ± 0.0009**	**0.9707 ± 0.0010**	**0.9356 ± 0.0018**	**0.6836 ± 0.0056**	0.7674 ± 0.0020

Test accuracy by varying training data proportions.

Effects of the Size of Labeled Data:
GCN can perform quite well with low label rate (Kipf and Welling, 2017)



(a) 20NG

Thanks!