香港浸會大學
HONG KONG BAPTIST UNIVERSITY

DEPARTMENT OF
COMPUTER SCIENCE
計算機科學系

COMP7015 Artificial Intelligence (S1, 2024-25)

Lecture 9: Deep Learning III

Instructor: Dr. Kejing Yin (cskjyin@hkbu.edu.hk)

Department of Computer Science
Hong Kong Baptist University

November 15, 2024

Announcements

- Lab 4 *Using Pre-trained Models in PyTorch*: Nov. 16.

- Course Project Submission is due on Nov. 22 at 11:59 am.
  - Submission box will open on Nov. 19 and automatically close past deadline.
  - Late submissions will NOT be accepted.
- Register for your project presentation as soon as possible.

- Programming Assignment 2 is due on Nov. 29 at 11:59 am.

Agenda for Today

- Recurrent Neural Network (RNN) Part II
  - Long-Short Term Model (LSTM)
  - Gated Recurrent Units (GRU)
- Language Modeling
  - Word Vectors
  - Pretrained Language Models
- Transformer Models
  - Attention Mechanism
  - The Transformer Architecture

Recap: Sequential Data

- Usually, we represent each sequential data sample as

$$X = [\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(t)}, \dots, \boldsymbol{x}^{(\tau)}]$$

  where the index t is the time step, τ is number of time steps in total, and each $\boldsymbol{x}^{(t)}$ is a d-dimensional vector.

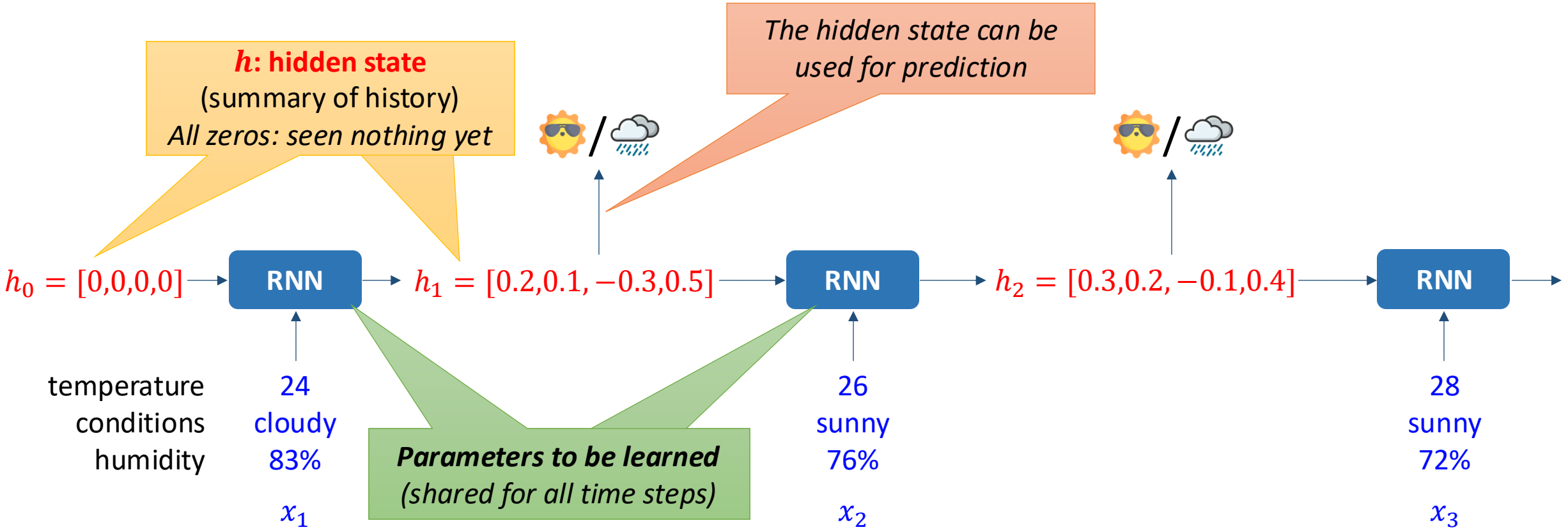
- $\boldsymbol{x}^{(t)}$ usually depends on $\boldsymbol{x}^{(t')}$ $(t' < t)$.

Problems of training sequential data with MLP:

- In any sequence, nearby items are related, and the order of items matters. Fully connected layer treats every item independently, unless it learns otherwise.
- Sequence length can vary across examples within the same task. For MLP, shape of input and output is determined at design time.

Recap: Recurrent Neural Networks (RNN)

Example: predicting rainfall using sequential weather data



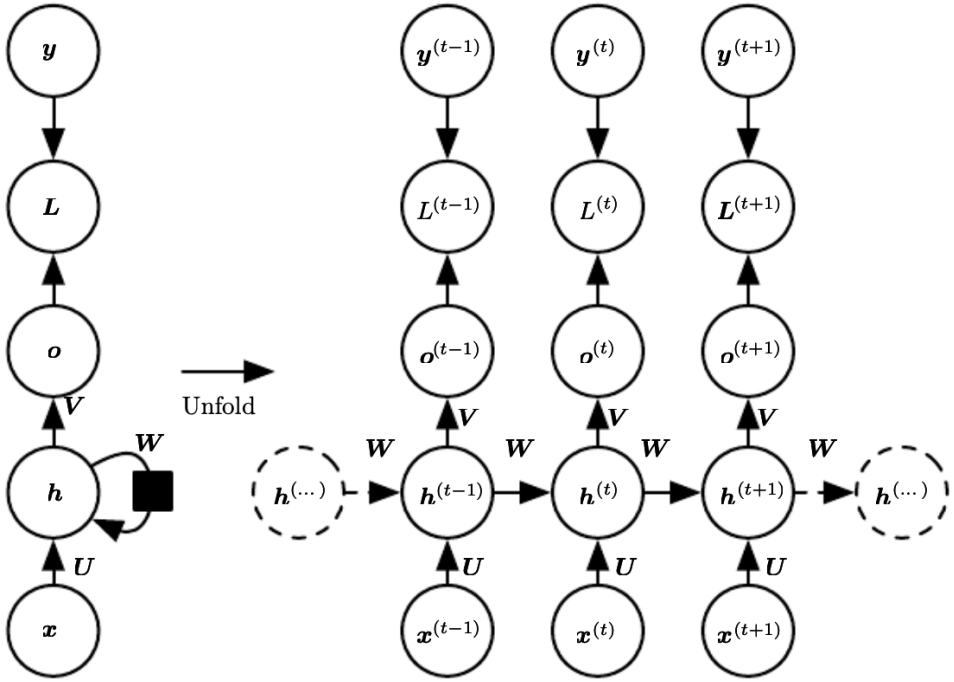
Recap: Vanilla RNN

- For each time step from $t = 1$ to $t = \tau$, we apply the following update equations:

combining history and new data

$$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)},$$
$$\boldsymbol{h}^{(t)} = \tanh\left(\boldsymbol{a}^{(t)}\right),$$
$$\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)},$$
$$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}\left(\boldsymbol{o}^{(t)}\right) \text{ or } \text{sigmoid}\left(\boldsymbol{o}^{(t)}\right)$$
$$L^{(t)} = Loss\left(\hat{\boldsymbol{y}}^{(t)}, \boldsymbol{y}^{(t)}\right).$$

Parameters to be learned: $\boldsymbol{b}, \boldsymbol{W}, \boldsymbol{U}, \boldsymbol{c}$, and $\boldsymbol{V}$



Vanilla RNN

Image source: Figure 10.2, Goodfellow, Bengio, and Courville, Deep Learning, Cambridge: MIT press, 2016.

Bidirectional RNNs

- Up to now, the state at time t only captures information from the past $x^{(1)}, x^{(2)}, \dots, x^{(t-1)}$.
- However, in many applications we want to output a prediction of $y(t)$ which may depend on the whole input sequence.
- Bidirectional RNNs combine an RNN that moves forward with another RNN that moves backward.

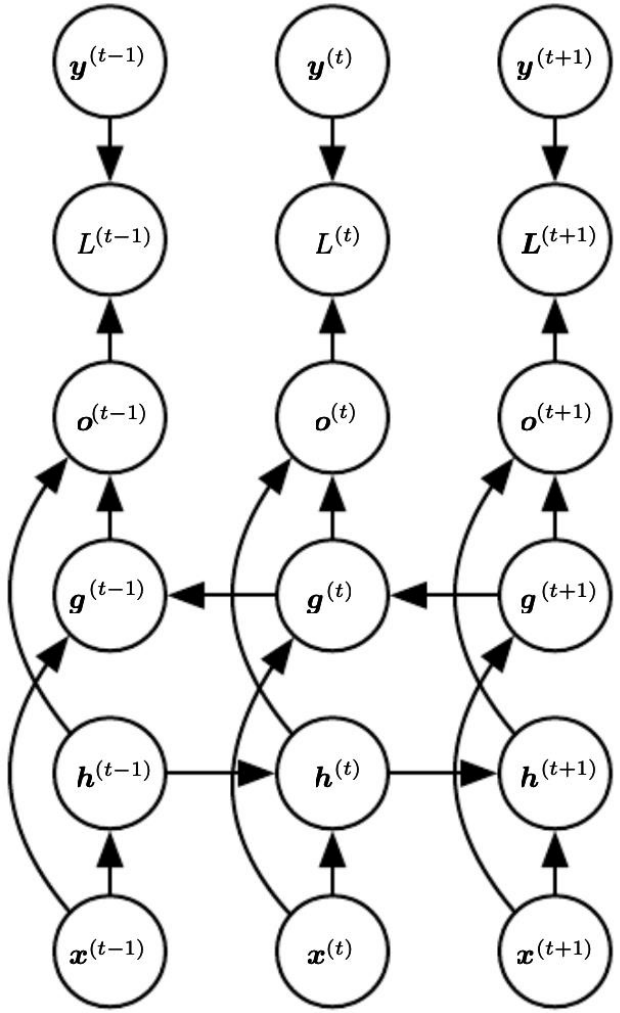


Image source: Figure 10.11, Goodfellow, Bengio, and Courville, Deep Learning, Cambridge: MIT press, 2016.

Deep Recurrent Networks

- The computation in most RNNs can be decomposed into three blocks of parameters and associated transformations:
  a. input to hidden;
  b. hidden to hidden;
  c. hidden to output.
- Would it be advantageous to introduce depth in each of these operations?
  - Of course, and the deep module can be incorporated into different RNN components.

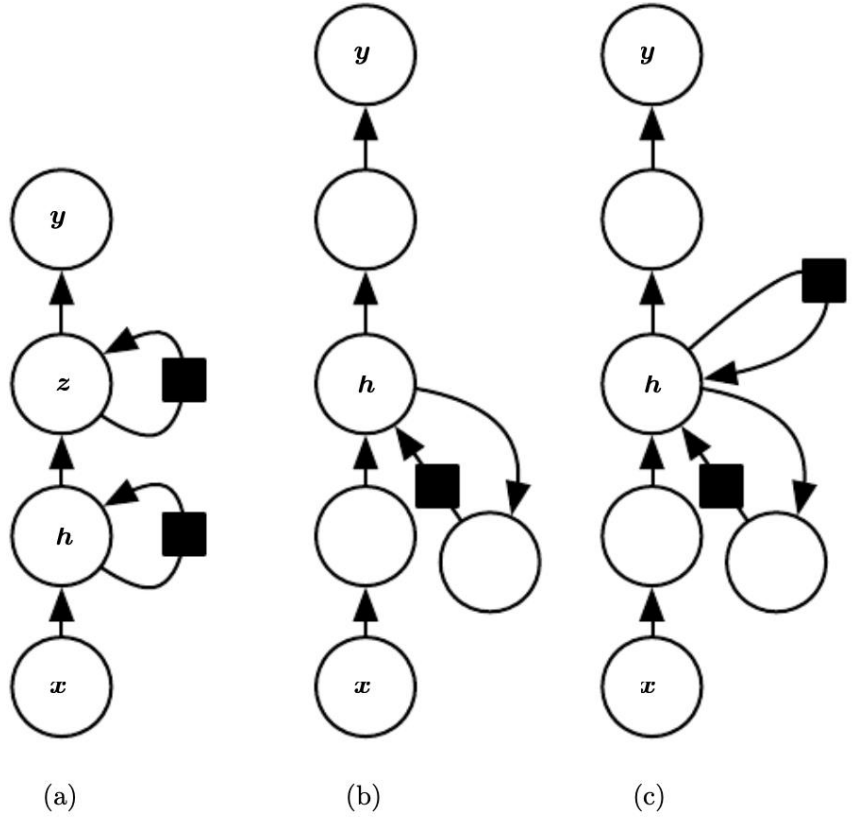
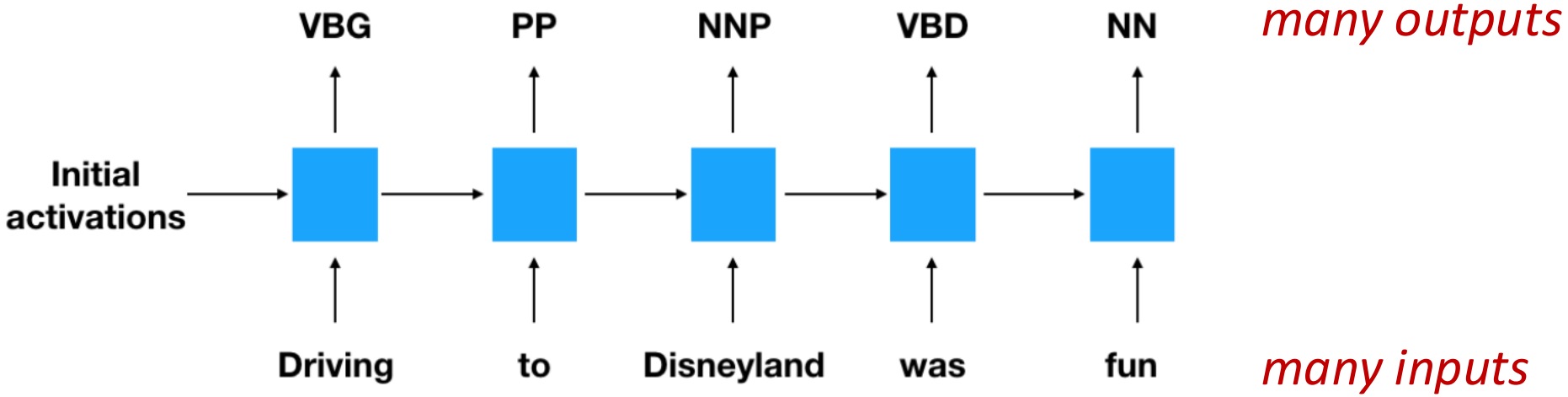


Image source: Figure 10.13, Goodfellow, Bengio, and Courville, Deep Learning, Cambridge: MIT press, 2016.

RNN Architecture: Many-to-Many

- Application: Part-of-Speech (POS) tagging



VBG: Verb, gerund or present participle
PP: Proper noun, singular
VBD: Verb, past tense
NN: Noun, singular or mass

Image source: https://stackoverflow.com/questions/50455556/rnn-with-simultaneous-pos-tagging-and-sentiment-classification

RNN Architecture: Many-to-One

- Application: Sentiment analysis

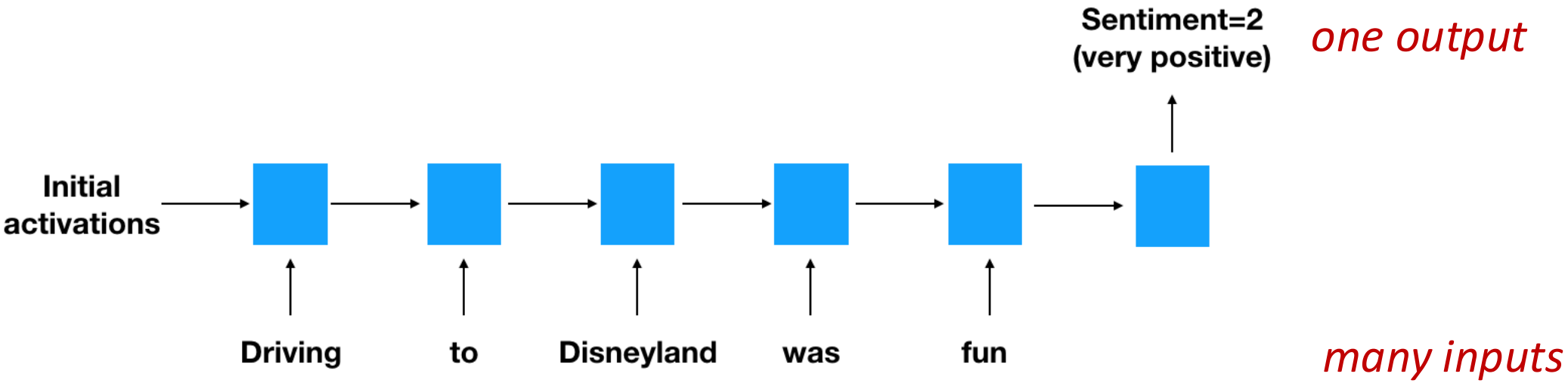
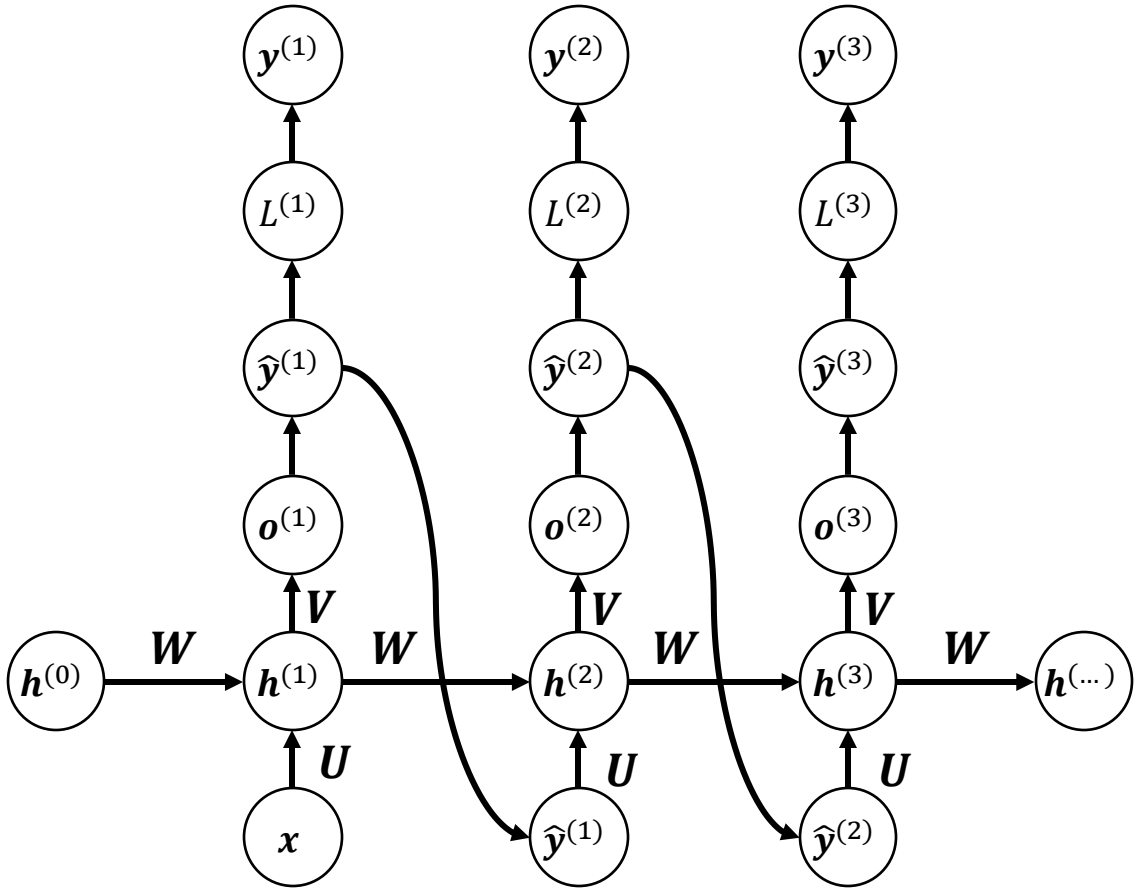
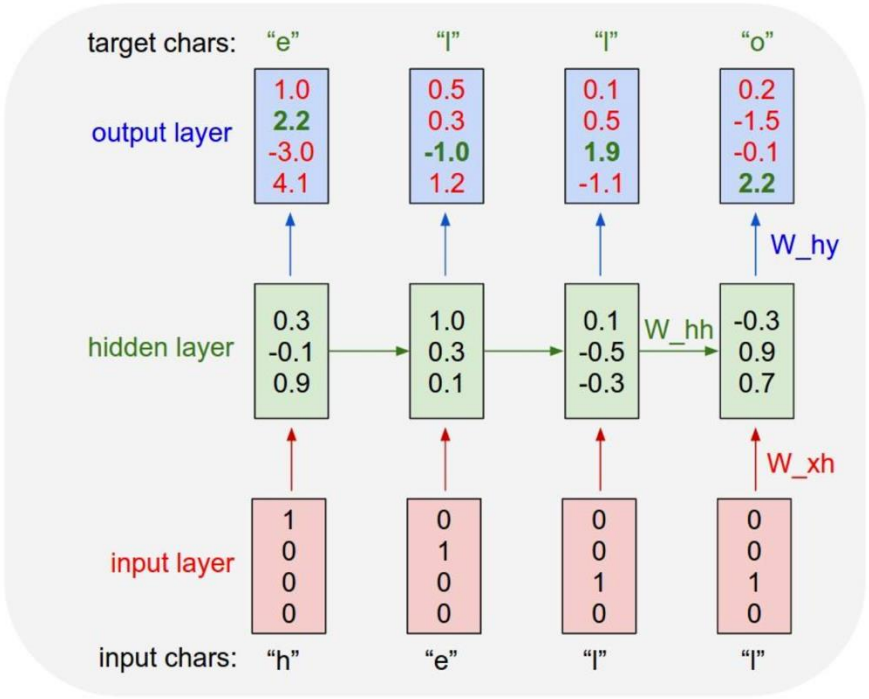


Image source: https://stackoverflow.com/questions/50455556/rnn-with-simultaneous-pos-tagging-and-sentiment-classification

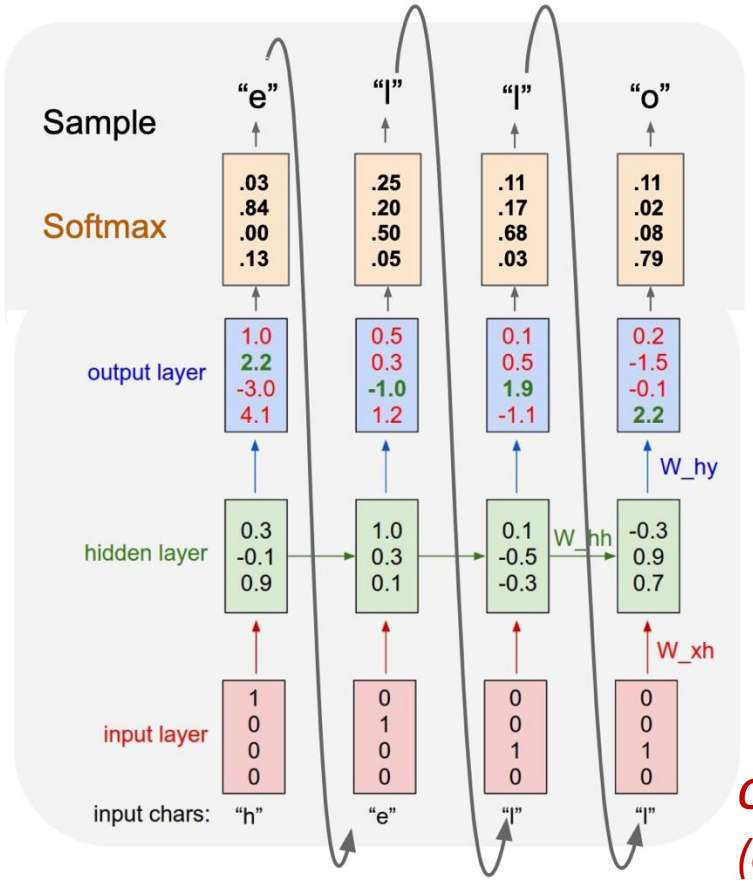
RNN Architecture: One-to-Many



RNN Architecture: One-to-Many



Training phase (many-to-many)



Test phase (one-to-many)

many outputs

one input

(output from previous time step as input)

Image source: Lecture 10, cs231n.

RNN Architecture: One-to-Many

- Application: Image captioning
  - The RNN takes a word, the context from previous time steps and defines a distribution over the next word in the sentence.
  - The RNN is conditioned on the image information at the first time step.
  - “START” and “END” are special tokens.

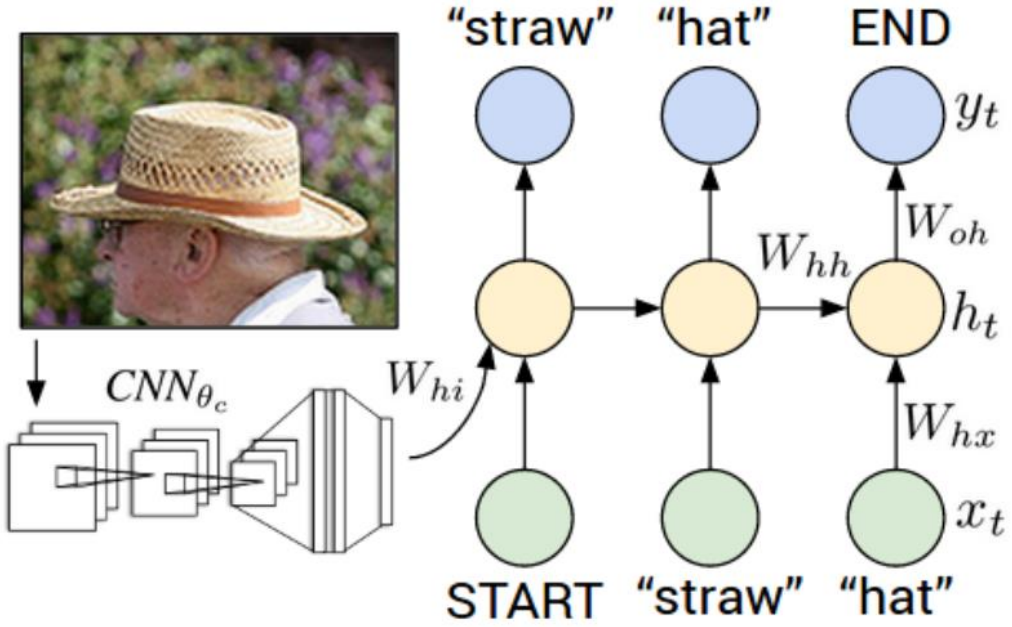
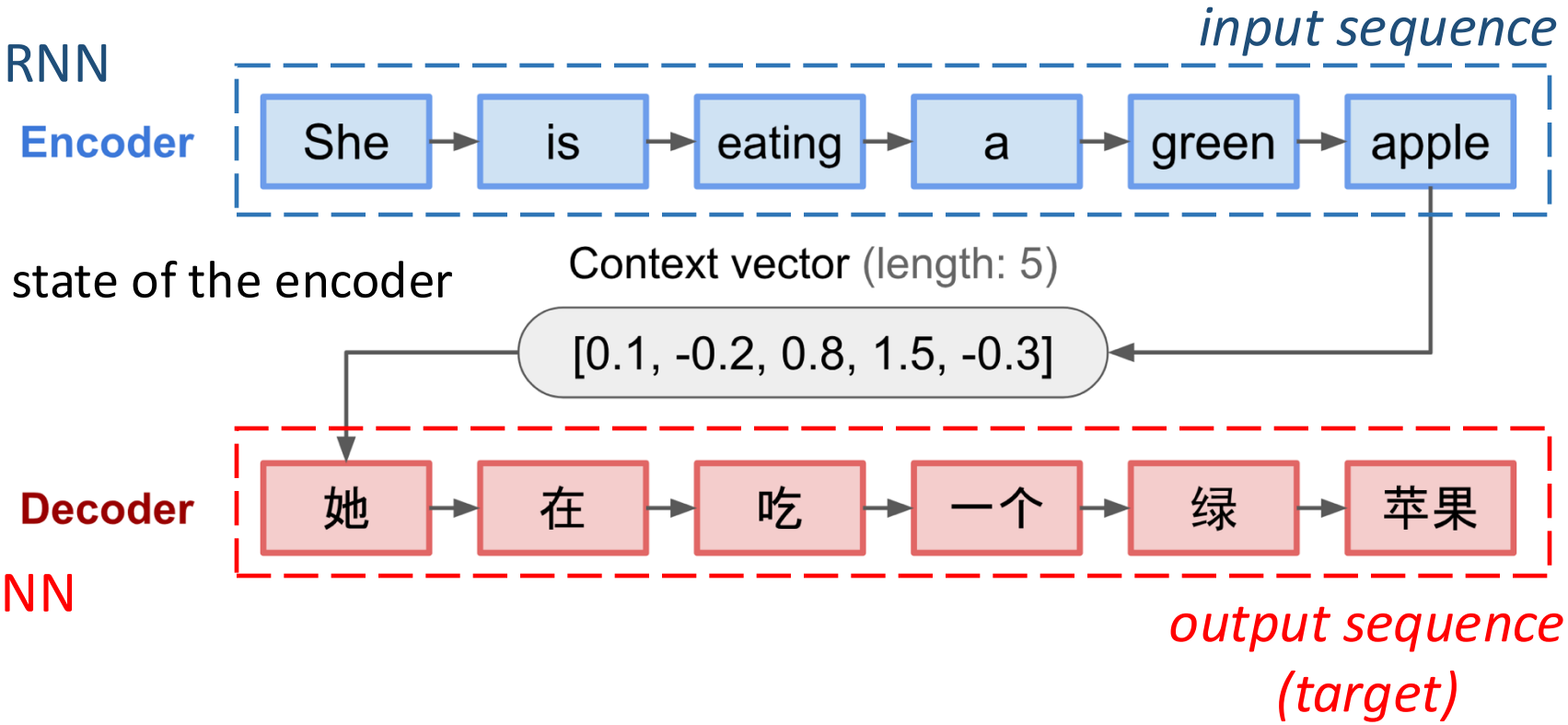


Image source: Karpathy, Andrej, and Li Fei-Fei. "Deep visual-semantic alignments for generating image descriptions." In CVPR. 2015.

Seq2Seq Models

- Application: Neural Machine Translation

Encoder: a many-to-one RNN

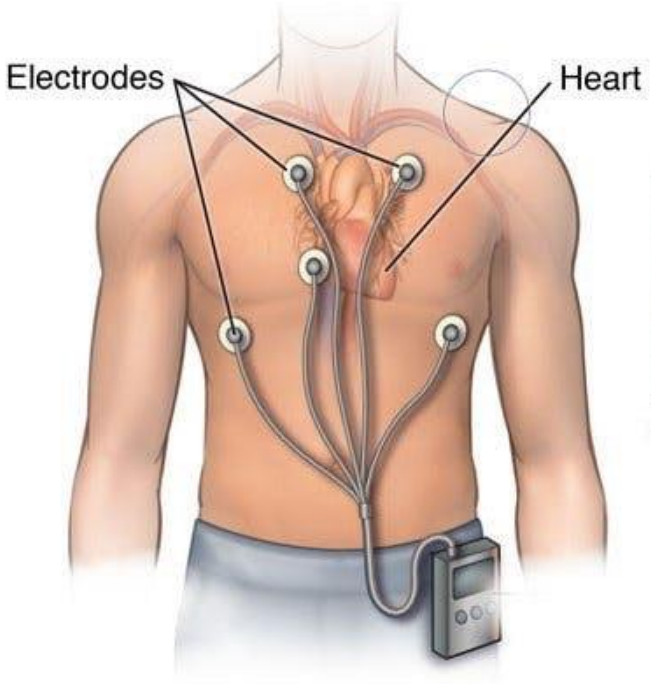


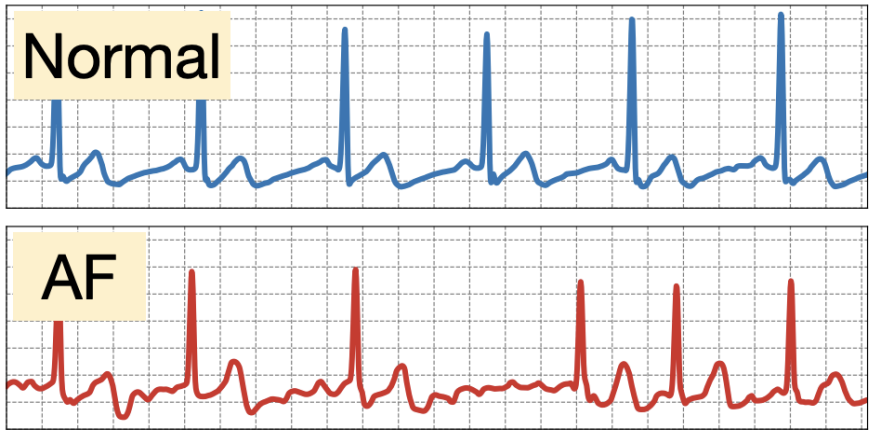
Context vector: the last hidden state of the encoder
(summary of input sequence)

Decoder: a one-to-many RNN

Image source: https://lilianweng.github.io/posts/2018-06-24-attention/

A Real Example: ECG Classification with CNN & RNN

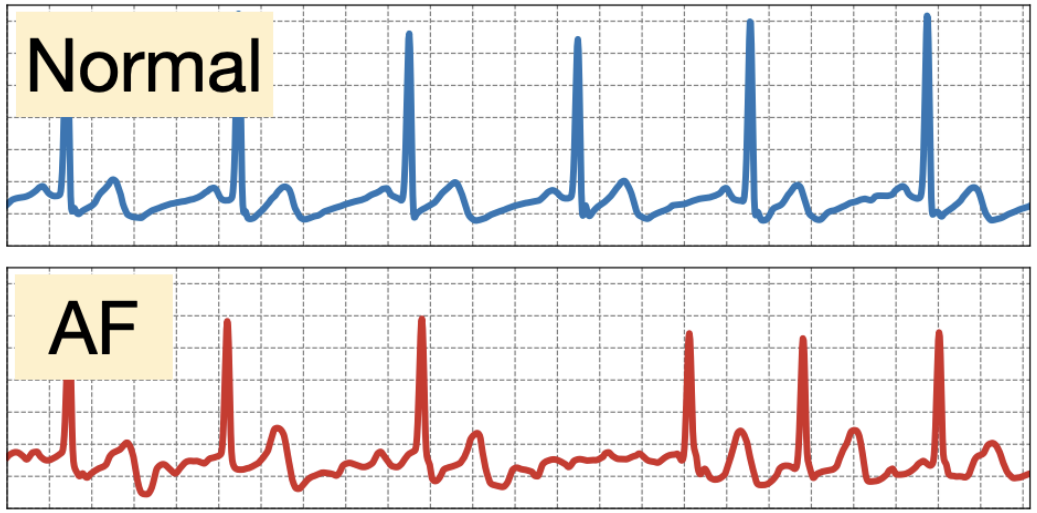




Electrocardiogram (**ECG** or **EKG**) is a recording of the heart's electrical activity.

Hong, Shenda, et al. "MINA: Multilevel Knowledge-Guided Attention for Modeling Electrocardiography Signals." *IJCAI*. 2019.

A Real Example: ECG Classification with CNN & RNN



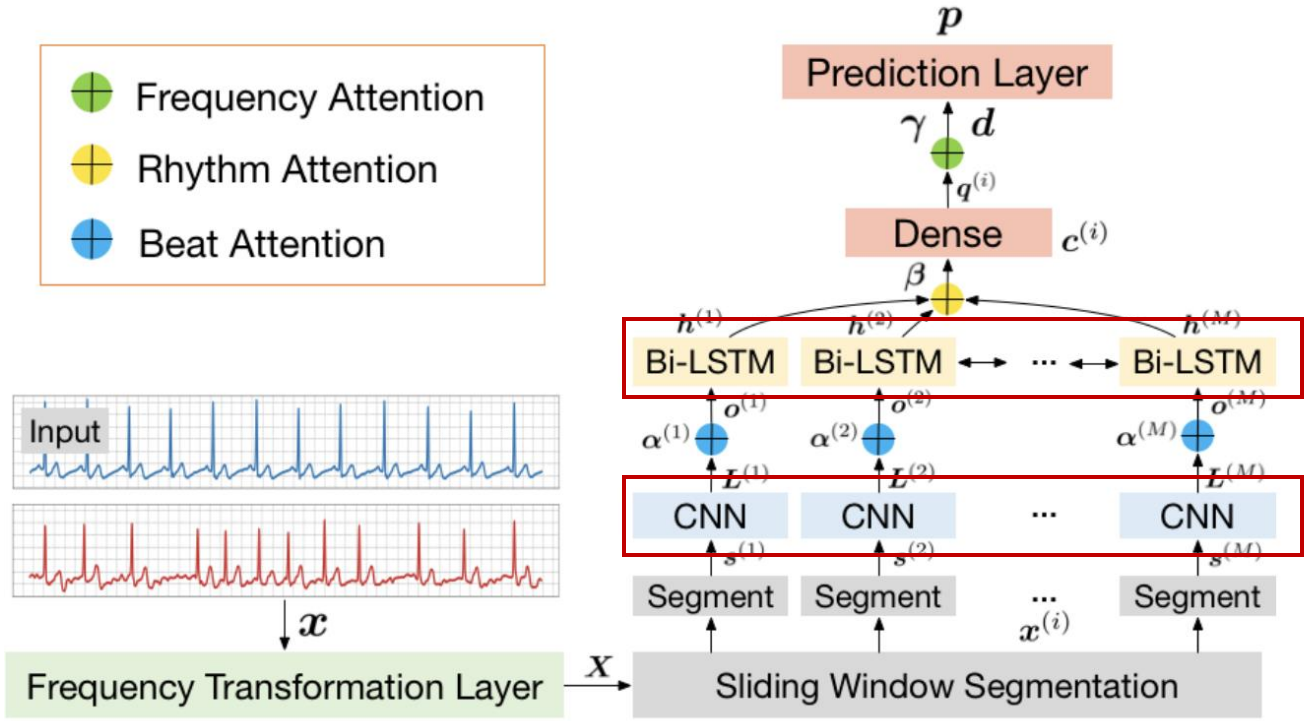
Atrial fibrillation (AF) is an irregular and often very rapid heart rhythm (arrhythmia) that can lead to blood clots in the heart.

Early detection of AF could decrease risk for heart failure and stroke.

Hong, Shenda, et al. "MINA: Multilevel Knowledge-Guided Attention for Modeling Electrocardiography Signals." *IJCAI*. 2019.

A Real Example: ECG Classification with CNN & RNN

A cutting-edge model called MINA (IJCAI-19):



Model the temporal dependency using RNN

Bi-LSTM: a variant of RNN

Extract features using CNN

Hong, Shenda, et al. "MINA: Multilevel Knowledge-Guided Attention for Modeling Electrocardiography Signals." *IJCAI*. 2019.

A Real Example: ECG Classification with CNN & RNN

A cutting-edge model called MINA (IJCAI-19):

	ROC-AUC	PR-AUC	F1
ExpertLR	0.9350 ± 0.0000	0.8730 ± 0.0000	0.8023 ± 0.0000
ExpertRF	0.9394 ± 0.0000	0.8816 ± 0.0000	0.8180 ± 0.0000
CNN	0.8711 ± 0.0036	0.8669 ± 0.0068	0.7914 ± 0.0090
CRNN	0.9040 ± 0.0115	0.8943 ± 0.0111	0.8262 ± 0.0215
ACRNN	0.9072 ± 0.0047	0.8935 ± 0.0087	0.8248 ± 0.0229
MINA	**0.9488** ± 0.0081	**0.9436** ± 0.0082	**0.8342** ± 0.0352

Table 3: Performance Comparison on AF Prediction

Hong, Shenda, et al. "MINA: Multilevel Knowledge-Guided Attention for Modeling Electrocardiography Signals." *IJCAI*. 2019.

Another Example: Cardiologist-level arrhythmia detection

Stanford ML Group

Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network

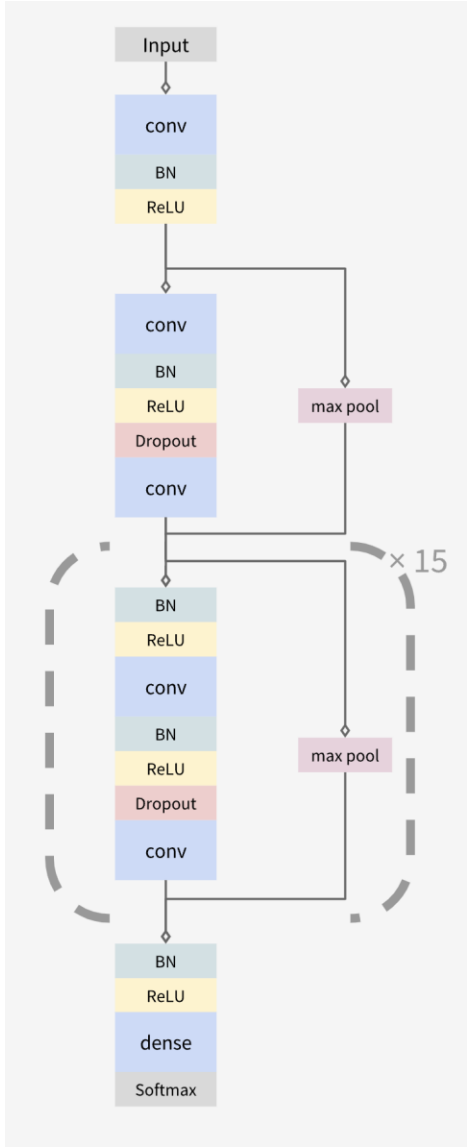
Awni Y. Hannun *, Pranav Rajpurkar *, Masoumeh Haghpanahi *, Geoffrey H. Tison *, Codie Bourn, Mintu P. Turakhia, Andrew Y. Ng

A collaboration between the Stanford Machine Learning Group and iRhythm Technologies

We developed a deep neural network which can diagnose irregular heart rhythms, also known as arrhythmias, from single-lead ECG signals at a high diagnostic performance similar to that of cardiologists.

The electrocardiogram (ECG) is a fundamental tool in the everyday practice of clinical medicine, with more than 300 million ECGs obtained annually worldwide, and is pivotal for diagnosing a wide spectrum of arrhythmias. In a study published in Nature Medicine, we developed a deep neural network to classify 10 arrhythmias as well as sinus rhythm and noise from a single-lead ECG signal, and compared its performance to that of cardiologists.

Can you identify the heart arrhythmia in the above example? The neural network is able to correctly detect AVB_TYPE2. Below, you can see other rhythms which the neural network is successfully able to detect.



1D convolutional deep neural network

https://stanfordmlgroup.github.io/projects/ecg2/

The Problem of Long-Term Dependencies

- The most appealing advantage of RNNs is the ability to connect previous information to the present task.
  - E.g., using previous video frames might inform the understanding of the present frame.
- If the gap is not very far, it seems ok.
  - E.g., the task of next word prediction: “the clouds are in the ___”.

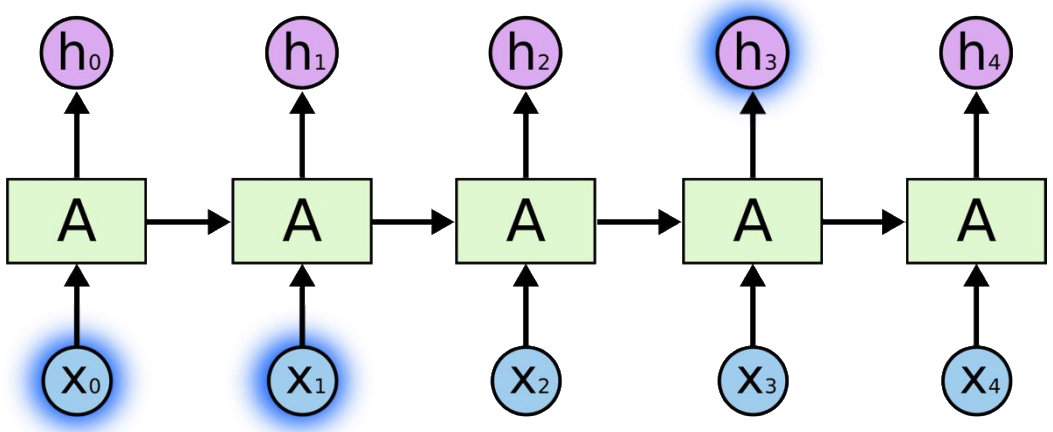


Image source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

The Problem of Long-Term Dependencies

- Unfortunately, as that gap grows, RNNs become unable to learn to connect the information.
  - E.g. “I grew up in China … (300 words)… Of course, I can speak fluent _____.”
- In theory, RNNs are absolutely capable of handling such “long-term dependencies.” But in practice, RNNs have difficulties to learn them.

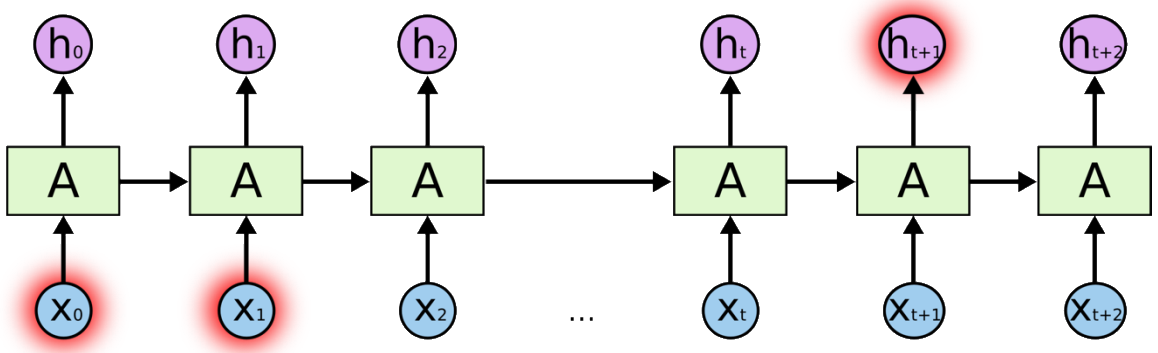


Image source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTMs

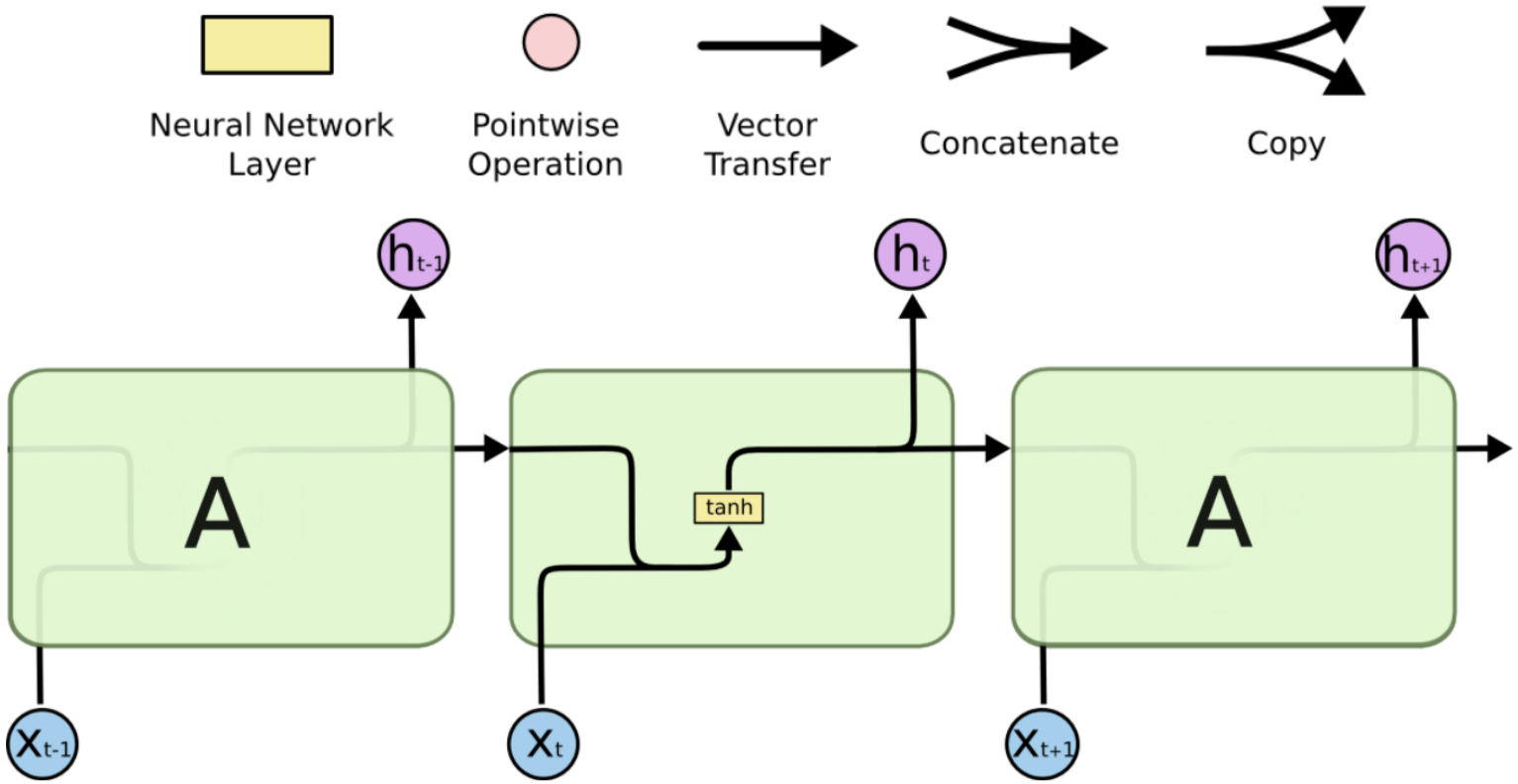
Long short-term memory
S Hochreiter, J Schmidhuber - Neural computation, 1997 - ieeexplore.ieee.org
Learning to store information over extended time intervals by recurrent backpropagation takes a very long time, mostly because of insufficient, decaying error backflow. We briefly review Hochreiter's (1991) analysis of this problem, then address it by introducing a novel …
☆ 99 Cited by 55956 Related articles All 50 versions »

- Long Short Term Memory networks (LSTMs) is a special kind of RNN, explicitly designed for learning long-term dependencies.
- It was proposed in 1997 by Jürgen Schmidhuber, but became popular until the deep learning era.

Jürgen Schmidhuber

Image source: https://www.bloomberg.com/news/features/2018-05-15/google-amazon-and-facebook-owe-j-rgen-schmidhuber-a-fortune

LSTMs



Standard form of vanilla RNNs

Image source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTMs

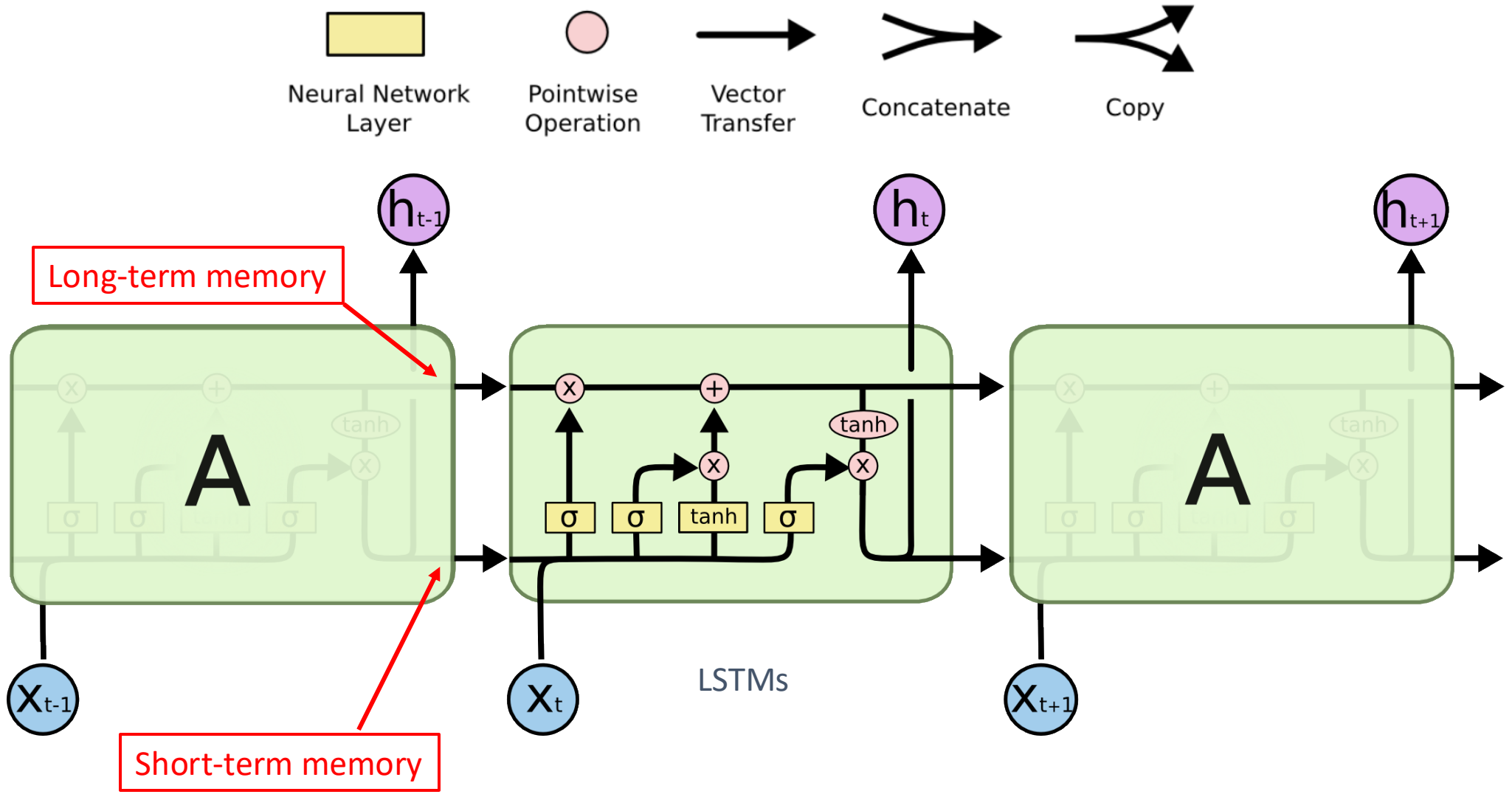


Image source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTMs: Cell State

- The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.
- The cell state is kind of like a conveyor belt.
- It runs straight down the entire chain, with only some minor linear interactions (no concat).
- It's very easy for information to just flow along it unchanged.

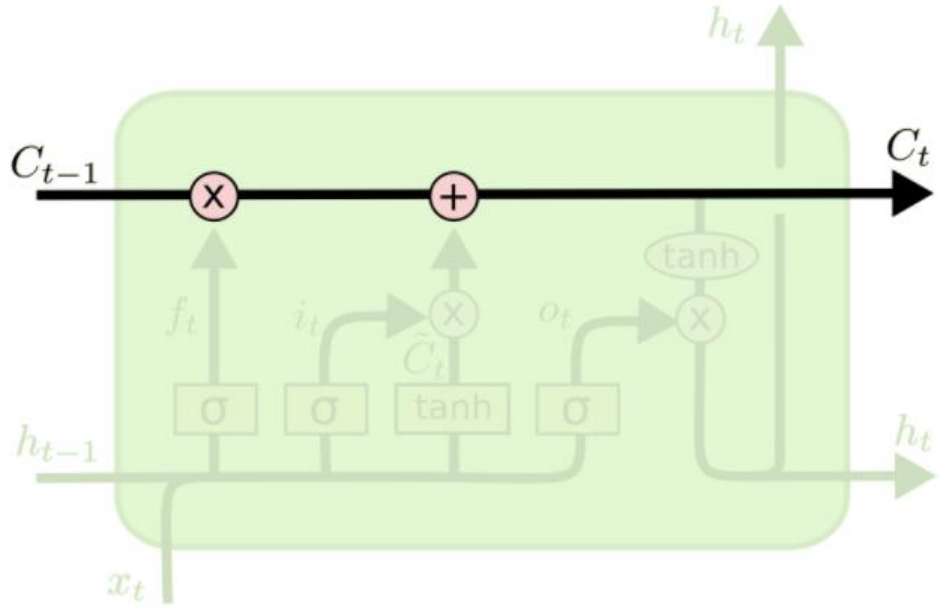
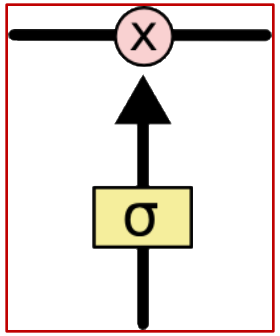


Image source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTMs: Gates

- Gates are a way to optionally let information through.
  - Composed of a sigmoid neural net layer and a pointwise multiplication operation.
- The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through.
  - A value of zero means “let nothing through,” while a value of one means “let everything through!”
- An LSTM has three of these gates, to control and exploit the cell state.
  - Forget gate.
  - Input gate.
  - Output gate.



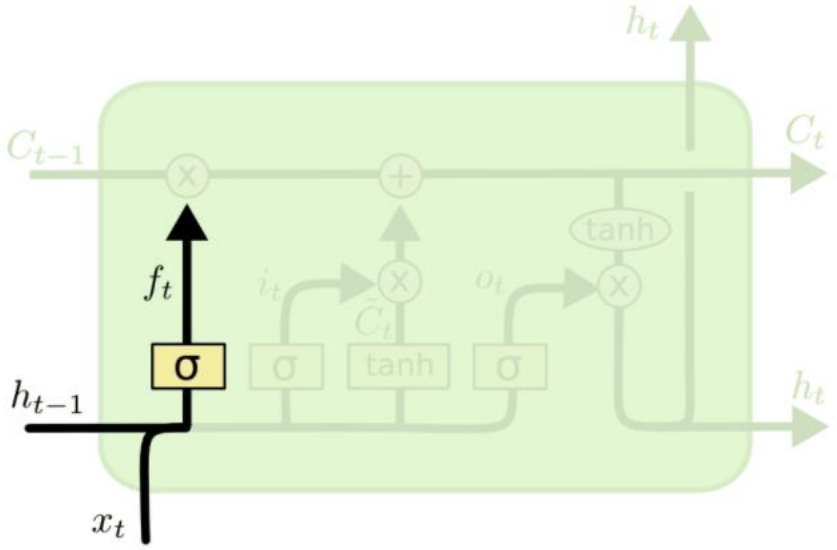
A gate in LSTMs

$$output = \sigma(W \cdot input + b)$$

Image source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTMs: Forget Gate

- Decide what information we're going to throw away from the cell state.
  - The output of sigmoid layer is between 0 and 1, and multiplied to each number in the cell state C_{t-1}.
- Example
  - The cell state might include the gender of the present subject, so that the correct pronouns can be used.
  - When we see a new subject, we want to forget the gender of the old subject.



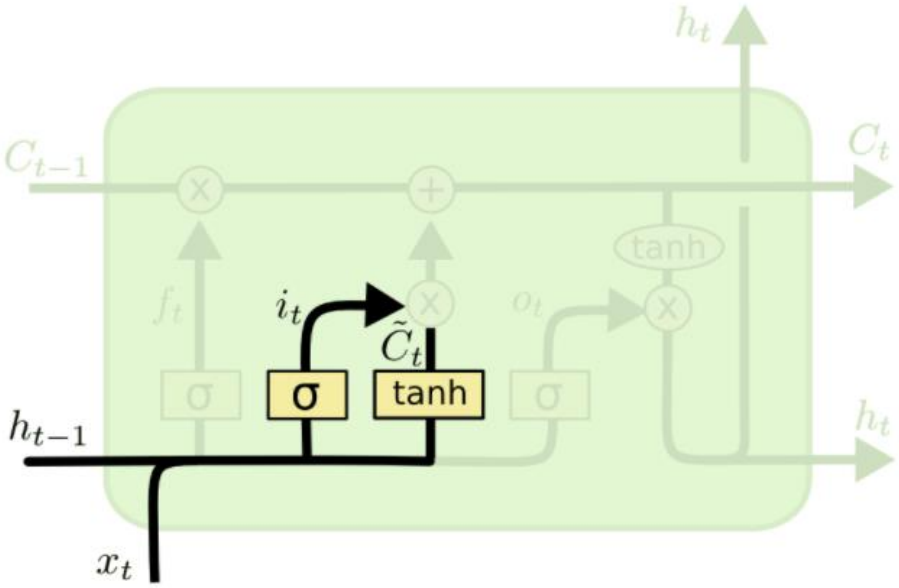
Forget gate

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

Image source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTMs: Input Gate

- Decide what new information we're going to store in the cell state.
  - A sigmoid layer decides which values we'll update.
  - A tanh layer creates a vector of new candidate values.
- Example
  - Add the gender of the new subject to the cell state, to replace the old one we're forgetting.



Input gate

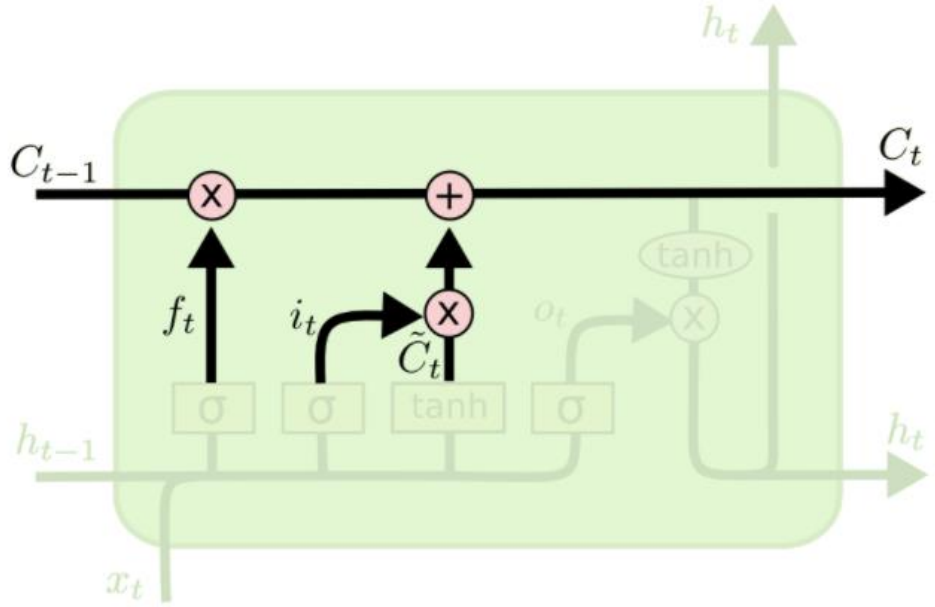
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\widetilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Image source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTMs: Forget and Input Gate

- Update the old cell state, C_{t-1}, into the new cell state C_t.
  - Multiply the old state by f_t, forgetting the things we decided to forget earlier.
  - Add new information, which is transformed by $\tilde{C}_t$ and selected by i_t.

- Example
  - Drop the information about the old subject's gender and add the new information.



Update cell state

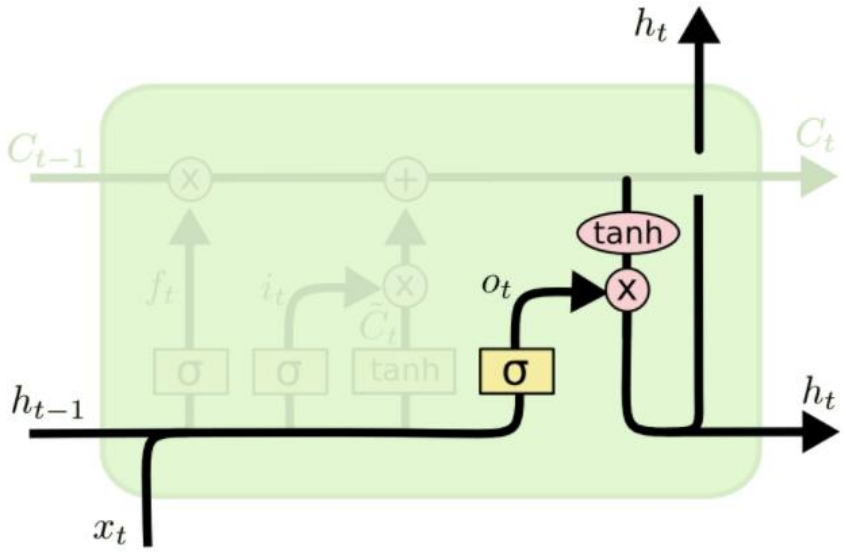
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

Image source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTMs: Output Gate

- Decide what we're going to output.
  - Run a sigmoid layer which decides what parts of the cell state we're going to output.
  - Put the cell state through tanh and multiply it by the output of the sigmoid layer.

- Example
  - It integrates the information of the new subject (e.g. singular or plural) with the scene or environment stored in the cell state to predict a coming relative verb.



Output gate

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t \odot \tanh(C_t)$$

Image source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

GRU

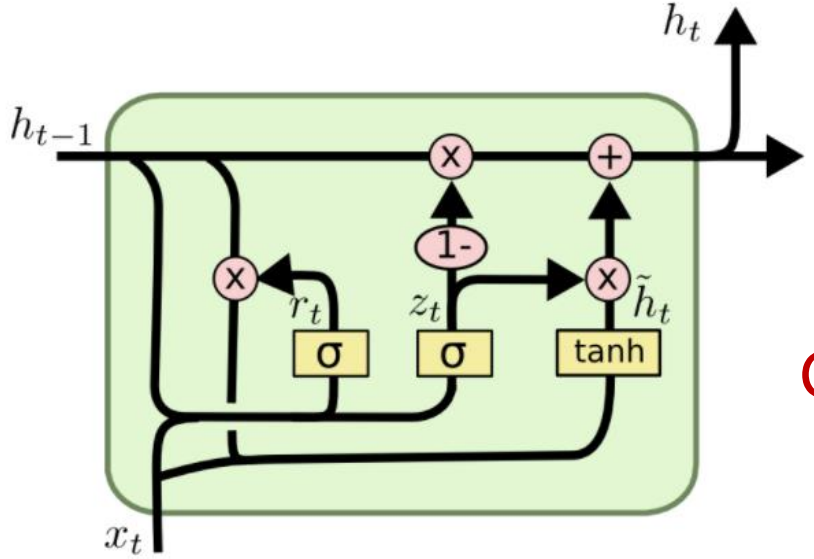
Learning phrase representations using RNN encoder-decoder for statistical machine translation

K Cho, B Van Merriënboer, C Gulcehre… - arXiv preprint arXiv …, 2014 - arxiv.org

In this paper, we propose a novel neural network model called RNN Encoder-Decoder that consists of two recurrent neural networks (RNN). One RNN encodes a sequence of symbols into a fixed-length vector representation, and the other decodes the representation into another sequence of symbols. The encoder and decoder of the proposed model are jointly trained to maximize the conditional probability of a target sequence given a source sequence. The performance of a statistical machine translation system is empirically found …

☆ 99 Cited by 11485 Related articles All 31 versions »

- Gated Recurrent Unit (GRU) is a variant of LSTM.
- GRU performs similarly to LSTM but is computationally cheaper.
- Merges the cell state and hidden state.
- Combine the forget and input gates into a single update gate.



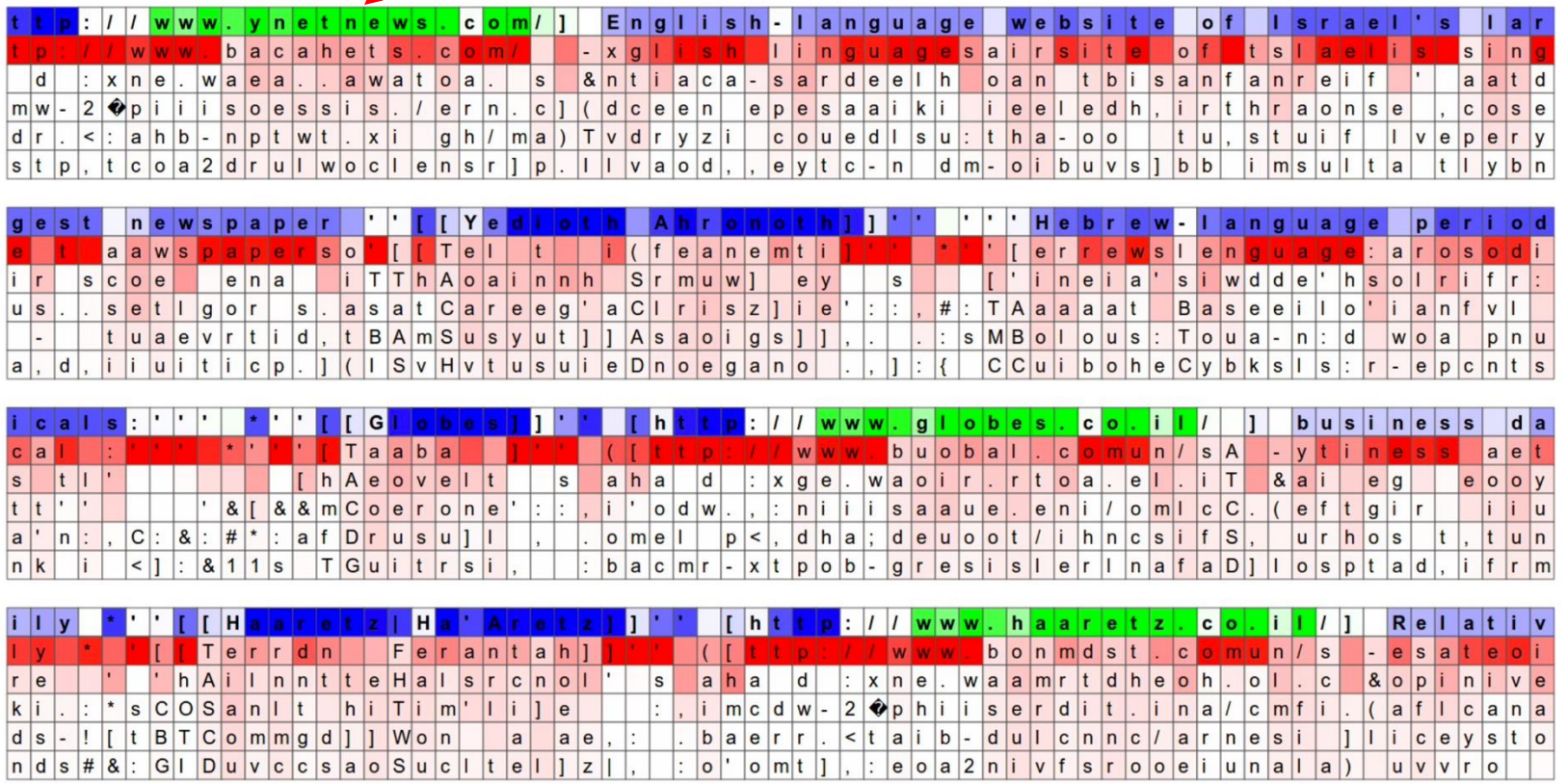
GRU

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$
$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$
$$\tilde{h}_t = \tanh(W \cdot [r_t \odot h_{t-1}, x_t])$$
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

Image source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Example

The LSTM is likely using this neuron to remember if it is inside a URL or not.

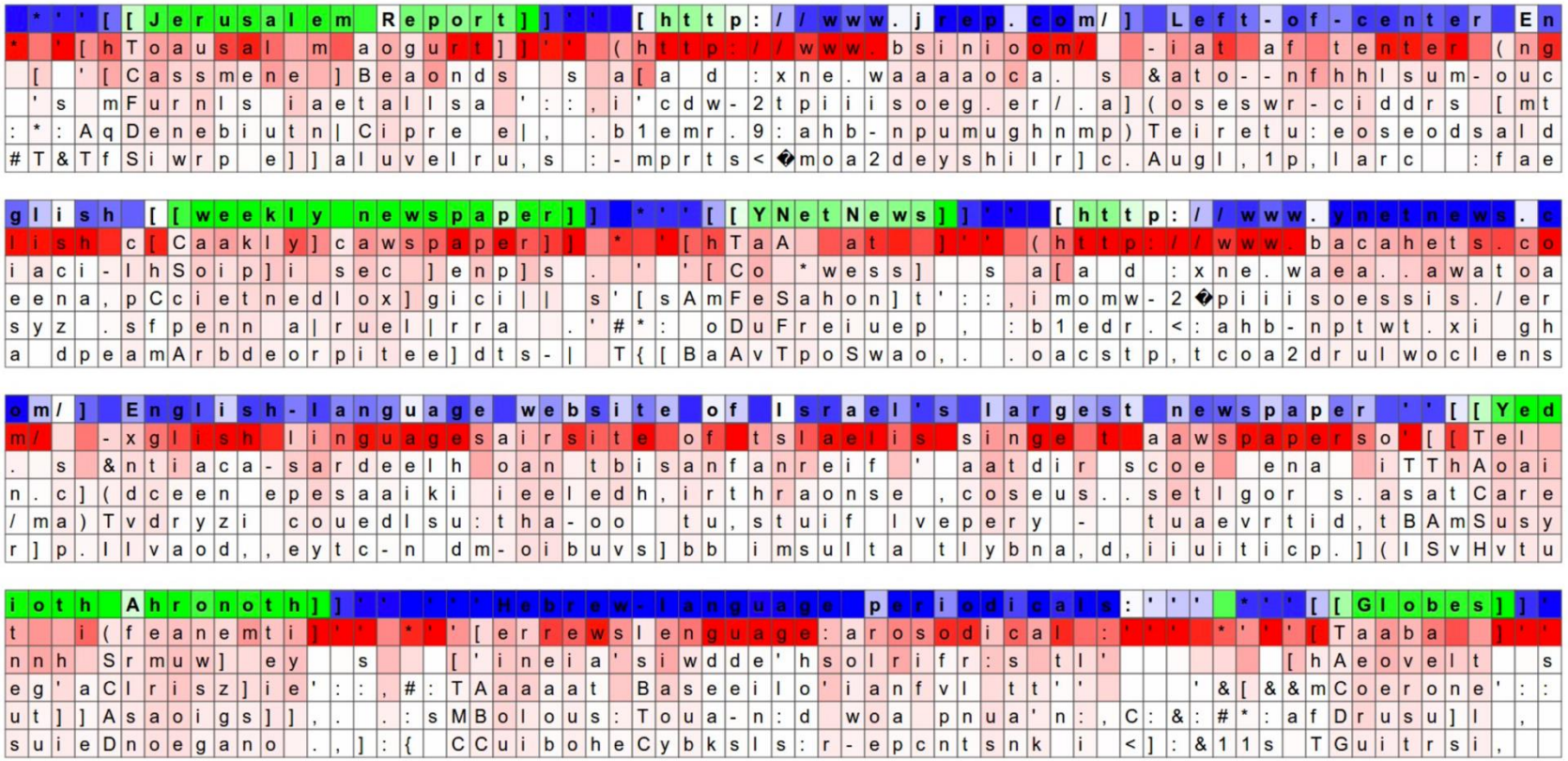


Next character prediction by LSTM. Red: top 5 prediction probability; Blue: negative value of some neuron in cell state; Green: positive value of some neuron in cell state.

Image source: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Example

The highlighted neuron here gets very activated when the RNN is inside the [[ ]] markdown environment and turns off outside of it.



Next character prediction by LSTM. Red: top 5 prediction probability; Blue: negative value some neuron in cell state; Green: positive value of some neuron in cell state.

Image source: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

A Brief Introduction to Modern Language Modeling

NLP Tasks using Language Model

- Sentiment analysis

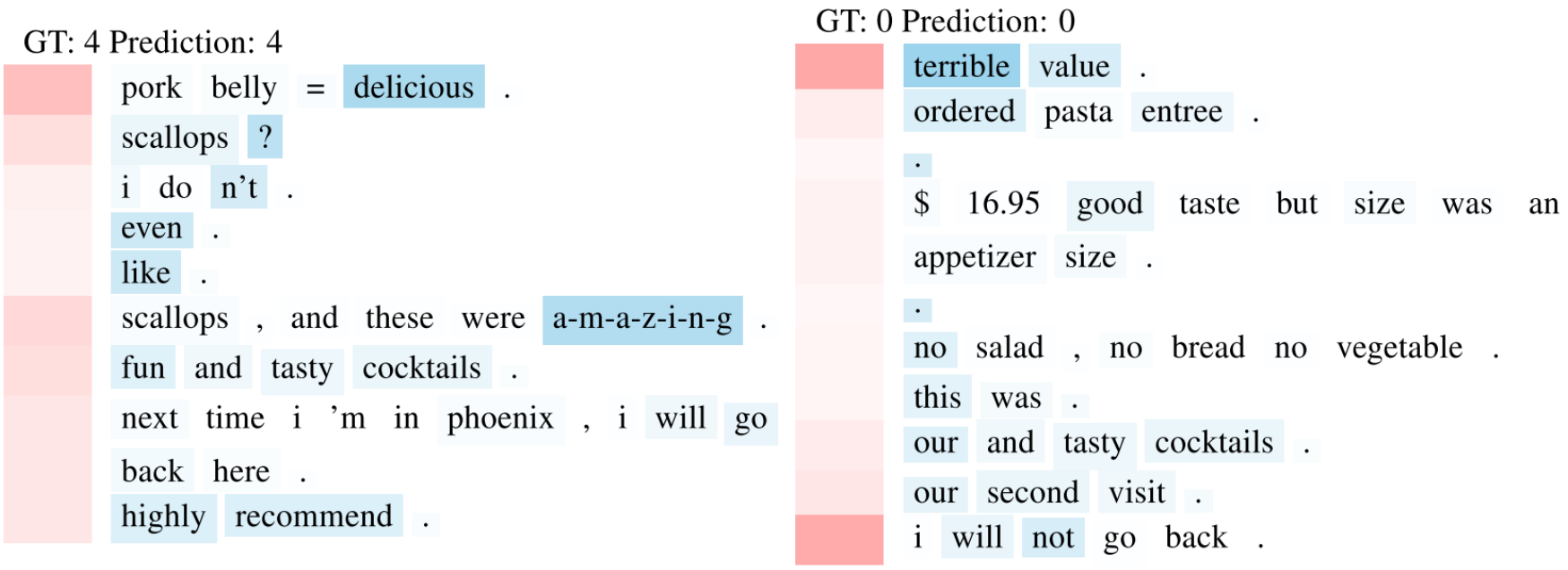
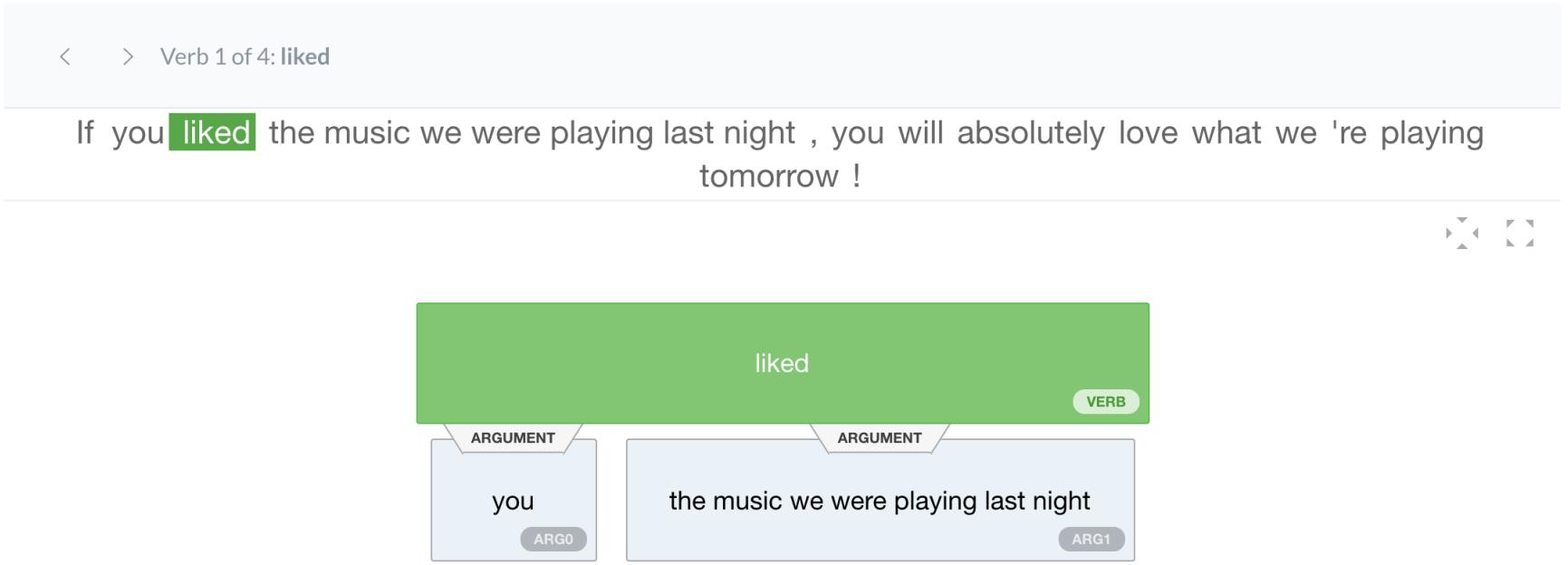


Image source: Yang, Zichao, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. "Hierarchical attention networks for document classification." In Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies, pp. 1480-1489. 2016.

NLP Tasks using Language Model

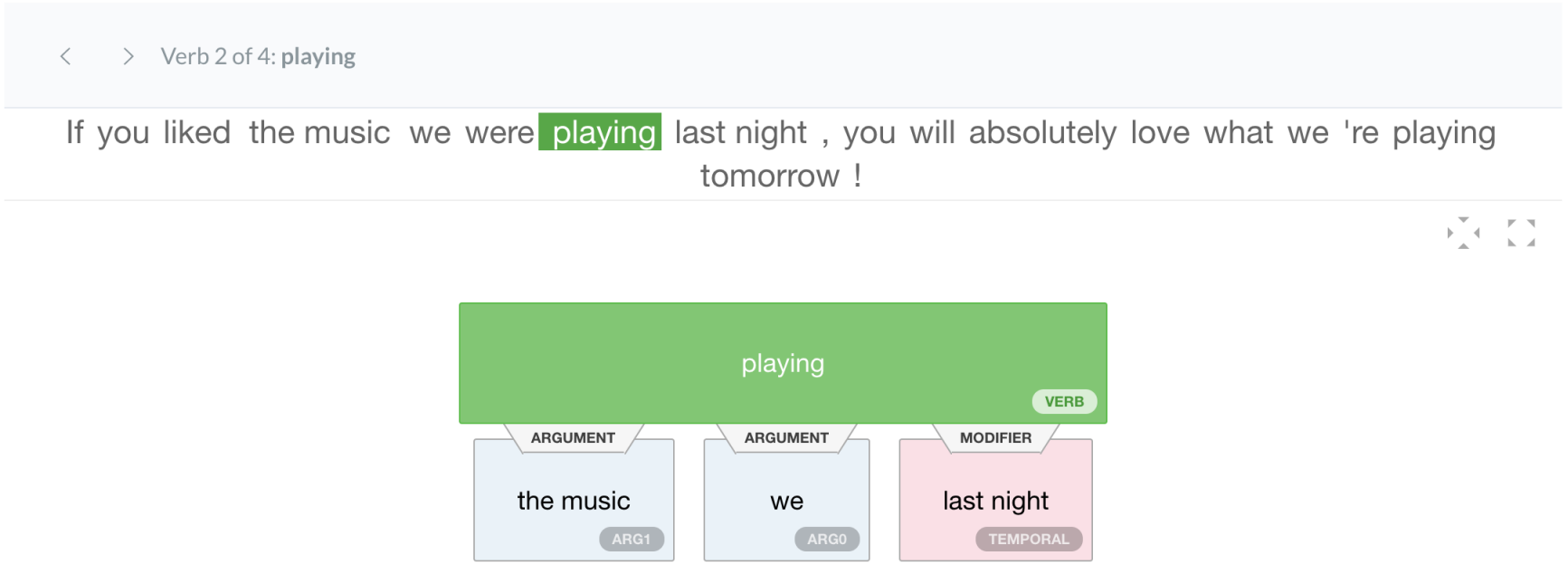
- Semantic role labeling



Source: https://demo.allennlp.org/semantic-role-labeling/MjQ1MjQxOA==

NLP Tasks using Language Model

- Semantic role labeling



Source: https://demo.allennlp.org/semantic-role-labeling/MjQ1MjQxOA==

NLP Tasks using Language Model

- Coreference resolution

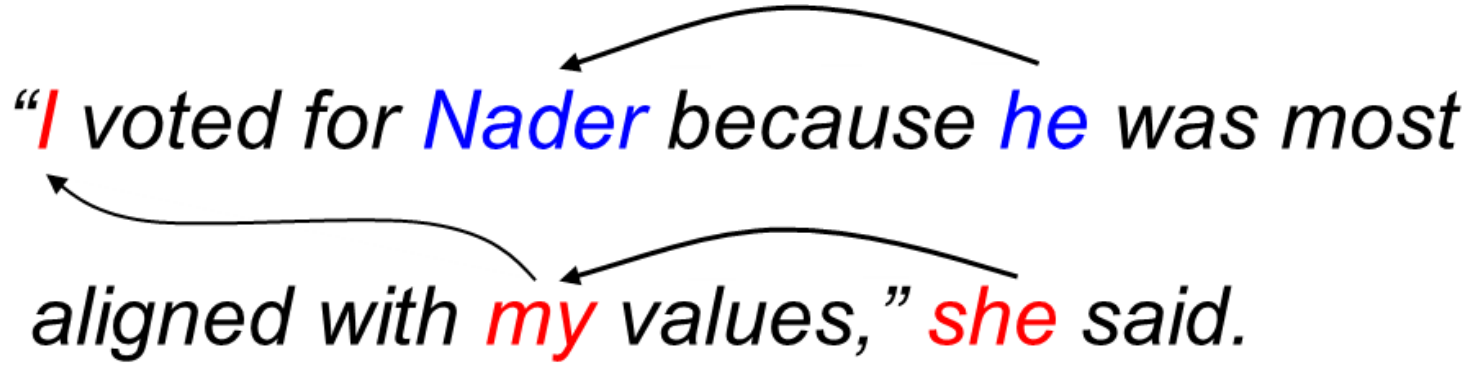


Image source: https://nlp.stanford.edu/projects/coref.shtml#:~:text=Coreference%20resolution%20is%20the%20task,question%20answering%2C%20and%20information%20extraction.

NLP Tasks using Language Model

- Named entity recognition (NER)

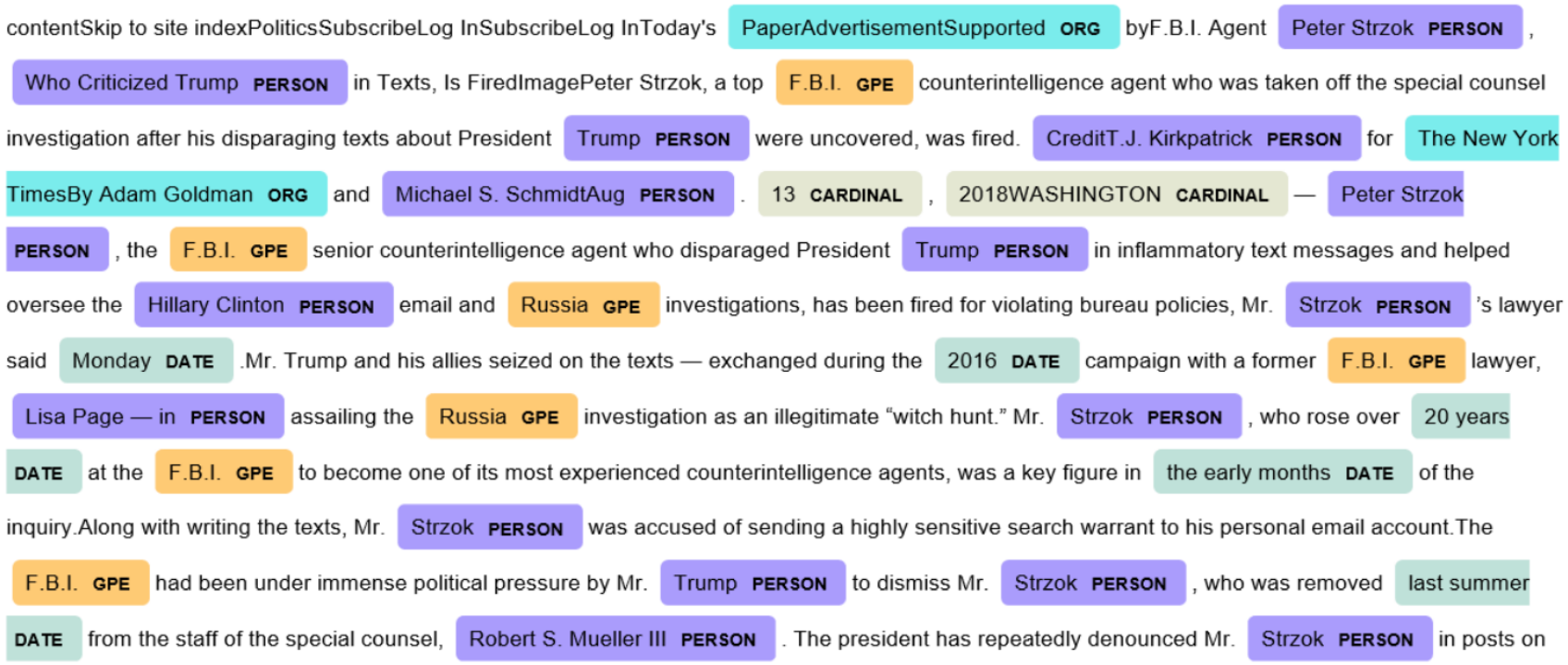


Image source: https://towardsdatascience.com/named-entity-recognition-and-classification-with-scikit-learn-f05372f07ba2

NLP Tasks using Language Model

- Machine translation

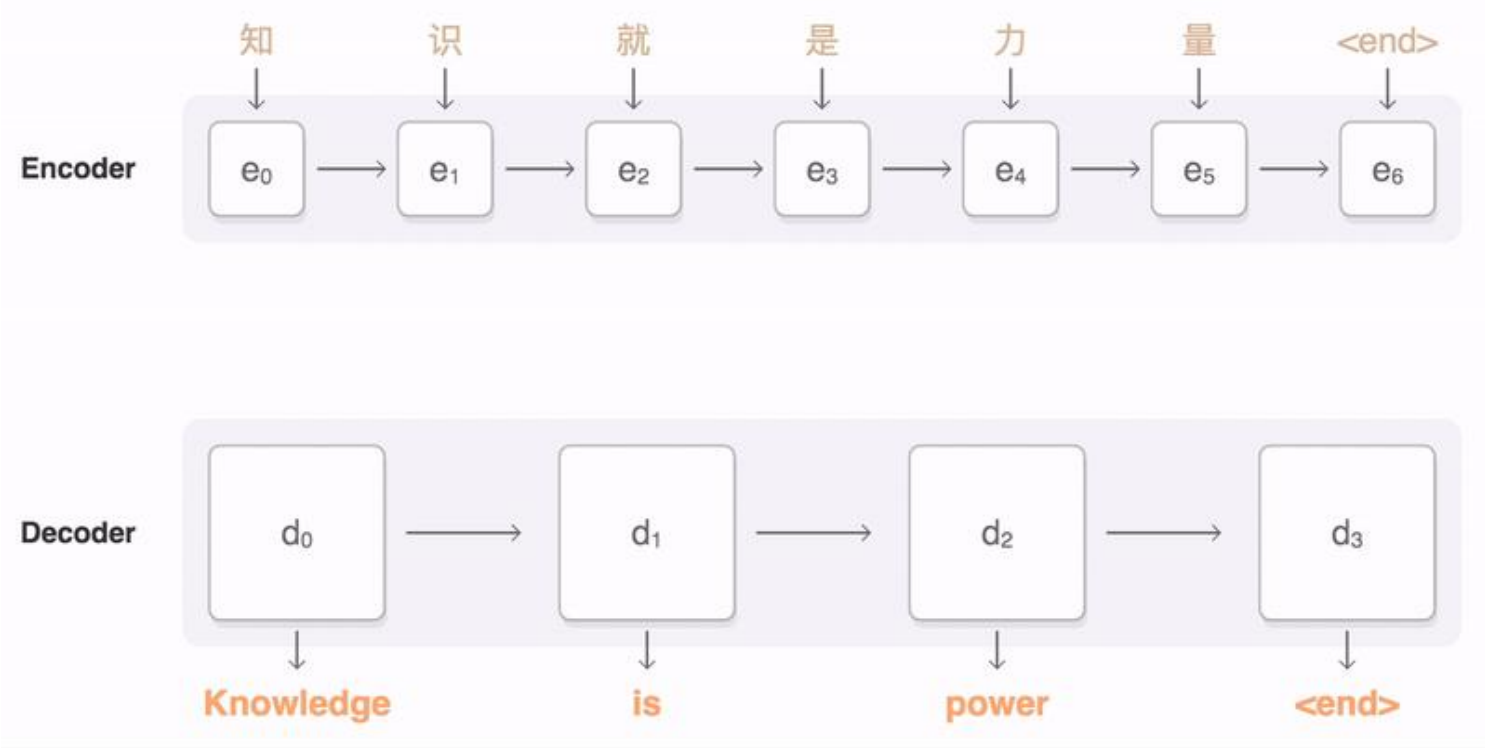


Image source: https://google.github.io/seq2seq/

Natural Language Processing

- "NLP is the crown jewel of Artificial Intelligence".
- It is very hard to make AI understand underlying meaning of human language.
- Among lots of problems, ambiguity is one of NLP's nightmares.

Image sources:
- https://examples.yourdictionary.com/reference/examples/examples-of-ambiguity.html
- https://www.pinterest.com/pin/113856696802262153/

Meaning of a Word

- How can computer know the meaning of a word?
- Use e.g., WordNet, a thesaurus containing lists of synonym sets and hypernyms ("is a" relationships).

```
from nltk.corpus import wordnet as wn
poses = { 'n':'noun', 'v':'verb', 's':'adj (s)', 'a':'adj', 'r':'adv'}
for synset in wn.synsets("good"):
    print("{}: {}".format(poses[synset.pos()],
            ", ".join([l.name() for l in synset.lemmas()])))
```

```
noun: good
noun: good, goodness
noun: good, goodness
noun: commodity, trade_good, good
adj: good
adj (sat): full, good
adj: good
adj (sat): estimable, good, honorable, respectable
adj (sat): beneficial, good
adj (sat): good
adj (sat): good, just, upright
…
adverb: well, good
adverb: thoroughly, soundly, good
```

Synonym of "good"

```
from nltk.corpus import wordnet as wn
panda = wn.synset("panda.n.01")
hyper = lambda s: s.hypernyms()
list(panda.closure(hyper))
```

```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
Synset('living_thing.n.01'),
Synset('whole.n.02'),
Synset('object.n.01'),
Synset('physical_entity.n.01'),
Synset('entity.n.01')]
```

Hypernyms of "panda"

Code source: Lecture 1, cs224n

Meaning of a Word

Problems of using dictionary library:

- Great as a resource but missing slight difference between words.
  - E.g., “proficient” is listed as a synonym for “good”. This is only correct in some contexts.
- Different meanings depending on the context.
- Missing new meanings of words, or new created words.
  - E.g., badass, lmao, skr, xswl…
  - Impossible to keep up-to-date!
- Requires human labor to create and adapt.

Meaning of a Word

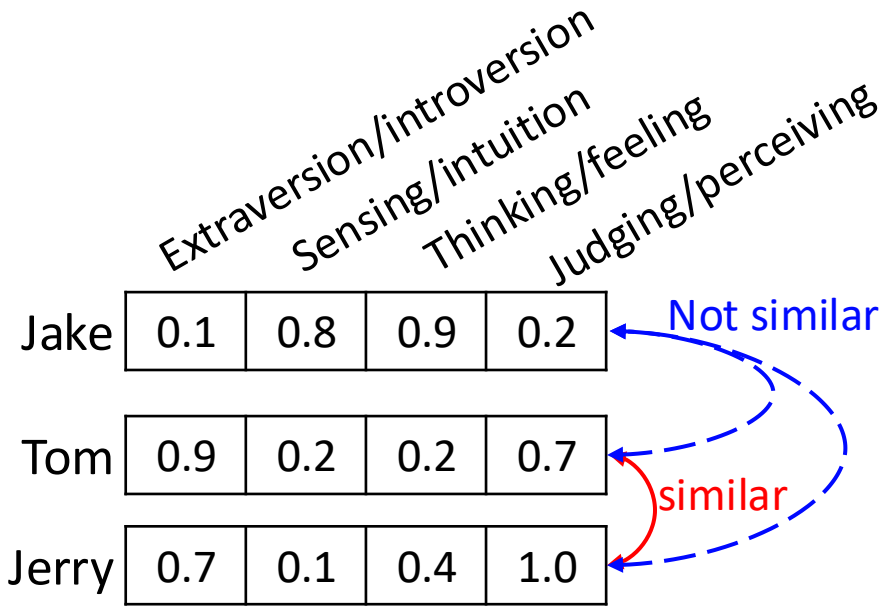
- In traditional NLP, we regard words as discrete symbols.
- Words can be represented by one-hot vectors:

Cat:	[0,0,0,0,0,0,…,1,0,0]
Dog:	[1,0,0,0,0,0,…,0,0,0]
Car:	[0,0,0,0,1,0,…,0,0,0]

- The length of the vector equals to the size of the corpus (e.g. 500,000).
- Problem: the distance between any pair of words are 1, except itself.
  - There is no natural notion of similarity for one-hot vectors.
- Solution: learn to encode similarity in the vectors themselves.

An Analogy: "Personality Embedding"

- One-hot representation of persons:
  - Jake: [0,1,0,0,0,0,…,0]
  - Tom: [1,0,0,0,0,0,…,0]
  - Jerry: [0,0,0,1,0,0,…,0]
- How to know who is similar with whom?
- We can ask for their MBTI:
  - We can compute distance between the vectors to measure their similarity!
  - "Semantics" of persons.



Example inspired by https://jalammar.github.io/illustrated-word2vec/

Word Vectors

- Build a dense vector for each word, chosen so that it is similar to vectors of words that appear in similar contexts.
  - Word vectors are sometimes called word embeddings or word representations. They are a distributed representation.
- Word vectors with small distance have the close meaning.

Cat:	[0.1,0.7,0.9,0.1,0.1]
Dog:	[0.2,0.7,0.8,0.2,0.1]
Car:	[0.9,0.1,0.5,0.6,0.8]

  - Usually hundreds of dimensions.

- However, there is no label to train these word embeddings in a supervised manner.
  - It is impossible to label the similarity between any two words.

Contextual Information

- Distributional semantics: words that are used and occur in the same contexts tend to purport similar meanings.
  - "A word is characterized by the company it keeps" was popularized by J. R. Firth, an English linguist, in the 1950s.
- When a word w appears in a text, its context is the set of words that appear nearby (within a fixed-size window).
- Use the many contexts of w to build up a representation of w.

…government debt problems turning into	***banking***	*crises as happened in 2009…*
…saying that Europe needs unified	***banking***	*regulation to replace the hodgepodge…*
…India has just given its	***banking***	*system a shot in the arm…*

"Banking" is represented by its context words

Word2vec

Distributed representations of words and phrases and their compositionality
T Mikolov, I Sutskever, K Chen, GS Corrado… - Advances in neural …, 2013 - papers.nips.cc
The recently introduced continuous Skip-gram model is an efficient method for learning high-quality distributed vector representations that capture a large number of precise syntactic and semantic word relationships. In this paper we present several improvements that make the Skip-gram model more expressive and enable it to learn higher quality vectors more rapidly. We show that by subsampling frequent words we obtain significant speedup, and also learn higher quality representations as measured by our tasks. We also introduce …
☆ 99 Cited by 23362 Related articles All 46 versions »

Idea:

- Every word in a fixed vocabulary is represented by a dense vector.
- Go through each position t in the text, which has
  - a center word c,
  - context words o.
- Use the similarity of the word vectors for c and o to calculate the probability of c given o (or vice versa).
- Keep adjusting the word vectors to maximize this probability.

Word2vec

- The authors proposed Skip-gram model to train word vectors.
- Given the center word c, predict the context words o.

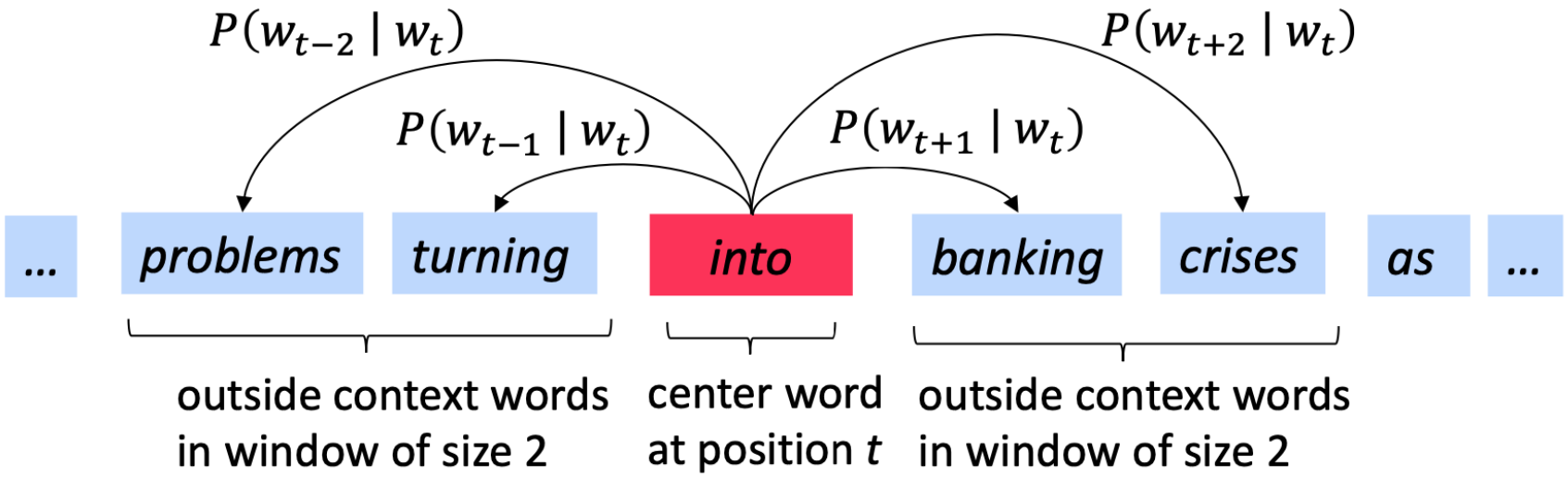


Image source: Lecture 1, cs224n

Word2vec

- The objective function $Loss(\theta)$ is the negative log likelihood:

$$Loss(\theta) = -\frac{1}{T}\log L(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\le j\le c, j\neq 0}\log p\left(w_{t+j}\middle|w_t;\theta\right).$$

- The probability is calculated by:

$$p(o|c) = \frac{\exp(\boldsymbol{w}_o'^{T}\boldsymbol{w}_c)}{\sum_{u\in V}\exp(\boldsymbol{w}_u'^{T}\boldsymbol{w}_c)}.$$

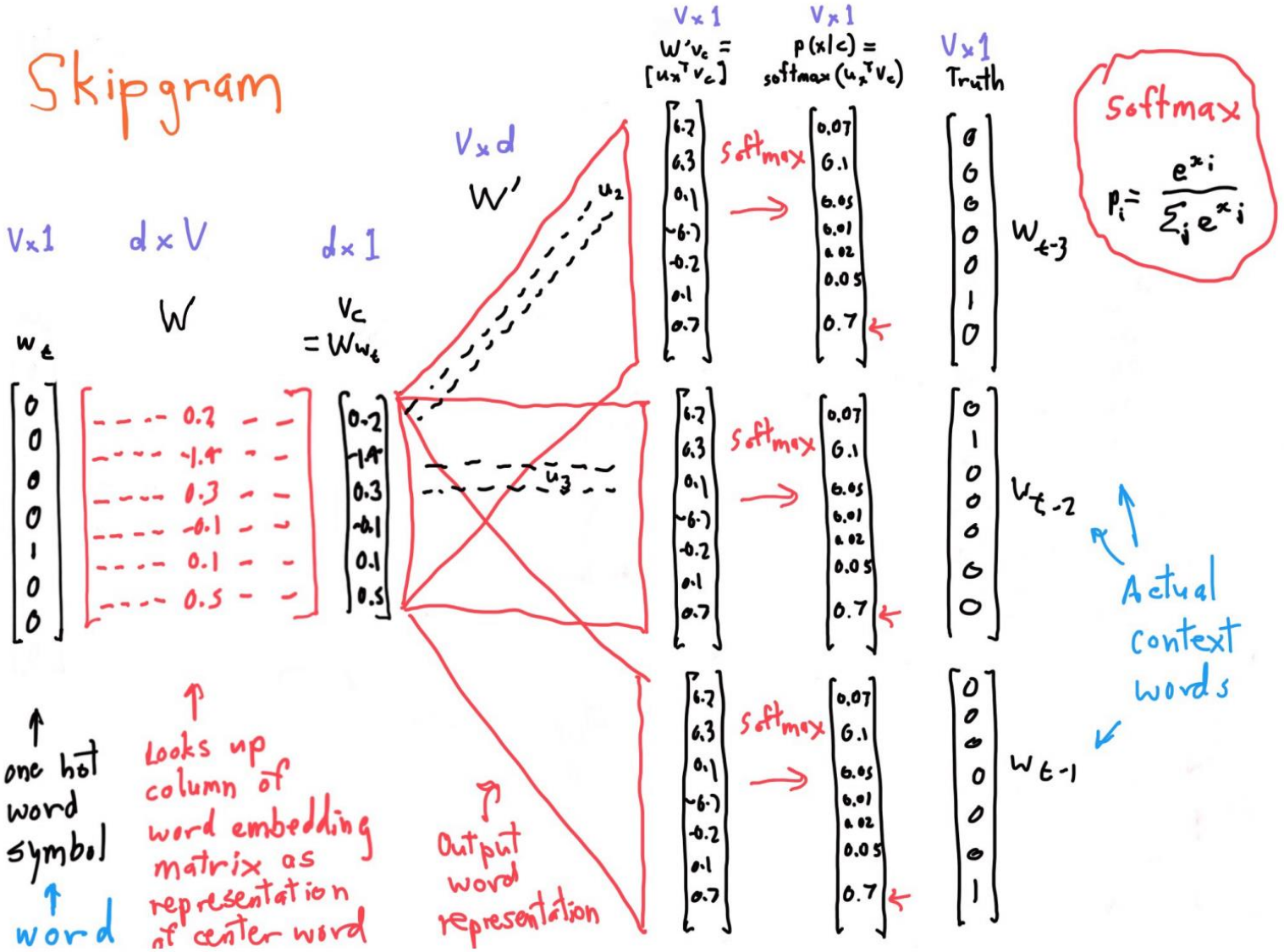
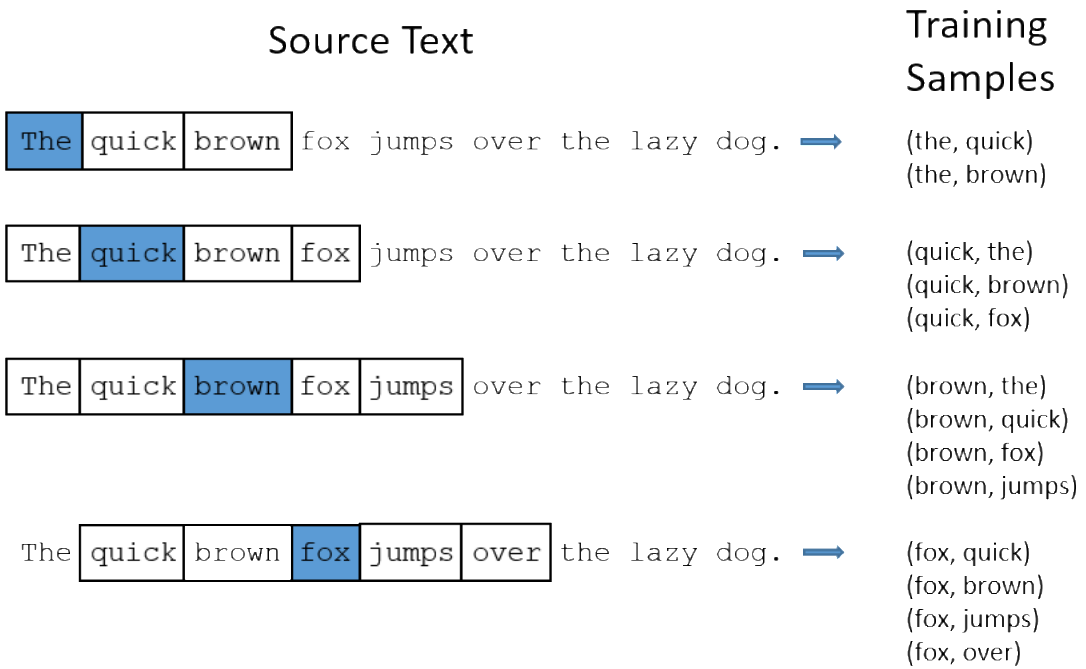


Image source: Lecture 2, cs224n

Word2vec

- In essence, Word2vec uses supervised manner to train word vectors.
  - The center word is the input, the context words are its labels.
  - Also called “self-supervised”

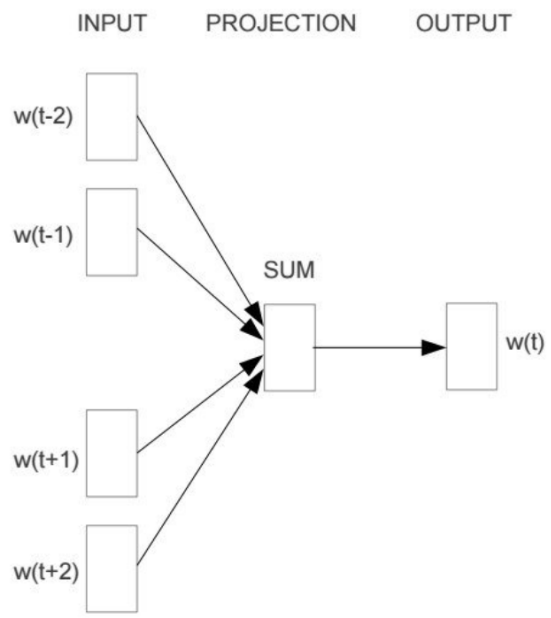


https://remykarem.github.io/word2vec-demo/

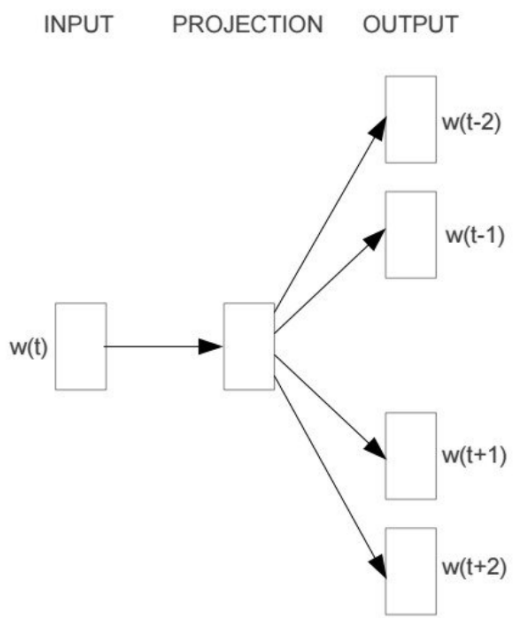
Image source: https://mubaris.com/posts/word2vec/

Word2vec

- Skip-gram model uses center word to predict context words.
- We can also use the context words to predict the center word. This model is called CBOW (Continuous Bag of Words).



CBOW



Skip-gram

Image source: https://mubaris.com/posts/word2vec/

Word Vectors

- By using word vectors, we can "calculate their meaning":

$$w['\text{king}'] \approx w['\text{queen}'] - w['\text{woman}'] + w['\text{man}']$$

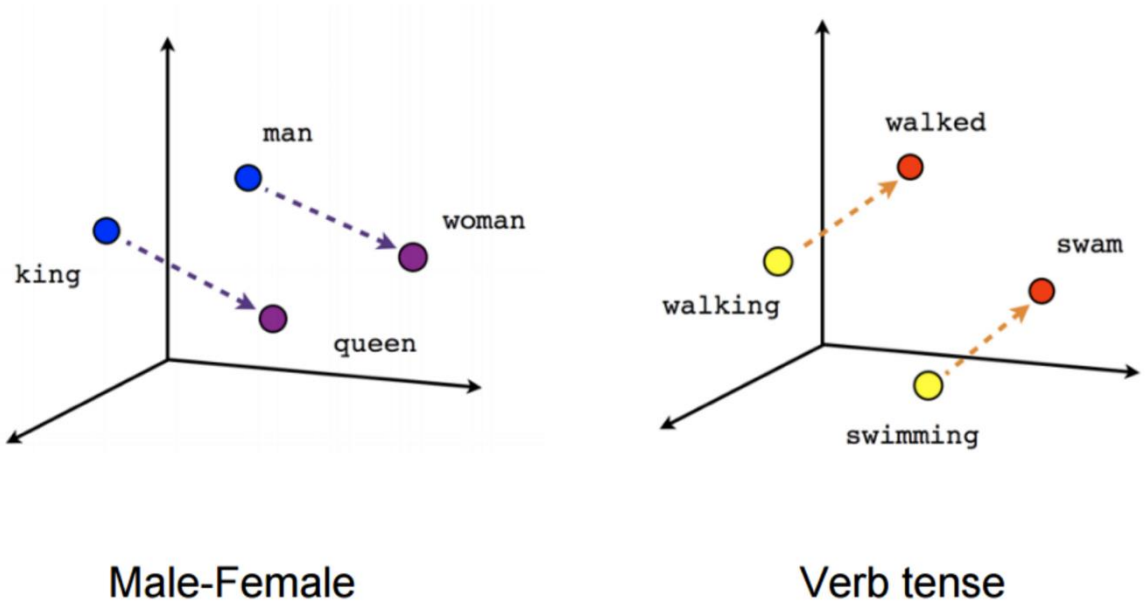


Image source: https://medium.com/@khulasaandh/word-embeddings-fun-with-word2vec-and-game-of-thrones-ea4c24fcf1b8

Word Vectors

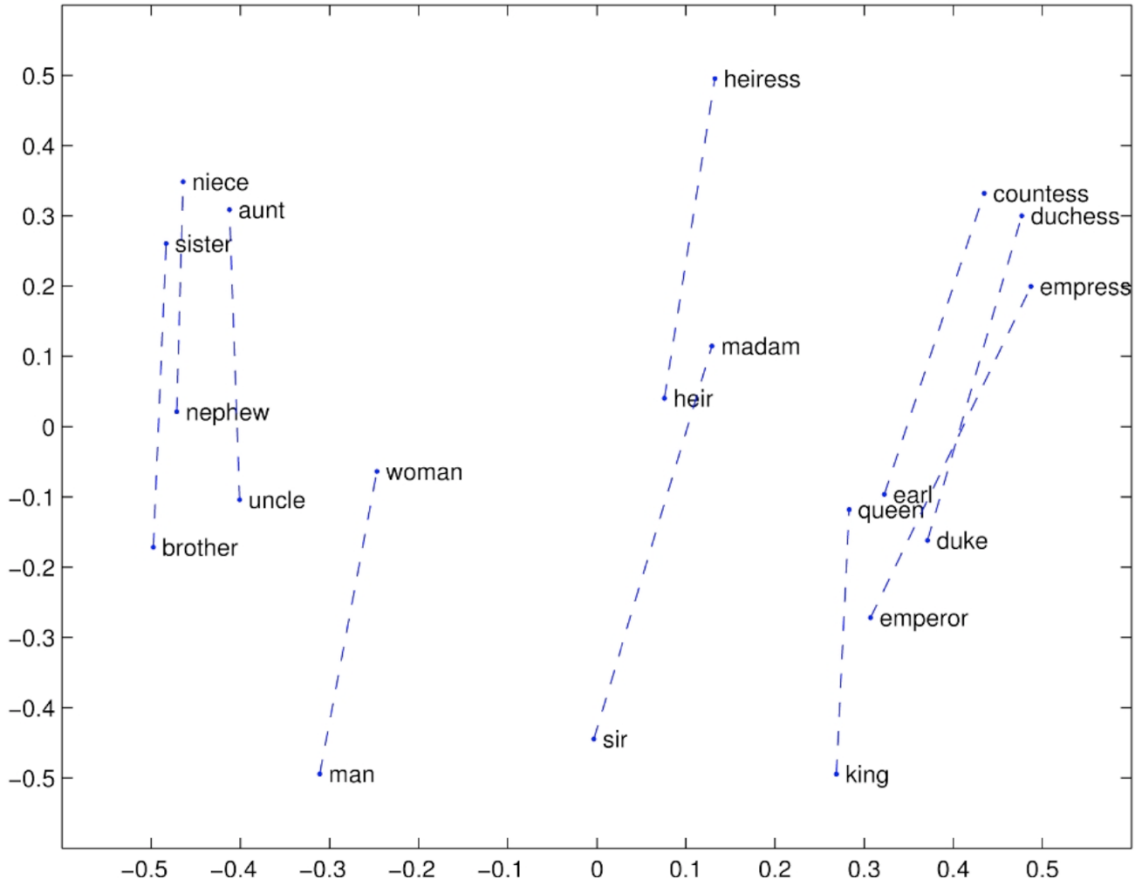


Image source: lecture 2, cs224n

Word Vectors

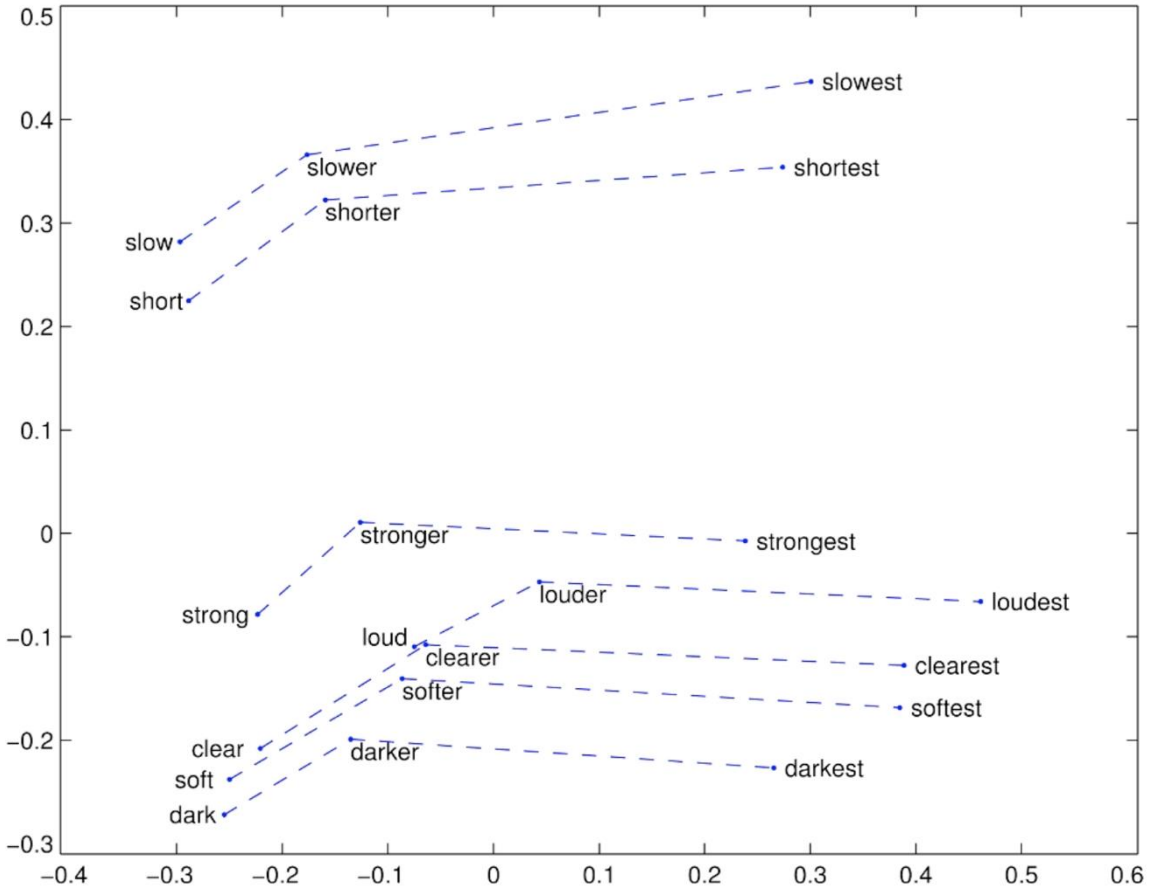
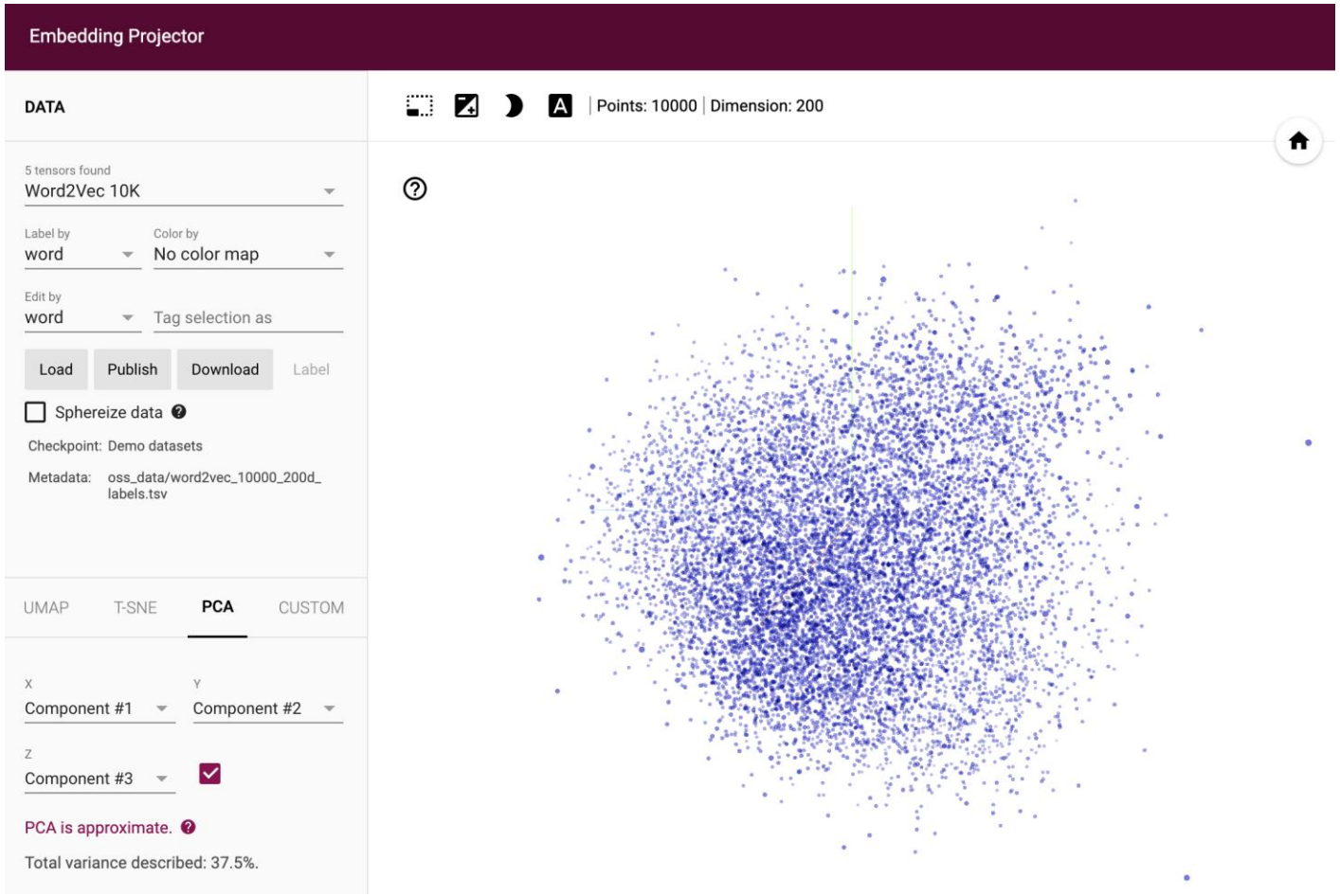


Image source: lecture 2, cs224n



http://projector.tensorflow.org/

XXX2vec

- Follow this idea, any pair frequently occur in a set can be represented by a vector:
  - Recommender system: item2vec, user2vec.
  - Graph: node2vec, edge2vec.
  - Social media: tweet2vec, emoji2vec.
  - Bioinformatics: protein2vec, dna2vec.
  - Health records: med2vec
  - Chemistry: molecule2vec, atom2vec.
  - Finance: stock2vec, fund2vec, company2vec.

Atom2vec

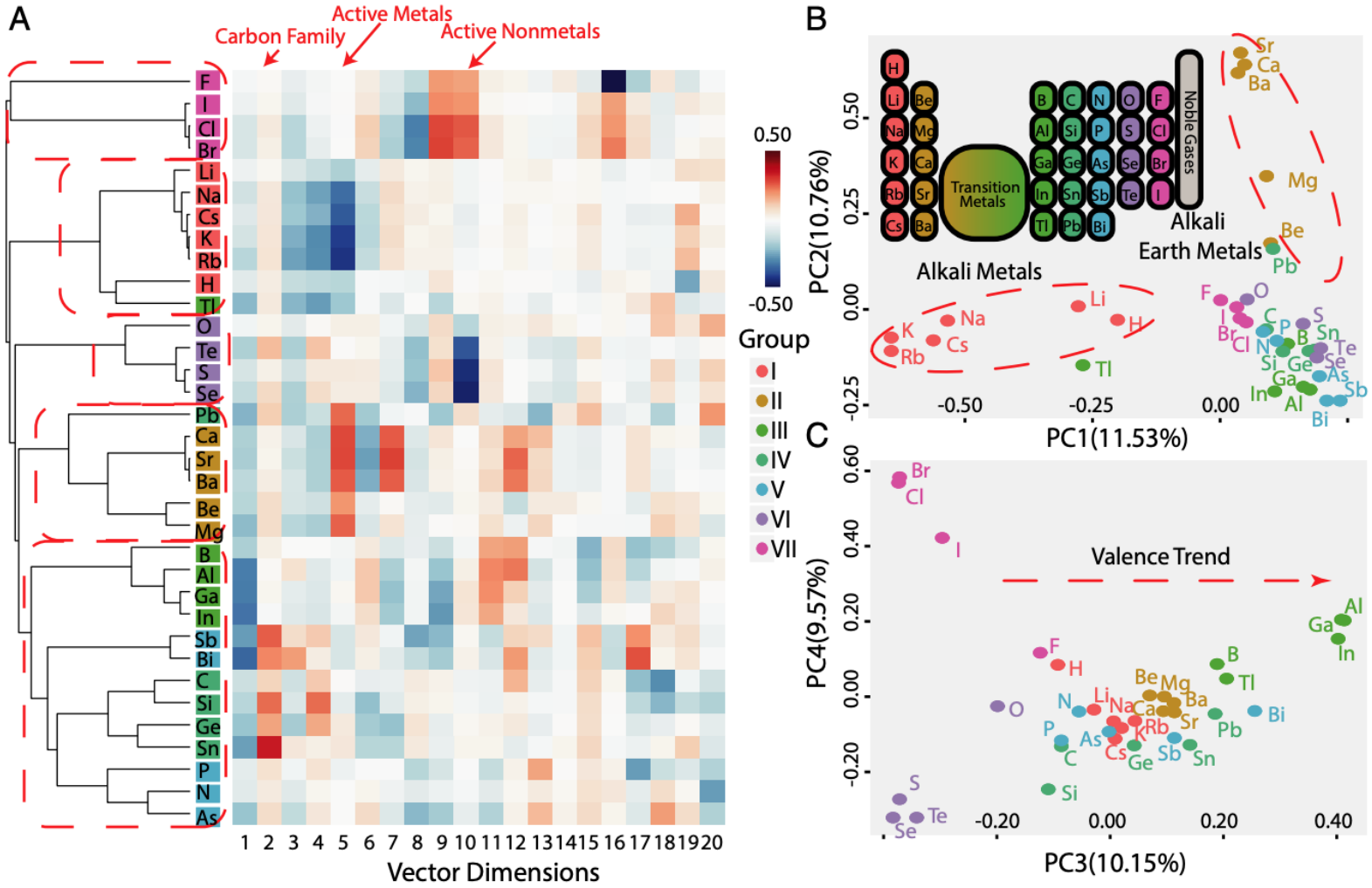


Image source: Zhou, Quan, Peizhe Tang, Shenxiu Liu, Jinbo Pan, Qimin Yan, and Shou-Cheng Zhang. "Learning atoms for materials discovery." Proceedings of the National Academy of Sciences 115, no. 28 (2018): E6411-E6417.

Emoji2vec

Image source: Eisner, Ben, Tim Rocktäschel, Isabelle Augenstein, Matko Bošnjak, and Sebastian Riedel. "emoji2vec: Learning emoji representations from their description." arXiv preprint arXiv:1609.08359 (2016).

Training Word2vec in PyTorch

- Use `nn.Embedding` for embedding loop-up.

```
word_to_ix = {"hello": 0, "world": 1}
embeds = nn.Embedding(2, 5)  # 2 words in vocab, 5 dimensional embeddings
lookup_tensor = torch.tensor([word_to_ix["hello"]], dtype=torch.long)
hello_embed = embeds(lookup_tensor)
print(hello_embed)
```

```
tensor([[ 0.6614,  0.2669,  0.0617,  0.6213, -0.4519]],
       grad_fn=<EmbeddingBackward>)
```

Code is adapted from https://pytorch.org/tutorials/beginner/nlp/word_embeddings_tutorial.html

```
EMBEDDING_DIM = 10
# We will use Shakespeare Sonnet 2
test_sentence = """When forty winters shall besiege thy brow,
And dig deep trenches in thy beauty's field,
Thy youth's proud livery so gazed on now,
Will be a totter'd weed of small worth held:
Then being asked, where all thy beauty lies,
Where all the treasure of thy lusty days;
To say, within thine own deep sunken eyes,
Were an all-eating shame, and thriftless praise.
How much more praise deserv'd thy beauty's use,
If thou couldst answer 'This fair child of mine
Shall sum my count, and make my old excuse,'
Proving his beauty by succession thine!
This were to be new made when thou art old,
And see thy blood warm when thou feel'st it cold.""".split()

# we should tokenize the input, but we will ignore that for now
# build a list of tuples.
# Each tuple is ([ word_i-2, word_i-1, word_i+1, word_i+2 ], target word)

context_tuple_list = [(test_sentence[i + 2],
                       [test_sentence[i], test_sentence[i + 1],
                        test_sentence[i + 3], test_sentence[i + 4]])
                        for i in range(len(test_sentence) - 4)]

# print the first 3, just so you can see what they look like
print(context_tuple_list[:3])

vocab = set(test_sentence)
word_to_ix = {word: i for i, word in enumerate(vocab)}
```

```
[('winters', ['When', 'forty', 'shall', 'besiege']), ('shall', ['forty',
'winters', 'besiege', 'thy']), ('besiege', ['winters', 'shall', 'thy', 'b
row,'])]
```

Code is adapted from https://pytorch.org/tutorials/beginner/nlp/word_embeddings_tutorial.html

```
class SkipGramLanguageModeler(nn.Module):

    def __init__(self, vocab_size, embedding_dim):
        super(SkipGramLanguageModeler, self).__init__()
        self.embeddings = nn.Embedding(vocab_size, embedding_dim)
        self.linear = nn.Linear(embedding_dim, vocab_size)

    def forward(self, inputs):
        embeds = self.embeddings(inputs).view((1, -1))
        out = self.linear(embeds)
        log_probs = F.log_softmax(out, dim=1)
        return log_probs


losses = []
loss_function = nn.NLLLoss()
model = SkipGramLanguageModeler(len(vocab), EMBEDDING_DIM)
optimizer = optim.SGD(model.parameters(), lr=0.001)
```

Code is adapted from https://pytorch.org/tutorials/beginner/nlp/word_embeddings_tutorial.html

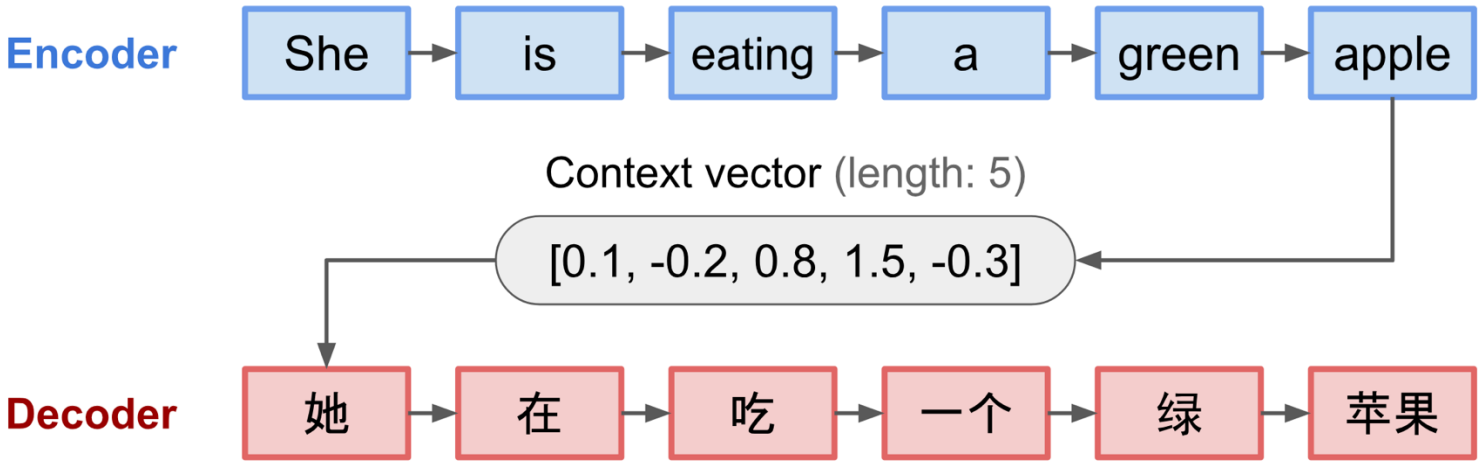
```
for epoch in range(10):
    total_loss = 0
    for target, context_list in context_tuple_list:
        for context in context_list:

            # Step 1. Prepare the inputs to be passed to the model (i.e, t
            # into integer indices and wrap them in tensors)
            context_idxs = torch.tensor(word_to_ix[context], dtype=torch.l

            # Step 2. Recall that torch *accumulates* gradients. Before pa
            # new instance, you need to zero out the gradients from the ol
            # instance
            model.zero_grad()

            # Step 3. Run the forward pass, getting log probabilities over
            # words
            log_probs = model(context_idxs)

            # Step 4. Compute your loss function. (Again, Torch wants the
            # word wrapped in a tensor)
            loss = loss_function(log_probs, torch.tensor([word_to_ix[targe

            # Step 5. Do the backward pass and update the gradient
            loss.backward()
            optimizer.step()

            # Get the Python number from a 1-element Tensor by calling ten
            total_loss += loss.item()
    print(total_loss)
    losses.append(total_loss)
```

```
2106.0324614048004
2099.963498353958
2093.969718694687
2088.050463914871
2082.2050607204437
2076.4329063892365
2070.7334401607513
2065.106065750122
2059.5502502918243
2054.065470457077
```

Code is adapted from https://pytorch.org/tutorials/beginner/nlp/word_embeddings_tutorial.html

Transformer Models

What's Wrong with Seq2Seq Models?



- Can the context vector (fixed-length) remember all details in the input sequence?
- Even with LSTM/GRU, remembering long sequence is very challenging!

Human Attention

What animal is in this photo?

Example: https://lilianweng.github.io/posts/2018-06-24-attention/
Original image credit: Instagram @mensweardog

Human Attention

What animal is in this photo?

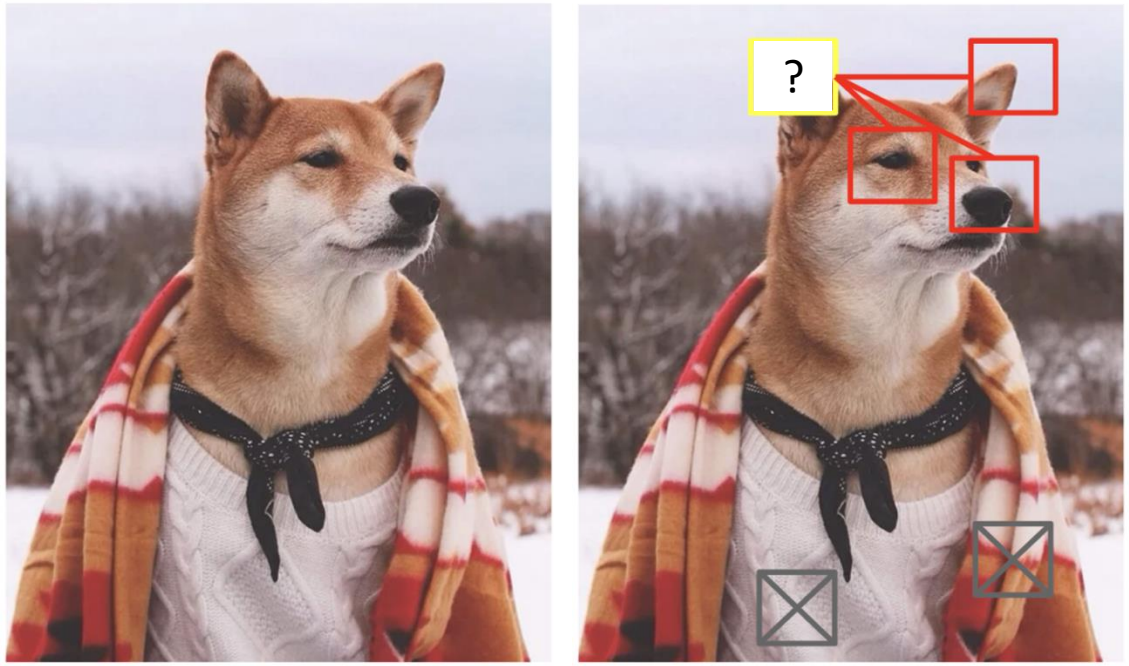
What is the background of this photo?

What color is the T-shirt the dog is wearing?

What material is the scarf the dog is wearing?

Example: https://lilianweng.github.io/posts/2018-06-24-attention/
Original image credit: Instagram @mensweardog

Human Attention



- Our human attention focus on only certain regions.
- We focus on different regions for answering different questions.
- Imagine what is in the white box: pointy ear.
  - Because we see the noise and another pointy ear in red boxes.
  - We will not focus on the outfit and background when imagining the masked pixels.

Example: https://lilianweng.github.io/posts/2018-06-24-attention/
Original image credit: Instagram @mensweardog

Attention Mechanisms

- ~~Using one context vector to remember the entire input sequence~~
  → Idea: Construct a different context when decoding different words.
- Make the context have access to the entire input sequence, no forgetting problem.
  (closed-book exam becomes open-book exam)

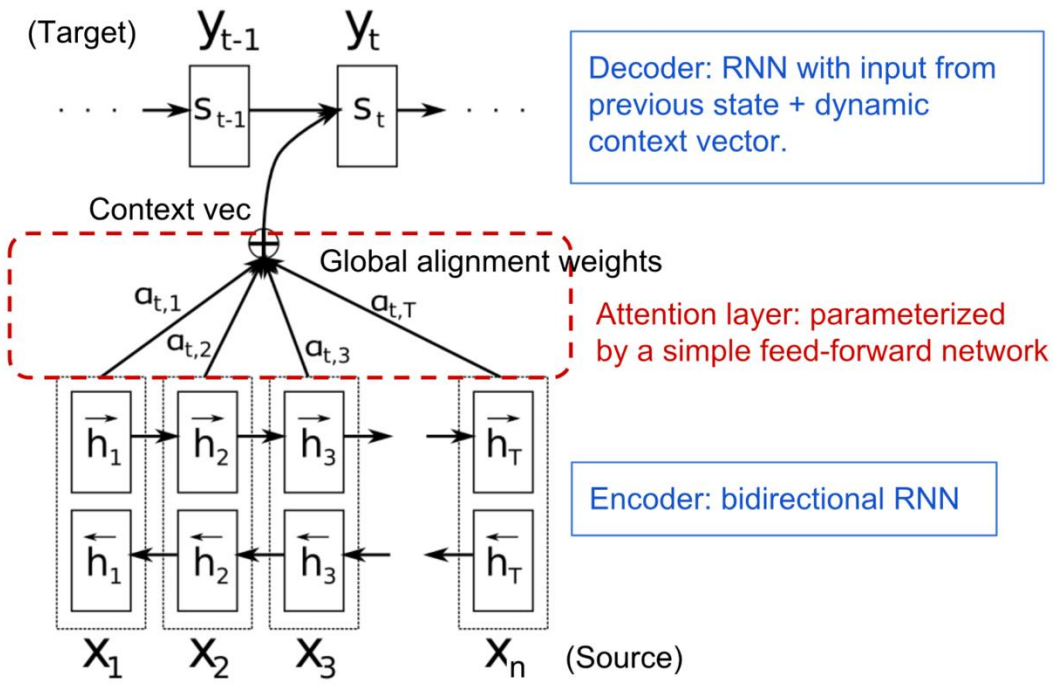


Figure: https://lilianweng.github.io/posts/2018-06-24-attention/

Context vector for output y_t: $c_t = \sum_{i=1}^{n} \alpha_{t,i} h_i$

- h_i: hidden state of encoder at step i
- $\alpha_{t,i}$: how y_t and x_i are aligned.

Additive Attention

$$\alpha_{t,i} = \text{align}(y_t, x_i) = \frac{\exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i))}{\sum_{i'=1}^{n} \exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_{i'}))}$$

$$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$$

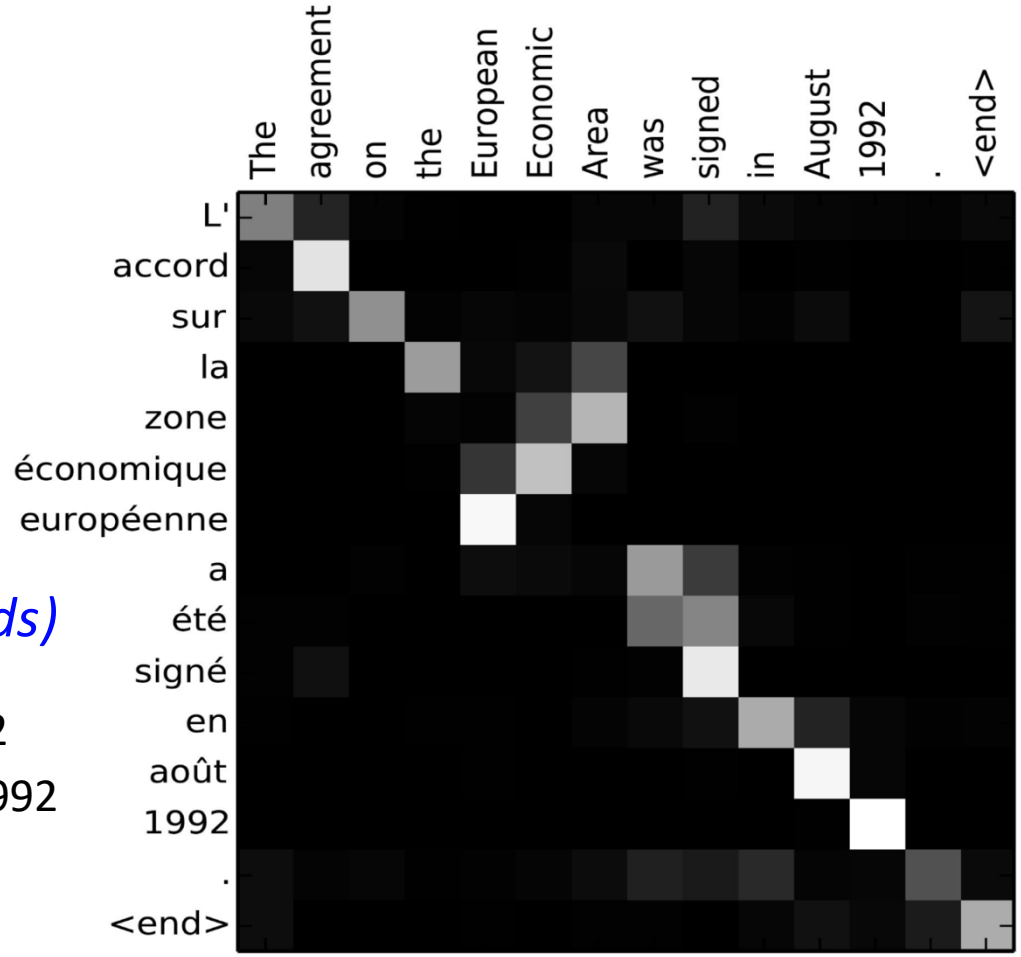
the extra parameter to be learned

Attention Mechanisms

- Idea: Construct a different context when decoding different words.
- Make the context have access to the entire input sequence, no forgetting problem.

$$\alpha_{t,i} = \text{align}(y_t, x_i) = \frac{\exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i))}{\sum_{i'=1}^{n} \exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_{i'}))}$$

$$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$$



Visualization of alignment in French-English translation:

(what input words to look at when generating the output words)

L'accord sur l'Espace économique européen a été signé en août 1992

The agreement on the European Economic Area was signed in August 1992

Figure: https://lilianweng.github.io/posts/2018-06-24-attention/

Attention Mechanism Family

Name	Alignment score function	Citation
Content-base attention	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$	Graves2014
Additive(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_{t-1}; \boldsymbol{h}_i])$	Bahdanau2015
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$ Note: This simplifies the softmax alignment to only depend on the target position.	Luong2015
General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.	Luong2015
Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$	Luong2015
Scaled Dot-Product(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017

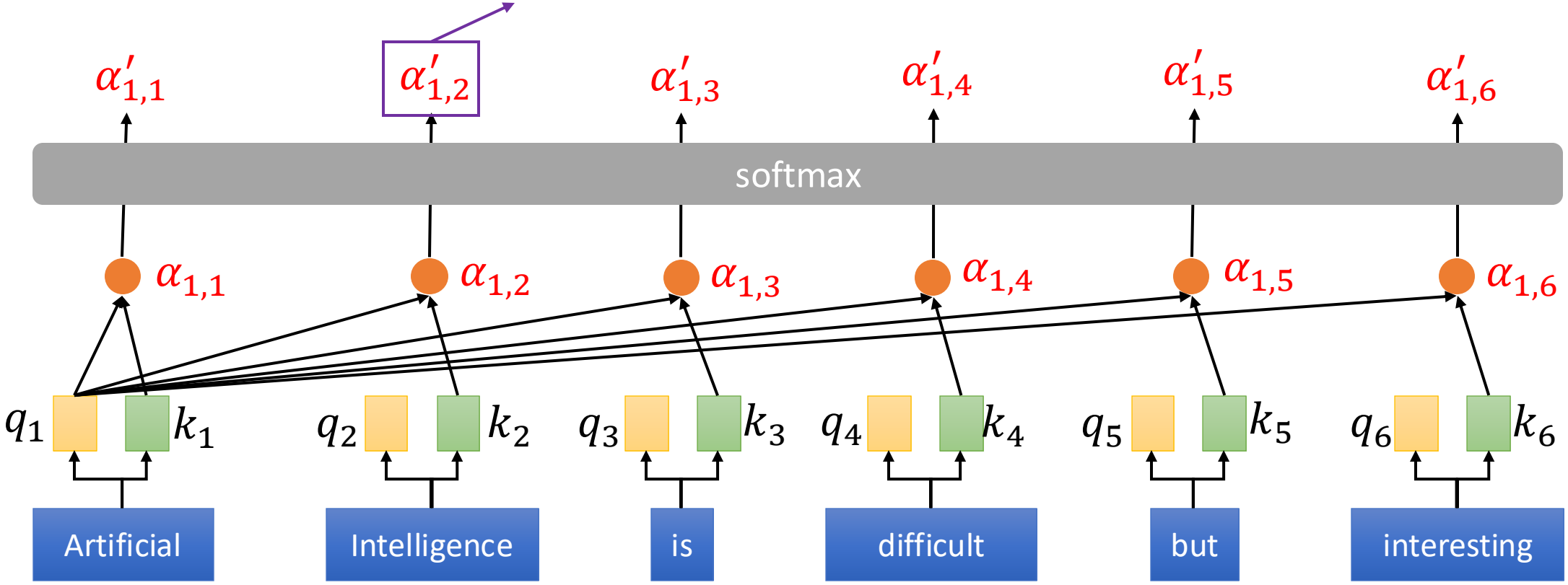
Figure: https://lilianweng.github.io/posts/2018-06-24-attention/

Attention is All You Need (forget about RNNs)

- Self-attention computes attention between words in a sentence

$$\alpha'_{i,j} = \frac{\exp(\alpha_{i,j})}{\sum_k \exp(\alpha_{i,k})}$$

how much we focus on the 2nd word when representing the 1st word



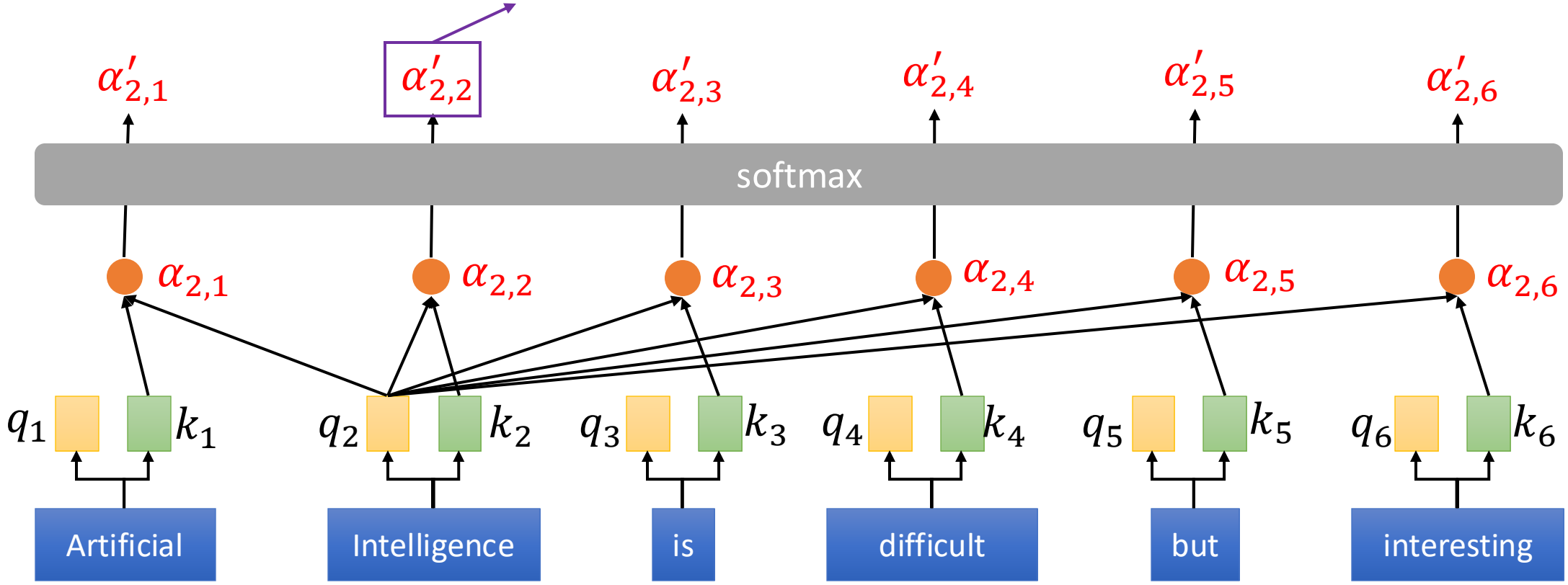
$q_i = W^Q x_i$: *query vector* $\quad k_i = W^K x_i$: *key vector*

Attention is All You Need (forget about RNNs)

- Self-attention computes attention between words in a sentence

$$\alpha'_{i,j} = \frac{\exp(\alpha_{i,j})}{\sum_k \exp(\alpha_{i,k})}$$

how much we focus on the 2nd word when representing the 2nd word



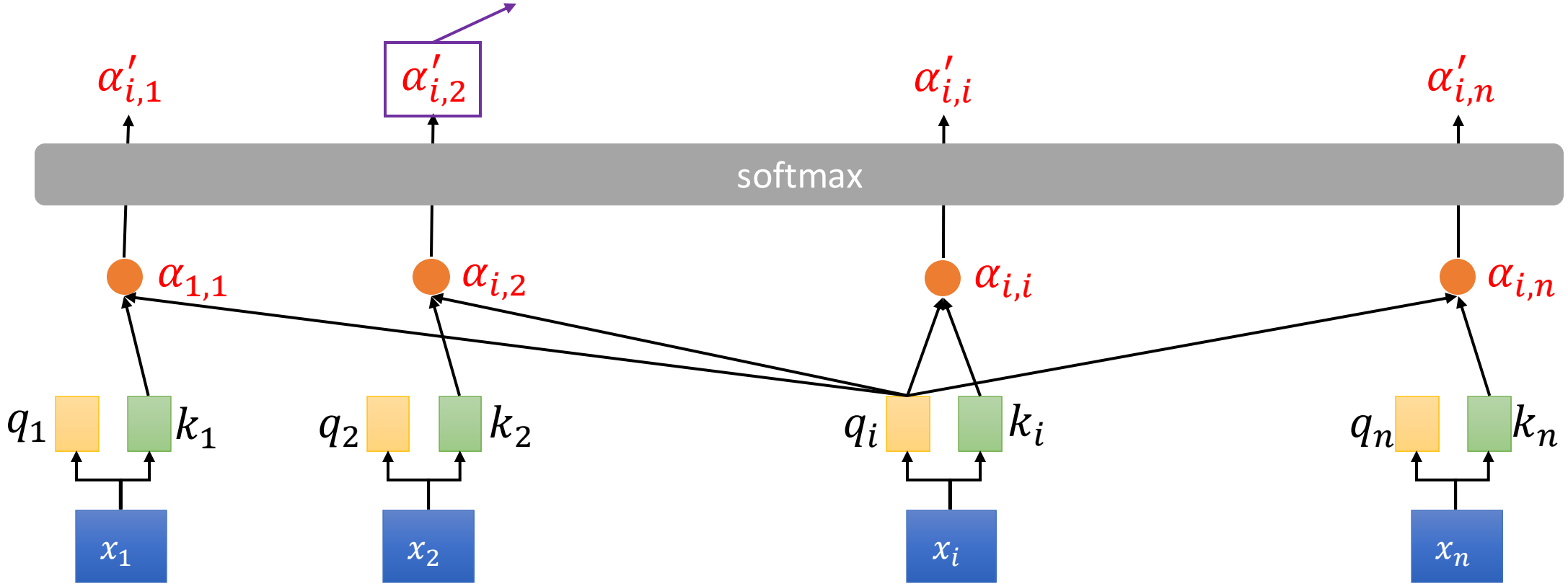
$q_i = W^Q x_i$: *query vector* $\quad$ $k_i = W^K x_i$: *key vector*

Attention is All You Need (forget about RNNs)

- Self-attention computes attention between words in a sentence

$$\alpha'_{i,j} = \frac{\exp(\alpha_{i,j})}{\sum_k \exp(\alpha_{i,k})}$$

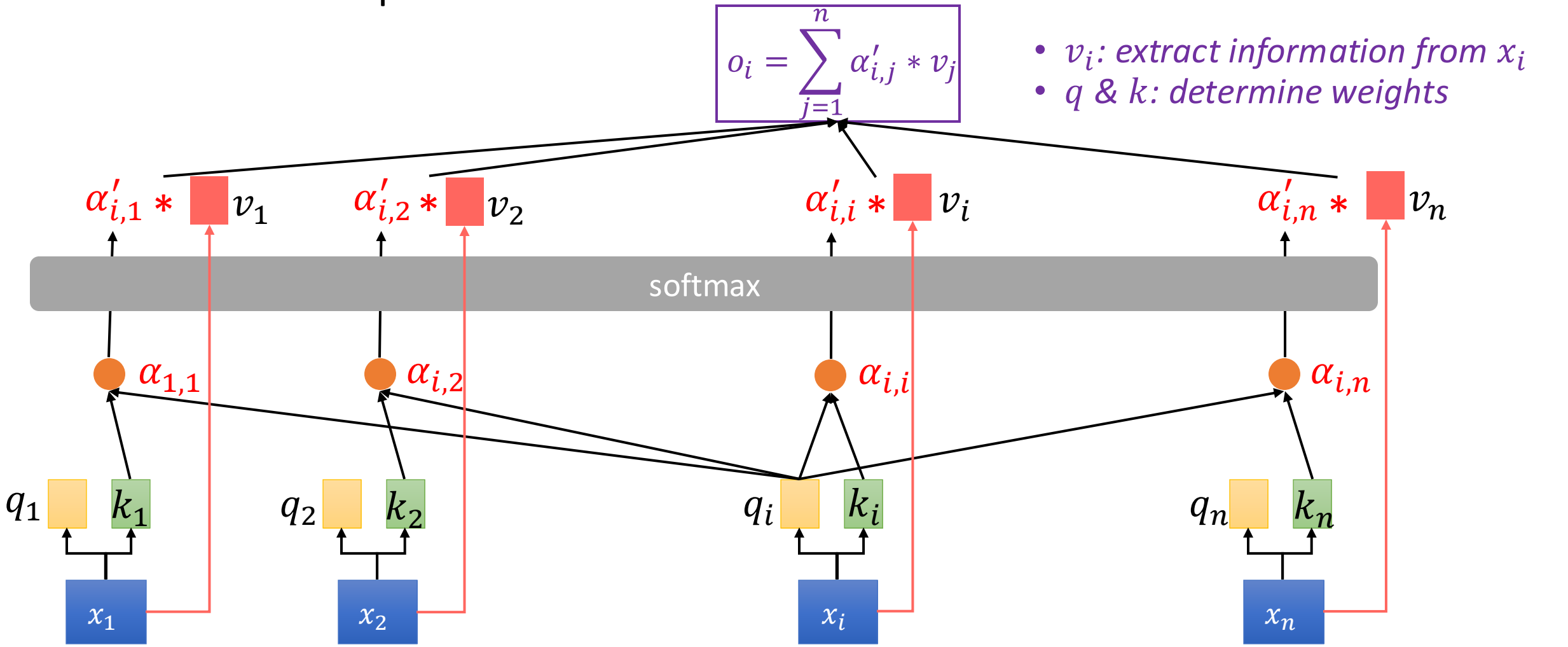
how much we focus on the 2nd word when representing the ith word



$q_i = W^Q x_i$: *query vector* $\quad$ $k_i = W^K x_i$: *key vector*

Attention is All You Need (forget about RNNs)

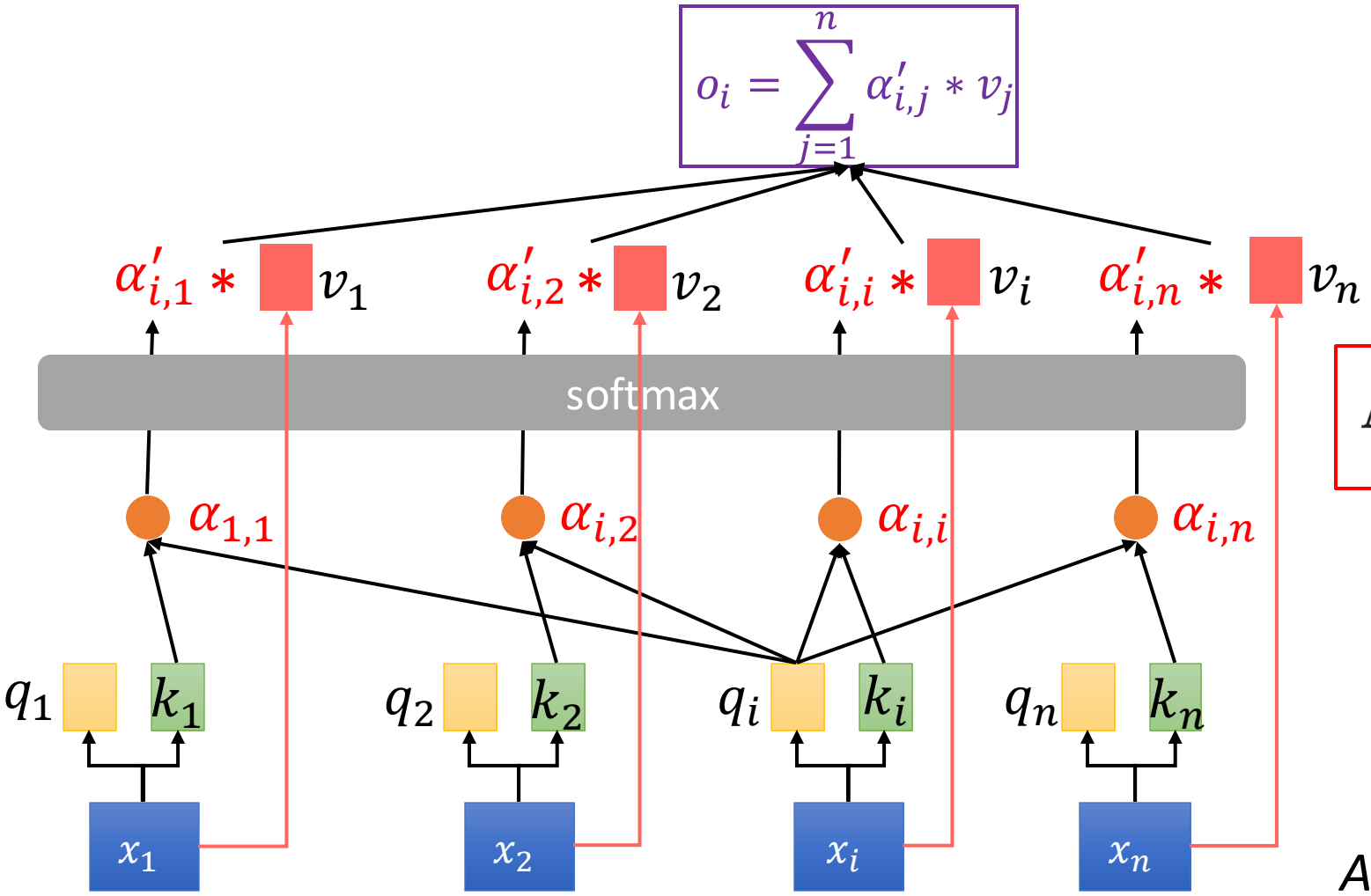
- Self-attention computes attention between words in a sentence



$q_i = W^Q x_i$: query vector $k_i = W^K x_i$: key vector $v_i = W^V x_i$: value vector

Attention is All You Need (forget about RNNs)

- Self-attention computes attention between words in a sentence



Scaled dot-product attention

Rewrite in matrix form:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{n}})\mathbf{V}$$

$$\mathbf{Q} = \mathbf{W}^Q\mathbf{X}$$
$$\mathbf{K} = \mathbf{W}^K\mathbf{X}$$
$$\mathbf{V} = \mathbf{W}^V\mathbf{X}$$

n: hidden dimension (NOT number of steps in the figure in left)

A core element of Transformer model

Transformer Architecture

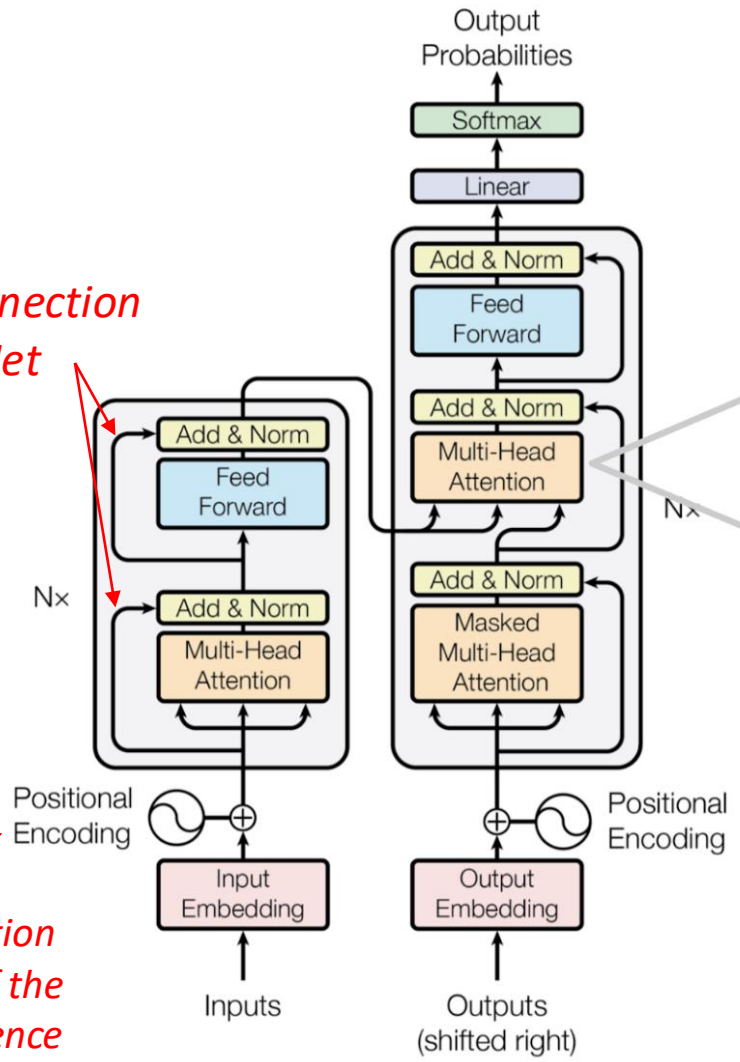
- Only use self-attention and feed forward neural network to model contextual information.
- Originally for machine translation by the encoder-decoder achitecture, but now widely used as a basic component of many NLP and CV tasks.

- Basis of GPT, ChatGPT, and Sora.

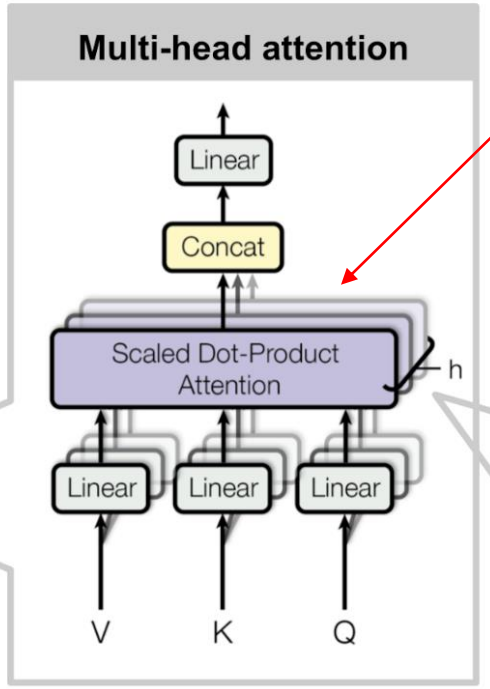
Transformer Architecture

Residual connection in ResNet

Provide information of the position of the word in the sentence

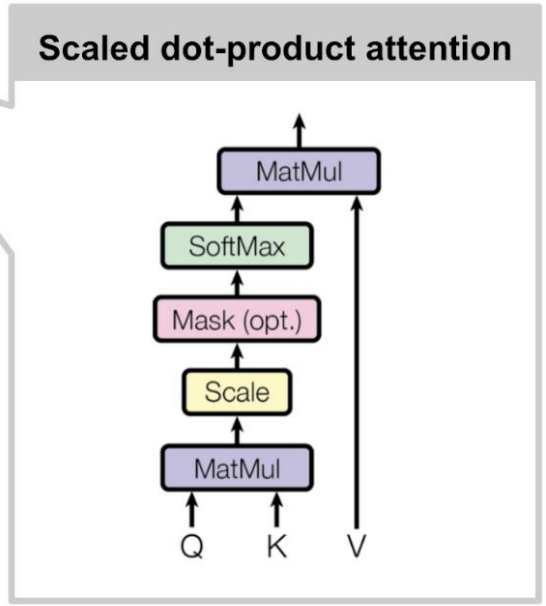


Run multiple attention and then combine them (the idea of ensembling)



Zoom-In!

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{n}})\mathbf{V}$$



Zoom-In!

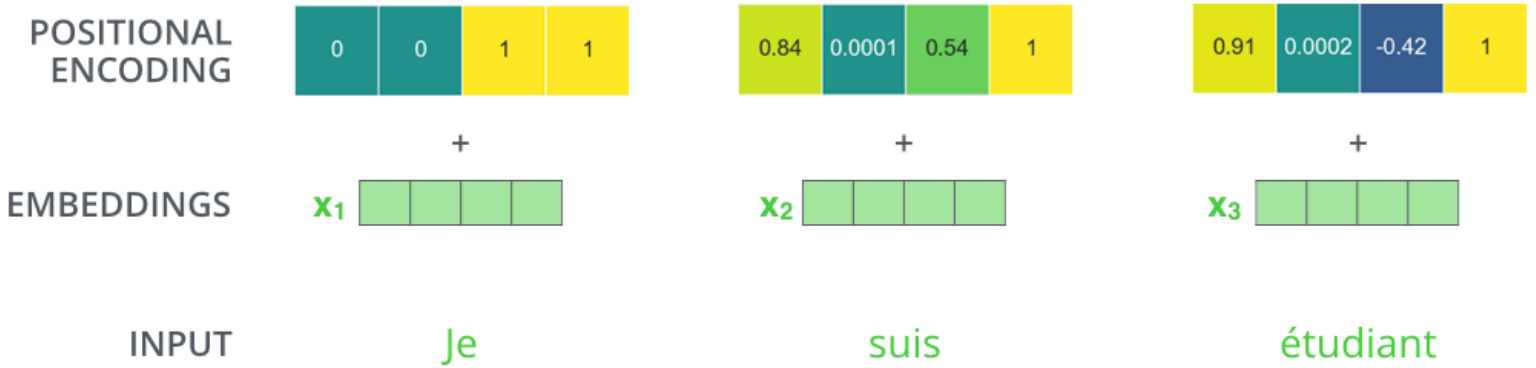
Figure: https://lilianweng.github.io/posts/2018-06-24-attention/ (original source: source: Fig 1 & 2 in Vaswani, et al., 2017.)

Transformer Architecture

- The sinusoid-wave-based positional encoding is fomulated as:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$
$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

pos is the position, i is the dimension index, d_{model} is word embedding dimension.



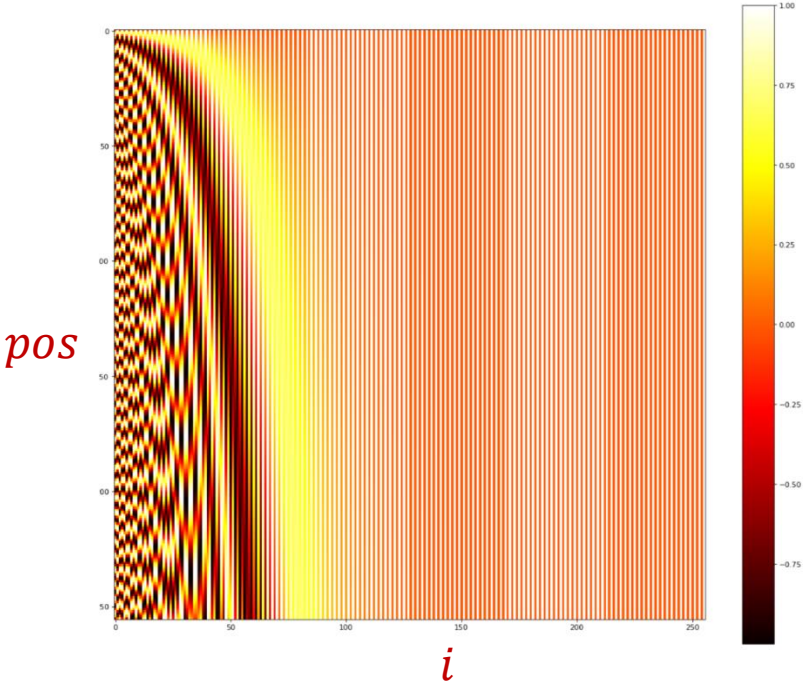


Image source: https://jalammar.github.io/illustrated-transformer/

Transformer Architecture

- For the positional encoding using sin and cos, the authors said:

> *We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k, PE_{pos+k} can be represented as a linear function of PE_{pos}.*

- Because:

$$\begin{cases} \sin(\alpha+\beta) = \sin\alpha\cos\beta + \cos\alpha\sin\beta \\ \cos(\alpha+\beta) = \cos\alpha\cos\beta - \sin\alpha\sin\beta \end{cases}$$

Vision Transformer

An **image** is **worth 16x16 words**: **Transformers** for **image recognition** at **scale**
A Dosovitskiy, L Beyer, A Kolesnikov… - arXiv preprint arXiv …, 2020 - arxiv.org
While the Transformer architecture has become the de-facto standard for natural language processing tasks, its applications to computer vision remain limited. In vision, attention is either applied in conjunction with convolutional networks, or used to replace certain …
☆ 99 Cited by 1341 Related articles All 5 versions »

The Vision Transformer treats an input image as a sequence of patches, akin to a series of word embeddings generated by an NLP Transformer.

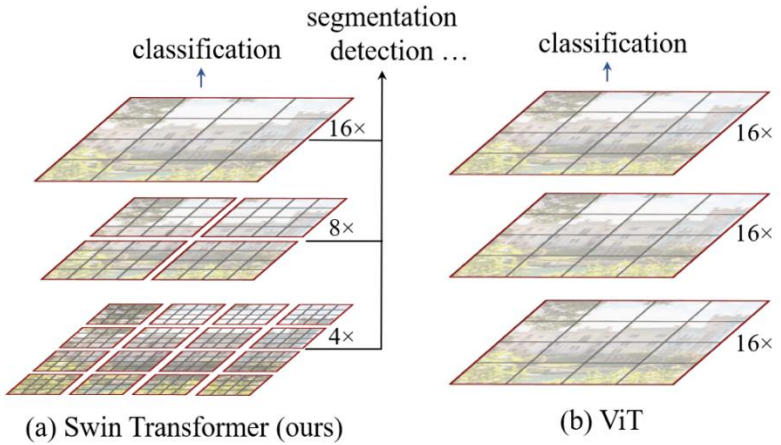
Image source: https://ai.googleblog.com/2020/12/transformers-for-image-recognition-at.html

Swin Transformer

Swin transformer: Hierarchical vision transformer using shifted windows
Z Liu, Y Lin, Y Cao, H Hu, Y Wei… - Proceedings of the …, 2021 - openaccess.thecvf.com
This paper presents a new vision Transformer, called Swin Transformer, that capably serves as a general-purpose backbone for computer vision. Challenges in adapting Transformer …
☆ 保存 99 引用 被引用次数：3581 相关文章 »

- Challenges in adapting Transformer from language to vision:

large variations in the scale of visual entities and the high resolution of pixels in images compared to words in text.



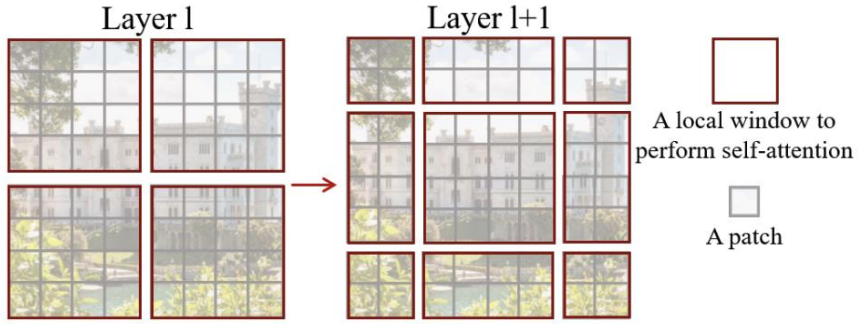


Image source: Liu, Ze, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. "Swin transformer: Hierarchical vision transformer using shifted windows." In Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 10012-10022. 2021.

BERT

Bert: Pre-training of deep bidirectional transformers for language understanding
J Devlin, MW Chang, K Lee, K Toutanova - arXiv preprint arXiv …, 2018 - arxiv.org
… Unlike Peters et al. (2018a) and Radford et al. (2018), we do not use traditional left-to-right or right-to-left language models to **pre-train BERT**. Instead, we **pre-train BERT** using two unsupervised tasks, described in this section. This step is presented in the left part of Figure 1 …
☆ 99 Cited by 28707 Related articles All 30 versions »

- BERT is a pre-training model using deep bidirectional transformers for language understanding. *(encoder part of Transformer)*
- It uses the idea of self-supervised learning, rather than training on any specific NLP task.
- After we obtain the BERT pre-trained model, we can fine-tune it for a specific NLP task. (*given a sentence, we can obtain a representation from BERT and use it for downstream tasks*)

BERT Model Architecture

- BERT's model architecture is a multi-layer bidirectional Transformer encoder.
- The input is word embedding and output is context sensitive word representation.

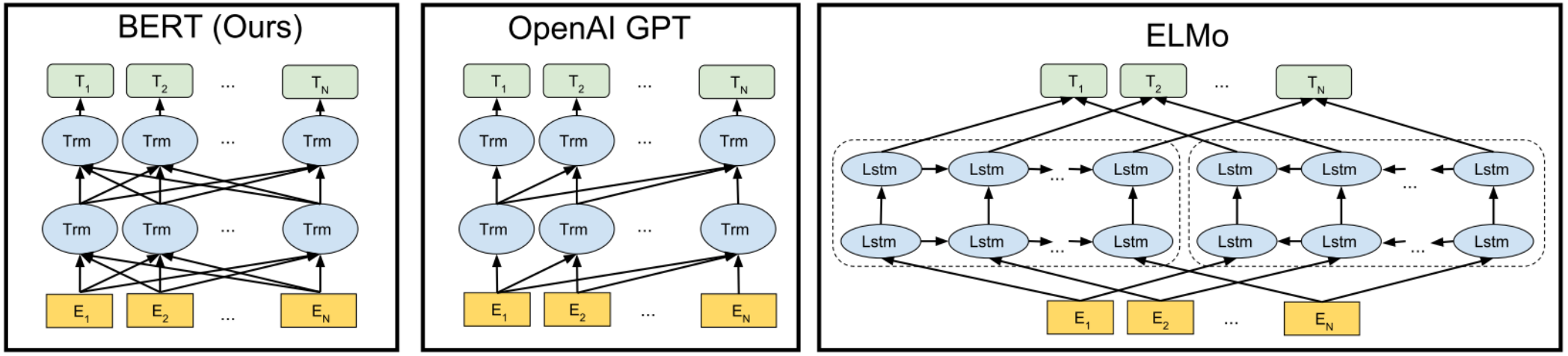


Image source: Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).

Input representation

- Positional embeddings are learnable, rather than fixed magic number as in the Transformer paper.
- Each input sequence is a pair of sentences, separated by the token `[SEP]`. It adopted two learnable embeddings to each sentence.
- `[CLS]` is the a special classification embedding for the first token of every sequence.

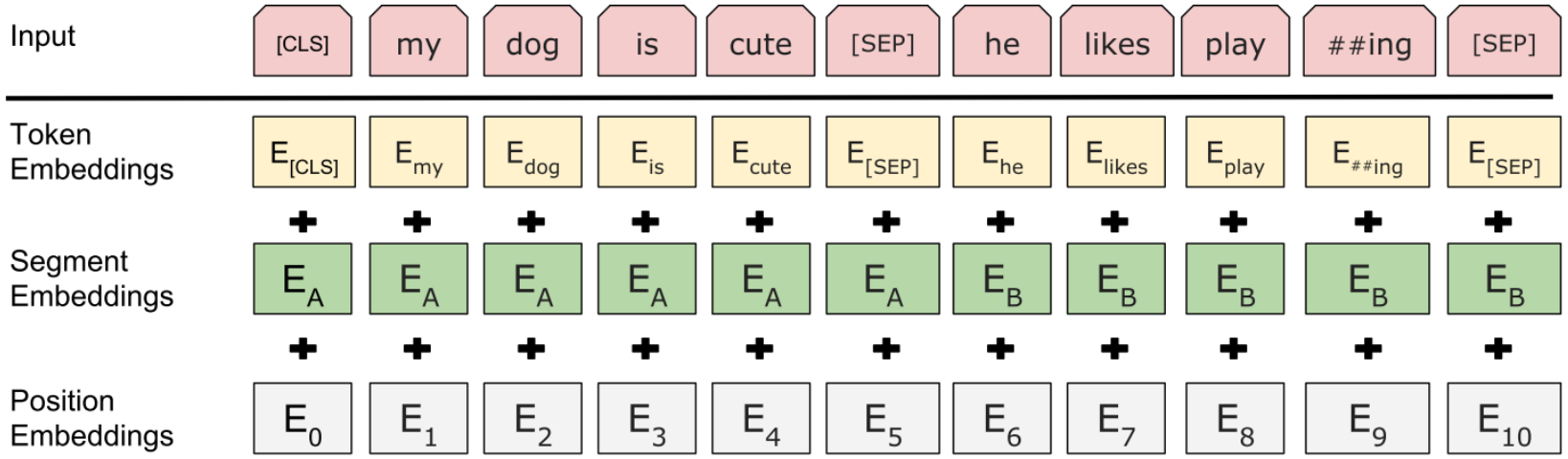


Image source: Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).

Pre-Training Task 1: Masked LM

- Mask some percentage of the input tokens at random, and then predicting only those masked tokens.
- Use `[MASK]` token to replace 15% tokens randomly, and use the real token as the label to make it predict.
- Howerver, the `[MASK]` token is never seen during fine-tuning. The authors proposed the following strategy:
  - 80% of the time: Replace the word with the `[MASK]` token.
    - e.g., my dog is hairy → my dog is `[MASK]`.
  - 10% of the time: Replace the word with a random word.
    - e.g., my dog is hairy → my dog is apple.
  - 10% of the time: Keep the word unchanged. The purpose of this is to bias the representation towards the actual observed word.
    - e.g., my dog is hairy → my dog is hairy.

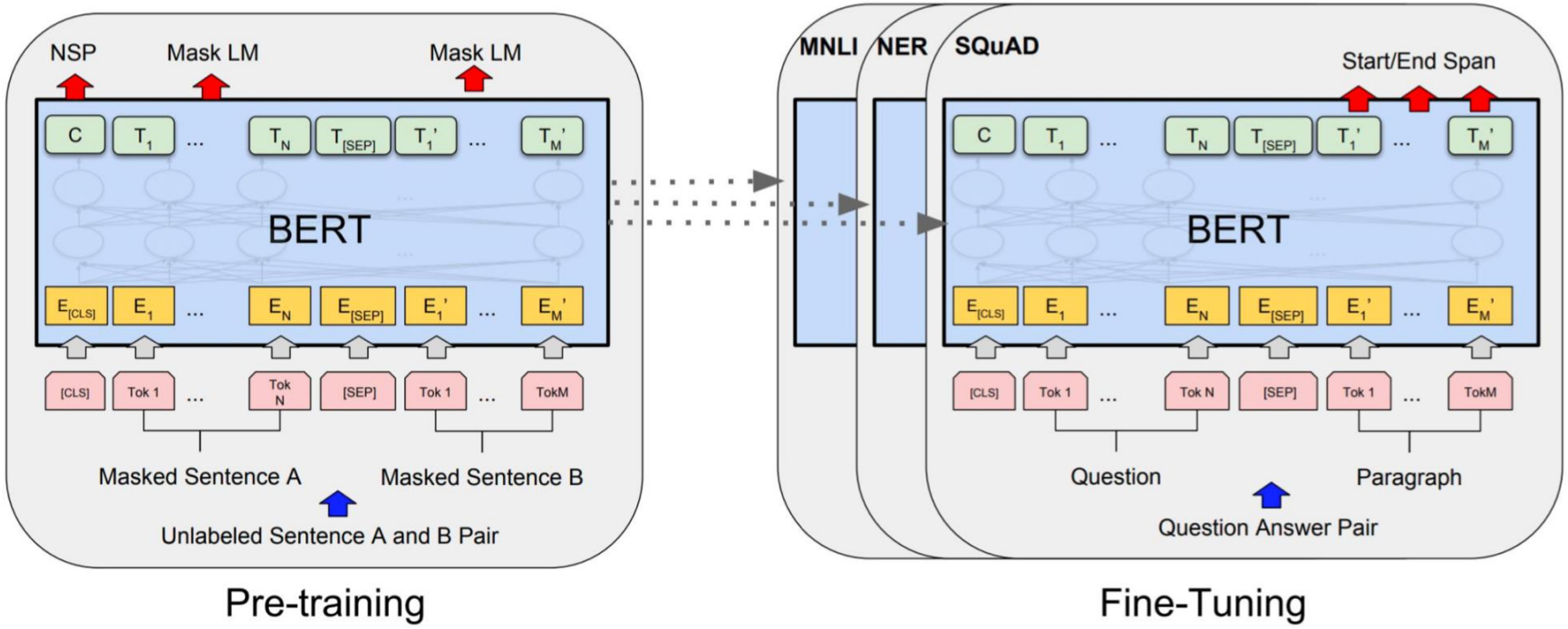
Pre-Training Task 2: Next Sentence Prediction

- Make the model understand the relationship between two text sentences.
- Choose the sentences A and B for each pre-training example.
  - 50% of the time B is the actual next sentence that follows A.
  - 50% of the time it is a random sentence from the corpus.
- Example:
  - Input = `[CLS] the man went to [MASK] store [SEP] he bought a gallon [MASK] milk [SEP]`

    Label = `IsNext`
  - Input = `[CLS] the man [MASK] to the store [SEP] penguin [MASK] are flight ##less birds [SEP]`

    Label = `NotNext`

Pre-Train and Fine-Tune



GPT-3

Language models are few-shot learners

TB Brown, B Mann, N Ryder, M Subbiah… - arXiv preprint arXiv …, 2020 - arxiv.org

Recent work has demonstrated substantial gains on many NLP tasks and benchmarks by pre-training on a large corpus of text followed by fine-tuning on a specific task. While typically task-agnostic in architecture, this method still requires task-specific fine-tuning datasets of thousands or tens of thousands of examples. By contrast, humans can generally perform a new language task from only a few examples or from simple instructions-something which current NLP systems still largely struggle to do. Here we show that scaling …

☆ 99 Cited by 180 Related articles All 2 versions »

- Generative Pre-trained Transformer 3 (GPT-3) is also a Transformer-based pre-trained language model. (decoder part of Transformer)

- It is the third generation in the GPT-n series created by OpenAI.
- It only uses the decoder part of Transformer ("decoder-only").
- It has 175 billion parameters.

GPT-n

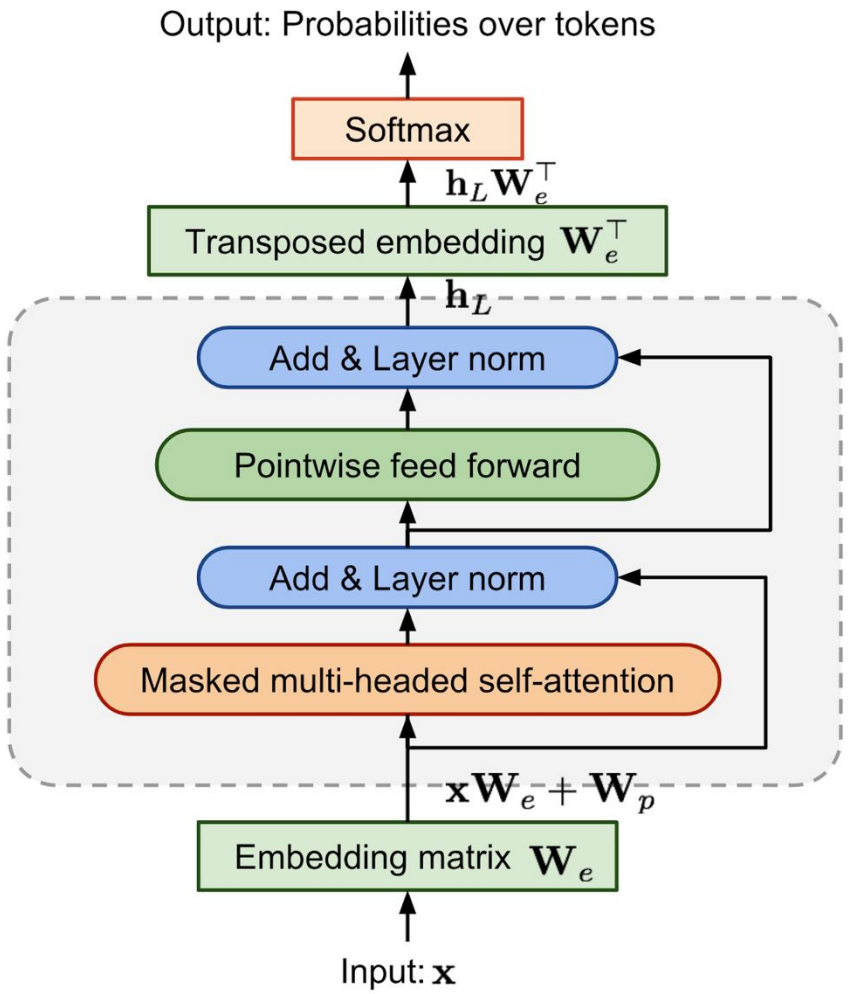
- Input is only one sentence.
- Predict the next word using previous words. Loss:

$$\mathcal{L}_{\text{LM}} = -\sum_i \log p(x_i \mid x_{i-k}, \ldots, x_{i-1})$$

- During inference, input a text prompt, e.g.,

```
Once upon a time in a faraway land,
```

- GPT then autoregressively predicts the next word.



Transformer Block
Repeat x L=12

$$\mathbf{h}_\ell = \text{transformer\_block}(\mathbf{h}_{\ell-1})$$
$$\ell = 1, \ldots, L$$

Source: https://spencergreenberg.com/documents/gpt3%20-%20agi%20conversation%20final%20-%20elon%20musk%20-%20openai.pdf

GPT-n

Have almost identical (but larger) architecture

ChatGPT: similar architecture with GPT, finetuned for conversations

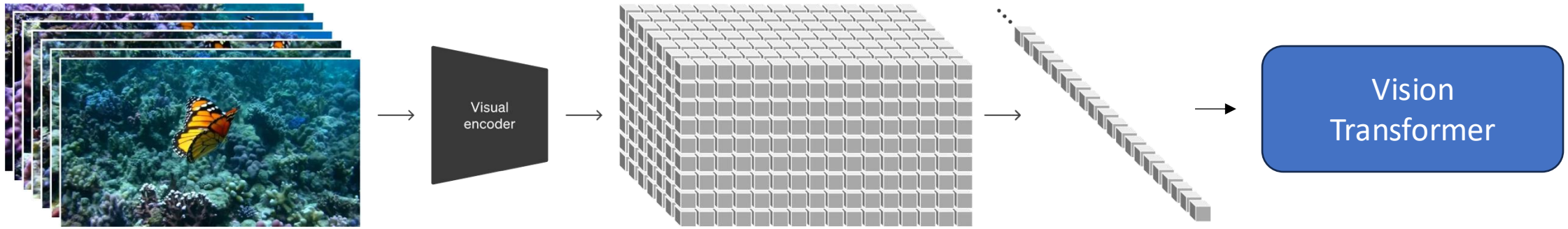
OpenAI's *GPT-n* series

Model	Architecture	Parameter count	Training data	Release date	Training cost
GPT-1	12-level, 12-headed Transformer decoder (no encoder), followed by linear-softmax.	117 million	BookCorpus:[39] 4.5 GB of text, from 7000 unpublished books of various genres.	June 11, 2018[9]	30 days on 8 P600 GPUs, or 1 petaFLOP/s-day.[9]
GPT-2	GPT-1, but with modified normalization	1.5 billion	WebText: 40 GB of text, 8 million documents, from 45 million webpages upvoted on Reddit.	February 14, 2019 (initial/limited version) and November 5, 2019 (full version)[40]	"tens of petaflop/s-day",[41] or 1.5e21 FLOP.[42]
GPT-3	GPT-2, but with modification to allow larger scaling	175 billion[43]	499 billion tokens consisting of CommonCrawl (570 GB), WebText, English Wikipedia, and two books corpora (Books1 and Books2).	May 28, 2020[41]	3640 petaflop/s-day (Table D.1[41]), or 3.1e23 FLOP.[42]
GPT-3.5	Undisclosed	175 billion[43]	Undisclosed	March 15, 2022	Undisclosed
GPT-4	Also trained with both text prediction and RLHF; accepts both text and images as input. Further details are not public.[38]	Undisclosed. Estimated 1.7 trillion.[44]	Undisclosed	March 14, 2023	Undisclosed. Estimated 2.1×10^{25} FLOP.[42]

Source: https://en.wikipedia.org/wiki/Generative_pre-trained_transformer

Sora: Video Generation

- Demo: https://openai.com/index/video-generation-models-as-world-simulators/
- Divide video frames into patches. Each patch act as a “word” in Transformer.



Source: https://openai.com/index/video-generation-models-as-world-simulators/

Let your voice be heard!

Thank you for your feedback! 🙌