PETUUM

Carnegie Mellon University
School of Computer Science

Probabilistic Graphical Models

Deep Sequence Models

Zhiting Hu

Lecture 14, March 2, 2020

Reading: see class homepage

Overview: Deep Learning & Generative Models

- 2/19 Lecture 11 Statistical and Algorithmic Foundations of Deep Learning
- 2/24 Lecture 12 Deep generative models (part 1)
- 2/26 Lecture 13 Deep generative models (part 2)
- 3/2 Lecture 14 Deep sequence models
- 3/4 Lecture 15 A unified view of deep generative models

Overview: Deep Learning & Generative Models

- 2/19 Lecture 11 Statistical and Algorithmic Foundations of Deep Learning
- 2/24 Lecture 12 Deep generative models (part 1)
- 2/26 Lecture 13 Deep generative models (part 2)
- 3/2 Lecture 14 Deep sequence models
- 3/4 Lecture 15 A unified view of deep generative models

Inference & Learning

Model Architectures

Outline

- Convolutional Networks (ConvNets)
- Recurrent Networks (RNNs)
  - Long-range dependency, vanishing gradients
  - LSTM
  - RNNs in different forms
- Attention Mechanisms
  - (Query, Key, Value)
  - Attention on Text and Images
- Transformers: Multi-head Attention
  - Transformer
  - BERT

Outline

- Convolutional Networks (ConvNets)
- Recurrent Networks (RNNs)
  - Long-range dependency, vanishing gradients
  - LSTM
  - RNNs in different forms
- Attention Mechanisms
  - (Query, Key, Value)
  - Attention on Text and Images
- Transformers: Multi-head Attention
  - Transformer
  - BERT

Convolutional Networks (ConvNets)

- Biologically-inspired variants of MLPs [LeCun et al. NIPS 1989]
  - Receptive field [Hubel & Wiesel 1962; Fukushima 1982]
    - Visual cortex contains a complex arrangement of cells
    - These cells are sensitive to small **sub-regions** of the visual field
  - The sub-regions are **tiled** to cover the entire visual field

Local Filters

Exploit the strong spatially local correlation present in natural images

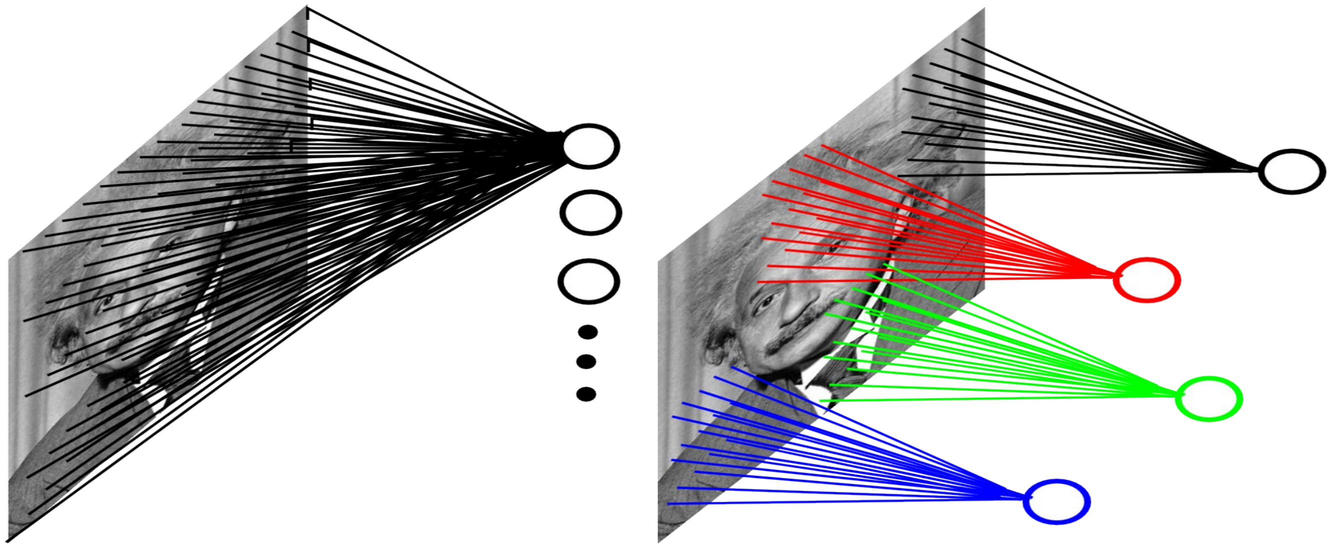


Figure courtesy: Yann LeCun

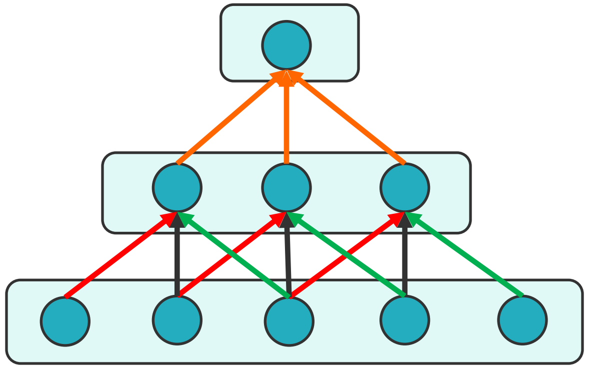
Convolutional Networks (ConvNets)

- Sparse connectivity
- Shared weights
- Increasingly "global" receptive fields
  - simple cells detect local features
  - complex cells "pool" the outputs of simple cells within a retinotopic neighborhood.

Feature maps $m+1$

Feature maps m

Feature maps $m-1$



Convolutional Networks (ConvNets)

- Hierarchical Representation Learning [Zeiler & Fergus 2013]

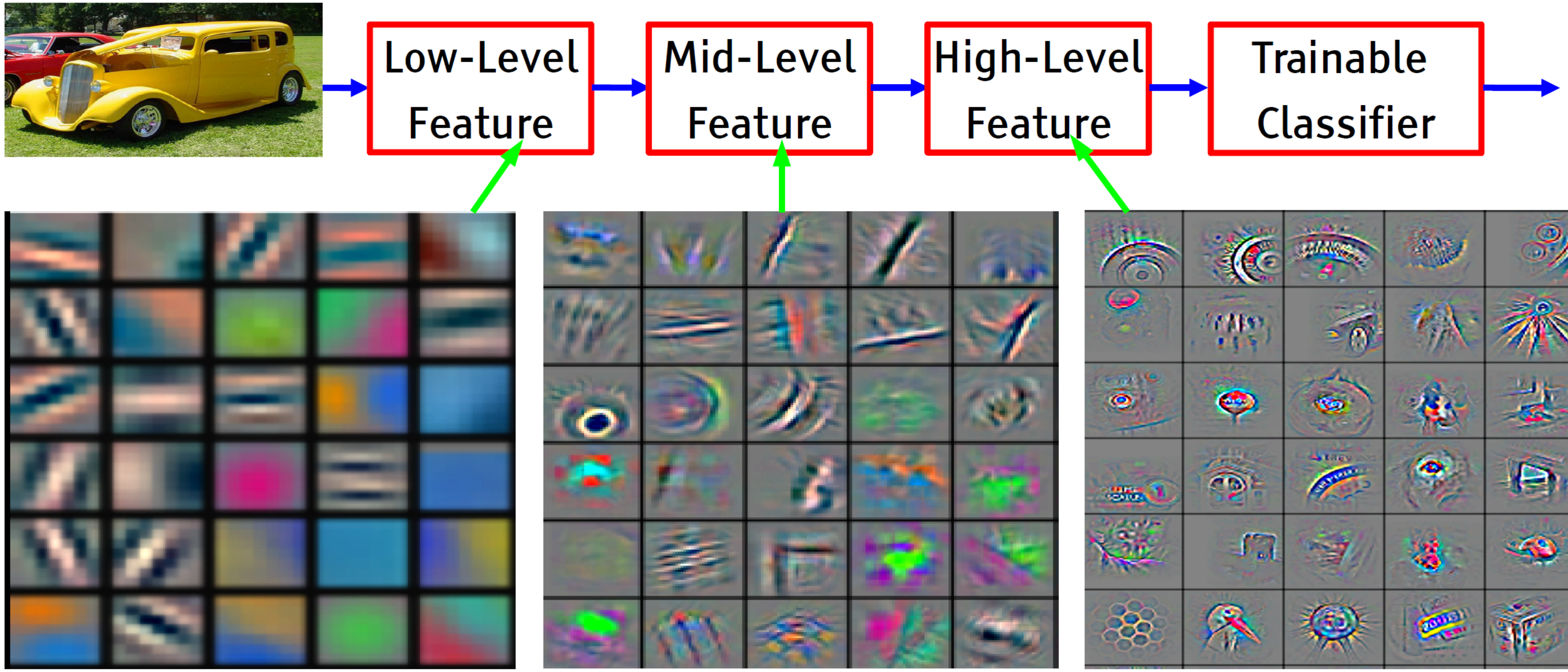
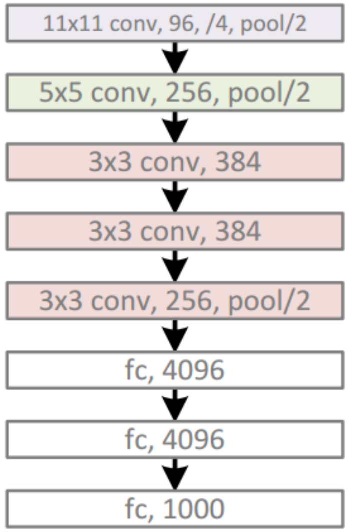


Figure courtesy: Yann LeCun

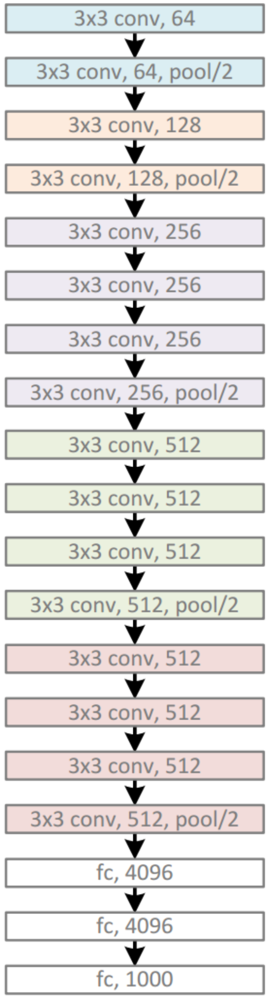
Evolution of ConvNets

AlexNet, 8 layers



2012

VGG, 19 layers

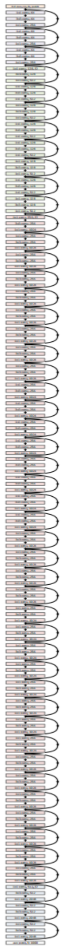


2015

GoogleNet, 22 layers

2014

ResNet, 152 layers



2015

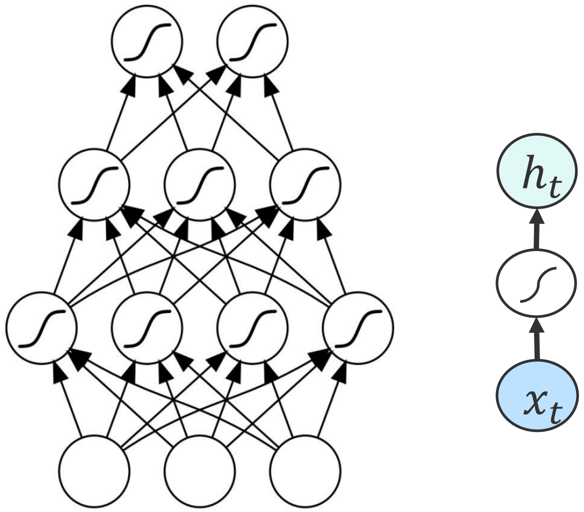
Figure courtesy: Kaiming He

Outline

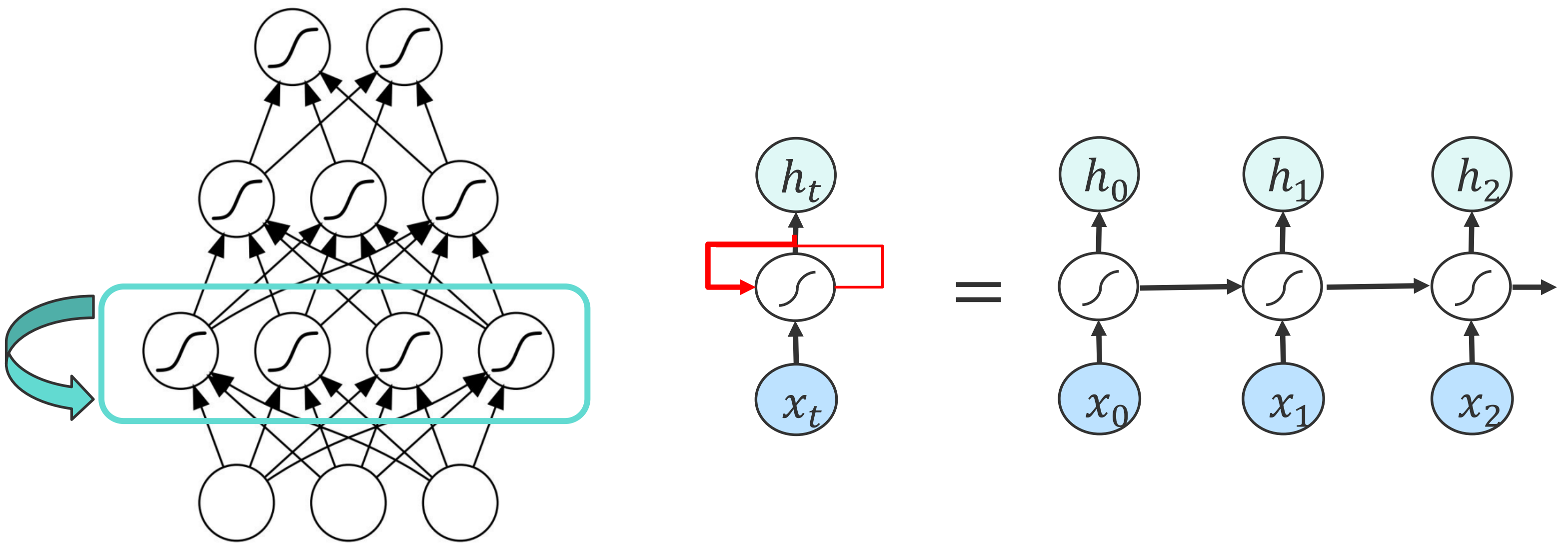
- Convolutional Networks (ConvNets)
- Recurrent Networks (RNNs)
  - Long-range dependency, vanishing
  - LSTM
  - RNNs in different forms
- Attention Mechanisms
  - (Query, Key, Value)
  - Attention on Text and Images
- Transformers: Multi-head Attention
  - Transformer
  - BERT

ConvNets v.s. Recurrent Networks (RNNs)

- Spatial Modeling *vs.* Sequential Modeling
- Fixed *vs.* variable number of computation steps.



The output depends ONLY on the current input



The hidden layers and the output additionally depend on previous states of the hidden layers

RNNs in Various Forms

One to One

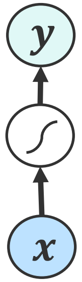


Image classification

One to Many

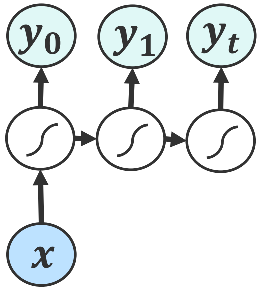
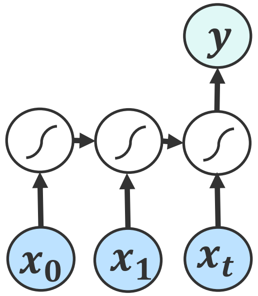


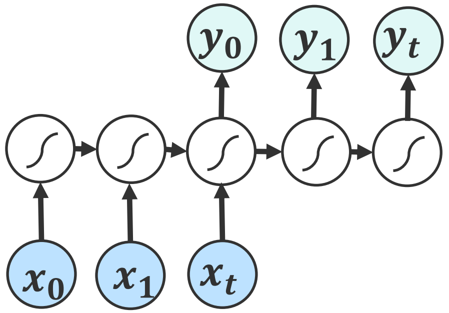
Image captioning

Many to One



Sentence sentiment analysis / Video recognition

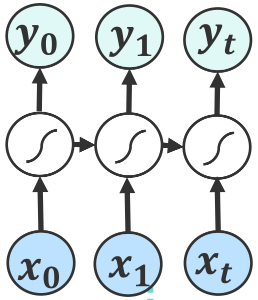
Many to Many



Machine Translation

(Sequence-to-sequence)

Many to Many

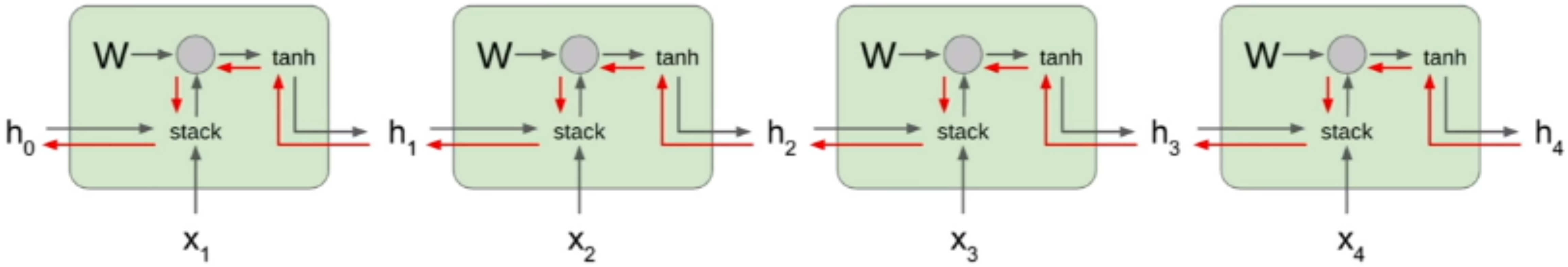


Named Entity Recognition

(Sequence tagging)

Vanishing / Exploding Gradients in RNNs

$$\boldsymbol{h}_t = tanh(W^{hh}\boldsymbol{h}_{t-1} + W^{hx}\boldsymbol{x}_t)$$

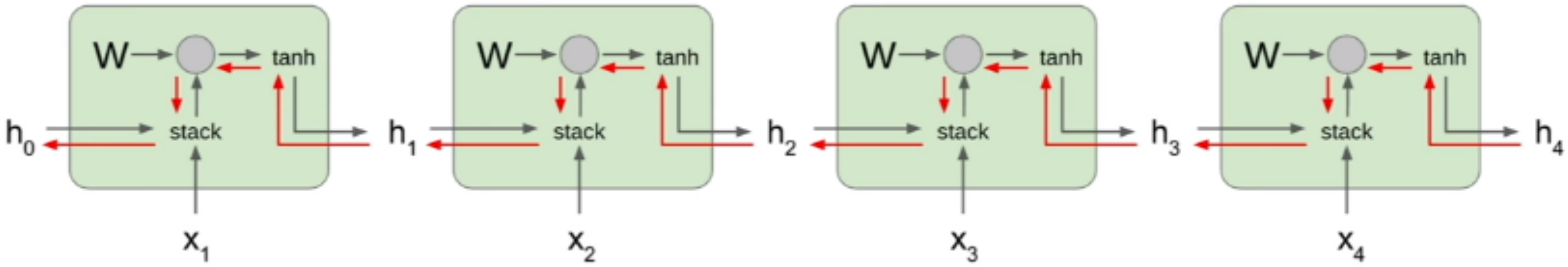


Bengio et al., 1994 “Learning long-term dependencies with gradient descent is difficult”
Pascanu et al., 2013 “On the difficulty of training recurrent neural networks”

Source: CS231N Stanford

Vanishing / Exploding Gradients in RNNs

$$\boldsymbol{h}_t = tanh(W^{hh}\boldsymbol{h}_{t-1} + W^{hx}\boldsymbol{x}_t)$$



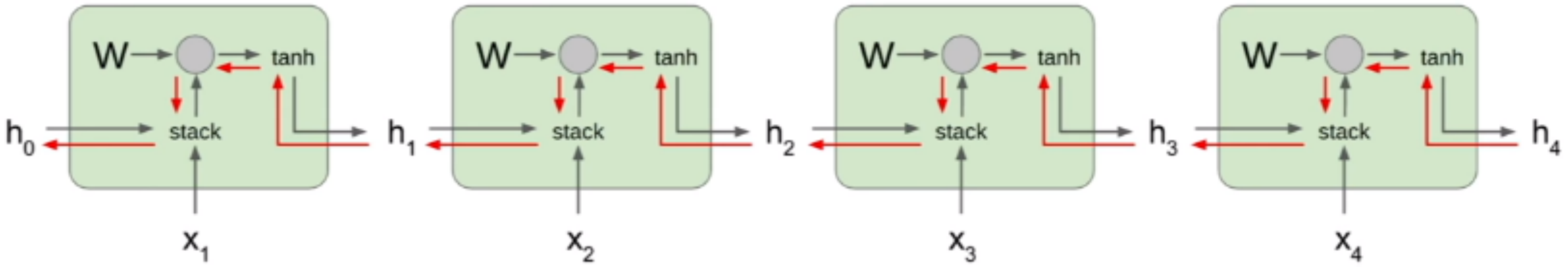
Computing gradient of h_0 involves many factors of W (and repeated tanh)

Source: CS231N Stanford

Bengio et al., 1994 "Learning long-term dependencies with gradient descent is difficult"
Pascanu et al., 2013 "On the difficulty of training recurrent neural networks"

Vanishing / Exploding Gradients in RNNs

$$\boldsymbol{h}_t = tanh(W^{hh}\boldsymbol{h}_{t-1} + W^{hx}\boldsymbol{x}_t)$$



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

→ **Gradient clipping**: Scale gradient if its norm is too big

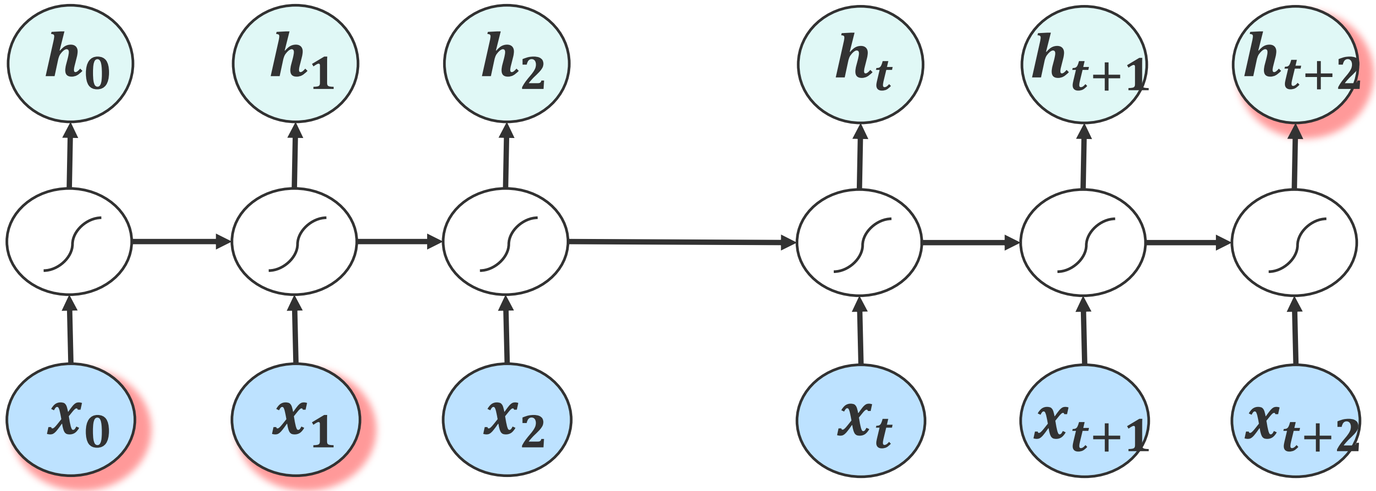
```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
    grad *= (threshold / grad_norm)
```

Largest singular value < 1:
Vanishing gradients

Source: CS231N Stanford

Bengio et al., 1994 "Learning long-term dependencies with gradient descent is difficult"
Pascanu et al., 2013 "On the difficulty of training recurrent neural networks"

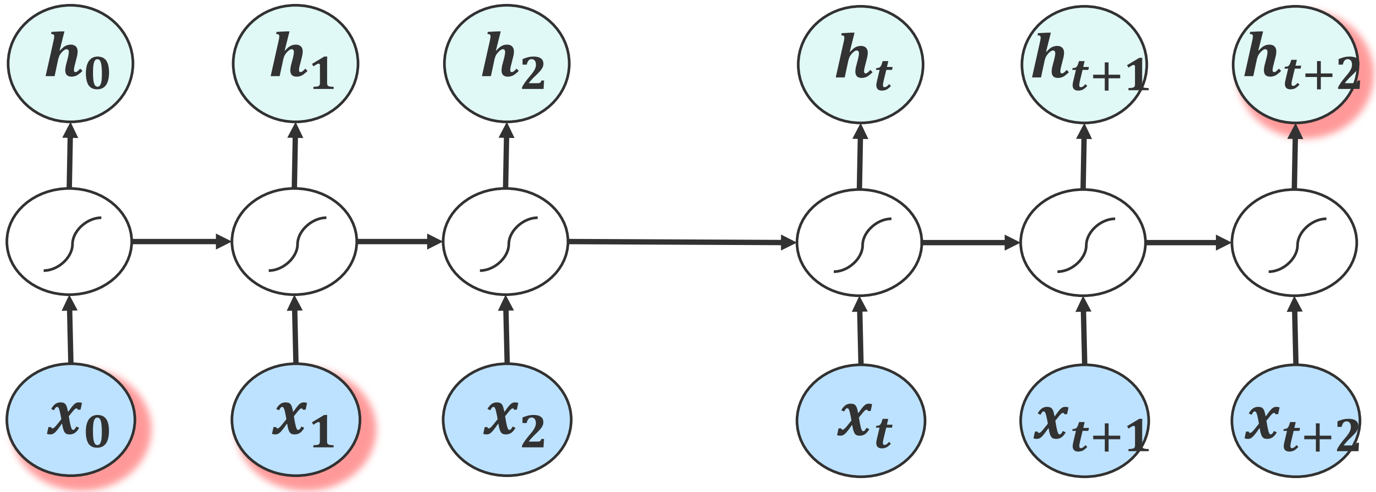
Long-term Dependency Problem



I live in France and I know ___________

Example courtesy: Manik Soni

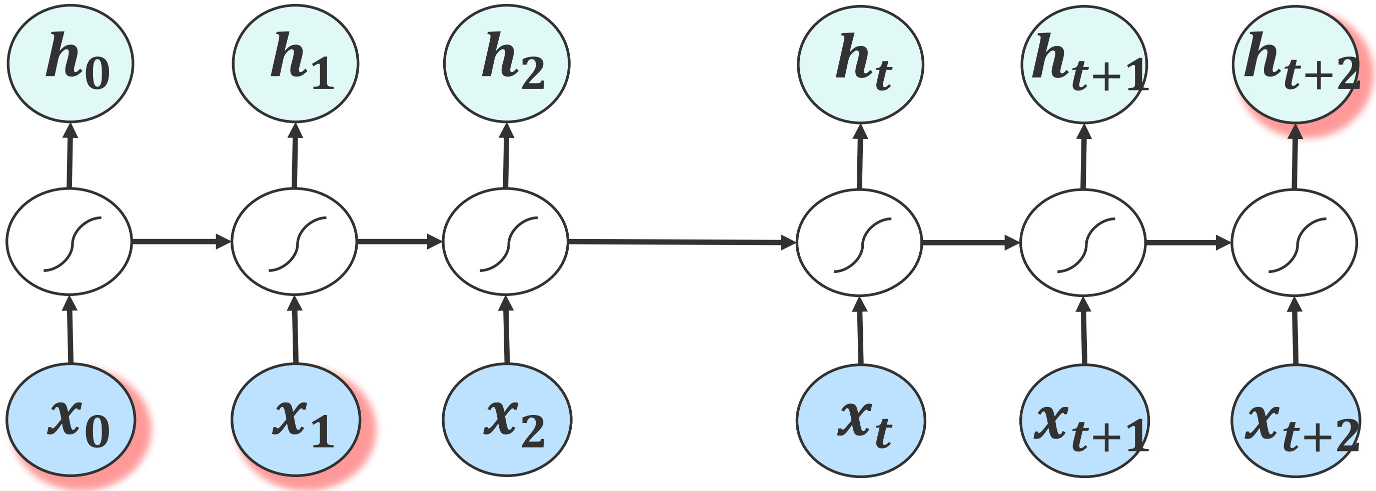
Long-term Dependency Problem



I live in France and I know ___*French*___

Example courtesy: Manik Soni

Long-term Dependency Problem



I live in France and I know __French__

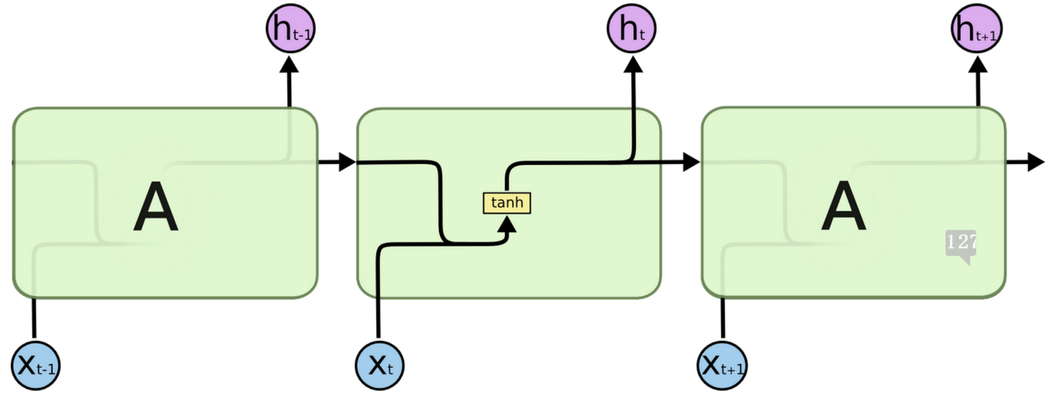
I live in France, a beautiful country, and I know __French__

Example courtesy: Manik Soni

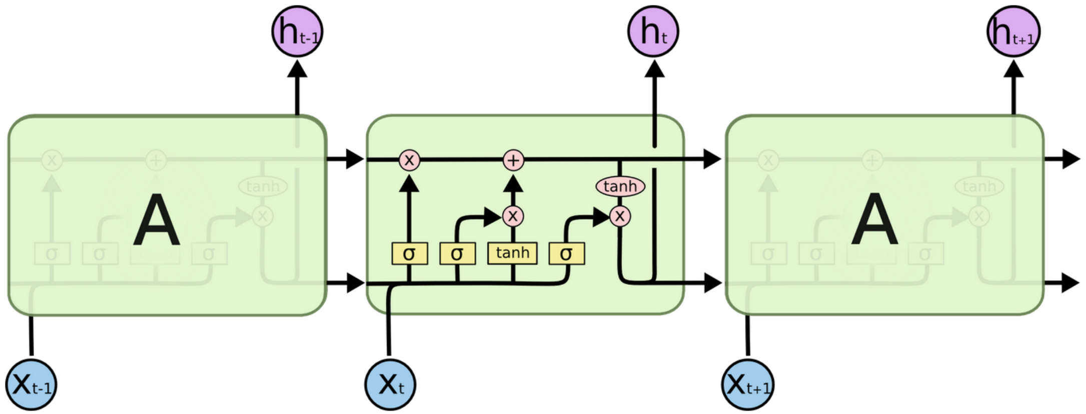
Long Short Term Memory (LSTM)

- LSTMs are designed to explicitly alleviate the long-term dependency problem [Horchreiter & Schmidhuber (1997)]

Standard RNN

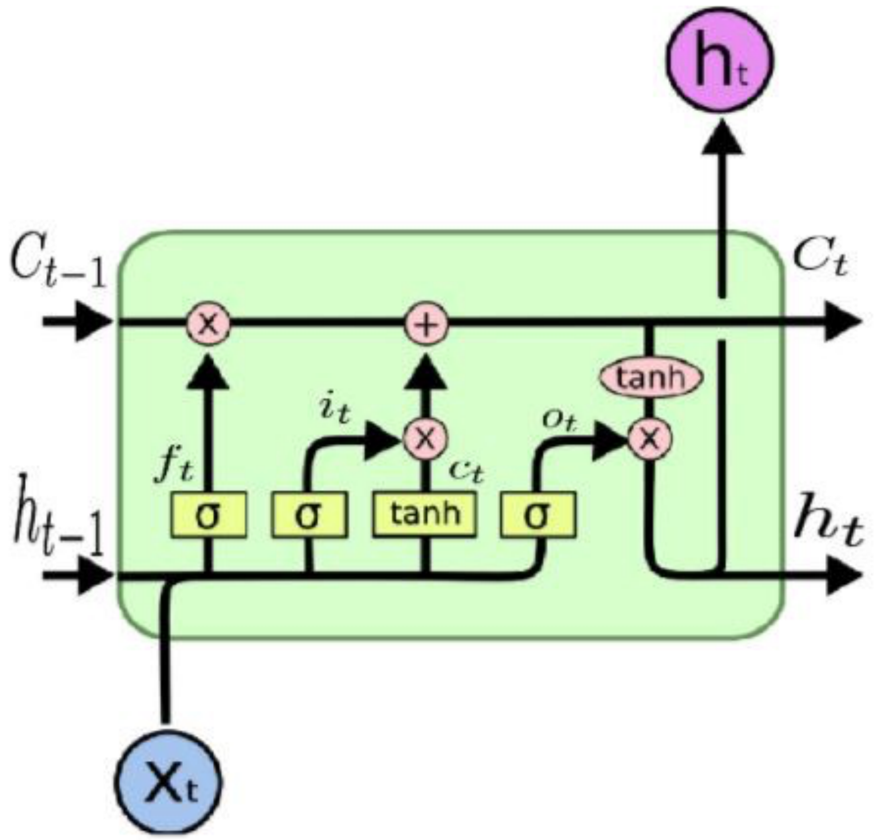


LSTM



Long Short Term Memory (LSTM)

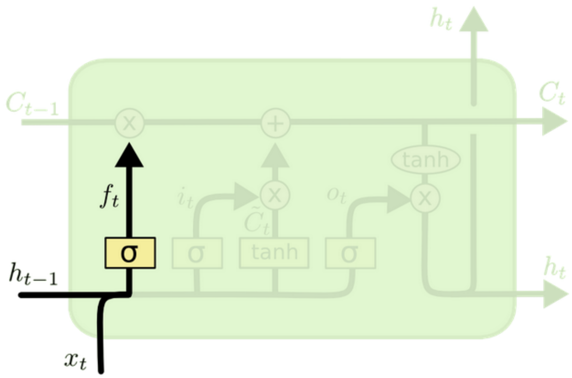
- Gate functions make decisions of reading, writing, and resetting information



- Forget gate: whether to erase cell (reset)
- Input gate: whether to write to cell (write)
- Output gate: how much to reveal cell (read)

Long Short Term Memory (LSTM)

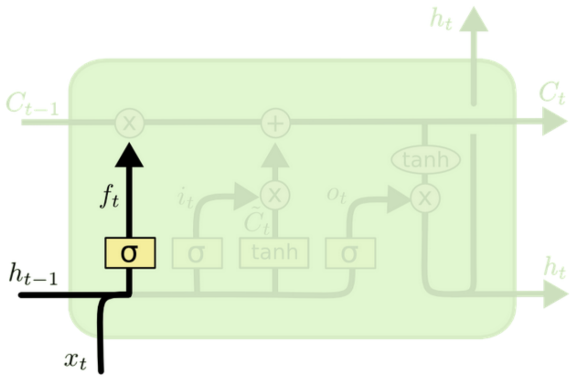
- **Forget gate:** decides what must be removed from $\boldsymbol{h_{t-1}}$



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

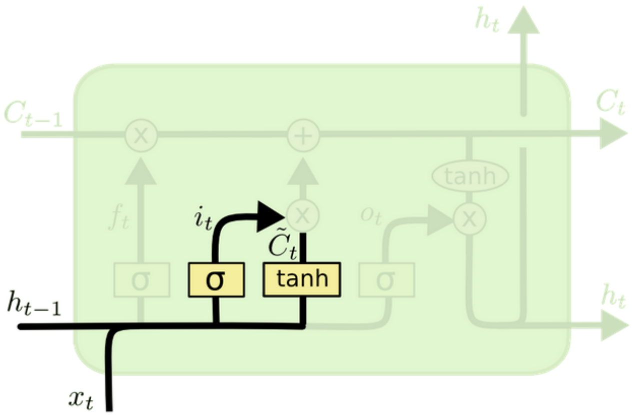
Long Short Term Memory (LSTM)

- **Forget gate:** decides what must be removed from $\boldsymbol{h}_{t-1}$



$$\boldsymbol{f}_t = \boldsymbol{\sigma}(\boldsymbol{W}_f \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_f)$$

- **Input gate:** decides what new information to store in the cell

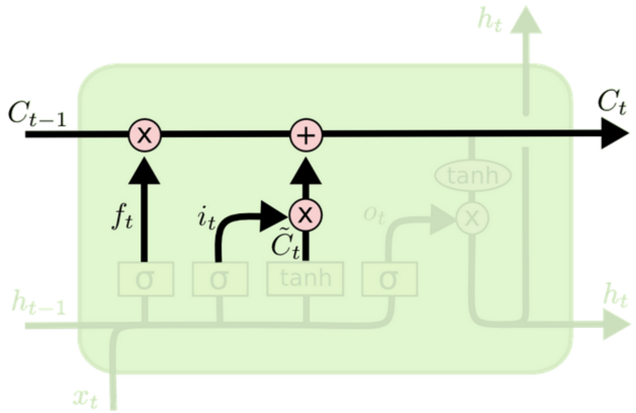


$$\boldsymbol{i}_t = \boldsymbol{\sigma}(\boldsymbol{W}_i \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_i)$$

$$\widetilde{\boldsymbol{C}}_t = \tanh(\boldsymbol{W}_C \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_C)$$

Long Short Term Memory (LSTM)

- Update cell state:



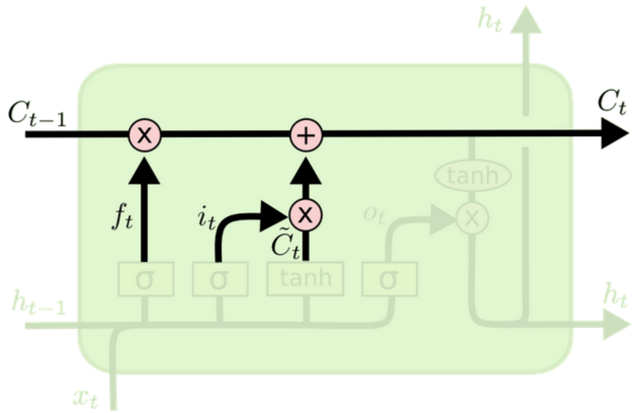
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

forgetting unneeded things

scaling the new candidate values by how much we decided to update each state value.

Long Short Term Memory (LSTM)

- Update cell state:

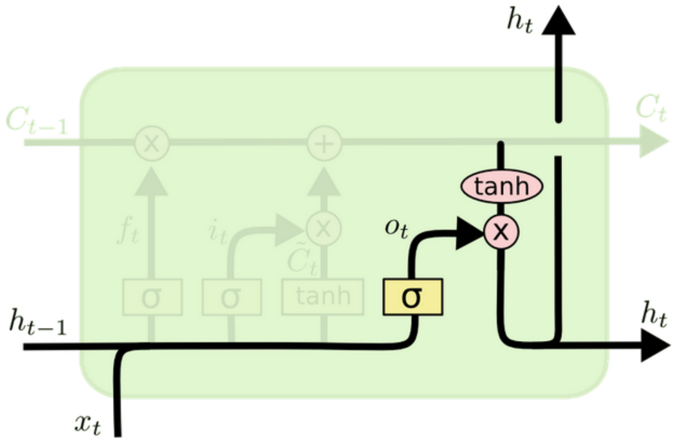


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

forgetting unneeded things

scaling the new candidate values by how much we decided to update each state value.

- Output gate: decides what to output from our cell state

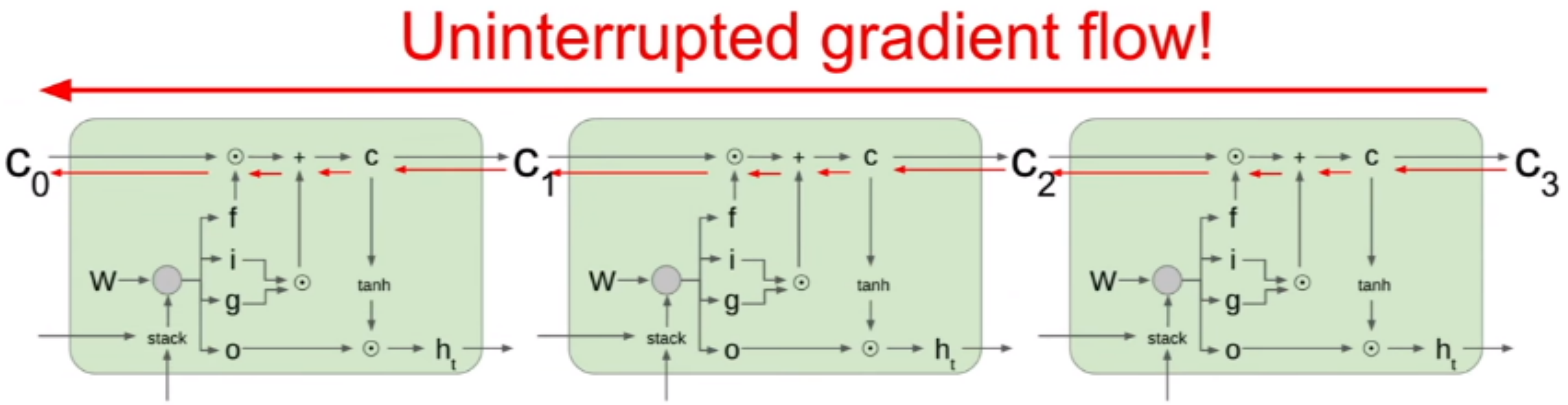


$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

sigmoid decides what parts of the cell state we're going to output

Backpropagation in LSTM



- No multiplication with matrix W during backprop
- Multiplied by different values of forget gate -> less prone to vanishing/exploding gradient

Source: CS231N Stanford

RNNs in Various Forms

One to One

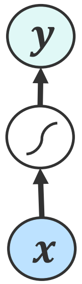


Image classification

One to Many

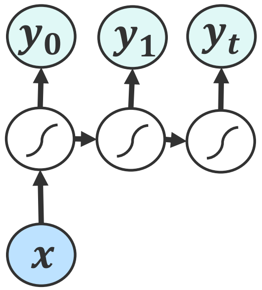
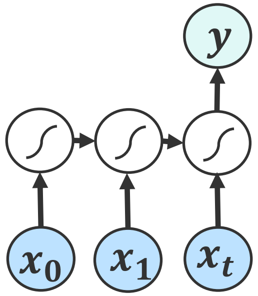


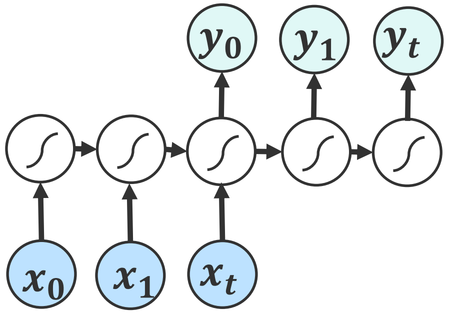
Image captioning

Many to One



Sentence sentiment analysis / Video recognition

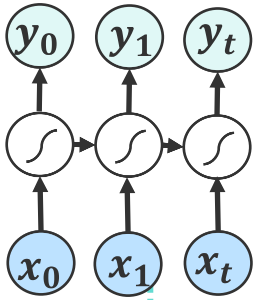
Many to Many



Machine Translation

(Sequence-to-sequence)

Many to Many

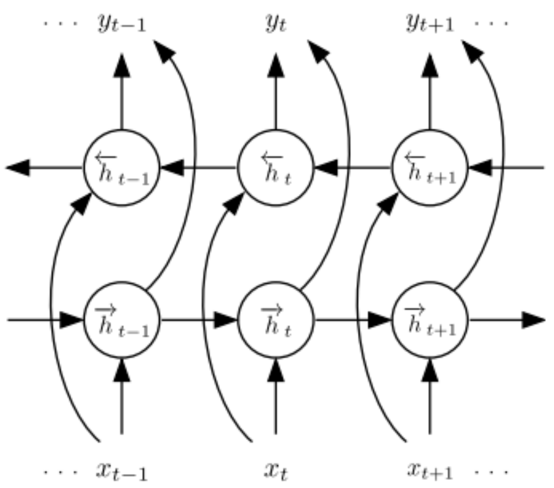


Named Entity Recognition

(Sequence tagging)

RNNs in Various Forms

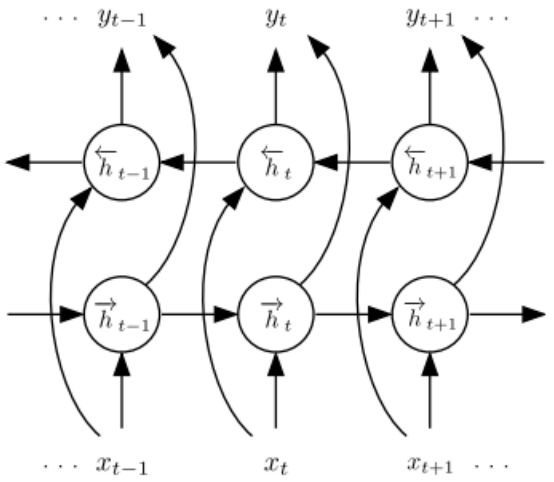
- Bi-directional RNN
  - Hidden state is the concatenation of both forward and backward hidden states.
  - Allows the hidden state to capture both past and future information.



[Speech Recognition with Deep Recurrent Neural Networks, Alex Graves]

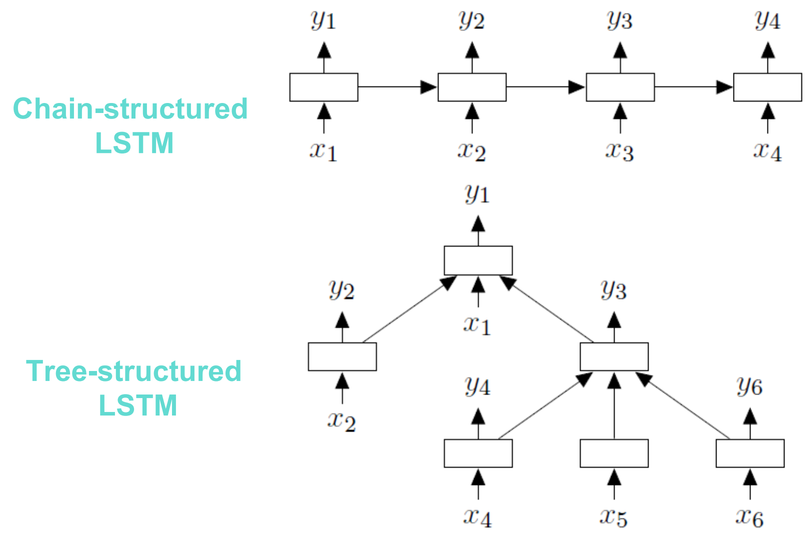
RNNs in Various Forms

- Bi-directional RNN
  - Hidden state is the concatenation of both forward and backward hidden states.
  - Allows the hidden state to capture both past and future information.



[Speech Recognition with Deep Recurrent Neural Networks, Alex Graves]

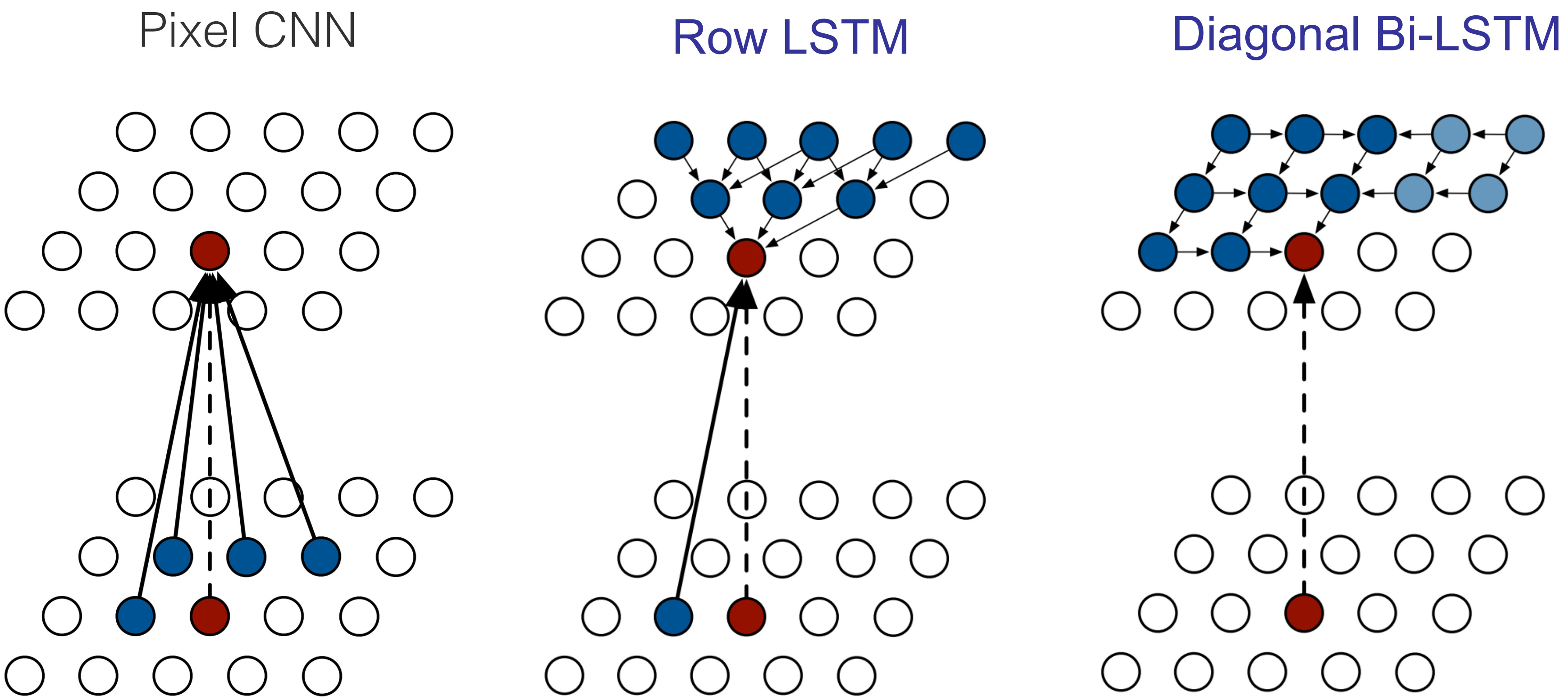
- Tree-structured RNN
  - Hidden states condition on both an input vector and the hidden states of arbitrarily many child units.
  - Standard LSTM = a special case of tree-LSTM where each internal node has exactly one child.



Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks, Tai. et al.

RNNs in Various Forms

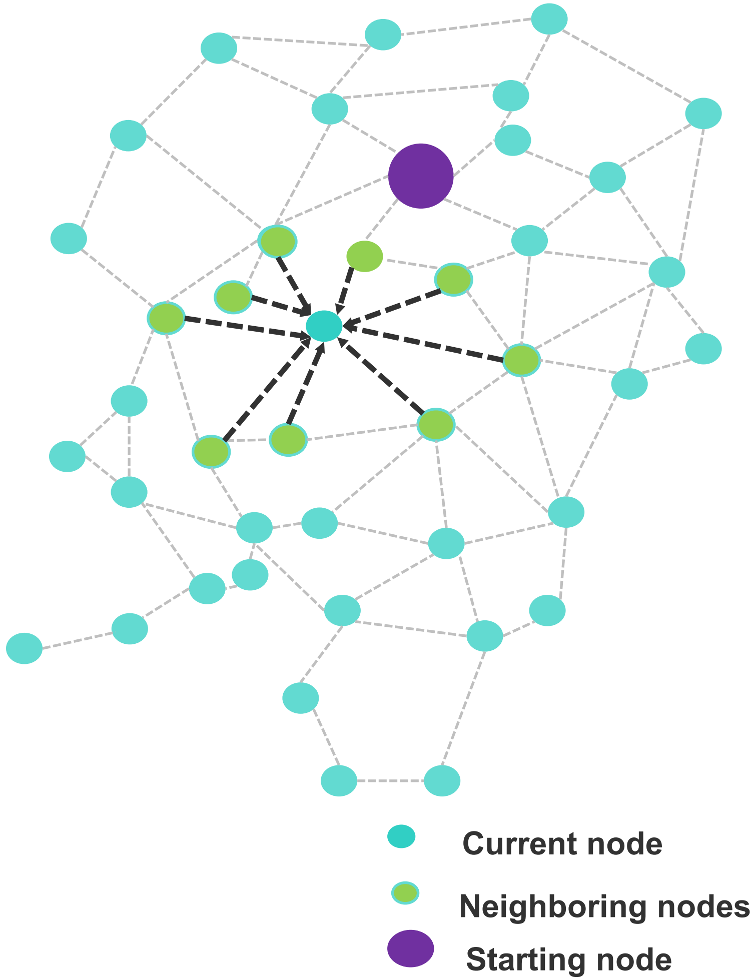
- RNN for 2-D sequences



[Pixel Recurrent Neural Networks, van den Oord. et al. 2016]

RNNs in Various Forms

- RNN for Graph Structures
  - Used in, e.g., image segmentation



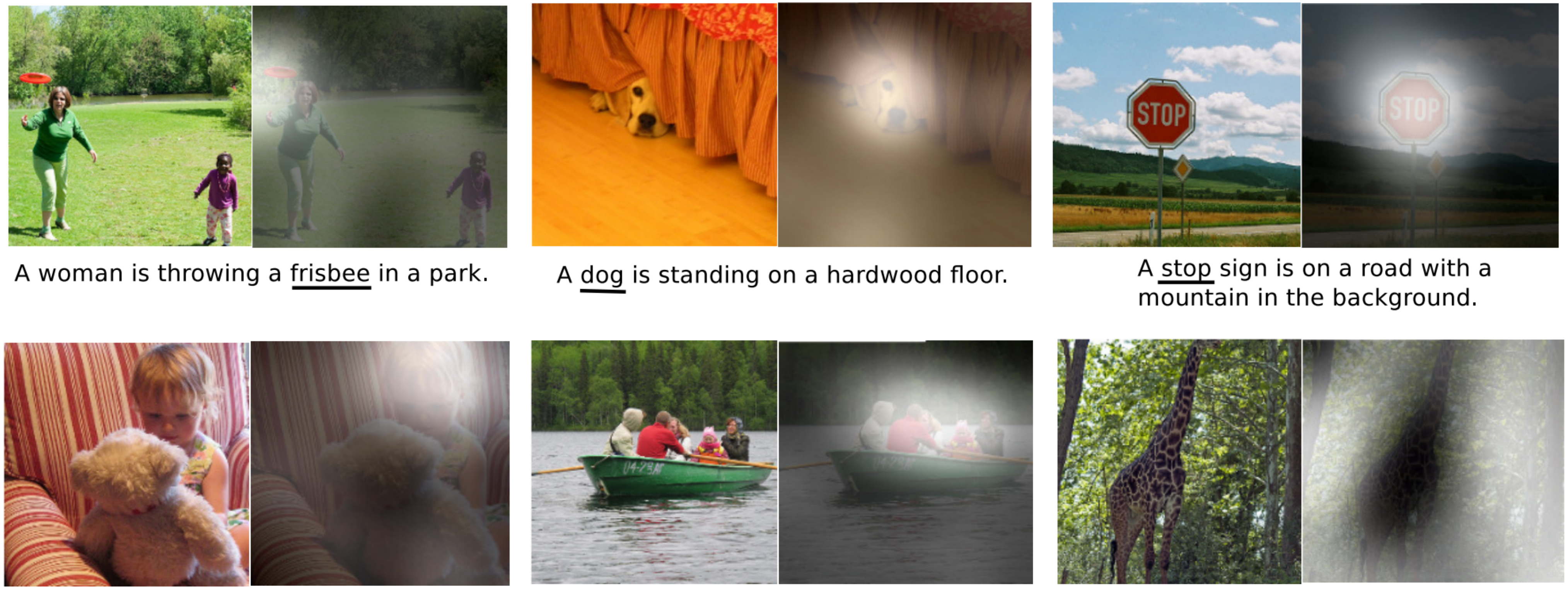
[Semantic Object Parsing with Graph LSTM. Liang et al. 2016]

Outline

- Convolutional Networks (ConvNets)
- Recurrent Networks (RNNs)
  - Long-range dependency, vanishing
  - LSTM
  - RNNs in different forms
- Attention Mechanisms
  - (Query, Key, Value)
  - Attention on Text and Images
- Transformers: Multi-head Attention
  - Transformer
  - BERT

Attention: Examples

- Chooses which features to pay attention to



A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.

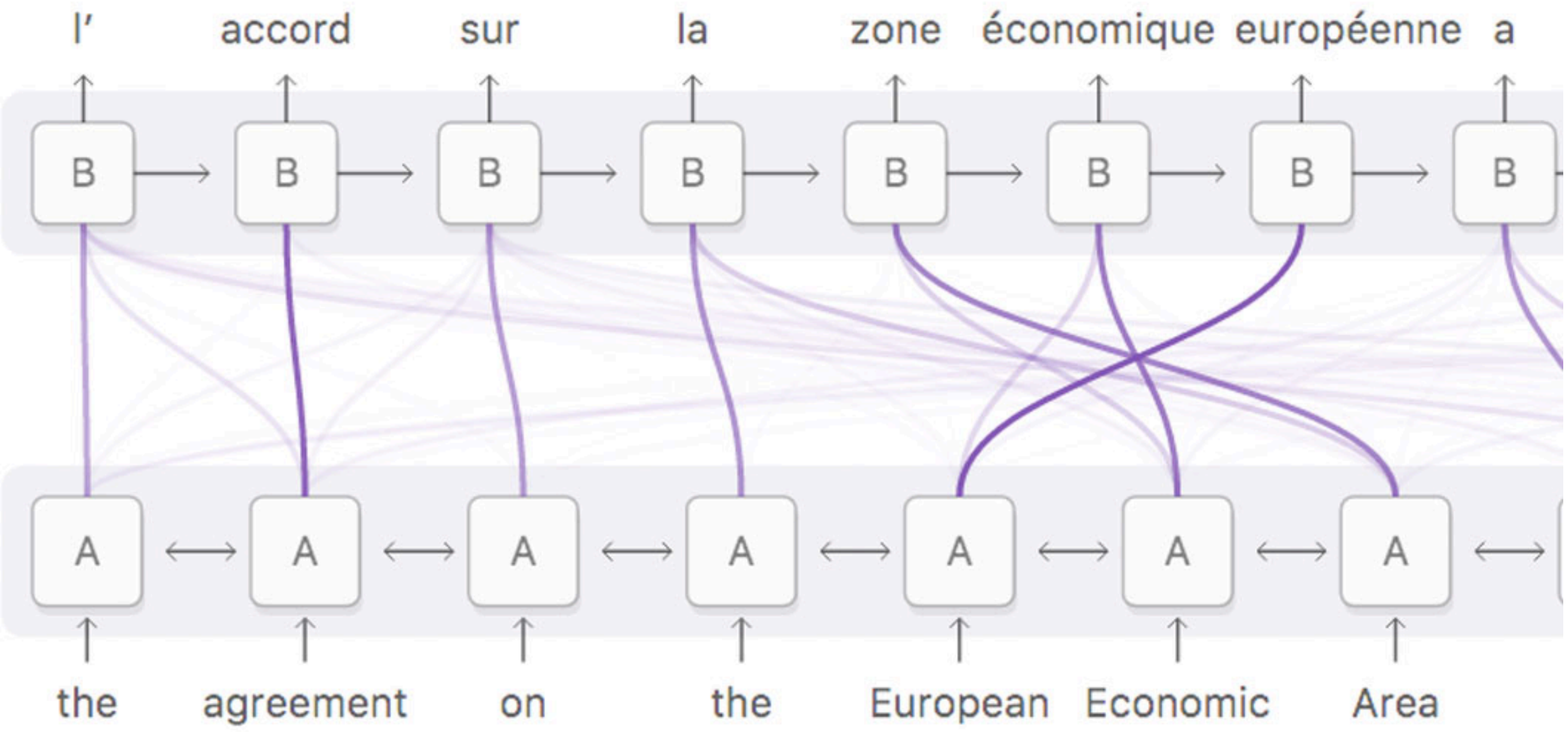
A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

Image captioning [Show, attend and tell. Xu et al. 15]

Attention: Examples

- Chooses which features to pay attention to



Machine Translation

Figure courtesy: Olah & Carter, 2016

Why Attention?

Figure courtesy: keitakurita

Why Attention?

- Long-range dependencies
  - Dealing with gradient vanishing problem

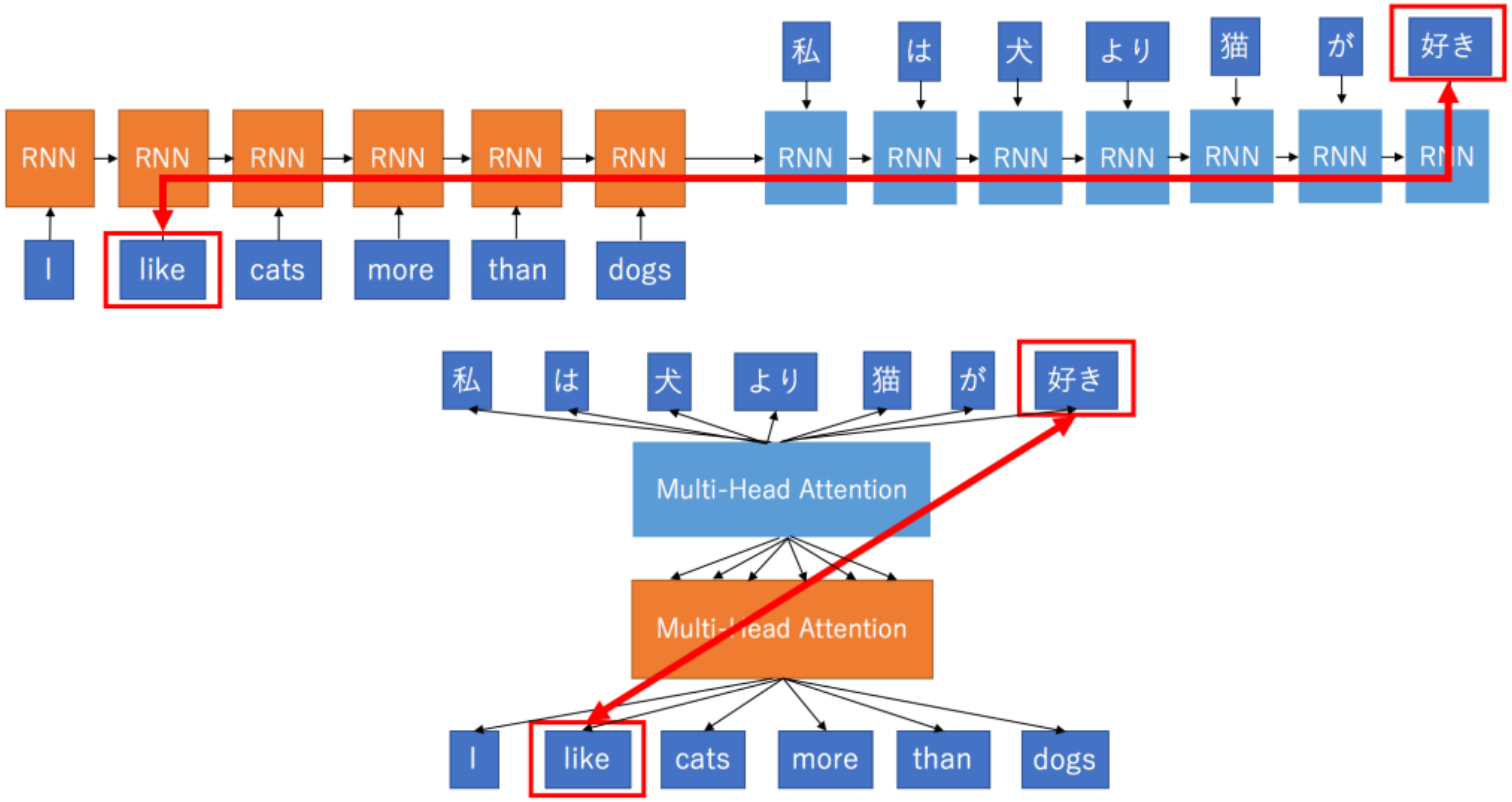


Figure courtesy: keitakurita

Why Attention?

- Long-range dependencies
  - Dealing with gradient vanishing problem
- Fine-grained representation instead of a single global representation
  - Attending to smaller parts of data: patches in images, words in sentences

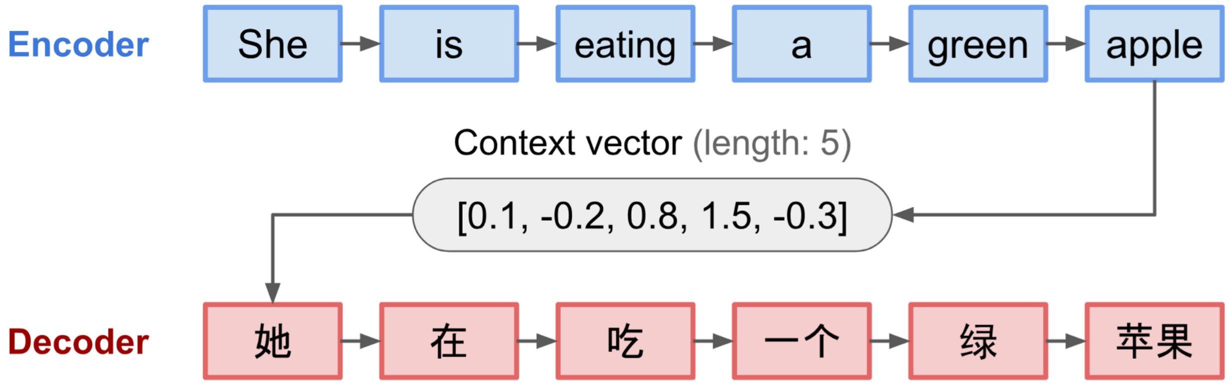


Figure courtesy: Lilian Weng

Why Attention?

- Long-range dependencies
  - Dealing with gradient vanishing problem
- Fine-grained representation instead of a single global representation
  - Attending to smaller parts of data: patches in images, words in sentences
- Improved Interpretability

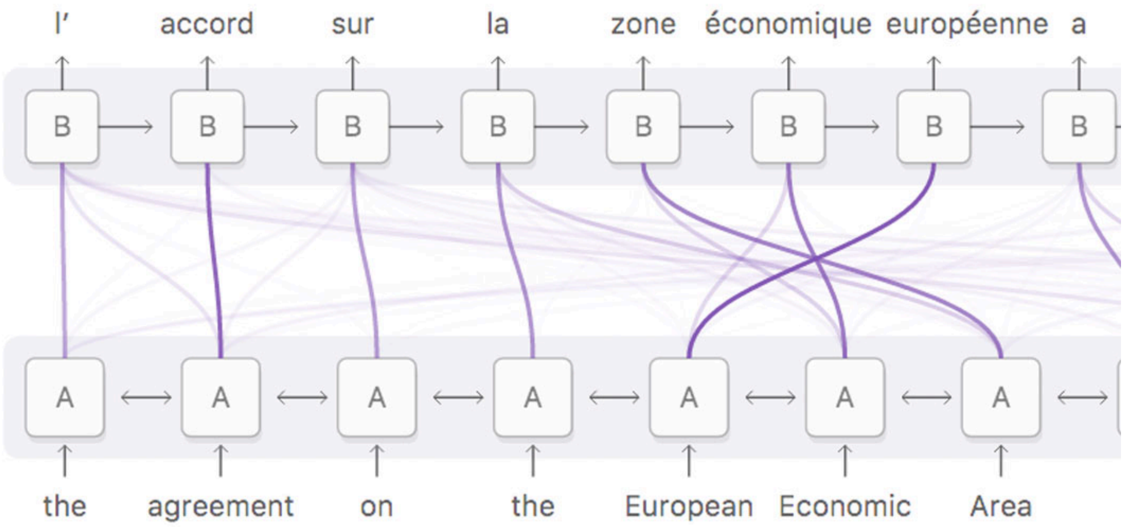


Figure courtesy: Olah & Carter, 2016

Attention Computation

- Encode each token in the input sentence into vectors
- When decoding, perform a linear combination of these vectors, weighted by “attention weights”
  - $\boldsymbol{a} = \mathrm{softmax}(\boldsymbol{alignment\_scores})$

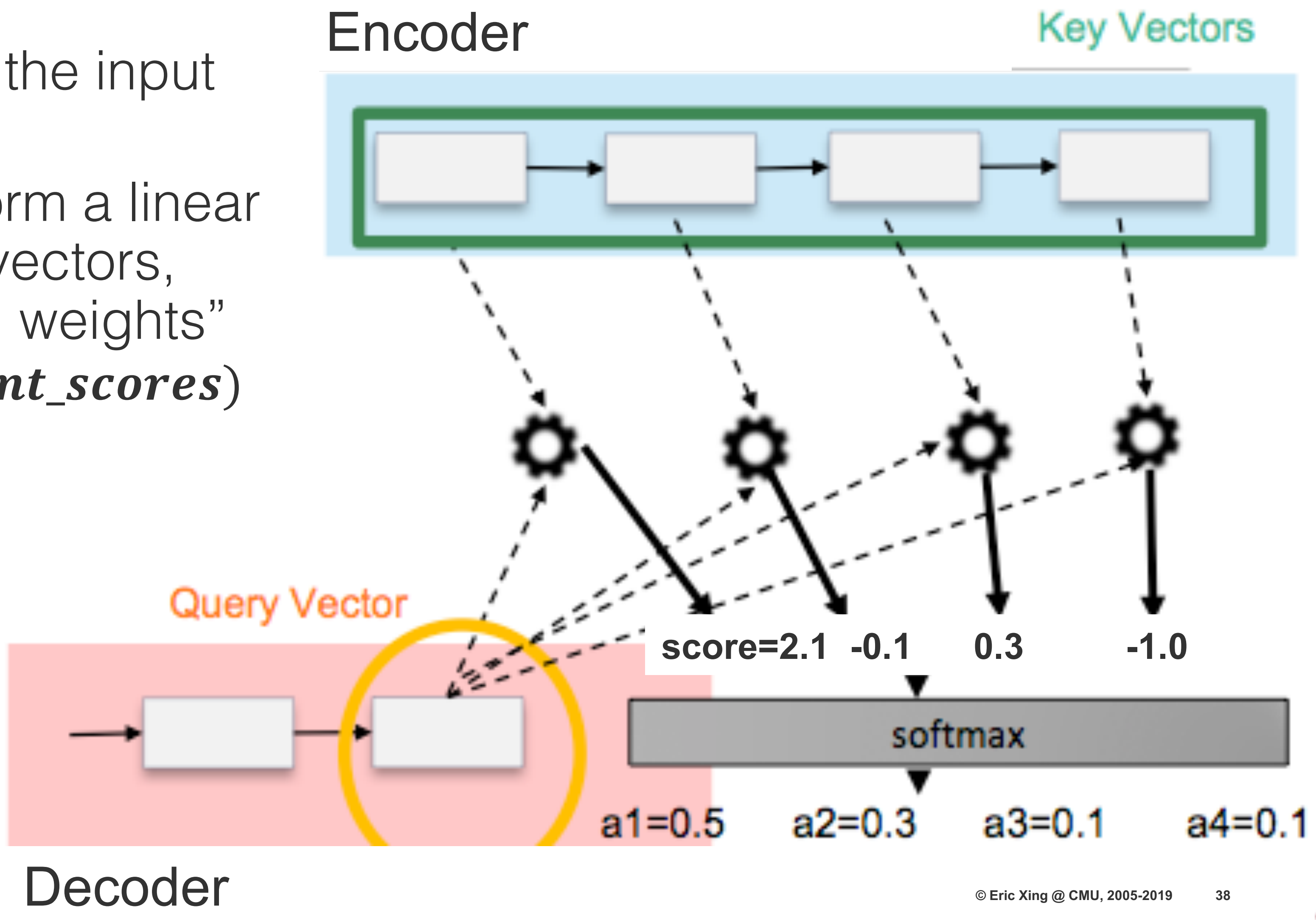
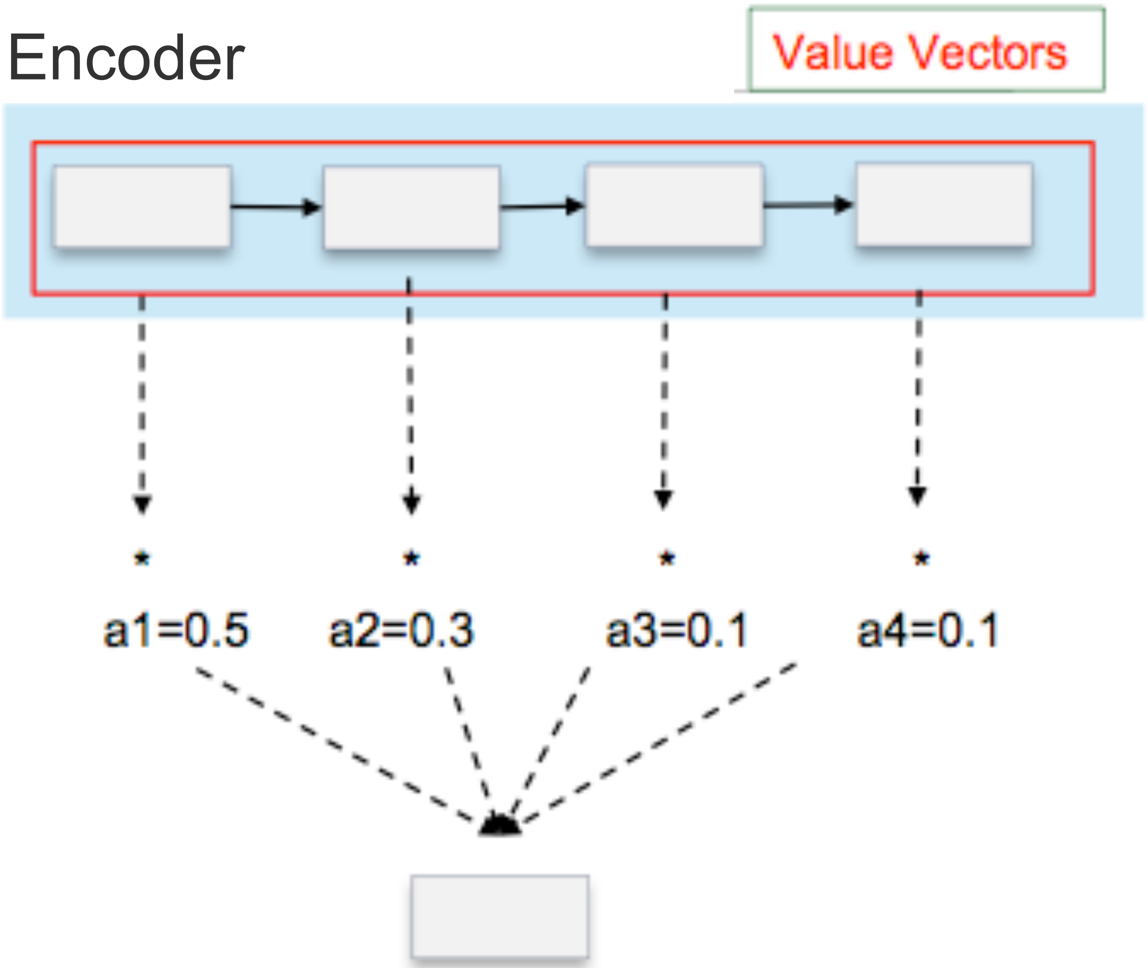


Figure courtesy: MARTA R. COSTA-JUSSÀ

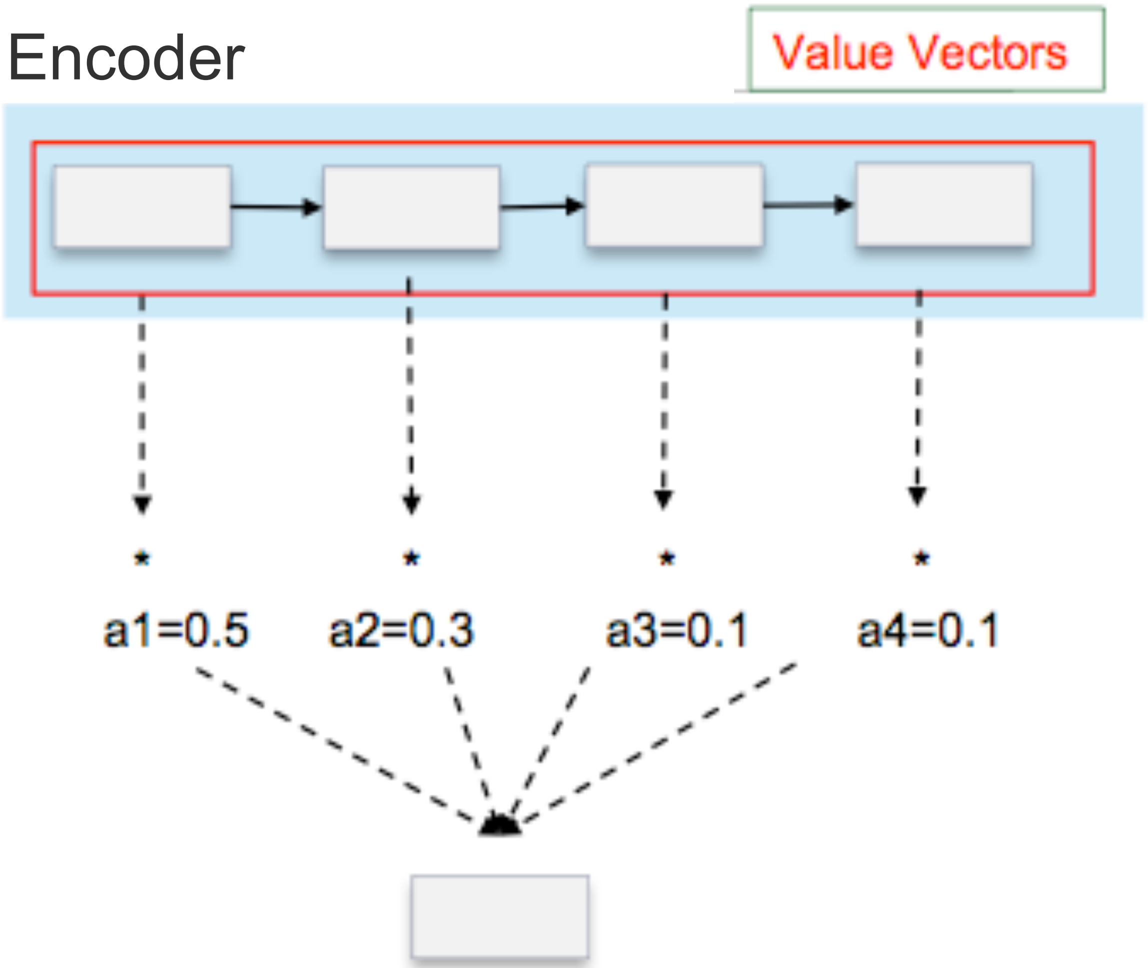
Attention Computation (cont'd)

- Combine together value by taking the weighted sum



Attention Computation (cont'd)

- Combine together value by taking the weighted sum



- **Query**: decoder state
- **Key**: all encoder states
- **Value**: all encoder states

Attention Variants

- Popular attention mechanisms with different alignment score functions

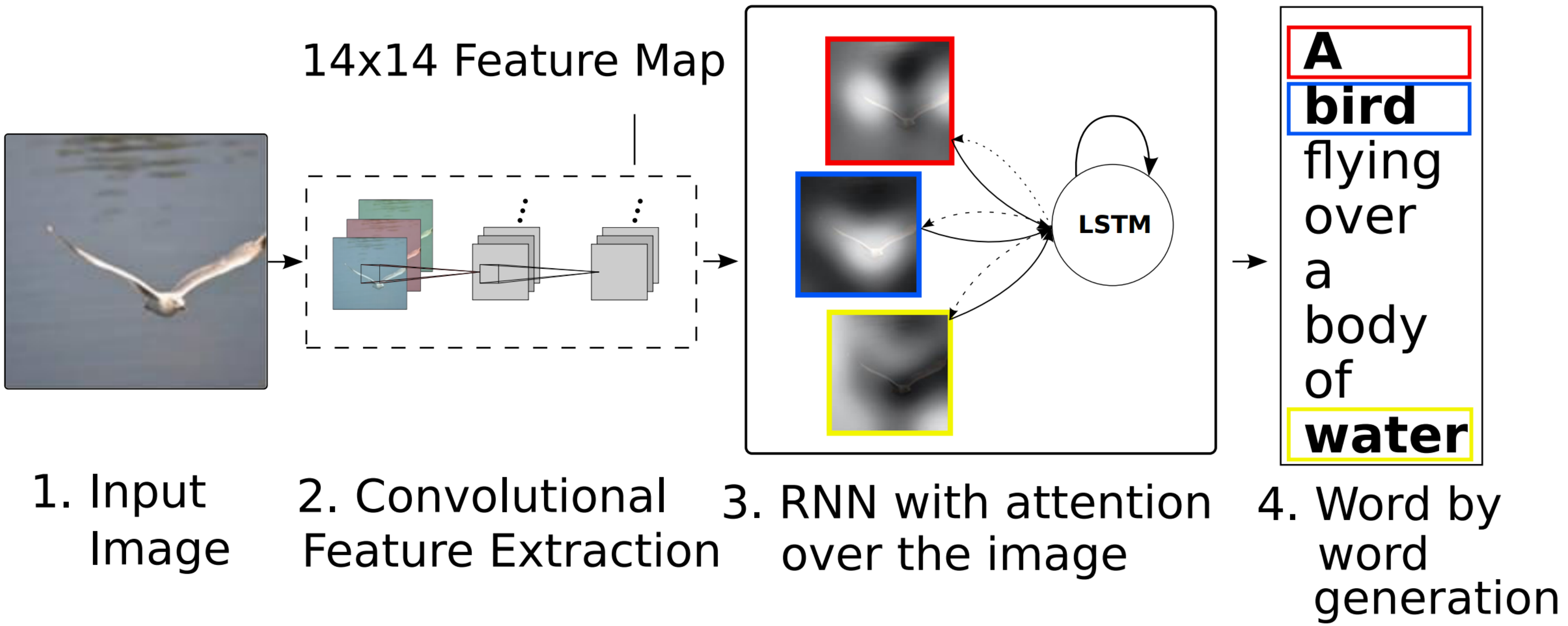
Alignment score = f(Query, Keys)

- **Query**: decoder state s_t
- **Key**: all encoder states h_i
- **Value**: all encoder states h_i

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(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$	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	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.	Luong2015
Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{s}_t^\top \boldsymbol{h}_i$	Luong2015
Scaled Dot-Product(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\mathbf{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017

Courtesy: Lilian Weng

Attention on Images – Image Captioning

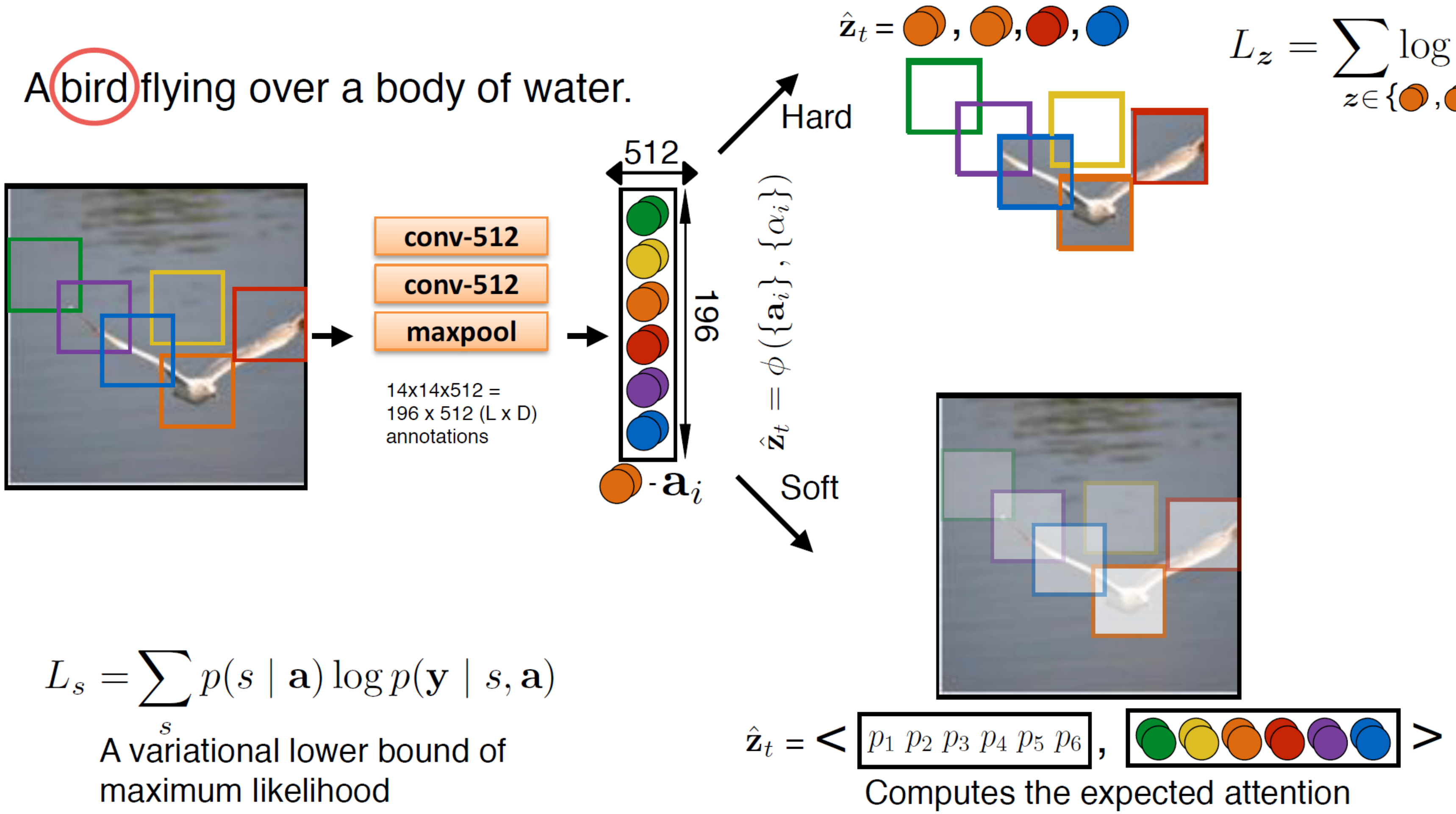


- **Query**: decoder state
- **Key**: visual feature maps
- **Value**: visual feature maps

[Show, attend and tell. Xu et al. 15]

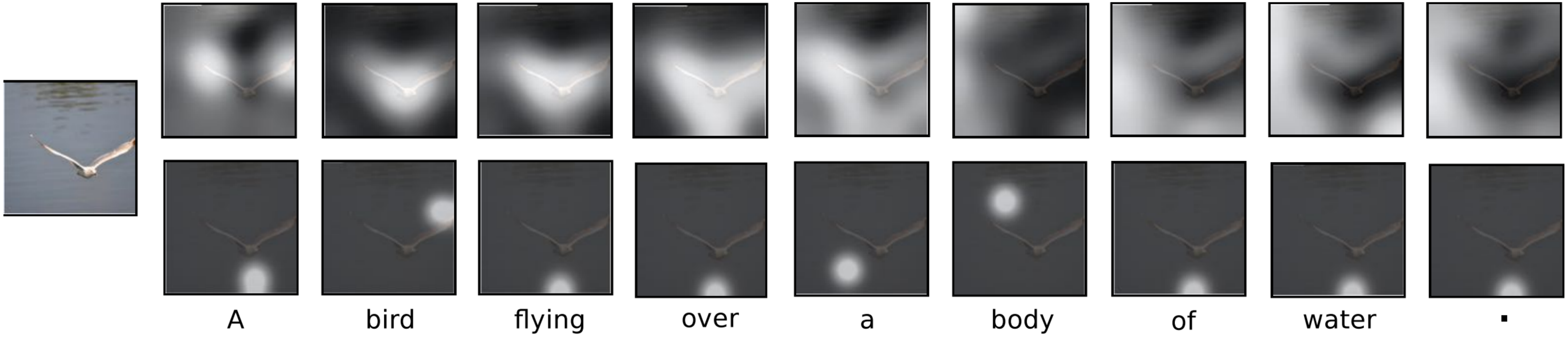
Attention on Images – Image Captioning

Hard attention *vs* Soft attention



Attention on Images – Image Captioning

Hard attention *vs* Soft attention



Attention on Images – Image Paragraph Generation

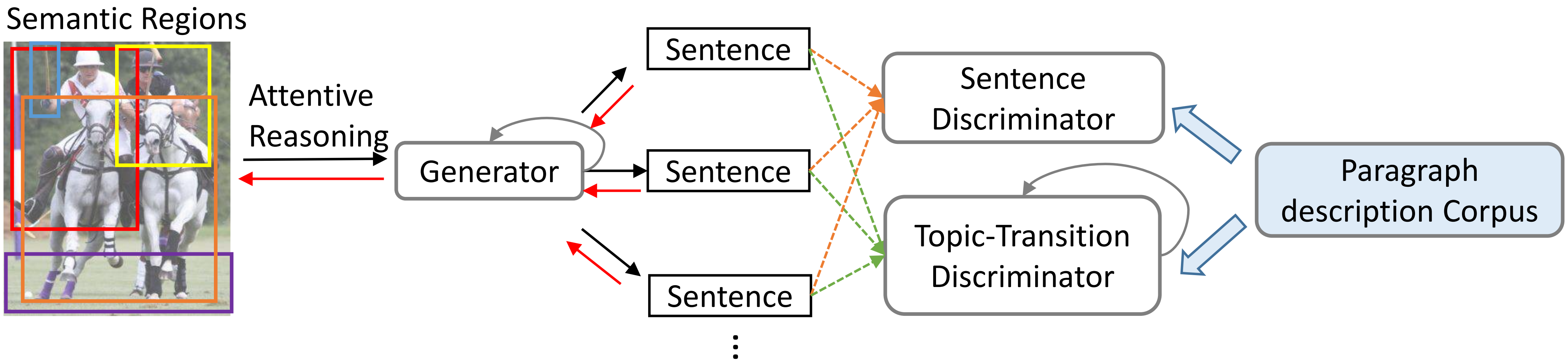
- Generate a long paragraph to describe an image
  - Long-term visual and language reasoning
  - Contentful descriptions -- ground sentences on visual features

This picture is taken for three baseball players on a field. The man on the left is wearing a blue baseball cap. The man has a red shirt and white pants. The man in the middle is in a wheelchair and holding a baseball bat. Two men are bending down behind a fence. There are words band on the fence.

A tennis player is attempting to hit the tennis ball with his left foot hand. He is holding a tennis racket. He is wearing a white shirt and white shorts. He has his right arm extended up. There is a crowd of people watching the game. A man is sitting on the chair.

A couple of zebra are standing next to each other on dirt ground near rocks. There are trees behind the zebras. There is a large log on the ground in front of the zebra. There is a large rock formation to the left of the zebra. There is a small hill near a small pond and a wooden log. There are green leaves on the tree.

Attention on Images – Image Paragraph Generation

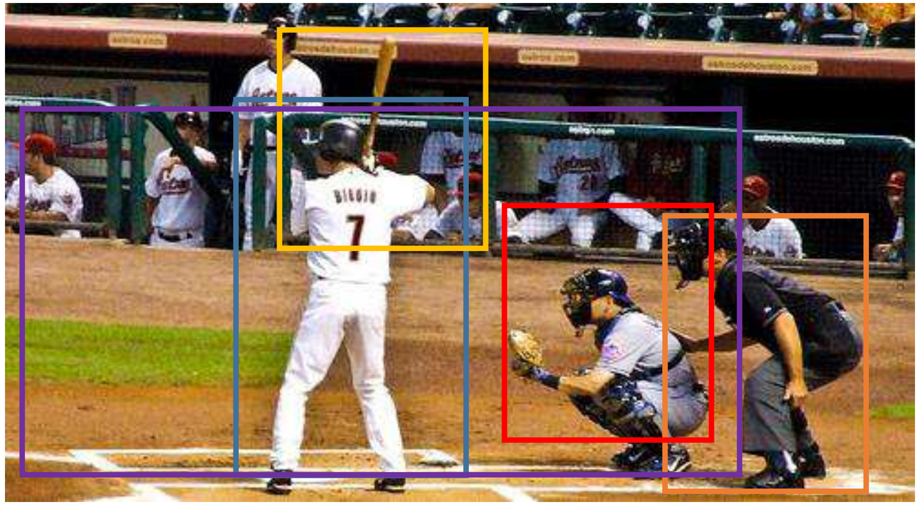


[Recurrent Topic-Transition GAN for Visual Paragraph Generation. Liang et al. 2017]

Attention on Images – Image Paragraph Generation

Semantic Regions

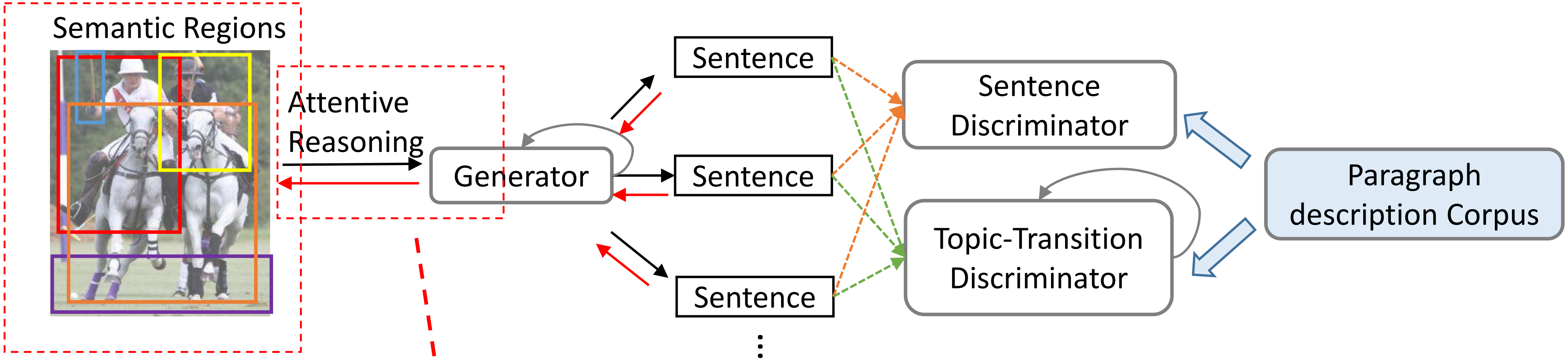
Semantic region detection & captioning



Local Phrases

- people playing baseball
- a man wearing white shirt and pants
- man holding a baseball bat
- person wearing a helmet in the field
- a man bending over

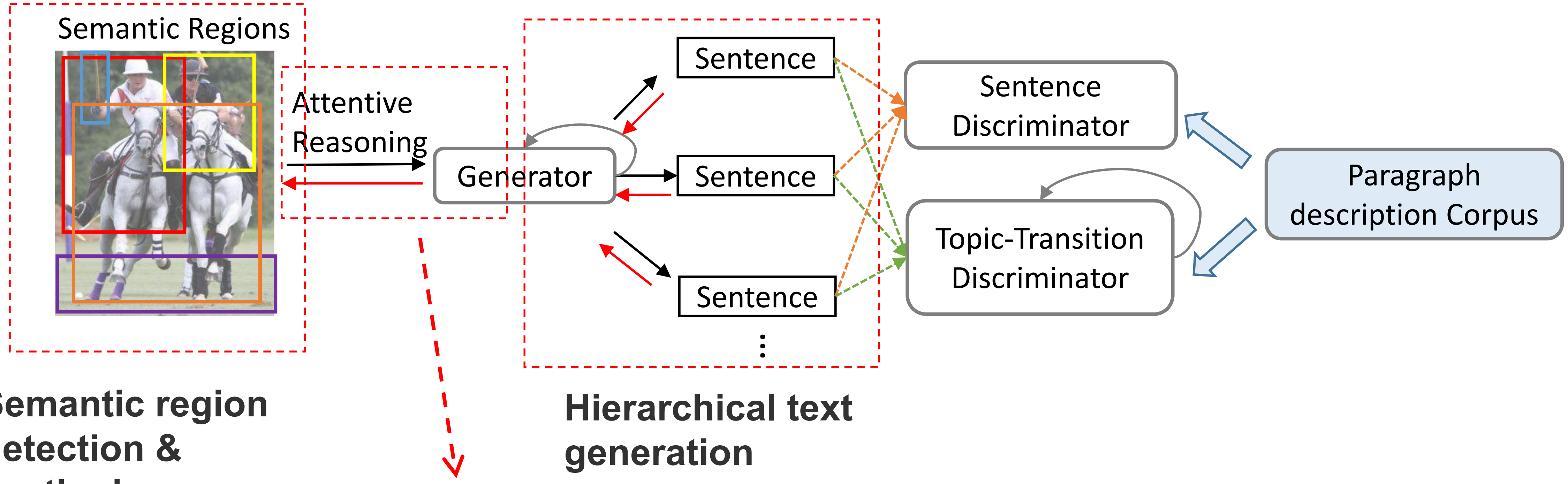
Attention on Images – Image Paragraph Generation



Semantic region detection & captioning

Attention on both visual regions and text phrases

Attention on Images – Image Paragraph Generation

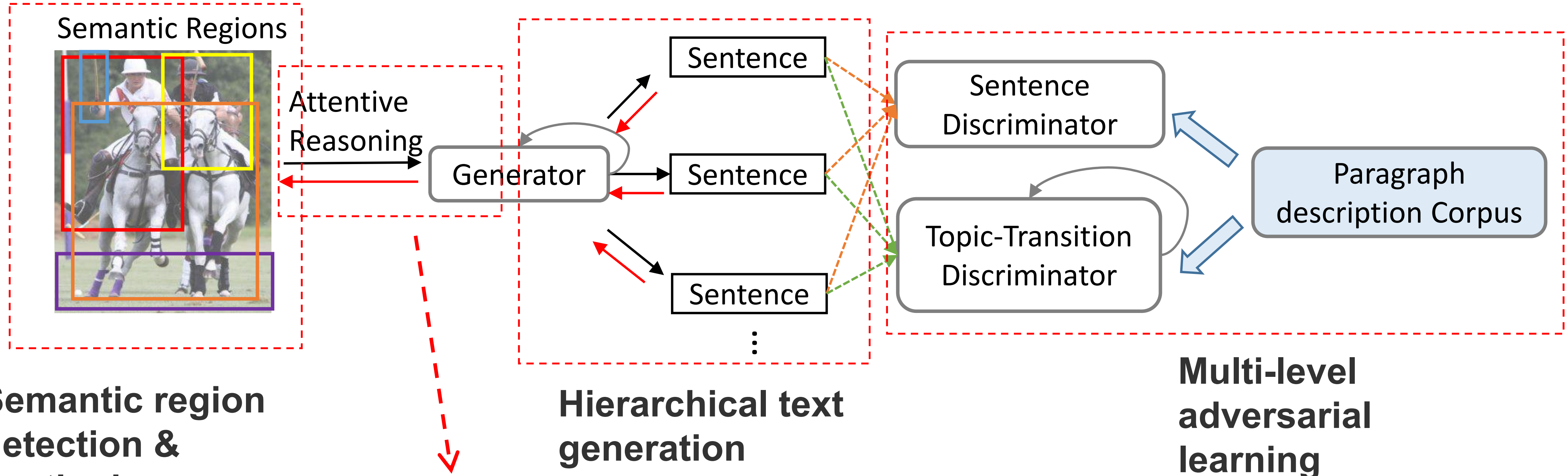


Semantic region detection & captioning

Attention on both visual regions and text phrases

Hierarchical text generation

Attention on Images – Image Paragraph Generation



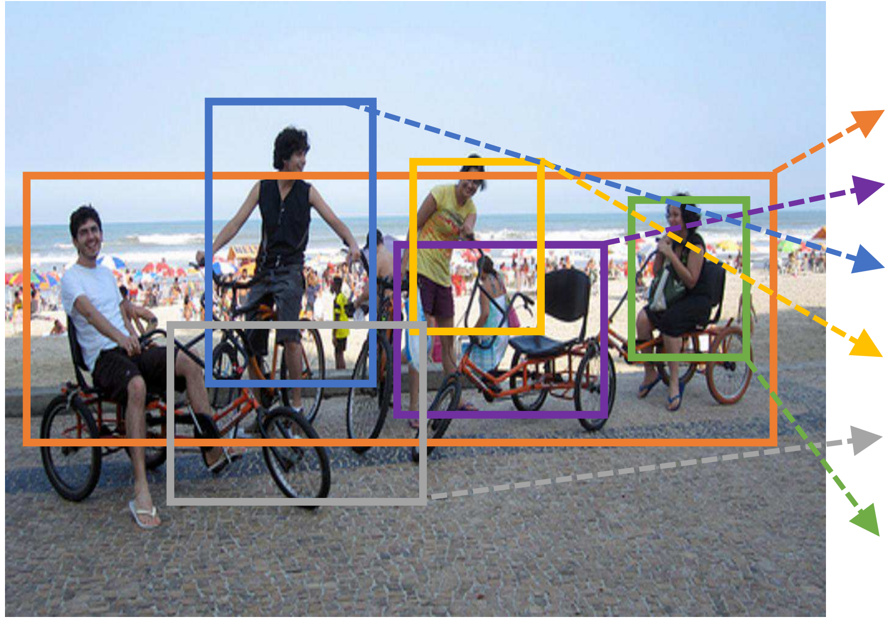
Semantic region detection & captioning

Attention on both visual regions and text phrases

Hierarchical text generation

Multi-level adversarial learning

Attention on Images – Image Paragraph Generation



1) people <u>riding</u> a bike
2) a bicycle parked on the <u>sidewalk</u>
3) man <u>wearing</u> a black shirt
4) a woman wearing a <u>yellow</u> shirt
5) a <u>red</u> and black bike
6) a woman wearing a <u>shirt</u>

Paragraph: *A group of people are riding bikes. There are two people riding bikes parked on the sidewalk. He is wearing a black shirt and jeans. A woman is wearing a short sleeve yellow shirt and shorts. There are many other people on the red and black bikes. A woman wearing a shirt is riding a bicycle.*

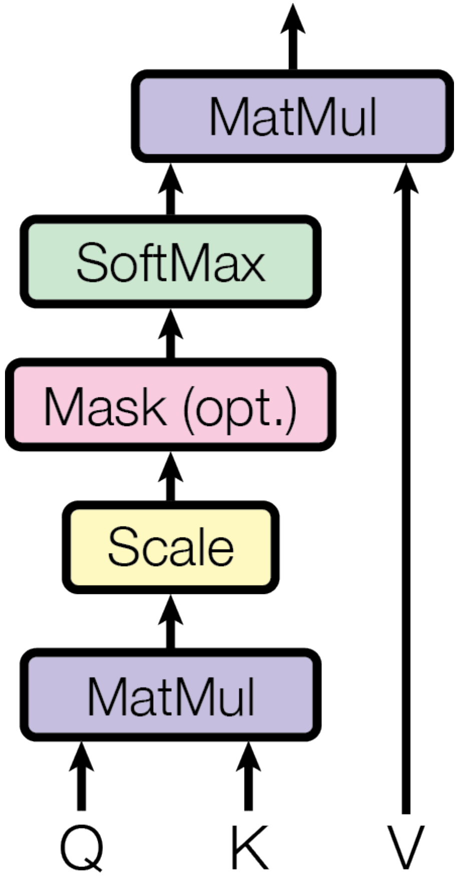
Outline

- Convolutional Networks (ConvNets)
- Recurrent Networks (RNNs)
  - Long-range dependency, vanishing
  - LSTM
  - RNNs in different forms
- Attention Mechanisms
  - (Query, Key, Value)
  - Attention on Text and Images
- Transformers: Multi-head Attention
  - Transformer
  - BERT

Transformers – Multi-head (Self-)Attention

- State-of-the-art Results by Transformers
  - [Vaswani et al., 2017] Attention Is All You Need
    - Machine Translation
  - [Devlin et al., 2018] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding
    - Pre-trained Text Representation
  - [Radford et al., 2019] Language Models are Unsupervised Multitask Learners
    - Language Models

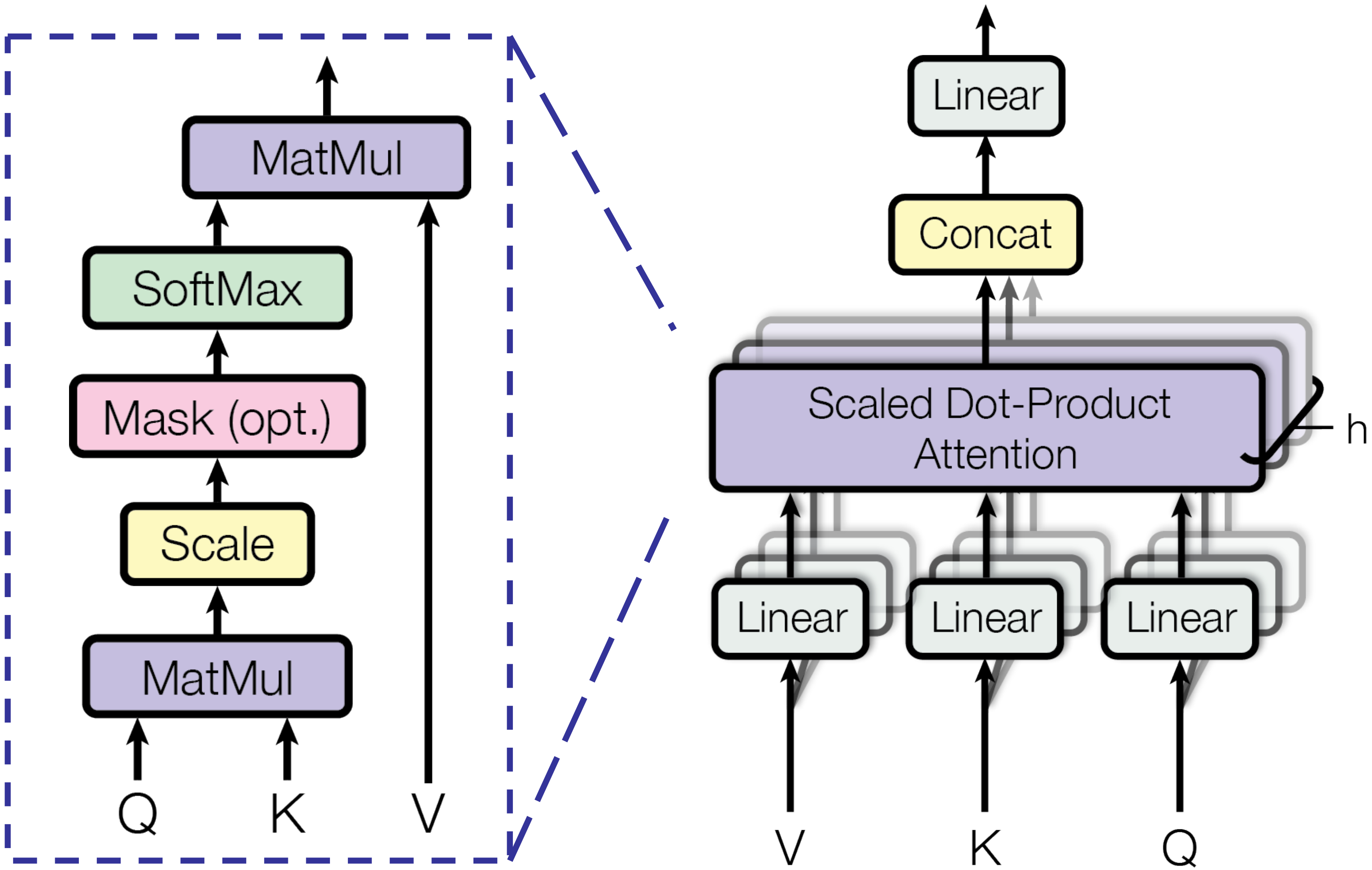
Multi-head Attention



Scaled Dot-Product Attention

Image source: Vaswani, et al., 2017

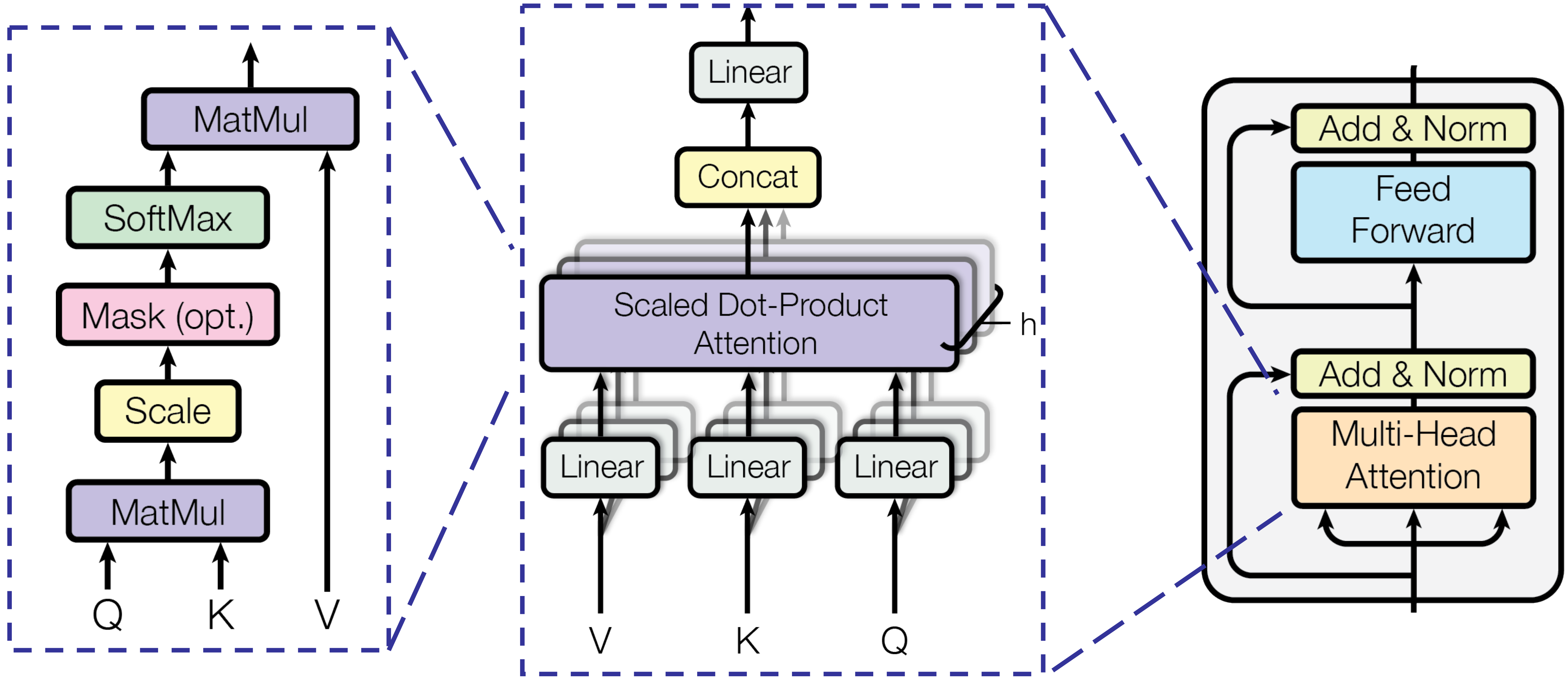
Multi-head Attention



Scaled Dot-Product Attention **Multi-head Attention**

Image source: Vaswani, et al., 2017

Multi-head Attention

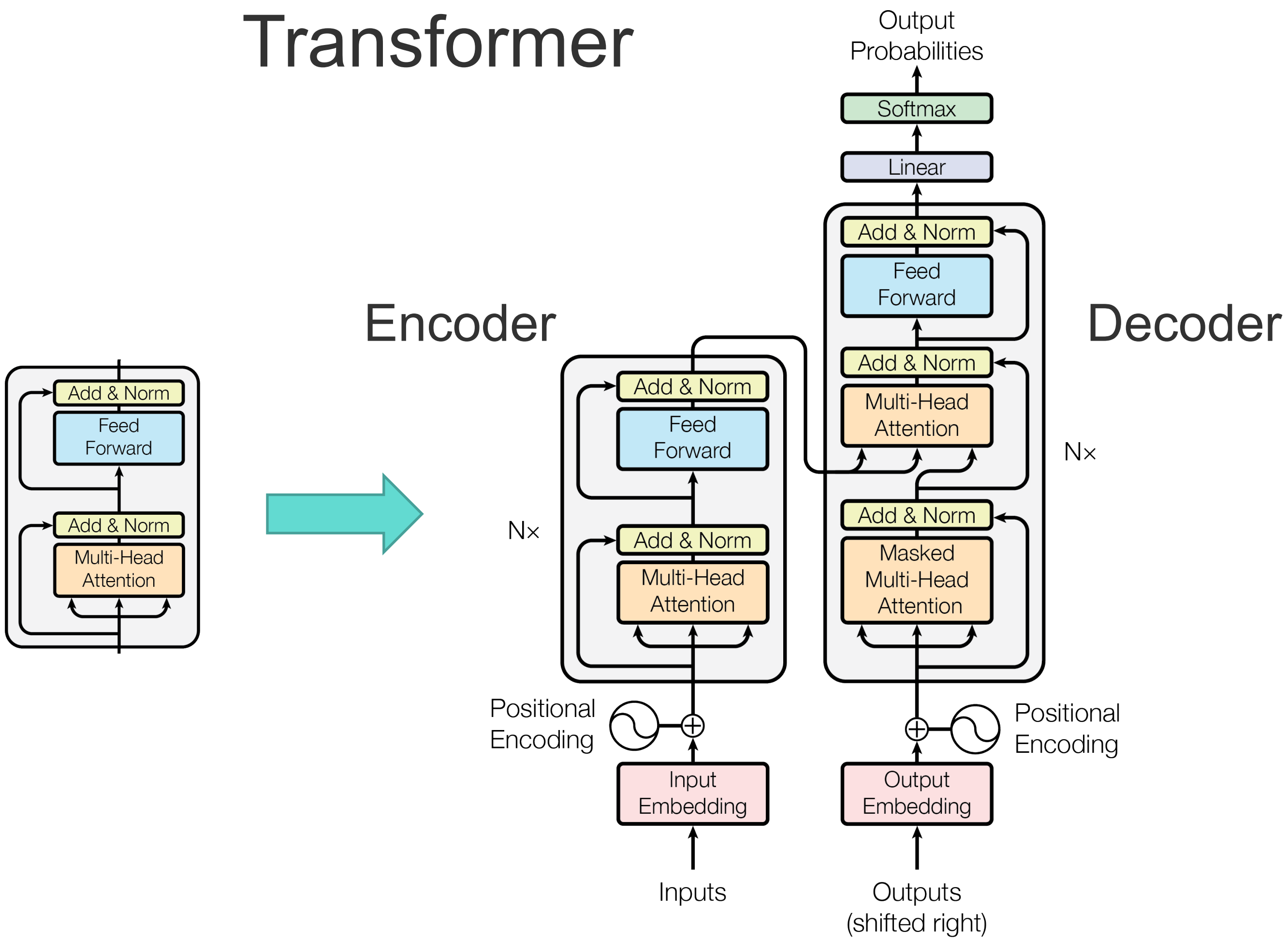


Scaled Dot-Product Attention

Multi-head Attention

Image source: Vaswani, et al., 2017

Multi-head Attention in Encoders and Decoders



Multi-head Attention in Encoders and Decoders

Transformer

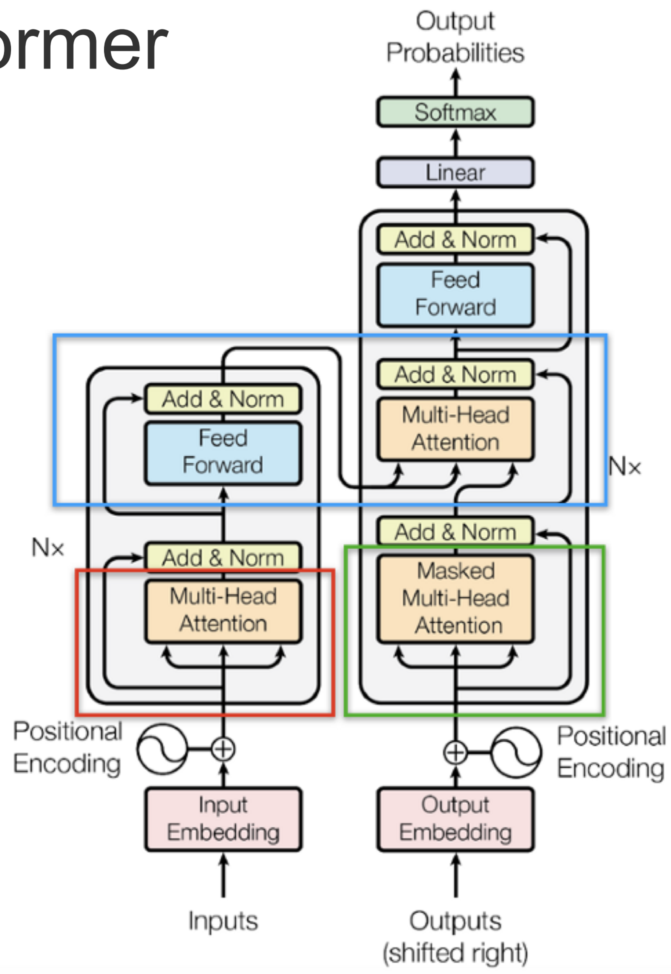


Figure 1: The Transformer - model architecture.

encoder self attention

1. Multi-head Attention
2. Query=Key=Value

decoder self attention

1. Masked Multi-head Attention
2. Query=Key=Value

encoder-decoder attention

1. Multi-head Attention
2. Encoder Self attention=Key=Value
3. Decoder Self attention=Query

Image source: Bgg

BERT: Pre-trained Text Representation Model

Image source: Vaswani, et al., 2017

BERT: Pre-trained Text Representation Model

- Conventional word embedding:
  - Word2vec, Glove
  - A pre-trained **matrix**, each row is an embedding vector of a word

	0	1	2	3	4	5	6	7	8	9	...
fox	-0.348680	-0.077720	0.177750	-0.094953	-0.452890	0.237790	0.209440	0.037886	0.035064	0.899010	...
ham	-0.773320	-0.282540	0.580760	0.841480	0.258540	0.585210	-0.021890	-0.463680	0.139070	0.658720	...
brown	-0.374120	-0.076264	0.109260	0.186620	0.029943	0.182700	-0.631980	0.133060	-0.128980	0.603430	...
beautiful	0.171200	0.534390	-0.348540	-0.097234	0.101800	-0.170860	0.295650	-0.041816	-0.516550	2.117200	...
jumps	-0.334840	0.215990	-0.350440	-0.260020	0.411070	0.154010	-0.386110	0.206380	0.386700	1.460500	...
eggs	-0.417810	-0.035192	-0.126150	-0.215930	-0.669740	0.513250	-0.797090	-0.068611	0.634660	1.256300	...
beans	-0.423290	-0.264500	0.200870	0.082187	0.066944	1.027600	-0.989140	-0.259950	0.145960	0.766450	...
sky	0.312550	-0.303080	0.019587	-0.354940	0.100180	-0.141530	-0.514270	0.886110	-0.530540	1.556600	...
bacon	-0.430730	-0.016025	0.484620	0.101390	-0.299200	0.761820	-0.353130	-0.325290	0.156730	0.873210	...
breakfast	0.073378	0.227670	0.208420	-0.456790	-0.078219	0.601960	-0.024494	-0.467980	0.054627	2.283700	...
toast	0.130740	-0.193730	0.253270	0.090102	-0.272580	-0.030571	0.096945	-0.115060	0.484000	0.848380	...
today	-0.156570	0.594890	-0.031445	-0.077586	0.278630	-0.509210	-0.066350	-0.081890	-0.047986	2.803600	...
blue	0.129450	0.036518	0.032298	-0.060034	0.399840	-0.103020	-0.507880	0.076630	-0.422920	0.815730	...
green	-0.072368	0.233200	0.137260	-0.156630	0.248440	0.349870	-0.241700	-0.091426	-0.530150	1.341300	...
kings	0.259230	-0.854690	0.360010	-0.642000	0.568530	-0.321420	0.173250	0.133030	-0.089720	1.528600	...
dog	-0.057120	0.052685	0.003026	-0.048517	0.007043	0.041856	-0.024704	-0.039783	0.009614	0.308416	...
sausages	-0.174290	-0.064869	-0.046976	0.287420	-0.128150	0.647630	0.056315	-0.240440	-0.025094	0.502220	...
lazy	-0.353320	-0.299710	-0.176230	-0.321940	-0.385640	0.586110	0.411160	-0.418680	0.073093	1.486500	...
love	0.139490	0.534530	-0.252470	-0.125650	0.048748	0.152440	0.199060	-0.065970	0.128830	2.055900	...
quick	-0.445630	0.191510	-0.249210	0.465900	0.161950	0.212780	-0.046480	0.021170	0.417660	1.686900	...

20 rows × 300 columns

Image source: Vaswani, et al., 2017

BERT: Pre-trained Text Representation Model

- Conventional word embedding:
  - Word2vec, Glove
  - A pre-trained **matrix**, each row is an embedding vector of a word

	0	1	2	3	4	5	6	7	8	9	...
fox	-0.348680	-0.077720	0.177750	-0.094953	-0.452890	0.237790	0.209440	0.037886	0.035064	0.899010	...
ham	-0.773320	-0.282540	0.580760	0.841480	0.258540	0.585210	-0.021890	-0.463680	0.139070	0.658720	...
brown	-0.374120	-0.076264	0.109260	0.186620	0.029943	0.182700	-0.631980	0.133060	-0.128980	0.603430	...
beautiful	0.171200	0.534390	-0.348540	-0.097234	0.101800	-0.170860	0.295650	-0.041816	-0.516550	2.117200	...
jumps	-0.334840	0.215990	-0.350440	-0.260020	0.411070	0.154010	-0.386110	0.206380	0.386700	1.460500	...
eggs	-0.417810	-0.035192	-0.126150	-0.215930	-0.669740	0.513250	-0.797090	-0.068611	0.634660	1.256300	...
beans	-0.423290	-0.264500	0.200870	0.082187	0.066944	1.027600	-0.989140	-0.259950	0.145960	0.766450	...
sky	0.312550	-0.303080	0.019587	-0.354940	0.100180	-0.141530	-0.514270	0.886110	-0.530540	1.556600	...
bacon	-0.430730	-0.016025	0.484620	0.101390	-0.299200	0.761820	-0.353130	-0.325290	0.156730	0.873210	...
breakfast	0.073378	0.227670	0.208420	-0.456790	-0.078219	0.601960	-0.024494	-0.467980	0.054627	2.283700	...
							945	-0.115060	0.484000	0.848380	...
							350	-0.081890	-0.047986	2.803600	...
							880	0.076630	-0.422920	0.815730	...
							700	-0.091426	-0.530150	1.341300	...
							250	0.133030	-0.089720	1.528600	...
							704	-0.039783	0.009614	0.308416	...
							315	-0.240440	-0.025094	0.502220	...
							160	-0.418680	0.073093	1.486500	...
							060	-0.065970	0.128830	2.055900	...
							480	0.021170	0.417660	1.686900	...

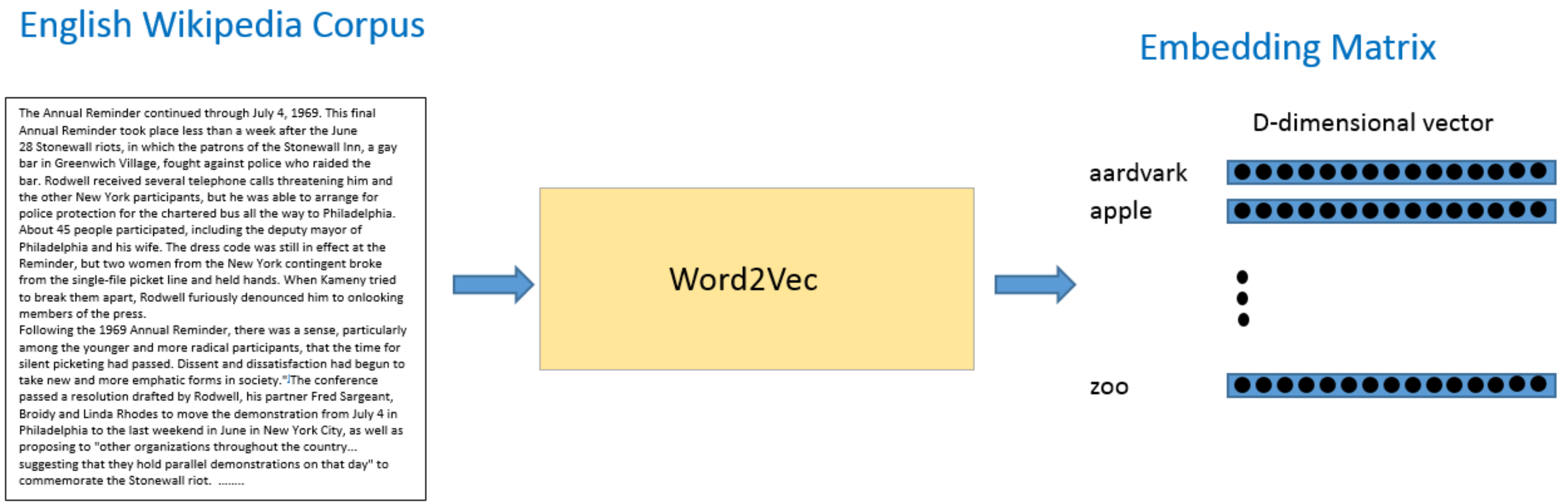
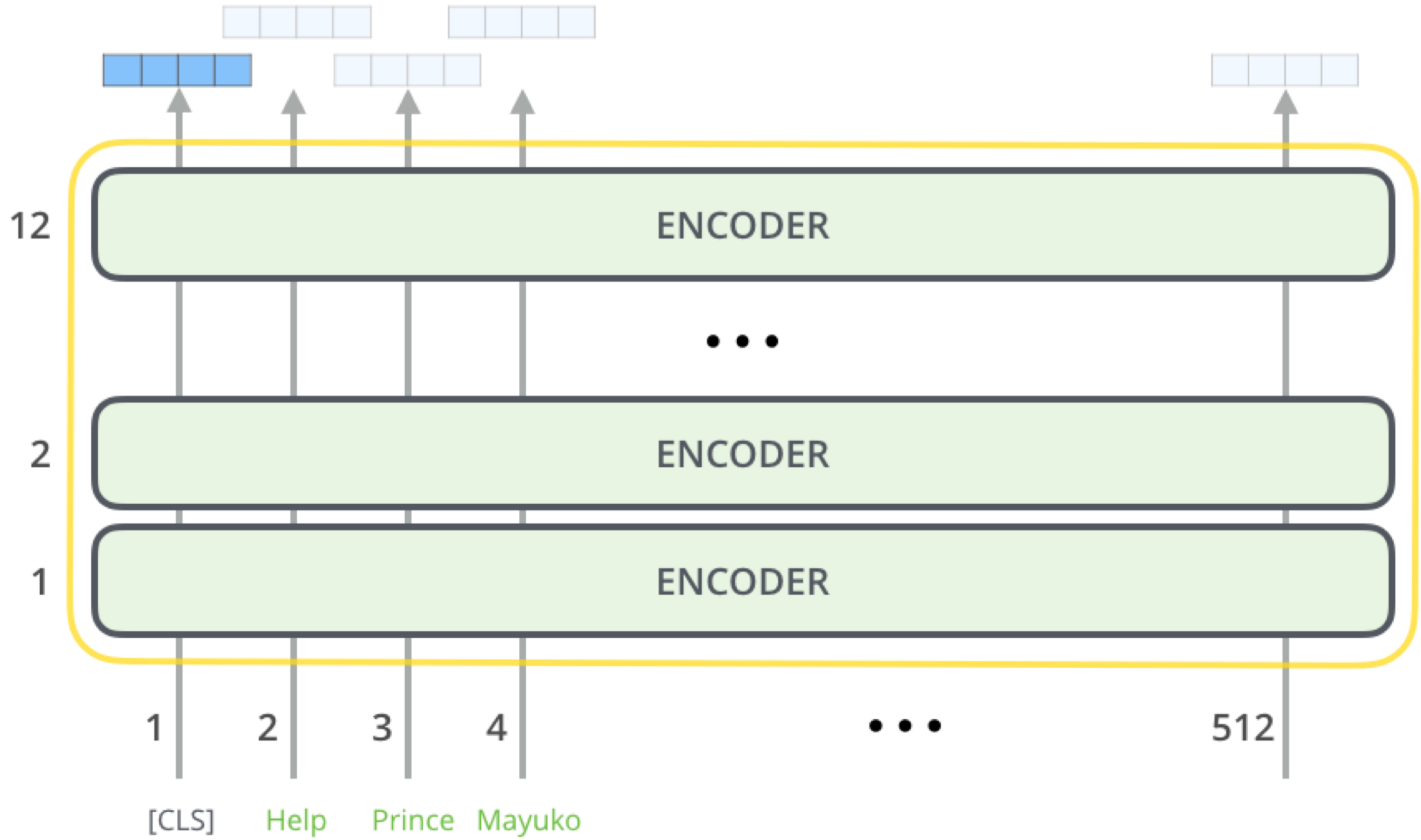


Image source: Vaswa

BERT: Pre-trained Text Representation Model

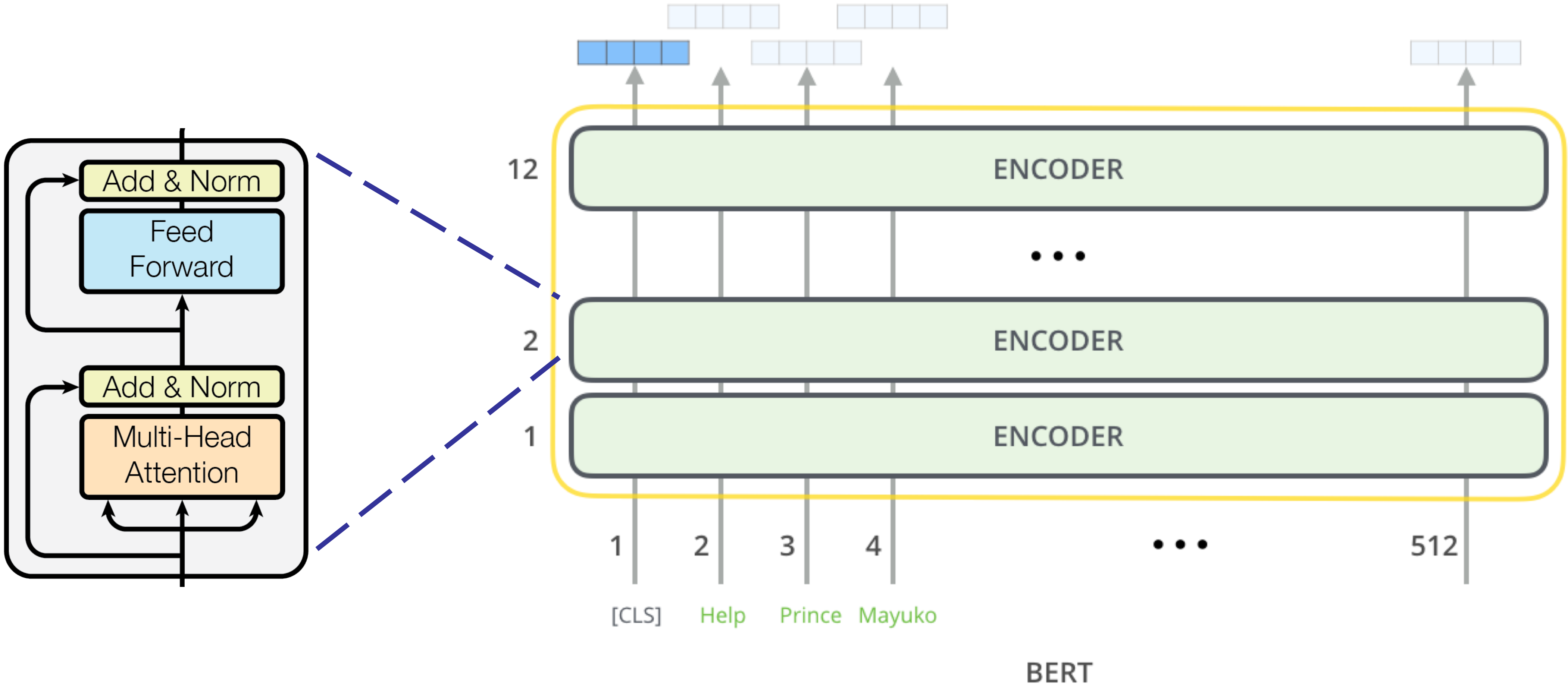
- BERT: A model to extract *contextualized* word embedding



BERT

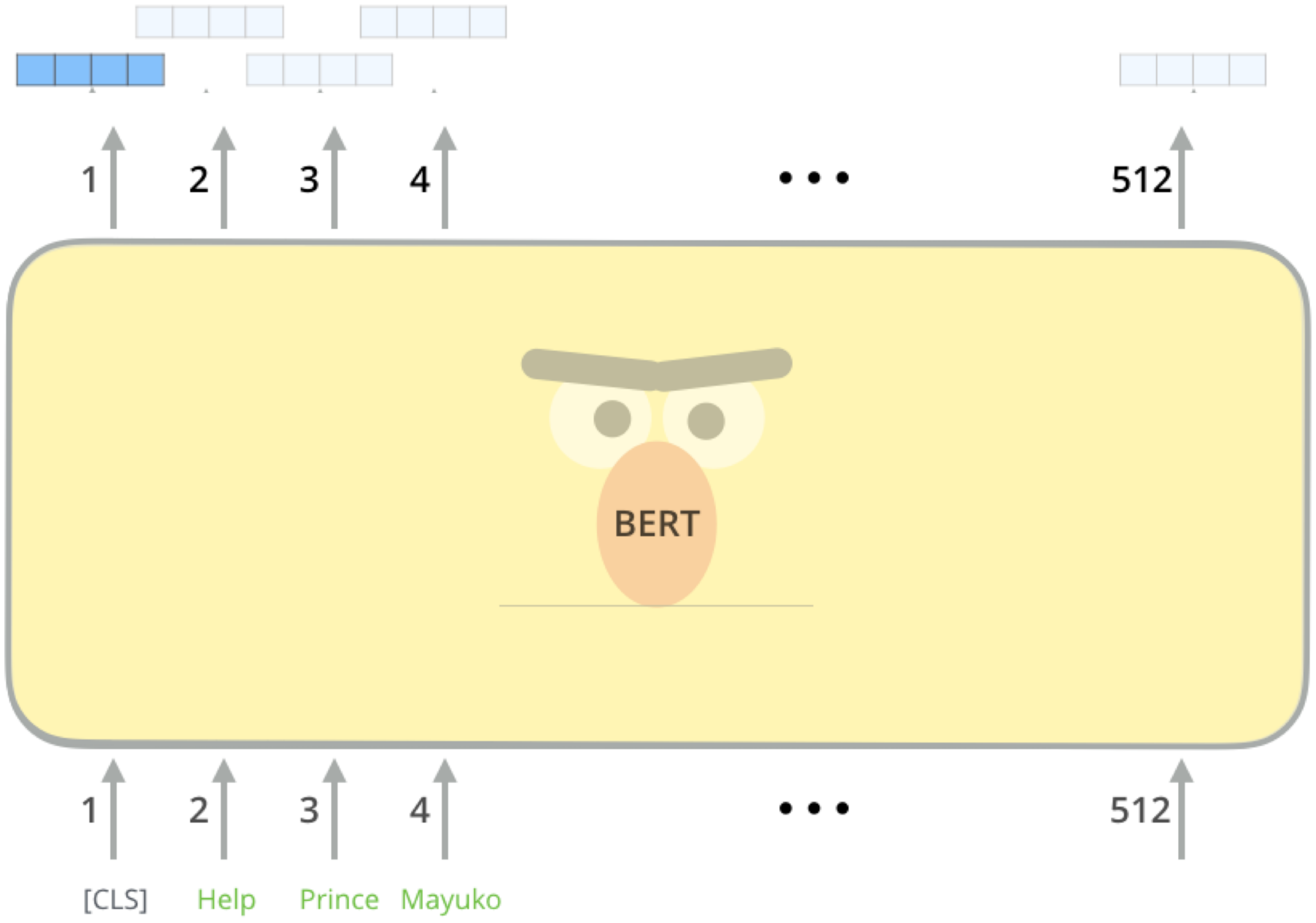
BERT: Pre-trained Text Representation Model

- BERT: A model to extract *contextualized* word embedding



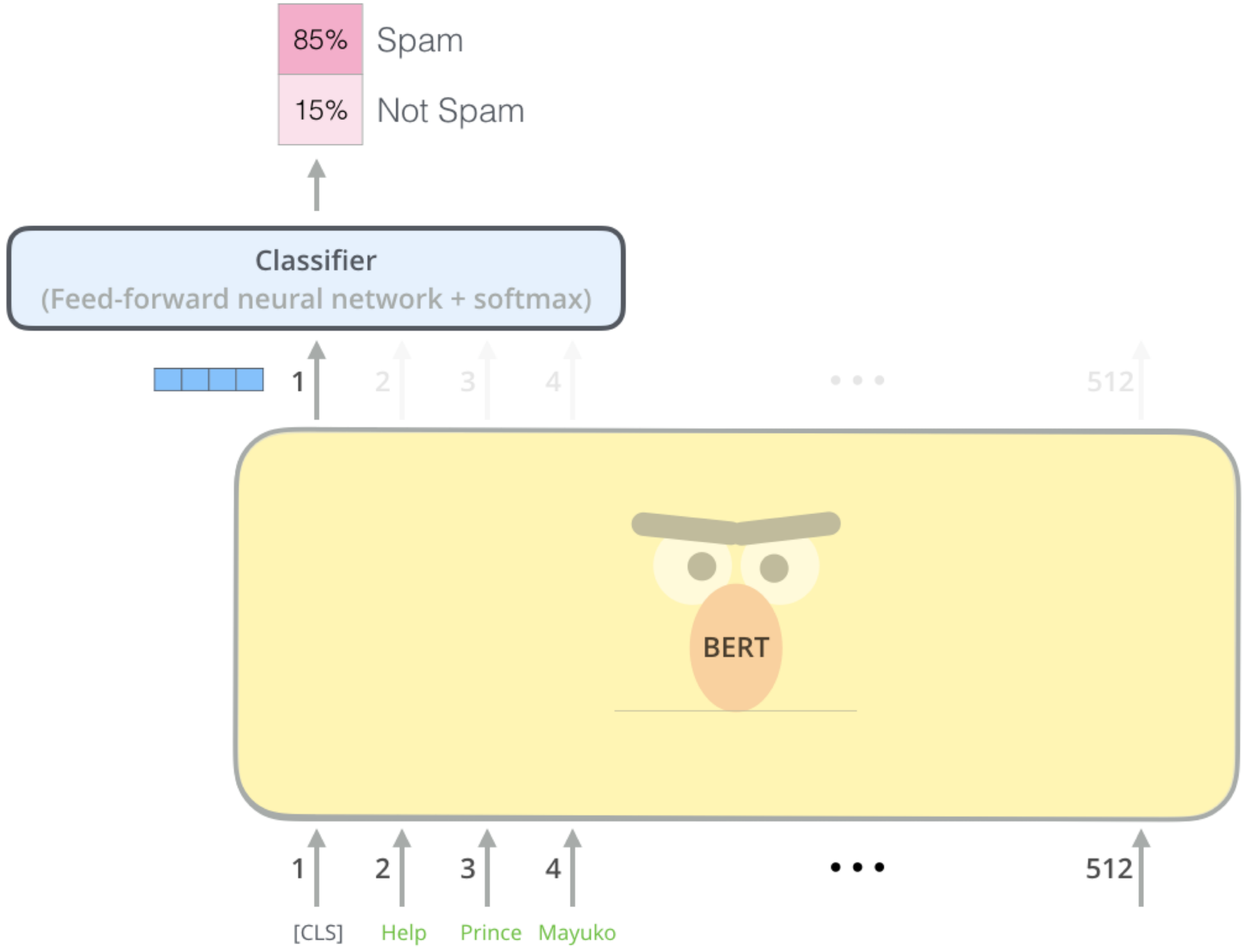
BERT: Pre-trained Text Representation Model

- BERT: A model to extract *contextualized* word embedding



BERT: Pre-trained Text Representation Model

- Use BERT for sentence classification



BERT Results

- **Huge improvements over SOTA on 12 NLP task**

System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
$\mathrm{BERT_{BASE}}$	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
$\mathrm{BERT_{LARGE}}$	**86.7/85.9**	**72.1**	**91.1**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**81.9**

Table 1: GLUE Test results, scored by the GLUE evaluation server. The number below each task denotes the number of training examples. The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set. OpenAI GPT = (L=12, H=768, A=12); $\mathrm{BERT_{BASE}}$ = (L=12, H=768, A=12); $\mathrm{BERT_{LARGE}}$ = (L=24, H=1024, A=16). BERT and OpenAI GPT are single-model, single task. All results obtained from https://gluebenchmark.com/leaderboard and https://blog.openai.com/language-unsupervised/.

BERT: Pre-training Procedure

- Model architecture:
  - A big Transformer Encoder (240M free parameters)
- Dataset:
  - Wikipedia (2.5B words) + a collection of free ebooks (800M words)

BERT: Pre-training Procedure

- Model architecture:
  - A big Transformer Encoder (240M free parameters)
- Dataset*:*
  - Wikipedia (2.5B words) + a collection of free ebooks (800M words)
- Training procedure
  - masked language model (masked LM)
    - Masks some percent of words from the input and has to reconstruct those words from context

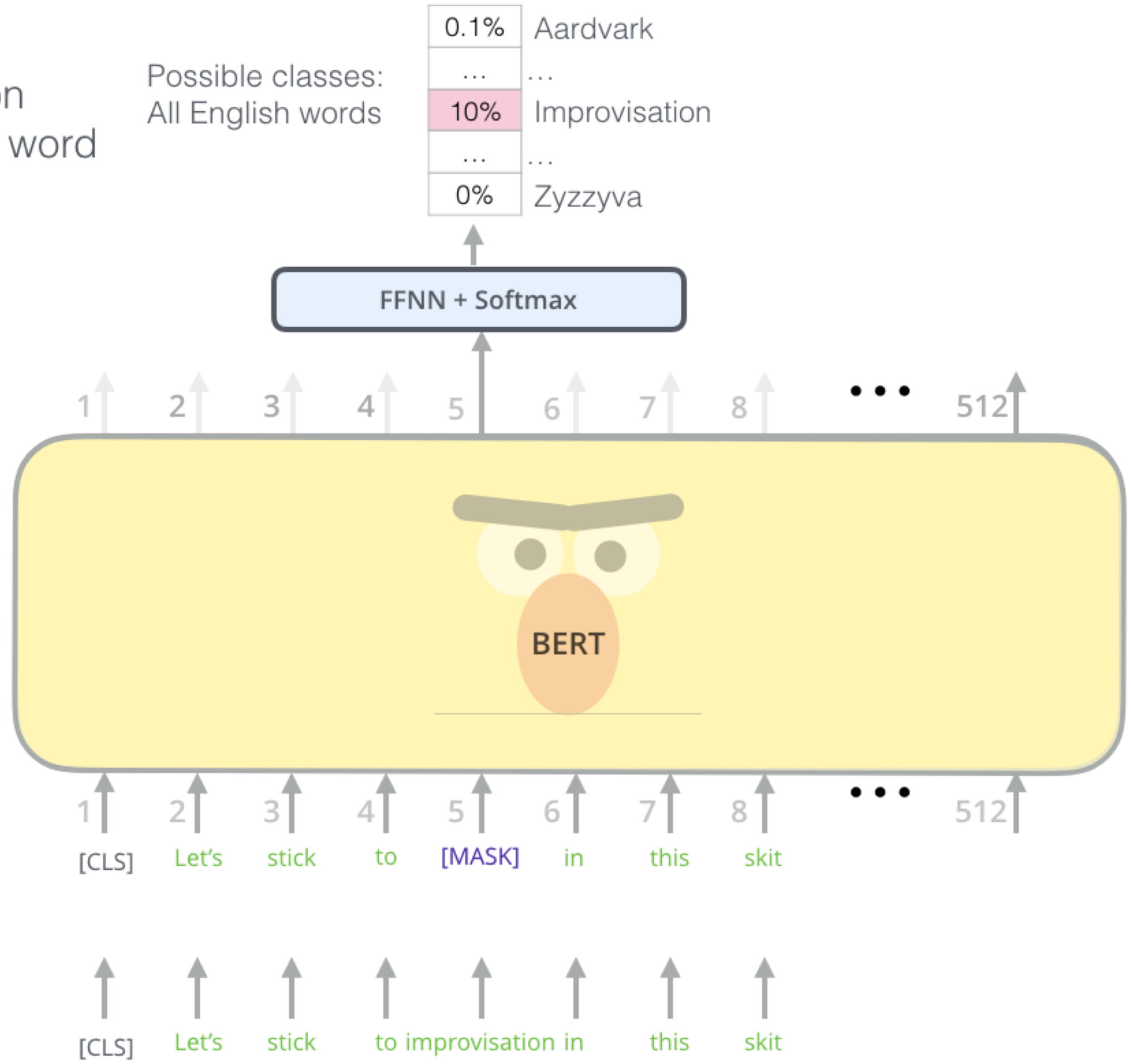
BERT: Pre-training Procedure

- Masked LM

Use the output of the masked word's position to predict the masked word

Randomly mask 15% of tokens

Input

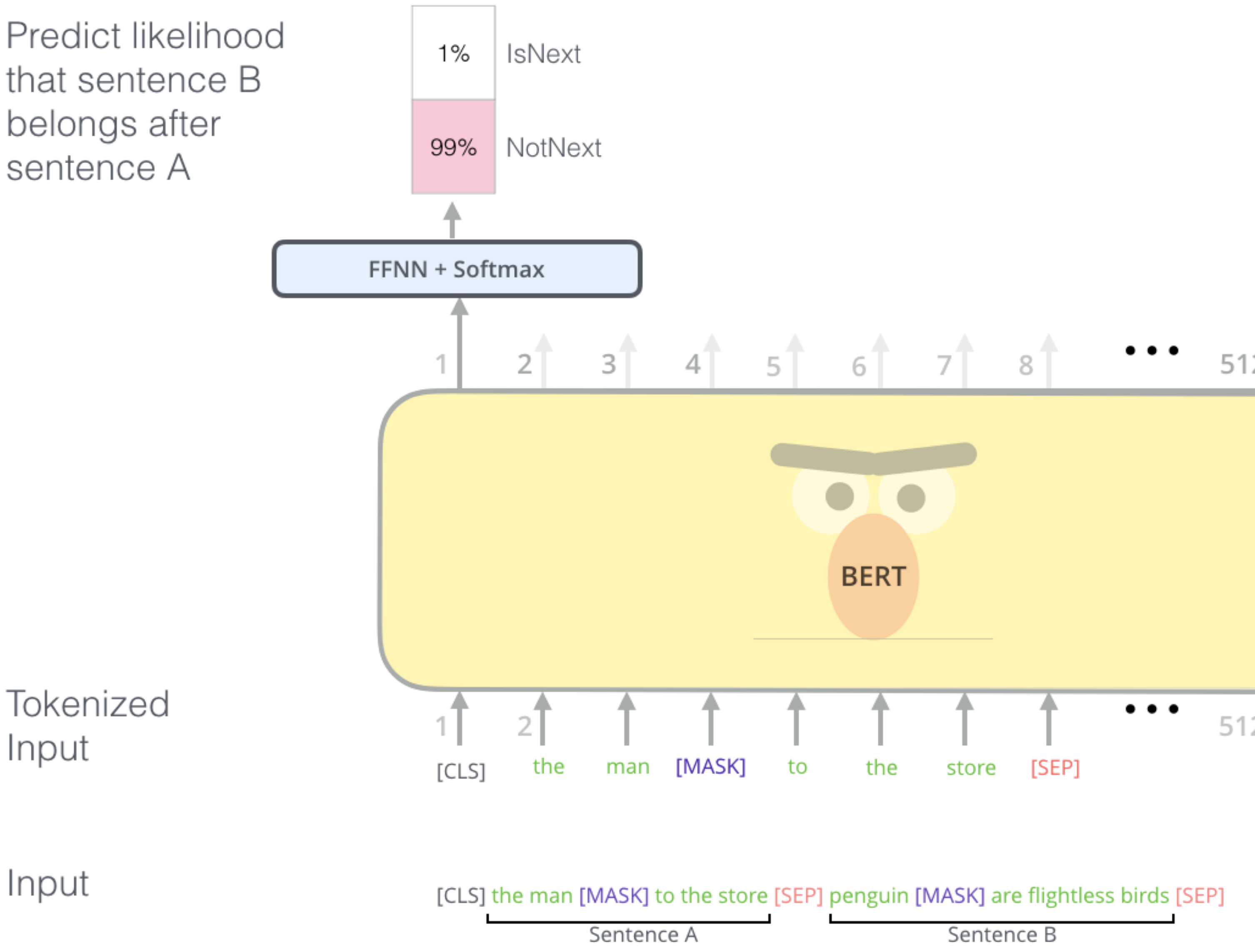


BERT: Pre-training Procedure

- Model architecture:
  - A big Transformer Encoder (240M free parameters)
- Dataset*:*
  - Wikipedia (2.5B words) + a collection of free ebooks (800M words)
- Training procedure
  - masked language model (masked LM)
    - Masks some percent of words from the input and has to reconstruct those words from context
  - Two-sentence task
    - To understand relationships between sentences
    - Concatenate two sentences A and B and predict whether B actually comes after A in the original text

BERT: Pre-training Procedure

- Two sentence task



BERT: Pre-training Procedure

- BERT is trained on 4 TPU pods (=256 TPU chips) in 4 days
  - TPU: a matrix multiplication engine
- = 64 V100 GPUs, Infiniband network, 5.3 days
- = a standard 4 GPU desktop with RTX 2080Ti, 99 days

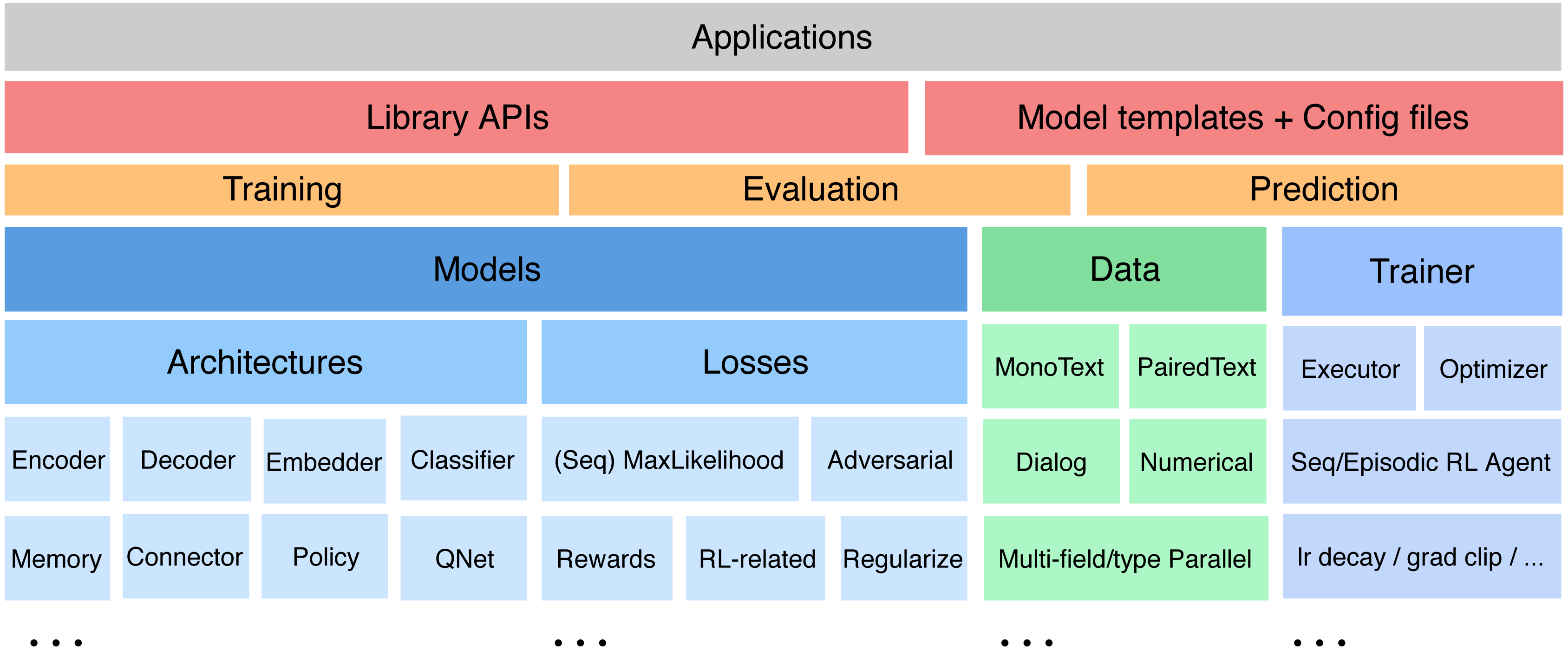
source: Tim Dettmers

Word Embedding on Texar

Texar

- A general-purpose text generation toolkit on TensorFlow

***Texar** stack*



Word Embedding on Texar

- Word2vec, Glove

```
import texar as tx

# Load data and pre-trained word embedding matrix
data = tx.data.MonoTextData(hparams=config.data)
iterator = tx.data.DataIterator(data)
data_batch = iterator.get_next()

# Create and initialize word embedder
embedder = texar.modules.WordEmbedder(
    init_value=data.embedding_init_value, hparams=config.emb)

# Embed text into vectors
data_embed = embedder(data_batch)

# Downstream tasks
classifier = tx.modules.Conv1DClassifier(hparams=config.clas)
logits, pred = classifier(input=data_embed)
```

```
config.data = {
    "embedding_init": {
        "file": "word2vec.pretrain.dat"
        "read_fn": "load_word2vec" # "load_glove"
    }
}
```

Texar

Word Embedding on Texar

- Word2vec, Glove

```
import texar as tx

# Load data and pre-trained word embedding matrix
data = tx.data.MonoTextData(hparams=config.data)
iterator = tx.data.DataIterator(data)
data_batch = iterator.get_next()

# Create and initialize word embedder
embedder = texar.modules.WordEmbedder(
    init_value=data.embedding_init_value, hparams=co

# Embed text into vectors
data_embed = embedder(data_batch)

# Downstream tasks
classifier = tx.modules.Conv1DClassifier(hparams=config.clas)
logits, pred = classifier(input=data_embed)
```

- BERT

```
# Create BERT embedder
embedder = tx.modules.TransformerEncoder(hparams=bert_config)
# Initialize BERT embedder
texar.init_bert_checkpoint("./bert.ckpt")

# Embed text into vectors
data_embed = embedder(data_batch)
```

Texar

Seq2seq Attention on Texar

```
# Read data
dataset = PairedTextData(data_hparams)
batch = DataIterator(dataset).get_next()

# Encode
embedder = WordEmbedder(dataset.vocab.size, hparams=embedder_hparams)
encoder = TransformerEncoder(hparams=encoder_hparams)
enc_outputs = encoder(embedder(batch['source_text_ids']),
                      batch['source_length'])

# Decode
decoder = AttentionRNNDecoder(memory=enc_outputs,
                              hparams=decoder_hparams)
outputs, length, _ = decoder(inputs=embedder(batch['target_text_ids']),
                             seq_length=batch['target_length']-1)

# Loss
loss = sequence_sparse_softmax_cross_entropy(
    labels=batch['target_text_ids'][:,1:], logits=outputs.logits, seq_length=length)
```

Seq2seq Attention on Texar

```python
# Read data
dataset = PairedTextData(data_hparams)
batch = DataIterator(dataset).get_next()

# Encode
embedder = WordEmbedder(dataset.vocab.size, hparams=embedder_hparams)
encoder = TransformerEncoder(hparams=encoder_hparams)
enc_outputs = encoder(embedder(batch['source_text_ids']),
                      batch['source_length'])

# Decode
decoder = AttentionRNNDecoder(memory=enc_outputs,
                              hparams=decoder_hparams)
outputs, length, _ = decoder(inputs=embedder(batch['target_text_ids']),
                             seq_length=batch['target_length']-1)

# Loss
loss = sequence_sparse_softmax_cross_entropy(
    labels=batch['target_text_ids'][:,1:], logits=outputs.logits, seq_length=length)
```

```python
decoder_hparams = {
    'rnn_cell': {
        'type': 'LSTMCell'
    }
    'num_layers': 2,
    'attention': {
        'type': 'LuongAttention',
        'kwargs': {
            'num_units': 256,
        }
    }
}
```

Takeaways

- Convolutional Networks (ConvNets)
- Recurrent Networks (RNNs)
  - LSTM designed for long-range dependency, vanishing gradients
  - RNNs not only for sequence data, but also 2D sequences, Trees, graphs
- Attention Mechanisms
  - Three core elements: (Query, Key, Value)
  - Many variants based on alignment score functions
  - Attention on Text and Images
- Transformers: Multi-head Attention
  - Transformer: encoder-decoder
  - BERT: pre-trained text representation
  - GPT-2: pre-trained language model

Texar