

Machine Learning 10-601/301

Tom M. Mitchell
Machine Learning Department
Carnegie Mellon University

March 17, 2021

This section:

- LSTMs
- Sequence to sequence models
- Transformer models
- Attention

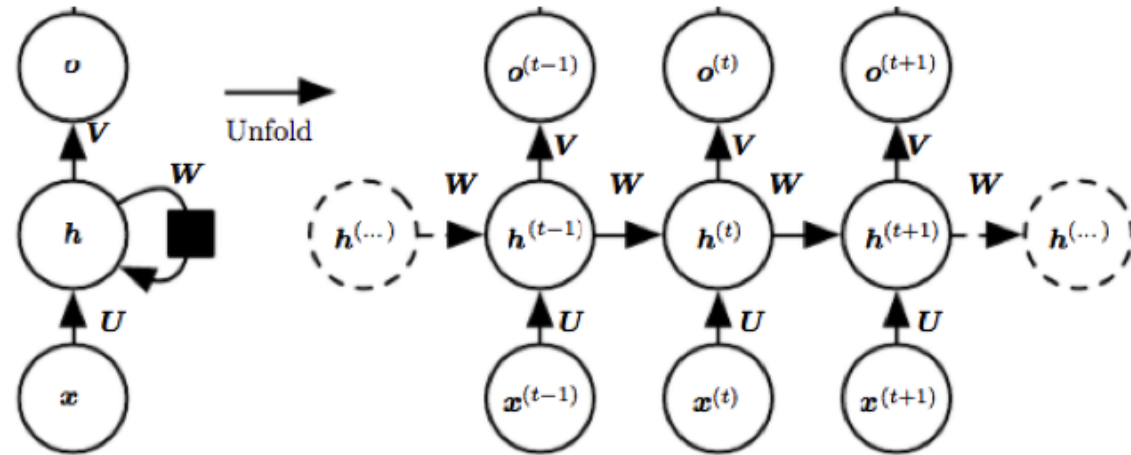
Readings: optional but recommended

- “Dive into Deep Learning” chapters 6.6, 8-8.4, 10.3-10.7
- This book is a free download on the web, and contains running code

Recurrent Neural Nets for Sequential Data

Recurrent Networks

- Key idea: recurrent network uses (part of) its state at t as input for $t+1$



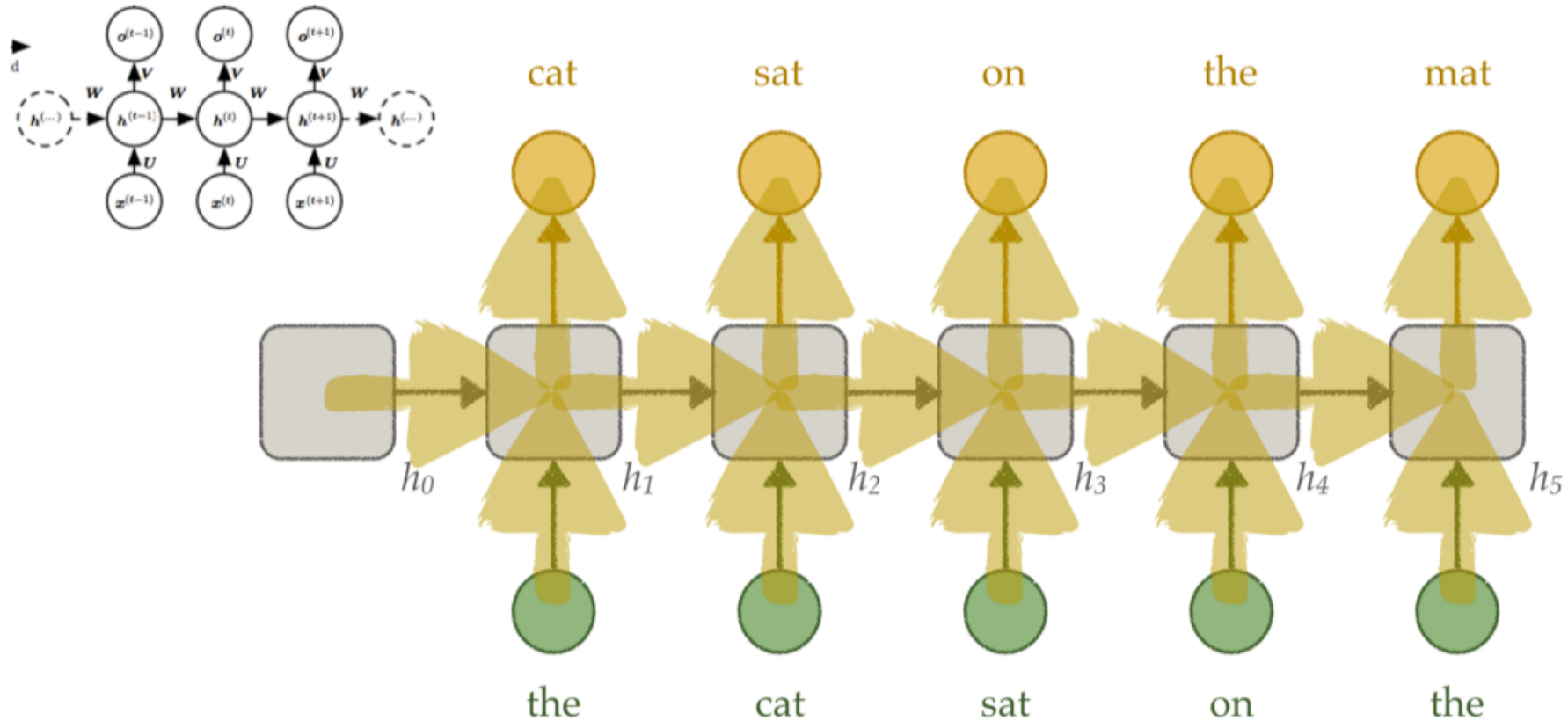
$$\mathbf{o}_t = \phi_2(\mathbf{V}\mathbf{h}_t + \mathbf{b}_o)$$

$$\mathbf{h}_t = \phi_1(\mathbf{U}\mathbf{x} + \mathbf{W}\mathbf{h}_{t-1} + \mathbf{b}_h)$$

Nonlinearity

Hidden State at
previous time
step

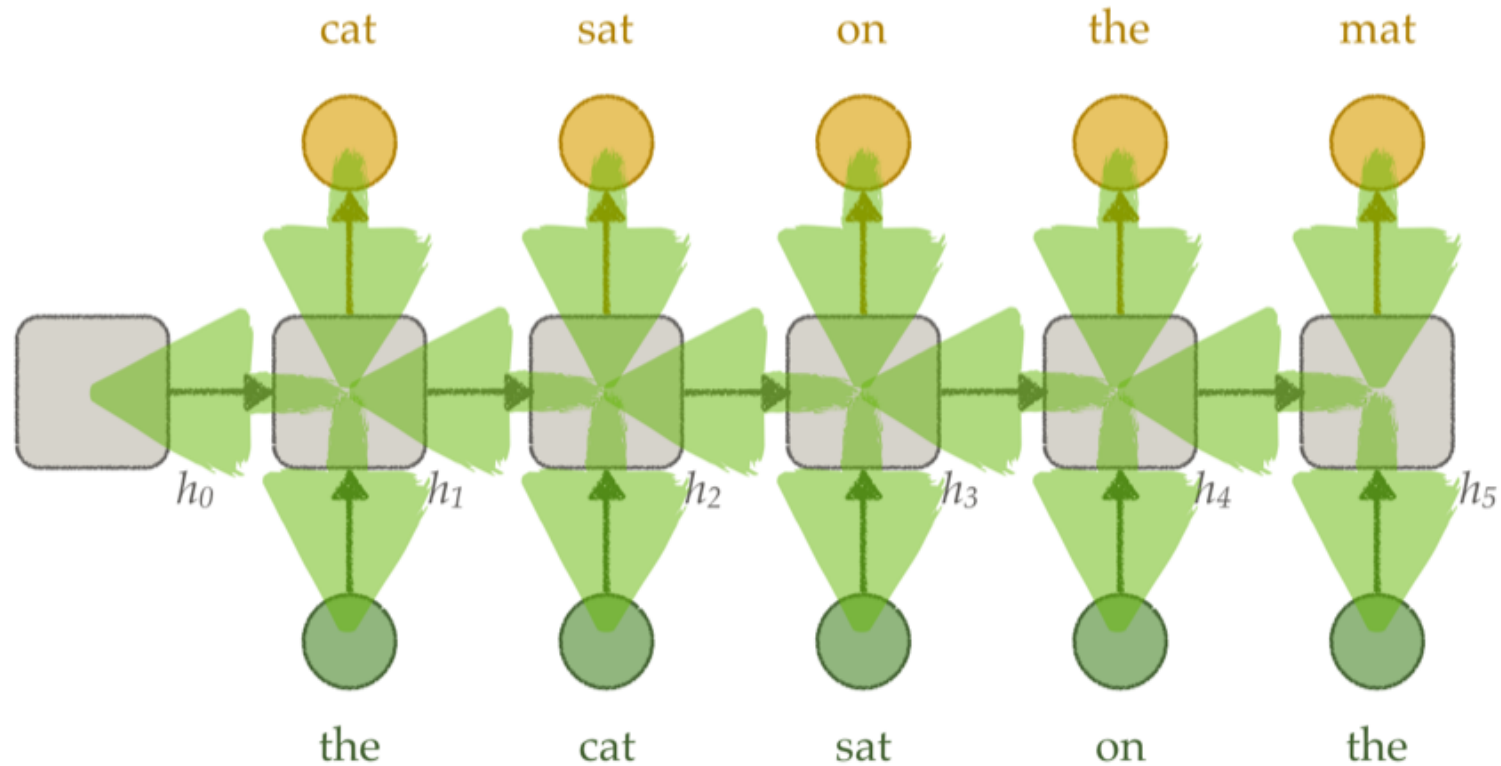
Recurrent Neural Network Language Models



Learning Sequences – Piotr Mirowski

- Forward Pass

Recurrent Neural Network Language Models



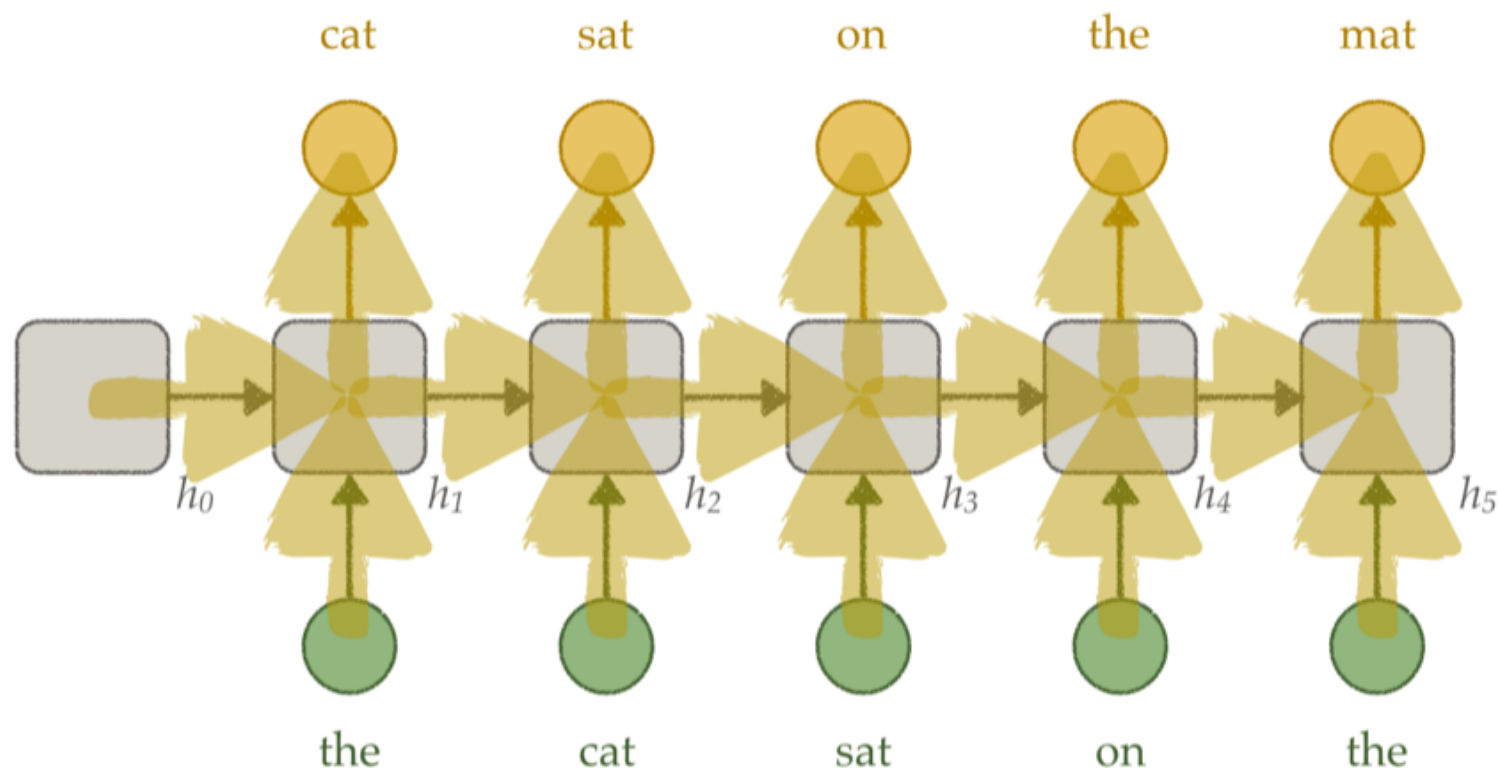
Learning Sequences – Piotr Mirowski

- Backward Pass

* problem: vanishing and/or exploding gradients

Slide Credit: Piotr Mirowski

Recurrent Neural Network Language Models



- Learned hidden representations of context can be useful for:
 - part of speech labeling
 - sentiment analysis
 - information extraction
 - ...
- Predict label for each word, instead of predicting next word

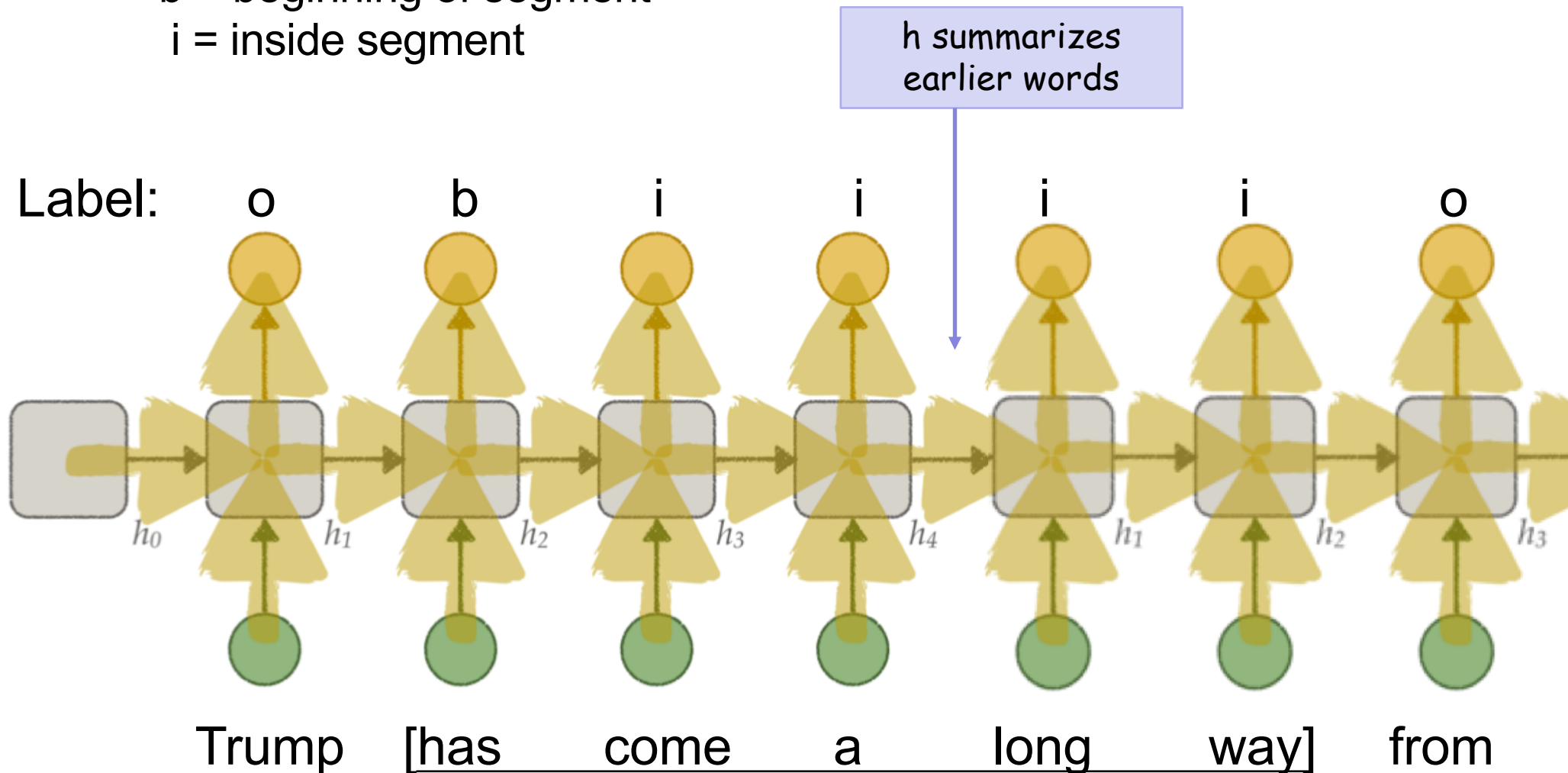
Example: Opinion Mining

Label opinion segments by labeling each word.

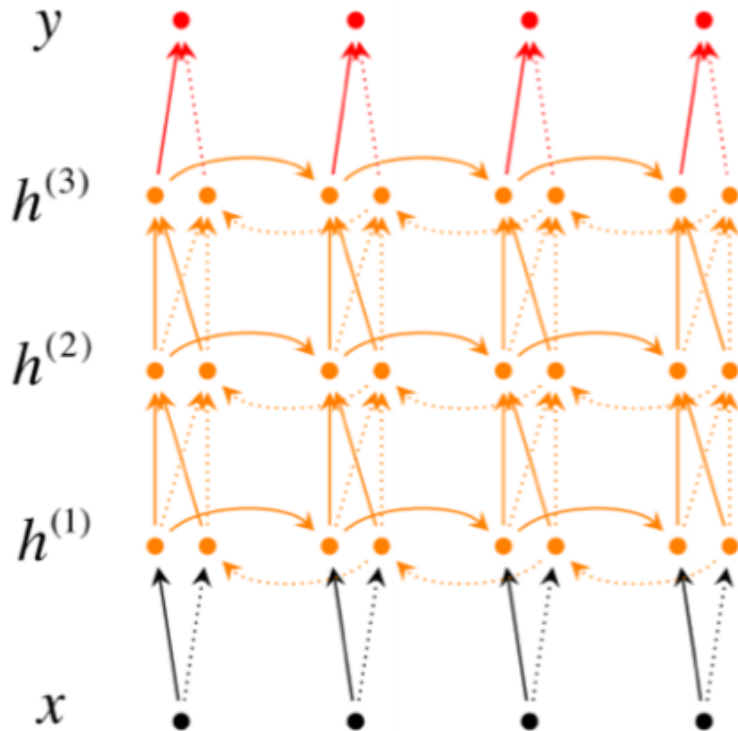
o = outside

b = beginning of segment

i = inside segment



Deep Bidirectional Recurrent Network



$$\vec{h}_t^{(i)} = f(\vec{W}^{(i)} h_t^{(i-1)} + \vec{V}^{(i)} \vec{h}_{t-1}^{(i)} + \vec{b}^{(i)})$$

$$\overleftarrow{h}_t^{(i)} = f(\overleftarrow{W}^{(i)} h_t^{(i-1)} + \overleftarrow{V}^{(i)} \overleftarrow{h}_{t+1}^{(i)} + \overleftarrow{b}^{(i)})$$

$$y_t = g(U[\vec{h}_t^{(L)}; \overleftarrow{h}_t^{(L)}] + c)$$

Two additional ideas:

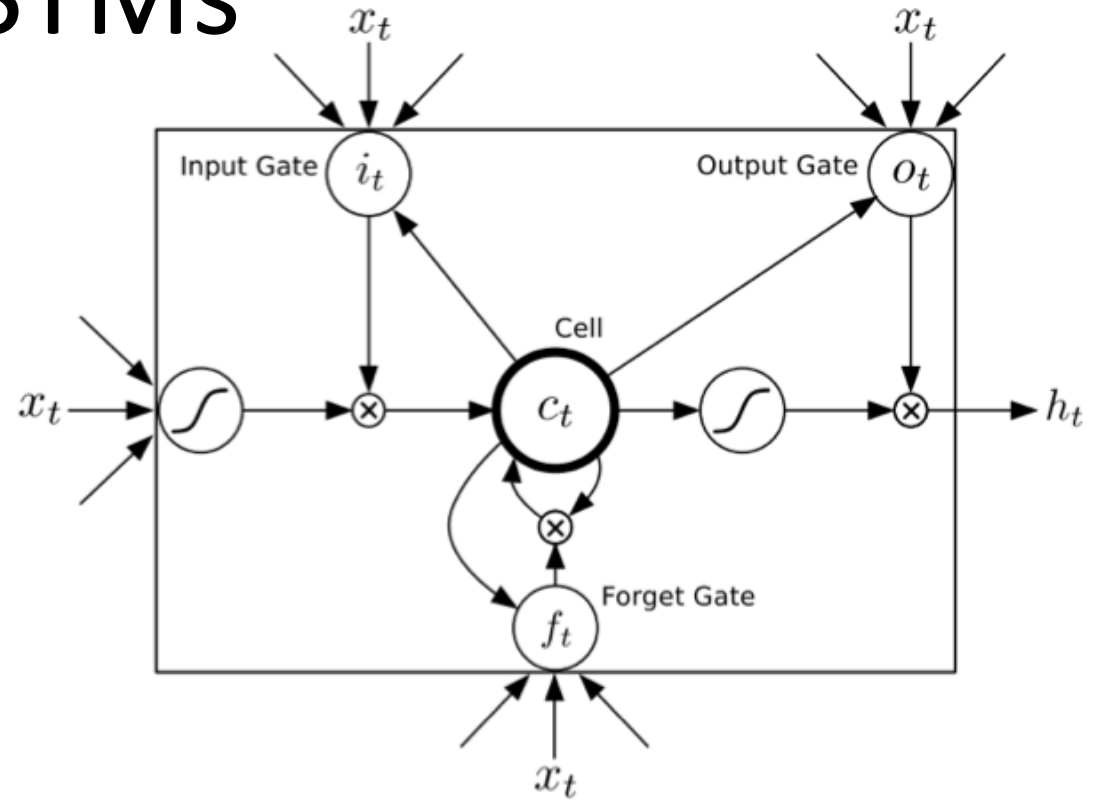
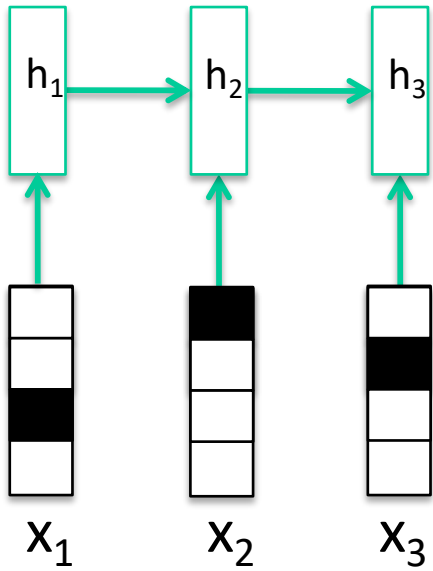
- Multiple layers to compute y from x
- A left-to-right RNN, plus right-to-left RNN

Example:

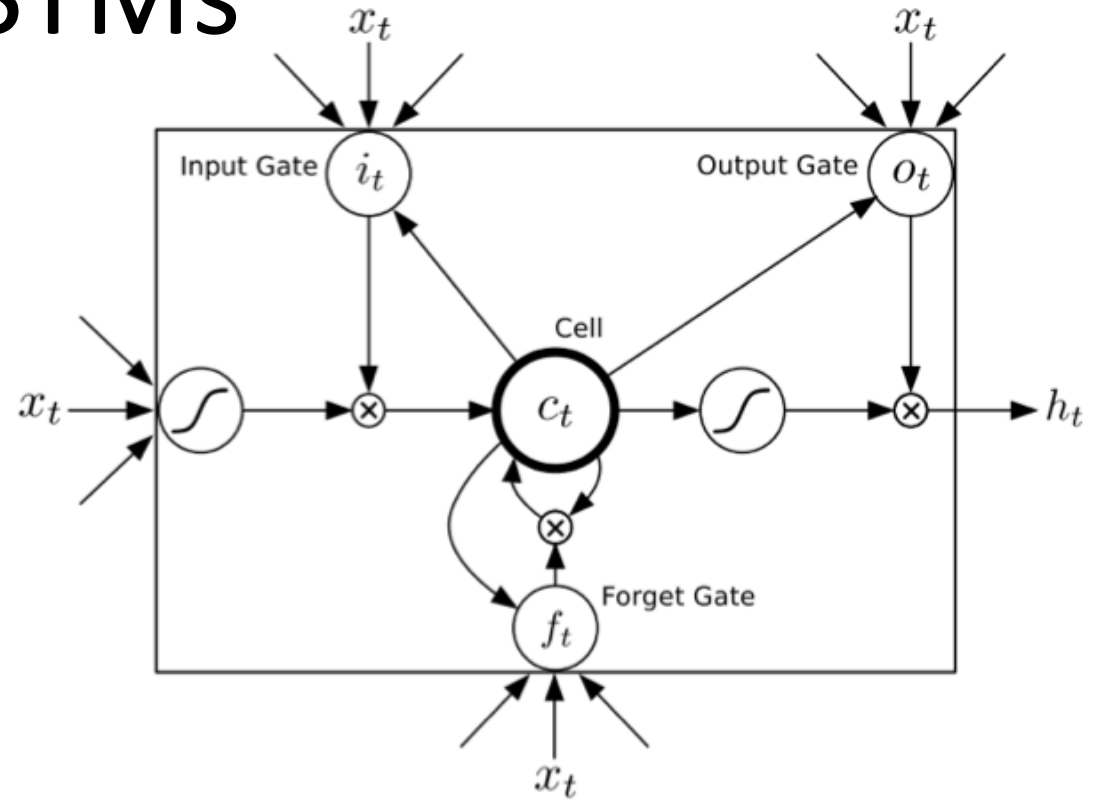
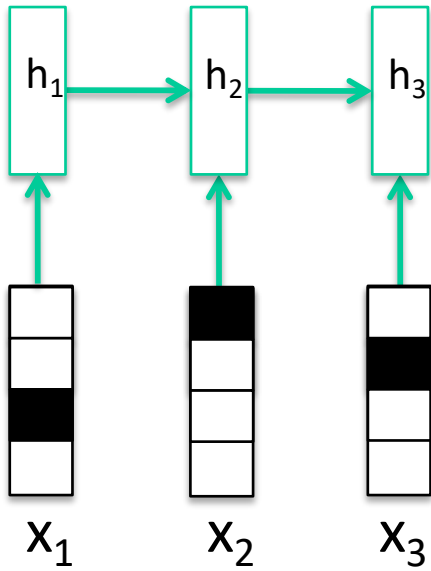
- Y label values {begin, inside, outside} for each word, to label contiguous text segments indicating opinions. [Irsoy & Cardie, 2014]

Long Short Term
Memory

LSTMs

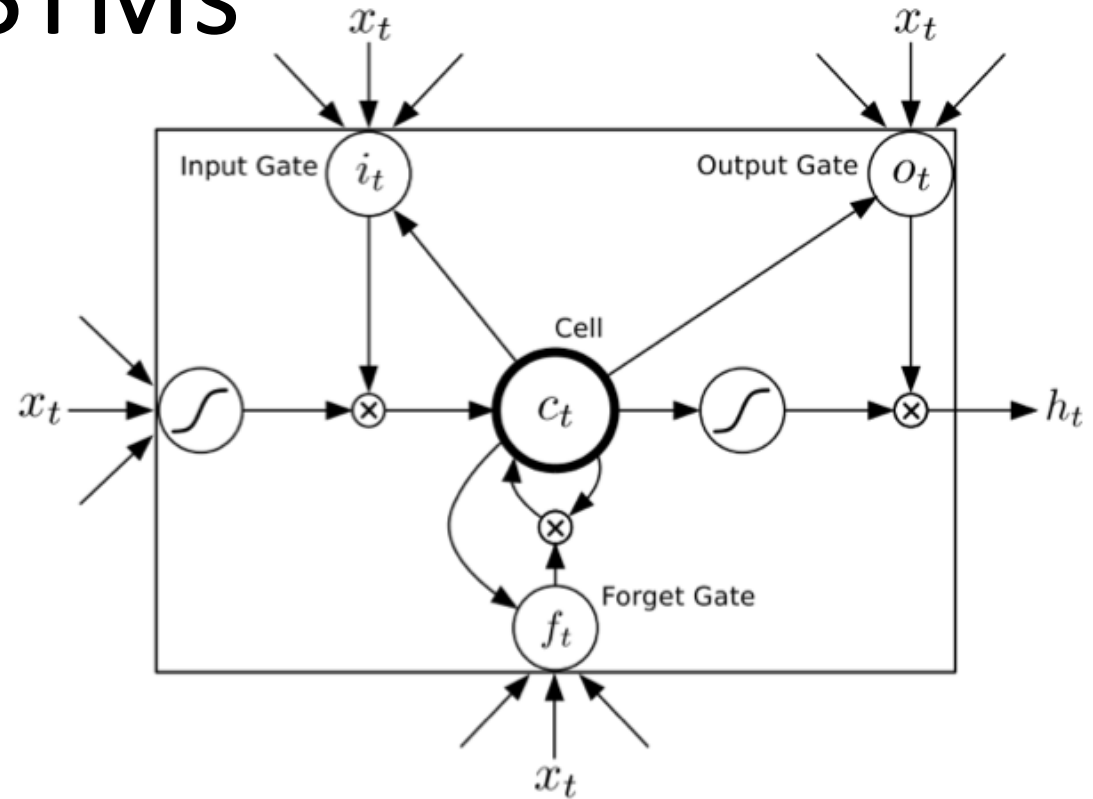
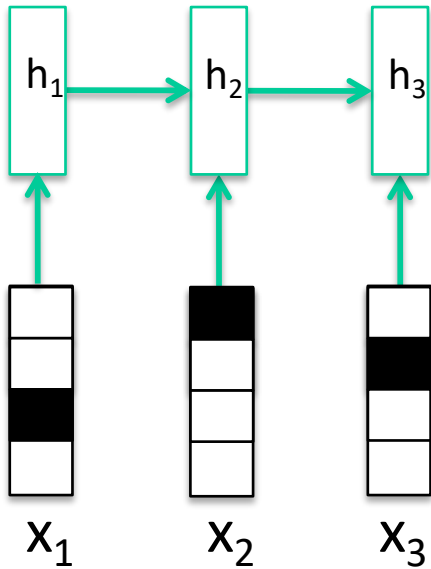


LSTMs



$$\mathbf{i}_t = \sigma(W_{xi}\mathbf{x}_t + W_{hi}\mathbf{h}_{t-1} + W_{ci}\mathbf{c}_{t-1} + \mathbf{b}_i),$$

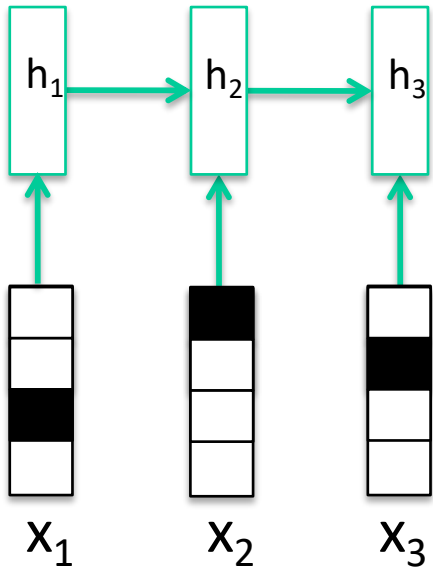
LSTMs



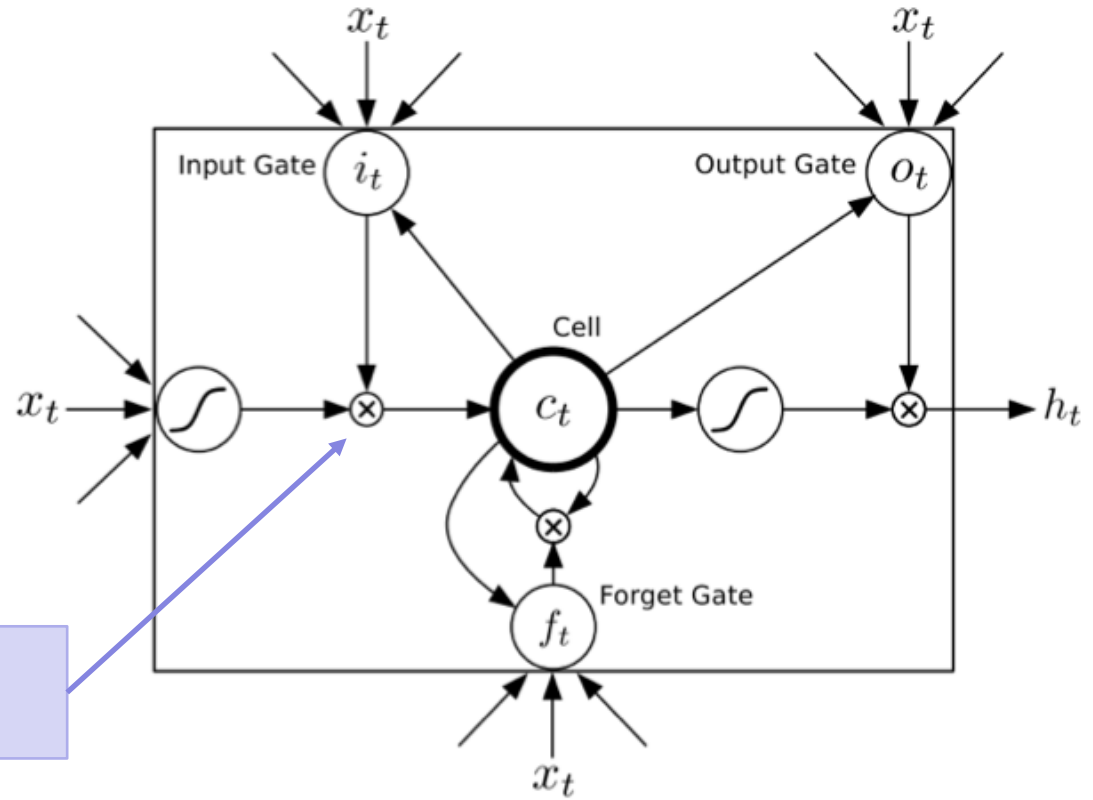
$$\mathbf{i}_t = \sigma(W_{xi}\mathbf{x}_t + W_{hi}\mathbf{h}_{t-1} + W_{ci}\mathbf{c}_{t-1} + \mathbf{b}_i),$$

$$\mathbf{f}_t = \sigma(W_{xf}\mathbf{x}_t + W_{hf}\mathbf{h}_{t-1} + W_{cf}\mathbf{c}_{t-1} + \mathbf{b}_f),$$

LSTM



Element-wise multiply

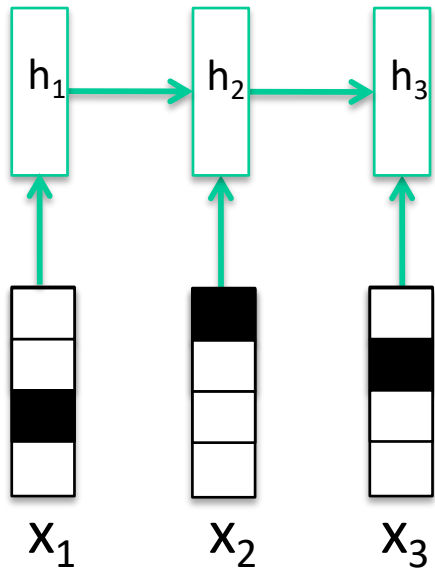


$$\mathbf{i}_t = \sigma(W_{xi}\mathbf{x}_t + W_{hi}\mathbf{h}_{t-1} + W_{ci}\mathbf{c}_{t-1} + \mathbf{b}_i),$$

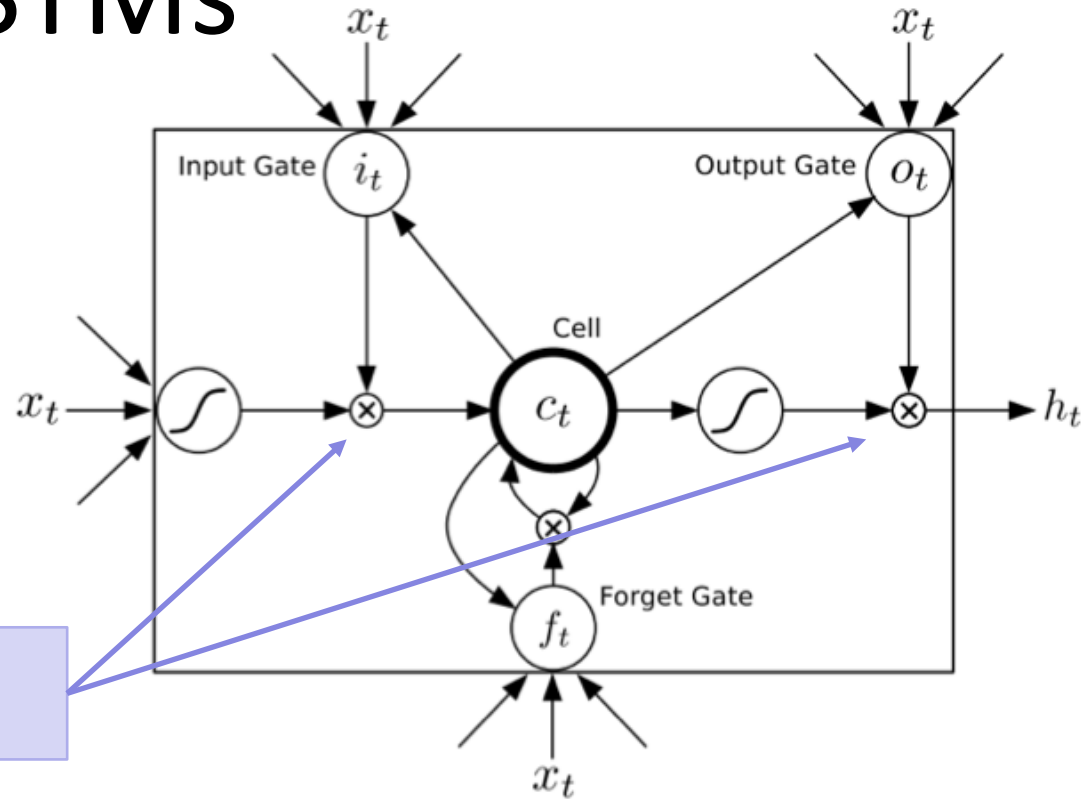
$$\mathbf{f}_t = \sigma(W_{xf}\mathbf{x}_t + W_{hf}\mathbf{h}_{t-1} + W_{cf}\mathbf{c}_{t-1} + \mathbf{b}_f),$$

$$\mathbf{c}_t = \mathbf{f}_t\mathbf{c}_{t-1} + \mathbf{i}_t \tanh(W_{xc}\mathbf{x}_t + W_{hc}\mathbf{h}_{t-1} + \mathbf{b}_c),$$

LSTMs



Element-wise multiply



$$\mathbf{i}_t = \sigma(W_{xi}\mathbf{x}_t + W_{hi}\mathbf{h}_{t-1} + W_{ci}\mathbf{c}_{t-1} + \mathbf{b}_i),$$

$$\mathbf{f}_t = \sigma(W_{xf}\mathbf{x}_t + W_{hf}\mathbf{h}_{t-1} + W_{cf}\mathbf{c}_{t-1} + \mathbf{b}_f),$$

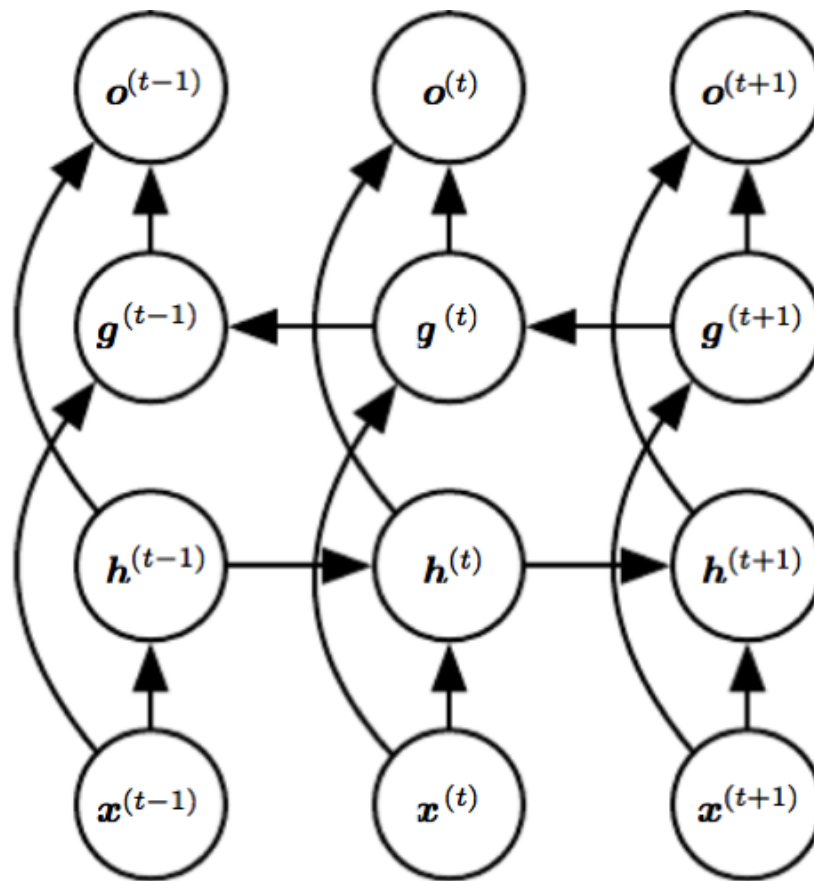
$$\mathbf{c}_t = \mathbf{f}_t\mathbf{c}_{t-1} + \mathbf{i}_t \tanh(W_{xc}\mathbf{x}_t + W_{hc}\mathbf{h}_{t-1} + \mathbf{b}_c),$$

$$\mathbf{o}_t = \sigma(W_{xo}\mathbf{x}_t + W_{ho}\mathbf{h}_{t-1} + W_{co}\mathbf{c}_t + \mathbf{b}_o),$$

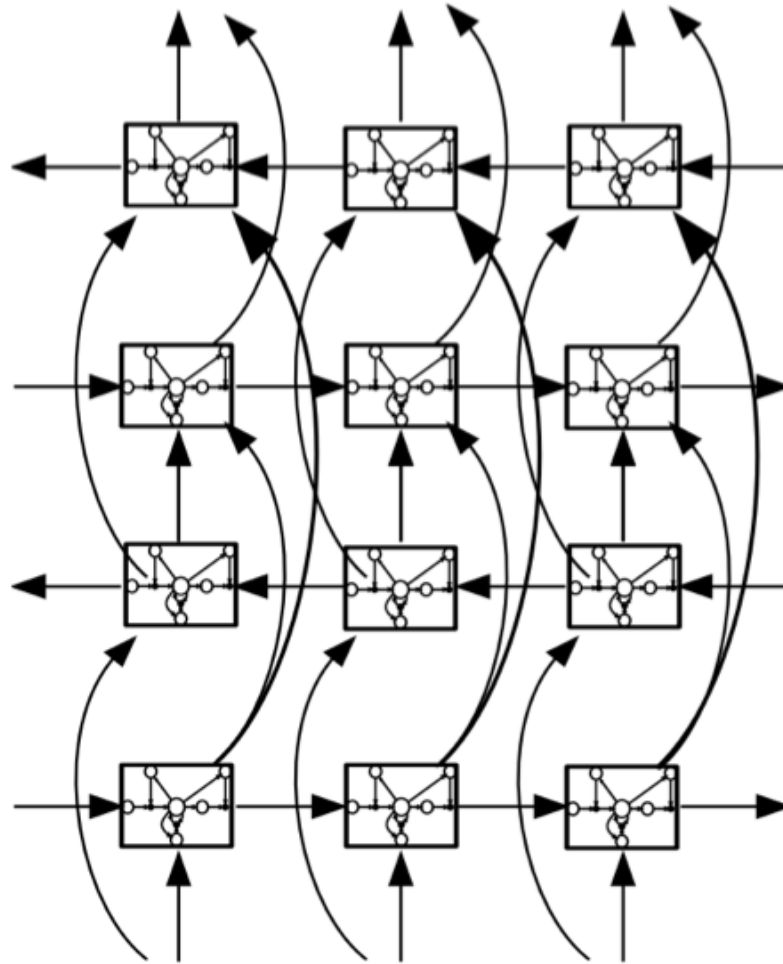
$$\mathbf{h}_t = \mathbf{o}_t \tanh(\mathbf{c}_t).$$

Bi-directional Recurrent Neural Networks

- Key idea: processing of word at position t can depend on following words too, not just preceding words



Deep Bidirectional LSTM Network



["Hybrid Speech Recognition with Deep Bidirectional LSTM,"
Graves et al., 2013]

Gated Recurrent Units (GRUs)

Element-wise
multiply

\circ denotes the **Hadamard product**. $h_0 = 0$.

$$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z)$$

$$r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r)$$

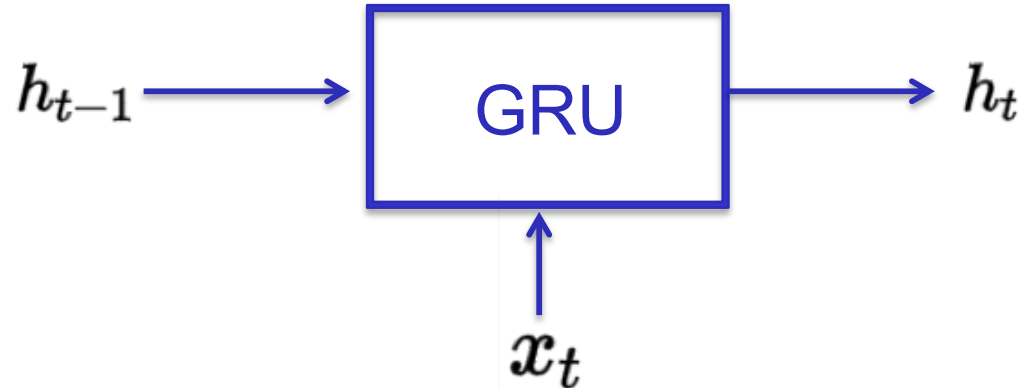
$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \sigma_h(W_h x_t + U_h(r_t \circ h_{t-1}) + b_h)$$

Variables

- x_t : input vector
- h_t : output vector
- z_t : update gate vector
- r_t : reset gate vector
- W , U and b : parameter matrices and vector

Activation functions

- σ_g : The original is a **sigmoid function**.
- σ_h : The original is a **hyperbolic tangent**.

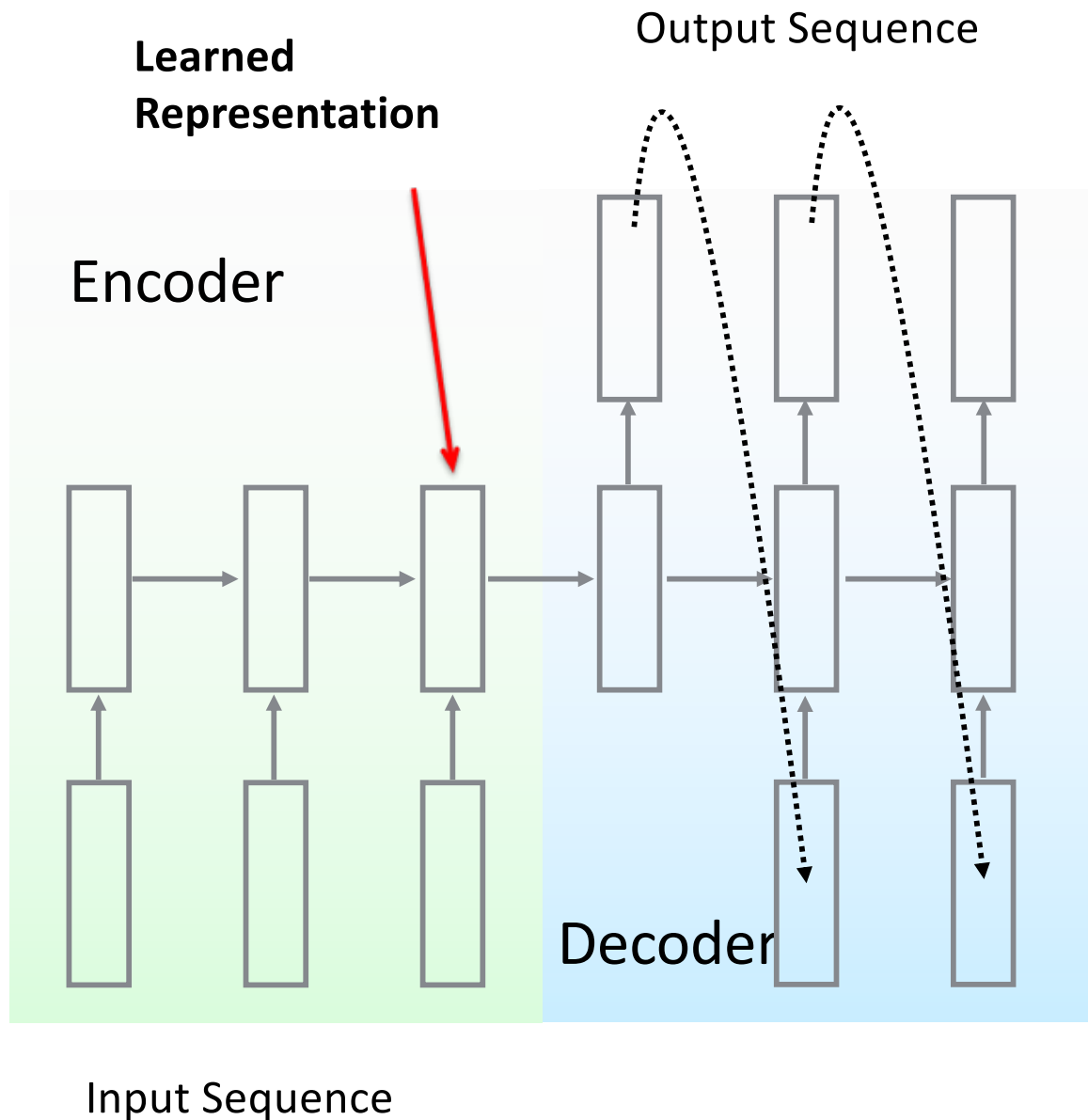


fewer parameters than LSTM
found equally effective in
some experiments involving

- speech recognition
- music analysis

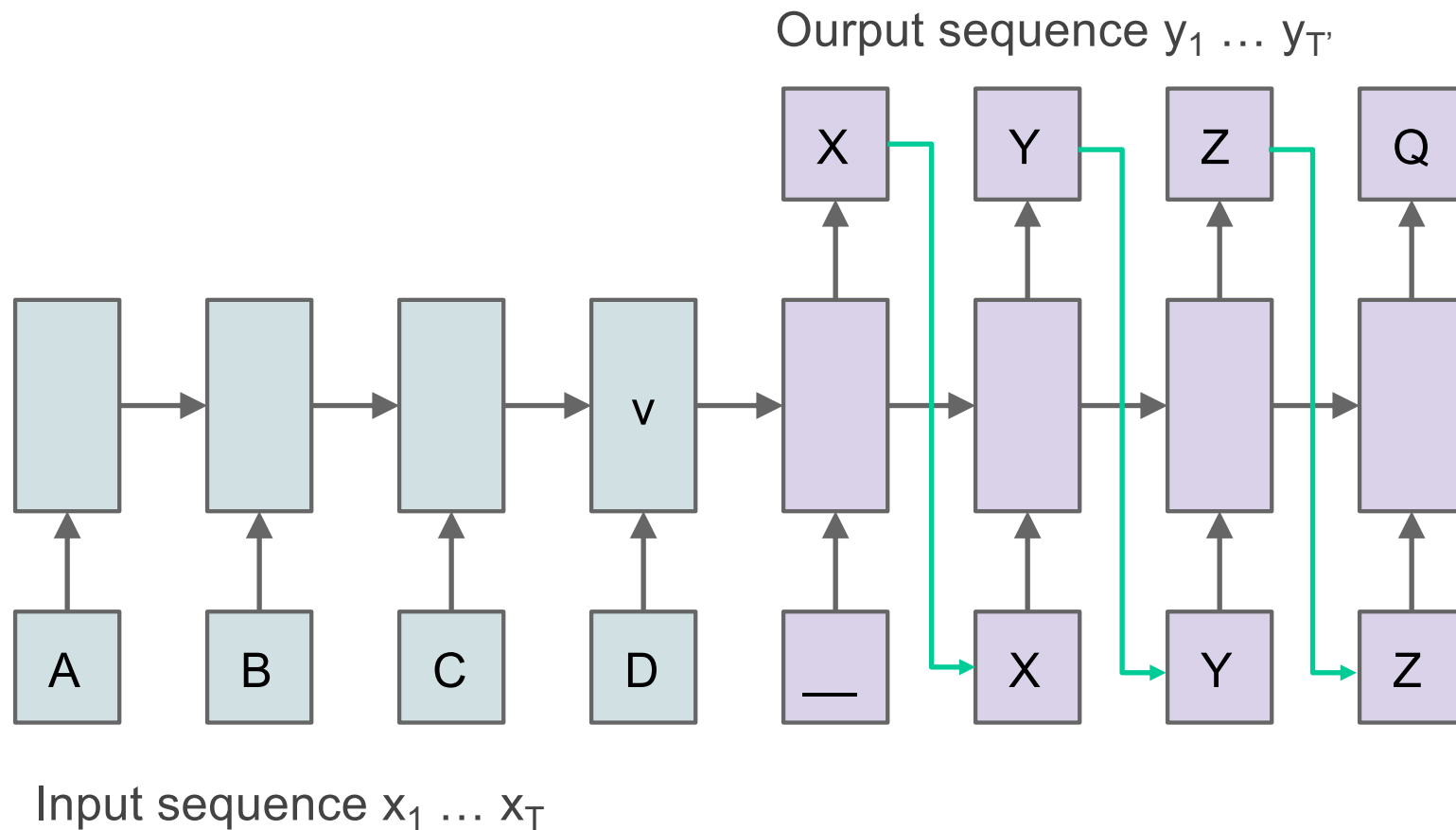
see [Chung et al., 2014]

Sequence to Sequence Learning



- RNN Encoder-Decoders for Machine Translation (Sutskever et al. 2014; Cho et al. 2014; Kalchbrenner et al. 2013, Srivastava et.al., 2015)

Seq2Seq. Encoder-Decoder Architecture

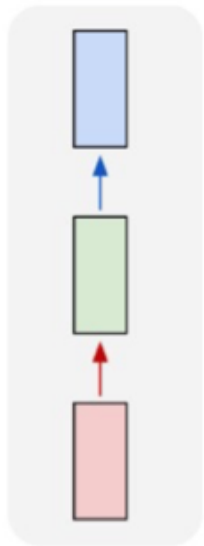


$$P(y_1, \dots, y_{T'} | x_1, \dots, x_T) = \prod_{t=1}^{T'} p(y_t | v, y_1, \dots, y_{t-1})$$

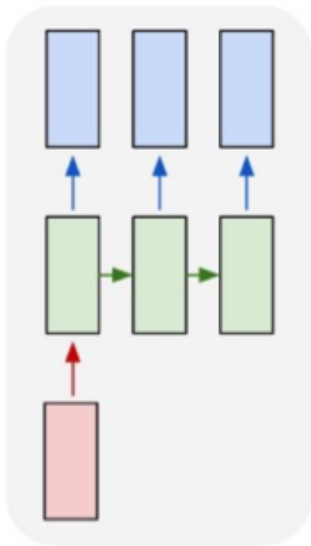
Sequence to Sequence Models

- machine translation
- text summarization
- text to computer command
- ...

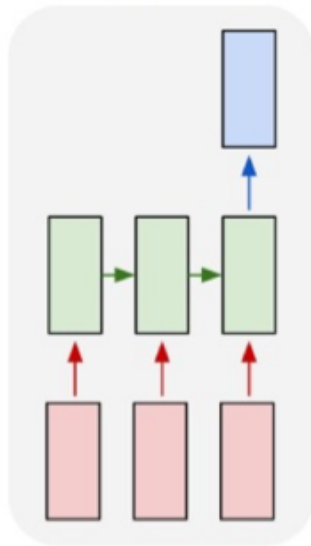
one to one



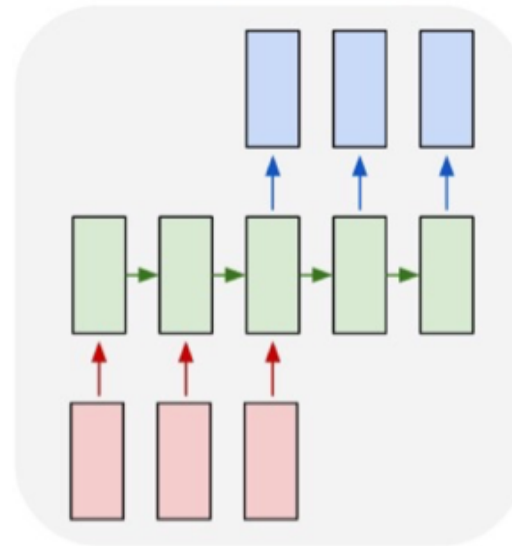
one to many



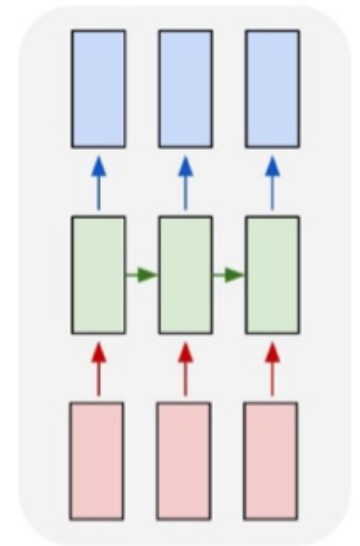
many to one



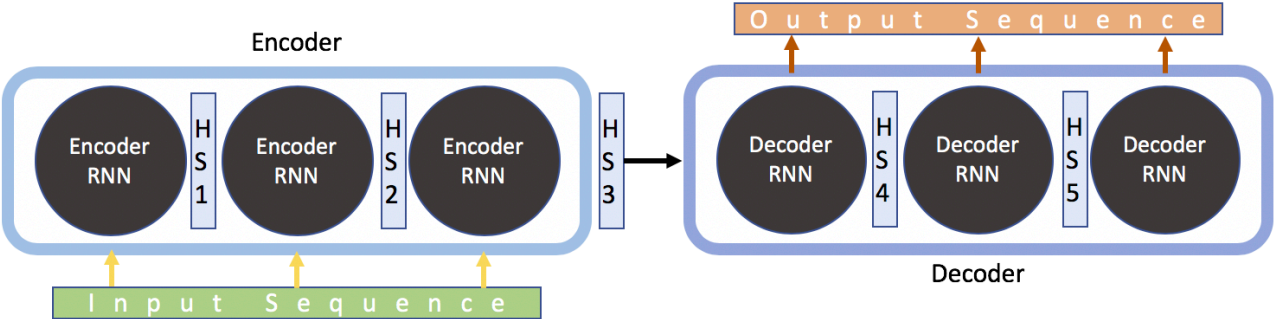
many to many



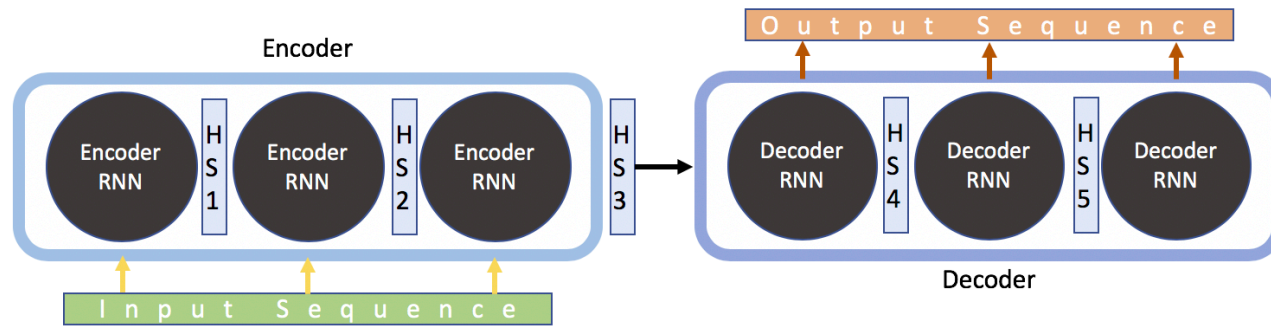
many to many



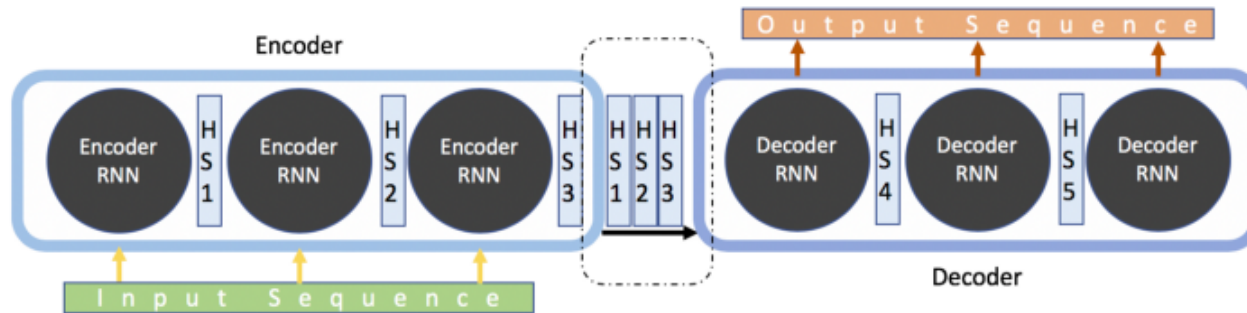
Problem: HS3 has to encode entire sequence...



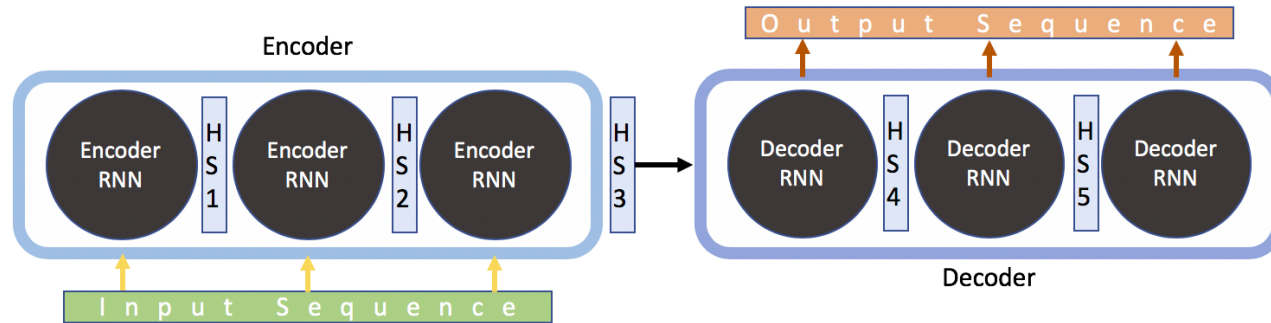
Problem: HS3 has to encode entire sequence...



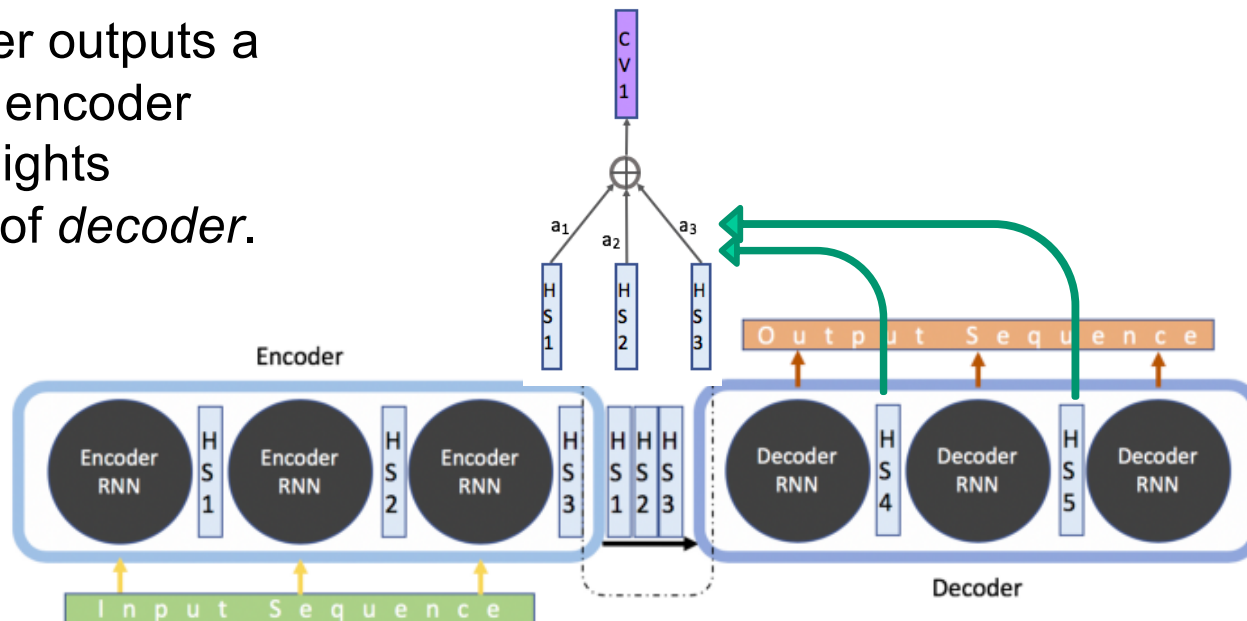
Maybe do this?



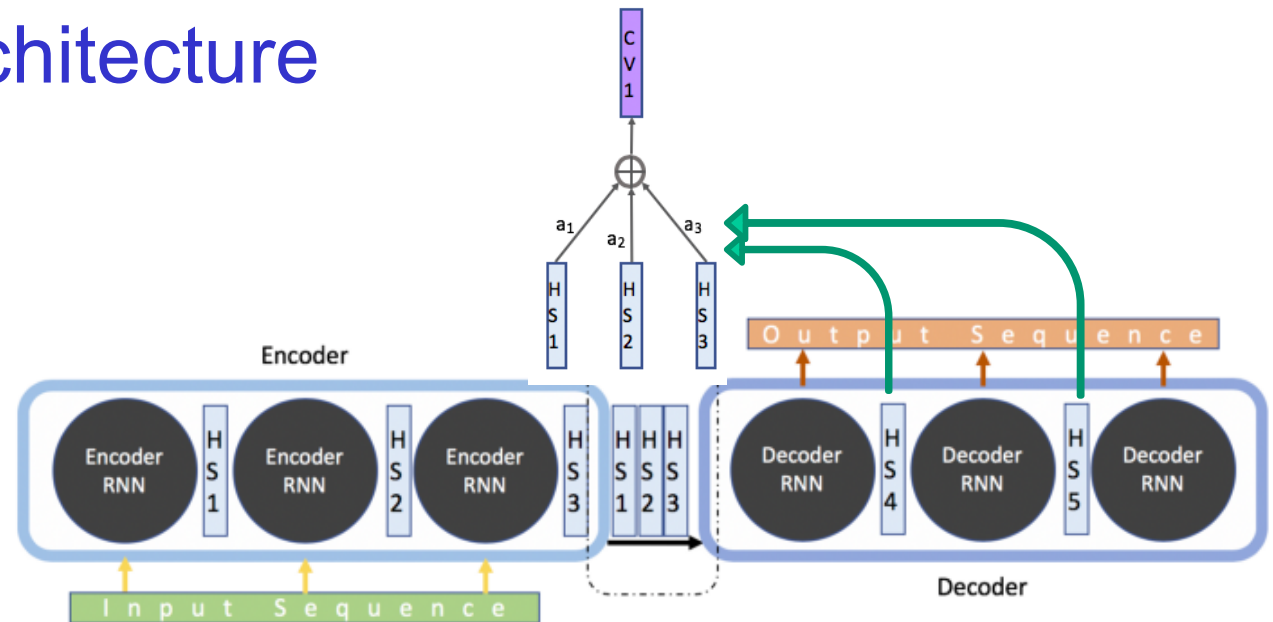
Problem: HS3 has to encode entire sequence...



Attention: encoder outputs a weighted avg. of encoder states, where weights depend on state of *decoder*.



Transformer Architecture



Transformer architecture uses attention in three ways:

1. By decoder: Attention on encoder states, based on decoder state
2. Inside encoder: Replace RNN by *self-attention* across input tokens
3. Inside decoder: Replace RNN by *self-attention* across output tokens

Scaled Dot Product Attention

Given:

- Set of <key, value> pairs $\langle \mathbf{k}_i, \mathbf{v}_i \rangle$
 - \mathbf{k}_i vector of dim d_k
 - \mathbf{v}_i vector of dim d_v
- Query \mathbf{q} which is vector of dim d_k

Return:

- Vector of dim d_v which is a weighted sum of the \mathbf{v}_i
- where weights given by the Softmax of dot products $\mathbf{q} \mathbf{k}_i / \text{sqrt}(d_k)$

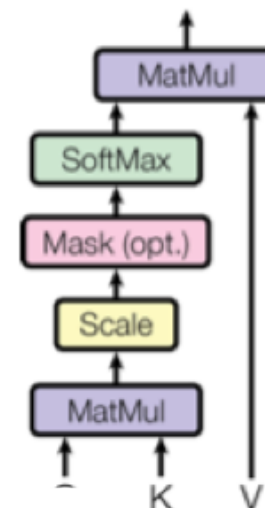
3.2.1 Scaled Dot-Product Attention

We call our particular attention "Scaled Dot-Product Attention" (Figure 2). The input consists of queries and keys of dimension d_k , and values of dimension d_v . We compute the dot products of the query with all keys, divide each by $\sqrt{d_k}$, and apply a softmax function to obtain the weights on the values.

In practice, we compute the attention function on a set of queries simultaneously, packed together into a matrix Q . The keys and values are also packed together into matrices K and V . We compute the matrix of outputs as:

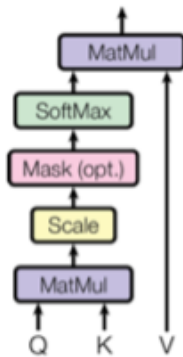
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (1)$$

Scaled Dot-Product Attention

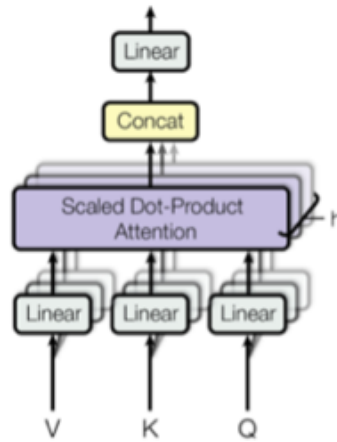


Multi-Head Attention

Scaled Dot-Product Attention



Multi-Head Attention



3.2.1 Scaled Dot-Product Attention

We call our particular attention "Scaled Dot-Product Attention" (Figure 2). The input consists of queries and keys of dimension d_k , and values of dimension d_v . We compute the dot products of the query with all keys, divide each by $\sqrt{d_k}$, and apply a softmax function to obtain the weights on the values.

In practice, we compute the attention function on a set of queries simultaneously, packed together into a matrix Q . The keys and values are also packed together into matrices K and V . We compute the matrix of outputs as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (1)$$

Transformer Architecture

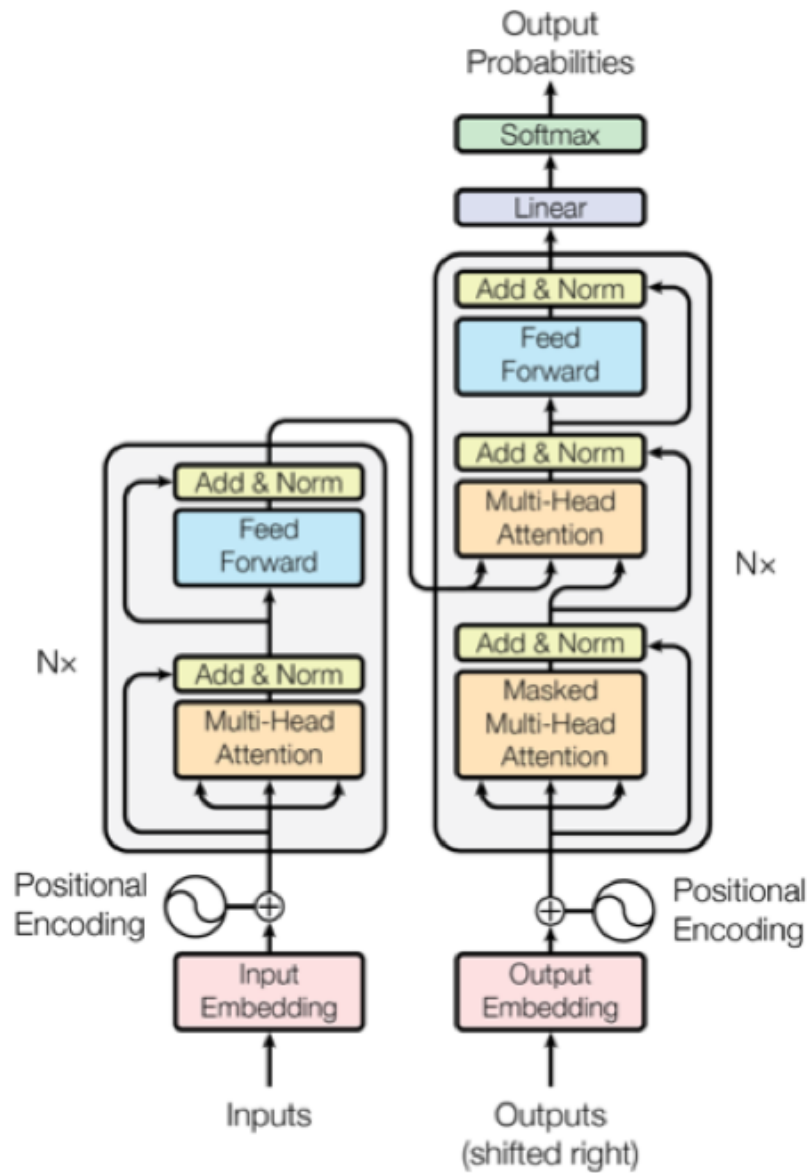
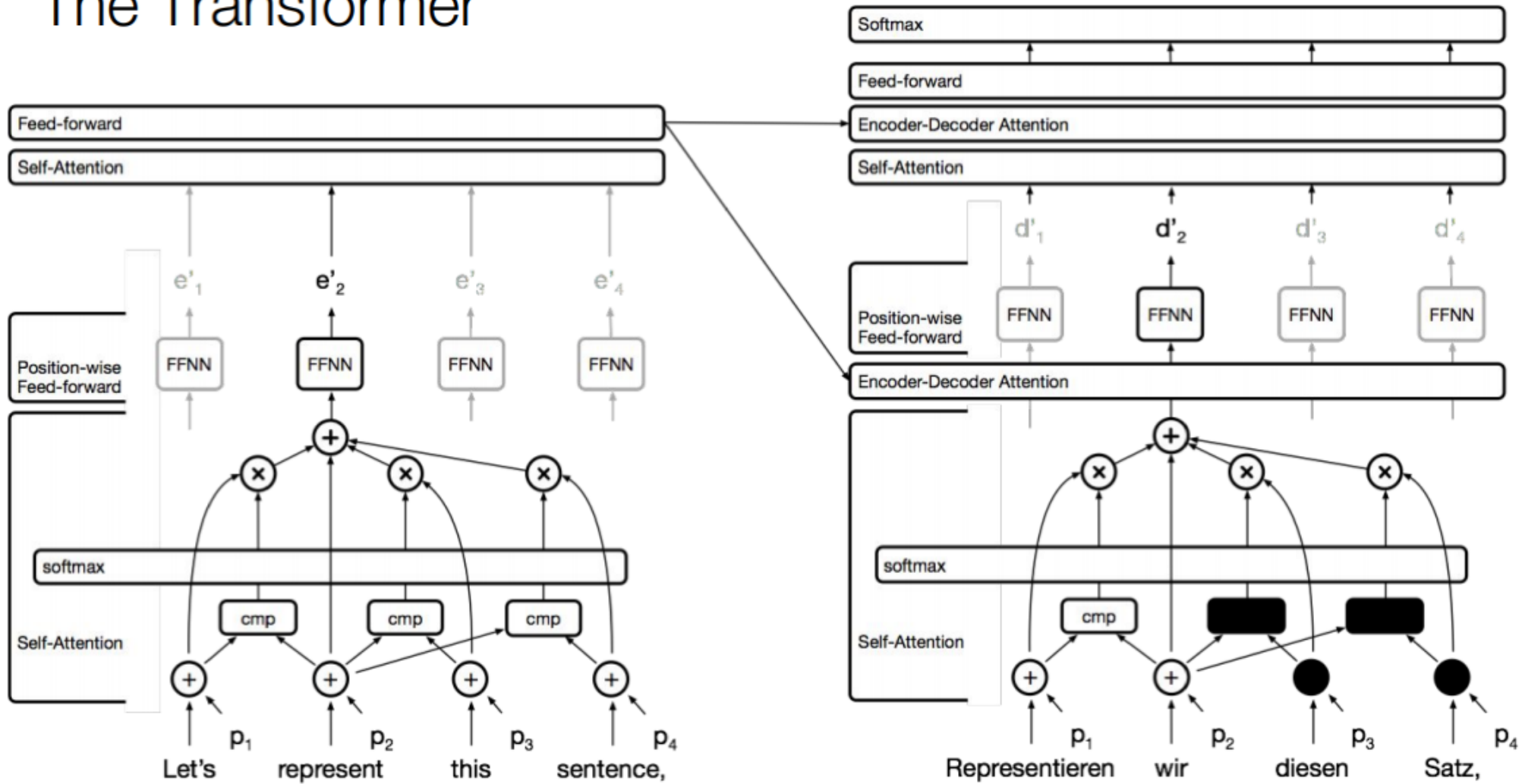
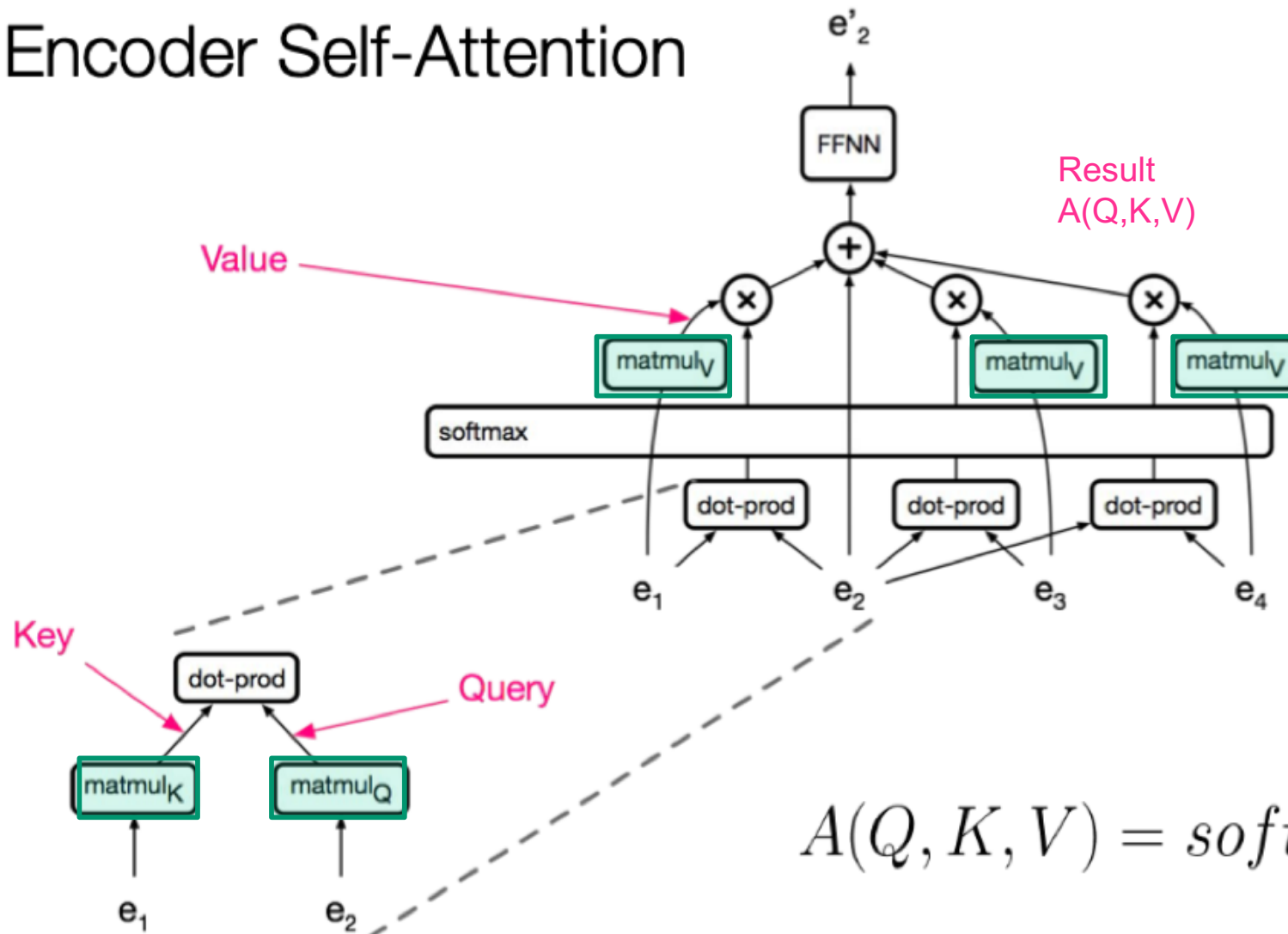


Figure 1: The Transformer - model architecture.

The Transformer

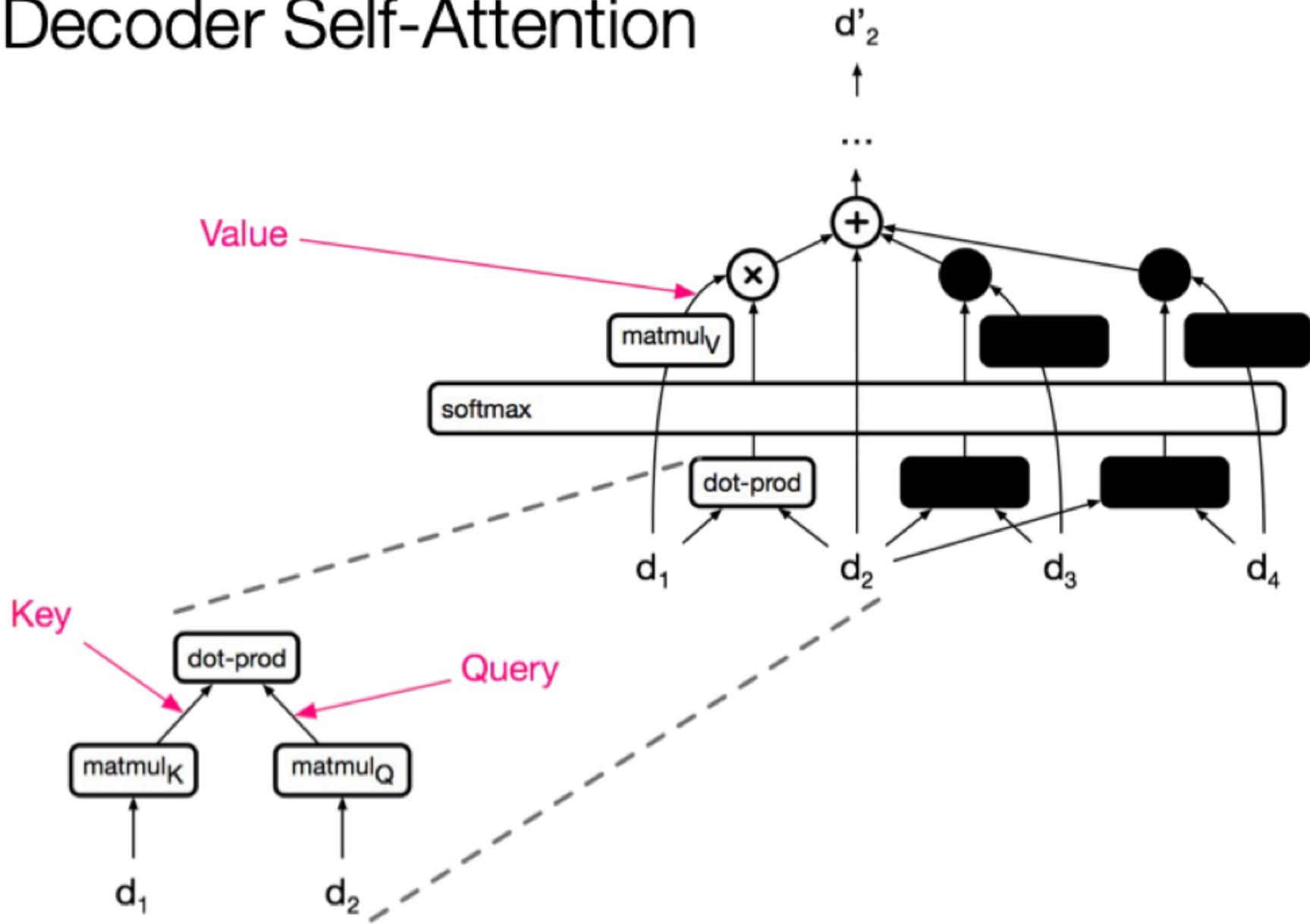


Encoder Self-Attention

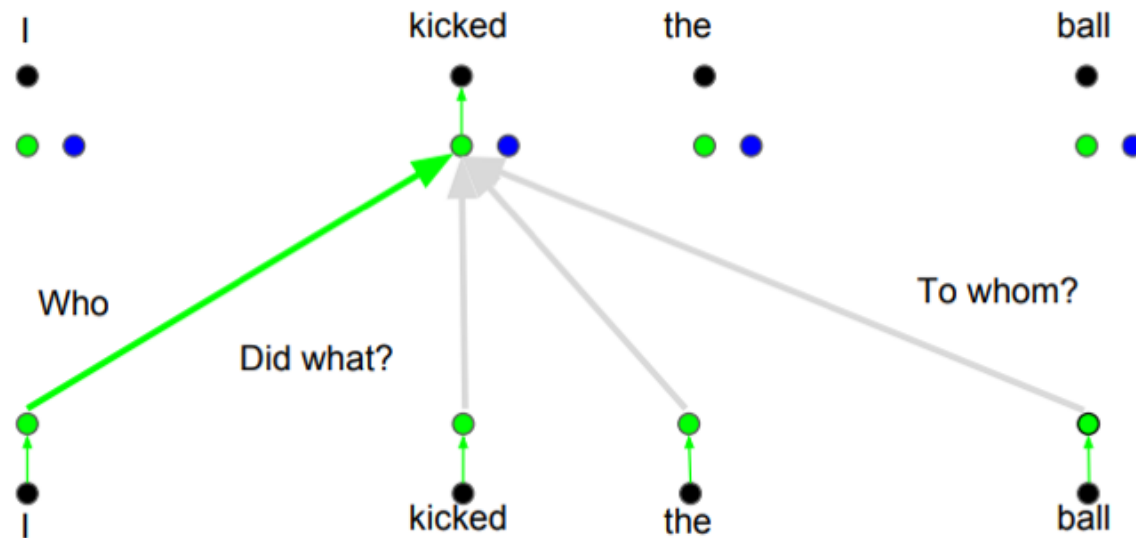
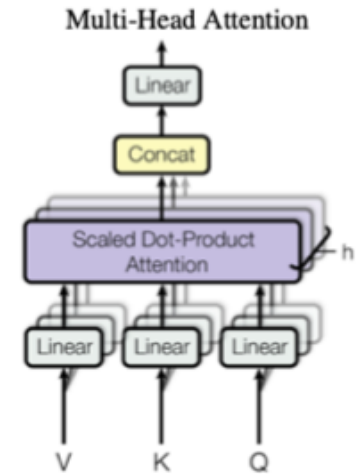


$$A(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

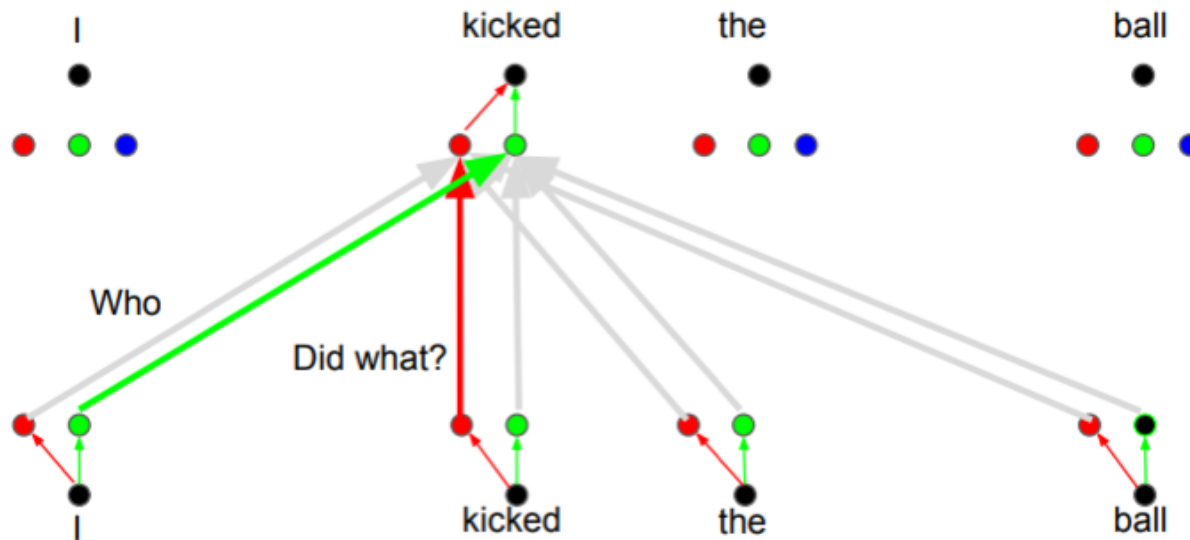
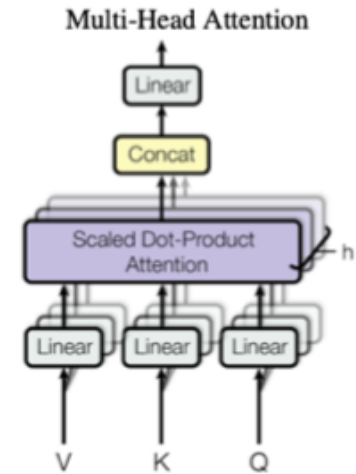
Decoder Self-Attention



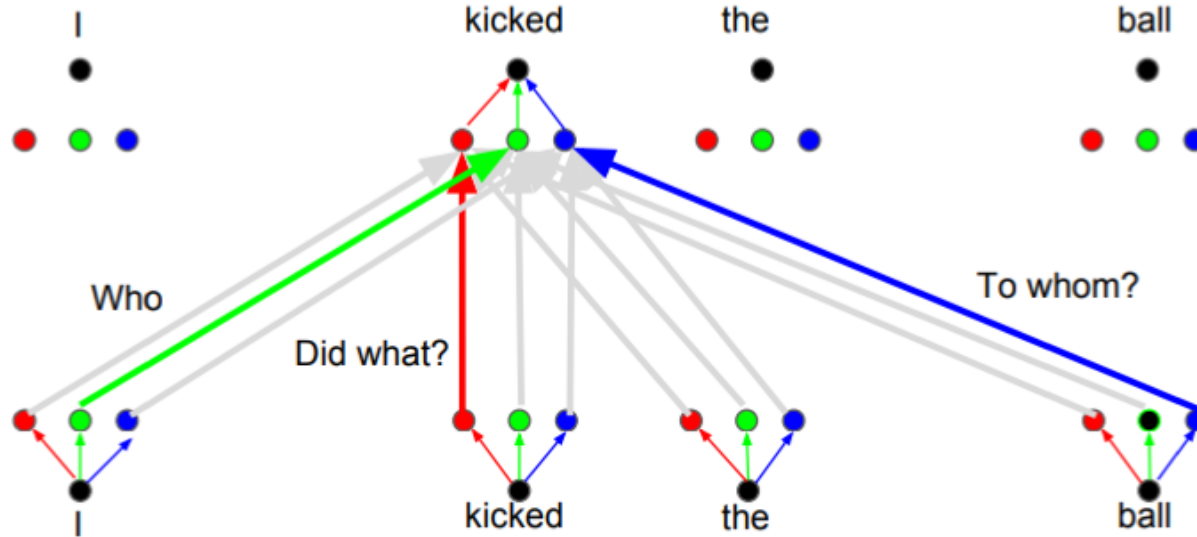
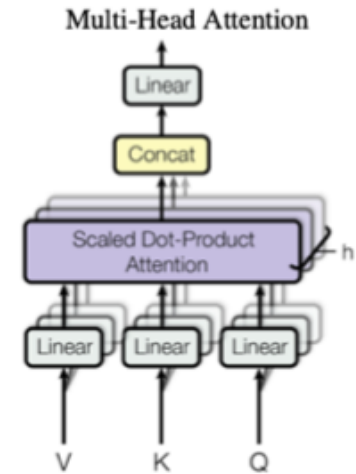
Attention head: Who



Parallel attention heads



Parallel attention heads



Attention is Cheap!

FLOPs

Self-Attention	$O(\text{length}^2 \cdot \text{dim})$	$= 4 \cdot 10^9$
RNN (LSTM)	$O(\text{length} \cdot \text{dim}^2)$	$= 16 \cdot 10^9$
Convolution	$O(\text{length} \cdot \text{dim}^2 \cdot \text{kernel_width})$	$= 6 \cdot 10^9$

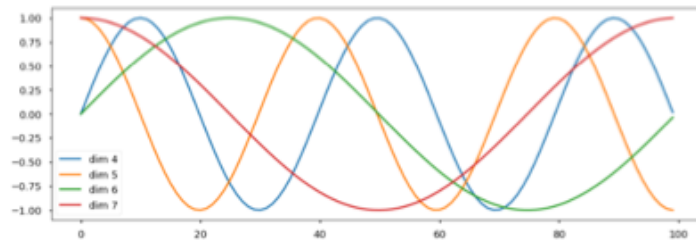
length=1000 dim=1000 kernel_width=3

Yields Significant Improvement in Machine Translation

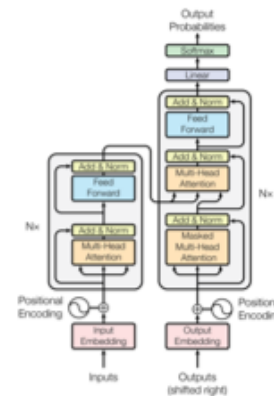
More details:

See <http://nlp.seas.harvard.edu/2018/04/03/attention.html>

- Multiple sine waves added as positional encoding of input tokens

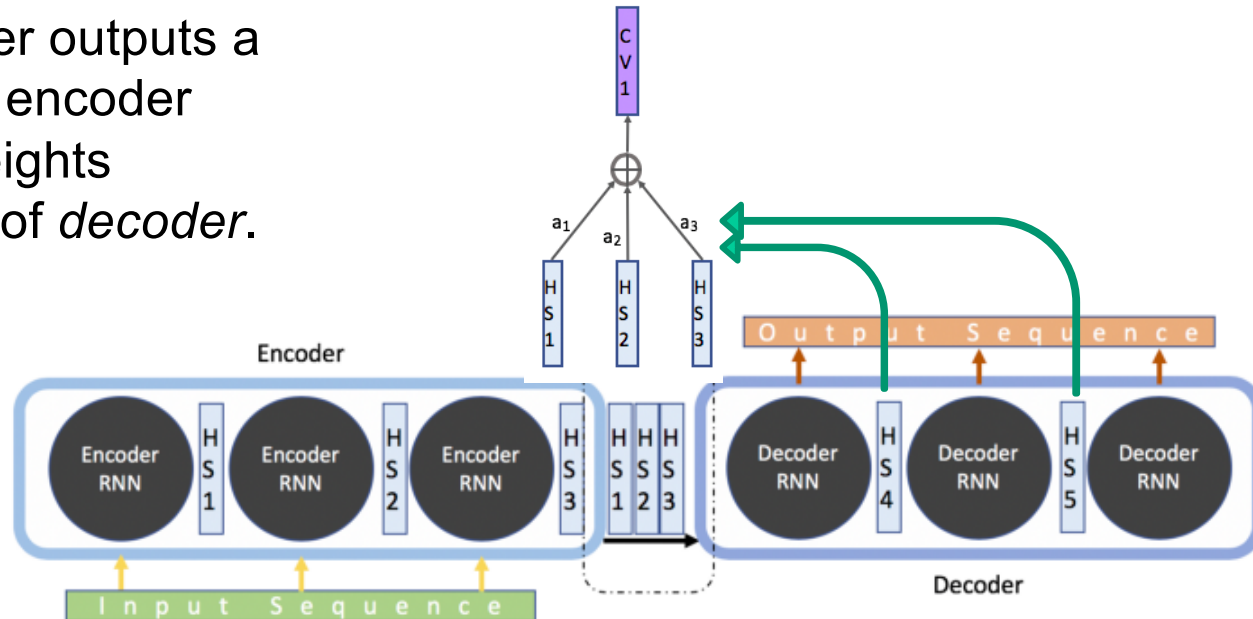


- Dropout during training at every layer
- Layer-norm
- ADAM optimizer with learning rate warmup (warmup + exponential decay in learning rate)
- Auto-regressive decoding with beam search and length biasing
- ...



How to Think About Transformers?

Attention: encoder outputs a weighted avg. of encoder states, where weights depend on state of *decoder*.



General program schema

- “Decoder” outputs a sequence of tokens
- Based on its perceived input, plus what it has already output
- With attention mechanism to focus on relevant subset of its input and output
- Learned parameters define both attention and operations it performs

BERT: Bidirectional Encoder Representations from Transformers

Goal: Trained model that will produce generally useful encodings of arbitrary text

- Uses transformer architecture
- Bidirectional attention across entire input
- Accept input sequences with multiple sentences
 - (e.g., question/answer pairs)
 - Special token [CLS] indicates beginning of sequence, output vector embedding for this token represents sequence for classification tasks
 - Special token [SEP] indicates beginning of new sentence
- Train by
 - masking out 15% of words, and predicting them
 - classify whether second sentence actually follows the first sentence
 - True in 50% of cases
- 24 layers deep, hidden unit dim = 1024, 12 self-attention heads, 340M trained parameters

BERT: Bidirectional Encoder Representations from Transformers

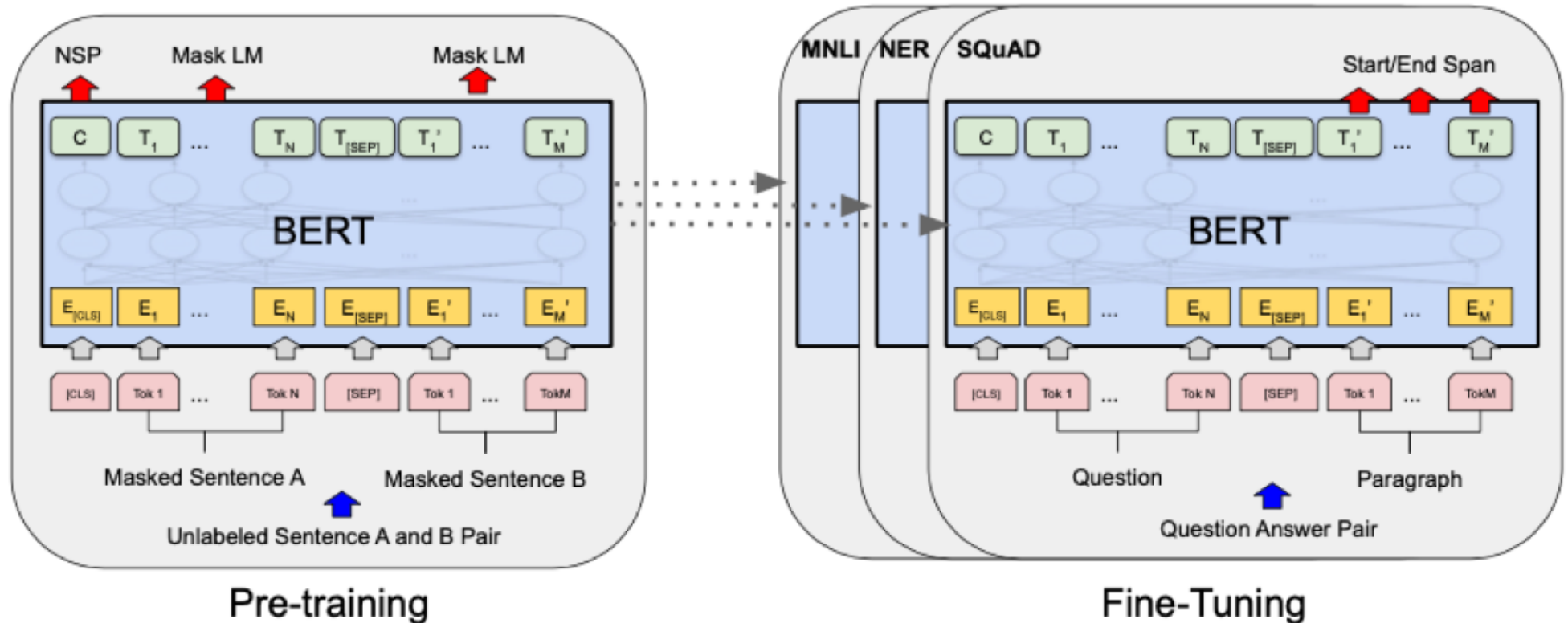


Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. [CLS] is a special symbol added in front of every input example, and [SEP] is a special separator token (e.g. separating questions/answers).

BERT: Bidirectional Encoder Representations from Transformers

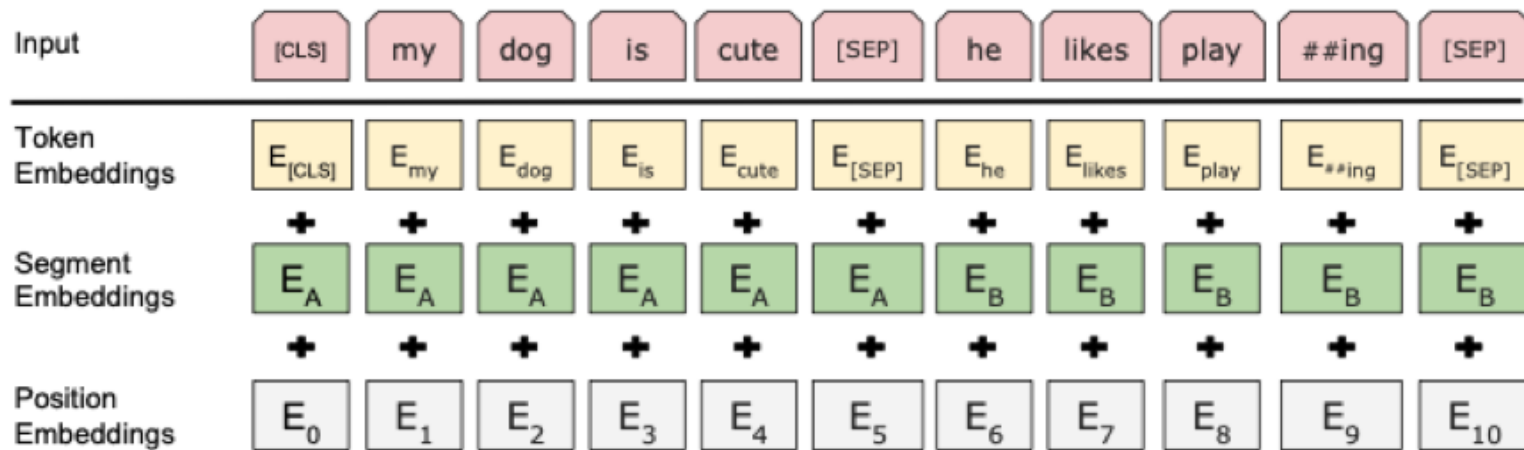


Figure 2: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

* “segment embedding” is a learned embedding indicating either sentence A or sentence B

BERT: Bidirectional Encoder Representations from Transformers

Goal: Trained model that will produce generally useful encodings of words and sentences

Fine tuning for new tasks:

- Adding an output layer for new tasks (Q/A, textual entailment, sentiment analysis, equivalence of two questions, ..) then fine-tuning by further training, advances state of the art performance on many language tasks
- Can “freeze” the 340M trained parameters, and fine-tune by training only the new output layer
- Or, fine tune end-to-end, tuning all parameters

COMET

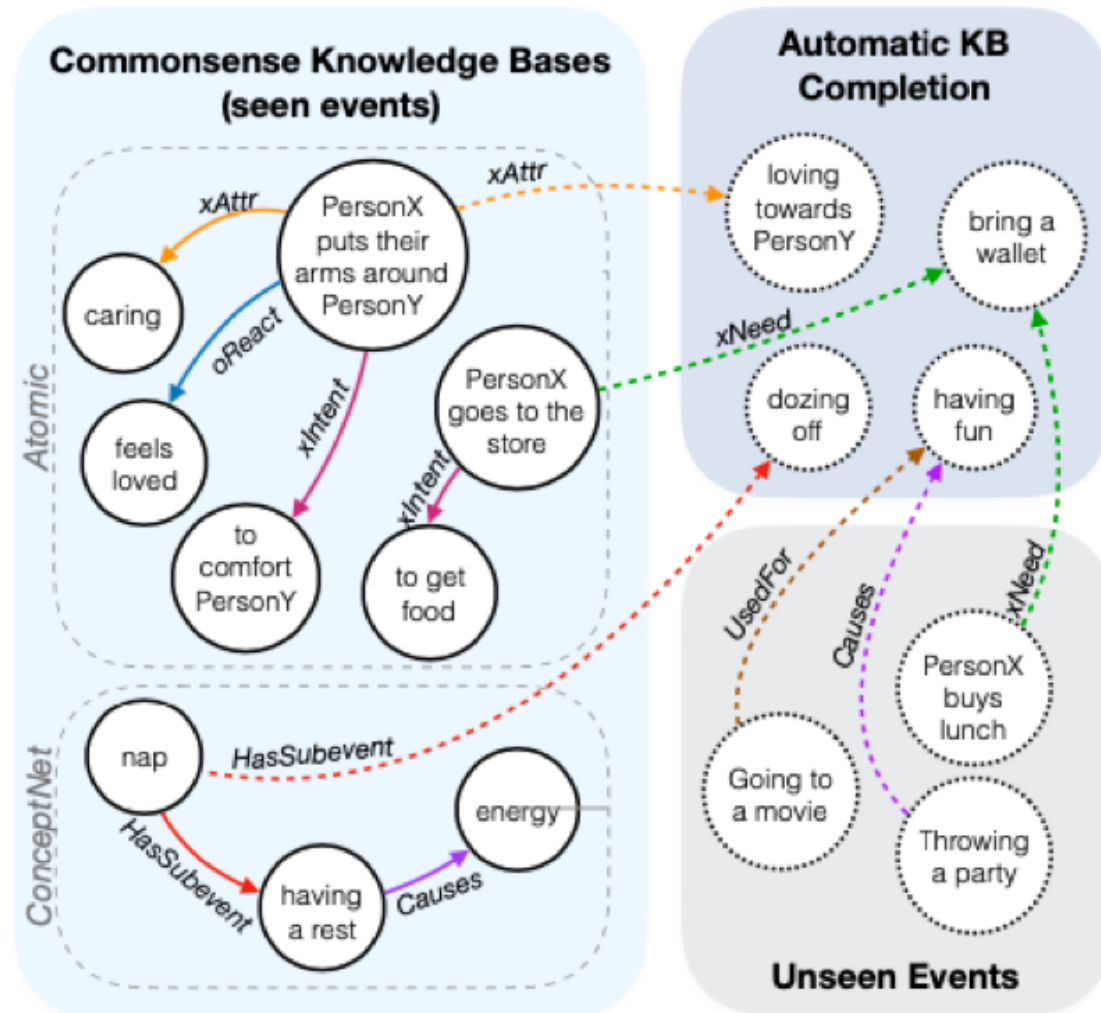



Figure 1: COMET  learns from an existing knowledge base (solid lines) to be able to generate novel nodes and edges (dashed lines).

COMET

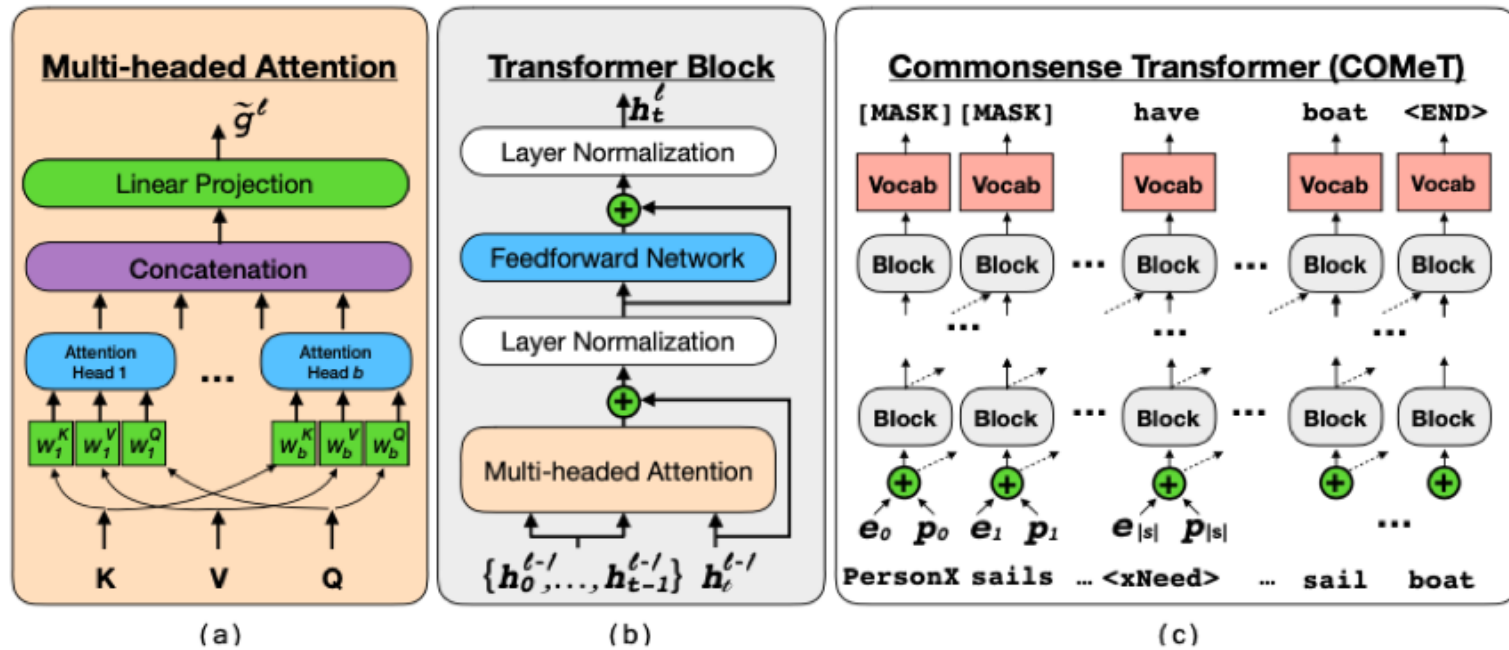


Figure 2: Model diagram. (a) In the multi-headed attention module, the key, value, and query all pass through a head-specific projection before a scaled dot-product attention is computed between them. The outputs of the heads are concatenated and projected. (b) Inside the transformer block, the outputs of all the previous layer blocks from earlier time steps are input to the multi-headed attention with the preceding block for the current time step as the query. (c) Each token is an input to a first-layer block along with all preceding tokens. Dotted lines indicate outputs to all future blocks in the next layer and inputs from all preceding blocks in the previous layer.

COMET

Seed Concept	Relation	Generated	Plausible
X holds out X's hand to Y	xAttr	helpful	✓
X meets Y eyes	xAttr	intense	✓
X watches Y every ____	xAttr	observant	✓
X eats red meat	xEffect	gets fat	✓
X makes crafts	xEffect	gets dirty	✓
X turns X's phone	xEffect	gets a text	
X pours ____ over Y's head	oEffect	gets hurt	✓
X takes Y's head off	oEffect	bleeds	✓
X pisses on Y's bonfire	oEffect	gets burned	
X spoils somebody rotten	xIntent	to be mean	
X gives Y some pills	xIntent	to help	✓
X provides for Y's needs	xIntent	to be helpful	✓
X explains Y's reasons	xNeed	to know Y	✓
X fulfils X's needs	xNeed	to have a plan	✓
X gives Y everything	xNeed	to buy something	✓
X eats pancakes	xReact	satisfied	✓
X makes ____ at work	xReact	proud	✓
X moves house	xReact	happy	✓
X gives birth to the Y	oReact	happy	✓
X gives Y's friend ____	oReact	grateful	✓
X goes ____ with friends	oReact	happy	✓
X gets all the supplies	xWant	to make a list	✓
X murders Y's wife	xWant	to hide the body	✓
X starts shopping	xWant	to go home	✓
X develops Y theory	oWant	to thank X	✓
X offer Y a position	oWant	to accept the job	✓
X takes ____ out for dinner	oWant	to eat	✓

Demo: https://mosaickg.apps.allenai.org/comet_atomic

Table 5: Generations that were **randomly selected** from a subset of **novel** generations from the ATOMIC development set. A novel generation is a *sro* tuple not found in the training set. Manual evaluation of each tuple indicates whether the tuple is considered plausible by a human annotator.

Programming Frameworks for Deep Nets

- Pytorch (Facebook)
- TensorFlow (Google)
- TFLearn (runs on top of TensorFlow, but simpler to use)
- Theano (University of Montreal)
- CNTK (Microsoft)
- Keras (can run on top of Theano, CNTK, TensorFlow)
- ...

Many support use of Graphics Processing Units (GPU's)

Major factor in dissemination of Deep Network technology

```
# Specify that all features have real-value data
feature_columns = [tf.feature_column.numeric_column("x", shape=[4])]

# Build 3 layer DNN with 10, 20, 10 units respectively.
classifier = tf.estimator.DNNClassifier(feature_columns=feature_columns,
                                       hidden_units=[10, 20, 10],
                                       n_classes=3,
                                       model_dir="/tmp/iris_model")

# Define the training inputs
train_input_fn = tf.estimator.inputs.numpy_input_fn(
    x={"x": np.array(training_set.data)},
    y=np.array(training_set.target),
    num_epochs=None,
    shuffle=True)

# Train model.
classifier.train(input_fn=train_input_fn, steps=2000)

# Define the test inputs
test_input_fn = tf.estimator.inputs.numpy_input_fn(
    x={"x": np.array(test_set.data)},
    y=np.array(test_set.target),
    num_epochs=1,
    shuffle=False)

# Evaluate accuracy.
accuracy_score = classifier.evaluate(input_fn=test_input_fn)["accuracy"]

print("\nTest Accuracy: {0:f}\n".format(accuracy_score))
```

TensorFlow example

Modern Deep Networks: 2021 vs 1987

- vastly more online data
- GPU's, TPU's
- Heterogenous units
 - Relu, sigmoid, tanh, linear
- including units composed from other units
 - LSTM, GRU, Attention, ...
- many new architectures
 - 100 layers deep, bidirectional LSTMs, Convolutional nets, Transformers...
- New ideas for gradient descent
 - dropout, batch normalization, Adagrad, layer normalization, ...
- unification with probabilistic models
 - train to output probabilities
- frameworks like TensorFlow
- online text with code: Dive into Deep Learning

What you should know:

- Representation learning
 - Hidden layers re-represent inputs in form to predict outputs
 - Autoencoders
 - Sometimes reused widely (e.g., word2vec word embeddings)
- Convolutional neural networks
 - Convolution provides translation invariance
 - Network stages with reducing spatial resolution, Mult. channels,...
- Recurrent neural networks
 - Learn to represent history in time series
 - Backpropagation as unfolding in time
 - LSTM memory units
- Neural architectures
 - Shared parameters across multiple computations
 - Layers with different structures/functions
 - RNN's, Seq2Seq, Transformer, ...
 - Probabilistic classification → output Softmax layer