

10707

Deep Learning: Spring 2021

Andrej Risteski

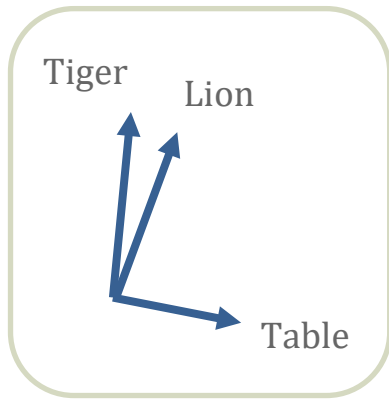
Machine Learning Department

Lecture 21:

The characters of modern NLP –
transformers, ELMo and BERT

Word embeddings

Semantically meaningful **vector representations** of words

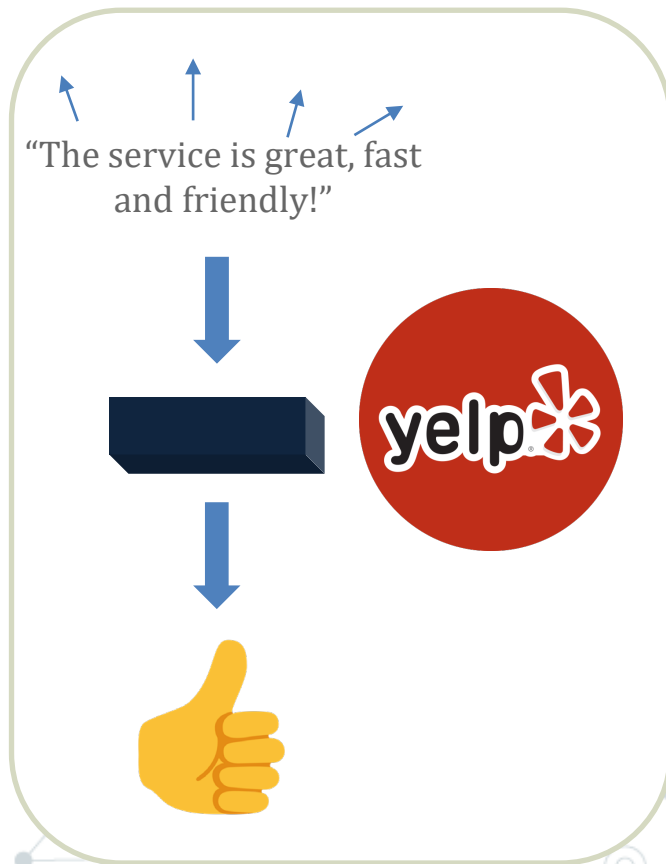


Example: Inner product (possibly scaled, i.e. cosine similarity) correlates with word similarity.



Word embeddings

Semantically meaningful **vector representations** of words



Example: Can use embeddings to do sentiment classification by training a simple (e.g. linear) classifier

Word embeddings

Semantically meaningful **vector representations** of words



Example: Can train a “simple” network that if fed word embeddings for two languages, can effectively translate.



Word embeddings

The main issue: embeddings are context-independent.

Seems intuitively wrong: determining the meaning of “bank” in
“*Walking along the river bank*” and “*A bank accepts deposits*”



depends on the surrounding context (“river” and “deposits” being the strong indicators of meaning).

One way to handle this? Recurrent neural nets!



Recurrent neural networks

Recurrent neural networks: autoregressive model that can be extended to arbitrary length sequences.

$$h_i = \tanh(W_{hh}h_{i-1} + W_{xh}x_i)$$

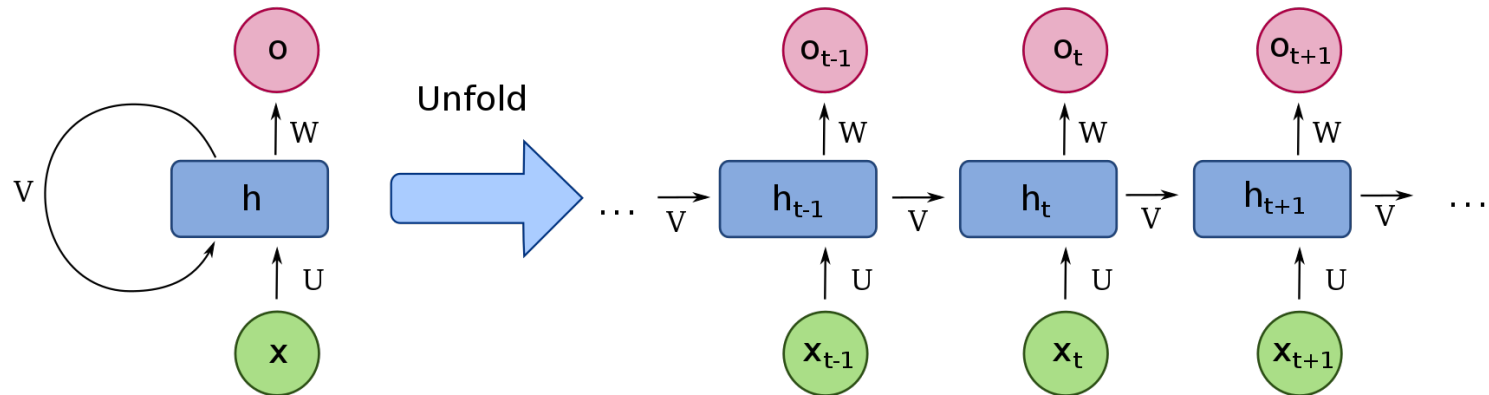
$$o_i = W_{hy}h_i$$

o_i specifies parameters for $p(x_i|x_{<i})$, e.g. $\text{softmax}(y_i)$

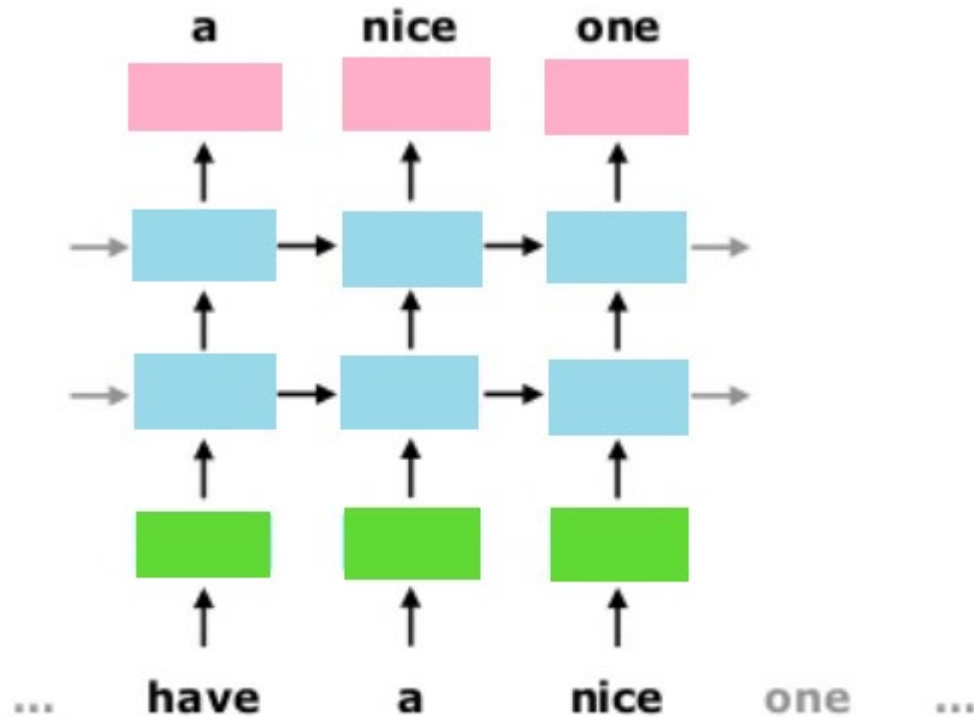
Repurposing RNNs to learn context-dependent embeddings:

think of x_i as one-hot encoding of words, W_{xh} as a $\text{dim}(h)$ word embedding matrix, and h as a context-dependent encoding

Recurrent neural networks: graphical representation

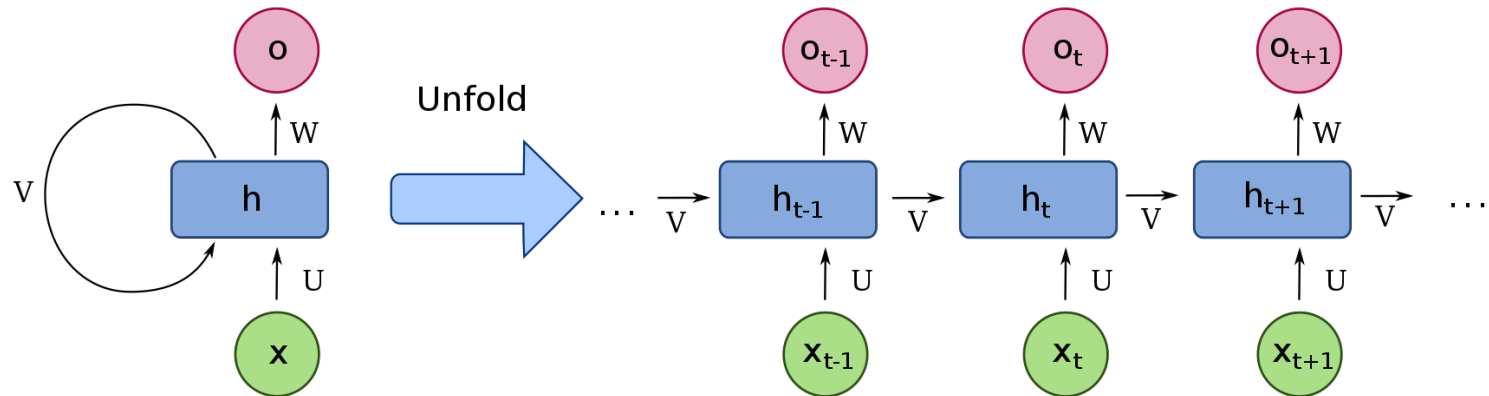


Recurrent neural networks: stacking multiple RNNs



It is quite natural to stack multiple RNNs in the obvious way

Recurrent neural networks



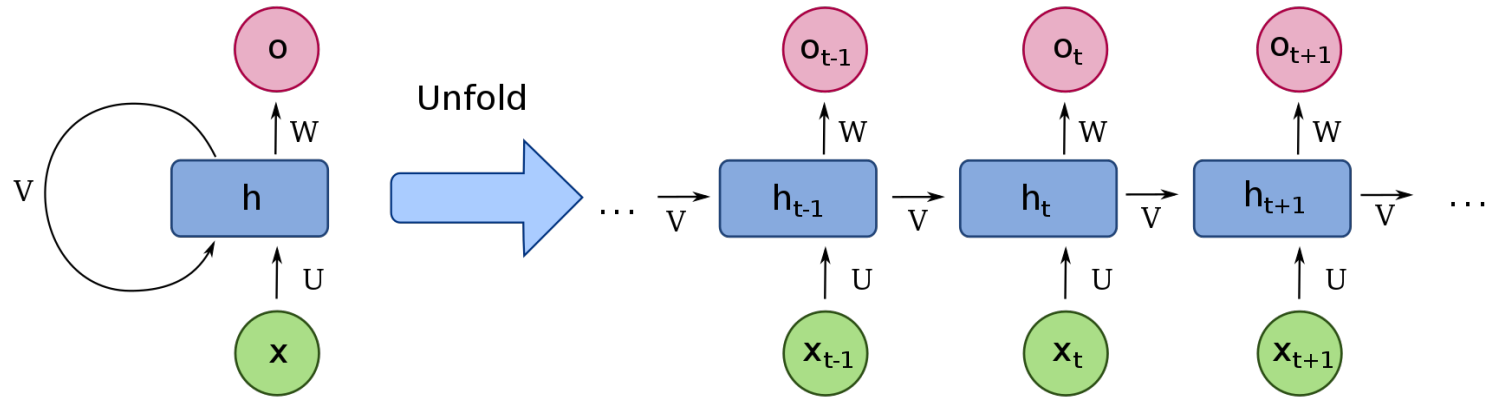
Obvious benefit: copious weight tying, number of params completely independent of length.

Drawbacks:

Training: backpropagation. Via unfolding equivalence above, same as calculating derivative of a length- t feedforward network.

Same problem as in very deep networks: *gradient explosion/vanishing!*

Recurrent neural networks



Obvious benefit: copious weight tying, number of params completely independent of length.

Drawbacks:

Training: *gradient explosion/vanishing!*

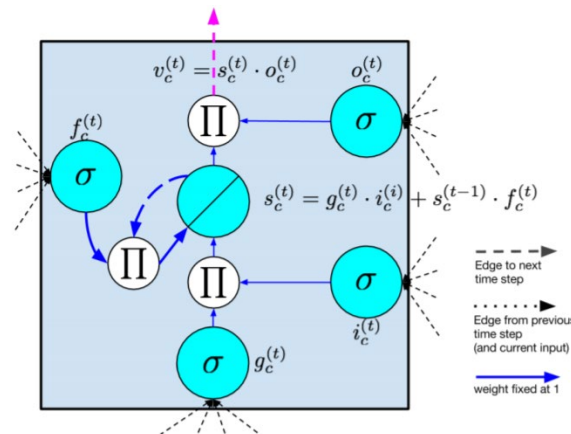
Likelihood evaluation: has to be done **sequentially** (no parallelization)

LSTM (Long Short-Term Memory)

The main issue with training RNN's is long-term dependencies (and correspondingly exploding/vanishing gradients)

The main idea of **LSTM's** (*Hochreiter and Schmidhuber '97*): are gating mechanisms that try to control the flow of information from past.

In practice, they seem to suffer much less from gradient vanishing/explosion. (No good theoretical understanding.)



LSTM (Long Short-Term Memory)

Ingredients:

Input node: $g^{(t)} = \phi(W^{gx}x^{(t)} + W^{gh}h^{(t-1)} + b_g)$

Input gate: $i^{(t)} = \sigma(W^{ix}x^{(t)} + W^{ih}h^{(t-1)} + b_i)$

Forget gate: $f^{(t)} = \sigma(W^{fx}x^{(t)} + W^{fh}h^{(t-1)} + b_f)$

Output gate: $o^{(t)} = \sigma(W^{ox}x^{(t)} + W^{oh}h^{(t-1)} + b_o)$

Internal state: $s^{(t)} = g^{(t)} \odot i^{(i)} + s^{(t-1)} \odot f^{(t)}$

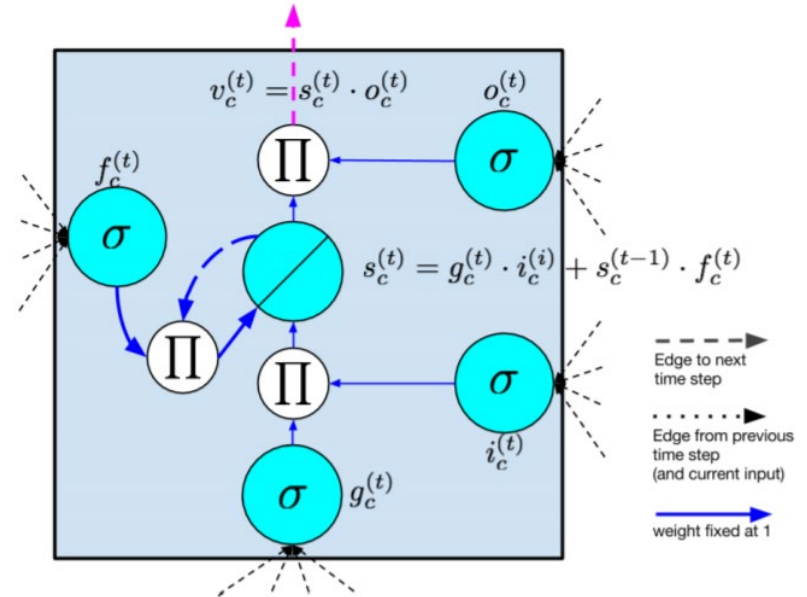
Hidden state: $h^{(t)} = \phi(s^{(t)}) \odot o^{(t)}$

Input node will be “gated” by pointwise multiplication with input gate: how input “flows”

Will gate the “internal state”

How output “flows”

Combine gated version of prev. state w/ forget gate, along with gated input node.



The parade of Sesame Street: ELMo

Recall our motivating example for the meaning of “bank”:

“Walking along the river bank” and “A bank accepts deposits”



The problem with using an RNN/LSTM is that in the first case, the context for bank is provided by “river” – which comes **before** it; in the latter by “deposits”, which comes **after** it.

A language model is “**unidirectional**” – the embedding for a position only depends on previous words.

How can we introduce bidirectionality?

The parade of Sesame Street: ELMo

Peters et al '18: **ELMo** (Embeddings from Language Models)

Train one (stacked) LSTM model to model $p_\theta(x_t|x_{<t})$ and another LSTM to model $p_\theta(x_t|x_{>t})$. Concatenate two representations. (For downstream tasks, linearly combine tasks with learned weights.)

ELMo represents a word t_k as a linear combination of corresponding hidden layers (inc. its embedding)

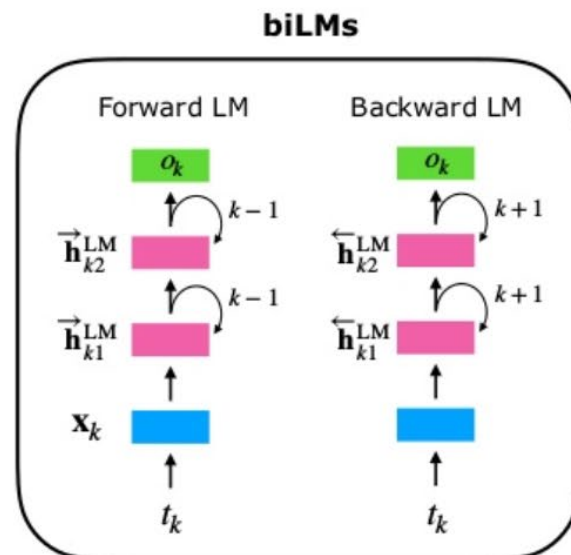
ELMo is a task specific representation. A down-stream task learns weighting parameters

$$\text{ELMo}_k^{\text{task}} = \gamma^{\text{task}} \times \sum \left\{ \begin{array}{l} s_2^{\text{task}} \times \mathbf{h}_{k2}^{\text{LM}} \\ s_1^{\text{task}} \times \mathbf{h}_{k1}^{\text{LM}} \\ s_0^{\text{task}} \times \mathbf{h}_{k0}^{\text{LM}} \end{array} \right. \quad \left([\mathbf{x}_k; \mathbf{x}_k] \right)$$

Concatenate hidden layers

$[\vec{\mathbf{h}}_{kj}^{\text{LM}}, \overleftarrow{\mathbf{h}}_{kj}^{\text{LM}}]$

Unlike usual word embeddings, ELMo is assigned to every *token* instead of a *type*



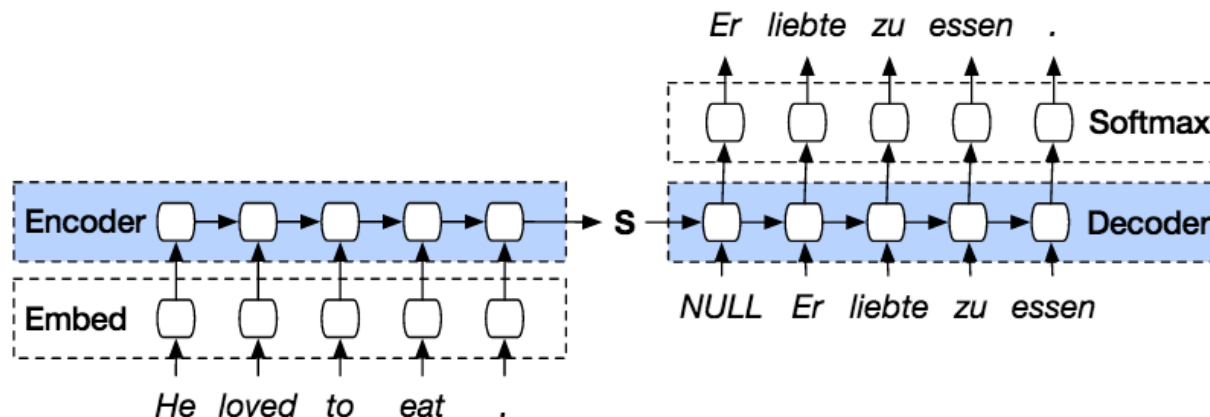
Brief diversion: machine translation

Despite handling gradients better than RNNs, LSTMs are still rather difficult to train, and struggle with long-term dependencies.

A new generation of models that displaced recurrent modern models altogether was initiated by a paper by *Vaswani et al '17*: “**Attention is all you need**”.

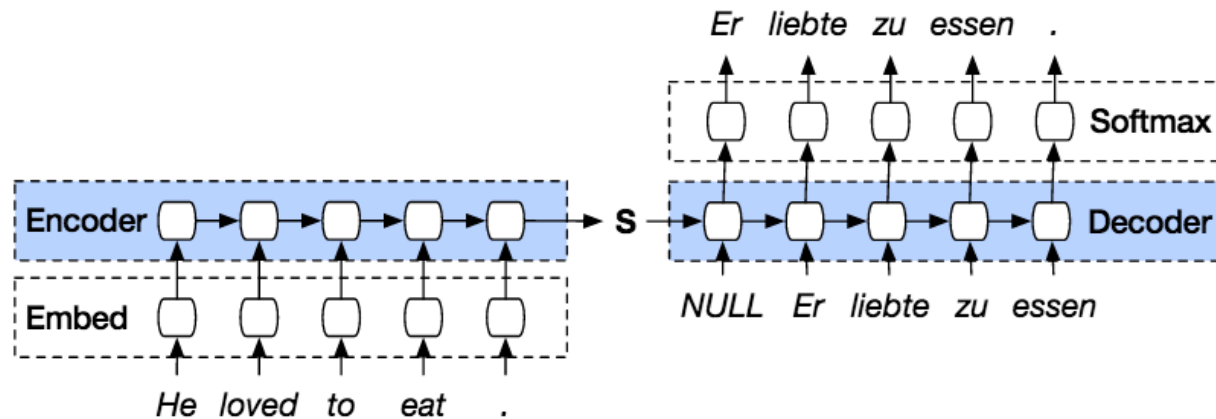
The most natural way to introduce the ideas is through **sequence to sequence** tasks, in which we try to map sequence in one domain to another.

Most natural instantiation: machine translation.



The encoder/decoder framework

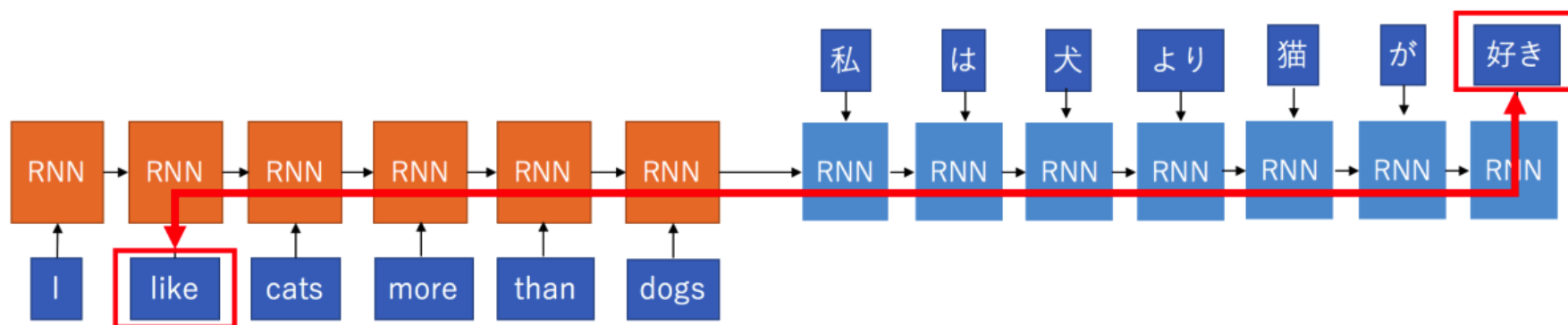
Most natural instantiation: machine translation.



Typical high-level architecture: an **encoder** transforms input to an intermediate representation (i.e. *lingua franca*), and a **decoder** transforms intermediate representation to target sequence.

RNN-based encoder/decoders

Earliest incarnation: Sutskever et al. '14 – just use RNNs.



As expected, long sentences are a problem. Training is also inherently sequential due to the nature of a recurrent network.

Attention

Attention: when translating a certain word in a sentence, certain words matter more than others. If we can determine which ones matter, that could replace recurrent network.

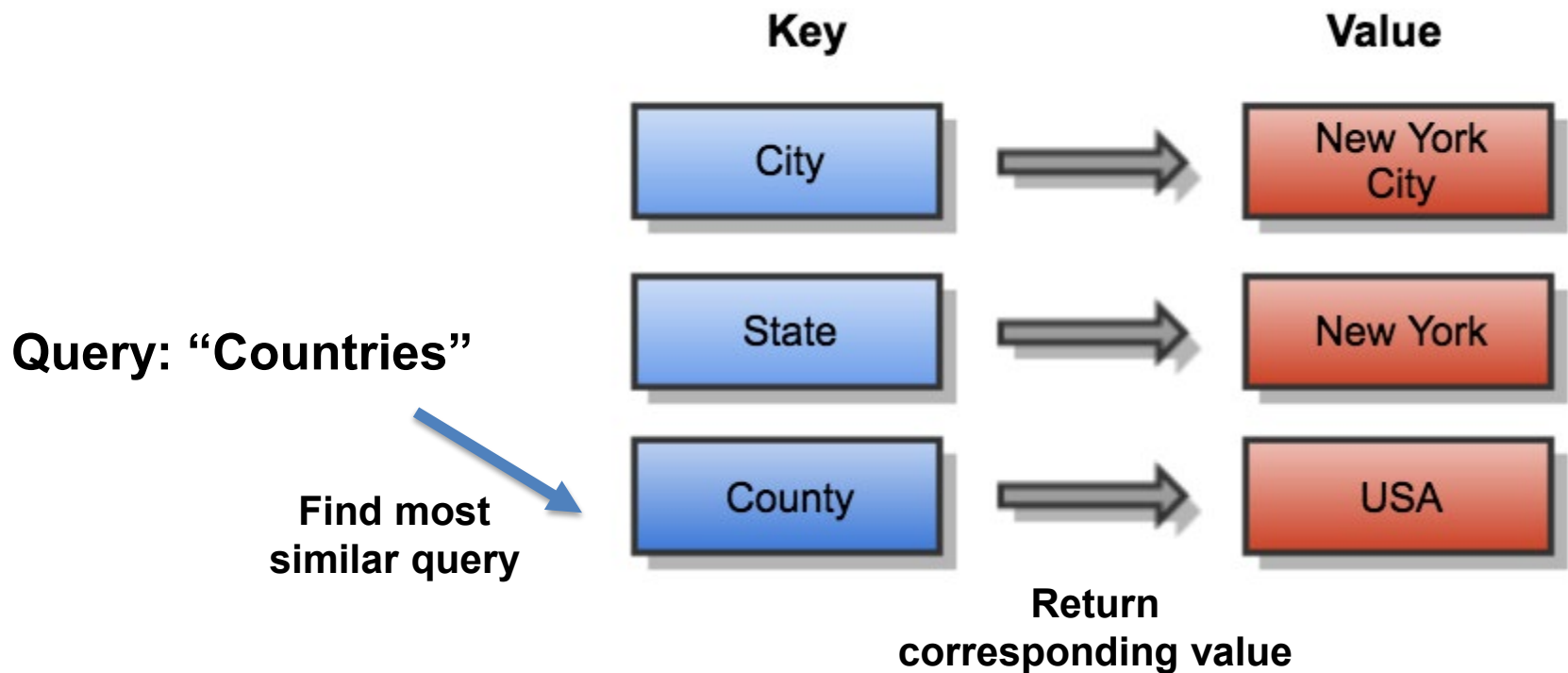
The animal didn't cross the street because it was too tired.
L'animal n'a pas traversé la rue parce qu'il était trop fatigué.

The animal didn't cross the street because it was too wide.
L'animal n'a pas traversé la rue parce qu'elle était trop large.

E.g. “it” refers to “animal” in first sentence, and “street” in second. These have different genders in French (“il” and “elle”), so system needs to understand which word to “attend” to when translating “it”.

Attention

Intuition for attention comes from databases: a key operation is given a **query**, find the relevant **key**, and lookup the corresponding **value**.



Attention

A more “differentiable” variant of this:

$$\text{Attention}(q, K, V) = \sum_i \text{similarity}(q, K_i) V_i$$

Vector

Matrices w/ rows
keys/values

Linear combination of
“most similar” values

The simplest notion of similarity: inner product.

$$\text{Attention}(q, K, V) = \sum_i \text{softmax}(\langle q, K_i \rangle) V_i$$

Produces distribution
over keys

Produces distribution
over values

Multiple queries combined in matrix Q:

$$\text{Attention}(Q, K, V) = \text{softmax}(QK^T)V$$

Transformers: using attention

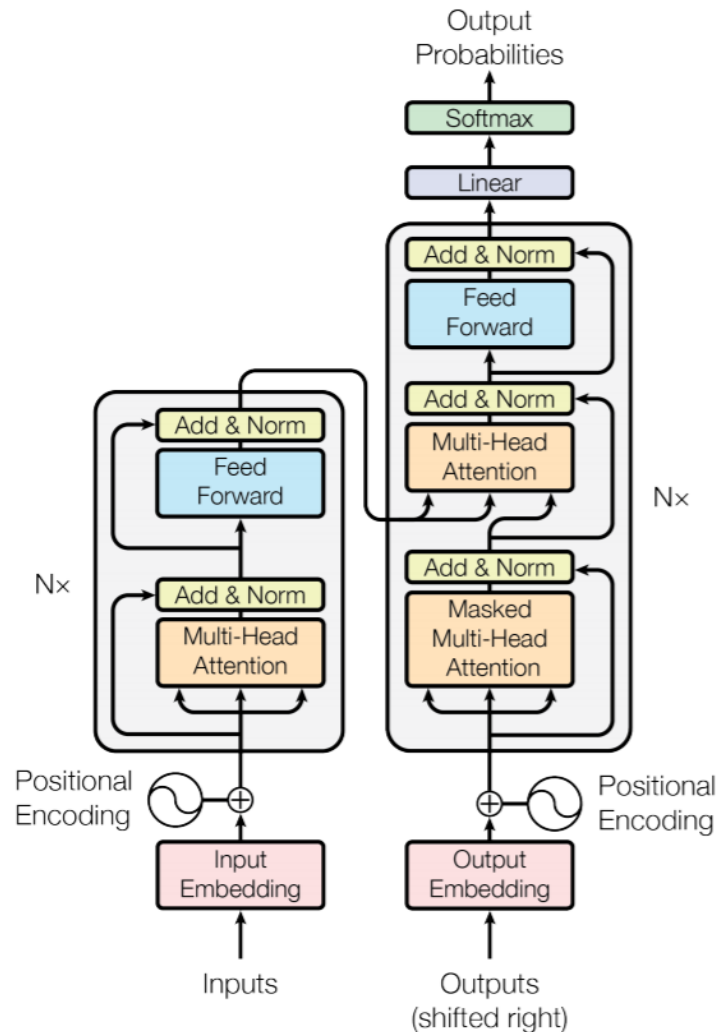


Figure from Vaswani et al '17: "Attention is all you need".

Transformers: encoder

Part 1: Multi-head self-attention

Self-attention: $\text{Attention}(Q, K, V)$ where:

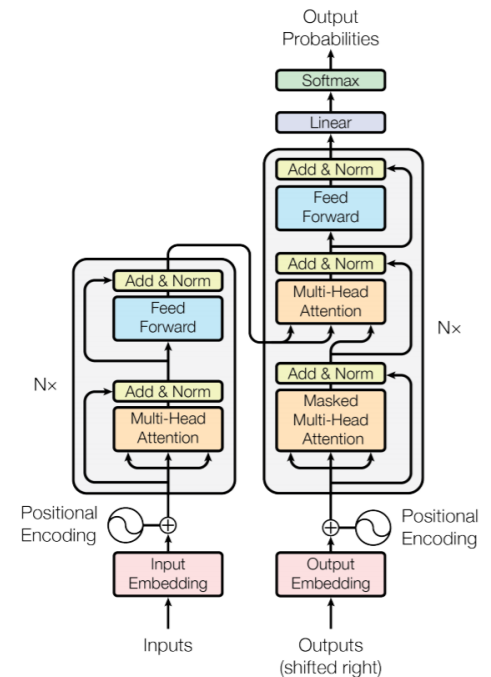
The queries/keys/values Q_i, K_i, V_i all correspond to vectors for each word i in input sequence.

In other words, decide what the relevant input context is for every input word.

Multi-head: instead of a single attention mechanism, (linearly) project Q, K, V multiple times, and learn self-attention w/ projected vectors.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$
$$\text{where } \text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

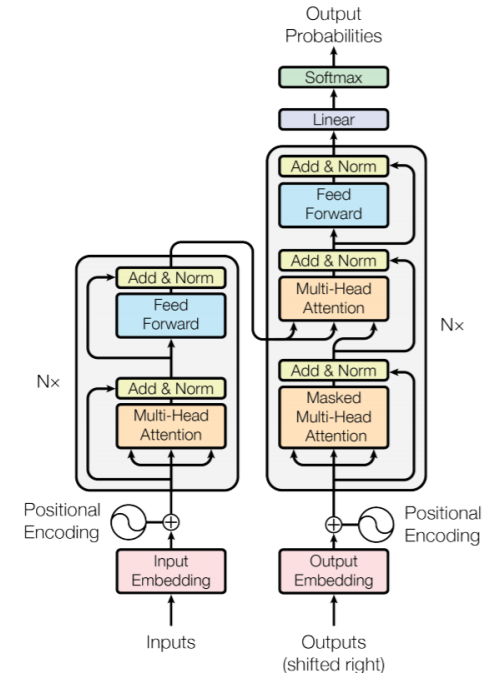


Transformers: encoder

Part 2: Positional encoding

We'd like the embedding fed into the stack of attention mechanisms to have some dependence on position. The authors choice: combine position-independent embedding (additively) with (*quite weird choice!*)

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



Part 3: Non-linearities

The output of each attention block is fed through a non-linearity, represented by a feed-forward network.

The “Add & norm” layers renormalize layers to whiten mean/covariance (similar as batch norm, except per layer rather than per node).

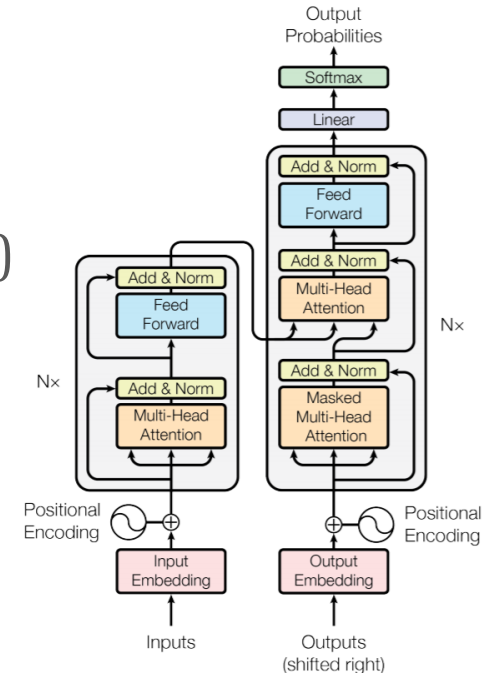
Transformers: decoder

Part 1: Attention b/w encoder and decoder

Queries: outputs of encoder (one per word in sequence)

Values/keys: output sequence (one per word in sequence)

Learn which parts of input to attend to
when building translations for parts of output.

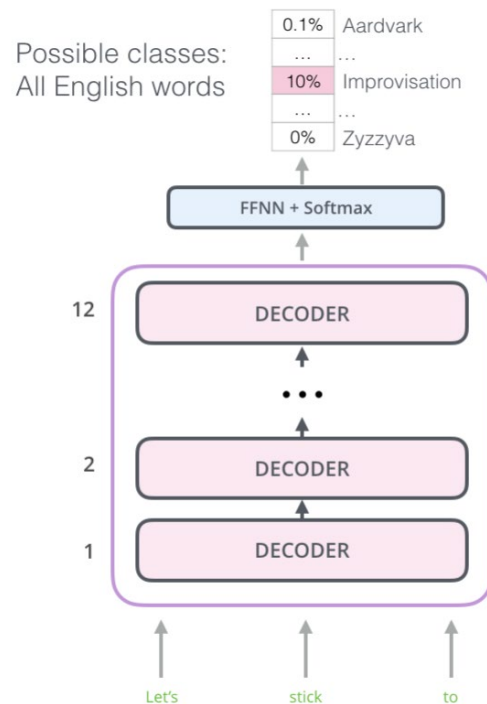


Remaining parts: basically the same as encoder.

Only important difference: multi-head self-attention is *masked* to only allow attending to prior symbols. (This is because the model can't be allowed to look ahead to predict next word in loss.)

Using transformers for feature learning: the obvious way

The obvious way to use this architecture to simply extract features: use a decoder only!



GPT2: Transformer-based architecture due to OpenAI

Figure from <http://jalammar.github.io/illustrated-bert/>

The less obvious way: BERT

Something is still amiss: the decoder from transformers still predicts the next word. *Still unidirectional!*

The insight of Devlin et al. '18: BERT (Bidirectional Encoder Representations from Transformers) – use an **encoder** instead!

But how? What are trying to predict?

Predict random 15% of the words, given the rest

The parade of Sesame Street: BERT

Masked LM task: Predict random 15% of the words, given the rest

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

Possible classes:
All English words

0.1%	Aardvark
...	...
10%	Improvisation
...	...
0%	Zyzyva

FFNN + Softmax

Randomly mask
15% of tokens

Input

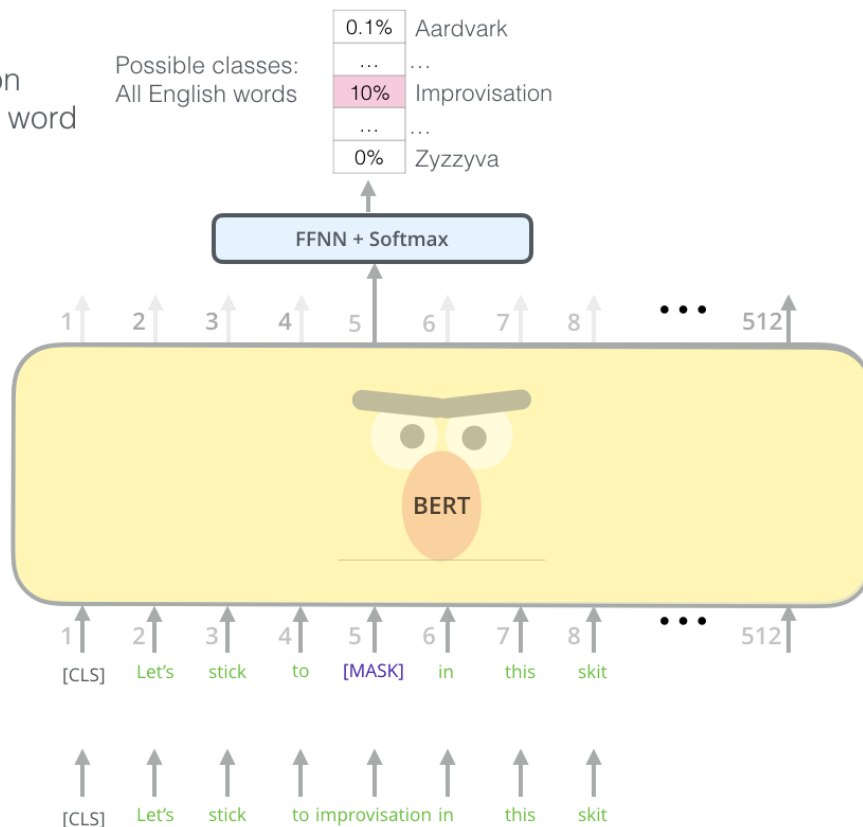


Figure from <http://jalamar.github.io/illustrated-bert/>

The parade of Sesame Street: BERT

Slight mismatch with downstream tasks: these will never have a token [MASK]. Can this be a bit ameliorated?

Yes, with some probability change word to be masked out to a random word, or leave as is.



Performance

At publication time, improved state of the art, by solid 5% on GLUE tasks.
Today, variants of BERT are pretty much defacto pretraining method.

The General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2018) is a collection of diverse natural language understanding tasks.

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	Average
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
BERT _{BASE}	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
BERT _{LARGE}	86.7/85.9	72.1	91.1	94.9	60.5	86.5	89.3	70.1	81.9

Performance

SQuAD 2.0 Question Answering leaderboard 2019-10-09

Passage

Super Bowl 50 was an American football game to determine the champion of the National Football League (NFL) for the 2015 season. The American Football Conference (AFC) champion Denver Broncos defeated the National Football Conference (NFC) champion Carolina Panthers 24–10 to earn their third Super Bowl title. The game was played on February 7, 2016, at Levi's Stadium in the San Francisco Bay Area at Santa Clara, California.

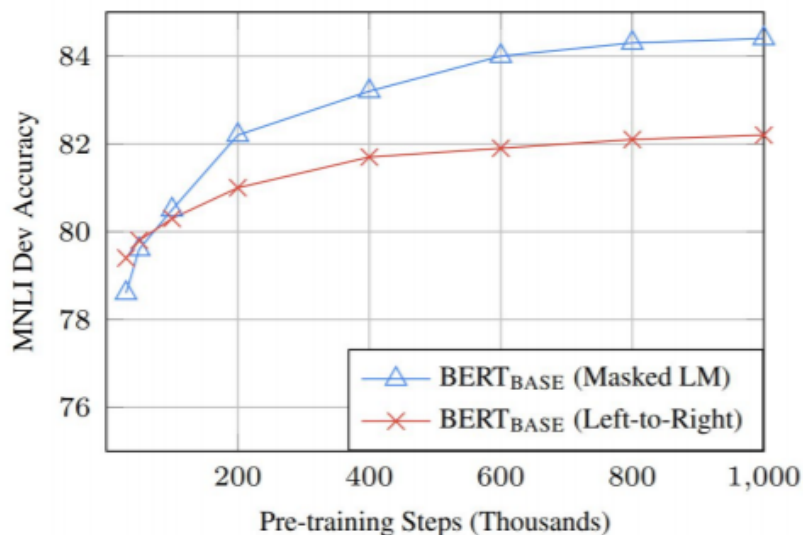
Question: Which team won Super Bowl 50?

26

Rank	Model	EM	F1
	Human Performance Stanford University (Rajpurkar & Jia et al. '18)	86.831	89.452
1 Sep 18, 2019	ALBERT (ensemble model) Google Research & TTIC https://arxiv.org/abs/1909.11942	89.731	92.215
2 Jul 22, 2019	XLNet + DAAF + Verifier (ensemble) PINGAN Omni-Sinitic	88.592	90.859
2 Sep 16, 2019	ALBERT (single model) Google Research & TTIC https://arxiv.org/abs/1909.11942	88.107	90.902
2 Jul 26, 2019	UPM (ensemble) Anonymous	88.231	90.713
3 Aug 04, 2019	XLNet + SG-Net Verifier (ensemble) Shanghai Jiao Tong University & CloudWalk https://arxiv.org/abs/1908.05147	88.174	90.702
4 Aug 04, 2019	XLNet + SG-Net Verifier++ (single model) Shanghai Jiao Tong University & CloudWalk https://arxiv.org/abs/1908.05147	87.238	90.071
5 Jul 26, 2019	UPM (single model) Anonymous	87.193	89.934
6 Mar 20, 2019	BERT + DAE + AoA (ensemble) Joint Laboratory of HIT and iFLYTEK Research	87.147	89.474
6 Jul 20, 2019	RoBERTa (single model) Facebook AI	86.820	89.795

Slide by Chris Manning.

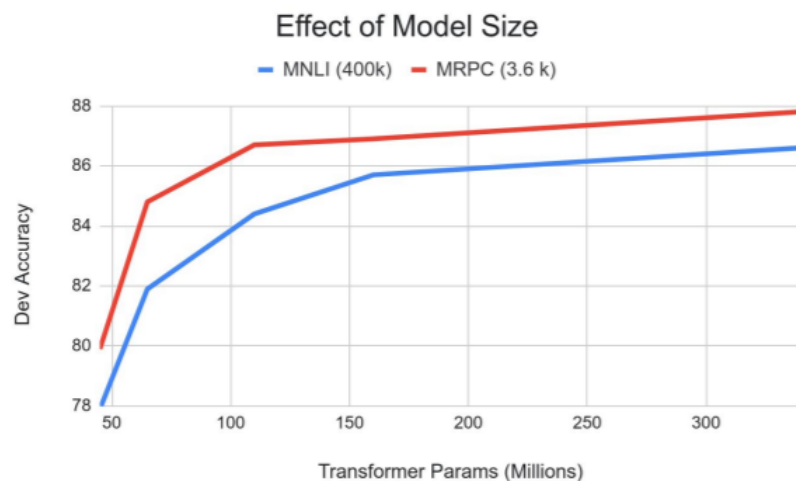
The value of bidirectionality



- Masked LM takes slightly longer to converge because we only predict 15% instead of 100%
- But absolute results are much better almost immediately

Slide from Jacob Devlin

Big models matter, a lot



- Big models help *a lot*
- Going from 110M -> 340M params helps even on datasets with 3,600 labeled examples
- Improvements have *not* asymptoted

Slide from Jacob Devlin

Big models matter, a lot

Transformer models

ELMo	GPT	BERT	GPT-2	XL-Net, ERNIE, Grover
Oct 2017	June 2018	Oct 2018	Feb 2019	RoBERTa, ...
Training:	Training	Training	Training	
800M words	800M words	3.3B words	40B words	
42 GPU days	240 GPU days	256 TPU days	~2048 TPU v3 days	July 2019

according to [a reddit thread](#)



Why did this not happen earlier?

- Why did no one think of this before?
- Better question: Why wasn't contextual pre-training popular before 2018 with ELMo?
- Good results on pre-training is $>1,000\times$ to 100,000 more expensive than supervised training.
 - E.g., $10\times$ - $100\times$ bigger model trained for $100\times$ - $1,000\times$ as many steps.
 - Imagine it's 2013: Well-tuned 2-layer, 512-dim LSTM sentiment analysis gets 80% accuracy, training for 8 hours.
 - Pre-train LM on same architecture for a week, get 80.5%.
 - Conference reviewers: "Who would do something so expensive for such a small gain?"

Slide from Jacob Devlin

Are we done with modeling in NLP?

Good systems are great, but still basic NLU errors

The Yuan dynasty is considered both a successor to the Mongol Empire and an imperial Chinese dynasty. It was the khanate ruled by the successors of Möngke Khan after the division of the Mongol Empire. In official Chinese histories, the Yuan dynasty bore the Mandate of Heaven, following the Song dynasty and preceding the Ming dynasty. The dynasty was established by Kublai Khan, yet he placed his grandfather Genghis Khan on the imperial records as the official founder of the

What dynasty came before the Yuan?

Gold Answers: ① Song dynasty ② Mongol Empire ③ the Song dynasty

Prediction: Ming dynasty [BERT (single model) (Google AI)]

Slide from Chris Manning

Are we done with modeling in NLP?

Two problems

BERT and friends are **awesome** as a universal pre-training base for natural language understanding tasks

Nevertheless, we still have work to do:

1. We've built powerful, **neural matching machines**, rather than devices that can **think**
2. We've built devices for one-step classification, QA etc., rather than devices that can **reason** through a series of steps

Are we done with modeling in NLP?

Answering Complex Open-Domain Questions: Single-step vs. Multi-step Question Answering

[Qi, Lin, Mehr, Wang, and Manning EMNLP-IJCNLP 2019]

Context: Aquaman is a **2018** American superhero film based on the DC Comics character of the same name. ...

Question: When was the movie Aquaman released?

Answer: **2018** ✓

Question: What's the Aquaman actor's next movie?

Answer: **Jason Momoa** ✗

Slide from Chris Manning

Are we done with modeling in NLP?

Answering Complex Open-Domain Questions: Single-step vs. Multi-step Question Answering

[Qi, Lin, Mehr, Wang, and Manning EMNLP-IJCNLP 2019]

Context: Aquaman is a **2018** American superhero film based on the DC Comics character of the same name. ...

Question: When was the movie Aquaman released?

Answer: **2018** ✓

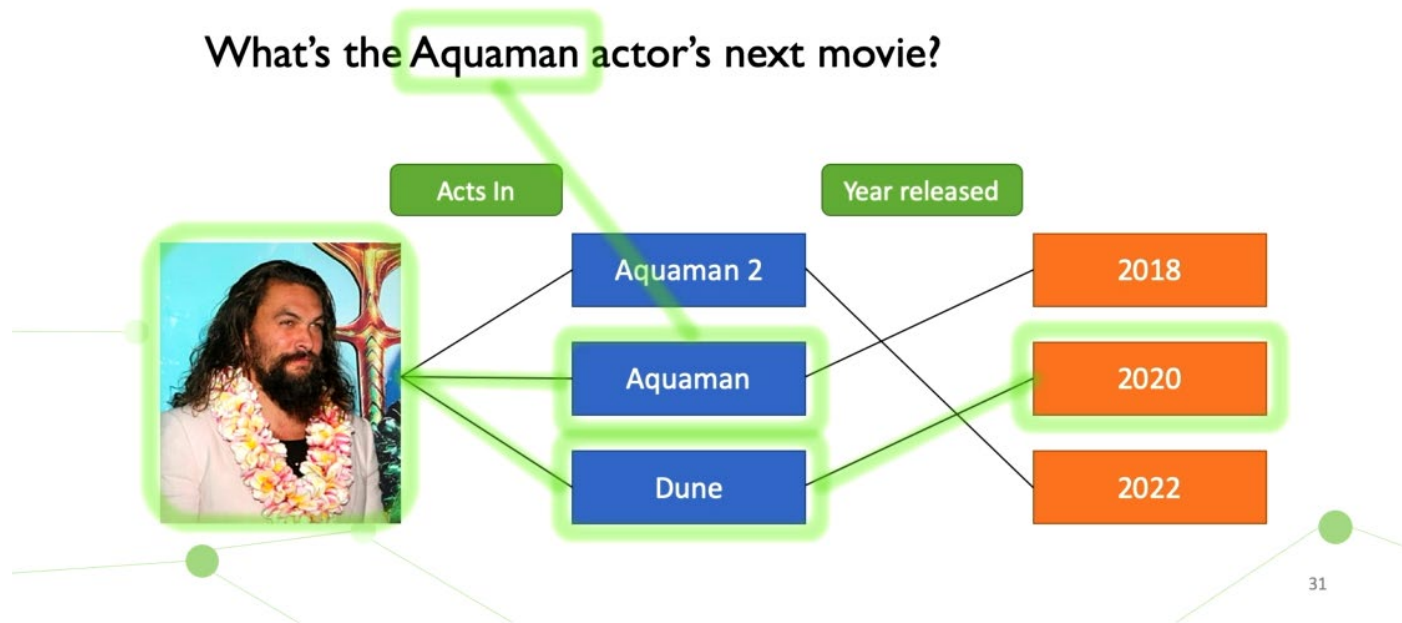
Question: What's the Aquaman actor's next movie?

Answer: **Jason Momoa** ✗

Slide from Chris Manning

More complex datasets

Multi-step Question Answering



Slide from Chris Manning

More complex datasets



1) What is the **giraffe** looking at?
person ✓

2) Is the **fence** in front of the **giraffe** made of metal? **no** ✓

3) Is the **woman's shirt** blue or yellow? **blue** ✓

4) On which side of the image is the **person**? **right** ✓

5) Is there a **child** behind the **giraffe**? **no** ✗

1) What is the **fruit** to the right of the **salad**? **strawberries** ✓

2) Is the **fork** to the right of the **salad**? **no** ✓

3) Is the **plate** white and square? **no** ✓

4) Is the **cup** behind the round **plate**? **yes** ✓

5) What is the **plate** made of? **paper** ✗

1) Are there either **scarves** or **hats** that are not pink? **no** ✓

2) Do the **bear's dress** and the **person's shirt** have the same color? **yes** ✓

3) Is the **bear** sitting or standing? **sitting** ✓

4) What is the green **object** that the **bear** is sitting on? **book** ✓

5) Is the **bear** wearing white **shoes**? **yes** ✗

1) Are there either a **chair** or a **clock** in the image? **no** ✓

2) Are there any **flowers** behind the **bed** on the left of the **room**? **yes** ✓

3) What color is the **appliance** on the right? **black** ✓

4) Is the **carpet** brown or blue? **brown** ✓

5) Is the **TV** turned on? **yes** ✗

Figure 2: Question examples along with answers predicted by the NSM. The questions involve diverse reasoning skills such as multi-step inference, relational and spatial reasoning, logic and comparisons.

Figure from Hudson & Manning '19,
VQA (visual question-answering) datasets

More complex models

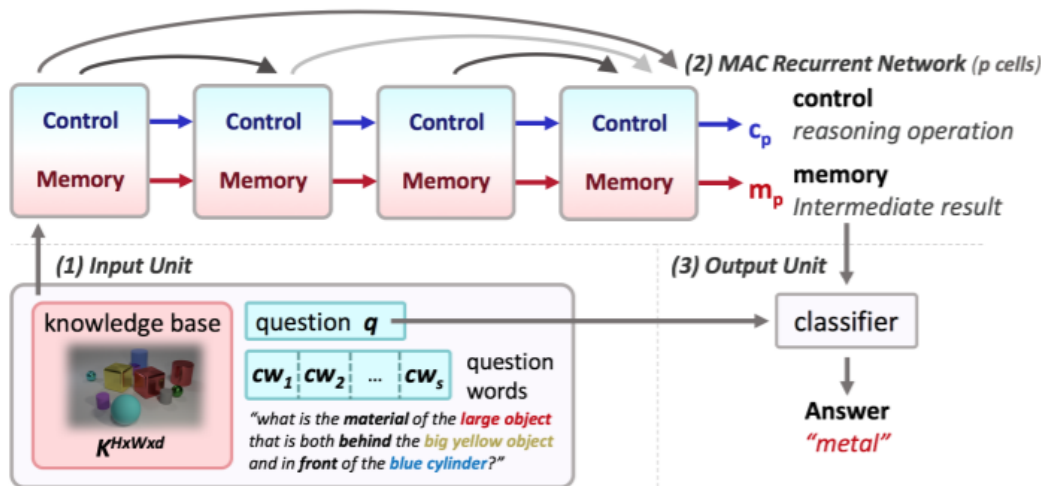


Figure 2: Model Overview. The MAC network consists of an input unit, a core recurrent network and an output unit. (1) The input unit transforms the raw image and question into distributed vector representations. (2) The core recurrent network reasons sequentially over the question by decomposing it into a series of operations (*control*) that retrieve information from the image (knowledge base) and aggregate the results into a recurrent *memory*. (3) The output classifier computes the final answer using the question and the final memory state.

More complex models

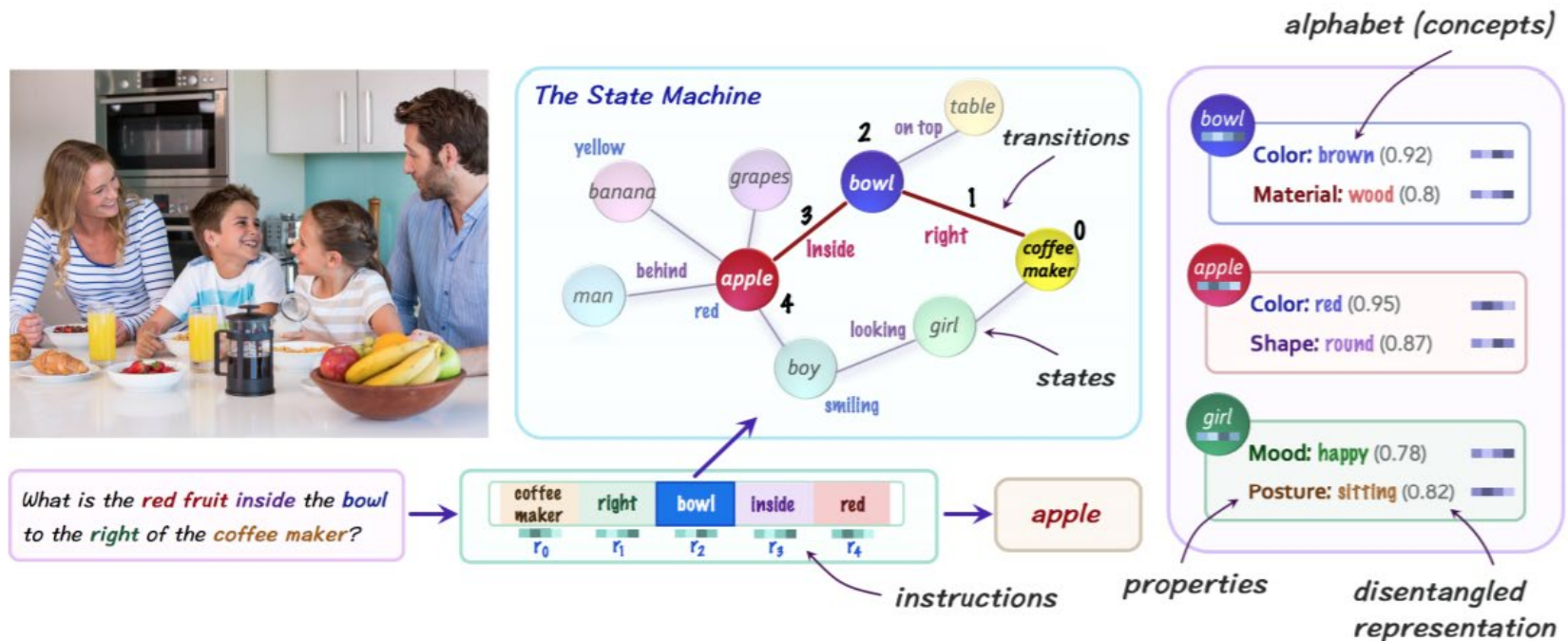


Figure 1: The Neural State Machine is a graph network that simulates the computation of an automaton. For the task of VQA, the model constructs a probabilistic scene graph to capture the semantics of a given image, which it then treats as a state machine, traversing its states as guided by the question to perform sequential reasoning.

Figure from Hudson & Manning '19