# Natural Language Processing with Deep Learning

# CS224N/Ling284


Christopher Manning

Lecture 10:
(Textual) Question Answering Architectures, Attention and Transformers

# Mid-quarter feedback survey

Thanks to the many of you (!) who have filled it in!

If you haven’t yet, today is a good time to do it 😉

# Lecture Plan

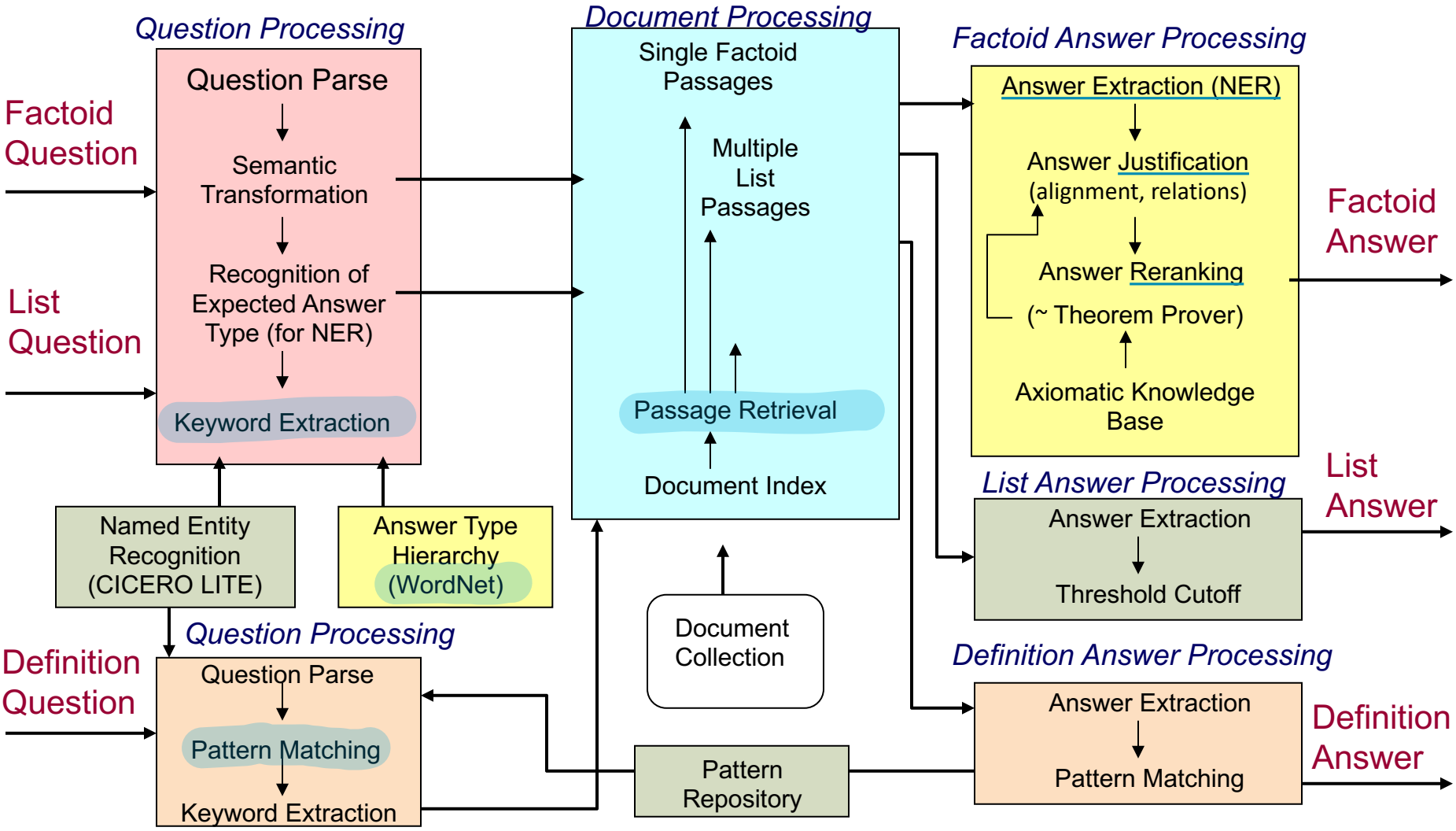
Lecture 10: (Textual) Question Answering

1. History/The SQuAD dataset (review)
2. The Stanford Attentive Reader model
3. BiDAF
4. Recent, more advanced architectures
5. Open-domain Question Answering: DrQA
6. Attention revisited; motivating transformers; ELMo and BERT preview
7. Training/dev/test data
8. Getting your neural network to train

# 1. Turn-of-the Millennium Full NLP QA:

## [architecture of LCC (Harabagiu/Moldovan) QA system, circa 2003]

## Complex systems but they did work fairly well on "factoid" questions



# Stanford Question Answering Dataset (SQuAD)

(Rajpurkar et al., 2016)

**Question:** Which team won Super Bowl 50?

**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.

100k examples

Answer must be a span in the passage

Extractive question answering/reading comprehension

# SQuAD 2.0 No Answer Example

Genghis Khan united the Mongol and Turkic tribes of the steppes and became Great Khan in 1206. He and his successors expanded the Mongol empire across Asia. Under the reign of Genghis' third son, Ögedei Khan, the Mongols destroyed the weakened Jin dynasty in 1234, conquering most of northern China. Ögedei offered his nephew Kublai a position in Xingzhou, Hebei. Kublai was unable to read Chinese but had several Han Chinese teachers attached to him since his early years by his mother Sorghaghtani. He sought the counsel of Chinese Buddhist and Confucian advisers. Möngke Khan succeeded Ögedei's son, Güyük, as Great Khan in 1251. He

**When did Genghis Khan kill Great Khan?**

*Gold Answers:* <No Answer>

*Prediction:* 1234 [from Microsoft nlnet]

# 2. Stanford Attentive Reader

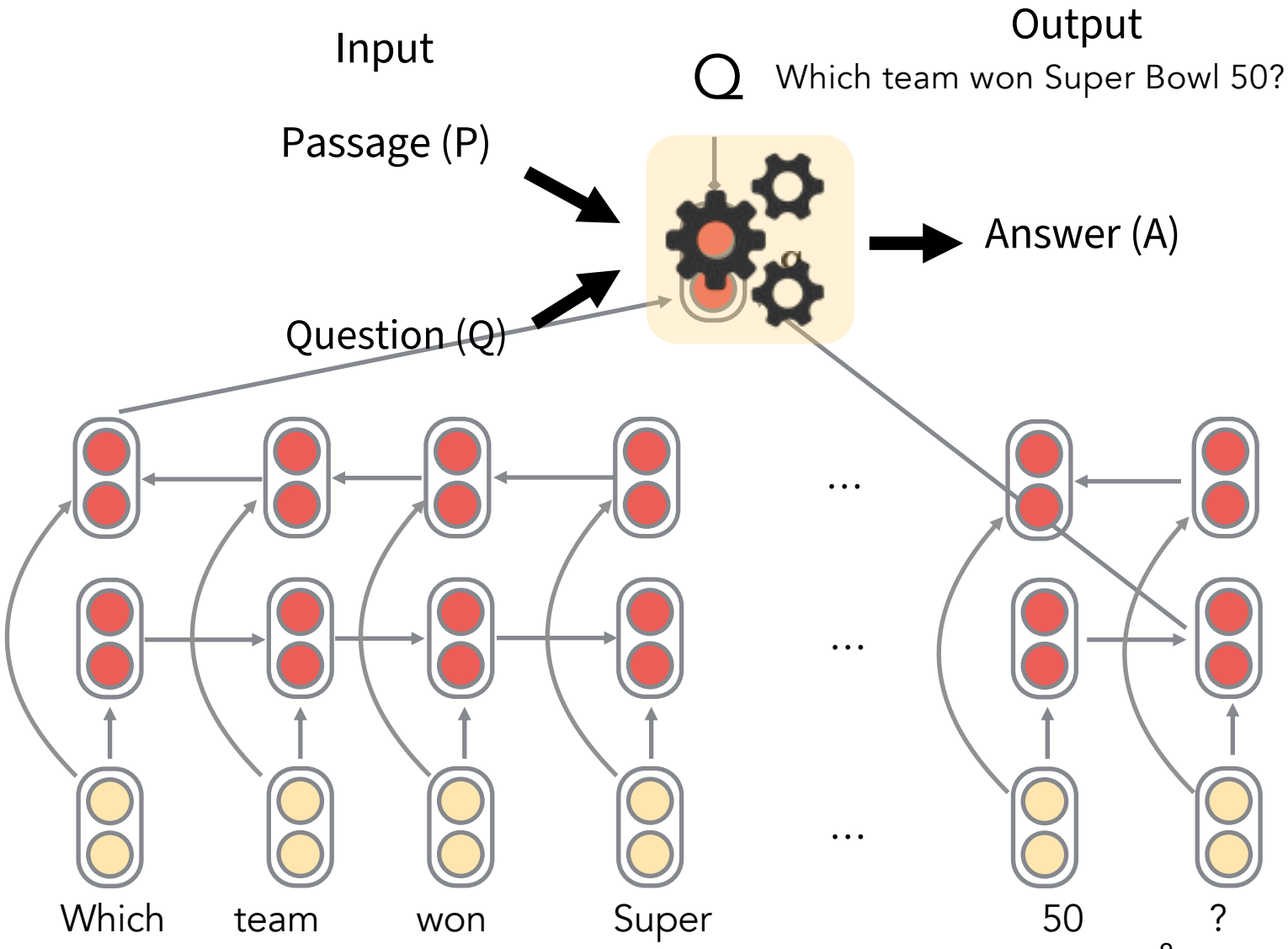
[Chen, Bolton, & Manning 2016]

[Chen, Fisch, Weston & Bordes 2017] DrQA

[Chen 2018]

- Demonstrated a minimal, highly successful architecture for reading comprehension and question answering
- Became known as the Stanford Attentive Reader

# The Stanford Attentive Reader



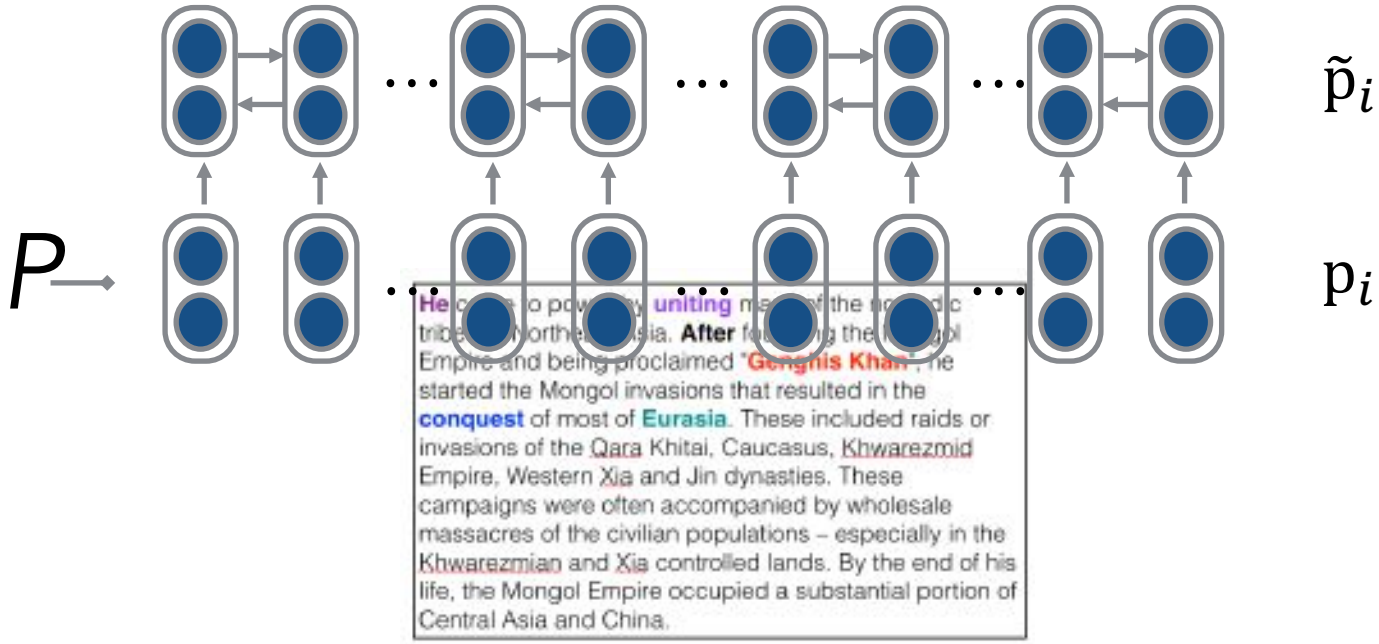
# Stanford Attentive Reader

Q

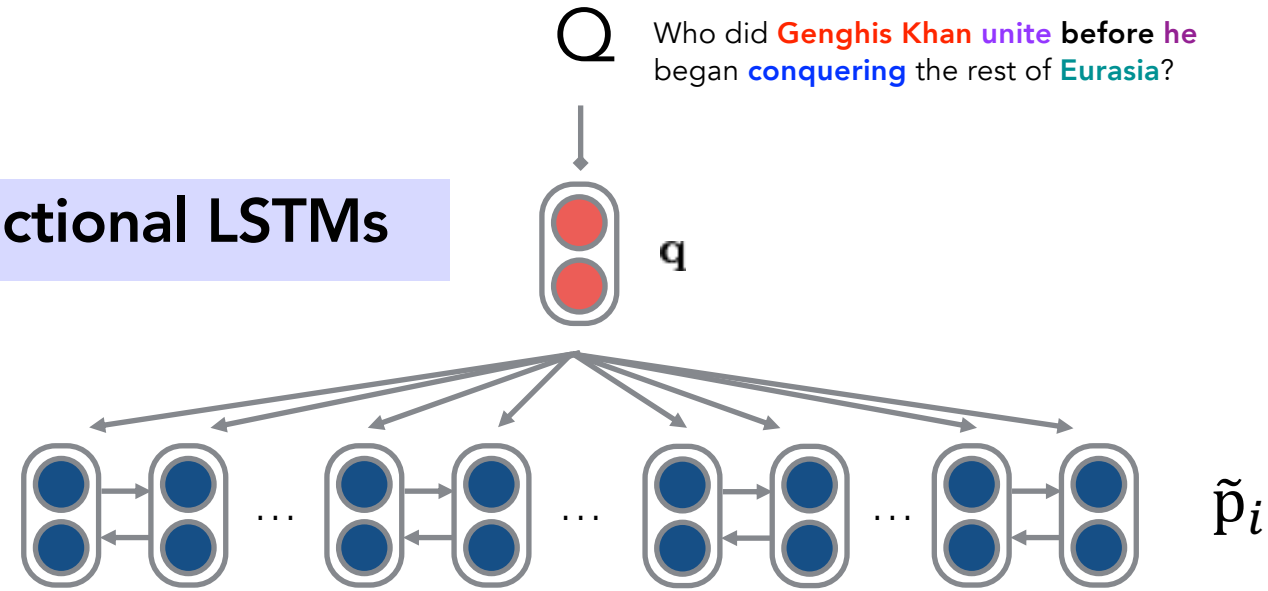
Who did **Genghis Khan unite** **before** **he** began **conquering** the rest of **Eurasia**?

**Bidirectional LSTMs**

$\mathbf{q}$



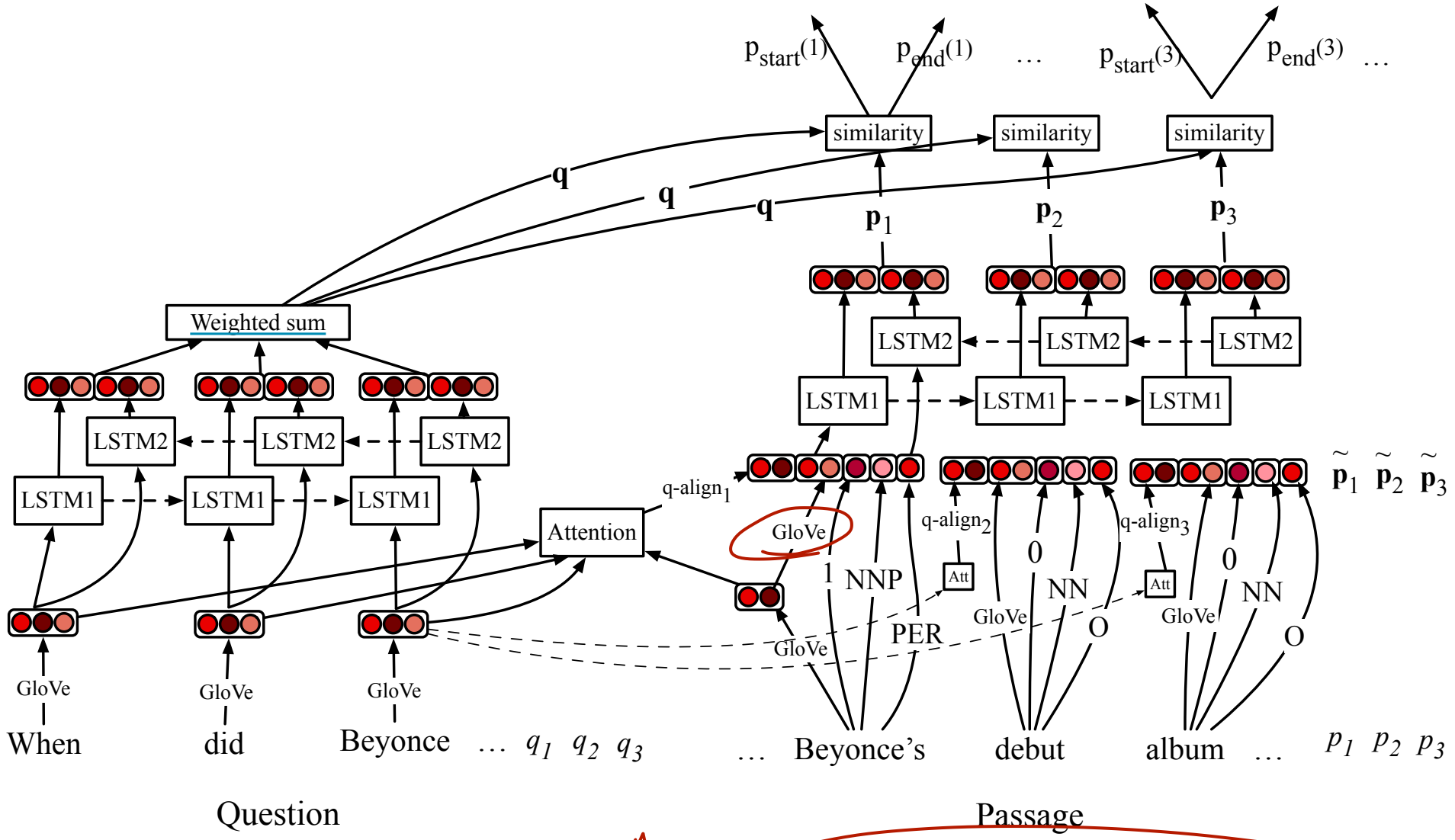
# Stanford Attentive Reader



# SQuAD 1.1 Results (single model, c. Feb 2017)

| | F1 |
|---|---|
| Logistic regression | 51.0 |
| Fine-Grained Gating (Carnegie Mellon U) | 73.3 |
| Match-LSTM (Singapore Management U) | 73.7 |
| DCN (Salesforce) | 75.9 |
| BiDAF (UW & Allen Institute) | 77.3 |
| Multi-Perspective Matching (IBM) | 78.7 |
| ReasoNet (MSR Redmond) | 79.4 |
| DrQA (Chen et al. 2017) | 79.4 |
| r-net (MSR Asia) [Wang et al., ACL 2017] | 79.7 |
| Human performance | 91.2 |

# Stanford Attentive Reader++




Training objective:

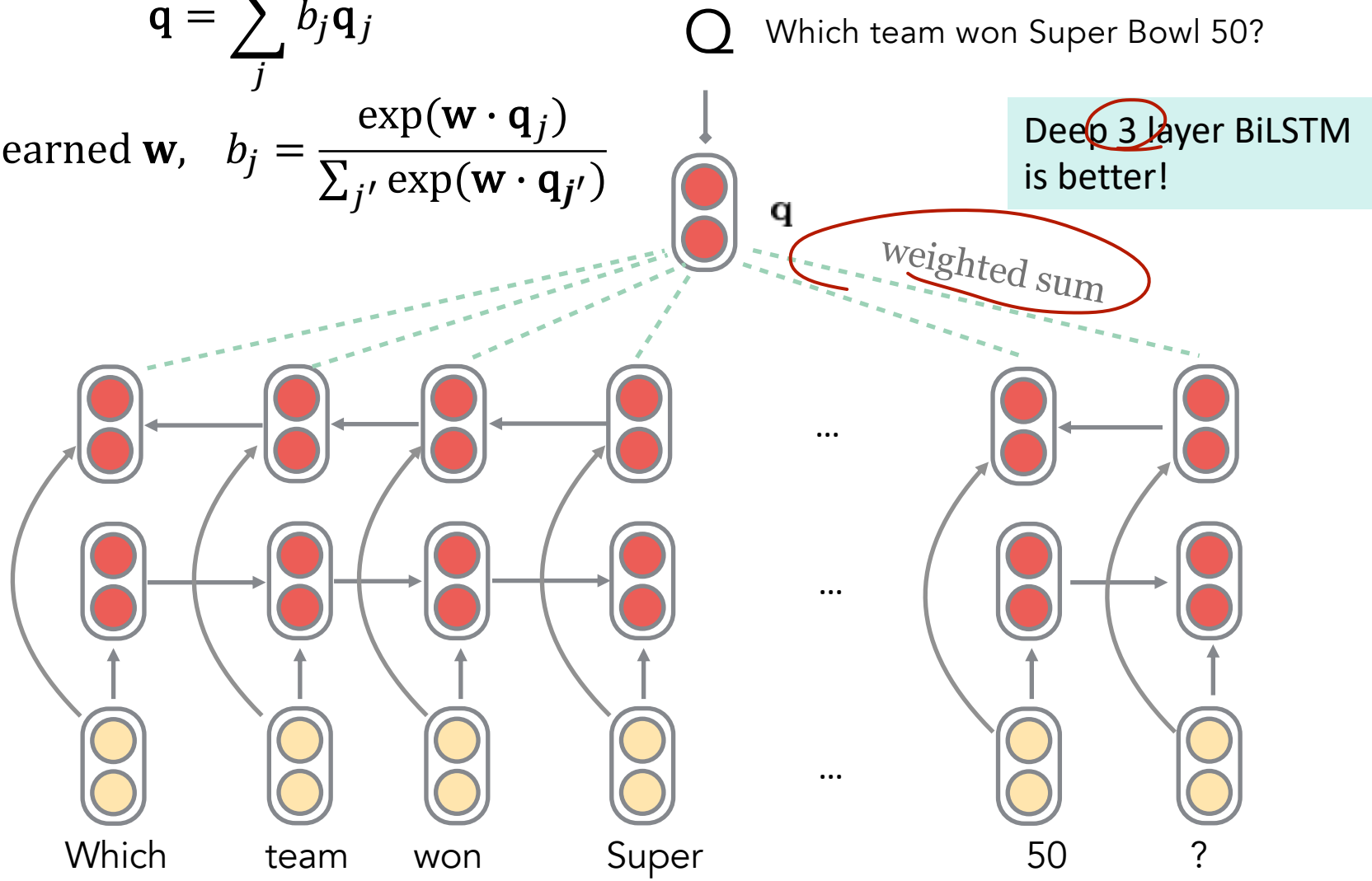
$$\mathcal{L} = -\sum \log P^{(\text{start})}(a_{\text{start}}) - \sum \log P^{(\text{end})}(a_{\text{end}})$$

(Chen et al., 2018)

# Stanford Attentive Reader++

$$\mathbf{q} = \sum_j b_j \mathbf{q}_j$$

For learned $\mathbf{w}$, $b_j = \frac{\exp(\mathbf{w} \cdot \mathbf{q}_j)}{\sum_{j'} \exp(\mathbf{w} \cdot \mathbf{q}_{j'})}$



# Stanford Attentive Reader++

- $\mathbf{p}_i$: Vector representation of each token in passage

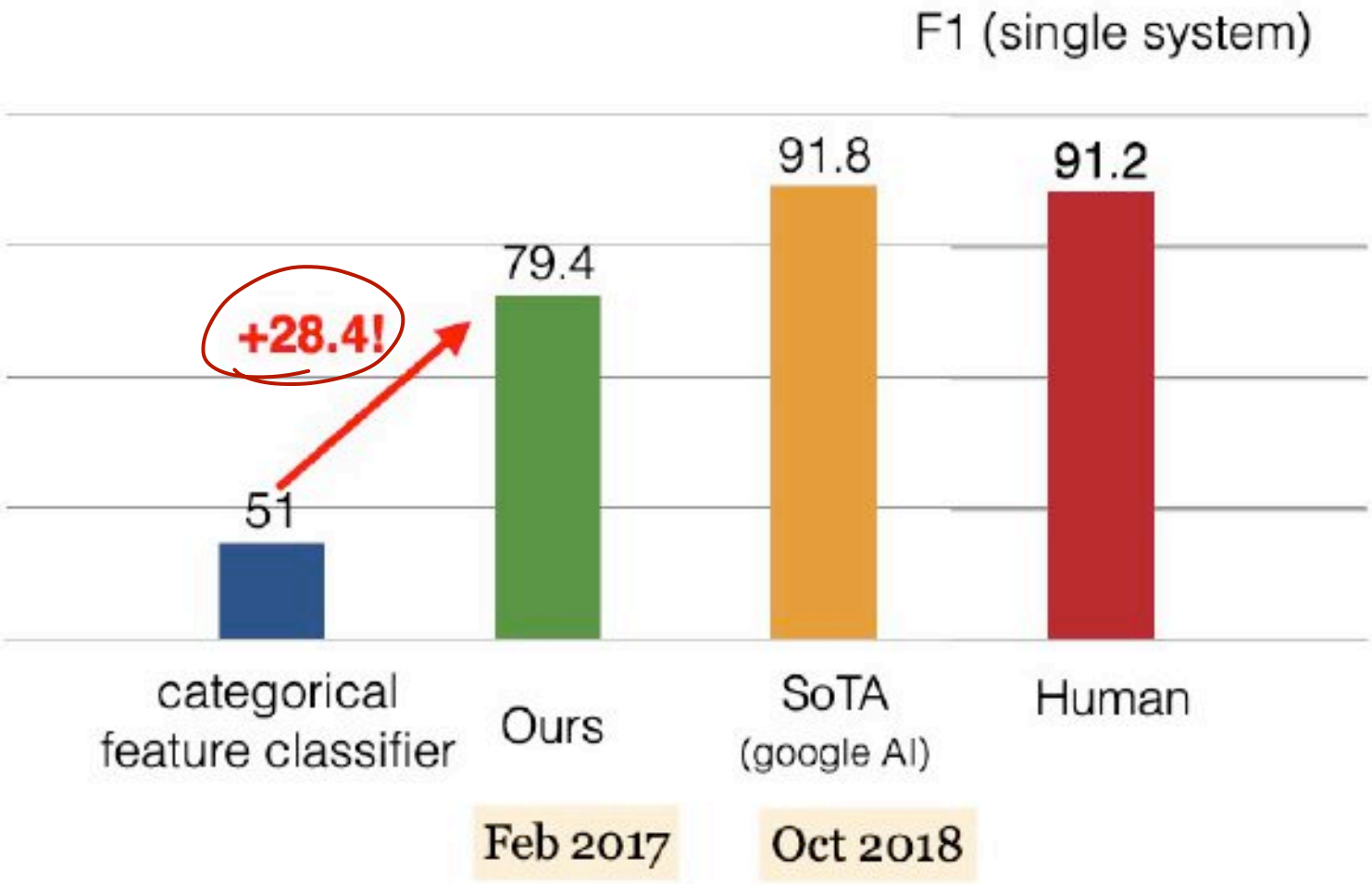
Made from concatenation of

- Word embedding (GloVe 300d)
- Linguistic features: POS & NER tags, one-hot encoded
- Term frequency (unigram probability)
- Exact match: whether the word appears in the question
  - 3 binary features: exact, uncased, lemma
- Aligned question embedding ("car" vs "vehicle")

$$f_{align}(p_i) = \sum_j a_{i,j}\mathbf{E}(q_j) \qquad q_{i,j} = \frac{\exp(\alpha(\mathbf{E}(p_i)) \cdot \alpha(\mathbf{E}(q_j)))}{\sum_{j'} \exp(\alpha(\mathbf{E}(p_i)) \cdot \alpha(\mathbf{E}(q'_j)))}$$

Where $\alpha$ is a simple one layer FFNN

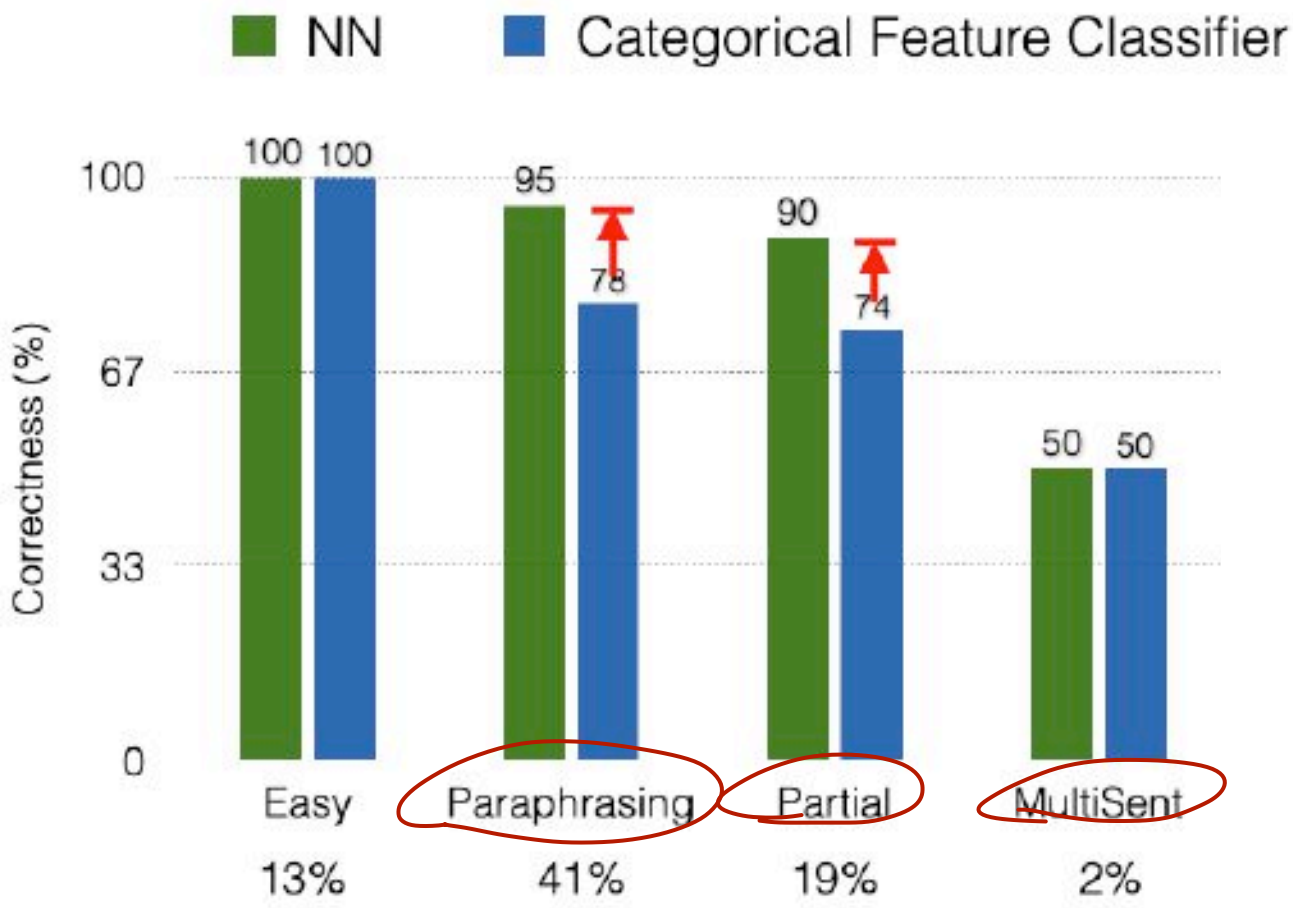
# A big win for neural models




x

(Chen, Bolton, Manning, 2016)

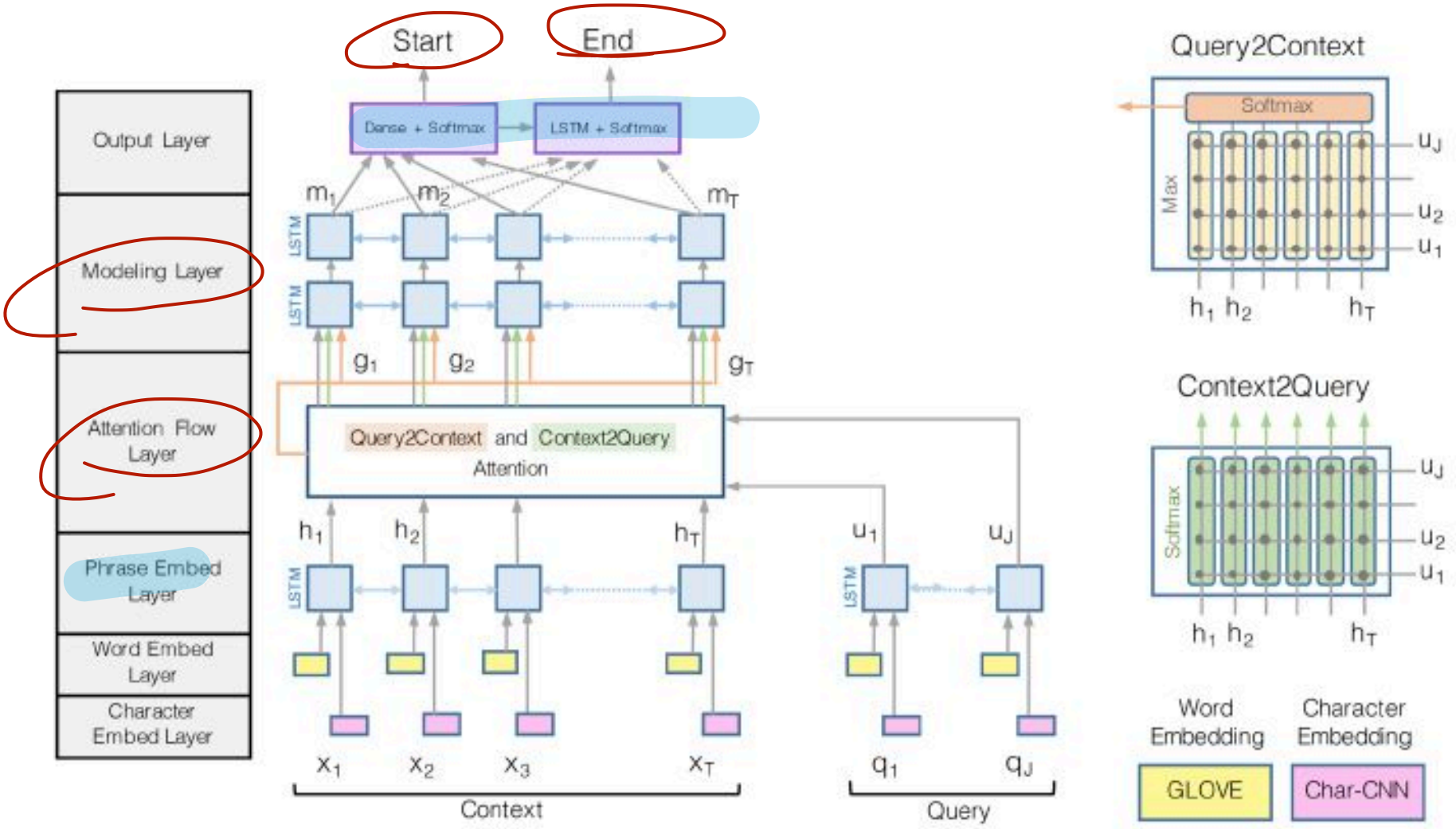
# What do these neural models do?




x

# 3. BiDAF: Bi-Directional Attention Flow for Machine Comprehension

## (Seo, Kembhavi, Farhadi, Hajishirzi, ICLR 2017)



# BiDAF – Roughly the CS224N DFP baseline

- There are variants of and improvements to the BiDAF architecture over the years, but the central idea is **the Attention Flow layer**
- **Idea:** attention should flow both ways – from the context to the question and from the question to the context
- Make similarity matrix (with **w** of dimension 6*d*):

$$\boldsymbol{S}_{ij} = \boldsymbol{w}_{\text{sim}}^T [\boldsymbol{c}_i; \boldsymbol{q}_j; \boldsymbol{c}_i \circ \boldsymbol{q}_j] \in \mathbb{R}$$

- Context-to-Question (C2Q) attention:
  (which query words are most relevant to each context word)

$$\alpha^i = \text{softmax}(\boldsymbol{S}_{i,:}) \in \mathbb{R}^M \quad \forall i \in \{1, \ldots, N\}$$

$$\boldsymbol{a}_i = \sum_{j=1}^{M} \alpha_j^i \boldsymbol{q}_j \in \mathbb{R}^{2h} \quad \forall i \in \{1, \ldots, N\}$$

# BiDAF

- **Attention Flow Idea:** attention should flow both ways – from the context to the question and from the question to the context
- Question-to-Context (Q2C) attention:
  (the weighted sum of the most important words in the context with respect to the query – slight asymmetry through max)

$$\boldsymbol{m}_i = \max_j \boldsymbol{S}_{ij} \in \mathbb{R} \quad \forall i \in \{1, \ldots, N\}$$

$$\beta = \text{softmax}(\boldsymbol{m}) \in \mathbb{R}^N$$

$$\boldsymbol{c}' = \sum_{i=1}^{N} \beta_i \boldsymbol{c}_i \in \mathbb{R}^{2h}$$

- For each passage position, output of BiDAF layer is:

$$\boldsymbol{b}_i = [\boldsymbol{c}_i; \boldsymbol{a}_i; \boldsymbol{c}_i \circ \boldsymbol{a}_i; \boldsymbol{c}_i \circ \boldsymbol{c}'] \in \mathbb{R}^{8h} \quad \forall i \in \{1, \ldots, N\}$$

# BiDAF

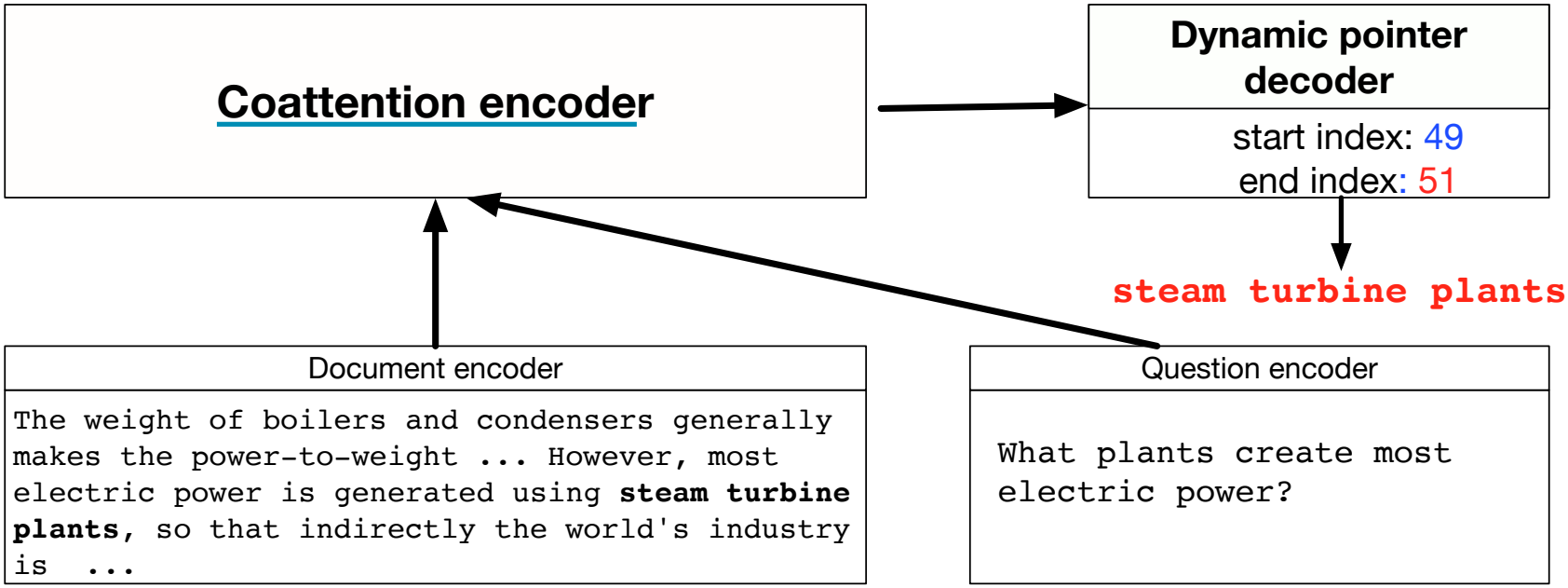
- There is then a "modelling" layer:
  - Another deep (2-layer) BiLSTM over the passage
- And answer span selection is more complex:
  - Start: Pass output of BiDAF and modelling layer concatenated to a dense FF layer and then a softmax
  - End: Put output of modelling layer M through another BiLSTM to give $M_2$ and then concatenate with BiDAF layer and again put through dense FF layer and a softmax
    - Editorial: Seems very complex, but it does seem like you should do a bit more than Stanford Attentive Reader, e.g., conditioning end also on start

# 4. Recent, more advanced architectures

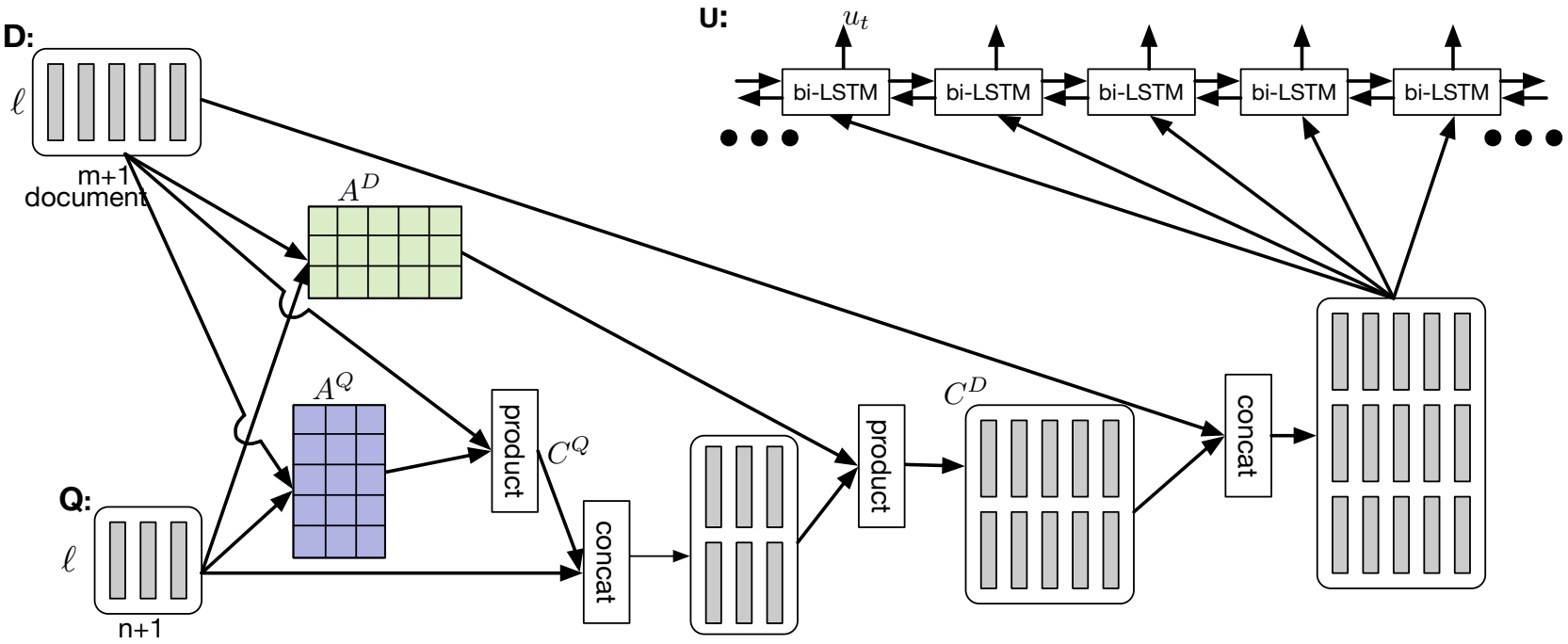
Most of the question answering work in 2016–2018 employed progressively more complex architectures with a multitude of variants of attention – often yielding good task gains

# Dynamic Coattention Networks for Question Answering (Caiming Xiong, Victor Zhong, Richard Socher ICLR 2017)

- Flaw: Questions have input-independent representations
- Interdependence needed for a comprehensive QA model



# Coattention Encoder



# Coattention layer

- Coattention layer again provides a two-way attention between the context and the question
- However, coattention involves a second-level attention computation:
  - attending over representations that are themselves attention outputs
- We use the C2Q attention distributions $\alpha_i$ to take weighted sums of the Q2C attention outputs $\mathbf{b}_j$. This gives us second-level attention outputs $\mathbf{s}_i$:

$$\boldsymbol{s}_i = \sum_{j=1}^{M+1} \alpha_j^i \boldsymbol{b}_j \in \mathbb{R}^l \quad \forall i \in \{1, \ldots, N\}$$

# Co-attention: Results on SQUAD Competition

| Model | Dev EM | Dev F1 | Test EM | Test F1 |
|---|---|---|---|---|
| *Ensemble* | | | | |
| DCN (Ours) | **70.3** | **79.4** | **71.2** | **80.4** |
| Microsoft Research Asia * | – | – | 69.4 | 78.3 |
| Allen Institute * | 69.2 | 77.8 | 69.9 | 78.1 |
| Singapore Management University * | 67.6 | 76.8 | 67.9 | 77.0 |
| Google NYC * | 68.2 | 76.7 | – | – |
| *Single model* | | | | |
| DCN (Ours) | 65.4 | **75.6** | **66.2** | **75.9** |
| Microsoft Research Asia * | 65.9 | 75.2 | 65.5 | 75.0 |
| Google NYC * | **66.4** | 74.9 | – | – |
| Singapore Management University * | – | – | 64.7 | 73.7 |
| Carnegie Mellon University * | – | – | 62.5 | 73.3 |
| Dynamic Chunk Reader (Yu et al., 2016) | 62.5 | 71.2 | 62.5 | 71.0 |
| Match-LSTM (Wang & Jiang, 2016) | 59.1 | 70.0 | 59.5 | 70.3 |
| Baseline (Rajpurkar et al., 2016) | 40.0 | 51.0 | 40.4 | 51.0 |
| Human (Rajpurkar et al., 2016) | 81.4 | 91.0 | 82.3 | 91.2 |

Results are at time of ICLR submission

See https://rajpurkar.github.io/SQuAD-explorer/ for latest results

# FusionNet (Huang, Zhu, Shen, Chen 2017)

## Attention functions

MLP (Additive) form:

$$S_{ij} = s^T \tanh(W_1 c_i + W_2 q_j)$$

Space: O(mnk), W is kxd

Bilinear (Product) form:

$$S_{ij} = c_i^T W q_j$$

$$S_{ij} = c_i^T U^T V q_j$$

$$S_{ij} = (U c_i)^T (V q_j)$$

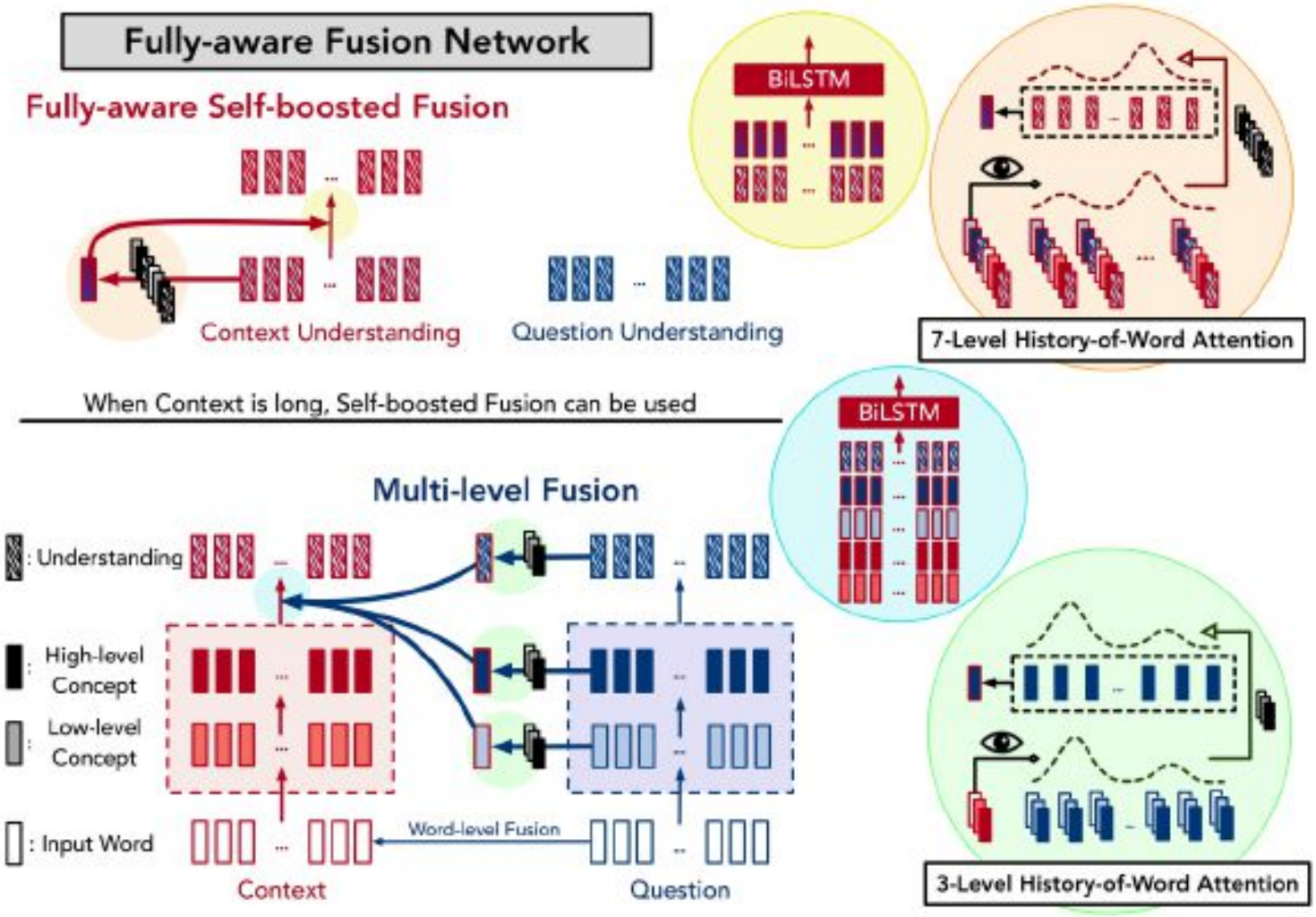
1. Smaller space
2. Non-linearity

Space: O((m+n)k)

$$S_{ij} = c_i^T W^T D W q_j$$

$$S_{ij} = Relu(c_i^T W^T) D Relu(W q_j)$$

# FusionNet tries to combine many forms of attention



# Multi-level inter-attention

$$\{\boldsymbol{m}_i^{(k),C}\}_{i=1}^m = \text{Attn}(\{\text{HoW}_i^C\}_{i=1}^m, \{\text{HoW}_i^Q\}_{i=1}^n, \{\boldsymbol{h}_i^{Q,k}\}_{i=1}^n), 1 \le k \le K+1$$

$$\text{HoW}_i^C = [\text{GloVe}(w_i^C); \text{BERT}_{w_i^C}; \boldsymbol{h}_i^{C,1}; \ldots, \boldsymbol{h}_i^{C,k}],$$

$$\text{HoW}_i^Q = [\text{GloVe}(w_i^Q); \text{BERT}_{w_i^Q}; \boldsymbol{h}_i^{Q,1}; \ldots, \boldsymbol{h}_i^{Q,k}].$$

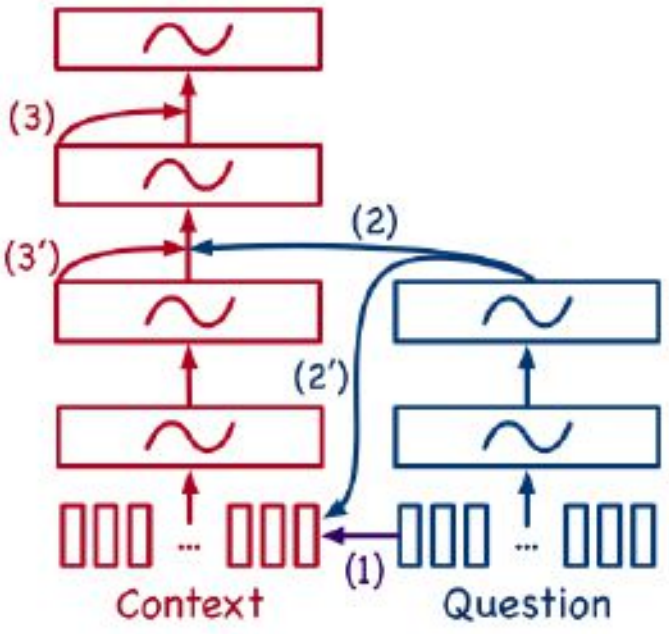
$$S_{ij} = (\text{HoW}_i^A)^T U^T V (\text{HoW}_j^B)$$

After multi-level inter-attention, use RNN, self-attention and another RNN to obtain the final representation of context: $\{\boldsymbol{u}_i^C\}$

# Recent, more advanced architectures

- Most of the question answering work in 2016–2018 employed progressively more complex architectures with a multitude of variants of attention – often yielding good task gains

| Architectures | (1) | (2) | (2') | (3) | (3') |
|---|---|---|---|---|---|
| Match-LSTM (Wang and Jiang, 2016) | | ✓ | | | |
| DCN (Xiong et al., 2017) | | ✓ | | | ✓ |
| FastQA (Weissenborn et al., 2017) | ✓ | | | | |
| FastQAExt (Weissenborn et al., 2017) | ✓ | ✓ | | ✓ | |
| BiDAF (Seo et al., 2017) | | ✓ | | | ✓ |
| RaSoR (Lee et al., 2016) | ✓ | | ✓ | | |
| DrQA (Chen et al., 2017) | ✓ | | | | |
| MPCM (Wang et al., 2016) | ✓ | ✓ | | | |
| Mnemonic Reader (Hu et al., 2017) | ✓ | ✓ | | ✓ | |
| R-net (Wang et al., 2017b) | | ✓ | | ✓ | |



# SQuAD limitations

- SQuAD has a number of key limitations:
  - Only span-based answers (no yes/no, counting, implicit why)
  - Questions were constructed looking at the passages
    - Not genuine information needs
    - Generally greater lexical and syntactic matching between questions and answer span than you get IRL
  - Barely any multi-fact/sentence inference beyond coreference

- Nevertheless, it is a well-targeted, well-structured, clean dataset
  - It has been the most used and competed on QA dataset
  - It has also been a useful starting point for building systems in industry (though in-domain data always really helps!)
  - And we're using it (SQuAD 2.0)

# 5. Open-domain Question Answering

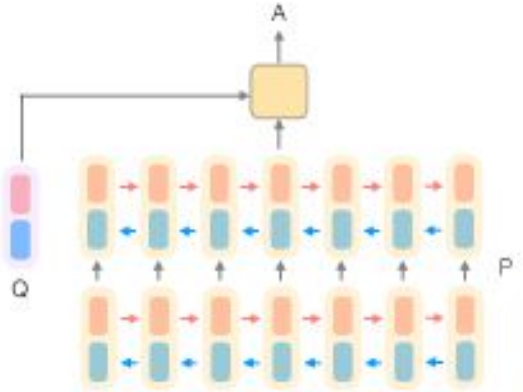
## DrQA (Chen, et al. ACL 2017) https://arxiv.org/abs/1704.00051

Q: How many of Warsaw's inhabitants spoke Polish in 1933?


**Document Retriever**

**Document Reader**

833,500



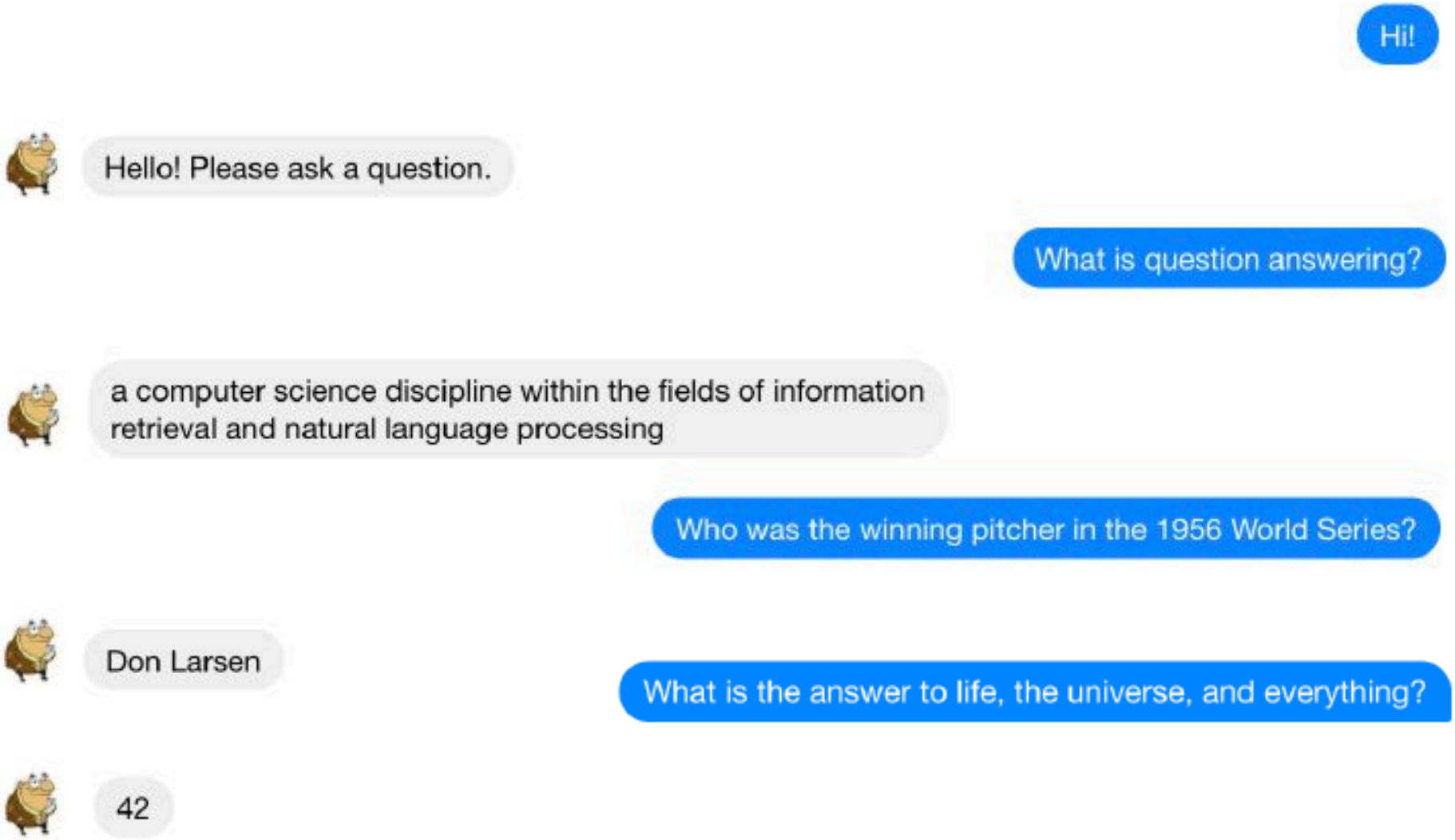
# Document Retriever

| Dataset | Wiki Search | Doc. Retriever plain | +bigrams |
|---|---|---|---|
| SQuAD | 62.7 | 76.1 | **77.8** |
| CuratedTREC | 81.0 | 85.2 | **86.0** |
| WebQuestions | 73.7 | **75.5** | 74.4 |
| WikiMovies | 61.7 | 54.4 | **70.3** |

Traditional tf.idf inverted index + efficient bigram hash

For **70–86%** of questions, the answer segment appears in the top 5 articles

# DrQA Demo


Hi!
Hello! Please ask a question.
What is question answering?
a computer science discipline within the fields of information retrieval and natural language processing
Who was the winning pitcher in the 1956 World Series?
Don Larsen
What is the answer to life, the universe, and everything?
42

# General questions

Combined with **Web search**, DrQA can answer **57.5%** of **trivia questions** correctly


bing

**Q**: The Dodecanese Campaign of WWII that was an attempt by the Allied forces to capture islands in the Aegean Sea was the inspiration for which acclaimed 1961 commando film?

**A**: The Guns of Navarone

**Q**: American Callan Pinckney's eponymously named system became a best-selling (1980s-2000s) book/video franchise in what genre?

**A**: Fitness

# 6. LSTMs, attention, and transformers intro

# SQuAD v1.1 leaderboard, 2019-02-07

| Rank | Model | EM | F1 |
|---|---|---|---|
| | Human Performance<br>*Stanford University*<br>(Rajpurkar et al. '16) | 82.304 | 91.221 |
| 1<br>Oct 05, 2018 | BERT (ensemble)<br>*Google AI Language*<br>https://arxiv.org/abs/1810.04805 | **87.433** | **93.160** |
| 2<br>Oct 05, 2018 | BERT (single model)<br>*Google AI Language*<br>https://arxiv.org/abs/1810.04805 | 85.083 | 91.835 |
| 2<br>Sep 09, 2018 | nlnet (ensemble)<br>*Microsoft Research Asia* | 85.356 | 91.202 |
| 2<br>Sep 26, 2018 | nlnet (ensemble)<br>*Microsoft Research Asia* | 85.954 | 91.677 |
| 3<br>Jul 11, 2018 | QANet (ensemble)<br>*Google Brain & CMU* | 84.454 | 90.490 |
| 4<br>Jul 08, 2018 | r-net (ensemble)<br>*Microsoft Research Asia* | 84.003 | 90.147 |
| 5<br>Mar 19, 2018 | QANet (ensemble)<br>*Google Brain & CMU* | 83.877 | 89.737 |
| 5<br>Sep 09, 2018 | nlnet (single model)<br>*Microsoft Research Asia* | 83.468 | 90.133 |

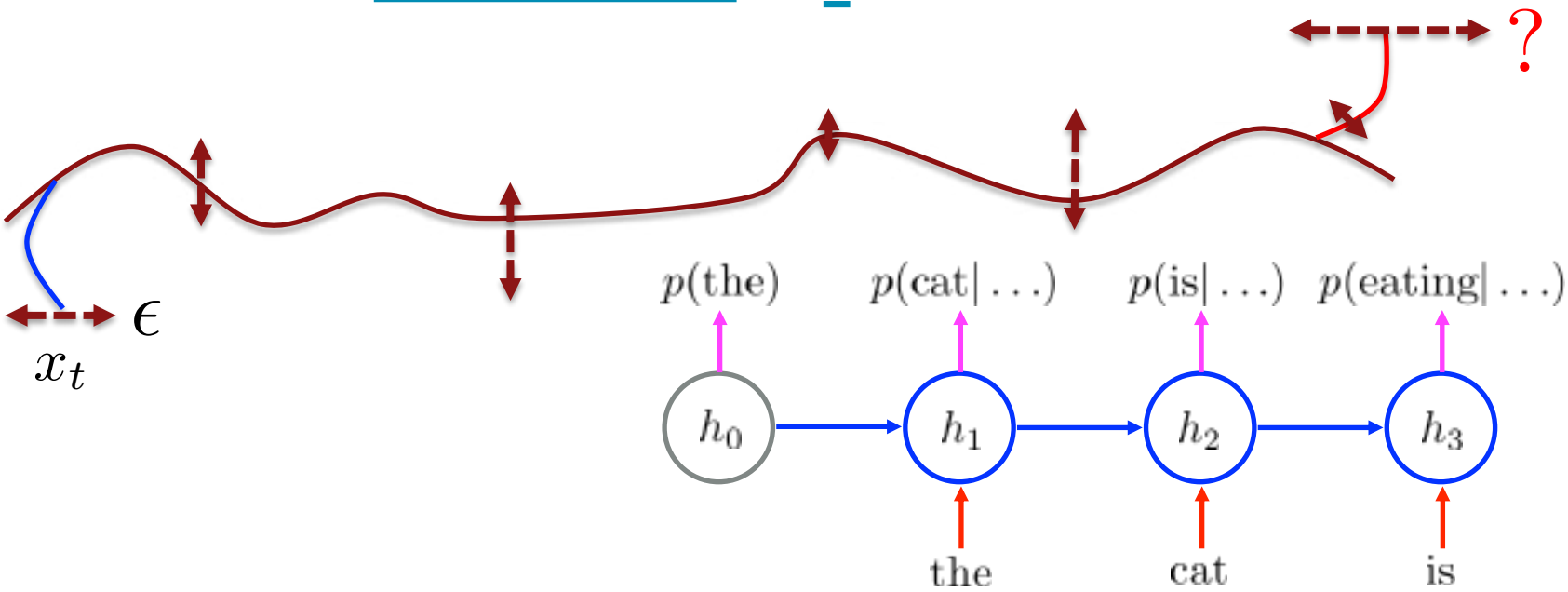
# Gated Recurrent Units, again

*Intuitively, what happens with RNNs?*

1. Measure the influence of the past on the future

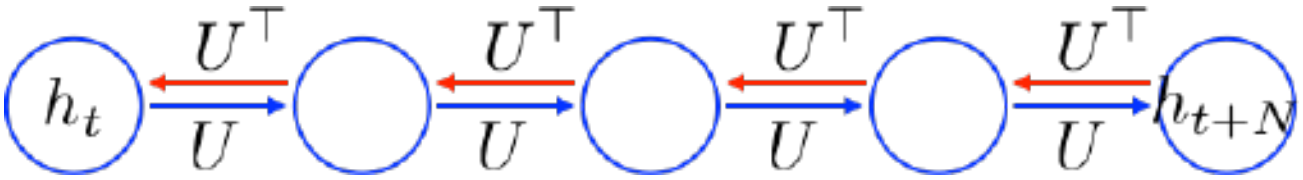
$$\frac{\partial \log p(x_{t+n}|x_{<t+n})}{\partial h_t} = \frac{\partial \log p(x_{t+n}|x_{<t+n})}{\partial g} \frac{\partial g}{\partial h_{t+n}} \frac{\partial h_{t+n}}{\partial h_{t+n-1}} \cdots \frac{\partial h_{t+1}}{\partial h_t}$$

2. How does the perturbation at $t$ affect $p(x_{t+n}|x_{<t+n})$?

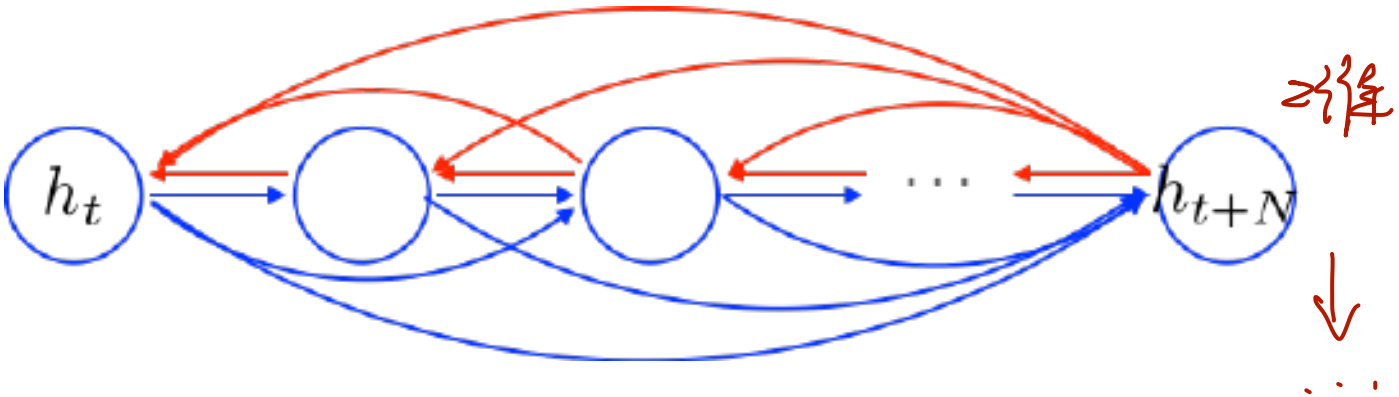


# Gated Recurrent Units : LSTM & GRU

- The signal and error must propagate through all the intermediate nodes:

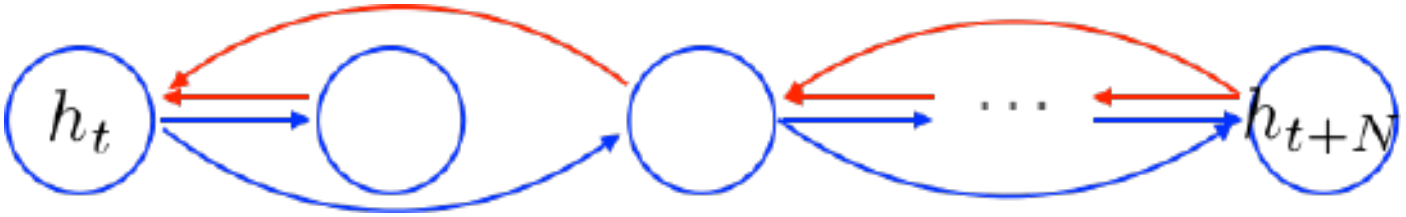



- Perhaps we can create shortcut connections.



# Gated Recurrent Unit

- Perhaps we can create *adaptive* shortcut connections.
- Let the net prune unnecessary connections *adaptively*.




$$f(h_{t-1}, x_t) = u_t \odot \tilde{h}_t + (1 - u_t) \odot h_{t-1}$$

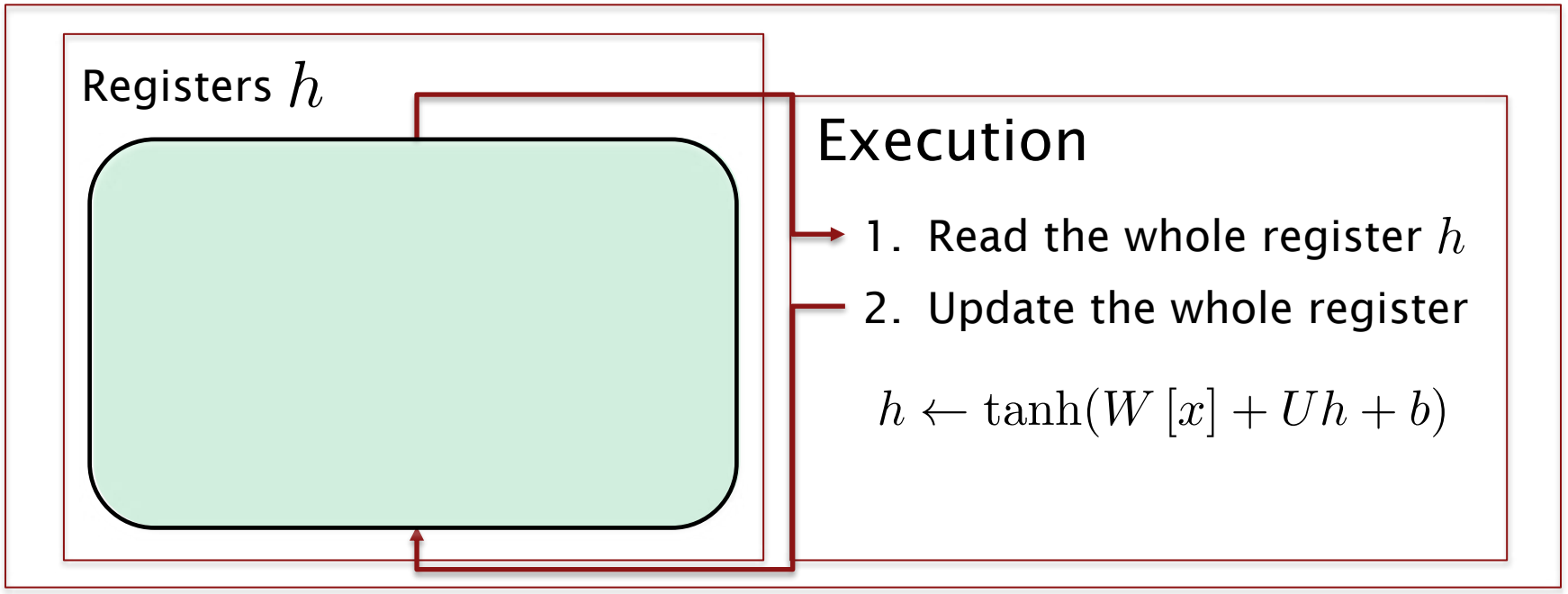
- Candidate Update $\tilde{h}_t = \tanh(W\,[x_t] + U(r_t \odot h_{t-1}) + b)$
- Reset gate $r_t = \sigma(W_r\,[x_t] + U_r h_{t-1} + b_r)$
- Update gate $u_t = \sigma(W_u\,[x_t] + U_u h_{t-1} + b_u)$



$\odot$: element-wise multiplication

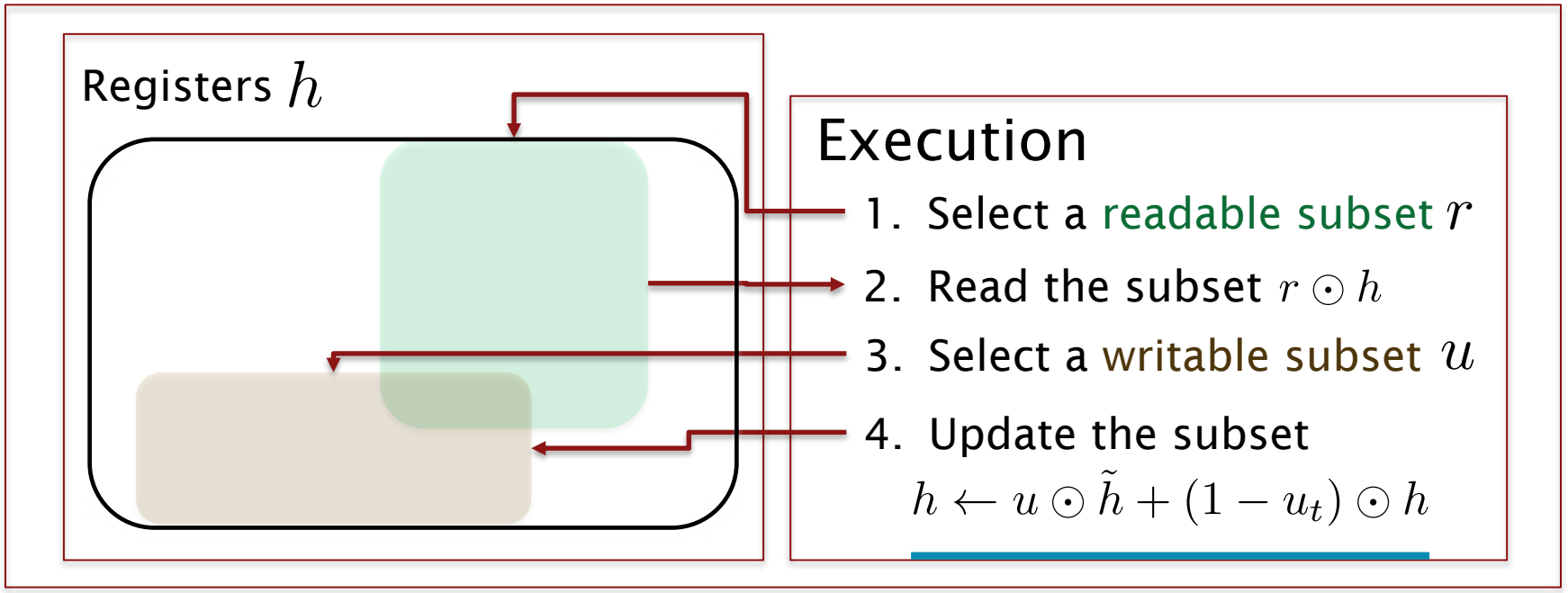
# Gated Recurrent Unit

*tanh-RNN* ….



# Gated Recurrent Unit

*GRU …*




Gated recurrent units are much more realistic for computation!

# Gated Recurrent Units: LSTM & GRU

*Two most widely used gated recurrent units: GRU and LSTM*

## Gated Recurrent Unit

**[Cho et al., EMNLP2014; Chung, Gulcehre, Cho, Bengio, DLUFL2014]**

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

$$\tilde{h}_t = \tanh(W\,[x_t] + U(r_t \odot h_{t-1}) + b)$$

$$u_t = \sigma(W_u\,[x_t] + U_u h_{t-1} + b_u)$$

$$r_t = \sigma(W_r\,[x_t] + U_r h_{t-1} + b_r)$$

## Long Short-Term Memory

**[Hochreiter & Schmidhuber, NC1999; Gers, Thesis2001]**

$$h_t = o_t \odot \tanh(c_t)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

$$\tilde{c}_t = \tanh(W_c\,[x_t] + U_c h_{t-1} + b_c)$$

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

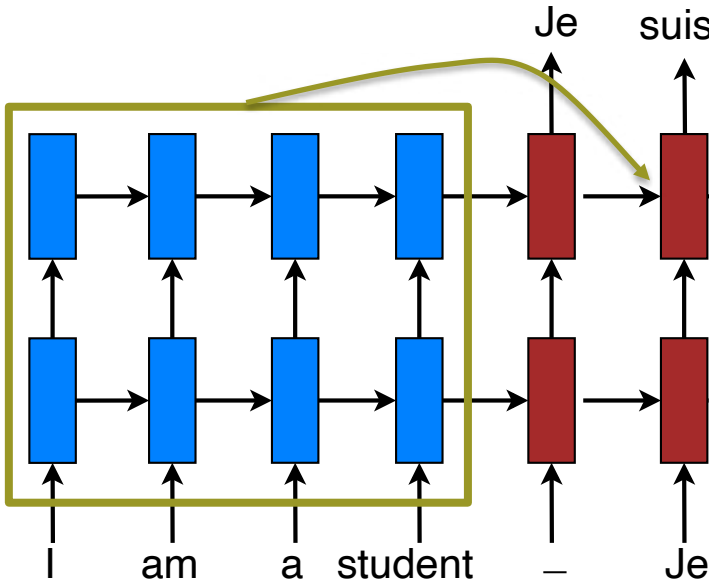
$$i_t = \sigma(W_i\,[x_t] + U_i h_{t-1} + b_i)$$

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

# Attention Mechanism

Started in computer vision! [Larochelle & Hinton, 2010], [Denil, Bazzani, Larochelle, Freitas, 2012]
Became famous in NMT/NLM

Pool of source states




- A second solution: random access memory
  - Retrieve past info as needed (but usually average)
  - Usually do content-similarity based addressing
    - Other things like positional are occasionally tried

# ELMo and BERT preview

**Contextual word representations**
Using language model-like objectives

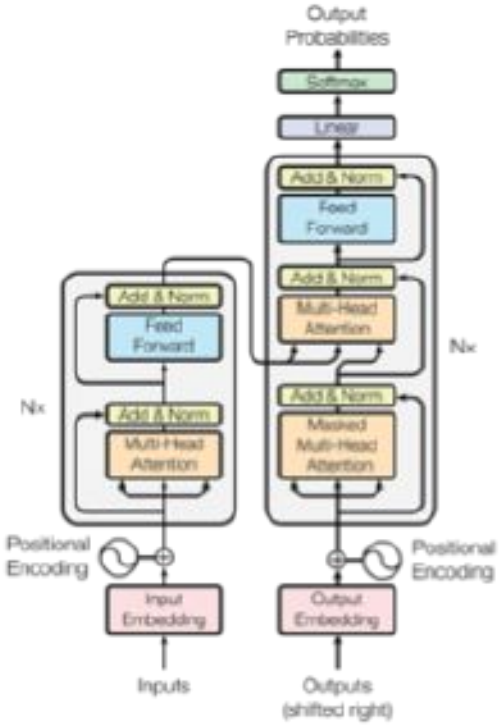
**Elmo**

(Peters et al, 2018)

**Bert**

(Devlin et al, 2018)

The transformer architecture used in BERT is sort of attention on steroids.

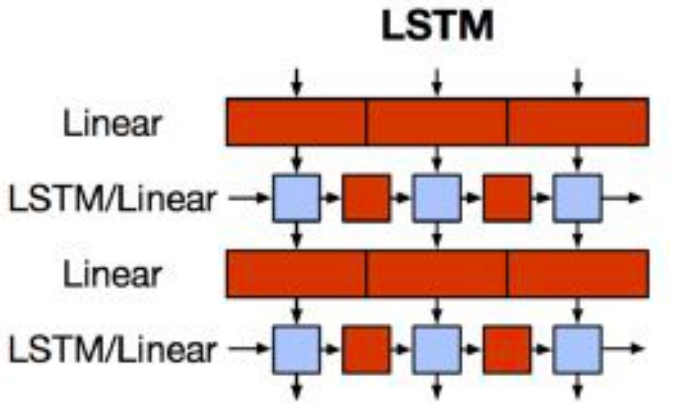



(Vaswani et al, 2017)

Look at SDNet as an example of how to use BERT as submodule: https://arxiv.org/abs/1812.03593

# The Motivation for Transformers

- We want **parallelization** but RNNs are inherently sequential



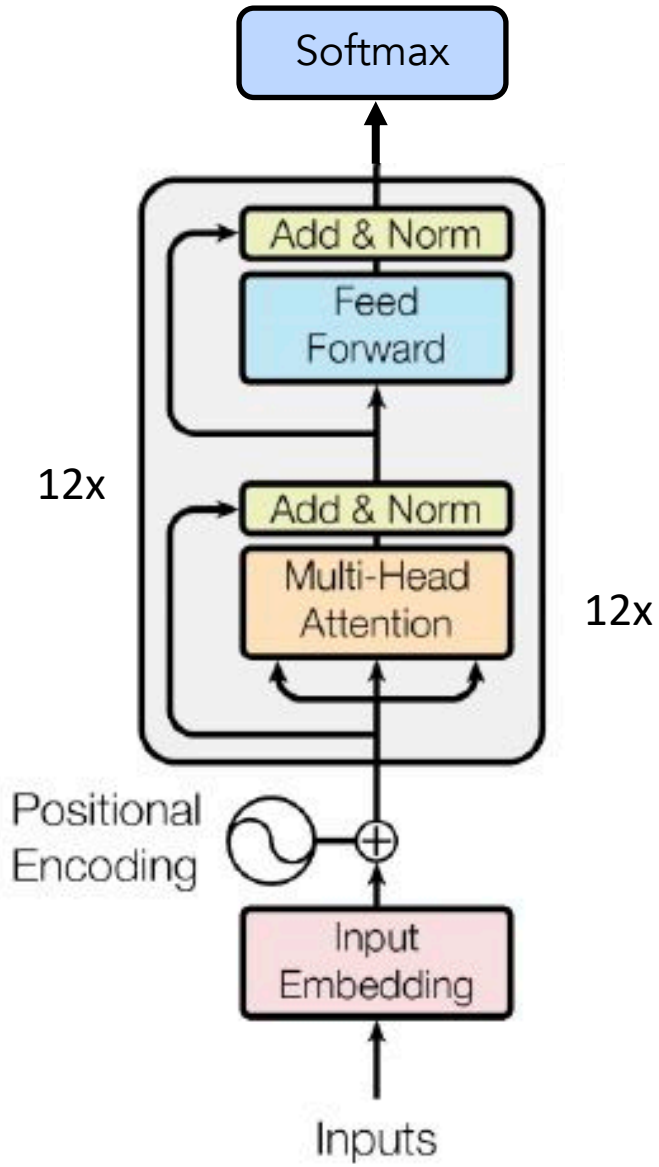

- Despite LSTMs, RNNs generally need attention mechanism to deal with long range dependencies – **path length** between states grows with distance otherwise
- But if **attention** gives us access to any state… maybe we can just use attention and don't need the RNN? 🤔
- And then NLP can have deep models … and solve our vision envy

# Transformer (Vaswani et al. 2017) "Attention is all you need"

https://arxiv.org/pdf/1706.03762.pdf

- **Non-recurrent** sequence (or sequence-to-sequence) model
- A **deep** model with a sequence of **attention**-based transformer blocks
- Depth allows a certain amount of lateral information transfer in understanding sentences, in slightly unclear ways
- Final cost/error function is standard cross-entropy error on top of a softmax classifier

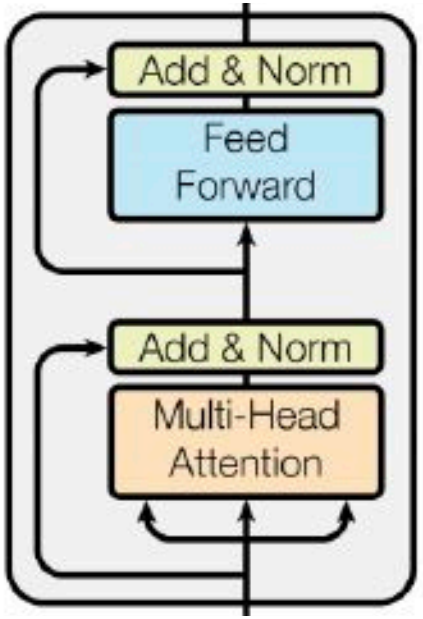
Initially built for NMT



# Transformer block

Each block has two "sublayers"

1. Multihead attention
2. 2-layer feed-forward NNet (with ReLU)




Each of these two steps also has:

Residual (short-circuit) connection

LayerNorm (scale to mean 0, var 1; Ba et al. 2016)

# Multi-head (self) attention

With simple self-attention: Only one way for a word to interact with others

Solution: Multi-head attention

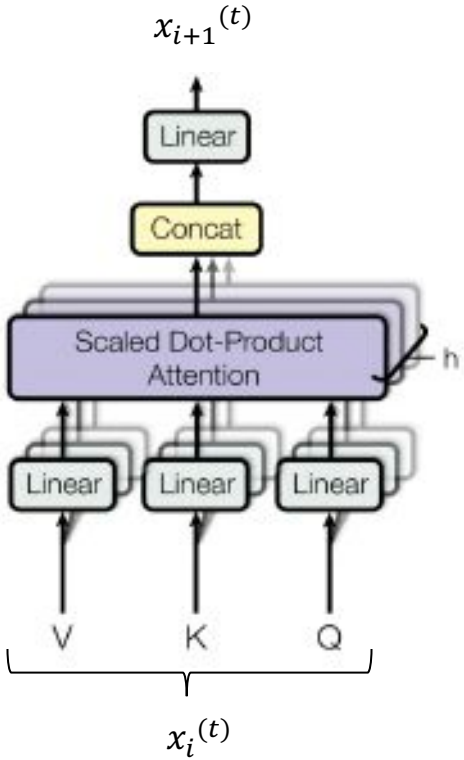
Map input into $h = 12$ many lower dimensional spaces via $W_h$ matrices

Then apply attention, then concatenate outputs and pipe through linear layer

$$\text{Multihead}(x_i^{(t)}) = \text{Concat}(head_j)W^O$$

$$head_j = \text{Attention}(x_i^{(t)}W_j^Q, x_i^{(t)}W_j^K, x_i^{(t)}W_j^V)$$

So attention is like bilinear: $x_i^{(t)}(W_j^Q(W_j^K)^T)x_i^{(l)}$



# Encoder Input

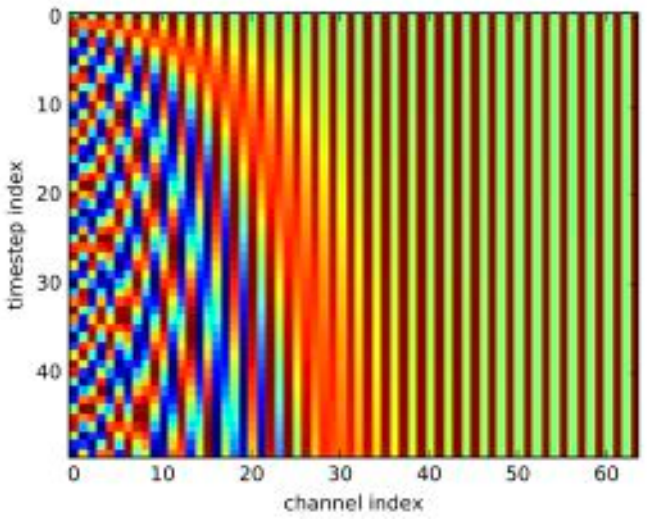
Actual word representations are word pieces (byte pair encoding)

- Topic of next week

Also added is a **positional encoding** so same words at different locations have different overall representations:

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

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



# BERT: Devlin, Chang, Lee, Toutanova (2018)

BERT (Bidirectional Encoder Representations from Transformers):

Pre-training of Deep Bidirectional Transformers for Language Understanding, which is then fine-tuned for a particular task

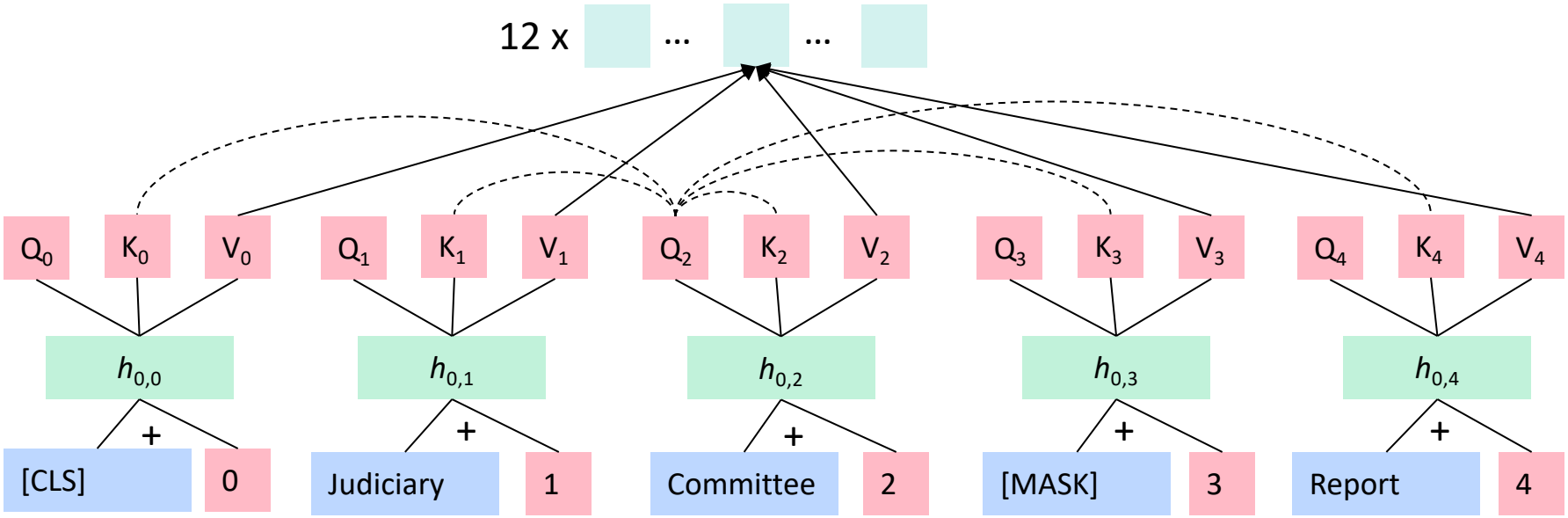
Pre-training uses a cloze task formulation where 15% of words are masked out and predicted:

store gallon

↑ ↑

the man went to the [MASK] to buy a [MASK] of milk

# Transformer (Vaswani et al. 2017)
# BERT (Devlin et al. 2018)

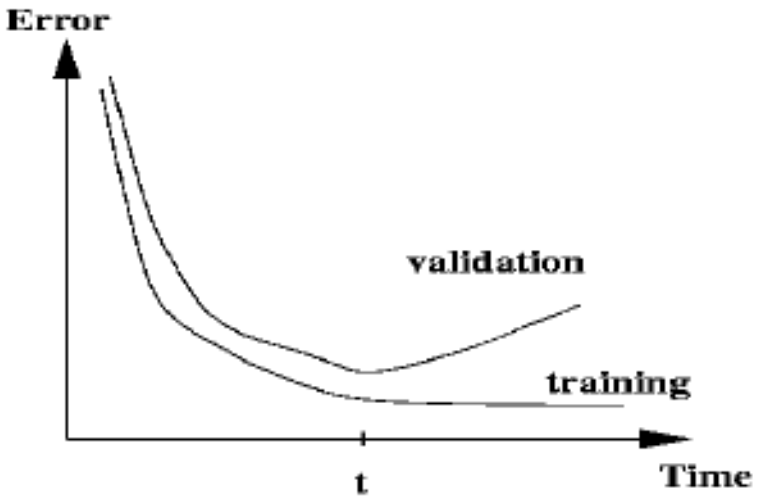


# 7. Pots of data

- Many publicly available datasets are released with a **train/dev/test** structure. **We're all on the honor system to do test-set runs only when development is complete.**
- Splits like this presuppose a fairly large dataset.
- If there is no dev set or you want a separate tune set, then you create one by splitting the training data, though you have to weigh its size/usefulness against the reduction in train-set size.
- Having a fixed test set ensures that all systems are assessed against the same gold data. This is generally good, but it is problematic where the test set turns out to have unusual properties that distort progress on the task.

# Training models and pots of data

- When training, models **overfit** to what you are training on
  - The model correctly describes what happened to occur in particular data you trained on, but the patterns are not general enough patterns to be likely to apply to new data
- The way to monitor and avoid problematic overfitting is using **independent** validation and test sets …



# Training models and pots of data

- You build (estimate/train) a model on a **training set**.
- Often, you then set further hyperparameters on another, independent set of data, the **tuning set**
  - The tuning set is the training set for the hyperparameters!
- You measure progress as you go on a **dev set** (development test set or validation set)
  - If you do that a lot you overfit to the dev set so it can be good to have a second dev set, the **dev2** set
- **Only at the end**, you evaluate and present final numbers on a **test set**
  - Use the final test set **extremely** few times … ideally only once

# Training models and pots of data

- The **train**, **tune**, **dev**, and **test** sets need to be completely distinct
- It is invalid to test on material you have trained on
  - You will get a falsely good performance. We usually overfit on train
- You need an independent tuning set
  - The hyperparameters won't be set right if tune is same as train
- If you keep running on the same evaluation set, you begin to overfit to that evaluation set
  - Effectively you are "training" on the evaluation set … you are learning things that do and don't work on that particular eval set and using the info
- To get a valid measure of system performance you need another untrained on, **independent** test set … hence dev2 and final test

# 8. Getting your neural network to train

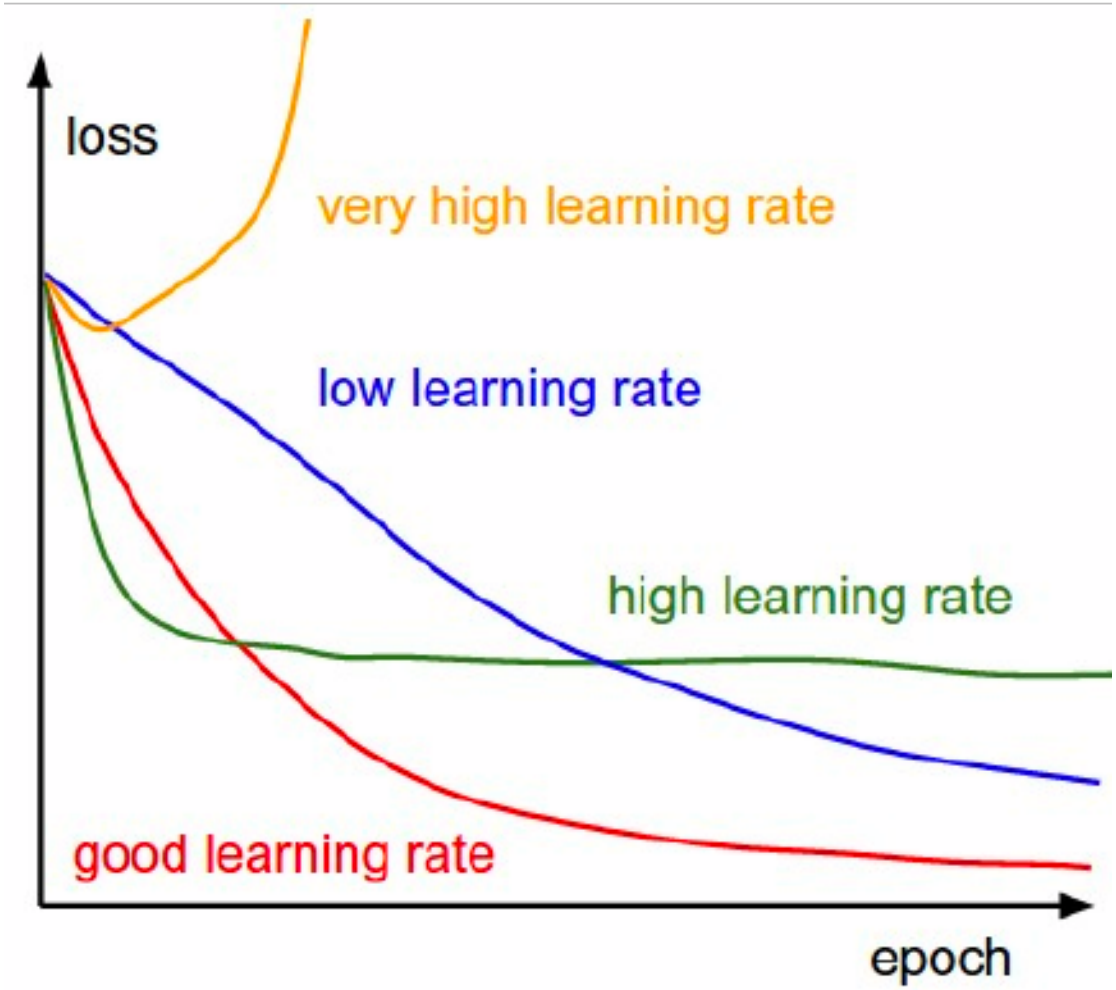
- Start with a positive attitude!
  - **Neural networks want to learn!**
    - If the network isn't learning, you're doing something to prevent it from learning successfully

- Realize the grim reality:
  - **There are lots of things that can cause neural nets to not learn at all or to not learn very well**
    - Finding and fixing them ("debugging and tuning") can often take more time than implementing your model

- It's hard to work out what these things are
  - But experience, experimental care, and rules of thumb help!

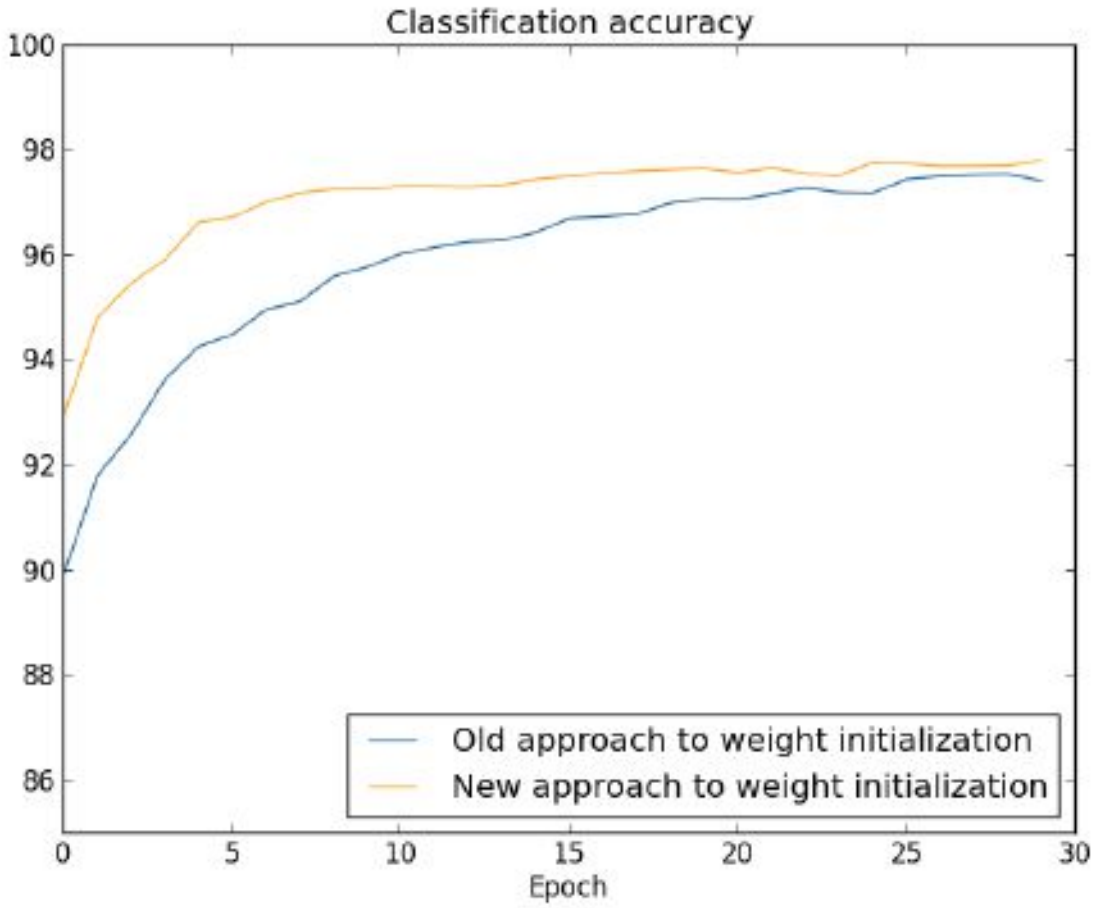
# Models are sensitive to learning rates

- From Andrej Karpathy, CS231n course notes



# Models are sensitive to initialization

- From Michael Nielsen http://neuralnetworksanddeeplearning.com/chap3.html



# Training a gated RNN

1. Use an LSTM or GRU: *it makes your life so much simpler!*
2. Initialize recurrent matrices to be orthogonal
3. Initialize other matrices with a sensible (**small!**) scale
4. Initialize forget gate bias to 1: *default to remembering*
5. Use adaptive learning rate algorithms: *Adam, AdaDelta,* …
6. Clip the norm of the gradient: *1–5 seems to be a reasonable threshold when used together with Adam or AdaDelta.*
7. Either only dropout vertically or look into using Bayesian Dropout (Gal and Gahramani – not natively in PyTorch)
8. *Be patient! Optimization takes time*

[Saxe et al., ICLR2014;
Ba, Kingma, ICLR2015;
Zeiler, arXiv2012;
Pascanu et al., ICML2013]

# Experimental strategy

- Work incrementally!
- Start with a very simple model and get it to work!
  - It's hard to fix a complex but broken model
- Add bells and whistles one-by-one and get the model working with each of them (or abandon them)

- Initially run on a tiny amount of data
  - You will see bugs much more easily on a tiny dataset
  - Something like 4–8 examples is good
  - Often synthetic data is useful for this
  - Make sure you can get 100% on this data
    - Otherwise your model is definitely either not powerful enough or it is broken

# Experimental strategy

- Run your model on a large dataset
  - It should still score close to 100% on the training data after optimization
    - Otherwise, you probably want to consider a more powerful model
    - Overfitting to training data is **not** something to be scared of when doing deep learning
      - These models are usually good at generalizing because of the way distributed representations share statistical strength regardless of overfitting to training data
- But, still, you now want good generalization performance:
  - Regularize your model until it doesn't overfit on dev data
    - Strategies like L2 regularization can be useful
    - But normally **generous dropout** is the secret to success

# Details matter!

- Look at your data, collect summary statistics
- Look at your model's outputs, do error analysis
- Tuning hyperparameters is **really** important to almost all of the successes of NNets

**Good luck with your projects!**