CS 533: Natural Language Processing

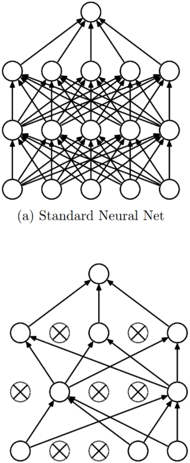
Backpropagation, Self-Attention, Text Representations Through Language Modeling

Karl Stratos

Rutgers University

Dropout

- Form of regularization for RNNs (and any NN in general)
- Idea: "Handicap" NN by removing hidden units **stochastically**
  - set each hidden unit in a layer to 0 with probability p during training ($p = 0.5$ usually works well)
  - scale outputs by $1/(1-p)$
  - hidden units forced to learn more general patterns
- Test time: Simply compute identity

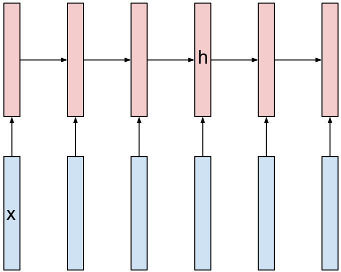


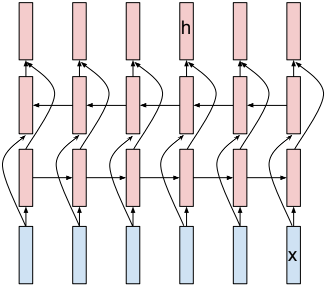
(a) Standard Neural Net

(b) After applying dropout.

(Slide credit Danqi Chen & Karthik Narasimhan)

Unidirectional vs Bidirectional RNN





Agenda

1. Backpropagation
2. Self-attention in NLP
3. Representation learning through language modeling

Backpropagation: Input and Output

- A technique to automatically calculate $\nabla J(\theta)$ for any definition of scalar-valued loss function $J(\theta) \in \mathbb{R}$.

  Input: loss function $J(\theta) \in \mathbb{R}$, parameter value $\hat{\theta}$
  Output: $\nabla J(\hat{\theta})$, the gradient of $J(\theta)$ at $\theta = \hat{\theta}$

- Calculates the gradient of an arbitrary differentiable function of parameter θ

  Including neural networks

Notation

- For the most part, we will consider (differentiable) function $f : \mathbb{R} \to \mathbb{R}$ with a single 1-dimensional parameter $x \in \mathbb{R}$.

- The gradient of f with respect to x is a **function** of x

$$\frac{\partial f(x)}{\partial x} : \mathbb{R} \to \mathbb{R}$$

- The gradient of f with respect to x evaluated at $x = a$ is written as

$$\left.\frac{\partial f(x)}{\partial x}\right|_{x=a} \in \mathbb{R}$$

Chain Rule

- Given any differentiable functions f, g from $\mathbb{R}$ to $\mathbb{R}$,

$$\frac{\partial g(f(x))}{\partial x} = \frac{\partial g(f(x))}{\partial f(x)} \times \underbrace{\frac{\partial f(x)}{\partial x}}_{\text{easy to calculate}}$$

- "Proof": Linearization of linearization of $g(z)$ around $f(x)$ around a

$$g(f(x)) \approx g(f(a)) + \underbrace{g'(f(a))f'(a)}_{\left.\frac{\partial g(f(x))}{\partial x}\right|_{x=a}}(x-a)$$

Exercises

At $x = 42$,

- What is the value of the gradient of $f(x) := 7$?
- What is the value of the gradient of $f(x) := 2x$?
- What is the value of the gradient of $f(x) := 2x + 99999$?
- What is the value of the gradient of $f(x) := x^3$?
- What is the value of the gradient of $f(x) := \exp(x)$?
- What is the value of the gradient of $f(x) := \exp(2x^3 + 10)$?
- What is the value of the gradient of

$$f(x) := \log(\exp(2x^3 + 10))$$

Chain Rule for a Function of Multiple Input Variables

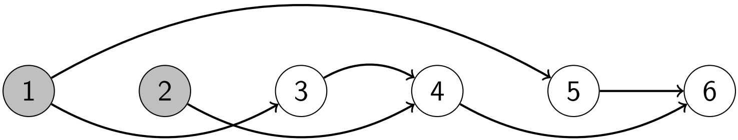
- Let $f_1 \ldots f_m$ denote any differentiable functions from $\mathbb{R}$ to $\mathbb{R}$.
- If $g : \mathbb{R}^m \rightarrow \mathbb{R}$ is a differentiable function from $\mathbb{R}^m$ to $\mathbb{R}$,

$$\frac{\partial g(f_1(x), \ldots, f_m(x))}{\partial x} = \sum_{i=1}^{m} \frac{\partial g(f_1(x), \ldots, f_m(x))}{\partial f_i(x)} \times \underbrace{\frac{\partial f_i(x)}{\partial x}}_{\text{easy to calculate}}$$

- Calculate the gradient of $x + x^2 + yx$ with respect to x using the chain rule.

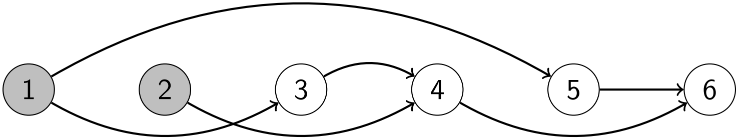
DAG

A **directed acylic graph (DAG)** is a directed graph $G = (V, A)$ with a **topological ordering**: a sequence π of V such that for every arc $(i, j) \in A$, i comes before j in π.



For backpropagation: usually assume have many roots and 1 leaf

Notation



$$V = \{1, 2, 3, 4, 5, 6\}$$

$$V_I = \{1, 2\}$$

$$V_N = \{3, 4, 5, 6\}$$

$$A = \{(1, 3), (1, 5), (2, 4), (3, 4), (4, 6), (5, 6)\}$$

$$\textbf{pa}(4) = \{2, 3\}$$

$$\textbf{ch}(1) = \{3, 5\}$$

$$\Pi_G = \{(1, 2, 3, 4, 5, 6),\ (2, 1, 3, 4, 5, 6)\}$$

Computation Graph

- DAG $G = (V, E)$ with a single output node $\omega \in V$.
- Every node $i \in V$ is equipped with a **value** $x^i \in \mathbb{R}$:
  1. For input node $i \in V_I$, we assume $x^i = a^i$ is given.
  2. For non-input node $i \in V_N$, we assume a differentiable **function** $f^i : \mathbb{R}^{|\mathbf{pa}(i)|} \to \mathbb{R}$ and compute

$$x^i = f^i((x^j)_{j \in \mathbf{pa}(i)})$$

- Thus G represents a *function*: it receives multiple values $x^i = a^i$ for $i \in V_I$ and calculates a scalar $x^\omega \in \mathbb{R}$.
  - We can calculate x^ω by a **forward pass**.

Forward Pass

Input: computation graph $G = (V, A)$ with output node $\omega \in V$
Result: populates $x^i = a^i$ for every $i \in V$

1. Pick some topological ordering π of V.
2. For i **in order** of π, if $i \in V_N$ is a non-input node, set

$$x^i \leftarrow a^i := f^i((a^j)_{j \in \mathbf{pa}(i)})$$

Why do we need a topological ordering?

Exercise

Construct the computation graph associated with the function

$$f(x, y) := (x + y)xy^2$$

Compute its output value at $x = 1$ and $y = 2$ by performing a forward pass.

For Notational Convenience...

- Collectively refer to **all input slots** by $x_I = (x^i)_{i \in V_I}$.
- Collectively refer to **all input values** by $a_I = (a^i)_{i \in V_I}$.

- At $i \in V$:

  Refer to its **parental slots** by $x_I^i = (x^j)_{j \in \textbf{pa}(i)}$.
  Refer to its **parental values** by $a_I^i = (a^j)_{j \in \textbf{pa}(i)}$.

Two equally valid ways of viewing any $a^i \in \mathbb{R}$ as a function:

- A "global" function of x_I evaluated at a_I.
- A "local" function of x_I^i evaluated at a_I^i.

Computation Graph: Gradients

- Now for every node $i \in V$, we introduce an additional slot $z^i \in \mathbb{R}$ defined as

$$z^i := \left. \frac{\partial x^{\omega}}{\partial x^i} \right|_{x_I = a_I}$$

- **The goal of backpropagation** is to calculate z^i for every $i \in V$.
  - Why are we done if we achieve this goal?

Key Ideas of Backpropagation

- Chain rule on the DAG structure

$$z^i := \left. \frac{\partial x^{\omega}}{\partial x^i} \right|_{x_I = a_I}$$

Key Ideas of Backpropagation

- Chain rule on the DAG structure

$$z^i := \left.\frac{\partial x^{\omega}}{\partial x^i}\right|_{x_I = a_I} = \sum_{j \in \textbf{ch}(i)} \left.\frac{\partial x^{\omega}}{\partial x^j}\right|_{x_I = a_I} \times \left.\frac{\partial x^j}{\partial x^i}\right|_{x_I^j = a_I^j}$$

Key Ideas of Backpropagation

- Chain rule on the DAG structure

$$z^i := \left.\frac{\partial x^{\omega}}{\partial x^i}\right|_{x_I = a_I} = \sum_{j \in \textbf{ch}(i)} \left.\frac{\partial x^{\omega}}{\partial x^j}\right|_{x_I = a_I} \times \left.\frac{\partial x^j}{\partial x^i}\right|_{x_I^j = a_I^j}$$

$$= \sum_{j \in \textbf{ch}(i)} z^j \times \underbrace{\left.\frac{\partial f^j(x_I^j)}{\partial x^i}\right|_{x_I^j = a_I^j}}_{\text{easy to calculate}}$$

Key Ideas of Backpropagation

- Chain rule on the DAG structure

$$z^i := \left.\frac{\partial x^\omega}{\partial x^i}\right|_{x_I = a_I} = \sum_{j \in \textbf{ch}(i)} \left.\frac{\partial x^\omega}{\partial x^j}\right|_{x_I = a_I} \times \left.\frac{\partial x^j}{\partial x^i}\right|_{x_I^j = a_I^j}$$

$$= \sum_{j \in \textbf{ch}(i)} z^j \times \underbrace{\left.\frac{\partial f^j(x_I^j)}{\partial x^i}\right|_{x_I^j = a_I^j}}_{\text{easy to calculate}}$$

- If we compute z^i in a **reverse topological ordering**, then we will have already computed z^j for all $j \in \textbf{ch}(i)$.

Key Ideas of Backpropagation

- Chain rule on the DAG structure

$$z^i := \left.\frac{\partial x^\omega}{\partial x^i}\right|_{x_I = a_I} = \sum_{j \in \textbf{ch}(i)} \left.\frac{\partial x^\omega}{\partial x^j}\right|_{x_I = a_I} \times \left.\frac{\partial x^j}{\partial x^i}\right|_{x_I^j = a_I^j}$$

$$= \sum_{j \in \textbf{ch}(i)} z^j \times \underbrace{\left.\frac{\partial f^j(x_I^j)}{\partial x^i}\right|_{x_I^j = a_I^j}}_{\text{easy to calculate}}$$

- If we compute z^i in a **reverse topological ordering**, then we will have already computed z^j for all $j \in \textbf{ch}(i)$.
  - What's the base case z^ω?

Backpropagation

Input: computation graph $G = (V, A)$ with output node $\omega \in V$ whose value slots $x^i = a^i$ are already populated for every $i \in V$
Result: populates z^i for every $i \in V$

1. Set $z^\omega \leftarrow 1$.
2. Pick some topological ordering π of V.
3. For i **in reverse order** of π, set

$$z^i \leftarrow \sum_{j \in \textbf{ch}(i)} z^j \times \left. \frac{\partial f^j(x_I^j)}{\partial x^i} \right|_{x_I^j = a_I^j}$$

Exercise

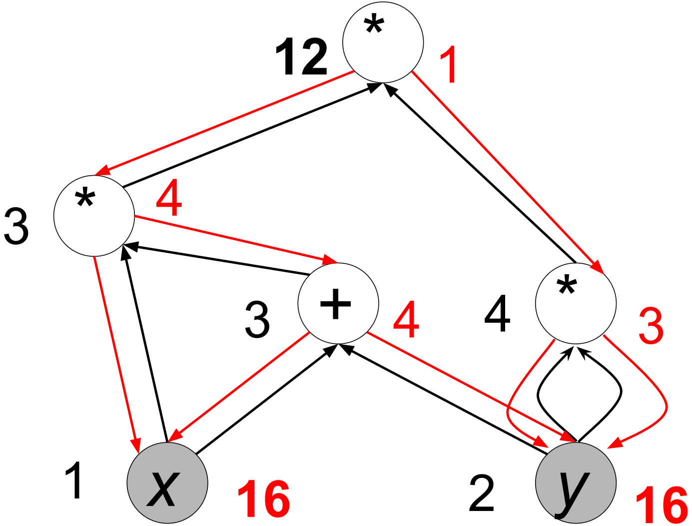
Calculate the gradient of

$$f(x, y) := (x + y)xy^2$$

with respect to x at $x = 1$ and $y = 2$ by performing backpropagation. That is, calculate the scalar

$$\left. \frac{\partial f(x, y)}{\partial x} \right|_{(x,y)=(1,2)}$$

Answer



Implementation

- Each type of function f creates a child node from parent nodes and initializes its gradient to zero.
  - "Add" function creates a child node c with two parents (a, b) and sets $c.z \leftarrow 0$.
- Each node has an associated **forward** function.
  - Calling forward at c populates $c.x = a.x + b.x$ (assumes parents have their values).
- Each node also has an associated **backward** function.
  - Calling backward at c "broadcasts" its gradient $c.z$ (assumes it's already calculated) to its parents

$$a.z \leftarrow a.z + c.z$$
$$b.z \leftarrow b.z + c.z$$

Implementation (Cont.)

- Express your loss $J_B(\theta)$ on minibatch B at $\theta = \hat{\theta}$ as a computation graph.

- **Forward pass.** For each node a in a topological ordering,

$$a.\textbf{forward}()$$

- **Backward pass.** For each node a in a reverse topological ordering,

$$a.\textbf{backward}()$$

- The gradient of $J_B(\theta)$ at $\theta = \hat{\theta}$ is stored in the input nodes of the computation graph.

General Backpropagation

- Computation graph in which input values that are vectors

$$x^i \in \mathbb{R}^{d^i} \qquad \forall i \in V$$

But the output value $x^\omega \in \mathbb{R}$ is always a scalar!

- The corresponding gradients are also vectors of the same size

$$z^i \in \mathbb{R}^{d^i} \qquad \forall i \in V$$

- Backpropagation has exactly the same structure using the generalized chain rule

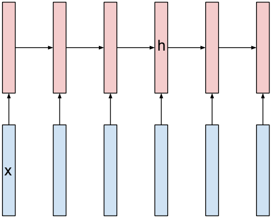
$$z^i = \sum_{j \in \textbf{ch}(i)} \underbrace{\left. \frac{\partial x^{\omega}}{\partial x^j} \right|_{x_I = a_I}}_{1 \times d^j} \times \underbrace{\left. \frac{\partial x^j}{\partial x^i} \right|_{x_I^j = a_I^j}}_{d^j \times d^i}$$

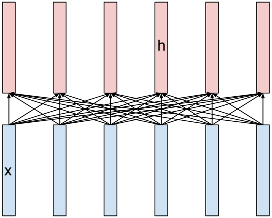
where second term is **Jacobian** of f^j wrt x^i evaluated at a_I

Agenda

1. Backpropagation
2. Self-attention in NLP
3. Representation learning through language modeling

Recurrent vs Self-Attention





$$\frac{\partial h}{\partial x}$$

Attention: General Form

Input

- $Q \in \mathbb{R}^{d \times T}$: T query vectors of the "asker"
- $K \in \mathbb{R}^{d \times T'}$: T' key vectors of the "answerer"
- $V \in \mathbb{R}^{d \times T'}$: T' value vectors of the "answerer"

Output

- $A \in \mathbb{R}^{d \times T}$: T answer vectors of the "asker" after asking

$$A = V \operatorname{softmax}\left(K^\top Q\right)$$

Example: Attention-Based Seq2Seq

Input

- $Q = Y$: target LSTM encodings
- $K = X$: source LSTM encodings
- $V = X$: source LSTM encodings

Output

- $A = \mathrm{Attention}(Y, X, X)$: new target encodings

$$A = X \mathrm{softmax}\left(X^\top Y\right)$$

Scaled Attention

Useful when d is large

$$A = V \mathrm{softmax}\left(\frac{K^\top Q}{\sqrt{d}}\right)$$

Exercise: $k, q \in \mathbb{R}^d$ elementwise independent, mean 0, variance 1

- $\mathrm{var}\left(k^\top q\right)$?
- $\mathrm{var}\left(k^\top q / \sqrt{d}\right)$?

Multi-Head Attention

Same input

Parameters

- $W_i^Q \in \mathbb{R}^{(d/H)\times d}$ for $i = 1 \ldots H$: query projectors
- $W_i^K \in \mathbb{R}^{(d/H)\times d}$ for $i = 1 \ldots H$: key projectors
- $W_i^V \in \mathbb{R}^{(d/H)\times d}$ for $i = 1 \ldots H$: value projectors
- $W \in \mathbb{R}^{d\times d}$

$$A = W \begin{bmatrix} \mathrm{Attention}\left(W_1^Q Q, W_1^K K, W_1^V V\right) \\ \vdots \\ \mathrm{Attention}\left(W_H^Q Q, W_H^K K, W_H^V V\right) \end{bmatrix}$$

Multi-Head Attention with Residual (or Skip) Connection

Plus regularization: dropout, layer normalization (Ba et al., 2016)

$$A = \mathrm{MultiHeadAttention}(Q, K, V)$$
$$A' = \mathrm{LayerNorm}\left(\mathrm{Drop}\left(A\right) + Q\right)$$

Henceforth $\mathrm{ResMHA}(Q, K, V)$

Transformer Encoder (Vaswani et al., 2017)

Using $H = 8$ heads, $l = 0 \ldots 5$

$$\widetilde{X}^{(l)} = \mathrm{ResMHA}\left(X^{(l)}, X^{(l)}, X^{(l)}\right)$$

$$X^{(l+1)} = \mathrm{ResFF}\left(\widetilde{X}^{(l)}\right)$$

$X^{(0)} = \mathrm{Drop}_{0.1}(E + \Pi)$

Transformer Decoder (Vaswani et al., 2017)

Using $H = 8$ heads, $l = 0 \dots 5$

$$\widetilde{Y}^{(l)} = \mathrm{ResMHA}\left(Y^{(l)}, Y^{(l)}, Y^{(l)}\right)$$

$$\overline{Y}^{(l)} = \mathrm{ResMHA}\left(\widetilde{Y}^{(l)}, X^{(6)}, X^{(6)}\right)$$

$$Y^{(l+1)} = \mathrm{ResFF}\left(\overline{Y}^{(l)}\right)$$

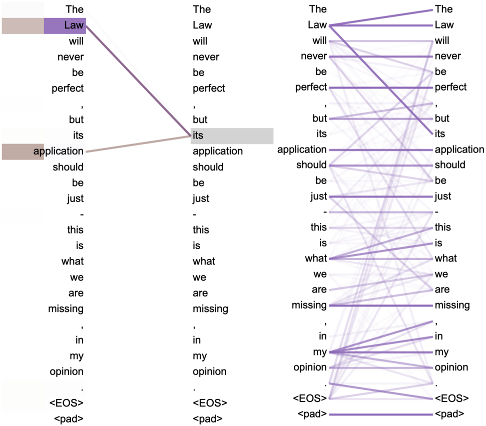
Prediction: softmax $\left(EY^{(6)}\right)$

Translation Performance (Vaswani et al., 2017)

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Self-Attention Visualization (Vaswani et al., 2017)

Layer 5 and 6, one of the "heads"



Different heads learn different weights

Agenda

1. Backpropagation
2. Self-attention in NLP
3. Representation learning through language modeling

Text Representations Through Neural Language Modeling

1. Language models can be trained on a **lot** of text (e.g., the web)
2. They yield text representations generally useful for downstream tasks

Example Downstream Tasks

Sentence classification

- Binary sentiment classification

$$\text{This film doesn't care about intelligent humor} \longrightarrow \begin{bmatrix} 0.05 \\ {\color{red}0.95} \end{bmatrix}$$

  or multi-class (e.g., 5 stars)

- Example datasets: SST-2, IMBb, Yelp Review, SemEval, CoLA
- Sentiment analysis results: http://nlpprogress.com/english/sentiment_analysis.html
- Other types of classification: grammatical vs ungrammatical (CoLA)

Example Downstream Tasks

Sentence pair classification, or natural language inference (NLI)

$$\big(\text{I am a lacto-vegetarian}, \text{I enjoy eating cheese}\big) \rightarrow \begin{bmatrix} 0.05 \\ 0.03 \\ \color{red}{0.92} \end{bmatrix}$$

Example dataset: MNLI (Williams et al., 2018)

Met my first girlfriend that way.	FACE-TO-FACE **contradiction** C C N C	I didn't meet my first girlfriend until later.
8 million in relief in the form of emergency housing.	GOVERNMENT **neutral** N N N N	The 8 million dollars for emergency housing was still not enough to solve the problem.
Now, as children tend their gardens, they have a new appreciation of their relationship to the land, their cultural heritage, and their community.	LETTERS **neutral** N N N N	All of the children love working in their gardens.
At 8:34, the Boston Center controller received a third transmission from American 11	9/11 **entailment** E E E E	The Boston Center controller got a third transmission from American 11.
I am a lacto-vegetarian.	SLATE **neutral** N N E N	I enjoy eating cheese too much to abstain from dairy.
someone else noticed it and i said well i guess that's true and it was somewhat melodious in other words it wasn't just you know it was really funny	TELEPHONE **contradiction** C C C C	No one noticed and it wasn't funny at all.

Example Downstream Tasks

SQuAD-style question answering (Rajpurkar et al., 2016)

Example dataset: SQuAD (Rajpurkar et al., 2016)

In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under **gravity**. The main forms of precipitation include drizzle, rain, sleet, snow, **graupel** and hail... Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals **within a cloud**. Short, intense periods of rain in scattered locations are called "showers".

What causes precipitation to fall?
gravity

What is another main form of precipitation besides drizzle, rain, snow, sleet and hail?
graupel

Where do water droplets collide with ice crystals to form precipitation?
within a cloud

Can be framed as predicting start/end index of the passage

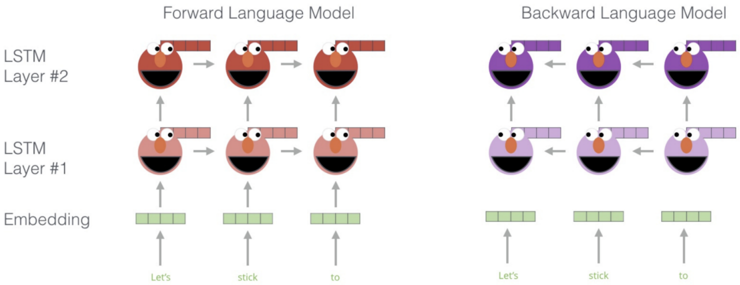
Setting

- Each such downstream task provides only a limited amount of labeled data
- Can we transfer a large-scale pretrained language model to improve performance in all these tasks simultaneously?
- Popular benchmarks (Wang et al, 2018):
  - GLUE: https://gluebenchmark.com/leaderboard
  - SuperGLUE: https://super.gluebenchmark.com/leaderboard

ELMo (Peters et al., 2018)

Trained 10 epochs on 1B Word Benchmark

https://arxiv.org/pdf/1802.05365.pdf



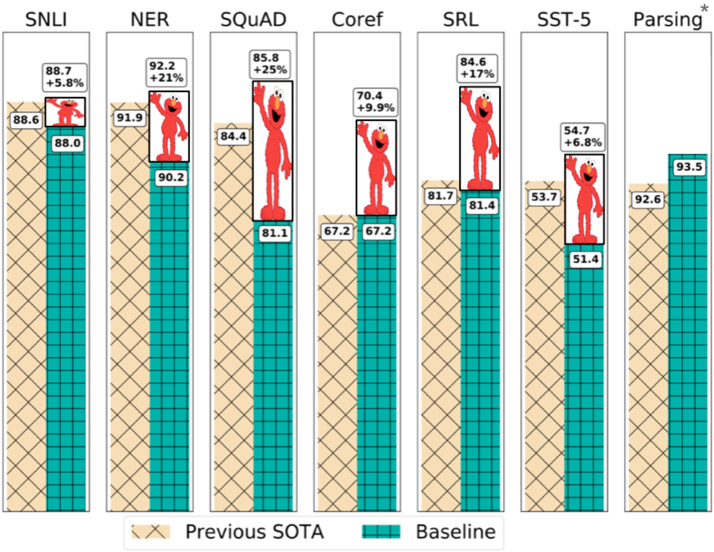
\# words in the sentence

$$\sum_{k=1}^{N} \Big(\log p(t_k \mid t_1, \ldots, t_{k-1}; \Theta_x, \overrightarrow{\Theta}_{LSTM}, \Theta_s) + \log p(t_k \mid t_{k+1}, \ldots, t_N; \Theta_x, \overleftarrow{\Theta}_{LSTM}, \Theta_s) \Big)$$

input

softmax

ELMo (Peters et al., 2018)



*Kitaev and Klein, ACL 2018 (see also Joshi et al., ACL 2018)

ELMo in Practice

1. ELMo layer: new representation of i-th token in a sequence

$$\mathrm{ELMo}_i(\gamma, s_1 \ldots s_L) = \gamma \sum_{l=0}^{L} s_l \begin{cases} e_i^{\mathrm{ELMo}} & \text{if } l = 0 \\ \begin{bmatrix} \overrightarrow{h}_i^{\mathrm{ELMo},l} \\ \overleftarrow{h}_i^{\mathrm{ELMo},l} \end{bmatrix} & \text{otherwise} \end{cases}$$

2. In your downstream task, concatenate $\mathrm{ELMo}_i(\gamma, s_1 \ldots s_L)$ to your i-th input embedding.

3. Train your original model **AND** $\gamma, s_1 \ldots s_L$ while keeping ELMo parameters fixed

Using ELMo

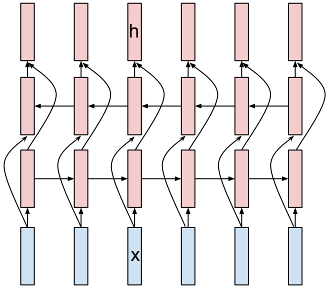
https://allennlp.org/elmo

Pre-trained ELMo Models

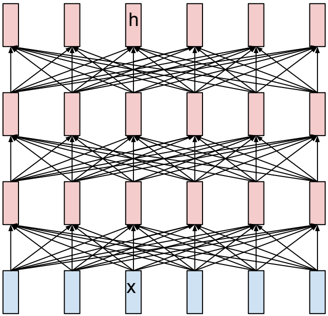
Model	Link(Weights/Options File)		# Parameters (Millions)	LSTM Hidden Size/Output size	# Highway Layers>
Small	weights	options	13.6	1024/128	1
Medium	weights	options	28.0	2048/256	1
Original	weights	options	93.6	4096/512	2
Original (5.5B)	weights	options	93.6	4096/512	2

```python
from allennlp.modules.elmo import Elmo, batch_to_ids

options_file = "https://allennlp.s3.amazonaws.com/models/elmo/2x409
weight_file = "https://allennlp.s3.amazonaws.com/models/elmo/2x4096

# Compute two different representation for each token.
# Each representation is a linear weighted combination for the
# 3 layers in ELMo (i.e., charcnn, the outputs of the two BiLSTM))
elmo = Elmo(options_file, weight_file, 2, dropout=0)

# use batch_to_ids to convert sentences to character ids
sentences = [['First', 'sentence', '.'], ['Another', '.']]
character_ids = batch_to_ids(sentences)

embeddings = elmo(character_ids)
```

Recurrent vs Self-Attention Encoding



not bidirectional until later



deeply bidirectional

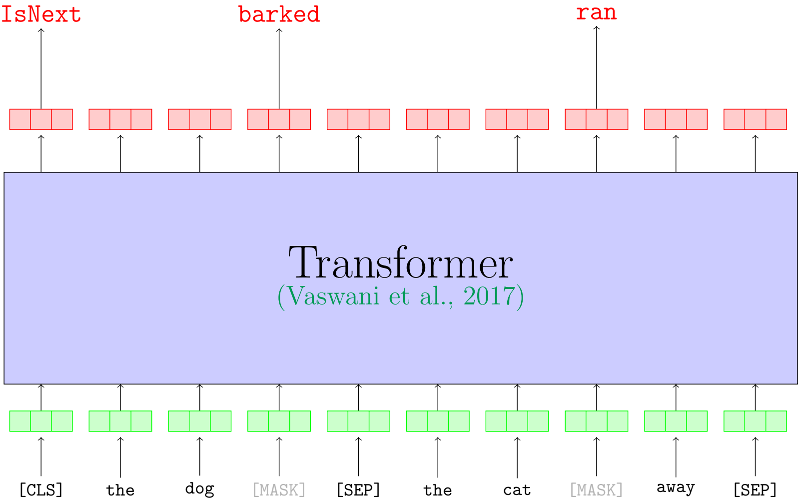
Masked Language Modeling

- **For the purposes of representation learning**, we don't care about defining a proper language model which only conditions on previous history.
- We want a prediction problem which conditions on entire context all the time, so that we can use deeply bidirectional encoders
- Solution: mask out words at random

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

- Need to be careful
  - Too little masking: too expensive to train
  - Too much masking: not enough context
  - Test time: no [MASK] input, so training should also handle no [MASK] input sometimes
  - Details: https://arxiv.org/pdf/1810.04805.pdf

BERT (Devlin et al., 2019)



BERT (Devlin et al., 2019)

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	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Number of parameters

- ELMo: 94 million
- BERT Base: 110 million
- BERT Large: 340 million

RoBERTa (Liu et al., 2019)

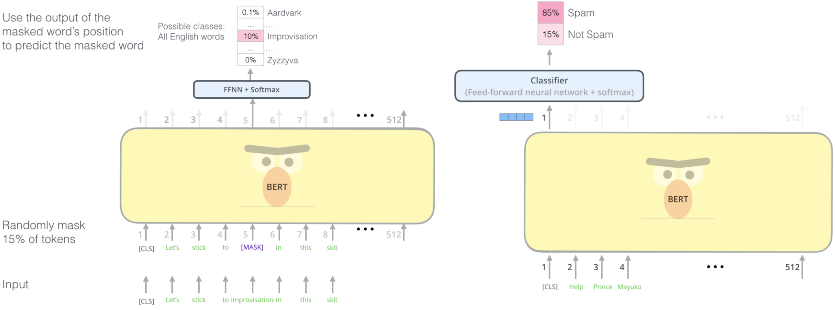
System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Average
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
$\text{BERT}_{\text{BASE}}$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

Model	data	bsz	steps	SQuAD (v1.1/2.0)	MNLI-m	SST-2
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	**94.6/89.4**	**90.2**	**96.4**
$\text{BERT}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7
$\text{XLNet}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	94.0/87.8	88.4	94.4
+ additional data	126GB	2K	500K	94.5/88.8	89.8	95.6

RoBERTa = BERT + more careful training + more data

https://github.com/pytorch/fairseq/tree/master/examples/roberta

BERT Manual

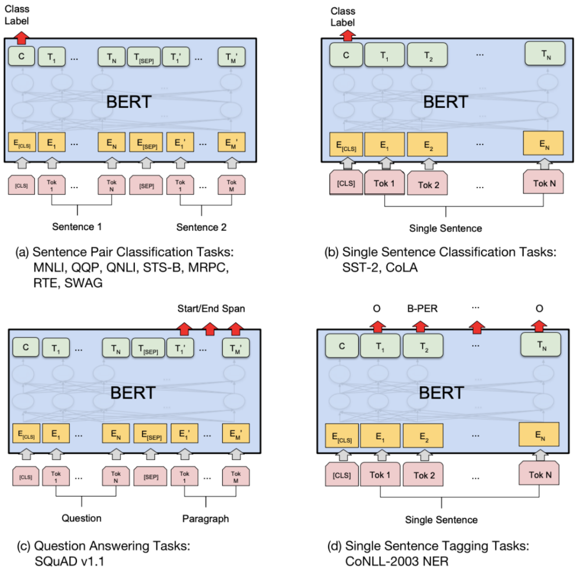


Pre-training

Fine-tuning

Critical difference from ELMo: **all BERT weights** are fine-tuned for the target task (expensive but worth it)

BERT Applications



Currently in NLP

Explosion of pretrained contextualized word embedding models

- TagLM (Peters et, 2017)
- CoVe (McCann et al. 2017)
- ULMfit (Howard and Ruder, 2018)
- ELMo (Peters et al, 2018)
- OpenAI GPT (Radford et al, 2018)
- BERT (Devlin et al, 2018)
- OpenAI GPT-2 (Radford et al, 2019)
- XLNet (Yang et al, 2019)
- SpanBERT (Joshi et al, 2019)
- RoBERTa (Liu et al, 2019)
- AlBERT (Anonymous)
- T5 (Raffel et al., 2019)
- ...