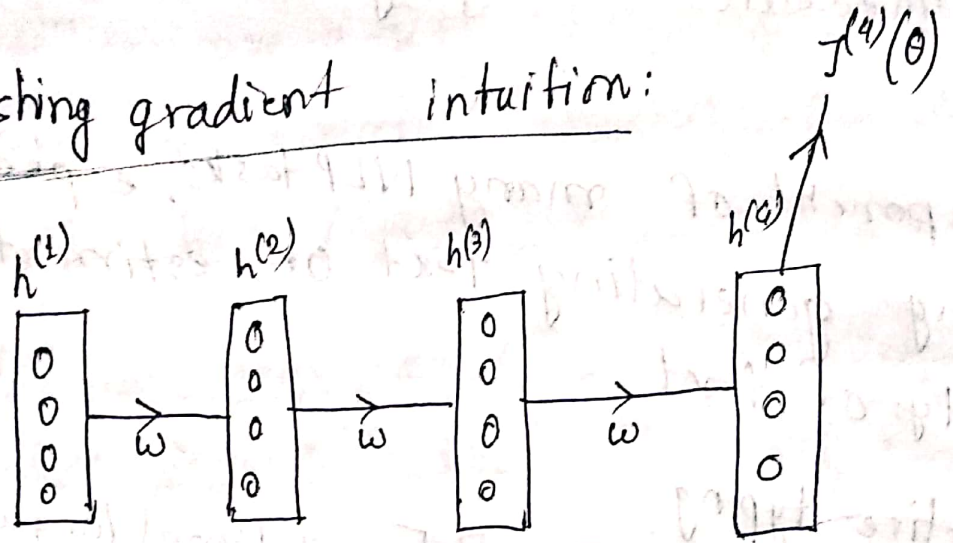
Lecture 7: Vanishing Gradients, Fancy RNNs

Vanishing gradient intuition:



$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial h^{(3)}}{\partial h^{(2)}} \times \frac{\partial h^{(4)}}{\partial h^{(3)}} \times \frac{\partial J^{(4)}}{\partial h^{(4)}}$$

what happens if these are small? → Vanishing gradient problem

Vanishing gradient problem: When these are small, the gradient signal gets smaller and smaller as it backpropagates further

$$h^{(t)} = \sigma\left(W_h h^{(t-1)} + W_x x^{(t)} + b_1\right)$$

$$\frac{\partial h^{(t)}}{\partial h^{(t-1)}} = \text{diag}\left(\sigma'\left(W_h h^{(t-1)} + W_x x^{(t)} + b_1\right)\right) W_h$$

$$\frac{\partial J^{(i)}(\theta)}{\partial h^{(j)}} = \frac{\partial J^{(i)}(\theta)}{\partial h^{(i)}} \prod_{j<t\leq i} \frac{\partial h^{(t)}}{\partial h^{(t-1)}}$$

$$= \frac{\partial J^{(i)}(\theta)}{\partial h^{(i)}} \underbrace{W_h^{(i-j)}} \prod_{j<t\leq i} \text{diag}\left(\sigma'\left(W_h h^{(t-1)} + W_x x^{(t)} + b_1\right)\right)$$

↑

If W_h is small, then this term get • vanishingly small as i and j get further apart.

Vanishing gradient proof sketch: (cont) Considering matrix L2 norm

$$\left\| \frac{\partial J^{(i)}(\theta)}{\partial h^{(j)}} \right\| \leq \left\| \frac{\partial J^{(i)}(\theta)}{\partial h^{(i)}} \right\| \|W_h\|^{(i-j)} \prod_{j<t\leq i} \|\text{diag}\left(\sigma'\left(W_h h^{(t-1)} + W_x x^{(t)} + b_1\right)\right)\|$$

→ Pascanu et al showed that if the largest eigenvalue of W_h is less than 1, then the gradient $\left\| \frac{\partial J^{(i)}(\theta)}{\partial h^{(j)}} \right\|$ will shrink exponentially.

→ Here the baund is 1 because we have sigmoid non-linearity

* Vanishing gradient is problematic because we will lose the long-term effect for this

Why is vanishing gradient a problem?

Another explanation

Gradient can be viewed as a measure of the effects of the past on the future.

⇒ If the gradient becomes vanishingly small over longer distances (step t to step $t+n$), then we can't tell whether:

i) There is no dependency between step t and step $t+n$ in the data.

ii) We have wrong parameters to capture the true dependency between t and $t+n$.

Due to vanishing gradient, RNN-LMs are better at learning from sequential recency than syntactical recency. ~~so they make~~

Why is exploding gradient a problem?

→ If the gradient becomes too big, then the SGD update step becomes too big

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$$

Gradient clipping: if the norm of the gradient is greater than some threshold, scale it down before apply SGD update

$$\hat{g} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$$

if $\|\hat{g}\| \geq$ threshold then

$$\hat{g} \leftarrow \frac{\text{threshold}}{\|\hat{g}\|} \hat{g}$$

end if

Intuition: take a step in the same direction, but a smaller step

How to fix vanishing gradient problem?

LSTM

Long short Term Memory (LSTM)

Hochreiter and Schmidhuber 1997

→ proposed as a solution to vanishing gradient problems

→ On step t, there is a hidden state $h^{(t)}$ and a cell state $c^{(t)}$

- Both are vectors length n
- The cell stores long-term information
- LSTM can erase, write and read information from the cell

The selection of which information is erased/written/read is controlled by three corresponding gates:

→ The gates are also vectors length n

→ On each timestep, each elements of the gate can be open (1), closed (0) or somewhere between

→ The gates are dynamic: their value is computed based on the current context

Forget gate: controlls what is kept vs forgotten

from previous cell state

$$f^{(t)} = \sigma\left(W_f h^{(t-1)} + U_f x^{(t)} + b_f\right)$$

$$i^{(t)} = \sigma\left(W_i h^{(t-1)} + U_i x^{(t)} + b_i\right)$$

$$o^{(t)} = \sigma\left(W_o h^{(t-1)} + U_o x^{(t)} + b_o\right)$$

Imput gate: controlls what parts of the new cell content are written to cell

Output Gate: controls what parts of cells are output to hidden state

New cell content: the new content to be written to the cell

$$\tilde{c}^{(t)} = \tanh\left(W_c h^{(t-1)} + U_c x^{(t)} + b_c\right)$$

Cell state: erase ("forget") some content from last cell state and write ("input") some new cell content

$$c^{(t)} = f^{(t)} \circ c^{(t-1)} + i^{(t)} \circ \tilde{c}^{(t)}$$

Hidden state: read ("output") some content from the cell

$$h^{(t)} = o^{(t)} \circ \tanh c^{(t)}$$

→ LSTM architecture makes it easier for the RNN to preserve information over many timesteps

eg. if the forget gate is set to remember everything on every timestep, then the info in the cell is preserved indefinitely

By contrast, it's harder for vanilla RNN to learn a recurrent weight matrix W_h that preserves info in hidden states.

→ However LSTM doesn't provide the ~~gaur~~ guarantee that there is no vanishing/exploding gradient, but it does provide an easier way for the model to learn long-distance dependencies.

→ In 2013-2015, LSTMs started achieving SOTA

..→ Successful tasks include: handwriting recognition, speech recognition, machine translation, parsing, image captioning.

→ LSTM became the dominant approach

Now, othe approaches ilike Transformers have become more dominant.

Gate Recurrent Units (GRU)

→ Proposed by Cho$ et. al. in 2014 as a simpler alternative to the LSTM

→ On each time step t we have input $x^{(t)}$ and hidden state $h^{(t)}$ (no cell state)

$$\Rightarrow u^{(t)} = \sigma\left(W_u h^{(t-1)} + U_u x^{(t)} + b_u\right)$$

$$\Rightarrow r^{(t)} = \sigma\left(W_r h^{(t-1)} + U_r x^{(t)} + b_r\right)$$

↓

Reset gate: controls what parts of previous hidden state are used to compute new content

→ **Update gate:** controls what parts of hidden state are updated vs preserved [plays the role of forget and input gate]

$$\tilde{h}^{(t)} = \tanh\left(W_h\left(r^{(t)} \circ h^{(t-1)}\right) + U_h x^{(t)} + b_h\right)$$

$$h^{(t)} = \tanh\left(1 - u^{(t)}\right) \circ h^{(t-1)} + u^{(t)} \circ \tilde{h}^{(t)}$$

↓

New hidden state content: reset gate selects useful parts of previous hidden state. Uses this and current input to compute new hidden state

→ **Hidden state:** (update gate) simultaneously controls what is kept from previous hidden state, and what is updated to new hidden state content

Like LSTM, GRU makes it easier to retain info long term (by setting update to 0)

LSTM vs GRU:

→ GRU is quicker to compute and has fewer parameters.

→ There is no conclusive evidence that one performs better than other

→ LSTM is a good default choice in practice

Rule of thumb: Start with LSTM, but switch to GRU if you want something more efficient

* Vanishing/exploding gradient is not just an RNN problem.

- Lots of new deep feedforward/convolutional architectures that add more direct connections (for example: Residual connections (skip connections))

- Dense connections, aka ~~Peeng~~ DenseNet

Highway connections:

- Similar to ~~resd~~ residual connections but the identity connections vs the transformation layer is controlled by a dynamic gate.

Conclusion: Though vanishing/exploding gradients are a general problem, RNNs are particularly unstable due to the repeated multiplication by the same weight matrix. [Bengio, 1994]

Bidirectional RNNs

forward RNN -> provides info. of left context
backward RNN -> provides info of right context

-> are only applicable if you have access to the entire sequence

↳ Not applicable to Language modeling.

BERT (Bidirectional Encoder Representations from Transformers) is a powerful pretrained contextual representation system built on bidirectionally.

Multi-layer RNNs (stacked RNNs)

→ This allows the network to compute more complex representations

- The lower RNNs should compute lower level features and higher RNNs should compute higher-level features

In practice

→ High performing RNNs are often multi-layer

→ In a 2017 paper, Britz et al find that for NMT, 2 to 4 layers are best for the encoder RNN and 4 layers are best for decoder RNN

→ Transformer-based network are frequently deeped like 12 to 24 layers. Transformers have alot of skipping-like connections

Lecture 8: Translation, Seq2Seq, Attention

Section 1: Pre-Neural Machine Translation

Machine Translation is the task of translating a sentence x ~~from~~ from one language (the source language) to a sentence y in another language (the target ~~sen~~ language)

Research began in the early 1950s

- Russian → English (motivated by the Cold War)
- Those systems were mostly rule-based, using bilingual dictionary to map Russian word to their English counterparts.

1990s-2010s: Statistical Machine Translation:

Core idea: Learn a probabilistic model from data

→ We want to find best English sentence y, given french sentence x

$$\text{argmax}_y \, P(y|x)$$

→ Using Bayes Rule to break this down to two components: $\text{argmax}_y \, P(x|y) \, P(y)$

$\text{argmax}_y \; P(x|y) \cdot P(y)$

Translation Model

Models how words and phrases should be translated (fidelity). Learnt from parallel data.

Language model

Models how to write good English (fluency) Learnt from monolingual data

How to learn translation model $P(x|y)$?

→ First, we need large amount of parallel data

Break that down further,

$P(x, a|y)$

where a is the alignment, i.e. word level correspondence between French x and English sentence y

What is alignment?

Alignment is the correspondence between particular words in the translated sentence pair.

Alignment can be complex

- one to one
- many to one
- one to many (fertile)
- many to many

Learning alignment for SMT

We learn $P(x, a|y)$ as a combination of many factors, including:

→ Probability of particular words aligning

→ Probability of particular words having particular fertility

- Alignments a are latent variables: They aren't explicitly specified in the data.

→ Require the use of special learning algos like Expectation Maximization for learning the params of distributions with latent variables

Decoding for SMT

→ We could enumerate every possible y and calculate the probability ⇒ Too expensive

Answer: Impose strong independence assumptions in model use dynamic programming for globally optimization solutions (Viterbi Algorithm)

Viterbi: Decoding with Dynamic Programming

$$P(x, a|y) = \prod_{j=1}^{|x|} P(x_j | f_a(j))$$

SMT was a huge field upto 2010s

The best systems were extremely complex

→ systems had many seperately-designed sub components

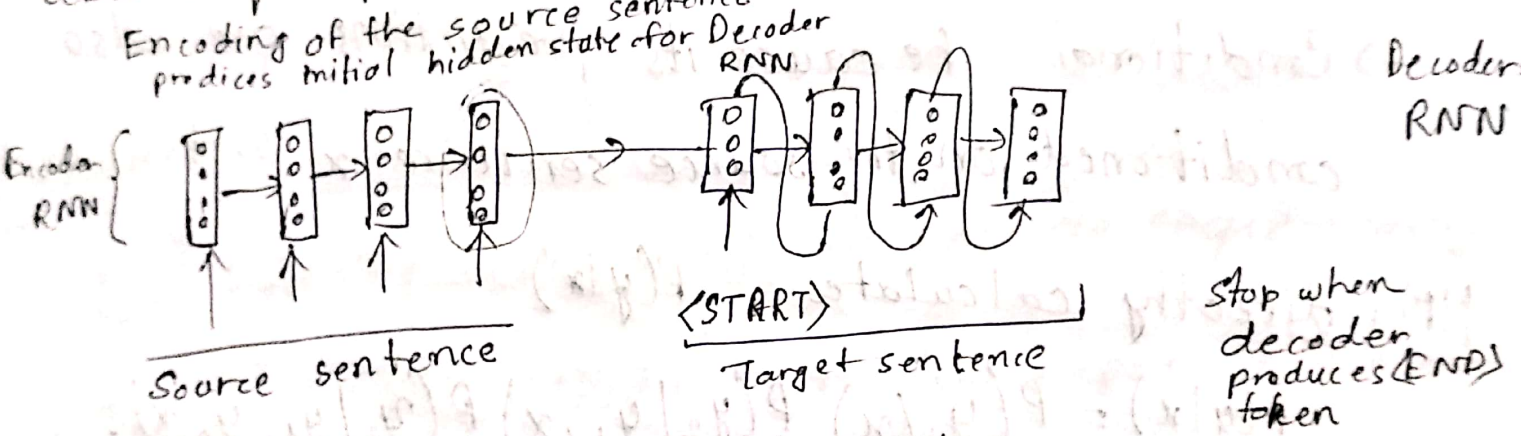
→ lots of feature engineering

→ Requires compiling and maintaining extra resources

→ Lots of human effort to maintain

Section 9: Neural Machine Translation.

- Neural Machine Translation (NMT) is a way to do Machine translation with a single neural network.
- The neural architecture is called sequence to sequence aka seq2seq and it involves two RNNs



Encoder RNN produces an encoding of the source sentence

~~Decoder RNN is~~ Decoder RNN is a language mode that generates target sentence, conditioned on encoding

* This architecture is versatile
  - Summarization (long text → short text)
  - Dialogue (previous utterances → next utterance)
  - Parsing (input text → output parse as sequence)
  - Code generation (natural language → Python code)

• The sequence to sequence is an example of a Conditional Language Model

→ Language model because the decoder is predicting the next word of the target sentence y.

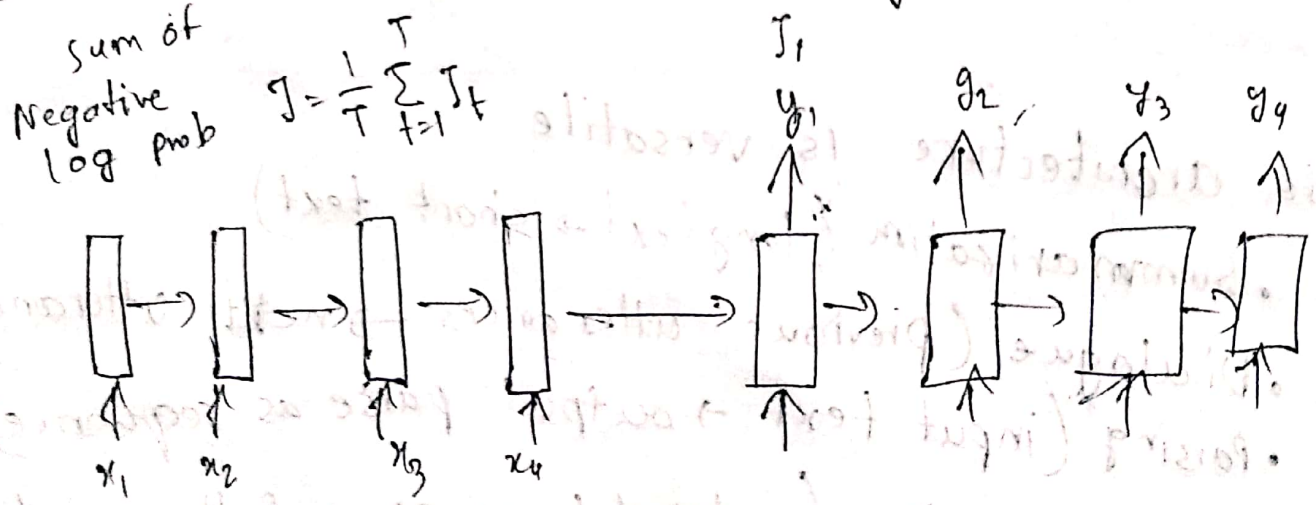
→ Conditional because its predictions are also conditioned on the source sentence x

NMT directly calculates $P(y|x)$

$$P(y|x) = P(y_1|x)\, P(y_2|y_1, x)\, P(y_3|y_1, y_2, x) \ldots P(y_T|y_1, \ldots y_{T-1}, x)$$

~~Question:~~

⇒ To train NMT we need a big parallel corpus.



- Seq2seq is optimized as a single system. Backprop operates "end-to-end"

There are many ways of decoding

* Greedy decoding (take most probable word on each step)
  Cons → no way to undo it

* Exhaustive search decoding:

We can try computing all possible sequences y
This $O(V^T)$ complexity is far too expensive

Beam search decoding:

On each step of decoder, keep track of the k-most probable partial transformations (hypotheses)
- k is the beam size (5-10)
- A hypothesis $y_1, \ldots, y_t$ has a score which is its log probability

$$\text{score}(y_1, \ldots, y_t) = \log P_{LM}(y_1, \ldots, y_t | x)$$

$$= \sum_{i=1}^{t} \log P_{LM}(y_i | y_1, \ldots, y_{i-1}, x)$$

- Scores are all negative and higher score is better
- We search for high-scoring hypothesis, ~~tracking~~ top k on each step

→ Beam search is not guaranteed to find optimal solution. But much more efficient than exhaustive search

Stopping criterion:

- In greedy decoding, usually we decode until the model produces a <END> token

→ In beam search decoding, different hypothesis may produce <END> tokens in different time step.

* When a hypothesis produces <END>, that hypothesis is complete

* Place it aside and continue exploring other hypothesis via beam search

→ Usually we continue beam search until:

- we reach timestep T
- We have at least n completed hypotheses

[T and n are predefined cutoffs]

Finishing up:

We have our list of completed hypotheses

Each hypothesis $y_1 \dots y_t$ on our list has a score

$$\text{score}(y_1, \dots y_t) = \log P_{LM}(y_1, \dots, y_t | x)$$

$$= \sum_{i=1}^{t} \log P_{LM}(y_i | y_1, \dots, y_{i-1}, x)$$

Problem ~~Cons~~ → longer hypotheses have lower scores

Fix: Normalize by length, Use this to select top one instead

$$\frac{1}{t} \sum_{i=1}^{t} \log P_{LM}(y_i | y_1, \dots, y_{i-1}, x)$$

Advantages of NMT

Compared to SMT

→ Better performance
- ~~Better p~~ More fluent
- Better use of context
- Better use of phrase similarities

→ A single neural network to be optimized end-to-end
- No subcomponents to be individually optimized

→ Requires much less human engineering effort
- No feature engineering
- Same method for all language pairs

Disadvantages

- less interpretable
- difficult to control
  → can't easily specify rules or guidelines for translation
  → Safety concerns

Evaluation:

BLEU (Bilingual Evaluation Understudy)

- compares the machine-generated translation to one or several human-written translation and compute a similarity score based on
  - → n-gram precision [usually for 1, 2, 3, 4-grams]
  - → plus a penalty for too-short system translations

★ BLEU is useful but imperfect

- There are many valid ways to translate a sentence
- So a good translation can get a poor BLEU score because it has low n-gram overlap with the human translation

NMT went from a fringe research activity in 2014 to the leading standard in 2016

- 2014: first seq2seq paper published
- 2016: Google Translate switches from SMT to NMT

→ Machine Translation is not get solved.

Difficulties

- Out-of-vocabulary words
- Domain mismatch between train and test data
- Maintaining context over longer text
- Low resource language pairs
- Using common sense is still hard
- Idioms are difficult to translate
- picks up biases in training data
- uninterpretable systems do strange things.

Section 3: Attention

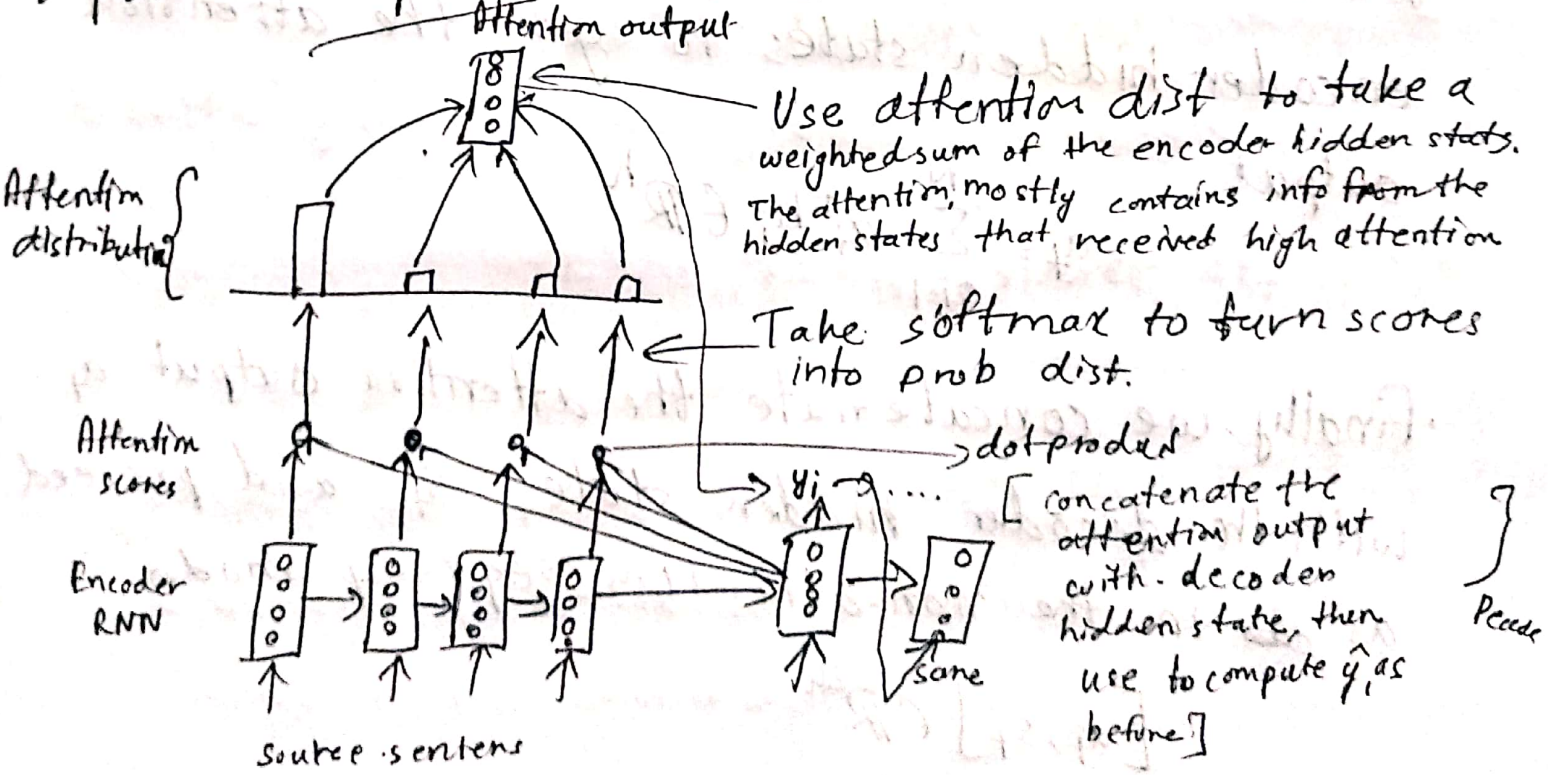
Sequence-to-sequence: the bottleneck

Encoding of the source sentence needs to capture all infos about the source sentence

~~Info bottlen~~ Info bottleneck

Attention provides solution to the bottleneck problem

→ Core idea: on each step of the decoder, use direct connection to the encoder to focus on a particular part of the source sequence



In equations

- We have encoder hidden states $h_1, \ldots h_N \in \mathbb{R}^h$
- On timestep t, we have decoder hidden state $s_t \in \mathbb{R}^h$
- We get the attention scores e^t for this step:

$$e^t = [s_t^T h_1, \ldots, s_t^T h_N] \in \mathbb{R}^N$$

- We take softmax to get the attention distribution α^t for this step (this is a prob dist and sums to 1)

$$\alpha^t = \text{softmax}(e^t) \in \mathbb{R}^N$$

- We use α^t to take weighted sum of the encoder hidden states to get the attention output

$$a_t = \sum_{i=1}^{N} \alpha_i^t h_i \in \mathbb{R}^h$$

- Finally, we concatenate the attention output a_t with the decoder hidden state s_t and proceed as in the non-attention seq2seq model

$$[a_t; s_t] \in \mathbb{R}^{2h}$$

Attention "is great"

- Attention significantly improves NMT performance
  - → It's very useful to allow to focus on certain parts of the source
- Attention solves bottleneck problem
  - → allows decoder look directly at source; bypass bottleneck
- Attention helps with vanishing gradient problem
  - → Provides shortcut to faraway states
- Attention provides some interpretability
  - → By inspecting attention distribution, we can see what the decoder was focusing on
  - → We get (soft) alignment for free
  - → This is cool because we never explicitly trained an alignment system
  - → The network just learned alignment by itself

Attention is general deep learning technique

We can use attention in many architectures and many tasks.

Attentions

Given a set of vector values and a vector query, attention is a technique to compute a weighted sum of the values, dependent on the query.

→ query attend to the values

→ For example, in the seq2seq + attention model, each decoder hidden state (query) attends to all the encoder hidden states (values)

Intuition:

The weighted sum is a selective summary of the information contained in the values, where the query determines which values to focus on.

- Attention is a way to obtain a fixed size representation of an arbitrary set of representations (the values), dependent on some other representation (the query)

There are several attention variants

- We have some values $h_1 \dots h_N \in \mathbb{R}^{d_1}$ and a $c_s \in \mathbb{R}^{d_2}$
- Attention always involves

1) Computing the attention scores $e \in \mathbb{R}^N$

2) Taking softmax to get attention distribution

$$\alpha = \text{softmax}(e) \in \mathbb{R}^N$$

3) Using attention distribution to take weighted sum of values

$$a = \sum_{i=1}^{N} \alpha_i h_i \in \mathbb{R}^{d_1}$$

thus obtaining the attention output a (sometimes called context vector)

Attention ~~variation~~ variants

- Basic dot product

$$e_i = s^T h_i \in \mathbb{R}$$

- Multiplicative attention

$$e_i = s^T W h_i \in \mathbb{R}$$

where $W \in \mathbb{R}^{d_2 \times d_1}$ is a weight matrix.

- ~~Additive~~ Additive attention

$$e_i = v^T \tanh(W_1 h_i + W_2 s) \in \mathbb{R}$$

→ where $W_1 \in \mathbb{R}^{d_3 \times d_1}$ $W_2 \in \mathbb{R}^{d_3 \times d_2}$ are weight matrices

$v \in \mathbb{R}^{d_3}$ is a weight vector

→ d_3 (the attention dimensionality) is a hyperparameter.

Lecture 9: Practical Tips for Project

Two basic starting points, for all of science

[Nails] Start with a (domain) problem of interest and try to find good/better ways to address it than are currently known/used.

[Hammer] Start with a technical approach of interest, and work out good ways to extend or improve it or new ways to apply it.

Herb Simon: "If you see a research area where many people are working, go somewhere else"

Wayne Gretzky: "I skate to where the puck is going, not where it has been"

Autoregressive language model (AR

Kind of model that ~~using~~ uses the context word to predict the next word. But here the context word is contrained to two directions either forward or backward

Autoencoder language model

Aims to reconstruct the original data from corrupted input. Context of both sides

Disadvantages

- It uses [MASK] in the pretraining, ~~but this~~

In XL-Net

Language model consists of two phases, the pre-train phase and fine-tune phase. XL-Net focus on pre-train phase. In the pretrain phase, it proposed a new objective called Permutation Language Modeling

Intuitively, what happens with RNNs?

1) Measure the influence of the past on the future

$$\frac{\partial \log p(x_{t+n} | x_{1:t+n})}{\partial h_t} =$$

5) The large output vocabulary proble in NMT (or all NLG)

- Word generation problem
  - Vocabs used are usually modest : 50K

Possible approach for output

- Hienarcial softmax → tree structure vocab
- Noise constrastive estimation → binary classification

→ Train on a subset of the vocab at a time.
test on a smart on the set of possible translations

→ Use attention to work out what you are translating
→ we can do something simple like dictimary lookup

BLE4 formula (counts n-grams up to length 4)

$$\exp\left(0.5 * \log p_1 + 0.25 \log p_2 + 0.125 \log p_3 + 0.125 \log p_4 - \max\left(\text{words in reference} / \text{words in machine} - 1.0\right)\right)$$

P = precision

BLEU is a weighted geometric mean, with a brevity penalty factor added.

- Initial results showed that BLEU predicts human judgement well.

Automatic evaluation of MT

- People started optimizing their systems to maximize BLEU score
  - → BLEU scores improved rapidly
  - → The correlation between BLEU and human judgements of quality went way down.
  - → MT BLEU scores now approach those of human translations but their true quality remains far below human translations

- Coming up with automatic evaluations has become its own research field.

There are many proposals: TER, METEOR, Maxsim, SEPIA, RTE-MT

TERpA is a representative good one that handles some word choices

* MT research requires some automatic metric to allow a rapid development and evaluation cycle.

Steps of Doing project

1) Define Task
2) Define Dataset
   - Search for academic datasets
   - Define your own data (harder, need new baselines)

3) Dataset hygiene

- Right at the beginning seperate of dev-test and test splits

4) Define metric

for Summarization [ROUGE - Recall Oriented Understudy for Gisting Evaluation]

5) Establish baseline

- Implement simplest model
- Compute metrics on train and dev
- Analyze errors
- If metrics are amazing and no error
  → Done! Problem was too easy. Need to restart

6) Implement existing neural net

- Compute metric on train and dev
- Analyze outputs and errors
- Minimum bar for this data

7) Always be close to your data! (Except for the final test set)

- Visualize the dataset
- Collect summary stats
- Look at errors
- Analyze how different hyper parans affect performance

8) Try out different models. and model variants

Aim to iterate quickly via having a good experimental setup

- Fixed window neural net
- Recurrent neural network
- Recursive neural network
- Convolutional neural network
- Attention-based model

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 runs only when development is done
- Splits like this presuppose a fairly large dataset
- If there is no dev set or you want a seperate tune set, then you create one by splitting training data though you have to weight its size/usefulness against the reduction in train-set size.
- Having a fixed test set ensures that all systems are assesed against the same gold standard.
- 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, model overfits to what you are training on

→ The model correctly describes what happened to occur in particular data we trained on, but the patterns are not general enough patterns to be likely to apply to new data.

- The way to monitor and avoid overfitting is using independent validation and test set.

- Only at the end, you evaluated and present final numbers on a test set ideally only once.

Getting your NN to train

- Positive attitude
  - NNs want to learn
    → If the net isn't learning, we'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 ~~well~~ at all or to not learn very well
    → Debugging might take more time than just implementing

- Hard to work out but experience, experimental care and rules of thumb ~~tea~~ help.

- Learning rate and initialization

Experimental Strategy:

- Work incrementally
- Start with a very simple model and get it to work
- 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
  - → bugs can be noticed easily
  - → 8 examples is good
  - → Make sure you can get 100% on this
    - Otherwise your model is definitely either not powerful enought or it is broken

- Run your model on a large dataset
  - → It should still close to 100% on the training data after optimization
  - Otherwise, you probably want to consider a more powerful model
  - Overfitting to train 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 train data
- We still need good generalization performance
  - Regularize your model until it doesn't overfit on dev data
    - L2 regularization
    - Generous dropout.

Some General Text Preprocessing Steps (Not all should be done in all cases)

1) Remove HTML Tags
2) Remove extra whitespaces
3) Convert accented characters to ASCII characters
4) Expand contradictions
5) Remove special characters
6) Lowercase all text
7) Convert numbers words to numeri form
8) Remove numbers
9) Remove stopwords
10) Lemmatization

Lecture 10: Question answering

Project Writeup

Abstract/Introduction

Prior related work

Model Model

Data

Experiments

Results

Analysis & Conclusion

Interesting thing
Google's Knowledge Graph

Question Answering:

1) Finding documents that (might) contain an answer [tradional info retrieval techniques are used]
2) finding an answer in a paragraph or a document [Reading Comprehension Problem]

A Brief History of Reading ~~Que~~ Comprehension

i) Schank, Abelson, Lehnert, et. al. 1977 - "Yale AI project"

ii) Revived by Lynette Hirschman in 1999

iii) Revived again by Chris Burges in 2013 with MC Test [Machine Comprehension]

iv) Floodgates opened in 2015/16 with the production of large datasets which permit supervised neural systems to be built

→ ~~Herman~~ Rajpukar et.al. (EMNLP 2016) SQuAD

→ Hermann et. al. (NIPS 2015) Deepmind CNN/DM dataset

A Brief History of Open-domain Question Answering

- Simmon's et.al. (1964) did first exploration on answering from an expository text based on matching ~~&~~ dependency parses
- Maron (1993) aimed to answer questions over an online encyclopedia using IR and shallow linguistic processing

- The NIST TREC QA track began in 1999 first rigorously investigated answering fact questions over a large collection of documents.
- IBM's Jeopardy! System (DeepQA, 2011) brought attention to a version of the problem; it used an ensemble of many methods.
- DrQA (Chen et.al. 2016) uses IR followed by neural reading comprehension to bring learning to Open-domain QA

Turn-of-the Millennium full NLP QA: [factoid QA]

Complex systems but they did work fairly well on "factoid" questions. [2003]

Factoid QA → answer is an entity

Evaluation

Exact Match

F1

SQuAD2.0

- All questions have an answer in the paragraph
- Systems (implicitly) rank candidates and choose the best one
- In SQuAD 2.0, 1/3 of the training questions have no answers and about 1/2 of the dev/test ques have no answers

Simplest system approach to SQuAD 2.0

- Have a threshold score for whether a span answers a question
- NLI (Natural Language Inference)
- Good systems are great, but still basic NLU errors

SQuAD limitations

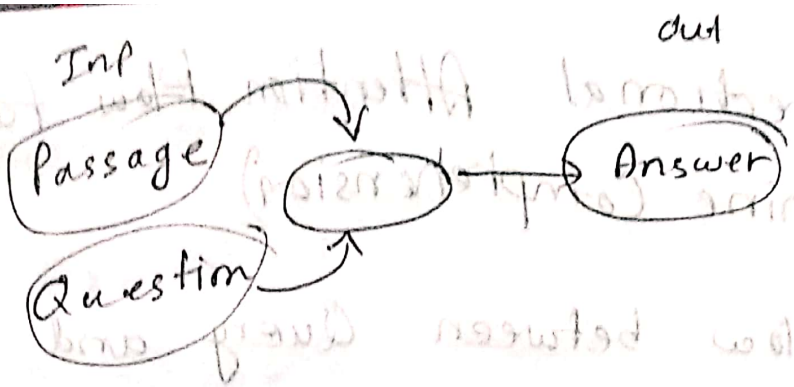
- only span-based answers (no, yes/no, counting, implicit, why)

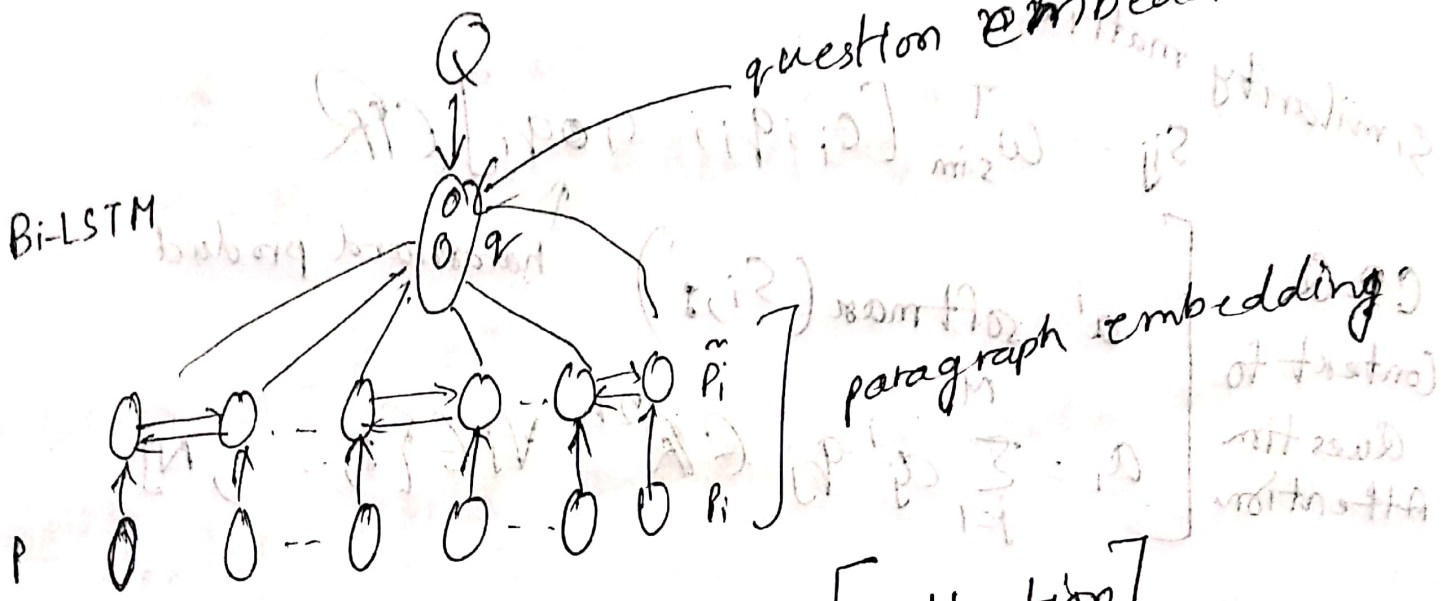
- Questions are constructed looking at the passages
  - → Not genuine info needs
  - → Generally greater lexical and syntactical matching between questions and answers spans than you get in real life
- Barely any multi-fact/sentence inference beyond coreference.

→ Nevertheless, it is well-targeted, well-structured and clean dataset
- most used and competed on QA data
- useful starting point for building systems in industry

Stanford Attentive Reader [DrQA]

- Demonstrated a minimal, highly successful architecture for reading comprehension and question answering





(start) $a_i = \text{softmax}(q^T W_s \tilde{P}_i)$ [Attention]

(end) $a_i' = \text{softmax}(q^T W_s' \tilde{P}_i)$

Stanford attentive reader

$q = \sum_j b_j q_j$ $\quad b_j = \frac{\exp(w \cdot q_j)}{\exp(w \cdot q_j)}$

[takes all states]

More deep and complicated embeddings

BiDAF (BiDirectional Attention Flow for Machine Comprehension)

→ Attention Flow between Query and Passage

Similarity matrix

$$S_{ij} = w_{sim}^T [c_i ; q_j ; c_i \circ q_j] \in \mathbb{R}$$

↑ harasmard product

C2Q Context to Question Attention

$$\begin{bmatrix} a^i = \text{softmax}(S_{i,:}) \\ a_i = \sum_{j=1}^{M} a_j^i q_j \in \mathbb{R}^{2h} \quad \forall i \in \{1, \ldots, N\} \end{bmatrix}$$

Q2C

$$\begin{bmatrix} m_i = \max_j S_{ij} \in \mathbb{R} \quad \forall i \{1, \ldots, N\} \\ \beta = \text{softmax}(m) \in \mathbb{R}^N \\ c' = \sum_{i=1}^{N} \beta_i c_i \in \mathbb{R}^{2h} \end{bmatrix}$$

→ for each position, output of BiDAF layer is

$$b_i = [c_i ; a_i ; c_i \circ a_i ; c_i \circ c'] \in \mathbb{R}^{8h}$$

FusionNet

$$S_{ij} = s^T \tanh(w_1 c_i + w_2 q_j) \quad \text{(MLP Additive)}$$

$O(mnk)$ - w is fixed

Bilinear form:

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

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

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

$$s_{ij} = \text{ReLU}(c_i^T w_i^T) \, D \, \text{ReLU}(W q_j)$$

$O((m+n)k)$

ELMo and BERT

[2018] Contextual word representation

The transformer architecture used in BERT is [2017] soft of attention on steroids. More Later

BERT achieves SoTA in QA

Lecture 11: ConvNets for NLP

From RNNs to ConvNets

→ RNN cannot capture phrases without prefix context

→ often capture too much of the last words in final vector

Main CNN idea:
What if we compute vectors for every possible word subsequences of a certain length?

Convolution $(f * g)[n] = \sum_{m=-M}^{M} f[n-m]\, g[m]$ (kernel)

↳ usually used to ~~classically use~~ extract features from images.

1D conv for text

kernel → context vector → padding → conv → maxpool

So, in PyTorch

batch_size = 16

word_embed_size = 4

seq_len = 7

input = torch.randn(batch_size, word_embed_size)

conv1 = conv1D(word_embed_size, 3, 3)

hidden1 = conv1(input)

hidden2 = torch.max(hidden1, dim=2) # maxpool

- Other less useful notions:
  - stride
  - local max pool, strides
  - k-max pooling
  - dilation (skipping values)

- Single Layer CNN for Text Classification

Yoon Kim (2014) CNN for Sentence classification

Multichannel input idea

- Initialized with pretrained word vectors
- Start with two copies
- Backprop into only one set, keep other "static"
- Both channels are added to c_i before max pooling

Zhang and Wallace A sensitivity Analysis of CNNs

Regularization:

Dropout: Create masking vector r for Bernoulli rv with probability p of being 1

Reasoning: Prevents co-adaption (overfitting to see specific feature contellations)

Bag of Words
Window Model
CNNs
RNNs

Gate units vertically

Gating/skipping that we saw in LSTMs and GRUs is a general idea and ~~are~~ ~~is~~ ~~in a~~ used in other place like Resnet

Batch normalization:

[Loffe and Szegedy 2015]

- Often used in CNNs
- Transform the convolution output of a batch by scaling the activations to have zero mean and unit variance
  - Familiar to Z-transform of statistic
  - Updated per batch so fluctuations don't affect things much
- Use BatchNorm makes models much less sensitive to parameter initialization, since outputs are automatically rescaled.
  - It also tends to make tuning of learning rates simpler.

1×1 convolutions (one convolutions)

[Lin, Chen, Yan 2013, Network in Network]

- 1×1 convolutions, a.k.a. Network-in-Networks (NiN) connections, are convolutional kernels with kernel-size = 1
- A 1×1 conv gives you fully connected linear layer across channels
- It can be used to map from many channels to fewer channels.
- 1×1 convs add additional NN layers with very few additional parameters
  → Unlike FC layers which add a lot of params

Learning Character-level Representations for POS

Dos Santos and Zadrozny [2014]

- Convolution over characters to generate word embeddings.
- Fixed window of word embeddings used for POS tagging.

Very Deep ConvNets for Text Classification

- Conneau, Schwenk, Lecun, Barrault, EACL 2017
- What happens when we build a vision-like deep ~~mo~~ ~~de~~ model for NLP

VD-CNN

Very unlike ~~DL AND~~ Deep learning NLP system

→ Each conv block is two convolutional layers, each followed by batch norm and a ReLU non-linearity

- Convolutions of size 3
- Pad to ~~reserve~~ preserve dimension

RNNs are Slow

- Parallelize badly (GPU ~~doesn't~~ can't work like CNN)

Idea: take the best and parallelizable parts of RNNs and CNNs

Qausi-RNN [ICLR 2017]

Lecture 12: SubWord Models

Human language sounds: phonetics and phonology

- Phonetics is the sound system
- Phonology posits a small set or sets of distinctive, categorical units: phenomes or distinctive features
  - → A perhaps universal typology, but language-particular realization
  - Best evidence of categorical perception comes from phonology.
    - → within phoneme differences shrink; between phoneme magnified.

Morphology: Parts of words:

- Traditionally we have morphemes as smallest semantic unit

Deep learning: Morphology little studied; one attempt with recurvise neural networks is (Luong, Socher & Manning 2013)

. This is seen as the minimal level that has meaning

This hasn't gotten much popular because close results can be obtained with character n grams

Related idea to use convolution layer

Can give many of the benifits on morphemes more easily?

Words in writing systems

Writing systems vary in how they represent words or don't

No word segmentation

Words (mainly) segmented

- Clitics?
- Compounds?

Models below word level

- Need to handle large, open vocabulary
- Transliteration
- Most deep learning NLP work begins with language in its written form

Purely character level model

Very Deep CNN for Text Classification
[Conneau, EACL 2017]

Strong results

Purely character level NMT model

- Initially, unsatisfactory performance [2001?]

↓

Decoder only [Chung 2016]

↓

Promising result

English - Czech WMT 2015

- Fully Character-level Neural Machine Translation with Explicit Segmentation [Lee 2017]

decoder is char-level
- BPE (Byte Pair Encoding)
- GRU

Revisiting char based NMT with capacity and compression [Cherry 2018]

Subword models : two trends

→ word pieces. (BPE)

Hybrid architectures

. Main model has words; something else for chars

BPE (Byte Pair Encoding)

Originally a compressing algorithm

NMT of Rare Words [ACL 2016]

Most frequent byte pair → a new byte

• Replace with character ngrams

⇒ A word segmentation algo:

Though done as bottom up clustering

• Start with a unigram ∞ vocabulary of all (Unicode) characters in data

• Most frequent ngram pairs → a new ngram

→ Have target vocab size and stop when you reach it

→ Do deterministic longest piece segmentation of words

→ Segmentation is only within words identified by some prior tokenizer (commonly Moses tokenizer for MT)

- Automatically decides vocab for system

→ Not longer "word" based in conventional way

Wordpiece / Sentencepiece model

- Google NMT uses a variant of this
  - V1: wordpiece model
  - V2: sentencepiece model
- Rather than char n-gram count uses a greedy approximation to maximizing language model log likelihood to choose the pieces.
  - Add n-gram that maximally reduces perplexity

→ Wordpiece model tokenizes inside words
→ Sentencepiece model works from raw text
- Whitespace is retained as special token and grouped normally
- You can reverse things at end by joining pieces and recoding them to spaces

Wordpiece/Sentencepiece model

BERT uses a variant of the wordpiece model

- Relatively common words are in the vocab. at fairfax 1910s
- Other words are built from word pieces

4 Character-level to build word-level [Dos Santos and Zadrozny 2014]

Learning Character-level Representations of POS Tagging

→ Convolution over characters to generate word embeddings
→ Fixed window of word embeddings used for POS tagging

* Character based LSTM to build word rep'n

Recurrent Language Model

Bi-LSTM builds word representation

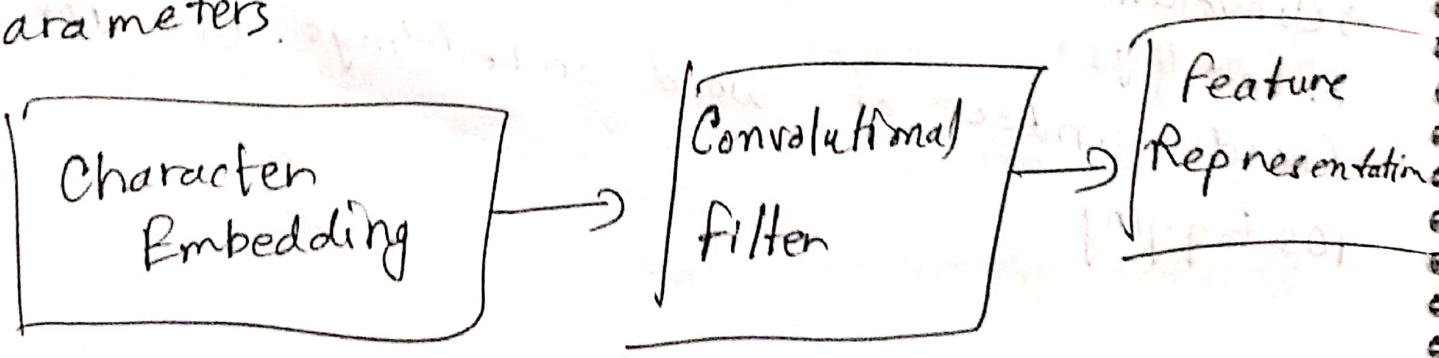
Used as LM and for POS tagging

Character-Aware Neural Language model (More complex)

Yoon Kim, Yacine Jernite, David Sontag 2015

Motivation

- Derive a powerful, robust language model effective across a variety of language
- Encode subword relatedness; eventful, eventfully
- Address rare-word problem of prior models
- Obtain comparable expressibility with fewer parameters



- Conv over char-level inputs
- Max over time pooling

Highway Network ('Srivastava et. al. 2015)

- Model n-grams interaction
- Apply transformation while carrying over original information
- Functions akin to an LSTM memory cell

⇝ Hierarchial Softmax to handle large output vocab
⇝ Trained with truncated BPTT backprop through time

~~Comparati~~
→ Comparable performance & with fewer params

Hybrid NMT (Best of Both world)

- Translate mostly at the word level
- Only go to the character level when needed

2 Stage Decoding

- Word-level beam search
- Character level beam search for <unk>

Char for word embeddings

- A Joint Model for Word Embedding and Word Morphology (Cao and Rei 2016)
  - → Same objective as w2v but using characters
  - → Bi-directional LSTM to compute embedding
  - → More attempts to capture morphology
  - → Models can infer roots of words

Fast Text embeddings

Enriching word vectors and subword information (Bojanowski, FAIR 2016)

- → Aim: a next gen efficient w2v-like word representations library butter better for rare words and languages with lots of morphology
- → An extension of the w2v skip-gram model with char n-grams

- Represent word as chor n-grams augmented with boundary symbols and as 'whole word'
- Represent word as sum of these representations

word in context score is

$$s(w,c) = \sum_{g \in G(w)} z_g^T v_c$$

Detail: rather than sharing representations for all n-grams use "hashing trick" to have fixed number of vectors.

FastText have more that 60-70 language word embedding which can be a good source for multi-lingual research.

Lecture 13: Contextual Word Embeddings

Contextual Word Representations and Pre-training:

i) Reflection on word representations
ii) Pre-ELMo and ELMo
iii) ULMFit and onward
iv) Transformer architectures
v) BERT

Reflection on word representations

Word vectors that we learned (Word2Vec, GloVe, fastText)

In early years supervised NN didn't word better for POS and NER while unsupervised pretraining worked better. But that was hard to train.

Current sense:

We can start from random word vectors and train them on our task of interest. Pretrained word vectors help a lot because we can train them for more words on much more data

Tips for unknown words with word vectors

- Vocab is {words occuring ≥ 5 time} ∪ {⟨UNK⟩}
- Map all rarer words to ⟨UNK⟩, train a word vector for it.
- Runtime: use ⟨UNK⟩ when out-of-vocabulary words occur

Problems → No way to distinguish between UNK words

Solution → char-level models

Problems with existing word representation

2 problems:

i) Always the same representation for a word type regardless of the context in which a word token occurs.

ii) We just have one representation for a word but words have different aspects, including semantics, syntactic behavior and register/connotations

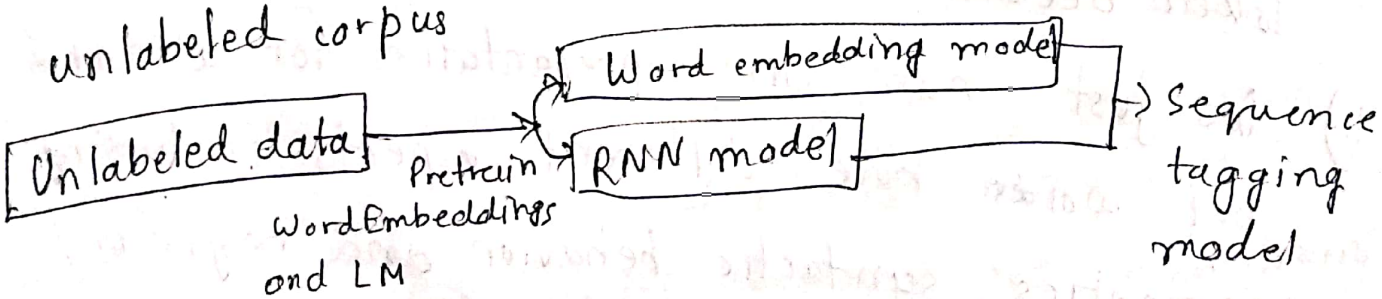
we unknowingly solved this problem earlier

- In NLM, we immediately stuck word vectors through LSTM layers
- Those LSTM layers are trained to predict the next word
- But those language models are producing context specific a word representation at each position

TagLM - Pre ELMo

Idea: Want meaning of word in context, but standardly learn task RNN only on small task labeled data

→ Semi supervise approat to train NLM on large unlabeled corpus



NER (Named Entity Recognition)

- A very important subtask: find and classify names in text, Person, date, location, organization

CoNLL 2003 NER dataset

LM mode observations

i) LM trained on supervised data doesn't help
ii) Having bidirectional LM helps over only forward, by about 0.2
iii) Having a huge LM design helps over a smaller model by about 0.3

ELMo (Embeddings from Language Model)

Peters, et al 2018

Deep contextualized word representations NAACL 2018

→ Breakout version of word token vectors or contextual word vectors
→ Learn word token vectors using long contexts not context windows
→ Learn a deep Bi-NLM and use all its layers in pred.

ELMo details

i) Train bidirectional LM

ii) Aim at performant but not overly large LM

- Use biLSTM layers
- Use character CNN to build initial word representations (only)
  - → 2048 char n-gram filters and 2 highway layers, 512 dim projection
  - → Use 4096 dim hidden/cell LSTM states with 512 dim projections to next input
  - → Use a residual connection
  - → Tie params of token input and output (softmax) and tie these between forward and backward LMs
- ELMo learns task specific combination of biLM representations

→ This is an innovation that improves on just using top-layer of LSTM stack

$$R_k = \{x_k^{LM}, \overrightarrow{h}_{k,j}^{LM}, \overleftarrow{h}_{k,j}^{LM} \mid j = 1, \ldots, L\}$$

$$= \{h_{k,j}^{LM} \mid j = 0, \ldots, L\}$$

$$ELMo_k^{task} = E(R_k; \theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} h_{k,j}^{LM}$$

- γ^{task} scales overall usefulness of ELMo to task
- s^{task} scales a softmax-normalized mixture model weight.

- Won the best paper award at NAACL 2018

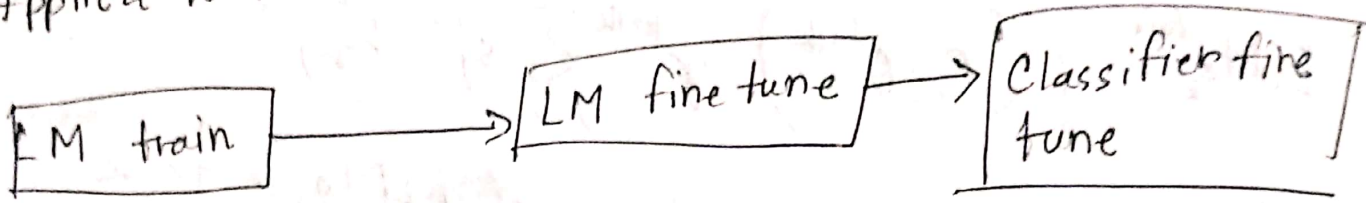
Two biLSTM NLM layers have differentiated uses/ meanings

→ Lower layer is better for lower-level syntax
POS, syntactic dependencies, NER

→ Higher layer is better for higher-level semantics
Sentiment, Semantic role labeling, question answering, SNLI

ULMFit [Howard and Ruder 2018]

- Same general idea of transfering NLM knowledge
- Applied to text classification



- Exception in freezing/unfreezing

- Use reasonable-size "1-GPU" ~~lem~~ LM not really huge one

→ A lot of care in LM fine-tuning

- Different per-layer learning rates
- Slanted Triangular learning rate (STLR) schedule

→ Gradual layer unfreezing and # STLR when learning classifier

→ classify using concatenatin $[h_T, maxpool(h), meanpool(h)]$

ULMFit	GPT	BERT	GPT-2
Jan 2018	June 2018	Oct 2018	feb 2019
Training:	Training:	Training	Training
1 GPU day	240 GPU day	256 TPU days	~2048 TPU v3
fast.ai	OpenAI	~320-560 GPU days	OpenAI
		Google AI	

Uses Transformer architecture

Transformers

We want parallelization but RNNs are inherently sequential.

→ Despite GRUs and LSTMs, RNNs still need attention mechanism to deal with long range dependencies- path length between states grows with sequence otherwise.

→ If attention gives us access to any state; maybe we can just use attention and don't need the RNN.

- Non-recurrent sequence-to-sequence encoder-decoder
- Machine translation with parallel corpus
- Predict each translated word
- Final cost/error function is standard cross-entropy error on top of softmax classifier.

Dot product Attention

Inputs: a query q and a set of ~~va~~ key value (k-v) pairs to an output

- Query, keys, values, and output are all vectors

Output is weighted sum of values, where weight of each value is computed by an inner product of query and corresponding key

$$A(q,K,v) = \sum_i \frac{e^{q \cdot k_i}}{\sum_j e^{q \cdot k_j}} v_i \quad = \text{softmax}(QK^T)V$$

Dot product attention

Scaled Dot Product Attention

As d_k gets large, the variance of $q^T k$ increases → some values inside the softmax get large → the softmax gets very peaked → hence its gradient get smaller

→ Scale by length of query/key vectors

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

Self attention in the encoder:

→ The input word vectors are the queries, key and value

→ In other words; the word vectors themselves select each other

→ Word vector stack = Q = K = V

→ ~~We~~

Multihead attention:

- First map Q, K, V into $h = 8$ many lower dim spaces via W matrices
- Then apply attention, then concatenate outputs and pipe through linear layers

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h) W^O$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Transformer block

Two sublayers

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

Each of these two two steps & also has

Residual (short-circuit) connection and LayerNorm

LayerNorm $(x + \text{Sublayer}(x))$ [Kiros, Hinton, 2016]

$$\mu^t = \frac{1}{H}\sum_{i=1}^{H} a_i^t \qquad \sigma^t = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^t - \mu^t)^2}$$

$$h_i = f\left(\frac{g_i}{\sigma_i}(a_i - \mu_i) + b_i\right)$$

• These are highly advantageous in GPU archs

Encoder input

• Actual word representation are BPE

Positional encoding

Some words at different locations have different overall representations

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

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

Transformer Decoder:

• 2 sublayers changes in decoder

• Masked decoder self-attention on previously generated outputs

• Encoder-Decoder attention: where queries of come from previous decoder layer and keys and values come from output of encoder

BERT (Devlin 2018)

Pre-training of Deep Bidirectional Transformers for Language Understanding

Problem: Language models use left context or right context, but language understanding is bidirectional

- Why are LMs unidirectional?

Reason 1: Directionality is needed to generate a well-formed probabilistic distribution

Reason 2: Words can "see themselves" in a bidirectional encoder

To solve bidirectionality problems in next layer they mask out k% of input words, and then predict the masked works

They always use k=15%

- Too little masking → Too expensive to train
- Too much masking → no enough context

BERT complication: Next sentence prediction

- To learn relationships between sentences, predict whether sentence B is not actual sentence that proceeds sentence A, or a random sentence

Input

Token Embeddings

+

Segment Embedding

+

Positional Embedding

BERT training

Transformer encoder

Self attention

Single multiplication per layer

→ Train on Wikipedia + Book Corpus

→ Train 2 model sizes

- BERT-base (12-layer, 768 hidden, 12-head)
- BERT-Large (24-layer 1024 hidden, 16-head)

- Trained on 4x4 or 8x8 TPU slice for 4 days

BERT Fine-tune

Simply learn a classifier on top layer for each task you fine tune for

BERT results on GLUE tasks

- GLUE benchmark is dominated by natural language inference task but also has sentence similarity and sentiment

→ MultiNLI

→ CoLA

(*) We Now have strongly performing deep, pre-trained, neural net stacks for NLP in the same way vision has had for 5 years