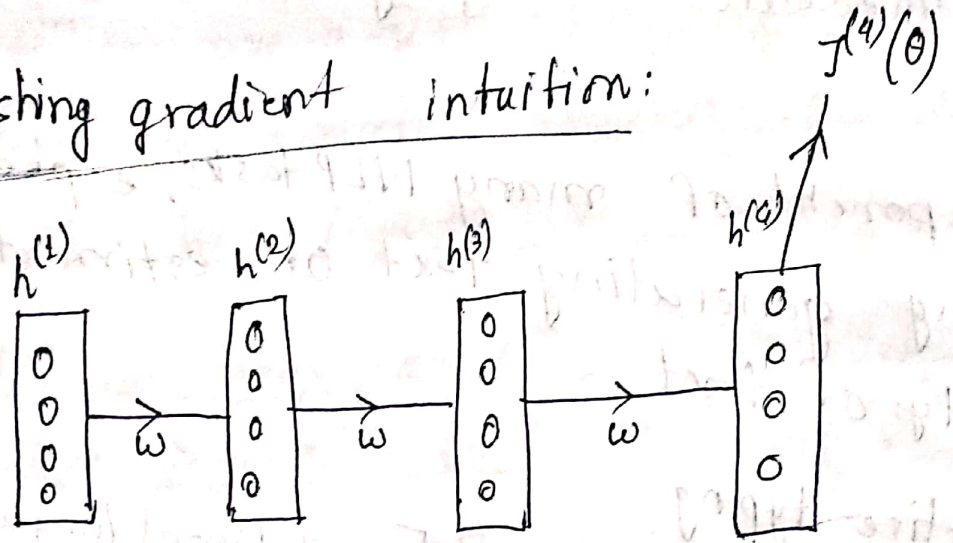
Lecture 7: Vanishing Gradients, Fancy RNNs

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

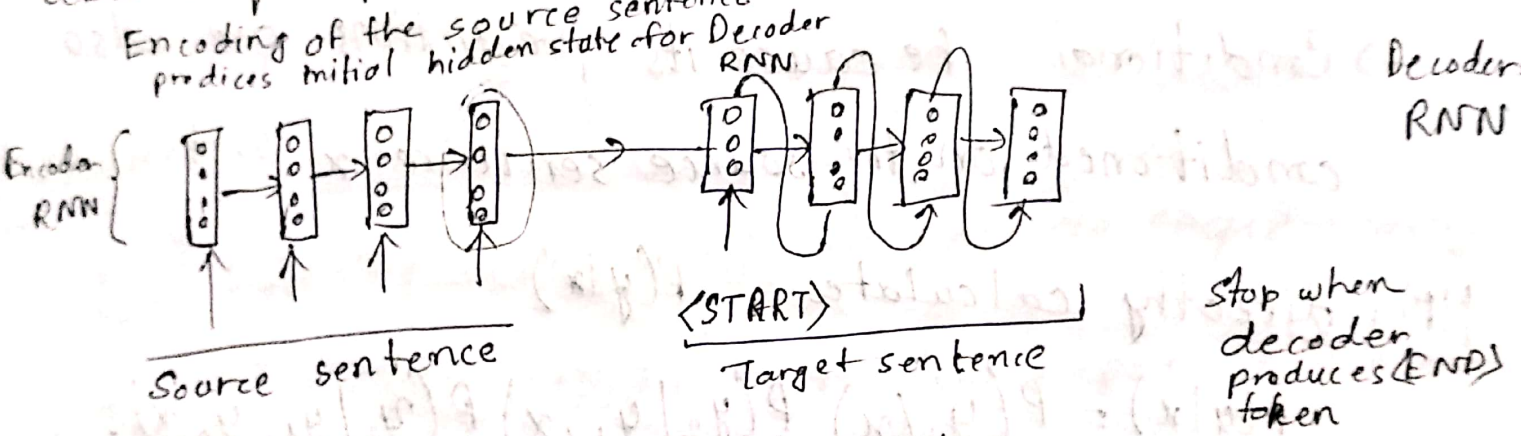
→ lots of feature engineering

→ 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.

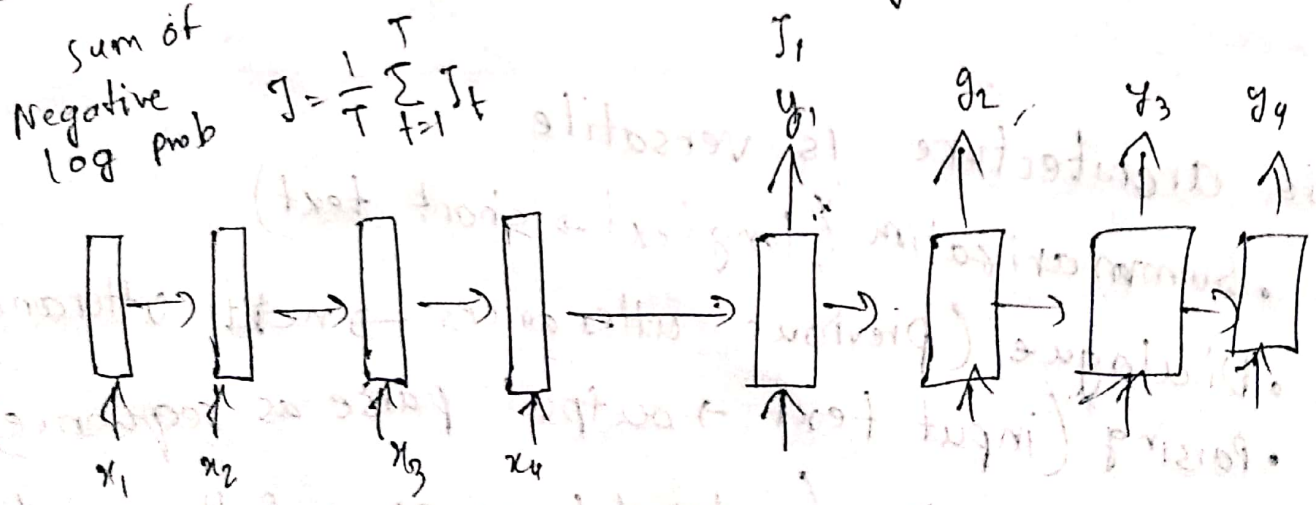
→ Conditional because its predictions are also conditioned on the source sentence x

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

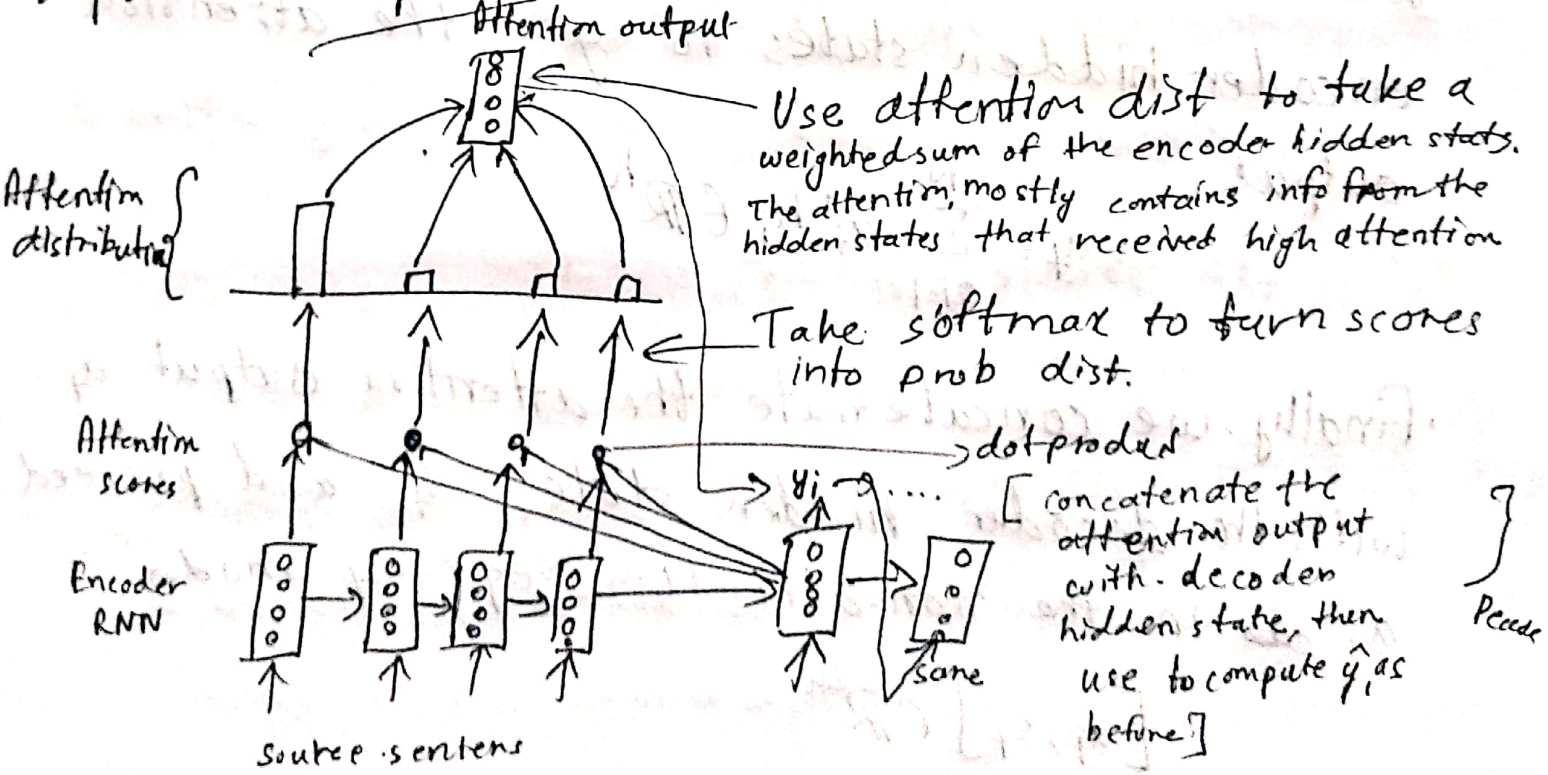
Sequence-to-sequence: the bottleneck

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.