Neural Machine Translation

Cyrus M. Vahid,
Principal DeepLearning Solution Architect
AWS DeepLearning
cyrusmv@amazon.com
Oct 2017
Problems of Translation

To perform translation we need to solve blow problems:

- **Modeling**: First, we need to decide what our model $P(E|F; \theta)$ will look like. What parameters will it have, and how will the parameters specify a probability distribution?

- **Learning**: Learning: Next, we need a method to learn appropriate values for parameters from training data.

- **Search**: Finally, we need to solve the problem of finding the most probable sentence (solving “argmax”). This process of searching for the best hypothesis and is often called decoding.
Background
Statistical Language Models
Machine Translation Statistical Prior

- Machine Translation Task is predicting a sentence in a language, given another sentence in another language, more formally:

\[ P(E|F; \theta) \text{ and } \theta \text{ is model parameters} \]

- The translation systems thus needs to be a language models itself, meaning being capable of building utterances that are naturally sounding, give a vocabulary. More formally:

\[ P(E; \theta) \text{ and } \theta \text{ is model parameters} \]
Consider we are calculating probability of a sentence of length $T$ being natural where the sentence is formed by words sequences: $e_1, e_2, \ldots, e_T$. We should this formally as:

$$P(E) = P(|E| = T, e_T^T) = \prod_{t=1}^{T+1} P(e_t | e_{t-1}^{t-1})$$

$$T = 3 \implies P(e_1, e_2, e_3) = P(e_1)P(e_2 | e_1)P(e_3 | e_1, e_2)$$

$P(|E| = 3, e_1 = \text{"she"}, e_2 = \text{"went"}, e_3 = \text{"home"}) =$

$P(e_1 = \text{"she"})$ * $P(e_2 = \text{"went"} | e_1 = \text{"she"})$

$P(e_3 = \text{"home"} | e_1 = \text{"she"}, e_2 = \text{"went"})$

$P(e_4 = \text{"</s>"} | e_1 = \text{"she"}, e_2 = \text{"went"}, e_3 = \text{"home"})$
Count-Based n-gram Language Models

- Prepare a set of training data from which we can count word strings
- Count up the number of times we have seen a particular string of words
- Divide it by the number of times we have seen the context.

More Formally:

\[
P_{\text{ML}}(e_t | e_{t-1}) = \frac{c_{\text{prefix}}(e_t)}{c_{\text{prefix}}(e_{t-1})}.
\]

Example:

- \( i \ am \ from \ pittsburgh . \)
- \( i \ study \ at \ a \ university . \)
- \( my \ mother \ is \ from \ utah . \)

\[
P(e_2=\text{am} | e_1=\text{i}) = \frac{c(\text{i}, \text{am})}{c(\text{i})} = \frac{1}{2} = 0.5
\]

\[
P(e_2=\text{study} | e_1=\text{i}) = \frac{c(\text{i}, \text{study})}{c(\text{i})} = \frac{1}{2} = 0.5
\]
Count-Based n-gram LM – The Problem

- If there is a sequence of four words such as ”I am from Utah” that does not exist in training set, then

\[
C_{\text{prefix}}(i, \text{am, from, utah}) = 0
\]

\[
\therefore P(e_4 = \text{utah}|e_1 = i, e_2 = \text{am}, e_3 = \text{from}) = 0
\]
Count-Based n-gram LM – The Solution

• If we just focus on the $n - 1$ previous words, we can eliminate the problem of assigning probability of zero to new sentences. These models are called $n$-grams.
• The maximum likelihood formula thus take the following form:

$$P(e_t \mid e_{1}^{t-1}) \approx P_{ML}(e_t \mid e_{t-n+1}^{t-1}).$$

• Example: ”i am”, “am from”, ”from utah”, “utah .” all seen in training set, so we can assign probability to “i am from utah.”

i am from pittsburgh .
i study at a university .
my mother is from utah .
Count-Based n-gram LM – The Problem Persists

• In bigrams we have the same problem of assigning probability 0 if there is a sequence of 2 words that does not exist in training data.
• In trigrams we have the same problem of assigning probability 0 if there is a sequence of 3 words that does not exist in training data.
• ...
• In n-grams we have the same problem of assigning probability 0 if there is a sequence of n words that does not exist in training data.
Evaluation of Accuracy - Likelihood

- **Likelihood**: Likelihood is probability of some observed data—e.g. test, given a model.

\[ P(\mathcal{E}_{\text{test}}; \theta) = \prod_{E \in \mathcal{E}_{\text{test}}} P(E; \theta) \]

- Example:

  \[ |\mathcal{E}_{\text{test}}| = 10^6 \]
  \[ E_1 = \text{I am from pittsburg}, P(E; \theta) = 2.1 \times 10^{-6} \]
  \[ E_2 = \text{I study at a university}, P(E; \theta) = 1.1 \times 10^{-8} \]
  \[ E_3 = \text{my mother is from utah}, P(E; \theta) = 2 \times 10^{-9} \]

\[ P(\mathcal{E}_{\text{test}}; \theta) = \prod_{E \in \mathcal{E}_{\text{test}}} P(E; \theta) = 2.1 \times 10^{-6} \times 1.1 \times 10^{-8} \times 2.0 \times 10^{-9} = 4.62 \times 10^{-23} \]

*probabilities are arbitrary just to demonstrate the range*
Evaluation of Accuracy – Log Likelihood

- **Theorem**: \( \log_b \prod_{i=1}^{n} x_i = \sum_{i=1}^{n} \log_b x_i \); **Example**: \( \log(x \cdot y) = \log(x) + \log(y) \)

- We have seen probabilities of likelihood are very tiny. Multiplying these small values make the probabilities even smaller numbers. It is easier to work with log probabilities for measuring accuracy.

\[
\log(P(\mathcal{E}_{test}; \theta)) = \log_b \prod_{E \in \mathcal{E}_{test}} P(E; \theta) = \sum_{E \in \mathcal{E}_{test}} \log(P(E; \theta)).
\]

- **Example**:

  \( |\mathcal{E}_{test}| = 10^6 \)

  \( \mathcal{E}_1 = \text{I am from pittsburg}, P(E; \theta) = 2.1 \times 10^{-6} \therefore \log(P(E; \theta)) = \log(2.1) + (-6) = -5.67 \)

  \( \mathcal{E}_2 = \text{I study at a university} P(E; \theta) = 1.1 \times 10^{-8} \therefore \log(P(E; \theta)) = \log(1.1) + (-8) = -7.96 \)

  \( \mathcal{E}_3 = \text{my mother is from utah}, P(E; \theta) = 2 \times 10^{-9} \therefore \log(P(E; \theta)) = \log(2.0) + (-9) = -8.70 \)

\[
P(\mathcal{E}_{test}; \theta) = \prod_{E \in \mathcal{E}_{test}} P(E; \theta) = \sum_{E \in \mathcal{E}_{test}} \log(P(E; \theta)) = -5.67 - 7.96 - 8.70 = -22.33
\]

*probabilities are arbitrary just to demonstrate the range*
Evaluation of Accuracy – Perplexity

• It is common to divide log-likelihood by the number of words in the corpus. This makes cross-corpora measurement easier.

\[
\text{length}(E_{test}) = \sum_{E \in E_{test}} |E|
\]

\[
ppl(E_{test}; \theta) = e^{-\left(\log(P(E_{test}; \theta)) \right) / \text{length}(E_{test})}
\]

• This accuracy indicator is called **perplexity**.
Neural Language Models
Motivation – MT and Context

• In SMT is count-based and looses so much of hidden information.
• There are other models such as log-linear where we actually include the context based on feature vectors, but we have other problems:

  **farmers eat**
  - steak $\rightarrow$ **high**
  - hay $\rightarrow$ **low**
  - cows eat
    - steak $\rightarrow$ **low**
    - hay $\rightarrow$ **high**

  **farmers grow**
  - steak $\rightarrow$ **low**
  - hay $\rightarrow$ **high**
  - cows grow
    - steak $\rightarrow$ **low**
    - hay $\rightarrow$ **low**

• In separate contexts $e_{t-2}$ = “farmers” is compatible with $e_t$ = “hay” and $e_{t-1}$ = “eats”.
• If we are using feature sets depending on $e_{t-1}$ and $e_{t-2}$, then a sentence like “farmers eat hey” cannot be ruled out, even though it is an unnatural sentence.
• The solution is adding future combination that increases dimensionality. It is time to find a better solution
Step 1, Word Embedding

- In Neural Language Models we represent words as word embedding, a vector representation of words.
- Then we concatenate these words to the length of the context (3 for trigrams).

![Diagram showing word embedding process with examples for j=1, j=2, j=3, j=4: a, the, hat, giving, respectively. Features for the previous word, two words ago, and both words are shown as vectors.](image-url)
Step 1, Uncovering Combination Features

- Different elements of a vector can potentially reflect different aspect of a word. For instance, different elements could represent countability, being and animal, ...
- We run the embedding vectors through hidden layer in order to learn combination features

https://www.tensorflow.org/tutorials/word2vec
Step 3, Calculating Scores and Probability

• Next, we calculate the score vector for each word. This is done by performing an affine* transform of the hidden vector $h$ with a weight matrix and adding a bias vector.
• Finally, we get a probability estimate $p$ by running the calculated scores through a softmax function.

\[
m = \text{concat}(M_.e_{t-2}, M_.e_{t-1})
\]
\[
h = \tanh(W_{mh}m + b_h)
\]
\[
s = W_{hs}h + b_s
\]
\[
p = \text{softmax}(s)
\]

* An affine transformation is any transformation that preserves collinearity (i.e., all points lying on a line initially still lie on a line after transformation) and ratios of distances (e.g., the midpoint of a line segment remains the midpoint after transformation). Ref: Wolfram Mathworld
Benefits of Neural Language Models

• **Better generalization of contexts:** n-gram models treat words as independent entities. Word embedding allows grouping of similar words that perform similarly in prediction of the next word.

• **More generalizable combination of words into contexts:** In an n-gram language model, we would have to remember parameters for all combinations of {cow; horse; goat} X {consume; chew; ingest} to represent the context of things farm animals eat". Hidden layer are more efficient in representing the concept.

• **Ability to skip previous words:** n-gram models refer to all sequential words depending the length of the context. Neural Models can skip words.

* An affine transformation is any transformation that preserves collinearity (i.e., all points lying on a line initially still lie on a line after transformation) and ratios of distances (e.g., the midpoint of a line segment remains the midpoint after transformation). Ref: [Wolfram Mathworld](https://mathworld.wolfram.com/AffineTransformation.html)
He doesn't have very much confidence in himself
She doesn't have very much confidence in herself
He doesn't have very much confidence in himself.
She doesn't have very much confidence in herself.

Seine steuerliche Bewertung wäre günstiger gewesen, wenn seine Steuererklärung für das Geschäftsjahr 1927 früher überprüft worden wäre.

His tax assessment would have been more favorable if his tax return for fiscal year 1927 had been checked earlier.
His tax assessment would have been more favorable if his tax return for fiscal year 1927 had been checked earlier.

Seine steuerliche Bewertung wäre günstiger gewesen, wenn seine Steuererklärung für das Geschäftsjahr 1927 früher überprüft worden wäre.

He doesn't have very much confidence in himself
She doesn't have very much confidence in herself
There is Still Something Missing...

He doesn't have very much confidence in himself.
She doesn't have very much confidence in herself.

Handling Long-Term Dependencies

His tax assessment would have been more favorable if his tax return for fiscal year 1927 had been checked earlier.
Sequences and Long-Term Dependency

- The Spanish King officially abdicated in … of {his} …, Felipe. Felipe will be confirmed tomorrow as the new Spanish … .
The Spanish King officially abdicated in favour of his son, Felipe. Felipe will be confirmed tomorrow as the new Spanish King.
Sequences and Long-Term Dependency

- Their duo was known as "Buck and Bubbles". Bubbles tapped along as Buck played on the stride piano and sang until Bubbles was totally tapped out.
Sequences and Long-Term Dependency

- CNN is news organization
- CNN is short for Convolutional Neural Network
- In our quest to implement perfect NLP tools, we have developed state of the art RNNs. Now we can use them to … .
Sequences and Long-Term Dependency

- CNN is news organization
- CNN is short for Convolutional Neural Network
- In our quest to implement perfect NLP tools, we have developed state of the art RNNs. Now we can use them to wreck a nice beech. (Jeoffy Hinton – Coursera)
Let $x$ and $y$ be matrices and $z$ the dot product; $z = x \cdot y$

$\hat{y} = x \cdot w$

weight decay penalty $= \lambda \Sigma_i w_i^2$
Unfolding Computational Graph

- \( s^{(t)} = f(s^{(t-1)}; \theta) \); where \( \theta \) is model params. (A)
  
  For instance if \( t = 3 \) then:
  \[
  s^{(3)} = f(s^{(2)}; \theta) = f(f(s^{(1)}; \theta); \theta)
  \]

- \( s^{(t)} = f(s^{(t-1)}, x^{(t)}; \theta) \); where \( x \) is external signal such as input vector (B)

- \( h^{(t)} = f(h^{(t-1)}, x^{(t)}; \theta) \); where \( h \) is the state of hidden node (C)

http://www.deeplearningbook.org/
RNN Unrolling Through Time

(a) A single RNN time step

\[
\begin{align*}
\mathbf{h}_t &= \tanh(\mathbf{x}_t \mathbf{W}_{xh} + \mathbf{h}_{t-1} \mathbf{W}_{hh} + \mathbf{b}_h) \\
\end{align*}
\]

(b) An unrolled RNN

\[
\begin{align*}
\mathbf{h}_0 &\rightarrow \mathbf{x}_1 \rightarrow \tanh \rightarrow \mathbf{x}_2 \rightarrow \tanh \rightarrow \mathbf{x}_3 \rightarrow \tanh \\
\end{align*}
\]

(c) A simplified view

\[
\begin{align*}
\mathbf{h}_0 &\rightarrow \text{RNN} \rightarrow \text{RNN} \rightarrow \text{RNN} \rightarrow \mathbf{x}_1 \rightarrow \mathbf{x}_2 \rightarrow \mathbf{x}_3 \\
\end{align*}
\]
RNN Unrolling - Example

\[ X = \text{emb}({"i", "am", "from", "earth"}) \]

\[ x_1 = \text{emb}("i"), x_2 = \text{emb}("am"), \]
\[ x_3 = \text{emb}("from"), x_4 = \text{emb}("earth") \]

\[ h(t) = \begin{cases} 
\tanh(W_{hx}x_t + W_{hh}h_{t-1}), & t \geq 1, \\
0, & \text{otherwise} 
\end{cases} \]

\[ P(t) = \text{softmax}(W_{hs}h_t + b_s) \]
Training RNN Networks

- Unrolling in time makes the turns the training task into a feed-forward network.
- Since RNN represents a dynamical system we carry weight matrices along the time steps.
Vanishing & Exploding Gradients

- In example of a document classification the networks makes a prediction over several timesteps.
- The model, then calculates a loss and back-propagates it over all the timesteps.
- At each timestep, running the BP algorithms causes the gradients to get smaller and smaller until negligible.
- Unless $\frac{dh_t}{dh_{t-1}} = 1$, it tends to either diminish ($< 1$) or explode ($> 1$) through repeated participation in computation.
Vanishing and Exploding Gradients; *the math*

Multiplying weight Matrix $W$, $t$ times to reach time-step $t \to W^t$

Suppose $W$ has decomposition $W = V\text{diag}(\lambda)$

$$W^t = (V\text{diag}(\lambda)V^{-1})^t = (V\text{diag}(\lambda)^t V^{-1})$$

When $\lambda \neq 1$ $W$ can become too large or too small.

- More specially in RNNs

$$h^t = W^T h^{(t-1)}$$

$$h^{(t)} = (W^t)^T h^{(0)}$$

*If $W$ admits eigendecomposition $Q\Lambda Q^T$*

$$\therefore h^{(t)} = Q^T \Lambda^t Q h^{(0)}$$

- If $\Lambda \neq I_t = \text{diag}(1, 1, \ldots, 1)_n$ then the gradients can diminish or explode.

http://www.deeplearningbook.org/
Long Short Term Memory (LSTM)
LSTM

- To resolve the vanishing gradient problem we want to design a network that ensures derivative of recurrent function is exactly 1.
- The idea behind LSTM is to add a gated memory cell $c$ to hidden nodes for which $\frac{dc_t}{dc_{t-1}}$ is exactly 1.
- Since derivative of the recurrent function applied to memory cell is exactly one, the value inside the cell does not suffer from vanishing gradient.

*Picture from Alex Grave PHD Thesis*
LSTM Cell—The Anatomy

Output gate (sigmoid) controls the output’s contribution

Linear self-loop

Forget gate (sigmoid) controls the value of state self-loop

Input gate (non-linear) controls the input

\[ \sigma(x) = \frac{1}{1 - e^{-x}} \]

\[ \tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \]
LSTM Cell—Input and Output Gates

- “gates” either allow information to pass or block it.
- Gate functions perform an affine transform followed by the **sigmoid** function, which squashes the input between 0 and 1.
- The output of the sigmoid is then used to perform a componentwise multiplication. This results in the “gating” effect:
  - if $\sigma \approx 1 \Rightarrow$ gate is open: it will have little effect on the input.
  - if $\sigma \approx 0 \Rightarrow$ gate is closed: it will block the input.
LSTM; \textit{the math}

\[
\begin{align*}
  u_t &= \tanh(W_{xu}x_t + W_{hu}h_{t-1} + b_u) \\
  i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \\
  o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \\
  c_t &= i_t \odot u_t + c_{t-1} \\
  h_t &= o_t \odot \tanh(c_t).
\end{align*}
\]

Calculating the Next Hidden State; used in probability calculation of lang. model:
\[p_t = \text{softmax}(W_{hs}h_t + b_s)\]

How Much to update?

Gate Functions

\[
\frac{dc_t}{dc_{t-1}} = \frac{d(i_t \odot u_t + c_{t-1})}{dc_{t-1}} = 0 + 1 = 1
\]

This is the main goal of LSTM to avoid vanishing or exploding gradients
Adding Forget Gate

- In most common variant of LSTM, a forget gate is added, so that the value of memory can be reset. We often would like to forget memory cell values after making an inference and no longer need the long term dependencies.

\[
\begin{align*}
    u_t &= \tanh(W_{xu}x_t + W_{hu}h_{t-1} + b_u) \\
    i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \\
    f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \\
    o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \\
    c_t &= i_t \odot u_t + f_t \odot c_{t-1} \\
    h_t &= o_t \odot \tanh(c_t).
\end{align*}
\]

Often a large bias to prevent the network from forgetting everything.

Modulating \(c_t\) with \(f_t\) enables the network to easily clear its memory when justified by setting \(f\text{-gate}\) to 0.
Handwriting Recognition by Alex Graves
Stacking RNNs

- Like all types of ANN, one can stack RNNs in order to detect more abstract features such as part of speech or tense.
- Stacking can however cause vanishing gradients in vertical direction.
- An easy solution is residual networks in which we add output of the previous layer to the next layer.

\[
\begin{align*}
    h_{1,t} &= \text{RNN}_1(x_t, h_{1,t-1}) + x_t \\
    h_{2,t} &= \text{RNN}_2(h_{1,t}, h_{2,t-1}) + h_{1,t} \\
    h_{3,t} &= \text{RNN}_3(h_{2,t}, h_{3,t-1}) + h_{2,t}.
\end{align*}
\]
Encoder-Decoder Models
Problems of Translation

To perform translation we need to solve blow problems:

• **Modeling:** First, we need to decide what our model $P(E|F; \theta)$ will look like. What parameters will it have, and how will the parameters specify a probability distribution?

• **Learning:** Learning: Next, we need a method to learn appropriate values for parameters from training data.

• **Search:** Finally, we need to solve the problem of finding the most probable sentence (solving “argmax”). This process of searching for the best hypothesis and is often called decoding.
Translation: Inferring a Sequence from another

http://opennmt.net/
Encode-Decoder Models

- We still intend to calculate $P(E; \theta)$, but before doing so we would like to calculate the initial state of the target model based on another RNN over the source sentence $F$.
- First RNN “encodes” source sentence $F$, and second RNN “decodes” the initial state into target sentence $E$. 
Encode-Decoder Models

**Encoder**

- $0 \rightarrow \text{RNN}^{(f)} \rightarrow \text{RNN}^{(f)} \rightarrow \cdots \rightarrow \text{RNN}^{(f)} \rightarrow h_{[F]}$
- $\text{lookup}^{(f)} \rightarrow \text{lookup}^{(f)} \rightarrow \text{lookup}^{(f)}$
- $f_1 \rightarrow f_2 \rightarrow f_{[F]}$

**Decoder**

- $p_1^{(e)} \rightarrow \text{softmax}^{(e)} \rightarrow \text{RNN}^{(e)} \rightarrow \text{RNN}^{(e)} \rightarrow \cdots \rightarrow \text{RNN}^{(e)}$
- $p_2^{(e)} \rightarrow \text{softmax}^{(e)} \rightarrow \text{RNN}^{(e)} \rightarrow \text{RNN}^{(e)} \rightarrow \cdots \rightarrow \text{RNN}^{(e)}$
- $p_{[E]}^{(e)} \rightarrow \text{softmax}^{(e)} \rightarrow \text{RNN}^{(e)} \rightarrow \text{RNN}^{(e)} \rightarrow \cdots \rightarrow \text{RNN}^{(e)}$
- $e_0 \rightarrow e_1 \rightarrow e_{[E]-1}$

Mathematical Formulas:

- $m_t^{(f)} = M_{:, f_t}^{(f)}$
- $h_t^{(f)} = \begin{cases} \text{RNN}^{(f)}(m_t^{(f)}, h_{t-1}^{(f)}) & t \geq 1, \\
0 & \text{otherwise.} \end{cases}$
- $m_t^{(e)} = M_{:, e_{t-1}}^{(e)}$
- $h_t^{(e)} = \begin{cases} \text{RNN}^{(e)}(m_t^{(e)}, h_{t-1}^{(e)}) & t \geq 1, \\
h_{[F]}^{(f)} & \text{otherwise.} \end{cases}$
- $p_t^{(e)} = \text{softmax}(W_{hs} h_t^{(e)} + b_s)$
Generating Output

• So far we have seen how to make a probability model. In order to perform translation, we need to generate output. Generally this is performed using search in parallel corpora, the corpora of target language.

• Search is performed based on several Criteria:
  • **Random Sampling**: Randomly selecting an output $E$ from a probability model. $\hat{E} \sim P(E|F)$
  • **Greedy 1-best search**: Finding $E$ that maximizes $P(E|F)$; $\hat{E} = \arg\max_E P(E|F)$
  • **n-best search**: Finding the n options with the highest probability $P(E|F)$
Random Sampling

- The method is used when we would like to find several outputs for an input such as **dialog models** in **chat bots**.
- Ancestral Sampling: Sampling variable values one at a time and gradually conditioning to more context.
- The method is unbiased, but has high variance and is inefficient.
  - Assume \( F \) represents \( X = \{ \text{"the", "actor", "walked", "onto", "the", "stage", ",", < eos >} \} \)
  - at time step \( t \) sample a word from distribution \( P(e_t | \hat{e}_{t-1}^t) \) until \( x_i = < \text{eos} > \)

```text
RNN: F

\[ h_0^g \]
\[ h_1^e \]
\[ h_1^e \]
```

```
the -> actor
the/F actor/the walked the, actor
```

```
```

```
Greedy 1-Best Search

- This is very similar to random sampling with one difference: The goal is to search for the word in target corpora that maximized probability;
  \[ \hat{e}_t = \arg \max_i \log(p_{t,i}^{(e)}) \]
- Greedy search is not guaranteed to find the translation with highest probability, but is efficient in terms of memory and computation.

https://github.com/tensorflow/nmt
Beam Search

- Beam Search is a search method that searched for b best hypotheses at each time step. b is called width of the beam.
- A larger beam size has a strong length bias.
- b is a hyperparameter that should be selected to maximize accuracy.
- Not very efficient.
Attentional NMT
Limitation of Encoder-Decoder

- Long distance dependencies between words that need to be translated to each other
- Storing information of sentences of arbitrary length in the same fixed-length vector.
  - Small model cannot store information about long sentences.
  - Large model is inefficient for small sentences.
  - Increasing number of parameters makes it difficult to train the model.

He purchased a car, so that he could drive to work and not spend his time on train.

Er kaufte ein Auto, damit er zur Arbeit fahren und seine Zeit nicht im Zug verbringen konnte.
Attention

• The attention model comes between the encoder and the decoder and helps the decoder to pick only the encoded inputs that are important for each step of the decoding process.

• For each encoded input from the encoder RNN, the attention mechanism calculates its importance:

https://talbaumel.github.io/attention/
Attention – Visualization of Importance

https://github.com/awslabs/sockeye/tree/master/tutorials/wmt
Attention Mechanism

the attention computation happens at every decoder time step. It consists of the following stages:

1. The current target hidden state is compared with all source states to derive **attention weights**
2. Based on the attention weights we compute a **context vector** as the weighted average of the source states.
3. Combine the context vector with the current target hidden state to yield the final **attention vector**
4. The attention vector is fed as an input to the next time step (**input feeding**). The first three steps can be summarized by the equations below:

\[
\alpha_{ts} = \frac{\exp\left(\text{score}(h_t, \tilde{h}_s)\right)}{\sum_{s'=1}^{S} \exp\left(\text{score}(h_t, \tilde{h}_{s'})\right)}
\]

\[
c_t = \sum_s \alpha_{ts} \tilde{h}_s
\]

\[
a_t = f(c_t, h_t) = \tanh(W_c[c_t; h_t])
\]

\[\alpha_{ts}: \text{attention weights}\]
\[c_t: \text{context vector}\]
\[a_t = \text{attention vector}\]
Attention Mechanism
# Calculating Attention Score

<table>
<thead>
<tr>
<th>Method</th>
<th>Pros</th>
<th>Cons</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dot Product</td>
<td>- Simple</td>
<td>- Input and output encoding in the same space</td>
<td>$attn_score\left(h^{(f)}_j, h^{(e)}_t\right) := h^{(f)}_j h^{(e)}_t$</td>
</tr>
<tr>
<td></td>
<td>- No additional Parameters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bilinear Functions</td>
<td>- Input and output of different sized</td>
<td>- More expensive than $\otimes$</td>
<td>$attn_score\left(h^{(f)}_j, h^{(e)}_t\right) := h^{(f)}_j W_a h^{(e)}_t$ $W_a$ is a linear transformation</td>
</tr>
<tr>
<td>MLP</td>
<td>- More flexible than $\otimes$</td>
<td></td>
<td>$attn_score\left(h^{(f)}<em>j, h^{(e)}<em>t\right) :=$ $w^{T}</em>{a2} \tanh \left(W</em>{a1}[h^{(e)}<em>t; h^{(f)}<em>j]\right)$ $W</em>{a1}$ and $w</em>{a2}$ are weight matrix and vector of the first and second layers of the MLP respectively</td>
</tr>
</tbody>
</table>
AWS Deep Learning Infrastructure Tools

P2 Instances: Up to 40K CUDA Cores

Deep Learning AMI, Preconfigured for Deep Learning mxnet, TensorFlow, …

CFM Template
Deep Learning Cluster
Apache MXNet

Most Open
Accepted into the Apache Incubator

Best On AWS
Optimized for deep learning on AWS
Amazon AI: Scaling With MXNet

88%

Efficiency

Ideal
Inception v3
Resnet

Alexnet
Sockeye Encode-Decoder Framework

- Sockeye is a sequence-to-sequence framework for Neural Machine Translation based on Apache MXNet Incubating. It implements the well-known encoder-decoder architecture with attention.
- Github repo: https://github.com/awslabs/sockeye
- Tutorials: https://github.com/awslabs/sockeye/tree/master/tutorials
- python3 -m sockeye.train -s train.source \ -t train.target \ -vs dev.source \ -vt dev.target \ --num-embed 32 \ --rnn-num-hidden 64 \ --attention-type dot \ --use-cpu \ --metrics perplexity accuracy \ --max-num-checkpoint-not-improved 3 \ -o seqcopy_model
References

- All references and images are from arXiv:1703.01619v1 [cs.CL] 5 Mar 2017 unless directly referenced on the page.
Thank you!

Cyrus M. Vahid
cyrusmv@amazon.com