Neural Machine Translation

Cyrus M. Vahid,

Principal DeepLearning Solution Architect

AWS DeepLearning

cyrusmv@amazon.com

Oct 2017

aws

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:

$$P(E|F;\theta)\ and\ \theta is\ model\ parameters$$

- 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:

$$P(E;\theta)\ and\ \theta is\ model\ parameters$$

Word by Word Computation of Probabilities

- Consider we are calculating probability of a sentence of length T being natural where the sentence is formed by words sequences: $e_1, e_2, \dots, e_T$. We should this formally as:

$$P(E) = P(|E| = T, e_1^T) = \prod_{t=1}^{T+1} P(e_t | e_1^{t-1})$$

$$T = 3 \ \therefore P(e_1, e_2, e_3) = P(e_1) P(e_2 | e_1) P(e\_3 | e_1, e_2)$$

P(|E| = 3, e_1="she", e_2="went", e_3="home") =

P(e_1="she")

* P(e_2="went" | e_1="she")

* P(e_3="home" | e_1="she", e_2="went")

* P(e_4="</s>" | e_1="she", e_2="went", e_3="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.

i am from pittsburgh .
i study at a university .
my mother is from utah .

$P(e_2=\text{am} \mid e_1=\text{i}) = c(e_1=\text{i}, e_2=\text{am})/c(e_1=\text{i}) = 1 / 2 = 0.5$

$P(e_2=\text{study} \mid e_1=\text{i}) = c(e_1=\text{i}, e_2=\text{study})/c(e_1=\text{i}) = 1 / 2 = 0.5$

More Formally:

$$P_{\text{ML}}(e_t \mid e_1^{t-1}) = \frac{c_{\text{prefix}}(e_1^t)}{c_{\text{prefix}}(e_1^{t-1})}.$$

aws

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

$$C_{prefix}(i, am, from, utah) = 0$$
$$\therefore$$
$$P(e_4 = utah | e_1 = i, e_2 = am, e_3 = from) = 0$$

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:

$$P(e_t \mid e_1^{t-1}) \approx P_{\mathrm{ML}}(e_t \mid e_{t-n+1}^{t-1}).$$

- 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 trigams 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.

$$\boldsymbol{P}(\mathcal{E}_{\boldsymbol{test}};\ \boldsymbol{\theta}) = \prod_{\boldsymbol{E}\ \epsilon\mathcal{E}_{\boldsymbol{test}}} \boldsymbol{P}(\boldsymbol{E};\boldsymbol{\theta})$$

- Example:

$|\mathcal{E}_{\boldsymbol{test}}| = \boldsymbol{10^6}$
E1 = I am from pittsburg, $\boldsymbol{P}(\boldsymbol{E};\boldsymbol{\theta}) = 2.1 \times 10^{-6}$ *
E2 = I study at a university $\boldsymbol{P}(\boldsymbol{E};\boldsymbol{\theta}) = 1.1 \times 10^{-8}$
E3 = my mother is from utah, $\boldsymbol{P}(\boldsymbol{E};\boldsymbol{\theta}) = 2 \times 10^{-9}$

$$\boldsymbol{P}(\mathcal{E}_{\boldsymbol{test}};\ \boldsymbol{\theta}) = \prod_{\boldsymbol{E}\ \epsilon\mathcal{E}_{\boldsymbol{test}}} \boldsymbol{P}(\boldsymbol{E};\boldsymbol{\theta}) = \boldsymbol{2.1 \times 10^{-6} \times 1.1 \times 10^{-8} \times 2.0 \times 10^{-9} = 4.62 \times 10^{-23}}$$

**probabilities are arbitrary just to demonstrate the range*

Evaluation of Accuracy – Log Likelihood

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

$$\log\big(P(\mathcal{E}_{test};\theta)\big) = \log_b \prod_{E\,\epsilon\,\mathcal{E}_{test}} P(E;\theta) = \sum_{E\,\epsilon\,\mathcal{E}_{test}} \log(\,P(E;\theta))\,.$$

- Example:

$|\mathcal{E}_{\boldsymbol{test}}| = \boldsymbol{10^6}$
E1 = I am from pittsburg, $\boldsymbol{P(E;\theta)} = 2.1 \times 10^{-6} \therefore \log(\mathrm{P(E};\theta) = \log(2.1) + (-6) = -5.67$ *
E2 = I study at a university $\boldsymbol{P(E;\theta)} = 1.1 \times 10^{-8} \therefore \log(\mathrm{P(E};\theta) = \log(1.1) + (-8) = -7.96$
E3 = my mother is from utah, $\boldsymbol{P(E;\theta)} = 2 \times 10^{-9} \therefore \log(\mathrm{P(E};\theta) = \log(2.0) + (-9) = -8.70$

$$\boldsymbol{P}(\mathcal{E}_{\boldsymbol{test}};\,\boldsymbol{\theta}) = \prod_{\boldsymbol{E}\,\boldsymbol{\epsilon}\mathcal{E}_{\boldsymbol{test}}} \boldsymbol{P(E;\theta)} = \sum_{E\,\epsilon\,\mathcal{E}_{test}} \log(\,P(E;\theta)) = \boldsymbol{-5.67 - 7.96\ - 8.70 = -22.33}$$

**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.

$$\text{length}(\mathcal{E}_{test}) = \sum_{E \,\epsilon\, \mathcal{E}_{test}} |E|$$

$$ppl(\mathcal{E}_{test}; \theta) = e^{-(\log(P(\mathcal{E}_{test};\theta))/lengt(\mathcal{E}_{test})}$$

- This accuracy indicator is called **perplexity**.

Neural Language Models

Motivation – MT and Context

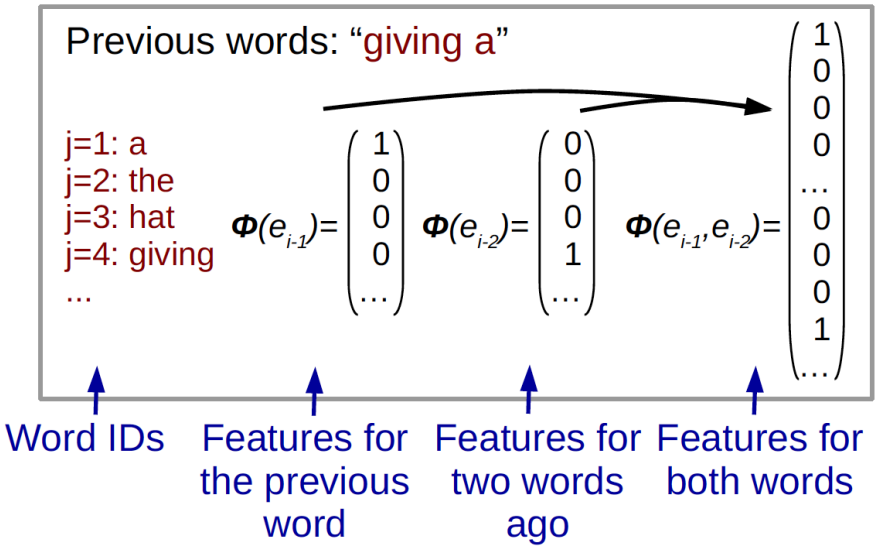
- In SMT is count-based and looses so much of hidden information.
- There are other models such as log-linear where we actually include the context based on feature vectors, but we have other problems:

farmers eat	steak → **high** hay → **low**	cows eat	steak → **low** hay → **high**
farmers grow	steak → **low** hay → **high**	cows grow	steak → **low** hay → **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

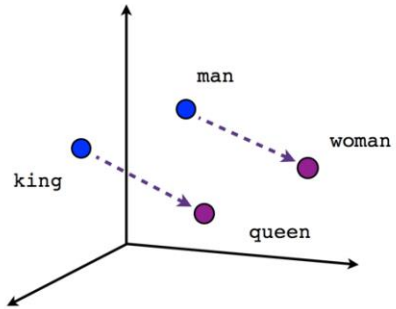
Step 1, Word Embedding

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

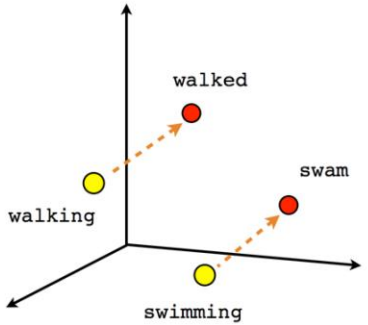


Step 1, Uncovering Combination Features

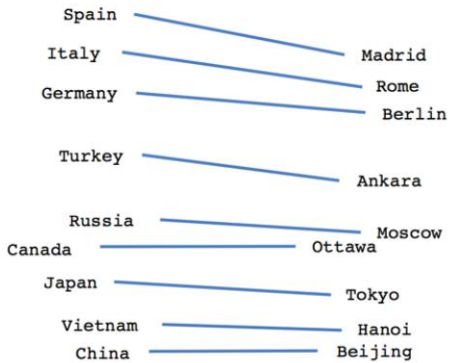
- Different elements of a vector can potentially reflect different aspect of a word. For instance, different elements could represent countability, being and animal, …
- We run the embedding vectors through hidden layer in order to learn combination features



Male-Female



Verb tense



Country-Capital

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 anaffine* 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.

$$\boldsymbol{m} = \text{concat}(M_{\cdot,e_{t-2}}, M_{\cdot,e_{t-1}})$$
$$\boldsymbol{h} = \tanh(W_{mh}\boldsymbol{m} + \boldsymbol{b}_h)$$
$$\boldsymbol{s} = W_{hs}\boldsymbol{h} + \boldsymbol{b}_s$$
$$\boldsymbol{p} = \text{softmax}(\boldsymbol{s})$$

* 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

There is Still Something Missing…

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

Seine steuerliche Bewertung wäre günstiger gewesen, wenn seine Steuererklärung ür 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.

There is Still Something Missing…

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

Who?

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.

There is Still Something Missing…

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

Who?

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

Handling Long-Term Dependencies

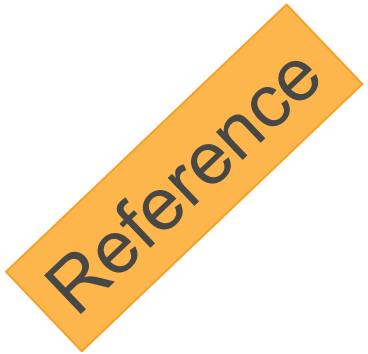
His tax assessment would have been more favorable if his tax return for fiscal year 1927 had been checked earlier.

Sequences and Long-Term Dependency

- The Spanish King officially abdicated in … of his …, Felipe. Felipe will be confirmed tomorrow as the new Spanish … .

Sequences and Long-Term Dependency

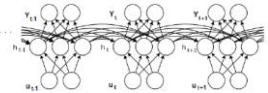
- The Spanish King officially abdicated in favour of his son, Felipe. Felipe will be confirmed tomorrow as the new Spanish King .



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

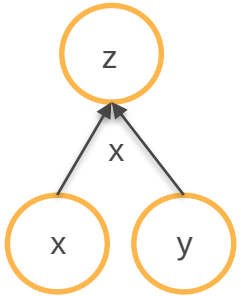


- CNN is news organization
- CNN is short for Convolutional Neural Network
- In our quest to implement perfect NLP tools, we have developed state of the art RNNs. Now we can use them to … .

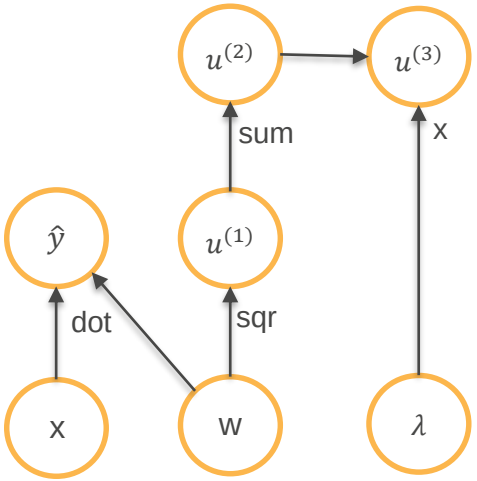
Sequences and Long-Term Dependency

- 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)

Computational Graphs



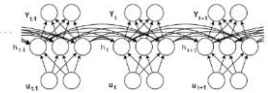
Let x and y be matrices and z the dot product;
$z = x \cdot y$



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

$weight\ decay\ penalty = \lambda \Sigma_i w_i^2$

Unfolding Computational Graph

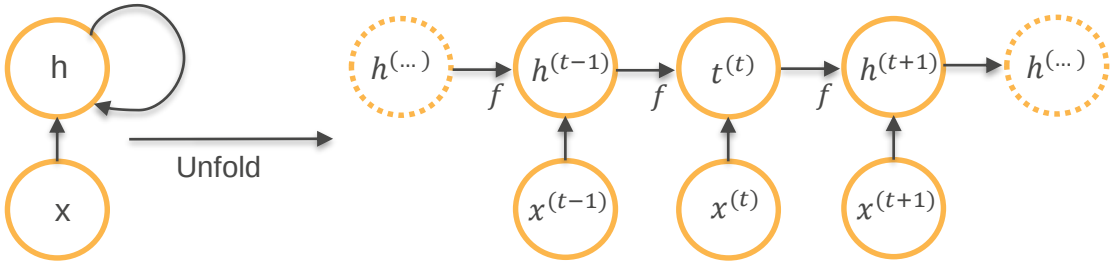


- $s^{(t)} = f\left(s^{(t-1)};\ \theta\right)$; where θ *is model params.* (A)

For instance if $t = 3$ then:

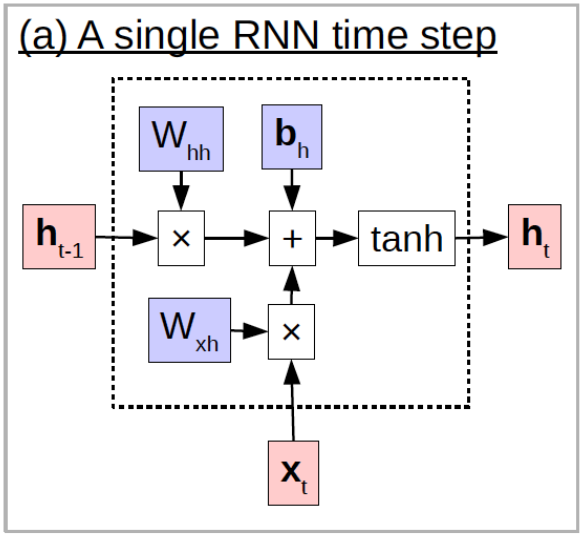
$$s^{(3)} = f\left(s^{(2)};\theta\right) = f\left(f\left(s^{(1)};\theta\right);\theta\right)$$

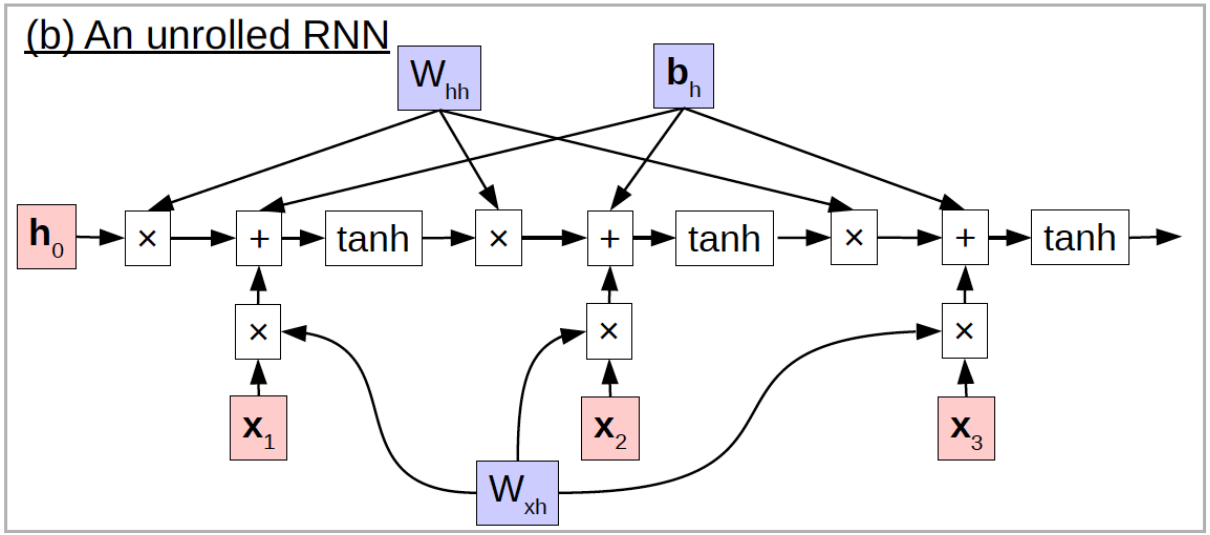
- $s^{(t)} = f\left(s^{(t-1)}, x^{(t)};\ \theta\right)$; *where x is external signal such as input vector* (B)
- $h^{(t)} = f\left(h^{(t-1)}, x^{(t)};\ \theta\right)$; *where h is the state of hidden node* (C)

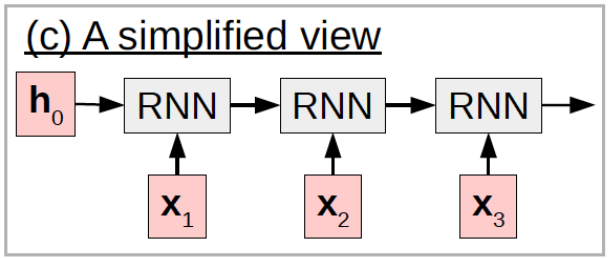


http://www.deeplearningbook.org/

RNN Unrolling Through Time







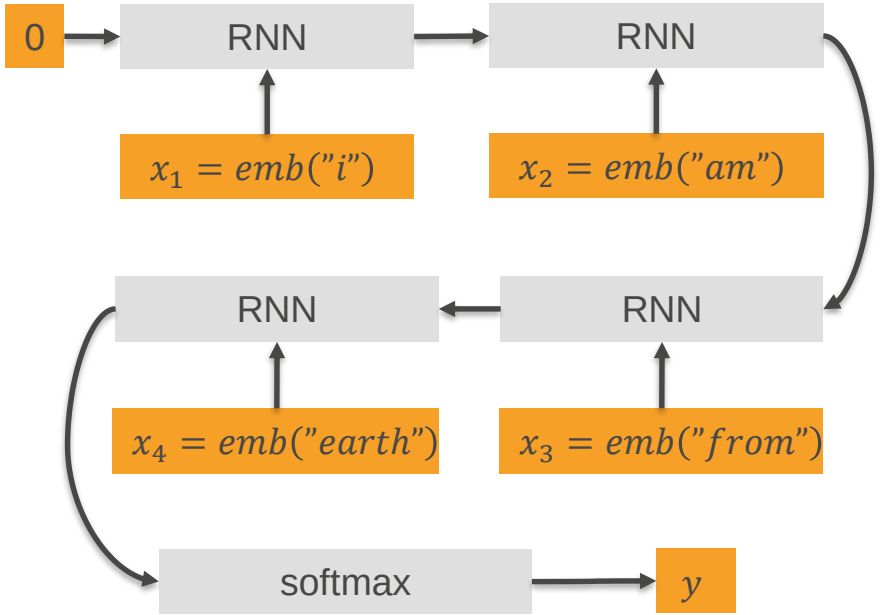
RNN Unrolling - Example

$$X = emb(\{\text{“}i\text{”}, \text{“}am\text{”}, \text{“}from\text{”}, \text{“}earth\text{”}\})$$

$$x_1 = emb(\text{”}i\text{”}), x_2 = emb(\text{”}am\text{”}),$$
$$x_3 = emb(\text{”}from\text{”}), x_4 = emb(\text{”}earth\text{”})$$

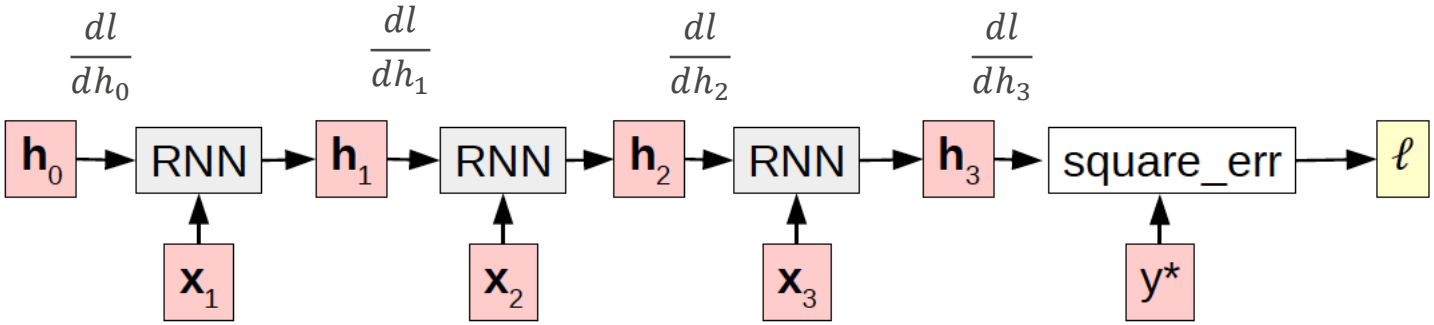
$$h(t) = \begin{cases} tanh(W_{hx}x_t + W_{hh}h_{t-1}), & t \geq 1, \\ 0 & , otherwise \end{cases}$$

$$P(t) = softmax(W_{hs}h_t + b_s)$$

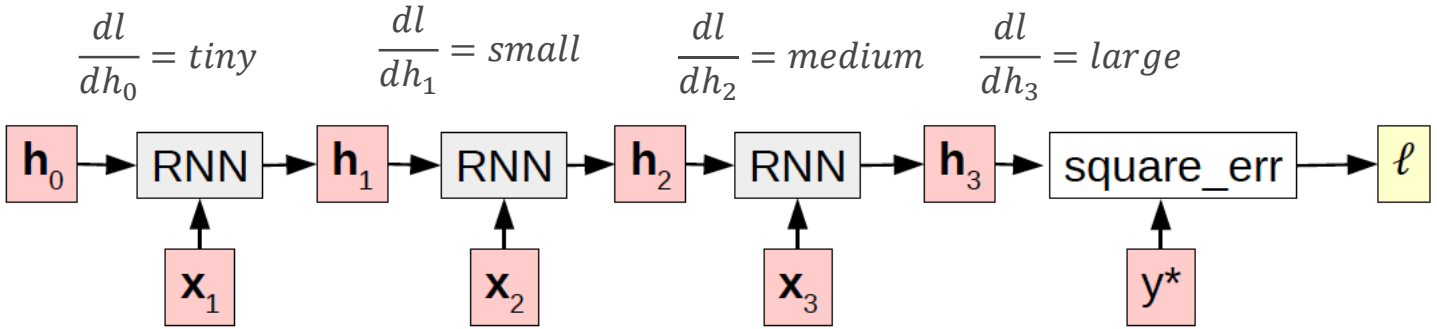


Training RNN Networks

- 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) $\frac{dl}{dh_t}$ through repeated participation in computation.

aws

Vanishing and Exploding Gradients; *the math*

$$Multipying\ weight\ Matix\ W,\ t\ times\ to\ rech\ time{-}step\ t \rightarrow W^t$$
$$Suppose\ W\ has\ decomposition\ W = Vdiag(\lambda)$$
$$W^t = (Vdiag(\lambda)V^{-1})^t = (Vdiag(\lambda)^t\ V^{-1})$$
$$When\ \lambda \neq 1\ W\ can\ become\ too\ large\ or\ too\ small.$$

- More specially in RNNs

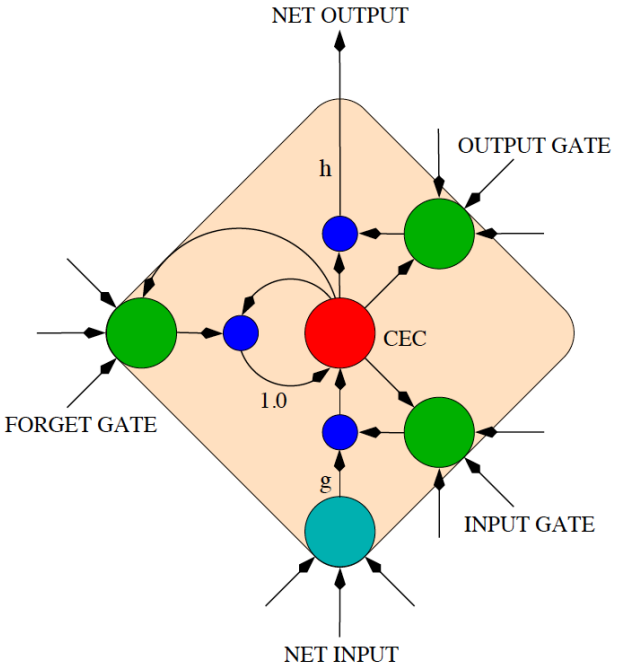
$$h^t = W^T h^{(t-1)}$$
$$h^{(t)} = (W^t)^T h^{(0)}$$
$$If\ W\ admits\ eigendecomposition\ Q\Lambda Q^T$$
$$\therefore\ h^{(t)} = Q^T \Lambda^t Q h^{(0)}$$

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

Long Short Term Memory (LSTM)

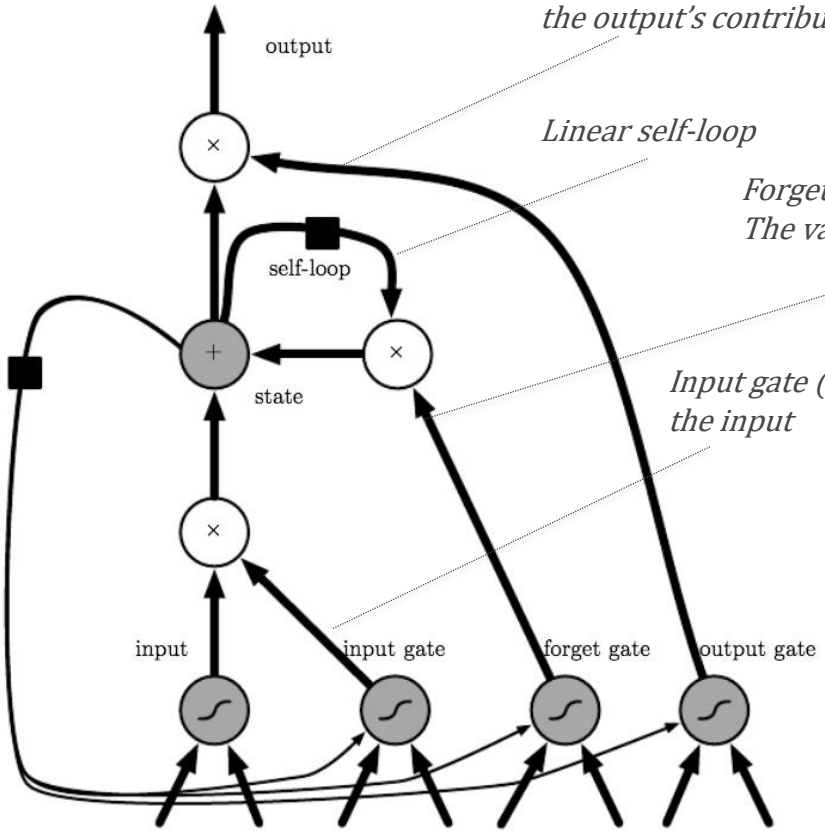
LSTM

- 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

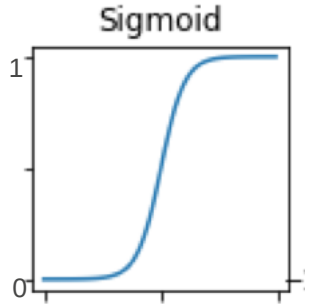


Output gate (sigmoid) controls the output's contribution

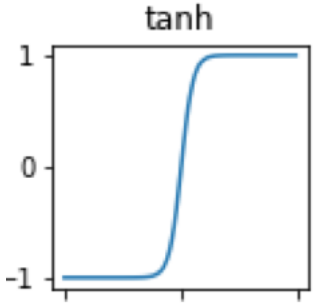
Linear self-loop

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

Input gate (non-linear) controls the input



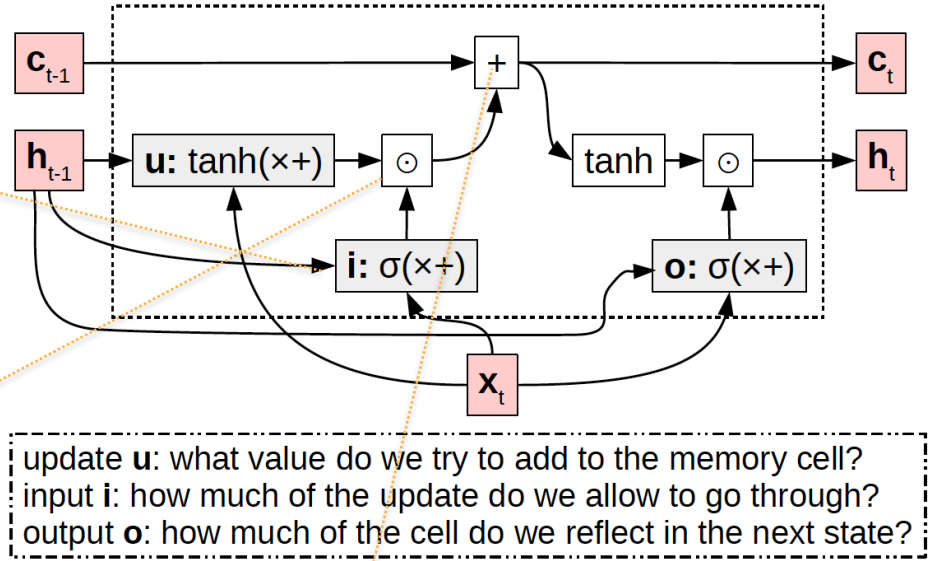
$$\sigma(x) = \frac{1}{1 - e^{-x}}$$



$$\tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

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 $\sigma \simeq 1 \Rightarrow gate\ is\ open$: it will have little effect on the input.
- if $\sigma \simeq 0 \Rightarrow gate\ is\ closed$: it will block the input.

LSTM; *the math*

How Much to update?

Gate Functions

$$\boldsymbol{u}_t = \tanh(W_{xu}\boldsymbol{x}_t + W_{hu}h_{t-1} + \boldsymbol{b}_u)$$
$$\boldsymbol{i}_t = \sigma(W_{xi}\boldsymbol{x}_t + W_{hi}h_{t-1} + \boldsymbol{b}_i)$$
$$\boldsymbol{o}_t = \sigma(W_{xo}\boldsymbol{x}_t + W_{ho}h_{t-1} + \boldsymbol{b}_o)$$
$$\boldsymbol{c}_t = \boldsymbol{i}_t \odot \boldsymbol{u}_t + \boldsymbol{c}_{t-1}$$
$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t).$$

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

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

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

aws

Adding Forget Gate

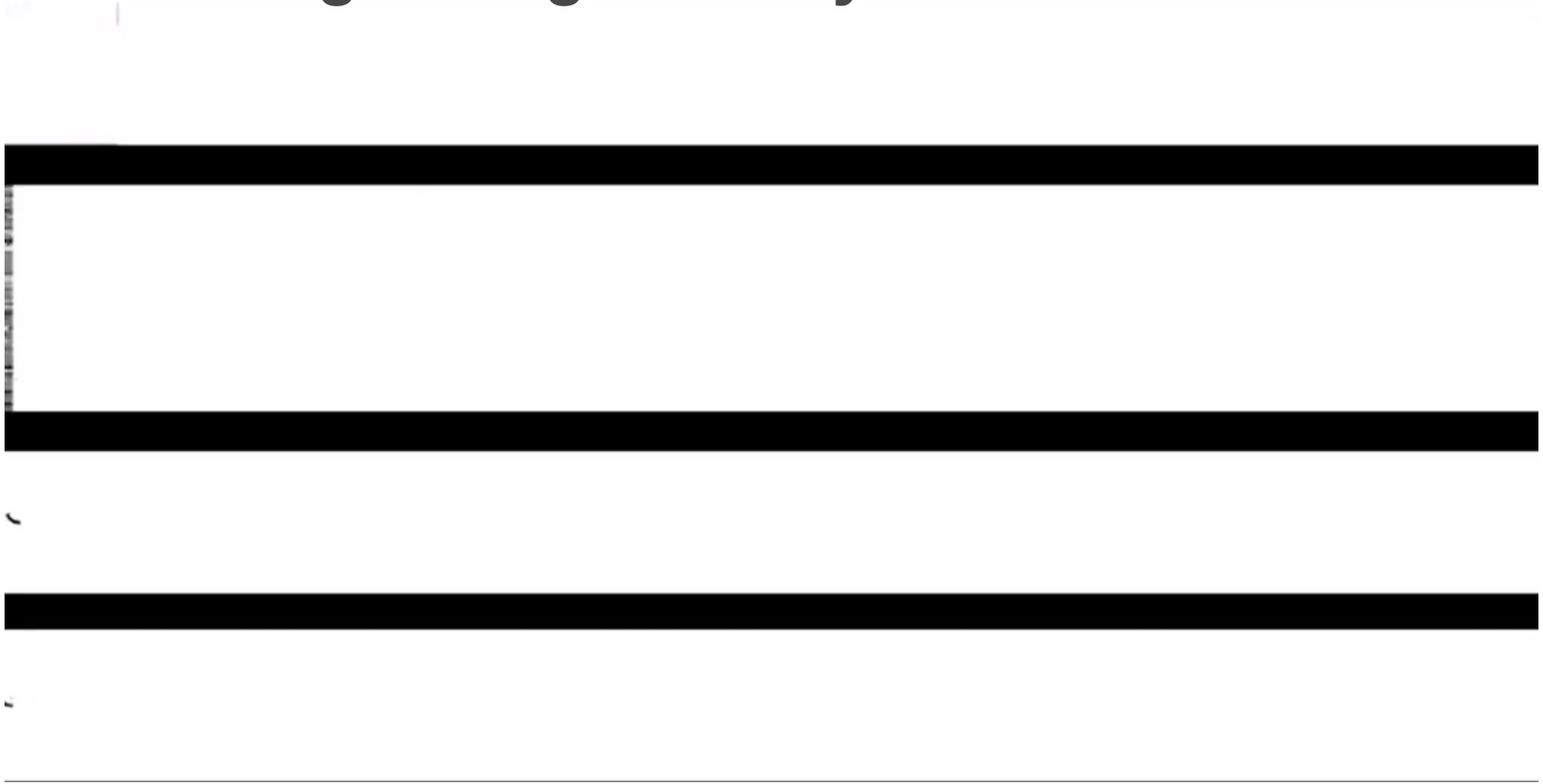
- In most common variant of LSTM, a forget gate is added, so that the value of memory can be reset. We often would like to forget memory cell values after making an inference and no longer need the long term dependencies.

$$\boldsymbol{u}_t = \tanh(W_{xu}\boldsymbol{x}_t + W_{hu}h_{t-1} + \boldsymbol{b}_u)$$
$$\boldsymbol{i}_t = \sigma(W_{xi}\boldsymbol{x}_t + W_{hi}h_{t-1} + \boldsymbol{b}_i)$$
$$\boldsymbol{f}_t = \sigma(W_{xf}\boldsymbol{x}_t + W_{hf}h_{t-1} + \boldsymbol{b}_f)$$
$$\boldsymbol{o}_t = \sigma(W_{xo}\boldsymbol{x}_t + W_{ho}h_{t-1} + \boldsymbol{b}_o)$$
$$\boldsymbol{c}_t = \boldsymbol{i}_t \odot \boldsymbol{u}_t + \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1}$$
$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t).$$

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

Modulating c_t with f_t enables the network to easily clear it memory when justified by setting *f-gate* to *0*.

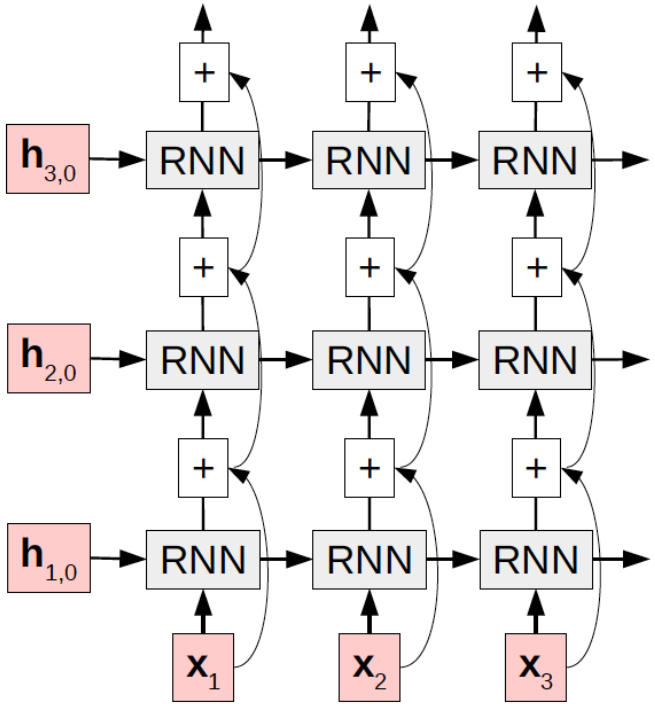
Handwriting Recognition by Alex Graves



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.

$$\boldsymbol{h}_{1,t} = \mathrm{RNN}_1(\boldsymbol{x}_t, \boldsymbol{h}_{1,t-1}) + \boldsymbol{x}_t$$
$$\boldsymbol{h}_{2,t} = \mathrm{RNN}_2(\boldsymbol{h}_{1,t}, \boldsymbol{h}_{2,t-1}) + \boldsymbol{h}_{1,t}$$
$$\boldsymbol{h}_{3,t} = \mathrm{RNN}_3(\boldsymbol{h}_{2,t}, \boldsymbol{h}_{3,t-1}) + \boldsymbol{h}_{2,t}.$$



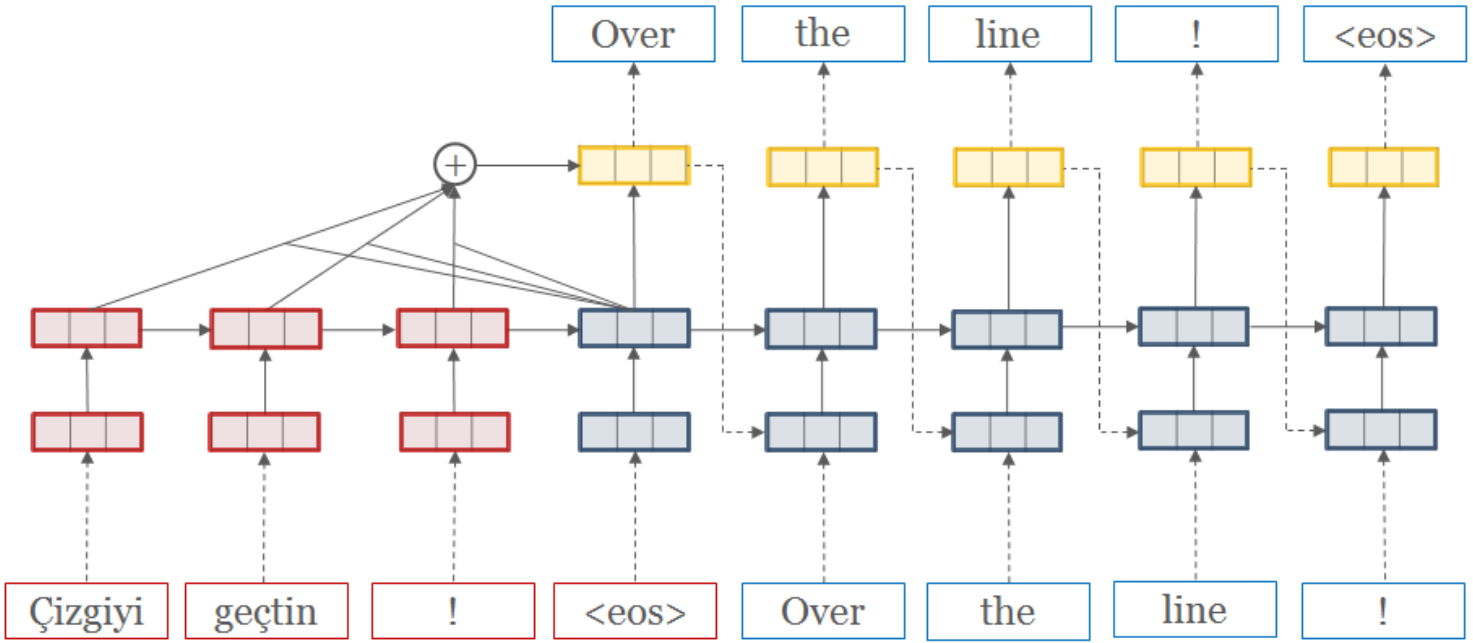
Encoder-Decoder Models

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

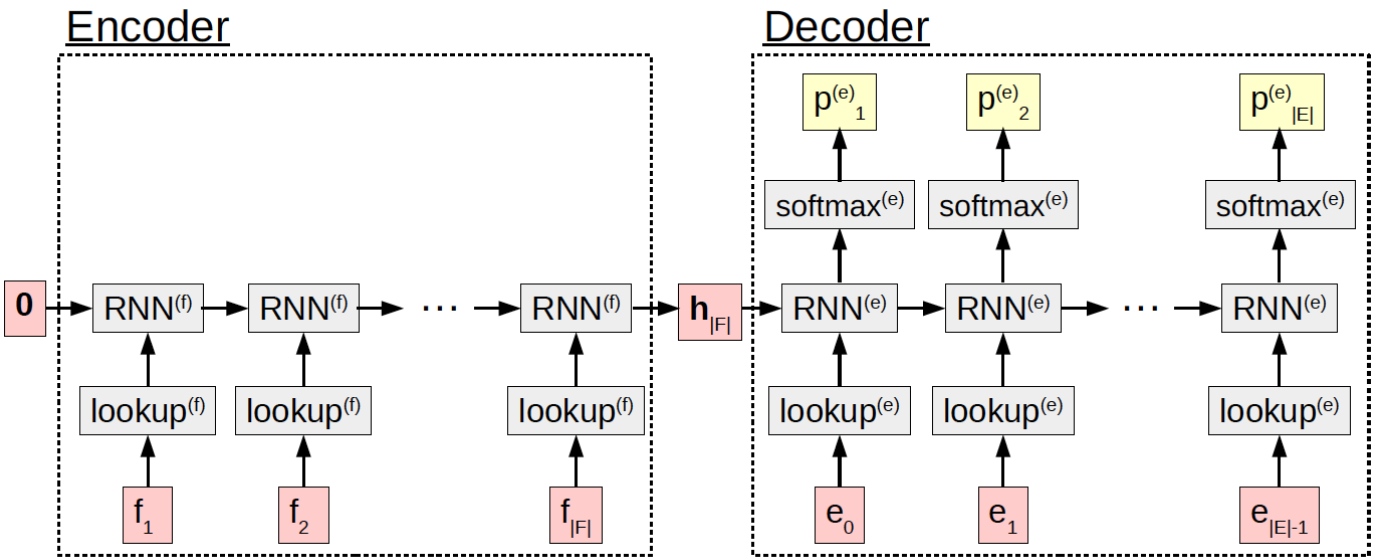


http://opennmt.net/

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 surce sentence F.
- First RNN ”encodes” source sentence F, and second RNN “decodes” the initial state into target sentence E.

Encode-Decoder Models



$$\boldsymbol{m}_t^{(f)} = M_{\cdot,f_t}^{(f)}$$

$$\boldsymbol{h}_t^{(f)} = \begin{cases} \text{RNN}^{(f)}(\boldsymbol{m}_t^{(f)}, \boldsymbol{h}_{t-1}^{(f)}) & t \geq 1, \\ \mathbf{0} & \text{otherwise.} \end{cases}$$

$$\boldsymbol{m}_t^{(e)} = M_{\cdot,e_{t-1}}^{(e)}$$

$$\boldsymbol{m}_t^{(e)} = M_{\cdot,e_{t-1}}^{(e)}$$

$$\boldsymbol{h}_t^{(e)} = \begin{cases} \text{RNN}^{(e)}(\boldsymbol{m}_t^{(e)}, \boldsymbol{h}_{t-1}^{(e)}) & t \geq 1, \\ \boldsymbol{h}_{|F|}^{(f)} & \text{otherwise.} \end{cases}$$

$$\boldsymbol{p}_t^{(e)} = \text{softmax}(W_{hs}\boldsymbol{h}_t^{(e)} + b_s)$$

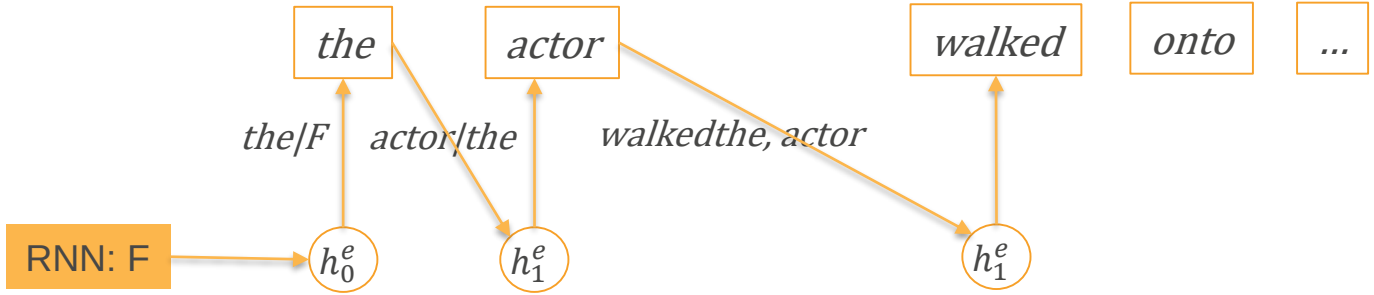
aws

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 $P(E|F)$; $\hat{E} = argmax_E\ P\ (E|F)$
    - **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.
  - $Assume\ F\ represents\ X = \{\text{“}the\text{”},\ \text{“}actor\text{”}, \text{“}walked\text{”}, \text{“}onto\text{”}, \text{”}the\text{”}, \text{“}stage\text{”}, \text{“}.\text{”}, < eos >\}$
  - $at\ time\ step\ t\ sample\ a\ word\ from\ distribution\ P(e_t|\hat{e}_1^{t-1})\ until\ x_i =< eos >$

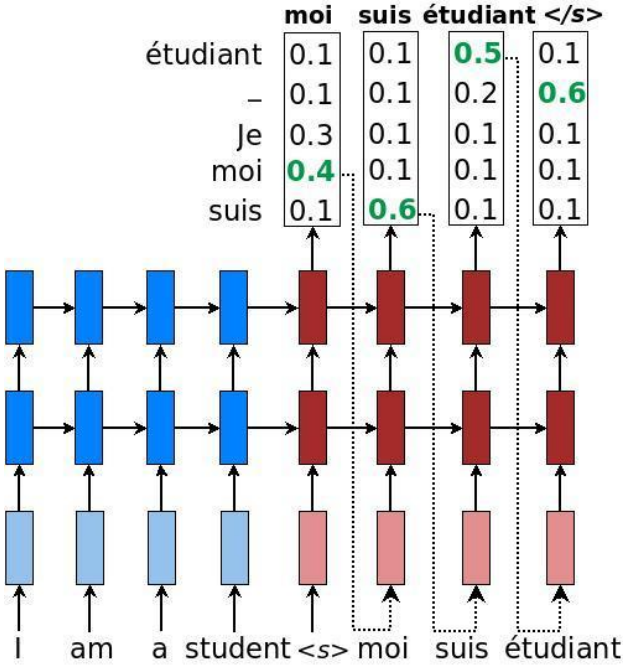


Greedy 1-Best Search

- This is very similar to random sampling with one difference: The goal is to search for the word in target corpora that maximized probability;

$$\hat{e}_t = argmax_i \log(p_{t,i}^{(e)})$$

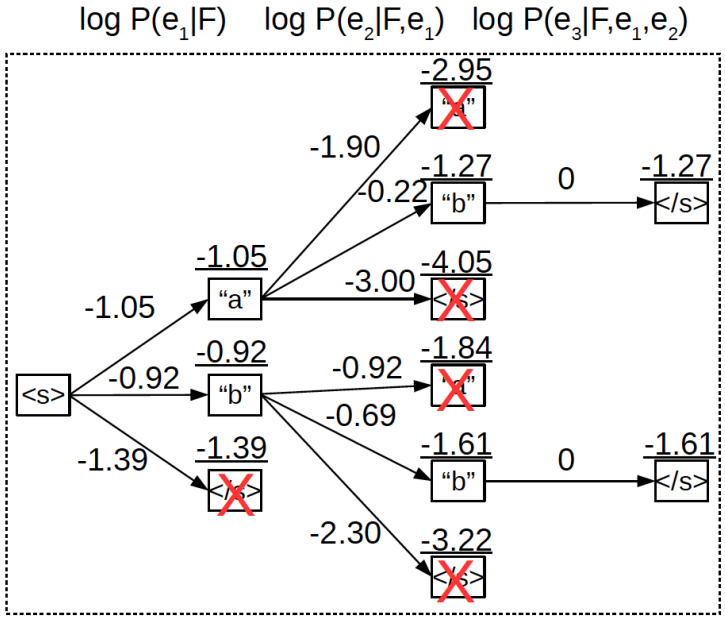
- Greedy search is not guaranteed to find the translation with highest probability, but is efficient in terms of memory and computation.



https://github.com/tensorflow/nmt

Beam Search

- Beam Search is a search method that searched for b best hypothesis at each time step. b is called width of the beam.
- 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

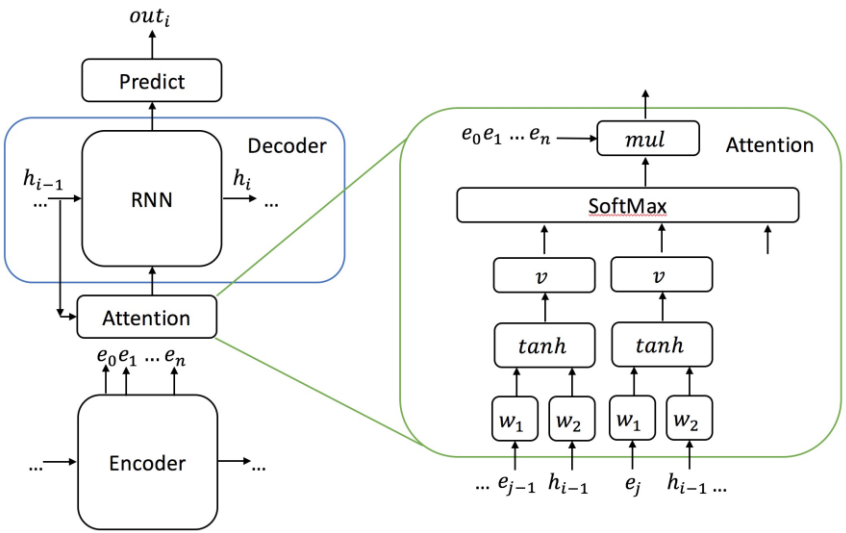
- Long distance dependencies between words that need to be translated to each other
- Storing information of sentences of arbitrary length in the same fixed-length vector.
    - Small model cannot store information about long sentences.
    - Large model is inefficient for small sentences.
    - Increasing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

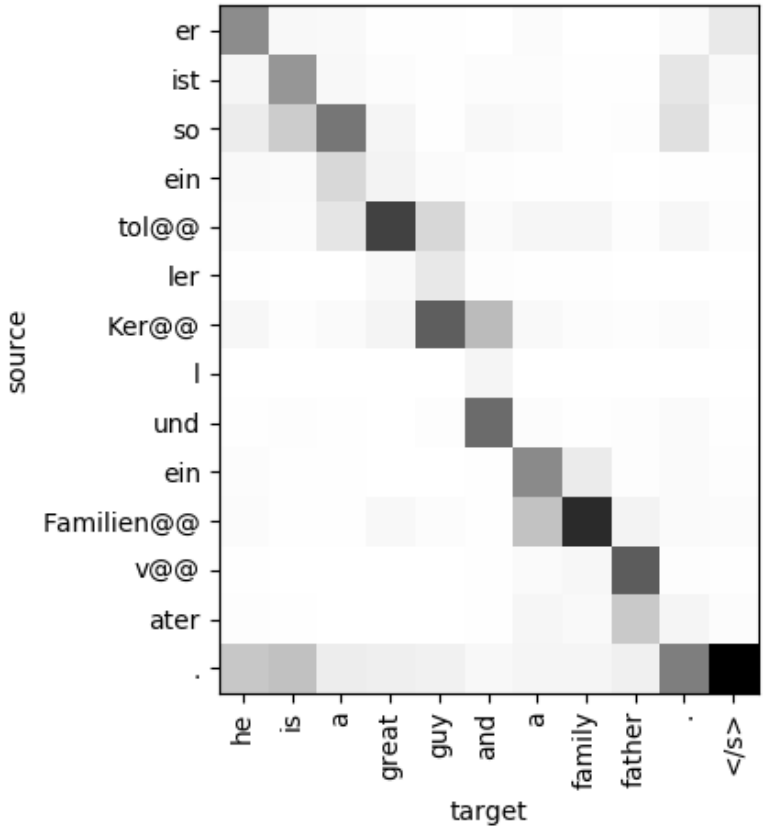
Attention

- 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

aws

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:

$$\alpha_{ts} = \frac{\exp\left(\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s)\right)}{\sum_{s'=1}^{S} \exp\left(\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_{s'})\right)}$$

$$\boldsymbol{c}_t = \sum_s \alpha_{ts} \bar{\boldsymbol{h}}_s$$

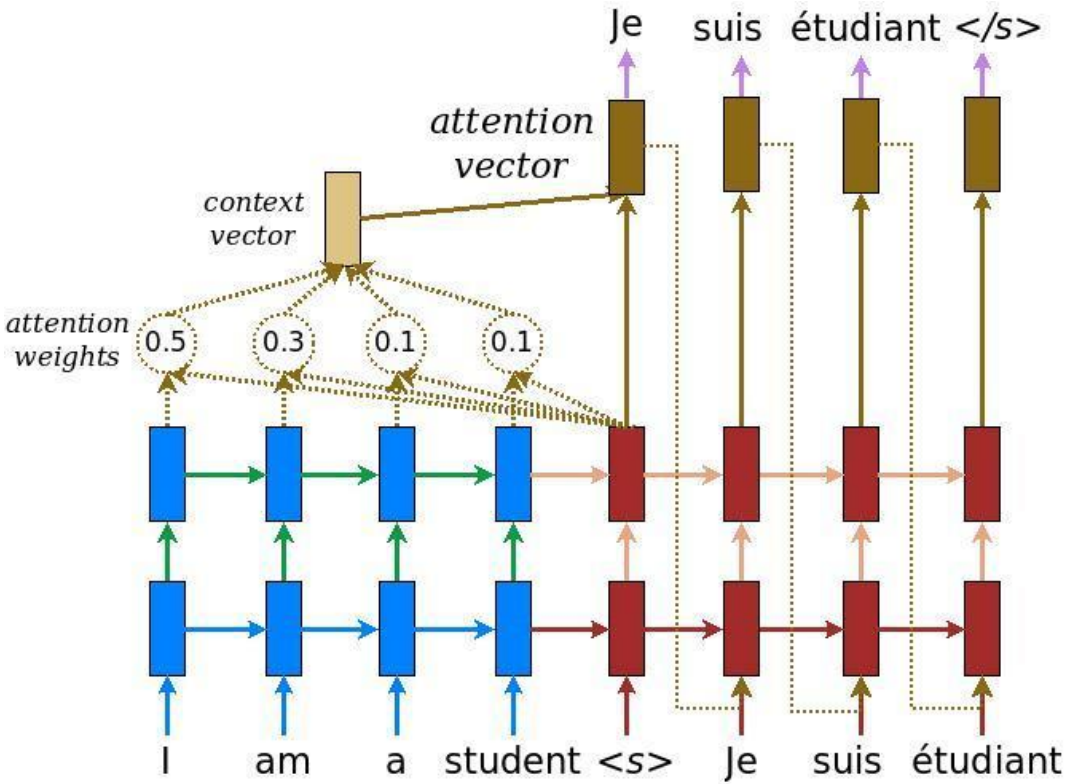
$$\boldsymbol{a}_t = f(\boldsymbol{c}_t, \boldsymbol{h}_t) = \tanh(\boldsymbol{W}_{\boldsymbol{c}}[\boldsymbol{c}_t; \boldsymbol{h}_t])$$

$\alpha_{ts}: attention\ weights$

$c_t: context\ vector$

$\alpha_t = attention\ vectorl$

Attention Mechanism

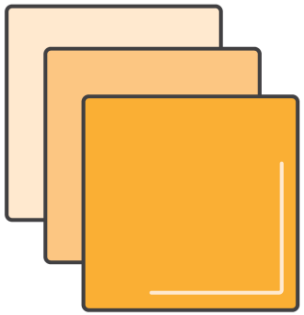


Calculating Attention Score

Method	Pros	Cons	Formula
Dot Product	- Simple - No additional Parameters	- Input and output encoding in the same space	$attn\_score\left(h_j^{(f)}, h_t^{(e)}\right) := h_j^{(f)T} h_t^{(e)}$
Bilinear Functions	- Input and output of different sized	- More expensive than $\otimes$	$attn\_score\left(h_j^{(f)}, h_t^{(e)}\right) := h_j^{(f)T} W_a h_t^{(e)}$ W_a *is a linear transformation*
MLP	- More flexible than $\otimes$ - Fewer parameters than BLF		$attn\_score\left(h_j^{(f)}, h_t^{(e)}\right) :=$ $w_{a2}^T \tanh\left(W_{a1}[h_t^{(e)}; h_j^{(f)}]\right)$ W_{a1} *and* w_{a2} *are weight matrix and vector of the first and second layers of the MLP respectively*

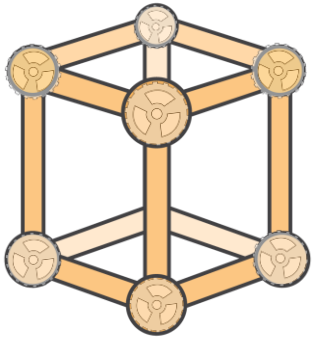

aws

AWS Deep Learning Infrastructure Tools



P2 Instances:
Up to 40K CUDA Cores

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

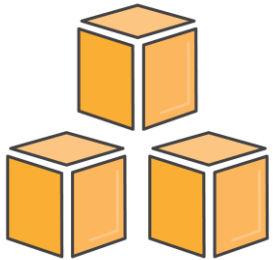


CFM Template
Deep Learning Cluster

Apache MXNet

Most Open

Accepted into the Apache Incubator



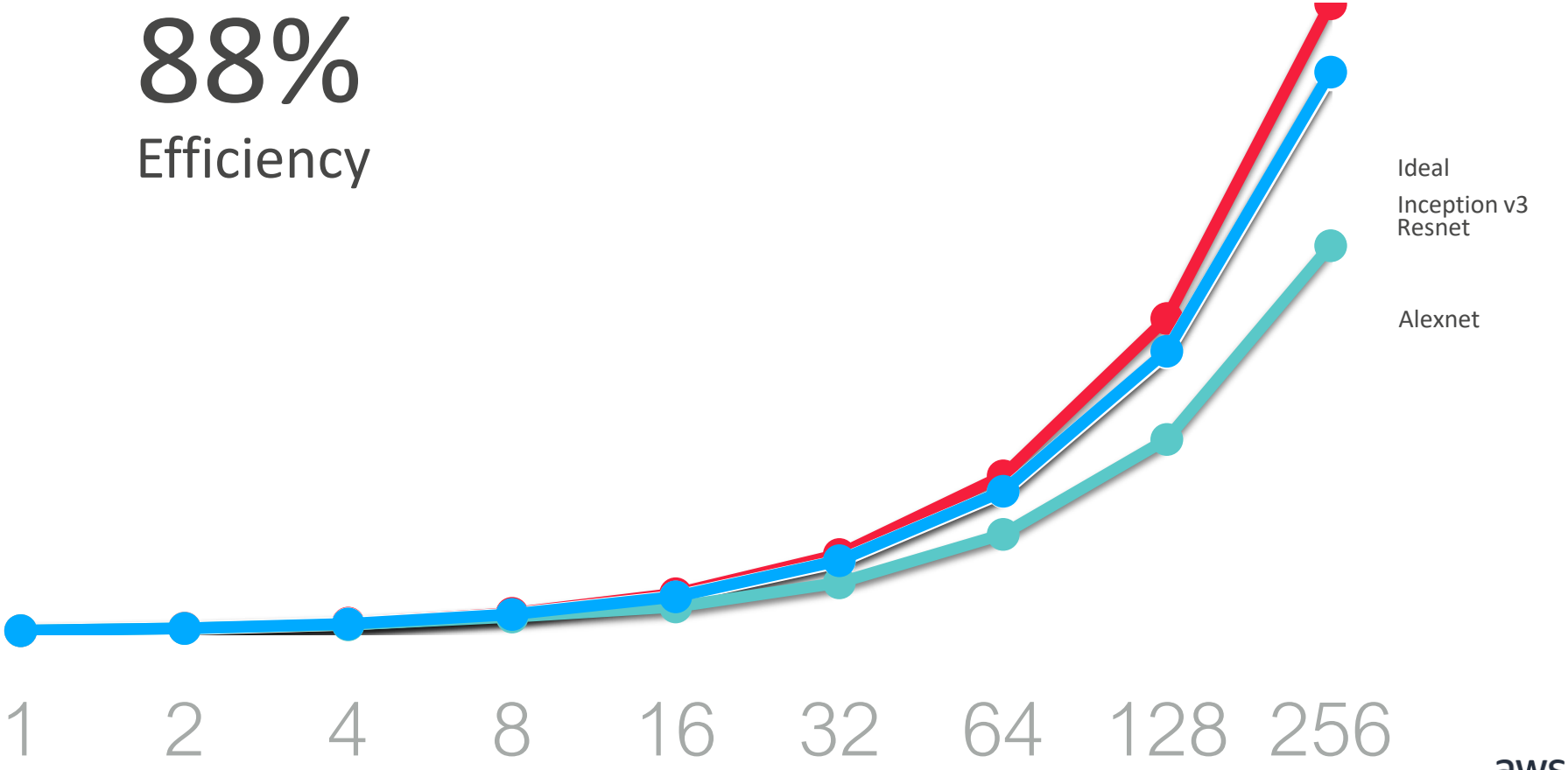
Best On AWS

Optimized for deep learning on AWS

Amazon AI: Scaling With MXNet

88%

Efficiency



aws

Sockeye Encode-Decoder Framework

- Sockeye is a 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

- arXiv:1703.01619v1 [cs.CL] 5 Mar 2017: Neural Machine Translation and Sequence-to-sequence Models: A Tutorial. Graham Neubig, Language Technologies Institute, Carnegie Mellon University
- 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

amazon web services