Sequence Learning with Neural Networks

Carolina Zheng

University of Pennsylvania

STAT 991

October 25, 2018

Overview

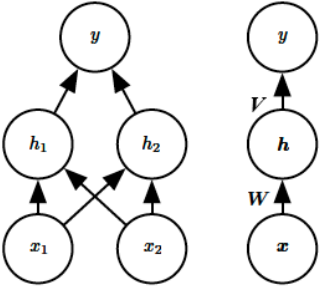
1. Motivation
2. Sequence Models
   - RNNs
   - Training Challenges
   - LSTM and GRU
3. Machine Translation
   - Attention
   - Transformer Model

Examples of problems involving sequence data

- Speech recognition
- Music generation
- Time series forecasting
- Machine translation
- Conversation agents
- Image captioning

Limitations of feedforward networks

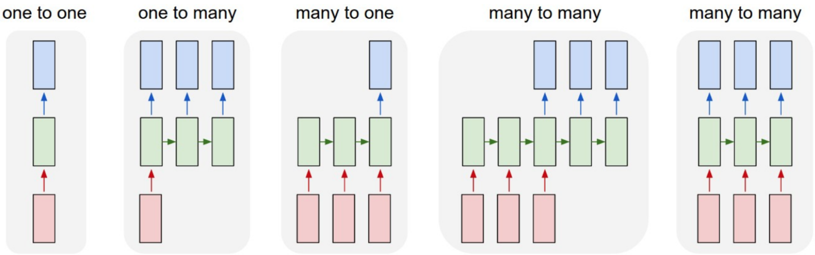
Recall the structure of a basic feedforward network:



$$h = \sigma_1(b + Wx)$$
$$o = c + Vh$$
$$\hat{y} = \mathrm{softmax}(o)$$

Problem: How to learn from a sequence of inputs $x_1, x_2, \ldots, x_\tau$?

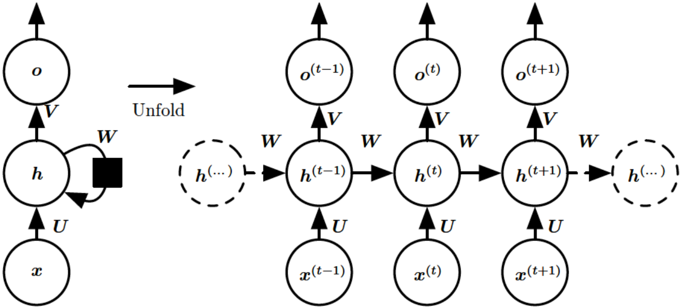
Sequence Models



We will focus on the many-to-many cases. Requirements:

1. Output $\hat{y}_t$ should depend on the sequence so far, $x_1, \ldots, x_t$.
2. We may not know the length of a particular input sequence ahead of time.

A Simple Recurrent Neural Network



$$h_t = \sigma_1(b + Wh_{t-1} + Ux_t)$$
$$o_t = c + Vh_t$$
$$\hat{y}_t = \text{softmax}(o_t)$$

A Simple Recurrent Neural Network

RNNs allow us to learn a single model, rather than a separate one for each time step.

1. The model is specified in terms of *transitions* from one state h_t to the next.
2. Parameters are shared across time.

Loss function is a sum of losses at each time step t:

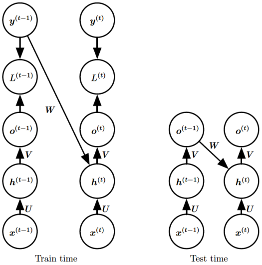
$$L(\hat{y}, y) = \sum_{t=1}^{T} \ell(\hat{y}_t, y_t)$$

Training RNNs

Training proceeds as before using **backpropagation through time** on the unrolled computation graph.

We cannot easily parallelize training since each step is dependent on the one before it.

Teacher forcing can be used when there are output-to-hidden recurrent connections. Ground truth is fed to the model instead of its own output.



Modeling Joint Probability Distributions

When we use a negative log-likelihood training objective,

$$\ell(\hat{y}_t, y_t) = -\log \Pr(y_t \mid x_1, \ldots, x_t)$$

we train the RNN to estimate the conditional distribution of the next sequence element y_t given the past inputs.

We typically use the softmax function as the output layer to obtain normalized probabilities for each class.

$$\text{softmax}(o)_i = \frac{e^{o_i}}{\sum_{j=1}^{\tau} e^{o_j}}$$

Modeling Joint Probability Distributions

RNNs can model arbitrary probability distributions of some sequence y over another sequence x.

$$\Pr(y_1, \ldots, y_\tau \mid x_1, \ldots, x_\tau) = \prod_{t=1}^{\tau} \Pr(y_t \mid x_1, \ldots, x_t)$$

To remove the conditional independence assumption, we can add output-to-hidden connections.

$$\Pr(y_1, \ldots, y_\tau \mid x_1, \ldots, x_\tau) = \prod_{t=1}^{\tau} \Pr(y_t \mid y_1, \ldots, y_{t-1}, x_1, \ldots, x_t)$$

Modeling Joint Probability Distributions

Some challenges:

1. What if we want a given output y_t to depend on the entire sequence $x_1, \ldots, x_T$?
2. How do we map a sequence x to a sequence y when they can differ in length from each other?

We will see the solutions later in the context of machine translation.

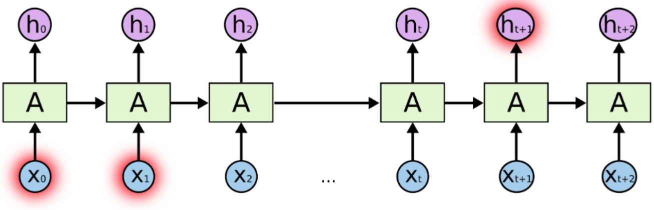
First, a fundamental problem:

RNNs have trouble learning long-term dependencies.

Learning Long-Term Dependencies

Sentence 1: "Jane walked into the room. John walked in too. Jane said hi to..."

Sentence 2: "Jane walked into the room. John walked in too. It was late in the day, and everyone was walking home after a long day at work. Jane said hi to..."



RNNs have trouble learning dependencies from inputs with a large time difference from the predicted output.

Vanishing or Exploding Gradients

Suppose we have an RNN with state h_t, input x_t, and cost $\mathcal{E}$.[1]

$$h_t = W\sigma(h_{t-1}) + Ux_t + b$$

$$\mathcal{E} = \sum_{1 \le t \le \tau} \mathcal{E}_t, \quad \mathcal{E}_t = \mathcal{L}(h_t)$$

Let's calculate the gradient over one input sequence.

$$\frac{\partial \mathcal{E}}{\partial W} = \sum_{1 \le t \le \tau} \frac{\partial \mathcal{E}_t}{\partial W} \tag{1}$$

$$\frac{\partial \mathcal{E}_t}{\partial W} = \sum_{1 \le k \le t} \left(\frac{\partial \mathcal{E}_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial^+ h_k}{\partial W} \right) \tag{2}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{t \ge i > k} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{t \ge i > k} W^\top \operatorname{diag}(\sigma'(h_{i-1})) \tag{3}$$

[1] The original paper uses this formulation instead of $h_t = \sigma(Wh_{t-1} + Ux_t + b)$ and says they are equivalent.

Vanishing or Exploding Gradients

The cause is the Jacobian matrix J. We have a product of $t-k$ Jacobians.

$$J = \frac{\partial h_{k+1}}{\partial h_k} = W^\top \text{diag}(\sigma'(h_k))$$

$$\|J\| = \left\| \frac{\partial h_{k+1}}{\partial h_k} \right\| \leq \underbrace{\left\| W^\top \right\|}_{\lambda_1} \underbrace{\left\| \text{diag}(\sigma'(h_k)) \right\|}_{\gamma} = \lambda_1 \gamma$$

λ_1 is the largest singular value of W. $\gamma = 1$ for tanh and $\gamma = \frac{1}{4}$ for sigmoid.

$$\left\| \prod_{i=k}^{t-1} \frac{\partial h_{i+1}}{\partial h_i} \right\| \leq (\lambda_1 \gamma)^{t-k}$$

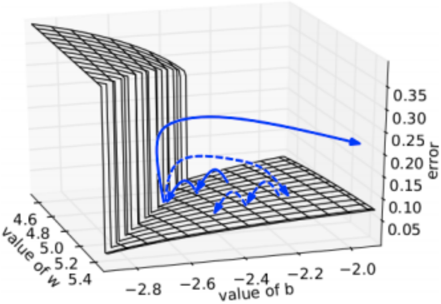
When $t \gg k$:

- If $\lambda_1 < \frac{1}{\gamma}$, the gradients *will* vanish.
- If $\lambda_1 > \frac{1}{\gamma}$, the gradients *may* explode.

How to deal with gradient problems?

1. Modify the training algorithm: **gradient clipping**.
    - Commonly used when training all variants of RNNs.
2. Use a different activation function: **ReLUs**.
    - Primarily used for deep neural nets (e.g. computer vision).
3. Use a more complex neural architecture: **LSTMs and GRUs**.

Gradient Clipping

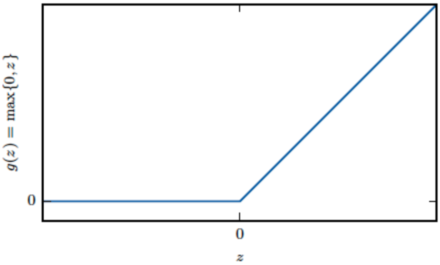


Addresses the problem of exploding gradients by preventing parameter updates from being too large.

$g \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|g\| \geq$ threshold **then**
  $g \leftarrow \frac{\text{threshold}}{\|g\|} g$
end if

Does gradient clipping affect convergence?

ReLU



Since the derivative is 1 when $z > 0$, gradients flow more easily compared to sigmoid or tanh units. Also computationally efficient.

Can lead to "dead neurons" during training. Is this a problem?

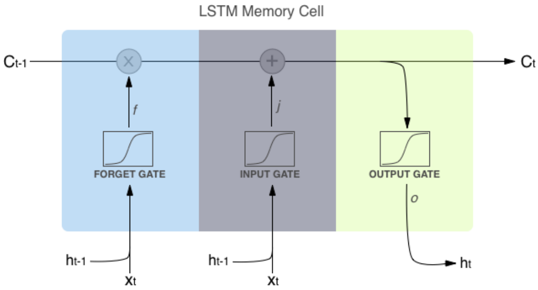
Empirically, ReLU has been shown to be very effective.

New Recurrent Architectures

We will introduce the Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) architectures.

- Address the problem of vanishing gradients.
- Considered the state of the art for many deep learning tasks.
  - Image captioning, parsing, speech recognition, machine translation, reinforcement learning

Long Short-Term Memory Network (Hochreiter 1997)



$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f)$$
$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i)$$
$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o)$$
$$a_t = \sigma_t(W_c x_t + U_c h_{t-1} + b_c)$$
$$c_t = i_t \odot a_t + f_t \odot c_{t-1}$$
$$h_t = o_t \odot \sigma_t(c_t)$$

LSTM and Backpropagating Gradients

Recall the equation to update the cell state:

$$c_t = i_t \odot a_t + f_t \odot c_{t-1}$$

The original LSTM did not have a forget gate, allowing error to flow unchanged from c_t to c_{t-1}.

- Referred to as the constant error carousel.

With a forget gate, if $f_t \approx 1$, we achieve the same effect.

- Can initialize the forget gate bias to 1 before training.

Gated Recurrent Unit (Cho et. al 2014)

$$u_t = \sigma(W_u x_t + U_u h_{t-1} + b_u)$$
$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r)$$
$$a_t = \sigma_t(W_i x_t + r_t \odot U_i h_{t-1} + b_i)$$
$$h_t = u_t \odot h_{t-1} + (1 - u_t) \odot a_t$$

Hidden units that learn to capture...

- short-term dependencies will tend to have reset gates that are frequently active.
- longer-term dependencies will have update gates that are mostly active.

Takeaway: Similar performance to LSTM, but fewer parameters.

Overview

1. Motivation
2. Sequence Models
   - RNNs
   - Training Challenges
   - LSTM and GRU
3. Machine Translation
   - Attention
   - Transformer Model

Neural Machine Translation in 2016

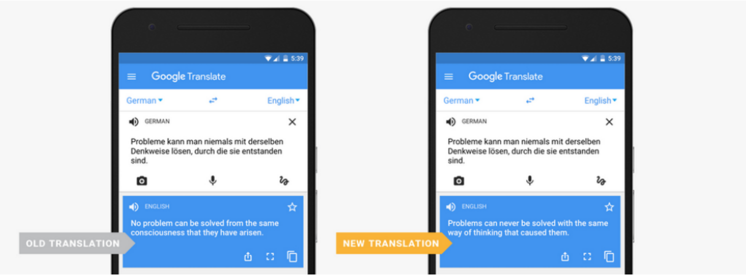
TRANSLATE

Found in translation: More accurate, fluent sentences in Google Translate

Barak Turovsky
Product Lead, Google Translate

Published Nov 15, 2016

In 10 years, Google Translate has gone from supporting just a few languages to 103, connecting strangers, reaching across language barriers and even helping people find love. At the start, we pioneered large-scale statistical machine translation, which uses statistical models to translate text. Today, we're introducing the next step in making Google Translate even better: Neural Machine Translation.



History

A bilingual translation task:
"The cat sat on the mat." $\rightarrow$ "Le chat s'est assis sur le tapis."

$* * *$

2003: Neural language model introduced by Bengio et al. This was incorporated into existing phrase-based statistical machine translation (SMT) systems.
2014: Encoder-decoder RNN introduced by Cho et. al for use in SMT.
2015: Attention model proposed by Bahdanu et. al for end-to-end neural machine translation (NMT).
2017: Transformer model proposed by Vaswani et. al. This is the current state-of-the-art.

Word Representations

How should the model encode a word token in a sequence?

Attempt 1: One-hot vectors. Represent every word as an $\mathbb{R}^{|V| \times 1}$ vector with all zero's except for a single one depending on the index of the word in the vocabulary V.

- $e(\text{cat}) = \begin{bmatrix} 0 & 0 & 0 & \ldots & 1 & \ldots & 0 \end{bmatrix}^\top$
- Does not capture semantic similarity between words!
- Scales poorly with size of vocabulary.

Word Representations

Attempt 2: Word embeddings. Model each word in a low-dimensional continuous space with dimension M.

- $e(\text{cat}) = \begin{bmatrix} 0.1 & 0.33 & 0.72 & \dots & 0.59 \end{bmatrix}^\top$
- Intuition: "A word is defined by the company it keeps."

Has revolutionized natural language processing (NLP) tasks since 2010.

- Popular word embeddings: word2vec, GloVe, ELMo

Encoder-Decoder Model

How to map an input sequence to an output sequence where the lengths are not necessarily the same?

Encoder RNN processes the input sequence and emits a context vector c, which is the model's final hidden state, h_{τ_x}.

$$h_t = f(h_{t-1}, e(x_t))$$

Decoder RNN generates the output sequence based on c and the previous output.

$$s_t = f(s_{t-1}, e(y_{t-1}), c)$$
$$\Pr(y_t) = g(s_t, e(y_{t-1}), c)$$

Jointly trained to maximize $\log \Pr_\theta(y_1, \ldots, y_{\tau_y} \mid x_1, \ldots, x_{\tau_x})$.

Task: English-to-French translation.
Training data: 348M sentences from Europarl, news commentary, etc.
Test data: 3000 sentences from a standard newstest dataset.

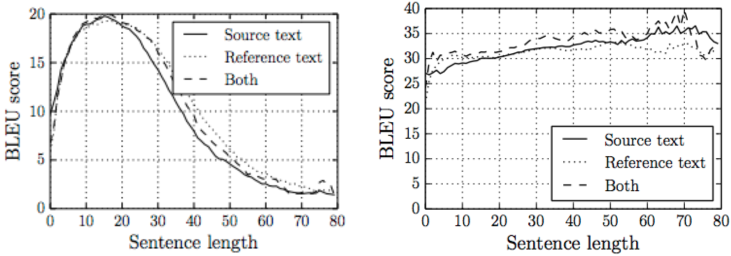


Figure: Translation quality vs. sentence length. RNN left, SMT right.

Encoder-Decoder Model

Source	She explained her new position of foreign affairs and security policy representative as a reply to a question: ”Who is the European Union? Which phone number should I call?”; i.e. as an important step to unification and better clarity of Union’s policy towards countries such as China or India.
Reference	Elle a expliqué le nouveau poste de la Haute représentante pour les affaires étrangères et la politique de défense dans le cadre d’une réponse à la question: ”Qui est qui à l’Union européenne?” ”A quel numéro de téléphone dois-je appeler?”, donc comme un pas important vers l’unicité et une plus grande lisibilité de la politique de l’Union face aux états, comme est la Chine ou bien l’Inde.
RNNEnc	Elle a décrit sa position en matière de politique étrangère et de sécurité ainsi que la politique de l’Union européenne en matière de gouvernance et de démocratie .
grConv	Elle a expliqué sa nouvelle politique étrangère et de sécurité en réponse à un certain nombre de questions : ”Qu’est-ce que l’Union européenne ? ” .
Moses	Elle a expliqué son nouveau poste des affaires étrangères et la politique de sécurité représentant en réponse à une question: ”Qui est l’Union européenne? Quel numéro de téléphone dois-je appeler?”; c’est comme une étape importante de l’unification et une meilleure lisibilité de la politique de l’Union à des pays comme la Chine ou l’Inde .

Figure: Sample output of two RNN models compared to the SMT system, Moses.

Problem: the neural network must compress all information in the source sentence into a single, fixed-length vector.

Encoder-Decoder Model with Attention

Decoder RNN is similar to before, but each context vector c_t is distinct for each target word y_t:

$$
\begin{aligned}
s_t &= f(s_{t-1}, e(y_{t-1}), c_t) \\
\Pr(y_t) &= g(s_t, e(y_{t-1}), c_t)
\end{aligned}
$$

c_t is a weighted sum of *annotations* $h_1, \ldots, h_{\tau_x}$ to which the encoder maps the input sentence.

$$
c_t = \sum_{j=1}^{\tau_x} \alpha_{tj} h_j
$$

How are α and h computed?

Encoder-Decoder Model with Attention

The encoder is a "**bidirectional RNN**."

Forward RNN reads the input as ordered and computes a sequence of forward hidden states $\overrightarrow{h}_1, \ldots, \overrightarrow{h}_{\tau_x}$.

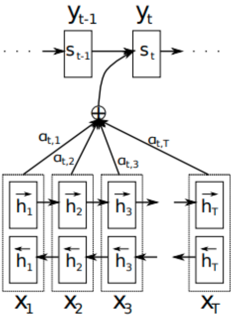
Backward RNN reads the input in reverse and computes a sequence of backward hidden states $\overleftarrow{h}_1, \ldots, \overleftarrow{h}_{\tau_x}$.

An annotation for a word x_j is the concatenation of the two hidden states.

$$h_j = \left[\overrightarrow{h}_j, \overleftarrow{h}_j \right]$$

Due to the tendency of RNNs to better represent recent inputs, the annotation h_j will be focused on the words around x_j.

Encoder-Decoder Model with Attention



To compute the weight α_{ij} of each annotation h_j, we use an **alignment model**. This is just a single-layer feedforward NN.

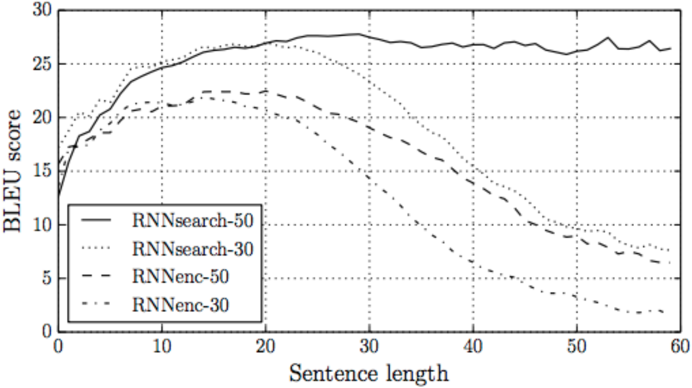
$$e_{ij} = a(s_{i-1}, h_j)$$

$$\alpha_{ij} = \text{softmax}(e_{ij})$$

The alignment model scores how well the inputs around position j and the output at position i match.

Encoder-Decoder Model with Attention

Same training and test data as before (English-to-French).



The encoder-decoder model with attention does much better on longer sentences.

Encoder-Decoder Model with Attention

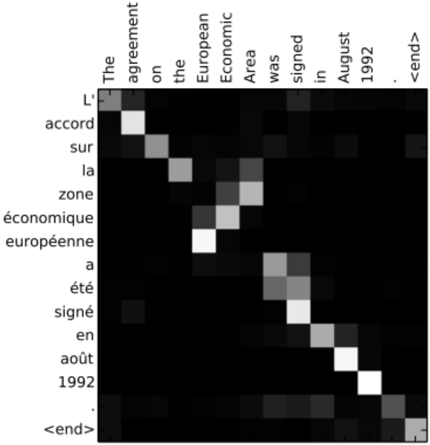


Figure: A sample alignment found by RNNsearch-50.

Transformer Model

"Attention is All You Need"

$* * *$

The transformer model utilizes an encoder/decoder architecture with attention, but no recurrent connections!

Recall that training RNNs is not parallelizable and requires more memory for each example due to the recurrence.

Also achieves state-of-the-art performance on translation tasks.

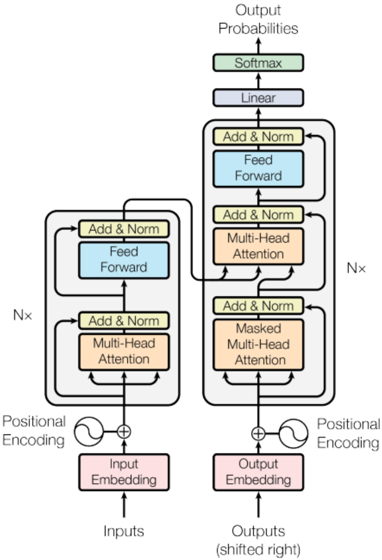
NMT Model Comparison (December 2017)

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Figure: n is sequence length, d is representation dimension, k is kernel size of convolutions, r is the size of the neighborhood in restricted self-attention.

Self-attention allows the Transformer to be trained in a more parallel fashion on GPU hardware. Typically, $d > n$.

Transformer Model



Transformer Model

No recurrent connections: the entire input sequence/output sequence so far is sent into the model at once (demo).

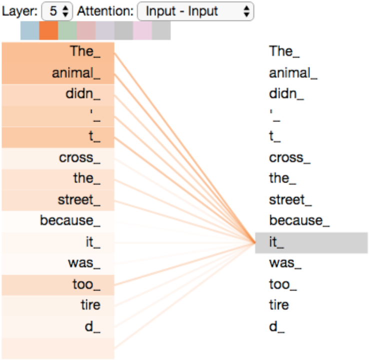
1. Input words get transformed into "positional embeddings."
2. For each encoder layer:
    1. Apply multi-headed self-attention.
    2. Send outputs through feedforward network.
3. For each output word:
    1. Transform all previous output words into positional embeddings.
        1. Apply masked multi-headed self-attention.
        2. Apply attention using the encoder outputs.
        3. Send output through feedforward network.
    2. Transform output vector into the next word using linear and softmax layers.

Self-Attention

Intuition: When the model processes a word, self-attention allows it to look at other positions in the input sequence to help it determine the optimal encoding for the word.

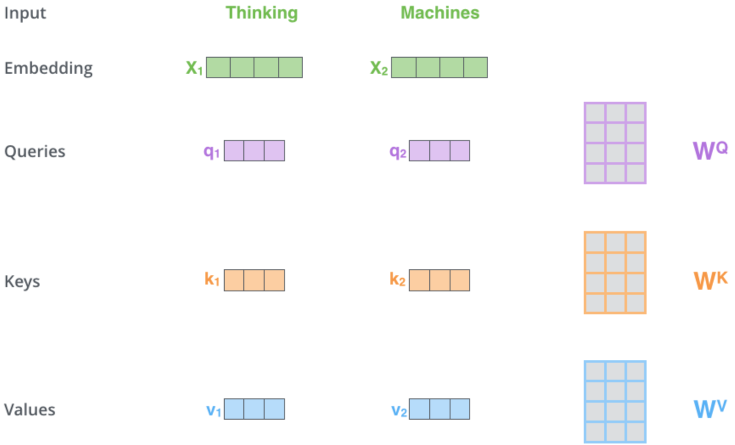
Example: "The animal didn't cross the street because it was too tired."

- Visualization



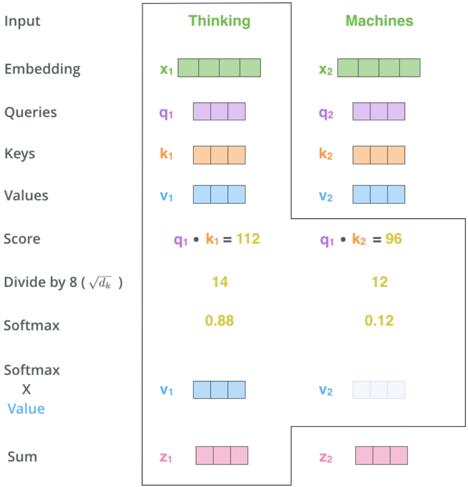
Self-Attention

Step 1: Generate query, key, and value vectors for each embedding using learned matrices W^Q, W^K, W^V.



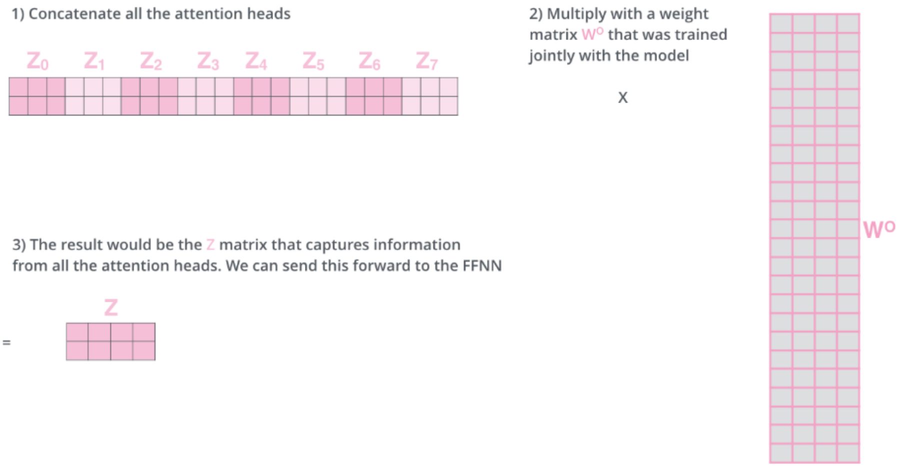
Self-Attention

Step 2: Compute a score for each key-value pair. Normalize these weights to sum to 1, and compute the output as the weighted sum of the values.

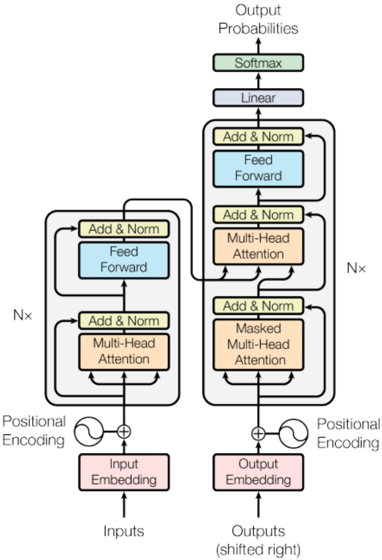


Self-Attention

Multi-headed attention means learning M attention layers in parallel (with different weight matrices).
Step 3 is combining the results using another learned matrix W^O.



Transformer Model



NMT Model Comparison (December 2017)

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

Figure: The Transformer outperforms other state-of-the-art models at a fraction of the training cost. FLOPS is floating point operations.

Future Directions for NMT

Explore attention and CNNs as an alternative to RNNs.

Many active research areas in NMT, including:

1. Sub-word or character-level models.
2. Rare word problem.
3. Low-resource languages.
4. Efficient decoding.

References I

Goodfellow, Bengio, Courville
Deep Learning
The MIT Press, 2017

Pascanu, Mikolov, Bengio (2013)
On the difficulty of training recurrent neural networks
arXiv preprint arXiv:1211.5063

Cho, Bahdanau, Bougares, Schwenk, Bengio (2014)
Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation
arXiv preprint arXiv:1406.1078

Bahdanau, Cho, Bengio (2015)
Neural Machine Translation by Jointly Learning to Align and Translate
arXiv preprint arXiv:1409.0473

References II

Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser, Polosukhin (2017)
Attention Is All You Need
arXiv preprint arXiv:1706.03762

Thank you!