# Deep Learning

## Lecture 05: Recurrent Neural Networks and Transformers

Wanxiang Che

## Recurrent Neural Networks

## Language Models

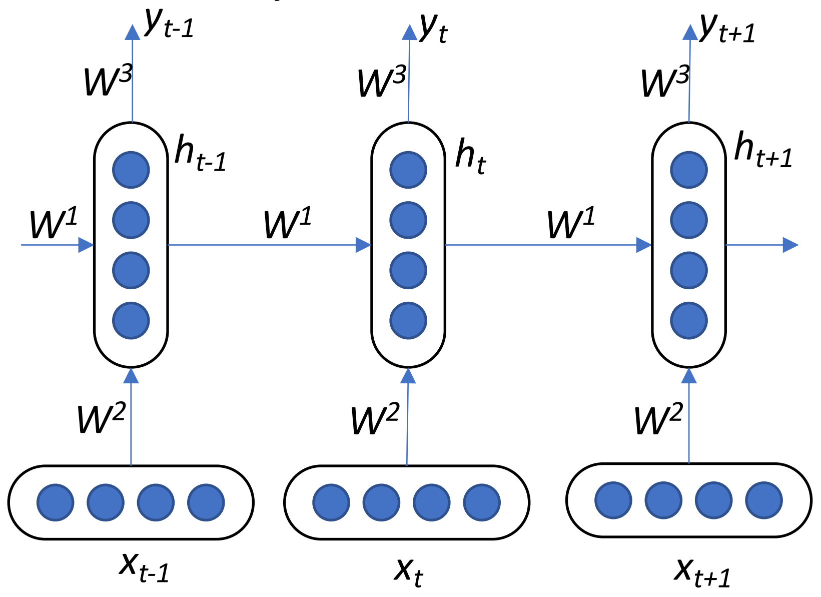
- A language model computes a probability for a sequence of word: $P(w_1, \cdots w_n)$ or predicts a probability for the next word: $P(w_{n+1}|w_1, \cdots w_n)$
- Useful for machine translation, speech recognition, and so on
  - Word ordering
    - *P*(the cat is small) > *P*(small the is cat)
  - Word choice
    - *P*(there are **four** cats) > *P*(there are **for** cats)

# Traditional Language Models

- An incorrect but necessary **Markov** assumption!
  - Probability is usually conditioned on window of *n*-1 previous words
  - $P(w_1, \cdots w_m) = \prod_{i=1}^{m} P(w_i | w_1, \cdots, w_{i-1}) \approx \prod_{i=1}^{m} P(w_i | w_{i-(n-1)}, \cdots, w_{i-1})$
- How to estimate probabilities
  - $p(w_2 | w_1) = \frac{count(w_1, w_2)}{count(w_1)}$ $\quad p(w_3 | w_1, w_2) = \frac{count(w_1, w_2, w_3)}{count(w_1, w_2)}$
- Performance improves with keeping around higher n-grams counts and doing **smoothing**, such as backoff (e.g. if 4-gram not found, try 3-gram, etc)
- Disadvantages
  - There are A LOT of n-grams!
  - Cannot see too long history
    - P(**坐/作** 了一整天的 **火车/作业**)

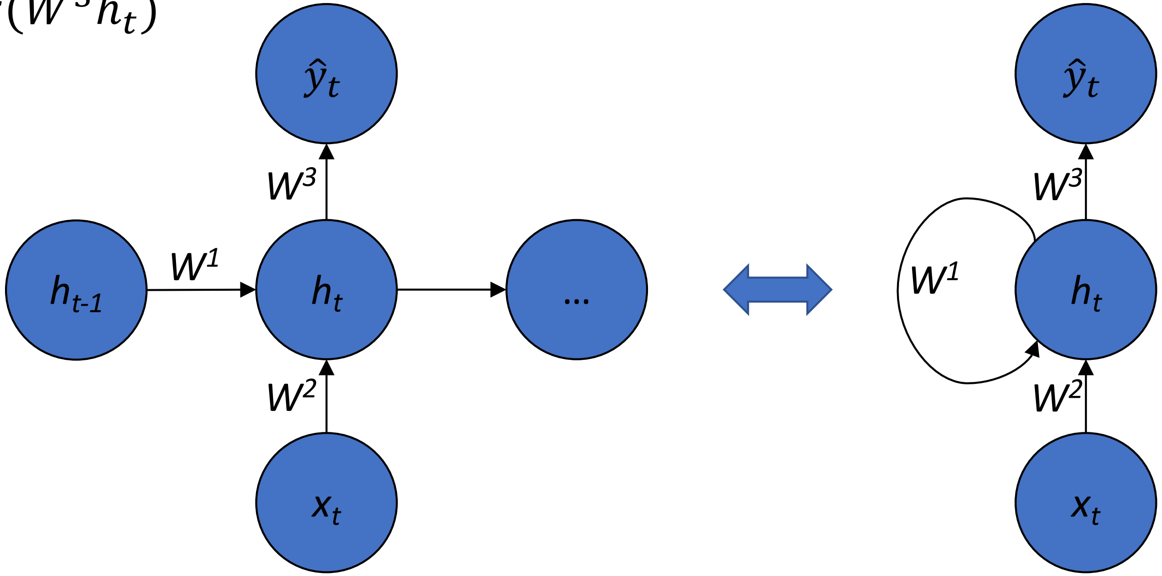
# Recurrent Neural Networks (RNNs)

- Condition the neural network on all previous inputs
- RAM requirement only scales with number of inputs



# Recurrent Neural Networks (RNNs)

- At a single time step *t*
  - $h_t = \tanh(W^1 h_{t-1} + W^2 x_t)$
  - $\hat{y}_t = softmax(W^3 h_t)$



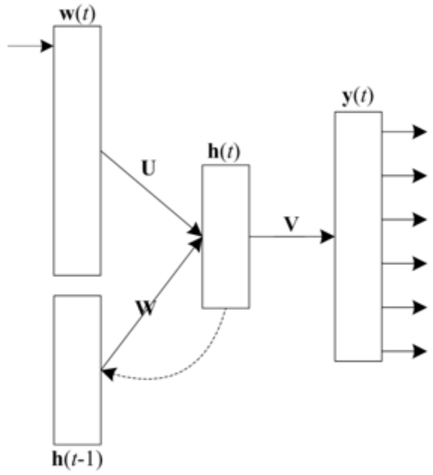
# Recurrent Neural Network Language Model (RNNLM)

- Modeling the infinite context (Mikolov et al. 2010-2013)
  - Compute:

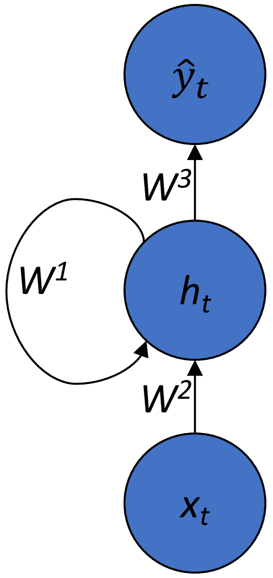
$$\mathbf{h}(t) = f(\mathbf{U}\mathbf{w}(t) + \mathbf{W}\mathbf{h}(t-1))$$

$$\mathbf{y}(t) = g(\mathbf{V}\mathbf{h}(t))$$

  - **U** is the Embedding Matrix



# BackPropagation Through Time (BPTT)



- Total error is the sum of each error at time step *t*
  - $\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W}$
- $\frac{\partial E_t}{\partial W^3} = \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial W^3}$ is easy to be calculated
- But to calculate $\frac{\partial E_t}{\partial W^1} = \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial W^1}$ is hard (also for $W^2$)
- Because $h_t = \tanh(W^1 h_{t-1} + W^2 x_t)$ depends on $h_{t-1}$, which depends on $W^1$ and $h_{t-2}$, and so on.

## BackPropagation Through Time (BPTT)

- Use the same as the backpropagation algorithm as we use in deep feedforward NN, but summing up the gradients for $W^1$
- BPTT is just a fancy name for standard backpropagation on an unrolled RNN
- $\frac{\partial E}{\partial W^1} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W^1}$
- $\frac{\partial E_t}{\partial W^1} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W^1}$

# The vanishing gradient problem

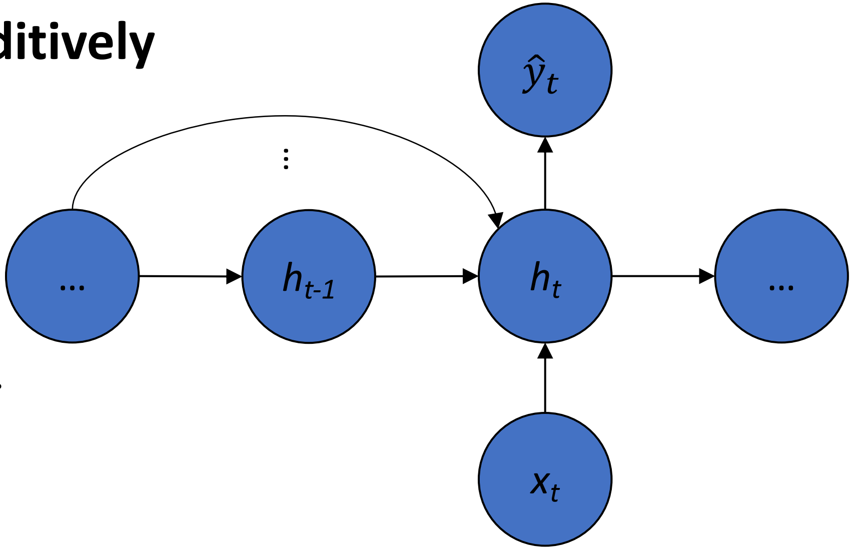
- $\frac{\partial E_t}{\partial W^1} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \boxed{\frac{\partial h_t}{\partial h_k}} \frac{\partial h_k}{\partial W^1}$, $h_t = \tanh(W^1 h_{t-1} + W^2 x_t)$
- $\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} = \prod_{j=k+1}^{t} W^1 \mathrm{diag}[\tanh'(\cdots)]$
- $\left\| \frac{\partial h_t}{\partial h_{t-1}} \right\| \leq \gamma \|W^1\| \leq \gamma \lambda_1$
  - where $\gamma$ is bound $\|\mathrm{diag}[\tanh'(\cdots)]\|$, $\lambda_1$ is the largest singular value of $W^1$
- $\left\| \frac{\partial h_t}{\partial h_k} \right\| \leq (\gamma \lambda_1)^{t-k}$→0, if $\gamma \lambda_1 < 1$
- This can become very small or very large quickly → Vanishing or exploding gradient
  - Trick for exploding gradient: clipping trick (set a threshold)

# A “solution”

- Intuition
  - Ensure $\gamma\lambda_1 \geq 1$ → to prevent vanishing gradients
- So …
  - Proper initialization of the W
  - To use ReLU instead of tanh or sigmoid activation functions

# A better “solution”

- Recall the original transition equation
  - $h_t = \tanh(W^1 h_{t-1} + W^2 x_t)$
- We can instead update the state **additively**
  - $u_t = \tanh(W^1 h_{t-1} + W^2 x_t)$
  - $h_t = h_{t-1} + u_t$
  - then, $\left\| \frac{\partial h_t}{\partial h_{t-1}} \right\| = 1 + \left\| \frac{\partial u_t}{\partial h_{t-1}} \right\| \geq 1$
  - On the other hand
    - $h_t = h_{t-1} + u_t = h_{t-2} + u_{t-1} + u_t = \cdots$



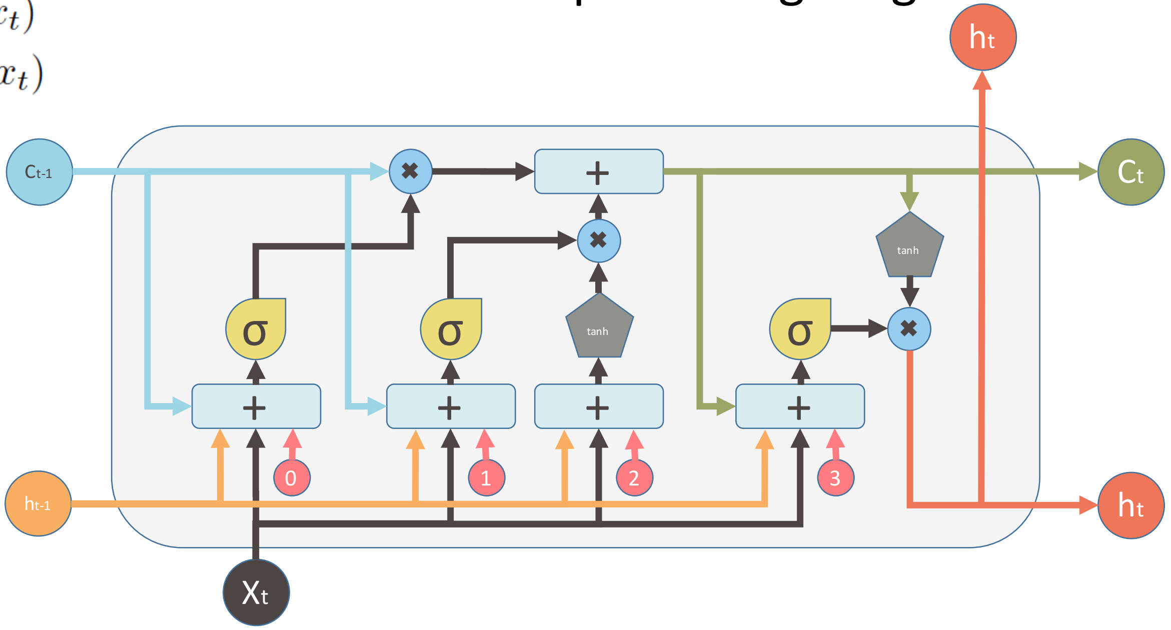
## A better "solution" (cont.)

- Interpolate between old state and new state ("choosing to **forget**")
  - $f_t = \sigma\left(W^f x_t + U^f h_{t-1}\right)$
  - $h_t = f_t \odot h_{t-1} + (1 - f_t) \odot u_t$
- Introduce a separate **input gate** $i_t$
  - $i_t = \sigma\left(W^i x_t + U^i h_{t-1}\right)$
  - $h_t = f_t \odot h_{t-1} + i_t \odot u_t$
- Selectively expose memory cell $c_t$ with an **output gate** $o_t$
  - $o_t = \sigma(W^o x_t + U^o h_{t-1})$
  - $c_t = f_t \odot c_{t-1} + i_t \odot u_t$
  - $h_t = o_t \odot \tanh(c_t)$

# Long Short-Term Memory (LSTM)

$$
\begin{aligned}
u_t &= \tanh\left(W h_{t-1} + V x_t\right) \\
f_t &= \mathrm{sigmoid}\left(W_f h_{t-1} + V_f x_t\right) \\
i_t &= \mathrm{sigmoid}\left(W_i h_{t-1} + V_i x_t\right) \\
o_t &= \mathrm{sigmoid}\left(W_o h_{t-1} + V_o x_t\right) \\
c_t &= f_t \odot c_{t-1} + i_t \odot u_t \\
h_t &= o_t \odot \tanh(c_t) \\
y_t &= U h_t
\end{aligned}
$$

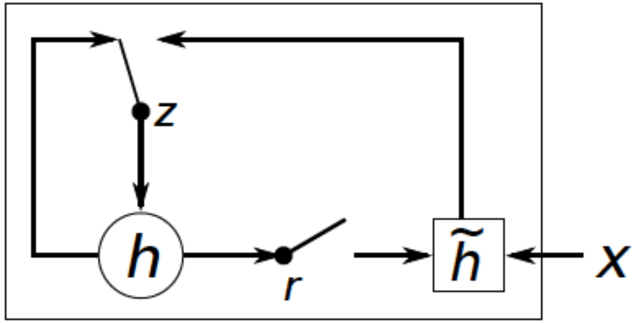
- Hochreiter & Schmidhuber, 1997
- LSTM = additive updates + gating



## Gated Recurrent Unites, GRU (Cho et al. 2014)

- Main ideas
  - Keep around memories to capture long distance dependencies
  - Allow error messages to flow at different strengths depending on the inputs
- Update gate
  - Based on current input and hidden state
  - $z_t = \sigma(W^z x_t + U^z h_{t-1})$
- Reset gate
  - Similarly but with different weights
  - $r_t = \sigma(W^r x_t + U^r h_{t-1})$

# GRU



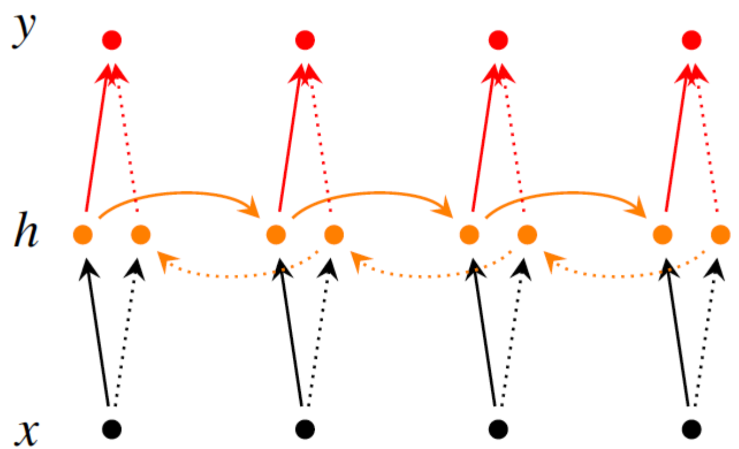
- New memory content
  - $\tilde{h}_t = \tanh(Wx_t + r_t \odot Uh_{t-1})$
- Final memory at time step combines current and previous time steps
  - $h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}$
  - Update gate *z* controls how much of past state should matter now
    - If *z* closed to 1, then we can copy information in that unit through many time steps → less vanishing gradient!
  - If reset gate *r* unit is close to 0, then this ignores previous memory and only stores the new input information → allows model to drop information that is irrelevant in the future
  - Units with long term dependencies have active update gates *z*
  - Units with short-term dependencies often have reset gates *r* very active

## LSTM vs. GRU

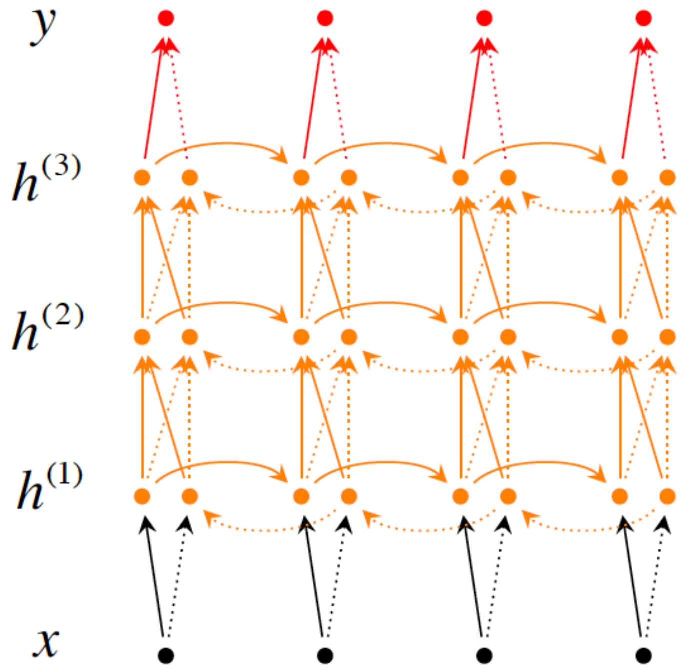
- No clear winner!
- Tuning hyperparameters like layer size is probably more important than picking the ideal architecture
- GRUs have fewer parameters and thus may train a bit faster or need less data to generalize
- If you have enough data, the greater expressive power of LSTMs may lead to better results.

# More RNNs

- Bidirectional RNN

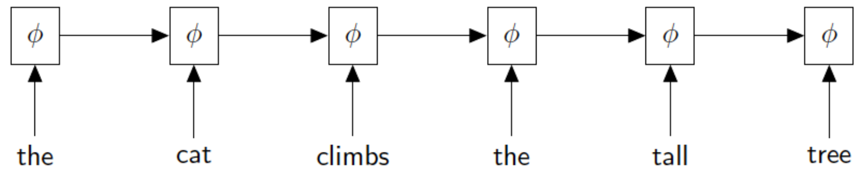


- Deep Bidirectional RNN

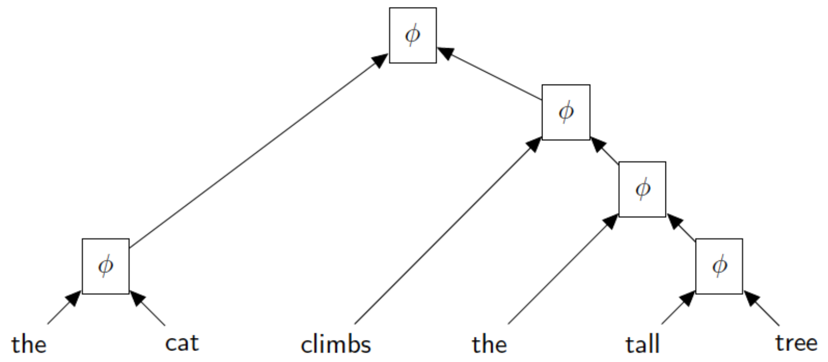


## Tree-LSTMs

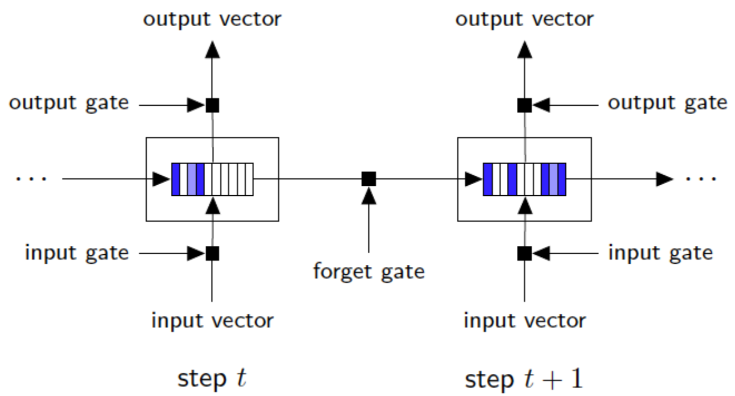
- Traditional Sequential Composition

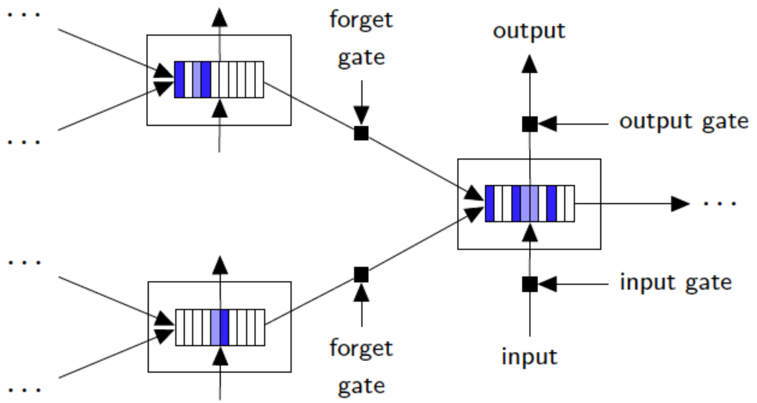


- Tree-Structured Composition



# Tree-Structured LSTMs





- Standard LSTMs
  - Each node has one child
- Tree-LSTMs
  - Accept multiple children

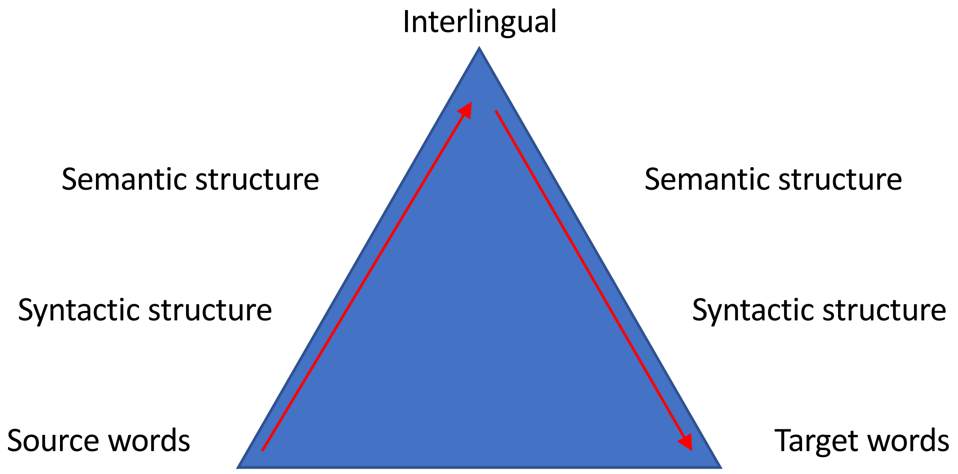
# Conditioned Language Models

- Not just generate text, generate text according to some specification

| Input *X* | Output *Y* (**Text**) | Task |
|---|---|---|
| Structured Data | NL Description | NL Generation |
| English | Japanese | Translation |
| Document | Short Description | Summarization |
| Utterance | Response | Response Generation |
| Image | Text | Image Captioning |
| Speech | Transcript | Speech Recognition |

# Neural Machine Translation

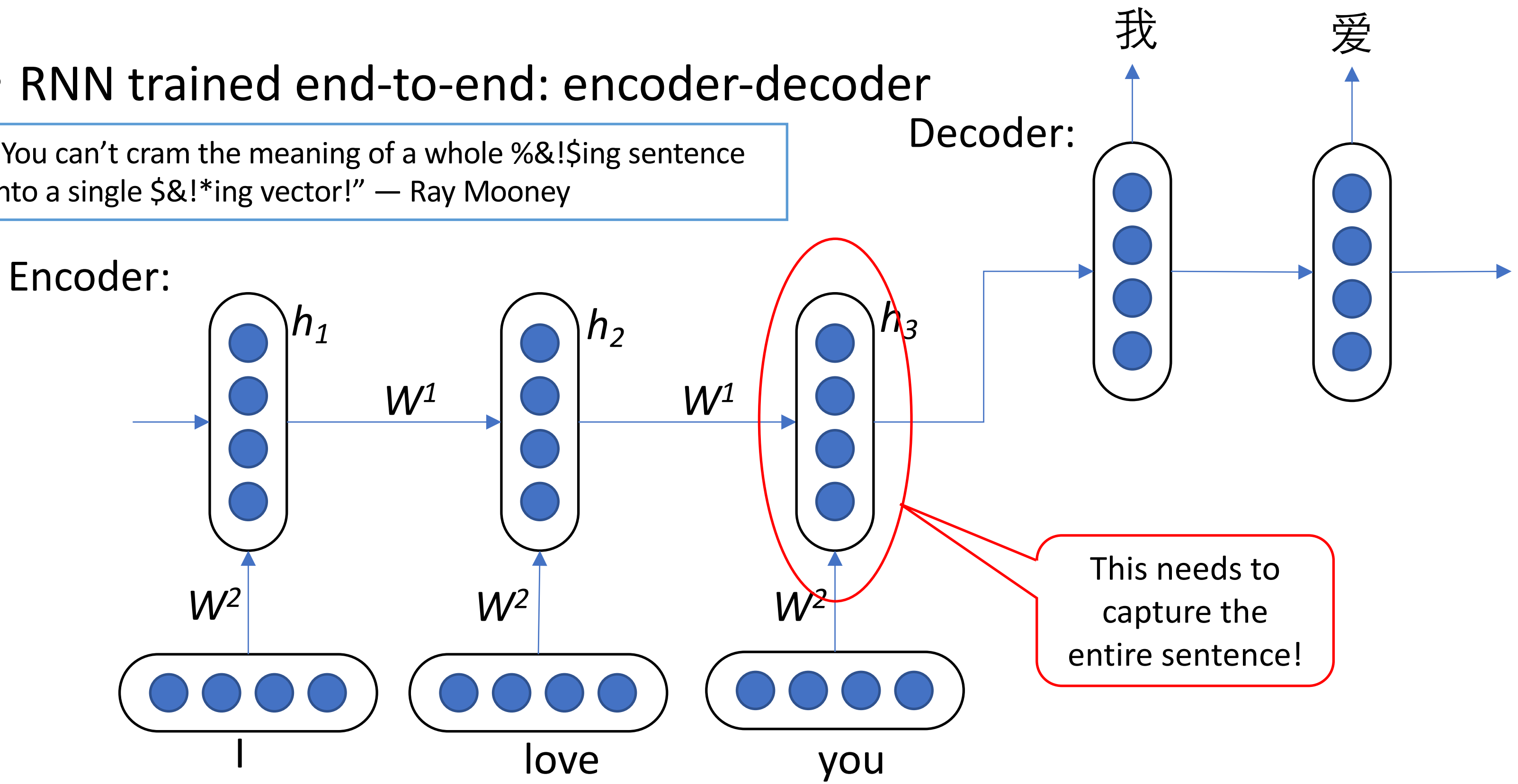
- The Vauquois Diagram (1968)



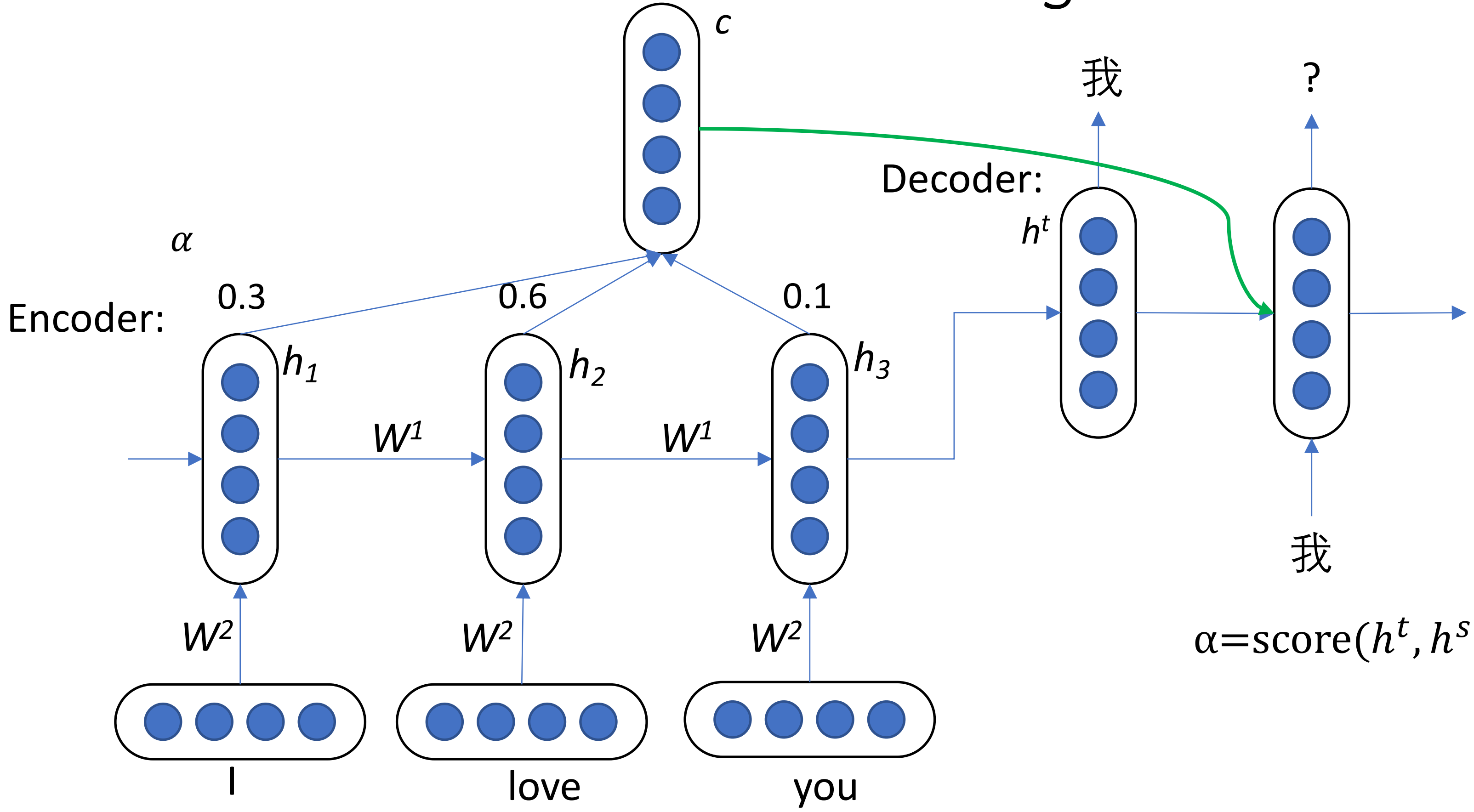
# Neural Machine Translation

- RNN trained end-to-end: encoder-decoder

"You can't cram the meaning of a whole %&!$ing sentence into a single $&!*ing vector!" — Ray Mooney



# Attention Mechanism – *Scoring*

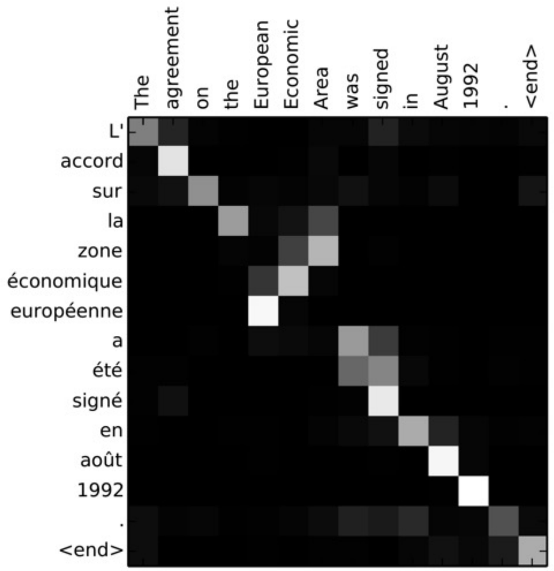


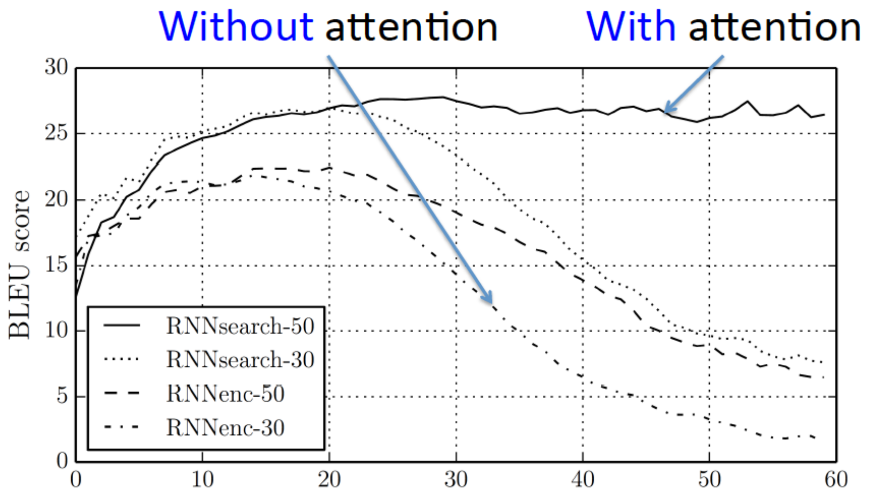
α=score($h^t, h^s$)

# Attention Score Functions

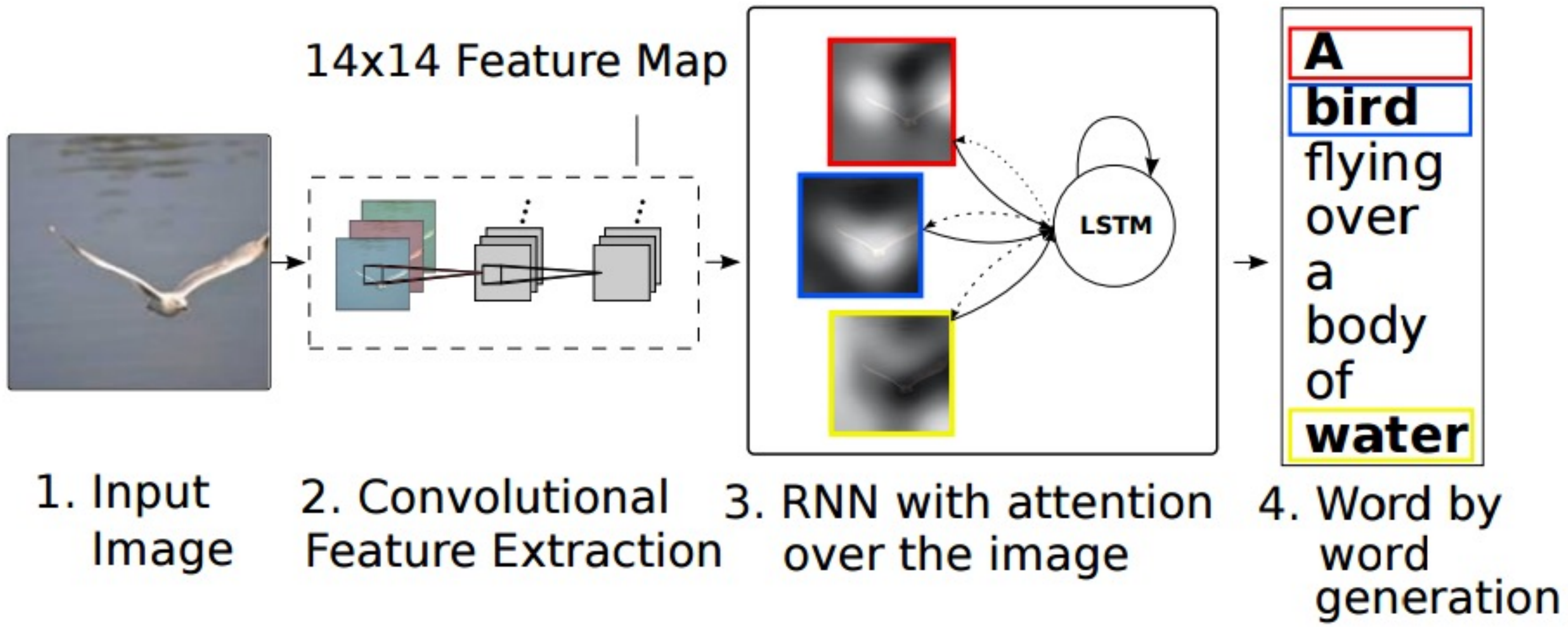
- $\boldsymbol{q}$ is the query and $\boldsymbol{k}$ is the key
- **Multi-layer Perceptron** (Bahdanau et al. 2015)
  - $a(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{w}_2^T \tanh(W_1[\boldsymbol{q}; \boldsymbol{k}])$
  - Flexible, often very good with large data
- **Bilinear** (Luong et al. 2015)
  - $a(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{q}^T W \boldsymbol{k}$
- **Dot Product** (Luong et al. 2015)
  - $a(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{q}^T \boldsymbol{k}$
  - No parameters! But requires sizes to be the same.
- **Scaled Dot Product** (Vaswani et al. 2017)
  - Problem: scale of dot product increases as dimensions get larger
  - Fix: scale by size of the vector
  - $a(\boldsymbol{q}, \boldsymbol{k}) = \frac{\boldsymbol{q}^T \boldsymbol{k}}{\sqrt{|\boldsymbol{k}|}}$

## Results

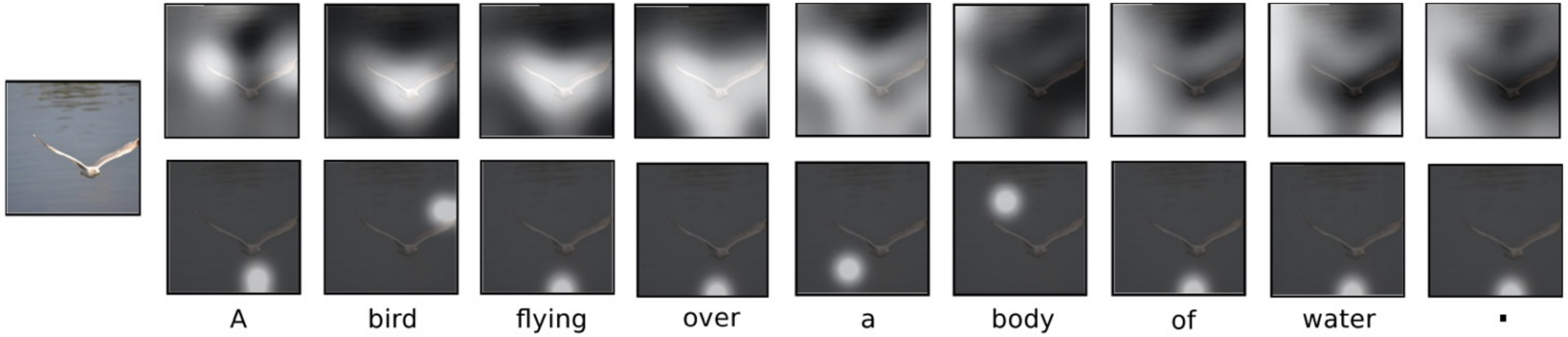

The
agreement
on
the
European
Economic
Area
was
signed
in
August
1992
.
<end>
L'
accord
sur
la
zone
économique
européenne
a
été
signé
en
août
1992
.
<end>


Without attention
With attention
BLEU score
RNNsearch-50
RNNsearch-30
RNNenc-50
RNNenc-30
0
5
10
15
20
25
30
0
10
20
30
40
50
60

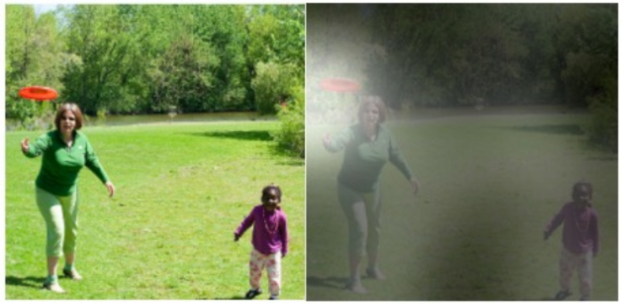
# Image Caption Generation (Xu et al., 2015)

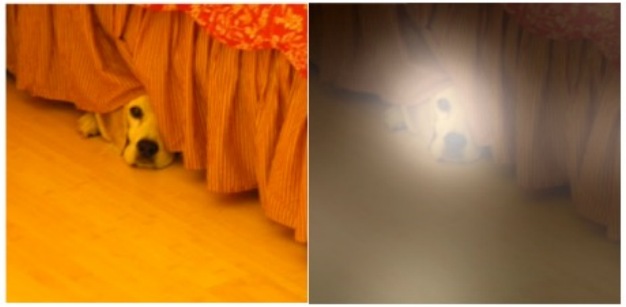

14x14 Feature Map
LSTM
A
bird
flying
over
a
body
of
water
1. Input Image
2. Convolutional Feature Extraction
3. RNN with attention over the image
4. Word by word generation

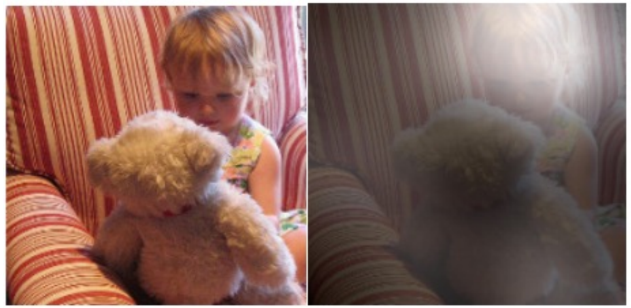
# Image Caption Generation (Xu et al., 2015)

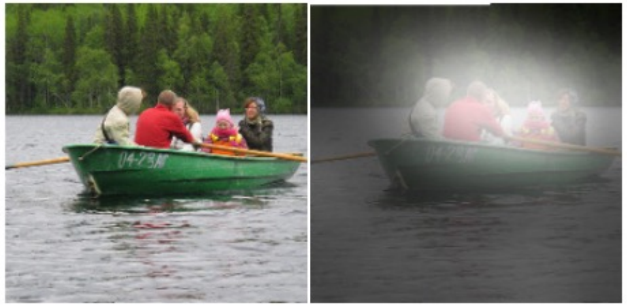


# Image Caption Generation (Xu et al., 2015)

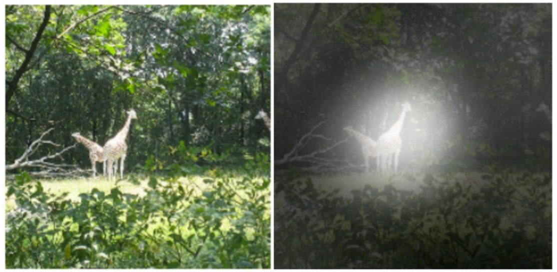

A woman is throwing a frisbee in a park.

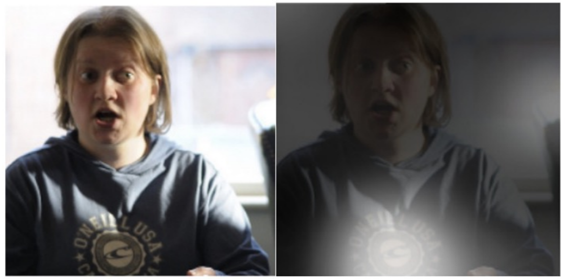

A dog is standing on a hardwood floor.

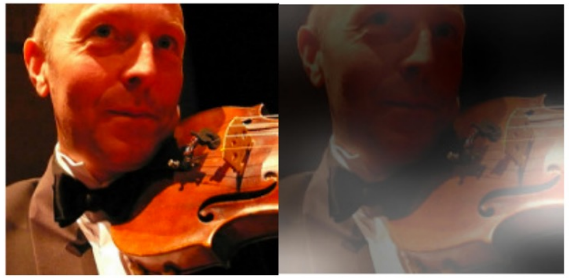

A little girl sitting on a bed with a teddy bear.

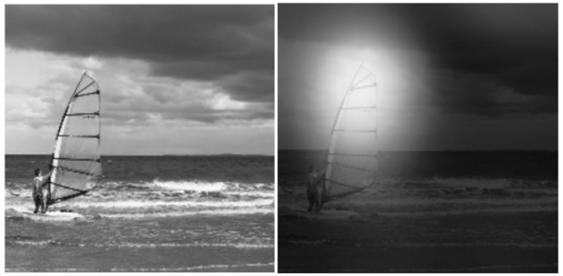

A group of people sitting on a boat in the water.

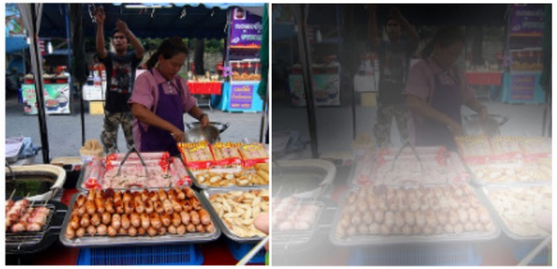
# Image Caption Generation (Xu et al., 2015)

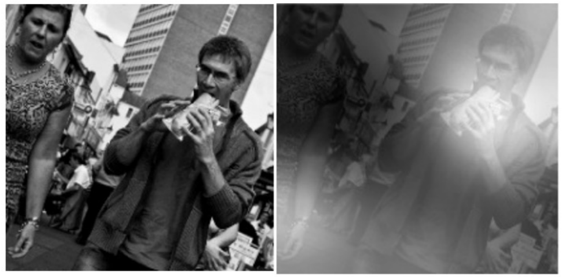

A large white bird standing in a forest.


A woman holding a clock in her hand.

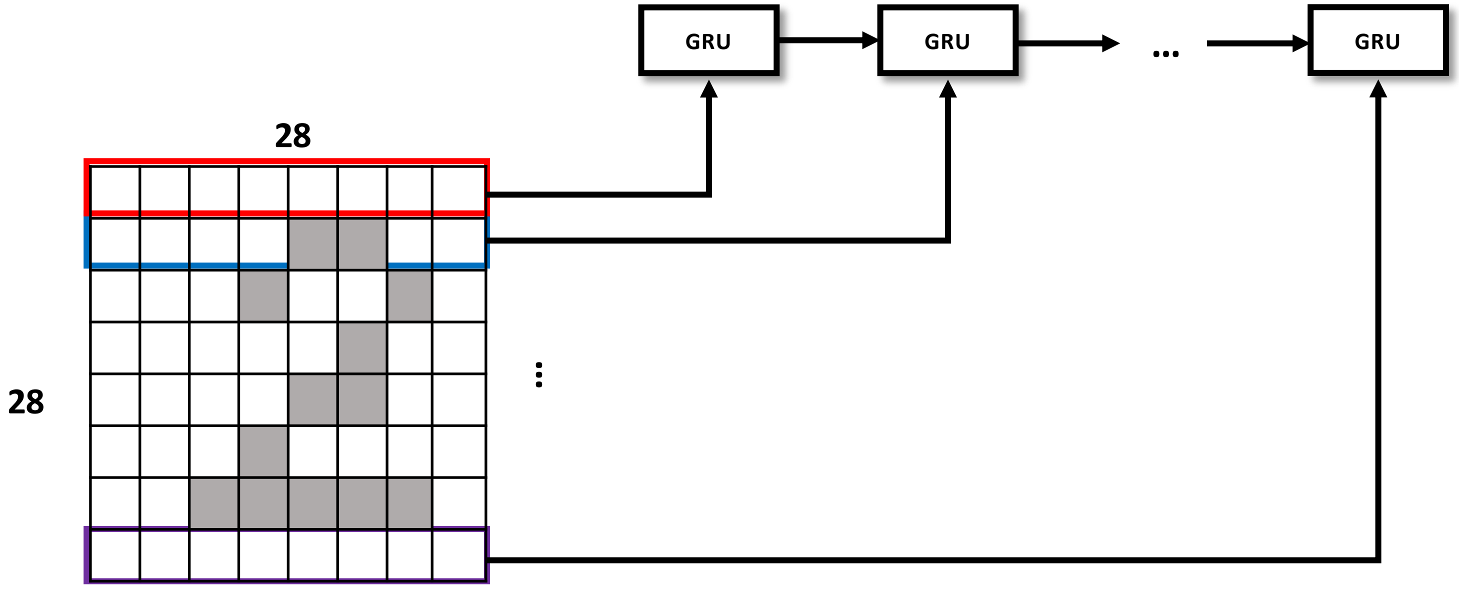

A man wearing a hat and
a hat on a skateboard.


A person is standing on a beach
with a surfboard.

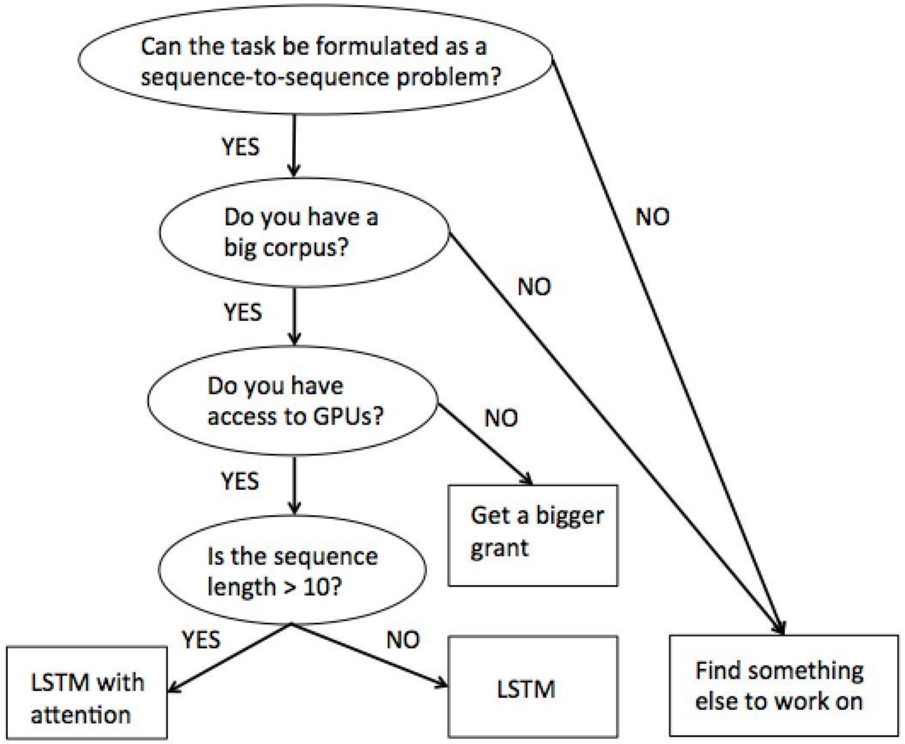

A woman is sitting at a table
with a large pizza.


A man is talking on his cell phone
while another man watches.

# RNN for MNIST



## How to do NLP research now?


Can the task be formulated as a sequence-to-sequence problem?
YES
NO
Do you have a big corpus?
YES
NO
Do you have access to GPUs?
YES
NO
Get a bigger grant
Is the sequence length > 10?
YES
NO
LSTM with attention
LSTM
Find something else to work on

## Transformers

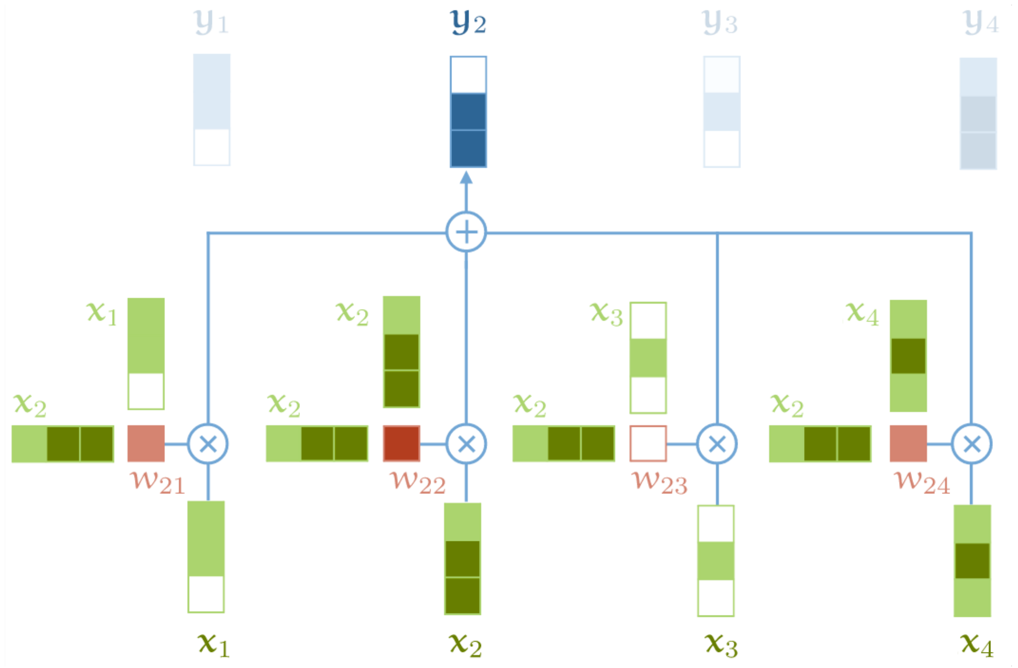
## What’s Transformers

- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. **Attention Is All You Need**. NIPS 2017.
- Definition
    - Any architecture designed to process a connected set of units—such as the tokens in a sequence or the pixels in an image—where the only interaction between units is through **self-attention**.
- Key component
    - Self-attention

## Self-attention

- A sequence-to-sequence operation
  - Input vectors: $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_t$
  - Output vectors: $\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_t$
  - The vectors all have dimension $k$
- $\boldsymbol{y}_i = \sum_j w_{ij} \boldsymbol{x}_j$, where $j$ indexes over the whole sequence
- $w_{ij}$ is derived from a function over $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$
- The simplest option for this function is the dot product:
  - $w_{ij}' = \boldsymbol{x}_i^T \boldsymbol{x}_j$
  - Then apply a softmax to map the values to [0, 1]: $w_{ij} = \frac{\exp w'_{ij}}{\sum_j \exp w'_{ij}}$

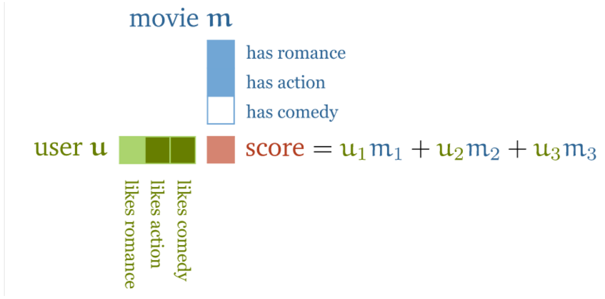
## A visual illustration of basic self-attention



Self-attention is the only operation in Transformers that propagates information between vectors

# Understanding why self-attention works

- For example, movie recommendation
- Using features to represent movies and users
  - Movies: how much romance there is in the movie, and how much action
  - Users: how much they enjoy romantic movies and how much they enjoy action-based movies
- The dot product between the two feature vectors would give you a score for how well a movie match what a user enjoys

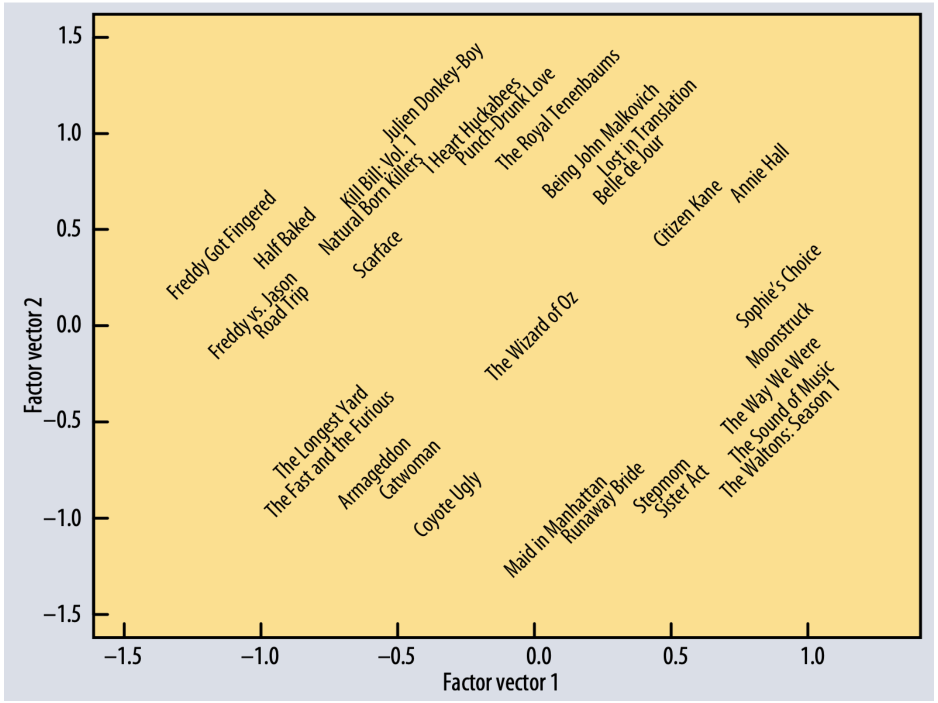


The signs of a feature are meaningful.

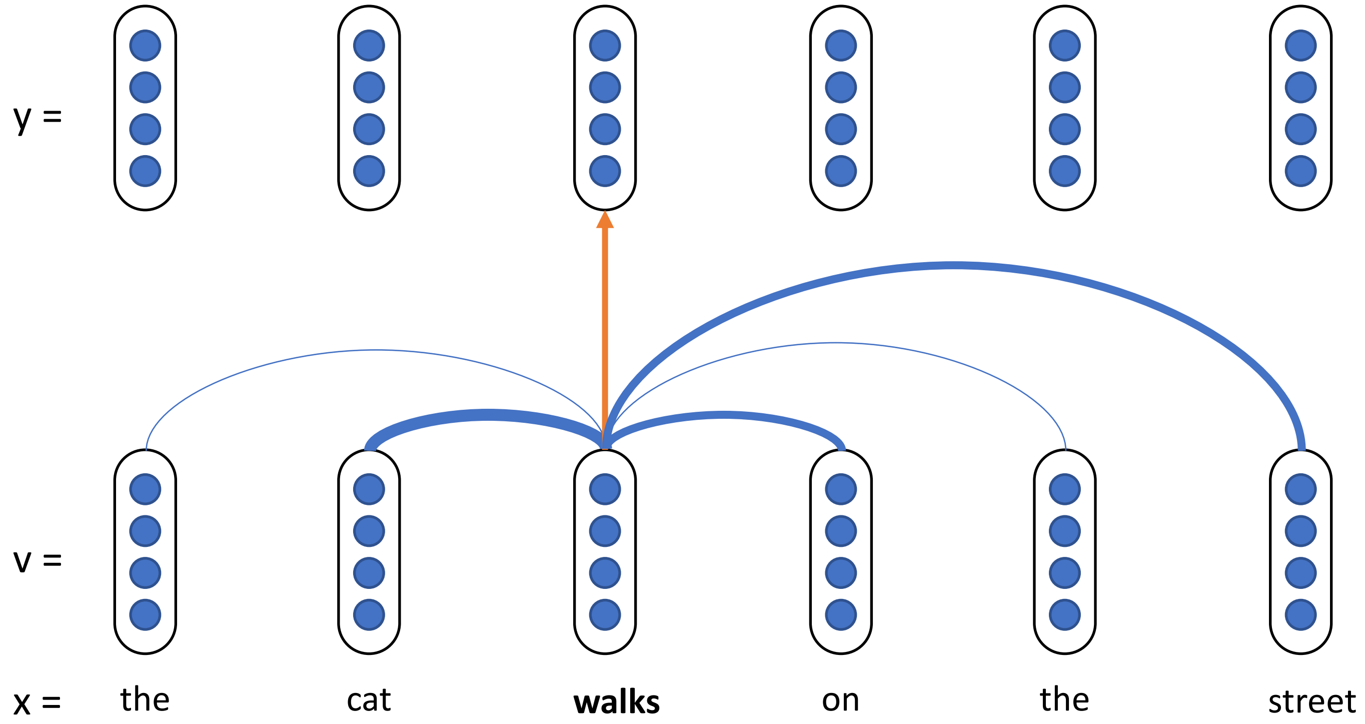
# How to assign features to movies and users?

- Ask users for a small number of movies that they like (training data)

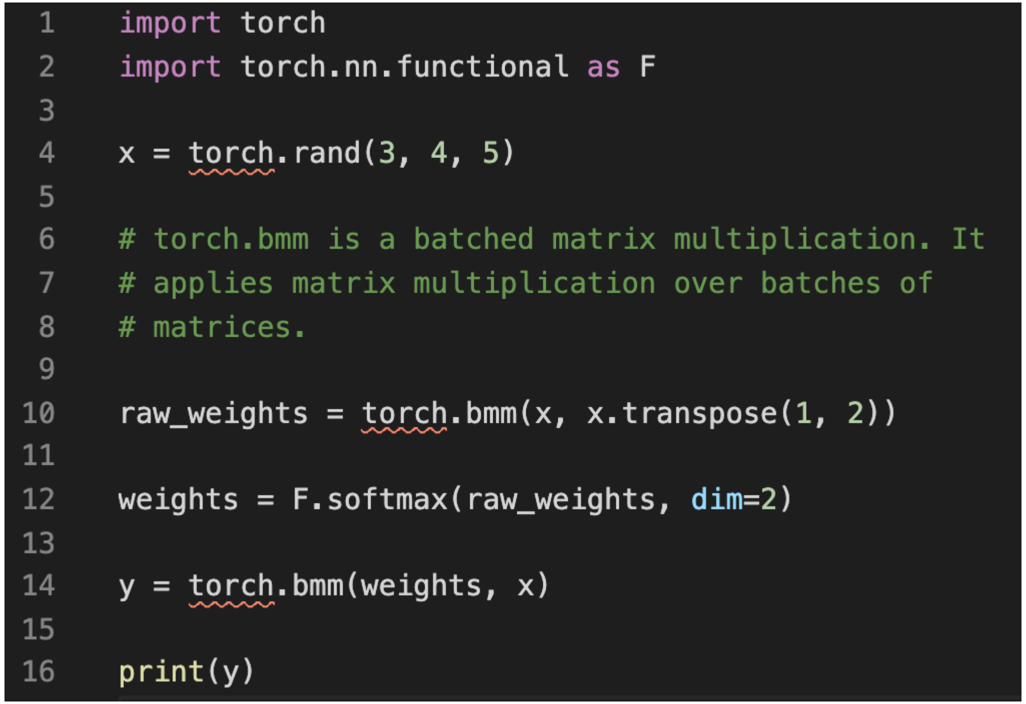
- Optimize the user and movie features so that their dot product matches the data



# Self-attention for a sequence of words



## Self-attention in action



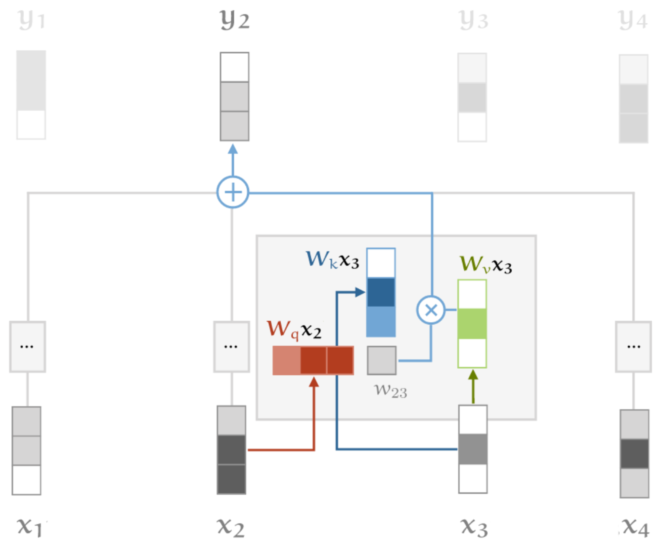
```python
import torch
import torch.nn.functional as F

x = torch.rand(3, 4, 5)

# torch.bmm is a batched matrix multiplication. It
# applies matrix multiplication over batches of
# matrices.

raw_weights = torch.bmm(x, x.transpose(1, 2))

weights = F.softmax(raw_weights, dim=2)

y = torch.bmm(weights, x)

print(y)
```

# Shortcomings of basic self-attention

- A vector plays three roles
  - Vectors $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ to establish the weights
  - Vectors $\boldsymbol{x}_j$ to establish the score
- Dot product grows with the embedding dimension $k$
  - Softmax function can be sensitive to very large input values, which kill the gradient and slow down learning
- A word can mean different things to different neighbors
- The results are position-independent

## Queries, keys and values

- A vector plays three roles
  - Using three *k* x *k* matrices to map $\boldsymbol{x}_i$ into three vectors to play different roles
- These roles are often called the **query**, the **key** and the **value**
  - $\boldsymbol{q}_i = \boldsymbol{W}_q\boldsymbol{x}_i$, $\boldsymbol{k}_i = \boldsymbol{W}_k\boldsymbol{x}_i$, $\boldsymbol{v}_i = \boldsymbol{W}_v\boldsymbol{x}_i$
  - $w_{ij}' = \boldsymbol{q}_i^T \boldsymbol{k}_j$
  - $w_{ij} = \text{softmax}(w_{ij}')$
  - $\boldsymbol{y}_i = \sum_j w_{ij} \boldsymbol{v}_j$

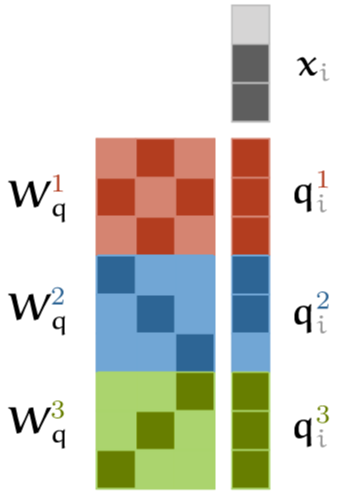


## Scaling the dot product

- $w'_{ij} = \frac{\boldsymbol{q}_i^T \boldsymbol{k}_j}{\sqrt{k}}$, where *k* is the dimension of vector *x*$_i$
- Why $\sqrt{k}$?
  - Imagine a vector in $\mathbb{R}^k$ with values all *c*. Its Euclidean length is $\sqrt{k}c$
  - Dividing out the amount by which the increase in dimension increases the length of the average vectors.

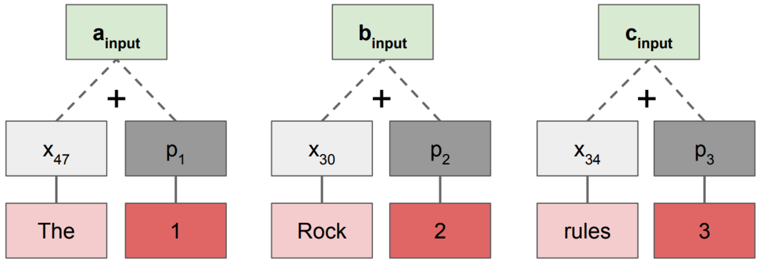
# Multi-head attention

- A word can mean different things to different neighbors
  - For example, “the cat **walks** on the street”
- Combining several self-attention mechanisms, each with different matrices $\boldsymbol{W}_q^r$, $\boldsymbol{W}_k^r$, $\boldsymbol{W}_v^r$ (r is the index of a head)
- For input $\boldsymbol{x}_i$ each attention head produces a different output vector $\mathbf{y}_i^r$. We concatenate these, and pass them through a linear transformation to reduce the dimension back to $k$



Multi-head attention seems ensemble of attention

# Using the positions



- Position embeddings
  - Embedding the positions like we did the words
  - For example
    - $\boldsymbol{v}_{12}$ = [0.4259, 0.1208, 0.5587], $\boldsymbol{v}_{25}$ = [0.7838, 0.4896, 0.5110]
- Position encodings
  - Using a function $f{:}\mathbb{N}\rightarrow\mathbb{R}^k$ to map a positions to a real valued vector

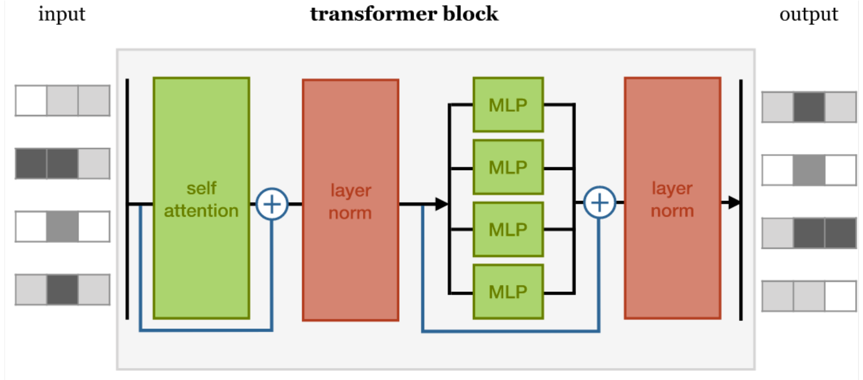
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

$$\begin{array}{c|ccccc|}
 & < & - & emb\_dim & - & > \\
The & \sin\left(\frac{0}{10000^{\frac{0}{emb_{dim}}}}\right) & \cos\left(\frac{0}{10000^{\frac{0}{emb_{dim}}}}\right) & \sin\left(\frac{0}{10000^{\frac{2}{emb_{dim}}}}\right) & \cos\left(\frac{0}{10000^{\frac{2}{emb_{dim}}}}\right) & \cdots \\
daughter & \sin\left(\frac{1}{10000^{\frac{0}{emb_{dim}}}}\right) & \cos\left(\frac{1}{10000^{\frac{0}{emb_{dim}}}}\right) & \sin\left(\frac{1}{10000^{\frac{2}{emb_{dim}}}}\right) & \cos\left(\frac{1}{10000^{\frac{2}{emb_{dim}}}}\right) & \cdots \\
had & \sin\left(\frac{2}{10000^{\frac{0}{emb_{dim}}}}\right) & \cos\left(\frac{2}{10000^{\frac{0}{emb_{dim}}}}\right) & \sin\left(\frac{2}{10000^{\frac{2}{emb_{dim}}}}\right) & \cos\left(\frac{2}{10000^{\frac{2}{emb_{dim}}}}\right) & \cdots \\
a & \sin\left(\frac{3}{10000^{\frac{0}{emb_{dim}}}}\right) & \cos\left(\frac{3}{10000^{\frac{0}{emb_{dim}}}}\right) & \sin\left(\frac{3}{10000^{\frac{2}{emb_{dim}}}}\right) & \cos\left(\frac{3}{10000^{\frac{2}{emb_{dim}}}}\right) & \cdots \\
nice & \sin\left(\frac{4}{10000^{\frac{0}{emb_{dim}}}}\right) & \cos\left(\frac{4}{10000^{\frac{0}{emb_{dim}}}}\right) & \sin\left(\frac{4}{10000^{\frac{2}{emb_{dim}}}}\right) & \cos\left(\frac{4}{10000^{\frac{2}{emb_{dim}}}}\right) & \cdots \\
umbrella & \sin\left(\frac{5}{10000^{\frac{0}{d_{emb_{dim}}}}}\right) & \cos\left(\frac{5}{10000^{\frac{0}{emb_{dim}}}}\right) & \sin\left(\frac{5}{10000^{\frac{2}{emb_{dim}}}}\right) & \cos\left(\frac{5}{10000^{\frac{2}{emb_{dim}}}}\right) & \cdots
\end{array}$$

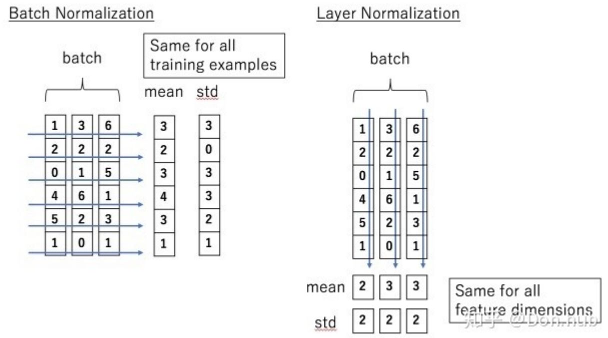
知乎 @Don.hub

# Transformer block

- Self-attention + non-linear
  - Combine self-attention with a local feedforward

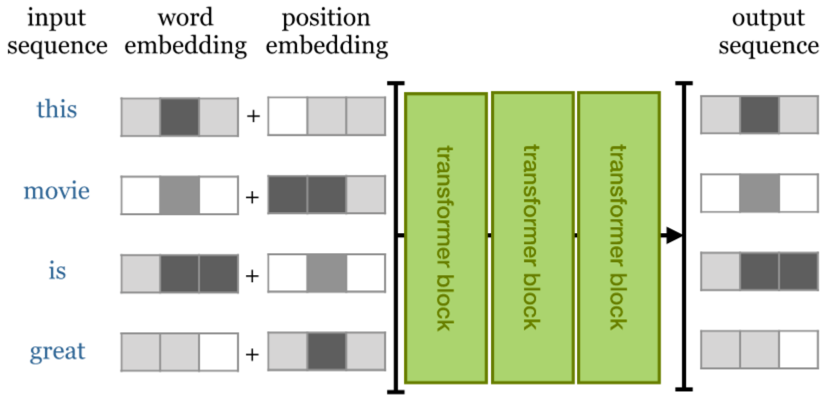


- Normalization and residual connections are standard tricks used to help DNNs train faster and more accurately



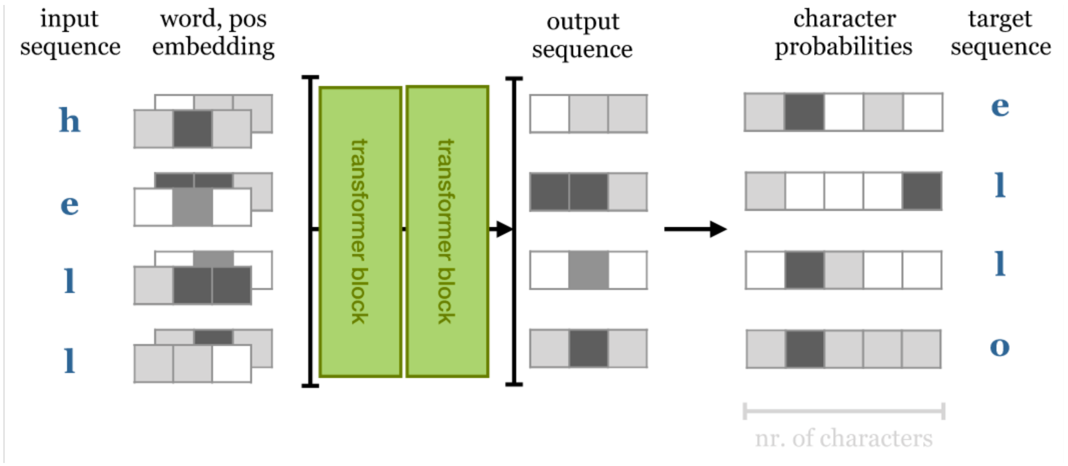
# Transformers for text classification

- Transformer
  - A chain of transformer blocks
- Text classification
  - Apply global average pooling to the final output sequence



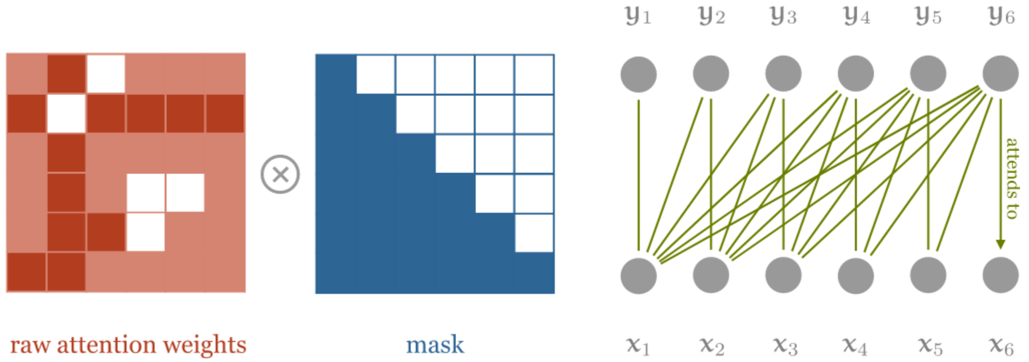
## Transformers for text generation

- Autoregressive model
- For example
  - Predict the next character in a sequence



# Transformers for text generation

- Output token should only depend only on previous inputs
- With a transformer, the output depends on the entire input sequence
- Applying a mask to the matrix of dot products, before the softmax is applied
  - Set the masked out elements to -inf



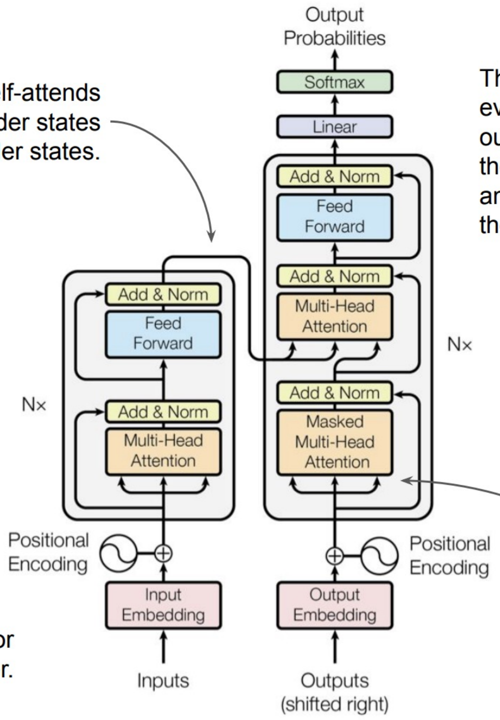
raw attention weights
mask
$y_1$ $y_2$ $y_3$ $y_4$ $y_5$ $y_6$
attends to
$x_1$ $x_2$ $x_3$ $x_4$ $x_5$ $x_6$

## Transformers for encoder-decoder

- The decoder generates the output sentence word for word based both on the input words and the words it has already generated

If you can use seq2seq, you can use transformer.

Each decoder state self-attends with all of its fellow decoder states and with all the encoder states.

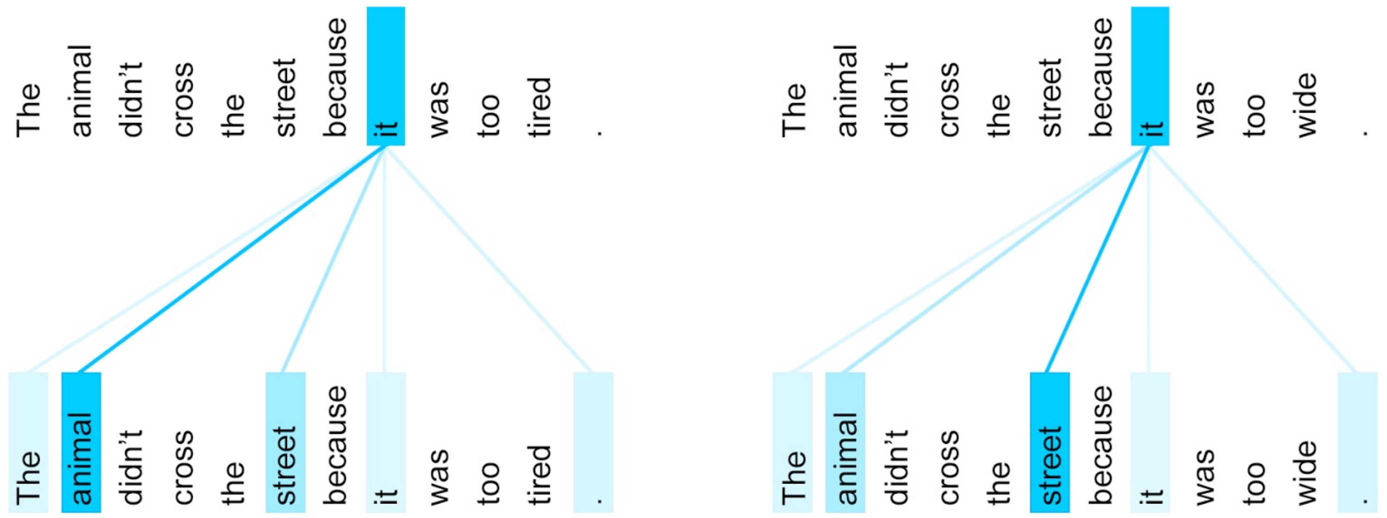


The right side is repeated for every decoder state, with outputs for each state that has them (all of them for dialogue and machine translation, only the final one for NLI).

The left side is repeated for every state in the encoder.

In the decoder, self-attention is limited to preceding words.

# Attention Visualization

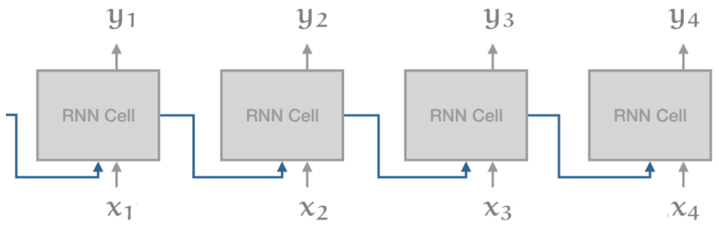


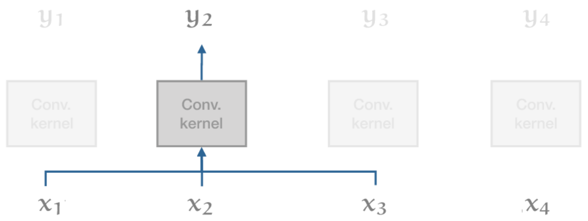
The encoder self-attention distribution for the word "it" from the 5th to the 6th layer of a Transformer trained on English to French translation (one of eight attention heads).
https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

# Advantages of Transformers

- Compared to RNN, Transformers have no recurrent connections
- So the whole model can be computed in a very efficient

- Compared to CNN, Transformers can model dependencies over the whole range of the input sequence





## Disadvantages of Transformers

- The model is too big to be fitted in a GPU
- Cannot handle too long sequences
- How to solve the problem?
  - Half precision
  - Gradient accumulation
  - Gradient checkpointing
    - https://medium.com/huggingface/training-larger-batches-practical-tips-on-1-gpu-multi-gpu-distributed-setups-ec88c3e51255

# Summary: RNNs and Transformers

**RNNs**

- Recurrent Neural Networks
- The vanishing gradient problem
- Long Short-Term Memory (LSTM)
- Gated Recurrent Unites (GRU)
- Attention mechanism
- Can be used in almost all applications

**Transformers**

- Basic self-attention
- Improvements
  - Query, key and value transformation
  - Scaling the doc product
  - Multi-head attention
  - Positional embeddings and encodings
- Transformers are extremely generic
- The current performance limit is purely in the hardware

## 参考

- 《自然语言处理：基于预训练模型的方法》
  - 出版社：电子工业出版社
  - 作者：车万翔，郭江，崔一鸣 著；刘挺 主审
  - 书号：ISBN 978-7-121-41512-8
  - 出版时间：2021.7
- 网购链接
  - https://item.jd.com/13344628.html
- 书中代码
  - https://github.com/HIT-SCIR/plm-nlp-code