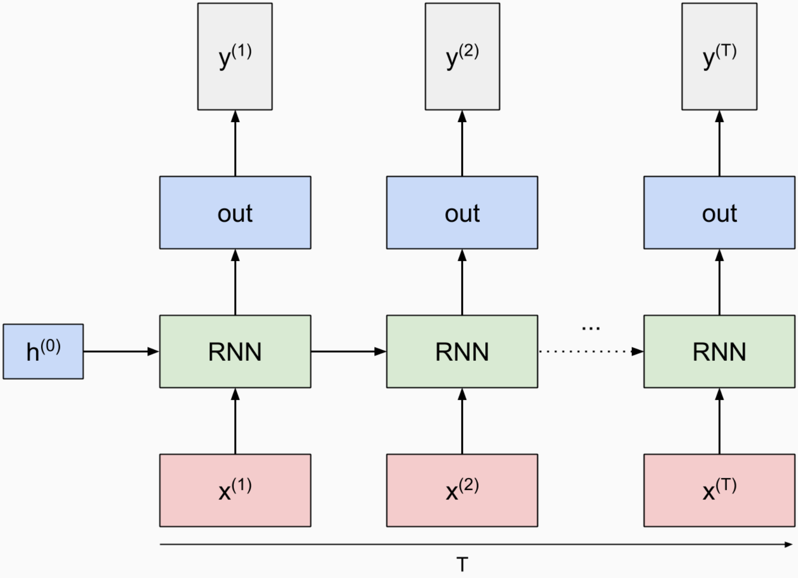
Advanced recurrent models

Martin Tutek

May 2022.

Recap

Plain recurrent model (RNN denotes recurrent cells)



Basic recurrent cell

Update of the hidden state of the recurrent cell:

$$h^{(t)} = tanh(\underbrace{W_{hh}h^{(t-1)} + W_{xh}x^{(t)} + b_h}_{a^{(t)}})$$

Output projection

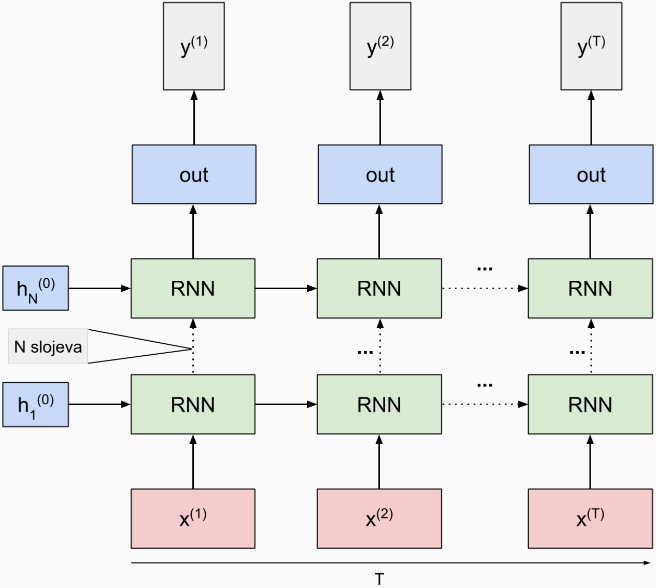
$$o^{(t)} = W_{hy}h^{(t)} + b_o \tag{1}$$

The plain formulation processes the whole sequence with *one* layer

- often insufficient for learning complex dependencies among the sequence elements
- it can be addressed by introducing latent recurrent layers between the input and the predictions!

Deep recurrent models

Deep (multi-layer) recurrent model



Deep recurrent model

Can you observe a problem in the precedent figure?

- $x^{(t)}$ and $h^{(t)}$ may have different dimensions
- dimensionality of $W_{xh} \in \mathbb{R}^{h \times h}$ may differ across layers

Layer $n = 1$:

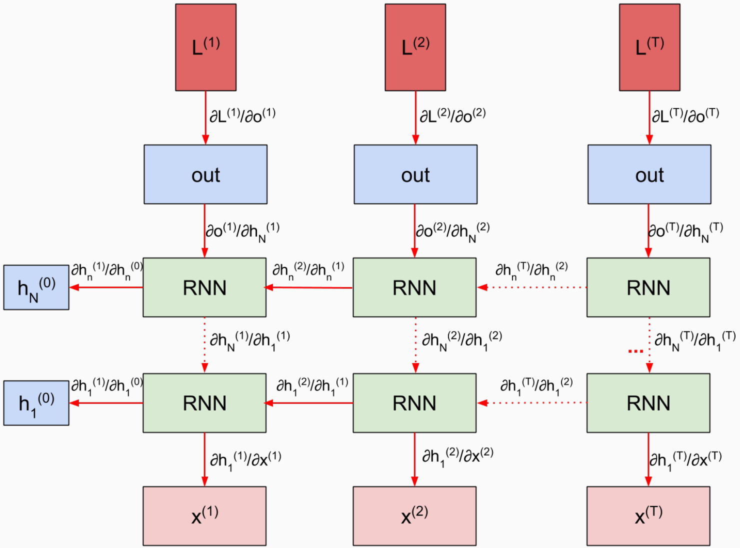
$$h_n^{(t)} = tanh(\underbrace{W_{nhh} h_n^{(t-1)} + W_{nxh} x^{(t)} + b_{nh}}_{a_n^{(t)}}) \tag{2}$$

Layers $n > 1$:

$$h_n^{(t)} = tanh(\underbrace{W_{nhh} h_n^{(t-1)} + W_{nxh} \boxed{h_{n-1}^{(t)}} + b_{nh}}_{a_n^{(t)}}) \tag{3}$$

[!!] Recent deep learning frameworks offer recurrent cells that adapt the matrix shapes automatically.

Deep recurrent model: backprop



Deep recurrent model: summary

Recurrent models extend through depth (vertically) and time (horizontally):

- practical configurations involve 4 to 8 layers depending on the quantity of training data
- more than 8 subsequent recurrent layers do not lead to significant performance improvements (even in the case of advanced cells)

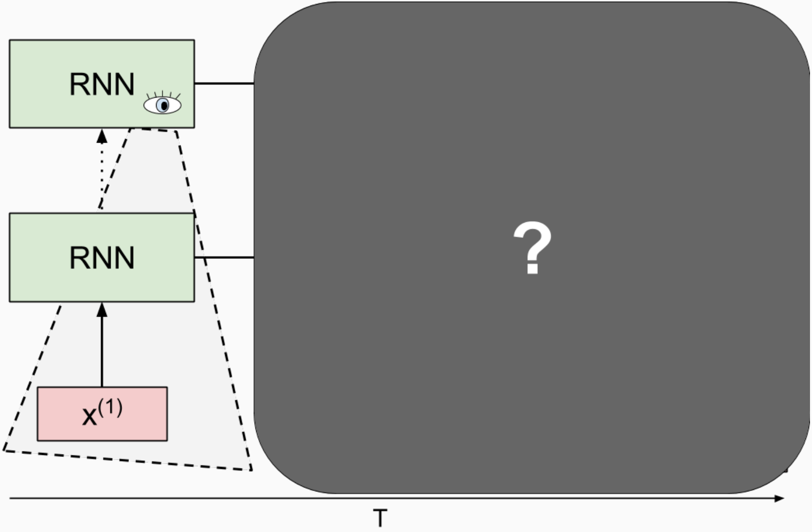
Different layers may have different dimensionalities:

- this does not complicate the implementation unless we start from scratch

Layers can be implemented by supplying constructor arguments to the chosen recurrent cell.

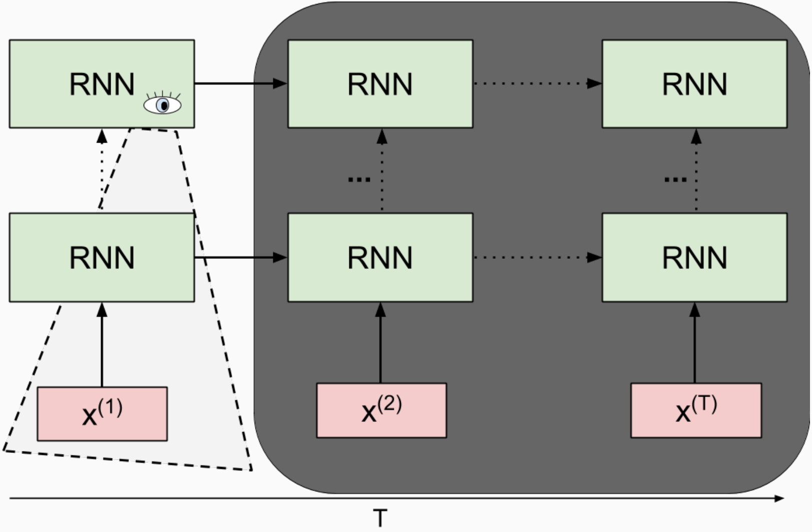
Troubleshooting

What is the **receptive field** of a recurrent cell?



Troubleshooting

What is the **receptive field** of a recurrent cell?



Troubleshooting: receptive field

Each recurrent cell (in any layer, at time t) observes only $x^{(t)} \leq t$:

- prediction at time t is conditioned only by observed inputs!
- if the problem at hand does not imply hiding the *future* context, we would like to allow the model to observe the **whole** sequence prior to making the prediction

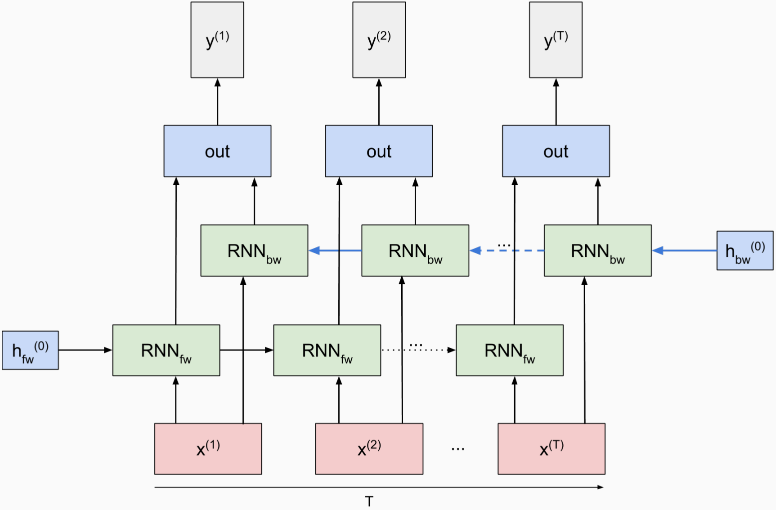
Idea: if the hidden state $h^{(t)}$ of a cell that reads from left to right sees $x^{(t)} \leq t$, then a cell that reads in the opposite direction sees the remaining inputs $x^{(t)} > t$

- together, these two cells observe the whole sequence

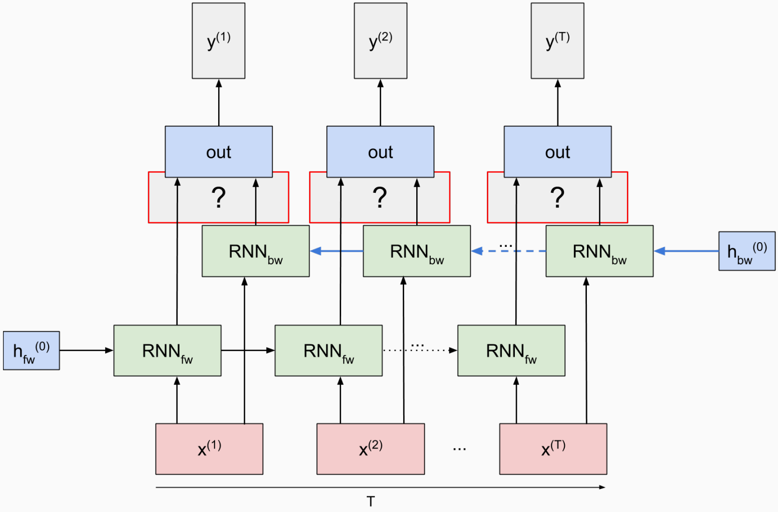
Bidirectional recurrent models

Bidirectional recurrent model

We add an **independent** recurrent model ($\overleftarrow{RNN}$) that operates in the **opposite** direction with respect to the original model ($\overrightarrow{RNN}$)



How to aggregate the hidden states?



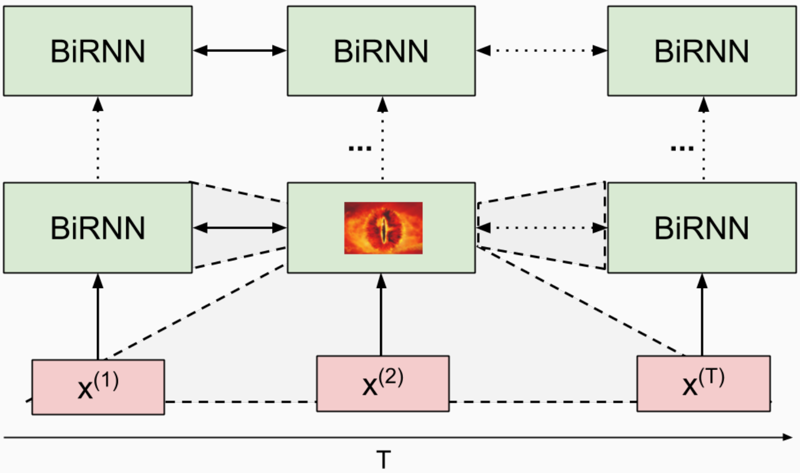
Bidirectional recurrent model

Bidirectional recurrent cell (BiRNN) consists of two unidirectional models that operate in opposite directions:

- $\overrightarrow{RNN}$ reads from left to right
- $\overleftarrow{RNN}$ reads from right to left

How to combine the hidden states?

1. concatenation:
$$h^{(t)} = [\overrightarrow{h}^{(t)}, \overleftarrow{h}^{(t)}]$$
    - this doubles the input dimensionality of the next layer
    - defaul behaviour in existing implementations
2. mean pooling
3. arbitrary (parameterized) function


BiRNN
BiRNN
BiRNN
...
...
BiRNN
BiRNN
$x^{(1)}$
$x^{(2)}$
$x^{(T)}$
T

Bidirectional recurrent models: summary

Bidirectional models consist of two recurrent models that iterirate in opposite directions:

- concatenation of the two states allow the next layer to receive the state that depends on all inputs

Concatenation increases the dimensionality of the subsequent layer:

- default behaviour
- alternatives: mean pooling, pooling + projection, ...

We must consider whether the model **is allowed** to access all input data (forecasting vs dense prediction).

Troubleshooting recurrent models

Vanishing and exploding gradients

Recurrent models are susceptible to vanishing and exploding gradients:

- caused by parameter sharing through subsequent operations
- more precisely: repeated multiplication by W_{hh}

Reminder:

$$h^{(t)} = tanh(W_{hh}h^{(t-1)} + W_{xh}x^{(t)} + b_h)$$

We first consider the *scalar* context:

$$h^{(t)} = tanh(\underbrace{w_{hh}h^{(t-1)} + w_{xh}x^{(t)} + b_h}_{a^{(t)}})$$

- $w_{hh}, w_{xh}, b_h, h, x \in \mathbb{R}$

Vanishing and exploding gradients (scalar context)

$$h^{(t)} = tanh(\underbrace{\boxed{w_{hh}h^{(t-1)}} + w_{xh}x^{(t)} + b_h}_{a^{(t)}})$$

We consider the magnitude of the gradijent between subsequent hidden states:

$$\begin{aligned} \frac{\partial h^{(t)}}{\partial h^{(t-1)}} &= \frac{\partial h^{(t)}}{\partial a^{(t)}} \frac{\partial a^{(t)}}{\partial h^{(t-1)}} \\ &= \frac{\partial th(a^{(t)})}{\partial a^{(t)}} w_{hh} \\ &= (1 - th^2(a^{(t)}))w_{hh} \end{aligned}$$

$tanh = th$

$\frac{dth(x)}{dx} = 1 - th^2(x)$

Tanh derivative is limited to unit interval:

$$tanh(x) \in (-1, 1)$$

$$\frac{\partial tanh(x)}{x} = (1 - tanh^2(x)) \in (0, 1)$$

Vanishing and exploding gradients (scalar context)

$$\frac{\partial h^{(t)}}{\partial h^{(t-1)}} = (1 - th^2(a^{(t)}))w_{hh}$$

Let us apply the following substitution:

$$\gamma_t = \partial tanh(x)/\partial x\big|_{a^{(t)}} < 1$$

Similiarly: $\gamma_{\sigma t} = \partial \sigma(x)/\partial x\big|_{a^{(t)}} < 1/4$

$$\begin{aligned} \frac{\partial h^{(t)}}{\partial h^{(t-1)}} &= \gamma_t w_{hh} \\ &\Big\downarrow\ t \to T \\ \frac{\partial h^{(T)}}{\partial h^{(t_0)}} &= \prod_{t_0}^{T} \gamma_t w_{hh} \\ &\Big\downarrow\ \overline{\gamma}, w_{hh} \text{ indep. } t \\ \frac{\partial h^{(T)}}{\partial h^{(t_0)}} &= (\overline{\gamma} w_{hh})^{T-t_0} \end{aligned}$$

Vanishing and exploding gradients (scalar context)

$$\frac{\partial h^{(T)}}{\partial h^{(t_0)}} = (\overline{\gamma} w_{hh})^{T-t_0}$$

When we have long sequences, $T - t_0 \gg 0$, the gradients may explode, vanish or be stable depending on $\overline{\gamma} w_{hh}$:

$$(\overline{\gamma} w_{hh})^{T-t_0} \to \begin{cases} \infty & \text{if } \overline{\gamma} w_{hh} > 1 \quad \text{(explodes)} \\ 0 & \text{if } \overline{\gamma} w_{hh} < 1 \quad \text{(vanishes)} \\ 1 & \text{if } \overline{\gamma} w_{hh} \approx 1 \quad \text{(stable)} \end{cases}$$

If we assume $\overline{\gamma} = 1$, then the above conditions apply to the **parameter** w_{hh}.

We proceed by repeating the analysis in the vector context.

Vanishing and exploding gradients (spectral norm)

We consider the following properties of the spectral norm of a square matrix A:

- norm of the product is less than or equal the product of the norms (this holds for all matrix norms):

$$\|AB\| \leq \|A\| \, \|B\|$$

- the spectral norm is equal to the largest singular value of A
  - or, equivalently, the largest eigenvalue of $A^\top A$
- the spectral norm is the natural norm induced by the L2-norm:

$$\|Ax\| \leq \|A\| \, \|x\|$$

Vanishing and exploding gradients (vector context)

Update of the hidden state of the recurrent cell (reminder):

$$h^{(t)} = tanh(\underbrace{W_{hh}h^{(t-1)} + W_{xh}x^{(t)} + b_h}_{a^{(t)}})$$

We are looking at the gradient between two subsequent states:

$$\frac{\partial h^{(t)}}{\partial h^{(t-1)}} = \frac{\partial h^{(t)}}{\partial a^{(t)}} W_{hh}^{\top}$$

We note that the gradient magnitude is bounded:

$$\left\| \frac{\partial h^{(t)}}{\partial h^{(t-1)}} \right\| \leq \left\| \frac{\partial h^{(t)}}{\partial a^{(t)}} \right\| \|W_{hh}^{T}\| \leq \gamma_{\mathrm{max}} \lambda_1$$

- λ_1 ... the greatest singular value of W_{hh}
- γ_{max} ... the upper bound of $\mathrm{max}(\frac{\partial tanh(a^{(t)})}{\partial a^{(t)}})$

Vanishing and exploding gradients (vector context)

We extend the last equation over several time-steps:

$$\frac{\partial h^{(T)}}{\partial h^{(t_0)}} \leq (\gamma_{\mathsf{max}}\, \lambda_1)^{T-t_0}$$

For long sequences ($T - t_0 \gg 0$) the gradient may explode, vanish or be stable depending on $\gamma_{\mathsf{max}}\, \lambda_1$:

$$(\gamma_{\mathsf{max}}\, \lambda_1)^{T-t_0} \rightarrow \begin{cases} \infty & \text{if } \gamma_{\mathsf{max}}\, \lambda_1 > 1 \ \text{(explode)} \\ 0 & \text{if } \gamma_{\mathsf{max}}\, \lambda_1 < 1 \ \text{(vanish)} \\ 1 & \text{if } \gamma_{\mathsf{max}}\, \lambda_1 \approx 1 \ \text{(stable)} \end{cases}$$

- Matrix W_{hh} must fulfill **strict requirements** in order to ensure smooth optimization
- For a more detailed analysis refer to [pascanu13icml]:
  Razvan Pascanu, Tomás Mikolov, Yoshua Bengio: On the difficulty of training recurrent neural networks. ICML 2013.

Long-term dependencies

Recurrent models consistently underperform on long sequences:

- The problem is widely known [bengio94tnn]: Y Bengio, PY Simard, P Frasconi: Learning long-term dependencies with gradient descent is difficult, IEEE TNN 1994.
- Prominent cause: unstability of the gradients during backprop.

Problem: exponentiation due to repeated multipliation with W_{hh} causes exploding and vanishing gradient

- ensure moderate singular values of the recurrent connection
  M Arjovsky, A Shah, Y Bengio: Unitary Evolution Recurrent Neural Networks. ICML 2016
- remove multiplication from the recurrent connection.

Solution: *decouple* functionalities of the the recurrent connection

- W_{hh} and W_{xh} couple information filtering, memorizing observed inputs, projection of new elements into the hidden state, …
- Hidden state h couples output projection and memorization.

Recurrent cell with long-term memory (LSTM)

Long short-term memory (LSTM)

Notation:

- Vector $h^{(t)}$ will be repurposed and used **only** for calculating the output
- We introduce a cell state vector $c^{(t)}$ which **only** holds information seen up to current point
- We introduce a new vector $\hat{c}^{(t)}$ which is used for updating the cell state
- We introduce $f^{(t)}$ i $i^{(t)}$:
    - we refer to $f^{(t)}$ as *forget gate*
    - we refer to $i^{(t)}$ as *input gate*

We wish to eliminate multiplication from the recurrent path:

$$c^{(t)} = \quad c^{(t-1)} + \quad \hat{c}^{(t)}$$

$$\frac{\partial c^{(t)}}{\partial c^{(t-1)}} = \mathbb{I}$$

Long short term memory (LSTM)

$$c^{(t)} = \quad c^{(t-1)} + \quad \hat{c}^{(t)}$$

We **forget** some information from the previous state

$$f^{(t)} = \sigma(W_{fhh} h^{(t-1)} + W_{fxh} x^{(t)} + b_{fh}) = \sigma(a_f^{(t)})$$

- each gate has its own set of parameters W_{hh}, W_{xh}, b_h.

We let through only a **subset** of the input information:

$$i^{(t)} = \sigma(W_{ihh} h^{(t-1)} + W_{ixh} x^{(t)} + b_{ih}) = \sigma(a_i^{(t)})$$

- we leverage these results in the recurrent path:

$$c^{(t)} = f^{(t)} \odot c^{(t-1)} + i^{(t)} \odot \hat{c}^{(t)}$$

Intuition behind the gates

$$c^{(t)} = f^{(t)} \odot c^{(t-1)} + i^{(t)} \odot \hat{c}^{(t)}$$

Hadamard product ($\odot$): element-wise matrix multiplication

$$a \odot b = \begin{pmatrix} a_0 b_0 \\ \dots \\ a_i b_i \end{pmatrix}$$

Purpose of the gates: filtering information ($\sigma : \mathbb{R} \rightarrow (0, 1)$).

Sigmoid function has a probabilistic interpretation - **the amount information that we wish to keep**.

Limiting $f^{(t)}$ and $i^{(t)}$ between $(0, 1)$ *eliminates* exploding gradients

- this interval is open in theory due to asymptotic behaviour of the sigmoid but closed in practice due to finite precision

Long short term memory (LSTM)

$$c^{(t)} = f^{(t)} \odot c^{(t-1)} + i^{(t)} \odot \hat{c}^{(t)}$$

$\hat{c}$: temporary result when updating cell state

$$\hat{c}^{(t)} = tanh(W_{chh}h^{(t-1)} + W_{cxh}x^{(t)} + b_{ch}) = tanh(a_c^{(t)})$$

- we determine $\hat{c}^{(t)}$ with respect to the **hidden state** and the input
- **note that notation is a bit different in the book**
  - in the book: $s^{(t)} := c^{(t)}; g^{(t)} := i^{(t)}; q^{(t)} := o^{(t)}$
  - the book does not use **aesthetic** substitution with $\hat{c}^{(t)}$
    $$s^{(t)} = f^{(t)} s^{(t-1)} + g^{(t)} \left(tanh(Wh^{(t-1)} + Ux^{(t)} + b_s) \right)$$

Long short term memory (LSTM)

$$c^{(t)} = f^{(t)} \odot c^{(t-1)} + i^{(t)} \odot \hat{c}^{(t)}$$

The hidden state is calculated with respect to the cell state:

$$h^{(t)} = o^{(t)} \odot tanh(c^{(t)})$$

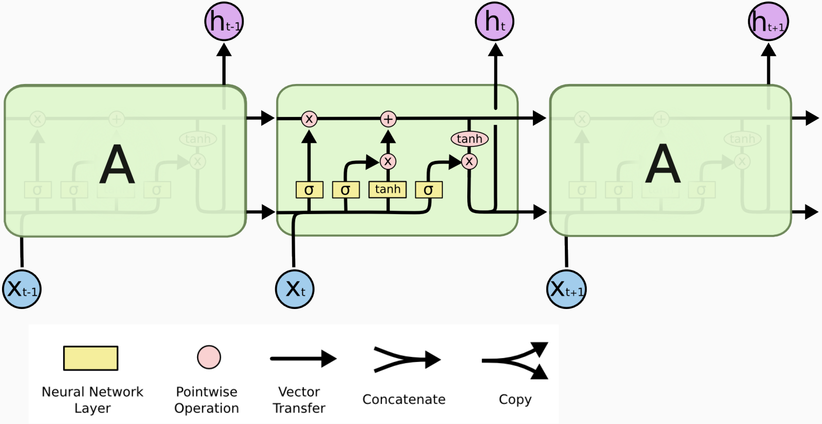
We refer to $o^{(t)}$ as the *output gate*:

$$o^{(t)} = \sigma(W_{ohh} h^{(t-1)} + W_{oxh} x^{(t)} + b_{oh}) = \sigma(a_o^{(t)})$$

Summary:

- we have separated the "memory" from the hidden state, thereby unburdening $h^{(t)}$, which has to do both task in the regular recurrent cell
- we introduce the "cell state" $c^{(t)}$ with a purpose to remember information (we ensure it is **hard** to change its value)
- there are **four** times more parameters than in a regular RNN cell

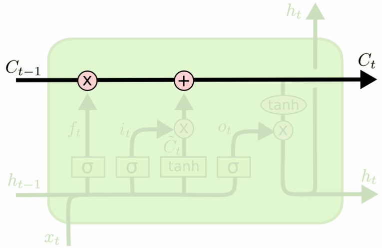
We reproduce several figures from the blog by Cristopher Olah [link]



Question: What is the order of gate names in the figure?

Visualizing the LSTM - cell state

$$\boxed{c^{(t)}} = f^{(t)} \odot \boxed{c^{(t-1)}} + i^{(t)} \odot \hat{c}^{(t)}$$

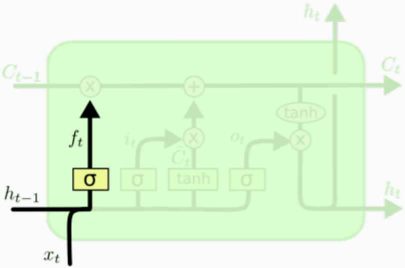


Cell state is modified with one multiplication and one summation - information flow is simple

- Cell state is **hard** to modify: the two gates have to allow the change to ``*pass through*"

Visualizing the LSTM - forget gate

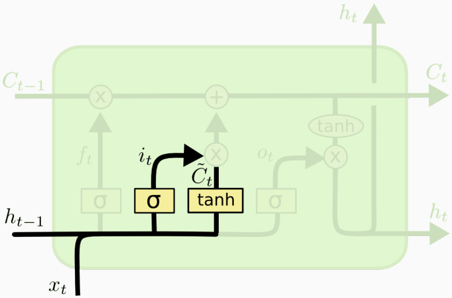
$$c^{(t)} = f^{(t)} \odot c^{(t-1)} + i^{(t)} \odot \hat{c}^{(t)}$$



$$f^{(t)} = \sigma(W_{fhh} h^{(t-1)} + W_{fxh} x^{(t)} + b_{fh})$$

Visualizing the LSTM - input gate

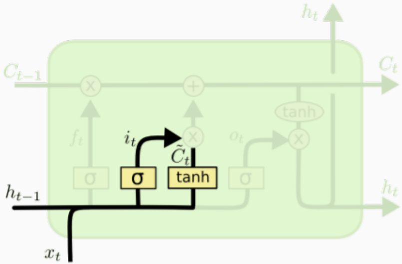
$$c^{(t)} = f^{(t)} \odot c^{(t-1)} + \boxed{i^{(t)} \odot \hat{c}^{(t)}}$$



$$i^{(t)} = \sigma(W_{ihh} h^{(t-1)} + W_{ixh} x^{(t)} + b_{ih})$$

$$\hat{c}^{(t)} = tanh(W_{chh} h^{(t-1)} + W_{cxh} x^{(t)} + b_{ch})$$

Visualizing the LSTM - input gate



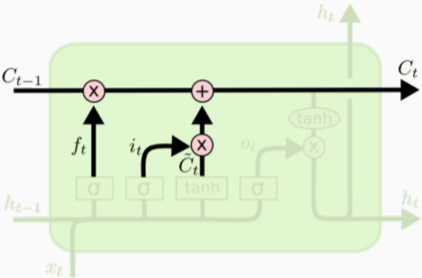
$$\hat{c}^{(t)} = tanh(W_{chh}h^{(t-1)} + W_{cxh}x^{(t)} + b_{ch})$$

According to the literature, either a sigmoid or a hyperbolic tangent can be used to activate $\hat{c}^{(t)}$

- PyTorch and Tensorflow use *tanh*

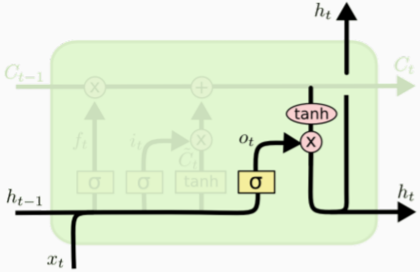
Visualizing the LSTM - updating the cell state

$$c^{(t)} = f^{(t)} \odot \boxed{c^{(t-1)}} + i^{(t)} \odot \hat{c}^{(t)}$$



$$\boxed{h^{(t)}} = \boxed{o^{(t)}} \odot tanh(c^{(t)})$$

$$o^{(t)} = \sigma(W_{ohh}h^{(t-1)} + W_{oxh}x^{(t)} + b_{oh})$$



LSTM as a better RNN: summary

Basic RNNs are difficult to optimize

- we often encounter exploding and wanishing gradients due to repeated multiplication with W_{hh}
- cell state has to memorize information and drive the output
- these models lose performance as sample length increases

We introduce the Long short term memory cell (LSTM)

$$c^{(t)} = f^{(t)} \odot c^{(t-1)} + i^{(t)} \odot \hat{c}^{(t)}$$

$$h^{(t)} = o^{(t)} \odot tanh(c^{(t)})$$

- we remove matrix multiplication from the forward and recurrent path
- we introduce gates to filter information
- decoupled **responsibility** reduces the load on the cell state

LSTM: exploding and vanishing gradients

$$c^{(t)} = f^{(t)} \odot c^{(t-1)} + i \odot \hat{c}^t$$

We consider the gradient for the recurrent link

$$\frac{\partial c^{(t)}}{\partial c^{(t-1)}} = f^{(t)} = \sigma(a_f^{(t)}) \in (0, 1)$$

The chain rule gives us:

$$\frac{\partial c^{(T)}}{\partial c^{(t_0)}} = \prod_{t=t_0}^{T} f^{(t)} \leq 1$$

It appears that the **exploding** gradients are eliminated!

- *Is that really so?*
- LSTM cell has *dual* hidden state $(c^{(t)}, h^{(t)})$

LSTM: exploding and wanishing gradients

$$h^{(t)} = o^{(t)} \odot tanh(c^{(t)})$$

Let us look at the output gate:

$$o^{(t)} = \sigma(a_o^{(t)}) = \sigma(W_{ohh} h^{(t-1)} + W_{oxh} x^{(t)} + b_{oh})$$

There is a similar pattern as in the basic RNN cell:

$$\boxed{h^{(t)}} = \sigma(W_{ohh} \boxed{h^{(t-1)}} + W_{oxh} x^{(t)} + b_{oh}) \odot tanh(c^{(t)})$$

$$\frac{\partial h^{(t)}}{\partial h^{(t-1)}} = \frac{\partial h^{(t)}}{\partial a_o^{(t)}} \frac{\partial a_o^{(t)}}{\partial h^{(t-1)}} = W_{ohh}^T \frac{\partial h^{(t)}}{\partial a_o^{(t)}} = \dots$$

Exploding gradient is therefore **still possible** when performing the backward pass through $h^{(t)}$

- it is, however, relatively rare in practice

LSTM variants - peepholes

Include the cell state $c^{(t-1)}$ while calculating the gate value:

$$f^{(t)} = \sigma(\underbrace{W_{fch}c^{(t-1)} + W_{fhh}h^{(t-1)} + W_{fxh}x^{(t)} + b_{fh}}_{a_f^{*(t)}})$$

This idea can be applied to all gates

- **Strength:** information about the current cell state may **help** [gers2000recurrent]
- **Weakness:** it increases the number of parameters
- **Weakness:** it introduces a second path for the exploding gradients to appear

LSTM variants - fused gates

$$c^{(t)} = \boxed{f^{(t)}} \odot c^{(t-1)} + \boxed{i^{(t)}} \odot \hat{c}^{(t)}$$

Idea: If some information is forgotten, it should be replaced

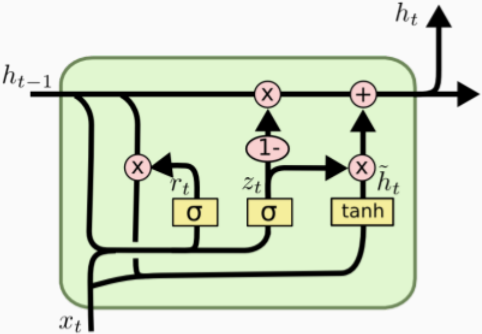
$$c^{(t)} = f^{(t)} \odot c^{(t-1)} + (1 - f^{(t)}) \odot \hat{c}^{(t)}$$

- **Strength:** 25% less parameters
- **Weakness:** it performs worse than the standard LSTM (more parameters helps)
- **Weakness:** it looses expressiveness (problem of addition)

Gated recurrent unit

Gated recurrent unit, a simpler LSTM variant

- works great even though it does not detach the context from the state
- this suggests that some intutitions about LSTM may be incomplete.



Gated recurrent network

GRU models involve two gates: $r^{(t)}$ and $u^{(t)}$:

- $u^{(t)}$ is the *(update gate)*

$$u^{(t)} = \sigma\left(W_{uhh}h^{(t-1)} + W_{uxh}x^{(t)} + b_{uh}\right) \tag{4}$$

- $r^{(t)}$ is the *reset gate*

$$r^{(t)} = \sigma\left(W_{rhh}h^{(t-1)} + W_{rxh}x^{(t)} + b_{rh}\right) \tag{5}$$

Important intermediate tensor: temporary state $\hat{h}^{(t)}$

$$\hat{h}^{(t)} = \sigma\left(W_{hh}\left(r^{(t)} \odot h^{(t-1)}\right) + W_{xh}x^{(t)} + b_h\right) \tag{6}$$

The recurrent state $h^{(t)}$ again has multiple responsibilities:

$$h^{(t)} = u^{(t)}h^{(t-1)} + (1 - u^{(t)})\hat{h}^{(t)} \tag{7}$$

Recurrent variations: summary

Exploding and vanishing gradients may appear in LSTM-s as well

- however, they appear less frequent in practice

Success of LSTM led to development of other gated recurrent cells.

1. **LSTM with peepholes**:
    - they return the state information into the update equation
    - it makes sense since LSTMs do not completely circumvent exploding and vanishing gradients
2. **LSTM with gate fusion**
    - fused forget and input gates for improved efficiency and less parameters
3. **Gated recurrent unit (GRU)**
    - fused gates and changed gate semantics
    - similar performance to LSTMs in spite of coupled state vector

Analysis: Sequence-to-sequence

Machine translation problem

We wish to generate an output sequence of words for a qiven input sequence of words:

- the model targets output sequences of **unknown length**.
- at each output position the model outputs a categoric distribution over the output vocabulary

Datasets: WMT, IWSLT (regularly updated):

- `https://www.statmt.org/wmt15/translation-task.html`
- `https://sites.google.com/site/iwsltevaluation2015/mt-track`

Example of input-output pair in the WMT-14 en-de dataset:

```
Parliament Does Not Support Amendment Freeing Tymoshenko
Keine befreiende Novelle für Tymoshenko durch das Parlament
```

Machine translation problem: preliminaries

Target variables:

- words in the target language: $\{keine, befreiende, Novelle, \ldots\}$
- we convert target variables into indices: $\{0, \ldots, V_{out}\}$
- we choose the size of the target sentence.

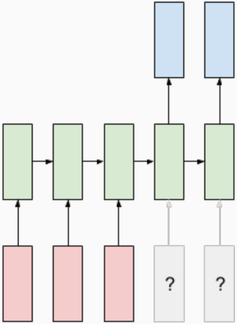
Input variables:

- we must fix the input vocabulary
- we map input words to rows of the embedding matrix,

This setup is similar to part-of-speech tagging, but:

1. we can start generation only after seeing the whole input
2. we do not know the number of output tokens
3. it is not clear whether what inputs should we use during output generation

Sequence-to-sequence



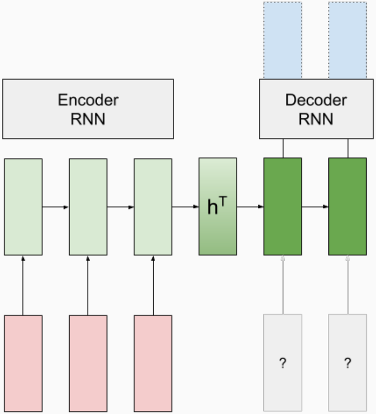
We start by formalizing the sequence-to-sequence problem (above).

Subsequently, we will present some concrete solutions.

Sequence-to-sequence: formalizing the problem

We envision a solution with two modules:

1. **encoder** ("reader"): reads the input sequence and builds the best possible hidden representation
2. **decoder** ("writer") uses the encoded representation to generate the most appropriate translation



Sequense-to-sequence: formalizing the problem

The last state of the encoder determines the first state of the decoder:

$$h_{dec}^{(0)} = f(h_{enc}^{(T)})$$

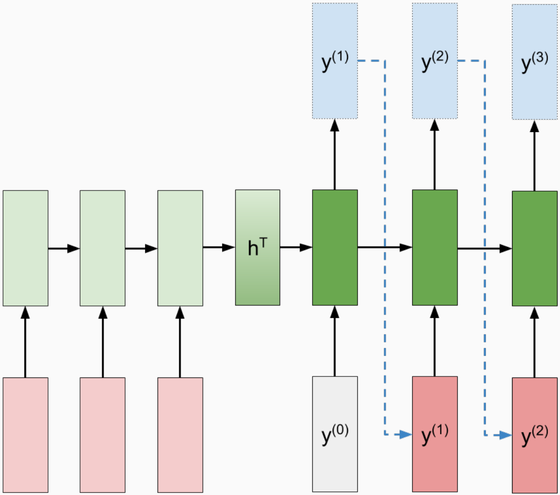
- in practice, we usually have $h_{dec}^{(0)} = h_{enc}^{(T)}$
- still, f can be any parameterized function)
- question: when would something like this be **necessary**?

The encoder **does not receive** the loss directly

- the loss propagates through the entire **decoder**

Problem: what are the inputs to the decoder module?

Sequence-to-sequence: decoder inputs



We use the most recent generated output as input in each step of the decoder

Sequence-to-sequence: generating the output

The decoder input at timestep $t > 0$ contains the most likely decoder output from the previous timestep

- what about timestep $t = 0$?
- the first input to the decoder is *a special symbol* that represents the start of sequence (`<sos>`)

How do we know that the output sequence is completed?

- the model signals the end of translation by outputting a *special symbol* that represents the end of sequence (`<eos>`)
- the token `<eos>` is appended at **the end** of each target sequence
- we stop the output generation when we get `<eos>` or when the generated sequence exeeds the maximum length

Sequence-to-sequence: teacher forcing

Sequence-to-sequence task is very hard to learn, especially in the early stages of optimization

Hence, we often train with **teacher forcing** where the decoder inputs are obtained stochastically as:

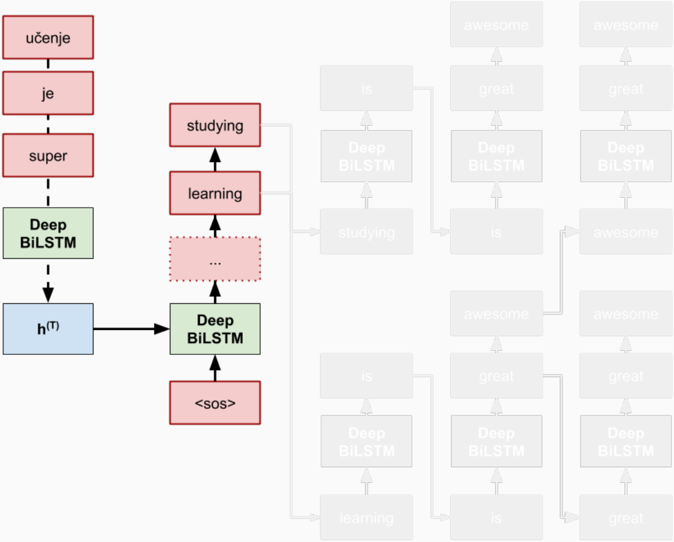
- **groundtruth tokens** of the previous step in p training samples
- **generated outputs** of the prevous step in $1 - p$ training samples
- $p \in [0, 1]$ is a hyper parameter that starts with $p = 1$ and may decrease as the training proceeds

Sequence-to-sequence: inference

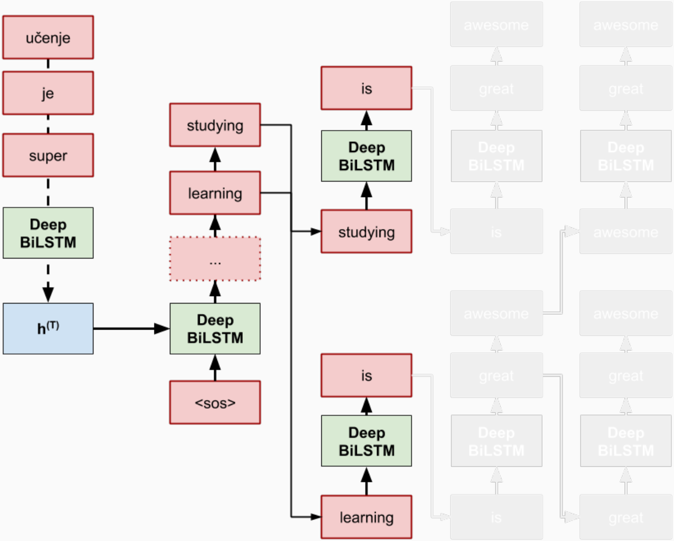
Target sequence generation approaches.

1. greedy approach: take the most probable word in each step:
    - as in other greedy approaches this may be suboptimal
    - it is possible that the best translation does not contain the most probable word in each step
    - moreover, this approach is not good for sampling as it generates **deterministic sequences**
2. random sampling according to probability (roulette wheel selection):
    - this injects rendomness into sequence generation and **encourages variability** of the output
    - it is not clear wheather we really desire to have **more than zero** probability for choosing a bad word
3. focused sampling by beam search:
    - consider k currently best scoring outputs in each set
    - hyper-parameter k denotes the beam width

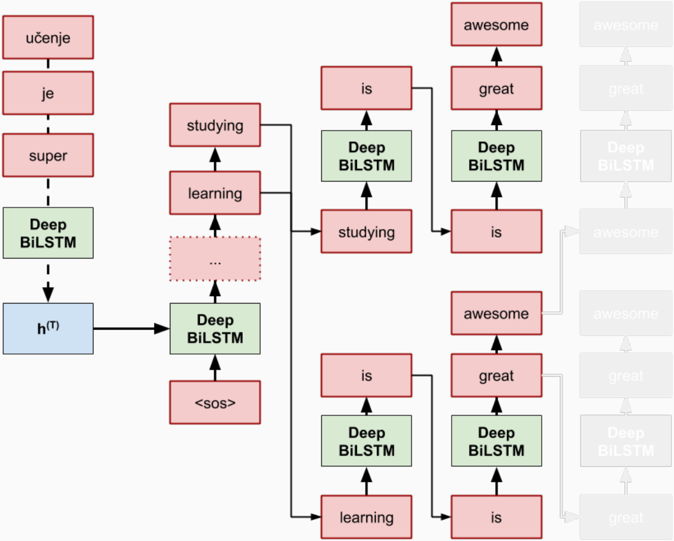
Sequence-to-sequence: beam search



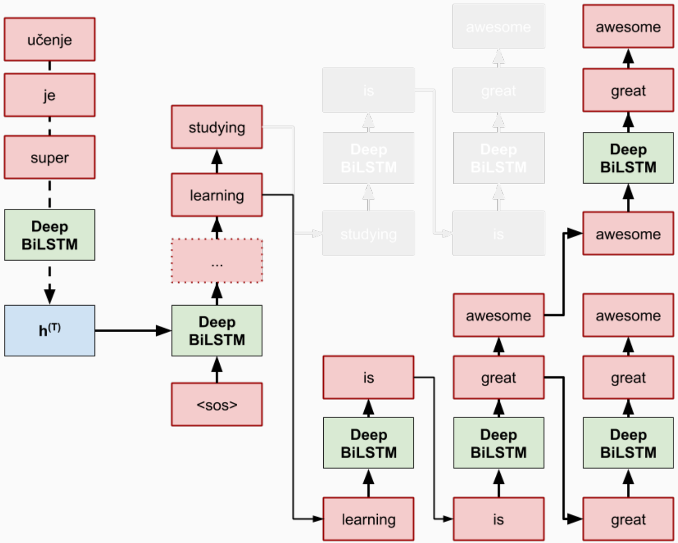
Sequence-to-sequence: vizualizing the beam search



Sequence-to-sequence: vizualizing the beam search



Sequence-to-sequence: vizualizing the beam search



Sequence-to-sequence: summary

Complexity of sequence-to-sequence prediction arises from the variable length of the target sequence:

- instead of ``simple'' context-based classification, the model has to learn to generate the entire sequence.

We approach this problem by separating it into i) reading the input sequence and ii) generating the output sequence:

- encoder and decoder have separate parameters

Sequence-to-sequence: summary (2)

Early training phase is problematic:

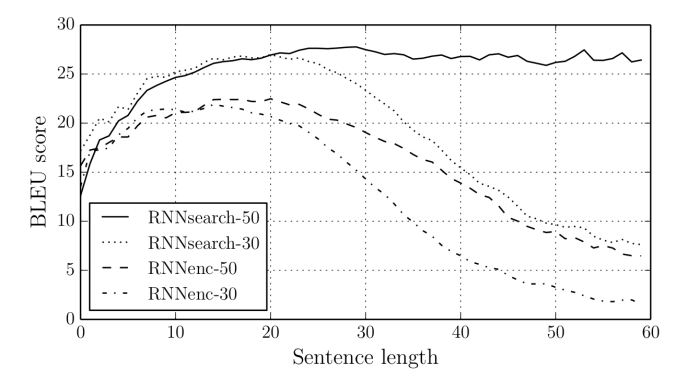
- early training can be improved with **teacher forcing** ("cheat-notes") in some percentage of training samples

Sequence generation is hard:

- it has to maximize **sequence** probability instead of probability of individual components
- it can be approached with **beam search** that tracks k most probable sequences in each step of the generation process

Attention

Success of machine translation for different sentence lengths. RNNsearch models use attention. Figure from [bahdanau14iclr].



Even with state-of-the-art recurrent cells, the translation success visibly deteriorates as the length of sentence increases

- this suggests that recurrent models have a poor memory.

Attention

Motivation:

- *"When I'm translating a sentence, I pay special attention to the word I'm presently translating. When I'm transcribing an audio recording, I listen carefully to the segment I'm actively writing down. And if you ask me to describe the room I'm sitting in, I'll glance around at the objects I'm describing as I do so."*

Our hidden representations are **not** perfect (they have a limited size)

If the cell can not remember all relevant information, could it at least recall where to find it? This idea leads to the new approach:

- denote the hidden representation as a **query**
- denote previous hidden representations as **keys** (memory)
- determine *similarity* between the query and all keys
- aggregate queries through **weighted pooling** where the weights correspond to the above similarity.

Attention: the baseline formulation

Attention returns a kind of scalar similarity between two vectors

- $q^{(t)} = h^{(t)}_{dec}$ and $k^{(t)} = h^{(t)}_{enc}$ correspond to the hidden states:

$$a^{(t)} = \mathrm{attn}(q^{(t)}, k^{(t)}), \qquad a^{(t)} \in \mathbb{R}, q \in \mathbb{R}^q, k \in \mathbb{R}^k \ .$$

We require similarity between the query and *all* keys:

$$a = \mathrm{attn}(q, K), \qquad a \in \mathbb{R}^T, K = [k^{(1)}, \ldots, k^{(T)}] \ .$$

Similarities are normalized to a probability distribution:

$$\alpha = \mathrm{softmax}(a) \ .$$

The attention output is a weighted pool of the hidden encoder states:

$$out_{\mathrm{attn}} = \sum_t^T \alpha_t k^{(t)} \ .$$

Attention: baseline formulation

Attention output is a weighted pool of encoder hidden states

- this vector is concatenated to the decoder state before a word is generated (**it is not used inside the recurrent cells**)

$$h_{dec}^{*(t)} = [h_{dec}^{(t)}; out_{\text{attn}}] \ .$$

How to formulate attention?

1. Bahdanau attention: differentiable module with parameters W_1 (matrix) and w_2 (vector):

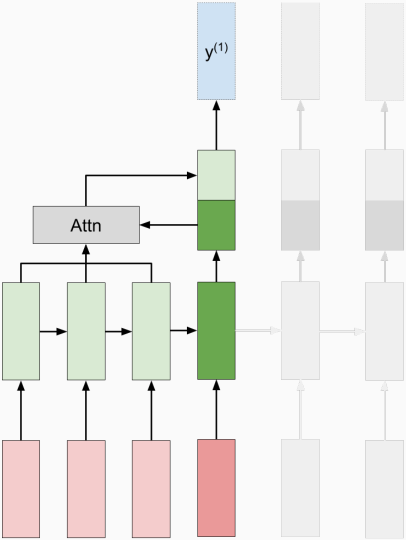
$$a^{(t)} = w_2^\top \cdot \tanh(W_1 \cdot [q^{(t)}; k^{(t)}]) \ .$$

2. Scalar product (condition: dim(q) = dim(k))

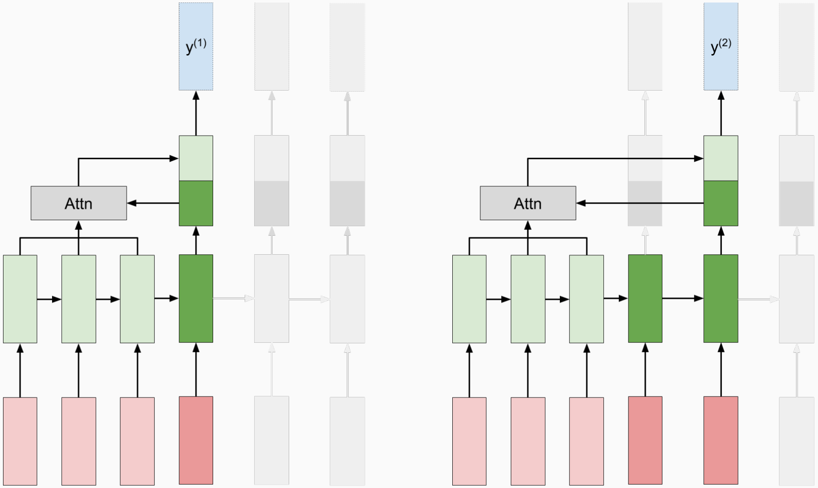
$$a^{(t)} = \frac{q^{(t)\top} \cdot k^{(t)}}{\sqrt{\dim(k)}} \ .$$

   - why do we scale with dimension size k?

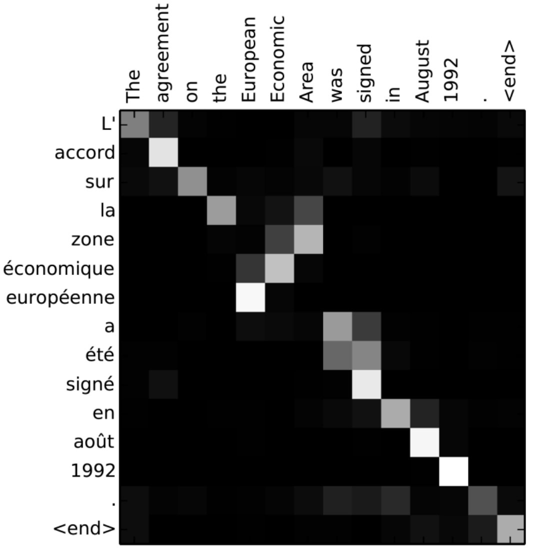
Attention: visualization



Attention: graphic overview



Attention: visualization of similarity



Similarity between the hidden encoder and decoder states for French to English translation.

Attention: extended formulation

We can differentiate between **keys** and **values**:

$$k^{(t)} = f_k(h_{enc}^{(t)}), \qquad v^{(t)} = f_v(h_{enc}^{(t)}) \ .$$

- f_k and f_v transform a hidden vector into **keys** and **values**
- in practice, f_k and f_v are projections.

$$k^{(t)} = W_k h_{enc}^{(t)}, \qquad v^{(t)} = W_v h_{enc}^{(t)} \ .$$

Extended attention:

$$\alpha = \mathrm{softmax}(\mathrm{attn}(q, K)),$$

$$out_{\mathrm{attn}} = \sum_{t}^{T} \alpha_t \, v^{(t)} \ .$$

This above formulation can also be useful beyond sequence-to-sequence translation

Self-attention in sequence classification

If the query comes from the **same** representation as the keys, the attention $\mathrm{attn}(k_i, K)$ may approach one-hot vector e_i:

- we can avoid this with **learned queries** as we show here;
- note that computer vision does not appear to suffer from this problem!

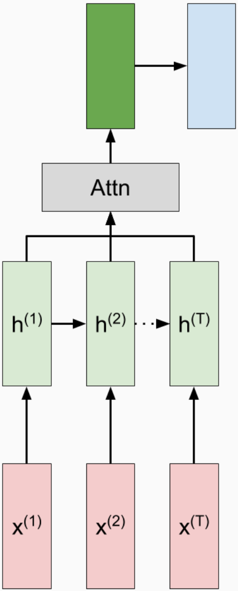
Self-attention with learned queries:

$$\hat{\alpha} = \mathrm{softmax}(attn(\boxed{w}, K)),$$

$$out_{attn} = \sum_{t}^{T} \hat{\alpha}_t v^{(t)} \ .$$

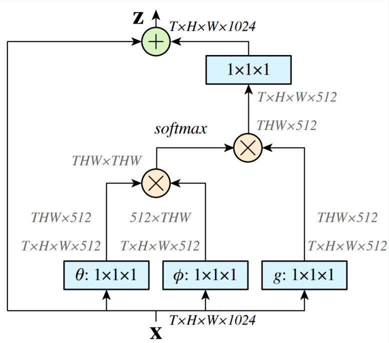
Intuitively, the learned queries correspond to abstract concepts such as *formal writing*, *slang*, *football*, *middle east*, etc.

Self-attention in sequence classification: visualization



Self-attention in computer vision

Some computer vision algorithms capture **long-range** dependencies by extended self-attention without learned-queries.



[wang18cvpr]

Input: abstract representation X

- 4th-o tensor T×H×W×1024
- can be viewed as THW×1024.
- H - height, W - width, T - time

Output: representation Z with improved long-distance connectivity

Input X is projected onto queries (θ), keys (ϕ) and values (g).

Each spatio-temporal feature $x_i \in R^{1024}$ is both a query and a key.

The similarity matrix A (THW $\times$ THW) compares queries with values.

Self-attention in computer vision (details)

The matrix A can be obtained through matrix multiplication:

- other formulations of similarity are easily plugged-in.

$$A = (W_\theta X^\top)^\top \cdot (W_\phi X^\top),$$
$$= (X W_\theta^\top) \cdot (W_\phi X^\top) \ .$$

The weight matrix α is obtained by activating rows of A with softmax.

- A_{ij} reflects similarity of the query $W_\theta x_i$ wrt the value $W_g x_j$

$$\alpha = softmax(A, axis = 1) \ .$$

` Outputs $Z = \{z_i\}$ are linear combinations of values $V = g(x_i)$:

- of course, the weights correspond to the elements of α

$$z_i = \sum_j \alpha_{ij} \cdot g(x_j) \ .$$

Attention: summary

Our best recurrent models still strougle with long sentences

Hence we introduce **attention** to model long-distance connections

- attention is a weighted poool of hidden states
- the weights model **similarity** with respect to some query
    - information relevance depends on the current need
- in recurrent sequence-to-sequence approaches, **query** is the current hidden state of the decoder, while we attend to the hidden states of the **encoder**
- extended variants may be applied to sequence classification and part-of-speech tagging

Attention: summary (2)

Ways to define similarity:

- Bahdanau attention: differentiable module operates on a concatenation of the query and the key
- scalar-product attention: *direct* comparison of the (projected) query with the (projected) value

Attention has been used with practically all RNN variants since its introduction.

Attention is a critical component of contemporaneous algorithms.

Attention is ... all we need?

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions

Models based exclusively on attention

Annotated paper with code samples:
http://nlp.seas.harvard.edu/2018/04/03/attention.html

Applications, examples, SOTA

gail weiss
@gail_w

Follow

Roses are red
RNNs process sequences
But if you look closely
You'll find they have weaknesses

New work with @yoavgo and @yahave on the practical power of different RNN architectures available now on arxiv, and soon at @acl2018!

Weaknesses of RNN Variants

Recent work has shown:

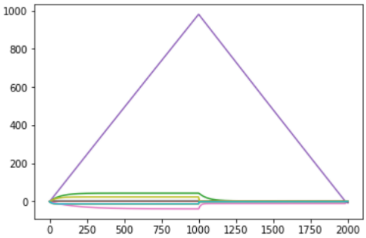
1. recurrent model with a sigmoidal activation and infinite state precision can simulate a Turing machine
2. more recently, this has been extended to include ReLU

However, this holds only under certain conditions:

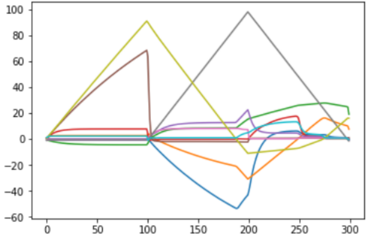
1. the whole sequence has been input into the recurrent neural network, and infinite inference time is available
2. precision is **infinite**.

Counting experiment: RNN architecture need to recognize sequences in the form of $a^n b^n$ or $a^n b^n c^n$

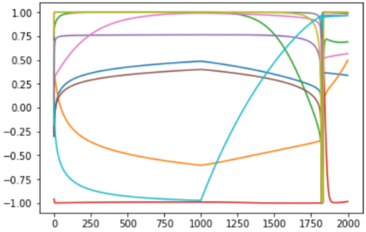
Paper: https://arxiv.org/pdf/1805.04908.pdf



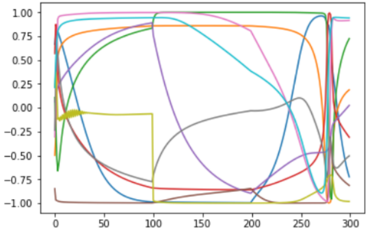
(a) $a^n b^n$-LSTM on $a^{1000}b^{1000}$



(b) $a^n b^n c^n$-LSTM on $a^{100}b^{100}c^{100}$



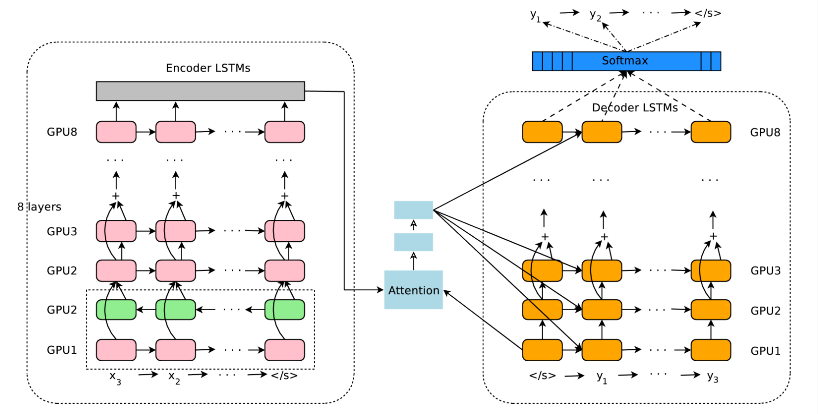
(c) $a^n b^n$-GRU on $a^{1000}b^{1000}$



(d) $a^n b^n c^n$-GRU on $a^{100}b^{100}c^{100}$

Machine translation

Google translate https://research.googleblog.com/2016/09/a-neural-network-for-machine.html



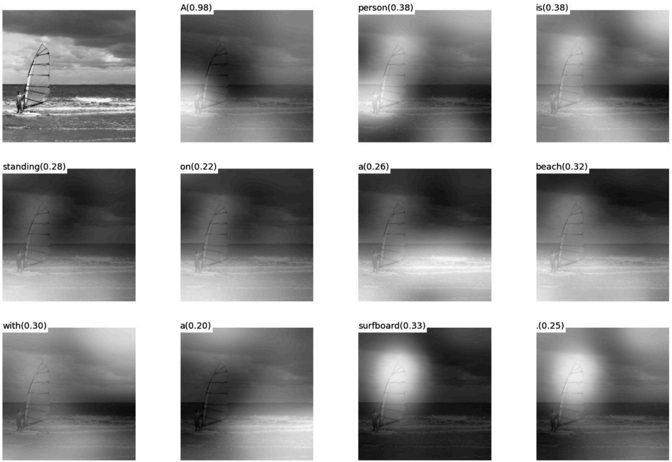
Google translate network arhitecture

Speech generation

https://google.github.io/tacotron/

Demo: `https://deepmind.com/blog/wavenet-generative-model-ra`

Image captioning



(b) A person is standing on a beach with a surfboard.

Image captioning with attention and recurrent networks

Image genration based on text description

TEXT DESCRIPTION

An astronaut   Teddy bears   A bowl of soup

riding a horse   lounging in a tropical resort in space   playing basketball with cats in space

in a photorealistic style   in the style of Andy Warhol   as a pencil drawing

→

DALL·E 2

DALLE-2 `https://openai.com/dall-e-2/`

A robot wrote this entire article. Are you scared yet, human?

GPT-3

We asked GPT-3, OpenAI's powerful new language generator, to write an essay for us from scratch. The assignment? To convince us robots come in peace

- For more about GPT-3 and how this essay was written and edited, please read our editor's note below

GPT-3 `https://www.theguardian.com/commentisfree/2020/sep/08/robot-wrote-this-article-gpt-3`; `https://arxiv.org/abs/2005.14165`

Solving textual mathematical problems

Question

Ali is a dean of a private school where he teaches one class. John is also a dean of a public school. John has two classes in his school. Each class has 1/8 the capacity of Ali's class which has the capacity of 120 students. What is the combined capacity of both schools?

Answer

Method: **175B Verification**

```
Ali's class has a capacity of 120 students.
Each of John's classes has a capacity of 120/8 = 15 students.
The total capacity of John's two classes is 15 students * 2 classes =
30 students.
The combined capacity of the two schools is 120 students + 30 students =
150 students.
```

Verifiers: `https://openai.com/blog/grade-school-math/`

Demo: `https://www.youtube.com/watch?v=SGUCcjHTmGY`

Blog post: `https://openai.com/blog/openai-codex/`

Questions?

Questions? :)

In the book

- Relevant chapters
  - 10.1, 10.2, 10.3, 10.4, 10.5, 10.7, 10.10, 10.11