Applied Machine Learning

Neural Networks for Sequences

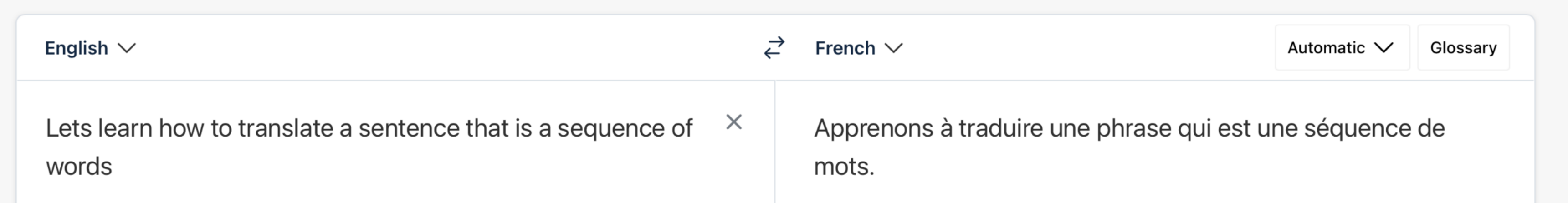
Isabeau Prémont-Schwarz

McGill
School of Computer Science

COMP 551 (Fall 2023)

Deep Neural Networks

- Neural Networks for **Tabular Data**
  - MLP
- Neural Networks for **Images**
  - CNN
- Neural Networks for **Sequences**
  - input is a sequence, the output is a sequence, or both are sequences
    - *e.g. machine translation, speech recognition, text classification, image captioning*



Learning objectives

- Recurrent neural networks (RNNs)
  - 3 different models for different input/output
  - training with back propagation through time
- understand the attention mechanisms
- The architecture of transformer

Recurrent neural networks (RNNs)

maps sequences to sequences in a stateful way

i.e. prediction $\hat{y}_t$ depends on x_t and hidden state of the network h_t, which is updated over time

- Vec2Seq (sequence generation)
- Seq2Vec (sequence classification)
- Seq2Seq (sequence translation)

Recurrent neural networks (RNNs)

- Vec2Seq (sequence generation)
  - output, $y_{1:T}$ is generated one token at a time
  - at each step we sample y_t from the hidden state h_t and then feed it back to the model to get h_{t+1}

arbitrary-length sequence of vectors

$$f_\theta : \mathbb{R}^D \to \mathbb{R}^{N_\infty C}$$

D: input vector size

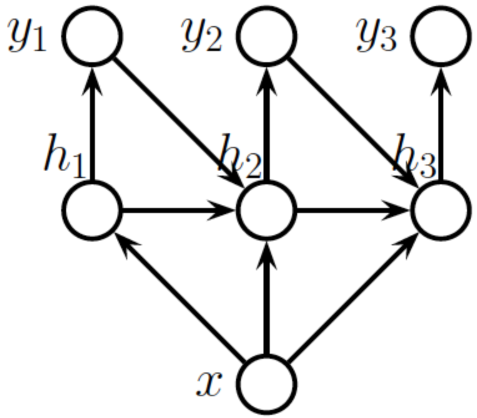
N_∞: *arbitrary-length sequence of vectors of length C*

C: each output vector size

conditional generative model:

$$p(y_{1:T}|x) = \sum_{h_{1:T}} p(y_{1:T}, h_{1:T}|x) = \sum_{h_{1:T}} \prod_{t=1}^{T} p(y_t|h_t)p(h_t|h_{t-1}, y_{t-1}, x)$$

with the initial hidden state$p(h_1|h0, y0, x) = p(h_1|x)$



Recurrent neural networks (RNNs)

- Vec2Seq (sequence generation) $f_\theta : \mathbb{R}^D \to \mathbb{R}^{TC}$

conditional generative model:

$$p(y_{1:T}|x) = \sum_{h_{1:T}} p(y_{1:T}, h_{1:T}|x) = \sum_{h_{1:T}} \prod_{t=1}^{T} p(y_t|h_t) p(h_t|h_{t-1}, y_{t-1}, x)$$

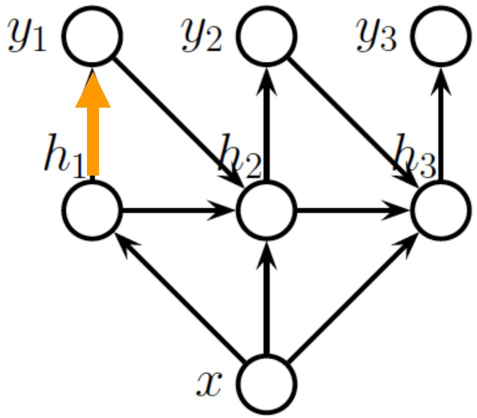
hidden-to-output weights

- real-valued output: $\hat{y}_t = W_{hy} h_t$

$$p(y_t|h_t) = \mathcal{N}(y_t|\hat{y}_t, \mathbf{I})$$

- categorical output: $\hat{y}_t = \mathrm{softmax}(W_{hy} h_t)$

$$p(y_t|h_t) = \mathrm{Categorical}(y_t|\hat{y}_t)$$



** per usual dropping bias terms for simplicity

Recurrent neural networks (RNNs)

- Vec2Seq (sequence generation)

$$f_\theta : \mathbb{R}^D \to \mathbb{R}^{TC}$$

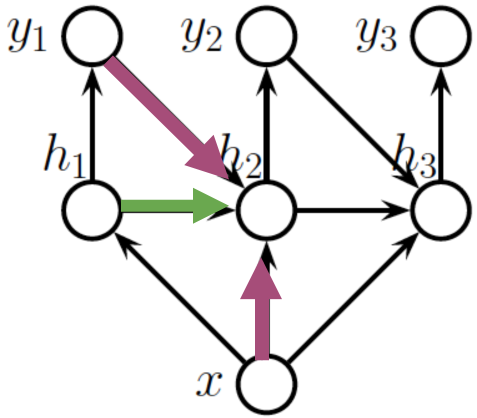
conditional generative model:

$$p(y_{1:T}|x) = \sum_{h_{1:T}} p(y_{1:T}, h_{1:T}|x) = \sum_{h_{1:T}} \prod_{t=1}^{T} p(y_t|h_t) p(h_t|h_{t-1}, y_{t-1}, x)$$

hidden state:

$$p(h_t|h_{t-1}, y_{t-1}, x) = \mathbb{I}(h_t = f(h_{t-1}, y_{t-1}, x))$$

$$h_t = \varphi(\underbrace{W_{xh}}_{\text{input-to-hidden weights}}[x; y_{t-1}] + \underbrace{W_{hh}}_{\text{hidden-to-hidden weights}} h_{t-1})$$



Recurrent neural networks (RNNs)

- Vec2Seq (sequence generation)

$$f_\theta : \mathbb{R}^D \to \mathbb{R}^{TC}$$

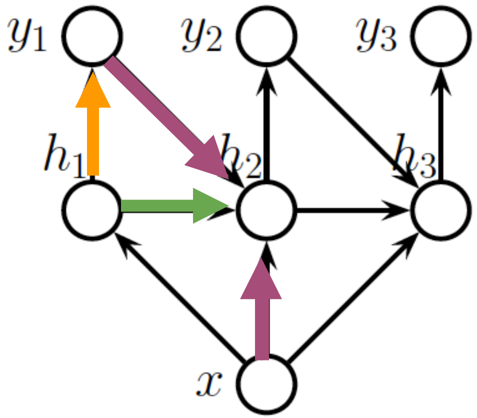
model

hidden-to-output weights

$$\hat{y}_t = g(W_{hy} h_t)$$

input-to-hidden weights hidden-to-hidden weights

$$h_t = \varphi(W_{xh}[x; y_{t-1}] + W_{hh} h_{t-1})$$



RNNs are powerful

- In theory can have unbounded memory and are as powerful as a Turing machine
- In practice, memory size is determined by the size of the latent space and strength of the parameters

Recurrent neural networks (RNNs)

- Vec2Seq (sequence generation)

conditional generative model:

$$p(y_{1:T}|x) = \sum_{h_{1:T}} p(y_{1:T}, h_{1:T}|x) = \sum_{h_{1:T}} \prod_{t=1}^{T} p(y_t|h_t) p(h_t|h_{t-1}, y_{t-1}, x)$$

language modelling: generating sequences unconditionally (by setting $x = \emptyset$) which is learning joint probability distributions over sequences of discrete tokens, i.e., $p(y_1, \ldots, y_T)$

Example:

character level RNN trained on the book The Time Machine by H. G. Wells (32,000 words and 170k character)

Output when given prefix "the":

the *githa some thong the time traveller held in his hand was a glitteringmetallic framework scarcely larger than a small clock and verydelicately made there was ivory in it and the latter than s bettyre tat howhong s ie time thave ler simk you a dimensions le ghat dionthat shall travel indifferently in any direction of space and timeas the driver determinesfilby contented himself with laughterbut i have experimental verification said the time travellerit would be remarkably convenient for the histo*

See the code here, read more here

Recurrent neural networks (RNNs)

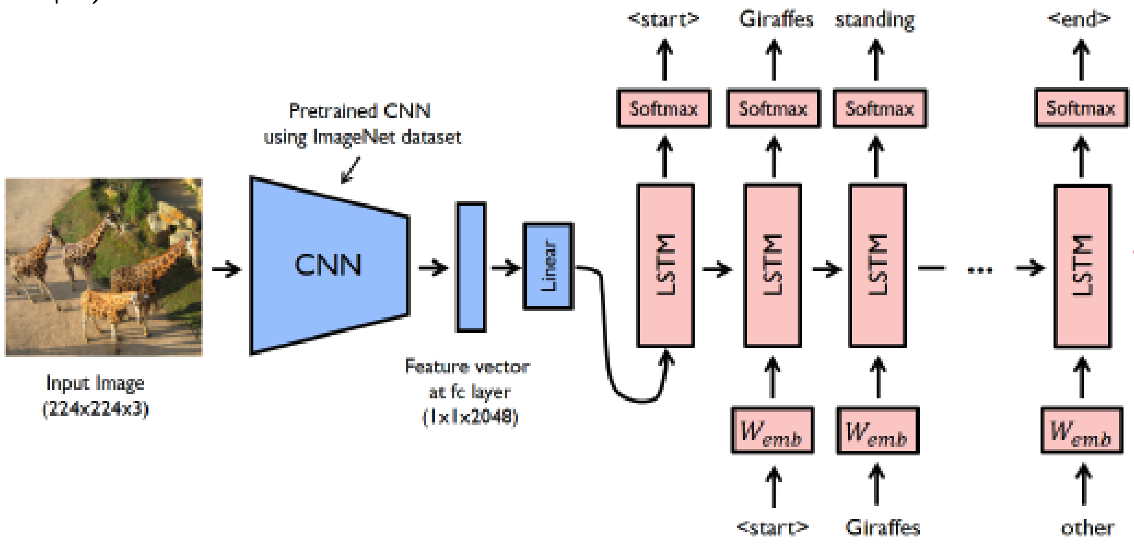
- Vec2Seq (sequence generation)

conditional generative model:

$$p(y_{1:T}|x) = \sum_{h_{1:T}} p(y_{1:T}, h_{1:T}|x)$$

Example:

CNN-RNN model for image captioning when x is embedding by a CNN



a specific kind of RNN

See more here

optional

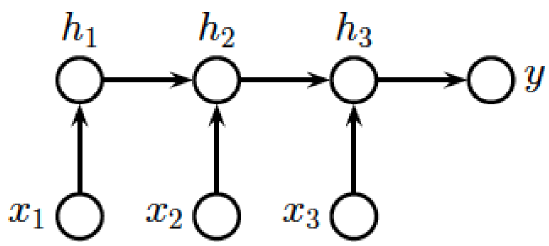
Recurrent neural networks (RNNs)

- Seq2Vec (sequence classification) $f_\theta : \mathbb{R}^{TD} \to \mathbb{R}^C$
  - predict a single fixed-length output vector given a variable length sequence as input $y \in \{1, \dots, C\}$

use the final state:

$$\hat{y} = \text{softmax}(W h_T)$$

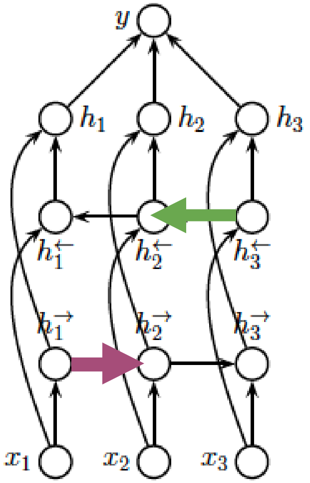
$$p(y|x_{1:T}) = \text{Categorical}(y|\hat{y})$$



Bi-directional RNN:

the hidden states of the RNN depend on the past and future context

gives better results



optional

Recurrent neural networks (RNNs)

- Seq2Vec (sequence classification) $f_\theta : \mathbb{R}^{TD} \to \mathbb{R}^C$
  - predict a single fixed-length output vector given a variable length sequence as input

$$h_t^{\rightarrow} = \varphi\left(W_{xh}^{\rightarrow} x_t + W_{hh}^{\rightarrow} h_{t-1}^{\rightarrow}\right)$$

$$h_t^{\leftarrow} = \varphi\left(W_{xh}^{\leftarrow} x_t + W_{hh}^{\leftarrow} h_{t+1}^{\leftarrow}\right)$$

$$h_t = [h_t^{\rightarrow}, h_t^{\leftarrow}]$$

$$\overline{h} = \frac{1}{T}\sum_{t=1}^{T} h_t$$

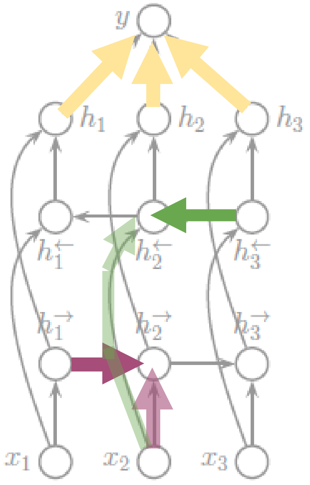
$$\hat{y} = \text{softmax}(W\overline{h})$$

$$p(y|x_{1:T}) = \text{Categorical}(y|\hat{y})$$

Bi-directional RNN:

the hidden states of the RNN depend on the past and future context

gives better results



optional

Recurrent neural networks (RNNs)

- Seq2Vec (sequence classification) $f_\theta : \mathbb{R}^{TD} \rightarrow \mathbb{R}^{C}$
  - predict a single fixed-length output vector given a variable length sequence as input

Example:

Sentiment classification with word level **bidirectional** LSTM trained on a subset of the Internet Movie Database (IMDB) reviews. (20k positive and 20k negative examples)

Prediction examples for two inputs:

'this movie is so great' $\Rightarrow$ 'positive'

'this movie is so bad' $\Rightarrow$ 'negative'

see the code here, and read more here

optional

Recurrent neural networks (RNNs)

- Seq2Seq (sequence translation)
  - aligned: $T = T'$
  - unaligned: $T \neq T'$

$$f_\theta : \mathbb{R}^{TD} \to \mathbb{R}^{T'C}$$

optional

Recurrent neural networks (RNNs)

- Seq2Seq (sequence translation)
  - aligned: $T = T'$

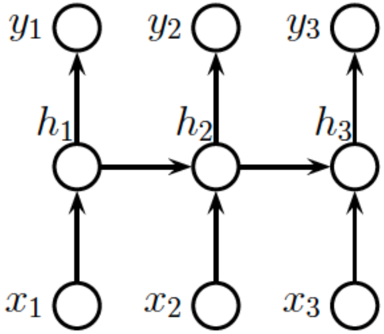
$$f_\theta : \mathbb{R}^{TD} \to \mathbb{R}^{TC}$$

modify the RNN as:

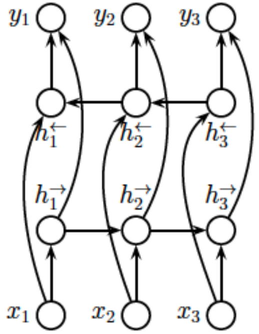
$$p\left(y_{1:T} \mid x_{1:T}\right) = \sum_{h_{1:T}} \prod_{t=1}^{T} p\left(y_t \mid h_t\right) \mathbb{I}\left(h_t = f\left(h_{t-1}, x_t\right)\right)$$

initial state: $h_1 = f\left(h_0, x_1\right) = f_0\left(x_1\right)$

dense sequence labeling:
predict one label per location



Bi-directional



optional

Recurrent neural networks (RNNs)

- Seq2Seq (sequence translation)
  - aligned: $T = T'$

$f_\theta : \mathbb{R}^{TD} \to \mathbb{R}^{TC}$

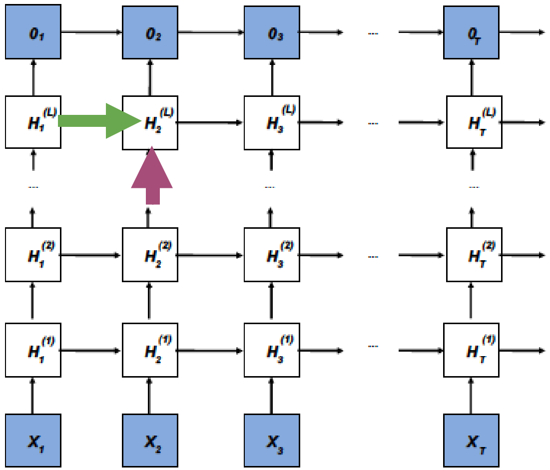
modify the RNN as:

$$p\left(y_{1:T} \mid x_{1:T}\right) = \sum_{h_{1:T}} \prod_{t=1}^{T} p\left(y_t \mid h_t\right) \mathbb{I}\left(h_t = f\left(h_{t-1}, x_t\right)\right)$$

more depth to be more expressive

$$h_t^l = \varphi_l \left(\underbrace{W_{xh}^l}_{\text{input-to-hidden weights}} h_t^{l-1} + \underbrace{W_{hh}^l}_{\text{hidden-to-hidden weights}} h_{t-1}^l \right)$$

$$y_t = W_{hy} h_t^L$$



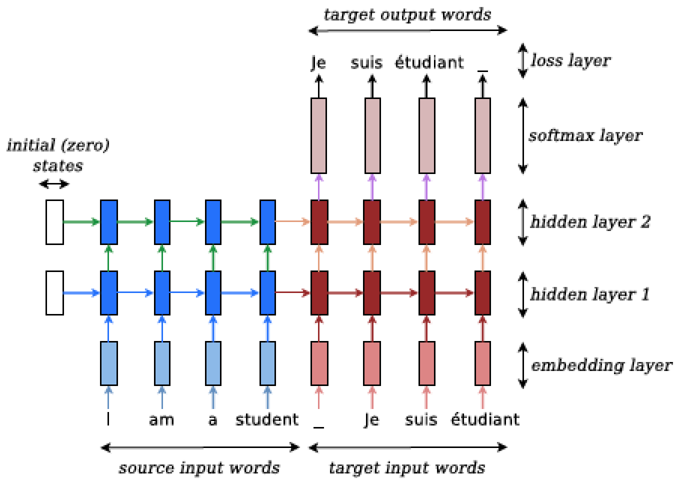
optional

Recurrent neural networks (RNNs)

- Seq2Seq (sequence translation)
  - unaligned: $T \neq T'$

$$f_\theta : \mathbb{R}^{TD} \to \mathbb{R}^{T'C}$$

Example: translating English to French



- encode the input sequence to get the context vector, the last state of an RNN, $c = f_e(x_{1:T})$
- generate the output sequence using an RNN decoder, $y_{1:T'} = f_d(c)$

see code here

Training: Backpropagation through time (BPTT)

unroll the computation graph, then apply the backpropagation

Example:

model

$$h_t = W_{hx} x_t + W_{hh} h_{t-1}$$

$$\hat{y}_t = W_{hy} h_t$$

loss

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

derivatives

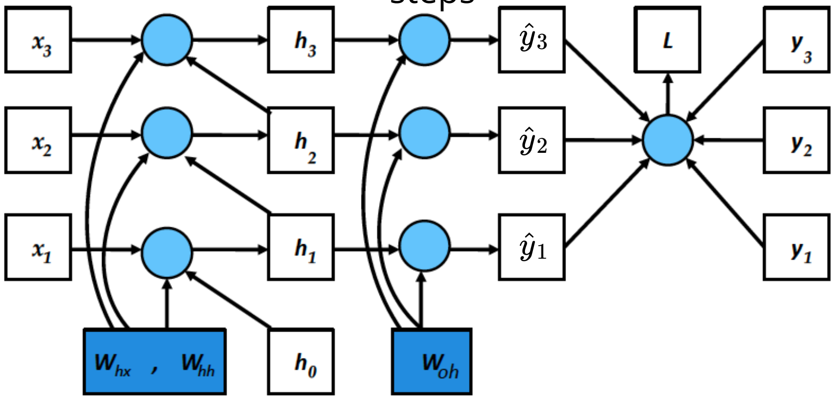
$$\frac{\partial L}{\partial W_{hx}}$$

$$\frac{\partial L}{\partial W_{hh}}$$

$$\frac{\partial L}{\partial W_{hy}}$$

Example:

An RNN unrolled (vertically) for 3 time steps



Training: Backpropagation through time (BPTT)

unroll the computation graph, then apply the backpropagation

Example:

model

$$h_t = W_{hx} x_t + W_{hh} h_{t-1} = f\left(x_t, h_{t-1}, \underbrace{w_h}_{[\mathrm{vec}(W_{hx});\, \mathrm{vec}(W_{hh})]}\right)$$

$$\hat{y}_t = W_{hy} h_t = g(h_t, w_y)$$

loss

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

derivatives

$$\left.\begin{array}{c} \frac{\partial L}{\partial W_{hx}} \\ \frac{\partial L}{\partial W_{hh}} \\ \frac{\partial L}{\partial W_{hy}} \end{array}\right\} \quad \frac{\partial L}{\partial w_h} = \frac{1}{T} \sum_{t=1}^{T} \frac{\partial \ell(y_t, \hat{y}_t)}{\partial w_h} = \frac{1}{T} \sum_{t=1}^{T} \frac{\partial \ell(y_t, \hat{y}_t)}{\partial \hat{y}_t} \frac{\partial g(h_t, w_y)}{\partial h_t} \frac{\partial h_t}{\partial w_h}$$

$$\frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial w_h}$$

expand this recursively

$$\frac{\partial h_t}{\partial w_h} = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^{t} \frac{\partial f(x_j, h_{j-1}, w_h)}{\partial h_{j-1}} \right) \frac{\partial f(x_i, h_{i-1}, w_h)}{\partial w_h}$$

see code here

Gating and long term memory

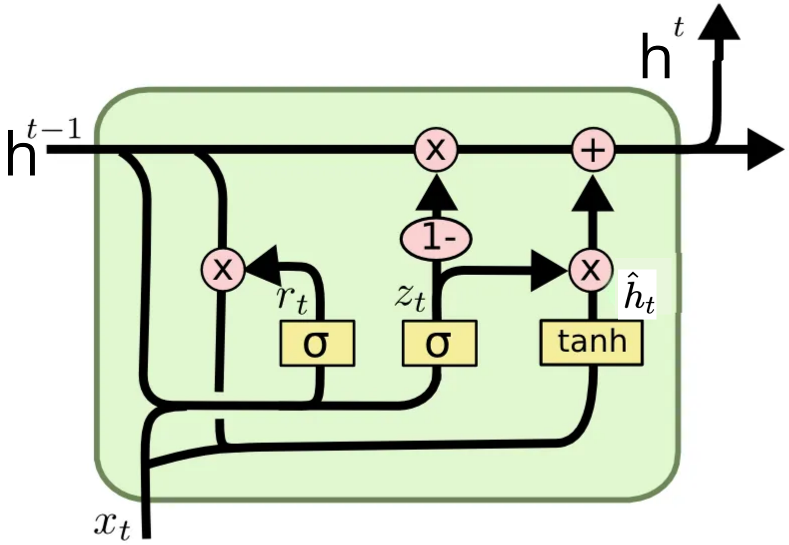
Vanishing and exploding gradients

activations can decay or explode as we go forwards and backwards in time

RNN variations that circumvent this:

- Gated recurrent units (GRU)
    - learns when to update the hidden state, by using a gating unit
- Long short term memory (LSTM)
    - augments the hidden state with a memory cell

The Gated Recurrent Unit (GRU):



$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$
$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r)$$
$$\hat{h}_t = \phi(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h)$$
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t$$

Attention

$$z = g(Wx)$$

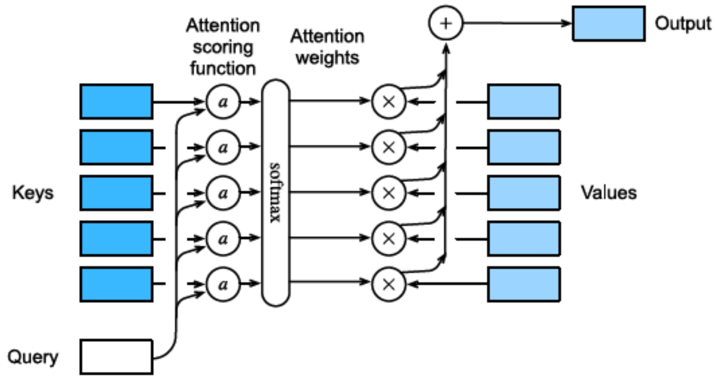
Instead of linear combination of the input activations, the model dynamically decides (in an input dependent way) which one to use based on how similar the input **query** vector $q \in \mathbb{R}^q$ is to a set of m **keys** K $\in \mathbb{R}^{m \times k}$. If q is most similar to key i, then we use value v_i.

$$\operatorname{Attn}(q, (k_1, v_1), \ldots, (k_m, v_m)) = \operatorname{Attn}(q, (k_{1:m}, v_{1:m})) = \sum_{i=1}^{m} \alpha_i(q, k_{1:m}) v_i \in \mathbb{R}^v$$

$$\alpha_i(q, k_{1:m}) = \operatorname{softmax}_i([a(q, k_1), \ldots, a(q, k_m)]) = \frac{\exp(a(q,k_i))}{\sum_{j=1}^{m} \exp(a(q,k_j))}$$

attention weight

The attention weights are computed from an attention score function $a(q, k_i) \in \mathbb{R}$, which gives the similarity of query q to key k_i.



Parametric Attention

The attention weights are computed from an attention score function $a(q, k_i) \in \mathbb{R}$, which gives the similarity of query $q \in \mathbb{R}^q$ to key $k_i \in \mathbb{R}^k$

- queries and keys both have different sizes
  - map them to a common embedding space of size h, then pass these into an MLP

$$a(q,k) = w_v^\top \tanh\left(\underbrace{W_q}_{\in \mathbb{R}^{h\times q}} q + \underbrace{W_k}_{\in \mathbb{R}^{h\times k}} k\right) \in \mathbb{R}$$

- queries and keys both have length $d = q = k$
  - so we can compute $q^T k$ directly: $a(q,k) = q^\top k / \sqrt{d} \in \mathbb{R}$
  - for minibatches of n vectors this gives:

$$\mathrm{Attn}(\underbrace{Q}_{\in \mathbb{R}^{n\times d}}, \overbrace{K}^{\in \mathbb{R}^{m\times d}}, \underbrace{V}_{\in \mathbb{R}^{m\times v}}) = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right) V \in \mathbb{R}^{n\times v}$$

Seq2Seq with attention

use attention to the input sequence in order to capture contexual embeddings of each input

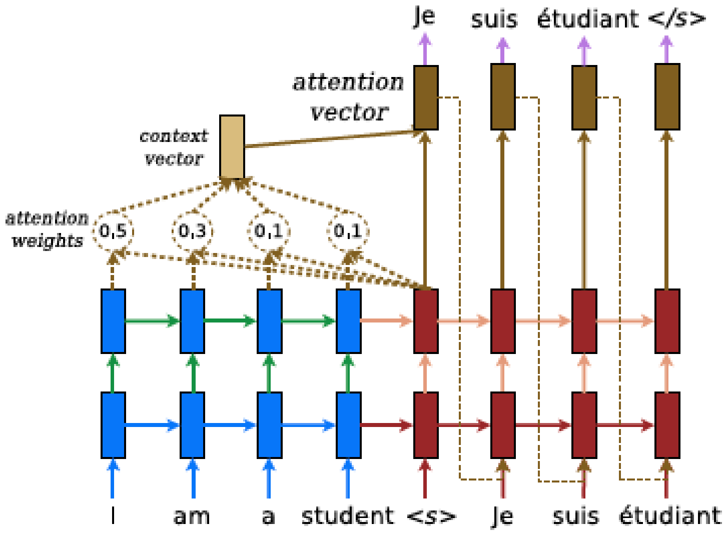
- query is the hidden state of the decoder at the previous step
- keys and values are the hidden states from the encoder

Gives better results for machine translations

self attention:

we can also modify the model so the encoder attends to itself

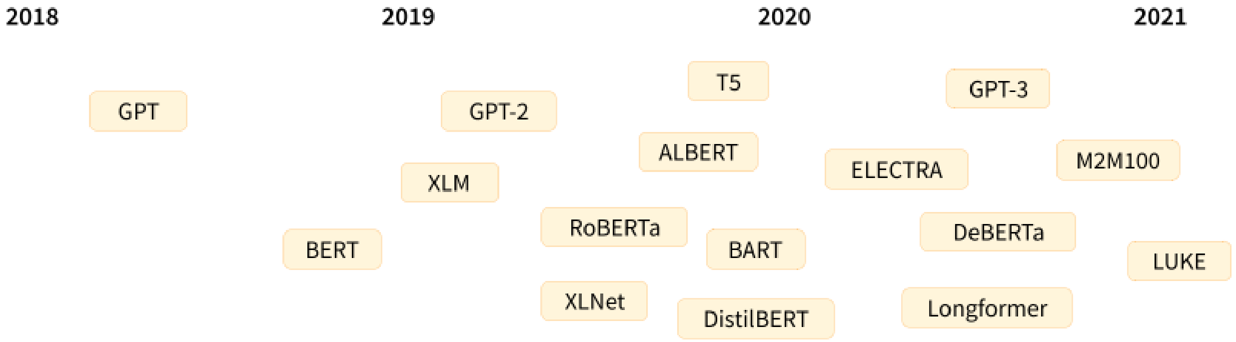
Example: translating English to French



Transformers

a seq2seq model which uses attention in the encoder as well as the decoder, thus eliminating the need for RNNs

- Self-attention
- Multi-headed attention
- Positional encoding



read more here

Transformers: self-attention

given a sequence of input tokens $x_1, \ldots, x_n$, generate a sequence of outputs of the same size with:

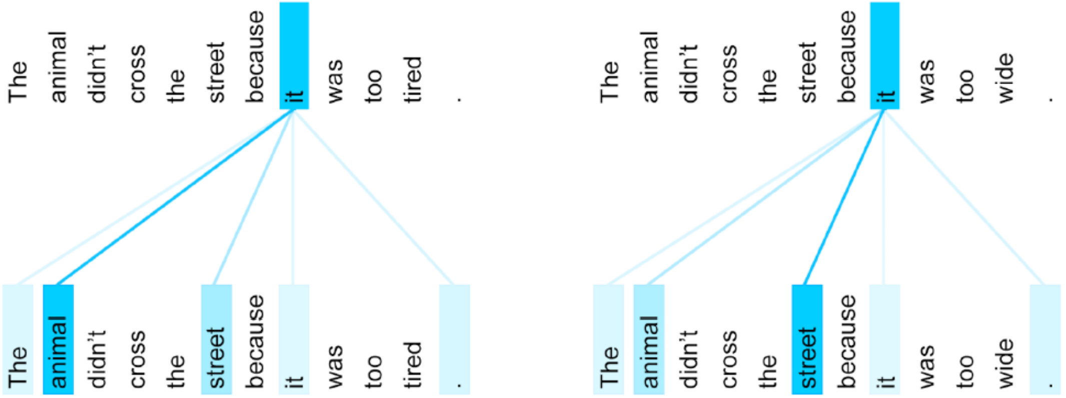
$$y_i = \mathrm{Attn}\Big(\overbrace{\underbrace{x_i}_{\in \mathbb{R}^d}}^{\text{query}}, \underbrace{(x_1, x_1), \ldots, (x_n, x_n)}_{\text{(key, value)s}}\Big)$$

for decoder we set $x_i = y_{i-1}$ and $n = i - 1$

this gives improved representations of context

Example:

coreference resolution: encoder self-attention for the word "it" differs depending on the input context which is important in translation, e.g. what pronoun to use in French



optional

Transformers: multi-headed attention

use multiple attention matrices, to capture different notions of similarity

with projection matrices: $W_i^{(q)} \in \mathbb{R}^{p_q \times d_q}, W_i^{(k)} \in \mathbb{R}^{p_k \times d_k}$, and $W_i^{(v)} \in \mathbb{R}^{p_v \times d_v}$

$$h_i = \mathrm{Attn}\left(W_i^{(q)} \underbrace{q}_{\in \mathbb{R}^{d_q}}, \left\{W_i^{(k)} \underbrace{k_j}_{\in \mathbb{R}^{d_k}}, W_i^{(v)} \underbrace{v_j}_{\in \mathbb{R}^{d_v}}\right\}\right) \in \mathbb{R}^{p_v}$$

We then stack the h heads together, and project with $W_o \in \mathbb{R}^{p_o \times h p_v}$:

$$h = \mathrm{MHA}\left(q, \{k_j, v_j\}\right) = W_o \begin{pmatrix} h_1 \\ \vdots \\ h_h \end{pmatrix} \in \mathbb{R}^{p_o}$$

optional

Transformers: positional encoding

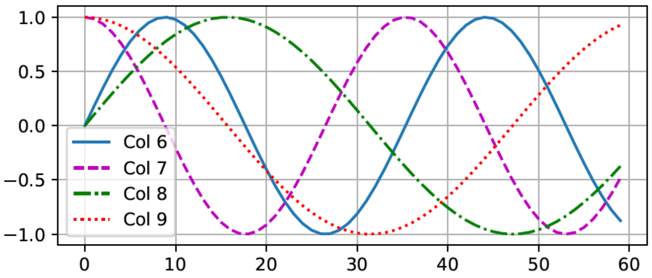
attention is permutation invariant, and hence ignores the input word ordering. To overcome this, we can concatenate the word embeddings with a positional embedding so that the model knows what order the words occur in

$$p_{i,2j} = \sin\left(\frac{i}{10000^{2j/d}}\right)$$
$$p_{i,2j+1} = \cos\left(\frac{i}{10000^{2j/d}}\right)$$

$$\mathrm{POS}(\mathrm{Embed}(X)) = \underbrace{X}_{\in \mathbb{R}^{n\times d}} + \underbrace{P}_{\in \mathbb{R}^{n\times d}}$$

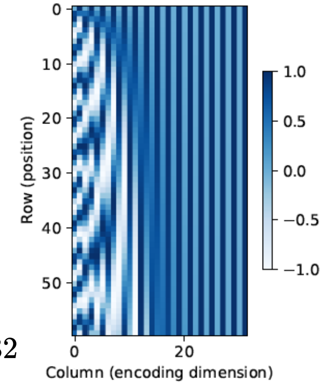
```
0 in binary is 000
1 in binary is 001
2 in binary is 010
3 in binary is 011
4 in binary is 100
5 in binary is 101
6 in binary is 110
7 in binary is 111
```

lower columns have higher frequencies



Example:

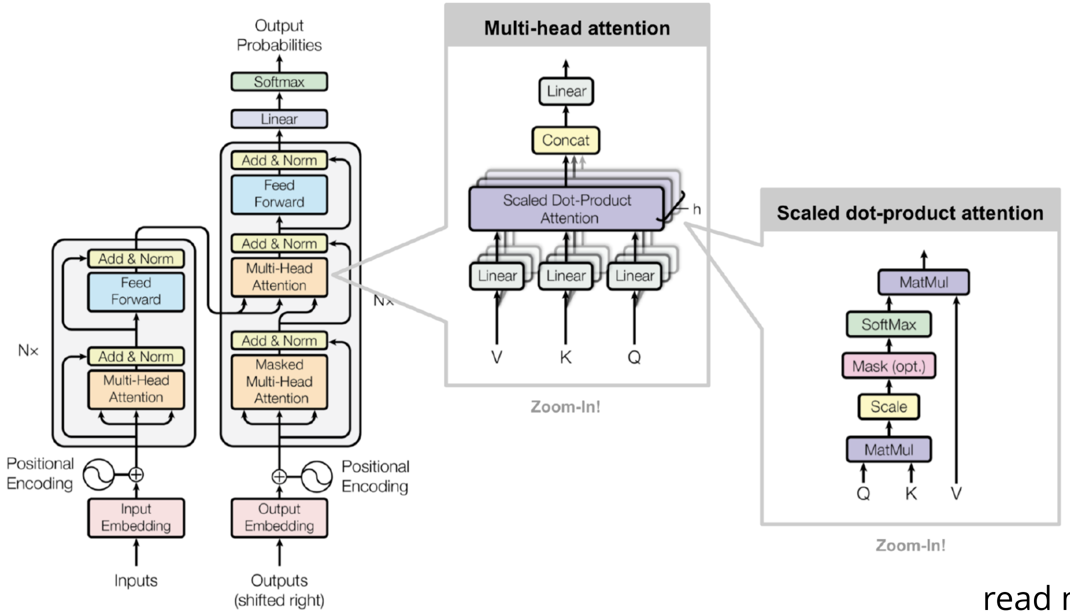
$n = 60,\ d = 32$



read more here

Transformers: putting it all together

A transformer is a seq2seq model that uses self-attention for the encoder and decoder rather than an RNN. The encoder uses a series of encoder blocks, each of which uses multi-headed attention, residual connections, and layer normalization



read more here, see the code here

Language models

- ELMo (Embeddings from Language Model)
  - RNN based, trained unsupervised to minimize the negative log likelihood of the input sentence, i.e. $y_t = x_{t-1}$
- BERT (Bidirectional Encoder Representations from Transformers)
  - Transformer-based: map a modified version of a sequence back to the unmodified form and compute the loss at the masked locations: fill-in-the-blank :

```
Let's make [MASK] chicken! [SEP] It [MASK] great with orange sauce
```

- GPT (Generative Pre-training Transformer)
  - uses a masked transformer as the decoder, see an open-source model here (20 billion parameters)

Summary

- Recurrent neural networks (RNNs)
  - Vec2Seq (sequence generation)
  - Seq2Vec (sequence classification)
  - Seq2Seq (sequence translation)
  - training with back propagation through time
- attention mechanisms, self-attention and multi-headed attention
- The architecture of transformer
- language models with transformer