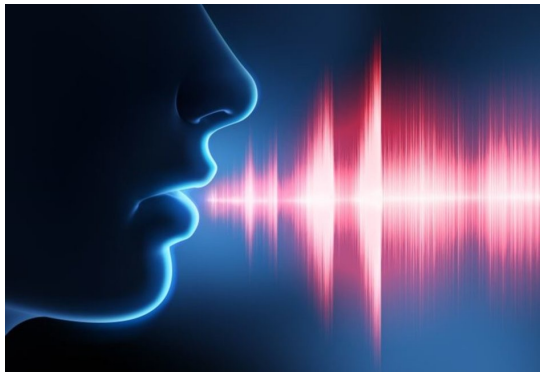
Recurrent Neural Networks

RNN basics

Yingzhen Li (yingzhen.li@imperial.ac.uk)

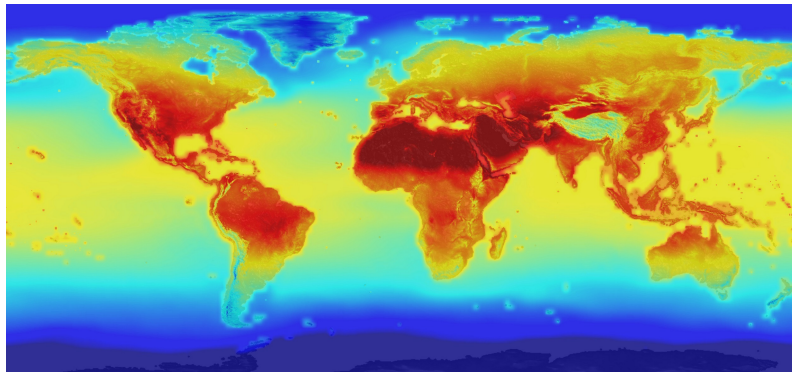
Sequential data is everywhere

Video data



Speech data

Financial time series data



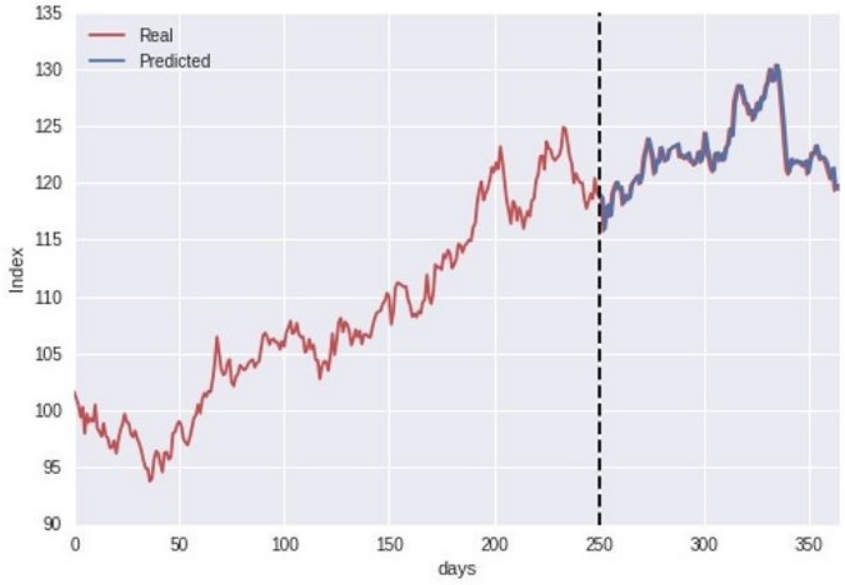
Climate science data (spatial-temporal)

Sequential data is everywhere

Time series prediction:

- Data sequence: $(x_1, \dots, x_T)$
  - x_t: data frame at time t
- Goal: predict future values

$$(x_1, \dots, x_T) \rightarrow x_{T+1}, x_{T+2}, \dots$$

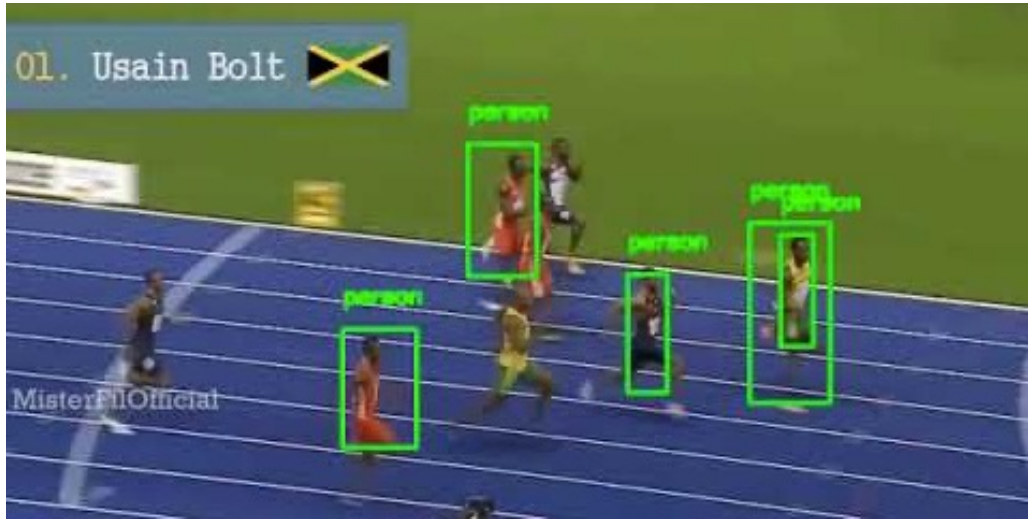


Sequential data is everywhere

Object tracking in videos:

- Data sequence: $(x_1, \dots, x_T)$
  - x_t: video frame at time t
- Label sequence: $(y_1, \dots, y_T)$
  - y_t: object identifier, bounding box coordinates, … at time t
- Goal: learn a mapping

$$(x_1, \dots, x_T) \rightarrow (y_1, \dots, y_T)$$

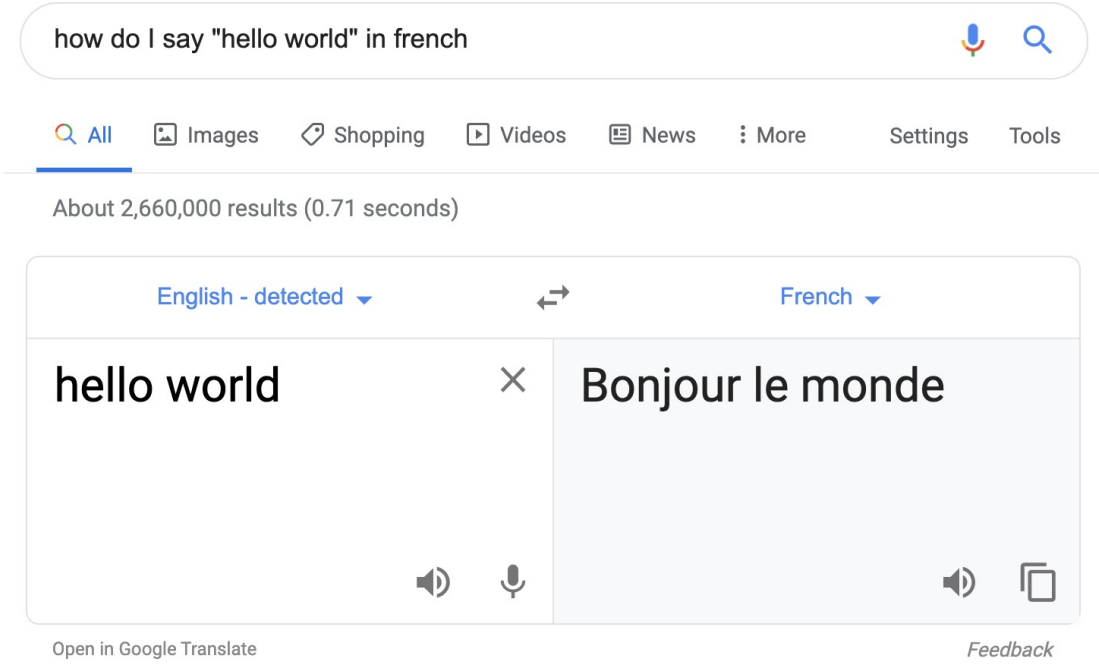


Sequential data is everywhere

Machine translation (e.g. EN to FR):

- Data sequence: $x = (x_1, \dots, x_T)$
  - x_t: the t^{th} word in the English sentence
- Output sequence: $y = (y_1, \dots, y_L)$
  - y_l: the l^{th} word in the French sentence
- Goal: learn a mapping

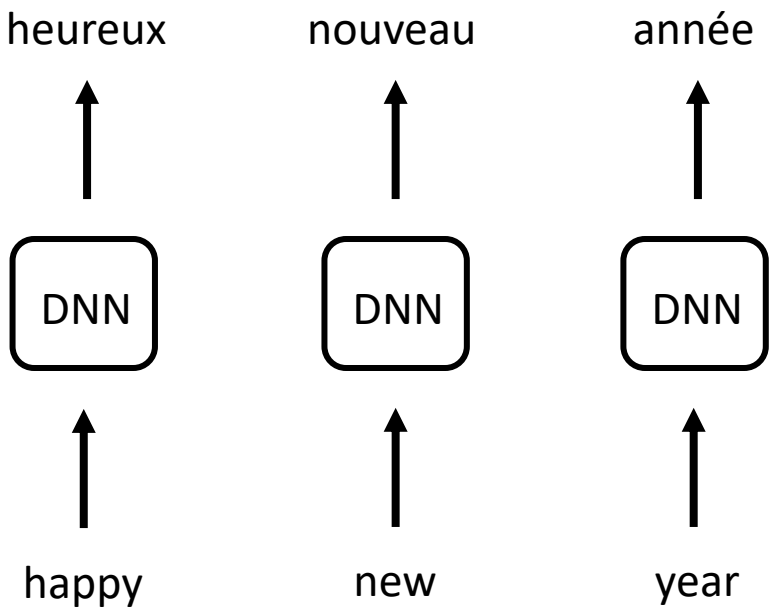
$$x \to y$$



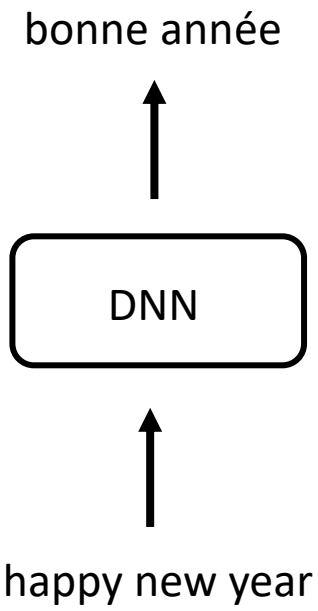
Deep learning's solution: recurrent neural networks

Why Recurrent Neural Networks

Machine translation as a motivating example:



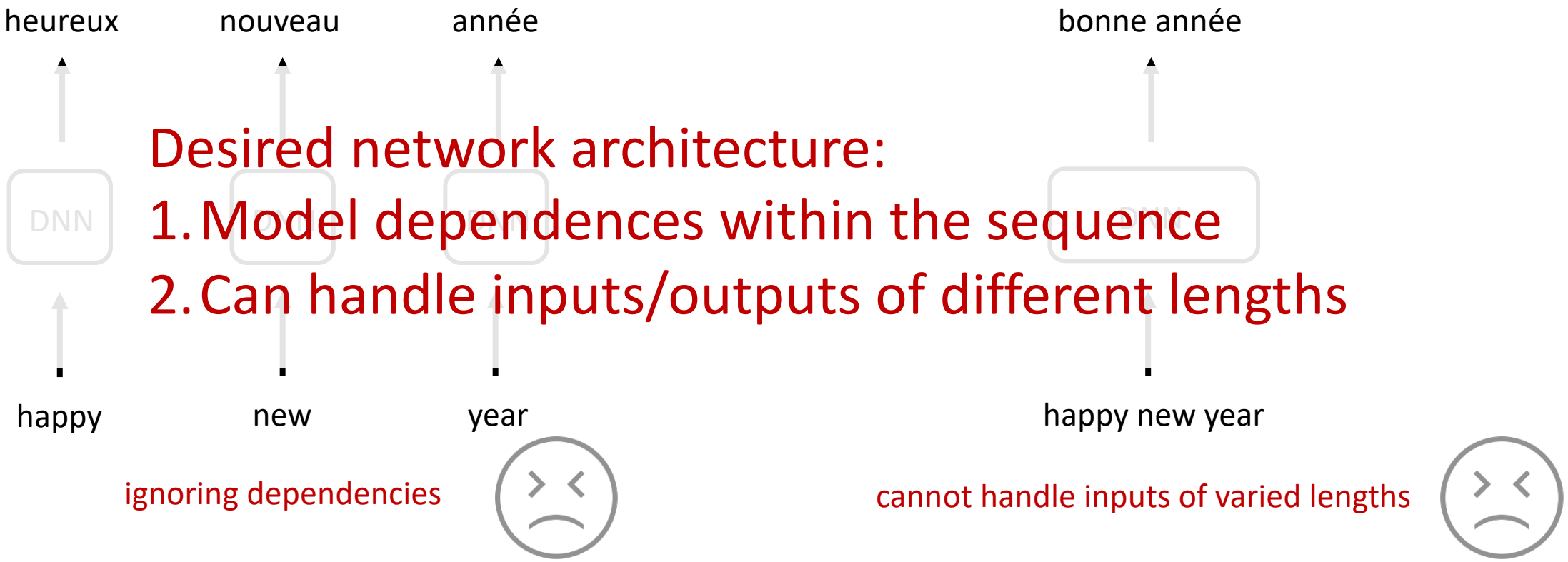
ignoring dependencies



cannot handle inputs of varied lengths

Why Recurrent Neural Networks

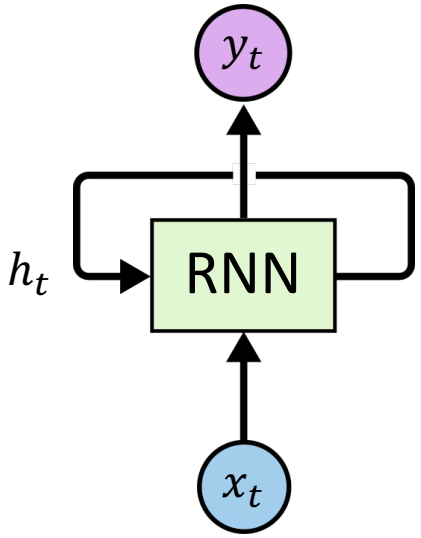
Machine translation as a motivating example:



Desired network architecture:

1. Model dependences within the sequence
2. Can handle inputs/outputs of different lengths

Simple RNNs



$$h_t = \phi_h(W_h h_{t-1} + W_x x_t + b_h)$$
$$y_t = \phi_y(W_y h_t + b_y)$$

ϕ_h: activation function for recurrent state

ϕ_y: activation function for output

Figure adapted from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Simple RNNs

Unrolling the RNN architecture through time:

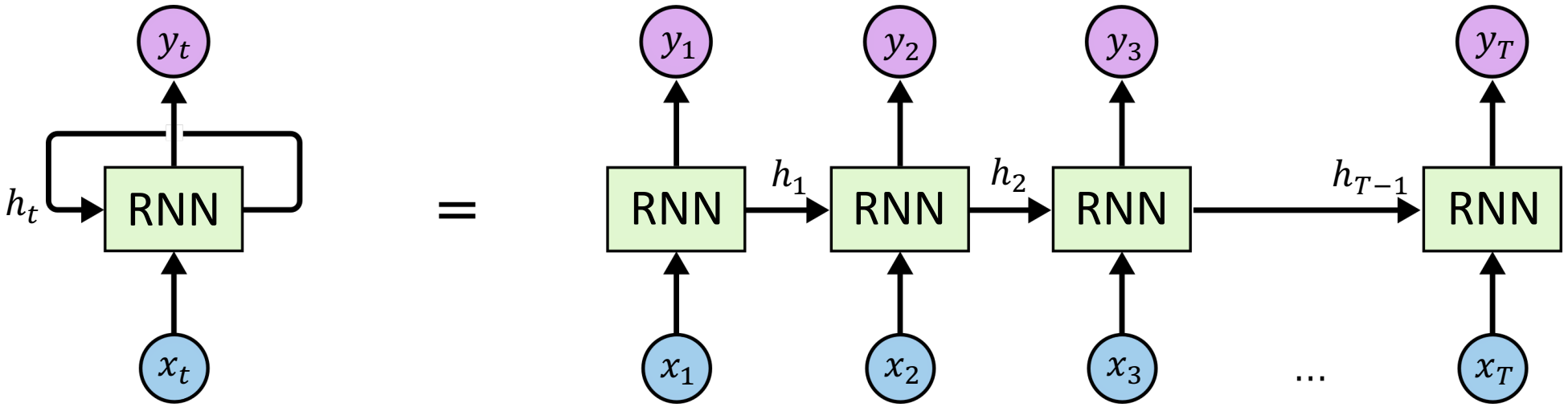
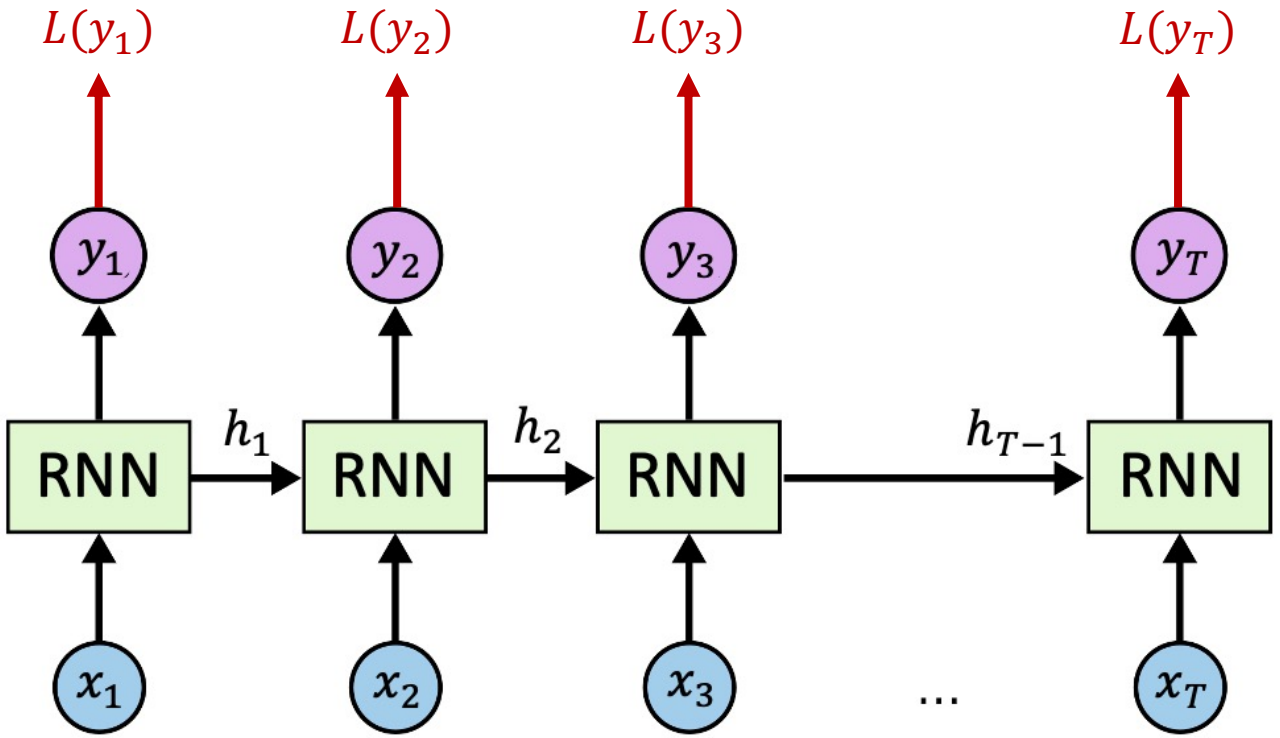


Figure adapted from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Training RNNs

Forward pass: $L_{total}(\theta) = \sum_{t=1}^{T} L(y_t),\ \theta = \{W_h, W_x, W_y, b_h, b_y\}$

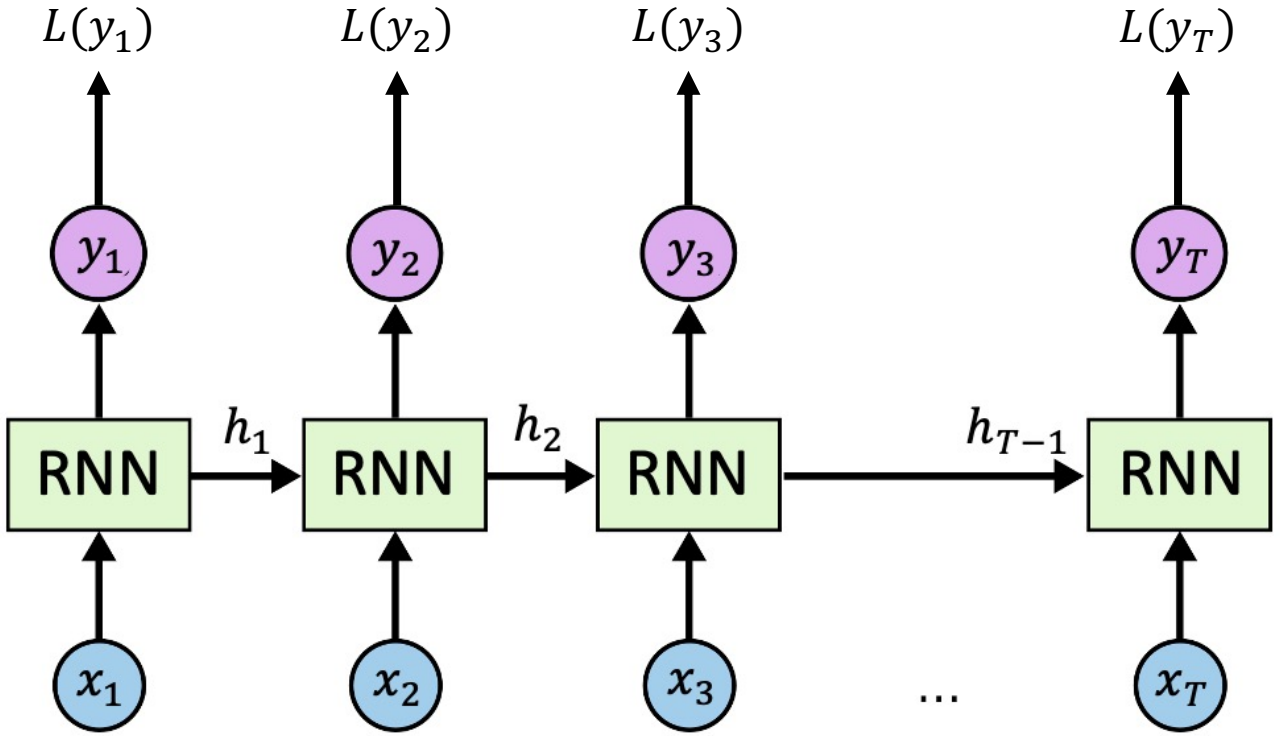


$$h_t = \phi_h(W_h h_{t-1} + W_x x_t + b_h)$$
$$y_t = \phi_y(W_y h_t + b_y)$$

Figure adapted from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Training RNNs

Backward pass: $\frac{d}{d\theta} L_{total}(\theta) = \sum_{t=1}^{T} \frac{d}{d\theta} L(y_t), \theta = \{W_h, W_x, W_y, b_h, b_y\}$



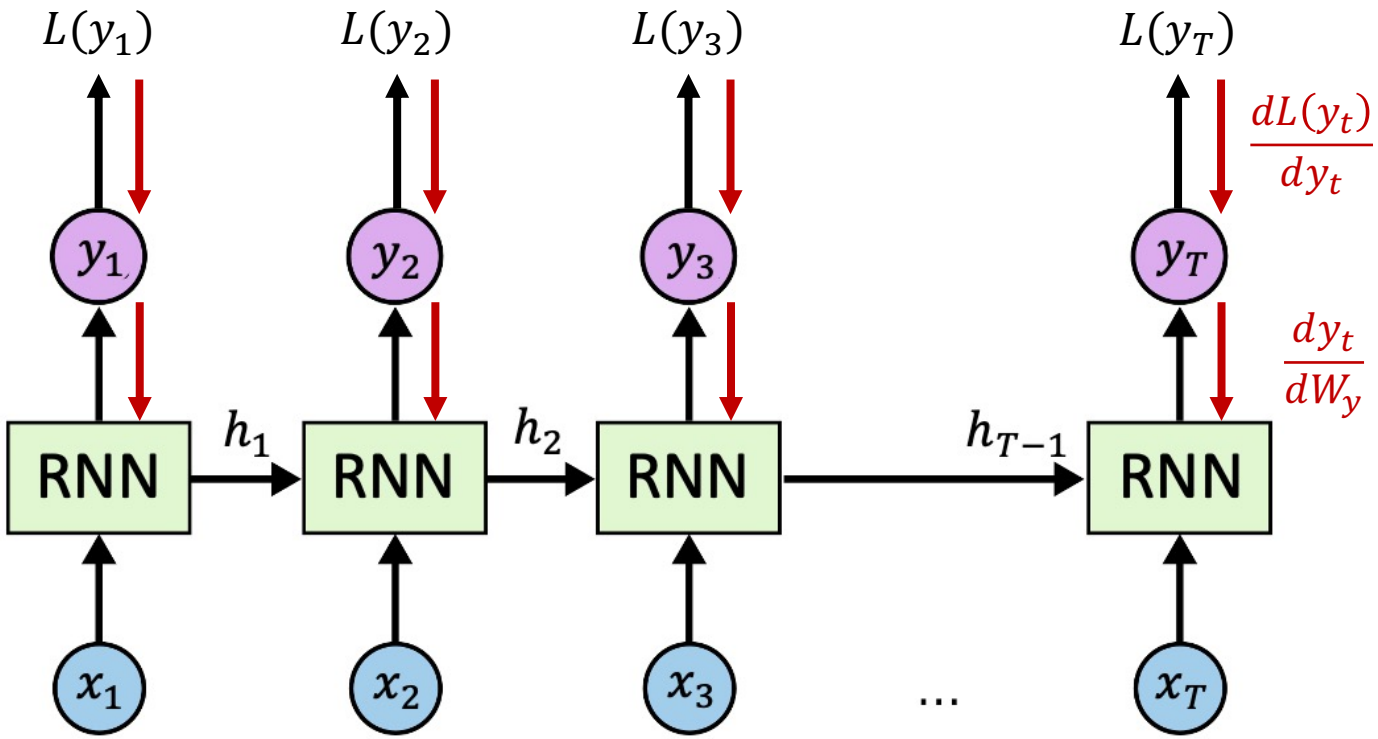
$$h_t = \phi_h(W_h h_{t-1} + W_x x_t + b_h)$$
$$y_t = \phi_y(W_y h_t + b_y)$$

Figure adapted from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Training RNNs

Backward pass: $\frac{d}{dW_y} L_{total} = \sum_{t=1}^{T} \frac{d}{dW_y} L(y_t)$

$\frac{d}{dW_y} L(y_t) = \frac{dL(y_t)}{dy_t} \frac{dy_t}{dW_y}$



$h_t = \phi_h(W_h h_{t-1} + W_x x_t + b_h)$
$y_t = \phi_y(W_y h_t + b_y)$

Figure adapted from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

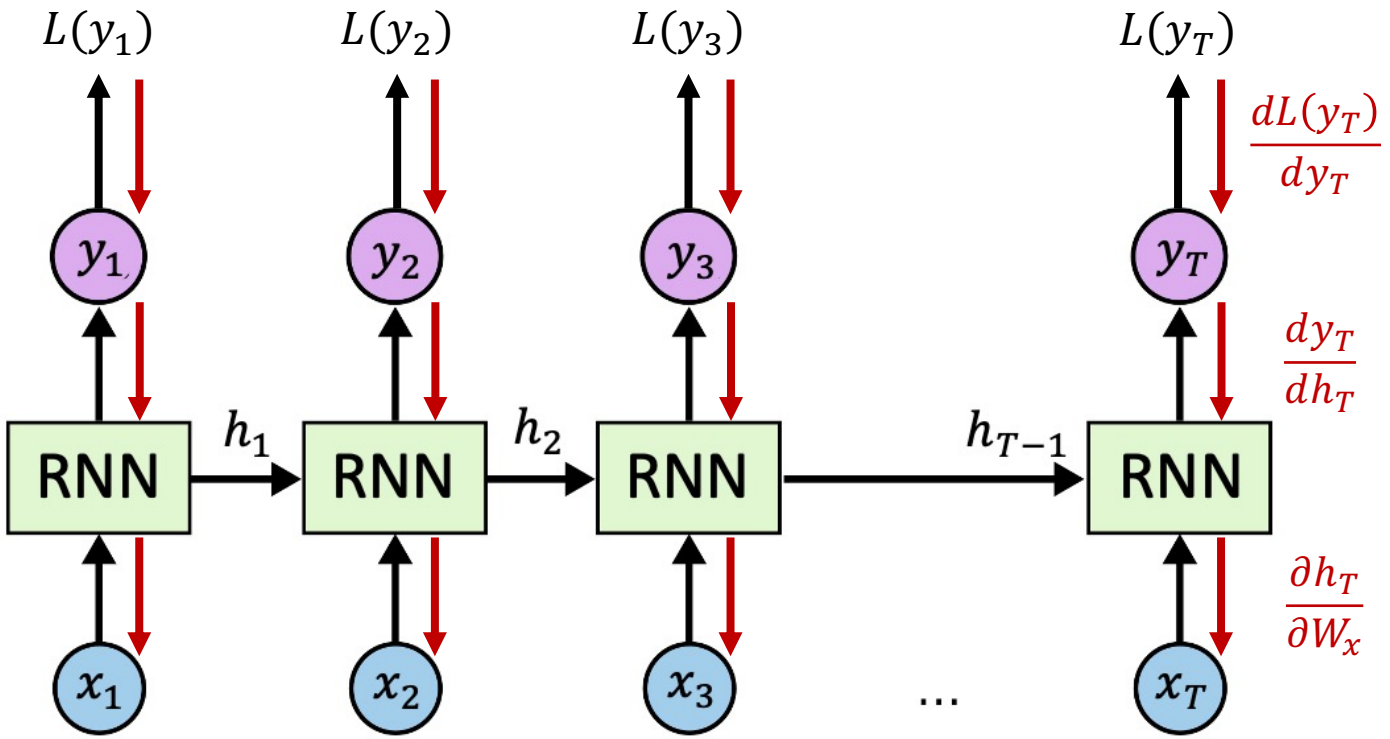
Training RNNs

$$\frac{dL(y_t)}{dW_x} = \frac{dL(y_t)}{dy_t}\frac{dy_t}{dh_t}\frac{dh_t}{dW_x}$$

$$\frac{dh_t}{dW_x} = \frac{\partial h_t}{\partial W_x} + \frac{\partial h_t}{\partial h_{t-1}}\frac{dh_{t-1}}{dW_x}$$

total gradient partial gradient

Backward pass: $\frac{d}{dW_x}L_{total} = \sum_{t=1}^{T}\frac{d}{dW_x}L(y_t)$



$$h_t = \phi_h(W_h h_{t-1} + W_x x_t + b_h)$$
$$y_t = \phi_y(W_y h_t + b_y)$$

Figure adapted from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

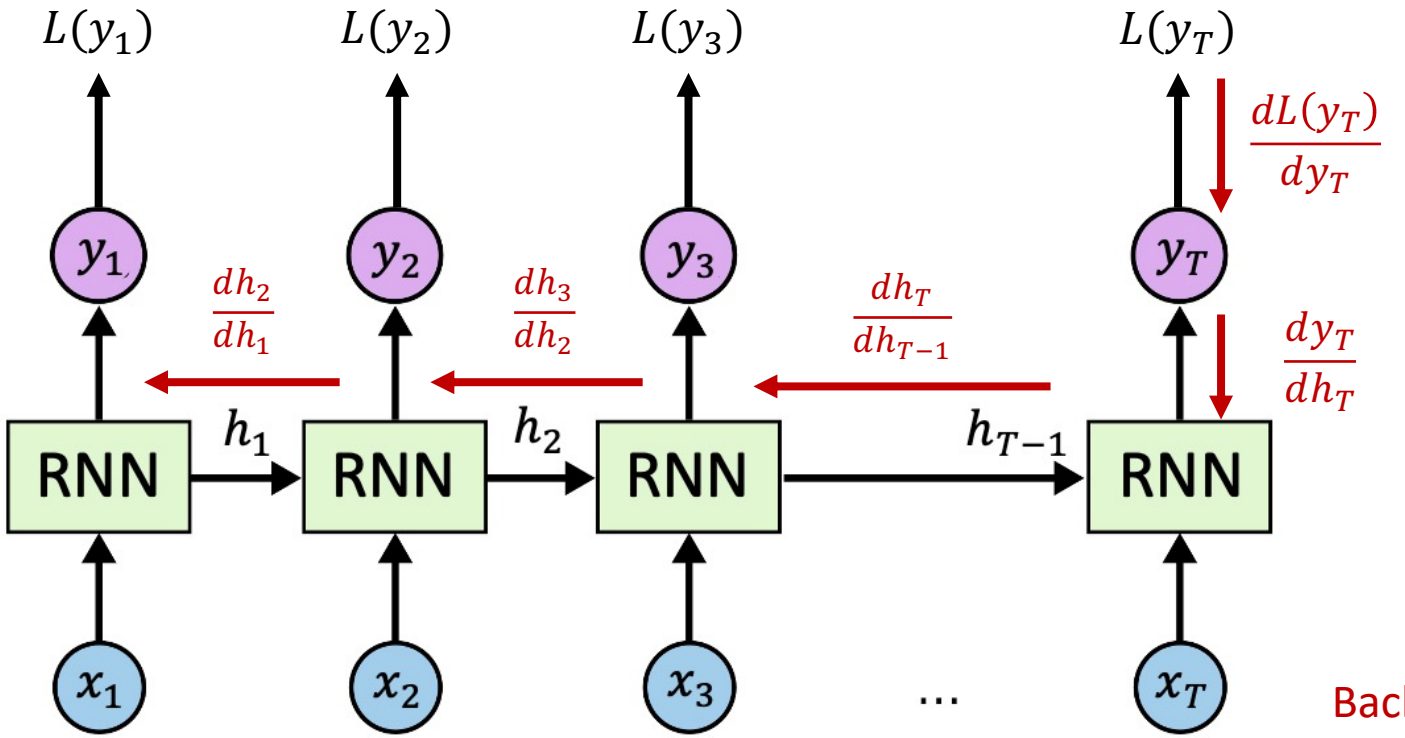
Training RNNs

$$\frac{dL(y_t)}{dW_h} = \frac{dL(y_t)}{dy_t}\frac{dy_t}{dh_t}\frac{dh_t}{dW_h}$$

$$\frac{dh_t}{dW_h} = \frac{\partial h_t}{\partial W_h} + \frac{\partial h_t}{\partial h_{t-1}}\frac{dh_{t-1}}{dW_h}$$

total gradient — partial gradient

Backward pass: $\frac{d}{dW_h}L_{total} = \sum_{t=1}^{T}\frac{d}{dW_h}L(y_t)$



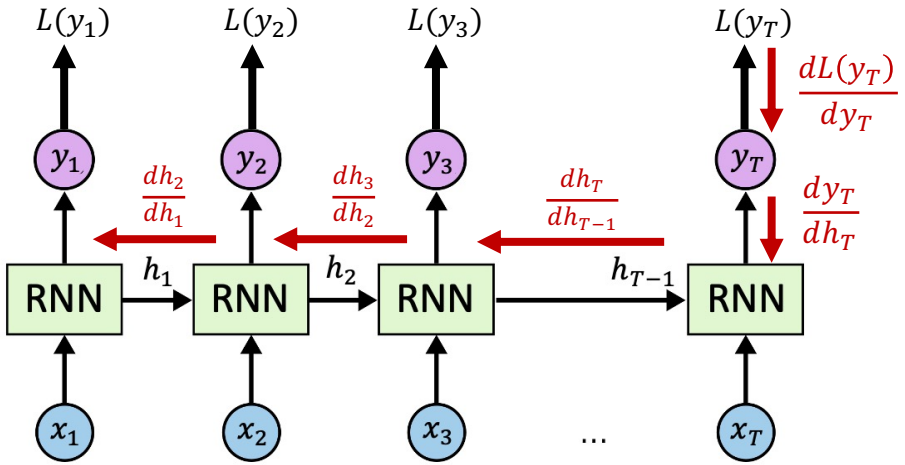
$$h_t = \phi_h(W_h h_{t-1} + W_x x_t + b_h)$$
$$y_t = \phi_y(W_y h_t + b_y)$$

Back-propagation through time (BPTT)

Figure adapted from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Issues of simple RNNs

Consider gradient of loss w.r.t. W_h:



$$\frac{dL(y_t)}{dW_h} = \frac{dL(y_t)}{dy_t}\frac{dy_t}{dh_t}\frac{dh_t}{dW_h}$$

$$\frac{dh_t}{dW_h} = \frac{\partial h_t}{\partial W_h} + \frac{dh_t}{dh_{t-1}}\frac{dh_{t-1}}{dW_h}$$

$$= \frac{\partial h_t}{\partial W_h} + \frac{dh_t}{dh_{t-1}}\left(\frac{\partial h_{t-1}}{\partial W_h} + \frac{dh_{t-1}}{dh_{t-2}}\frac{dh_{t-2}}{dW_h}\right) = \ \ldots$$

$$\Rightarrow \frac{dh_t}{dW_h} = \sum_{\tau=1}^{t}\left(\prod_{l=\tau}^{t-1}\frac{dh_{l+1}}{dh_l}\right)\frac{\partial h_\tau}{\partial W_h},$$

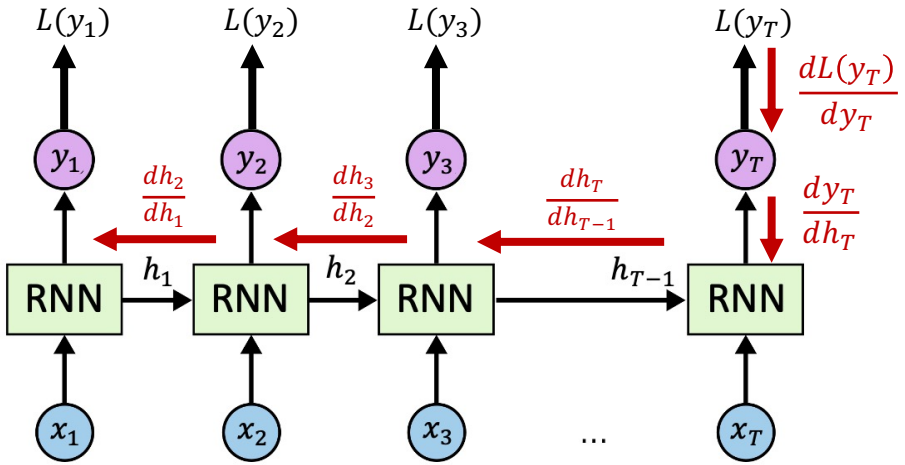
$$\frac{dh_{l+1}}{dh_l}^{\top} = \phi_h'(W_h h_l + W_x x_{l+1} + b_h) \odot W_h$$

contain products of W_h and ϕ_h' for $t - \tau$ times

Depending on W_h and the non-linearity ϕ_h, when $t \to \infty$, $\prod_{l=1}^{t-1}\frac{dh_{l+1}}{dh_l}$ can vanish or explode!

Issues of simple RNNs

Consider gradient of loss w.r.t. W_h:

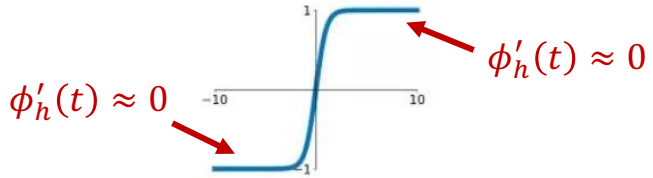


Identity mapping: $\phi_h(t) = t$

$$\prod_{l=1}^{t-1} \frac{dh_{l+1}}{dh_l} = (W_h^{t-1})^\top$$

(explode or vanish depending on the largest singular value)

Tanh mapping: $\phi_h(t) = \tanh(t)$



$$\Rightarrow \frac{dh_t}{dW_h} = \sum_{\tau=1}^{t} \left(\prod_{l=\tau}^{t-1} \frac{dh_{l+1}}{dh_l} \right) \frac{\partial h_\tau}{\partial W_h},$$

$$\frac{dh_{l+1}}{dh_l}^\top = \phi_h'(W_h h_l + W_x x_{l+1} + b_h) \odot W_h$$

contain products of W_h and ϕ_h' for $t - \tau$ times

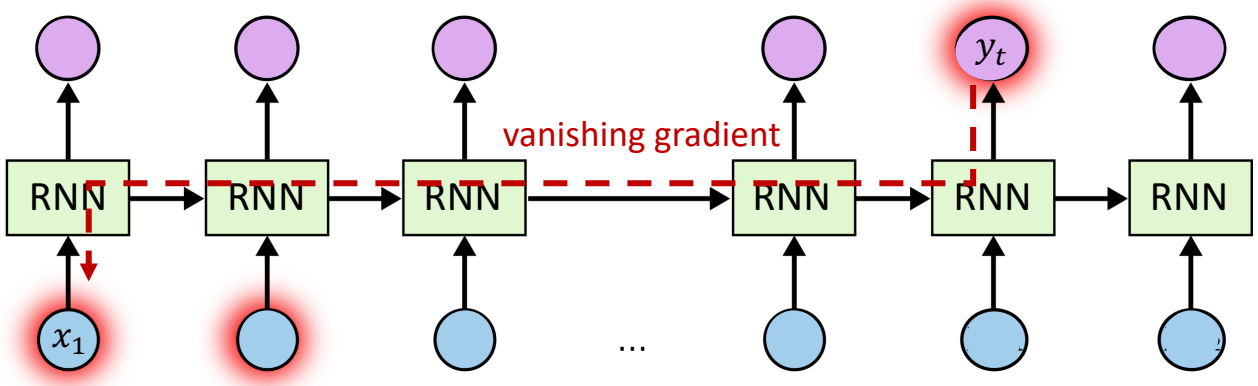
Depending on the non-linearity ϕ_h, when $t \to \infty$, $\prod_{l=1}^{t-1} \frac{dh_{l+1}}{dh_l}$ can vanish or explode!

Issues of simple RNNs

- Consider gradient of loss w.r.t. W_h:

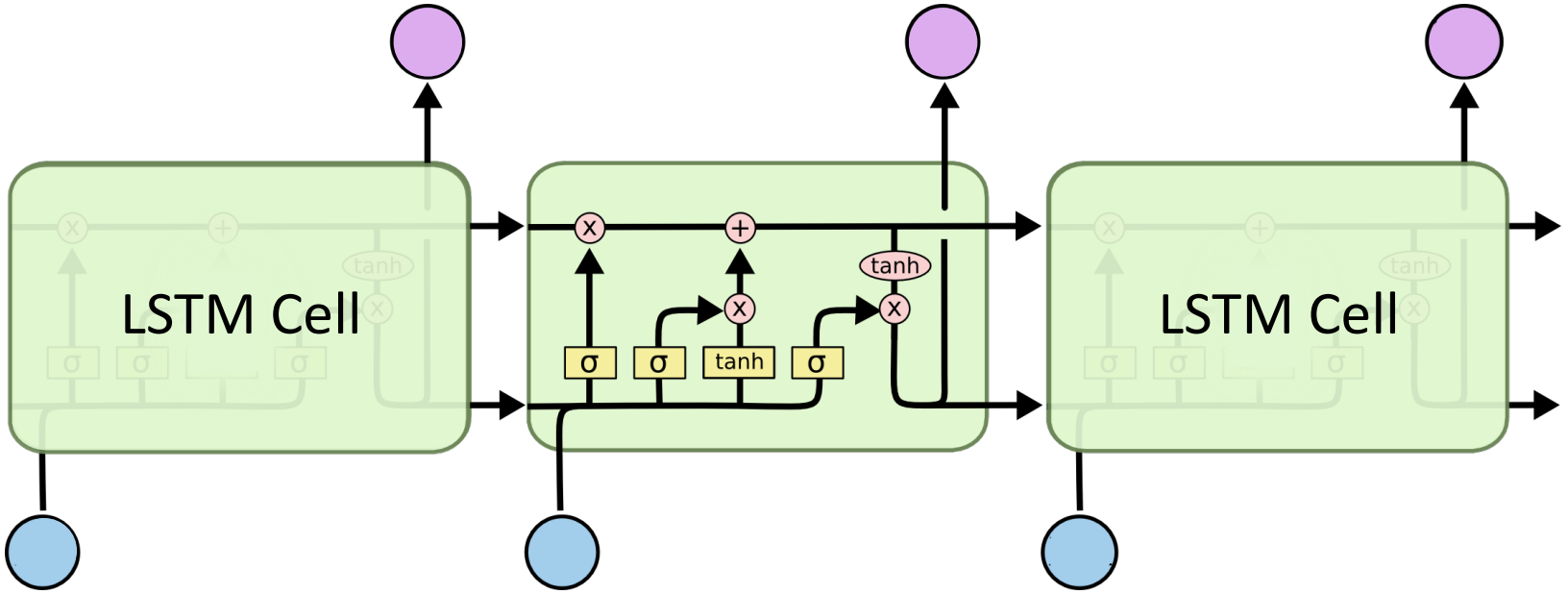
$$\Rightarrow \frac{dh_t}{dW_h} = \sum_{\tau=1}^{t} \left(\prod_{l=\tau}^{t-1} \frac{dh_{l+1}}{dh_l} \right) \frac{\partial h_\tau}{\partial W_h}, \qquad \frac{dh_{l+1}}{dh_l}^{\top} = \phi_h'(W_h h_l + W_x x_{l+1} + b_h) \odot W_h$$

can vanish or explode (especially when $t \gg \tau$)



Dependency of y_t on x_1 gets harder to learn as t increases

Long Short-Term Memory (LSTM)

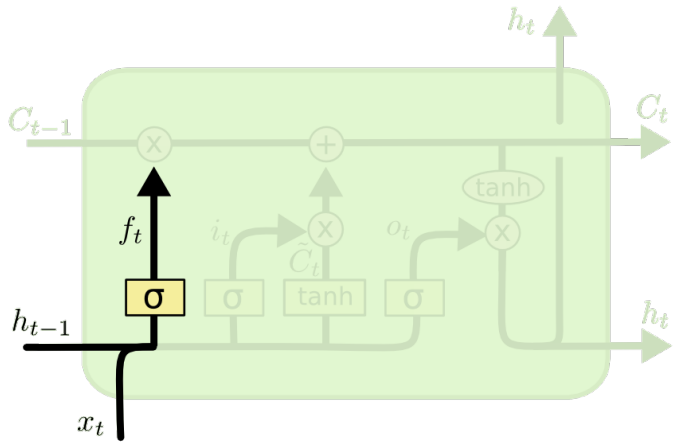


Key ideas of LSTM:

- Introduce cell state C_t
- Gating mechanisms to control cell state updates and output values

Hochreiter and Schmidhuber (1997). Long Short-Term Memory. Neuro Computation.
Figure adapted from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short-Term Memory (LSTM)



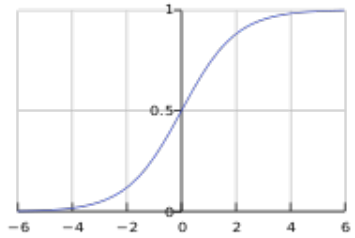
Forget gate f_t:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

concatenate

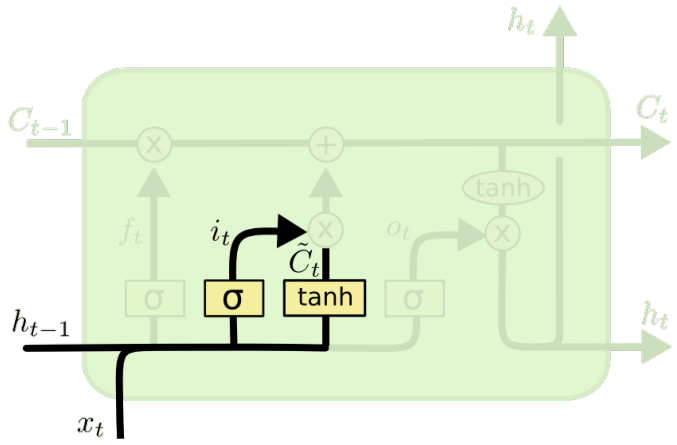
sigmoid activation function

$$\sigma(t) = \frac{1}{1 + \exp(-t)}$$



Hochreiter and Schmidhuber (1997). Long Short-Term Memory. Neuro Computation.
Figure adapted from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short-Term Memory (LSTM)



Input gate: i_t

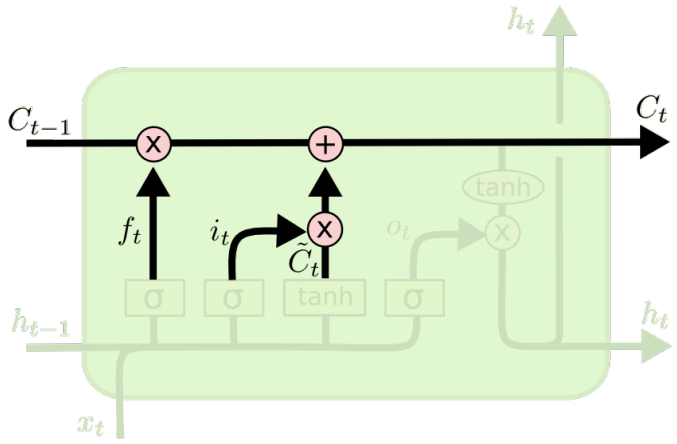
Candidate cell state update: $\tilde{c}_t$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{c}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

Hochreiter and Schmidhuber (1997). Long Short-Term Memory. Neuro Computation.

Figure adapted from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short-Term Memory (LSTM)



Cell state update:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

$f_t[d] \in (0,1)$

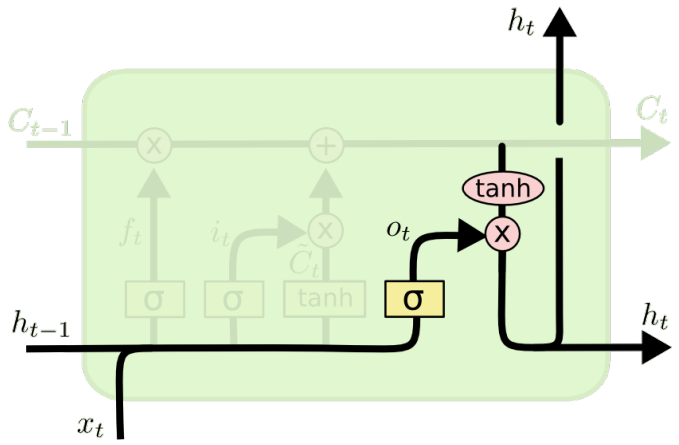
$f_t[d] \to 0$: forget previous state
$f_t[d] \to 1$: maintain previous state

$i_t[d] \in (0,1)$

$i_t[d] \to 0$: discard candidate cell state update
$i_t[d] \to 1$: enable cell state update

Hochreiter and Schmidhuber (1997). Long Short-Term Memory. Neuro Computation.
Figure adapted from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short-Term Memory (LSTM)



Update the hidden state h_t
with output gating o_t

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t \odot \tanh(c_t)$$

$o_t[d] \in (0,1)$

$o_t[d] \to 0$: zero output
$o_t[d] \to 1$: output cell state (squashed in $(-1, 1)$)

Prediction of y_t can proceed in a similar way as in simple RNNs:

$$y_t = \phi_y(W_y h_t + b_y)$$

Hochreiter and Schmidhuber (1997). Long Short-Term Memory. Neuro Computation.
Figure adapted from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short-Term Memory (LSTM)

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{c}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t \odot \tanh(c_t)$$
$$y_t = \phi_y(W_y h_t + b_y)$$

BPTT in LSTMs:

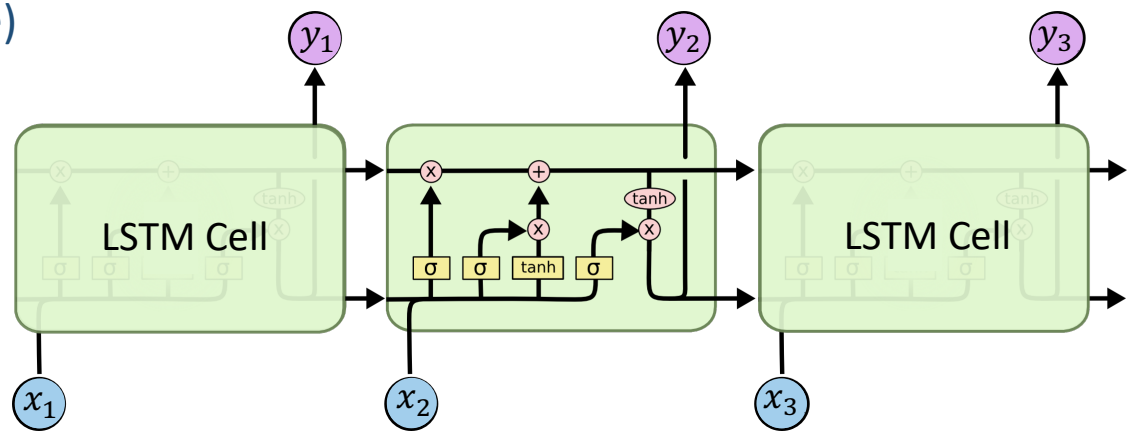
Requires computing $\prod_{l=1}^{t-1} \frac{dc_{l+1}}{dc_l} = \prod_{l=1}^{t-1}(f_{l+1} + o_l \odot \frac{dtanh(c_l)}{dc_l} \odot \frac{dc_{l+1}}{dh_l})$

How to derive $\frac{dc_t}{dc_{t-1}}$: Notice c_t depends on c_{t-1} in 4 paths:

direct dependence

$$\boldsymbol{c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t}$$

Depends on $h_{t-1} = o_{t-1} \odot c_{t-1}$ (indirect dependence)



Hochreiter and Schmidhuber (1997). Long Short-Term Memory. Neuro Computation.
Figure adapted from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short-Term Memory (LSTM)

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{c}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t \odot \tanh(c_t)$$
$$y_t = \phi_y(W_y h_t + b_y)$$

BPTT in LSTMs:

Requires computing $\prod_{l=1}^{t-1} \frac{dc_{l+1}}{dc_l} = \prod_{l=1}^{t-1}(f_{l+1} + o_l \odot \frac{dtanh(c_l)}{dc_l} \odot \frac{dc_{l+1}}{dh_l})$

Alleviating gradient explosion:

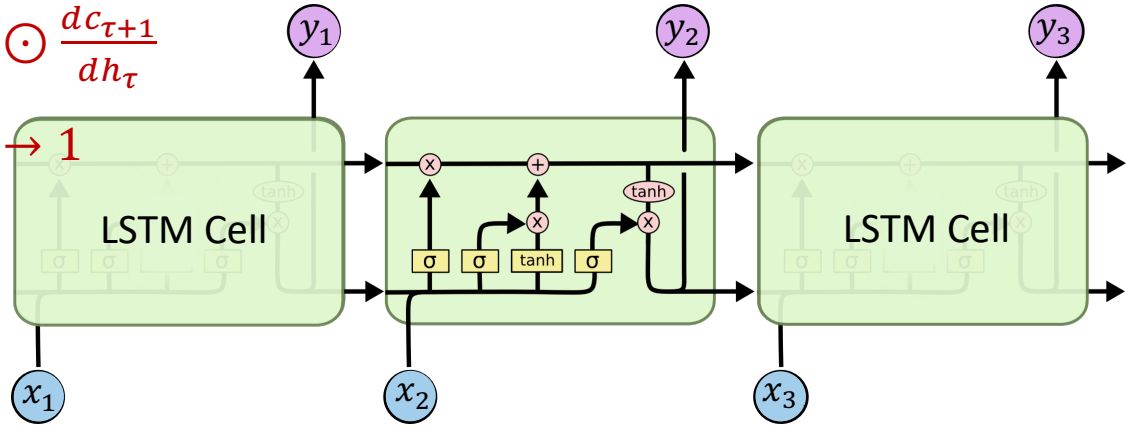
The gradient contains terms proportional to $f_{i+1} \prod_{l=1}^{i-1} o_l \odot \frac{dc_{l+1}}{dh_l}$

≈ 0 when $f_{i+1} \approx 0$

Alleviating gradient vanishing:

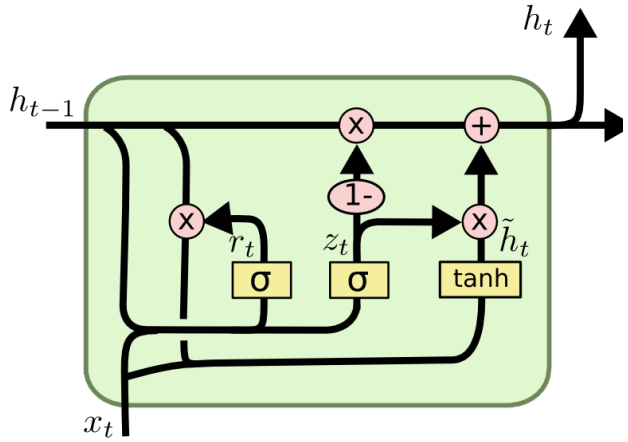
The gradient contains terms proportional to $\prod_{l=\tau+1}^{i} f_l \odot o_\tau \odot \frac{dc_{\tau+1}}{dh_\tau}$

$\approx o_\tau \odot \frac{dc_{\tau+1}}{dh_\tau}$ when $f_l \to 1$ for $l = \tau + 1, \ldots, i$



Hochreiter and Schmidhuber (1997). Long Short-Term Memory. Neuro Computation.
Figure adapted from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Gated Recurrent Unit (GRU)



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z)$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r)$$

$$\tilde{h}_t = \tanh(W_h \cdot [r_t \odot h_{t-1}, x_t] + b_h)$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

Cho et al. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. EMNLP 2014
Figure adapted from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

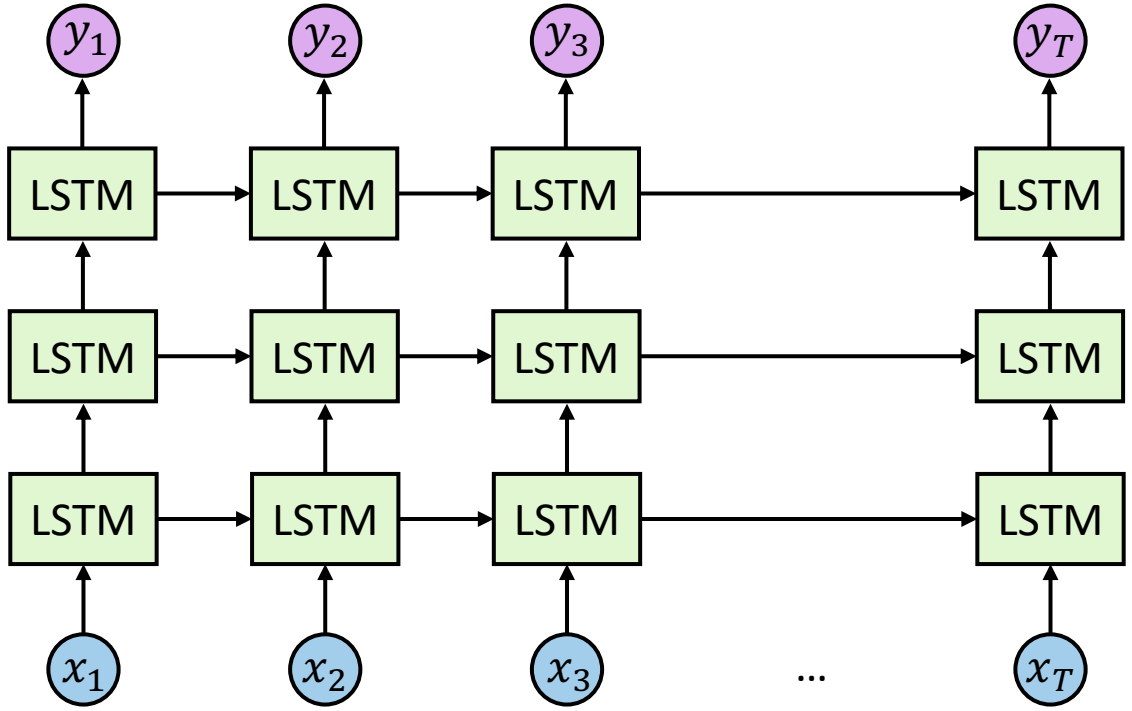
LSTM vs GRU

- Other gated RNN variants exists, but LSTM and GRU are the most widely-used
- GRU is quicker to compute and has fewer parameters
- No conclusive evidence for LSTM > GRU or vice versa
- LSTM is a good default choice (especially if your data has particularly long dependencies, or you have lots of training data)
- Switch to GRU if you want more efficient compute & less overfitting

Stacking LSTMs

Stacking multiple LSTM layers:

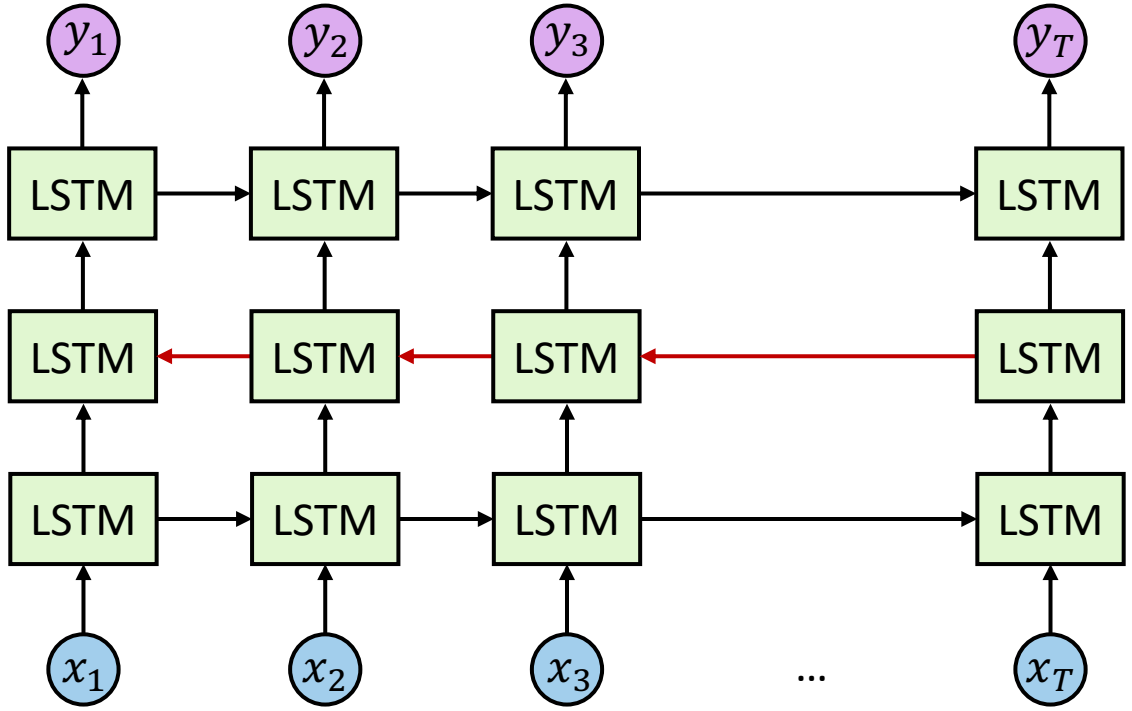
- Hidden states of the previous LSTM later as inputs to the next LSTM layer;
- No need to wait for previous LSTM layer to finish forward pass;



Bidirectional LSTMs

Bidirectional LSTM:

- Stacking some LSTM layers;
- For some LSTM layers, the forward pass is reversed from time $\mathrm{t} = T$ to $\mathrm{t} = 1$;
- If two consecutive LSTM layers are of reversed time ordering, then the top layer needs to wait for the bottom one to finish forward pass.



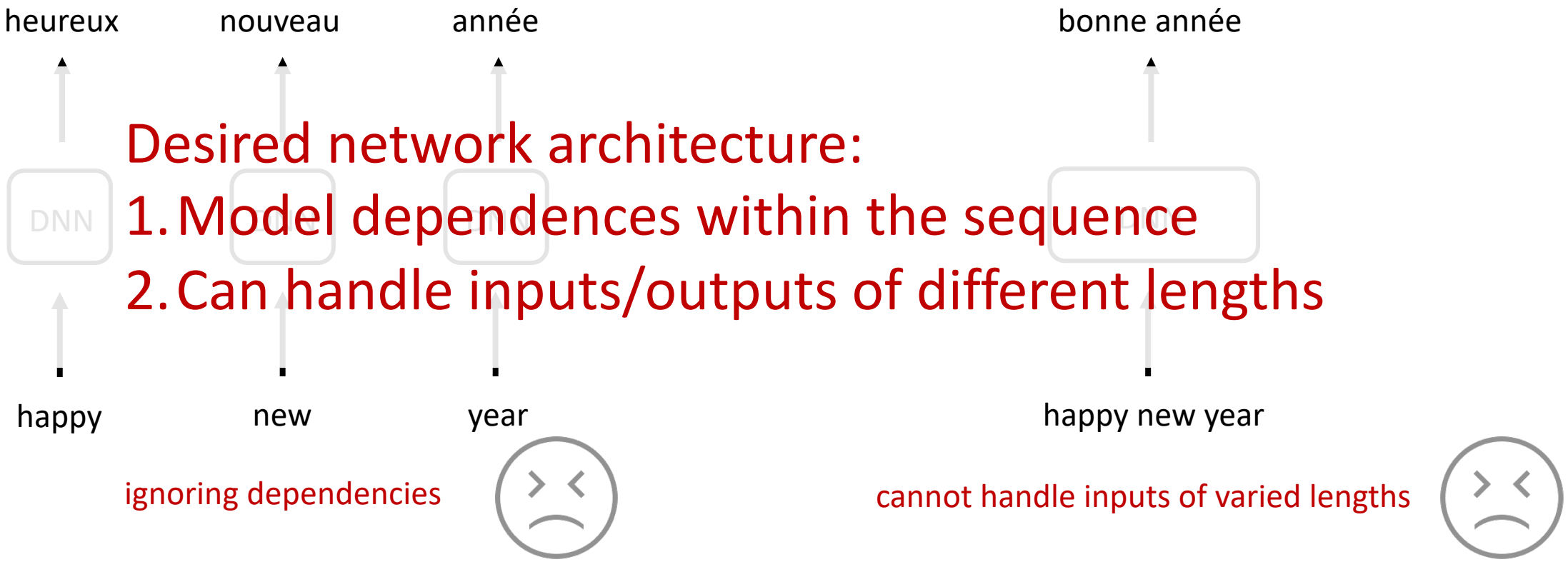
Recurrent Neural Networks

Applications

Yingzhen Li (yingzhen.li@imperial.ac.uk)

Why Recurrent Neural Networks

Machine translation as a motivating example:



Desired network architecture:

1. Model dependences within the sequence
2. Can handle inputs/outputs of different lengths

Sequence-to-Sequence Model

Machine translation (e.g. EN to FR):

- Data sequence: $x_{1:T} = (x_1, \dots, x_T)$
  - x_t: the t^{th} word in the English sentence
- Label sequence: $y_{1:L} = (y_1, \dots, y_L)$
  - y_l: the l^{th} word in the French sentence
- Goal: learn the conditional distribution

$$p_\theta(y_{1:L}|x_{1:T})$$

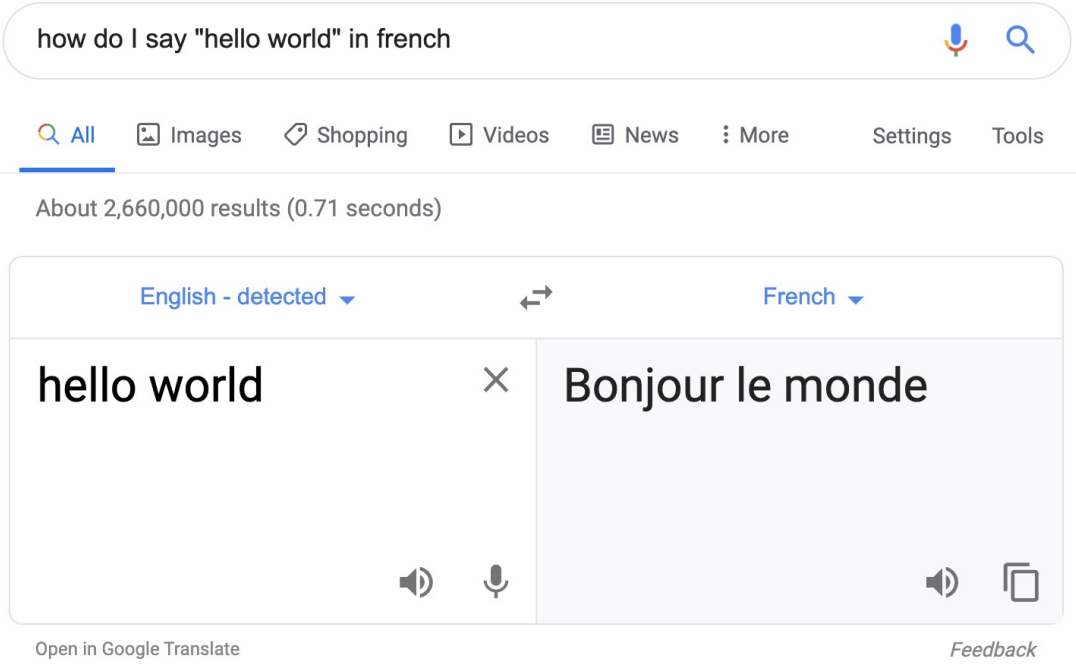
for x, y sequences of any length

- Idea: define an auto-regressive model

$$p_\theta(y_{1:L}|x_{1:T}) = \prod_{l=1}^{L} p_\theta(y_l|y_{<l}, v),\ v = enc(x_{1:T})$$

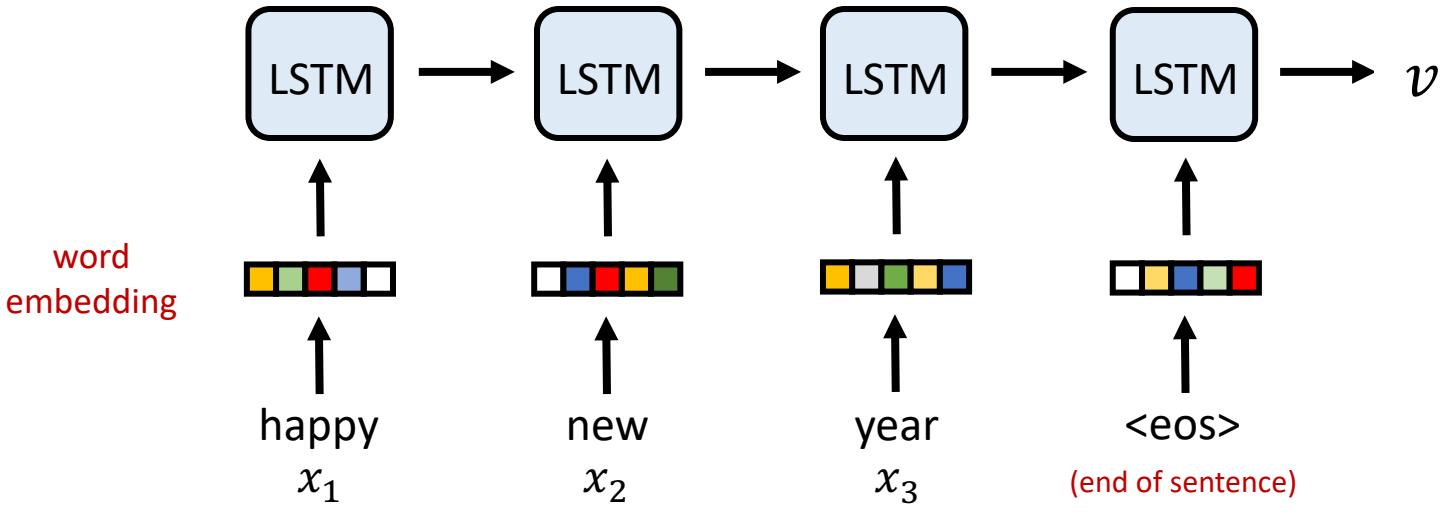
Sequence decoder

Sequence encoder



Sequence-to-Sequence Model

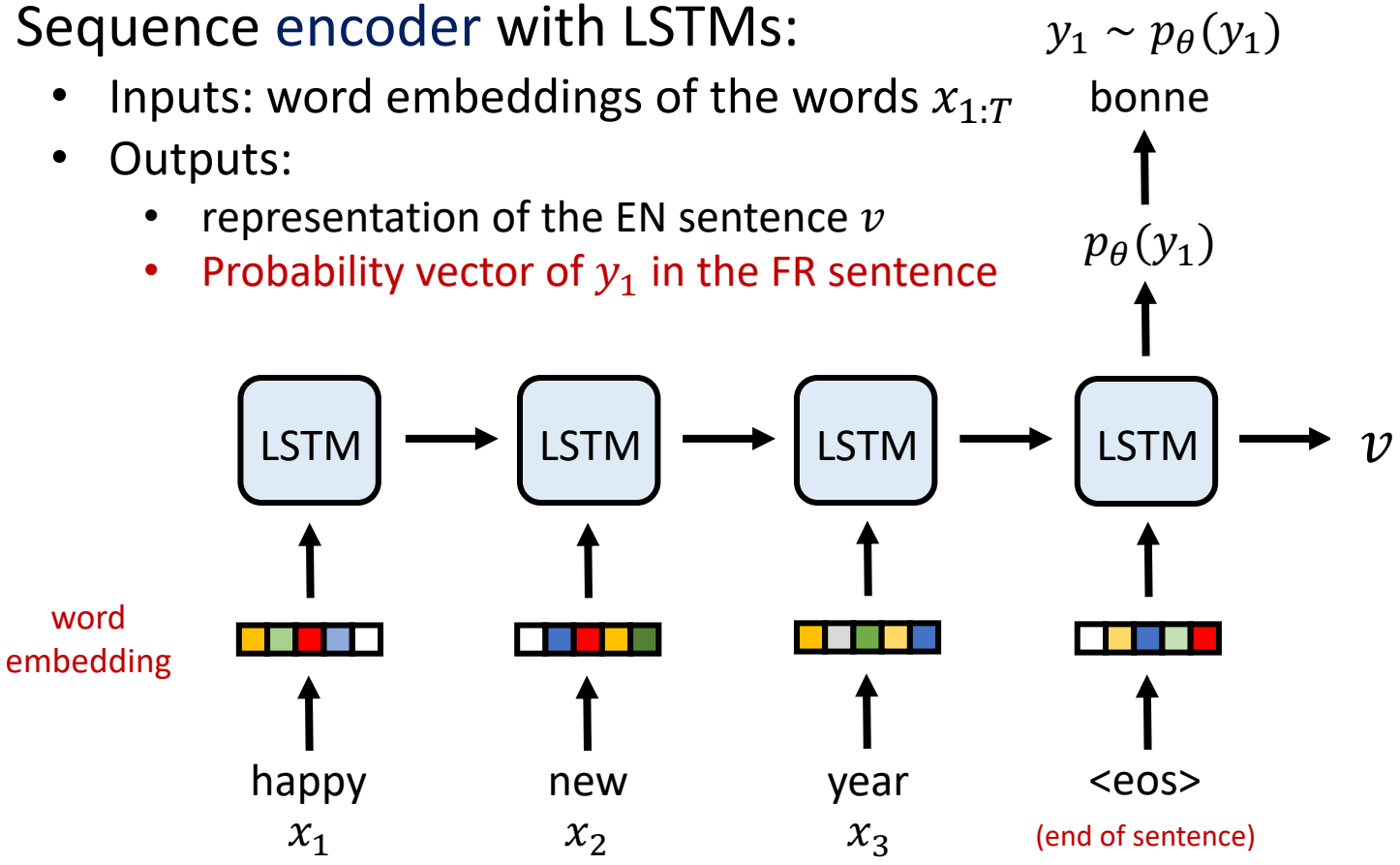
- Sequence encoder with LSTMs:
  - Inputs: word embeddings of the words $x_{1:T}$
  - Outputs:
    - representation of the EN sentence v



Sutskever et al. Sequence to Sequence Learning with Neural Networks. NeurIPS 2014

Sequence-to-Sequence Model

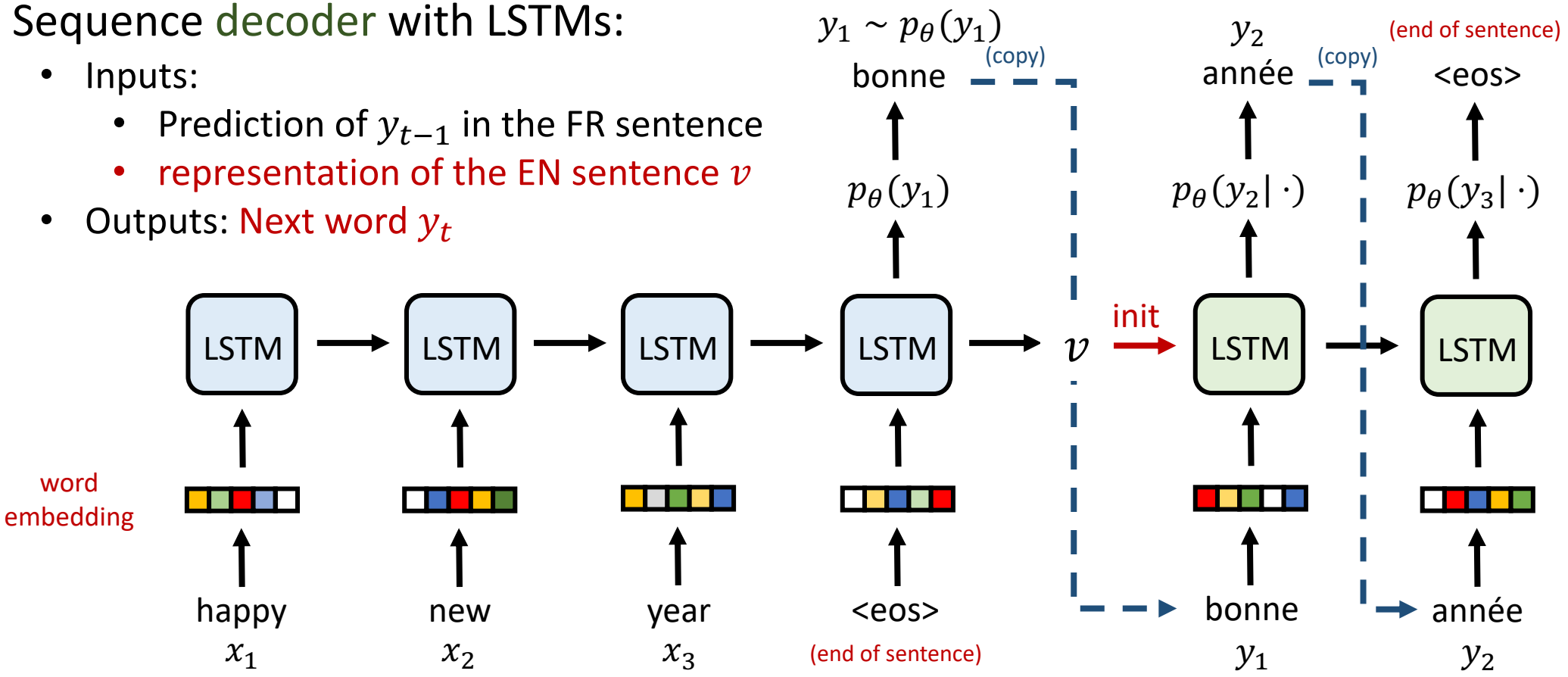
- Sequence encoder with LSTMs:
  - Inputs: word embeddings of the words $x_{1:T}$
  - Outputs:
    - representation of the EN sentence v
    - Probability vector of y_1 in the FR sentence



Sutskever et al. Sequence to Sequence Learning with Neural Networks. NeurIPS 2014

Sequence-to-Sequence Model

- Sequence decoder with LSTMs:
  - Inputs:
    - Prediction of y_{t-1} in the FR sentence
    - representation of the EN sentence v
  - Outputs: Next word y_t



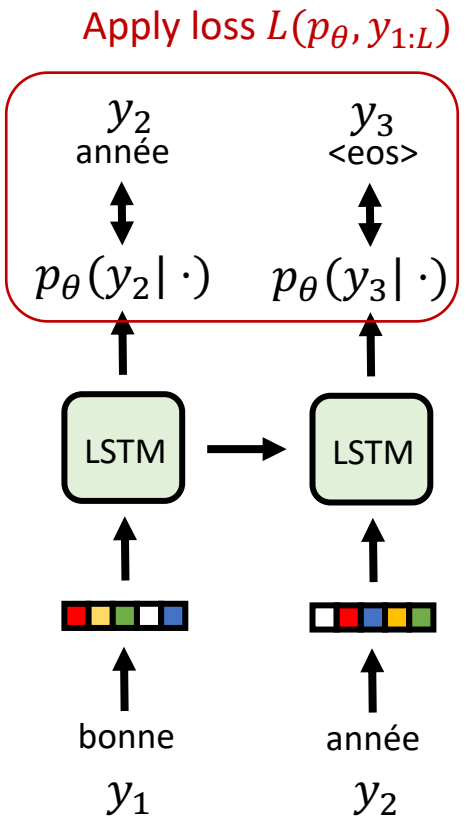
Sutskever et al. Sequence to Sequence Learning with Neural Networks. NeurIPS 2014

Sequence-to-Sequence Model

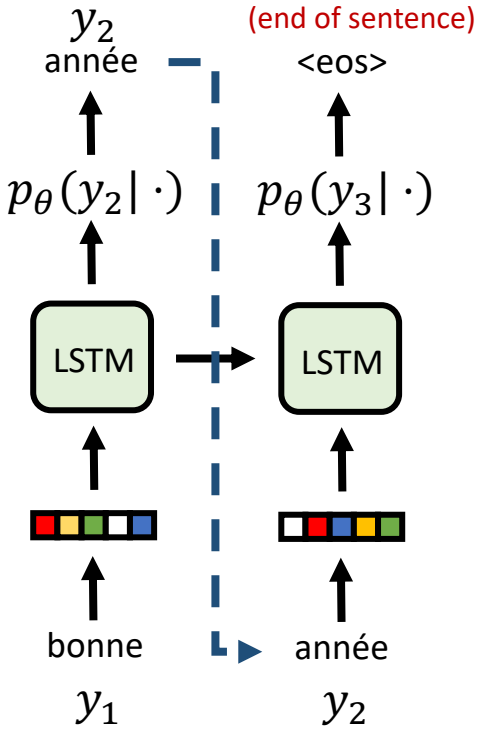
- Sequence decoder inputs during training/test:

Training:
Data: $(x_{1:T}, y_{1:L})$
Inputs: the y_l word in the provided output supervision sequence

MLE training require computing $p_\theta(y_{1:L}|x_{1:T})$ using data



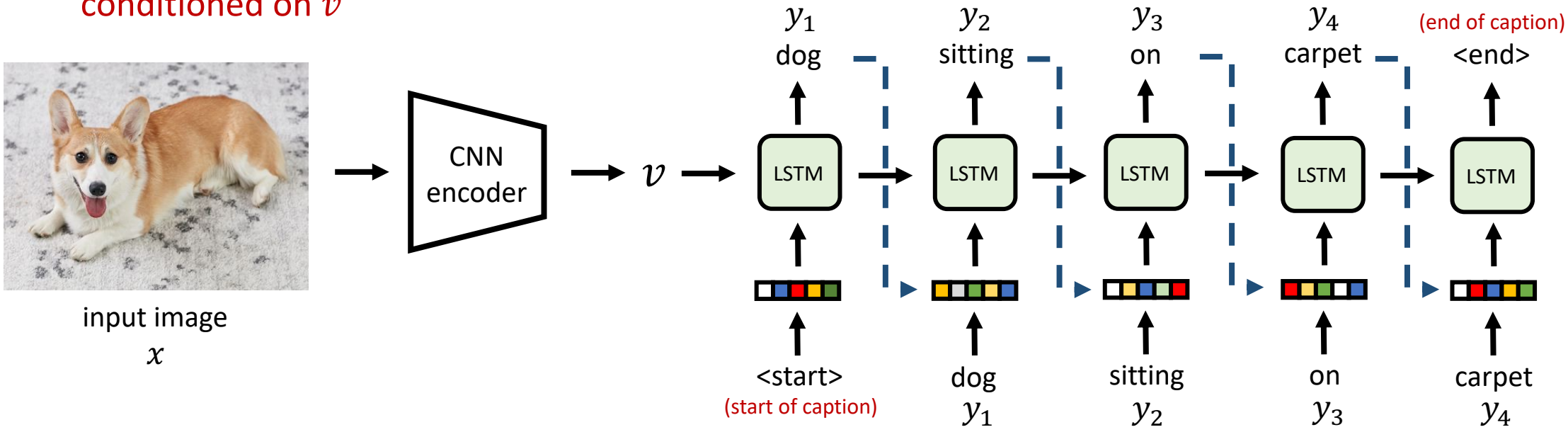
Test:
Data: $x_{1:T}$
Inputs: the y_l word predicted from the last decoding step



Sutskever et al. Sequence to Sequence Learning with Neural Networks. NeurIPS 2014

Image Captioning

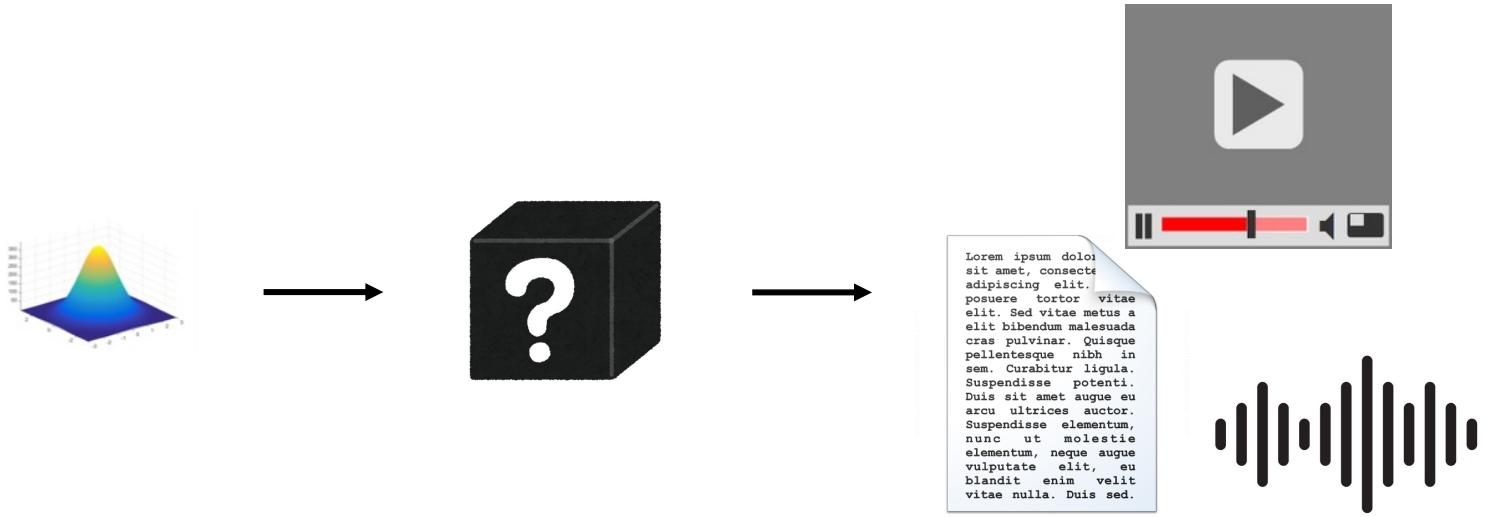
- Image captioning with CNN encoder + LSTM decoder:
  - CNN encoder extract representation v of image x
  - LSTM decoder generate caption conditioned on v



Vinyals et al. Show and tell: A neural image caption generator. CVPR 2015

Sequence generation models

- Latent variable model for sequence generation:



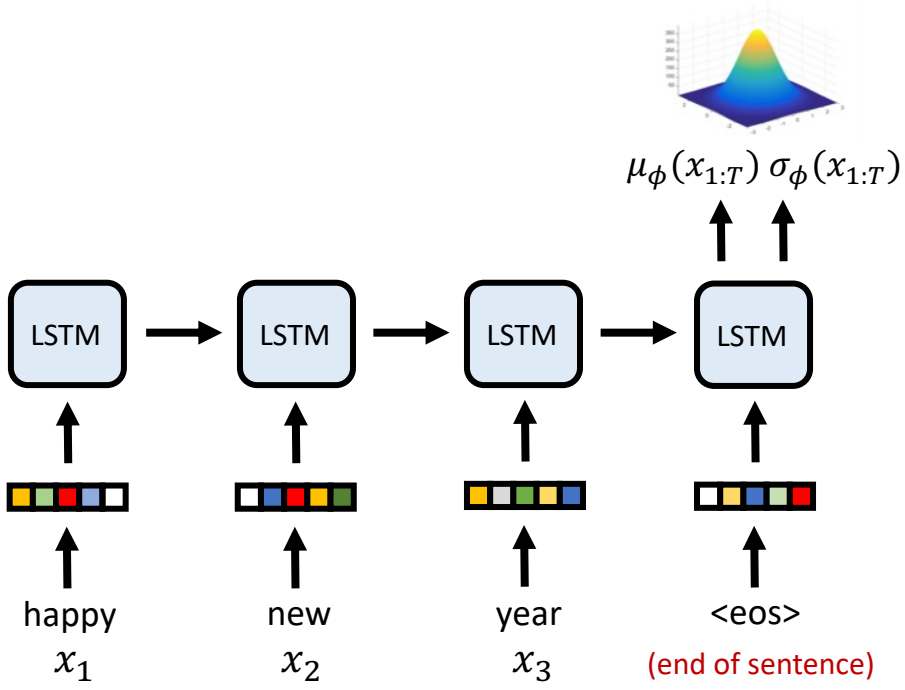
$$z \sim p(z), \;\; x_{1:T} \sim p_\theta(x_{1:T}|z)$$
$$p_\theta(x_{1:T}) = \int p_\theta(x_{1:T}|z)p(z)dz$$

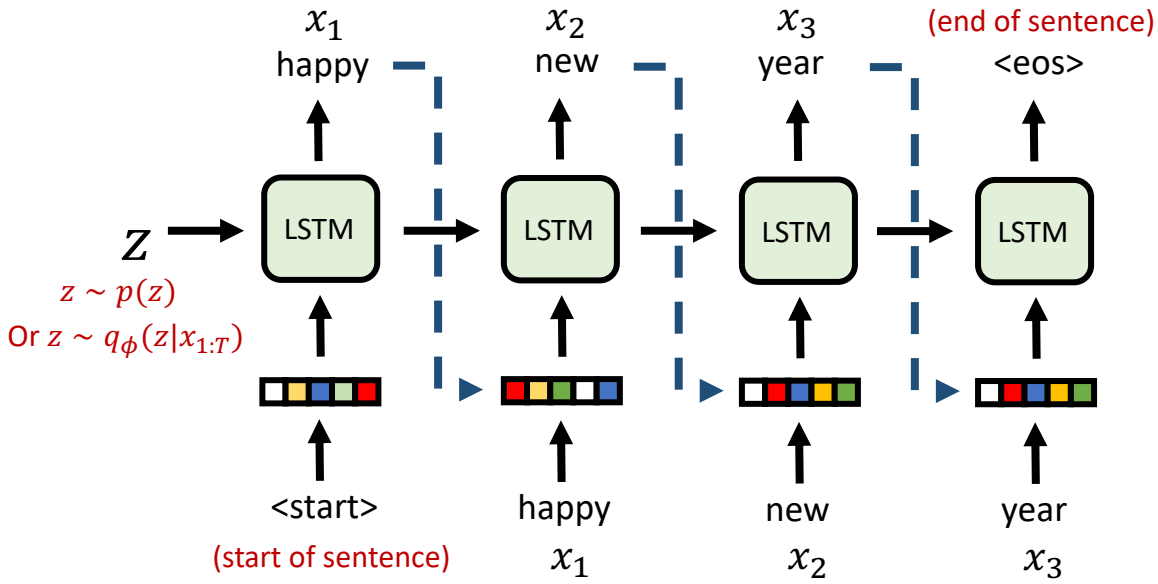
Sequence generation models

- Sequence VAE for language modelling:

Encoder: $q_\phi(z|x_{1:T}) = N(z; \mu_\phi(x_{1:T}), diag(\sigma_\phi^2(x_{1:T})))$

Generator: $p_\theta(x_{1:T}|z) = \prod_{t=1}^{T} p_\theta(x_t|x_{<t}, z)$





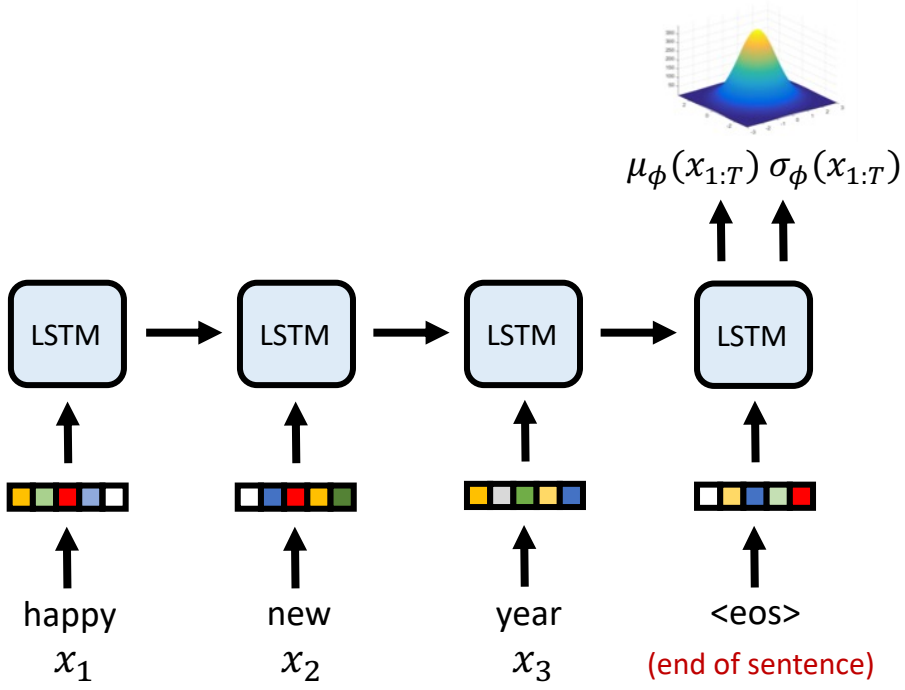
Bowman et al. Generating Sentences from a Continuous Space. CoNLL 2016

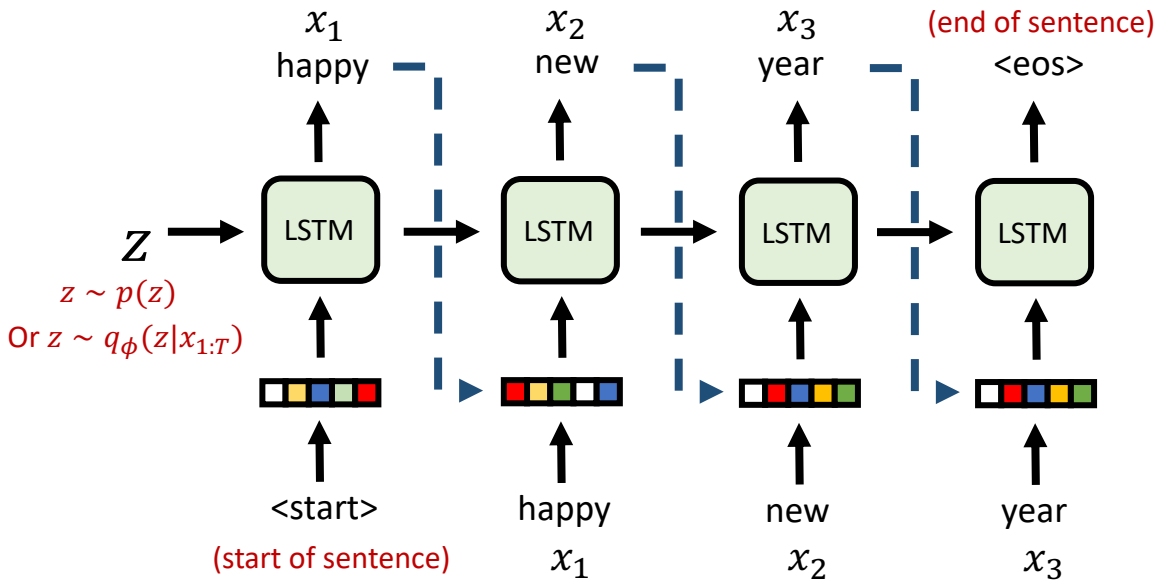
Sequence generation models

- Sequence VAE for language modelling:

$$= \sum_{t=1}^{T} \log p_\theta(x_t | x_{<t}, z)$$

$$L(\theta, \phi) = E_{p_{data}(x_{1:T})}[E_{q_\phi(z|x_{1:T})}[\log p_\theta(x_{1:T}|z)] - \beta\, KL[q_\phi(z|x_{1:T})\|p(z)]]$$





Bowman et al. Generating Sentences from a Continuous Space. CoNLL 2016

Sequence generation models

- Combining state-space models and RNNs:

State-space models:

- Stochastic dynamic model for the latent state:

$$p_\theta(z_{1:T}) = p_\theta(z_1)\prod_{t=2}^{T} p_\theta(z_t|z_{t-1})$$

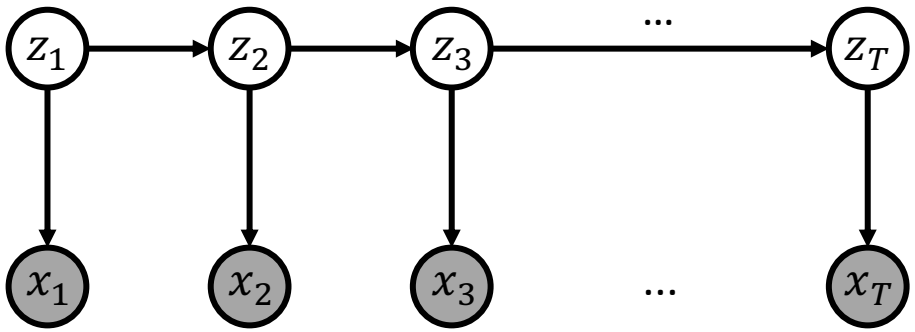
- Emission model:

$$p_\theta(x_t|z_t)$$

- Joint distribution:

$$p_\theta(x_{1:T}, z_{1:T}) = \prod_{t=1}^{T} p_\theta(z_t|z_{t-1})p_\theta(x_t|z_t)$$

(with the convention that $p_\theta(z_1|z_0) \coloneqq p_\theta(z_1)$)



Sequence generation models

- Combining state-space models with RNNs:

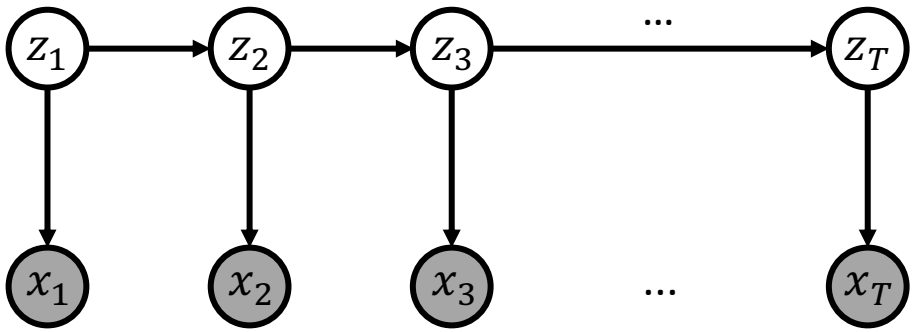
Example: Hidden Markov Model (HMM)

- Stochastic linear dynamic model for the latent state:

$$z_t = Az_{t-1} + B\epsilon_t, \epsilon_t \sim N(0, I)$$

- Linear Gaussian emission model:

$$x_t = Cz_t + D\psi_t, \psi_t \sim N(0, I)$$



Sequence generation models

- Combining state-space models with RNNs:

State-space models + non-linear dynamics:

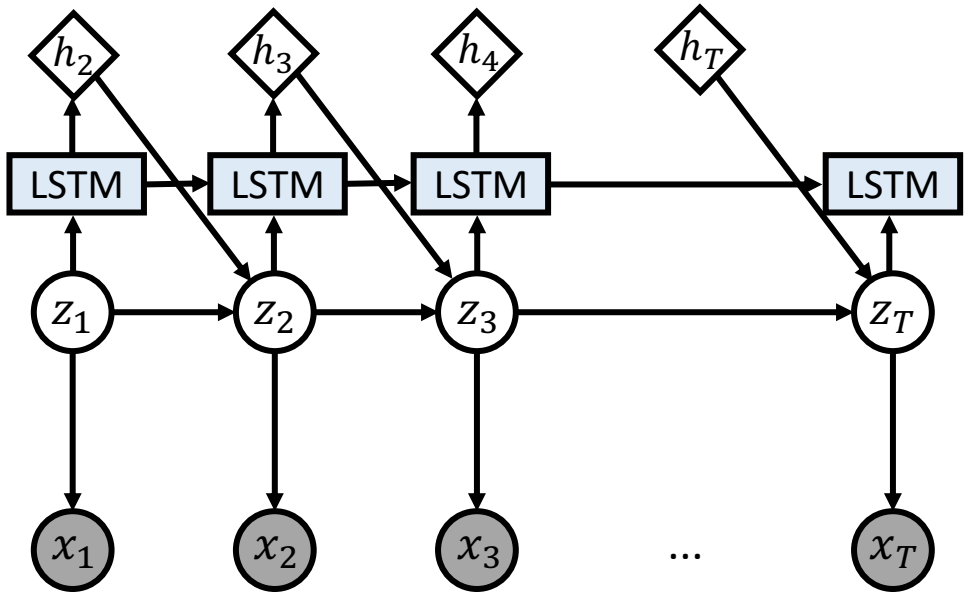
- Stochastic dynamic model parameterized by RNNs:

$$z_t = \mu_\theta^z(t) + \sigma_\theta^z(t)\epsilon_t, \epsilon_t \sim N(0, I),$$

$$\mu_\theta^z(t), \sigma_\theta^z(t) = NN_\theta(h_t^d), [h_t^d, c_t^d] = LSTM_\theta(z_{t-1}, h_{t-1}^d, c_{t-1}^d)$$

- Non-linear emission model:

$$x_t = \mu_\theta^x(z_t) + \sigma_\theta^x(z_t)\psi_t, \psi_t \sim N(0, I)$$



Sequence generation models

- Combining state-space models with RNNs:

State-space models + non-linear dynamics:

- Stochastic dynamic model parameterized by RNNs:

$$p_\theta(z_{1:T}) = \prod_{t=1}^{T} p_\theta(z_t|z_{<t})$$

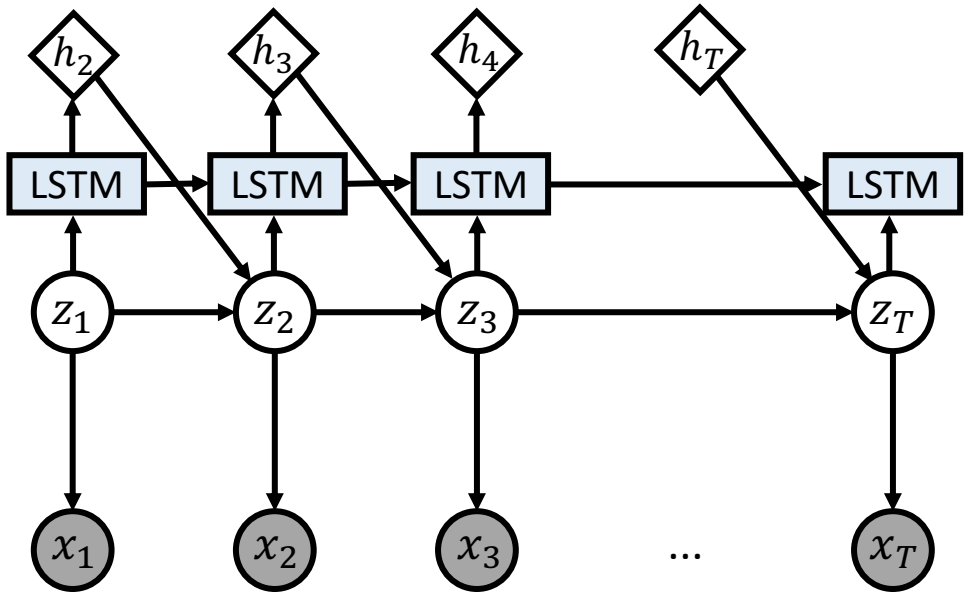
$p_\theta(z_t|z_{<t}) \neq p_\theta(z_t|z_{t-1})$ (LSTM makes all historical states relevant)

- Non-linear emission model:

$p_\theta(y_t|z_t)$: $x_t = \mu_\theta^x(z_t) + \sigma_\theta^x(z_t)\psi_t, \psi_t \sim N(0, I)$

- Joint distribution:

$$p_\theta(x_{1:T}, z_{1:T}) = \prod_{t=1}^{T} p_\theta(z_t|z_{<t}) p_\theta(x_t|z_t)$$



Sequence generation models

- Combining state-space models with RNNs:

  Training:

$$E_{p_{data}(x_{1:T})}[\log p_\theta(x_{1:T})] \geq E_{p_{data}(x_{1:T})}[E_{q_\phi(z_{1:T}|x_{1:T})}[\log p_\theta(x_{1:T}|z_{1:T})] - KL[q_\phi(z_{1:T}|x_{1:T})||p_\theta(z_{1:T})]]$$

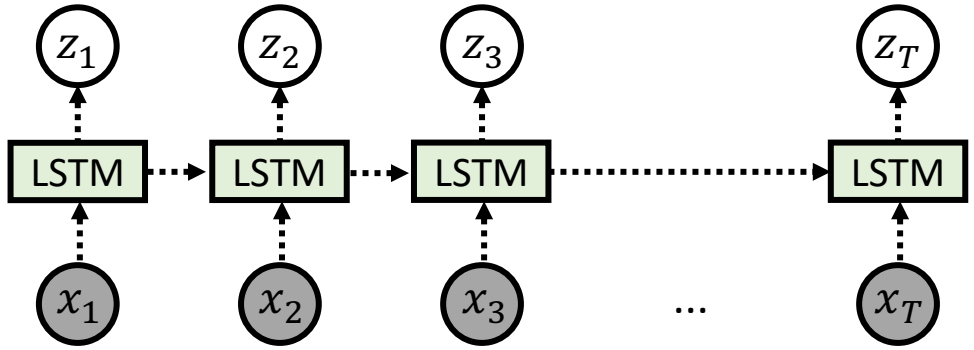
Prior parameters to be learned!

  - Generative model: $p_\theta(x_{1:T}, z_{1:T}) = \prod_{t=1}^{T} p_\theta(z_t|z_{<t})p_\theta(x_t|z_t)$
  - Designing an LSTM-based encoder:

$$q_\phi(z_{1:T}|x_{1:T}) = \prod_{t=1}^{T} q_\phi(z_t|x_{\leq t})$$

$$q_\phi(z_t|x_{\leq t}) = N(z_t; \mu_\phi^z(h_t^e), diag(\sigma_\phi^z(h_t^e)^2))$$

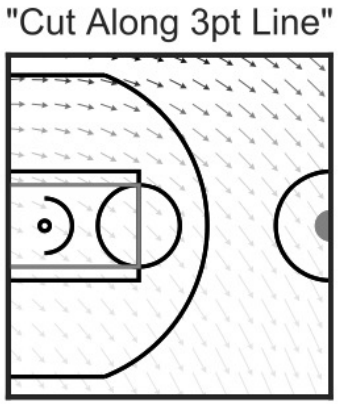
$$[h_t^e, c_t^e] = LSTM_\phi(x_t, h_{t-1}^e, c_{t-1}^e)$$



Sequence generation models

Neural state-space models have been applied to:

Model-based RL



Basketball player trajectory analysis

Speech synthesis

Fraccaro et al. Sequential Neural Models with Stochastic Layers. NeuIPS 2016
Linderman et al. Recurrent Switching Linear Dynamical Systems. AISTATS 2017
Hafner et al. Learning Latent Dynamics for Planning from Pixels. ICML 2019

Attention & Transformers

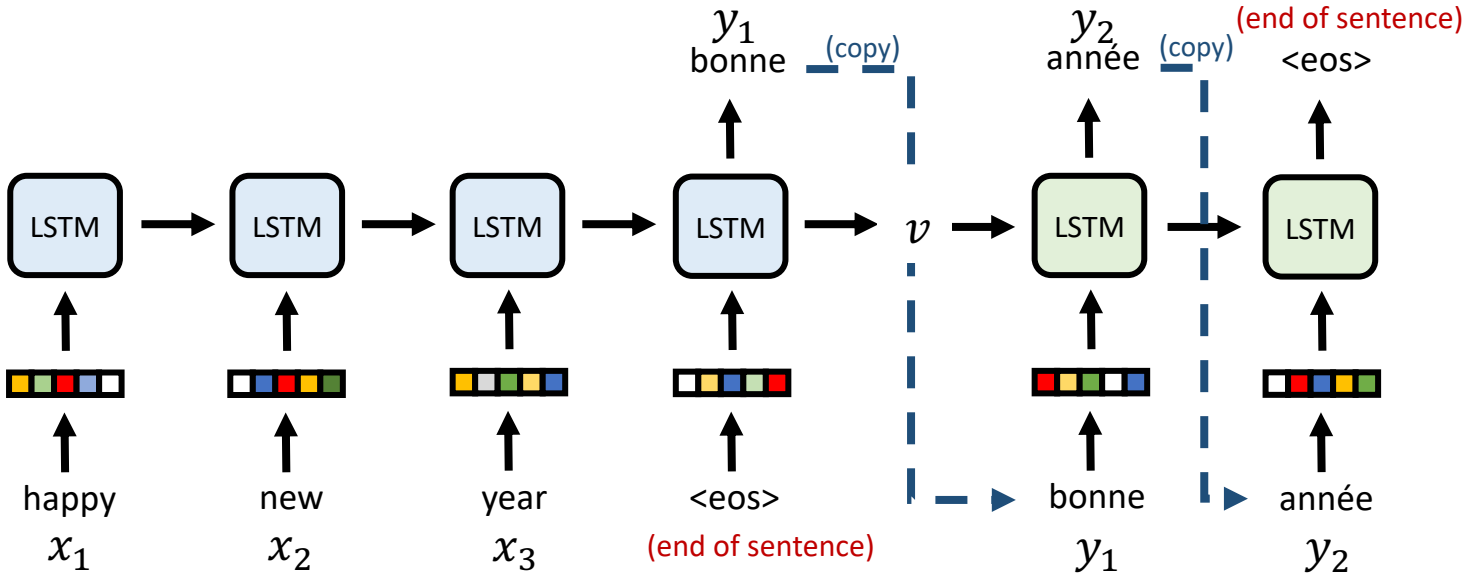
Basics

Yingzhen Li (yingzhen.li@imperial.ac.uk)

Motivation

Recap A Seq2Seq model for machine translation:

What if the sequence is very long?

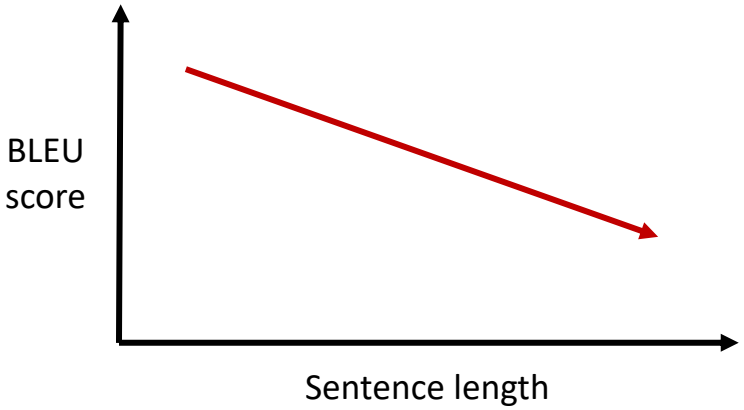


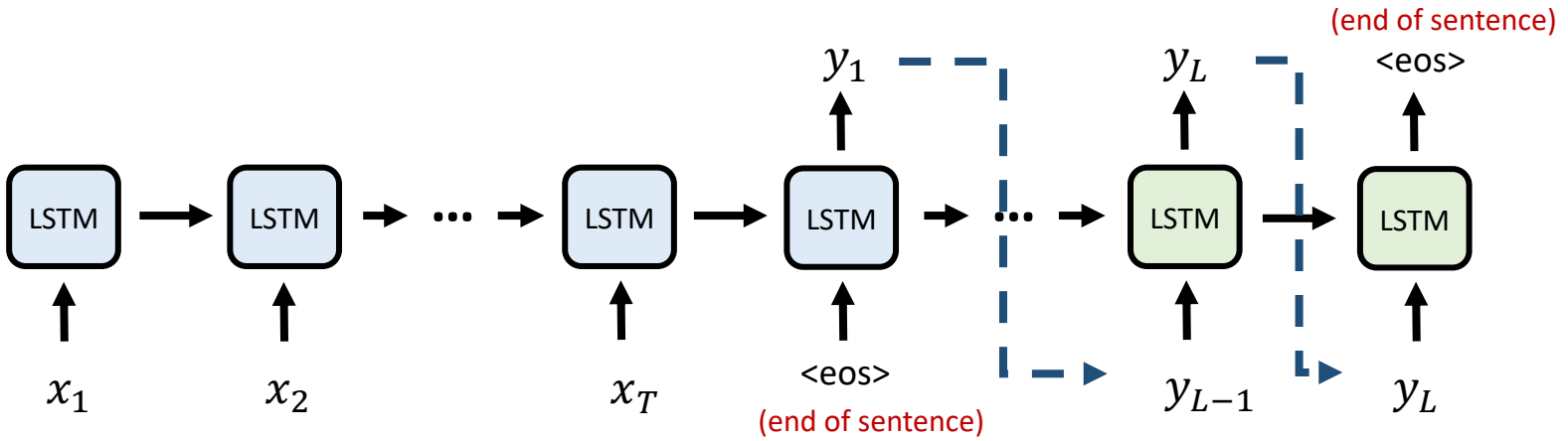
Sutskever et al. Sequence to Sequence Learning with Neural Networks. NeurIPS 2014

Motivation

Recap A Seq2Seq model for machine translation:

What if the sequence is very long?





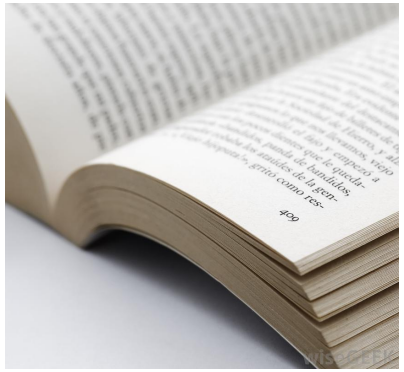
Input:

Since the release of Cyberpunk 2077 on Dec. 10, thousands of gamers have created viral videos featuring a multitude of glitches and bugs — many hilarious — that mar the game. They include tiny trees covering the floors of buildings, tanks falling from the sky and characters standing up, inexplicably pantless, while riding motorcycles...

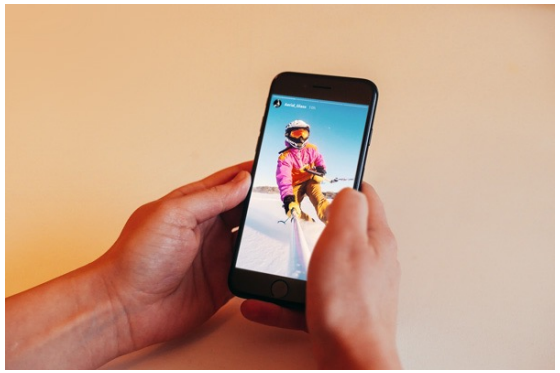
Sutskever et al. Sequence to Sequence Learning with Neural Networks. NeurIPS 2014

Motivation

- Long sequence is everywhere!



A paragraph typically contains hundreds of words



A 30sec short video contains $30 \times 60 = 1{,}800$ frames (60Hz frame rate)

Need efficient ways to handle long-term dependencies!

Attention in *Bahdanau et al.* NMT model

In Seq2Seq model, decoder is defined as

$$p_\theta(y_{1:L}|x_{1:T}) = \prod_{l=1}^{L} p_\theta(y_l|y_{<l}, v)$$

Shared representation of the entire input $x_{1:T}$

With attention:

$$p_\theta(y_{1:L}|x_{1:T}) = \prod_{l=1}^{L} p_\theta(y_l|y_{<l}, v_l)$$

Each y_l refers to the input sequence differently

$$v_l = \sum_{t=1}^{T} \alpha_{lt} f_t$$

aggregate features by weighted sum

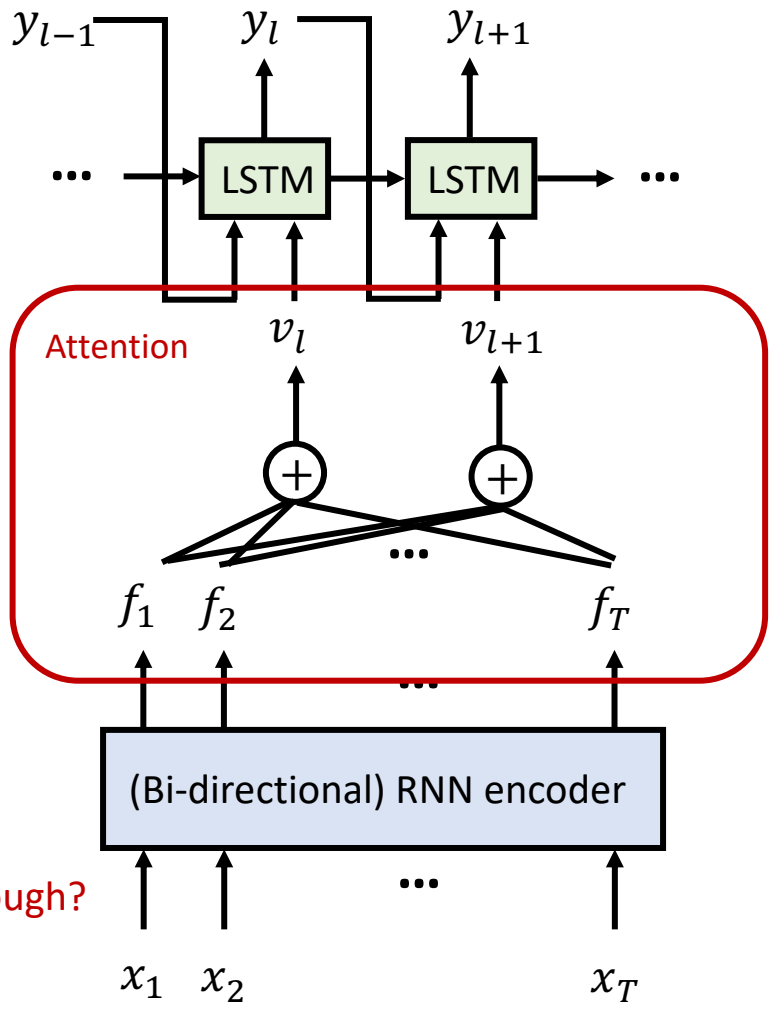
$$\alpha_l = softmax(e_l), e_l = (e_{l1}, \dots, e_{lT})$$

$$e_{lt} = a(h_{l-1}^d, f_t)$$

Decoder RNN state at step $l-1$

Encoder feature output at time t

Alignment model $a(\cdot,\cdot)$ score the "similarity/alignment" between two inputs



Still using RNNs for encoder feature extraction -- Can we do attention all the way through?

Bahdanau et al. Neural Machine Translation by Jointly Learning to Align and Translate. ICLR 2015

Attention in Transformers

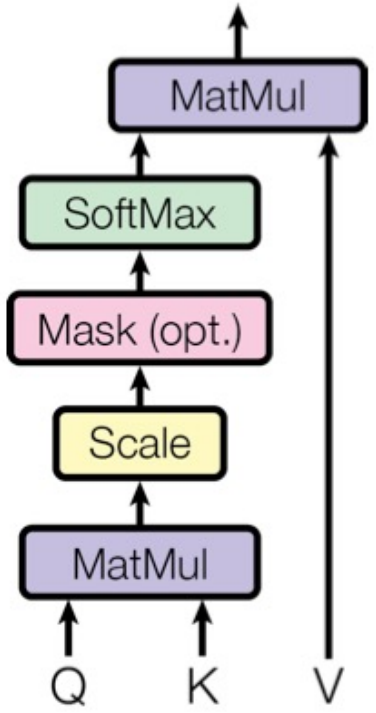
- Single head attention

Attention weights

$$Attention(Q, K, V; a) = a\left(\frac{QK^T}{\sqrt{d_q}}\right)V$$

$Q \in R^{N \times d_q}$: N query inputs, each of dimension d_q
$K \in R^{M \times d_q}$: M key vectors, each of dimension d_q
$V \in R^{M \times d_v}$: M value vectors, each of dimension d_v
$a(\cdot)$: activation function applied row-wise

Self attention: $K = Q$



Vaswani et al. Attention is all you need. NeurIPS 2017

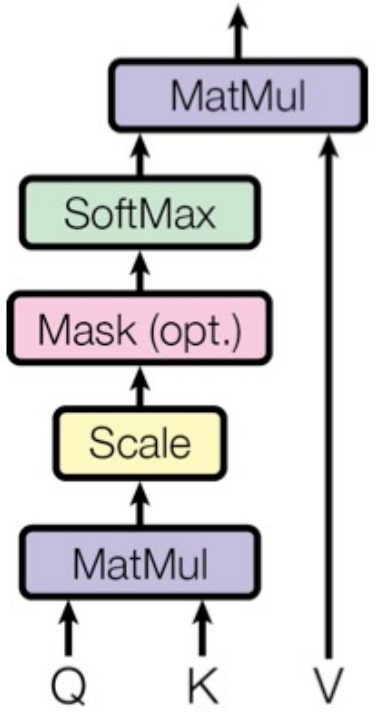
Attention in Transformers

- Single head attention

$(QK^T)_{ij} = \langle q_i, k_j \rangle$: similarity between query q_i and key k_j

$$Attention(Q, K, V; a) = a\left(\frac{QK^T}{\sqrt{d_q}}\right)V$$

Retrieve values according to the weighting



Hard attention: $a(x) = onehot(argmax\ x_i)$ for $x = (x_1, \dots, x_d)$

- Each key vector k_i is associated with a value vector v_i $(K = (k_1, \dots, k_M)^T, V = (v_1, \dots, v_M)^T)$
- The query matrix packs the query vectors $Q = (q_1, \dots, q_N)^T$
- $a(\cdot)$ applied row-wise: the n^{th} row of the output is $onehot(argmax\ \langle q_n, k_i \rangle)$
- Therefore, the n^{th} row of the attention output will be v_{i_n} for $i_n = argmax\ \langle q_n, k_i \rangle$

Vaswani et al. Attention is all you need. NeurIPS 2017

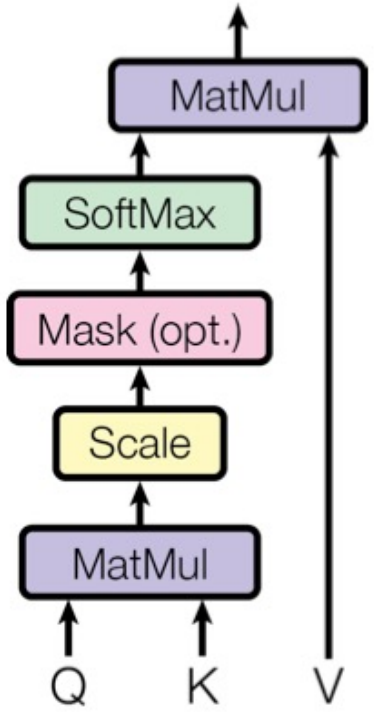
Attention in Transformers

- Single head attention

$(QK^T)_{ij} = \langle q_i, k_j \rangle$: similarity between query q_i and key k_j

$$Attention(Q, K, V; a) = a\left(\frac{QK^T}{\sqrt{d_q}}\right)V$$

Retrieve values according to the weighting



Soft attention: $a(x) = softmax(x)$ for $x = (x_1, \ldots, x_d)$

- Each key vector k_i is associated with a value vector v_i $(K = (k_1, \ldots, k_M)^T, V = (v_1, \ldots, v_M)^T)$
- The query matrix packs the query vectors $Q = (q_1, \ldots, q_N)^T$
- $a(\cdot)$ applied row-wise: the n^{th} row of the output is $softmax(q_n K^T)$
- Therefore, the n^{th} row of the attention output will be a weighted sum of value vectors v_j with weighting proportional to $\exp(\langle q_n, k_j \rangle)$

Vaswani et al. Attention is all you need. NeurIPS 2017

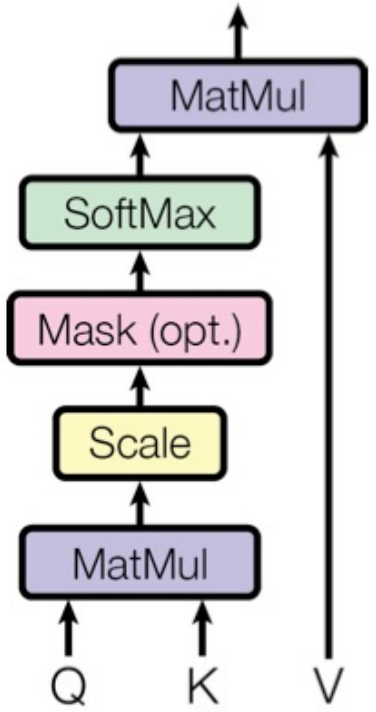
Attention in Transformers

- Single head attention

$(QK^T)_{ij} = \langle q_i, k_j \rangle$: similarity between query q_i and key k_j

$$MaskedAttention(Q, K, V; a, M) = a\left(mask\left(\frac{QK^T}{\sqrt{d_q}}, M\right)\right)V$$

Retrieve values according to the weighting and masking



Masked attention: mask out some of the attention values,
Example when $a(\cdot)$ is softmax:

- M_{nm} takes values 0 (mask out) or 1 (keep in)
- With $M_{nm} = 0$, set $(QK^T)_{nm} = -\infty$ so that $\exp\left(\frac{QK^T}{\sqrt{d}}\right)_{nm} = 0$
- So value v_m will NOT contribute to the attention output for query q_n
- Useful for sequence prediction with a given ordering: in test time, "future" is not available for the "current" to attend

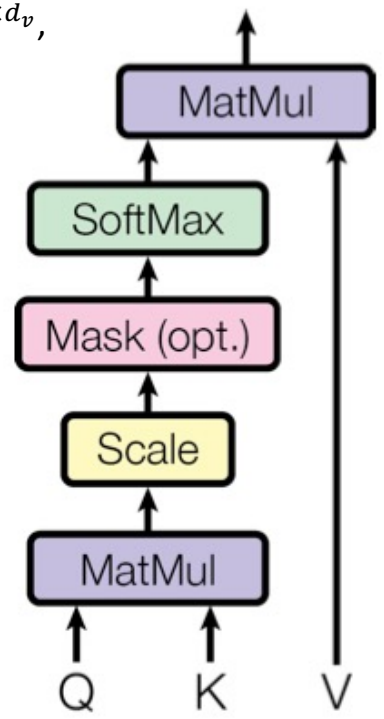
Vaswani et al. Attention is all you need. NeurIPS 2017

Attention in Transformers

- Single head attention

$Q \in R^{N \times d_q}, K \in R^{M \times d_q}, V \in R^{M \times d_v}$,
$a(\cdot)$ applied row-wise

$$Attention(Q, K, V; a) = a\left(\frac{QK^T}{\sqrt{d_q}}\right)V$$



Complexity analysis:

- Time complexity: $O(MNd_q + MNd_v)$
- Space complexity: $O(MN + Nd_v)$ (incl. intermediate steps)
- Parameters to learn:
  - K, V in the usual form: $O(Md_q + Md_v)$
  - V only for self attention: $O(Nd_v)$
  - Can also use $V = K$ (meaning $Q = V = K$ in self attention)

Vaswani et al. Attention is all you need. NeurIPS 2017

Attention in Transformers

- Multi-head attention

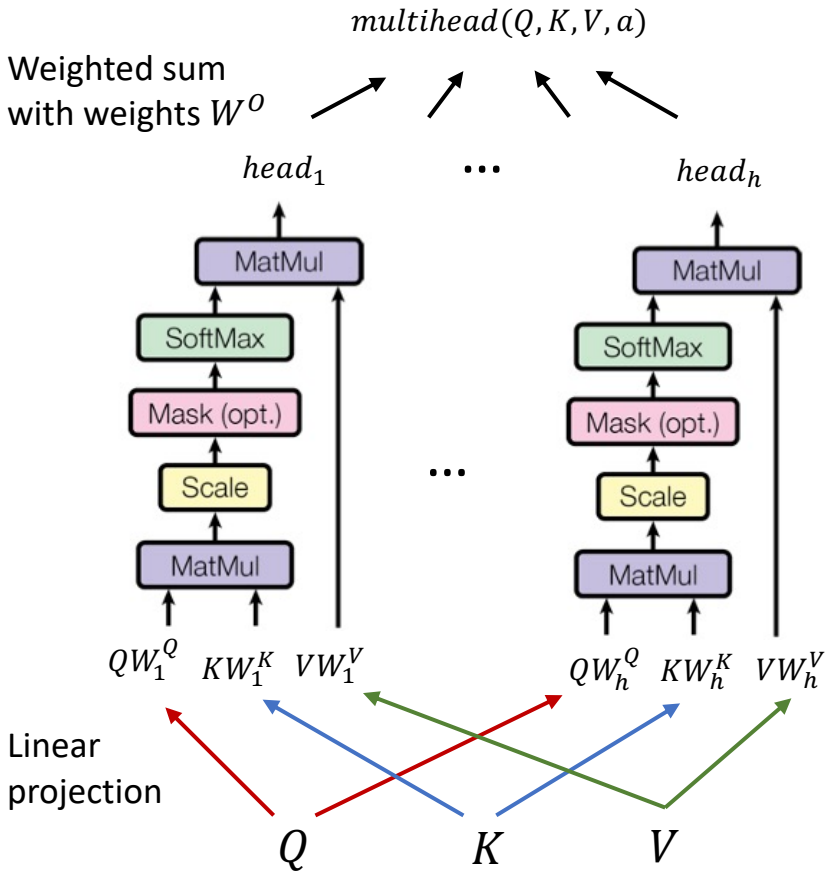
$$Multihead(Q, K, V, a) = concat(head_1, \dots, head_h)W^O$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V, a)$$

$$Attention(Q, K, V; a) = a\left(\frac{QK^T}{\sqrt{d}}\right)V$$

Different head represent different alignments, e.g.
when each query represents a word and self-attention is used:

- Head 1: find keys that are semantically similar to q
- Head 2: find keys that makes (q, k) as a subject-verb pair
- …



Vaswani et al. Attention is all you need. NeurIPS 2017

Attention in Transformers

- Multi-head attention

$$Multihead(Q, K, V, a) = concat(head_1, \dots, head_h)W^O$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V, a)$$

Complexity analysis (assume $Q \in R^{N \times d_q}$ projected to $R^{N \times \tilde{d}_q}$ and so on):

- Time complexity:

Attention heads | projections | combined output

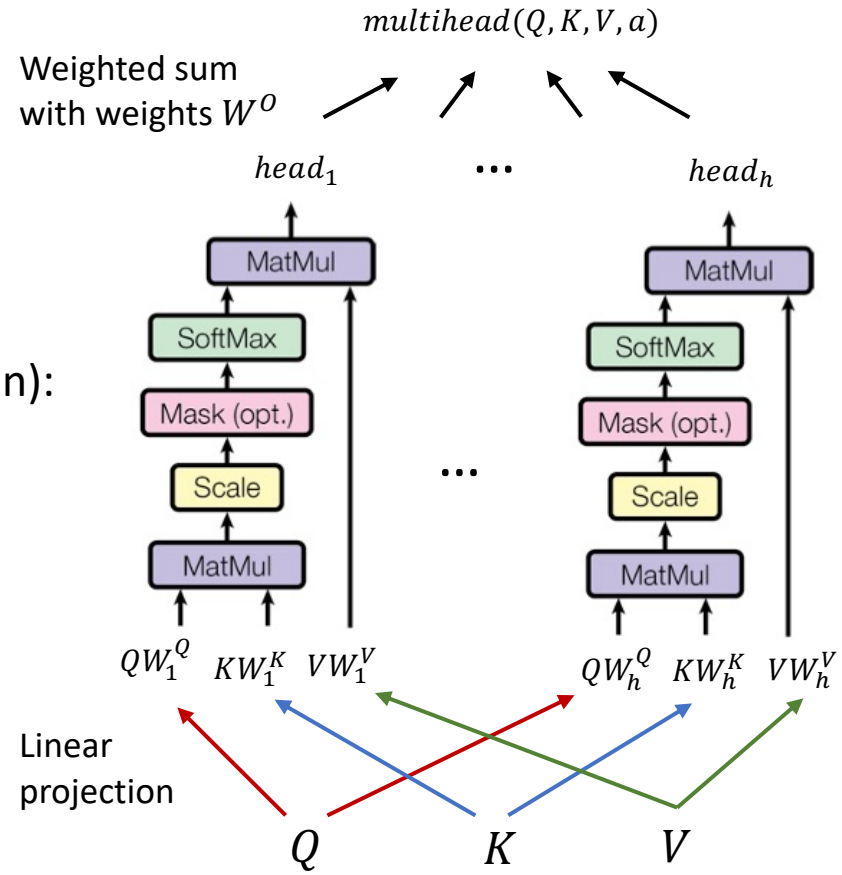
$$O(hMN(\tilde{d}_q + \tilde{d}_v) + h(\tilde{d}_q d_q(M+N) + \tilde{d}_v d_v M) + Nh\tilde{d}_v d_{out})$$

- Space complexity:

Attention heads | projections | combined output

$$O(hN(M + \tilde{d}_v) + h((N+M)\tilde{d}_q + M\tilde{d}_v) + Nd_{out})$$

- Parameters to learn (apart from K and V):
  - Projection parameters W_i^Q, W_i^K, W_i^V
  - Output weight matrix W^O



Vaswani et al. Attention is all you need. NeurIPS 2017

Transformer Architecture

Attention based encoder + decoder:

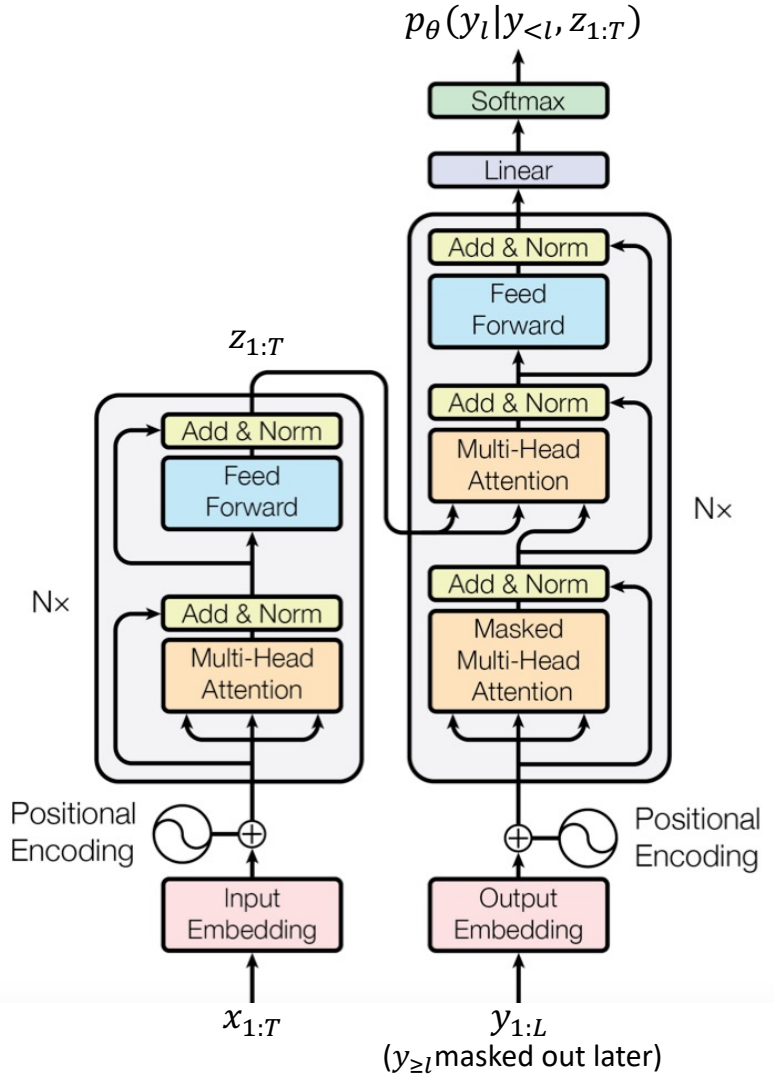
$$p_\theta(y_{1:L}|x_{1:T}) = \prod_{l=1}^{L} p_\theta(y_l|y_{<l}, z_{1:T})$$

$y_{\geq l}$ will be masked out in the first layer of decoder

Encoder attention outputs used in decoder

The input to the decoder:

- Training time: $y_{1:L}$
- Test time: $(y_1, \dots, y_{l-1}, \emptyset, \dots, \emptyset)$



Vaswani et al. Attention is all you need. NeurIPS 2017

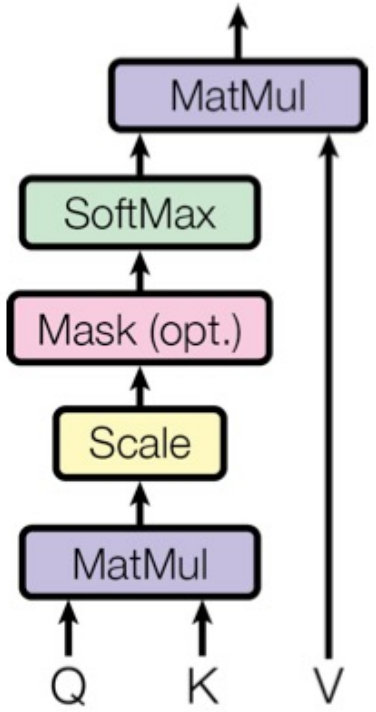
Transformer Architecture

- Single head attention

$$Attention(Q, K, V; a) = a\left(\frac{QK^T}{\sqrt{d_q}}\right)V$$

Permutation equivariant:

- $\tilde{Q}$ constructed by swapping the i^{th} and j^{th} row in Q
  $\Rightarrow Attention(\tilde{Q}, K, V, a)$ equals to $Attention(Q, K, V, a)$
  except that the i^{th} and j^{th} rows are swapped
- The ordering information is irrelevant!



Vaswani et al. Attention is all you need. NeurIPS 2017

Transformer Architecture

Attention based encoder + decoder:

$$p_\theta(y_{1:L}|x_{1:T}) = \prod_{l=1}^{L} p_\theta(y_l|y_{<l}, z_{1:T})$$

$y_{\geq l}$ will be masked out in the first layer of decoder

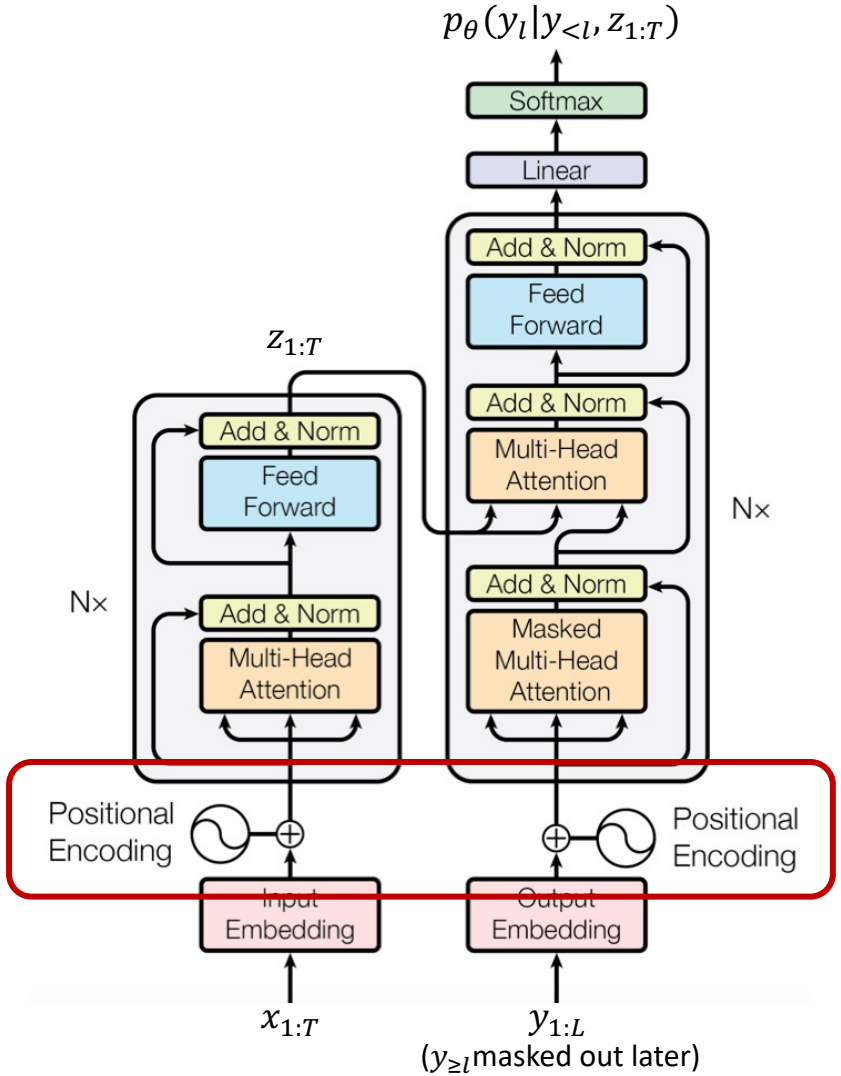
Encoder attention outputs used in decoder

Position encoding: inject ordering information
Can either be learned or be a pre-defined mapping, e.g.:

$$PE(pos, 2i) = \sin(pos/10000^{2i/d_{out}})$$

$$PE(pos, 2i+1) = \cos(pos/10000^{2i/d_{out}})$$

output: $word\ embedding(x_t) + PE(t, 1{:}\, d_{emb})$



Vaswani et al. Attention is all you need. NeurIPS 2017

Transformer Architecture

Attention based encoder + decoder:

$$p_\theta(y_{1:L}|x_{1:T}) = \prod_{l=1}^{L} p_\theta(y_l|y_{<l}, z_{1:T})$$

$y_{\geq l}$ will be masked out in the first layer of decoder

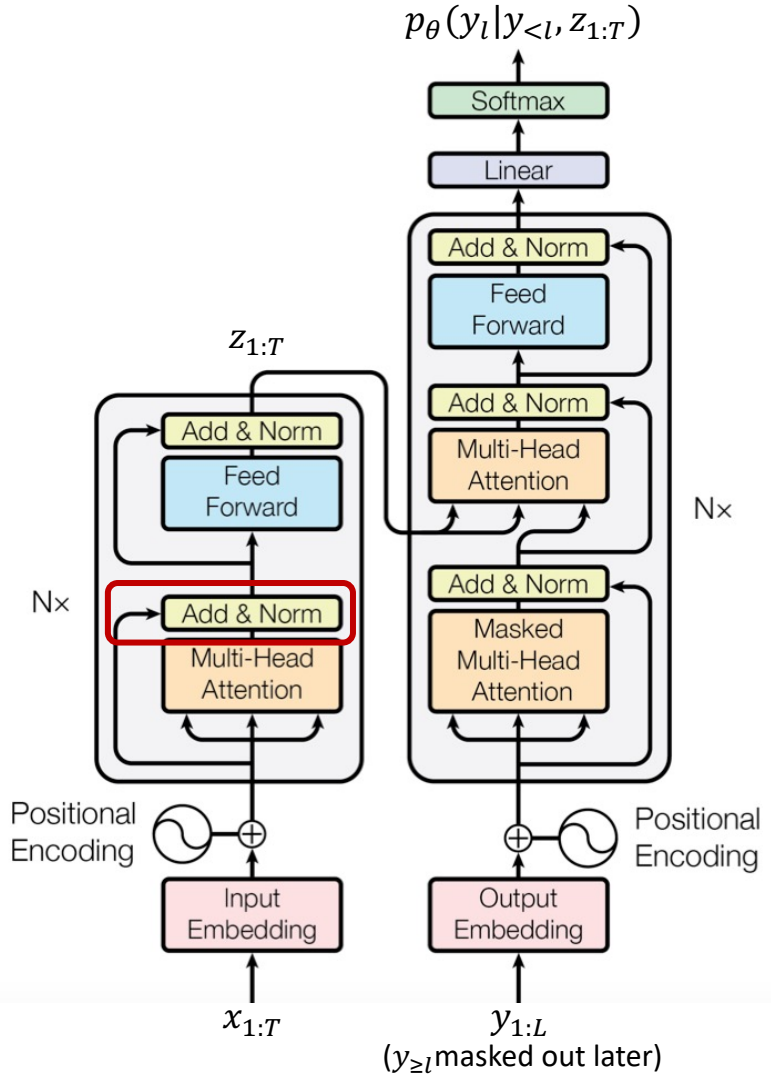
Encoder attention outputs used in decoder

Add & Norm:

Residual connection

$$LayerNorm(x + Sublayer(x))$$

Layer normalization is similar to batch normalization except that it is performed within a single hidden layer output



Vaswani et al. Attention is all you need. NeurIPS 2017

Transformer Architecture

Attention based encoder + decoder:

$$p_\theta(y_{1:L}|x_{1:T}) = \prod_{l=1}^{L} p_\theta(y_l|y_{<l}, z_{1:T})$$

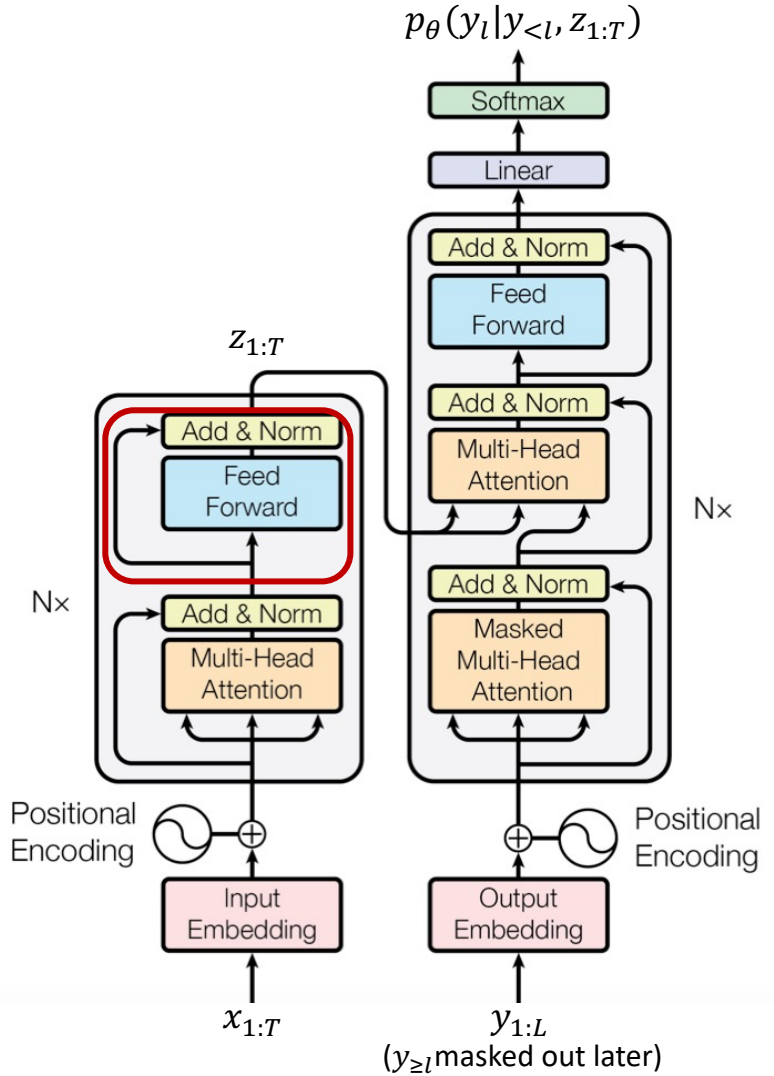
$y_{\geq l}$ will be masked out in the first layer of decoder

Encoder attention outputs used in decoder

Feed-forward network:

Applied to each of the output value vectors (i.e. row vectors)

independently and identically



Vaswani et al. Attention is all you need. NeurIPS 2017

Transformer Architecture

Attention based encoder + decoder:

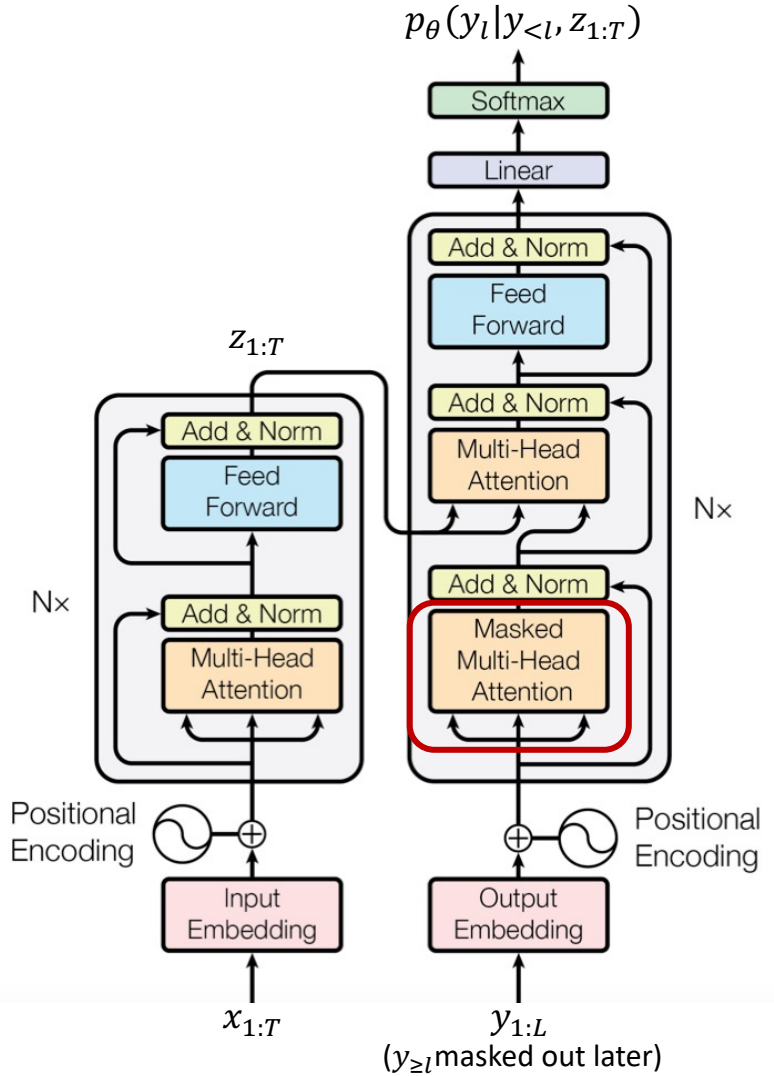
$$p_\theta(y_{1:L}|x_{1:T}) = \prod_{l=1}^{L} p_\theta(y_l|y_{<l}, z_{1:T})$$

$y_{\geq l}$ will be masked out in the first layer of decoder

Encoder attention outputs used in decoder

Maked Multi-head Attention:
Prevent the model to use "future" information for predicting the current output
(training time input: $(y_1, \dots, y_{l-1}, , y_l \dots, y_L)$)
(test time input: $(y_1, \dots, y_{l-1}, \emptyset, \dots, \emptyset)$)

should be masked out



Vaswani et al. Attention is all you need. NeurIPS 2017

Transformer Architecture

Attention based encoder + decoder:

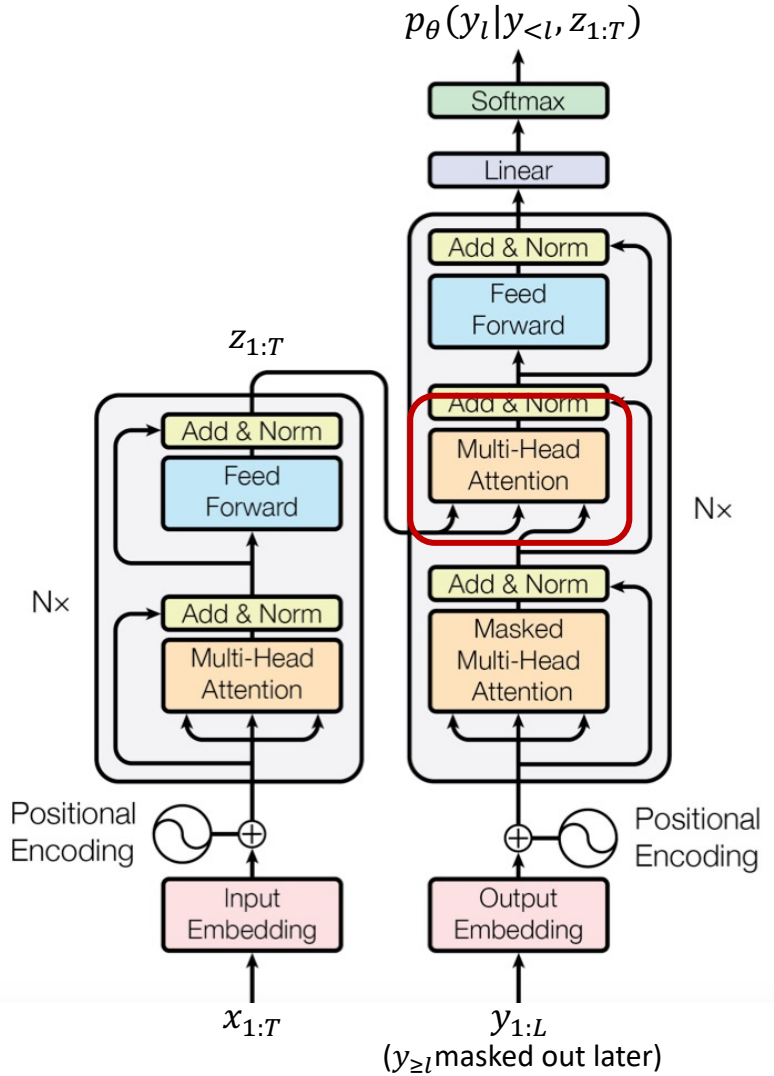
$$p_\theta(y_{1:L}|x_{1:T}) = \prod_{l=1}^{L} p_\theta(y_l|y_{<l}, z_{1:T})$$

$y_{\geq l}$ will be masked out in the first layer of decoder

Encoder attention outputs used in decoder

Multi-head Attention using encoder output $z_{1:T}$

- $z_{1:T}$ are used as the keys and values of this attention module
- Allow the decoder to attend every word in the input $x_{1:T}$ for each of the predicted output $y_{1:l-1}$ so far

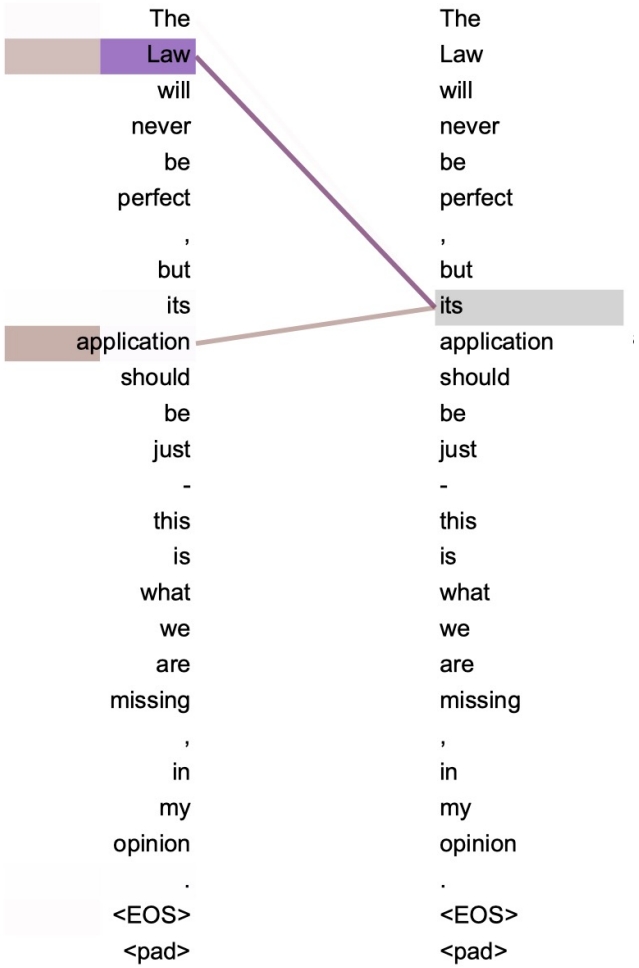


Vaswani et al. Attention is all you need. NeurIPS 2017

Visualising Learned Attentions

reflect structure of the sentence

Different head show different patterns



Vaswani et al. Attention is all you need. NeurIPS 2017

Attention & Transformers

Advances & Applications

Yingzhen Li (yingzhen.li@imperial.ac.uk)

Attention applications pre-2017

Attention used in RNN generators back in 2013:

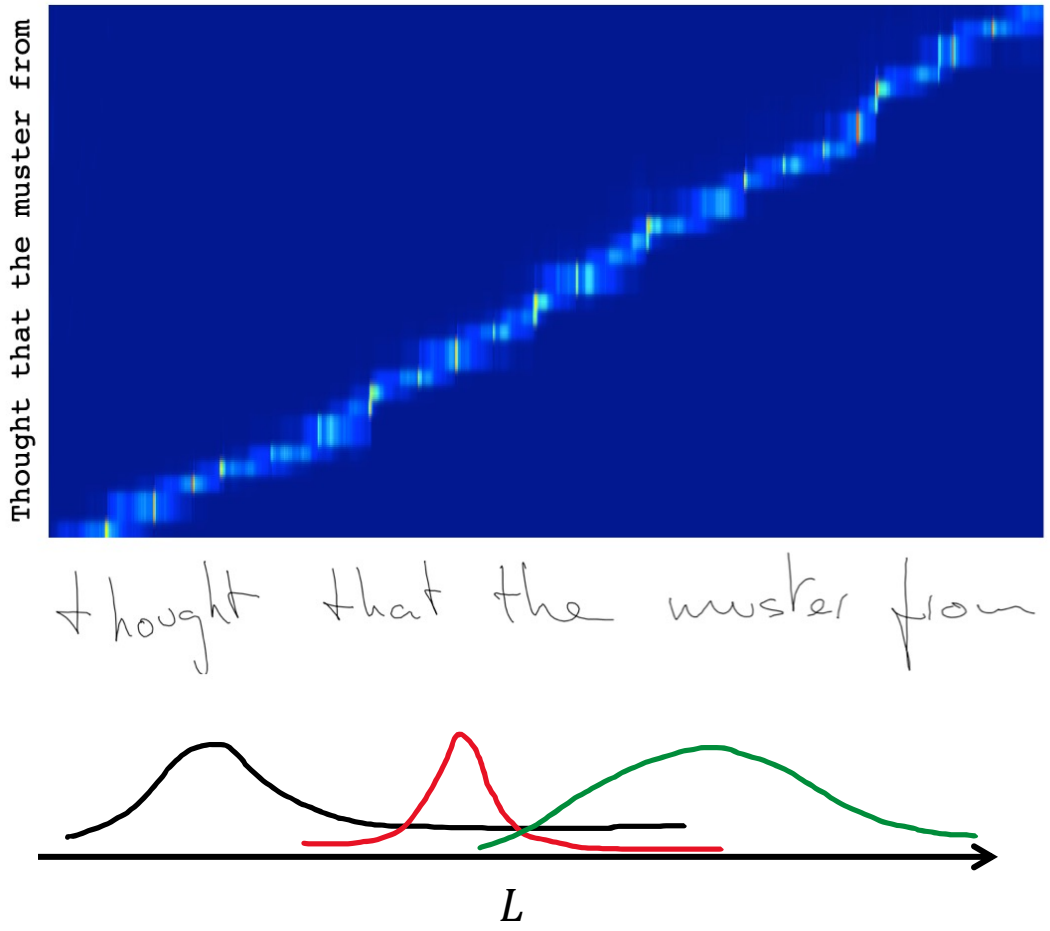
- Idea: align two different sequences with different length
  - $x_{1:T}$: x_t is the pen position at time t
  - $y_{1:L}$: y_l is the l^{th} character in the text

RNN parameterised mixture of Gaussian attention

$$Attention(x_t, y_{1:L}) = \sum_{l=1}^{L} \phi(t,l) emb(y_l)$$

$$\phi(t,l) = \sum_{k=1}^{K} \alpha_t^k \exp(-\beta_t^k (l - \kappa_t^k)^2)$$

$$\alpha_t^k, \beta_t^k, \kappa_t^k = MLP(h_t), h_{1:T} = RNN(x_{1:T})$$



Graves. Generating Sequences with Recurrent Neural Networks. arXiv:1308.0850

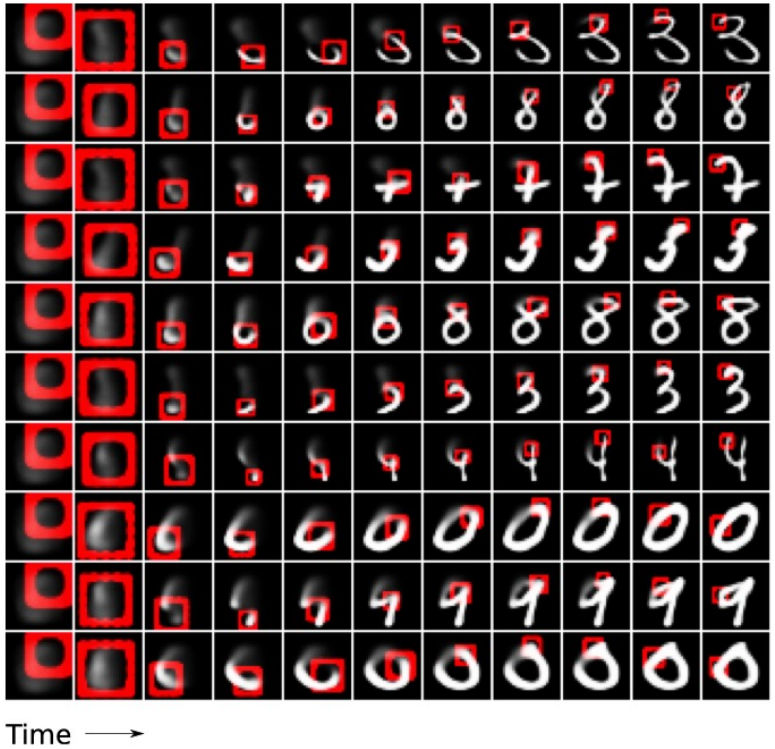
Attention applications pre-2017

DRAW model, extending Graves (2013):

- A VAE generative model for images combining RNNs and Gaussian attention
- Treat the intermediate "drawing state" as latent variable:

$$p(x|z) = p(x|z_{1:T})$$

- Decoder uses "read" and "write" operations guided by attention
- Gaussian attention to select focused patch at time t
- The Gaussian attention filter parameters are obtained from the RNN in the decoder

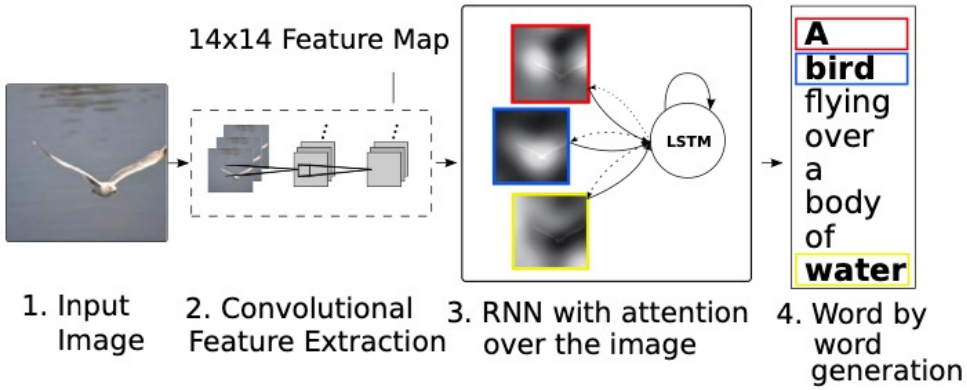


Gregor et al. DRAW: A Recurrent Neural Network For Image Generation. ICML 2015

Attention applications pre-2017

Attention applied in image captioning:

- CNN low-level features are indexed by spatial location
- Sort CNN features and then process with RNNs
- Apply similar RNN attention methods in Bahdanau et al. (2014)



A woman is throwing a frisbee in a park.

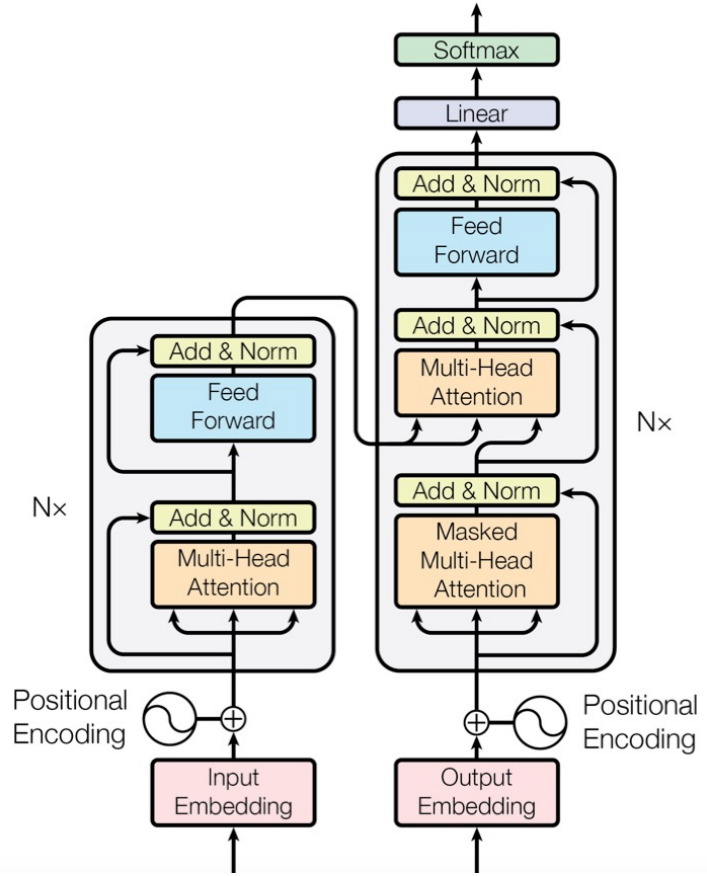
A dog is standing on a hardwood floor.

Xu et al. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. ICML 2015

Transformer in NLP applications

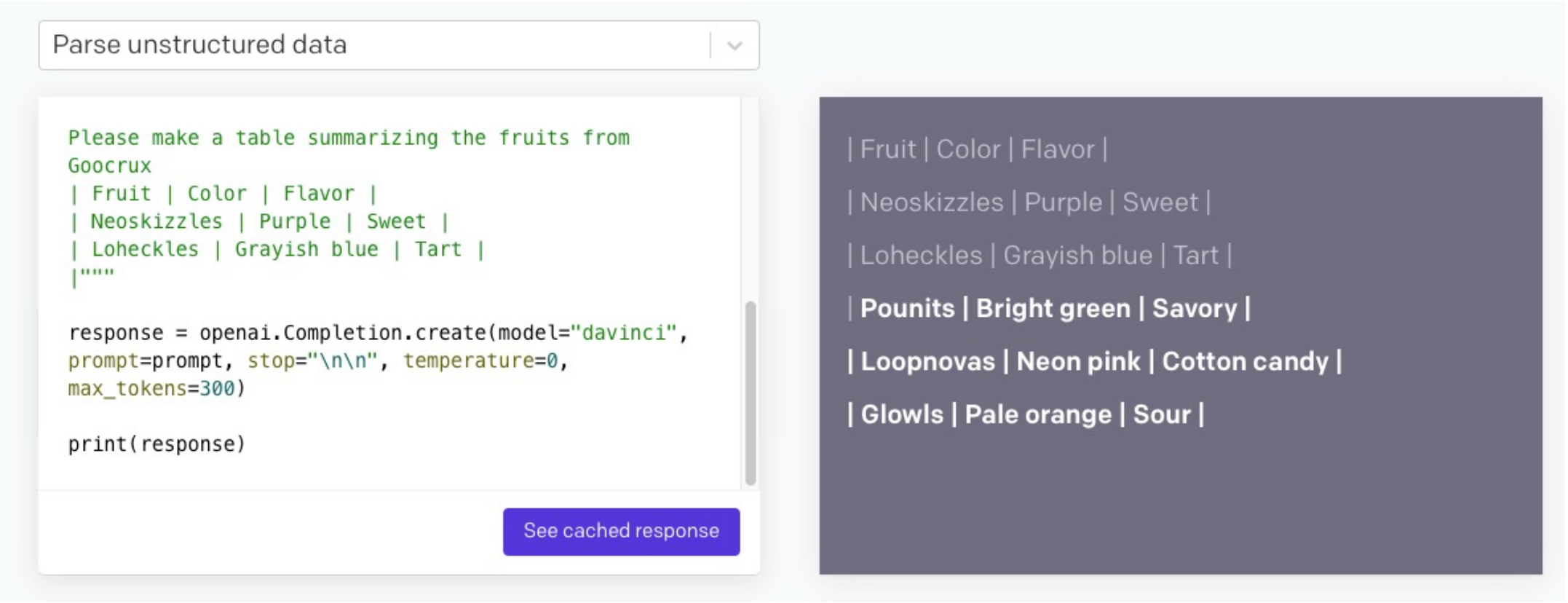
- Pretraining Transformer models on massive data

- Deeper & bigger Transformer architecture with modifications
- Pretrained on very big corpus by e.g. randomly masking out and predicting words in a sequence
- Fine-tune on specific tasks that the user cares



Devlin et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. NAACL 2019
Brown et al. Language Models are Few-shot Learners. NeurIPS 2020

Transformer in NLP applications



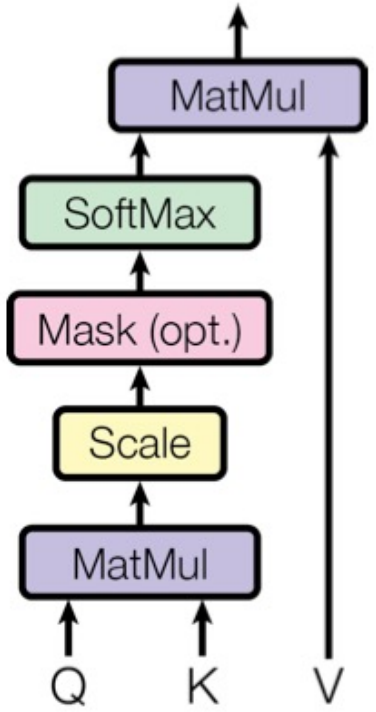
beta.openai.com

Brown et al. Language Models are Few-shot Learners. NeurIPS 2020

Multi-head Attention in Other Applications

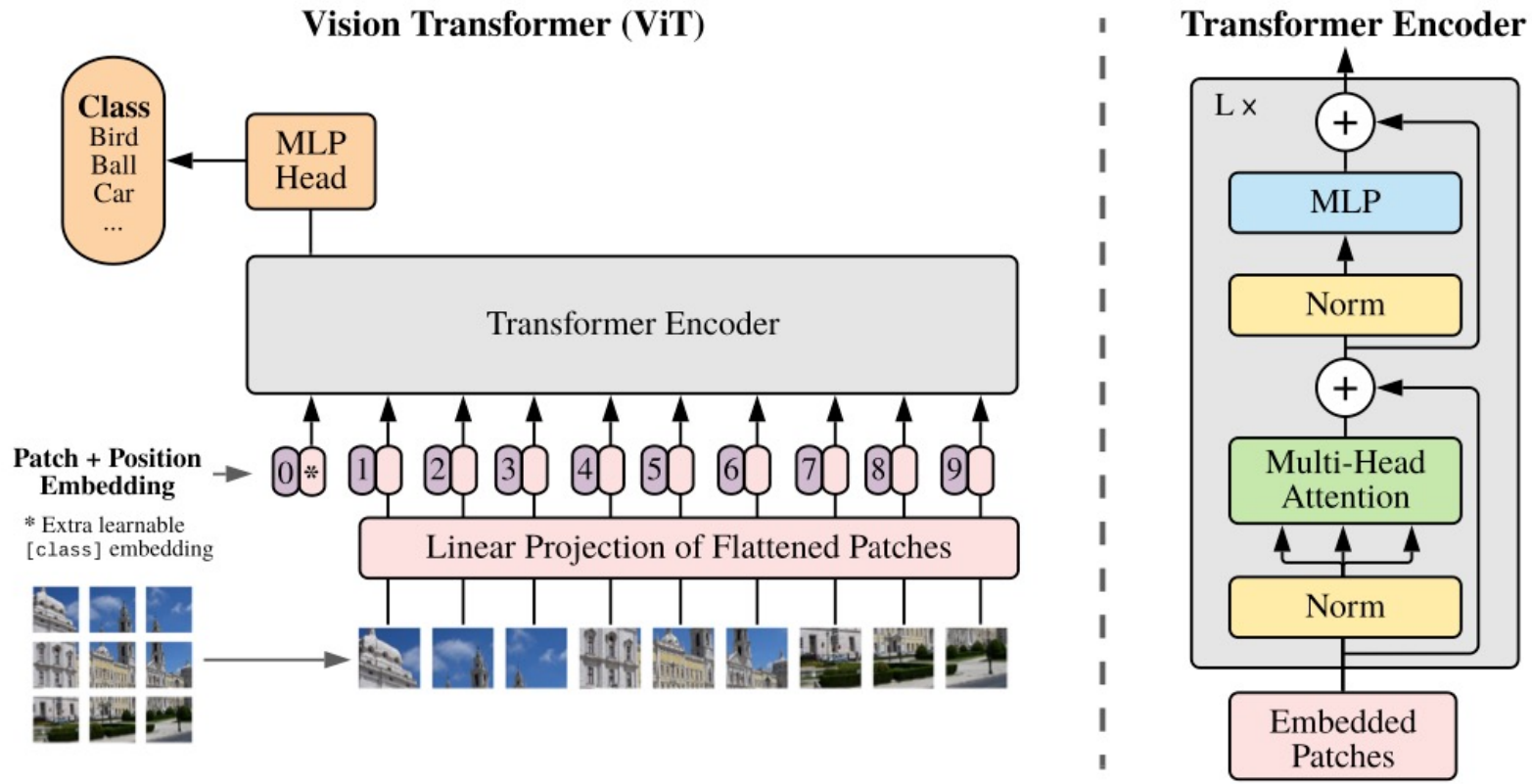
⇒ Can be applied to any Set Data with/out indexing!

- Text = set of words, Image = set of pixels (or set of patches), Graph = set of nodes and edges, point cloud = set of points, …
- Cross-modality application: embed points from diff. modality to the same space then apply attention



Vaswani et al. Attention is all you need. NeurIPS 2017

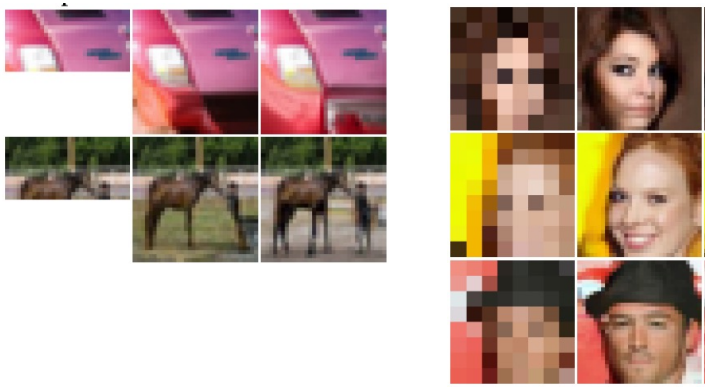
Multi-head Attention in Other Applications



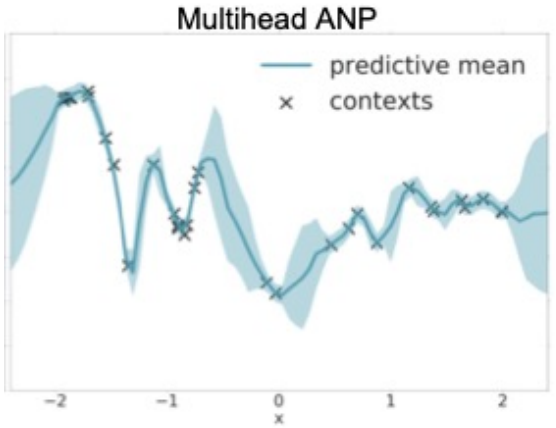
Dosovitskiy et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. ICLR 2021

Multi-head Attention in Other Applications

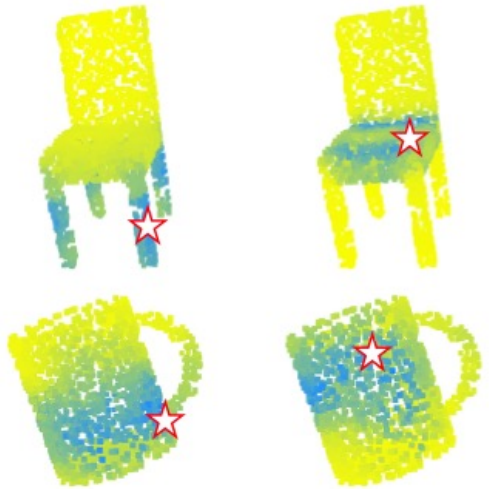
Image completion & super-resolution



Regression & supervised learning



Point cloud applications



Parmar et al. Image Transformer. ICML 2018
Kim et al. Attentive Neural Processes. ICLR 2019
Guo et al. PCT: Point Cloud Transformer. arXiv:2012.09688

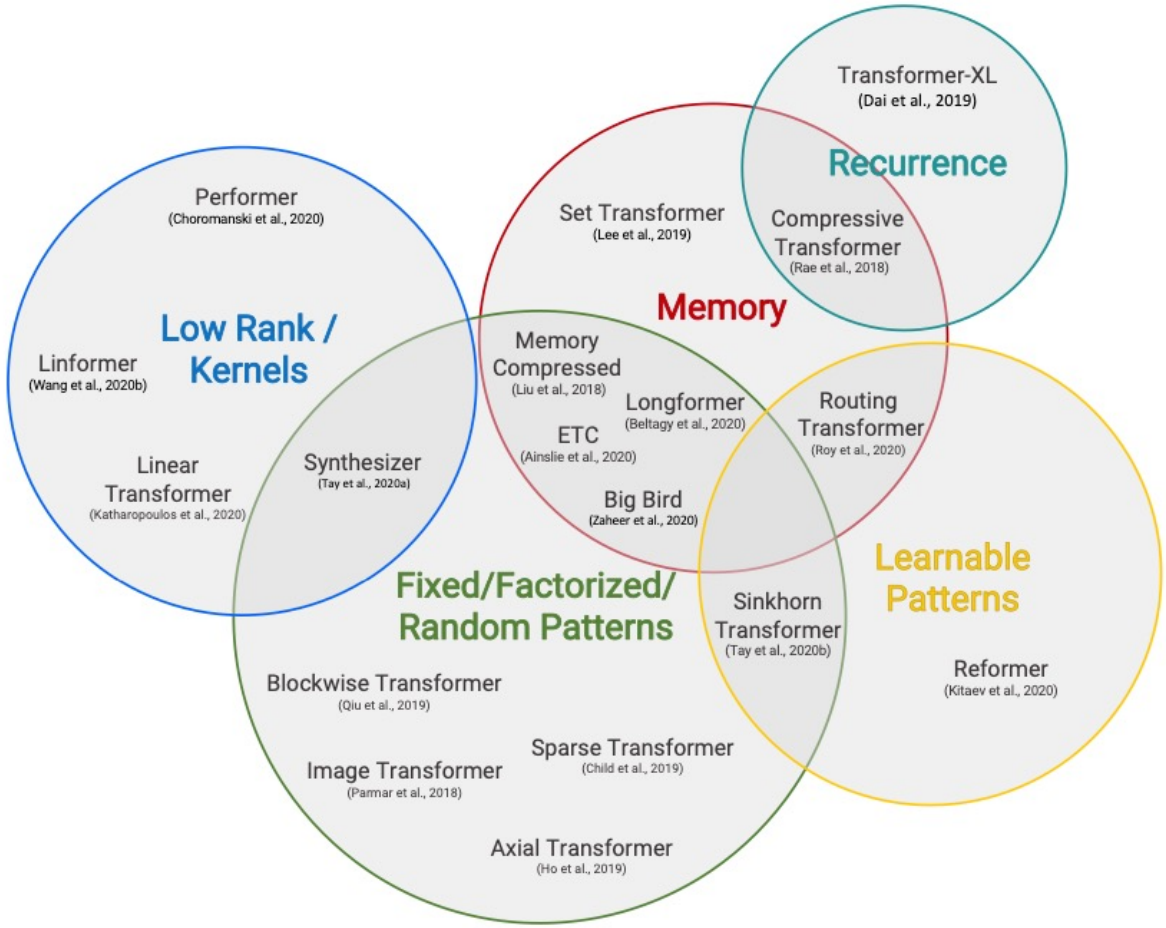
Efficient Transformers

Self attention complexities:

Assume the query $Q \in R^{N \times d_q}$

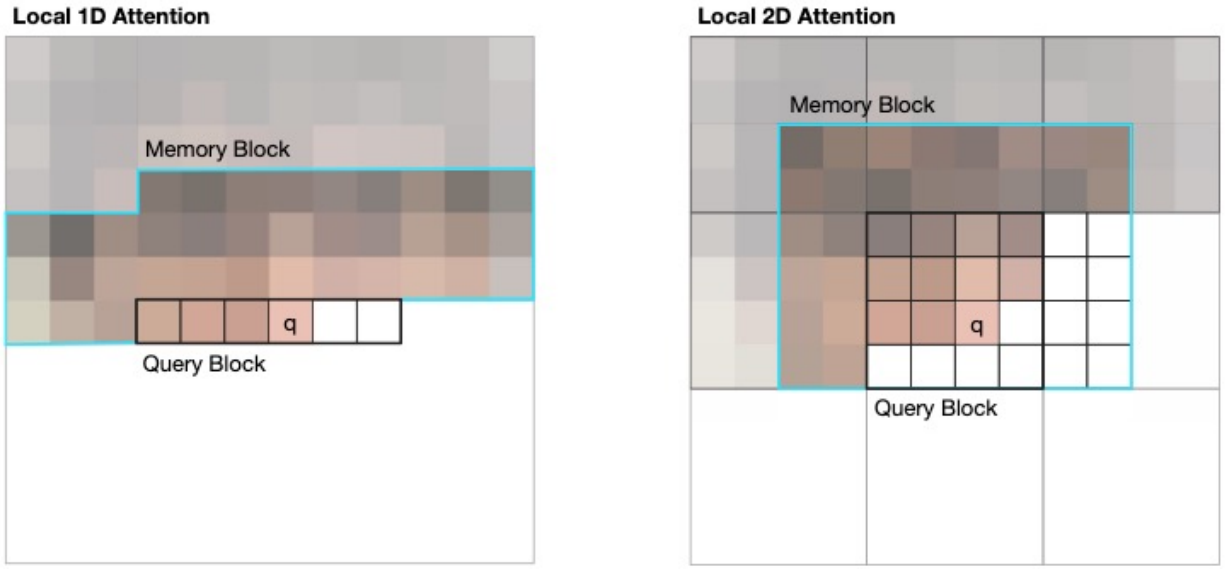
- Time complexity: $O(N^2)$
- Space complexity: $O(N^2)$
- Both complexities also scales linearly with d_q

Many existing approaches for improving efficiency!



Tay et al. Efficient Transformers: A Survey. arXiv:2009.06732

Efficient Transformers

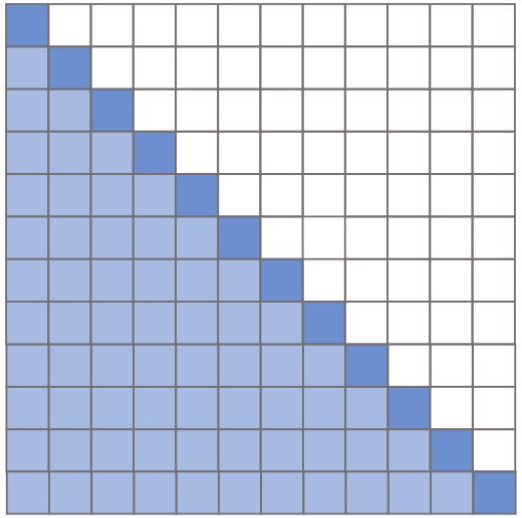


Local attention by defining “neighbourhood”:

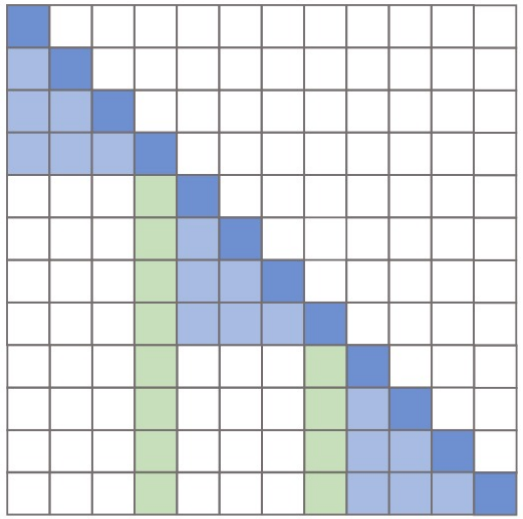
- Treat the input as a sequence with position ordering and build an auto-regressive model
- At current position, both the query and the key (or memory) inputs are “local”

Parmar et al. Image Transformer. ICML 2018

Efficient Transformers



(a) Transformer



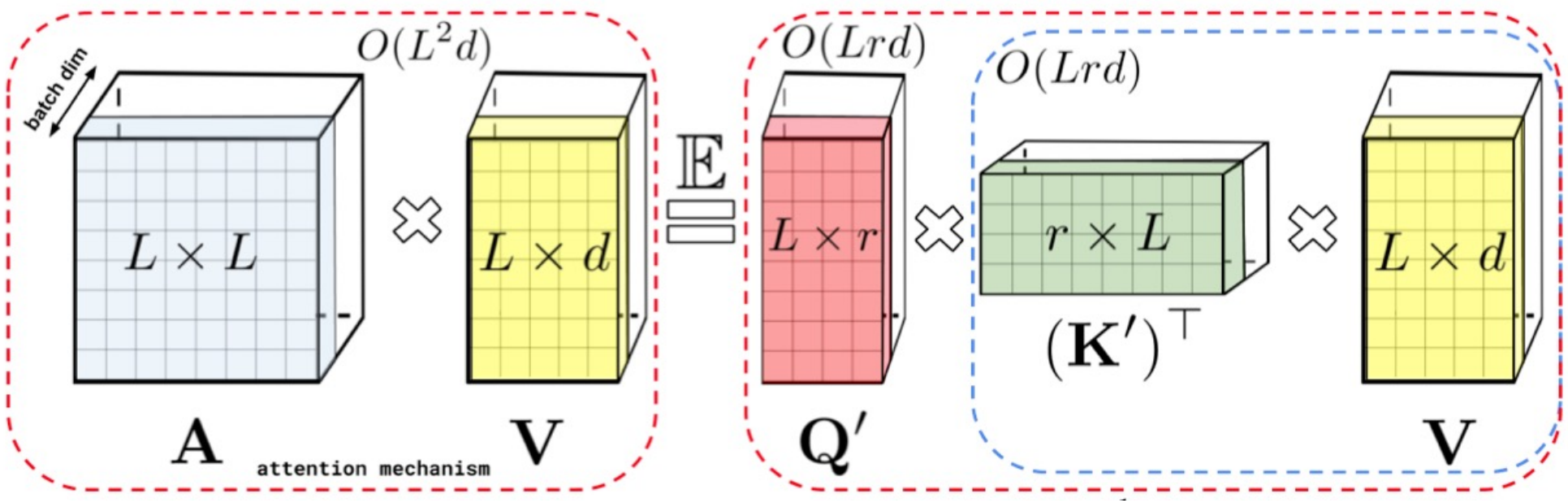
(b) Sparse Transformer

Sparse attention by defining attention weight matrix patterns:

- Full attention matrix before non-linearity: $(QK^T)_{ij} = \langle q_i, k_j \rangle$
- Sparse attention matrix: fix rules such that only for certain pairs of (i, j) the attention entry is computed
- Different attention heads can have different sparsity patterns

Child et al. Generating Long Sequences with Sparse Transformers. arXiv:1904.10509

Efficient Transformers



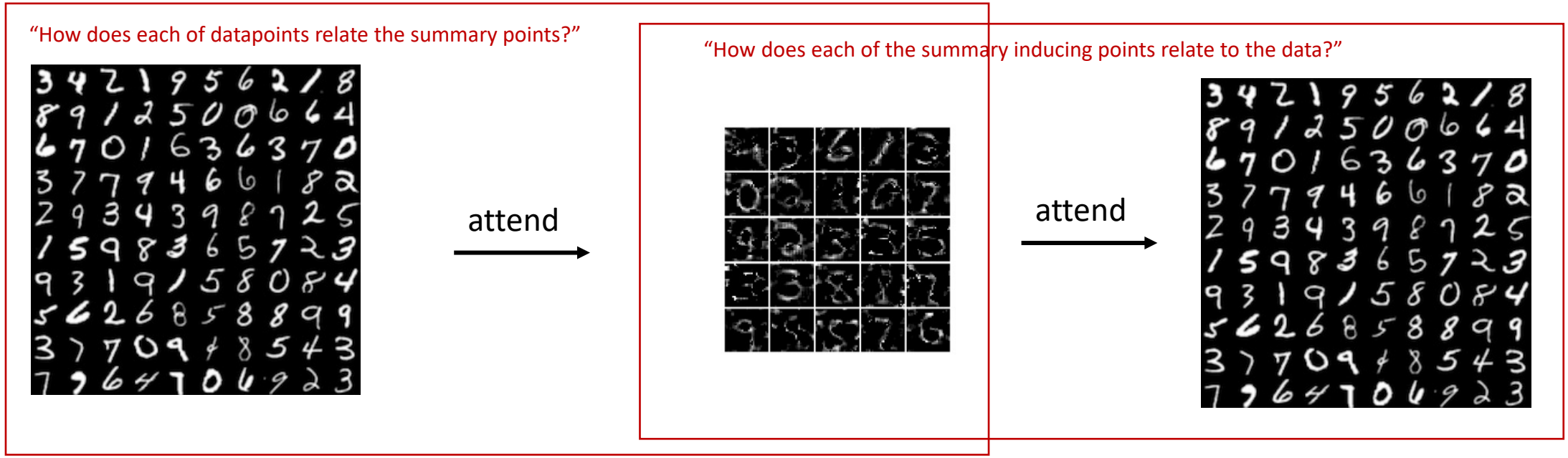
Low-rank approximations:

Can be much faster to compute!

- If the activation $a(\cdot)$ in $Attention(Q, K, V, a)$ is linear: $(QK^T)V = Q(K^TV)$
- In general the attention matrix has entries $A_{ij} = K(q_i, k_j)$ with $K(\cdot,\cdot)$ a kernel function
- Random feature approximation can be done to (approximately) compute A fastly

Choromanski et al. Rethinking attention with Performers. ICLR 2021

Efficient Transformers



Learnable inducing points for set summary:

- $MAB(X, X)$ as building block, requiring $MultiHeadAttention(X, X)$
- Idea: Using learnable inducing points I_M as a "bridge":

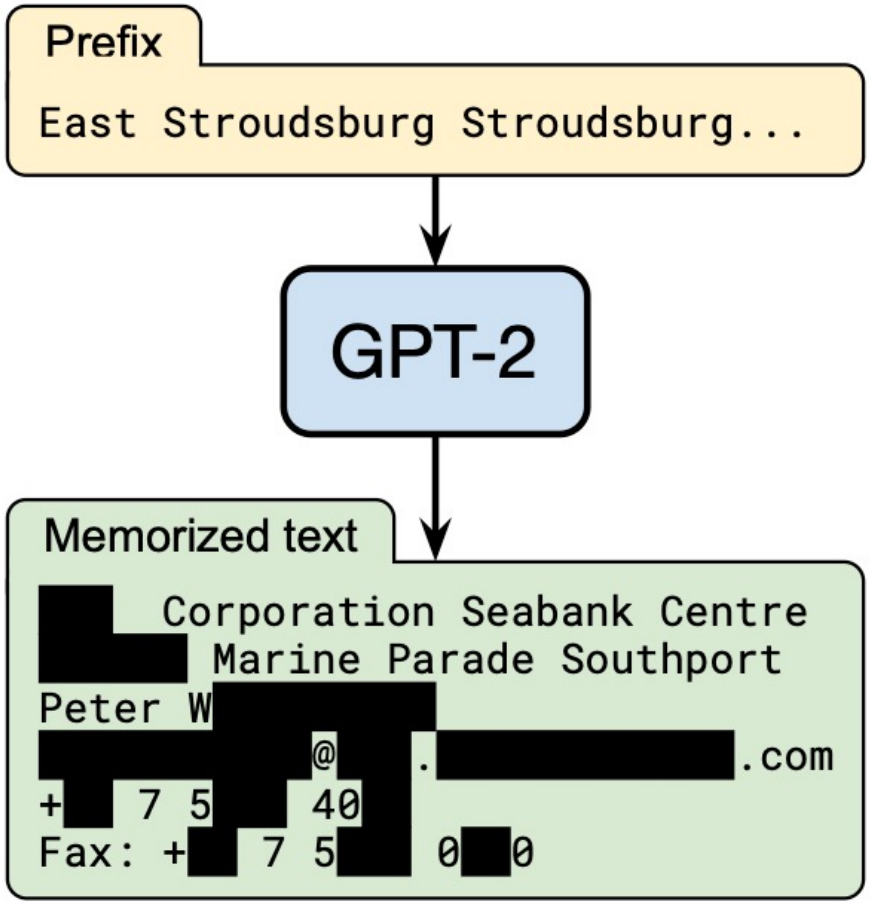
$$ISAB = MAB(X, MAB(I_M, X))$$

Lee et al. Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks. ICML 2019
Figure adapted from: Tomczak and Welling. VAE with a VampPrior. AISTATS 2018.

Memorisation in Transformers

Potential privacy issues:

- Transformer-based Language models has billions of parameters
- Capable for memorising the input data
- Currently they are pre-trained on open-web text without any privacy protection
- Attacks can steal sensitive input data



Carlini et al. Extracting Training Data from Large Language Models. arXiv:2012.07805