Learning (IV)

Mingsheng Long

Tsinghua University

Spring 2023

Outline

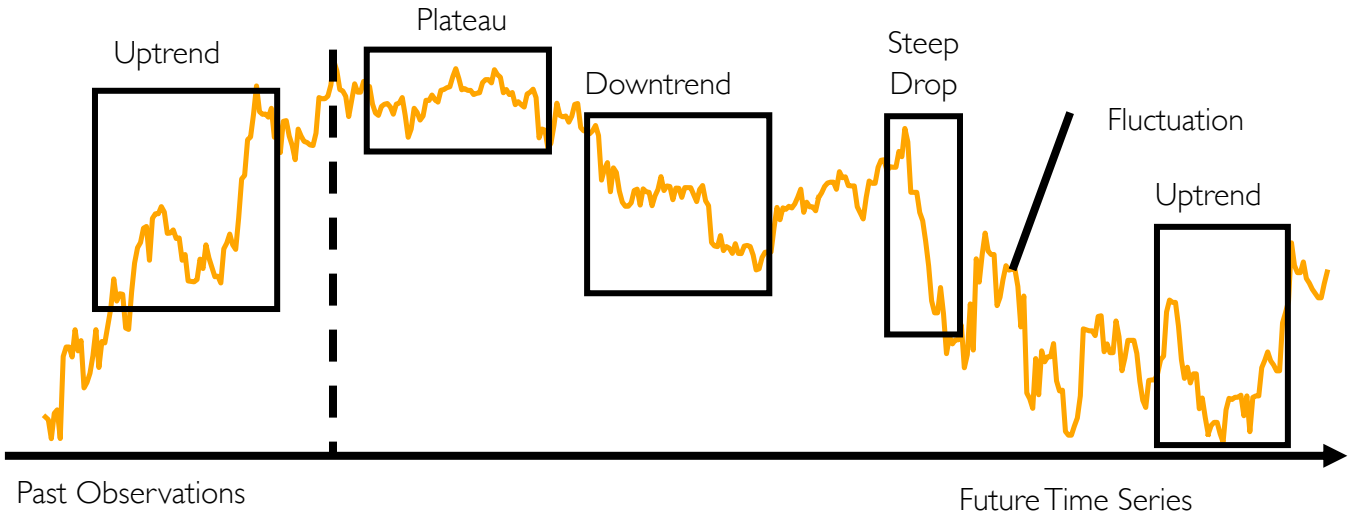
- **Recurrent Neural Network (RNN)**
  - LSTM: Long Short-Term Memory
  - Training Strategies
- Transformers: Attention is All You Need
  - Language Transformers
    - > GPT: Generative Pre-Training
  - Vision Transformers

Sequence Modeling

- Modeling sequences for prediction and recognition is ubiquitous
  - Language, time series, video, action trajectories in robotics…
- The key to sequence modeling is capturing the sequential context

boring movie and not great **?** **=** *great movie and not boring*

Speech



Language Model

- A language model (LM) aims at providing a probability distribution over every word, given all the words before it:

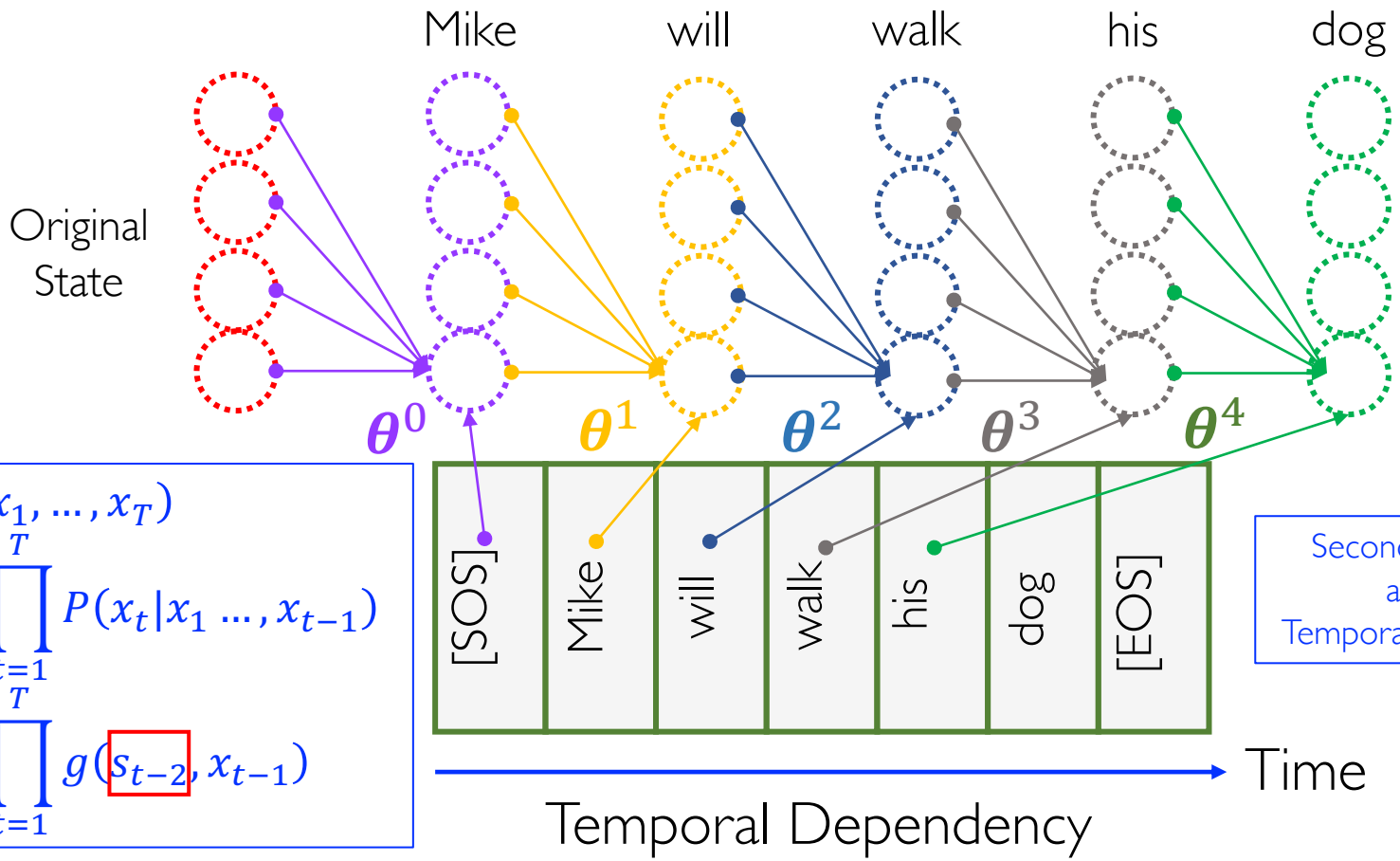
$$P(word_i \mid word_1, \cdots, word_{i-1})$$

- Humans have good sense of what the next word will be:
  - *Deep learning is the study and practice of how we can learn feature representations from large amounts of* ____.

$$P(word_i = \text{“data”} \mid word_1, \cdots, word_{i-1}) = 0.3$$
$$P(word_i = \text{“information”} \mid word_1, \cdots, word_{i-1}) = 0.1$$
$$P(word_i = \text{“hotdogs”} \mid word_1, \cdots, word_{i-1}) = 0.01$$

- How can machines learn useful sequential knowledge from data?

Idea 1: Local Dependency

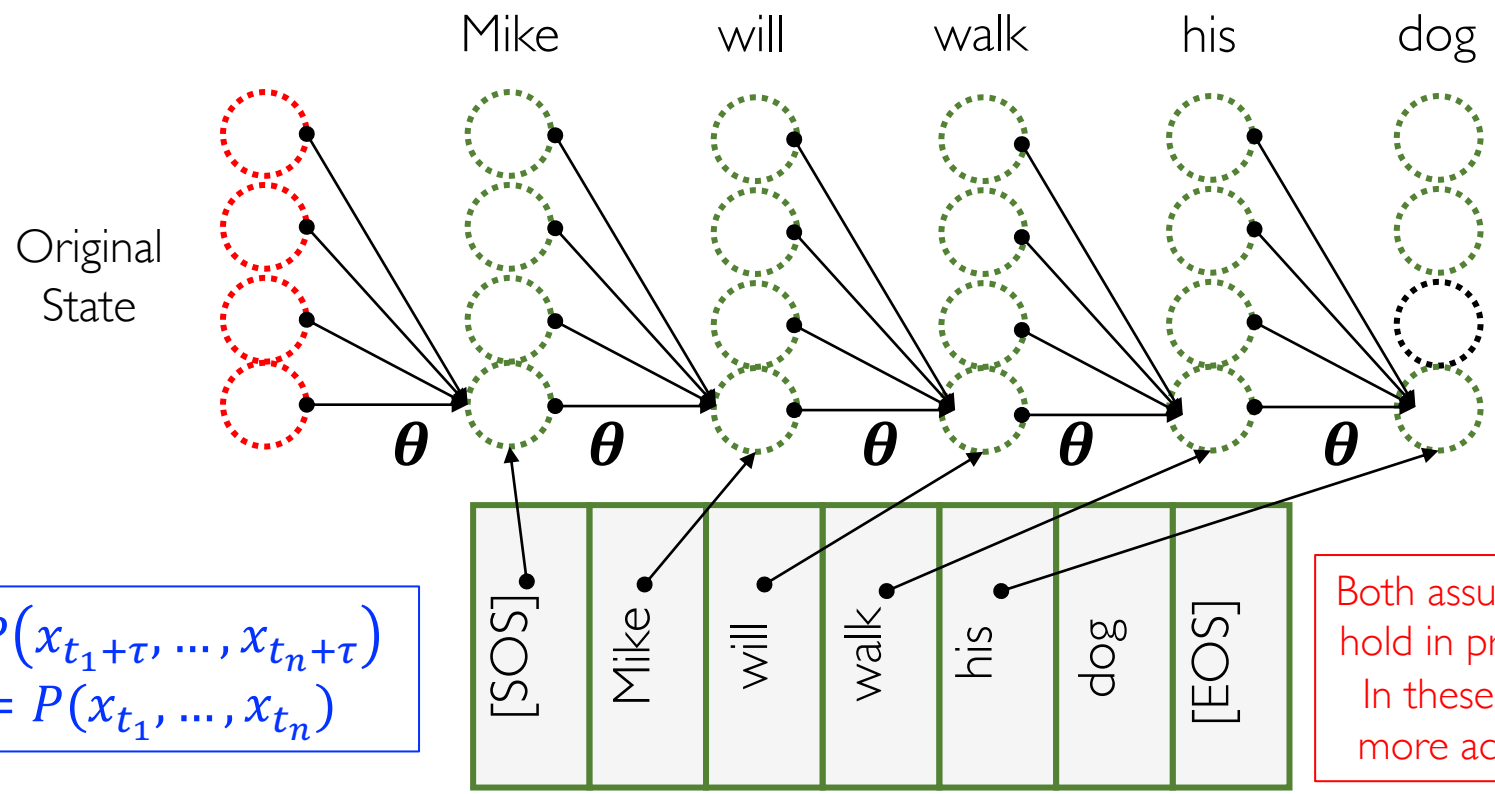


$$P(x_1, \dots, x_T) = \prod_{t=1}^{T} P(x_t | x_1 \dots, x_{t-1}) = \prod_{t=1}^{T} g(s_{t-2}, x_{t-1})$$

Second-order Markov assumption
Temporal receptive field: 2

- **[Local Dependency Assumption]:** The sequential information of all previous timestamps can be encoded into one hidden representation

Idea 2: Parameter Sharing



$$P(x_{t_1+\tau}, \dots, x_{t_n+\tau}) = P(x_{t_1}, \dots, x_{t_n})$$

Both assumptions may not hold in practical problems. In these cases, we need more advanced models.

- **[Temporal Stationarity Assumption]:** If a feature is useful at time t_1, then it should also be useful for all time stamps t_2.
- Thus can reduce the redundant parameters with parameter sharing.

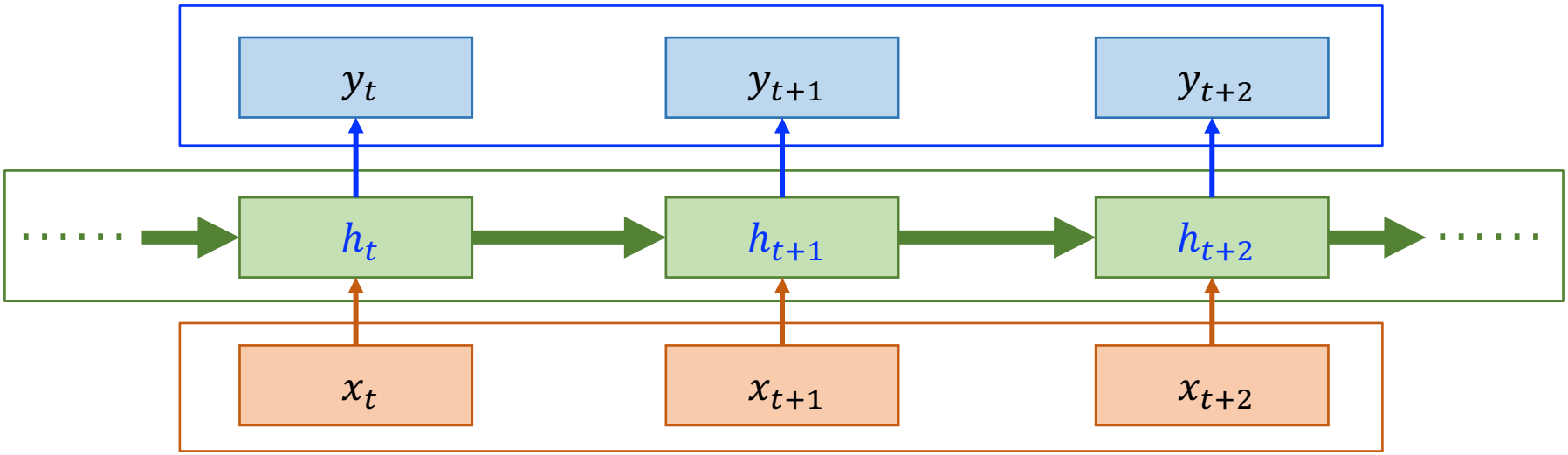
Recurrent Neural Network (RNN)

Prediction | Recognition | Sequence-to-Sequence Modeling | .....

$\{x_t\}$ and $\{y_t\}$ can be different

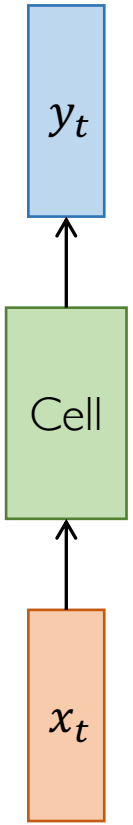
Use different outputs $\{y_t\}|_{t=0}^{T}$ for different tasks

Generate sequential features $\{h_t\}|_{t=0}^{T}$



Process the input sequence $\{x_t\}|_{t=0}^{T}$, where $x_t \in \mathbb{R}^D$ is feature vector

Recurrent Layer



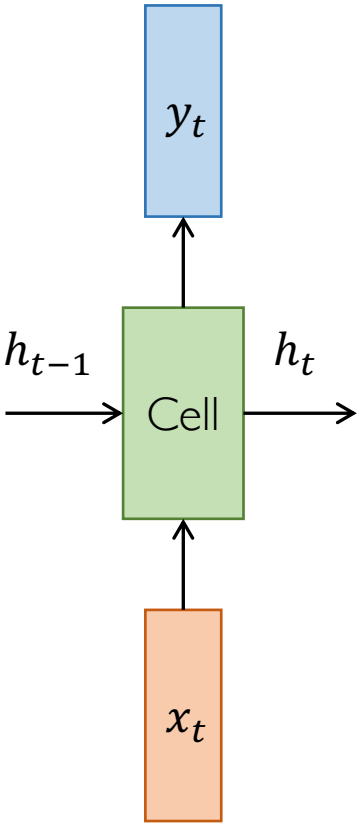
Output

Hidden State

Input

This is a simple MLP.

Recurrent Layer

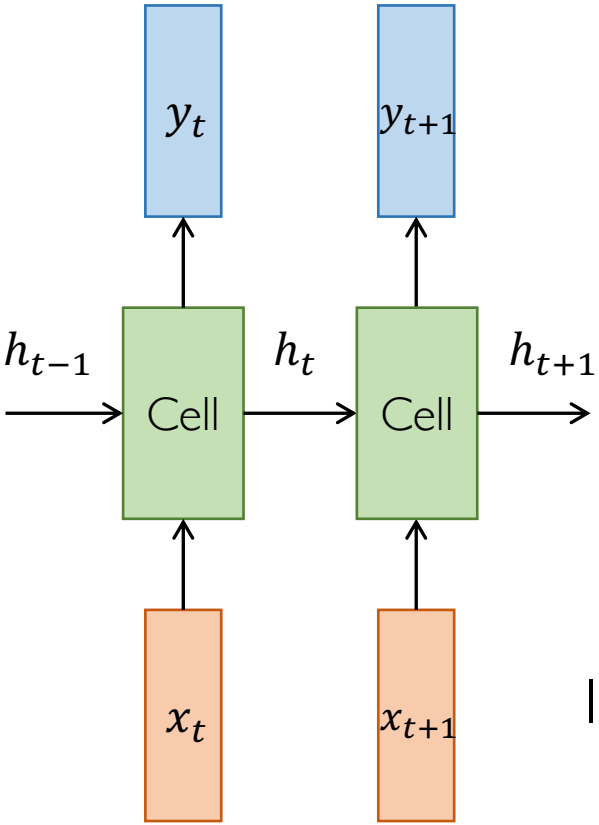


Grows over time…

Hidden State: h_t is used to encode the information from all past timesteps

Implementation of local dependency

Recurrent Layer

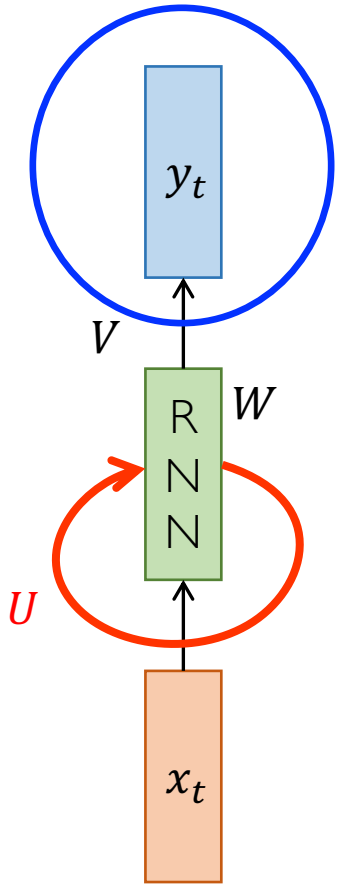


(y_t, y_{t+1}) is a length-2 sequence

Hidden State: h_t is used to encode the information from all past timesteps

Implementation of local dependency

Recurrent Layer



predict a vector at some time steps

Other activations (ReLU) are applicable

$$y_t = Vh_t$$

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

New state

Old state

input vector at some time step

some function with parameters W

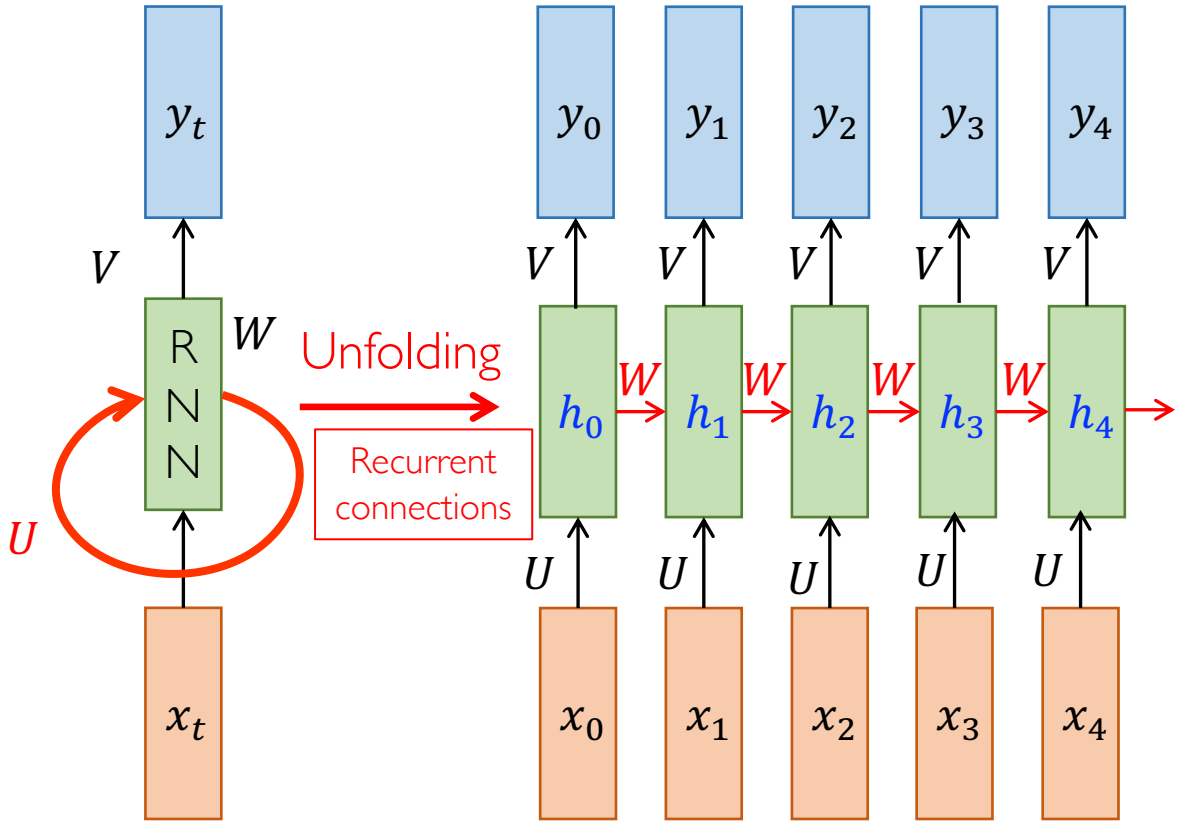
Hidden State: h_t is used to encode the information from all past timesteps

$$h_t = \tanh(Wh_{t-1} + Ux_t)$$

Other activations (ReLU) are applicable

Concat of x_t and h_{t-1}

Unfolding over Time



Note: parameters are not shared over layers

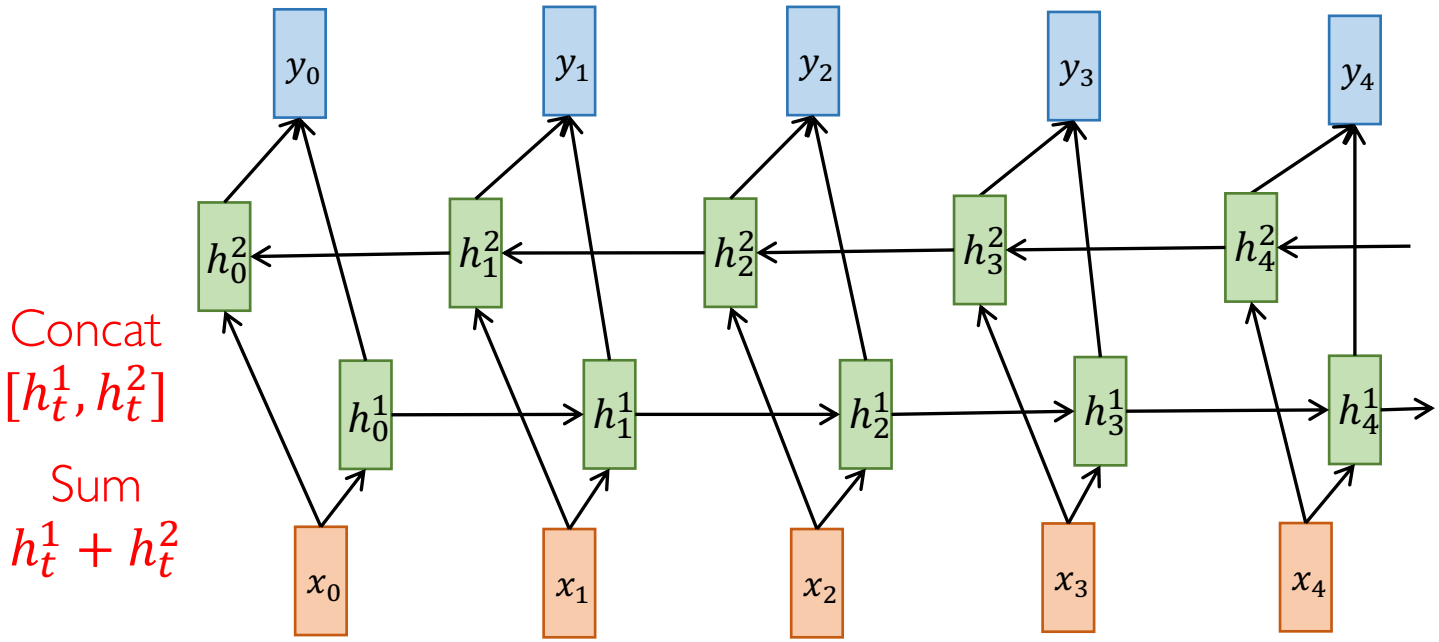
h_t is used to encode the information from all past timesteps

Stationary assumption

We use temporally shared parameters W, U, V

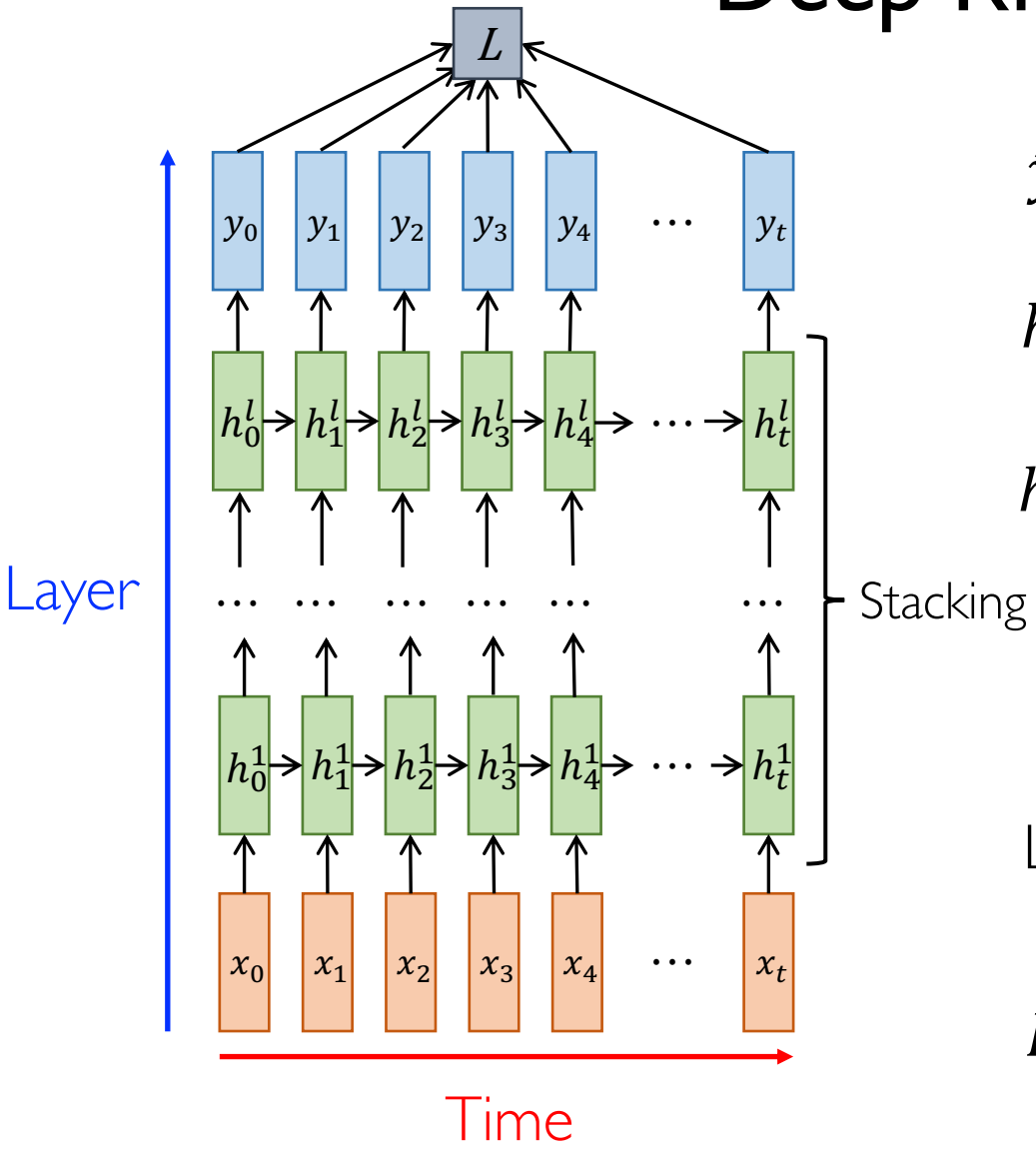
Bidirectional RNN

- When conditioning on a full input sequence, traversing from both directions can enhance the temporal dependency
- Used in sequence classification, e.g. speech recognition



Graves et al. Framewise phoneme classification with bidirectional LSTM. **Neural Networks**, 2005

Deep RNN



$$\hat{y}_t = \text{softmax}(V h_t^l)$$

$$h_t^1 = \tanh(W_1 h_{t-1}^1 + U_1 x_t)$$

……

$$h_t^l = \tanh(W_l h_{t-1}^l + U_l h_t^{l-1})$$

Temporally shared parameters W, U, V (no superscript t)

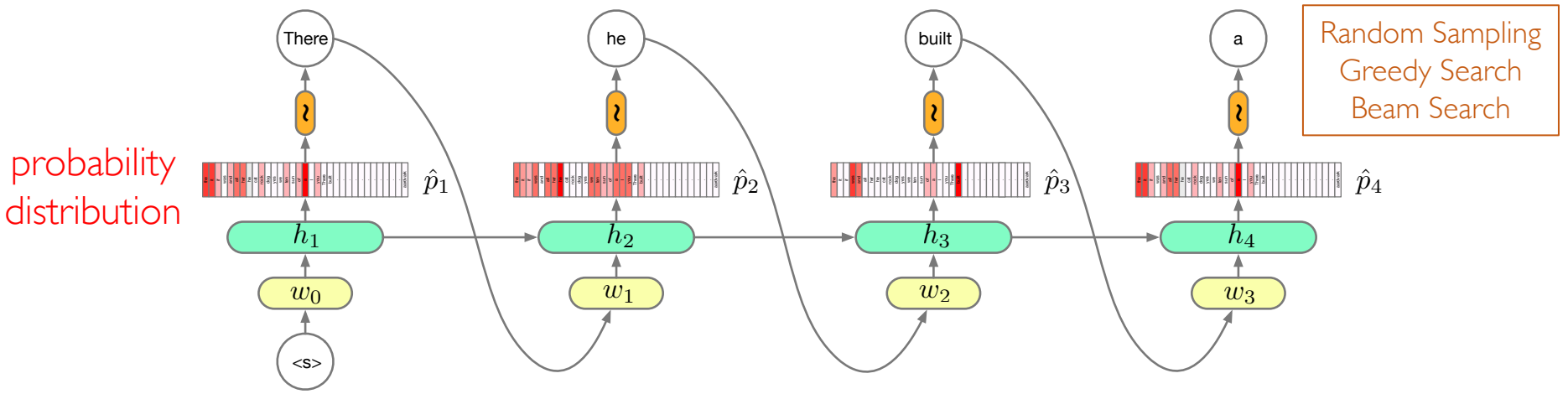
Loss function (cross-entropy):

$$L = -\frac{1}{T}\sum_{t=1}^{T}\sum_{c=1}^{C} y_{t,c}\log(\hat{y}_{t,c})$$

RNN for LM

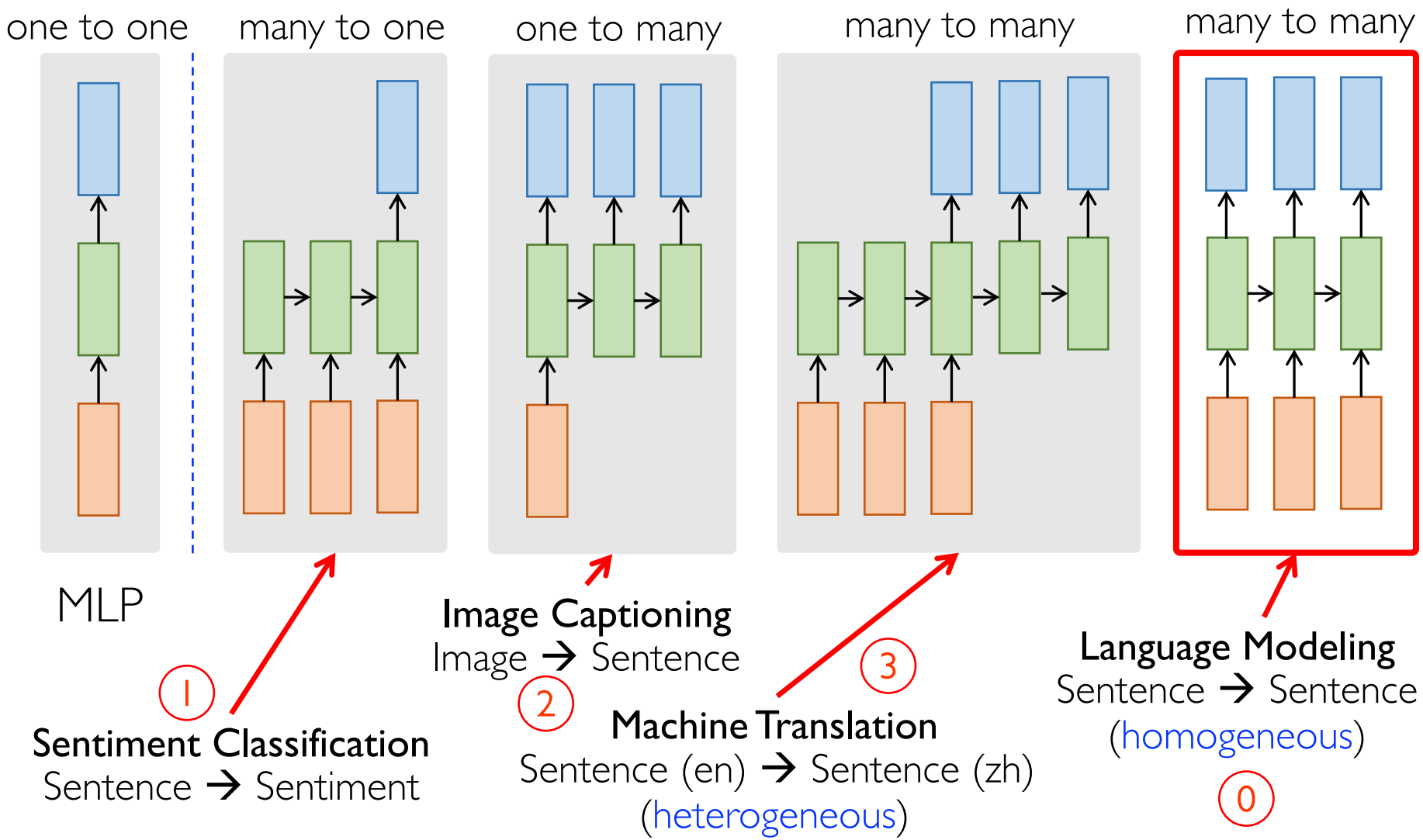
n-gram LM model

$$h_t = g(Wh_{t-1} + Ux_t)$$



- RNNs can represent unbounded temporal dependencies ☺
- RNNs encode histories of words into a fixed size hidden vector ☺
- Parameter size does not grow with the length of dependencies ☺
- RNNs are hard to learn long range dependencies present in data

Standard Architectures



Outline

- Recurrent Neural Network (RNN)
  - **LSTM: Long Short-Term Memory**
  - Training Strategies
- Transformers: Attention is All You Need
  - Language Transformers
    - > GPT: Generative Pre-Training
  - Vision Transformers

Long Short-Term Memory (LSTM)

Difficulty of learning is caused by interrupted gradient flow in RNNs

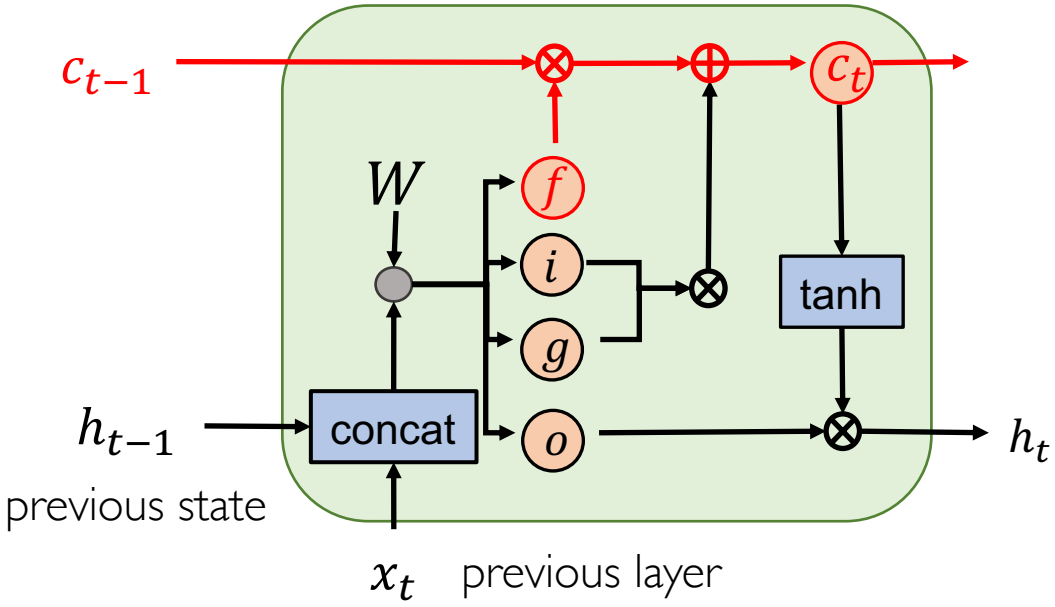
J. Schmidhuber

How to Improve Gradient Flow?

Hochreiter and J. Schmidhuber. Long short-term memory,1995 (Cited 79717)

Long Short-Term Memory (LSTM)

Improve Gradient Flow



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

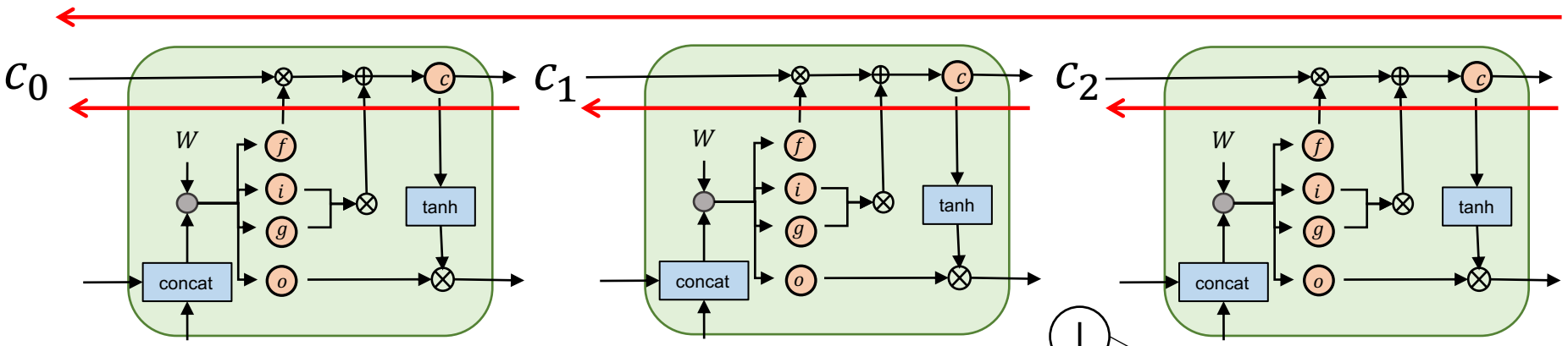
f: **Forget gate**, Whether to erase cell
i: **Input gate**, whether to write to cell
g: **Gate gate**, How much to write to cell
o: **Output gate**, How much to reveal cell

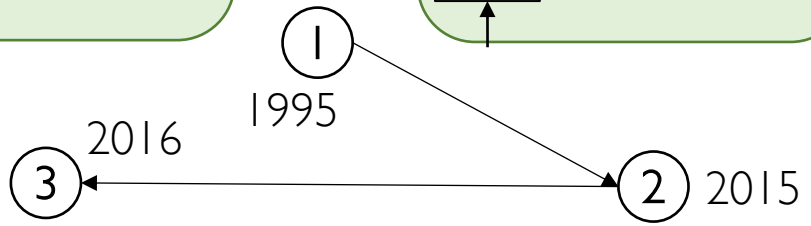
Hochreiter and J. Schmidhuber. Long short-term memory,1995 (Cited 79717)

LSTM: Gradient Flow

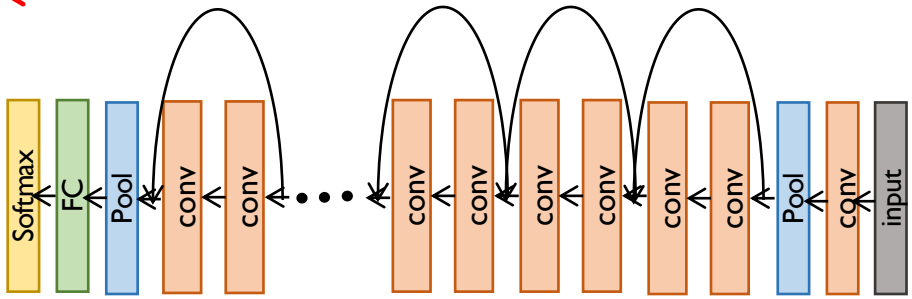
Gradient Flow Highway

Back-propagate from c_t to c_{t-1} via only element-wise multiplication, no matrix multiply.





ResNet motivated from Highway Network



Highway Network motivated from LSTM.

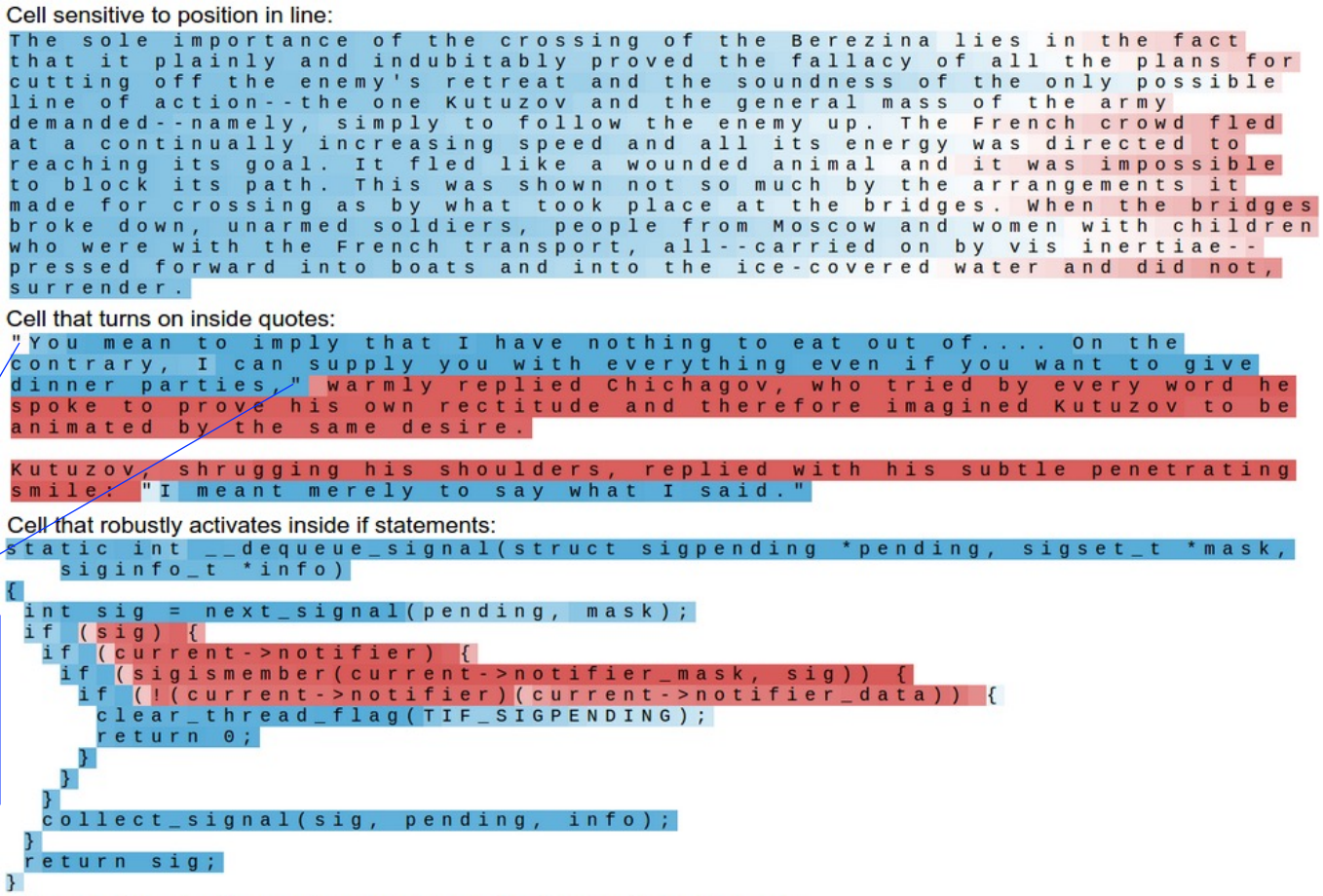
$$g = T(x, W_T)$$

$$y = g \odot H(x, W_h) + (1 - g) \odot x$$

Schmidhuber et al. Highway Networks. **NIPS** 2015

Visualization: Character LM

- Visualization of some cell activation: $\tanh(c)$
  - red for -1
  - blue for +1
- Line counter
- Track quotes
- "If" statements



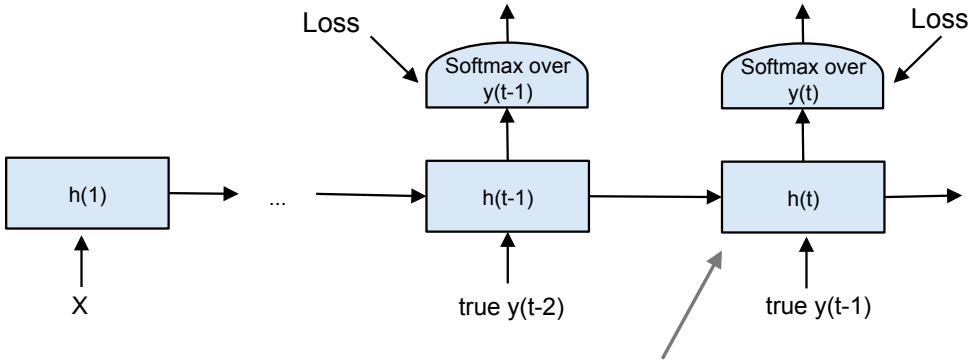
Long-Term Dependencies! (more than 100+)

Karpathy et al. "Visualizing and understanding recurrent networks" 2015

Outline

- Recurrent Neural Network (RNN)
  - LSTM: Long Short-Term Memory
  - **Training Strategies**
- Transformers: Attention is All You Need
  - Language Transformers
    - > GPT: Generative Pre-Training
  - Vision Transformers

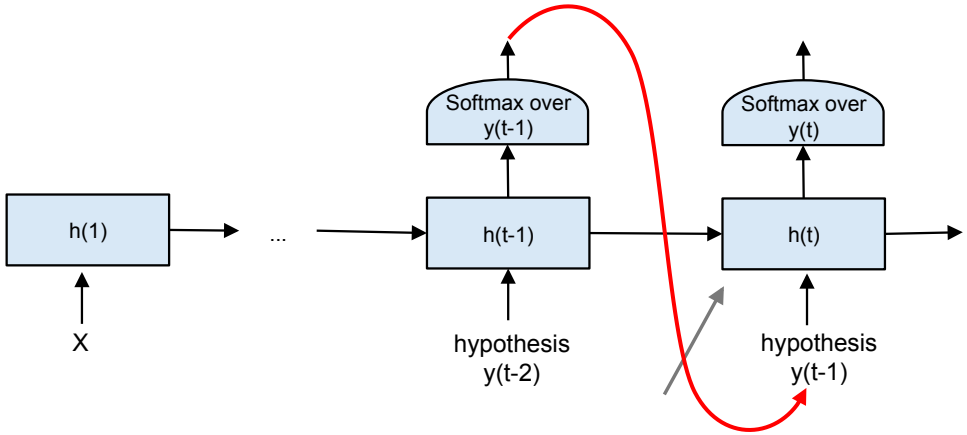
Shift in Training & Inference



Training

$$P(y_t|h_t)$$
$$h_t = f(h_{t-1}, y_{t-1}; \theta)$$

Sampling $\hat{y}_t$ from Softmax at time $(t-1)$



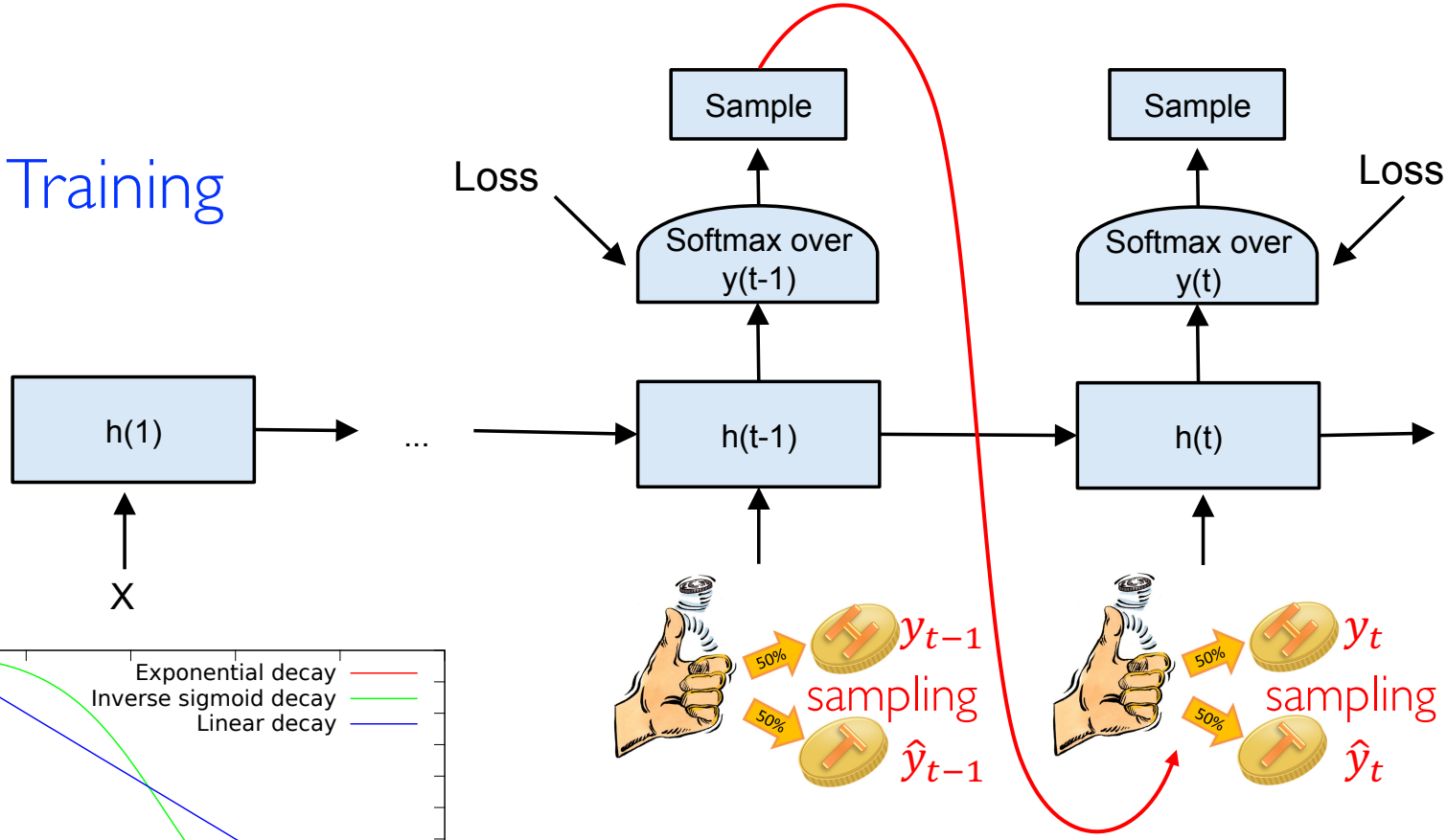
difficulty

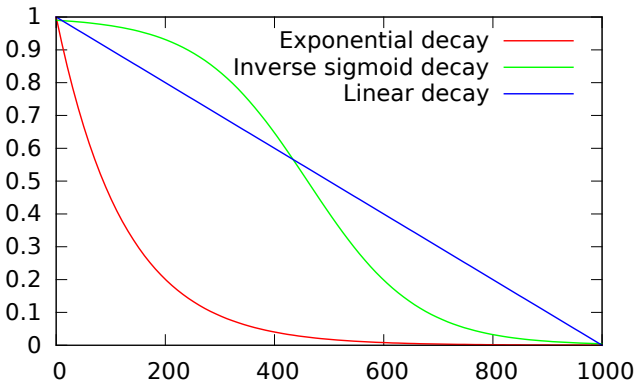
Inference

$$P(y_t|h_t)$$
$$h_t = f(h_{t-1}, \hat{y}_{t-1}; \theta)$$

Discrepancy between training and inference $\hat{y}_{t-1} \neq y_{t-1} \rightarrow$ mistakes accumulate quickly.

Scheduled Sampling

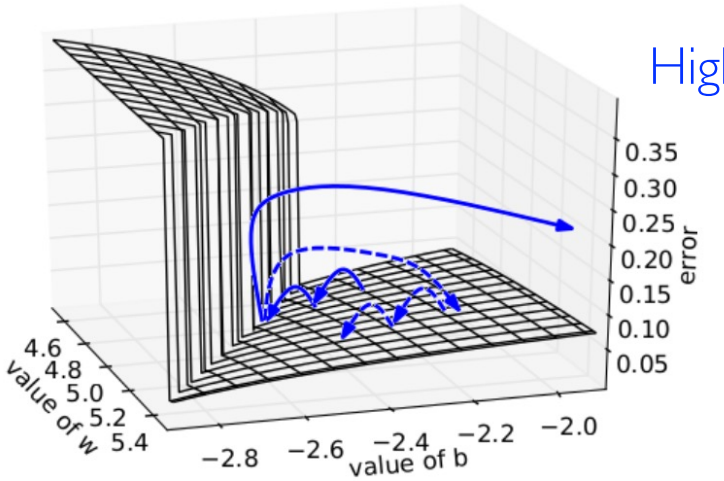




- Scheduled sampling rate (exponential, linear…)
- Curriculum learning: from $\boldsymbol{y_t}$ (easy) to $\hat{\boldsymbol{y}}_{\boldsymbol{t}}$ (hard)

Bengio et al. Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks. *NIPS 2015*.

Gradient Clipping



High-curvature walls → exploding gradient

Algorithm 1 Pseudo-code for norm clipping

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$

if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**

$\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|}\hat{\mathbf{g}}$

end if

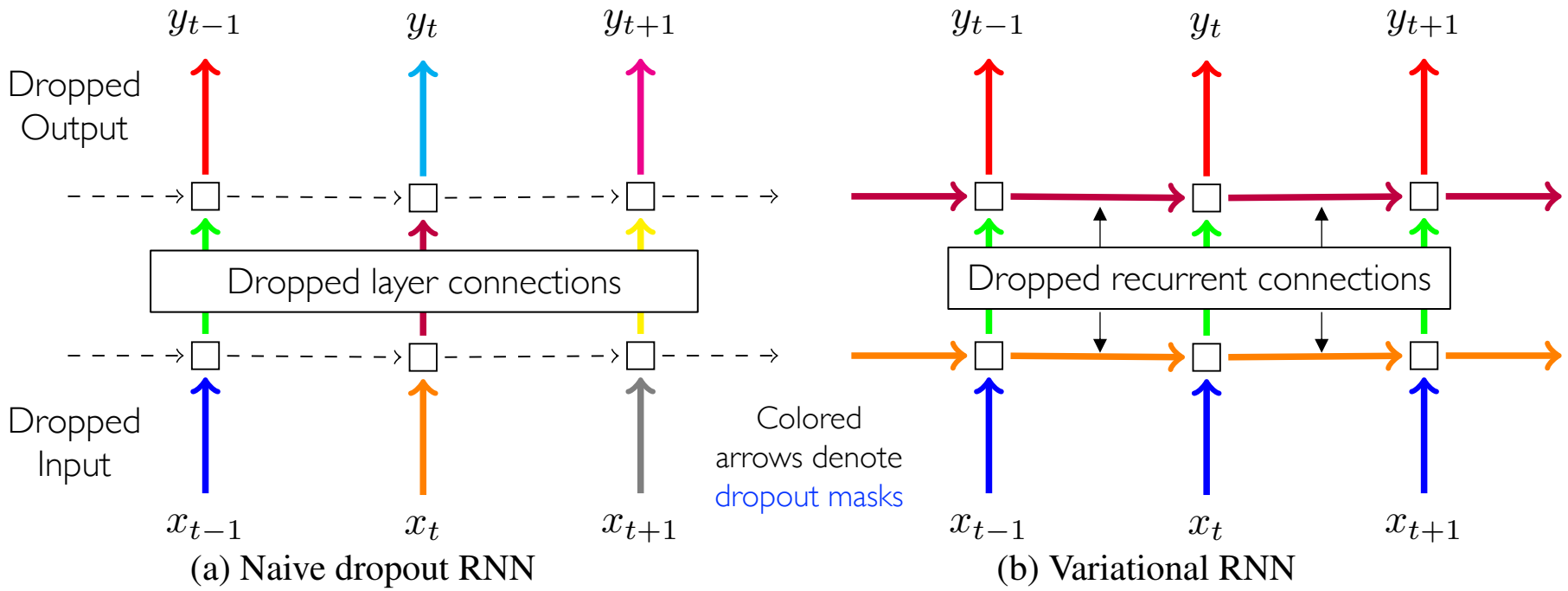
Set to average of gradient norms

- The error surface of a single hidden unit RNN
  - The existence of high-curvature walls gives rise to training difficulty
- Solid lines: standard gradient descent trajectories (diverge!)
- Dashed lines: gradients rescaled to fixed size when its norm is above a threshold

Bengio et al. "Learning long-term dependencies with gradient descent is difficult." IEEE TNN, 1994 (Cited 8755)
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013 (Cited 5466)

Variational Dropout

- Mitigate overfitting
  - Tie the recurrent dropout mask and sampling at evaluation time
  - Implementation: same network units dropped at each time step, randomly dropping inputs, outputs, and recurrent connections.



Gal et al. A Theoretically Grounded Application of Dropout in RNN. NIPS, 2016 (Cited 1667)

Layer Normalization

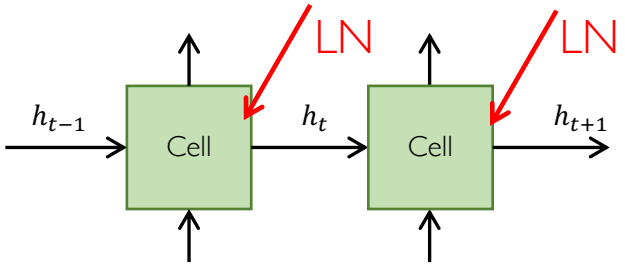
Recap: Batch Normalization (BN)

$$\bar{a}_i^l = \frac{g_i^l}{\sigma_i^l}(a_i^l - \mu_i^l) \qquad \mu_i^l = E_{x\sim P(x)}[a_i^l] \qquad \sigma_i^l = \sqrt{E_{x\sim P(x)}\left[(a_i^l - \mu_i^l)^2\right]}$$

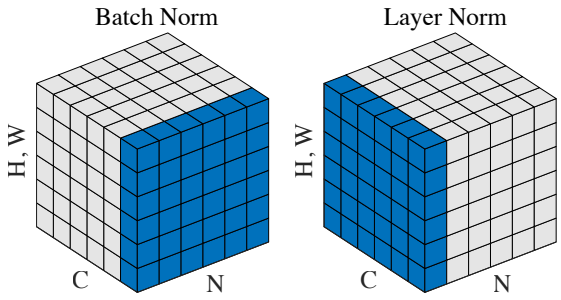
Layer normalization (LN) in RNNs

$$h^t = f\left[\frac{g}{\sigma^t} \odot (a^t - \mu^t) + b\right] \qquad \mu^t = \frac{1}{C}\sum_{i=1}^{C} a_i^t \qquad \sigma^t = \sqrt{\frac{1}{C}\sum_{i=1}^{C}(a_i^t - \mu^t)^2}$$

- $\boldsymbol{g}$ and $\boldsymbol{b}$ are shared over all time-steps, the statistics are computed across each feature (channel) and are independent of the batch size
- After multiplying inputs and states by weights but before nonlinearity



C: channel dim
N: example dim
(H, W): spatial dims



Ba et al. Layer normalization. *NIPS, 2016.*

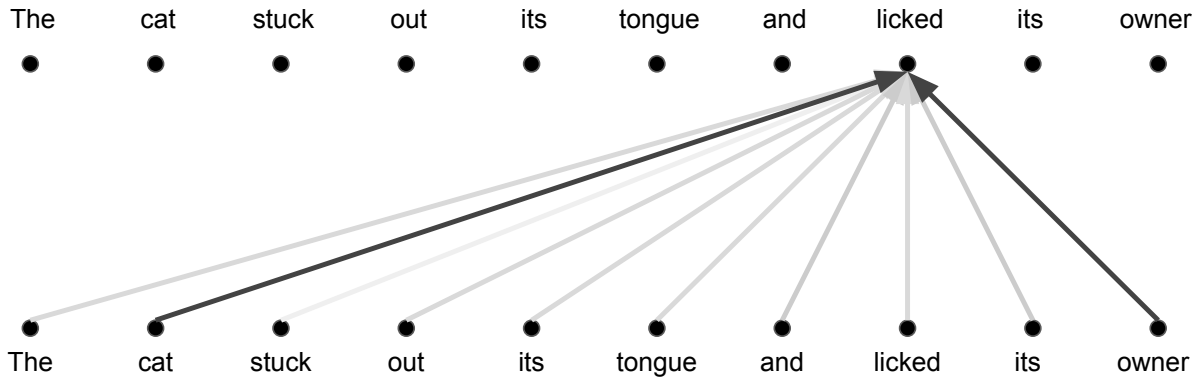
Outline

- Recurrent Neural Network (RNN)
  - LSTM: Long Short-Term Memory
  - Training Strategies
- **Transformers: Attention is All You Need**
  - Language Transformers
    - > GPT: Generative Pre-Training
  - Vision Transformers

Self-Attention

- In RNN, we need attention modules!
- Sure. With attention, we even don't need RNN.

- What is self-attention?
  - Matrix form of the key $K = [k_1, \ldots, k_n]$, query Q and value V:
  - Self-attention: K, Q, V are from the same place.
    - »Compute weights with each others in a large whole.



Vaswani et al. Attention is all you need. *NIPS 2017*. (Cited 70781)

Solution: Transformer (Trm)

- Main Components:
  - Scaled Dot-Product Attention
  - (Masked) Multi-Head Attention
  - Position-wise FFN
  - Residual Connections
  - Layer Normalization

  Self-attention blocks
- Encoder-decoder architecture
  - Positional encoding
  - Masking in decoder
  - Stack $N = 6$ identical self-attention blocks for encoder and decoder

Scaled Dot-Product Attention

- Recall $e_{ij} = v_a^\top \tanh\left(W_a s_{i-1} + U_a h_j\right)$: too complex!
- Given query q (to match others) and key k (to be matched)
  - **Bilinear**

$$a(q,k) = q^T W k$$

Require parameters

  - **Dot Product**

$$a(q,k) = q^T k$$

Increase as dimensions get larger

  - **Scaled Dot Product**

Dimension of q and k

$$a(q,k) = q^T k / \sqrt{d_k}$$

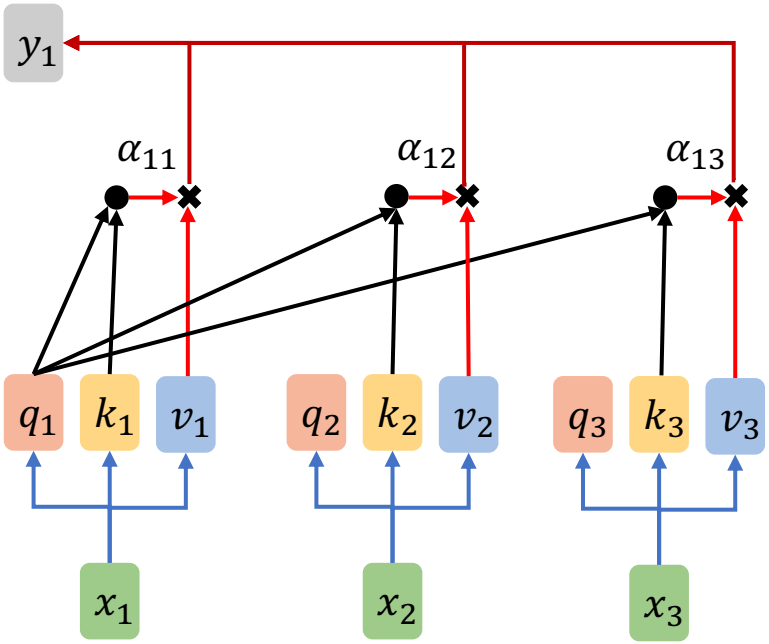
Scale by size of the vector

- For large values of d_k the dot products grow large in magnitude:
  - Cause small gradients after Softmax → Scaled dot product
  - Assume independent q_i, k_i with mean 0 and variance 1 mean, dot product $q^T k = \sum_{i=1}^{d_k} q_i k_i$ has mean 0 and variance d_k.

Scaled Dot-Product Attention

$$y_1 = \sum_i \alpha_{1i} v_i \qquad \alpha_{1i} = \frac{\exp(q_1 k_i)}{\sum_j \exp(q_1 k_j)}$$

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

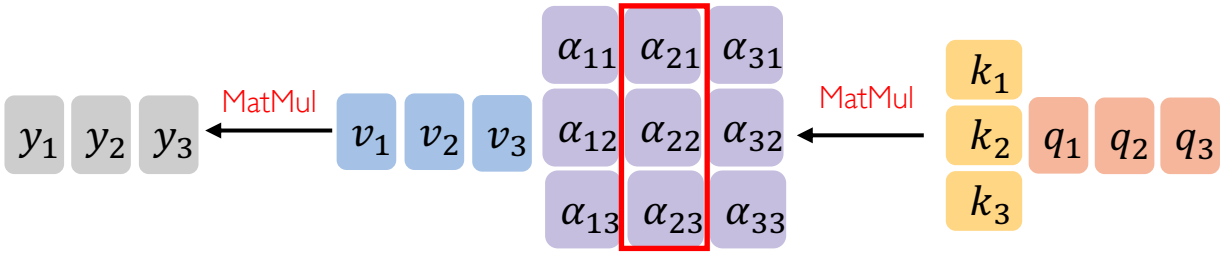


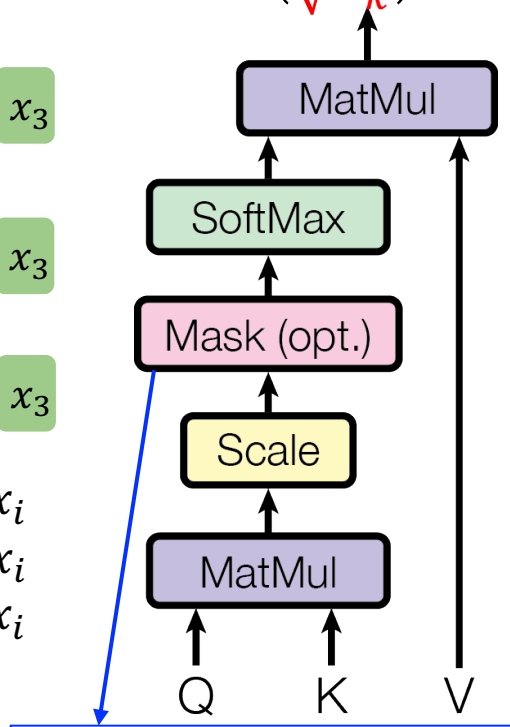
$$q_1\ q_2\ q_3 = W^q\ x_1\ x_2\ x_3 \quad Q$$

$$k_1\ k_2\ k_3 = W^k\ x_1\ x_2\ x_3 \quad K$$

$$v_1\ v_2\ v_3 = W^v\ x_1\ x_2\ x_3 \quad V$$

$$q_i = W^q x_i$$
$$k_i = W^k x_i$$
$$v_i = W^v x_i$$





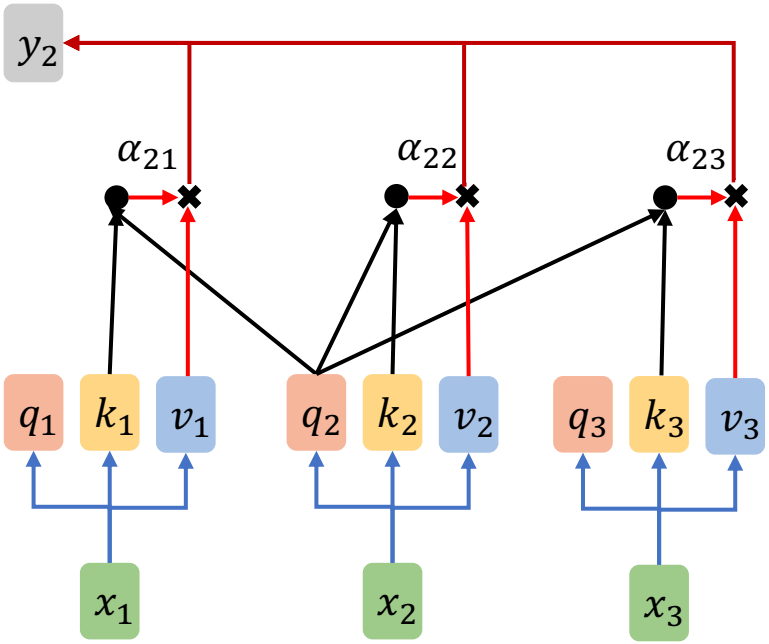
Mask will be discussed later

Ignore $\sqrt{d_k}$ for simplicity

Scaled Dot-Product Attention

$$y_1 = \sum_i \alpha_{1i} v_i \qquad \alpha_{1i} = \frac{\exp(q_1 k_i)}{\sum_j \exp(q_1 k_j)}$$

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

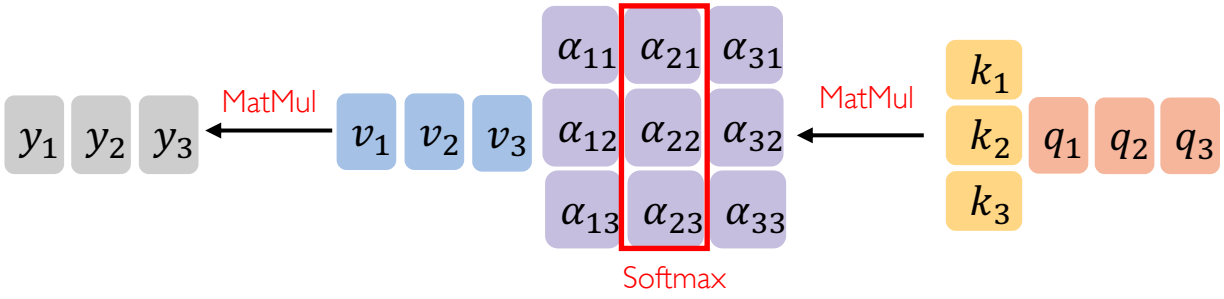


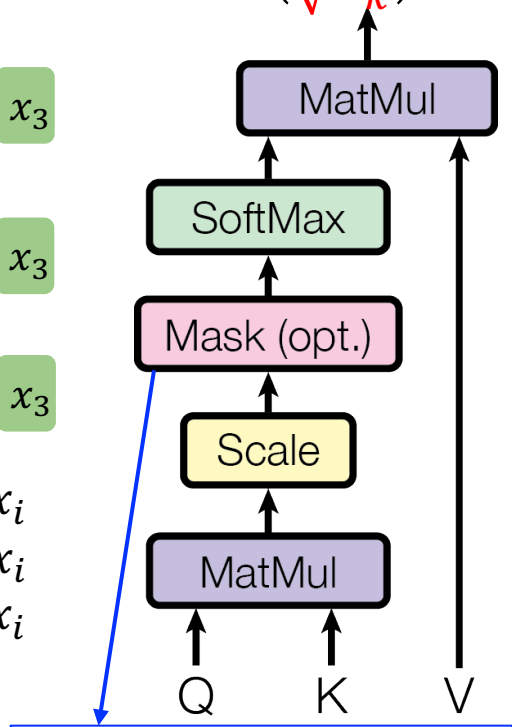
$$q_1\ q_2\ q_3 = W^q\ x_1\ x_2\ x_3 \quad Q$$

$$k_1\ k_2\ k_3 = W^k\ x_1\ x_2\ x_3 \quad K$$

$$v_1\ v_2\ v_3 = W^v\ x_1\ x_2\ x_3 \quad V$$

$$q_i = W^q x_i$$
$$k_i = W^k x_i$$
$$v_i = W^v x_i$$





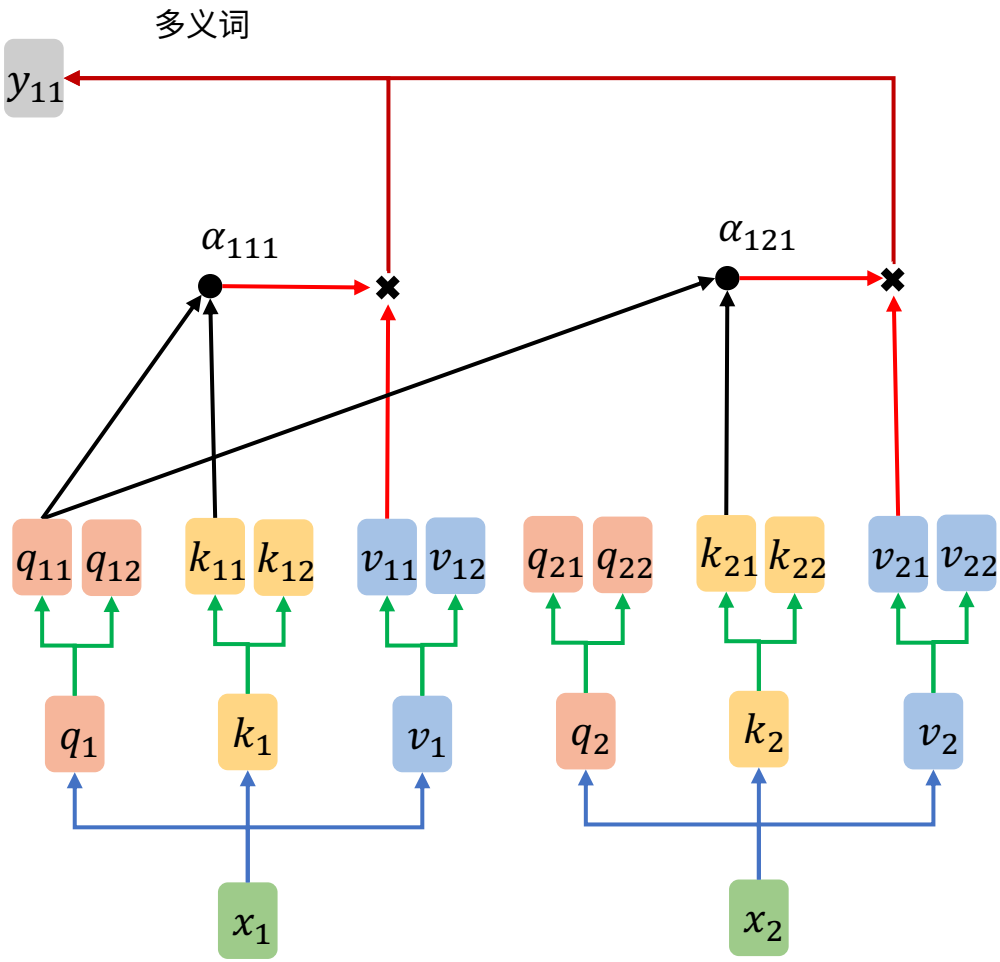
Mask will be discussed later

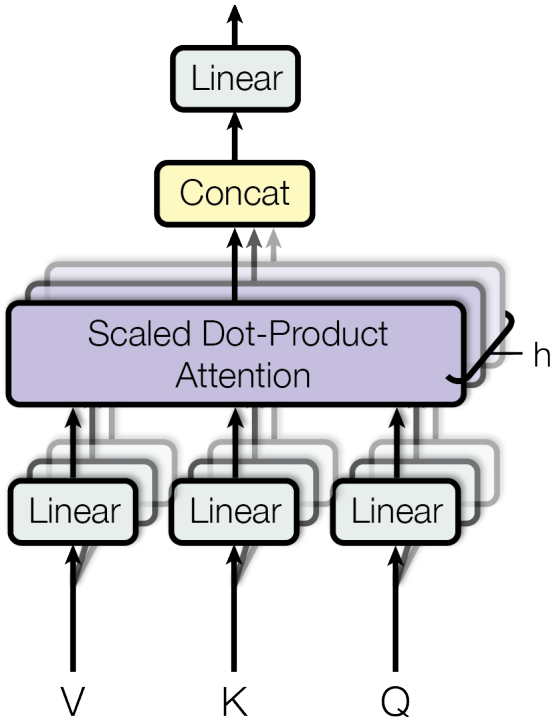
Ignore $\sqrt{d_k}$ for simplicity

Multi-Head Attention

Expressivity

$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1,\dots,\text{head}_h)W^O$$
$$\text{where head}_i = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right)$$





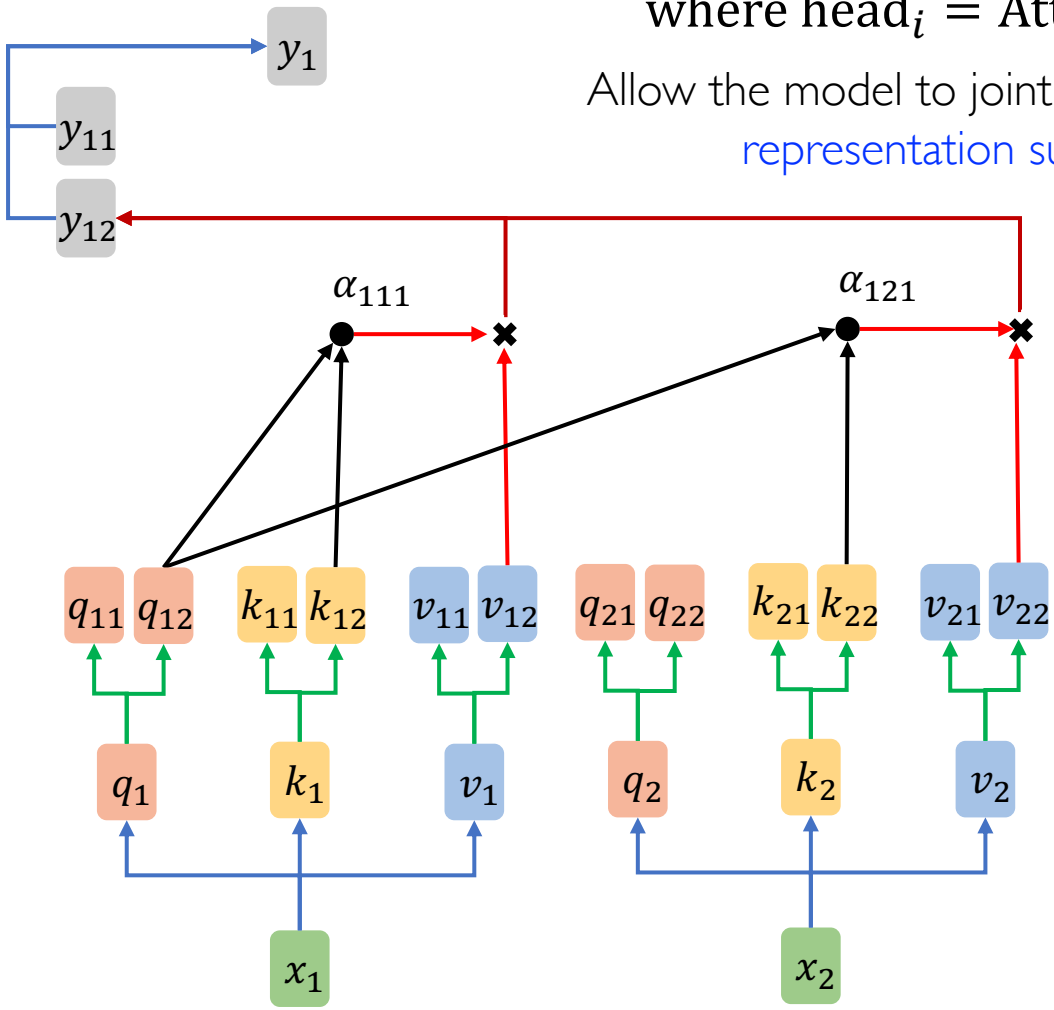
Similar to #Channels in CNNs

Multi-Head Attention

Expressivity

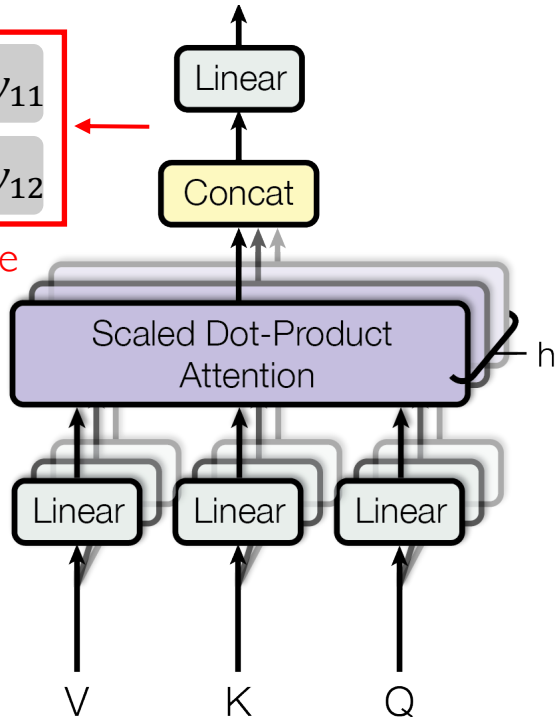
$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1,\dots,\text{head}_h)W^O$$
$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Allow the model to jointly attend to information from different representation subspaces at different positions.



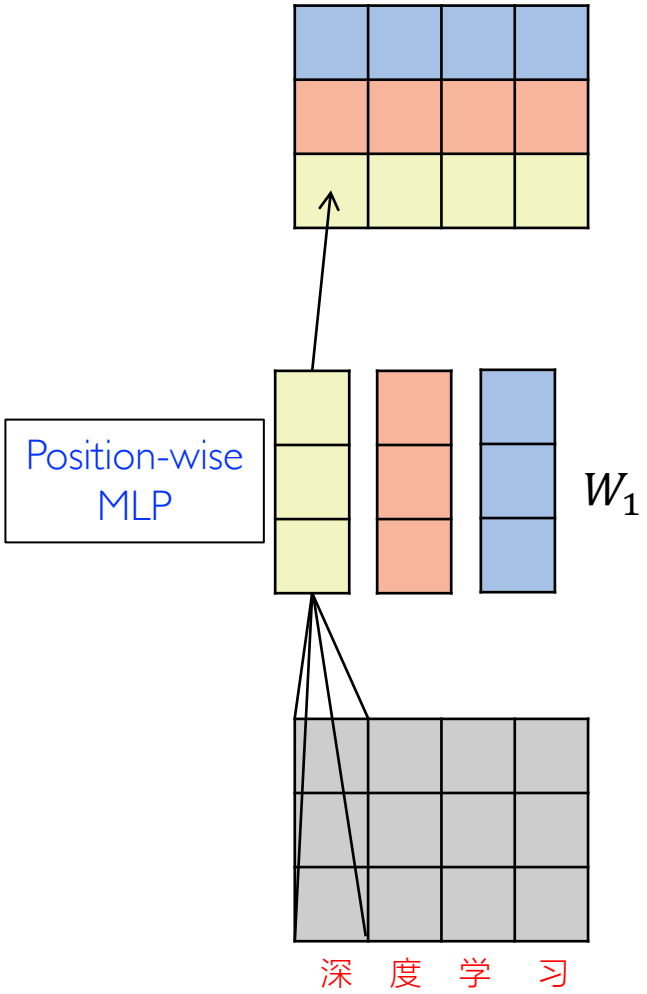
$$y_1 = W^0 \begin{matrix} y_{11} \\ y_{12} \end{matrix}$$

Maintain shape



Similar to #Channels in CNNs

Position-wise FFN



- Apply a two-layer position-wise MLP
  - Consisting of two linear transformations with a ReLU activation in between

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

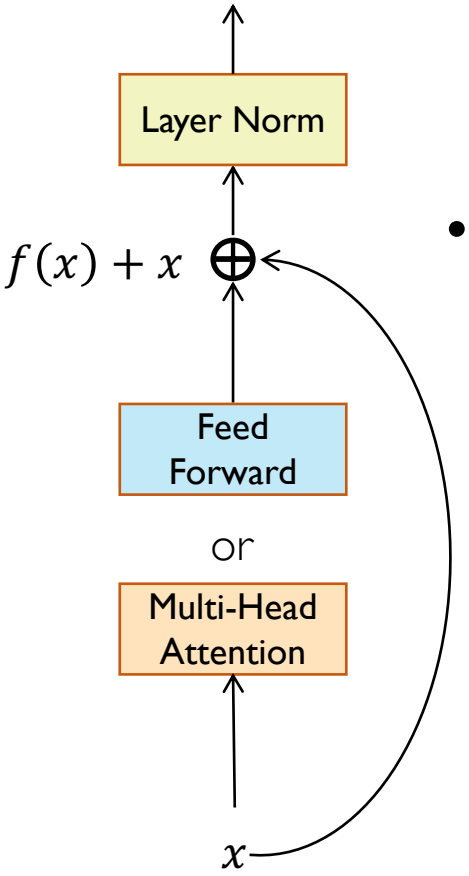
- Equivalent to apply two 1D conv layers

(Recall point-wise convolution)

  - Parameter are sharing among all words
  - Applied to sequences with arbitrary length

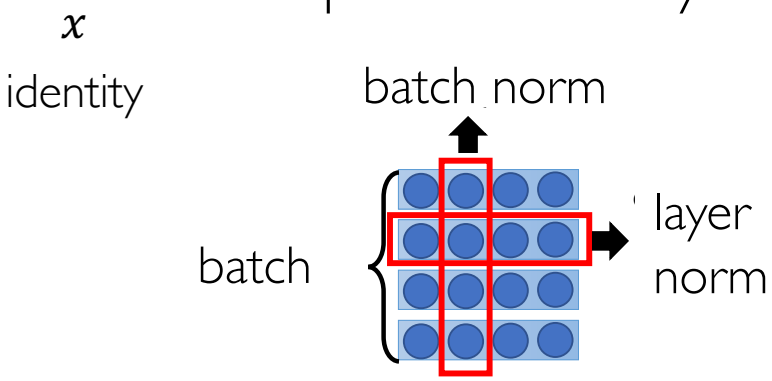
只学习单个词的特征

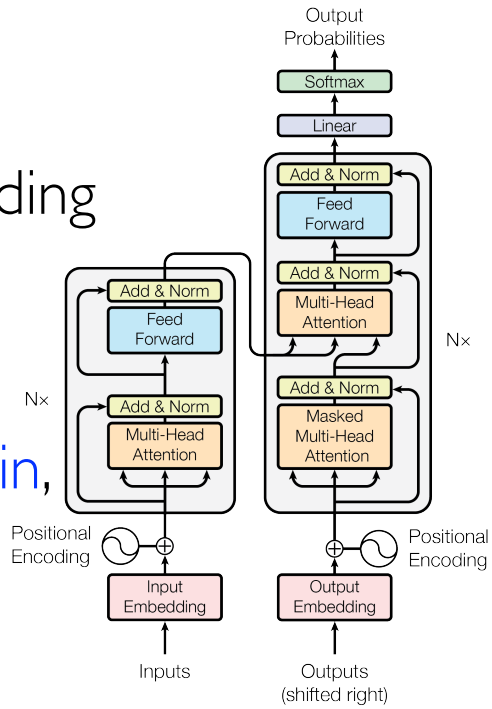
Residual Connections & Layer Normalization



Residual Connections

- **Residual Connections:**
  - Stronger gradients
  - Help remember positional encoding
- **Layer Normalization**
  - Covariate shift ?
  - Center embeddings around origin, which helps attention layers





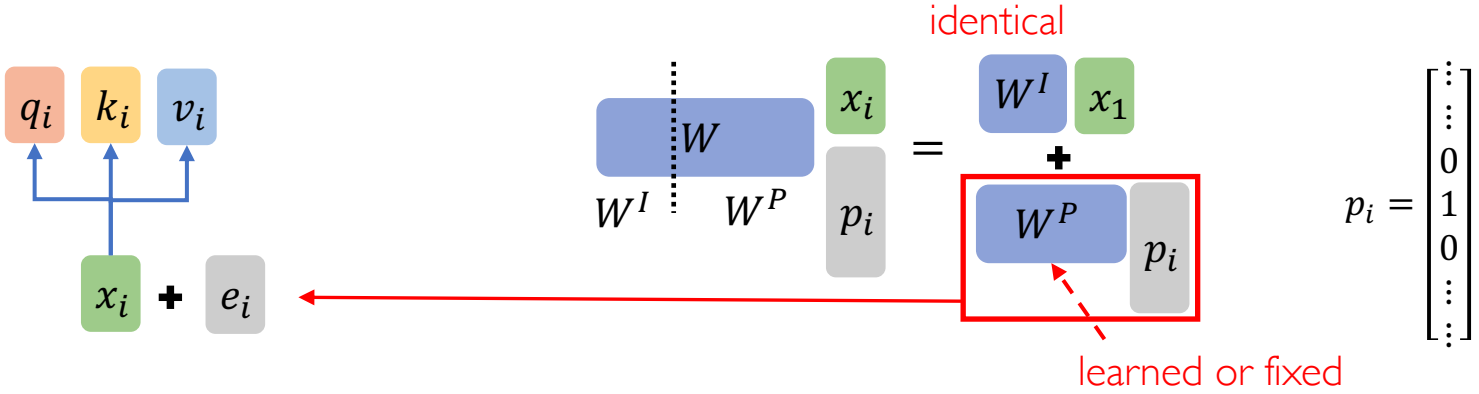
performs exactly the same computation at training and test times

So we can stack many self-attention blocks! 😊

Positional Encoding

- No position information in self-attention!

Inject some information about the relative or absolute position of the tokens in the sequence!



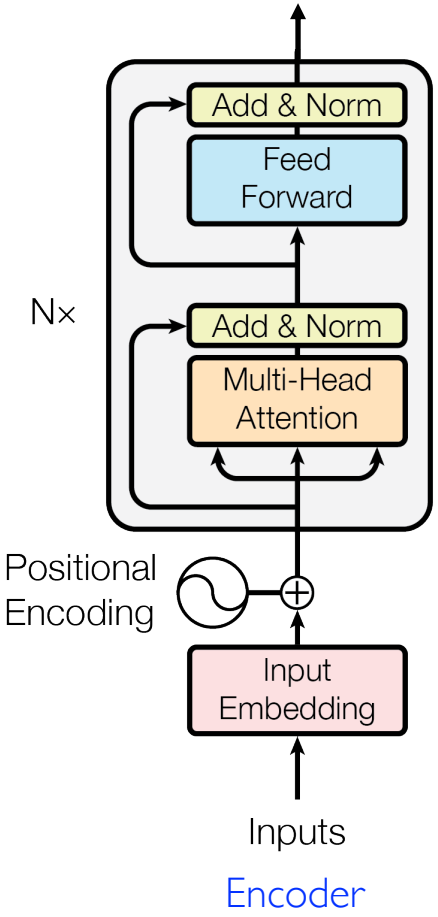
- In original paper: each position has a unique positional vector e_i (not learned from data)

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$

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

pos is the position and i is the dimension

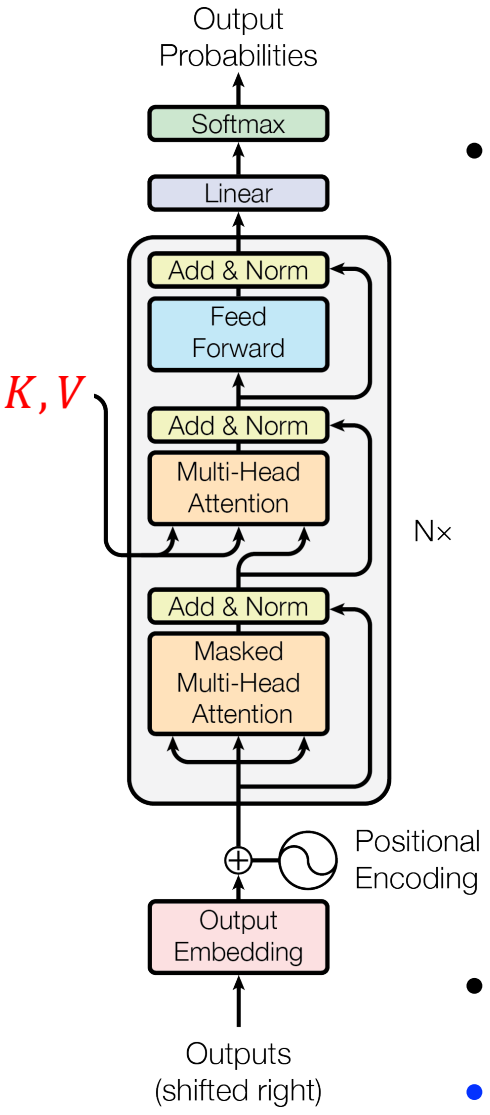
Encoder Architecture



- The encoder stacks N ($N = 6$ in original paper) encoder blocks
  - Multi-head self-attention
  - Position-wise FFN
  - Perform positional encoding at the bottoms of the encoder
- Each block maintains shape (#vectors, vector length)
- Potential weakness
  - Complexity is $O(n^2)$, where n is sequence length
  - Can be difficult to train
  - Many heads are unimportant and can be pruned with little impact on performance

Efficient transformer?

Decoder Architecture



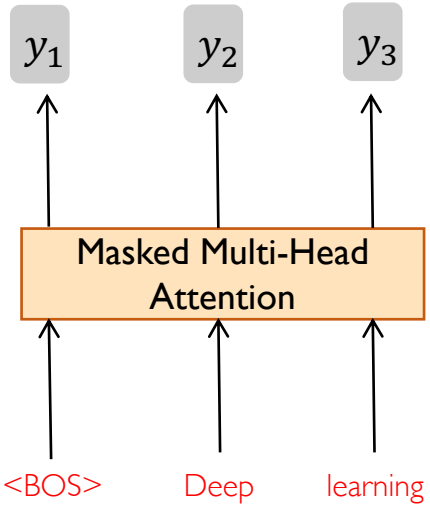
- The encoder stacks N ($N = 6$ in original paper) decoder blocks
  - Double multi-head self-attention layers
    - »Masked multi-head self-attention layer
    - »K and V in the second multi-head self-attention layer are from encoder outputs.
  - Position-wise FFN
  - Perform positional encoding at the bottoms of the encoder
- Each block maintains shape (#vectors, vector length)
- How decoder works during testing and training?

Masked Multi-Head Self-Attention

Mask is used in the decoder to keep it autoregressive

Train self-attention blocks in parallel, and avoid the model cheating!

$$q_i = W^q x_i$$
$$k_i = W^k x_i$$
$$v_i = W^v x_i$$

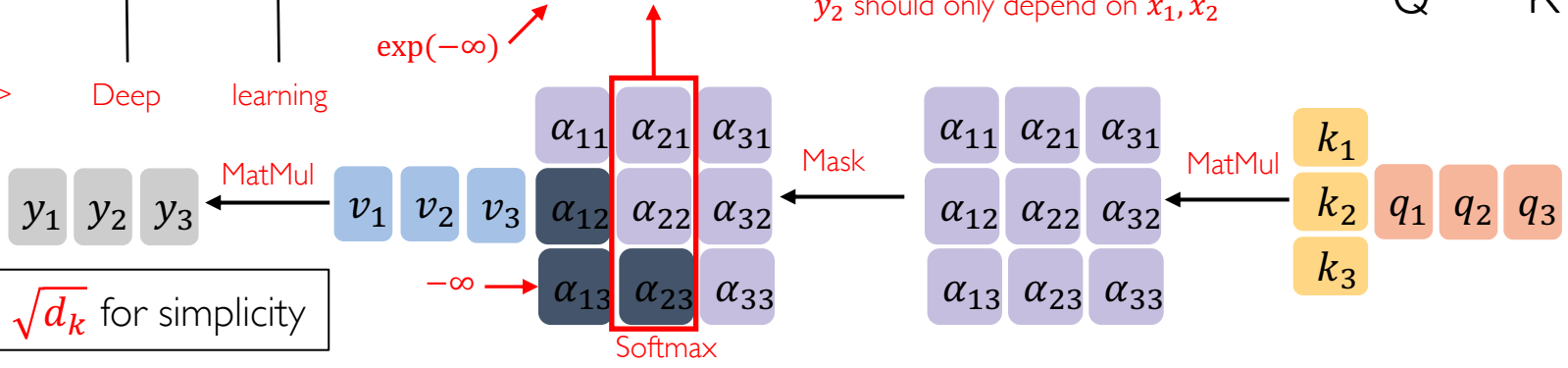


$$y_1 = \sum_i \alpha_{1i} v_i \qquad \alpha_{1i} = \frac{\exp(q_1 k_i)}{\sum_j \exp(q_1 k_j)}$$

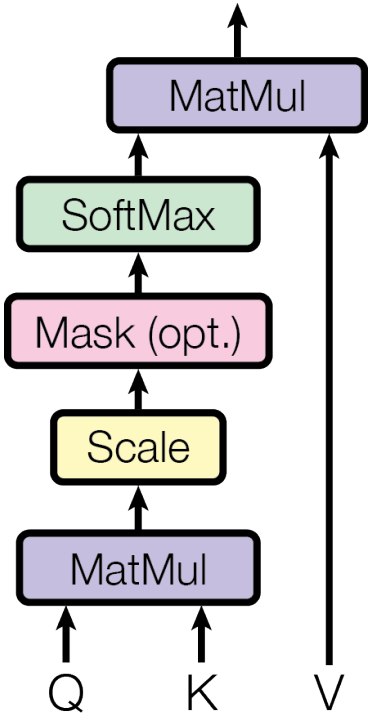
$$\alpha_i = \begin{bmatrix} \exp(\alpha_{21}) \\ \exp(\alpha_{22}) \\ 0 \end{bmatrix} \frac{1}{\exp(\alpha_{21}) + \exp(\alpha_{22}) + 0}$$

$\exp(-\infty)$

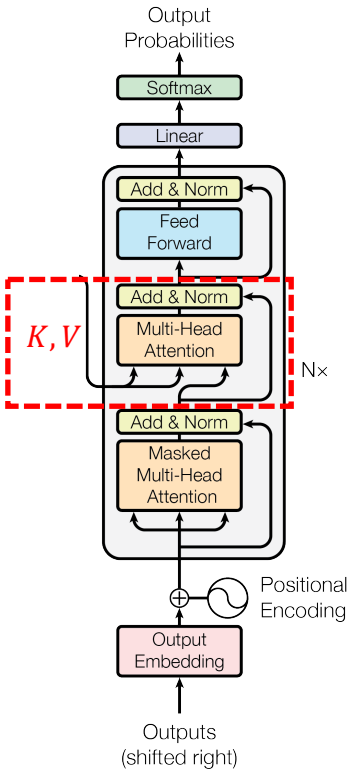
y_2 should only depend on x_1, x_2



Ignore $\sqrt{d_k}$ for simplicity



Encoder-Decoder Self-Attention



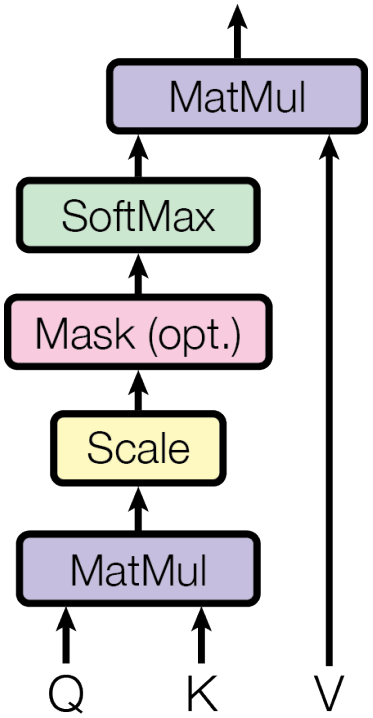
- Encoder-decoder self-attention takes two inputs:
  - X^e: $d^e \times \text{len}_e$ (encoder outputs)
  - X^d: $d^d \times \text{len}_d$ (decoder inputs)

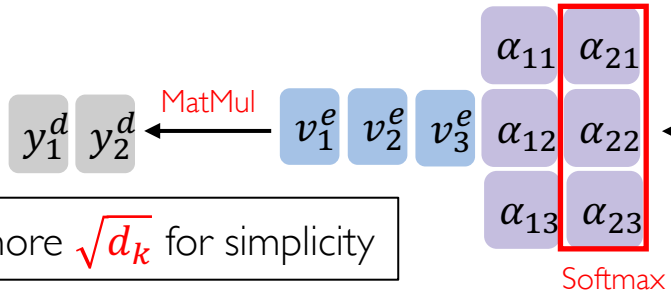
$$q_i^d = W^q x_i^d$$
$$k_i^e = W^k x_i^e$$
$$v_i^e = W^v x_i^e$$

$$\underbrace{q_1^d \; q_2^d}_{Q^d} = W^q \; x_1^d \; x_2^d$$

$$\underbrace{k_1^e \; k_2^e \; k_3^e}_{K^e} = W^k \; x_1^e \; x_2^e \; x_3^e$$

$$\underbrace{v_1^e \; v_2^e \; v_3^e}_{V^e} = W^v \; x_1^e \; x_2^e \; x_3^e$$

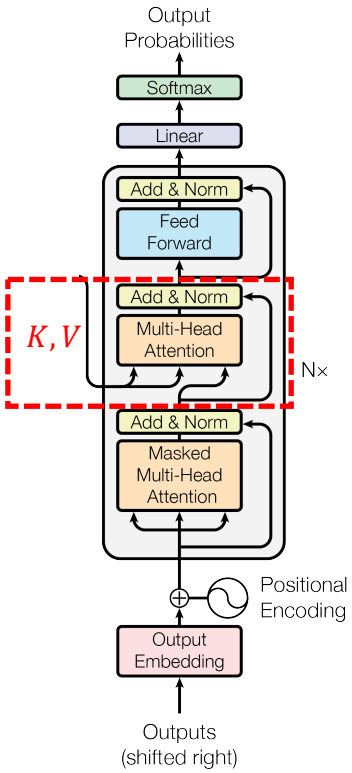




Ignore $\sqrt{d_k}$ for simplicity

$$\text{Attention}(Q^d, K^e, V^e) = \text{Softmax}\left(\frac{Q^d K^{eT}}{\sqrt{d_k}}\right) V$$

Encoder-Decoder Self-Attention



- Difference to standard self-attention:
  - Queries are computed by X^d
  - Keys and values are computed by X^e
- Number of outputs is len_d
- X^d only influences outputs by attention matrix A
- Value vector determined by X^e
- No Mask needed! y_i depends on x_i^d and X^e

$$\underbrace{q_1^d \; q_2^d}_{Q^d} = W^q \; x_1^d \; x_2^d$$

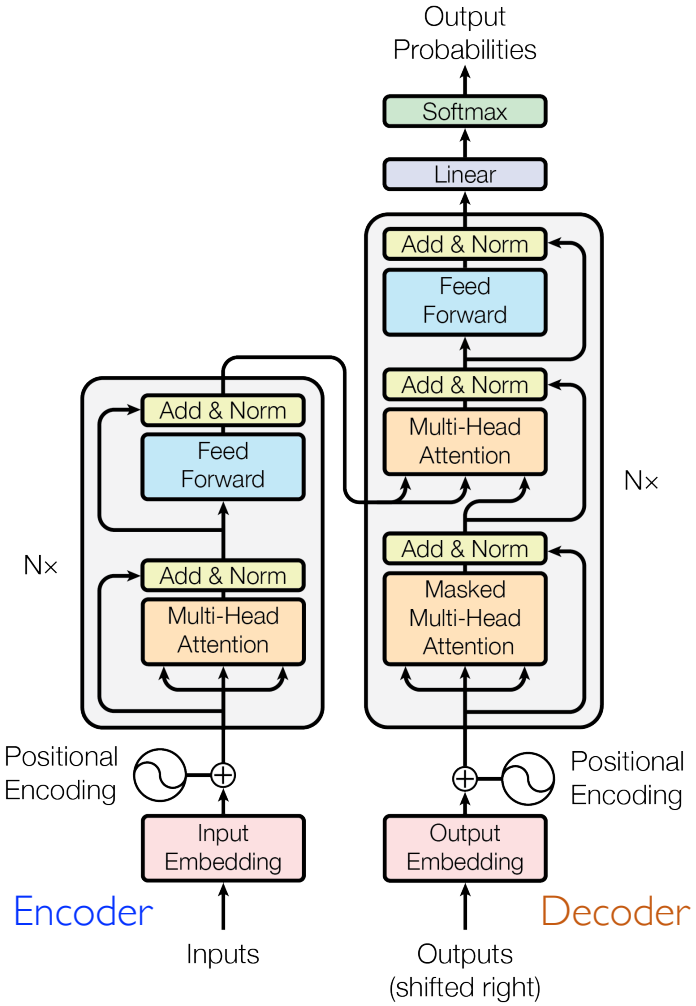
$$\underbrace{k_1^e \; k_2^e \; k_3^e}_{K^e} = W^k \; x_1^e \; x_2^e \; x_3^e$$

$$\underbrace{v_1^e \; v_2^e \; v_3^e}_{V^e} = W^v \; x_1^e \; x_2^e \; x_3^e$$

$$y_1^d \; y_2^d \xleftarrow{\text{MatMul}} v_1^e \; v_2^e \; v_3^e \; \underbrace{\begin{matrix} \alpha_{11} & \alpha_{21} \\ \alpha_{12} & \alpha_{22} \\ \alpha_{13} & \alpha_{23} \end{matrix}}_{\text{Softmax}} \xleftarrow{\text{MatMul}} \begin{matrix} k_1^e \\ k_2^e \\ k_3^e \end{matrix} \; q_1^d \; q_2^d$$

Ignore $\sqrt{d_k}$ for simplicity

Transformer



Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [15]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [32]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [8]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.0**	$2.3 \cdot 10^{19}$	

State-of-the-art results (At that time)

[31] Wu et al. "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation." 2016.

[8] Gehring et al. "Convolutional Sequence to Sequence Learning." *ICML 2017.*

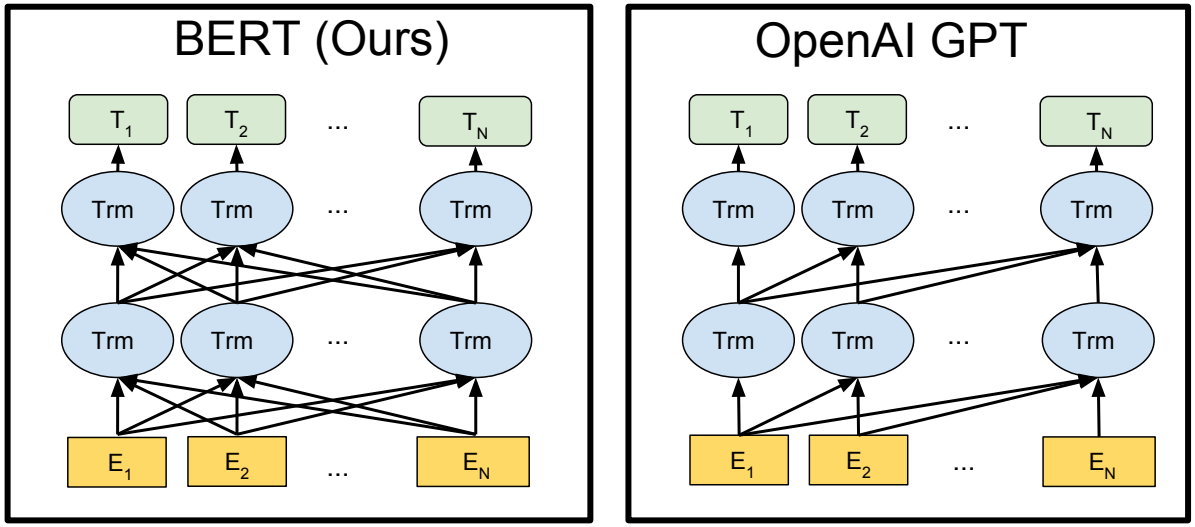
Vaswani et al. Attention is all you need. ***NIPS*** *2017.* (Cited 70781)

Outline

- Recurrent Neural Network (RNN)
  - LSTM: Long Short-Term Memory
  - Training Strategies
- Transformers: Attention is All You Need
  - **Language Transformers**
    - **> GPT: Generative Pre-Training**
  - Vision Transformers

BERT

- **B**idirectional **E**ncoder **R**epresentations from **T**ransformers (**BERT**)
- Pre-train deep bidirectional representations by jointly conditioning on both left and right contexts in all layers.



$\mathrm{BERT_{BASE}}$:
L =12, H=768, A =12,
Total Parameters=110M

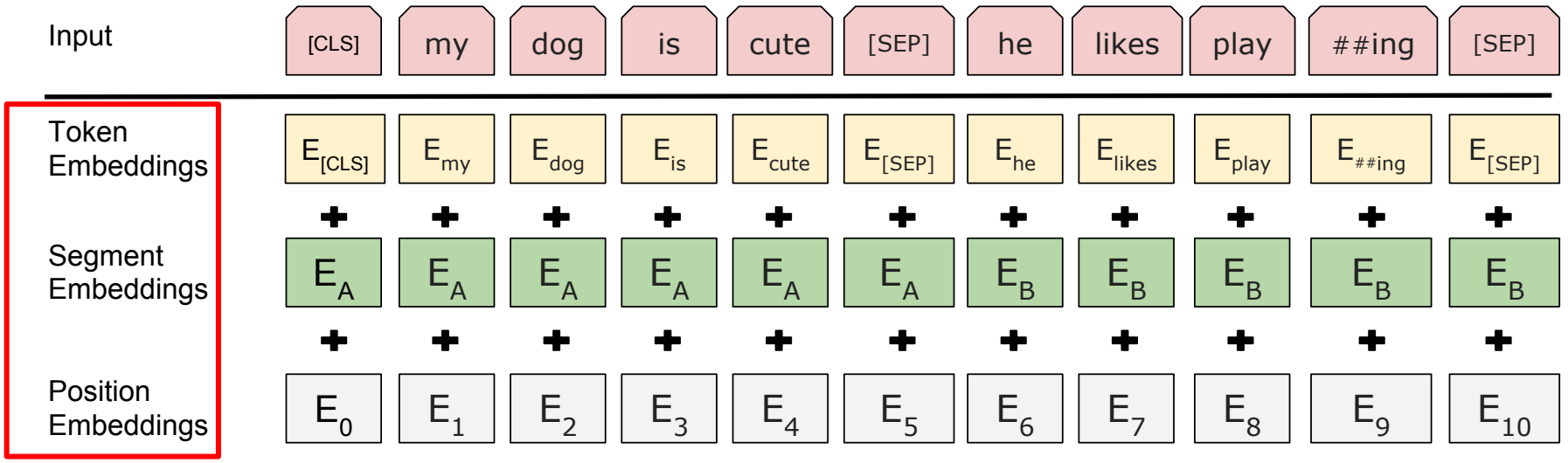
$\mathrm{BERT_{LARGE}}$:
L =24, H =1024, A=16,
Total Parameters=340M

L: the number of Transformer blocks; H: the hidden size; A : the number of self-attention heads.

Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" (cite 63204)

BERT: Input Representation

- The input embeddings are the sum of token embeddings, segment embeddings and position embeddings.
  - Add additional segment embeddings (feature engineering is back!)
- Each example is a pair of sentences



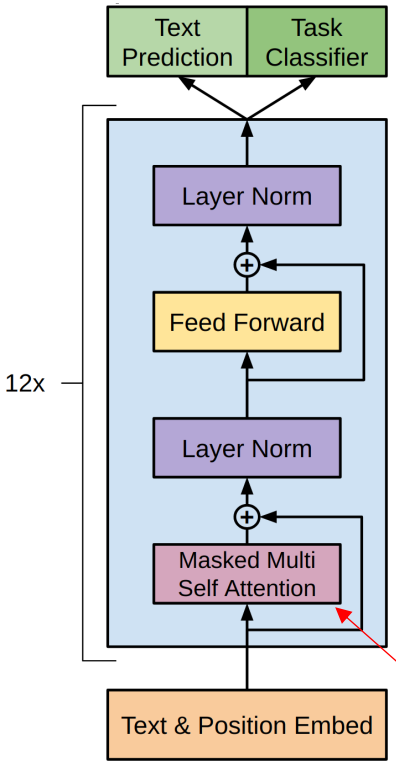
Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" (cite 63204)

Transfer Learning

- Pre-training + fine-tuning
- Increasing the training data set and the model size has a noticeable improvement on the transformer language model.
- Similar to pre-training computer vision models on ImageNet, we can pre-train a language model for NLP tasks and then finetune to downstream tasks.
  - **Stage 1: Unsupervised pre-training**
  - **Stage 2: Supervised fine-tuning**
- Generative Pre-Training (GPT)
- Word2vec is a pretrained model but it ignores sequential information

GPT: Generative Pre-trained Transformer

- **Stage 1: Unsupervised pre-training**
  - Learning a high-capacity language model on a large corpus of text.



The context vectors of tokens

The token embedding matrix

$$h_0 = UW_e + W_p$$

The position embedding matrix

$$h_l = \text{transformer\_block}(h_{l-1}) \qquad \forall\, i \in [1, n]$$

$$P(u) = \text{softmax}(h_n W_e^T)$$

Given an unsupervised corpus of tokens $\mathcal{U} = \{u_1, \ldots, u_n\}$

Maximize likelihood:

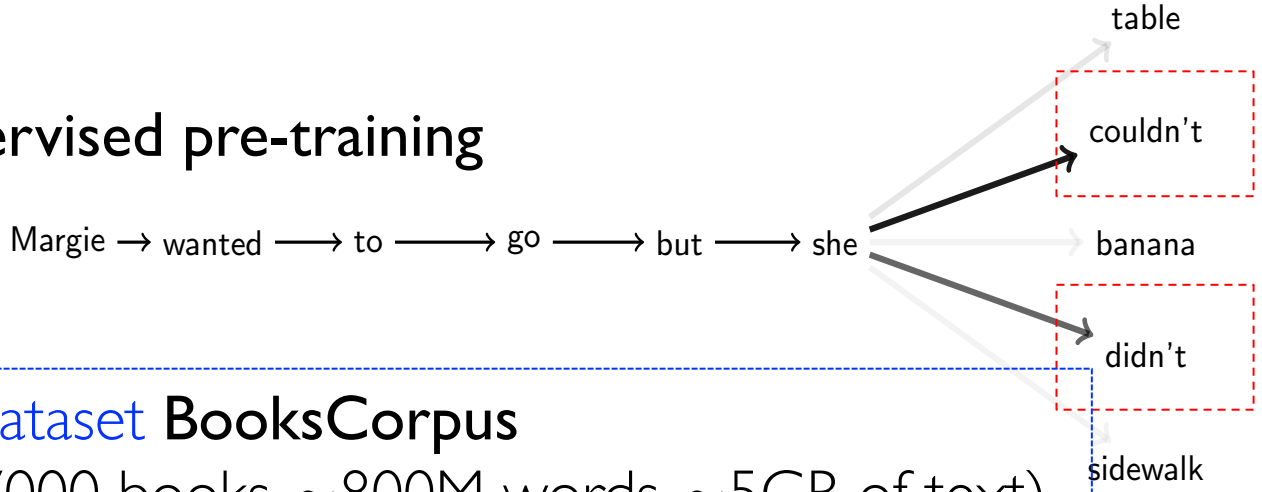
$$L_1(\mathcal{U}) = \sum_i \log P(u_i | u_{i-k}, \ldots, u_{i-1}; \Theta)$$

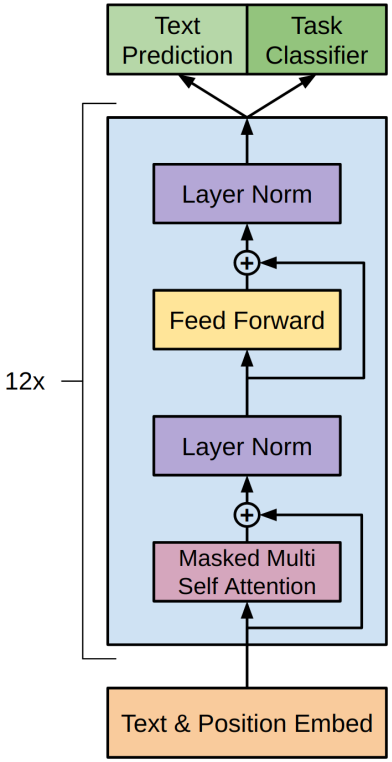
Similar to decoder block

Radford et al. Improving Language Understanding by Generative Pre-Training. (Cited 5169)

GPT: Generative Pre-trained Transformer

- Stage 1: Unsupervised pre-training





Dataset **BooksCorpus**
(7000 books, ~800M words, ~5GB of text)
Duration 1 month
Hardware 8 GPUs.

Given an unsupervised corpus of tokens $\mathcal{U} = \{u_1, \ldots, u_n\}$
Maximize likelihood:

$$L_1(\mathcal{U}) = \sum_i \log P(u_i | u_{i-k}, \ldots, u_{i-1}; \Theta)$$

Radford et al. Improving Language Understanding by Generative Pre-Training. (Cited 5169)

GPT: Generative Pre-trained Transformer

- **Stage 2: Supervised fine-tuning**
- Keep the pre-trained Transformers
- Replace the final linear layer
  - Replace W_e with W_y
- Given a labeled dataset $\mathcal{C}$, maximize the following objective:

$$L_2(\mathcal{C}) = \sum_{x,y} \log P(y|x^1, \ldots, x^m) = \sum_{x,y} \log \left(\text{softmax}\left(h_l^m W_y\right) \right)$$

The final transformer block's activation

A linear output layer

- Final loss function

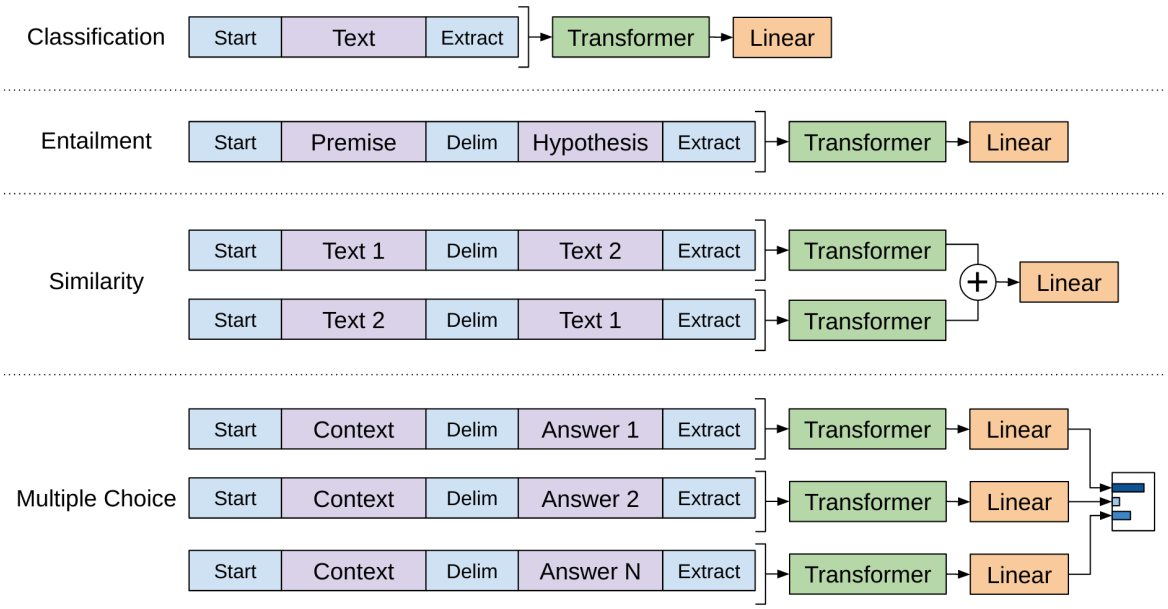
$$L_3(\mathcal{C}) = L_2(\mathcal{C}) + \lambda L_1(\mathcal{C})$$

Multi-Task Learning

Radford et al. Improving Language Understanding by Generative Pre-Training. (Cited 5169)

GPT: Generative Pre-trained Transformer

- Stage 2: Task-Specific Adaptation



Text classification.
Directly fine-tune

Other tasks, like
textual entailment, similarity.
Convert structured inputs into an ordered sequence that our pre-trained model can process

Given a labeled dataset $\mathcal{C}$, maximize the following objective:

$$L_2(\mathcal{C}) = \sum_{x,y} \log P(y|x^1, \ldots, x^m) = \sum_{x,y} \log(\text{softmax}(h_l^m W_y))$$

The final transformer block's activation

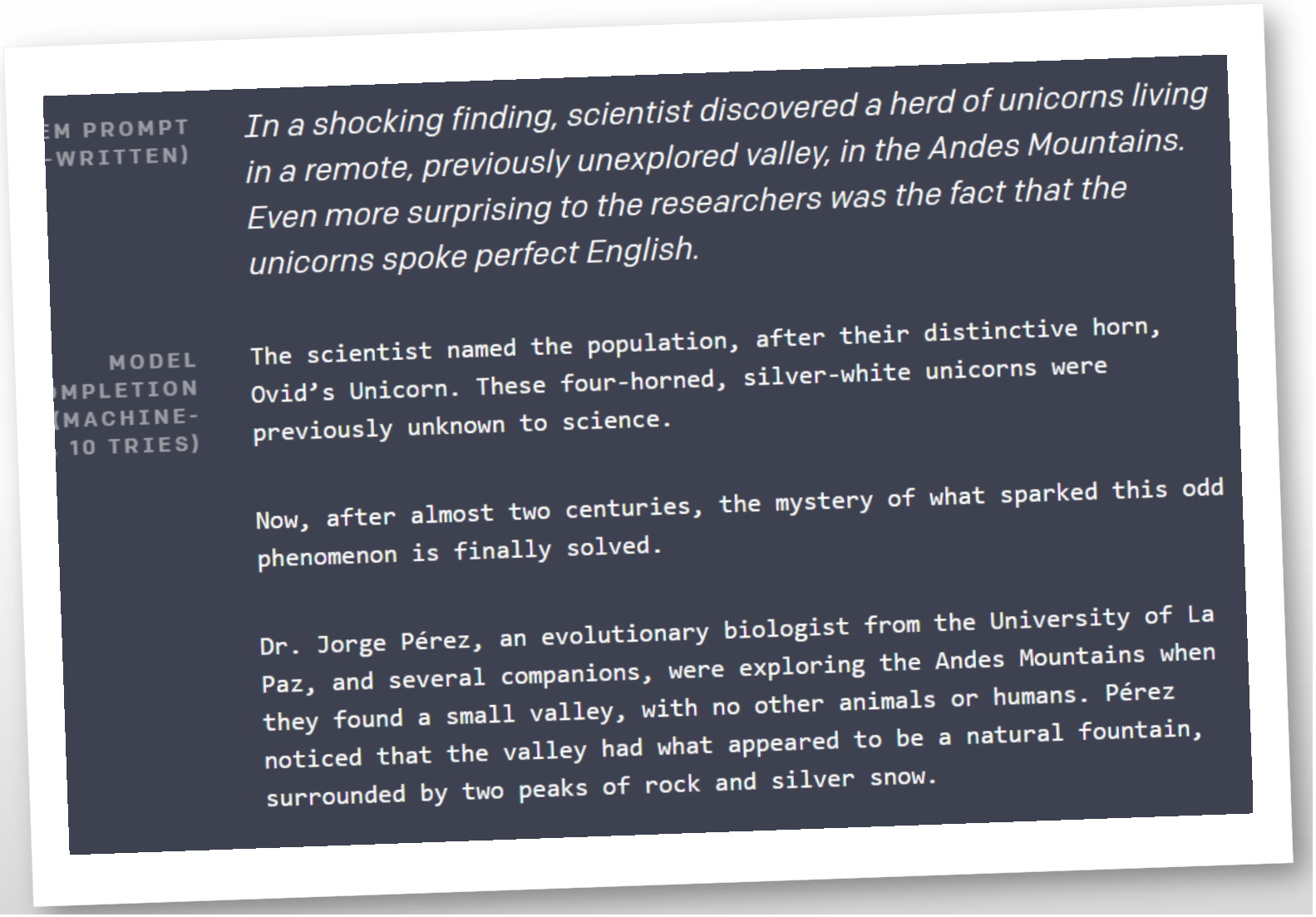
A linear output layer

Benchmark

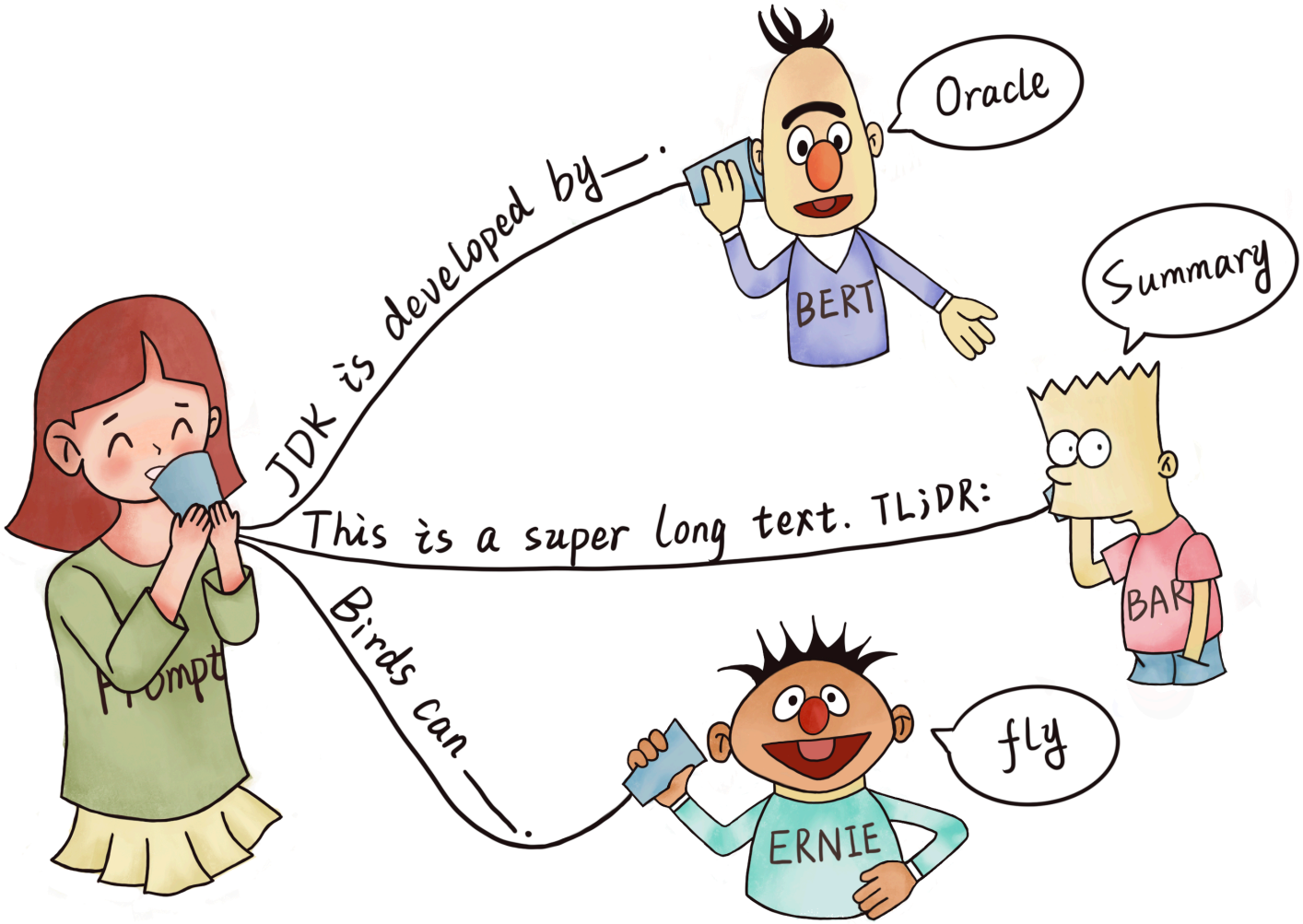
- **Textual Entailment**
  - SNLI 89.3 → 89.9
  - MNLI Matched 80.6 → 82.1
  - MNLI Mismatched 80.1 → 81.4
  - **SciTail 83.3 → 88.3**
  - QNLI 82.3 → 88.1
- **Semantic Similarity**
  - STS-B 81.0 → 82.0
  - QQP 66.1 → 70.3
- **Reading Comprehension**
  - RACE 53.3 → 59.0
- **Commonsense Reasoning**
  - ROCStories 77.6 → 86.5
  - COPA 71.2 → 78.6
- **Linguistic Acceptability**
  - CoLA 35.0 → 45.4
- **Multi-Task Benchmark**
  - GLUE 68.9 → 72.8

Radford et al. Improving Language Understanding by Generative Pre-Training. (Cited 5169)

Writing with GPT-2



Prompting in GPT-3



Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing

GPT-2 and GPT-3

- The developments in GPT-2 model:
  - Move norm layers before attention modules
  - Perform multi-task learning on a larger dataset (40GB)
  - More parameters (1542M in GPT-2 vs. 117M in GPT)
- GPT-3 use the same model and architecture as GPT-2, but with 45TB training data and 175 Billions parameters
- Zero-shot learning 😄
  - Reading Comprehension: $d_1, d_2, \cdots, d_N, \text{“}Q:\text{”}, q_1, q_2, \cdots, \text{“}A:\text{”}$
  - Summarization: $d_1, d_2, \cdots, d_N, \text{“TL; DR.”}$
  - Translation: $\mathbf{English} \rightarrow \mathbf{French}$

Radford et al. Language Models are Unsupervised Multitask Learners. (Cited 5169)

Brown et al. Language Models are Few-Shot Learners. NIPS 2020. (Best Paper, Cited 9019)

OpenAI Models

Models

Overview

The OpenAI API is powered by a diverse set of models with different capabilities and price points. You can also make limited customizations to our original base models for your specific use case with fine-tuning.

MODELS	DESCRIPTION
GPT-4 Limited beta	A set of models that improve on GPT-3.5 and can understand as well as generate natural language or code
GPT-3.5	A set of models that improve on GPT-3 and can understand as well as generate natural language or code
DALL·E Beta	A model that can generate and edit images given a natural language prompt
Whisper Beta	A model that can convert audio into text
Embeddings	A set of models that can convert text into a numerical form
Moderation	A fine-tuned model that can detect whether text may be sensitive or unsafe
GPT-3	A set of models that can understand and generate natural language
Codex Deprecated	A set of models that can understand and generate code, including translating natural language to code

We have also published open source models including Point-E, Whisper, Jukebox, and CLIP.

Visit our model index for researchers to learn more about which models have been featured in our research papers and the differences between model series like InstructGPT and GPT-3.5.

GPT-4 Limited beta

GPT-4 is a large multimodal model (accepting text inputs and emitting text outputs today, with image inputs coming in the future) that can solve difficult problems with greater accuracy than any of our previous models, thanks to its broader general knowledge and advanced reasoning capabilities. Like `gpt-3.5-turbo`, GPT-4 is optimized for chat but works well for traditional completions tasks. Learn how to use GPT-4 in our chat guide.

LATEST MODEL	DESCRIPTION	MAX TOKENS	TRAINING DATA
`gpt-4`	More capable than any GPT-3.5 model, able to do more complex tasks, and optimized for chat. Will be updated with our latest model iteration.	8,192 tokens	Up to Sep 2021
`gpt-4-0314`	Snapshot of `gpt-4` from March 14th 2023. Unlike `gpt-4`, this model will not receive updates, and will only be supported for a three month period ending on June 14th 2023.	8,192 tokens	Up to Sep 2021
`gpt-4-32k`	Same capabilities as the base `gpt-4` mode but with 4x the context length. Will be updated with our latest model iteration.	32,768 tokens	Up to Sep 2021
`gpt-4-32k-0314`	Snapshot of `gpt-4-32` from March 14th 2023. Unlike `gpt-4-32k`, this model will not receive updates, and will only be supported for a three month period ending on June 14th 2023.	32,768 tokens	Up to Sep 2021

For many basic tasks, the difference between GPT-4 and GPT-3.5 models is not significant. However, in more complex reasoning situations, GPT-4 is much more capable than any of our previous models.

GPT-3.5

GPT-3.5 models can understand and generate natural language or code. Our most capable and cost effective model in the GPT-3.5 family is `gpt-3.5-turbo` which has been optimized for chat but works well for traditional completions tasks as well.

LATEST MODEL	DESCRIPTION	MAX TOKENS	TRAINING DATA
`gpt-3.5-turbo`	Most capable GPT-3.5 model and optimized for chat at 1/10th the cost of `text-davinci-003`. Will be updated with our latest model iteration.	4,096 tokens	Up to Sep 2021
`gpt-3.5-turbo-0301`	Snapshot of `gpt-3.5-turbo` from March 1st 2023. Unlike `gpt-3.5-turbo`, this model will not receive updates, and will only be supported for a three month period ending on June 1st 2023.	4,096 tokens	Up to Sep 2021
`text-davinci-003`	Can do any language task with better quality, longer output, and consistent instruction-following than the curie, babbage, or ada models. Also supports inserting completions within text.	4,097 tokens	Up to Jun 2021
`text-davinci-002`	Similar capabilities to `text-davinci-003` but trained with supervised fine-tuning instead of reinforcement learning	4,097 tokens	Up to Jun 2021
`code-davinci-002`	Optimized for code-completion tasks	8,001 tokens	Up to Jun 2021

https://platform.openai.com/docs/models/overview

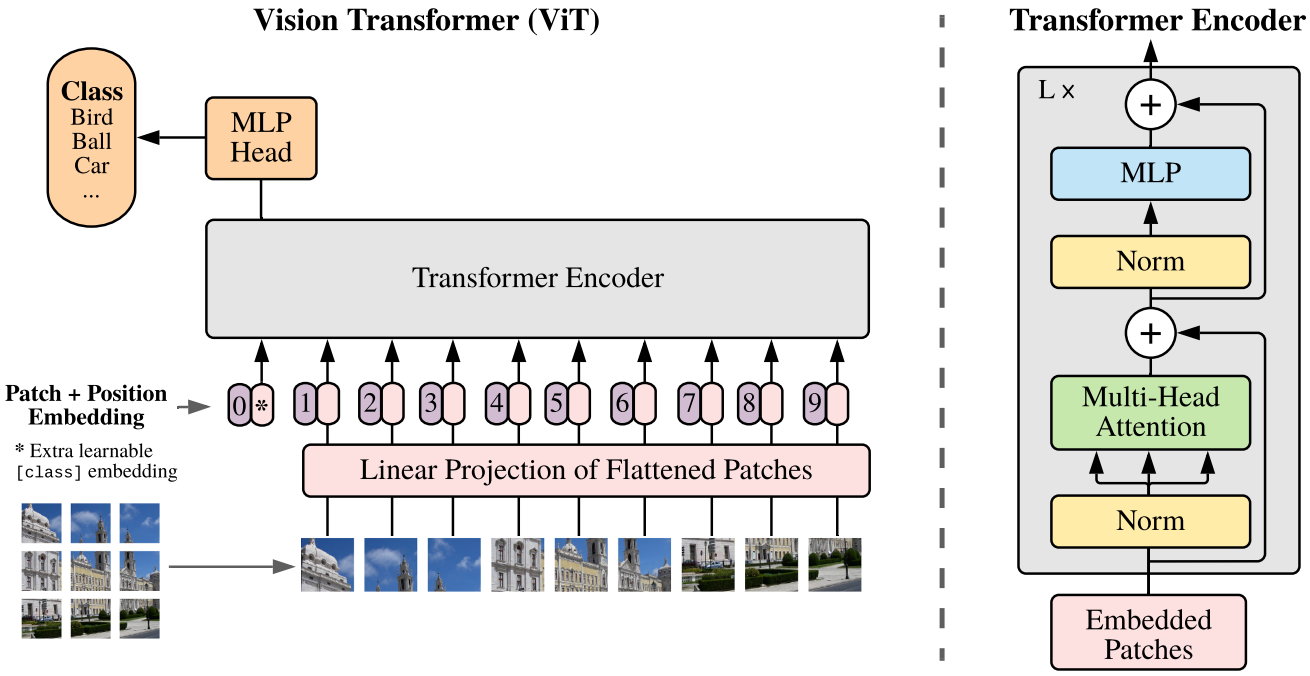
Outline

- Recurrent Neural Network (RNN)
  - LSTM: Long Short-Term Memory
  - Training Strategies
- Transformers: Attention is All You Need
  - Language Transformers
    - > GPT: Generative Pre-Training
  - **Vision Transformers**

ViT

- Split image into patches and provide the sequence of these patches as an input to a standard transformer (patch = convolution)

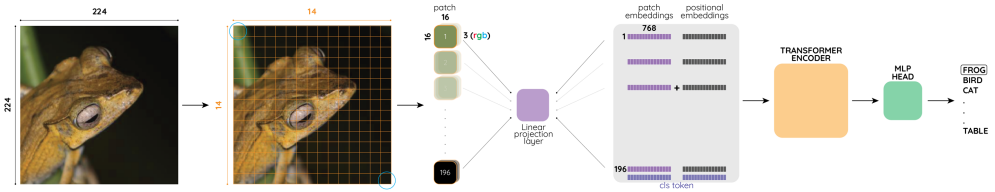
patches vs. tokens



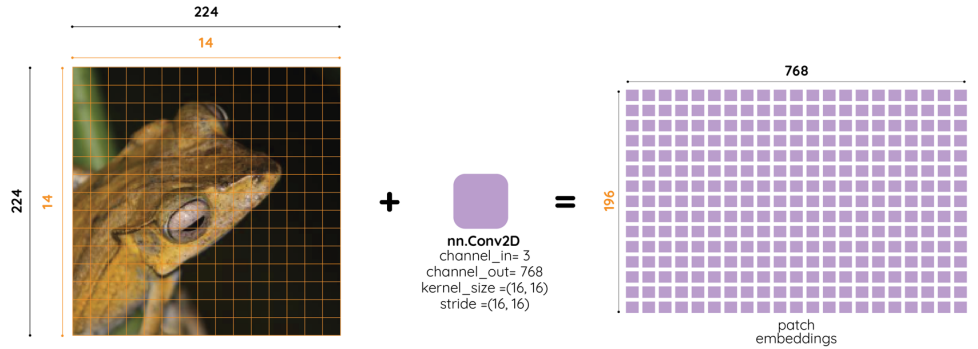
Dosovitskiy et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale (cite 14053)

ViT

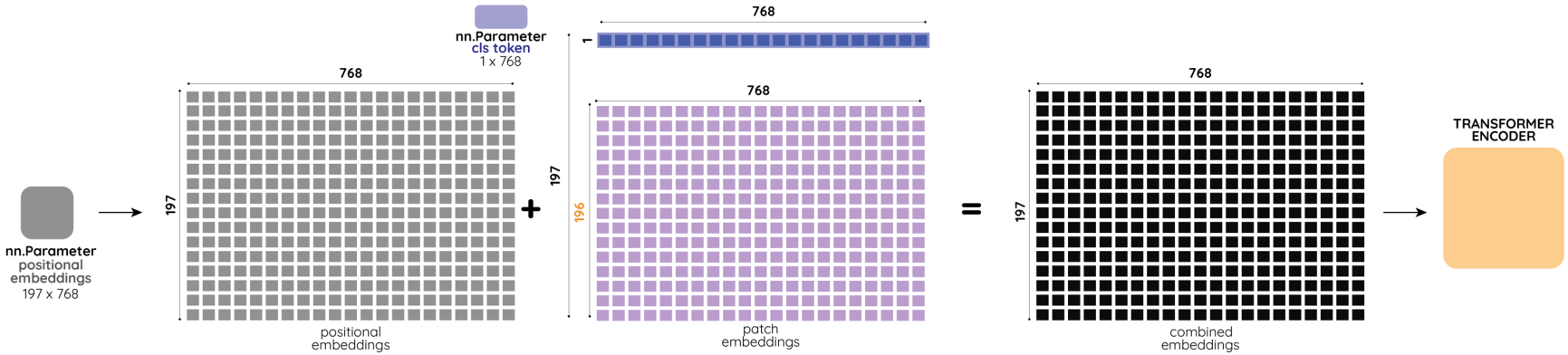
- Simplified Overview



- Patch Embeddings

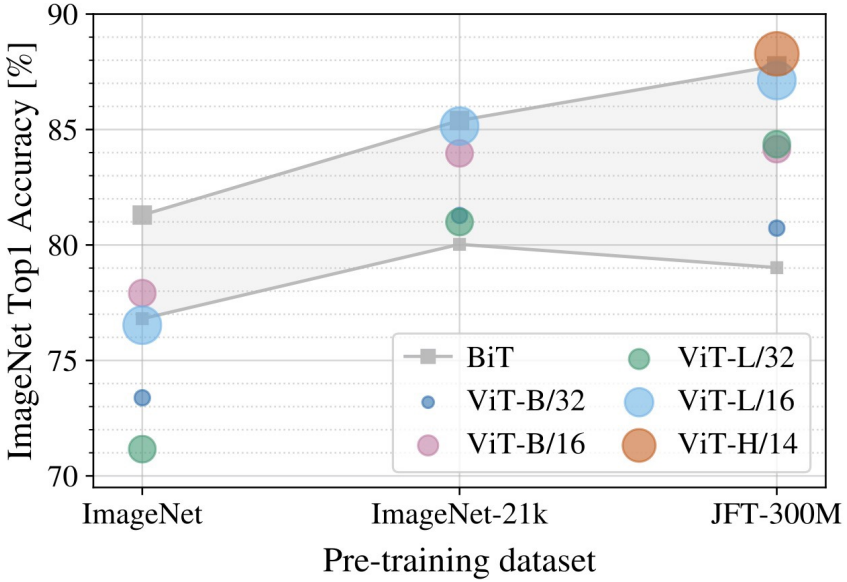


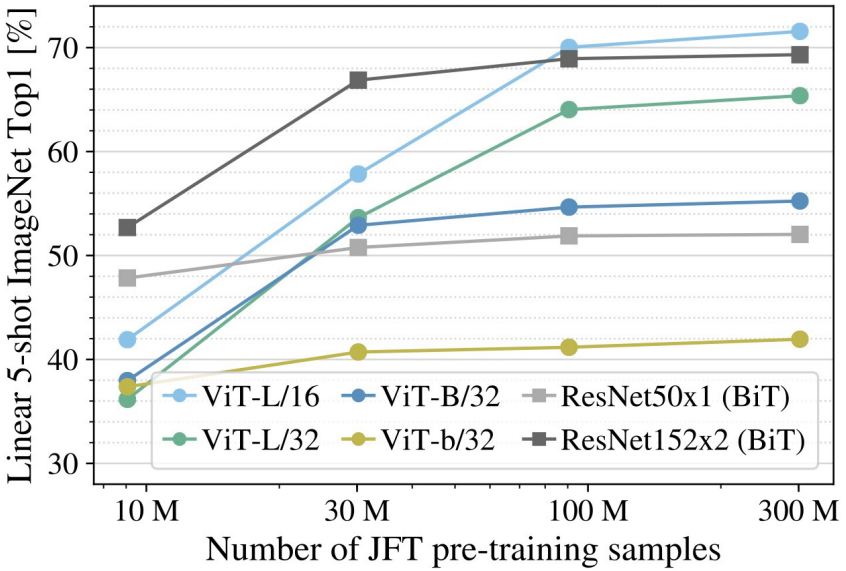
- [class] token & Position Embeddings



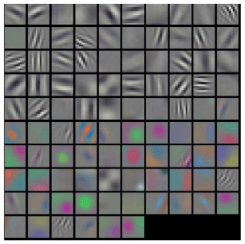
ViT

- New state-of-the-art on ImageNet. Very fast improvement nowadays!

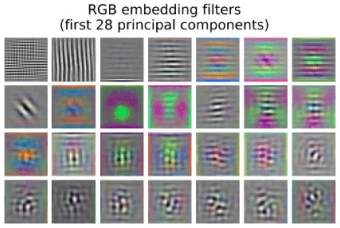




Alexnet 1st conv filters

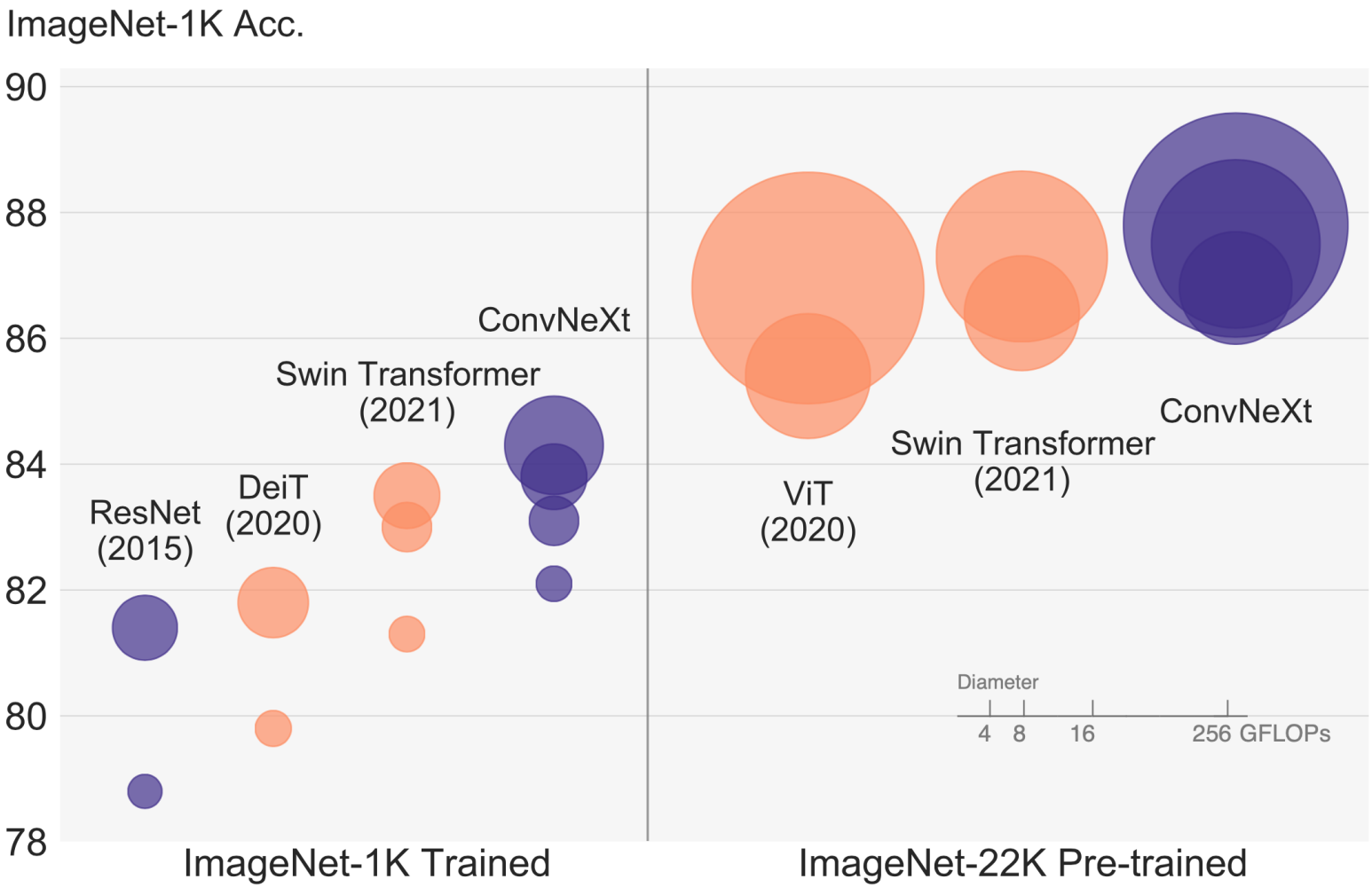


ViT 1st linear embedding filters



Attention from output token to input space

CNN vs. Transformer



Liu et al. A ConvNet for the 2020s. https://arxiv.org/pdf/2201.03545.pdf

Thank You

Questions?

Mingsheng Long
mingsheng@tsinghua.edu.cn
http://ise.thss.tsinghua.edu.cn/~mlong

答疑：东主楼11区413室