

Recurrent networks

Kyunghyun Cho

Feedforward networks

- A usual feedforward network
 - $f(x) = \sigma(W^L \sigma(W^{L-1} \sigma(\cdots \sigma(W^1 x + b^1) \cdots) + b^{L-1}) + b^L)$
- Consider a feedforward network with two hidden layers
 - A neural network: $f(x) = \sigma(W^2 \sigma(W^1 x + b^1) + b^2)$
 - Loss function: $l(f(x), y) = \|f(x) - y\|^2$

Feedforward networks

Gradient computation

- Gradient computation: backpropagation
 - $\partial l / \partial f(x) = 2(f(x) - y)$
 - $\partial f(x) / \partial a^2 = \sigma'(a^2)$, where $a^2 = W^2 \sigma(W^1 x + b^1) + b^2$
 - $\partial a^2 / \partial W^2 = \sigma(a^1)$, where $a^1 = W^1 x + b^1$: $\partial l / \partial W^2 = \partial l / \partial f(x) \times \partial f(x) / \partial a^2 \times \partial a^2 / \partial W^2$
 - $\partial a^2 / \partial b^2 = 1$: $\partial l / \partial b^2 = \partial l / \partial f(x) \times \partial f(x) / \partial a^2 \times \partial a^2 / \partial b^2$
 - $\partial a^2 / \partial \sigma(a^1) = W^2$ and $\partial \sigma(a^1) / \partial a^1 = \sigma'(a^1)$
 - $\partial a^1 / \partial W^1 = x$: $\partial l / \partial W^1 = \partial l / \partial f(x) \times \partial f(x) / \partial a^2 \times \partial a^2 / \partial \sigma(a^1) \times \partial \sigma(a^1) / \partial a^1 \times \partial a^1 / \partial W^1$
 - $\partial a_1 / \partial b^1 = 1$: $\partial l / \partial b^1 = \partial l / \partial f(x) \times \partial f(x) / \partial a^2 \times \partial a^2 / \partial \sigma(a^1) \times \partial \sigma(a^1) / \partial a^1 \times \partial a^1 / \partial b^1$

Feedforward networks

With tied weights

- Do the weights and biases have to be unique? What if...?
 - $f(x) = \sigma(W\sigma(W\sigma(\cdots\sigma(Wx + b)\cdots) + b) + b)$
 - Perfectly fine as long as W is a square matrix and $\dim(x) = \dim(b)$
 - But, does it change backpropagation?
 - Not really, as you have learned already earlier in the course.

Feedforward network

With tied weights

- The number of layers does not have to be fixed
 - $f(x) = \sigma(Wx + b)$
 - $f(x) = \sigma(W\sigma(Wx + b) + b)$
 - ...
 - $f(x) = \sigma(W\sigma(W\sigma(\cdots\sigma(Wx + b)\cdots) + b) + b)$
- The number of parameters is not tied to the amount of computation

Feedforward networks

With tied weights

- Consider a two-layer computation
 - $f(x) = \sigma(W\sigma(Wx + b) + b)$
 - Loss function: $l(f(x), y) = \|f(x) - y\|^2$
- Nothing has changed, but there is only a single pair of weight and bias.

Feedforward networks

With tied weights

- Gradient computation: backpropagation
 - $\partial l / \partial W = (\partial l / \partial f) \times (\partial f / \partial a^2) \times (\sigma(a^1)) + W(\partial \sigma(a^1) / \partial a^1) \times (\partial a^1 / \partial W)$
 - Equivalently, $\partial l / \partial W = \partial l / \partial [W]^2 + \partial l / \partial [W]^1$, where
 - $f(x) = \sigma([W]^2 \sigma([W]^1 x + b) + b)$
 - $\partial l / \partial [W]^2 = (\partial l / \partial f) \times (\partial f / \partial a^2) \times (\partial a^2 / \partial [W]^2)$
 - $\partial l / \partial [W]^1 = (\partial l / \partial f) \times (\partial f / \partial a^2) \times (\partial a^2 / \partial \sigma(a^1)) \times (\partial \sigma(a^1) / \partial a^1) \times (\partial a^1 / \partial [W]^1)$

Feedforward networks

Gradient computation with tied weights

- Gradient computation: backpropagation
 - Compute the gradient w.r.t. each *application* of the tied weight, and
 - Sum all those gradients to compute the gradient of the weight.
 - $f(x) = \sigma([W]^2 \sigma([W]^1 x + [b]^1) + [b]^2)$
 - $\partial l / \partial W = \partial l / \partial [W]^2 + \partial l / \partial [W]^1$
 - $\partial l / \partial b = \partial l / \partial [b]^2 + \partial l / \partial [b]^1$
- Generalizable to an arbitrary number of applications of an individual parameter

Feedforward networks

With tied weights

- Three typical use cases
 - Convolutional networks: one filter matrix is slid through the entire image.

- $\sum_{i=1}^{w-w'} \sum_{j=1}^{h-h'} \sum_{k=1}^{w'} \sum_{l=1}^{h'} (w \odot x_{i:i+w', j:j+h'})_{k,l}$, where w is a $w' \times h'$ filter and x is a $w \times h$ with a single channel (grayscale). More in Oct. 2 and 9!

- Recurrent networks: now, in this and next lectures!
- Attention: In the next week!

Recurrent networks

Handling a variable-length sequence

- Typically, we expect a fixed-size input
 - $x \in \mathbb{R}^d$ (a vector)
 - $x \in \mathbb{R}^{w \times h}$ (a b/w image)
 - $x \in \mathbb{R}^{w \times h \times c}$ (a color image)
- What if the size of the input varies?
 - $x \in \mathbb{R}^{d \times T}$ (a spectrogram)
 - $x \in \mathbb{R}^{w \times h \times c \times T}$ (a video clip)
 - $x \in V^T$, where V is a set of unique tokens (a sentence)
 - T varies from one example to another.

Recurrent networks

Handling a variable-length sequence

- Consider a *vector sequence* input and a *vector* output
 - (x^1, x^2, \dots, x^T) , where $T > 0$ and $x^t \in \mathbb{R}^d$
- A recurrent network for sequence classification/regression
 - A slight tweak to the feedforward net with tied weights
 - $f(x) = R\sigma(Wx^T + U\sigma(Wx^{T-1} + U\sigma(\dots\sigma(Wx^1 + Uh^0 + b)\dots) + b) + b) + c$
 - Equivalently, $f(x) = Rh^T + c$, where
 - $h^t = \sigma(Wx^t + Uh^{t-1} + b)$
 - h^0 is a learnable parameter (or often fixed to an all-zero vector.)

Recurrent networks

Handling a variable-length sequence

- A recurrent network for sequence regression
 - $f(x) = R\sigma(Wx^T + U\sigma(Wx^{T-1} + U\sigma(\cdots\sigma(Wx^1 + Uh^0 + b)\cdots) + b) + b) + c$
 - Loss function: $l(f(x), y) = \|f(x) - y\|^2$, when classifying a sequence.

Recurrent neural networks

Gradient computation by backpropagation through time

- Gradient computation: backpropagation through time

- $f(x) = R\sigma([W]^T x^T + [U]^T \sigma(\cdots \sigma([W]^1 x^1 + [U]^1 h^0 + [b]^1) \cdots) + [b]^T) + c$

- $\partial l / \partial R = \partial l / \partial f \times \partial f / \partial R$ and $\partial l / \partial c = \partial l / \partial f \times \partial f / \partial c$


- $\partial l / \partial U = \partial l / \partial f \times \sum_{t=1}^T (\partial f / \partial h^t \times \partial h^t / \partial [U]^t)$

- $\partial l / \partial W = \partial l / \partial f \times \sum_{t=1}^T (\partial f / \partial h^t \times \partial h^t / \partial [W]^t)$

- $\partial l / \partial b = \partial l / \partial f \times \sum_{t=1}^T (\partial f / \partial h^t \times \partial h^t / \partial [b]^t)$

Recurrent neural networks

Gradient computation by backpropagation through time

- Backpropagation through time
 - $\partial f / \partial h^t = \partial f / \partial h^T \times \partial h^T / \partial h^{T-1} \times \dots \times \partial h^{t+1} / \partial h^t$

 - Propagate the sensitivity to the perturbation *backward* in time.
- Temporal derivative
 - $\partial h^t / \partial h^{t-1} = \text{diag}(\sigma'(a^t))U$, where
 - $a^t = Wx^t + Uh^{t-1} + b$
 - $\sigma'(a^t)$ is the derivative of the element-wise nonlinearity σ .

Recurrent neural networks

Gradient computation by backpropagation through time

- The norm of the temporal derivative
 - Choose $\sigma : \mathbb{R}^d \rightarrow (-1,1)^d$ such as $\sigma(a) = \tanh(a)$
 - Both the output and its derivate are bounded from above and below.
 - $0 < \tanh'(x) \leq 1$
 - The upper bound on the norm of the temporal derivative
 - $\|\partial h^t / \partial h^{t-1}\| = \|\text{diag}(\sigma'(a^t))U\| \leq \|\text{diag}(\sigma'(a^t))\| \|U\|$ (Cauchy-Schwartz inequality)
 - $\|\text{diag}(\sigma'(a^t))\| \|U\| = \max \sigma'(a^t) \|U\| \leq \|U\|$ (because $\tanh'(x) \leq 1$)

Recurrent neural networks

Vanishing gradient

- The (spectral) norm of the temporal derivative
 - $\|\partial h^t / \partial h^{t-1}\| \leq \|U\|$
 - $\|\partial f / \partial h^t\| \leq \|\partial f / \partial h^T\| \|\partial h^T / \partial h^{T-1}\| \cdots \|\partial h^{t+1} / \partial h^t\| \leq \|\partial f / \partial h^T\| \|U\|^{T-t}$
- The norm of the gradient in a recurrent network is expressed as a *repeated product* of the norm of the transition matrix U .
 - When a positive number < 1 is repeatedly multiplied, it shrinks to zero exponentially fast.
 - When a positive number > 1 is repeatedly multiplied, it explodes to infinity exponentially fast.
 - When a positive number $= 1$ is repeatedly multiplied, it stays same ($=1$).

Recurrent neural networks

Vanishing gradient

- $\frac{\partial l}{\partial U} = \frac{\partial l}{\partial f} \sum_{t=1}^T \left(\frac{\partial f}{\partial h^t} \frac{\partial h^t}{\partial [U]^t} \right)$, where
 - $\|\partial f / \partial h^t\| \leq \|\partial f / \partial h^T\| \|\partial h^T / \partial h^{T-1}\| \cdots \|\partial h^{t+1} / \partial h^t\| \leq \|\partial f / \partial h^T\| \|U\|^{T-t}$
- $\left\| \frac{\partial l}{\partial U} \right\| \leq \left\| \frac{\partial l}{\partial f} \right\| \sum_{t=1}^T \left\| \frac{\partial f}{\partial h^t} \right\| \left\| \frac{\partial h^t}{\partial [U]^t} \right\| \leq \left\| \frac{\partial l}{\partial f} \right\| \sum_{t=1}^T \|U\|^{T-t} \left\| \frac{\partial h^t}{\partial [U]^t} \right\|$
- When the spectral norm is smaller than 1, the norm of the gradient w.r.t $[U]^t$ for small t rapidly *vanishes*, making it impossible to capture the *long-term dependencies*.

Recurrent neural networks

Vanishing gradient

- What does it mean for $\|\partial l / \partial [U]^t\| \rightarrow 0$?
 1. We don't know how to change U at time t to change the loss value, because the largest singular value of U ($= \|U\|$) is smaller than 1.
 2. We have changed U at time t enough already and reached the local minimum.
- It is often the mix of both:
 - A recurrent net has now captured short-term dependencies, but
 - A recurrent net cannot capture any long-term dependencies.

Recurrent neural networks

Quick digression: exploding gradient

- The (spectral) norm of the temporal derivative
 - $\|\partial h^t / \partial h^{t-1}\| \leq \|U\|$
 - $\|\partial f / \partial h^t\| \leq \|\partial f / \partial h^T\| \|\partial h^T / \partial h^{T-1}\| \cdots \|\partial h^{t+1} / \partial h^t\| \leq \|\partial f / \partial h^T\| \|U\|^{T-t}$
- The norm of the gradient in a recurrent network is expressed as a *repeated product* of the norm of the transition matrix U .
 - When a positive number < 1 is repeatedly multiplied, it shrinks to zero exponentially fast.
 - When a positive number > 1 is repeatedly multiplied, it explodes to infinity exponentially fast.
 - When a positive number $= 1$ is repeatedly multiplied, it stays same ($=1$).

Recurrent neural networks

Quick digression: exploding gradient

- The (spectral) norm of the temporal derivative
 - $\|\partial h^t / \partial h^{t-1}\| = \|\text{diag}(\sigma'(a^t))U\| \geq 1$, if $\|U\| \geq 1 / \max \sigma'(a^t)$, $\because \sigma'(a^t) > 0$.
 - This implies that for a certain configuration of the transition matrix U , the norm of the temporal derivative could explode rather than vanish.
- We address exploding gradient in practice by
 - Clipping the norm of the backpropagating gradient $\|\partial l / \partial h^t\|$, or
 - Clipping the norm of the gradient $\|\partial l / \partial U\|$.

Recurrent neural networks

Back to vanishing gradient

- A recurrent net cannot capture any long-term dependency, because the gradient norm *vanishes as it backpropagates through many time steps*.
- We change the network architecture to avoid repeated multiplication:

- Introduce **linear shortcuts** that bypass intermediate steps

- $$h^t = \sigma(Uh^{t-1} + Wx^t + b) + \sum_{t'=1}^{t-1} g^t(h^{t'})$$

- Equivalently,
$$h^t = \sigma(Uh^{t-1} + Wx^t + b) + \sum_{t'=1}^{t-1} g^t([h^{t'}]^t),$$
 where

- $[h^{t'}]^t$ refers to the use of $h^{t'}$ for computing h^t , similar to how we defined $[U]^t$ earlier.

Recurrent neural networks

Vanishing gradient and shortcut connections

- Gradient computation: backpropagation through time

- Recall $h^t = \sigma(Uh^{t-1} + Wx^t + b) + \sum_{t'=1}^{t-1} g^t([h^{t'}]^t)$

- $\frac{\partial h^t}{\partial h^{t'}} = \prod_{t''=1}^{t-t'} \frac{\partial h^{t-t''+1}}{\partial h^{t-t''}} + \frac{\partial h^t}{\partial g^t([h^{t'}]^t)} \frac{\partial g^t([h^{t'}]^t)}{\partial [h^{t'}]^t}$

- There is a one-hop path from h^t to $h^{t'}$.

Recurrent neural networks

Vanishing gradient and shortcut connections

- Gradient computation: backpropagation through time

$$\bullet \quad \frac{\partial h^t}{\partial h^{t'}} = \frac{\partial h^t}{\partial h^{t-1}} \frac{\partial h^{t-1}}{\partial h^{t'}} + \frac{\partial h^t}{\partial g^t([h^{t'}]^t)} \frac{\partial g^t([h^{t'}]^t)}{\partial [h^{t'}]^t}$$

$$\bullet \quad \frac{\partial h^{t-1}}{\partial h^{t'}} = \frac{\partial h^{t-1}}{\partial h^{t-2}} \frac{\partial h^{t-2}}{\partial h^{t'}} + \frac{\partial h^{t-1}}{\partial g^t([h^{t'}]^{t-1})} \frac{\partial g^t([h^{t'}]^{t-1})}{\partial [h^{t'}]^t}$$

- There are paths from h^t to $h^{t'}$ of all possible lengths $\in \{1, \dots, t' - t\}$

Recurrent neural networks

Vanishing gradient and shortcut connections

- Looks like this would solve the problem of vanishing gradient, but

- $$h^t = \sigma(Uh^{t-1} + Wx^t + b) + \sum_{t'=1}^{t-1} g^{t'}([h^{t'}]^t)$$

- The amount of computation grows as the sequence length grows.
- If each $g^{t'}$ requires a separate set of parameters, the number of parameters also grows as the length grows.

Recurrent neural networks

Vanishing gradient and linear shortcut connections

- An identity matrix is a transition matrix with the largest singular value=1:
 - $I^T = I$: Repeated product of an identity matrix does not change anything
- Residual connection creates shortcuts that do not vanish
 - $h^t = \sigma(Uh^{t-1} + Wx^t + b) + Ih^{t-1} = \sigma(Uh^{t-1} + Wx^t + b) + h^{t-1}$
 - $h^t = \sum_{t' \leq t} \sigma(Uh^{t'-1} + Wx^{t'} + b) + \sum_{t' \leq t} h^{t'-1}$
 - *Without* introducing any additional parameters nor computation (constant)

Recurrent neural networks

Vanishing gradient and linear shortcut connections

- It may *explode* in the forward pass!

$$\bullet \quad \|h^t\| \leq \left\| \sum_{t' \leq t} \sigma(Uh^{t'-1} + Wx^{t'} + b) \right\| + \left\| \sum_{t' \leq t} h^{t'-1} \right\| \leq \left\| \sum_{t' \leq t} \sigma(Uh^{t'-1} + Wx^{t'} + b) \right\| + \sum_{t' \leq t} \|h^{t'-1}\|$$

- It likely *explodes* in the backward pass as well, since the forward and backward passes are mirror of each other in a linear function

- Forward: Wx

$$\bullet \quad \text{Backward: } \frac{\partial l}{\partial Wx} \frac{\partial Wx}{\partial x} = \left(\frac{\partial l}{\partial Wx} \right)^\top W^\top$$

Recurrent neural networks

Vanishing gradient and gated shortcut connections

- Sparse short connections

- $$h^t = \underbrace{u^t \odot}_{\text{optional}} \sigma(Uh^{t-1} + Wx^t + b) + (1 - u^t) \odot h^{t-1}, \text{ where}$$

- $u^t = \sigma(U_u h^{t-1} + W_u x^t + b_u) \in [0, 1]^d$

- Only when the update gate u^t determines the shortcut is needed, the shortcut is introduced.

Recurrent neural networks

Vanishing gradient and gated shortcut connections

- Sparse short connections

- $h^t = \sigma(Uh^{t-1} + Wx^t + b) + (1 - u^t) \odot h^{t-1}$

- In the forward pass,

- $\|h^t\| \leq \left\| \sum_{t' \leq t} \sigma(Uh^{t'-1} + Wx^{t'} + b) \right\| + \sum_{t' \leq t} \left\| (1 - u^{t'}) \cdot h^{t'-1} \right\| < \left\| \sum_{t' \leq t} \sigma(Uh^{t'-1} + Wx^{t'} + b) \right\| + \sum_{t' \leq t} \|h^{t'-1}\|$

- The norm of the hidden state via shortcut is largely reduced by the update gate*: avoiding the explosion of both forward and backward pass.
- Meanwhile, it still captures long-term dependencies.

* strictly saying, it's not easy to establish this, as the update gate affects $h^{t'}$ s as well.

Recurrent neural networks

Gated recurrent units

- Sparse short connections & sparse recurrent connection
- $h^t = u^t \odot \sigma(U[(1 - r^t) \odot h^{t-1}] + Wx^t + b) + (1 - u^t) \odot h^{t-1}$, where
 - Update gate: $u^t = \sigma(U_u h^{t-1} + W_u x^t + b_u) \in [0,1]^d$
 - Reset gate: $r^t = \sigma(U_r h^{t-1} + W_r x^t + b_r) \in [0,1]^d$
- Only when the reset gate u^t determines the previous memory is important, the memory is used.
- There are a few variants, but all behave more or less similarly.

Recurrent neural networks

Long short-term memory (LSTM)

- Gated recurrent units (GRU) are a simpler variant of LSTM from 90's.
- Hidden state: $h^t = o^t \odot c^t$
- Cell state: $c^t = f^t \odot c^{t-1} + i^t \odot \tanh(W^c x^t + U^c c^{t-1} + V^c h^{t-1} + b^c)$
- Forget gate: $f^t = \sigma(W^f x^t + U^f c^{t-1} + V^f h^{t-1} + b^f)$
- Input gate: $i^t = \sigma(W^i x^t + U^i c^{t-1} + V^i h^{t-1} + b^i)$
- Output gate: $o^t = \sigma(W^o x^t + U^o c^{t-1} + V^o h^{t-1} + b^o)$

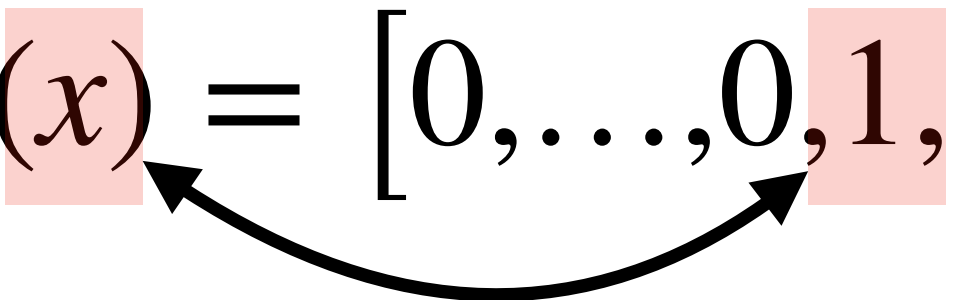
Recurrent neural networks

Some properties

- Learning by backpropagation-through-time (BPTT)
 1. Process the entire sequence
 2. Backpropagate the gradient through the entire sequence
 3. Update the parameters
- Online learning algorithms exist: variants of real-time recurrent learning (RTRL)
 - Particularly suitable when each sequence is very or infinitely long.
 - Out of the scope of this course
- Gated recurrent variants are much easier to use: LSTM, GRU, etc.
- They form a backbone of building a neural net that processes long sequences

Recurrent language modeling

An example of using a recurrent neural network

- Preliminary: embedding
 - When an input is an item $x \in V$ from a finite set $V = \{v_1, v_2, \dots, v_M\}$, we represent it as a one-hot vector: $\text{1-hot}(x) = [0, \dots, 0, 1, 0, \dots, 0]^\top$ 
 - Linear transformation is equivalent to table lookup.
 - $W \cdot \text{1-hot}(x) = W[:, x]$: slice out the x -th column of $W \in \mathbb{R}^{d \times |V|}$
 - We *embed* V in a d -dimensional real-valued space.

Recurrent language modeling

An example of using a recurrent neural network

- Preliminary: cross-entropy loss for multi-class classification
 - When an output is an item $y \in V$ from a finite set $V = \{v_1, v_2, \dots, v_M\}$, a neural net outputs a categorical distribution over V . In other words, it outputs M probabilities that sum to 1, using *softmax*:

$$\bullet \log p(y^* | x) = W[:, y^*]^\top h - \log \sum_{y \in V} \exp(W[:, y]^\top h)$$

- Let me omit the bias for now without loss of generality.

Recurrent language modeling

An example of using a recurrent neural network

- Preliminary: cross-entropy loss for multi-class classification

- When an output is an item $y \in V$ from a finite set $V = \{v_1, v_2, \dots, v_M\}$:

- $\log p(y^* | x) = W[:, y^*]^\top h - \log \sum_{y \in V} \exp(W[:, y]^\top h)$

- $\frac{\partial \log p(y^* | x)}{\partial h} = W_{y^*} - \sum_{y \in V} W_y \frac{\exp(W_y^\top h)}{\sum_{y' \in V} \exp(W_{y'}^\top h)} = W^\top (1\text{-hot}(y^*) - p(y | x))$

- Learning is equivalent to making the output closer to the true one-hot vector.

Recurrent language modeling

An example of using a recurrent neural network

- Language modeling: how likely is a given sentence?
- Autoregressive language modeling

$$p(X) = p(x_1)p(x_2 | x_1)p(x_3 | x_1, x_2) \cdots p(x_T | x_{<T}) = \prod_{t=1}^T p(x_t | x_{<t}),$$

- based on the definition of conditional probability.
- An autoregressive language model is parametrized using a recurrent network:
 - At time t , the input is a prefix $(x_1, x_2, \dots, x_{t-1})$ and the target is x_t .

Recurrent language modeling

An example of using a recurrent neural network

- An autoregressive language model is parametrized using a recurrent network:

$$p(X) = p(x_1)p(x_2 | x_1)p(x_3 | x_1, x_2) \cdots p(x_T | x_{<T}) = \prod_{t=1}^T p(x_t | x_{<t})$$

- At time t , the input is a prefix $(x_1, x_2, \dots, x_{t-1})$ and the target is x_t :

- h^0 , then $p(x^1) = G_{x^1}(h^0)$

- $h^1 = F(h^0, \text{1-one}(x^1))$, then $p(x^2) = G_{x^2}(h^1)$

- $h^2 = F(h^1, \text{1-one}(x^2))$, then $p(x^3) = G_{x^3}(h^2)$

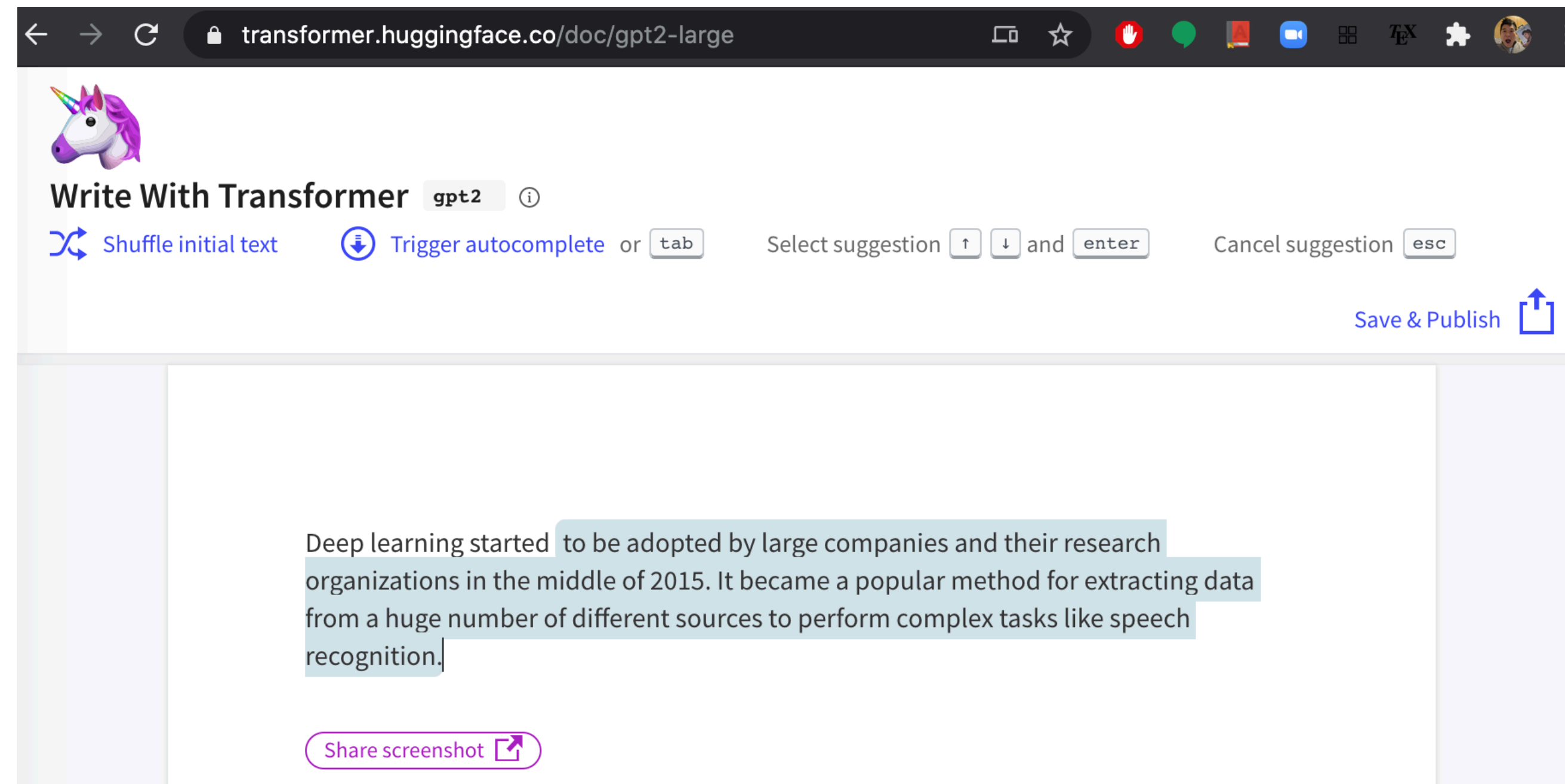
⋮

- $h^{T-1} = F(h^{T-2}, \text{1-one}(x^{T-1}))$, then $p(x^T) = G_{x^T}(h^{T-1})$

Recurrent language modeling

An example of using a recurrent neural network

- We can train a recurrent language model on a large amount of unlabelled text.

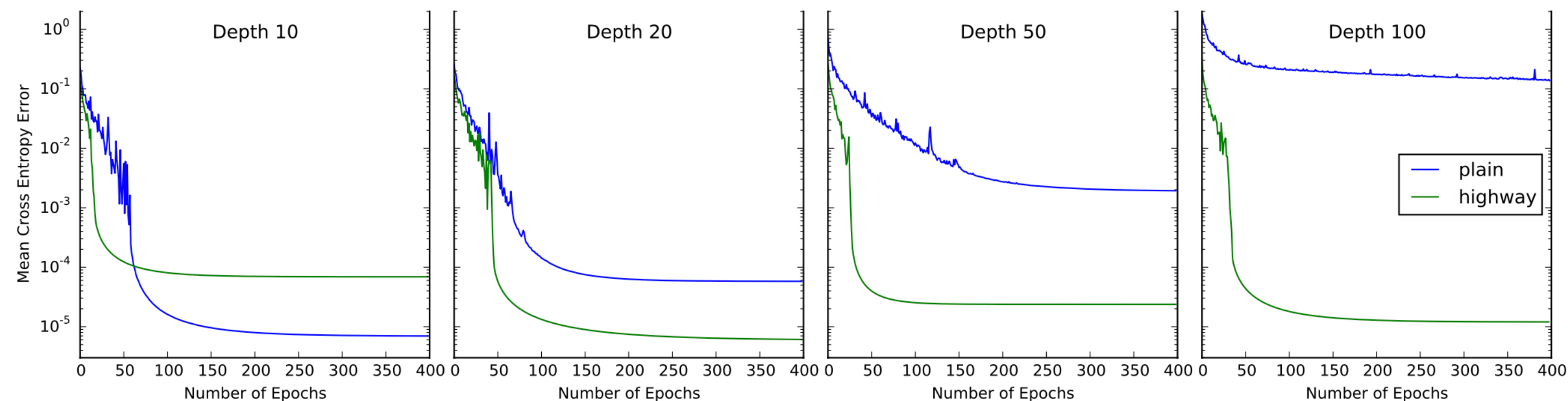


- We'll see more about it in the lab session 4.

Highway networks

Back to feedforward networks

- Bring adaptive shortcut connections to feedforward networks
 - $h^l = u^l \odot \sigma(U^l h^{l-1} + b^l) + (1 - u^l) \odot h^{l-1}$, where $u^l = \sigma(U_u^l h^{l-1} + b_u^l) \in [0,1]^d$
 - No extra input at each layer
 - No parameter sharing across layers
- This facilitates both optimization and generalization



Highway networks

Residual connections for feedforward networks

- Smooth interpolation between a linear network and a nonlinear network
 - $h^l = u^l \odot \sigma(U^l h^{l-1} + b^l) + (1 - u^l) \odot h^{l-1}$, where $u^l = \sigma(U_u^l h^{l-1} + b_u^l) \in [0,1]^d$
 - A linear network when $u^l = 0$ for all $l = 1, \dots, L$
 - The most complicated network when $u^l = 1$ for all $l = 1, \dots, L$
- Nonlinear components capture residual from more linear parts
 - When initializing $b_u^l \ll 0$, the network starts out as a linear network.
 - As training continues, nonlinear parts are activated and capture the residual.

Residual networks

Perhaps update gates are not necessary

- The feedforward network's depth is fixed or upper-bounded by a constant.
 - We do *not* need to worry too much about the *exploding gradient*: we just rescale the gradient, as the norm will never reach infinity.
 - We *still* need to worry about the *vanishing gradient*: we cannot really re-scale non-existent signal.
- How about $h^l = F(h^{l-1}) + h^{l-1}$?
 - Residual network: *de facto* standard.
 - We will learn more about it in the lectures 5 and 6 on convolutional networks

Attention

Gated recurrent units to attention

Replacing the recurrence with feedforward computation

- Start from GRU w/o reset: $h^t = u^t \odot \sigma(Uh^{t-1} + Wx^t + b) + (1 - u^t) \odot h^{t-1}$
- Let us unroll this recurrence equation
 - $h^1 = u^1 \odot \hat{h}^1 + (1 - u^1) \odot h^0$
 - $h^2 = u^2 \odot \hat{h}^2 + (1 - u^2) \odot u^1 \odot \hat{h}^1 + (1 - u^2) \odot (1 - u^1) \odot h^0$
 - $h^3 = u^3 \odot \hat{h}^3 + (1 - u^3) \odot u^2 \odot \hat{h}^2 + (1 - u^3) \odot (1 - u^2) \odot u^1 \odot \hat{h}^1 + (1 - u^3) \odot (1 - u^2) \odot (1 - u^1) \odot h^0$
 - ...
- $h^t = \sum_{t'=1}^t u^{t'} \odot \left(\prod_{t''=t'}^{t-1} (1 - u^{t''}) \right) \odot \hat{h}^{t'}$

Gated recurrent units to attention

Replacing the recurrence with feedforward computation

- Recurrence: $h^t = u^t \odot \sigma(Uh^{t-1} + Wx^t + b) + (1 - u^t) \odot h^{t-1}$
- Unrolled: $h^t = \sum_{t'=1}^t u^{t'} \odot \left(\prod_{t''=t'}^{t-1} (1 - u^{t''}) \right) \odot \hat{h}^{t'}$
- The state at time t is the weighted sum of all candidates states.
- Each candidate state encodes the input at the corresponding time step.

Gated recurrent units to attention

Replacing the recurrence with feedforward computation

- Recurrence: $h^t = u^t \odot \sigma(Uh^{t-1} + Wx^t + b) + (1 - u^t) \odot h^{t-1}$
- Unrolled: $h^t = \sum_{t'=1}^t u^{t'} \odot \left(\prod_{t''=t'}^{t-1} (1 - u^{t''}) \right) \odot \hat{h}^{t'}$
 - **The state at time t is the weighted sum of all candidates states.**
 - Each candidate state encodes the input at the corresponding time step.

Gated recurrent units to attention

Replacing the recurrence with feedforward computation

- Rewrite GRU as the weighted sum of inputs from earlier time steps

- Original:
$$h^t = \sum_{t'=1}^t u^{t'} \odot \left(\prod_{t''=t'}^{t-1} (1 - u^{t''}) \right) \odot \hat{h}^{t'}$$

- New:
$$h^t = \sum_{t'=1}^t w^{t'}(x^{t'}, h^{t'-1}) \odot \hat{h}^{t'}(x^{t'}, h^{t'-1})$$

- See the dependencies: it is still *recurrent*.

Gated recurrent units to attention

Replacing the recurrence with feedforward computation

- Relaxing one component at a time:
$$h^t = \sum_{t'=1}^t w^{t'}(x^{t'}, h^{t'-1}) \odot \hat{h}^{t'}(x^{t'}, h^{t'-1})$$

- What if each candidate state was computed independently?

- $$h^t = \sum_{t'=1}^t w^{t'}(x^{t'}, h^{t'-1}) \odot \hat{h}(x^{t'}, t')$$

- All the candidate states can be computed in *parallel*.
- We still want to ensure some *positional* information is included via embedding

Gated recurrent units to attention

Replacing the recurrence with feedforward computation

- Relaxing one component at a time:
$$h^t = \sum_{t'=1}^t w^{t'}(x^{t'}, h^{t'-1}) \odot \hat{h}(x^{t'}, t')$$

- What if each weight was computed independently?

- $$h^t = \sum_{t'=1}^t w(x^{t'}, x^t, t', t) \odot \hat{h}(x^{t'}, t')$$

- All the weights can be computed in *parallel* of course with positional information.

Gated recurrent units to attention

Replacing the recurrence with feedforward computation

- Implementing one component at a time:
$$h^t = \sum_{t'=1}^t w(x^{t'}, x^t, t', t) \odot \hat{h}(x^{t'}, t')$$
 - The weight represents the compatibility between the **query** and **key** vectors
 - $$w(x^{t'}, x^t, t', t) = \frac{\exp(Q(x^t, t)^\top K(x^{t'}, t'))}{\sum_{t'=1}^t \exp(Q(x^t, t)^\top K(x^{t'}, t'))}$$
 - The candidate state represents the **position**-sensitive **values** that are averaged
 - $$\hat{h} = V(x^{t'} + P(t'))$$

Gated recurrent units to attention

Replacing the recurrence with feedforward computation

- Multiple attention heads: $h^t = [h^{t,1}; \dots; h^{t,M}]^\top$

- $$h^{t,m} = \sum_{t'=1}^t w^m(x^{t'}, x^t) \odot \hat{h}^m(x^{t'}, t')$$

- $$w^m(x^{t'}, x^t, t', t) = \exp(Q^m(x^t, t)^\top K^m(x^{t'}, t')) / \sum_{t'=1}^t \exp(Q^m(x^t, t)^\top K^m(x^{t'}, t'))$$

- $$\hat{h} = V^m(x^{t'} + P(t'))$$

Gated recurrent units to attention

Replacing the recurrence with feedforward computation

- Nonlinear fusion: $h^t = f([h^{t,1}; \dots; h^{t,M}])$
 - Attention is largely linear: $h^{t,m} = \sum_{t'=1}^t w^m(x^{t'}, x^t, t', t) \odot \hat{h}^m(x^{t'}, t')$
 - $Q^m(x, t) = W_Q(x + P(t))$, $K^m(x, t) = W_K(x + P(t))$ and $V^m(x, t) = W_V(x + P(t))$
 - Nonlinearity is crucial to capture complicated relationship in the input
 - f is often designed as a multilayer perceptron with a residual connection
 - $f(\tilde{h}^t) = \text{ReLU}(U_f \text{ReLU}(W_f \tilde{h}^t + b_f) + c_f) + x^t$, where $\tilde{h}^t = [h^{t,1}; \dots; h^{t,M}]$

Gated recurrent units to attention

Replacing the recurrence with feedforward computation

- Self-attention: non-causal attention

- $h^t = f([h^{t,1}; \dots; h^{t,M}])$, where

- $$h^{t,m} = \sum_{t'=1}^T w^m(x^{t'}, x^t, t', t) \odot \hat{h}^m(x^{t'}, t')$$

- $Q^m(x, t) = W_Q(x + P(t))$, $K^m(x, t) = W_K(x + P(t))$ and $V^m(x, t) = W_V(x + P(t))$

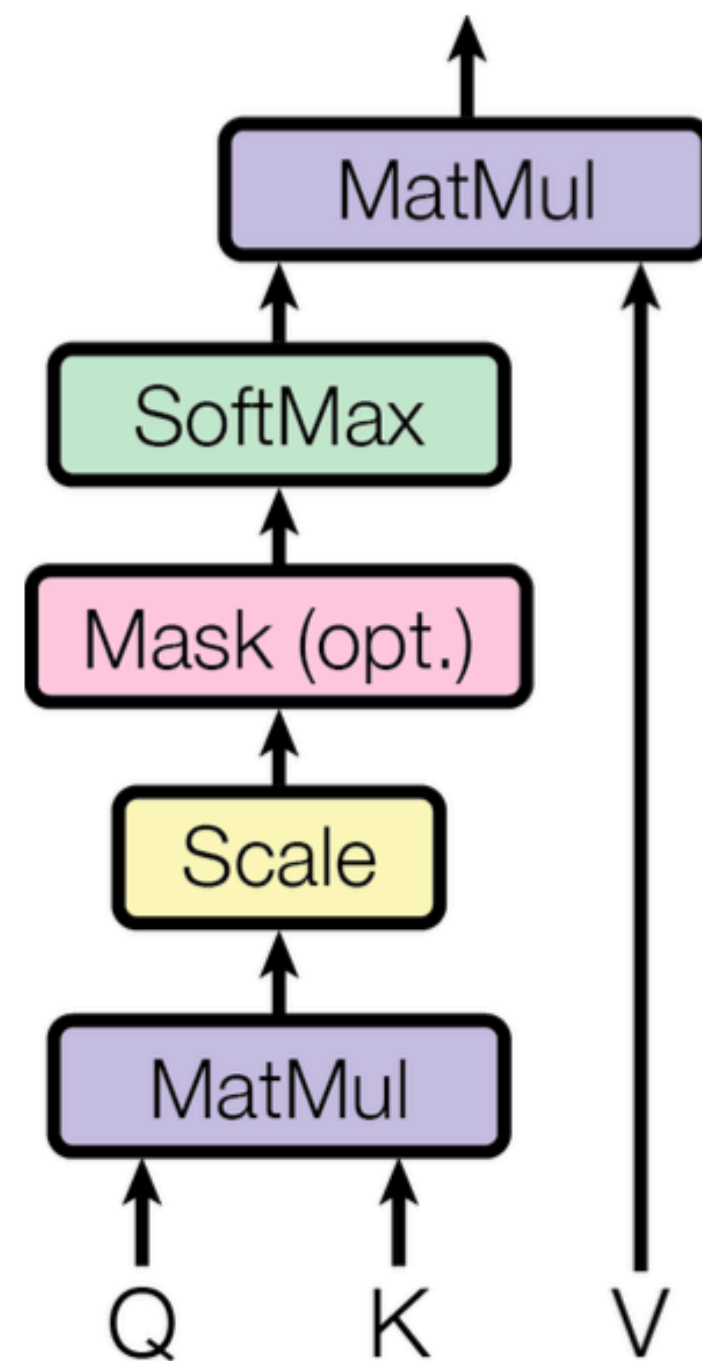
- $$w^m(x^{t'}, x^t, t', t) = \exp(Q^m(x^t, t)^\top K^m(x^{t'}, t')) / \sum_{t'=1}^t \exp(Q^m(x^t, t)^\top K^m(x^{t'}, t'))$$

- $\hat{h} = V^m(x^{t'} + P(t'))$

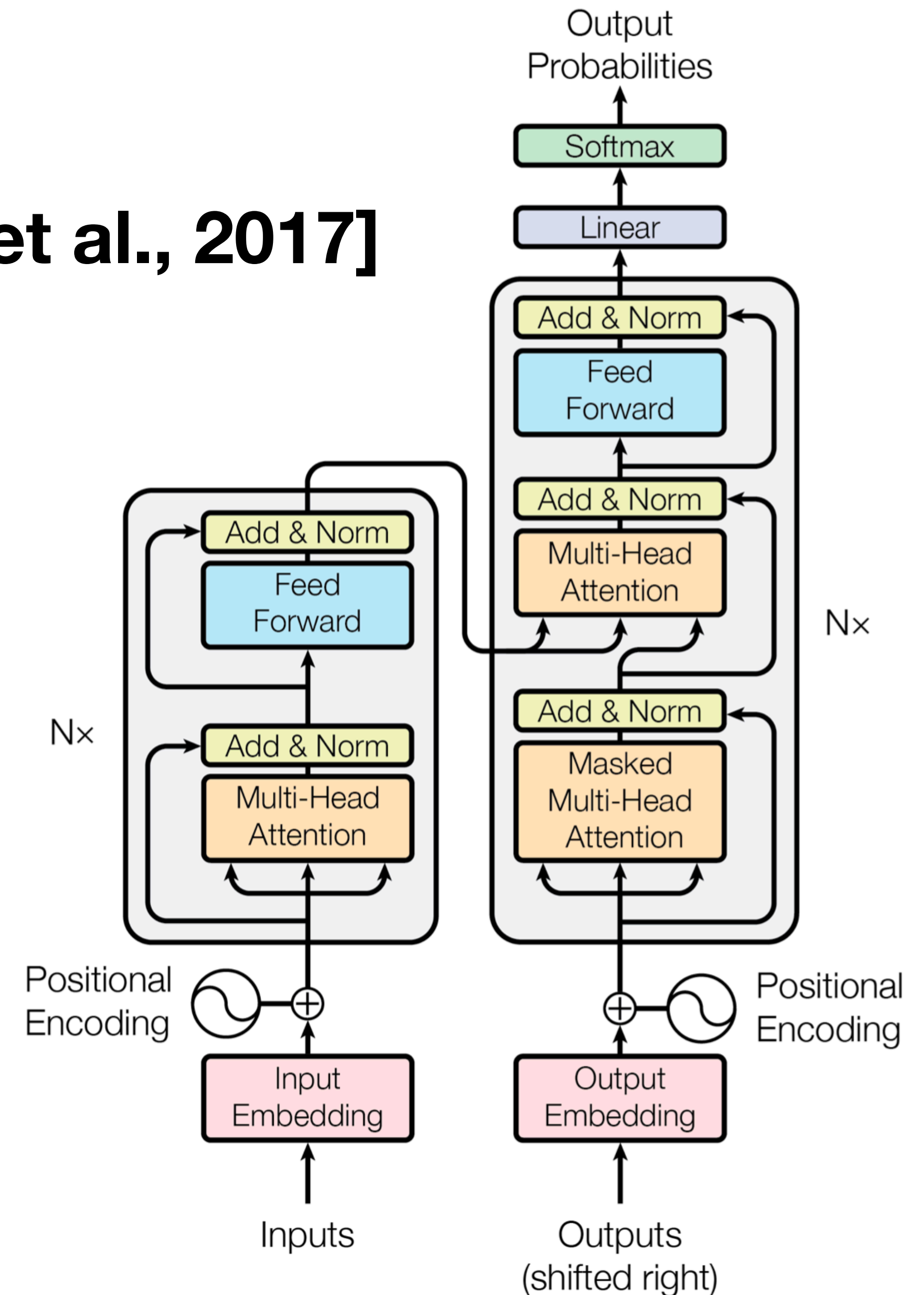
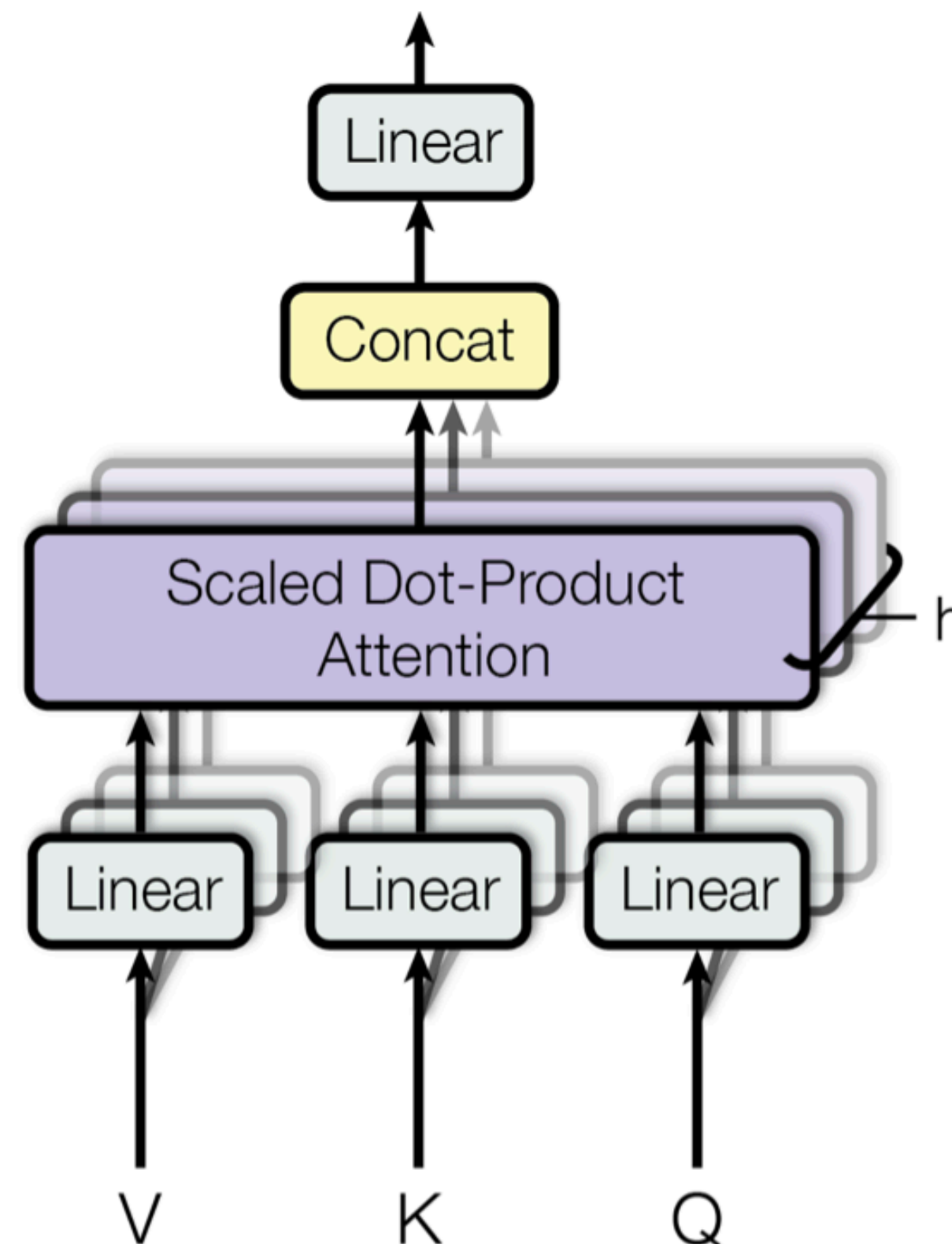
Transformers

Attention-only neural networks [Vaswani et al., 2017]

Scaled Dot-Product Attention



Multi-Head Attention



Gated recurrent units to attention

Properties

- Attention does not suffer from vanishing gradient.
- Attention enables heavy parallelization:
 - The only sequential computation is reduction.

- $$h^{t,m} = \sum_{t'=1}^T w^m(x^{t'}, x^t, t', t) \odot \hat{h}^m(x^{t'}, t')$$

- Attention however requires $O(T^2)$ computation, as opposed to $O(T)$ of RNN

Learning to deal with a set

Preview: deep set networks

- Input to the attention block is a set of position-augmented items
 - An unstructured feature vector is a set of one vector
 - A sequence of words is a set of position-augmented words
 - An image is a set of position-augmented pixels (or pixel blocks)
 - Not everything admits this view: a generic graph requires a bit more.. why?
- Attention allows us to build a neural net that works with *structured input*.
- More to be discussed in November further by Alfredo.