전반적인 흐름 이해

CNN

RNN

LSTM GRU.

transformer

↳ motivation

actual computation (▷ GRU
RNN
Trans

(intuition)

AE VAE GAN.

Convolution Neural Network

Convolution ⇒ spatial filtering

kernel. ⇒ input = m kernel size = k stride = S output size = $\frac{(m-k)}{S}+1$

padding : zeroed frame. border effects

Compute Output Shape = $\left(\frac{H+2P-F}{S}+1\right) \times \left(\frac{W+2P-F}{S}+1\right)$

H: input height W: input width
P: padding size F: filter size S: Stride

Conv properties
- preserve signal support structure
- sparse interaction (local connectivity)
- parameter sharing
- equivariance to transition conv(shift(input)) = shift(conv(input))
- high K ⇒ larger receptive field (local region of input)

Pooling : replace output of layer at certain location with summary stat. of nearby outputs.
- smaller representation, more manageable
- ⇒ max pooling, average pooling, ...
- invariant to small translations of the input. small change of input ⇒ most pooled output no change.

Parameter : by model in training. (W, b)
Hyperparameter : by user before training. (LR, Epoch, ...)

Calculate # params
1. Linear: $(I \times O)+O$ (I: input size, O: output size)
2. Conv: $((K_w \times K_h \times C_{in})+1) \times C_{out}$ (K: kernel C: channel)
3. Batch Norm: $C \times 2$ C: channel.
4. pooling : 0
5. RNN: $H \times I + H \times H + H$ (H: hidden state, I: input)
6. LSTM: $4 \times (RNN)$ (∵ 4 gates)
7. GRU: $3 \times (RNN)$ (∵ 3 gates)
8. Transformer : (d: model dimension, V: vocab size)
   - Each Encoder : Multi-head Attention : $4 \times d^2$ (K, Q, V, Out)
   - Feed Forward Network: $2(d \times 4d) = 8d^2$ ($d \to 4d \to d$)

Total: $(V \times d) + N \times (\quad)$

VGGNet

ResNet

Sequential Data : Sentences, video frames, speech waveform, ...
bag of words: vocab set. no order. ⇒ one hot encode, ...

model for sequential data : deal with variable length sequence
maintain sequence order
keep track of long term dependencies
share parameterers across the sequence.

Vectorizing : transform text into numeric tensors.
⇒ break down text into (word, character, n-gram)

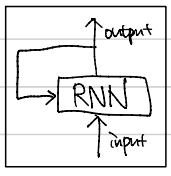
Tokenization: word : delimeter = space. most common. big vocab. (embedding matrix), OOV
character : small vocab. no meaning long seq length.
Subword : no split frequent words, split rare words into meaningful subword, zero-shot ok
Byte-Pair Encoding : iteratively find most frequent pair of adjacent characters in data
merge them into a new single token. ⇒ add new tokens until desired vocab size.

Vectorization :
one-hot encoding : size of vector = size of vocab. each token has unique index of vector
binary, sparse, high-dimensional, hard-coded

token(word) embedding : use of dense word vectors. lower-dimension, learn from data

Why Embedding? efficiency : one-hot ⇒ large, most zero vectors. embedding ⇒ direct row indexing, O(1) MEM efficient.
interpretability : directly storing token vector. ⇒ pretrained, visualized, reused.
simplicity : each token has its own learnable vectors.

Recurrent Neural Networks => process sequence by iterating through seq elem and maintain state
state: containing info relative to what it has seen so far




```
state = 0
for input in input_seq:
    output = activation(dot(W, input) + dot(U, state) + b)
    state = output
```

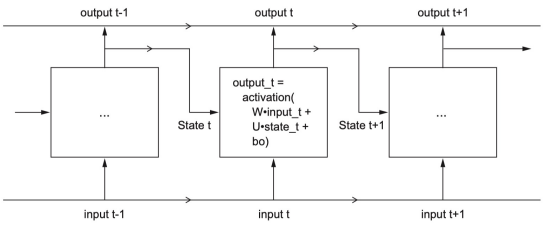


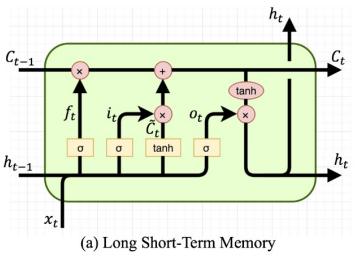
Figure 6.10 A simple RNN, unrolled over time

$y = \sigma(State \cdot U + input \cdot W) + b)$

param of simple RNN.
Embedding layer : Vocab size x Embedding Dim (10000x64)
RNN layer : $H \times (H+I) + B$ $64(64+64) + (64+2)$
Linear layer : $(I \times O) + B$ $(64 \times 1) + 1$

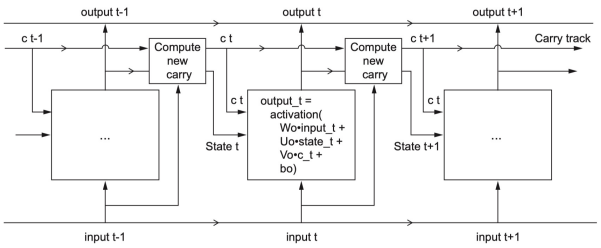
Packing ⇒ padding 없이 처리가능.

LSTM : carry info across many timesteps.
Simple RNN + additional data flow. (carry)

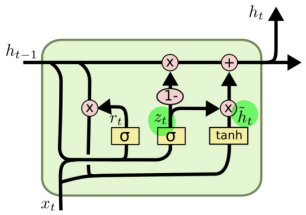


(a) Long Short-Term Memory

$$i_t = \sigma(x_t U^i + h_{t-1} W^i)$$
$$f_t = \sigma(x_t U^f + h_{t-1} W^f)$$
$$o_t = \sigma(x_t U^o + h_{t-1} W^o)$$
$$\tilde{C}_t = \tanh\left(x_t U^g + h_{t-1} W^g\right)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$h_t = \tanh(C_t) * o_t$$



GRU



(b) Gated Recurrent Unit

$$z_t = \sigma(x_t U^z + h_{t-1} W^z)$$
$$r_t = \sigma(x_t U^r + h_{t-1} W^r)$$
$$\tilde{h}_t = \tanh\left(x_t U^h + (r_t * h_{t-1}) W^h\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Bidirectional RNNs
- forward
- backward

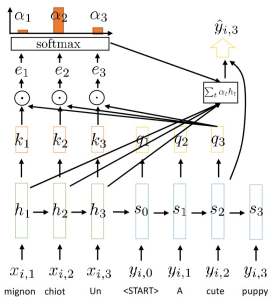
} knowledge coming next

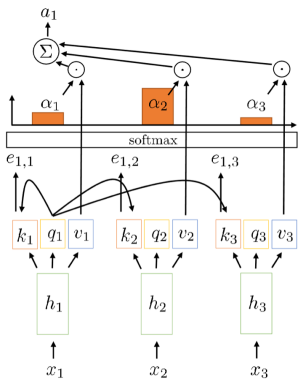
Attention ⇒ vector of importance weights. To predict one elem, using attention how strongly correlated with others.

Seq2seq ⇒ fixed size embedding. is incapable of remembering long seq. decode from last hidden state.
Attention ⇒ context vector(decoder state) from entire encoded vectors.
teacher forcing in training by feeding GT token.

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

attention can access every time step. RNN → pure attention (OK)
no temporal dependencies.





$a_l = \sum_t \alpha_{l,t} v_t$

$\alpha_{l,t} = \exp(e_{l,t}) / \sum_{t'} \exp(e_{l,t'})$

$e_{l,t} = q_l \cdot k_t$

we'll see why this is important soon

$v_t = v(h_t)$ before just had $v(h_t) = h_t$, now e.g. $v(h_t) = W_v h_t$

$k_t = k(h_t)$ (just like before) e.g., $k_t = W_k h_t$

$q_t = q(h_t)$ e.g., $q_t = W_q h_t$

this is *not* a recurrent model!

but still weight sharing:

$h_t = \sigma(W x_t + b)$

shared weights at all time steps

(or any other nonlinear function)

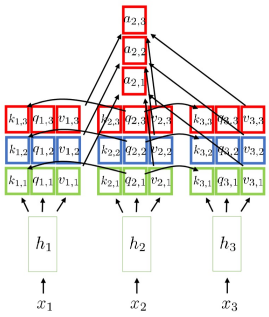
Self-attention limitation

1. positional encoding : { position of words has meaning. represent relative position. by frequency-based sin/cos encoding.
   { learning positional encoding ⇒ more flexible, complex. cannot generalize.

   input = emb(x_t) + p_t

2. Multi-head attention: softmax ⇒ dominated by one value. ⋯ hard to specify 2 diff things.

   then... multiple keys, queries, values for every time step.



full attention vector formed by concatenation:

$$a_2 = \begin{bmatrix} a_{2,1} \\ a_{2,2} \\ a_{2,3} \end{bmatrix}$$

compute weights **independently** for each head

$e_{l,t,i} = q_{l,i} \cdot k_{l,i}$

$\alpha_{l,t,i} = \exp(e_{l,t,i}) / \sum_{t'} \exp(e_{l,t',i})$

$a_{l,i} = \sum_t \alpha_{l,t,i} v_{t,i}$

around **8** heads seems to work pretty well for big models

3. Adding nonlinearity : every self attention layer is a linear transformation of previous layer

   $h_t^l = \sigma(W^l a_t^l + b^l)$

4. Masked encoding : prevent attention looking into the future.

   problem: self attention can look at the value at next steps. based on inputs at next steps.

   During test time ⇒ for output of step 1 requires input of 2 & 3.

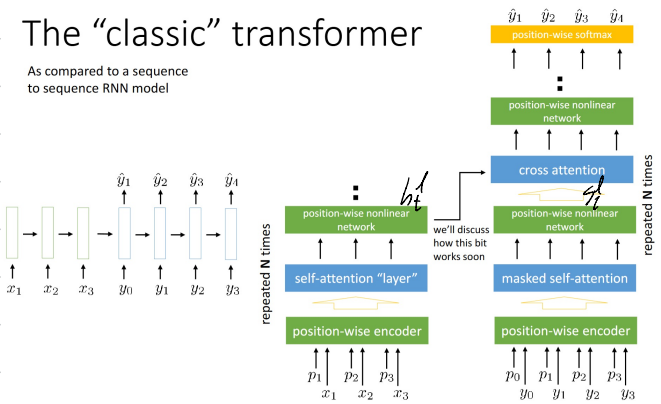
   Self attention ⇒ allow past info. but not future.

   solution : $e_{l,t} = \begin{cases} q_l \cdot k_t & (\text{if } l > t) \\ -\infty & (\text{o.w.}) \end{cases}$ ⇒ Zero value in Softmax.

Transformer

The "classic" transformer

As compared to a sequence to sequence RNN model



Cross Attention

query: $q_t^l = W_q^l s_t^l$

key: $k_t^l = W_k^l h_t^l$

value: $v_t^l = W_v^l h_t^l$

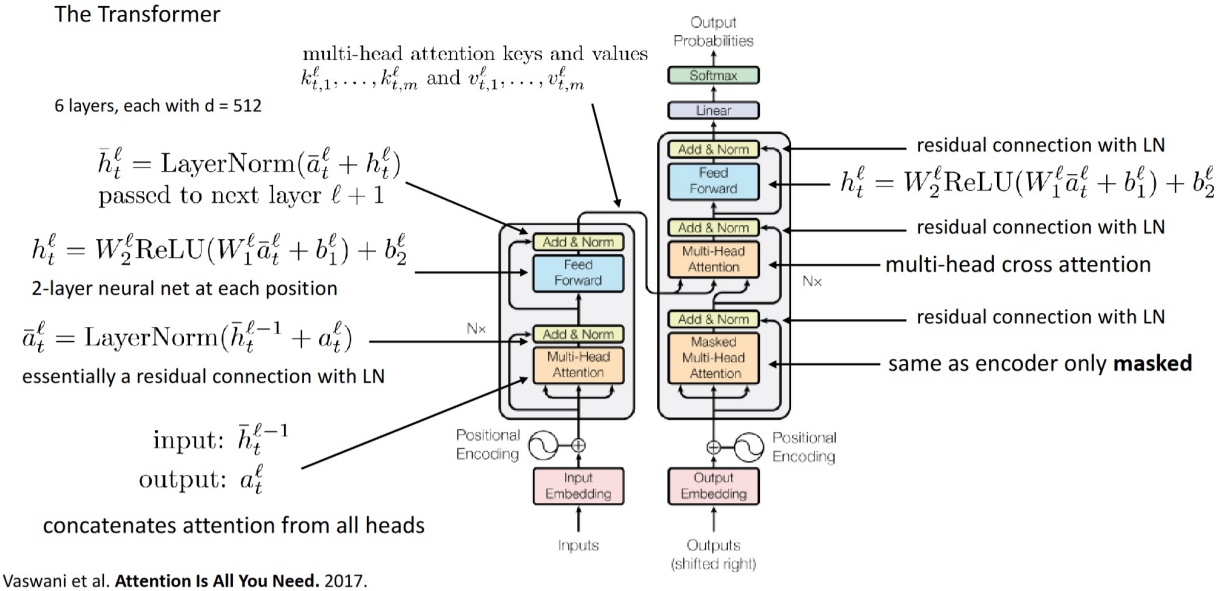
$e_{t,t'}^l = q_t^l \cdot k_{t'}^l$, $\alpha_{t,t'}^l = \exp(e_{t,t'}^l) / \sum_{t'} (\exp(e_{t,t'}^l))$, $c_t^l = \sum_{t'} \alpha_{t,t'}^l v_{t'}^l$

* Layer Normalization for variable length of text. (dimension-wise)
Each data get its own stat(μ, σ) ⇒ norm

Putting it all together

Decoder decodes one position at a time with masked attention

The Transformer



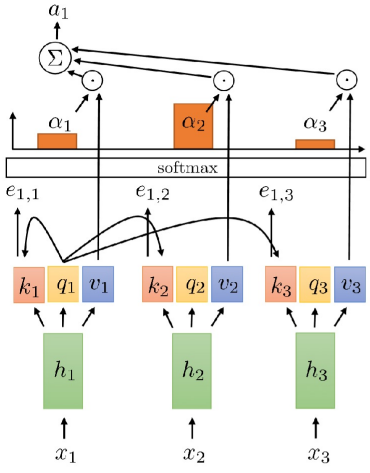
Vaswani et al. **Attention Is All You Need.** 2017.

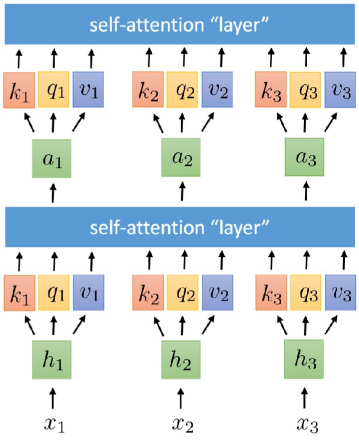
Drawback
→ $O(n^2)$
complex implementation

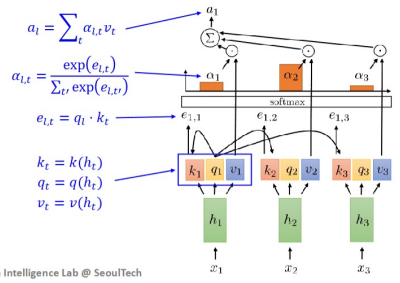
Benefits
→ long range connection
easy to parallelize
deeper network.

Language Model. : probabilistic model that assigns a probability $P(w_1, w_2, \cdots, w_n)$ to every finite seq $w_1, w_2, \cdots, w_n$.
chain rule of conditional prob. $P(w_1, w_2, \cdots, w_n) = P(w_1) P(w_2|w_1) \cdots P(w_n | w_1, w_2, \cdots, w_{n-1})$
⇒ equivalent to being able to predict next word.

Attention can access every time step. RNN can do, Attention can do more.





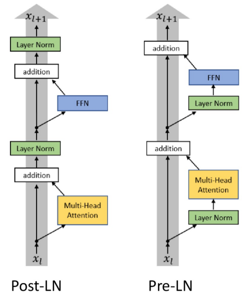


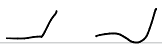
Normalization Position. LN ⇒ $\frac{x-\mu}{\sigma}$, $\sigma = \sqrt{\frac{1}{n}\Sigma(x-\mu)^2}$

Post-LN : vanilla. between residual blocks. instable due to large gradients near the output layer.

Pre-LN : LN applied before each sub-layer.

RMSNorm : Simplify LN by removing mean stat. $RMS(a) = \sqrt{\frac{1}{n}\sum_{i}^{n} a_i^2}$





Activation Function : ReLU GeLU ···

Positional Embedding : APE : added to content representation,
RPE : add position info to model.

FLAN : data ⇒ Instruction, improve zero-shot.

Alignment & RLHF ⇒ ① Supervised Fine Tuning. ② Reward Model. ③ Proximal Policy Optimization

↳ objective = follow user's instruction usefully safely

Discriminative vs. Generative Models. (Give observed variable x, target variable y)

Discriminative ⇒ estimate $P(y|x)$ conditional prob.

Generative ⇒ estimate $P(x,y)$ joint prob.

w/o assuming y, learn $p(x)$ from given data.

↳ enable use to generate new data similar to training dataset

how to sample new plausible data?

Statistical perspective : generation / sampling ⇒ data point from probability distribution

Map simple dist $P(z)$ to data dist $p(x)$. z: latent variable. $p(z)$: latent dist.

Sample from $p(z)$, map it to data point ⓧ $\xleftarrow{P(z)}$ ⓩ

AutoEncoder. $x \xrightarrow[\text{enc}]{\text{deterministic}} z \xrightarrow[\text{dec}]{\text{deterministic}} \hat{x}$ decoder (latent → input) ⇒ latent space is unstructured.

⇒ enable accurate reconstruction $e_\theta(x)$ $d_\phi(z)$

guarantee latent is mapped to data point

random sampling latent code produce meaningless outputs.

Variational AutoEncoder. (probabilistic structure on latent space)

$x - e_\theta(x) - \frac{\mu}{\sigma} - z \sim N(\mu, \sigma^2) - z - d_\phi(z) - \hat{x}$ ⓧ $\underset{\mu_\theta(x)\ \sigma_\theta(x)}{\overset{D(z)}{\rightleftarrows}}$ ⓩ $p(x|z) = N(x; D(z), \sigma^2)$, $p(z) = N(z; 0, I)$

Max ELBO. Max $P(x)$

$ELBO = E_{q_\phi(z|x)}[\log(p_\theta(x|z))] - D_{KL}(q_\phi(x|z) \| P(z))$

⇒ reconstruction (max) accurate recovery of x from z. ⇒ prior matching term (mini) $q_\phi(x|z)$ stay close to $p(z)$

$p(x) = \int p_\theta(x,z)dz = \int p_\theta(x|z)p(z)dz$, $p_\theta(x|z) = N(x; D_\theta(z), \sigma^2 I)$, $P(z) = N(z; 0, I)$: decoder

$q_\phi(z|x) = N(z; \mu_\phi(x), \sigma_\phi(x)I)$: encoder.

VAE training

1. Feed data point x to encoder to predict $\mu_\phi(x)$, $\tilde{\sigma}_\phi(x)$
2. Sample latent variable z from $q_\phi(z|x) = N(z; \mu_\phi(x), \tilde{\sigma}_\phi(x) I)$
3. Feed z to decoder to predict $\hat{x} = D_\theta(z)$
4. Perform gradient descent through $(-ELBO)$ $\Leftrightarrow$ $\max(+ELBO)$

(2) $\Rightarrow \mu + \sigma \cdot \varepsilon$ $\Leftrightarrow$ differentiable. ε from $N(0,1)$ by reparameterization trick. Randomness separation to deterministic

VAE Generation.

1. Sample latent value z from $p(z) = N(z; 0, I)$
2. Feed z to decoder to predict $\hat{x} = D_\theta(z)$

Generative Adversarial Networks. $\Rightarrow$ minimax game D vs. G.

Objective: $\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p(data)}[\log D(x)] + \mathbb{E}_{z \sim p(z)}[\log(1 - D(G(z)))]$

$D(x)$ = prob. input x is real.

Heuristics $\Rightarrow$ Saturating Gradients. $D(G(z))$ instead of $(1 - D(G(z)))$.

D-G 번갈아 until $D = 1/2$ $\begin{cases} D(x) & 1 \to 0.5 \\ D(G(z)) & 0 \to 0.5 \end{cases}$

$l(x, y) = -[y \log x + (1-y) \log(1-x)]$ real: 1 fake: 0