12 - RNN

$$Y_t = f(X_t, X_{t-1}, \dots, X_{t-N})$$

→ consider the history

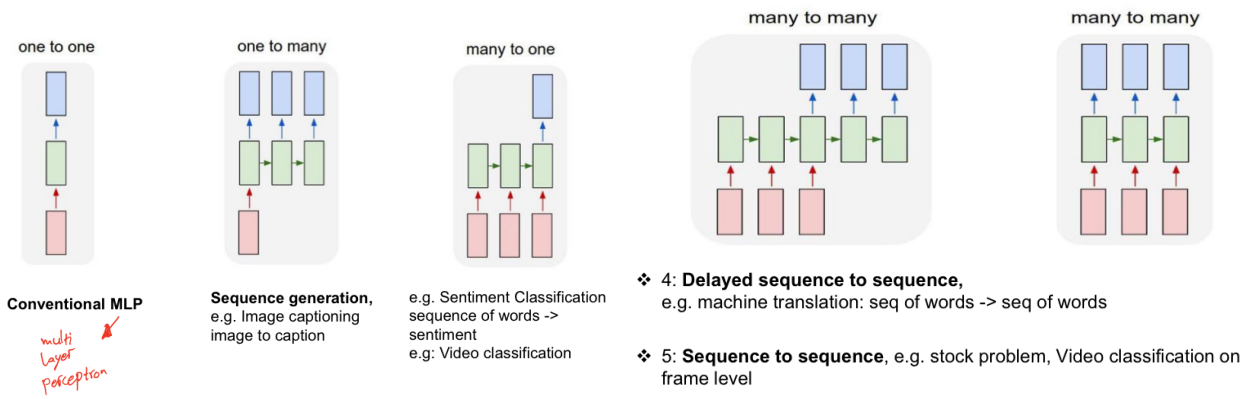
→ finite-Response model

$$Y_t = f(X_t, X_{t-1}, \dots, X_{t-\infty})$$

→ Stock market

→ Infinite-Response model

types of RNN



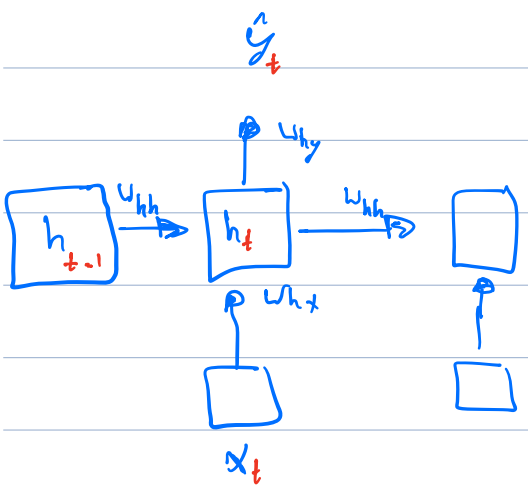
Conventional MLP

multi layer perceptron

Sequence generation, e.g. Image captioning image to caption

e.g. Sentiment Classification sequence of words -> sentiment
e.g: Video classification

- 4: **Delayed sequence to sequence,** e.g. machine translation: seq of words -> seq of words
- 5: **Sequence to sequence**, e.g. stock problem, Video classification on frame level

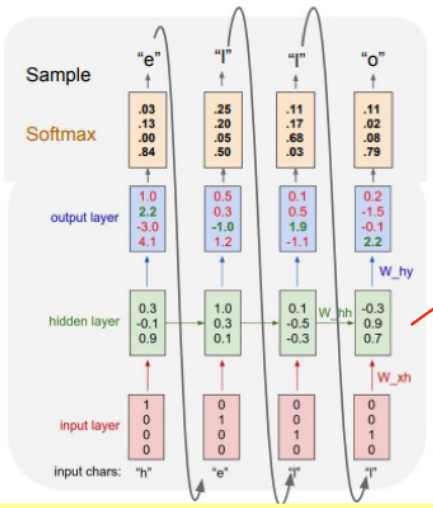


Same W in every time-step

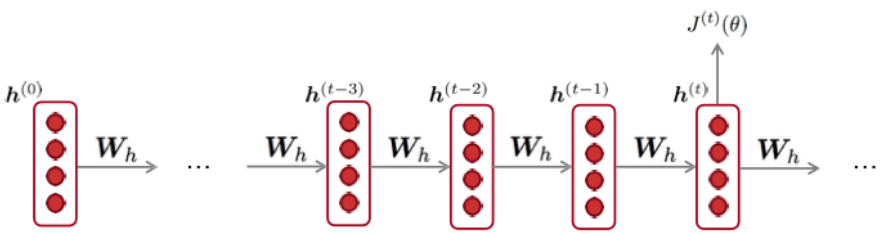
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

or σ

❖ At test-time sample characters one at a time, feed back to model



from training



Question: What's the derivative of $J^{(t)}(\theta)$ w.r.t. the repeated weight matrix W_h?

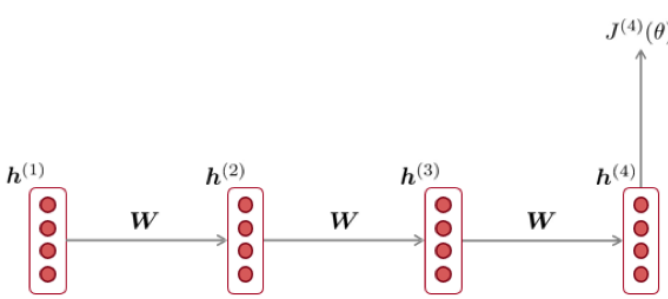
Answer:

$$\frac{\partial J^{(t)}}{\partial W_h} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial W_h}\Big|_{(i)}$$

"The gradient w.r.t. a repeated weight is the sum of the gradient w.r.t. each time it appears"

called

"backpropagation through time"



$$\frac{dJ^4}{dh^1} = \frac{dJ^4}{dh^4} \times \frac{dh^4}{dh^3} \times \frac{dh^3}{dh^2} \times \frac{dh^2}{dh^1}$$

- Recall: $h^{(t)} = \sigma\left(W_h h^{(t-1)} + W_x x^{(t)} + b_1\right)$
- Therefore: $\frac{\partial h^{(t)}}{\partial h^{(t-1)}} = \text{diag}\left(\sigma'\left(W_h h^{(t-1)} + W_x x^{(t)} + b_1\right)\right) W_h$ (chain rule)

L2-Regularization will make vanishing worse (set $W=0$)

Vanilla RNN has 2 problems:-

① vanishing the gradient: loss the history (in LM tasks = loss the syntatic and just keep the sequential recency)

when largest eigenvalue of $W_h < 1$

Language model

Solutions:-

1- LSTMs (Long-Short-term-memories)

2- GRUs (Gated-Recurrent-Units)

② exploding the gradient:- bad update (learning) = NaN or Inf

when largest eigenvalue of $W_h > 1$

Solutions:-

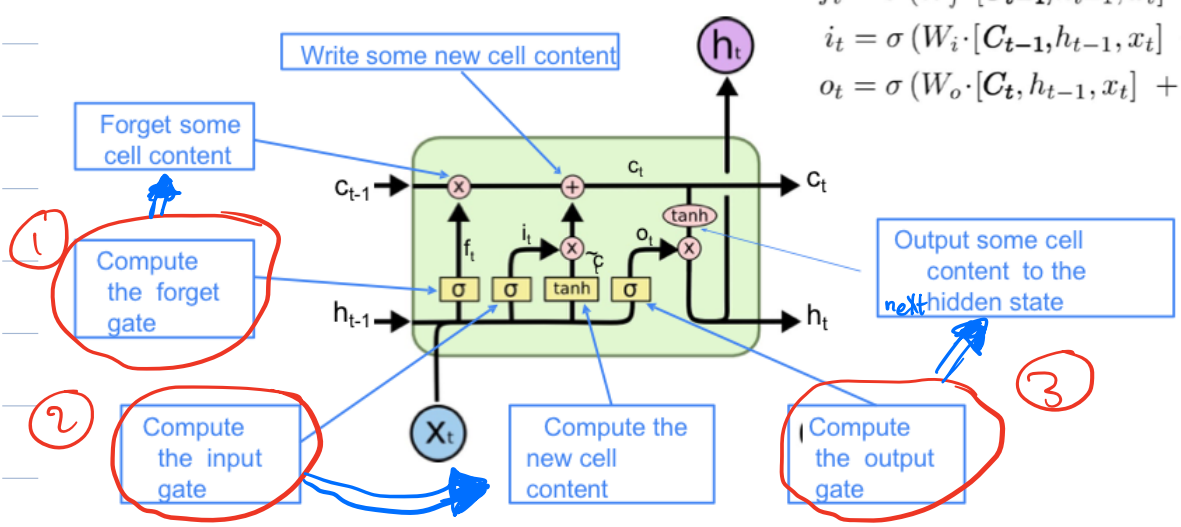
1- gradient clipping:- if it ≥ threshold ⇒ scale it befor update

- Intuition: take a step in the same direction, but a smaller step

LSTM

1 ⇒ keep
0 ⇒ forget

$$f_t = \sigma\left(W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [C_{t-1}, h_{t-1}, x_t] + b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [C_t, h_{t-1}, x_t] + b_o\right)$$

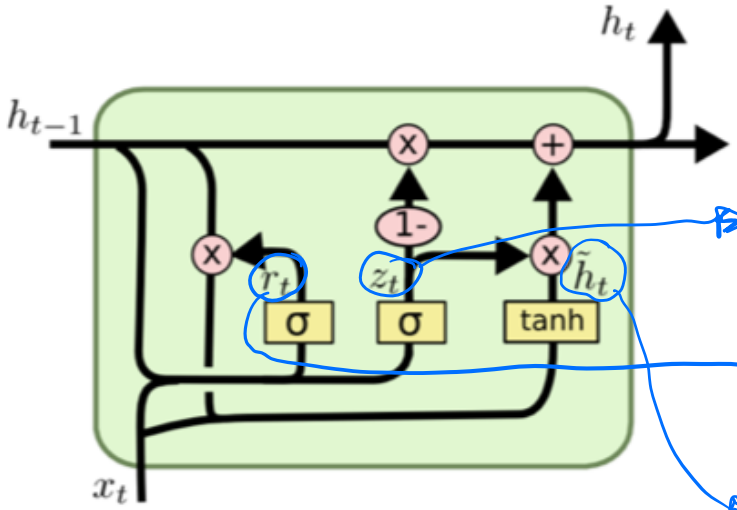


GRUs

GRU first computes an update gate

Less vanishing gradient

copy-paste the h_{t-1}



① if $z=1 \Rightarrow$ update gate $= 0$

update Gate — like input gate

②

Reset Gate

if $r_t = 0$ ignore history

new memory content

$$z_t = \sigma(W_z \cdot [x_t, h_{t-1}])$$
$$r_t = \sigma(W_r \cdot [x_t, h_{t-1}])$$
$$\tilde{h}_t = \tanh(W \cdot [x_t, r_t * h_{t-1}])$$
$$h_t = \tilde{h}_t * z_t + (1 - z_t) * h_{t-1}$$

How they solve vanishing?

LSTM = add constant-error carousel that carry information ($c_{t-1}, c_t, c_{t+1}, \ldots$)
→ good as default (with larg training data)

but only affected by a gates (forget and Input)

GRU = allow error messages to flow at different strengths depending on the inputs

↳ quicker

✳ Vanishing is not just in RNN

any deep network has this problem

Solutions:

- add more direct connections
  - ResNet skip-connection
  - DenseNet dense-connection

✳ Bidirectional RNN

- could be a vanilla, LSTM, or GRU
- every RNN has it's own W
- for classification $\boxed{\boldsymbol{h}^{(t)}} = [\overrightarrow{\boldsymbol{h}}^{(t)}; \overleftarrow{\boldsymbol{h}}^{(t)}]$ concatenation

✳ Multi-layer RNNs

stack RNN

13 - Encoder-decoder (Time asyncronance)

In situations where the start of sequence is obvious, the <sos> may not be needed, but <eos> is required to terminate sequences

Sometimes we will use a single symbol to represent both start and end of sentence, e.g just <eos> , or even a separate symbol, e.g. <s>

- The input sequence is terminated by an explicit <eos> symbol
  - The hidden activation at the <eos> "stores" all information about the sentence
- Output production continues until an <eos> is produced

at time K — take argmax — output — input — greedy → take most probable word on each step

$$\hat{y}_k^w = P(O_k = w | O_{k-1}, \ldots, O_1, I_1, \ldots, I_N)$$

Vanilla E/D is greedy decoder (no way to undo) — stops when we got <end>

Solutions:-

① Exhaustive search decoding (all possible seq.) too expensive $O(V^T)$

② Beam search decoding (keep track of the K most probable results)

no gurantee to find the optimal

Stops when:- — input of encoder

① maximum T timesteps

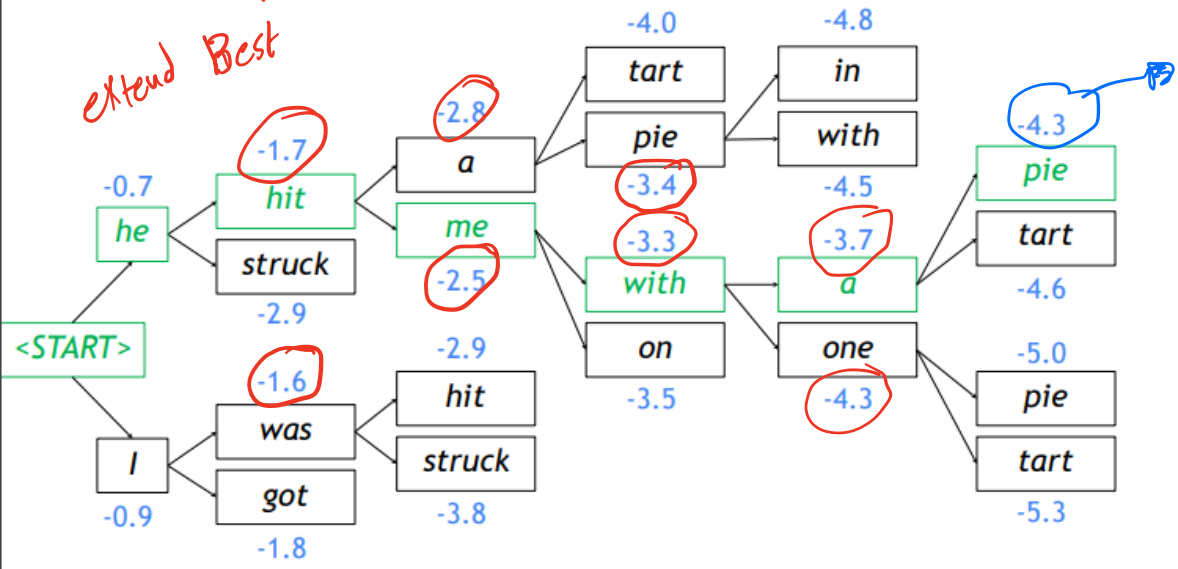
② maximum N hypotheses — output of decoder

③ got <end>

# of complete output

Beam search decoding: example

Beam size = k = 2. Blue numbers $= \text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Backtrack to obtain the full hypothesis

extend Best

highest

long hypotheses ⇓ lower scores ⇓ do normalize by length

⁂ Training E/D

- For each iteration
  - ① – Randomly select training instance: (input, output)
  - ② – Forward pass
  - ③ – Randomly select a single output y(t) and corresponding desired output d(t) for backprop
  - ④ - update parameters

* Reversing the Input

– Things work better this way

This happens both for training and during inference on test data

⁂ ways to treat videos for video captioning:-

1 - for each frame (CNN) (to extract features) then get the mean across all frame then RNN

2 - for each frame (CNN) then encoder-decoder

All types of E/D has a bottleneck problem (context vector)

Solution:- Attention

① it also helps with vanishing

② not just for Seq2Seq

❖ More general definition of attention: (input of encoder / input of decoders)

➢ Given a set of vector values, and a vector query, attention is a technique to compute a weighted sum of the values, dependent on the query.

Ways to calculate e_i :-

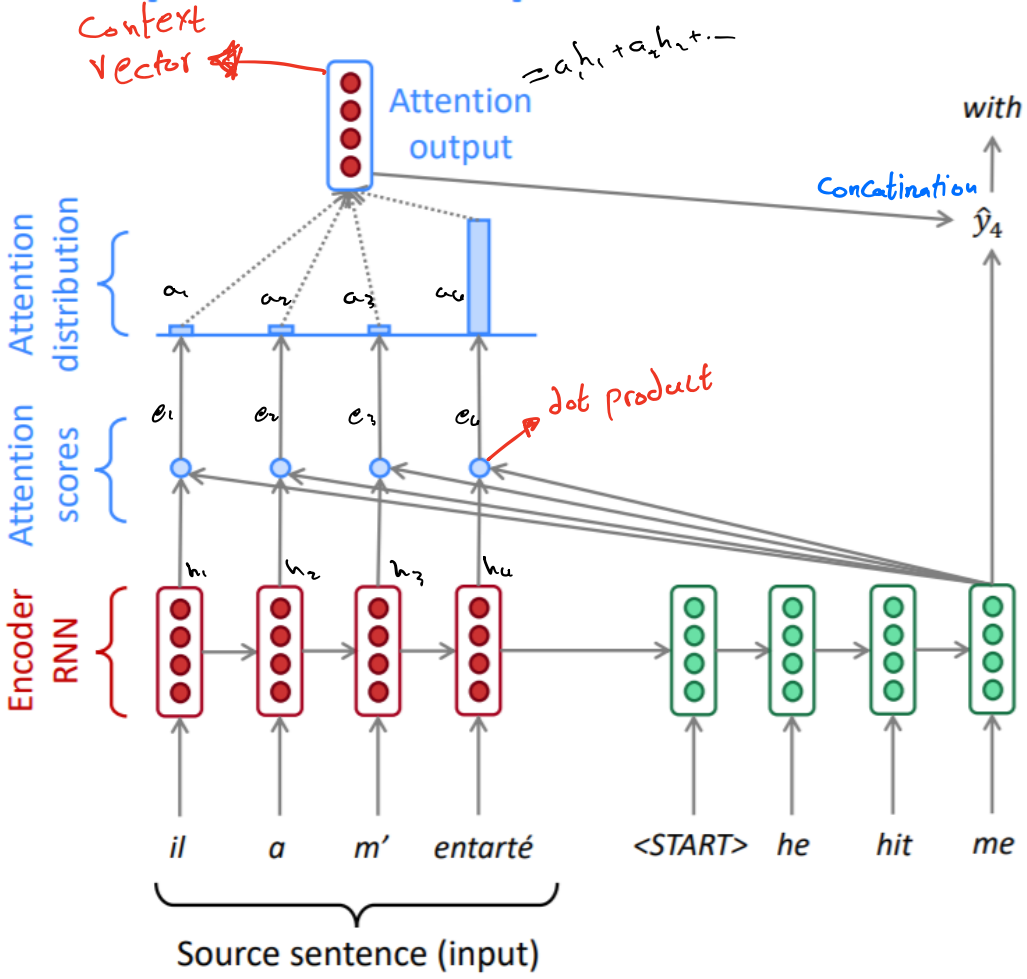
① dot-product

② multiplicative attention — multiply with W

③ additive attention — addition with W

→ hyperparameters

Sequence-to-sequence with attention

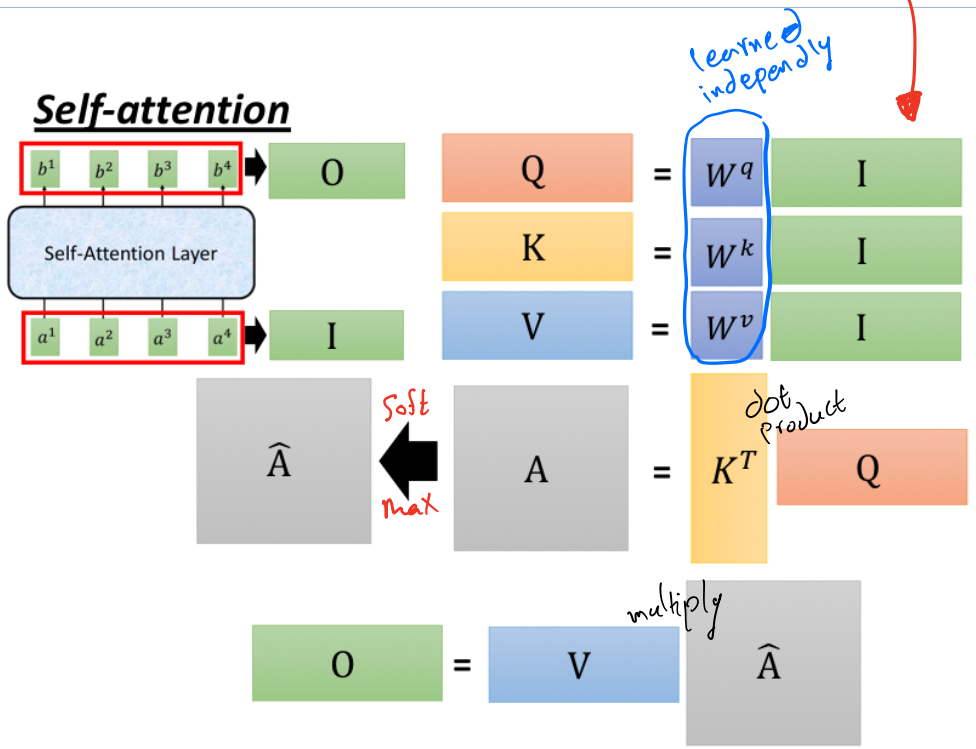


14- Self-Attention and Transformers

Self-Attentions-

- inputs = outputs
- inputs interact with each other to decide which one should have attention
- output = these interactions and attention scores

for each input q, K, V, are same (query, key, value)



We use multiple Head to focus on multiple places at once
each one will do attention independently then we concatenate them

Self Attention has problems:-

1. Doesn't care about the order (may change the meaning)
2. Doesn't have nonlinearities
3. May in decoding it look at the future

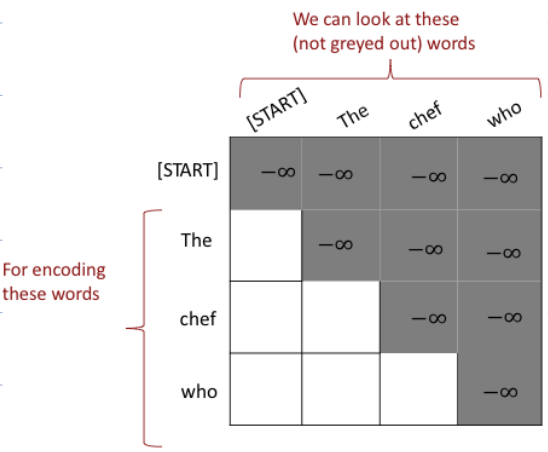
Transformers => are self-Attention and more

- positional encoding → **absolute** position is less important than **relative** position → So we use sin/cos
- Non-linearities → adding FF
- Masked decoding → just include the past word → by setting the future = $-\infty$

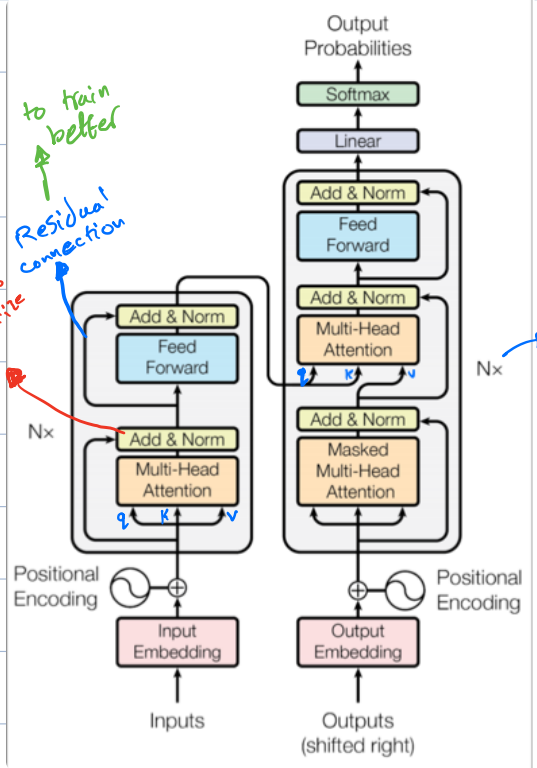
Necessities for a self-attention building block:

- **Self-attention**:
  - the basis of the method.
- **Position representations**:
  - Specify the sequence order, since self-attention is an unordered function of its inputs.
- **Nonlinearities**:
  - At the output of the self-attention block
  - Frequently implemented as a simple feed-forward network.
- **Masking**:
  - In order to parallelize operations while not looking at the future.
  - Keeps information about the future from "leaking" to the past.
- That's it! But this is not the **Transformer** model we've been hearing about?

John Hewitt



Transformers



19- GAN generative adversarial networks

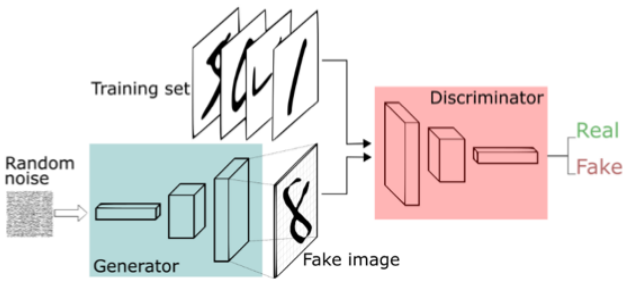
gradient descent

two player game:-

① Generative network $P(X, Y)$ Minimizer

② Discriminative network $P(Y|X)$ Maximizer

gradient ascent



Train jointly in **minimax game**

Discriminator outputs likelihood in (0,1) of real image

Minimax objective function:

$$\min_{\theta_g}\max_{\theta_d}\left[\mathbb{E}_{x\sim p_{data}}\log D_{\theta_d}(x) + \mathbb{E}_{z\sim p(z)}\log(1 - D_{\theta_d}(G_{\theta_g}(z)))\right]$$

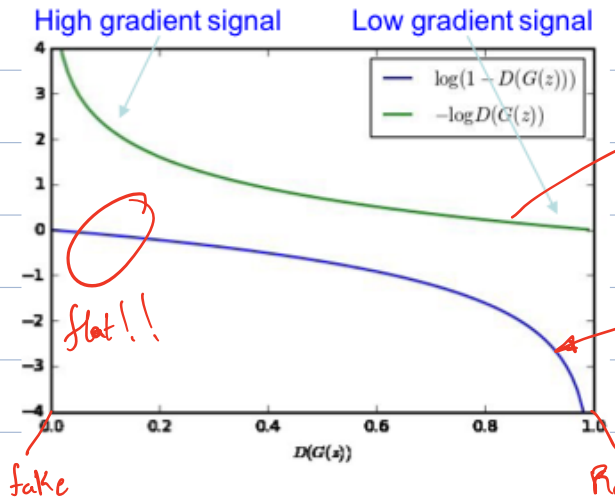
$D_{\theta_d}(x)$: Discriminator output for real data x

$D_{\theta_d}(G_{\theta_g}(z))$: Discriminator output for generated fake data G(z)

- Discriminator (θ_d) wants to **maximize objective** such that D(x) is close to 1 (real) and D(G(z)) is close to 0 (fake)
- Generator (θ_g) wants to **minimize objective** such that D(G(z)) is close to 1 (discriminator is fooled into thinking generated G(z) is real)

for generator:

gradient ascent is better



gradient ascent

gradient descent

✳ Architectures

generator = upsampling network with fractionally-strided convolutions

discriminator = CNN

Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

✳ Challenging in Training:-

① Vanishing gradient => When discriminator is very good

② Mode Collapse => When too little diversity in the generated samples

③ Normal loss metric doesn't correspond to image quality

Frechet Inception Distance is a decent choice for calc. loss

Types of GANs

① Conditional GAN

generate x (random) given y (labels) So the input for G and D are x and y

$$\min_G \max_D V(D,G) = \mathbb{E}_X \log D(X,Y) + \mathbb{E}_Z \log D(G(Z,Y),Y)$$

Example:- Edge → Image

Sketche → Image

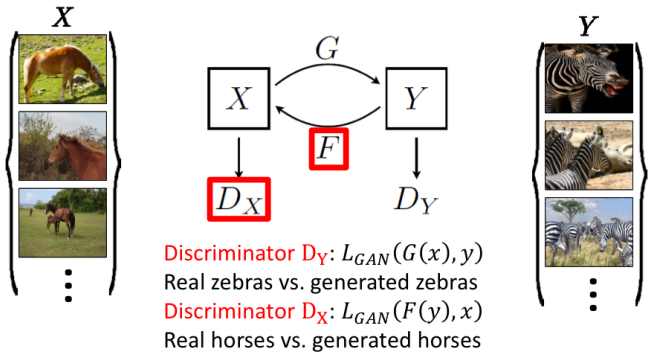
② LapGAN

Chain of conditional GANs

③ CycleGAN

pair of conditional GANs

Image-to-Image translation



Discriminator D_Y: $L_{GAN}(G(x), y)$
Real zebras vs. generated zebras
Discriminator D_X: $L_{GAN}(F(y), x)$
Real horses vs. generated horses

they use cycle consistency as loss fun.