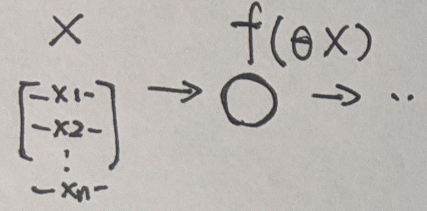
Deep Learning

- Neuron:



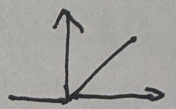
X : data matrix

$\theta = \begin{bmatrix} 1 \\ \theta_1 \\ \theta_2 \\ \vdots \end{bmatrix}$ params

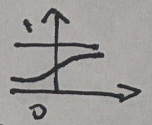
θX : linear comb of param * data

f : activation function:

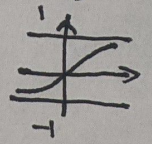
Relu:

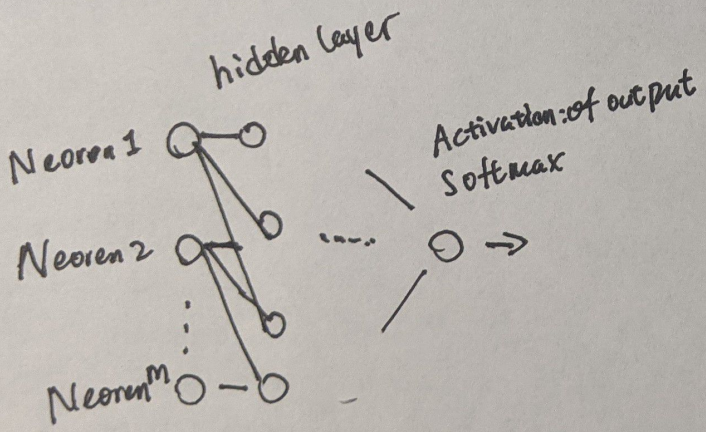


Sigmoid :



tanh:



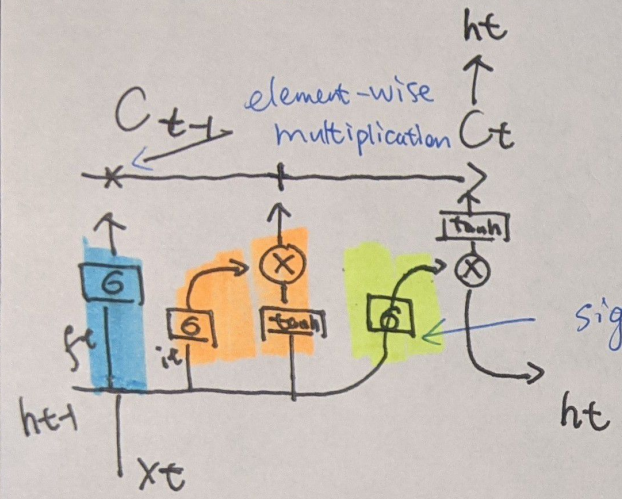


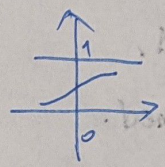
Softmax:

output a array of prob,
sum of these is 1

- Backpropergation: ~~xxxxx~~ update params by gradient descent & chain rule.

LSTM





Computation: Foget Gate Input Gate

$$\tanh\left[\sigma\left(W_f\begin{bmatrix}h_{t-1}\\x_t\end{bmatrix}\right)\sigma\left(W_i\begin{bmatrix}h_{t-1}\\x_t\end{bmatrix}\right)*\tanh\left(W_c\begin{bmatrix}h_{t-1}\\x_t\end{bmatrix}\right)\right] + \sigma\left[W_o\begin{bmatrix}h_{t-1}\\x_t\end{bmatrix}\right]$$

new Value

Output gate

params:

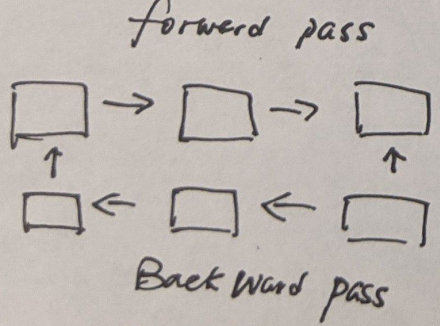
$4 \times Shape(h) \times (Shape(h) + Shape(x))$

W_f, W_i, W_c, W_o

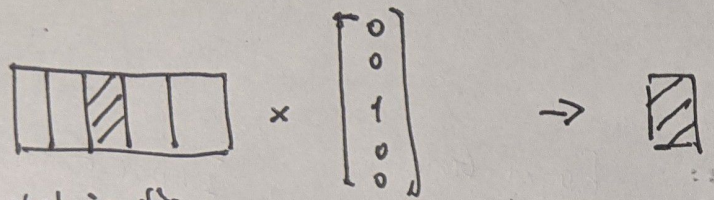
$h\begin{bmatrix}\quad\\\quad\end{bmatrix}$

$h + x$

Bidirectional Cells:



Word Embedding

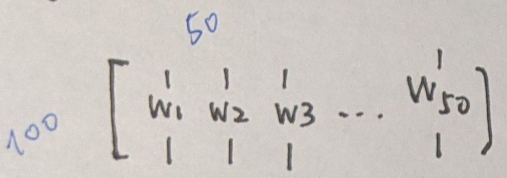


Veb in dic　　one hot encoding　　encoding for one word,
length is customed.

Model From Internship,

text classification for cleaning text.

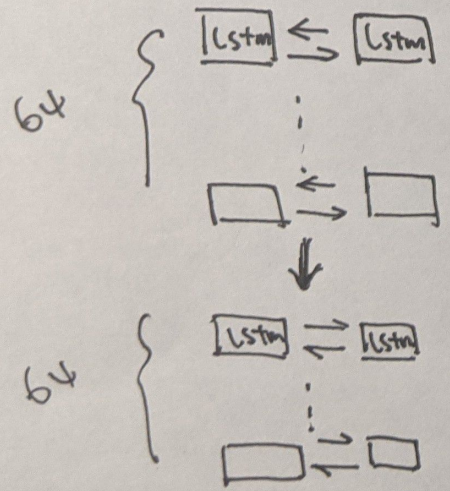
- each piece of doc are sep by " " (triple space)
- 0 as need to keep
  1 as need to remove
-



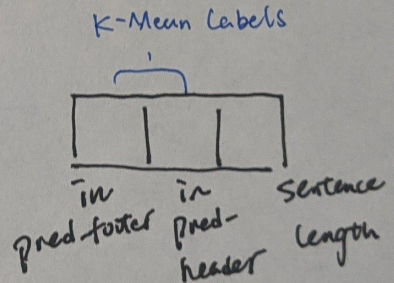
---- Embeding layer

100 for embedding vector length

50 for max length of each doc.



64 Bidirection
---- Lstm units

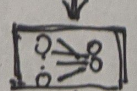


... # params:

$2 \times 4 \times [(64+100) \times 64 + 64] \approx 80k$ (bias)

use layer regularizer to prevent overfitting.

Concatenate



DNN, Relu

Sigmoid

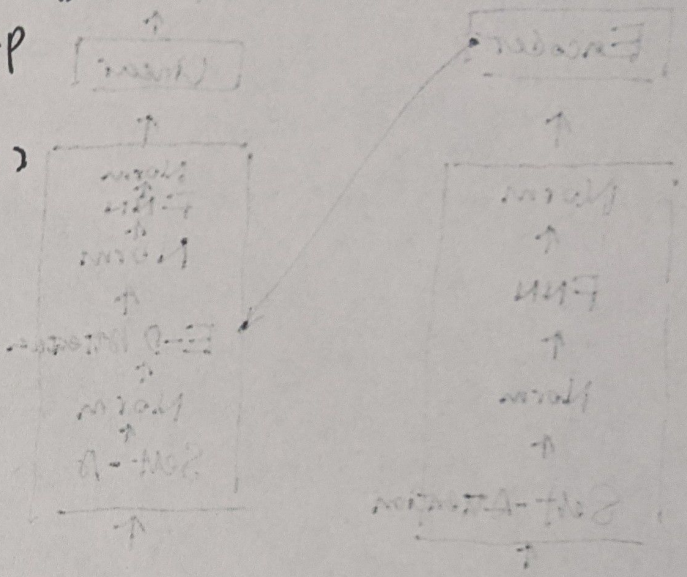
Cross Entropy Loss

- Transformer Networks

- Vector - Seq Model : Dog img → "Dog in a cup"

  Seq - vec Model : "Text..." → [0~1]

  Seq - seq Model : Translation



- RNN ↑ Transformer ↑

  x^t x^{t+1} x_1 x_2 x_3 ... — Positional Encoder:

  vector that gives context based on position of word in sentence

Seq. model

- Attention Layer. What part of the input should we focus?

E-C Attention Structure:

Enc: x_1 x_2 x_3 ... x_m

- $k_i = W_k x_i$ ...
- $v_i = W_v x_i$ ...

Dec: $x_1' x_2' \dots x_m'$

- $q_j = W_q x_j'$ ...

Formulas $C = Attn(X, X')$

$\alpha_1 = \text{Softmax}(K^T q_{:1}) \in \mathbb{R}^m$

$C_1 = \alpha_{11} v_1 + \dots \alpha_{m1} v_m = V \alpha_1$

$\alpha_2 = \dots$

$C_2 = \dots$

v_m

Notes:

C can be used to generate next word (Translation Task)

Self-Attention Layer

- Query $q_i = W_Q x_i$

  Key $k_i = W_K x_i$

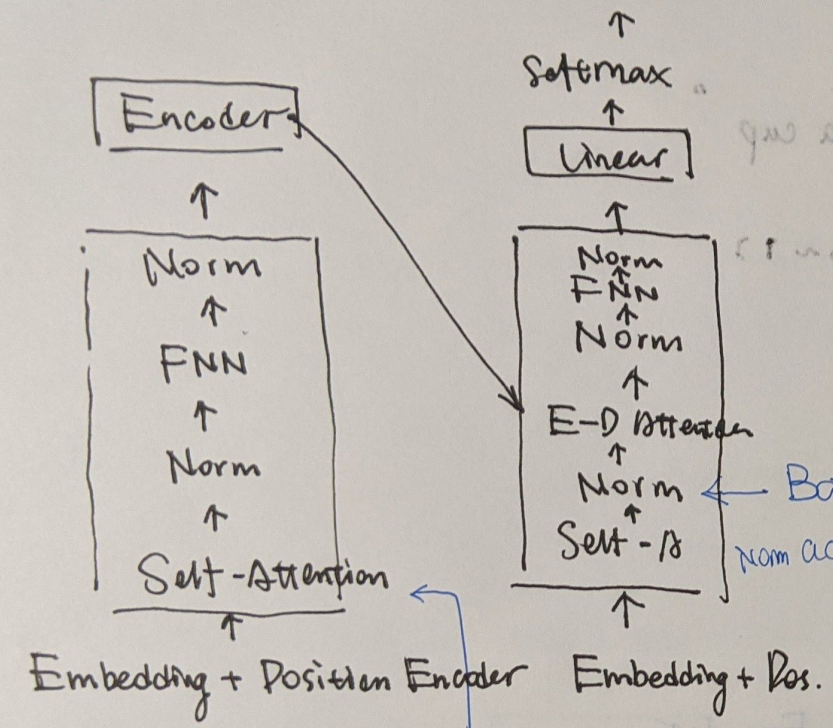
  Value $v_i = W_V x_i$

- $\alpha_j = \text{Softmax}(K^T q_{:j})$

  $c_1 = \alpha_{11} V_{:1} + \cdots \alpha_{m1} v_m = V\alpha_{:2}$ $\Big\} \Rightarrow c_{:j} = V \cdot \text{Softmax}(K^T q_{:j})$

- Self-attention layer: $C = \text{Attn}(X, X)$

- Encoder - Decoder



Batch Norm. faster Learning.

Norm across samples

different attention vectors,

avg. for each word vector

BERT

- is for pre-training Transformer's encoders
- Predict ① masked word ② next sentence.

① mask 15% words randomly.

output a prob @ masked position.

Calculate Loss = CrossEntropy(e, p)

ground-truth ↑ e, p ↑ Prob

Perform GD to update parameters

② use 50% true linked 2 sentences, 50% faked

[CLS] Sent 1 [SEP] Sent 2

↑ token for classification ↑ for seperating sentences

- Combing Two Tasks

input

example: "[CLS] Calculus is a [MASK] of math, [SEP] it [MASK] developed by newton and leibniz."

Target: true, "branch", "was"

Loss:

next-sent: binary clf: cross Entropy

masked word: multi-class Classifiaro

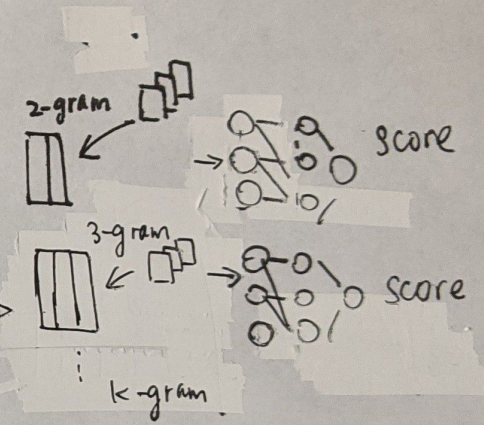
Sum of three loss

Model From Internship. , keyphrase extraction task

- Unsupervised Keyphrase Extraction:

n-grams → Pretrained Bert → Word Embedding ↑
doc → .. → Word Embedding ↓
Cosine similarity to select keywords.

- Supervised Keyphrase Extraction. (BERT-KPE)



- n-grams → Pretrained Bert (Base/SpanBert/RoBert) →

- Loss:

Hingle Loss: pairwise ranking loss $\sum \max(0, 1 - f^*(p_+, D) + f^*(p_-, D))$ phase, doc

CE Loss: is or not true keyphrase $\mathcal{L}_{chunk} = \text{CrossEntropy}(p(c_i^k = y_i^k))$

Multi-task training $\mathcal{L}_{oss} = \mathcal{L}_{Rank} + \mathcal{L}_{chunk}$.

Pytorch:

Pros : many things are modulizable

Cons : may not good for launching model into production

Code example for train model (Stats 507)

```
for epoch in range(n_epochs)
    model.train()
    for bdx, batch in enumerate(train_loader):
        # forward passing
        Logits, probs = model(...)
        #          ↑ input   ↑ output # softmax
        Cost = F.cross_entropy(Logit, true_counts)
        optimizer.zero_grad()
        Cost.backward()
        # update model parameters
        optimizer.step()
    # evaluation
    model.eval()
    Cost = Compute_epoch_loss(model, train_loader)
    :
```