1011-Notes

Wenxin Zhang

December 2021

Contents

1 Machine Learning Basics

1.1 Model Parameterization

1.1.1 Figure out p(y | x)

Given the input, what's the potential output?
- p(y|x) is proportional to exp(F(x,y))
- Here, we are interested in the distribution of y.

1.1.2 Latent variable approach p(y,z | x)

- Example: Will the car drive left or right to pass its front obstacle? This may depend on the driver's hand habit(z: the driver is right-handed or left-handed).
- p(y,z|x) is proportional to exp(F(x,y,z))
- p(z|x) is proportional to exp(F(x,z))

1.1.3 Multi-class Classification

$y \in \{1, ..., c\}$
$F(x, y) = w_y^T \phi(x) + b_y$
$\text{p(y|x)} = \frac{exp(w_y^T \phi(x)+b_y))}{\sum_{y'=1}^{c} exp(w_{y'}^T \phi(x)+b_{y'})}$

1.1.4 Mixture Density Model

[In statistics, a mixture model is a probabilistic model for representing the presence of sub-populations within an overall population, without requiring that an observed data set should identify the sub-population to which an individual observation belongs.](WIKI)
$y \in R^{d'}$; $z \in \{1, ..., c\}$
$F'(x, z) = w_z^T \phi(x) + b_z$
$F(x, y, z) = -||y - w_\mu^z \phi(x)||^2$

$$p(y|x) = \sum_{z=1}^{c} p(y, z|x) = \sum_{z=1}^{c} p(y|z, x)p(z|x)$$

$$p(y|x) = \sum_{z=1}^{c} [\frac{exp(w_z^T \phi(x) + b_z)}{\sum_{z'=1}^{c} exp(w_{z'}^T \phi(x) + b_{z'})}][\frac{exp(-||y - w_\mu^z \phi(x)||^2)}{\int exp(-||y' - w_\mu^z \phi(x)||^2) dy'}]$$

1.2 Defining a loss function

Consider only one example: $f(x, y^*)$, the higher $p(y^*|x)$ the model gives, the better the function is. Loss(Negative Log Likelihood Loss) is defined as: l=-logp(y^*|x).

For the training dataset: $D = (x_1, y_1^*), (x_2, y_2^*), ...(x_N, y_N^*)$;

$p(D) = \prod_{n=1}^{N} p(x_n, y_n^*) = \prod_{n=1}^{N} p(y_n^*|x_n)p(x_n)$, where $p(x_n)$ is uniform.
$logp(D) = \sum_{n=1}^{N} logp(y_n^*|x_n) + constant$
Here, we choose log: since p(x) is uniform, we can just consider it as a constant, so that we can replace multiplication with addition to simplify.

1.2.1 Multi-way Classification

NLL $= -logp(y^*|x) = [-w_{y^*}^T \phi(x) - b_{y^*}] + log \sum_{y'=1}^{c} exp[w_{y'}^T \phi(x) + b_{y'}]$
To minimize the NLL:
- let $y^*, \phi(x)$ be close to each other, so that $y^*\phi(x)$ is larger; let b_{y^*} be as large as possible.
- let $y', \phi(x)$ be far away from each other, so that $y'\phi(x)$ is small; let $b_{y'}$ be as small as possible.

1.3 Optimization

Optimization: Stochastic Gradient Descent(Mini-Batch).
L(D; θ): a data-level loss function
L(D; θ) = $\frac{1}{|D|} \sum_{(x,y)\in D} \ell(x, y; \theta)$

1.4 Model Selection

Hyper-parameter Selection/Optimization
- Training Set D for training/optimization
- Validation Set D for model selection
- Test Set D for generalization ability test

1.5 Reporting

2 Text Classification

2.1 Classification Problem

2.1.1 What does translation mean?

Input: natural language
Input is actually a sequence of discrete symbols
$X = (w_1, w_2, ..., w_T)$ where w_t belongs to V(vocabulary)

To predict Y based on X, where Y $\in$ {1, 2, ..., C} is to translate.

2.1.2 Multi-Classification Problem

$Y \in \{1, ..., c\}$
$F(X, Y) = w_Y^T \phi(X) + b_Y$
p(Y|X) = $\frac{exp(w_Y^T \phi(X) + b_Y))}{\sum_{Y'=1}^{c} exp(w_{Y'}^T \phi(X) + b_{Y'})}$

2.2 Encoding(One-hot Embedding & Dense Embedding)

2.2.1 One-hot Encoding

$$|e(w) - e(u)| = c, \forall w \neq u$$

one-hot encoding (1-q-k):

$$1h(w) \in \{0, 1\}^{|V|}$$

$$1h(w)^i = 1, \ if \ i = hash(w); \ 1h(w)^i = 0, otherwise$$

2.2.2 Dense Embedding

$$e(w) = W1h(w) = W[:, hash(w)]$$

W is the weight matrix, where $W \in R^{d' \times |V|}; 1h(w) \in R^{|V| \times 1}; e(w) \in R^{d'}$

2.2.3 Continuous Bag Of Words

$$X = (e(w_1), e(w_2), e(w_3), ..., e(w_T))$$

$$\phi(X) = \sum_{t=1}^{T} e(w_t)$$

2.3 Linear Classification Problems

$$F(X,Y) = u_Y^T \phi(X) + b_Y = u_Y^T (\sum_{t=1}^{T} e(w_t)) + b_Y = \sum_{t=1}^{T} u_Y^T e(w_t) + b_Y$$

$$a = U\phi(X) + b$$

Mention that: F(X,Y) is the y_{th} coordinate of Y.
Assume that $b_y = 0$:

$$logp(Y^*|X) = u_{Y^*}^T \phi(X) - log \sum_{Y=1}^{c} exp(u_Y^T \phi(X))) = F(X, Y*) - logZ(X)$$

1. Gradient with respect to feature vector

$$\nabla_{\phi(X)} = u_{Y^*} - \sum_{Y=1}^{c} \frac{exp(u_Y^T \phi(X))}{\sum_{Y'=1}^{c} exp(u_Y^T \phi(X))} u_Y = u_{Y^*} - E_{Y|X}[u_Y]$$

$$\nabla_{e(w)} = \nabla_{\phi(X)} \frac{\partial \phi(X)}{\partial e(w)} = \nabla_{\phi(X)} = u_{Y^*} - E_{Y|X}[u_Y]$$

Hence, Gradient with respect to feature vector: Push towards the correct class embedding and push far away from the incorrect class embedding.

2. Gradient with respect to class embedding

$$\nabla_{u_{Y^*}} = \phi(X) - \frac{exp(u_{Y^*}^T \phi(X)}{\sum_{Y'=1}^{c} exp(u_{Y'}^T \phi(X))} \phi(X) = \phi(X) - p(Y^*|X)\phi(X) = (1 - p(Y^*|X))\phi(X)$$

$$\nabla_{u_Y} = -\frac{exp(u_Y^T \phi(X))}{\sum_{Y'=1}^{c} exp(u_{Y'}^T \phi(X))} \phi(X) = -p(Y|X)\phi(X) = (0 - p(Y|X))\phi(X)$$

Hence, Gradient with respect to class embedding also acts as the resort of the alignment.
Here, the correct class would have gradient: $u_{Y^*} = (1 - p(Y^*|X)\phi(X)$, which pushes towards the feature vector; while the wrong class would have gradient: $u_Y = (0 - p(Y|X))\phi(X)$, which pushes far away from the feature vector.

$\nabla_{e(w)}$ *and* $\nabla_{u_{Y^*}}$ would be estimated simultaneously, and initialized randomly!

2.4 Non-Linear Classification Problems

In a movie review, what if the "hate" and "love" both appear? The linear classifier may not be strong enough. Therefore, we introduce non-linear classification and creat a NN manually.

2.5 Back-Propagation

3 Language Modeling(Distribution Hypothesis)

3.1 Language Modeling

What can be a right way to represent a word?
- **Context** of the word can represent meaning of itself.

A statistical language model is a probability distribution over sequences of words. Given such a sequence, say of length m, it assigns a probability $p(w_1, w_2, ...w_m)$ to the whole sequence.(WIKI)

Fully Factorial: $p(x_1, x_2, ..., x_T) = p(x_1)p(x_2)...p(x_T)$; order doesn't matter
Chain Rule of Probability: $p(x_1, x_2, ..., x_T) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2)...p(xT|x_1, x_2, ..., x_{T-1})$
Latent Variable Models: $p(x_1, x_2, ..., x_T) = \int p(z) \prod_{t=1}^{T} p(x_t|z)dz$

Goal of Language Modeling: $p(x; \theta) > p(x'|\theta)$ if $x \sim D^*$ & $x' \not\sim D^*$
Learning process of Language Modeling: 1. Maximize LogLikelihood(correct answers | input x) OR 2.Minimize LogLikelihood(wrong answers | input x)

3.2 Auto-regressive Modeling

$$L(\theta; D) = -\frac{1}{|D|} \sum_{x \in D} logp(x; \theta) = -\frac{1}{|D|} \sum_{n=1}^{|D|} \sum_{t=1}^{T_n} logp(x_t|x_1 : x_{t-1}; \theta)$$

$logp(x_t|x_1 : x_{t-1}; \theta)$: per word
$\sum_{n=1}^{T_n} logp(x_t|x_1 : x_{t-1}; \theta)$: per sentence
$\sum_{n=1}^{|D|} \sum_{t=1}^{T_n} logp(x_t|x_1 : x_{t-1}; \theta)$: per document

3.3 Cont-based Estimation

$$p(x_t|x_{<t}) = \frac{p(x_{<t}, x_t)}{p(x_{<t})} = \frac{p(x_{<t}, x_t)}{\sum_{x' \in r} p(x_{<t}, x_{t'})} = \frac{c(x_{<t}, x_t)}{\sum_{x' \in r} c(x_{<t}, x_{t'})}$$

c(): count the occurrence
This method faces the issue of sparsity, since $|V(vocabulary)| >> |D(corpus)|$
Due to the large size of $|V|$ and the limit size of $|D|$, here, the $p(x_t|x_{<t})$ would intend to be 0, but it doesn't imply the impossibilities here.

3.4 N-th order — Markov Assumption

$$p(x_t|x_1, x_2, ..., x_{t-1}) = p(x_t|x_{t-n} : x_{t-1})$$

consider that

$$\frac{|D|}{O(|V|^n)} >> \frac{|D|}{O(|V|^T)}$$

$$p(x_t|x_{<t}) = p(x_t|x_{t-n} : x_{t-1}) = \frac{c(x_{t-n}, ..., x_t)}{\sum_{x' \in r} c(x_{t-n}, ..., x')}$$

To smooth the Markov Assumption, we can assume that the word must appear at least z times.

$$p(x_t|x_{<t}) = p(x_t|x_{t-n} : x_{t-1}) = \frac{c(x_{t-n}, ..., x_t) + z}{\sum_{x' \in r} c(x_{t-n}, ..., x') + z}$$

3.5 Feed-forward Language Modeling

$$F(x_{t-n}, x_{t-n+1}, ..., x_{t-1}, x) = u_x^T \phi(x_{t-n}, x_{t-n+1}, ..., x_{t-1}) + b_x$$

where F() = class embedding $\times$ word embedding + bias

4 Recurrent Language Modeling

Recurrent Language Modeling: A language model that can handle unbounded context(prefix).

$$p(x_1, x_2, ..., x_T) = \prod_{t=1}^{T} p(x_t|x_1, ...x_{t-1}) \approx p(x_t|x_{t-n}, ..., x_{t-1})$$

4.1 Neural Probabilistic Language Model

$$e(x_{t-n}, ..., e_{t-1}) = [e(x_{t-n}^T, ..., e_{t-1}^T)]^T \in R^{nd}$$

$$h = tanh(We(x_{t-n}, ..., x_{t-1}) + b) \in R^{d'}, where\ W \in R^{d' \times nd}$$

Compression of the input: $\phi(X) = \sigma(w \sum_{t=1}^{T} e(x_t) + b)$
Class embedding: $F(X, Y) = u_Y^T \phi(X) + e_Y$

4.2 Recurrent Neural Network Language Model

The intuition of RNN is to recursively compress the input sequence/ whatever we have seen so far/ prefix of any sequence into a single vector.
if $h_{t-1} = \phi(X = (w_1, ..., w_{t-1}))$, how can we add w_t?

$$h_t = R(h_{t-1}, w_t) = \sigma(w_r h_{t-1} + w_x e(w_t) + b)$$

where $h_{t-1}, h_t \in R^{d'}; w_r \in R^{d' \times d'}; w_x \in R^{d' \times d}; e(w_t) \in R^d; b \in R^{d'}$
$h_0 \in R^{d'}$ *or* $h_0 = 0$

RNN consist of multiple **repeated layers** of non-linear functions.

4.3 Perplexity

[In information theory, perplexity is a measurement of how well a probability distribution or probability model predicts a sample. It may be used to compare probability models. A low perplexity indicates the probability distribution is good at predicting the sample.](WIKI)

5 Vanishing Gradient and Gated Recurrent Units

5.1 Sequence Classification/ Regression

5.2 Vanishing Gradient & Exploding Gradient

$f(x) = R\sigma(W^T X_T + U^T \sigma(W^{T-1} X_{T-1} + U^{T-1} \sigma(...\sigma(W^1 X_1 + U^1 h_0 + b^1)...) + b^{T-1}) + b^T) + c$

$$\frac{\partial \ell}{\partial U} = \frac{\partial \ell}{\partial f}(\sum_{t=1}^{T} \frac{\partial f}{\partial h_t} \frac{\partial h_t}{\partial U^T})$$

$$\frac{\partial \ell}{\partial W} = \frac{\partial \ell}{\partial f}(\sum_{t=1}^{T} \frac{\partial f}{\partial h_t} \frac{\partial h_t}{\partial W^T})$$

$$\frac{\partial \ell}{\partial b} = \frac{\partial \ell}{\partial f}(\sum_{t=1}^{T} \frac{\partial f}{\partial h_t} \frac{\partial h_t}{\partial b^T})$$

$$\frac{\partial f}{\partial h_t} = \frac{\partial f}{\partial h_T} \frac{\partial h_T}{\partial h_{T-1}} ... \frac{\partial h_{t+1}}{\partial h_t}$$

$h_t = \sigma(a_t) = \sigma(W x_t + u h_{t-1} + b)$; $\frac{\partial h_t}{\partial h_{t-1}} = diag(\sigma'(a_t))u$, $where\, \sigma'(a_t) = 0$, $if\ a_t \leq 0; and = 1\ o.w.$

$$||\frac{\partial h_t}{\partial h_{t-1}}|| = ||diag(\sigma'(a_t))u|| \leq ||diag(\sigma'(a_t))|| \cdot ||u|| = max(\sigma'(a_t))) \cdot ||u|| \leq ||u||$$

$$\frac{\partial f}{\partial h_t} \leq ||\frac{\partial f}{\partial h_T}|| \cdot ||\frac{\partial h_T}{\partial h_{T-1}}||...||\frac{\partial h_{t+1}}{\partial h_t}|| \leq ||\frac{\partial f}{\partial h_T}|| \cdot ||u||^{T-t}$$

Exploding gradient may happen when $||u|| >> 1$;

Vanishing gradient may happen when $||u|| < 1$;

5.3 Linear Shortcuts/ Residual Connection/ Gated Recurrent Unit

Logic Line So Far:
To predict the next word, many solutions are raised. Word Embedding:
step 1. One Hot: Feature Vector Sparsity Problem
step 2. N-Gram(CBOW & Skip-gram): N-Gram appearance Sparsity Problem

Sequence Embedding:
step 1. Recurrent Neural Network: Vanishing Gradient or Exploding Gradient
step 2. Linear Shortcuts

$$h_t = \sigma(U h_{t-1} + W x_t + b) + \sum_{t'=1}^{t-1} g_t([h_{t'}]^t)$$

$$\frac{\partial h_t}{\partial h_{t'}} = \prod_{t''}^{t-t'} \frac{\partial h_{t-t''+1}}{\partial h_{t-t'}} + \frac{\partial h_t}{\partial g_t([h_{t'}]^t)} \frac{\partial g_t([h_{t'}]^t)}{\partial [h_{t,}]^t}$$

Residual Connection: $g_{t-1}(a) = Ia;\ g_{t'}(a) = 0\ for\ t' < t-1$

$$h_t = \sigma(Uh^{t-1}+Wx^t+b)+h_{t-1} = \sigma(Uh^{t-1}+Wx^t+b)+\sigma(Uh^{t-2}+Wx^{t-1}+b)+h_{t-2}$$

$$h_t = \sum_{t' \leq t} \sigma(Uh_{t'-1} + Wx_{t'} + b)$$

$$||h_t|| \leq \sum_{t' \leq t} ||\sigma(Uh_{t'-1} + Wx_{t'} + b)||$$

There can still be exploding/vanishing gradient problems. Here, we introduce update gate(u_t) to convexly combine two bounded functions! This combination guarantees h_t be bounded.

$$h_t = U_t \bigodot \sigma_n(Uh_{t-1} + Wx_t + b_n) + (1 - U_t) \bigodot h_{t-1}$$

$$U_t = \sigma_u(Uh_{t-1} + Wx_t + b_u) \in [0, 1]^d$$

5.4 Networks with update gates

- Residual Networks
- Highway Networks
- Gate Recurrent Networks
- Long Short Memory Networks
- Google Networks

6 Attention and Masked Language Modeling

6.1 Break the Recurrence Dependence

Attention is all you need.

6.1.1 Update Gate/Forget Gate

$$h_t = u_t \bigodot \hat{h_t} + (1-u_t) \bigodot h_{t-1} = u_t \bigodot \sigma(Uh_{t-1} + Wx_t + b) + (1-u_t) \bigodot h_{t-1}$$

$$h_1 = u_1 \bigodot \hat{h_1} + (1-u_1) \bigodot h_0$$

$$h_2 = u_2 \bigodot \hat{h_2} + (1-u_2) \bigodot u_1 \bigodot \hat{h_1} + (1-u_2) \bigodot (1-u_1) \bigodot h_0$$

$$h_3 = u_3 \bigodot \hat{h_3} + (1-u_3) \bigodot u_2 \bigodot \hat{h_2} + (1-u_3) \bigodot (1-u_2) \bigodot u_1 \bigodot \hat{h_1} + (1-u_3) \bigodot (1-u_2) \bigodot (1-u_1) \bigodot h_0$$

$$h_t = \sum_{t'=1}^{t} w_{t'}(x_{t'}, h_{t'-1}) \bigodot \hat{h_{t'}}(x_{t'}, h_{t'-1})$$

At any time of t, we look into the hidden activation of the gated RNN, it's the weighted sum of the candidate vectors at all of the previous layers. And this can happen due to the linear short connections that they create.

6.1.2 Implement of Parallel Computing:

Next, we want to break this temporal/ recurrence dependency, so that we can compute parallelly using GPU to improve the efficiency.

$$h_t = \sum_{t'=1}^{t} w_(x_{t'}, x_t, t', t) \bigodot \hat{h}(x_{t'}, t')$$

where t' is positional embedding.

$$w(x_{t'}, x_t, t', t) = \frac{exp(Q(x_t, t)^T K(x_{t'}, t'))}{\sum_{t'=1}^{t} exp(Q(x_t, t)^T K(x_{t'}, t'))}$$

$$\hat{h}(x_{t'}, t') = V(x_{t'} + p(t'))$$

where Q() represents query; K() represents key; V() represents the "value"; p() represents the position embedding;

About the weights:
Here we use vector product (dot product) to tell how good the compatibility is.
- Query * Key tells us whether the particular row in the key table is compatible with the query.
- Use SoftArgMax to normalize them, so that the weights are non-negative and sum to 1.

6.2 Multiple Attention Heads

$$h_{t,m} = \sum_{t'=1}^{t} w_m(x_{t'}, x_t, t', t) \bigodot \hat{h_m}(x_{t'}, t')$$

6.3 Masked-out Indices

Problem: Predict the missing parts
Model:

$$argmax_y \ log \ p(x_{sequence \ prefix}, \ y_{missing \ part}, \ x_{sequence \ suffix})$$

Analysis:
- Without language model, we can hardly solve this problem.
- Recap the language model, it scores how probable a given sentence is.
- We are going to try all possible ways to fill in the missing part, and get the one with the highest probability. However, this can be troublesome! The missing parts y can have variable length; we should predict what the missing parts are.

$$argmax_y \ [log \ p(y_{missing \ part}|x_{prefix}, \ x_{suffix}) + logp(x_{prefix}, \ x_{suffix})]$$

$$argmax_y \ [log \ p(y_{missing \ part}|x_{prefix}, \ x_{suffix})]$$

For the sequence

$$(x_1, x_2, ..., x_t, < mask >, < mask >, ..., < mask >, x_{t+|y|}, ..., x_T)$$

- If the length of y is 1, it's like a usual classification problem.
- But if the length of y is really long, we should introduce the in-painting method.

6.4 BERT/ [Masked] Langauge Modeling

Masked-out Indices: $m_1, m_2, ..., m_{T_m}$
Observed Indices: $o_1, o_2, ..., o_{T_o}$
where $T_m + T_o = T$

$$p(x_{m_1}, x_{m_2}, ..., x_{m_{T_m}}|corrupt(x)) = \prod_{i=1}^{T_m^n} p(x_{m_i}|corrupt(x))$$

Objective function:

$$J(\theta) = \frac{1}{N} \sum_{n=1}^{N} E_{m,o \sim Y(T_n)}[\sum_{i=1}^{T_m^n} logp(x_{m_i}|x_{o_1}, x_{o_2}, ..., x_{o_x}; \theta)]$$

- Go through N examples
- T_n represents length of the nth example
- T_m^n represents the number of missing words within the nth example
- Get the log probability of all of the missing words given all of the observed ones

There are many ways to define probability distribution. One way is to define how we generate samples. Without knowing what the probability is, we can come up with some kind of stochastic process, just want to get as much as samples.

Three ways to define probability distribution:
1. Turn this mask language model into auto-regressive language model.
(left to right modeling)

$$p(x) = \prod_{t=1}^{T} p(x_t | x_1, x_2, ..., x_{t+1} =< mask >, ..., x_T =< mask >)$$

- $\mathbf{p(x)}$: correctly given all of the previous ones up until x_{t-1}, while all of the future ones are masked.
- Intuitively: We trust this model has to learn everything that comes before, and will miss everything that come after. And it tries to predict exactly what we want.
- The space of partitioning is pretty large, for every word, we can imagine that we are like posting a coin. Extending the sequence would exponentially increase the size. Hence, we need enough data to learn from. In reality, this method cannot work well.

To deal with the curse of dimensionality (weights' space increases exponentially), we turn this mask language modeling into a score function.
2. Turn this mask language model into a score function. Pseudo-likelihood

$$F(x) = PLL(x) = \sum_{t=1}^{T} logp(x_t | x_1, x_2, ..., x_T)$$

$$p(x) = \frac{exp(F(x))}{\sum_{x'} exp(F(x'))}$$

- This pseudo-likelihood can approximate exponential complexity using linear complexity.
- Here, we are given as many as possible words to recove just one missing variable. Once we have the score F(x), we normalize it over all of the possible sequences.
- Two tricky reasons: 1) It's hard to compute this normalization constant. 2) We cannot get the exact distribution to sample from.

Actually we don't know the exact distribution, but we just need to know how

to sample from the true distribution.

3. Sample from the true distribution for k=1,2,...,K; we uniformly sample an index i at random.

$$\hat{x_i} \sim p(x_i|x_1, ...x_{i-1}, < mask >, x_{i+1}, ..., x_T)$$

Building and training language model is pretty universal to solve any problems, since human language is like how we encode.

7 Semi-Supervised Learning and Transfer Learning in NLP

7.1 Classification: Constrained Optimization

- Classification: Solving a constrained problem.
- There are a number of parameters that we have to estimate.
- Solve the optimization problem with the constraints for each sample. (In fact, all of the machine learning problems can be conduced as solving a constrained optimization problem.) We want to find the beta that's going to satisfy these constraints.

8 Conditional Language Modeling

8.1 Machine Translation

What we have learned so far is how to build a language model; language model is a black box that tells us how likely or how probable the given sequence is, under the dataset that we used to train this language model.

1. Language modeling seems to encapsulate a lot of problems that we aim at solving.
- Natural language is how we convey ideas from one to another.
- LM can be powerful, since it can frame any sophisticated, low/high-level questions into natural language.

2. Language modeling can be extremely fundamental.
- In the sense of text classification, the idea of LM is to identify one of the classes that the input sequence belong to.

Parameterization:
Auto-regressive modeling:
Model:

$$p(y|x) = \prod_{t=1}^{T_y} p(y_t|y_{<t}, x)$$

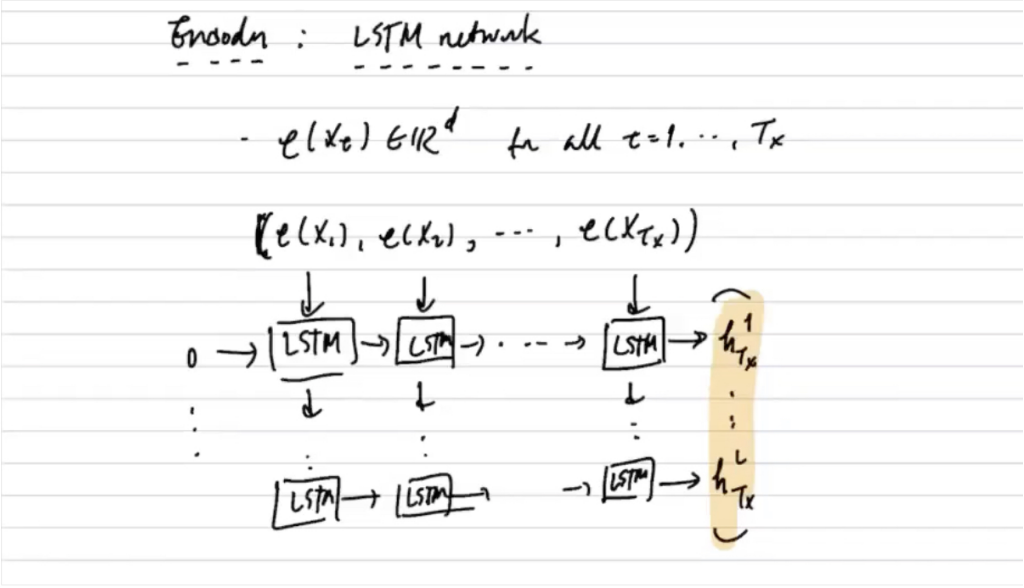
Related Paper: [Sequence to sequence learning with Neural Networks]

8.2 Encoder & Decoder: LSTM Network

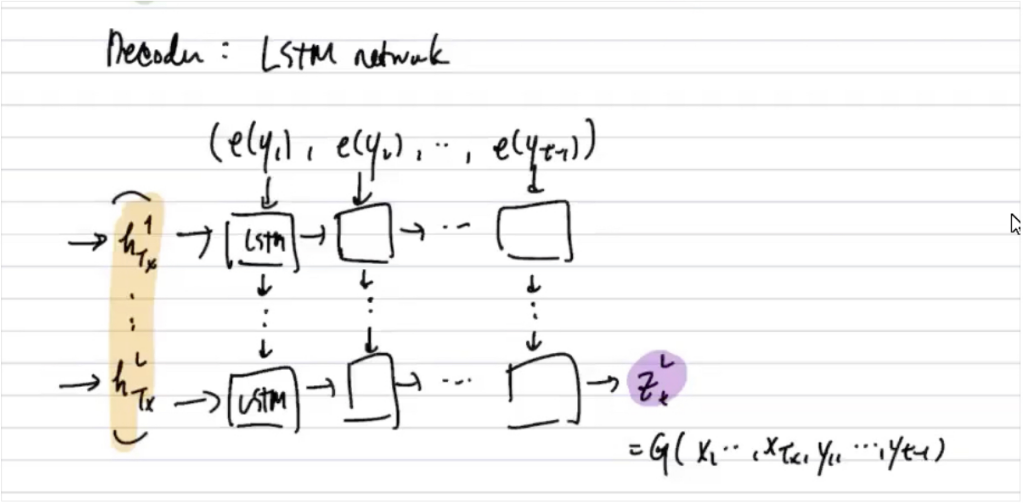
- Use the same set of parameters between the Encoder and the Decoder.

Disadvantages:
- Encoder had a pretty huge burden of having to compress the content of sentences into a fixed size of vector.
- Short and tiny architecture cannot address long sequences well.

- Encoder:



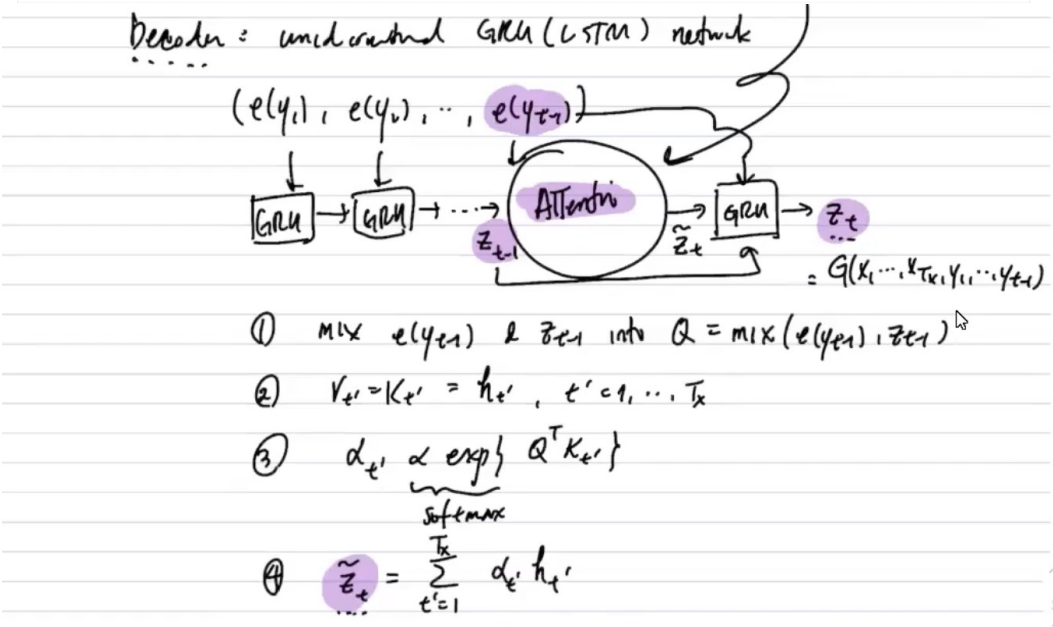
- Decoder:



8.3 Encoder: Bidirectional GRU/LSTM Network; Decoder: Undirectional GRU/LSTM Network

Encoder:

- Bidirectional GRU/LSTM network
- We are still using the continuous embedding
- We are not going to look at the final hidden layers, but all of the hidden layers



Related Paper: Neural Machine Translation by Jointly Learning to Align and Translate

9 Matrix Factorization and Language Modeling

9.1 Matrix Factorization

PCA, SVD...Many different ways to factorize matrix

Matrix would allow us to represent data in a way that:
- ROW: example in the dataset
- COLUMN: value that we assign to one of the features of the dataset

Distribution Hypothesis:
- word context: meaning of the word is determined by other words that appear together
- masked language modeling

Consider one word w: which word does w appear together with within some distance(small window)?
$\Phi(x)$: a feature vector of w; this can be a large vector that is as large as the number of the words; and this vector can be seen as the co-occurrence vector of w. $\Phi(w)_i$ represents the count of the word i that appears in the context of w.

9.2 Co-occurrence matrix:

- size: $\|V\| * \|V\|$ (It can be a large and sparsity matrix.)
- How to choose size of the window is up to us.

Once we have the vectorzied feature of the word, we can always check the similarity by checking at the dot product.

- Idea of the mean vector μ: if we don't know which word we are talking about, what would be my best guess.
- Subtract the mean: center all the words representations

- The objective function: look into the ith & the jth word, $\Phi(w_{ij})$ tells us how many times these two words can appear together.
- $\Phi(w_{ij}) = u_i^T v_j$; where u_i represents the ith word and v_j represents the jth word.