1. **Overfitting**

Reason:

* model is too complex and has too many parameters
* the training dataset is small
* the distribution of training dataset and test dataset is different

Solution:

1. Using simple model: to decrease the complexity of model, we can remove some hidden layer if there exists any and reduce the number of neurons to make the network smaller.
2. Changing a larger dataset
3. data augmentation (computer vision): Multiple similar images can be generated using data augmentation. This can help us increase the dataset size and reduce overfitting. Because the model cannot overfit all the samples as the amount of data increases, it has to generalize.
4. regularization: Add a regularization term after the loss function. If the data is too complex to model accurately, L2 is a better choice because it is able to learn the intrinsic patterns present in the data. And when the data is simple enough to model accurately, L1 is more suitable.
5. Dropout: It randomly drops neurons in the neural network during each iteration of training. When we discard different sets of neurons, it is equivalent to training different neural networks. Different neural networks can overfit in different ways, so the net effect of discarding will be to reduce the occurrence of overfitting.
6. Early stop: after several iteration of training, the training error doesn’t have significant decline but the test error starts increase. Early stop is a method that shut down training before the test error increase too high.
7. **Analysis of train loss and test loss**

* The **train loss** keeps dropping, and the **test loss** keeps dropping, indicating that the network is still learning;
* The **train loss** keeps decreasing, and the **test loss** tends to remain unchanged, indicating that the network is overfitting;
* The **train loss** tends to remain unchanged, and the **test loss** continues to decline, indicating that the data set is 100% problematic;
* The **train loss** tends to remain unchanged, and the **test loss** tends to remain unchanged, indicating that learning has encountered a bottleneck, and the learning rate or batch number needs to be reduced;
* The **train loss** continues to rise, and the **test loss** continues to rise, indicating that the network structure is improperly designed, the training hyperparameters are improperly set, and the data set has been cleaned.

**The reason why training loss doesn’t decrease.**

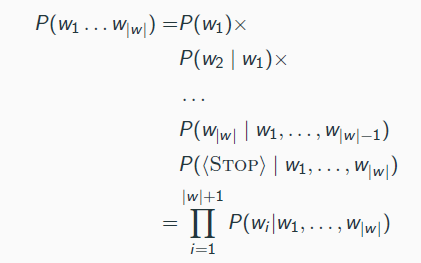
Weight initialization is not correct. In a neural network, if the weights are initialized to 0, or other uniform constants, it will cause subsequent activation units to have the same value. All units are the same, meaning they are all computing the same feature, and the network becomes the same as only one hidden unit. Like a layered node, this makes the neural network lose the ability to learn different features.

Activation function, loss function is not appropriate, which leads to the loss is 0 and model weight doesn’t have the ability to update correctly.

1. **Language model and probabilistic model**

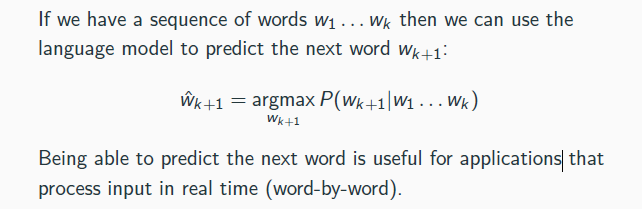
Defines a joint distribution over infinite sample space in terms of conditional distributions, each over finite sample spaces (but with potentially infinite history!)

n-gram models make all terms finite with a Markov assumption, the probabilities must be non-negative and sum to 1.



Sample zero: it appears but not sampled

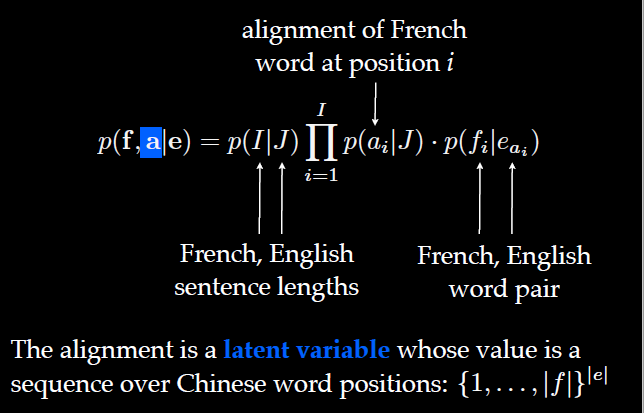
Structure zero: it shouldn't appear in the language



**Problem of machine translation when using conditional language model.**

* model forget Chinese sentence after generating first n-1 words of English
* sentences might not have the same length or have the same word order

**IBM model:**



**EM model:**

Basic idea:

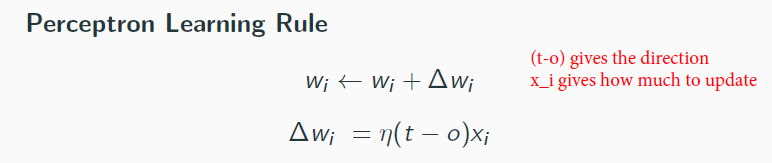
1. arbitrarily select a set of parameters (i.e. uniform)
2. calculate the expected counts of the unseen events
3. choose a set of parameters that maximize the likelihood using the expected counts as proxy for observed counts.

E-step: computes the expected counts of an alignment link between a target word and source word

M-step: normalises the expected counts for a source word by the expected counts for the source word involved in that alignment.

1. **Perceptron**

y = f(u(x)), f(x) is nonlinear; u(x) is linear



**Multilayer perceptron:** are feed-forward neural network, the weight could be zero.

Q: Why using many hidden layers rather than a big layer?

A: The big layer is hard to train because it has more parameters, which may also lead to overfitting problem. It is better to separate the big layer into multiple layers. A\*B >= A+B

1. **Feed-forward language model**

Multilayer perceptron (universal function approximators) aka neural networks have exactly one data type: vectors.

Softmax function: converts vector into a probability distribution.

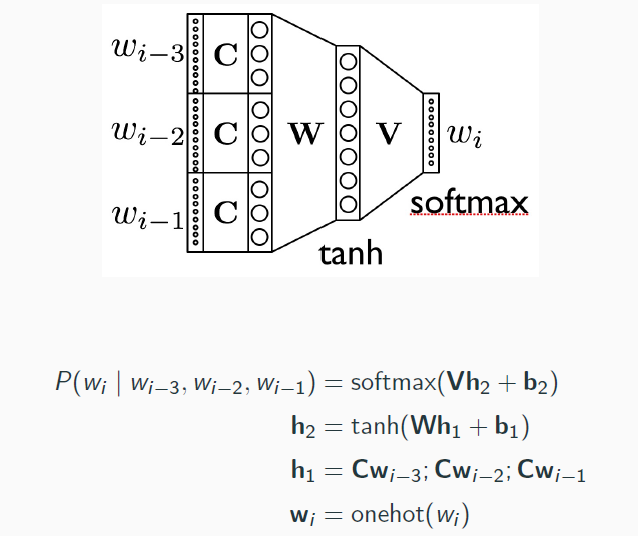
The feedforward model can be easily extended to larger n, which might be advantageous, while n-gram model are parameter sensitive when increasing n.

The views of learning: (1) choose parameters to maximize likelihood when estimating an n-gram model; (2) update parameters to minimize error as in the perceptron.

All the partial derivatives called gradient.

Learning rate:

* Small leads to convergence.
* Very small may takes long time.
* Large leads to divergence.



C: Word Embedding

W: contextual embedding

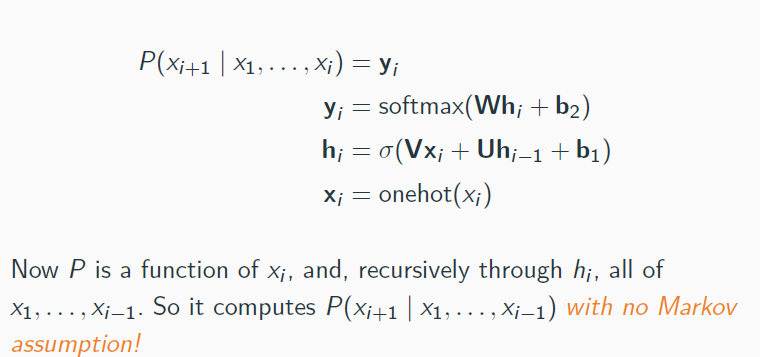
SGD: update weight by using an example at each iteration to minimize cross-entropy loss

Mini-batch GD: update weight by using a batch of examples at each iteration

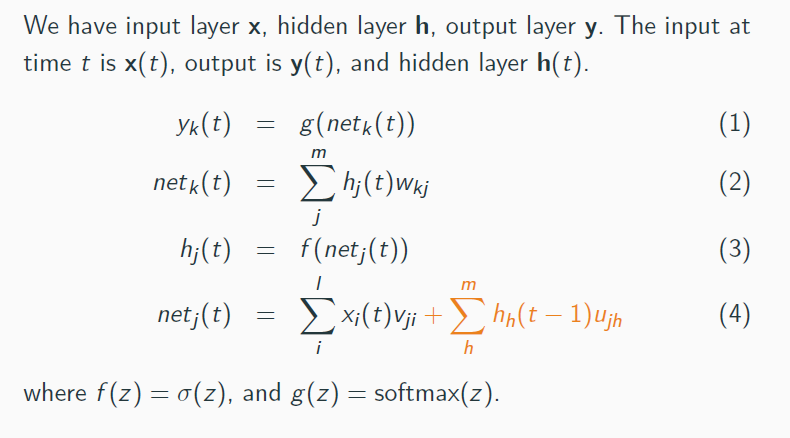
1. **RNN**

Q: Why need to remove the Markov assumption?

A: The real-world data input is various in length, Markov method need to fix the input length, which may cause sparse input or cannot have ability to deal with the long dependency (large context).



**RNN forward propagation:**

Gradient vanishing: RNN is not able to learn long-range dependencies well, as the gradient vanishes: at every time step, we multiply the weights with another gradient. The gradients are < 1 so the deltas become smaller and smaller.

1. **LSTM**

Idea: deal with arbitrary length input

Q: Why LSTM could solve the vanishing gradient problem?

A: The memory cell is linear, so its gradient doesn’t vanish

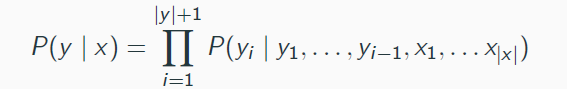
LSTM through BPTT: (review later)

<http://arunmallya.github.io/writeups/nn/lstm/index.html#/>

1. **Seq2Seq**

The desirable property is that the model is a universal approximator for sequence-to sequence functions since inputs and outputs are now unique given the additional information of positional embeddings.

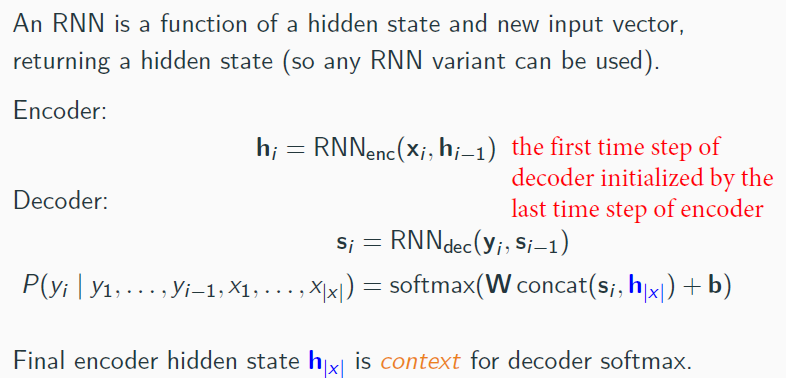
Model the machine translation from x to y



We could treat sentence pairs as one long sequence, but:

* The source and target model may have different vocabulary and morphosyntactic structure

The first time step of decoder is initialized by the last time step of encoder



The shortage of Encoder-Decoder structure:

* Hidden representation (the last encoder unit’s output) needs to represent the sentence at any length but its size is fixed.
* RNN has gradient vanishing.
* The representation (the last encoder unit’s output) has a recency bias: it mostly represents final word of the input.
* Target words whose corresponding source words are at the beginning of the sentence therefore depend on information that originates further away in the computation graph.

Decoder cannot be bidirectional, because the right information is missing.

1. **Attention**

Attention computes a distribution over source words, learn which input words are important for the decoder.

Instead of just using the last hidden state of encoder, attention uses weight average over all hidden states at encoder.

**Attention solves a problem:** we have a variable length input, and we want to summarize it into a fixed vector, focusing only on the relevant parts of the input.

The relevance

Query = output hidden state

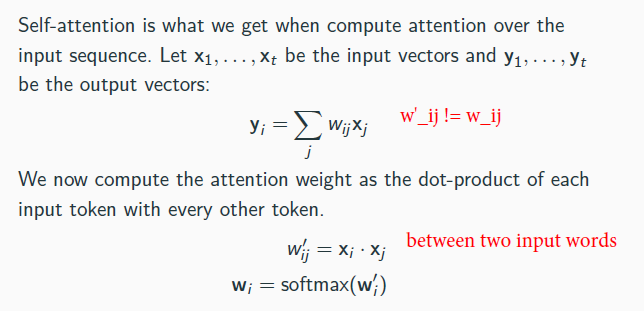
Key = encoder output (input vector)?

Value = encoder output

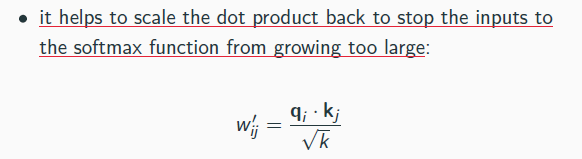
The relation between dependency and attention?

**Self-attention：**The self-attention weight is computed from the dot-product of each input token with every other token. Because the weight is computed from input, the self-attention only relies on input.

The QKV of self-attention are all from the input x.



Scaling the pot-product of K\*Q

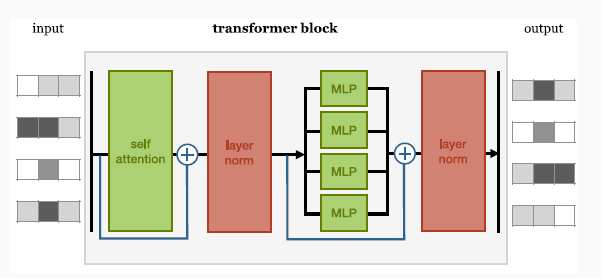


**Multi-head attention:** a word could relate to many words for different head (purpose)

Narrow self-attention: k/h \* k/h

Wide self-attention: weight matrix k\*k

1. **Transformer**



**Position encoding:** Transformer cannot directly represent the word orders; position encoding is then added to the input embedding.

**Masking:** The model can no longer look forward and just copy the upcoming input, which is possible for autoregressive way for sequence-to-sequence tasks

**key padding mask** is used to make the model ignore padding tokens within the encoder-decoder attention layer (i.e. ensure values corresponding to padded tokens are not baked into the representation of other words). **attn mask** is used to prevent the model from cheating (i.e. prevent it from copying the representation of future words in the input into words at an earlier timestep).

**The transformer has only two sources of non-linearity**: the feedforward layer and the SoftMax in the self-attention.

**Classification using transformer**: stack transformers and use them for classification. We apply mean pooling to the output and map it onto a class vector.

1. **Word Embedding**

**Pre-training:** train a generic source model on a standard, large dataset, a way of initializing the parameters of your target model to good values.

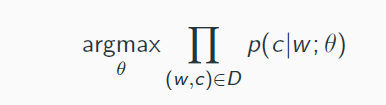
**Finetuning:** then take the resulting model, keep its parameters, and replace the output layer to suit the new task. Now train this target model on the dataset for the new task.

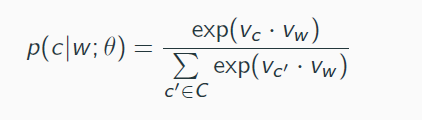
* Finetuning often truncate the last layer when you replace it
* Use a smaller learning rate (already initialize a good weight)
* Weight freezing

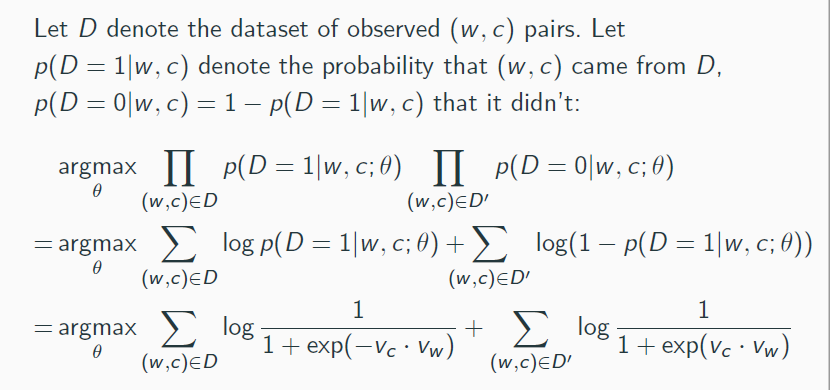
**Static Word Embedding**: design a fixed vector to each word, independent of its context.

**CBOW:** usesthe words within small window to predict the current word.

**Skipgram**: uses the current word to predict the context words

Skipgram’s objective function: 



* Seek for parameter values such that the dot product is maximized
* Impractical because the denominator is over all context c’.
* The hidden layer does not use activation function, because non-linearity does not improve the performance of the model (any other model need non-linearity?)
* Problem: if large output layer (size equal to vocabulary size)
* Negative sampling objective 

**Contextual Word Embedding**: They assign a vector to a word that depends on its context, i.e., on preceding and following words.

**BERT (bidirectional Encoder Representations from Transformer):**

Basic BERT architecture:

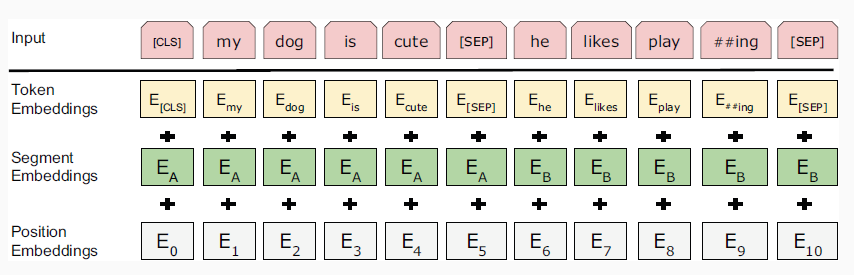
* Multi-layer bidirectional transformer
* L: transformer blocks; H: dimensionality of hidden layer; A: number of self-attention heads;
* Bert Base: L = 12, H = 768, A = 12, 110M parameters;
* Bert Large: L = 24, H = 1024, A = 16, 340M parameters.

Input: could be <Q, A> pairs, single sentence, or other string of tokens

If first token is always [CLS]: aggregate sentence representation for classification tasks;

Sentence pairs are separated by [SEP] token; and by segment embedding

The input need to go through token embedding, segment embedding and position embedding.



**Masked Language Model:**

Instead of predicting the next word, we train the model to predict the whole context.

Because copying in self-attention could achieve the highest attention score by dot-product, we need to mask 15% of the tokens in the input sequence; train the model to predict this.

However, due to [mask] makes mismatch between pre-training and finetuning, we change the previous masking schema into:

* The [mask] token 80% of the time;
* Random token 10% of the time;
* unchanged token 10% of the time;

The reason of this masking schema:

* if we always use [MASK] token, the model would not have to learn good representation for other words;
* if we only use [MASK] token or random word, model would learn that observed word is never correct;
* if we only use [MASK] token or observed word, model would just learn to trivially copy.

**Next sentence prediction (includes QA, NLI)**: generate training data 50% is the true pairs and other 50% is random pairs

Pre-training and finetuning with BERT:

Input:

* sentence pairs in paraphrasing
* hypothesis-premise in entailment
* question-answer pairs in QA
* text-0 pairs in text classification task or sequence tagging

Output:

* sequence of tokens in task such as QS
* sequence of tagging labels in NER
* [CLS] representation is fed into an output layer for classification, such as entailment or sentiment.

Bert cannot tell how good is the paraphrasing but only tell from the language model whether the output is paraphrasing or not.

1. **Neural Parsing**

Encoder-decoder for parsing

* Add [END] symbol
* Reverse the input
* Make network deeper
* Add attention
* Use pre-trained word embedding
* A lot of training data

Q: How to make sure the opening and closing bracket match?

A: This situation is very rare, is it occurs, just fix the bracket with post-processing.

Q: How to associate the word with leaves of tree?

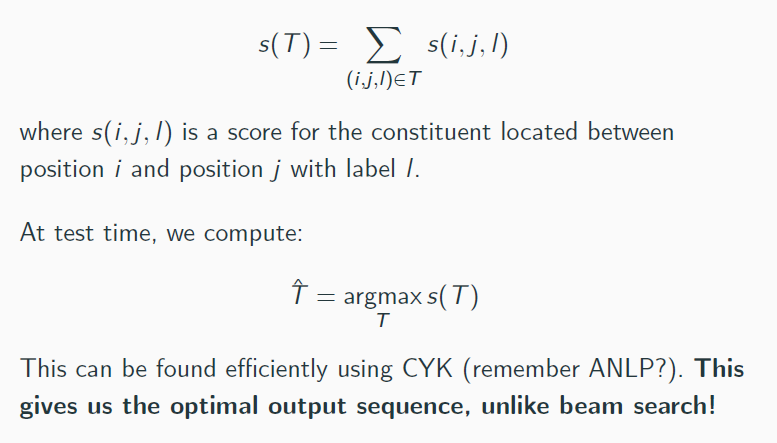
A: we could associate each input word with a PoS in the output.

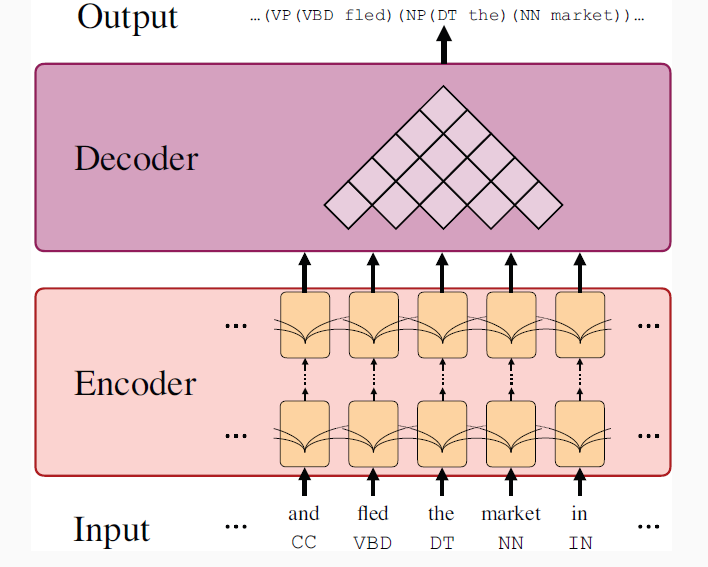
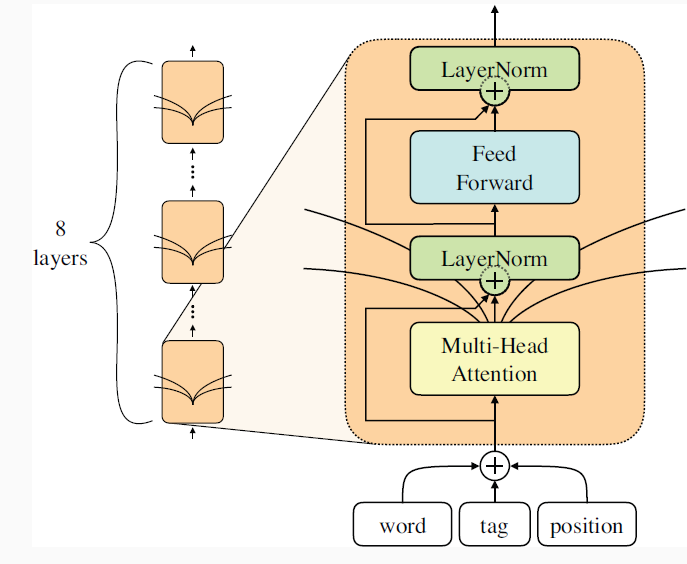
Q: How to make sure the model output the overall best sequence?

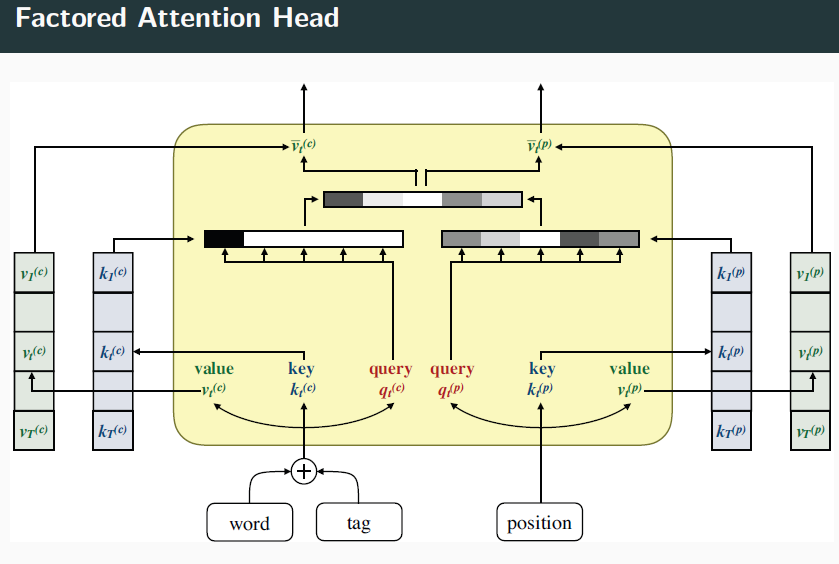
A: Beam search.

**The encoder-decoder model only works when attention is used.**

Parsing with transformer:

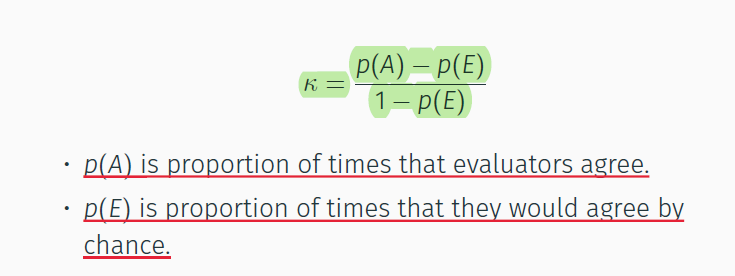


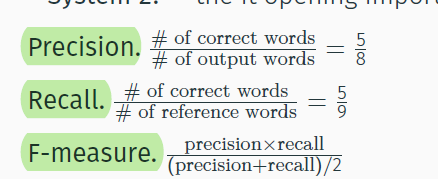


1. **Evaluating of machine translation**

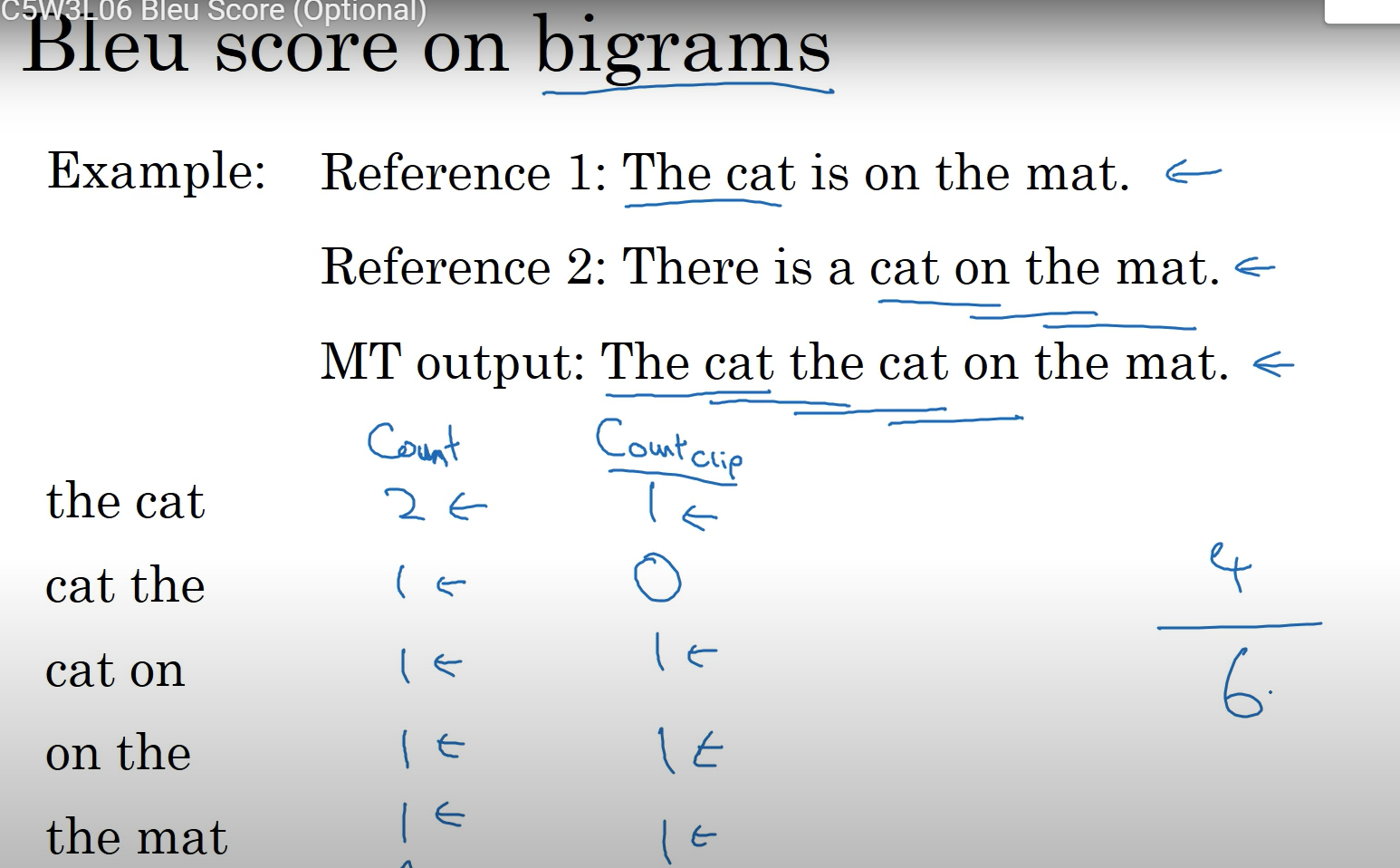
A good translation needs both adequate and fluency.



Evaluate automatically:



BLEU score:



惩罚参数BP:当存在多个参考译文时，选取和翻译译文最接近的长度

BLEU’s drawback:

* BLEU treat punctuation the same as other context words
* Poor proxy of adequate and fluency
* Score human translation low
* Not interpretable across datasets

1. **Open vocabulary model**

**Non-solution**: ignore rare word, replace the word out of vocabulary with UNK, although the vocabulary covers 95% of words, but it’s not enough, because rare word often have high self-information.

**Approximative softmax:** at training time, the vocabulary based on the words occurring in training set; at test time, determine likely target words based on source text (still not open)

**Back-off model:** replace rare words with UNK at training time, when system produce UNK, align UNK to source word, and translate this with back-off model (hard to model 1-many relationships; hard to predict inflection with back-off dictionary)

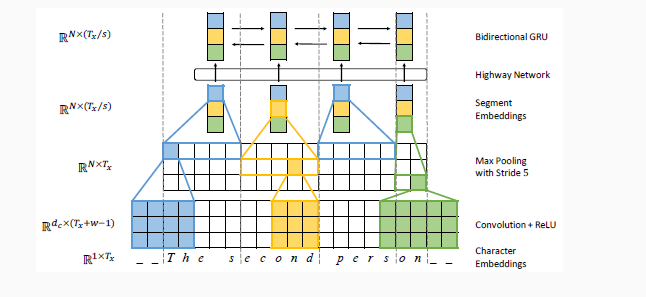
**Subword NMT**:

BPE (byte pair encoding): character level representation, repeatedly replace most frequent symbol pairs (A, B) with AB.

BPE-dropout: most frequent words are intact in vocabulary, learns how to compose with infrequent words; sometimes forget to merge, we will learn how words compose, and better transliteration

**Character-level NMT**:

Most open-vocabulary, no heuristic or language specific segmentation (drawback: increase the sentence length)

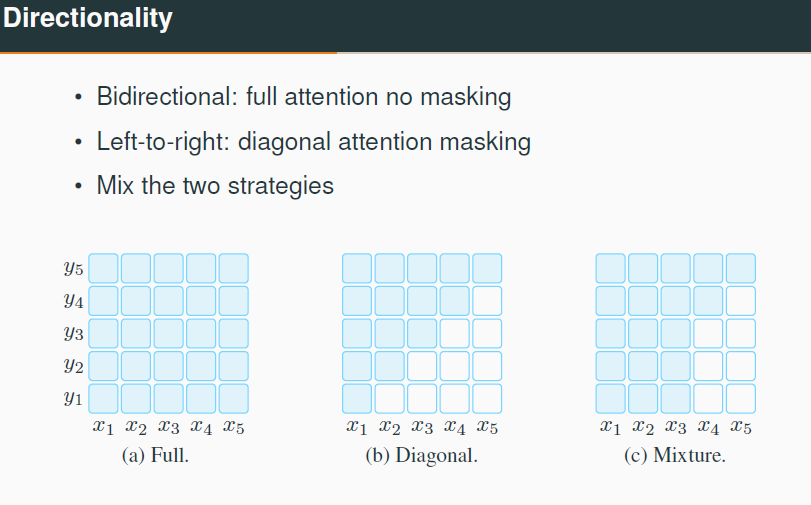


1. **Prompting**

Pre-training language model:

* Main objective: Predict P(x\_i|x\_<i)
* Denoising function: x’ = f\_noise(x), predict P(x|x’)
  + corrupted text reconstruction, loss over noised part
  + full text reconstruction, loss over entire input
* Auxiliary objective:
  + Next sentence prediction
  + Discourse relation prediction
  + Image region prediction

Directionality

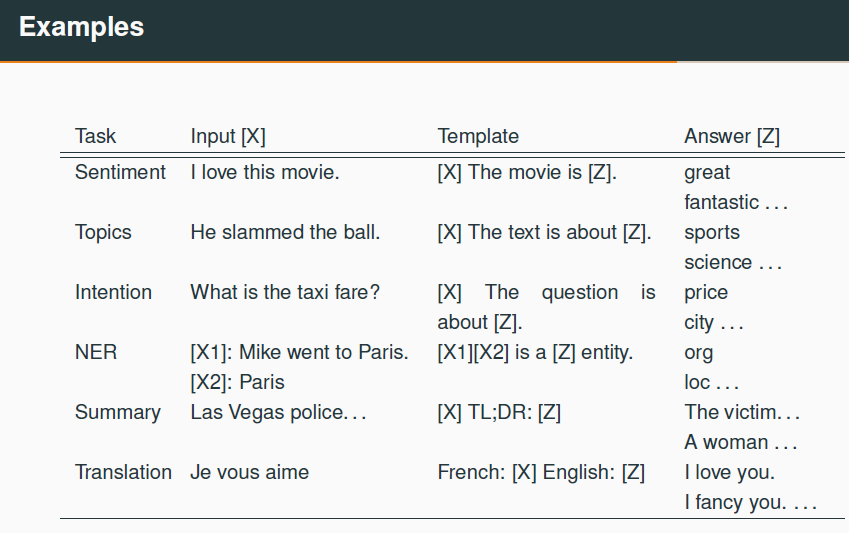


**Prompting:**

Pre-train->finetuning: adapting LMs to downstream tasks

Pre-train->prompt: adapting downstream tasks to LMs

* Add a prompt f(.) = [x][z], apply it to input x
* Search answers: a set of z
* Map highest scoring answer to desired output



**Prompt engineering**:

* Creating prompt function
* Automated template learning of discrete prompts
* Continuous prompts: initialize with discrete prompt, finetuing

1. **Low resource MT**

**Monolingual Data**: need synthetic parallel data, back translation, can fail if the initial system is too weak.

**Multilingual data**: using de-en to initialize sw-en

Q: why large pre-training model had little influence on MT?

A: MT is highly-resourced task for the most studied language pairs; MT is usually encoder-decoder model, while pre-train is an encoder model.

**mBART**:

* Objective: loss over full text reconstruction
* Noise
  + Mask 35% of text
  + Permute the order of the sentence
* Input: source and target sentence

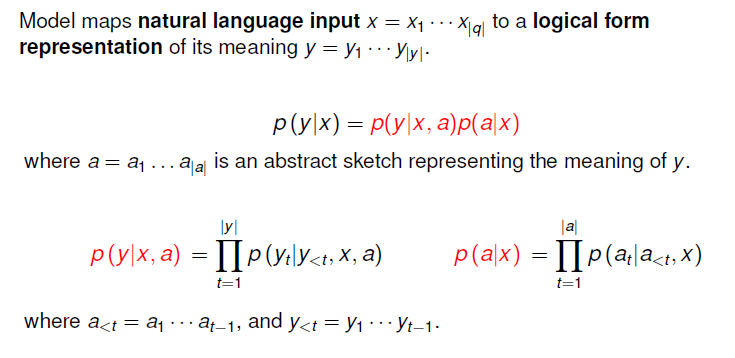
Evaluation: metrics are less reliable on poor system; human evaluation is always preferable

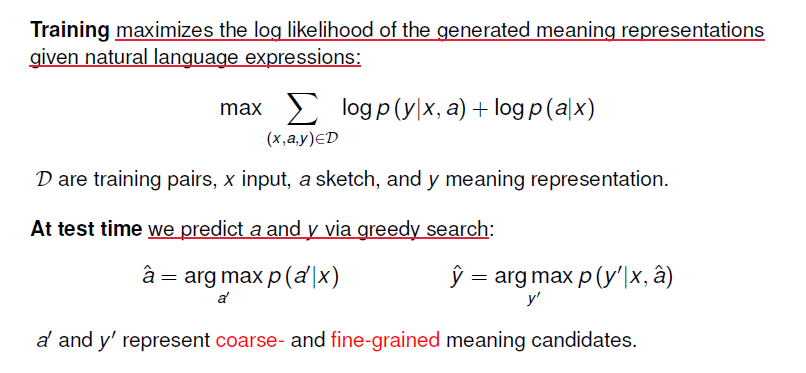
1. **Semantic parsing**

Challenges:

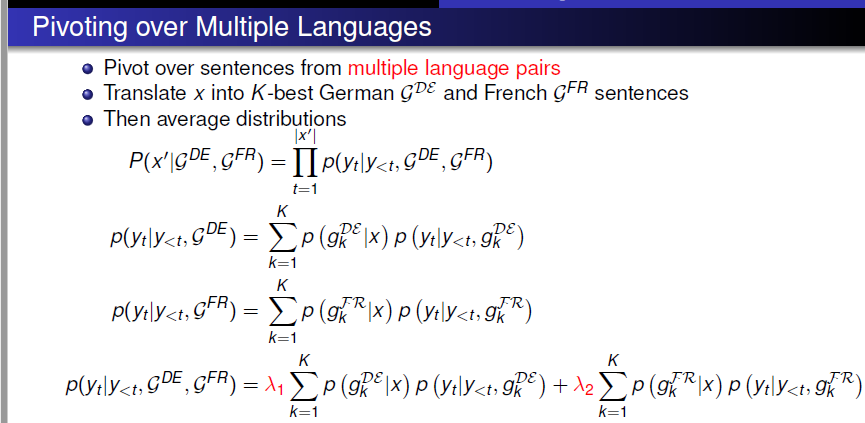
* **Structural mismatching**: neural encoder-decoder architecture for mapping NL into logical forms
* **Well-formedness Constrains**: coarse-to-fine decoding
* **Linguistic Coverage**: query parsing framework

**Coarse-to-fine model**: input-encoding -> sketch decoding ->sketch encoding -> output decoding, the objective is to maximize sum log likelihood



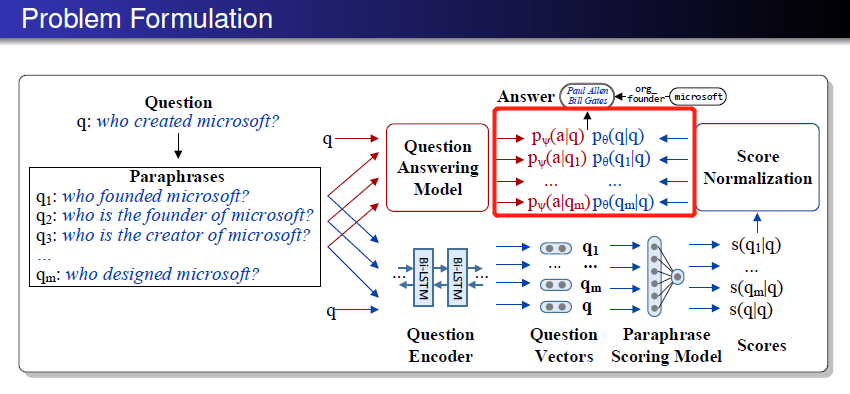


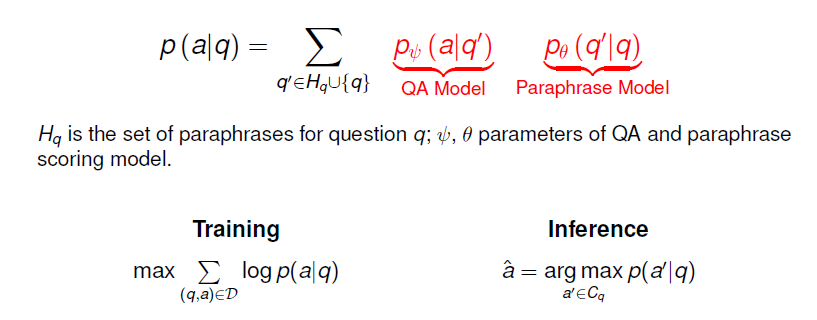
ParaNet: multiple Pivot (translate)



**Paraphrase’s evaluation**: sentence length, 1-4 gram similarity, paraphrase probability, language model score, cosine distance, attention scores

**Paraphrase for QA**:





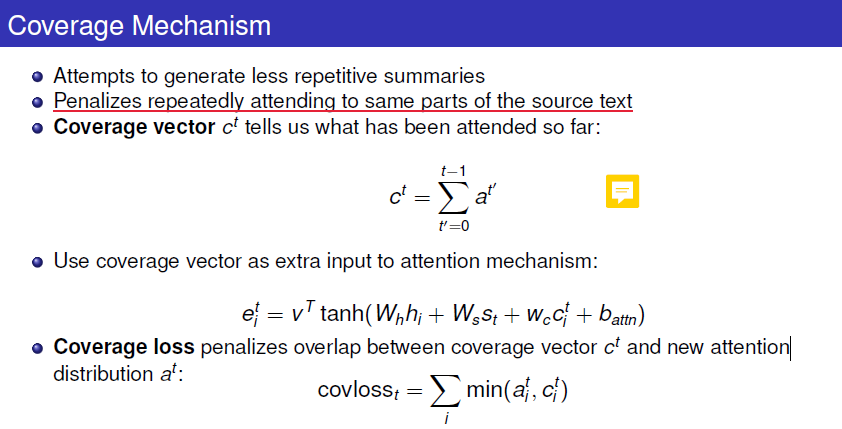
1. **Summarization**

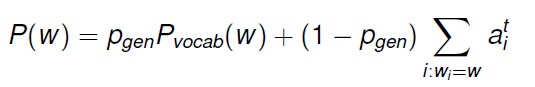
Extractive Summarization: select parts in the document.

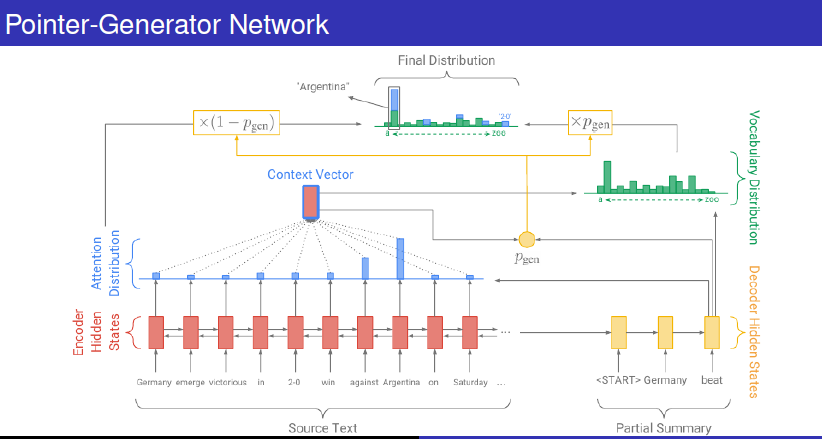
Abstractive Summarization: generate new text.

**Pointer-Generator Network:**

* Implement **copying mechanism**
* **Solving problem:** The copy mechanism is a way to handle one of the issues that is out-of-vocabulary words.
* Coverage Mechanism: Penalizes repeatedly attending to same parts of the source text

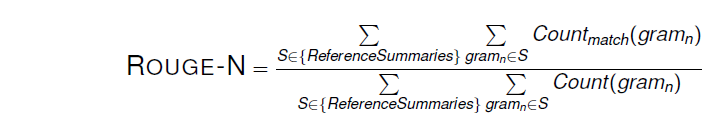


* Copying by point or generating words from a fixed vocabulary 



Smaller learning rate and longer warm-up for encoder; larger learning rate and shorter warm up for decoder

Summarization Evaluation: ROUGE



Potential problem:

1. Factual error
2. Nonsense sentence
3. Repeat itself
4. No way to module the sentence length

Since it is computationally expensive to find a globally optimal subset of sentences that maximizes the Rouge score, we employ a greedy approach, where we add one sentence at a time incrementally to the summary, such that the ROUGE score of the current set of selected sentences is maximized with respect to the entire gold summary.

1. **Data-to-Text Generation**

Input: a table of unordered record, each represented as features, type of record, entity, value; features are embedded into vectors; multilayer perceptron yields vector representation of each record. (MLP is used for dimensionality reduction)

**Greedy decoding:** on each step, take the most probable word, and use the word to feed the next input at next step; keep going until produce <END>

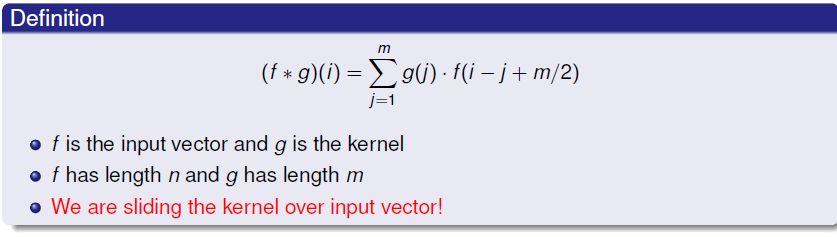
**Beam Search decoding:** on each step of decoder, keep tracing k (beam size) most probable sentence (hypothesis); after reaching some stop criteria, choose the highest probable sentence.

**Sample-based decoding**:

* Pure sampling: on each step. Randomly sample next word from probability distribution P.
* Top-n sampling: select top n words with highest probability, and then randomly sample from them

1. **CNN**

**Convolution:**



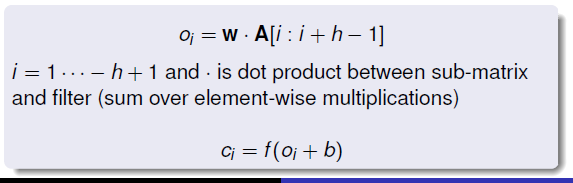
Hyperparameters of CNN:

**Zero-padding:** using when there’s not enough elements in the filer

**Stride size:** decide how much the filter shift at each step

**Objective:** minimize the cross entropy loss

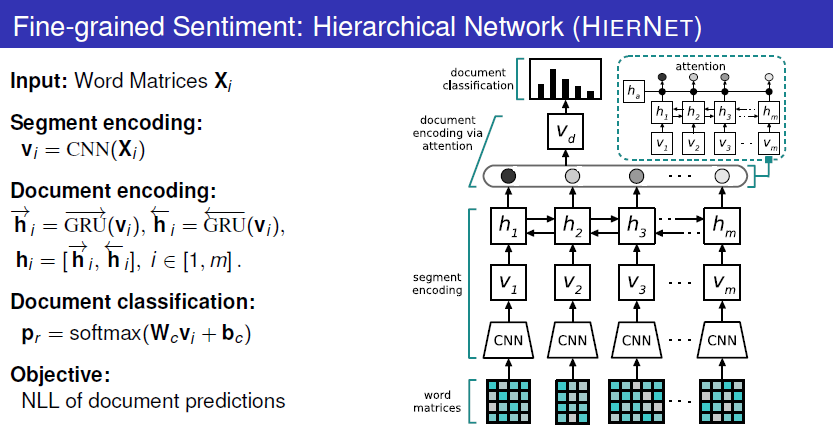
**Parameters:** weight matrix in the filter, weight matrix in the softmax, bias in the activation function.



1. **Semantic analysis**

Fine-grained Sentiment: build a document level classifier, hierarchical model (first build representations for sentences, then aggerate those into document level.); use document level classifier for label segment.

**HierNet**: Get segment encoding from CNN layer, then the document encoding is generated from bidirectional GRU layers, after get hidden states at each time step, we calculate the attention weight on each word embedding, depending on the attention weight, we finally generate the document encoding. Through softmax get the classification result. The model’s training objective is maximize the NLL of document prediction.



**Multi instance learning network:**

**Model assumption**:

* Segments have sentiment polarity [-1, +1]
* Segments have various importance a \in [0,1]

