Stanford NLP: https://www.youtube.com/playlist?list=PLuqhl4iqeAZYU3nRJQ9sgLAqSDTAsuVOX
http://web.stanford.edu/class/cs224n/
Natural language understanding Stanford:
https://www.youtube.com/playlist?list=PLuqhl4iqeAZYID9q_KZh4qZ2ycKXIh0BS
http://web.stanford.edu/class/cs224u/

Stanford NLP:

https://www.youtube.com/playlist?list=PLuqhl4iqeAZYU3nRJQ9sgLAqSDTAsuVOX

Lec1

Goals (3:04)
- Difficulties in understanding and producing human languages
- Ability to build systems for major problems in NLP using deep learning frameworks such as pytorch

GPT:
- (11:45) a model that could understand, not just doing mechanical tasks

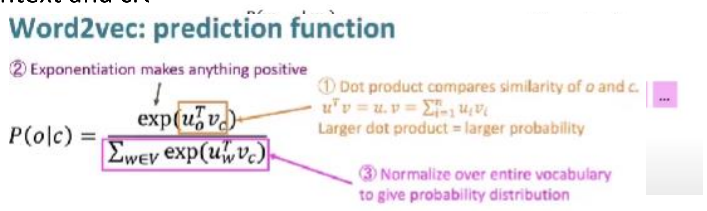
Wordnet:
- synonyms and hypernyms, like MS Word's synonym suggestions?
  But words doesn't mean the same in different contexts

Represent words as discrete symbols:
- one hot vector(18:59)

distributional semantics: (20:40)
- a words' meaning is given by the words that frequently appear close-by
- use near by words, altogether determines current word's context
- (2332): differences between tokens and types: tokens refer to one word's apparence in one sentence, while type refers to one word appearing in multiple sentences, and each one of them might have a different meaning.
- (2618): 2D simplified word embedding space, where words with more similar meanings has a smaller absolute distance between word embedding vector

Example windows and process for computing $P(w_{t+j} \mid w_t)$

Word2Vec algorithm(2742), bag of words(Lec2, 0251)
- Given a sampled body of text; go through context and cre
  likelihood of occurrence of surrounding context words GIVEN a center word, and optimize the probability along the way
- Objective function (3100)
- (3420): the probability is calculated using softmax;

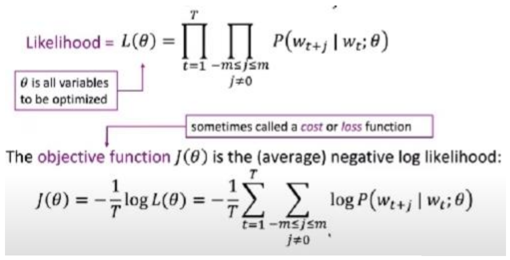

Word2vec: prediction function
② Exponentiation makes anything positive
① Dot product compares similarity of o and c.
Larger dot product = larger probability
③ Normalize over entire vocabulary to give probability distribution

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

$$u^T v = u.v = \sum_{i=1}^{n} u_i v_i$$

- Prediction function(3645)
- Elaboration: u and v comes from the same parameterization(3837), where each row is a feature vector;

(4549): derivation of softmax partial derivatives, using logs;



Likelihood = $L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m\le j\le m \\ j\ne 0}} P(w_{t+j} \mid w_t; \theta)$

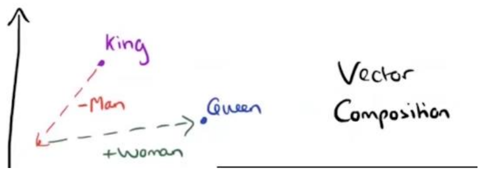
θ is all variables to be optimized

sometimes called a *cost* or *loss* function

The objective function $J(\theta)$ is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T}\log L(\theta) = -\frac{1}{T}\sum_{t=1}^{T} \sum_{\substack{-m\le j\le m \\ j\ne 0}} \log P(w_{t+j} \mid w_t; \theta)$$

Word vector code demonstrations (5704)
Numpy, sklearn, matplotlib
- (10022): vector composition example
- Demo of analogy(10400)



Lec2: Neural classifiers

Word2vec review(0414):
Group words having similar meaning together, the idea of embedding

Sparse gradient update(1215) for only context words around one center word -> remaining words' are not updated, perhaps a waste of resource as those gradients are still being calculated…
Solution(1215): using rows to represent each word, and update certain rows for only each of those update operations; or using a hash of words to keep track
Reason: for a 2D matrix, rows are accessed first -> a[x]; but if column: a[:][x] -> somehow inefficient, refer to numpy manual on row and column operation speed

Continuous bag of words method: (1607)
Predict center word from (bag of)context words

$$J_{neg-sample}(\boldsymbol{u}_o, \boldsymbol{v}_c, U) = -\log\sigma(\boldsymbol{u}_o^T\boldsymbol{v}_c) - \sum_{k\in\{K\ sampled\ indices\}} \log\sigma(-\boldsymbol{u}_k^T\boldsymbol{v}_c)$$

Negative sampling: (1822)

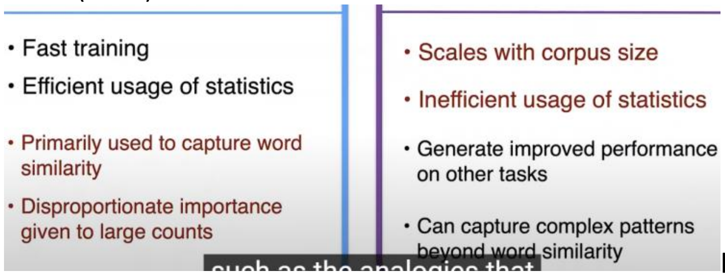
$$J_t(\theta) = \log\sigma\left(u_o^T v_c\right) + \sum_{i=1}^{k} \mathbb{E}_{j\sim P(w)}\left[\log\sigma\left(-u_j^T v_c\right)\right]$$

Sigmoid function: for closing context words, maximize the sigmoid; for all remaining words(in the summation), make the sigmoid function have probability as small as possible
Realize only sampling "k" words from total of so many words can still make a significant progress, similar reason to batch gradient descent

Co-occurrence matrix(2343):
How-often words occur together in a window of 2/3/4… -> numbers indicates matrix dimensions. Can also be done using paragraphs/whole web page etc.
Smaller number of dimensionsalso makes prediction result relatively good.

Singular value decomposition(3246)
After doing that, take only those parts with higher diagonal values, and use those diagonals to construct a lower dimensional approximation of words
Applying SVD(3601): functional/too-frequent words can affect results. -> log frequencies, take restrict the maximum of a word occurrence, or ignore those too frequent but meaningless words

GloVe(3813):



- Fast training
- Efficient usage of statistics
- Primarily used to capture word similarity
- Disproportionate importance given to large counts

- Scales with corpus size
- Inefficient usage of statistics
- Generate improved performance on other tasks
- Can capture complex patterns beyond word similarity

such as the analogies that

linear models vs neural models

Co-occurrence can encode meaning components(a kind of description with the object)(4128)

(4246) log bilinear models with vector differences;

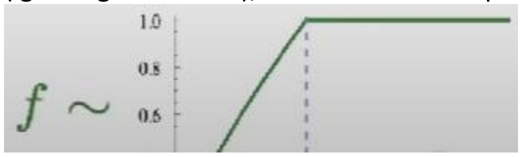
$$w_i \cdot w_j = \log P(i|j)$$

$$w_x \cdot (w_a - w_b) = \log \frac{P(x|a)}{P(x|b)}$$

all are dot products

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

Objective: make words dot product be as close to co-occurrence matrix value as possible (ignoring bias terms); “f”: MODIFIED frequency of words, having below shape: (flat later)



Intrinsic vs extrinsic evaluation of word vectors(4700)

Intrinsic: specific/intermediate subtask; short time to compute; not sure if helps in real world

Extrinsic: evaluation on real task; take long time; but works better in real

Intrinsic: (4852)

Cosine distance between two words; cannot handle non-linear cases

$\boxed{\text{a:b :: c:?}}$ $\longrightarrow$ $\boxed{d = \arg\max_i \frac{(x_b - x_a + x_c)^T x_i}{||x_b - x_a + x_c||}}$

man:woman :: king:?

Analogy evaluation and hyperparameters evaluations(5200)

- Wikipedia tokens are better than common text tokens;
- Larger vector size leads to better accuracy, but suffers from inveted scale problem

10800: one word can appear in multiple locations to ensure being grouped and close to all relevant information, while still maintaining different context words belongs to different clusters.

11200: integrated meaning of words in each context using linear combination:

Word senses -> one word can indeed have different contexts, and can be integrated with different classes.

$$v_{\text{pike}} = \alpha_1 v_{\text{pike}_1} + \alpha_2 v_{\text{pike}_2} + \alpha_3 v_{\text{pike}_3}$$

Where $\alpha_1 = \frac{f_1}{f_1+f_2+f_3}$, etc., for frequency f

Missing extrinsic evaluation methods

Lec3: backprop and neural netowrks, pytorch implementations require reviewing

Named entity recognition: 0541

- Find and classify names in text;
- Also useful for tracking particular entities from documents, or question answering memorization;
- Simple method: (0633): using context windows to acquire a set of vectors, and feed to NN with logistic classifier, which gives a likelihood that the word is a named entity

Stochastic gradient descent and backpropagation((1300)

- Calculate gradient by hand(matrix calculus)
- Definition of gradient(1600): how much will the output change if we change the input a bit
  The gradient calculated represents the “multiplication factor” w.r.t output’s slight change amount, to represent the parameter’s change amount -> \delta_p=\delta_z*gradient
  Partial derivatives, gradient of vectors

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

- Jacobian(1813): multiple functions, each function need to have a vector of partial derivatives on each input it has.

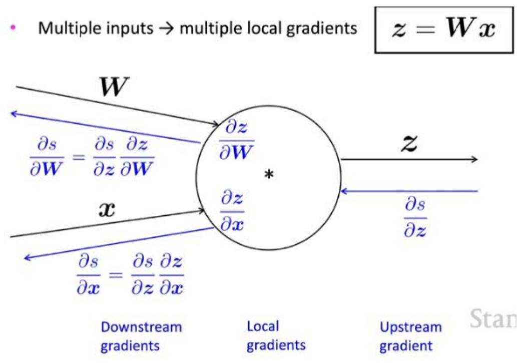
$$\frac{\partial}{\partial \boldsymbol{x}}(\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b}) = \boldsymbol{W}$$
$$\frac{\partial}{\partial \boldsymbol{b}}(\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b}) = \boldsymbol{I} \ \text{(Identity matrix)}$$
$$\frac{\partial}{\partial \boldsymbol{u}}(\boldsymbol{u}^T\boldsymbol{h}) = \boldsymbol{h^T}$$

$$\frac{\partial s}{\partial \boldsymbol{W}} = \boldsymbol{\delta}^T \boldsymbol{x}^T$$

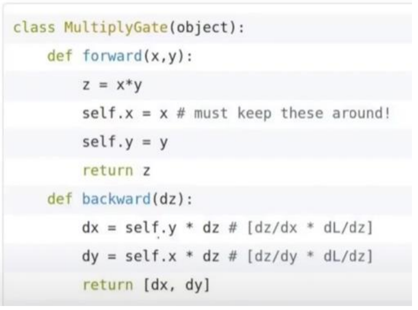
So $\frac{\partial s}{\partial \boldsymbol{W}}$ is n by m: $\begin{bmatrix} \frac{\partial s}{\partial W_{11}} & \cdots & \frac{\partial s}{\partial W_{1m}} \\ \vdots & \ddots & \vdots \\ \frac{\partial s}{\partial W_{n1}} & \cdots & \frac{\partial s}{\partial W_{nm}} \end{bmatrix}$

- Chain rule(2015),
  Jacobians multiplication(2313): treat elementwise operations as a diagonal matrix multiplication/operation
  Gradient of linear operations: refer to video location (1800), and treat each row of resulting vector “h” as a function. Final jacobian should give desired results
  Backpropagation for eliminating duplicated computations to save time(3431)(3453)
- Realize: dh/dw is the outer product of vector h and x!!! which has the same shape as input matrix(3544) shape convention is used for making weight update convenient(3716) sometimes pursuing pure correctness won’t be an optimal solution
  Further explanation(3923) idea: weights are only being used for calculating a particular h’s element, and using one particular x’s elements. Thus other elements won’t affect this weight element’s change(4042). Outer product matrix(4142)

- Dimension transpose/permute (not reshape!!!)(4446)

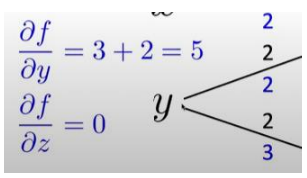

Multiple inputs → multiple local gradients
$z = Wx$
W
x
z
$\frac{\partial s}{\partial W} = \frac{\partial s}{\partial z}\frac{\partial z}{\partial W}$
$\frac{\partial z}{\partial W}$
$*$
$\frac{\partial z}{\partial x}$
$\frac{\partial s}{\partial z}$
$\frac{\partial s}{\partial x} = \frac{\partial s}{\partial z}\frac{\partial z}{\partial x}$
Downstream gradients
Local gradients
Upstream gradient
Stan

Backpropagation algorithm in software
- Re-use derivatives from higher layers(4800). Computation graph(4842)
- Forward propagation(5000): calculate outputs given inputs
- Backward propagation: send back gradients for parameters to update(4950)
- Multiple input cases(5349) (image shown right)
- Variable being used for multiple calculations (10112)(10308)



```
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        self.x = x # must keep these around!
        self.y = y
        return z
    def backward(dz):
        dx = self.y * dz # [dz/dx * dL/dz]
        dy = self.x * dz # [dz/dy * dL/dz]
        return [dx, dy]
```

Summary(10748)
Abstract implementations(11500)(11650)
Manual gradient checking (11800)
Useful for checking whether customly defined methods (backward()) is implemented correctly


$\frac{\partial f}{\partial y} = 3 + 2 = 5$
$\frac{\partial f}{\partial z} = 0$
y
2
2
2
2
3

Lec4: Dependency Parsing

Consistency and dependency, transition-based dependency, neural dependency parsing

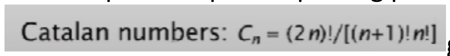
Phrase structure: (0512) organize words into nested constituents
- Words -> phrases -> bigger phrases
- Context free grammars: nested phrases(0716)
- Pay attention to how professor defined grammar rules(0900-1156): lexicon + grammar (consisting of English word types: noun, verb… and used regex notations)

Dependency structure(1423)
- Shows which words depend on which other words(modify, attach to, being arguments of) (1946) is a good example
- Humans convey ideas by composing multiple ideas into complex structures. Thus language models might require disecting each sentence structure and figure out dependency relations between words to understand.

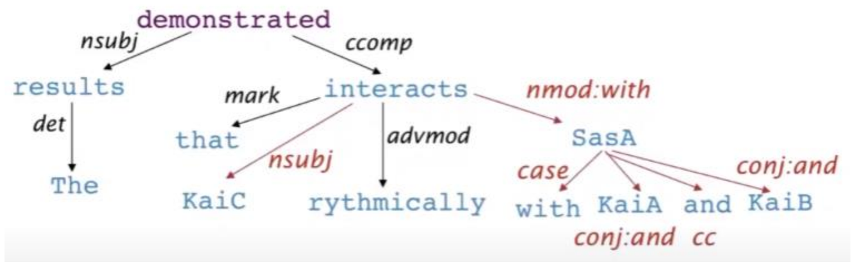
Ambiguity analysis(recall CSC485's materials): A kill man with knife: "man with knife", or "kill with knife?"(2500)

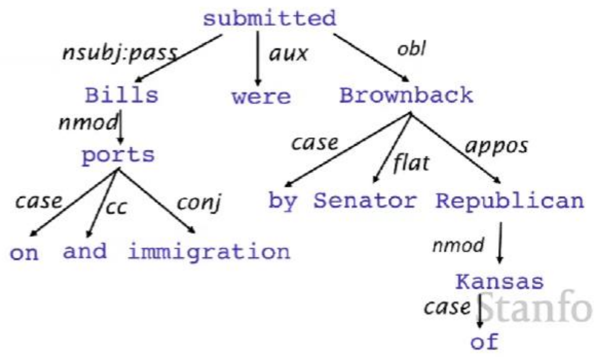
Comparisons with Chinese ambiguities(2720)
Idea of phrase attachments matching(2928)

- # of ALL possible phrase parsing possibilities[including all wrong ones]: (3151)


Catalan numbers: $C_n = (2n)!/[(n+1)!n!]$ given a total of "n" many phrases-; exponentially grow

Coordination scope ambiguity(3446)
- [veteran and executive] [jack]/[veteran] and [executive jack]





Adjectival/adverbial modifier ambiguity(3803)

- Students get [first hand] [job experience] / students get [first] [hand job] experience

Verb phrase attachment ambiguity(4002)

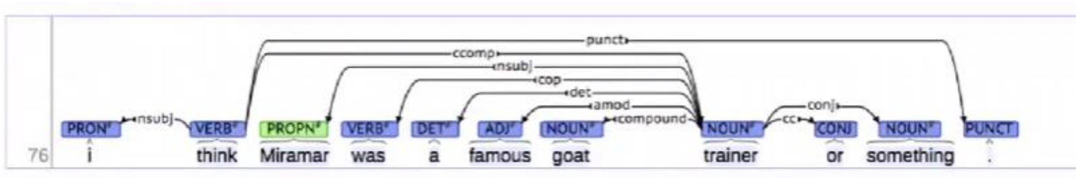
- Body washes up on [beach] [to be used for a task] / [body] washes up on beach [to be used for a task]

Semantic interpretation(4200)

Dependency grammar and dependency structure(4300)

- Binary asymmetric relations(arrows) called dependencies
  Arrows are “typed”(labelled) with grammatical relations
  Dependencies form a tree;

Annotated dependency graphs(5100) comes from human-worked datasets



Advantages of annotated treebank structure over writing grammar rules(5413)

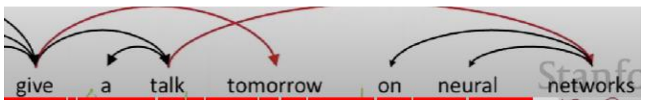
- Labor re-using
- Broad coverage, not just few intuitions
- Frequencies and distributional information acquired
- Better for evaluate NLP system

Sources of information for dependency parsing(5700)

- Bilexical affinities: plausibility of logics and pairing(adj for noun, adv for verb)
- Dependency distance: nearby words have more dependencies
- Intervening material
- Valency of heads: some words can only occur on one side of a word;
  “the” can only modify words appear later

Dependency parsing(10000)

- Constraints: ROOT only connects to one word
- No cycles, only a tree;
- Arrow crossing (non-projective or not)
  Red arrows are crossed; require shifting of phrases?

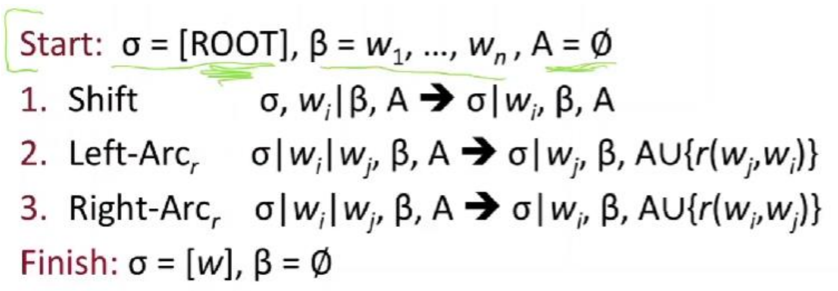


Methods for dependency parsing(10400)

- Dynamic programming(reduce from exponential time to polynomial)
- Graph algorithms

- CSP: constraint satisfaction (csc384): applied on edges (“valency of head”-like constraints)
- Transition-based parsing/deterministic dependency parsing: greedy method

Greedy transition-based parsing (10630)
- Bottom up actions for a PARSER
- Parser properties:
  Stack \sigma
  Buffer \beta
  Dependency arcs **A**
  Set of actions (3, shown as shift, left arc, right arc in example)
- Example parsing(11133)
- Time complexity & accuracy: linear but low(11630)
- Choosing next action(11230)
  Machine learning methods? SVM? Beam search?(require further studying this method)



Start: $\sigma = [\text{ROOT}],\ \beta = w_1, \ldots, w_n,\ A = \emptyset$

1. Shift $\quad \sigma, w_i|\beta, A \rightarrow \sigma|w_i, \beta, A$
2. Left-Arc$_r$ $\quad \sigma|w_i|w_j, \beta, A \rightarrow \sigma|w_j, \beta, A\cup\{r(w_j,w_i)\}$
3. Right-Arc$_r$ $\quad \sigma|w_i|w_j, \beta, A \rightarrow \sigma|w_i, \beta, A\cup\{r(w_i,w_j)\}$

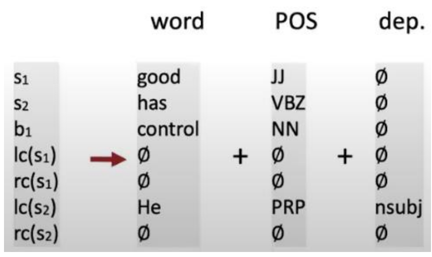
Finish: $\sigma = [w],\ \beta = \emptyset$

Parsing evaluation(11900)
- Direct accuracy calculation: unlabelled/labelled accuracy score

Lec5: Dependency Parsing continued and language modelling using RNN

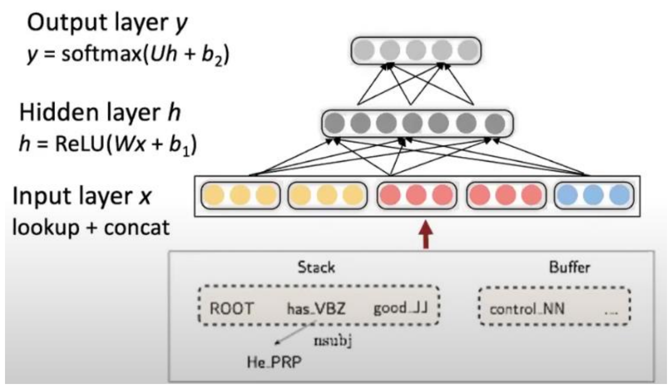
Indicator features revisited(0340)
- Problems of dependency parsing using indicator features: sparse, incomplete, and expensive to compute(0453)

Neural dependency parser(0600)
- Mechanism: (0810):
  distributed representations: d-dimensional dense vector (word-embedding)
  part of speech(POS) + dependency labels as d-dimensional vectors(0930)
  word type, such as noun… such as modifying, subj, etc.
- Neural network allows using softmax for classification(1200)
  Traditional classifiers only provide linear decision boundaries;



	word		POS		dep.
s_1	good		JJ		∅
s_2	has		VBZ		∅
b_1	control		NN		∅
$lc(s_1)$	→ ∅	+	∅	+	∅
$rc(s_1)$	∅		∅		∅
$lc(s_2)$	He		PRP		nsubj
$rc(s_2)$	∅		∅		∅

Neural depenency parser model architecture(1755)



Input layer consist of feature vectors; after MLP layers, a softmax will output probability

Graph based dependency parsing(2032)

- Showing likelihood of word dependency relationships   minimum spanning tree

Neural network extended

- Regularization(2500)&overfitting
  Regularization only ensures model generalization; but overfitting is not completely resolved;
- Dropout(3030)
  Prevents feature co-adaptation; can be thought of emsembling
  Backward pass: no gradient going through(3500)
- Vectorization(3300): use vectors, matrices, tensors instead of for-loops;
- Nonlinearities; (3640)
  Logistic, tang, relu, softmax…
  Relu is usually the first option to try due to ease of training
- Parameter initialization: (4410)
  Random, small values; uniform distribution for weights initialization
- Optimizers: (4600)
  SGD requires choosing appropriate learning rate;
  Adam: usually the best?
- Learning rate elaboration(4855)
  Learning rate scheduling for decreasing and leading to better convergence

Language modelling(5017)

- Task of predicting the word coming next; giving a conditional probability
  $P(\boldsymbol{x}^{(t+1)}|\ \boldsymbol{x}^{(t)},\dots,\boldsymbol{x}^{(1)})$ predict "t+1" given previous observations

n-gram language model(5314)

$$P(\boldsymbol{x}^{(t+1)}|\boldsymbol{x}^{(t)},\dots,\boldsymbol{x}^{(1)}) = P(\boldsymbol{x}^{(t+1)}|\overbrace{\boldsymbol{x}^{(t)},\dots,\boldsymbol{x}^{(t-n+2)}}^{n\text{-1 words}})$$

prob of a n-gram

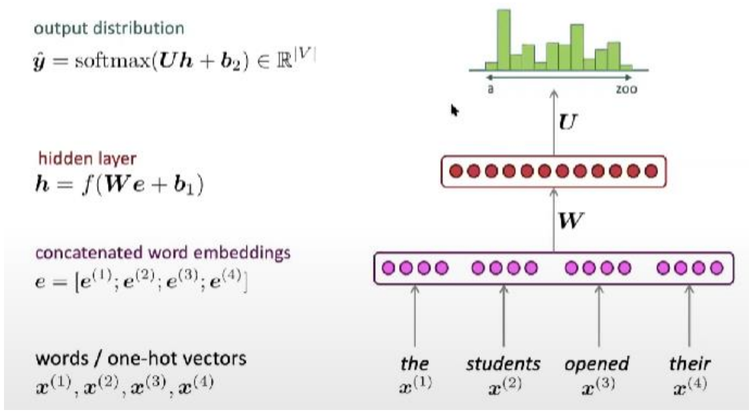
prob of a (n-1)-gram

$$= \frac{\boxed{P(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\dots,\boldsymbol{x}^{(t-n+2)})}}{\boxed{P(\boldsymbol{x}^{(t)},\dots,\boldsymbol{x}^{(t-n+2)})}}$$

- Chunk of n-consecutive words
- Markov assumption: current word prediction only depends on PREVIOUS words
- Statistical approximation using counts(5530)
- Problems: (5900)
  Sparsity of data, some combinations may never occur in training
    Adding numeric stable terms: epsilon
    Backoff is another alternative, which further shortens the context
  Storage problems(10100)
- Example of predicting, and generating using sampling (10200-)

Building a neural network:

- Setting(10600)
  Input: sequence of words in feature vectors; output: probability of next word given those observations
  Simple example (10700) (also see image below)
- Require neural architecture that can process arbitrary amount of context, sharing parameters and still preserve sensitivity;

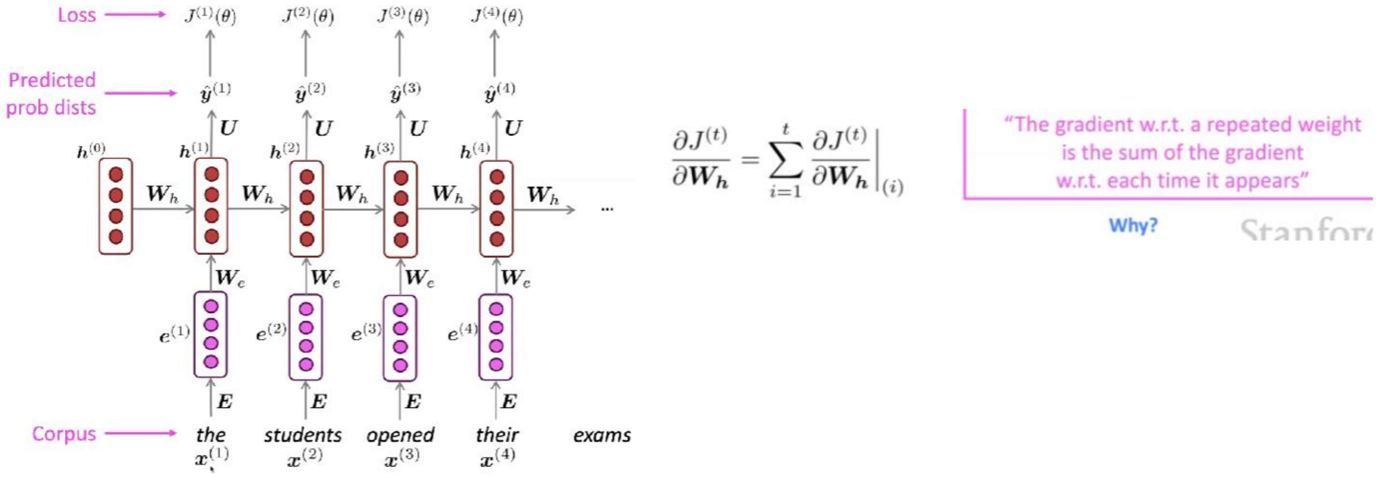

output distribution
$\hat{y} = \text{softmax}(Uh + b_2) \in \mathbb{R}^{|V|}$
hidden layer
$h = f(We + b_1)$
concatenated word embeddings
$e = [e^{(1)}; e^{(2)}; e^{(3)}; e^{(4)}]$
words / one-hot vectors
$x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}$
a
zoo
U
W
the
students
opened
their

RNN(11200)

- Self-feeding hidden layers
- Input words will be treated as a sequence, each part of sequence contains only one word, and used for predicting future results given current word and previous results(hidden states)
- Advantages & disadvantages(11730)
  Can process any length input with fixed model size, can use information from many steps back;
  But slow computation, gradient vanishing problem when accessing past context

Lec 6: RNN, exploding&vanishing gradient; LSTM, bidirectional RNN

Train RNN: (0450)



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

- Sequence, compute output distribution of word come next, use cross entropy loss
- Batch update using SGD to gain speed and efficiency (1030)
- Backpropagation of RNN(1140)
  Weight gradient is the sum of loss over weight FOR EACH TIME STEP(1330)
  Multivariable chain rule(1530)

Generating text using RNN (1900)

- BOS & starting word, predict probability&sample next output, take next output as part of input, continue generating. EOS

Evaluating language models(2430)

- Perplexity:
  Inverse(1/x) likelihood(product of conditional probabilities, predicting next word using current sequence), which is equivalent to exponential of cross entropy

$$\text{perplexity} = \prod_{t=1}^{T} \underbrace{\left(\frac{1}{P_{\text{LM}}(\boldsymbol{x}^{(t+1)} \mid \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})} \right)}_{}^{1/T} = \prod_{t=1}^{T} \left(\frac{1}{\hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}}} \right)^{1/T} = \exp\left(\frac{1}{T} \sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}} \right) = \exp(J(\theta))$$

RNN in NLP:

- Sequence tagging, part of speech: (3200)
  Given a sequence of words, predict the role of each word in a sentence -> verb, noun, …
- Sentiment classification(3340)
  Sentence encoding
- Question answering/text generation(3440)
  Use RNN to acquire sentence encoding, and combine with context(perhaps also in RNN) to generate answer
  Speech recognization, machine translation, summarization

Vanishing gradient in RNN(3730)

- Long sequence, long chain rule, small partials -> small gradient for upstream nodes;
- Problems with vanishing gradient in RNN(4330)
  Long-term effects are hard to acquire, only close-by signals
  Cannot learn long-term dependency
- A problem not only occur among RNNs, but also “quite deep” NN(11000)
  Solution: skip-connection, direct connections, highway connections(using gates to control information sent on skip-connection paths)
-

Exploding gradient: lead to “large scaled” parameter update

- Gradient clipping (4730): use a threshold on gradients.

LSTM(4930)

- Hidden state (h) and cell state (c) same vector length
- “cell” store long term information. Can be read, erase and write
- Gate vectors: probability for read, erase, write
  Forget gate(f), input gate(i), output gate(o)
- Update: (involves element-wise product)
  New cell content(c\bar), cell state update(forget+input), hidden state(output + cell)
- Demo (5430)
- Complete mechanism graph(5921)
- Preserves information over many time steps(10300)

Bidirectional RNN, multi-layer RNN(11600)

- Some times one word’s interpretation might also depend on future information (require another direction for RNN) -> forward RNN + backward RNN; concatenate representation along the way, for a particular time step’s states.
- Only applicable if have access to entire sequence(12000)

Transformer also includes bidirectional property

Lec7: machine translation, sequence2sequence, attention

Statistical machine translation
Learning alignment(1140)

- Capturing the differences in language grammar (position of verb etc…)
- Model reason jointly on alignment and source context to generate target
- Examples of complex alignments(1330)
  Translating multiple words into one word, or several words into another several words, paraphrasing

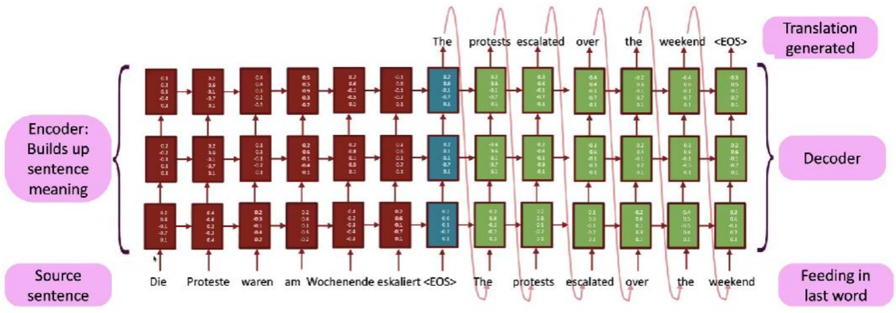
Decoding mechanism&examples(1700)

Neural machine translation(2000)

- Single end2end neural network
- Sequence2sequence -> 2 RNNs(2400)
  One RNN for encoding(encoded features), another RNN for decoding(translated sentence)
- Sequence 2 sequence application: summarization + dialogue + parsing + code generation

Probabilistic model(2900)

- Using conditional probability calculation on a markov chain
- Training procedure(3000)
  Trained on node by node procedure, one output after another
  Gradient descent graph(3200)
- Multi-layer RNN/stack RNN: (3400)
  Lower RNN compute lower-level features, higher RNNs computer higher-level features
  See one example(along with its application on NMT) right;
  Features: (3700)
    Yield high performance, with skip connections, and consist of 2-4 layers usually


Translation generated
The protests escalated over the weekend <EOS>
Encoder: Builds up sentence meaning
Decoder
Source sentence
Die Proteste waren am Wochenende eskaliert <EOS> The protests escalated over the weekend
Feeding in last word

Decoding methods

- Greedy decoding(3900): taking probable word on each step without referring to future decoded results->cannot undo decisions
  Solution(4200): exhaustive search decoding:
    Generate all possible sequences, and find the one maximizing the likelihood function below:

$$P(y|x) = P(y_1|x)\,P(y_2|y_1,x)\,P(y_3|y_1,y_2,x)\ldots,P(y_T|y_1,\ldots,y_{T-1},x)$$
$$= \prod_{t=1}^{T} P(y_t|y_1,\ldots,y_{t-1},x)$$

- Beam search decoding(4330)
  Keep track of “k” most probable partial translations(hypothesis) along the way of decoding

(reduced possibility compared to exhaustive search)

Not guaranteed to be always optimal, but works well in practise when also considering other factors

Hypothesis(4400)

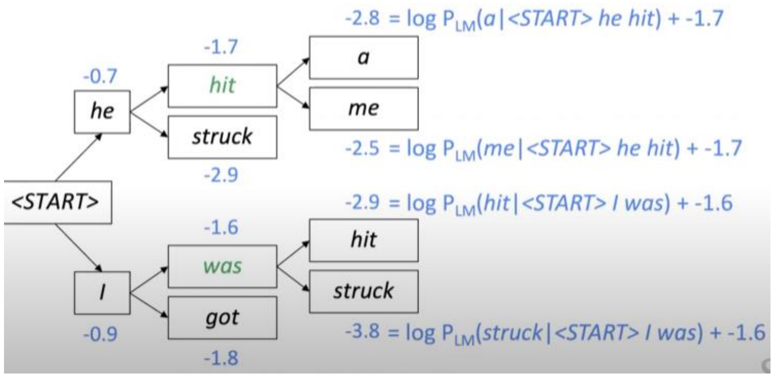
$$\text{score}(y_1, \ldots, y_t) = \log P_{\text{LM}}(y_1, \ldots, y_t|x) = \sum^{t} \log P_{\text{LM}}(y_i|y_1, \ldots, y_{i-1}, x)$$

Heuristics using likelihood

Example(4500)

quite useful for understanding with specific values for likelihood representation;

pay attention to elimination ordering


-2.8 = log P_LM(a|<START> he hit) + -1.7
-1.7
a
-0.7
hit
he
me
struck
-2.5 = log P_LM(me|<START> he hit) + -1.7
-2.9
<START>
-2.9 = log P_LM(hit|<START> I was) + -1.6
-1.6
hit
was
I
struck
got
-0.9
-3.8 = log P_LM(struck|<START> I was) + -1.6
-1.8

Exploring other hypothesis for potentially better results(4900)

Longer hypothesis have lower scores;

Normalize by sentence length

Advantages&disadvantages of neural machine translation(5200)(5400)

- Better performance; can be optimized end2end; less human effort to build
- Not interpretable; hard to control; ethics/AI safety

Evaluation metrics(5500)

- BLEU: bilingual evaluation understudy; n-gram precision/similarity score
  Inperfect; not fflexible to various valid ways

Current problems of NMT: (10200)

- Out of vocabulary words; domain mismatch; problems maintaining context; low data is not supported; failures of capturing meaning; pronoun resolution issues; morphological agreement;
  Bias in training data occurs when using

Problem of RNN/sequence2sequence(11300)

- Bottleneck: nodes at back of RNN requires capturing all information from previous sentence -> can be biased or lower performance of later nodes of RNN;
- Require attention: direct link between encoder and decoder, focus on a particular part of input for ALL the time, and perhaps make changes along the way.
- Compare decoder state with encoder state and generate attention score using dot product attention(and softmax, in particular)(11600)

Lec8: attention + final project

Lec9: self attention & transformers

RNN doesn't allow parallelizability when making operations (0900)

Word windows(1D convolution over word embedding sequences?)

- Still weak in dealing with long-term contexts(1300)

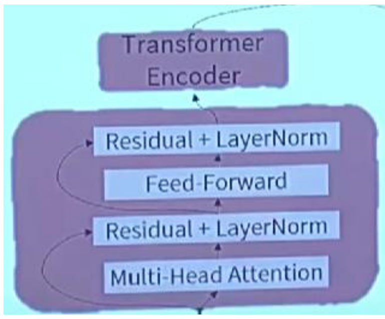
Attention(1600)
Attention weights are calculated along "key" dimension (1900)

Ordering information, positional embedding?(2500)
- Representing each sequence index as a vector
- Sinusoidal position representations(2700)
- Learn about positional embeddings(2800) -> but cannot generalize to indices outside defined range.

Non-linearities: add another feed-forward neural network with non-linear activations.

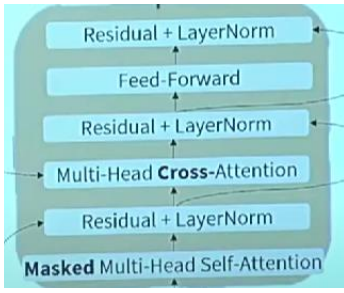
Masking future information(3400)

Issues and solutions with attention(3600)



Transformer
- Cross attention, multi-head attention(3800)
- Self-attention(4000)
  Note: softmax should be taken via row dimension at time (4230)
- Cross attention(10200)

- Multi-head attention: parallel processing of different parts of one sentence
- Significance of residual connection(4900)
  In training(make loss function landscape more smooth) and information required to learn -> skip connections allows layers to learn only extra information required
- Layer normalization: normalize over all the summed inputs to one neuron's output. (5100)
  (10900): questions:
  Losing dynamic range as sequences get longer, layer norm might not fix that problem and require scaling.
- Scale dot product: divide matrix score by \sqrt(feature_dim/num_heads) before feeding into softmax operation.
- Encoder and decoder block (5900)



Fixing transformer model(11000)
- Dot product between pairs of inputs requires quadratic computation time, while for RNN the computation time grows linearly.
  Mapping sequence length dimension to lower dimension for values and keys(11300) -> Linformer
  using random interactions(sample from keys and values to do dot product with, instead of acrossing all key-values) -> BigBird
- Finding better positional embeddings

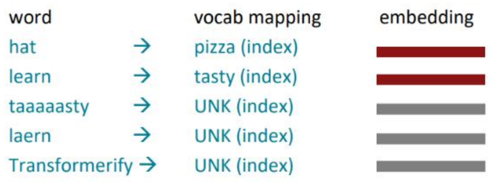
Lec10: model pre-training, transformers reviewing, subword models

TODO: go through

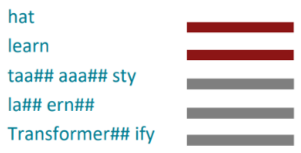
https://web.stanford.edu/class/cs224n/slides/cs224n-2023-lecture9-pretraining.pdf

to fill in the gaps!!!

Using sub-words instead of whole word:
- UNK(nown) token: back-ups when approaching some new words not learned before
- Handling misspellings of words or other inaccurate language syntax



word		vocab mapping	embedding
hat	→	pizza (index)	🟥
learn	→	tasty (index)	🟥
taaaaasty	→	UNK (index)	⬜
laern	→	UNK (index)	⬜
Transformerify	→	UNK (index)	⬜



hat	🟥
learn	🟥
taa## aaa## sty	⬜
la## ern##	⬜
Transformer## ify	⬜

- Byte-pair encoding algorithm(0730)
  Start with only characters, integrate adjacent characters as subwords
  Effectiveness: two images shown above gives a quite explicit comparison of using subwords than using whole words. (1210)

Pre-trained word-embeddings(1800)

Subwords mechanism(0000-1400)
- Parts of words(instead of whole word?) -> split one word into subwords; most extremely: character-by-character encoding
- How subwords handle wrong syntax

Pre-train & fine-tune
- Effectiveness(2200)
  Representations of language, strong parameter initialization, probability distribution over languages to sample from
- Input-reconstruction learning(2300-2600)
  Examples involves blanking/masking part of input, and let the model predict the masked words.
- Problem of overfitting(student question, 2800): actually underfitting… large models need wayyyyyyyyy more data than expected actually
- Learning general things -> adapt to custom tasks

Stochastic gradient descent and finetuning loss objectives(3800)

Fine-tuning for pre-trained transformers
Order: decoder, encoder, finally enc-dec connection
- Decoder: finetune on last word's hidden state, possibly also with a randomly initialized linear decoder layer, which backpropagates to whole decoder(4300)

- Dialogue (context=dialogue history)
- Summarization (context=document)

$$h_1, \dots, h_T = \text{Decoder}(w_1, \dots, w_T)$$
$$w_t \sim Aw_{t-1} + b$$

GPT(5000)
- Encoder: finetune by masking out part of input, and then combine with finetuned decoder to learn predicting those masked words. (5600)
  BERT
  Limitations: not useful for word by word autoregressive generation tasks

$$h_1, \dots, h_T = \text{Encoder}(w_1, \dots, w_T)$$
$$y_i \sim Aw_i + b$$

- Encoder-decoder connection(11200)
  Pretraining: see the right image. Encoder generate first set of hidden layers, and decoder generate another set of hidden layers(distinguish by time-steps/sequence positions). When fine-tuning, using another linear layer (perhaps with bias), and backpropagate to whole model

$$h_1, \dots, h_T = \text{Encoder}(w_1, \dots, w_T)$$
$$h_{T+1}, \dots, h_2 = Decoder(w_1, \dots, w_T, h_1, \dots, h_T)$$
$$y_i \sim Aw_i + b, i > T$$

Learnings from pre-training:
- Semantics

GPT3: LLM and in-context learning
- Learning without gradient steps: from examples provided within contexts

Lec11: Question answering

SpanBERT(00000)
Into to question answering(0100)
System structure(0800) & fine tune on pretrained BERT
Questions based on unstructured text(1100)
Reading comprehension(1200)
Importance of studying question answering problems:
- Important for understanding whether computers "understand" human languages(1500)
- Many NLP tasks can be reduced to NLP problems(1600)
- Semantic role labeling(1700) -> converting roles of labelling into questions

SquAD dataset(2100)
- Exact match and partial credit(F1 criterion) evaluation metrics
- Comparing predicted answers with gold answers(3 gold answers would be collected due to having multiple plausible answers)

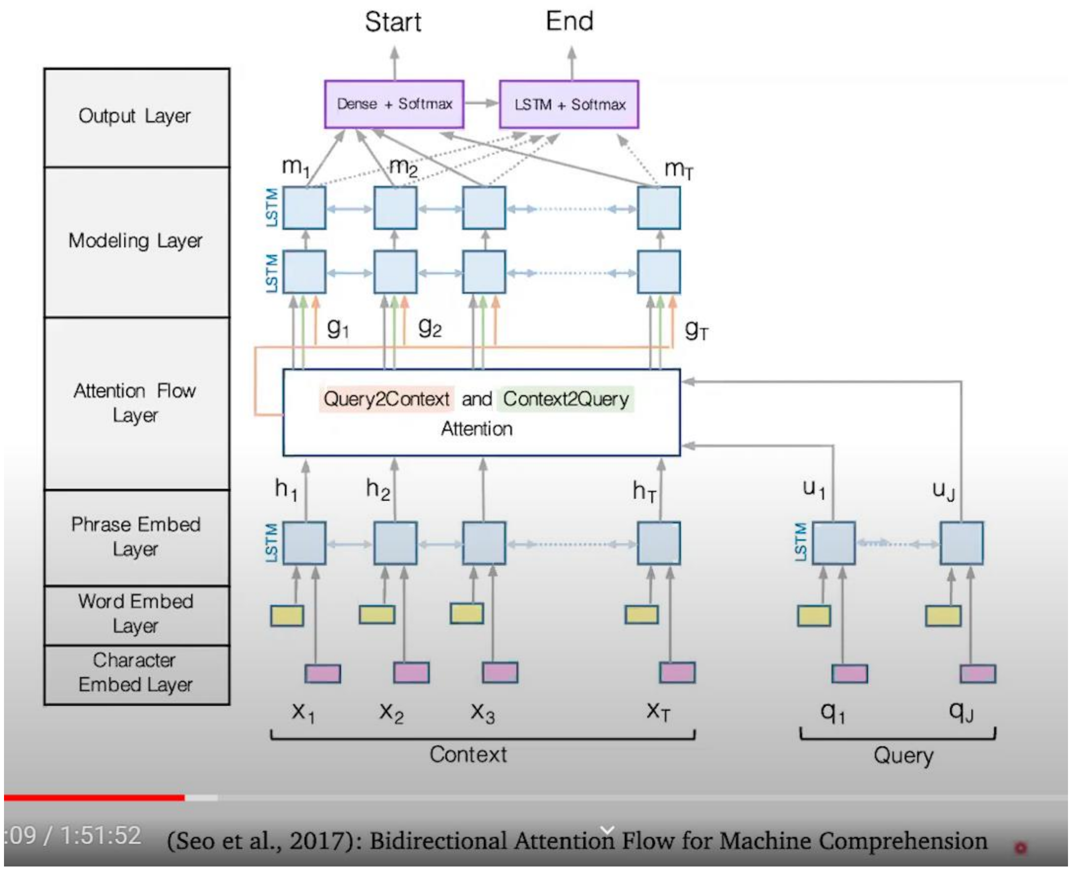
Neural models for reading comprehension(2400)
- Input: context token vector, question token vector
- Output: answer with specific lengths.

LSTM+Attention; fine tune BERT; (2700)

- Problem formulation
  - Input: $C = (c_1, c_2, \dots, c_N)$, $Q = (q_1, q_2, \dots, q_M)$, $c_i, q_i \in V$
  - Output: $1 \leq$ start $\leq$ end $\leq N$

LSTM+Attention details(2800)
- Two sequences: passage+questions
- Model words in passage are relevant to questions (attention comes in place?)

- Model architecture: bidirectional attention flow for machine comprehension: BIDAF(3000)



Encoding layer(bottom 3 layers)(3100)

GloVe word embedding + CNN character embedding -> 2 bidirectional LSTM for contextual embeddings for context and query

Attention layer

Context-to-query attention: Q is context, K is question

Query to context attention: context is K, question is Q

Modeling+output layer(4600)

Modelling interactions between context words

Output layer: predict start and end position of context that indicates where the answers in context for the question lies. (using softmax)

Training loss: (4900): cross entropy

$$\mathcal{L} = -\log p_{\text{start}}(s^*) - \log p_{\text{end}}(e^*)$$

BERT fine-tuning(5200)

Using models as testing datasets? Using a technique similar to GAN(5600)

Comparison of BERT and BIDAF models(5700)

- BERT uses attention between passages and questions(self-attention and cross attention)
- BIDAF doesn’t use attention, it models the interactions between questions and passages. Adding attention to BIDAF also improves performance(10000)

Pre-training objectives(10200)

- Masking consecutive set of words instead of randomly masking individual words

- Using two end points to predict the whole context between two end points

NLP adversarial attacks & out of distribution data(10500)

Open domain question answering(10900)

- Instead of reading comprehension where answers lie in context, this time language models have access to search engines and webs. However more challenging in practise
- Retriever-reading framweork: (11000)
  First use retriever to search and select all relevant webs/articles; then reader go through all of them and extract useful information.
  (11200) gives details of procedure.
- Training retriever based on BERT model, which encodes questions and context;
- Hypothesis that reader model maybe is not required; encoding every phrase of WIKIPEDIA and use techniques similar to GLOVE(distance metrics) for matching questions with potentially relevant WIKIPEDIA articles. -> makes looking up very fast!!!

Lec12: NLG: generation;

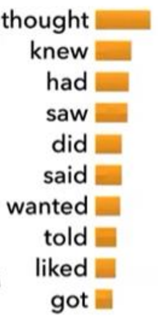
Relevant background(0913)

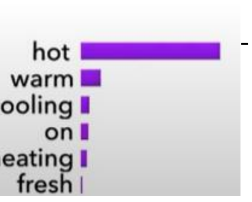
- Using softmax and previous context, output probability for each word token/choice.
- Using a language decoder, takes in probabilities and output a deterministic results(1100)
- Minimize neg-log-likelihood(1140); can be treated as classification tasks
- Summary(1300)

Decoding mechanism(1300)

$$\hat{y}_t = \underset{w \in V}{\textbf{argmax}}\, P(y_t = w | y_{<t})$$

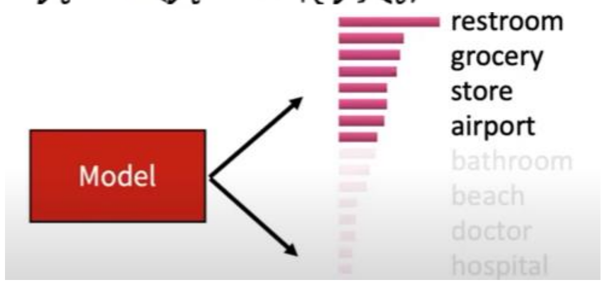
- Greedy methods: using argmax;
- Beam search: (lecture 7): keep track of "k" most probable tokens, and form a spanning tree for future predictions;
- Problems of repeating generated results(1500)
  If a phrase is being repeated for multiple times, the model is more confident that the next token would also be the same -> being fooled?(1600)
- Reducing it(1800): using a loss function to penalize generating repetitive outputs; minimize embedding distance between consecutive sentences; unlikelihood(two words are unlikely to occur in consecution)



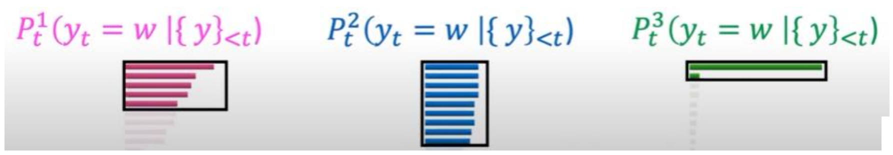


Sampling(2000)

$$\hat{y}_t \sim P(y_t = w \,|\{ y\}_{<t})$$



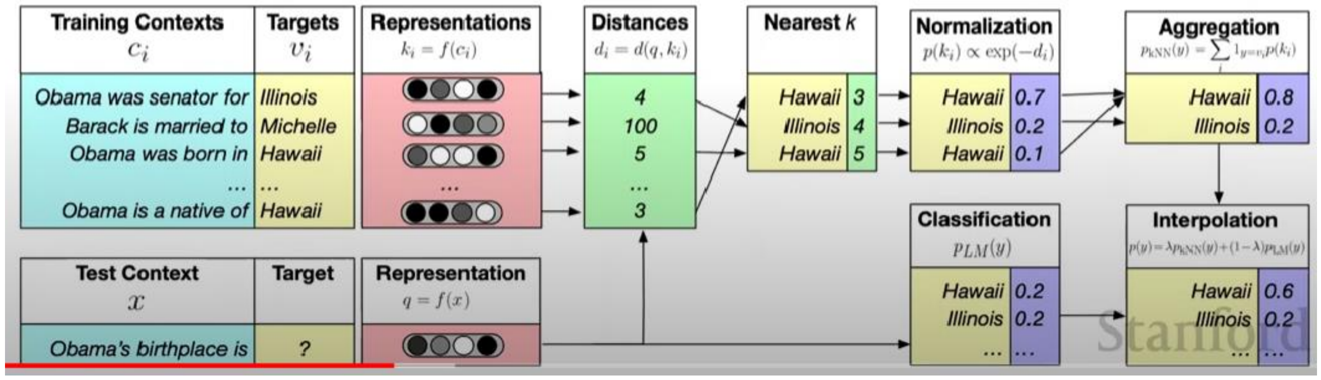
- Sample tokens/output words from a distribution of tokens
- Top-k-sampling decoding(2100)
  Most tokens should not be sampled at all (they should have absolutely zero probability of being sampled)
- Image shown left(2300)
  Idea of choosing appropriate "k": top: should include diversity; bottom: should cut out other possibilities;
  Is it possible to use a movable "k"? i.e, using a criterion to determine "k" for each output?
- Top-p sampling: (2400)
  Resolves above problem



$$P_t(y_t = w) = \frac{\exp(S_w/\tau)}{\sum_{w' \in V} \exp(S_{w'}/\tau)}$$

Sampling temperature(2500)
- See the formula shown right;
  \tau > 1: more diverse output; \tau<1: more convergent output.
- This method applies to all sampling methods(2600)

Re-balancing distributions using retrieval from n-gram phrase statistics(2700)



- Above is one example of using re-balancing distributions;
  The idea is to create a dataset(upper) acquired from training set, and calculate similarity score between context phrases and dataset phrases, use that as reference.

Bakcpropagation-based distribution re-balancing(3000)
- Define a loss criterion that imposes the properties of the desired output. This is done using a classifier-like structure. Then during training, the classifiers will be backpropagated with gradients that updates "intermediate activations/apply gradients on latent variables, NOT parameters" for generating a desired outputs

Re-ranking(3300)
- Using a specifically designed scores to assess the selected/sampled sequences' quality, and use the ranking to determine which sequence to output.
- Common criterion is using "perplexity"

Takeaways of decoding algorithms(3637)

Training issues using maximum likelihood(4200) teacher forcing?
- Repetitive phrases(addressed before)
  Unlikelihood training(4300): "y_neg" are "UNdesired tokens", can be set to words being generated before.

$$\mathcal{L}_{UL}^t = -\sum_{y_{neg} \in \mathcal{C}} \log(1 - P(y_{neg} \mid \{y^*\}_{<t}))$$

$$\mathcal{L}_{MLE}^t = -\log P(y_t^* \mid \{y^*\}_{<t}) \qquad \mathcal{L}_{ULE}^t = \mathcal{L}_{MLE}^t + \alpha \mathcal{L}_{UL}^t$$

- Exposure bias: models have access to complete sequences during training, yet it's not the case during generation: only have access to previously generated sequence tokens. (4600)

Solution: scheduled sampling, dataset aggregation(during training)

Sequence re-writing(4700)

Given a dataset, first use techniques similar to n-gram phrase(explained before) to find probable sequence outputs, then perform minor editing to contents based on previously generated sequences to better match with contexts.

Reinforcement learning techniques(4800)

Cast text generation model as Markov decision process, for sequence generation

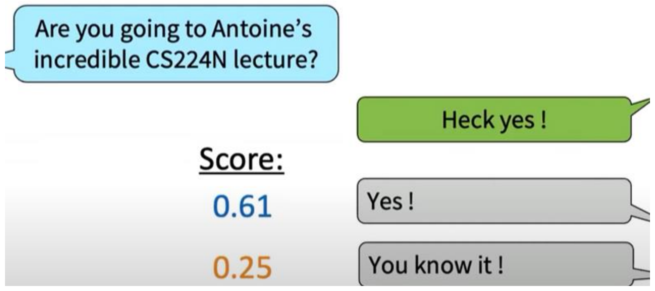
Perhaps different from RLHF?

Choosing reward function: (5000)

BLEU(machine translation), ROUGE(summarization)

But scores can be deceiving: high scores don't guarantee qualitative improvements(5139)

DRL for reward estimation (5207) (usually trained offline)



Evaluating NLG(5600)

- Content-overlap metrics(5800)
  Comparison between generated sequences and human-written ones(gold-standard)
  Computing a score: n-gram overlap, semantic overlap
  n-gram are not useful for machine translation(5900): no idea of semantic relatedness
- Model-based metrics(10100)
  Recall vector similarities in lecture 2/3? Using word distance
  BERTSCORE;
  Sentence movers similarity: using RNN to evaluate similarities in a continuous space
- Human-evaluations(10300)
  Need to develop better criterions(10500)
  Possibilities to consider: fluency, coherence/consistency, factuality, commonsense, style, grammaticality, typicality, redundancy
  Human evaluations can be subjective… be careful though;

Lec13: coreference resolution

Example(0700):

- Mentions of specific people and objects, and formulate relationship webs perhaps?
- Also identify what each pronoun represents?
- Applications(1700)
  Full-text&dialogue understanding, machine translation, etc.

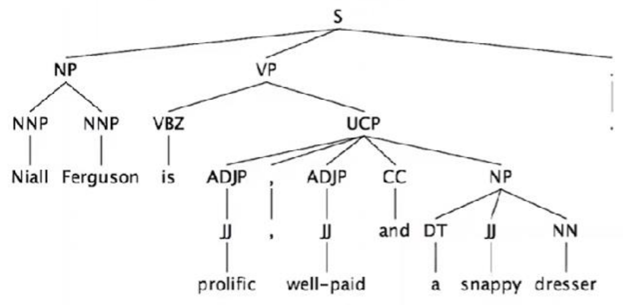
Mechanism(1900)

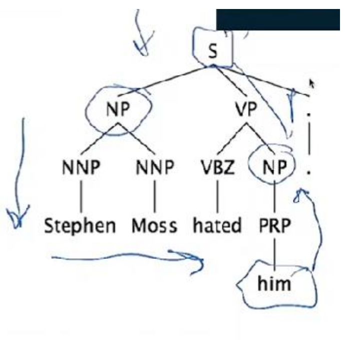
- Detect mentions; (2000) This is relatively easy;
  Pronous: part-of-speech-tagger;
  Named entities: named entity recognition system
  Noun parser: use a parser;

  Dealing with bad mentions: (2400)
  - Keep them first, and discard later if they are not being matched with anything with coreference relation

- Cluster mentions;
  Anaphora, antecedent, coreference(3100)

Coreference models(4000)
- Rule-based(4100): Hobbs algorithm




[redacted]

- Knowledge-based pronominal coreference(4900)
- Mention pair algorithm(5600)
  Binary classifier for EVERY pair…; trained using cross entropy loss

$$J = -\sum_{i=2}^{N}\sum_{j=1}^{i} y_{ij} \log p(m_j, m_i)$$

  Coreference links(5800): A->B and B->C then A-> C; pronouns and mentions
  - Realizing some pronous are used frequently to refer to DIFFERENT mentions… (10000)
  - Apply mention ranking to pick the most likely mentions for each pronoun. (10100)
    - Softmax
  - Using statistical classifier, simple NN, or advanced ones like LSTM/transformer.

Convnet in language(10300)
- Realizing the kernels have size “kernel_size * embedding_dim”, applied on word-embeddings of sequences. (10600)
- Convolution over sequences of “characters” (instead of words or phrases) (11000)
  Compute a representation of every sequences of characters

Explanation of one benchmark algorithm(11400~)
- Used LSTM+attention mechanism, where each word representations are then fed into a model for calculating coreference scores between two words.
- Using BERT is much better: spanBERT for coreference and Question-Answering. (11700)

Lec14: