CS 190I Notes

Alex Mei

Winter 2022

1 Unix Text Processing

1.1 Regular Expressions

- **Disjunction**: square bracket [ ] (e.g., [Ww]oodchuck for Woodchuck or woodchuck).
- **Range**: [A-Z] denotes a capital letter, [a-z] denotes lower-case letter, [0-9] denotes a digit.
- **Disjunction Negation**: carat ˆ means negation only when inside square bracket (e.g., [ˆ 0-9] for non-numeric character).
- **Global Disjunction**: pipe operator |(e.g., Woodchuck |woodchuck for Woodchuck or woodchuck).
- **Optional Character**: question mark ? denotes previous character is optional (e.g., colou?r for color or colour).
- **Klenne Star**: asterik * denotes 0 or more of previous character (e.g., oo*h! includes oh! ooh! ...).
- **Klenne Plus**: plus + denotes 1 or more of previous character (e.g., o+h! includes oh! ooh! ...).
- **Wildcard**: dot . denotes any character (e.g., beg.n for begin beg3n ...).
- **Anchors**: beginning carat ˆ denotes beginning of regex; ending dollar $ denotes end of regex.
- **Takeaway**: regex provides strong first model for text processing; harder tasks can use ML, but regex helps identify features.

1.2 Unix and Bash Commands

- **echo**: send arguments into stdout.
- **cat**: send file arguments' content into stdout.
- **head -n [line num]**: output up and not including [line num] into stdout. (Can use negative indices like Python.)
- **tail -n [line num]**: output starting [line num] into stdout. (Can use negative indices like Python.)
- **shuf**: shuffles line of text.
- **grep**: search for arg1 regex pattern in arg2
- **sort**: sort argument; use -n to sort using numeric string value; use -r to sort in reverse order.
- **uniq**: removes adjacent duplicates; use -c to group by count as first column.
- **tr**: replaces anything in arg1 with arg2; use -c to use the complement or arg1; use -s to squeeze repeats into one replace.
- **wc**: output newline, word, and byte count in 3 columns.
- **cut -f [col num]**: outputs [col num] column of tsv argument to stdout.
- **paste**: joints arguments as tsv columns and outputs to stdout.
- **rev**: reverses lines character wise.
- **sed**: search arg2 for all strings that matches [search] in arg1='s/[search]/[replace]/g' and replace with [replace].
- **awk –F [field] "{print $[col num]}"**: splits argument by field into columns, then prints content in [col num].

1.3 Text Normalization

- **Wordform**: an inflected form of a base word; asymmetric expansion is mapping word to wordform set.
- **Lemma**: set of wordforms for a base word. Lemmatization is mapping terms to base word.
- **Morpheme**: meaningful units that make up words.
- **Stem**: core meaning-bearing units; **Affix**: grammatical functions that adhere to stems; **Stemming**: removing affixes.
- **Porter's Algorithm**: series of rules to help stem English words.
- **Type**: element in the vocabulary (V); **Token**: instance of a type; total instances (N).
- **Sentence Segmentation**: split by unambiguous terms (!, ?). Period is ambiguous; build a binary classifier.
- **Language Issues**: Some languages have no spaces, multiple date formats, multiple alphabets.

2 Probabilistic Language Models

- **Prior (Unconditional) Probability**: probabilities associated with a proposition or variable, prior to any evidence.
- **Goal**: assign probability to a sequence of words; **Related Task**: assign probability for an upcoming word.

- **Chain Rule**:

$$P(w_1, ..., w_n) = \Pi_i P(w_i | w_1, ..., w_{i-1})$$

- **kth Order Markov Assumption**:

$$P(w_i | w_1, ..., w_{i-1}) \approx P(w_i | w_{i-k}, ..., w_{i-1})$$

- **Unigram Model**:

$$P(w_1, ..., w_n) = \Pi_i P(w_i)$$

- **Bigram Model**:

$$P(w_1, ..., w_n) = \Pi_i P(w_i | w_{i-1})$$

- **N-Gram Model**: lower order n may not capture long range dependencies, but higher order n require big data to generalize.
- **Start / End Tokens**: tokens added for estimating higher order n-grams; otherwise, excludes n-grams that give context.
- **Maximum Likelihood Estimate**:

$$P(w_i) = \frac{count(w_i)}{count(w_*)}; P(w_i | w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

- **Underflow Issue**: transform operations to log space to prevent near zero-probabilities and increase efficiency with addition.
- **Perplexity**: inverse probability of test set; average branching factor.

$$PP(w_1, ..., w_n) = P(w_1, ..., w_n)^{1/n} = \exp(-\frac{1}{n} * \ln(P(w_1, ..., w_n)))$$

- **Extrinsic Evaluation**: best evaluation for model comparison; time consuming and resource intensive.
- **Intrinsic Evaluation**: bad generalization approximation and only useful for pilot tests; easier to evaluate.
- **Type I / False Positives**: strings matched when they should not. Minimize by increasing accuracy / precision.
- **Type II / False Negatives**: strings not matched when they should. Minimize by increasing coverage / recall.

2.1 Smoothing

- **Out of Vocabulary**: open vocabulary task where entire vocabulary is unknown; use UNK token.
- **OOV Training**: normalize unknown text with UNK token and train as normal.
- **OOV Evaluation**: replace unknown words with UNK token to calculate probability.
- **Storage**: prune infrequent n-grams; use data structures and efficient storage processes (bits, indices).
- **Zeros**: occurrences in test set not seen in training data; assigns 0 probability and undefined perplexity.
- **Smoothing**: shift probability mass to lower frequencies to generalize better.
- **Laplace Add-One Smoothing**: better in domains with few zeros. For bigram MLE,

$$P(w_i | w_{i-1}) = \frac{count(w_{i-1}, w_i) + 1}{count(w_{i-1}) + V} = \frac{count^*(w_{i-1}, w_i)}{count(w_{i-1})}$$

- **Reconstituted Counts**: smoothing-adjusted frequencies.

$$count^*(w_{i-1}, w_i) = \frac{(count(w_{i-1}, w_i) + 1) * count(w_{i-1})}{count(w_{i-1}) + V}$$

- **Backoff**: use trigram if good evidence exists, otherwise use bigram, otherwise unigram.
- **Interpolation**: adaptation of backoff that weights trigram, bigram, and unigram models.
- **Discriminative Modeling**: choosing weights to improve a task instead of the training data.
- **Cache Models**: assumes recent words are more likely to appear.
- **Absolute Discounting**: discount each term a constant amount. (d = 0.75 works well.)

$$P(w_i | w_{i-1}) = \frac{count(w_{i-1}, w_i - d)}{count(w_{i-1})} + \lambda(w_{i-1}) P(w_i)$$

- **Kneser-Ney Smoothing**: discounting with continuation probability (likelihood a word completes a bigram, normalized by bigram count). Lambda uses the normalized discounting * times discounted needed to transfer probability mass.

$$P_{continuation}(w_i) = \frac{|\{w_{i-1} : c(w_{i-1}, w_i) > 0\}|}{|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}|}$$

$$P(w_i | w_{i-1}) = \frac{\max(c(w_{i-1}, w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1}) * P_{continuation}(w_i)$$

$$\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})} * |\{w_i : c(w_{i-1}, w_i) > 0)\}|$$

3 Machine Learning

- **Generative Models**: estimate joint probability, uses bayes rules; estimates actual distributions.
- **Discriminative Models**: estimate conditional probability without Bayes; estimates decision boundaries.

3.1 Optimization

- **Optimization Step**: $w(t+1) = w(t) + \eta\hat{v}$
- **Direction Vector**: pick $\hat{v}$ to minimize $E(w(t))$ direction.

$$g_t = \nabla_w E(w(t)); v_t = -g_t; \hat{v} = -\frac{g_t}{||g_t||_2}$$

- **Takeaways**: simple, effective, scalable; assumes smooth (differentiable) functions; need convexity to guarantee global min.

3.2 Stochastic Gradient Descent

- Sample a subset of data to calculate gradient to be less memory intensive; more scalable.
- **Tradeoff**: improves efficiency by introducing noise, making convergence rate longer. The average move is the same as gradient descent, but simpler, and can help escape local minimum for nonconvex functions.

3.3 Naive Bayes

- **Baye's Rule**:

$$P(X|Y) = \frac{P(Y|X) * P(X)}{P(Y)}$$

- **Naive Bayes Model**:

$$c(x) = \max_{c_j \in C} P(x_1, ..., x_n|c_j) * P(c_j) = \max_{c_j \in C} P(c_j) * \Pi_{1 \leq i \leq n} P(x_i|c_j)$$

- **Naive Bayes Assumption**: Assume the probability of observing $P(x_1, ..., x_n|c_j)$ is equal the product of $P(x_i|c_j)$.
- **Text-based Naive Bayes**: represent feature vector as a bag of words model.

$$P(w_i|c_j) = \frac{count(w_i) \in (texts \in c_j) + \alpha}{count(w_*) \in (texts \in c_j) + \alpha * V}$$

3.4 Perceptron

- **Input**: vector $x = [x_1, ..., x_d]^T$ with d dimensions.
- **Output**: binary class mapped from a score $= \sum_{i=0}^{d} w_i x_i$ based on some threshold.
- **Perceptron Function**: a linear/hyperplane separator $h(x) = sign(w^T x)$.
- **Multiclass Perceptron**: train weights to classify whether an element is in a class; take argmax of weights to classify.
- **Issues**: cannot calculate model uncertainty; sign function not differentiable.

3.5 Voted Perceptron

- **Intuition**: when an example is misclassified, save previous hyperplane and initialize a new one with an update.
- **Prediction**:

$$\hat{y}_j = sign(\sum_{i}^{k} c_i * sign(v_i^T x_j))$$

3.6 Logistic Regression

- **Classifier**:

$$P(y|x) = \theta(w^T x)$$

- **Sigmoid Function**:

$$\theta(s) = \frac{e^s}{1+e^s} = \frac{1}{1+e^{-s}}; \theta(s) + \theta(-s) = 1$$

- **Log Conditional Likelihood**:

$$LCL(w) = \sum_{i}^{n} \ln(P(y_i|x_i, w)) = \sum_{i}^{n} \frac{1}{1+\exp(-w^T x_i)}; \nabla_w LCL(w(t)) = \sum_{i}^{n} (y_i - p_i) x_i$$

- **Softmax Function**: given K classes,

$$softmax(z) = \frac{\exp(z)}{\sum_k^K \exp(z_k)}$$

- **Multiclass Formulation**: use softmax function for K classes and pick highest probable class,

$$P(y = k \in K|x) = softmax(w^T x) = \frac{\exp(w_k^T x)}{\sum_k^K \exp(w_k^T x)}$$

4 Hidden Markov Models

- **Sequence Labeling Task**: assign tokens in a sequence a label; labels depend on neighbor context (not iid).
- **Sequence Labeling Issues**: difficult to integrate nearby context; difficult to propagate uncertainty between decisions and collectively determine joint classification of entire sequence.
- **Probabilistic Sequence Model**: integrates uncertainty over multiple, interdependent classifications and collectively determine the most likely global assignment (e.g., HMM, CRF).
- **Issues**: only use word identity as features when we can use richer representations; also, task for training and testing differ.
- **Assumptions**: Model is in finite hidden set of states; state transitions and token generation given state are probabilistic.

4.1 Formal Definition of HMM

- Set S of $N + 2$ states (including start s_0 and end s_f). Set V of M vocabulary observations.
- Set A of transition probabilities:

$$a_{ij} = P(q_{t+1} = s_j|q_t = s_i)$$

$$\sum_j^N a_{ij} + a_{if} = 1$$

- Set of observation probabilities for each state:

$$b_j(k) = P(v_k|q_t = s_j)$$

4.2 Observation Likelihood (Emission) Task

- **Definition**: determine the likelihood of an observation sequence O given a learned HMM $\lambda = (A, B)$; $P(O|\lambda)$.
- **Naive Solution**: sum the probabilities of all possible state sequences of $len(O) = T$. $O(TN^T)$ time complexity.

$$P(O|\lambda) = \sum_Q P(O, Q|\lambda) = \sum_Q P(O|Q, \lambda)P(Q|\lambda)$$

4.2.1 Forward Algorithm

- **Assumption**: utilizes first-order Markov assumption to compute probabilities.

$$a_1(j) = a_{0j} * b_j(o_1)$$

$$a_t(j) = [\sum_{i=1}^N a_{t-1}(i) * a_{ij}] * b_j(o_t)$$

$$P(O|\lambda) = \sum_{i=1}^N a_t(i) * a_{if}$$

- **Runtime**: $O(TN^2)$.

4.3 Most Likely State Sequence (Decoding) Task

- **Definition**: determine the state sequence Q given a learned HMM λ and a sequence O; $P(Q|O, \lambda)$.
- **Viterbi Algorithm**: same as forward algorithm, except use max instead of sum and keep back pointers for final sequence.

4.4 Maximum Likelihood Training (Learning) Task

- **Definition**: learn HMM parameters λ given a set of sequences and labels.

$$a_{ij} = \frac{count(q_t = s_i, q_{t+1} = s_j)}{count(q_t = s_i)}; b_j(k) = \frac{count(q_i = s_j, o_i = v_k)}{count(q_i = s_j)}$$

4.5 Maximum Entropy Markov Model

- **Definition**: log-linear discriminative model where each state depends on the previous and the observation sequence; more expressive than HMM since we can use more descriptive (and real-valued) features as input to logistic classifier for each state.
- **Formal Definition**: given state q, observation o, weight w, feature function f,

$$P(Q|O) = \prod_{i=1}^{n} P(q_i|q_{i-1}, o_i); P(q_i|q_{i-1}, o) = \frac{\exp(\sum_i^n w_i f_i(o, q_i))}{Z(o, q_{i-1})}$$

$$Z(o, q_i) = \sum_{o \in O} \exp(\sum_{i=1}^{n} w_i f_i(q_i, o))$$

- **Training and Inference**: use viterbi, replacing prior and likelihood probabilities with posterior:

$$v_t(j) = \max_i(v_{t-1}(i) * P(q_j|q_i, o_t))$$

- **Probability Estimation**:

$$a_t(q_j) = \sum_{q_i \in Q} a_{t-1}(q_i) * P(q_j|q_i, o_t)$$

- **Label Bias Problem**: no global view of features, probabilities of outgoing state transitions are normalized.
- **BIOES Tagging Scheme**: Beginning, Inside, Outside, End, Single Token (both beginning and end).

5 Conditional Random Field

- **Graphical Models**: represent variable dependencies in directed (Bayesian) or undirected (Markov) graph format.
- **Structure Learning**: learning to determine edges.
- **Conditional Probability Table (CPT)**: associated with each node, defined by $P(node|parents)$.
- **Markov Blanket**: adjacent nodes of a given node.
- **Clique**: subset of vertices in an undirected graph where every pair of vertices are adjacent.
- **Markov Net**: defined by a set of potential functions ϕ_k for each clique k, assigning a nonnegative value for a pair of nodes denoting their compatibility; no independence assumptions needed.

$$P(x_1, ..., x_n) = \frac{1}{Z} \prod_k \forall_{i,j \in k} \phi_k(x_i, x_j)$$

$$Z = \sum_x \prod_k \forall_{i,j \in k} \phi_k(x_i, x_j)$$

- **Linear Chain CRF**: specific log-linear case of Markov Network, given feature k, input data X and labels Y of length t,

$$P(Y|X) = \frac{1}{Z(X)} \exp(\sum_{t=1}^{T} \sum_{k=1}^{K} \lambda_k f_k(Y_t, Y_{t-1}, X_t))$$

$$Z(X) = \sum_Y \exp(\sum_{t=1}^{T} \sum_{k=1}^{K} \lambda_k f_k(Y_t, Y_{t-1}, X_t))$$

- **Skip Chain CRF**: include long distance edges to incorporate longer dependencies, but makes exact inference difficult.
- **CRF Training**: use SGD to optimize λ, easier to do and converse since log-linear.
- **Summary**: CRFs are discriminative, SOTA for sequence labeling, and improve results on IE, but does not scale well.

6 Distributional Semantics

- **Intuition**: two words are similar if they have similar word context.
- **Embeddings**: encode the meaning of a word in a vector (rather than index).
- **Dense Vectors**: reduces number of weights to tune, generalizes better, and captures synonyms.

6.1 Term Frequency, Inverse Document Frequency (TFIDF)

$$TF(term) = \frac{count(term, doc)}{count(words \in doc)}$$

$$IDF(term) = 1 + log(count(docs)/count(docs_{term}))$$

$$weight(term, doc) = TF(term, doc) * IDF(term)$$

- **Advantages**: effective, intuitive, easy to implement, well-studied and evaluated.
- **Disadvantages**: assumes term independence, doc/query equivalence, many hyperparameters, not for word-word similarity.
- **Term Document Matrix**: document columns, word rows, tf cells.
- **Term Context Matrix**: use window of context instead of entire document; smaller windows highlight syntax, larger windows highlight semantics; very sparse matrix.
- **Syntagmatic Association**: first-order co-occurence; words are nearby each other.
- **Paradigmatic Association**: second-order co-occurence; words have similar neighbors.
- **Syntactic Dependencies**: count of context in dependencies instead of raw words.

6.2 Positive Pointwise Mutual Information (PPMI)

- **Pointwise Mutual Information (PMI)**: measures co-occurence versus independence, intersection of conditional entropy

$$PMI(x, y) = \log_2(\frac{P(x, y)}{P(x)P(y)})$$

- **PPMI**: negative values uninterpretable, replace with 0.
- **PMI Bias**: rare words have very high PMI, want to decrease; can use smoothing or increase probability.
- **Weighting Context**: give rare context higher probability using $\alpha < 1$

$$P_\alpha(c) = \frac{count(c)^\alpha}{\sum_c count(c)^\alpha}$$

- **Cosine Similarity**: normalized dot product, fixes dimensionality and size issues.

$$\frac{q^T d}{||q||_2||d||_2}$$

6.3 Singular Value Decomposition (SVD)

- **Eigendecomposition**: a real-valued linearly independent matrix A can be written as $U\Lambda U^{-1}$ where U is an orthogonal matrix containing the the eigenvectors of A and Λ is a diagonal matrix with the associated eigenvalues in decreasing order.
- **SVD Embeddings**: each row of the U matrix is a $k \leq M$ dimensional representation.
- **Principal Component Analysis**: SVD applied on covariance matrix XX^T to capture the highest variance dimensions.
- **Benefits**: denoise unimportant information, capture higher order co-occurence.
- **SVD Algorithm**: Solve for λ with $(A^T A - \lambda^2 I)x = 0$; Compute corresponding eigenvectors using $A - \lambda I = 0$.

6.4 Prediction Based Models

- **Definition**: train NN to predict neighbors to learn dense representation; faster than SVD, available online and pretrained.
- **Skip Gram**: predict each neighboring words in context window (i.e., window of plus or minus 2).
- **Skip Gram Learning**: Learns an input (input to projection weight) and output (projection to output) embedding, which we can use one, concatenate, or sum.
- **Skip Gram Task**: Pick highest probability neighbors given projected input.

7 Machine Translation

- **Premise**: decode messages by denoising source language to target language.
- **Bayesian Analysis**: best translation $\hat{T}$ given translations T and source S. $P(S|T)$ and $P(T)$ denotes the translation and language models respectively.

$$\hat{T} = \max_{T \in Target} P(T|S) = \max_{T \in Target} P(S|T) * P(T)$$

- **Phrase-Based Models**: translating segmented phrases t_i into s_i with translation probability $\phi(s_i, t_i)$ and distorted probability $d(i)$ for reordering phrase i, and stopping when end token is generated.

$$P(S|T) = \prod_i \phi(s_i, t_i) d(i)$$

- **Translation Probabilities**: assumes phrased perfectly-aligned parallel corpus, can use random and train to convergence.

$$\phi(s, t) = \frac{count(s, t)}{\sum_s count(s, t)}$$

- **Distortion Probability**: language-pair sensitive, given decay $0 < \alpha < 1$ and normalization c s.t. $\sum_i d(i) = 1$

$$d(i) = c\alpha^{|index(translation(t_i)) - index(translation(t_{i-1}))|}$$

- **Word Alignment**: easier than phrases, but supervised is expensive and rare; use unsupervised expectation-maximization.
- **BLEU**: precision measure of n-gram overlap with reference summary; can clip to single reference.

$$BLEU = \frac{grams_{generated} \in reference}{grams_{generated}}$$

7.1 Human Evaluation

- **Fluency**: grammatical, understandable, readable. **Edit Cost**: number of corrections human translator must make.
- **Adequacy**: human judgement on completeness, using bilinguists and monolinquists. **Fidelity**: correct information.
- **Informativeness/Task-Based Evaluation**: use translation to answer questions about source.

8 Question Answering

- **Factoid Questions**: factual, in commercial systems, as opposed to more complex narrative questions.
- **Information Retrieval Approach**: directly retrieve answer from documents in corpus.
- **Knowledge Based Approach**: build semantic representation of query and use to fetch from knowledge structure.
- **Hybrid Approach**: find candidates using shallow augmented IR; score candidates with full knowledge source.
- **Accuracy**: does answer match gold-label?
- **Mean Reciprocal Rank**: N queries, query score is 1/rank of first correct answer in list of M candidates; 0 if incorrect.

$$MRR = \frac{\sum_{i=1}^{N} rank_i^{-1}}{N}$$

8.1 Watson Architecture

1. Question Processing: focus, answer type, question classification, relation extraction, tagging, parsing, coreference.
- **Focus**: part of question that co-refers with answer; replace with answer to find passage; extract with production rules.
- **Lexical Answer Type**: semantic class of answer; extracted using production rules.
- **Relation Extraction**: production rules for most frequent, and distant supervision for rest.

2. Candidate Answer Generation: query existing DB using 3-tuples with extracted relations.
- **Fetching Candidates**: use standard IR QA.
- **Fetching Answers**: use production rules.

3. Candidate Answer Scoring: ensemble evidence sources, including lexical answer type (strict filter for IR factoid QA).

9 Neural Networks

- **Expressiveness**: single-layer is a linear separator, two-layer can make a ridge, and three-layers can make a bump.
- **Deep Neural Net**: multiple hidden layers.
- **Formulation**:

$$y = f(x) = \sigma(W^L ... \sigma(W^1 x))$$

- **Hyperbolic Tangent**:

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- **Rectified Linear Unit**:

$$RELU(x) = \max(x, 0)$$

- **Non Linearity**: layers can approximate more complex functions; without it, would just be linear transform.
- **Dropout**: randomly delete features/weights and train ensemble; convolve predictions of nets to prevent adapatation to specific features/weights; help with NN overfitting.

9.1 Backprogation

- **Premise**: too many parameters and strong dependencies to use forward gradient.
- **Computational Graph**: inward arrow indicate dependencies of node.

$$\frac{\partial}{\partial a}(a+b) = \frac{\partial a}{\partial a} + \frac{\partial b}{\partial a} = 1; \frac{\partial}{\partial a}(a*b) = a\frac{\partial b}{\partial a} + b\frac{\partial a}{\partial a} = b$$

- The derivative of a path is a product of the edges of the path.
- The derivative from node a to b is the sum of all derivative paths $a \to b$ (in forward mode).
- **Forward Differentiation**: tracks how one input affects every node; tells us how to adjust the input.
- **Backward Differentiation**: tracks how one output affects every node; tells us how to adjust the weights.

9.2 Recurrent Neural Network

- **Goal**: pass local information through hidden state.
- **Input**: list of word vectors $x_1, ..., x_t$.
- **Timestep Definitions**: given activation σ_i applied element-wise ($\sigma_1 = \tanh$, $\sigma_2 = softmax$) initialized hidden vector a_0.

$$a_t = \sigma_1(W^{aa}a_{t-1} + W^{ax}x_t)$$

$$h_t = \sigma_2(W^{ha}a_t)$$

- **Vanishing Gradient**: multiplicative gradient (from weights) exponentially magnifying with respect to number of layers.
- **Long Term Dependencies**: unlikely to learn from passing hidden vector due to vanishing gradient.
- **Autoregressive**: next stage depends on previous, not independent; preserves some order.
- **Advantages**: handles any input length using same model; computation considers history; shared weights across time.
- **Other Issues**: difficult to parallelize, cannot access future input.

9.3 Long Short Term Memory Networks (LSTMs)

- **Core Idea**: cell state can easily pass information to next with minor interactions (forget gate, new memory cell).
- **Gating**: use sigmoid to describe how much information is let in.
- **Forget Gate**: how much previous information to forget

$$f_t = \sigma(W_f[h_{t-1}, x_t])$$

- **Input Gate**: adds current information

$$i_t = \sigma(W_i[h_{t-1}, x_t])$$

- **New Memory Cell**: creates new memories

$$c_t = \tanh(W_c[h_{t-1}, x_t])$$

- **Final Memory Cell**: use element-wise multiplication, stores long term memory

$$C_t = f_t * C_{t-1} + i_t * c_t$$

- **Output Gate**: filtered version of cell state

$$o_t = \sigma(W_o[h_{t-1}, x_t])$$

$$h_t = o_t * \tanh(C_t)$$

- **Advantages**: fixes long term dependencies, and mitigates gradient issues.
- **Disadvantages**: resource intensive, prone to overfitting, difficult to apply dropout, more complex gradient issues.

9.4 Sequence to Sequence Models

- **Encoder**: incorporate inputs into hidden state.
- **Decoder**: translate hidden state into output.
- **Issues**: does the last hidden state capture all the information?

9.5 Convolutional Neural Nets (CNNs)

- **Goal**: recognize spacial invariants and capture local context.
- **Convolution Layer**: perform convolution operation on filter size F with stride S.
- **Max Pooling Layer**: perform max (or mean) operation on regions to downsample and capture most important activation.
- **Fully Connected Layer**: flatten input.
- Unlike feed-forward NNs, CNNs do not have global view of all features.
- **Multi Channel**: one static copy of input weights + one backprop optimized copy; merge channels before pooling.
- **Multi Filter**: useful to have different window sizes; can also learn complimentary features of same size; concatenate pooled features of different filters and apply softmax to get merged feature.
- **Advantages**: easy to parallelize, exploits local dependencies.
- **Disadvantages**: long-distance dependencies need many layers.

10 Transformers

- **Premise**: non-recurrent encoder-decoder for machine translation.
- **Encoder Block**: multi-head-attention and 2-layer feed forward NN with RELU, with residual connection and layer norm.
- **Layer Norm**: normalize input to have 0 mean and 1 variance per layer and training point.
- **Positional Encoding**: add temporal information.
- **Training**: use byte-pair encodings, checkpoint averaging, ADAM optimizers, dropout at every layer before adding residual, smoothing, auto-regressive decoding with beam search and length penalities.

10.1 Attention

- **Motivation**: alignment of terms in translation.
- **Idea**: replace recurrence architecture and allow access into any state for long term dependencies.
- **Input**: query $q \in R^k$ against a set of keys $k \in R^k$ to determine candidates $v \in R^v$.
- **Dot Product Attention**:

$$A(q, K, V) = \sum_i \frac{\exp(q^T k_i)}{\sum_j \exp(q^T k_j)} v_i = softmax(QK^T)V$$

- **Self-Attention**: take current timestep input as query with other inputs as keys; not autoregressive.
- **Encoder Self-Attention**: can use both past and future inputs.
- **Decoder Self-Attention**: cannot use future inputs; partial attention, only see encoder info.
- **Label Bias**: training data can see into future but testing task cannot.
- **Advantages**: easy to parallelize, constant path length between two positions, global model that looks at all pairs.
- **Multihead Attention**: capture semantic dimensions with parallel attention layers with different linear transforms on input/output; concatenate multiple attention heads.
- **Scaled Dot Product Attention**: fix variance increase with dimension size increase; softmax large and gradient small.

$$A(Q, K, V) = softmax(QK^T/\sqrt{d_k})V$$

10.2 Bidirectional Encoder Representations from Transformers (BERT)

- Learn word vectors using long context with transformers.
- Language understanding is bidirectional.
- Model relationship between sentences with next sentence prediction.
- **Input**: word-level token embeddings, sentence-level segment embeddings, position embeddings.