

Backpropagation
<p>Linear-time dynamic program for computing derivatives.</p> <ol style="list-style-type: none"> Write down composite function as a labeled, acyclic, directed hypergraph with intermediate variables as nodes and hyperedges labeled with primitive functions; Given input, perform forward propagation; Run backpropagation on the graph using forward values. $\frac{\partial y_i}{\partial x_j} = \sum_{p \in P(j,i)} \prod_{(k \rightarrow \ell) \in p} \frac{\partial z_\ell}{\partial z_k}$ $\sin'(x) = \cos(x), \cos'(x) = -\sin(x), y^x = \exp(x \log(y)), \log'(x) = \frac{1}{x}, \exp'(x) = \exp(x).$

Log-linear modelling
$\text{score}(y, x) = \theta^\top f(x, y).$ <p>Result of NLL gradient equal to 0:</p> $\sum_{i=1}^n f(x_i, y_i) = \sum_{i=1}^n \mathbb{E}_{y_i x_i, \theta} [f(x_i, y_i)].$ <p>Hessian ($H_\theta = \nabla_{\theta^\top} \nabla_{\theta}$):</p> $H_\theta \left(\sum_{i=1}^n -\log p(y_i x_i) \right) = \sum_{i=1}^n \text{Cov}_{y_i x_i, \theta} [f(x_i, y_i)].$ <p>To derive this, first derive the following:</p> $\nabla_{\theta^\top} p(y_i x_i) = p(y_i x_i) \left(f(x_i, y_i)^\top - \mathbb{E}_{y' x_i} [f(x_i, y')]^\top \right).$ <p>Softmax: $\text{SOFTMAX}(h)_y = \frac{\exp(h_y/T)}{\sum_{y' \in \mathcal{Y}} \exp(h_{y'}/T)}$.</p> <p>$T \rightarrow 0$: argmax (structured perceptron). $T \rightarrow \infty$: uniform.</p> <p>Exponential family: $p(x \theta) = \frac{1}{Z(\theta)} h(x) \exp(\theta^\top \phi(x))$.</p> <p>$Z(\theta)$: partition function. $h(x)$: support. θ: parameters. $\phi(x)$ feature function.</p> <p>Advantage: conjugate priors, compress all data into finite parameters $\phi(x)$ without losing information.</p>

Multi-layer perceptron
<p>Problem with log-linear modelling: Data must be linearly separable. Solution: Hack the non-linearity into the feature function.</p> <p>Problem: We need to know the decision boundary shape a priori.</p> <p>Solution: Learn the non-linear feature function with an MLP:</p> $h_k = \sigma_k \left(W_k^\top h_{k-1} \right), \quad h_1 = \sigma_1 \left(W_1^\top e(x) \right),$ <p>Then $\text{SOFTMAX}(\theta^\top h_n)$ for prob. dist.</p> <p>$e(x)$ obtained with Skip-Gram: predict whether 2 words are within same context. Positive and negative samples (no normalizer needed). Intuition: need good representation of words for this task that can be used by other tasks.</p> <p>Derivative:</p> $\frac{\partial \ell}{\partial W_k} = \frac{\partial \ell}{\partial y} \frac{\partial y}{\partial h_n} \left(\prod_{m=k+1}^n \sigma'_m(\dots) W_m \right) \sigma'_k(\dots) h_{k-1}.$

Structured prediction
<p>To be able to train a model, we need to compute:</p> $p(y x) = \frac{\exp \text{score}(y, x)}{Z(x)}, \quad Z(x) = \sum_{y' \in \mathcal{Y}} \exp \text{score}(y', x).$ <p>Problem: \mathcal{Y} is exponentially or infinitely large. This makes $Z(x)$ inefficient/impossible to compute during training, and it is hard to find the best $\hat{y} \in \mathcal{Y}$ during inference. Solution: Design algorithms that make use of the structure of the input and output.</p>

Language modelling
$p(\mathbf{y}) = p(\text{EOS} \mathbf{y}) \cdot \prod_{i=1}^N p(y_i \mathbf{y}_{<i})$ $p(y_i \mathbf{y}_{<i}) = \frac{1}{Z(\mathbf{y}_{<i})} \exp \text{score}(y_{<i}, y_i).$ <p>Problem: Model is non-tight (total prob. $\neq 1$) if we have a history that never ends in EOS. Solution: Force $p(\text{EOS} \mathbf{y}_{<i}) > \xi > 0$ for every possible history $\mathbf{y}_{<i}$.</p> <p>Problem: infinitely many histories. n-gram: $p(y_i \mathbf{y}_{<i}) = p(y_i y_{i-n+1}, \dots, y_{i-1})$ (only look at last $n-1$ word counts). Problem: probability can be 0. Solution: Laplace smoothing. Problem: Related sentences have no dependency. Neural n-gram (Bengio et al.): Use embeddings and MLP with $n-1$ history as input and output dist. over next words. Problem: Unrealistic assumption of n-grams. RNN: $h_i = \sigma(W_h h_{i-1} + W_x e(y_{i-1}) + b)$ (encode entire history using hidden states).</p> <p>Derivative:</p> $\frac{\partial \ell_i}{\partial W_h} = \frac{\partial \ell_i}{\partial y_i} \frac{\partial y_i}{\partial h_N} \sum_{j=1}^i \left(\prod_{m=j+1}^i \sigma'(\dots) W_h \right) \sigma'(\dots) h_{i-1}.$ <p>Problem: Vanishing gradient. Solution: LSTM/GRU.</p>

Semirings
<p>Definition monoid $\langle \mathbb{K}, \odot, e \rangle$:</p> <ol style="list-style-type: none"> \odot is associative: $(x \odot y) \odot z = x \odot (y \odot z)$. $e \in \mathbb{K}$ is the identity element: $x \odot e = x$. <p>Definition semiring $\langle \mathbb{K}, \oplus, \otimes, \mathbf{0}, \mathbf{1} \rangle$:</p> <ol style="list-style-type: none"> $\langle \mathbb{K}, \oplus, \mathbf{0} \rangle$ is a commutative monoid: $x \oplus y = y \oplus x$. $\langle \mathbb{K}, \otimes, \mathbf{1} \rangle$ is a monoid. \otimes distributes over \oplus: $(x \oplus y) \otimes z = (x \otimes z) \oplus (y \otimes z)$ and $z \otimes (x \oplus y) = (z \otimes x) \oplus (z \otimes y)$. $\mathbf{0}$ is an annihilator of \otimes: $\mathbf{0} \otimes x = \mathbf{0}$ and $x \otimes \mathbf{0} = \mathbf{0}$. <p>Closed: $x^* = \bigoplus_{n=0}^{\infty} x^{\otimes n}$. The Kleene star must be in \mathbb{K} and obey the following two axioms: $x^* = \mathbf{1} \oplus x \otimes x^*$ and $x^* = \mathbf{1} \oplus x^* \otimes x$.</p> <p>Boolean: $\langle \{0, 1\}, \vee, \wedge, 0, 1 \rangle$. Viterbi: $\langle [0, 1], \max, \times, 0, 1 \rangle$. Inside: $\langle \mathbb{R}^+ \cup \{\infty\}, +, \times, 0, 1 \rangle$, Real: $\langle \mathbb{R} \cup \{\infty\}, \min, +, \infty, 0 \rangle$, Tropical: $\langle \mathbb{R}^+ \cup \{\infty\}, \min, +, \infty, 0 \rangle$, Log: $\langle \mathbb{R} \cup \{\infty\}, \log(\exp a + \exp b), +, -\infty, 0 \rangle$, Expectation: $\langle \mathbb{R}^2, \langle a_1 + b_1, a_2 + b_2 \rangle, \langle a_1 b_1, a_1 b_2 + a_2 b_1 \rangle, \langle 0, 0 \rangle, \langle 1, 0 \rangle \rangle$, Counting: $\langle \mathbb{N}, +, \times, 0, 1 \rangle$.</p>

Part-of-speech tagging
<p>Input: $w \in \Sigma^N$. Output: $t \in \mathcal{T}^N$.</p> <p>CRF: Classifier that makes use of the structure. (It classifies 1 tag per 1 word while considering the tags of the other words.) We will assume that tags only depend on their immediate neighbors,</p> $\text{score}(t, w) = \sum_{n=1}^N \text{score}(\langle t_{n-1}, t_n \rangle, w, n)$ $\text{score}(\langle t_{n-1}, t_n \rangle, w, n) = \text{trans}(t_{n-1}, t_n) + \text{em}(w_n, t_n),$ <p>We can "neuralize" a CRF by making the transition and emission scores the learned weights of the model.</p> <p>Forward algorithm: Start at BOT and go word by word, tag by tag. (Viterbi: for every node, compute the argmax and keep track). Then,</p> $\alpha_{n,t_n} \leftarrow \bigoplus_{t_{n-1} \in \mathcal{T}} \exp \text{score}(\langle t_{n-1}, t_n \rangle, w, n) \otimes \alpha_{n-1, t_{n-1}}.$ <p>Return: $\alpha_{N, \text{EOS}}$. Runtime: $\mathcal{O}(N \mathcal{T} ^2)$. Intuition: sum over all paths in POS graph.</p> <p>Forward-backward algorithm is an instantiation of backpropagation, because it is analogous to the forward and backward pass. It is a sum over products on the paths.</p> <p>Dijkstra's decoding: Paths can only get worse $\forall a, b \in \mathbb{K} : a \preceq (b \otimes a)$, o-closedness). Thus, the first N-length tagging popped from the priority queue must be the best. Runtime: $\mathcal{O}(N \mathcal{T} ^2 + N \mathcal{T} \log(N \mathcal{T}))$.</p>

Finite-state automata
<p>Definition WFST: input alphabet Σ, output alphabet Ω, finite states Q, initial states $I \subseteq Q$, final states $F \subseteq Q$, initial state weights $\lambda : I \rightarrow \mathbb{K}$, final state weights $\rho : F \rightarrow \mathbb{K}$, transitions $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Omega \cup \{\epsilon\}) \times \mathbb{K} \times Q$.</p> <p>Pathsum (R_{ik} is inner pathsum i to k):</p> $Z(\mathcal{T}) = \bigoplus_{i,k \in Q} \lambda(q_i) \otimes R_{ik} \otimes \rho(q_k).$ <p>Infinite paths because of cycles: need Lehmann's algorithm to compute semiring-sum over inner paths between all nodes under a closed semiring. Recurrence:</p> $R_{ik}^{(j)} \leftarrow R_{ik}^{(j-1)} \oplus R_{ij}^{(j-1)} \otimes \left(R_{jj}^{(j-1)} \right)^* \otimes R_{jk}^{(j-1)}.$ <p>Return $I \oplus R^{(Q)}$. Runtime: $\mathcal{O}(Q ^3)$. Intuition: $R_{ik}^{(j)}$ is the semiring-sum over all paths between i and k through $\{1, \dots, j\}$.</p> <p>Floyd-Warshall (min path in graph with no negative cycles):</p> $R_{ik}^{(j)} \leftarrow \min \left\{ R_{ik}^{(j-1)}, R_{ij}^{(j-1)} + R_{jk}^{(j-1)} \right\}.$ <p>Return: $\min \{ I, R^{(Q)} \}$.</p> <p>WFST composition $\mathcal{T} = \mathcal{T}_1 \circ \mathcal{T}_2 : \Sigma^* \xrightarrow{\mathcal{T}_1} \Omega^* \xrightarrow{\mathcal{T}_2} \Xi^*$ with</p>

weights such that

$$T(x, y) = \bigoplus_{z \in \Omega^*} T_1(x, z) \otimes T_2(z, y).$$

Transliteration

Transliteration is the mapping of strings in one character set to strings in another. Formally, we want to compute $p(y | x)$ for all $x \in \Sigma^*$ to $y \in \Omega^*$. We use WFSTs to specify it as a globally normalized model. We need three transducers:

1. T_x is the transducer that maps x to x ;
2. T_θ is the transducer that maps any string in Σ^* to any string in Ω^* ;
3. T_y is the transducer that maps y to y .

Compose $T_x \circ T_\theta$ to compute $Z_\theta(x)$ with Lehmann's. Compose $T_x \circ T_\theta \circ T_y$ to compute $\text{score}_\theta(y, x)$.

Constituency parsing

Definition CFG: Finite non-terminals \mathcal{N} , start symbol S , terminals Σ , production rules \mathcal{R} of form $N \rightarrow \alpha$.

PCFG: Locally normalized probability distribution $p: \mathcal{R} \rightarrow [0, 1]$.

WCFG: Globally normalized with $\text{score}(t, w) = \sum_{r \in t} \text{score}(r)$, where t is a multiset of rules.

Problem: Cycles. **Solution:** A grammar is in CNF if all rules look like $N_1 \rightarrow N_2 N_3$ or $N \rightarrow a$.

Cocke-Kasami-Younger algorithm: Draw chart. Look at iteratively larger spans, finding children that fill up the entire span. For each span, make sure to check all smaller spans that make up this span, because there might be sneaky rules in the grammar.

$$C_{i,k,X} \leftarrow \bigoplus_{X \rightarrow YZ} \text{exp score}(X \rightarrow YZ) \otimes C_{i,j,Y} \otimes C_{j,k,Z}.$$

Return: $C_{1,N+1,S}$. **Runtime:** $\mathcal{O}(N^3 |\mathcal{R}|)$.

Dependency parsing

There are $(N - 1)^{N-2}$ spanning trees with the single root constraint, thus we need an efficient algorithm to compute $Z(w)$. Scoring function:

$$\text{score}(t, w) = \underbrace{\text{score}(r, w)}_{\rho_r} + \sum_{(i \rightarrow j) \in t} \underbrace{\text{score}(i, j, w)}_{A_{ij}}$$

Problem with Kirchhoff's MTT: Undirected graph.

Problem with Tutte's MTT: No single-root constraint.

Solution Koo MTT: To compute $Z(w)$ with single-root constraint, we construct the Laplacian matrix (root scores on first row, sum over columns) (Kirchhoff removes i -th row and column),

$$L_{ij} = \begin{cases} \rho_j & i = 1 \text{ (KH and Tutte is without this condition)} \\ -A_{ij} & i \neq j \\ \sum_{k \neq i} A_{kj} & \text{otherwise.} \end{cases}$$

$Z(w) = \det(L)$. **Runtime:** $\mathcal{O}(N^3)$.

Chu-Liu-Edmonds algorithm: (1) Compute **greedy graph** by

taking all max incoming edges for each node. (2) If cycle, treat as single node and **contract** it by reweighting incoming edges, such that they break the cycle (add weight of all other edges in cycle node). (3) If there are multiple edges emanating from the root, remove (permanently) the edge with the **lowest swap loss** (weight of current edge - next best incoming edge of child). (4) **Repeat** until there are no more cycles (cycle nodes may contain smaller cycle nodes). (5) **Expand** the contractions by deleting the edge from the cycle that is pointed at the node with an edge in the greedy graph that contains this cycle node. (6) **Repeat** until there are no more cycle nodes, making sure to remove cycles if they come up. **Runtime:** $\mathcal{O}(N^2)$.

Semantic parsing

Lambda calculus: variables x, y, z , lambda operator $(\lambda x.f(x))$, application of two lambda terms $(M N)$.

β -reduction: $((\lambda x.M) N) \rightarrow M[x := N]$.

α -conversion: $\lambda x.M[x] \rightarrow \lambda y.M[y]$.

If free variable becomes bound after β -reduction, first do α -conversion.

Combinatory categorial grammar rules:

$$\begin{aligned} X/Y \quad Y &\implies X && (>) \\ Y \quad X \setminus Y &\implies X && (<) \\ X/Y \quad Y/Z &\implies X/Z && (\mathbf{B}_>) \\ Y/Z \quad X/Y &\implies X/Z && (\mathbf{B}_<) \\ X &\implies T/(T \setminus X) && (\mathbf{T}_>) \\ X &\implies T \setminus (T/X) && (\mathbf{T}_<). \end{aligned}$$

CCGs are mildly context-sensitive (because there are infinite non-terminals, any recursive combination of atomic non-terminals and $/, \setminus$), but using only $>$ and $<$ is a context-free grammar. CCGs allow for coordination and cross-serial dependencies.

If **lambda calculus with CCG**, first derive the CCG parse tree. Then, combine the λ expressions. M on the left and N on the right, then $>$ combines $(M N)$ and $<$ combines $(N M)$. Iteratively simplify the λ expressions of greater spans using the smaller spans.

Linear-indexed grammar: Mildly context-sensitive grammar is a CFG where the non-terminals have stacks that contain indices from a finite set I . Rules have the following form (RHS can be ϵ):

$$\begin{aligned} N[\sigma] &\rightarrow \alpha M[\sigma] \beta \\ N[\sigma] &\rightarrow \alpha M[f\sigma] \beta && (\text{push}) \\ N[f\sigma] &\rightarrow \alpha M[\sigma] \beta && (\text{pop}). \end{aligned}$$

Transformers

Attention: Query, key, and values where the query and keys decide how much is "attended" to the values (soft-lookup in hashmap). **Self-attention:** Learn the queries $W_Q \in \mathbb{R}^{d \times d}$, keys $W_K \in \mathbb{R}^{d \times d}$, and values $W_V \in \mathbb{R}^{d \times d}$ from a single input

$X \in \mathbb{R}^{d \times n}$:

$$\text{SELFATT}(X) = \text{SOFTMAX}\left(\left(W_Q^\top X\right)^\top \left(W_K^\top X\right) / \sqrt{d_q}\right) \left(W_V^\top X\right)^\top.$$

Runtime: $\mathcal{O}(nd^2 + dn^2)$. **Problem:** Permutation equivariant.

Solution: Add positional encoding: $P_{pi} = \sin(p/10000^{i/d})$ if i is even, and $P_{pi} = \cos(p/10000^{i/d})$ if i is odd.

1. **Encoder:** $\oplus P \rightarrow \text{MHSA} \rightarrow \oplus \rightarrow \text{LN} \rightarrow \text{MLPs} \rightarrow \oplus \rightarrow \text{LN}$;
2. **Decoder:** Add linear projection and softmax.

Residual layers help with gradient problems. **Layer norm** helps with internal covariate shifts. **MHSA:** Concatenate heads and project to \mathbb{R}^d .

Sequence-to-sequence: Encode input sequence with n sequential encoders. Then, input this to the decoders together with a representation of previous output tokens (these output a prob. dist. over next tokens).

Problem: Greedily picking the most probable next token does not necessarily result in the globally maximum output, and we cannot go over every single possible output, which is infinite.

Solution: **Beam search** (keep track of at most k paths at once), **nucleus sampling** (only consider top $p\%$ probability mass).

Axes of modelling

Bias-variance tradeoff. High bias: Low model complexity, underfitting, misses correlations in training data. High variance: High model complexity, overfitting, fits too well on the training data (captures noise).

Regularization: Tradeoff between bias and variance, e.g., $\ell(\theta) + \lambda \|\theta\|_2^2$.

Cross-entropy \equiv NLL. Minimize NLL \equiv MLE. $\hat{\theta} = \text{argmin}_{\theta \in \Theta} -\log \prod_{(x,y) \in \mathcal{D}} p_\theta(y | x)$. **MLE problems:** only suitable for probabilistic models, easily overfit if small training dataset.

Precision = $\frac{\text{true positive}}{\text{predicted positive}}$, **Recall** = $\frac{\text{true positive}}{\text{true positive} + \text{false negative}}$, **F1** = $\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$.

Locally normalized: + Efficient to train, - Label bias.

Globally normalized: + Scores at each step can have differing "importance". - Requires computation of normalizer, which usually requires independence assumptions.

Tips

When **deriving the gradient of a model**, think of which paths exist in the computational graph between the two nodes. Sum over the paths and product within the paths.

If it is possible to **reuse terms during backpropagation**, adjust the time complexity to do that.

Linear algebra multiplication complexities:

1. Vector-vector ($\mathbb{R}^{1 \times d}, \mathbb{R}^{d \times 1}$): $\mathcal{O}(d)$;
2. Matrix-vector ($\mathbb{R}^{n \times m}, \mathbb{R}^{m \times 1}$): $\mathcal{O}(nm)$;
3. Matrix-matrix ($\mathbb{R}^{n \times m}, \mathbb{R}^{m \times \ell}$): $\mathcal{O}(nml)$.

Activation functions:

1. $\sigma(x) = \frac{1}{1 + \exp(-x)}$ ($\mathbb{R} \rightarrow [0, 1]$),
 $\sigma'(x) = \sigma(x)(1 - \sigma(x))$;

2. $\text{ReLU}(x) = \max\{0, x\}$ ($\mathbb{R} \rightarrow \mathbb{R}_{\geq 0}$),

$$\text{ReLU}'(x) = \mathbb{1}\{x > 0\};$$

3. $\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$ ($\mathbb{R} \rightarrow [-1, 1]$),

$$\tanh'(x) = 1 - \tanh^2(x).$$