Backpropagation

Linear-time dynamic program for computing derivatives.

1. Write down composite function as a labeled, acyclic, directed hypergraph with intermediate variables as nodes and hyperedges labeled with primitive functions;
2. Given input, perform forward propagation;
3. Run backpropagation on the graph using forward values. $\frac{\partial y_i}{\partial x_j} = \sum_{p \in P(j,i)} \prod_{(k \to \ell) \in p} \frac{\partial z_\ell}{\partial z_k}$.

$\sin'(x) = \cos(x)$, $\cos'(x) = -\sin(x)$, $y^x = \exp(x \log(y))$, $\log'(x) = \frac{1}{x}$, $\exp'(x) = \exp(x)$.

Log-linear modelling

$$\text{score}(y, x) = \boldsymbol{\theta}^\top \boldsymbol{f}(x, y).$$

Result of NLL gradient equal to 0:

$$\sum_{i=1}^{n} \boldsymbol{f}(x_i, y_i) = \sum_{i=1}^{n} \mathbb{E}_{y|x_i,\boldsymbol{\theta}}[\boldsymbol{f}(x_i, y)].$$

Hessian ($\boldsymbol{H}_{\boldsymbol{\theta}} = \nabla_{\boldsymbol{\theta}^\top} \nabla_{\boldsymbol{\theta}}$):

$$\boldsymbol{H}_{\boldsymbol{\theta}} \left(\sum_{i=1}^{n} -\log p(y_i \mid x_i) \right) = \sum_{i=1}^{n} \text{Cov}_{y|x_i,\boldsymbol{\theta}}[\boldsymbol{f}(x_i, y)].$$

To derive this, first derive the following:

$$\nabla_{\boldsymbol{\theta}^\top} p(y_i \mid x_i) = p(y_i \mid x_i) \Big(\boldsymbol{f}(x_i, y_i)^\top - \mathbb{E}_{y'|x_i} \Big[\boldsymbol{f}(x_i, y')^\top \Big] \Big).$$

Softmax: $\textsc{Softmax}(\boldsymbol{h})_y = \frac{\exp(h_y/T)}{\sum_{y' \in \mathcal{Y}} \exp(h_{y'}/T)}$.

$T \to 0$: argmax (structured perceptron). $T \to \infty$: uniform.

Exponential family: $p(x \mid \boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} h(x) \exp(\boldsymbol{\theta}^\top \boldsymbol{\phi}(x))$.

$Z(\boldsymbol{\theta})$: partition function. $h(x)$: support. $\boldsymbol{\theta}$: parameters. $\boldsymbol{\phi}(x)$ feature function.

Advantage: conjugate priors, compress all data into finite parameters $\boldsymbol{\phi}(x)$ without losing information.

Multi-layer perceptron

Problem with log-linear modelling: Data must be linearly separable. **Solution**: Hack the non-linearity into the feature function. **Problem**: We need to know the decision boundary shape a priori. **Solution**: Learn the non-linear feature function with an MLP:

$$\boldsymbol{h}_k = \sigma_k \Big(\boldsymbol{W}_k^\top \boldsymbol{h}_{k-1} \Big), \qquad \boldsymbol{h}_1 = \sigma_1 \Big(\boldsymbol{W}_1^\top \boldsymbol{e}(x) \Big),$$

Then $\textsc{Softmax}(\boldsymbol{\theta}^\top \boldsymbol{h}_n)$ for prob. dist.

$\boldsymbol{e}(x)$ 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.

Derivative:

$$\frac{\partial \ell}{\partial \boldsymbol{W}_k} = \frac{\partial \ell}{\partial y} \frac{\partial y}{\partial \boldsymbol{h}_n} \left(\prod_{m=k+1}^{n} \sigma_m'(\cdots) \boldsymbol{W}_m \right) \sigma_k'(\cdots) \boldsymbol{h}_{k-1}.$$

Structured prediction

To be able to train a model, we need to compute:

$$p(y \mid x) = \frac{\exp \text{score}(y, x)}{Z(x)}, \quad Z(x) = \sum_{y' \in \mathcal{Y}} \exp \text{score}(y', x).$$

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.

Language modelling

$$p(\boldsymbol{y}) = p(\textsc{eos} \mid \boldsymbol{y}) \cdot \prod_{i=1}^{N} p(y_i \mid \boldsymbol{y}_{<i})$$

$$p(y_i \mid \boldsymbol{y}_{<i}) = \frac{1}{Z(\boldsymbol{y}_{<i})} \exp \text{score}(\boldsymbol{y}_{<i}, y_i).$$

Problem: Model is non-tight (total prob. $\neq 1$) if we have a history that never ends in EOS. **Solution**: Force $p(\textsc{eos} \mid \boldsymbol{y}_{<i}) > \xi > 0$ for every possible history $\boldsymbol{y}_{<i}$.

Problem: infinitely many histories. **n-gram**: $p(y_i \mid \boldsymbol{y}_{<i}) = p(y_i \mid y_{i-n+1}, \ldots, y_{i-1})$ (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**: $\boldsymbol{h}_i = \sigma(\boldsymbol{W}_h \boldsymbol{h}_{i-1} + \boldsymbol{W}_x \boldsymbol{e}(y_{i-1}) + \boldsymbol{b})$ (encode entire history using hidden states).

Derivative:

$$\frac{\partial \ell_i}{\partial \boldsymbol{W}_h} = \frac{\partial \ell_i}{\partial y_i} \frac{\partial y_i}{\partial \boldsymbol{h}_N} \sum_{j=1}^{i} \left(\prod_{m=j+1}^{i} \sigma'(\cdots) \boldsymbol{W}_h \right) \sigma'(\cdots) \boldsymbol{h}_{i-1}.$$

Problem: Vanishing gradient. **Solution**: LSTM/GRU.

Semirings

Definition **monoid** $\langle \mathbb{K}, \odot, \boldsymbol{e} \rangle$:

1. $\odot$ is associative: $(x \odot y) \odot z = x \odot (y \odot z)$.
2. $\boldsymbol{e} \in \mathbb{K}$ is the identity element: $x \odot \boldsymbol{e} = x$.

Definition **semiring** $\langle \mathbb{K}, \oplus, \otimes, \boldsymbol{0}, \boldsymbol{1} \rangle$:

1. $\langle \mathbb{K}, \oplus, \boldsymbol{0} \rangle$ is a commutative monoid: $x \oplus y = y \oplus x$.
2. $\langle \mathbb{K}, \otimes, \boldsymbol{1} \rangle$ is a monoid.
3. $\otimes$ distributes over $\oplus$: $(x \oplus y) \otimes z = (x \otimes z) \oplus (y \otimes z)$ and $z \otimes (x \oplus y) = (z \otimes x) \oplus (z \otimes y)$.
4. $\boldsymbol{0}$ is an annihilator of $\otimes$: $\boldsymbol{0} \otimes x = \boldsymbol{0}$ and $x \otimes \boldsymbol{0} = \boldsymbol{0}$.

Closed: $x^* = \bigoplus_{n=0}^{\infty} x^{\otimes n}$. The Kleene star must be in $\mathbb{K}$ and obey the following two axioms: $x^* = \boldsymbol{1} \oplus x \otimes x^*$ and $x^* = \boldsymbol{1} \oplus x^* \otimes x$.

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**: $\langle \mathbb{N}, +, \times, 0, 1 \rangle$.

Part-of-speech tagging

Input: $\boldsymbol{w} \in \Sigma^N$. **Output**: $\boldsymbol{t} \in \mathcal{T}^N$.

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,

$$\text{score}(\boldsymbol{t}, \boldsymbol{w}) = \sum_{n=1}^{N} \text{score}(\langle t_{n-1}, t_n \rangle, \boldsymbol{w}, n)$$

$$\text{score}(\langle t_{n-1}, t_n \rangle, \boldsymbol{w}, n) = \text{trans}(t_{n-1}, t_n) + \text{em}(w_n, t_n),$$

We can "neuralize" a CRF by making the transition and emission scores the learned weights of the model.

Forward algorithm: Start at BOT and go word by word, tag by tag. (**Viterbi**: for every node, compute the argmax and keep track). Then,

$$\alpha_{n,t_n} \leftarrow \bigoplus_{t_{n-1} \in \mathcal{T}} \exp \text{score}(\langle t_{n-1}, t_n \rangle, \boldsymbol{w}, n) \otimes \alpha_{n-1,t_{n-1}}.$$

Return: $\alpha_{N,\textsc{eot}}$. **Runtime**: $\mathcal{O}(N|\mathcal{T}|^2)$. **Intuition**: sum over all paths in POS graph.

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.

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**: $\mathcal{O}(N|\mathcal{T}|^2 + N|\mathcal{T}| \log(N|\mathcal{T}|))$.

Finite-state automata

Definition **WFST**: input alphabet Σ, output alphabet Ω, finite states Q, initial states $I \subseteq Q$, final states $F \subseteq Q$, initial state weights $\lambda : I \to \mathbb{K}$, final state weights $\rho : F \to \mathbb{K}$, transitions $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Omega \cup \{\epsilon\}) \times \mathbb{K} \times Q$.

Pathsum ($\boldsymbol{R}_{ik}$ is inner pathsum i to k):

$$Z(\mathcal{T}) = \bigoplus_{i,k \in Q} \lambda(q_i) \otimes \boldsymbol{R}_{ik} \otimes \rho(q_k).$$

Infinite paths because of cycles: need **Lehmann's algorithm** to compute semiring-sum over inner paths between all nodes under a closed semiring. Recurrence:

$$\boldsymbol{R}_{ik}^{(j)} \leftarrow \boldsymbol{R}_{ik}^{(j-1)} \oplus \boldsymbol{R}_{ij}^{(j-1)} \otimes \Big(\boldsymbol{R}_{jj}^{(j-1)} \Big)^* \otimes \boldsymbol{R}_{jk}^{(j-1)}.$$

Return $\boldsymbol{I} \oplus \boldsymbol{R}^{(|Q|)}$. **Runtime**: $\mathcal{O}(|Q|^3)$. **Intuition**: $\boldsymbol{R}_{ik}^{(j)}$ is the semiring-sum over all paths between i and k through $\{1, \ldots, j\}$.

Floyd-Warshall (min path in graph with no negative cycles):

$$\boldsymbol{R}_{ik}^{(j)} \leftarrow \min \Big\{ \boldsymbol{R}_{ik}^{(j-1)}, \boldsymbol{R}_{ij}^{(j-1)} + \boldsymbol{R}_{jk}^{(j-1)} \Big\}.$$

Return: $\min \Big\{ \boldsymbol{I}, \boldsymbol{R}^{(|Q|)} \Big\}$.

WFST composition $\mathcal{T} = \mathcal{T}_1 \circ \mathcal{T}_2$: $\Sigma^* \xrightarrow{\mathcal{T}_1} \Omega^* \xrightarrow{\mathcal{T}_2} \Xi^*$ with

weights such that

$$\mathcal{T}(x,y) = \bigoplus_{z \in \Omega^*} \mathcal{T}_1(x,z) \otimes \mathcal{T}_2(z,y).$$

Transliteration

Transliteration is the mapping of strings in one character set to strings in another. Formally, we want to compute $p(y \mid x)$ 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. $\mathcal{T}_x$ is the transducer that maps x to x;
2. $\mathcal{T}_\theta$ is the transducer that maps any string in Σ^* to any string in Ω^*;
3. $\mathcal{T}_y$ is the transducer that maps y to y.

Compose $\mathcal{T}_x \circ \mathcal{T}_\theta$ to compute $Z_\theta(x)$ with Lehmann's. Compose $\mathcal{T}_x \circ \mathcal{T}_\theta \circ \mathcal{T}_y$ 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 $N \to \boldsymbol{\alpha}$.
PCFG: Locally normalized probability distribution $p : \mathcal{R} \to [0,1]$.
WCFG: Globally normalized with $\text{score}(\boldsymbol{t}, \boldsymbol{w}) = \sum_{r \in \boldsymbol{t}} \text{score}(r)$, where $\boldsymbol{t}$ is a multiset of rules.
Problem: Cycles. **Solution**: A grammar is in CNF if all rules look like $N_1 \to N_2\ N_3$ or $N \to a$.
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.

$$\boldsymbol{C}_{i,k,X} \leftarrow \bigoplus_{X \to YZ} \exp \text{score}(X \to YZ) \otimes \boldsymbol{C}_{i,j,Y} \otimes \boldsymbol{C}_{j,k,Z}.$$

Return: $\boldsymbol{C}_{1,N+1,S}$. **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 $Z(\boldsymbol{w})$. Scoring function:

$$\text{score}(\boldsymbol{t}, \boldsymbol{w}) = \underbrace{\text{score}(r, \boldsymbol{w})}_{\boldsymbol{\rho}_r} + \sum_{(i \to j) \in \boldsymbol{t}} \underbrace{\text{score}(i, j, \boldsymbol{w})}_{\boldsymbol{A}_{ij}}.$$

Problem with Kirchhoff's MTT: Undirected graph.
Problem with Tutte's MTT: No single-root constraint.
Solution Koo MTT: To compute $Z(\boldsymbol{w})$ with single-root constraint, we construct the Laplacian matrix (root scores on first row, sum over columns) (Kirchhoff removes i-th row and column),

$$\boldsymbol{L}_{ij} = \begin{cases} \boldsymbol{\rho}_j & i = 1 \quad \text{(KH and Tutte is without this condition)} \\ -\boldsymbol{A}_{ij} & i \neq j \\ \sum_{k \neq i} \boldsymbol{A}_{kj} & \text{otherwise.} \end{cases}$$

$Z(\boldsymbol{w}) = \det(\boldsymbol{L})$. **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: $((\lambda x.M)\ N) \to M[x := N]$.
α-conversion: $\lambda x.M[x] \to \lambda y.M[y]$.
If free variable becomes bound after β-reduction, first do α-conversion.
Combinatory categorial grammar rules:

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

CCGs are mildy context-sensitive (because there are infinite non-terminals, any recursive combination of atomic non-terminals and $/, \backslash$), 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):

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

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 $\boldsymbol{W}_Q \in \mathbb{R}^{d \times d}$, keys $\boldsymbol{W}_K \in \mathbb{R}^{d \times d}$, and values $\boldsymbol{W}_V \in \mathbb{R}^{d \times d}$ from a single input $\boldsymbol{X} \in \mathbb{R}^{d \times n}$:

$$\textsc{SelfAtt}(\boldsymbol{X}) = \textsc{Softmax}\left(\left(\boldsymbol{W}_Q^\top \boldsymbol{X}\right)^\top \left(\boldsymbol{W}_K^\top \boldsymbol{X}\right) / \sqrt{d_q}\right) \left(\boldsymbol{W}_V^\top \boldsymbol{X}\right)^\top.$$

Runtime: $\mathcal{O}(nd^2 + dn^2)$. **Problem**: Permutation equivariant. **Solution**: Add positional encoding: $\boldsymbol{P}_{pi} = \sin(p/10000^{i/d})$ if i is even, and $\boldsymbol{P}_{pi} = \cos(p/10000^{i/d})$ if i is odd.
1. **Encoder**: $\oplus \boldsymbol{P} \to$ MHSA $\to \oplus \to$ LN $\to$ MLPs $\to \oplus \to$ 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., $\ell(\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|_2^2$.
Cross-entropy $\equiv$ NLL. Minimize NLL $\equiv$ MLE. $\hat{\boldsymbol{\theta}} = \text{argmin}_{\boldsymbol{\theta} \in \Theta} -\log \prod_{(x,y) \in \mathcal{D}} p_{\boldsymbol{\theta}}(y \mid x)$. **MLE problems**: only suitable for probabilistic models, easily overfit if small training dataset.
Precision $= \frac{\text{true positive}}{\text{predicted positive}}$, **Recall** $= \frac{\text{true positive}}{\text{true positive+false negative}}$, **F1** $= \frac{\text{precision} \cdot \text{recall}}{\text{precision+recall}}$.
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 ($\mathbb{R}^{n\times m}$, $\mathbb{R}^{m\times \ell}$): $\mathcal{O}(nm\ell)$.

Activation functions:
1. $\sigma(x) = \frac{1}{1+\exp(-x)}$ ($\mathbb{R} \to [0,1]$),
   $\sigma'(x) = \sigma(x)(1-\sigma(x))$;
2. $\mathrm{ReLU}(x) = \max\{0, x\}$ ($\mathbb{R} \to \mathbb{R}_{\geq 0}$),
   $\mathrm{ReLU}'(x) = \mathbb{1}\{x > 0\}$;
3. $\tanh(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$ ($\mathbb{R} \to [-1,1]$),
   $\tanh'(x) = 1 - \tanh^2(x)$.