1 Linear Algebra

Semiring

Monoid —

- Consists of a set $\mathcal{A}$, an operation $*$, and an identity element e, such that:
  - Associativity: $(a*b)*c = a*(b*c)$
  - Identity: $a*e = e*a = a$
- *Commutative monoid*: Additionally commutative: $a*b = b*a$

Semiring —

- $(\mathcal{A}, \oplus, \otimes, \overline{0}, \overline{1})$:
  - $(\mathcal{A}, \oplus, \overline{0})$ is a commutative monoid
  - $(\mathcal{A}, \otimes, \overline{1})$ is a monoid
  - $\otimes$ distributes over $\oplus$: $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c), \quad c \otimes (a \oplus b) = (c \otimes a) \oplus (c \otimes b)$
  - $\overline{0}$ is an annihilator for $\otimes$: $a \otimes \overline{0} = \overline{0}, \quad \overline{0} \otimes a = \overline{0}$
- *Commutative semiring*: $\otimes$ is commutative: $a \otimes b = b \otimes a$
- *Idempotent semiring*: $\oplus$ is idempotent: $a \oplus a = a$
  - For idempotent semirings, $\bigoplus_{k=0}^{K} \boldsymbol{M}^k = (\boldsymbol{I} + \boldsymbol{M})^K$
    Proof:
    - Base case: $(\boldsymbol{I} \oplus \boldsymbol{M})^1 = \boldsymbol{I} \oplus \boldsymbol{M} = \boldsymbol{M}^0 \oplus \boldsymbol{M}^1 = \bigoplus_{k=0}^{1} \boldsymbol{M}^k$
    - Inductive step: $(\boldsymbol{I} \oplus \boldsymbol{M})^{K+1} = \left(\bigoplus_{k=0}^{K} \boldsymbol{M}^k\right) \otimes (\boldsymbol{I} \oplus \boldsymbol{M}) = \bigoplus_{k=0}^{K} (\boldsymbol{M}^k \otimes \boldsymbol{I}) \oplus \bigoplus_{k=0}^{K} (\boldsymbol{M}^k \otimes \boldsymbol{M}) = \bigoplus_{k=0}^{K} \boldsymbol{M}^k \oplus \bigoplus_{k=0}^{K} \boldsymbol{M}^{k+1} = \bigoplus_{k=0}^{K} \boldsymbol{M}^k \oplus \bigoplus_{k=1}^{K+1} \boldsymbol{M}^k$
    - Because of idempotency, the repeated terms $\boldsymbol{M}^k$ can be simplified ($\boldsymbol{M}^k \oplus \boldsymbol{M}^k = \boldsymbol{M}^k$)
    - Then, we have: $(\boldsymbol{I} \oplus \boldsymbol{M})^{K+1} = \bigoplus_{k=0}^{K+1} \boldsymbol{M}^k$
  - We can also show that, $\bigoplus_{k=0}^{K} \boldsymbol{M}^k = \bigoplus_{k=0}^{K} \bigotimes_{n=0}^{\lfloor \log_2 N \rfloor} \boldsymbol{M}^{\alpha_n 2^n}$ if we use binary decomposition on matrix $\boldsymbol{M}$
    Proof:
    - According to binary decomposition: $k = \sum_{n=0}^{\lfloor \log_2 N \rfloor} \alpha_n 2^n$ where $\alpha_n \in \{0, 1\}$ and $\alpha_n = 1$ if 2^n is part of the decomposition, otherwise $\alpha_n = 0$
    - Then, $\boldsymbol{M}^k = \boldsymbol{M}^{\sum_{n=0}^{\lfloor \log_2 N \rfloor} \alpha_n 2^n} = \bigotimes_{n=0}^{\lfloor \log_2 N \rfloor} \boldsymbol{M}^{\alpha_n 2^n}$
- *Closed semiring*: Additional unary operation: *Kleene star* $*$ (*asteration*):
  - $x^* = \bigoplus_{n=0}^{\infty} x^{\otimes n} = \overline{1} \oplus x \oplus x^{\otimes 2} \oplus x^{\otimes 3} \oplus \cdots$
  - Properties: $x^* = \overline{1} \oplus x \otimes x^* = \overline{1} \oplus x^* \otimes x$
    Proof:
    - $\overline{1} \oplus x \otimes x^* = \overline{1} \oplus x \otimes \left(\overline{1} \oplus x \oplus x^{\otimes 2} \oplus x^{\otimes 3} \oplus \cdots\right)$
    - Using the distributive property of $\oplus$ over $\otimes$, we distribute $x \otimes$ across the terms and get: $\overline{1} \oplus x \otimes x^* = \overline{1} \oplus x \oplus (x \otimes x) \oplus (x \otimes x^{\otimes 2}) \oplus (x \otimes x^{\otimes 3}) \oplus \cdots = \overline{1} \oplus x \oplus x^{\otimes 2} \oplus x^{\otimes 3} \oplus \cdots$
  - E.g.:
    - For the log-sum-exp semiring, the Kleene star is the geometric series: $x^* = \log(\sum_{n=0}^{\infty} e^{n \times x}) = \log(\frac{1}{1-e^x})$ for $x < 0$
    - For the first part of the expectation semiring, the Kleene star is the geometric series: $x^* = \sum_{n=0}^{\infty} x^n = \frac{1}{1-x}$ for $x \in (0, 1)$
      Proof: $x^* = \frac{1}{1-x} = 1 + \frac{1}{1-x} - 1 = 1 + \frac{1-1+x}{1-x} = 1 + \frac{x}{1-x} = 1 + x\frac{1}{1-x} = 1 + xx^*$
  - As an alternative to Lehmann's algorithm, we can approximate Kleene star: $\sum_{k=0}^{K} \boldsymbol{M}^k \approx \boldsymbol{M}^*$ as $K \to \infty$ if $\rho(\boldsymbol{M}) < 1$ resp. $\sigma_{\max}(\boldsymbol{M}) = \|\boldsymbol{M}\|^2 < 1$, since then $\boldsymbol{M}^k \to 0$ as $k \to \infty$
  - *Truncation error* of this approximation: $\|\boldsymbol{M}^* - \sum_{k=0}^{K} \boldsymbol{M}^k\| \leq \frac{\sigma_{\max}(\boldsymbol{M})^{K+1}}{1-\sigma_{\max}(\boldsymbol{M})}$
    Proof:
    - $\boldsymbol{M}^* - \sum_{k=0}^{K} \boldsymbol{M}^k = \sum_{k=K+1}^{\infty} \boldsymbol{M}^k = \boldsymbol{M}^{K+1} \sum_{k=K+1}^{\infty} \boldsymbol{M}^{k-(K+1)} = \boldsymbol{M}^{K+1} \sum_{m=0}^{\infty} \boldsymbol{M}^m = \boldsymbol{M}^{K+1} \boldsymbol{M}^*$
    - Then: $\|\boldsymbol{M}^* - \sum_{k=0}^{K} \boldsymbol{M}^k\| = \|\boldsymbol{M}^{K+1} \boldsymbol{M}^*\|$
    - Using the Cauchy-Schwarz inequality: $\|\boldsymbol{M}^{K+1} \boldsymbol{M}^*\| < \|\boldsymbol{M}^{K+1}\| \, \|\boldsymbol{M}^*\|$
    - For $\|\boldsymbol{M}^{K+1}\|$: $\|\boldsymbol{M}^{K+1}\| \leq \|\boldsymbol{M}\|^{K+1} = \sigma_{\max}(\boldsymbol{M})^{K+1}$
    - For $\|\boldsymbol{M}^*\|$: $\sum_{n=0}^{\infty} \|\boldsymbol{M}^n\| \leq \sum_{n=0}^{\infty} \|\boldsymbol{M}\|^n = \sum_{n=0}^{\infty} \sigma_{\max}(\boldsymbol{M})^n = \frac{1}{1-\sigma_{\max}(\boldsymbol{M})}$ where the second-to-last term is a geometric series
    - Then: $\|\boldsymbol{M}^* - \sum_{k=0}^{K} \boldsymbol{M}^k\| \leq \frac{\sigma_{\max}(\boldsymbol{M})^{K+1}}{1-\sigma_{\max}(\boldsymbol{M})}$
  - Good approximation, especially if $\sigma_{\max} \ll 1$, since then the error becomes very small
  - Runtime complexity exponential in K
- $\overline{0}$-*closed semiring*: $\overline{1} \oplus a = \overline{1}$
  - Properties:
    - $x^* = \bigoplus_{n=0}^{N-1} x^{\otimes n}$ since cycles in a path of length $\geq N$ do not contribute
    - Idempotent
      Proof: $a \oplus a = a \otimes (\overline{1} + \overline{1}) = a \otimes \overline{1} = a$ where the second last step follows due to defining property of $\overline{0}$-closed semiring
  - E.g.: Tropical and arctic semiring

Common semirings:

2 Probability and Statistics

Probability Distributions

Normal distribution — $\mathcal{X} \sim \mathcal{N}(\mu, \sigma^2)$

For univariate, PDF: $\frac{1}{\sqrt{2\pi}\sigma} exp(\frac{-(x-\mu)^2}{2\sigma^2})$

For multivariate, PDF: $\frac{1}{2\pi\sigma^{n/2}} \frac{1}{|\Sigma|^{1/2}} exp(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\top \Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu}))$

Standard normal distribution — Normal distribution, standardized via z-score $z = \frac{x-\mu}{\sigma}$, which results in $\mu = 0$ and $\sigma = 1$

Bernoulli distribution — Trial with success (probability p) or failure (probability $1-p$)

- $\mathcal{X} \sim \text{Bernoulli}(p)$
- PDF: $p(x)p^x(1-p)^x$
- Mean: $\mathbb{E}(x) = p$
- Variance: $\mathbb{V}(x) = p(1-p)$

Binomial distribution — n independent Bernoulli trials with k successes

- $\mathcal{X} \sim \text{Bin}(n, p)$
- PDF: $\binom{n}{k} p^k (1-p)^{n-k}$
- Mean: $\mathbb{E}(x) = np$
- Variance: $\mathbb{V}(x) = np(1-p)$

Poisson distribution —

- $\mathcal{X} \sim \text{Pois}(\lambda)$
- PDF: $e^{-\lambda} \frac{\lambda^x}{x}$
- Mean: $\mathbb{E}(x) = \lambda$
- Variance: $\mathbb{V}(x) = \lambda$

Beta distribution —

- X takes values $\in [0, 1]$
- Represents the probability of a Bernoulli process after observing $\alpha - 1$ successes and $\beta - 1$ failures
- $\mathcal{X} \sim \text{Beta}(\alpha, \beta)$ where $\alpha, \beta > 0$
- PDF: $\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)+\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}$ where $\Gamma(\alpha) = \int_0^\infty u^{\alpha-1} e^{-u} du$
- Mean: $\mathbb{E}(x) = \frac{\alpha}{\alpha+\beta}$
- Variance: $\mathbb{V}(x) = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$

Exponential families — Family of probability distributions (incl. Gaussian, Poisson, Bernoulli, Categorical, Gamma, Beta) of the form: $p(x \mid \theta) = \frac{1}{Z(\theta)} \boldsymbol{h}(x) \exp(\theta \cdot \boldsymbol{\Phi}(x))$ where

- $Z(\theta)$ is a *partition function* which normalizes
- $\boldsymbol{h}(x)$ is the *support*
- θ are the *parameters*
- $\boldsymbol{\Phi}(x)$ are the *sufficient statistics*

Properties:

- Finite sufficient statistics
- *Conjugate priors*:
  - Prior distribution p for a likelihood distribution q such that the posterior distribution induced by p and q is of the same form as p
  - This helps to conduct Bayesian machine learning: Posterior parameters can be updated using prior parameters and do not have to be re-derived
  - E.g.: Bernoulli distribution with Beta prior:
    - Assume prior $P(\theta) \sim \text{Beta}(\alpha, \beta)$
    - Likelihood $P(X \mid \theta) = \theta^k (1-\theta)^{n-k}$, where $n = \text{trials}$ and $k = \text{successes}$
    - Posterior $P(\theta \mid X) \propto P(X \mid \theta) P(\theta) \propto \theta^k (1-\theta)^{n-k} \theta^{\alpha-1} (1-\theta)^{\beta-1} \propto \theta^{k+\alpha-1} (1-\theta)^{n-k+\beta-1}$
    - Due to conjugate prior, posterior $\sim \text{Beta}(\alpha', \beta')$, with $\alpha' = k + \alpha$ and $\beta' = n - k + \beta$

Derivation from maximum entropy:

- *Maximum entropy principle*: When estimating a probability distribution over some constraints (e.g. known $\mathbb{E}$), we should choose the distribution that has the highest entropy
- *Entropy* in discrete case: $H(x) = -\sum_{x \in \mathcal{X}} p(x) \log p(x)$
- Assume we know $\mathbb{E}(\boldsymbol{\Phi}(x)) = F$
- Using the maximum entropy principle, we want to find $J(p) = -\sum_{x \in \mathcal{X}} p(x) \log p(x)$ subject to
  - Non-negativity constraint: $p(x) \geq 0 \quad \forall x$
  - Sum-to-one constraint: $\sum_{x \in \mathcal{X}} p(x) = 1$
  - User-defined constraint: $\mathbb{E}(\boldsymbol{\Phi}(x)) = \sum_{x \in \mathcal{X}} p(x) \boldsymbol{\Phi}(x) = F$
- This is a constrained optimization problem with the objective function: $J(p, \lambda) = -\sum_{x \in \mathcal{X}} p(x) \log p(x) - \lambda_0 (1 - \sum_{x \in \mathcal{X}} p(x)) + \sum_k \lambda_k (F_k - \sum_{x \in \mathcal{X}} p(x) \Phi_k(x))$
- Then, the gradient is given by: $\frac{\partial J(p,\lambda)}{\partial p(x)} = -1 - \log(p(x)) - \lambda_0 - \sum_k \lambda_k \Phi_k(x) = 0$
- $\Rightarrow \log(p(x)) = -1 - \lambda_0 - \sum_k \lambda_k \Phi_k(x) = \log(\frac{1}{Z} \exp(-\sum_k \lambda_k \Phi_k(x)))$ where $Z = \exp(1 + \lambda_0)$
- $\Rightarrow p(x) = \frac{1}{Z} \exp(-\sum_k \lambda_k \Phi_k(x))$
- $\Rightarrow \sum_{x \in \mathcal{X}} p(x) = \sum_{x \in \mathcal{X}} \frac{1}{Z} \exp(-\sum_k \lambda_k \Phi_k(x)) = 1$ by sum-to-one constraint
- $\Rightarrow Z = \sum_{x \in \mathcal{X}} \exp(-\sum_k \lambda_k \Phi_k(x))$
- If we plug this Z back into $p(x)$, we get: $p(x) = \frac{\exp(-\sum_k \lambda_k \Phi_k(x))}{\sum_{x \in \mathcal{X}} \exp(-\sum_k \lambda_k \Phi_k(x))} = \frac{1}{\sum_{x \in \mathcal{X}} \exp(-\boldsymbol{\lambda}^\top \boldsymbol{\Phi}(x))} \exp(-\boldsymbol{\lambda}^\top \boldsymbol{\Phi}(x))$
- Then, we see that this is in the exponential family

Proof that Gaussian is an exponential distribution:

- Gaussian PDF: $f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$
- $= \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2 - 2\mu x + \mu^2}{2\sigma^2}}$
- $= \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2} + \frac{2\mu x}{2\sigma^2} - \frac{\mu^2}{2\sigma^2}}$
- $= \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} e^{\frac{\mu x}{\sigma^2}} e^{-\frac{\mu^2}{2\sigma^2}}$
- $= \frac{e^{-\frac{\mu^2}{2\sigma^2}}}{\sqrt{2\pi\sigma^2}} \times 1 \times e^{\frac{\mu x}{\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$

- $= \frac{e^{-\frac{\mu^2}{2\sigma^2}}}{\sqrt{2\pi\sigma^2}} \times 1 \times e^{\frac{\mu}{\sigma^2}x - \frac{1}{2\sigma^2}x^2}$
- $= \frac{e^{-\frac{\mu^2}{2\sigma^2}}}{\sqrt{2\pi\sigma^2}} \times 1 \times e^{[\frac{\mu}{\sigma^2}, \frac{-1}{2\sigma^2}] \cdot [x, x^2]}$
- Thus, parameters of exponential family given by:
  - $\frac{1}{Z(\theta)} = \frac{e^{-\frac{\mu^2}{2\sigma^2}}}{\sqrt{2\pi\sigma^2}} \Rightarrow Z(\theta) = \sqrt{2\pi\sigma^2} \times \frac{1}{e^{-\frac{\mu^2}{2\sigma^2}}} = \sqrt{2\pi\sigma^2} e^{\frac{\mu^2}{2\sigma^2}}$
  - $\boldsymbol{h}(x) = 1$
  - $\boldsymbol{\theta} = \begin{bmatrix} \mu/\sigma^2 \\ -1/2\sigma^2 \end{bmatrix}$
  - $\boldsymbol{\Phi}(x) = \begin{bmatrix} x \\ x^2 \end{bmatrix}$

Proof that Poisson is an exponential distribution:

- Poisson PDF: $f(x \mid \lambda) = \frac{\lambda^x e^{-\lambda}}{x!}$
- $= \frac{1}{x!} e^{-\lambda} \lambda^x$
- $= \frac{1}{e^\lambda} \frac{1}{x!} e^{log(\lambda^x)}$
- Thus, parameters of exponential family given by:
  - $\frac{1}{Z(\theta)} = \frac{1}{e^\lambda} \Rightarrow Z(\theta) = e^\lambda$
  - $\boldsymbol{h}(x) = \frac{1}{x!}$
  - $\boldsymbol{\theta} = [log(\lambda)]$
  - $\boldsymbol{\Phi}(x) = [x]$

Proof that Beta is an exponential distribution:

- Beta PDF: $f(x \mid \alpha, \beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)}$
- $= \frac{1}{B(\alpha,\beta)} \times 1 \times e^{(\alpha-1)\ln(x) + (\beta-1)\ln(1-x)}$
- $= \frac{1}{B(\alpha,\beta)} \times 1 \times e^{[(\alpha-1),(\beta-1)] \times [\ln(x), \ln(1-x)]}$
- Thus, parameters of exponential family given by:
  - $\frac{1}{Z(\alpha,\beta)} = \frac{1}{B(\alpha,\beta)} \Rightarrow Z(\alpha,\beta) = B(\alpha,\beta) \Rightarrow Z(\boldsymbol{\theta}) = B(\theta_1 + 1, \theta_2 + 1)$
  - $\boldsymbol{h}(x) = 1$
  - $\boldsymbol{\theta} = \begin{bmatrix} (\alpha - 1) \\ (\beta - 1) \end{bmatrix}$
  - $\boldsymbol{\Phi}(x) = \begin{bmatrix} \ln(x) \\ \ln(1-x) \end{bmatrix}$

Proof that Chi Square is an exponential distribution:

- Chi Square PDF: $f(x \mid k) = \frac{x^{k/2-1} e^{-x/2}}{2^{k/2}\Gamma(k/2)}$
- $= \frac{1}{2^{k/2}\Gamma(k/2)} e^{-x/2} e^{(k/2-1)\ln(x)}$
- Thus, parameters of exponential family given by:
  - $\frac{1}{Z(k)} = \frac{1}{2^{k/2}\Gamma(k/2)} \Rightarrow Z(k) = 2^{k/2}\Gamma(k/2) \Rightarrow Z(\boldsymbol{\theta}) = 2^{\theta+1}\Gamma(\boldsymbol{\theta} + 1)$
  - $\boldsymbol{h}(x) = e^{-x/2}$
  - $\boldsymbol{\theta} = [k/2 - 1]$
  - $\boldsymbol{\Phi}(x) = [\ln(x)]$

Proof that Binomial is an exponential distribution:

- Binomial PDF: $f(x \mid n, \pi) = \binom{n}{x} \pi^x (1-\pi)^{n-x}$
- $= \binom{n}{x} e^{x\ln(\pi)} e^{(n-x)\ln(1-\pi)}$
- $= \binom{n}{x} e^{x\ln(\pi) - x\ln(1-\pi) + n\ln(1-\pi)}$
- $= \binom{n}{x} e^{x\ln(\frac{\pi}{1-\pi})} e^{n\ln(1-\pi)}$
- $= e^{n\ln(1-\pi)} \binom{n}{x} e^{\ln(\frac{\pi}{1-\pi})x}$
- Thus, parameters of exponential family given by:
  - $\frac{1}{Z(\pi)} = e^{n\ln(1-\pi)} \Rightarrow Z(\pi) = \frac{1}{e^{n\ln(1-\pi)}} \Rightarrow Z(\boldsymbol{\theta}) = e^{n\ln(1+e^\theta)}$
  - $\boldsymbol{h}(x) = \binom{n}{x}$
  - $\boldsymbol{\theta} = \left[\ln(\frac{\pi}{1-\pi})\right]$
  - $\boldsymbol{\Phi}(x) = [x]$

Hypothesis Testing

Terminology —

- *Hypothesis*:
  - H_0: Accepted null hypothesis, e.g. $p = p_0$, $p_1 - p_2 = p_{0,1} - p_{0,2} = 0$
  - H_A: Alternative hypothesis, e.g. $p \neq p_0$, $p_1 - p_2 \neq p_{0,1} - p_{0,2} \neq 0$
- Errors:
  - *True positive*: Chose H_0, and H_0 obtains
  - *False negative, type I error*: Chose H_A, but H_0 obtains
  - *True negative*: Chose H_A, and H_A obtains
  - *False positive, type II error*: Chose H_0, but H_A obtains
- *Significance level* α:
  - $\alpha \geq p(\text{type I error}) = p(\overline{x} \geq c \mid H_0)$ with equality for continuous variables
  - If α is small, the probability that we are erroneously rejecting H_0 is very small
  - Set by us, typically at 5%
  - If $\mathcal{X} \sim \mathcal{N}(\theta, 1)$ and $H_0 : \theta = 0$: $\alpha = p(\overline{x} \geq c \mid H_0) = p(\sqrt{n}\overline{x} \geq \sqrt{n}c \mid H_0) = p(z_n \geq \sqrt{n}c \mid H_0) = 1 - \Phi(\sqrt{n}c)$ where
    - Φ is the CDF of the normal distribution
    - $z_n \mid H_0 = \frac{\overline{x}-0}{1/\sqrt{n}} = \sqrt{n}\overline{x}$
- *Critical value* z:
  - For two-sided: $z_{\alpha/2}$, $z_{1-\alpha/2}$
  - For one-sided upper tail: $z_{1-\alpha}$
  - For one-sided lower tail: z_α
  - Associated z-score with α
  - Corresponds to critical value c prior to z-score transformation
  - If $\mathcal{X} \sim \mathcal{N}(\theta, 1)$ and $H_0 : \theta = 0$: $\alpha = 1 - \Phi(\sqrt{n}c) \Rightarrow c = \frac{1}{\sqrt{n}}\Phi^{-1}(1-\alpha)$ where Φ is the CDF of the normal distribution
- *P-value* p:
  - For two-sided: $p = P(|z| \geq z_n)$
  - For one-sided upper tail: $p = P(z \geq z_n)$
  - For one-sided lower tail: $p = P(z \leq z_n)$
  - Probability, given H_0 that we observe a value as or more extreme as the observed value z_n
  - Smallest significance level resp. largest confidence level, at which we can reject H_0 given the sample observed
  - If p-value is less than significance level resp. if observed value is more extreme than critical value, reject H_0, because the probability that we are erroneously doing so is very small
- *Confidence level*: $1 - \alpha$, probability, given H_0, that we retain H_0
- *Beta*: $\beta = p(\text{type II error})$
- *Power*:
  - $1 - \beta = p(\text{type II error}) = p(\overline{x} \geq c \mid H_1)$
  - Probability, given H_A, that we reject H_0
  - If $\mathcal{X} \sim \mathcal{N}(\theta, 1)$ and $H_0 : \theta = 0$: $1 - \beta = p(\overline{x} \geq c \mid H_1) = p(\sqrt{n}(\overline{x} - 1) \geq \sqrt{n}(c-1) \mid H_1) = p(z_n \geq \sqrt{n}(c-1) \mid H_1) = p(z_n \geq \sqrt{n}(c-1) \mid H_0) = 1 - \Phi(\sqrt{n}(c-1))$ where
    - Φ is the CDF of the normal distribution
    - $z_n \mid H_1 = \frac{\overline{x}-1}{1/\sqrt{n}} = \sqrt{n}(\overline{x} - 1)$
    - We can switch from $\mid H_1$ to $\mid H_2$ because the two distributions follow the same form, just shifted
- Test types:
  - *Two-sided*: $H_0 : p = p_0, H_A : p \neq p_0$
  - *One-sided upper tail*: $H_0 : p \leq p_0, H_A : p > p_0$
  - *One-sided lower tail*: $H_0 : p \geq p_0, H_A : p < p_0$
- Calculating *test statistic*:
  - $z_n \mid H_0 = \frac{\overline{x}-\mu_0}{\sigma/\sqrt{n}}$

Multiple comparisons problem — Accumulation of false positive rate (α) for K tests, due to independence of tests: $P(|\text{false rejections of } H_0| > 0) = 1 - P(|\text{false rejections of } H_0| = 0) = 1 - (1-\alpha)^K$

Corrections for multiple comparisons problem — *Bonferroni correction*: New significance level set to $\alpha* = \alpha/K$

3 ML Paradigms

MLE estimator

- Maximizes *log likelihood*: $\hat{\theta} = argmax_\theta(L) = argmax_\theta(\prod_{i=1}^n p(y_i|\boldsymbol{x_i}, \theta)) = argmax_\theta(\sum_{i=1}^n log(p(y_i|\boldsymbol{x_i}, \theta)))$
- In the case of classification for 2 classes, *log loss* is equivalent to *binary cross entropy*:
  - Given two classes $y \in \{0, 1\}$:
    - Predicted probability for class 1: $p_1 = \sigma(z) = \frac{1}{1+e^{-z}}$
    - Predicted probability for class 0: $p_0 = 1 - p_1$
  - *Binary cross entropy*: $\text{BCE}(y, p_1) = -[y\log(p_1) + (1-y)\log(1-p_1)]$
    - When $y = 1$: $\text{BCE}(1, p_1) = -\log(p_1)$
    - When $y = 0$: $\text{BCE}(0, p_1) = -\log(1-p_1)$
  - *Log loss*:$\text{LL}(y, p) = -\log(p_y)$
    - When $y = 1$: $\text{LL}(y, p) = -\log(p_1)$
    - When $y = 0$: $\text{LL}(y, p) = -\log(p_0) = -\log(1-p_1)$

NLP-specific model taxonomy

- Locally normalized models: Normalization at each token prediction step within sequence
  - $p(y_i \mid \mathbf{x}) = \frac{\exp(\text{score}(\mathbf{x}, y_i))}{\sum_{y'} \exp(\text{score}(\mathbf{x}, y'))}$
  - $p(\mathbf{y} \mid \mathbf{x}) = \prod_{i=1}^{|\mathbf{y}|} p(y_i \mid \mathbf{x})$
- Globally normalized models: Normalization over entire sequence
  - $p(\mathbf{y} \mid \mathbf{x}) = \frac{\exp(\text{score}(\mathbf{x}, \mathbf{y}))}{\sum_{\mathbf{y}' \in \mathcal{Y}} \exp(\text{score}(\mathbf{x}, \mathbf{y}'))}$

  Structured prediction task: Predict over exponentially (or even infinitely) large set of classes

4 Model Evaluation

Common Loss Measures

Confusion matrix —

- *Precision* = Correctly predicted positive / predicted positive
- *Recall* = Correctly predicted positive / positive in reality
- *Accuracy* = $(TP + TN)/N$
- *F1 score* = Harmonic mean between precision and recall = $(2PR)/(P + R)$

Curve scores —

- *ROC AUC curve*:
  - Plots the TPR (recall, y) against the False Positive Rate FPR (x) at various threshold levels
  - A perfect model has an AUC of 1, a random model has an AUC of 0.5
- *Precision recall curve*:
  - Plots the precision (y) against the recall (x) at various threshold levels
  - A perfect model has precision and recall of 1, the curve for a

random model is a horizontal line at the baseline precision, reflecting the proportion of positive samples in the dataset

Structured prediction and NLP —
- Some outcomes are more similar than others, e.g. if a translation is almost right vs. a translation is completely wrong
- NLP evaluation metrics:
  - Intrinsic evaluation:
    - Log likelihood
    - *Perplexity*:
    - $perplexity(w) = 2^{-\frac{l(w)}{M}}$
    - Lower perplexity = higher likelihood, with perplexity 1 being perfect
    - Cosine similarity of embeddings for words, which are humanly considered similar
    - Word analogy testing (e.g. king vs. queen, man vs. woman)
  - Extrinsic evaluation on downstream task

5 Estimating Common Distributions

Gaussian

Frequentism (MLE) —
- Likelihood (excl. constants): $L = (\frac{1}{\sigma})^n \prod_{i=1}^n exp(-\frac{1}{2\sigma^2}(x^{(i)} - \mu)^2)$
- Log likelihood: $LL = -nlog(\sigma) - \sum_{i=1}^n (\frac{1}{2\sigma^2}(x^{(i)} - \mu)^2)$
- μ_{MLE} is sample mean: $\frac{1}{n}\sum_{i=1}^n x^{(i)}$:
  - Derivative of log likelihood wrt μ:

$$\nabla_\mu LL = \nabla_\mu(-\sum_{i=1}^n(\frac{x^{(i)^2} - 2x^{(i)}\mu + \mu^2}{2\sigma^2})) = \nabla_\mu(-\sum_{i=1}^n(-\frac{x^{(i)}\mu}{\sigma^2} + \frac{\mu^2}{2\sigma^2})) = -\sum_{i=1}^n(-\frac{x^{(i)}}{\sigma^2} + \frac{2\mu}{2\sigma^2}) = \sum_{i=1}^n(\frac{x^{(i)} - \mu}{\sigma^2}) = \sum_{i=1}^n x^{(i)} - n\mu = 0$$

- σ^2_{MLE} is sample variance: $\frac{1}{n}\sum_{i=1}^n (x^{(i)} - \mu)^2$:
  - Derivative of log likelihood wrt σ:

$$\nabla_\sigma LL = -n\nabla_\sigma log(\sigma) - \nabla_\sigma(\sum_{i=1}^n(\frac{(x^{(i)} - \mu)^2}{2\sigma^2})) = \frac{-n}{\sigma} - \nabla_\sigma(\sum_{i=1}^n \frac{1}{2}\sigma^{-2}(x^{(i)} - \mu)^2) = \frac{-n}{\sigma} - (\sum_{i=1}^n -1\sigma^{-3}(x^{(i)} - \mu)^2) = -n + \sum_{i=1}^n(\frac{(x^{(i)} - \mu)^2}{\sigma^2}) = 0$$

Bayesianism —
- Assume Σ is known and $\mu \sim \mathcal{N}(\mu_0, \Sigma_0)$ is the outcome of a random variable
- $p(\mu|X, \mu_0, \Sigma_0) \propto p(X|\mu, \Sigma)p(\mu|\mu_0, \Sigma_0)$
- $p(X|\mu, \Sigma) = \frac{1}{2\pi^{mn/2}}\frac{1}{|\Sigma|^{n/2}}exp(\frac{1}{2}\sum_{i=1}^n (x^{(i)} - \mu)^\top \Sigma^{-1}(x^{(i)} - \mu))$
- $p(\mu|\mu_0, \Sigma_0) = \frac{1}{2\pi^{m/2}}\frac{1}{|\Sigma_0|^{1/2}}exp(\frac{1}{2}\sum_{i=1}^n (\mu - \mu_0)^\top {\Sigma_0}^{-1}(\mu - \mu_0))$
- $p(\mu|X, \mu_0, \Sigma_0) \propto exp(-\frac{1}{2}(\mu^\top {\Sigma_0}^{-1}\mu + n\mu^\top \Sigma^{-1}\mu - 2\mu_0{}^\top {\Sigma_0}^{-1}\mu - 2n\bar{x}^\top \Sigma^{-1}\mu))$ after combining exponents of the prior and likelihood, expanding, absorbing terms unrelated to μ into a constant, and replacing $\sum_{i=1}^n x^{(i)\top}$ by $n\bar{x}^\top$
- We now apply a symmetric matrix property $x^\top Ax + 2x^\top b = (x + A^{-1}b)^\top A(x + A^{-1}b) - b^\top A^{-1}b$, with $\mu = x$, $-({\Sigma_0}^{-1} + n\Sigma^{-1})^{-1} = A^{-1}$ and $(\Sigma^{-1}n\bar{x} + {\Sigma_0}^{-1}\mu_0) = b$
- Through this, we get $p(\mu|X, \mu_0, \Sigma_0) \propto exp(\frac{1}{2}(\mu({\Sigma_0}^{-1} + n\Sigma^{-1})^{-1}(\Sigma^{-1}n\bar{x} + {\Sigma_0}^{-1}\mu_0))^\top({\Sigma_0}^{-1} + n\Sigma^{-1})(\mu - ({\Sigma_0}^{-1} + n\Sigma^{-1})^{-1}(\Sigma^{-1}n\bar{x} + {\Sigma_0}^{-1}\mu_0))) = exp(\frac{1}{2}(\mu - \mu_n)^\top \Sigma_n^{-1}(\mu - \mu_n))$
- Thus, $p(\mu|X, \mu_0, \Sigma_0) \sim \mathcal{N}(\mu_n, \Sigma_n)$ with
  - $\mu_n = ({\Sigma_0}^{-1} + n\Sigma^{-1})^{-1}(\Sigma^{-1}n\bar{x} + {\Sigma_0}^{-1}\mu_0) =$ (if Σ equals 1) $\frac{n\bar{x}\Sigma_0 + \mu_0}{n\Sigma_0 + 1}$
  - $\Sigma_n = ({\Sigma_0}^{-1} + n\Sigma^{-1})^{-1} =$ (if Σ equals 1) $\frac{\Sigma_0}{n\Sigma_0 + 1}$
- Bayesian parameters approximate prior for small n and MLE for large n

6 Linear Regression

Formulation

- $y^{(i)} = \beta \cdot x^{(i)}$ resp. $y = X\beta$ where X contains n rows, each of which represents an instance, and m columns, each of which represents a feature

Optimization

Parameters — Find parameters β

Objective function — Ordinary least squares estimator (OLSE):
- Minimize mean squared error: $LO = (y - X\beta)^\top(y - X\beta)$

Optimization —
- $\nabla_\beta LO = \frac{1}{2}\nabla_\beta((y - X\beta)^\top(y - X\beta)) = \frac{1}{2}\nabla_\beta(\beta^\top X^\top X\beta - 2y^\top X\beta) = X^\top X\beta - X^\top y = X^\top(X\beta - y) = 0$
- $\Rightarrow \beta = (X^\top X)^{-1}X^\top y$

7 Ridge (ℓ_2) Regression

Optimization

Parameters — Find parameters β subject to $\|\beta\|^2 - t \leq 0$

Objective function —
- Minimize mean squared error subject to constraint
- Lagrangian formulation: $LO = (y - X\beta)^\top(y - X\beta) + \lambda(\|\beta\|^2 - t)$

Optimization —
- $\nabla_\beta LO = 0$
- $\Rightarrow \beta = (X^\top X + \lambda I)^{-1}X^\top y$

Characteristics —
- Shrinks certain elements of β to near 0
- Strictly convex with global minimum, unique solution (linearly independent columns), and analytic solution (always invertible)

8 Lasso (ℓ_1) Regression

Optimization

Parameters — Find parameters β subject to $|\beta| - t \leq 0$

Objective function —
- Minimize mean squared error subject to constraint
- Lagrangian formulation: $LO = (y - X\beta)^\top(y - X\beta) + \lambda(|\beta| - t)$

Characteristics —
- Shrinks certain elements of β to 0
- Convex with global minimum, unique or infinite solutions, and numeric solution (not differentiable at $\beta_i = 0$)

9 Polynomial Regression

Formulation

- $y = \Phi\beta$ where Φ is the transformed design matrix with rows $\phi(x^{(i)})^\top$
- If ϕ is a non-linear transformation, we can produce non-linear decision boundaries with a linear regressor

10 Log Linear Models

Formulation

- Family of probability distributions defined as:
  $p(y \mid x, \theta) = \frac{\exp(\theta \cdot f(x,y))}{Z(\theta)} = \frac{\exp(\theta \cdot f(x,y))}{\sum_{y'}\exp(\theta \cdot f(x,y'))}$ where:
  - Z is a *partition function* which normalizes
  - $\theta \cdot f(x, y) = \sum_{i=1}^m \theta_i f_i(x, y)$ is a freely chosen scoring function, that says how good x and y are together, where θ are the *parameters* and $f(x, y)$ is a *feature function*
- If we take the log of $p(y \mid x, \theta)$, we get a linear function:
  $\log(p(y \mid x, \theta)) = \theta \cdot f(x, y) + \text{constant}$

Optimization

Parameters — Find parameters θ

Objective function —
- $\theta = argmin_\theta$ log loss
- Log loss $= -\sum_{i=1}^n \log(p(y^{(i)} \mid x^{(i)}, \theta))$
  $= -\sum_{i=1}^n \log(\frac{\exp(\theta \cdot f(x^{(i)}, y^{(i)}))}{\sum_{y'}\exp(\theta \cdot f(x^{(i)}, y'))})$
  $= -\sum_{i=1}^n \theta \cdot f(x^{(i)}, y^{(i)}) - \log(\sum_{y'}\exp(\theta \cdot f(x^{(i)}, y')))$

Optimization — First-order derivative:
- Gradient of log loss given by:
  $\frac{\partial}{\partial\theta}\log \text{loss} = -\sum_{i=1}^n \frac{\partial}{\partial\theta}(\theta \cdot f(x^{(i)}, y^{(i)}) - \log(\sum_{y'}\exp(\theta \cdot f(x^{(i)}, y'))))$
- $= -\sum_{i=1}^n (f(x^{(i)}, y^{(i)}) - \frac{\sum_{y'}\exp(\theta \cdot f(x^{(i)}, y')) \times f(x^{(i)}, y')}{\sum_{y'}\exp(\theta \cdot f(x^{(i)}, y'))})$
  since: $\frac{\partial}{\partial x}\log e^{\alpha x} = \frac{1}{e^{\alpha x}} \times \frac{\partial e^{\alpha x}}{\partial x} = \frac{1}{e^{\alpha x}} \times e^{\alpha x} \times \frac{\partial \alpha x}{\partial x} = \frac{1}{e^{\alpha x}} \times e^{\alpha x} \times \alpha$
- $= -\sum_{i=1}^n (f(x^{(i)}, y^{(i)}) - \sum_{y'} p(y' \mid x^{(i)}, \theta) \times f(x^{(i)}, y'))$
- $= -\sum_{i=1}^n f(x^{(i)}, y^{(i)}) + \sum_{i=1}^n \sum_{y'} p(y' \mid x^{(i)}, \theta) \times f(x^{(i)}, y')$
- $= -\sum_{i=1}^n f(x^{(i)}, y^{(i)}) + \sum_{i=1}^n \mathbb{E}(f(x^{(i)}, y'))$
- If we set gradient to 0, we have *expectation matching* (observed features = expected features):
  $\sum_{i=1}^n f(x^{(i)}, y^{(i)}) = \sum_{i=1}^n \mathbb{E}(f(x^{(i)}, y'))$

Second-order derivative:
- Hessian of log loss given by:
  $\frac{\partial^2 \log \text{loss}}{\partial\theta\partial\theta^\top} = \frac{\partial}{\partial\theta^\top}$ gradient of log loss
- $= \frac{\partial}{\partial\theta^\top}(-\sum_{i=1}^n (f(x^{(i)}, y^{(i)}) - \frac{\sum_{y'}\exp(\theta \cdot f(x^{(i)}, y')) \times f(x^{(i)}, y')}{\sum_{y'}\exp(\theta \cdot f(x^{(i)}, y'))}))$ where
  - $f(x) = h(x) \times k(x)$
  - $h(x) = \exp(\theta \cdot f(x^{(i)}, y'))$
  - $k(x) = \frac{1}{\sum_{y'}\exp(\theta \cdot f(x^{(i)}, y'))}$
  - $g(x) = f(x^{(i)}, y')$
- $= 0 + \sum_{i=1}^n \sum_{y'} (h'(x)k(x) + k'(x)h(x))g(x) + g'(x)h(x)k(x)$ where
  - $h'(x) = \exp(\theta \cdot f(x^{(i)}, y')) \times f(x^{(i)}, y')$
  - $k'(x) = -\frac{1}{(\sum_{y'}\exp(\theta \cdot f(x^{(i)}, y')))^2} \times \sum_{y'}\exp(\theta \cdot f(x^{(i)}, y')) \times f(x^{(i)}, y')$
    since: $\frac{\partial}{\partial x}\frac{1}{e^{\alpha x}} = \frac{\partial}{\partial x}(e^{\alpha x})^{-1} = -\frac{1}{(e^{\alpha x})^2} \times \frac{\partial e^{\alpha x}}{\partial x} = -\frac{1}{(e^{\alpha x})^2} \times e^{\alpha x} \times \frac{\partial \alpha x}{\partial x} = -\frac{1}{(e^{\alpha x})^2} \times e^{\alpha x} \times \alpha$
  - $g'(x) = 0$
- $= \sum_{i=1}^n \sum_{y'} \exp(\theta \cdot f(x^{(i)}, y')) \times f(x^{(i)}, y') \times \frac{1}{\sum_{y'}\exp(\theta \cdot f(x^{(i)}, y'))} \times f(x^{(i)}, y') - \frac{1}{(\sum_{y'}\exp(\theta \cdot f(x^{(i)}, y')))^2} \times \sum_{y'}\exp(\theta \cdot f(x^{(i)}, y')) \times f(x^{(i)}, y') \times \exp(\theta \cdot f(x^{(i)}, y')) \times f(x^{(i)}, y') + 0$
- $= \sum_{i=1}^n \sum_{y'} p(y' \mid x^{(i)}, \theta) f(x^{(i)}, y') f(x^{(i)}, y')^\top - p(y' \mid x^{(i)}, \theta) f(x^{(i)}, y') \sum_{y'} p(y' \mid x^{(i)}, \theta) f(x^{(i)}, y')^\top$
- $= \sum_{i=1}^n \mathbb{E}[f(x^{(i)}, y') f(x^{(i)}, y')^\top] - \sum_{i=1}^n ((\sum_{y'} p(y' \mid x^{(i)}, \theta) f(x^{(i)}, y'))(\sum_{y'} p(y' \mid x^{(i)}, \theta) f(x^{(i)}, y')^\top))$
- $= \sum_{i=1}^n \mathbb{E}[f(x^{(i)}, y') f(x^{(i)}, y')^\top] - \sum_{i=1}^n (\mathbb{E}[f(x^{(i)}, y')]\mathbb{E}[f(x^{(i)}, y')])$

- $= \sum_{i=1}^{n} Cov(f(\boldsymbol{x}^{(i)}, y'))$
- We can set a different prior by introducing an offset:
  - $p_k = \frac{e^{z_k + t_k}}{\sum_{\ell=1}^{k} e^{z_\ell + t_\ell}}$ where $t_k = \ln(\frac{q(x_k)}{p(x_k)})$
  - Then, the prior on x ($p(x)$) is replaced by $q(x)$

11 Logistic Regression

Formulation

- Probability of each class is estimated via *sigmoid function*:
  $\sigma(z) = \frac{1}{1+e^{-z}} = \frac{e^z}{1+e^z}$
- $P(y=1|\boldsymbol{x}) = \frac{1}{1+e^{-\beta \cdot \boldsymbol{x}}} = \frac{e^{\beta \cdot \boldsymbol{x}}}{1+e^{\beta \cdot \boldsymbol{x}}}$, $P(y=0|\boldsymbol{x}) = \frac{1}{1+e^{\beta \cdot \boldsymbol{x}}} = \frac{e^{-\beta \cdot \boldsymbol{x}}}{1+e^{-\beta \cdot \boldsymbol{x}}}$
- *Odds*: $\frac{P(y=1|\boldsymbol{x})}{P(y=0|\boldsymbol{x})} = e^{\beta \cdot \boldsymbol{x}}$
- *Log odds*: $ln(\frac{P(y=1|\boldsymbol{x})}{P(y=0|\boldsymbol{x})}) = \boldsymbol{\beta} \cdot \boldsymbol{x} = ln(\frac{P(y=1)}{P(y=0)}) + ln(\frac{P(\boldsymbol{x}|y=1)}{P(\boldsymbol{x}|y=0)})$
- Geometrically, $z = \beta \cdot x$ defines a linear separating hyperplane: When $z > 0$ resp. log-odds > 0, then odds > 1 resp. $P(y=1|\boldsymbol{x}) > P(y=0|\boldsymbol{x})$, then predict $y = 1$, otherwise $y = 0$

Optimization

Parameters — Find parameters β

Objective function —

- Likelihood:
  $L(\beta) = \prod_{i=1}^{n} P(y^{(i)}|\boldsymbol{x}^{(i)}; \beta) = \prod_{i=1}^{n} \sigma(z^{(i)})^{y^{(i)}} (1-\sigma(z^{(i)}))^{1-y^{(i)}}$
- Log likelihood:
  $\log L(\beta) = \sum_{i=1}^{n} \left[y^{(i)} \log \sigma(z^{(i)}) + (1-y^{(i)}) \log(1-\sigma(z^{(i)})) \right] =$
  $\sum_{i=1}^{n} \left[y^{(i)} \log \frac{1}{1+e^{-z^{(i)}}} + (1-y^{(i)}) \log \frac{e^{-z^{(i)}}}{1+e^{-z^{(i)}}} \right] =$
  $\sum_{i=1}^{n} \left[y^{(i)} z^{(i)} - \log(1+e^{z^{(i)}}) \right]$
- Log loss: $-\log L(\beta)$

Optimization —

- $\frac{\partial -\log L(\beta)}{\partial \beta} = -\sum_{i=1}^{n} \frac{\partial}{\partial \beta} [y^{(i)} \log \sigma(z^{(i)}) + (1-y^{(i)}) \log(1-\sigma(z^{(i)}))] = \sum_{i=1}^{n} [\sigma(z^{(i)}) - y^{(i)}] \boldsymbol{x}^{(i)}$

Proof:

- Derivative of the sigmoid: $\frac{\partial \sigma(z^{(i)})}{\partial z^{(i)}} = \sigma(z^{(i)})(1-\sigma(z^{(i)}))$
- Derivative of the first term: $\frac{\partial}{\partial \beta} \left[y^{(i)} \log \sigma(z^{(i)}) \right] =$
  $y^{(i)} \frac{1}{\sigma(z^{(i)})} \frac{\partial \sigma(z^{(i)})}{\partial z^{(i)}} \frac{\partial z^{(i)}}{\partial \beta} = y^{(i)} \frac{1}{\sigma(z^{(i)})} \sigma(z^{(i)})(1-\sigma(z^{(i)})) \boldsymbol{x}^{(i)} =$
  $y^{(i)}(1-\sigma(z^{(i)})) \boldsymbol{x}^{(i)} = y^{(i)} \boldsymbol{x}^{(i)} - y^{(i)} \sigma(z^{(i)}) \boldsymbol{x}^{(i)}$
- Derivative of the second term: $\frac{\partial}{\partial \beta} \left[(1-y^{(i)}) \log(1-\sigma(z^{(i)})) \right] =$
  $(1-y^{(i)}) \frac{1}{1-\sigma(z^{(i)})} (-1) \frac{\partial \sigma(z^{(i)})}{\partial z^{(i)}} \frac{\partial z^{(i)}}{\partial \beta} =$
  $(1-y^{(i)}) \frac{1}{1-\sigma(z^{(i)})} (-1) \sigma(z^{(i)})(1-\sigma(z^{(i)})) \boldsymbol{x}^{(i)} =$
  $-(1-y^{(i)}) \sigma(z^{(i)}) \boldsymbol{x}^{(i)} = y^{(i)} \sigma(z^{(i)}) \boldsymbol{x}^{(i)} - \sigma(z^{(i)}) \boldsymbol{x}^{(i)}$

If we set gradient to 0, we have *expectation matching*: $\sum_{i=1}^{n} \sigma(z^{(i)}) \boldsymbol{x}^{(i)} = \sum_{i=1}^{n} y^{(i)} \boldsymbol{x}^{(i)}$

Characteristics — Log loss is convex – Proof:

- Sum of convex functions is convex
- Thus, we need to prove convexity of $\ln(1+e^{\beta \cdot \boldsymbol{x}^{(i)}})$ and $-y^{(i)} \beta \cdot \boldsymbol{x}^{(i)}$
- For second term:
  - $\mathcal{H}(\beta) = \nabla_{\beta}^2 \left(-y^{(i)} \beta \cdot \boldsymbol{x}^{(i)} \right) = 0$
  - $\mathcal{H} \succcurlyeq 0$
- For first term:
  - $\mathbb{H}(\beta) = \nabla_{\beta}^2 \ln(1+e^{\beta \cdot \boldsymbol{x}^{(i)}})$
  - $= v(x) \times u'(x) - v'(x) \times u(x) =$
    $\frac{1}{1+e^{\beta \cdot \boldsymbol{x}}} \times e^{\beta \cdot \boldsymbol{x}} \times \boldsymbol{x}\boldsymbol{x}^\intercal - \frac{e^{\beta \cdot \boldsymbol{x}} \times \boldsymbol{x}}{(1+e^{\beta \cdot \boldsymbol{x}})^2} \times \boldsymbol{x}^\intercal \times e^{\beta \cdot \boldsymbol{x}}$
  - $= \frac{e^{\beta \cdot \boldsymbol{x}} \boldsymbol{x}\boldsymbol{x}^\intercal (1+e^{\beta \cdot \boldsymbol{x}}) - e^{\beta \cdot \boldsymbol{x}} \boldsymbol{x}\boldsymbol{x}^\intercal e^{\beta \cdot \boldsymbol{x}}}{(1+e^{\beta \cdot \boldsymbol{x}})^2}$
  - $= \frac{e^{\beta \cdot \boldsymbol{x}} \boldsymbol{x}\boldsymbol{x}^T}{(1+e^{\beta \cdot \boldsymbol{x}})^2}$
  - $\mathcal{H} \succcurlyeq 0$
    Proof: $\boldsymbol{a}^T \mathcal{H} \boldsymbol{a} = \frac{e^{\beta \cdot \boldsymbol{x}}}{(1+e^{\beta \cdot \boldsymbol{x}})^2} \boldsymbol{a}^T \boldsymbol{x}\boldsymbol{x}^T \boldsymbol{a} = \frac{e^{\beta \cdot \boldsymbol{x}}}{(1+e^{\beta \cdot \boldsymbol{x}})^2} \|\boldsymbol{a}^T \boldsymbol{x}\|^2 \geq 0$

Perfectly separable data requires regularization – Proof:

- Let weights for each class k be scaled by c as $c\tilde{\beta}_k$
- Gradient of log-loss with respect to c is always negative, causing gradient descent to grow c without bound
  - Log loss $= \sum_{i=1}^{n} \ln\left(1+e^{c\tilde{\beta} \cdot \boldsymbol{x}^{(i)}}\right) - y^{(i)} c\tilde{\beta} \cdot \boldsymbol{x}^{(i)}$
  - ∇_c log loss $= \sum_{i=1}^{n} \frac{1}{1+e^{c\tilde{\beta} \cdot \boldsymbol{x}^{(i)}}} \times e^{c\tilde{\beta} \cdot \boldsymbol{x}^{(i)}} \times \tilde{\beta} \cdot \boldsymbol{x}^{(i)} - y^{(i)} \tilde{\beta} \cdot \boldsymbol{x}^{(i)}$
  - $= \sum_{i=1}^{n} \tilde{\beta} \cdot \boldsymbol{x}^{(i)} \left(\frac{e^{c\tilde{\beta} \cdot \boldsymbol{x}^{(i)}}}{1+e^{c\tilde{\beta} \cdot \boldsymbol{x}^{(i)}}} - y^{(i)} \right)$
  - Given perfect separation:
    - If $y^{(i)} = 1$, $\tilde{\beta} \cdot \boldsymbol{x}^{(i)} > 0$, $\frac{e^{c\tilde{\beta} \cdot \boldsymbol{x}^{(i)}}}{1+e^{c\tilde{\beta} \cdot \boldsymbol{x}^{(i)}}} - y^{(i)} < 0$
    - If $y^{(i)} = 0$, $\tilde{\beta} \cdot \boldsymbol{x}^{(i)} < 0$, $\frac{e^{c\tilde{\beta} \cdot \boldsymbol{x}^{(i)}}}{1+e^{c\tilde{\beta} \cdot \boldsymbol{x}^{(i)}}} - y^{(i)} > 0$
  - Thus, for all i, ∇_c log loss < 0

12 Multinomial Logistic Regression

Formulation

- Probability of each class is estimated via the *softmax function*:
  $P(y=k|\boldsymbol{x}) = \frac{e^{f_i(\boldsymbol{x})/T}}{\sum_{j=1}^{k} e^{f_j(\boldsymbol{x})/T}} = \frac{e^{\beta_k \cdot \boldsymbol{x}/T}}{\sum_{j=1}^{k} e^{\beta_j \cdot \boldsymbol{x}/T}}$ where T is the *temperature*

Softmax – Proof:

- Take class k as reference class
- We start with log-odds:
  - $\log\left(\frac{P_y(y=i|\boldsymbol{x})}{P_y(y=k|\boldsymbol{x})}\right) = f_i(\boldsymbol{x}) - f_k(\boldsymbol{x}) = (\boldsymbol{\beta}_i - \boldsymbol{\beta}_k) \cdot \boldsymbol{x} + (\beta_{i0} - \beta_{k0})$
  - $\frac{P_y(y=i|\boldsymbol{x})}{P_y(y=k|\boldsymbol{x})} = \exp(f_i(\boldsymbol{x}) - f_k(\boldsymbol{x})) = \exp((\boldsymbol{\beta}_i - \boldsymbol{\beta}_k) \cdot \boldsymbol{x} + (\beta_{i0} - \beta_{k0}))$
  - $\sum_{i=1}^{k-1} (\frac{P_y(y=i|\boldsymbol{x})}{P_y(y=k|\boldsymbol{x})}) = \frac{1-P_y(y=k|\boldsymbol{x})}{P_y(y=k|\boldsymbol{x})} = \sum_{i=1}^{k-1} \exp(f_i(\boldsymbol{x}) - f_k(\boldsymbol{x}))$
- We can re-form last equation to posterior:
  $P_y(y=k \mid \boldsymbol{x}) = \frac{1}{1+\sum_{i=1}^{k-1} \exp(f_i(\boldsymbol{x}) - f_k(\boldsymbol{x}))}$
- We can re-form secondlast equation to posterior:
  - $P_y(y=i \mid \boldsymbol{x}) = 1 - \frac{1}{1+\sum_{i=1}^{k-1} \exp(f_i(\boldsymbol{x}) - f_k(\boldsymbol{x}))}$
  - $= \frac{\exp(f_i(\boldsymbol{x}) - f_k(\boldsymbol{x}))}{1+\sum_{i=1}^{k-1} \exp(f_i(\boldsymbol{x}) - f_k(\boldsymbol{x}))}$
  - $= \frac{\frac{\exp(f_i(\boldsymbol{x}))}{\exp(f_k(\boldsymbol{x}))}}{1+\sum_{i=1}^{k-1} \frac{\exp(f_i(\boldsymbol{x}))}{\exp(f_k(\boldsymbol{x}))}}$
  - $= \frac{\frac{\exp(f_i(\boldsymbol{x}))}{\exp(f_k(\boldsymbol{x}))}}{\sum_{i=1}^{k-1} \frac{\exp(f_k(\boldsymbol{x})) + \exp(f_i(\boldsymbol{x}))}{\exp(f_k(\boldsymbol{x}))}}$
  - $= \frac{\exp(f_i(\boldsymbol{x})) \times \exp(f_k(\boldsymbol{x}))}{\exp(f_k(\boldsymbol{x})) \times \sum_{i=1}^{k-1} (\exp(f_k(\boldsymbol{x})) + \exp(f_i(\boldsymbol{x})))}$
  - $= \frac{\exp(f_i(\boldsymbol{x}))}{\sum_{j=1}^{k} \exp(f_j(\boldsymbol{x}))}$

Softmax ($T=1$) vs. argmax ($T=0$) – Proof:

- Assume $\boldsymbol{x} = [x_1, x_2]^\intercal$
- $\lim_{T \to 0} p(\boldsymbol{x}) = \lim_{T \to 0} \frac{e^{x_1/T}}{e^{x_1/T} + e^{x_2/T}}$
- $= \lim_{T \to 0} \frac{e^{x_1/T} e^{-x_1/T}}{(e^{x_1/T} + e^{x_2/T}) e^{-x_1/T}}$
- $= \lim_{T \to 0} \frac{1}{1+e^{(x_2-x_1)/T}}$
- $\lim_{T \to 0} e^{(x_2-x_1)/T} = \begin{cases} 0 & \text{if } x_1 > x_2 \\ 1 & \text{if } x_1 = x_2 \\ \infty & \text{otherwise} \end{cases}$
- Plugging back into $\lim_{T \to 0} p(\boldsymbol{x}) = \frac{1}{1+e^{(x_2-x_1)/T}}$:
  $p(\boldsymbol{x}) = \begin{cases} [1,0]^\intercal & \text{if } x_1 > x_2 \\ [0.5,0.5]^\intercal & \text{if } x_1 = x_2 \\ [0,1]^\intercal & \text{otherwise} \end{cases}$

Optimization

Parameters — Find parameters $\beta_1, \ldots, \beta_k$

Objective function —

- Likelihood:
  $L(\beta) = \prod_{i=1}^{n} P(y^{(i)}|\boldsymbol{x}^{(i)}; \beta) = \prod_{i=1}^{n} \prod_{\ell=1}^{k} (\frac{e^{\beta_\ell \cdot \boldsymbol{x}^{(i)}}}{\sum_{j=1}^{k} e^{\beta_j \cdot \boldsymbol{x}^{(i)}}})^{\delta\{y^{(i)}=\ell\}}$
- Log likelihood:
  $\log L(\beta) = \sum_{i=1}^{n} \sum_{\ell=1}^{k} \delta\{y^{(i)} = \ell\} [\beta_\ell \cdot \boldsymbol{x}^{(i)} - \log(\sum_{j=1}^{k} e^{\beta_j \cdot \boldsymbol{x}^{(i)}})]$
- Log loss: $-\log L(\beta)$

Optimization —

- $\frac{\partial -\log L(\beta)}{\partial \beta_k} = -\sum_{i=1}^{n} \sum_{\ell=1}^{k} \frac{\partial}{\partial \beta_k} [\delta\{y^{(i)} = \ell\} [\beta_\ell \cdot \boldsymbol{x}^{(i)} - \log(\sum_{j=1}^{k} e^{\beta_j \cdot \boldsymbol{x}^{(i)}})]] = -\sum_{i=1}^{n} \delta\{y^{(i)} = k\} \boldsymbol{x}^{(i)} - P(y=k|\boldsymbol{x}^{(i)}) \boldsymbol{x}^{(i)}$

Proof:

- Derivative of the first term:
  $\frac{\partial}{\partial \beta_k} (\sum_{\ell=1}^{k} \delta\{y^{(i)} = \ell\} \beta_\ell \cdot \boldsymbol{x}^{(i)}) = \delta\{y^{(i)} = k\} \boldsymbol{x}^{(i)}$
- Derivative of the second term:
  $\frac{\partial}{\partial \beta_k} (-\log(\sum_{j=1}^{k} e^{\beta_j \cdot \boldsymbol{x}^{(i)}})) = -\frac{1}{\sum_{j=1}^{k} e^{\beta_j \cdot \boldsymbol{x}^{(i)}}} \frac{\partial}{\partial \beta_k} (\sum_{j=1}^{k} e^{\beta_j \cdot \boldsymbol{x}^{(i)}}) =$
  $-\frac{e^{\beta_k \cdot \boldsymbol{x}^{(i)}}}{\sum_{j=1}^{k} e^{\beta_j \cdot \boldsymbol{x}^{(i)}}} \boldsymbol{x}^{(i)} = -P(y=k|\boldsymbol{x}^{(i)}) \boldsymbol{x}^{(i)}$
- For reference: Softmax derivative if $\ell = k$: $\frac{\partial P(y=\ell|\boldsymbol{x})}{\partial \beta_k}$:
  - Using the quotient rule:
    $\frac{\partial P(y=\ell|\boldsymbol{x})}{\partial \beta_\ell} = \frac{\frac{\partial}{\partial \beta_\ell} e^{\beta_\ell \cdot \boldsymbol{x}} (\sum_{j=1}^{k} e^{\beta_j \cdot \boldsymbol{x}}) - e^{\beta_\ell \cdot \boldsymbol{x}} \frac{\partial}{\partial \beta_\ell} (\sum_{j=1}^{k} e^{\beta_j \cdot \boldsymbol{x}})}{(\sum_{j=1}^{k} e^{\beta_j \cdot \boldsymbol{x}})^2}$
  - $\frac{\partial}{\partial \beta_\ell} e^{\beta_\ell \cdot \boldsymbol{x}} = \frac{\partial}{\partial \beta_\ell} (\sum_{j=1}^{k} e^{\beta_j \cdot \boldsymbol{x}}) = e^{\beta_\ell \cdot \boldsymbol{x}} \boldsymbol{x}$
  - After plugging this in, we get:
    $\frac{\partial P_\ell}{\partial \beta_\ell} = \frac{e^{\beta_\ell \cdot \boldsymbol{x}} (\sum_{j=1}^{k} e^{\beta_j \cdot \boldsymbol{x}}) - e^{\beta_\ell \cdot \boldsymbol{x}} e^{\beta_\ell \cdot \boldsymbol{x}} \boldsymbol{x}}{(\sum_{j=1}^{k} e^{\beta_j \cdot \boldsymbol{x}})^2} = P(y=\ell|\boldsymbol{x})(1-P(y=\ell|\boldsymbol{x}))x$
- For reference: Softmax derivative if $\ell \neq k$: $\frac{\partial P(y=\ell|\boldsymbol{x})}{\partial \beta_k}$:
  - Using the quotient rule:

$$\frac{\partial P(y=\ell|\boldsymbol{x})}{\partial \boldsymbol{\beta}_k} = \frac{\frac{\partial}{\partial \boldsymbol{\beta}_k} e^{\boldsymbol{\beta}_\ell \cdot \boldsymbol{x}} (\sum_{j=1}^k e^{\boldsymbol{\beta}_j \cdot \boldsymbol{x}}) - e^{\boldsymbol{\beta}_\ell \cdot \boldsymbol{x}} \frac{\partial}{\partial \boldsymbol{\beta}_k} (\sum_{j=1}^k e^{\boldsymbol{\beta}_j \cdot \boldsymbol{x}})}{(\sum_{j=1}^k e^{\boldsymbol{\beta}_j \cdot \boldsymbol{x}})^2}$$

- First term vanishes, since it does not depend on $\boldsymbol{\beta}_k$
- $\frac{\partial}{\partial \boldsymbol{\beta}_k} (\sum_{j=1}^k e^{\boldsymbol{\beta}_j \cdot \boldsymbol{x}}) = e^{\boldsymbol{\beta}_k \cdot \boldsymbol{x}} \boldsymbol{x}$
- After plugging this in, we get:
  $\frac{\partial P_\ell}{\partial \boldsymbol{\beta}_k} = \frac{-e^{\boldsymbol{\beta}_\ell \cdot \boldsymbol{x}} e^{\boldsymbol{\beta}_k \cdot \boldsymbol{x}} \boldsymbol{x}}{(\sum_{j=1}^k e^{\boldsymbol{\beta}_j \cdot \boldsymbol{x}})^2} = -P(y=\ell|\boldsymbol{x}) P(y=k|\boldsymbol{x}) \boldsymbol{x}$

– If we set gradient to 0, we have expectation matching:
$\sum_{i=1}^n \delta\{y^{(i)}=k\} \boldsymbol{x}^{(i)} = \sum_{i=1}^n P(y=k|\boldsymbol{x}^{(i)}) \boldsymbol{x}^{(i)}$ resp.
$\boldsymbol{x}^{(l)} = \mathbb{E}_{j \sim \text{softmax}[\boldsymbol{x}^{(l)}]}$

13 Neural Networks

Formulation

Formulation —
– Model architecture:
  * Inputs: n words
  * Transformation: word embedding $\boldsymbol{e}(w_i)$
  * Concatenated vector of word embeddings: $\boldsymbol{e}(\boldsymbol{x}) = \frac{1}{n} \sum_{w_i} \boldsymbol{e}(w_i)$
  * Hidden layer: $\boldsymbol{h}^{(1)} = \sigma(\boldsymbol{w}^{(1)} \boldsymbol{e}(\boldsymbol{x}))$
    ...
    $\boldsymbol{h}^{(K)} = \sigma(\boldsymbol{w}^{(K)} \boldsymbol{h}^{(K-1)})$
  * Activation: Softmax: $p(y \mid \boldsymbol{x}) = \frac{\exp\left(\boldsymbol{h}_y^{(K)}\right)}{\sum_{y'} \exp\left(\boldsymbol{h}_{y'}^{(K)}\right)}$
– Neuron (j) in layer $[k]$ given training instance $\boldsymbol{x}^{(i)[0]}$ resp. instance from previous layer $\boldsymbol{h}^{(i)[k-1]}$ given by:
$\boldsymbol{h}^{(j)[1]} = \varphi(\boldsymbol{x}^{(i)[0]} \cdot \boldsymbol{\beta}^{(j)[1]})$ resp. $\boldsymbol{h}^{(j)[k]} = \varphi(\boldsymbol{h}^{(i)[k-1]} \cdot \boldsymbol{\beta}^{(j)[k]})$
– Outputs for neurons $1, \ldots, j$ in fixed layer (notation for layer omitted below) given by:
$\boldsymbol{H} = \varphi(\boldsymbol{X}\boldsymbol{B}) = \varphi(\boldsymbol{S})$ where
  * $\boldsymbol{X} \in \mathbb{R}^{n \times m+1}$ (incl. bias term)
  * $\boldsymbol{B} \in \mathbb{R}^{m+1 \times j}$ with a weight vector for each neuron in each column (incl. bias term)
  * $\boldsymbol{S} \in \mathbb{R}^{n \times j}$ with the weighted sum (prior to activation) for instance i in neuron j is on the i^{th} row and j^{th} column

Activation functions —
– Introduce non-linearities
– Differ by neuron
– *Sigmoid*:
  * $[0,1]$
  * $\varphi(z) = \sigma(z) = \frac{1}{1+e^{-z}} = \frac{e^z}{e^z+1}$
  * $\varphi'(z) = \frac{e^{-z}}{(1+e^{-z})^2}$ with maximum at 0.25
– *Hyperbolic tangent*:
  * $[-1,1]$
  * $\varphi(z) = tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} = \frac{1-e^{-2z}}{1+e^{-2z}}$
  * $\varphi'(z) = 1 - tanh(z)^2$
– *ReLU*:
  * $\varphi(z) = max(0,z)$
  * $\varphi'(z) = 1$ if $z > 0$; 0 otherwise

Optimization

Parameters — Find parameters $\theta = \boldsymbol{B}$

Objective function —
– Minimize standard objectives, e.g. MSE

Optimization —
– Perform forward pass with randomly initialized parameters, to calculate loss
– Perform backpropagation, to calculate gradient:
  * $\frac{\partial L}{\partial \theta} = [\frac{\partial L}{\partial \boldsymbol{B}^{[0]}}, \ldots, \frac{\partial L}{\partial \boldsymbol{B}^{[output]}}]$
  * $\frac{\partial L}{\partial \boldsymbol{B}^{[k]}} = \frac{\partial L}{\partial \boldsymbol{H}^{[l]}} \frac{\partial \boldsymbol{H}^{[l]}}{\partial \boldsymbol{B}^{[k]}} = C$
    - When $l > k+1$, i.e. when going several layers back:
      $\frac{\partial L}{\partial \boldsymbol{B}^{[k]}} = \frac{\partial L}{\partial \boldsymbol{H}^{[l]}} \frac{\partial \boldsymbol{H}^{[l]}}{\partial \boldsymbol{S}^{[l-1]}} \frac{\partial \boldsymbol{S}^{[l-1]}}{\partial \boldsymbol{H}^{[l-1]}} \frac{\partial \boldsymbol{H}^{[l-1]}}{\partial \boldsymbol{B}^{[k]}}$
    - When $l = k+1$, i.e. when going one layer back:
      $\frac{\partial L}{\partial \boldsymbol{B}^{[k]}} = \frac{\partial L}{\partial \boldsymbol{H}^{[k+1]}} \frac{\partial \boldsymbol{H}^{[k+1]}}{\partial \boldsymbol{S}^{[k]}} \frac{\partial \boldsymbol{S}^{[k]}}{\partial \boldsymbol{B}^{[k]}}$
– Perform gradient descent to find best weights

Challenges —
– Unstable gradients:
  * Can happen since backpropagation computes gradients using the chain rule, meaning many gradients are multiplied across many layers
  * Caused by poor choice of activation, typically sigmoid or tanh with high absolute input values
  * Caused when weights are shared across many layers, especially in RNNs
  * *Exploding gradients*: If gradients are > 1, gradients grow bigger and bigger during backpropagation, algorithm diverges
  * *Vanishing gradients*: If gradients are < 1 (resp. parameter θ is < 1), gradients approach 0 during backpropagation, algorithm fails to converge
    Proof:
    - $h_m = \sigma(\theta h_{m-1} + x_m)$
    - $\frac{\partial h_{m+k}}{\partial h_m} = \prod_{i=0}^{k-1} \frac{\partial h_{m+k-i}}{\partial h_{m+k-i-1}} = \prod_{i=1}^{k} \theta \times \sigma'(\theta h_{m+k-i-1} + x_{m+k-i})$
    - $\leq \prod_{i=1}^{k} \theta \times 0.25 = \theta^k \times 0.25^k$ since derivative of sigmoid has 0.25 as maximum value
    - $\to 0$ as $k \to \infty$, since $\theta < 1$
  * Solution:
    - Use fewer layers
    - Use ReLU activation function
    - Use residual networks (ResNet)
    - Use LSTM or GRU units
    - Glorot or He initialization: Connection weights of each layer are initialized randomly
    - Batch normalization
    - Gradient clipping: Set maximum threshold for gradients during backpropagation
– Dying ReLUs:
  * Caused when weights are tweaked such that a neuron becomes negative, causing ReLU activation to output 0
  * Can happen if β_0 in $x^T w + \beta_0$ is large and negative
  * Dead neuron cannot be brought back:
    - Let $z = x^T w + b^{[l]}$.
    - $\text{ReLU}\left(b^{[l]}\right) = 0$ if $b^{[l]} \leq 0$
    - Then $\frac{\partial l}{\partial z} = 0$, $\frac{\partial z}{\partial h} = 0$, $\frac{\partial h}{\partial b^{[l]}} = 0$
    - $h_{b^{[l]}}$ is guaranteed to be zero for all inputs if h is dead
    - Then, parameters cannot change and h will remain dead
  * Solution: Leaky ReLU, ELU, scaled ELU

14 Backpropagation

Big Picture

– In ANNs, we learn both f and θ, meaning that the log loss is not convex
– Backpropagation uses the chain rule in combination with dynamic programming techniques to compute the gradient of the log loss $\nabla_\theta L(\theta)$

Point of Departure

Composite function — Ordered series of equations (*primitives*), where each equation is a function only of the preceding equations. E.g.:
$f(x,z) = x^2 + 3z, \quad \text{with } a = x^2, \quad b = 3z, \quad c = a + b, \quad f(x,z) = c$

Computation graph $\mathcal{G}$ — Graphical representation of a composite function
– *Directed acyclic hypergraph* (E,V) where V is the set of vertices (nodes) representing variables and E is the set of edges representing functions
  * Directed: One direction
  * Acyclic: No cycles
  * An edge can connect any number of vertices, not just two
– Edge from $v' \to v$: labeled with a differentiable function f_v
– Vertex v with outgoing edges: argument to f_v
– Vertex v with incoming edges: result of f_v
– If we have n input nodes and $|E| = M$ edges, we have $|V| = M + n$ total nodes

Bauer's Formula —
– For two nodes z_i and z_j in $\mathcal{G}$, let the Bauer path $P(i,j)$ define the set of all directed paths starting at j and ending at i
– Partial derivative: $\frac{\partial z_i}{\partial z_j} = \sum_{p \in P(j,i)} \prod_{(k,l) \in p} \frac{\partial z_l}{\partial z_k}$ where we sum over all paths and take the product over all nodes on a given path
  Proof:
  * Base case: $i = j$, i.e., $z_i = z_j$: $\frac{\partial z_i}{\partial z_j} = 1$
  * Inductive hypothesis: Bauer's formula holds for all $j \leq i$
  * Inductive step:
    - Let $m < j$ (i.e., z_m comes before z_j in topological order, and $j \in \text{out}(m)$)
    - $\frac{\partial z_i}{\partial z_m} = \sum_{j \in \text{out}(m)} \frac{\partial z_i}{\partial z_j} \frac{\partial z_j}{\partial z_m}$ by the chain rule
    - $\frac{\partial z_i}{\partial z_m} = \sum_{j \in \text{out}(m)} (\sum_{p \in P(j,i)} \prod_{(k,l) \in p} \frac{\partial z_l}{\partial z_k}) \frac{\partial z_j}{\partial z_m}$ due to inductive hypothesis
    - $\frac{\partial z_i}{\partial z_m} = \sum_{j \in \text{out}(m)} (\sum_{p \in P(j,i)} \frac{\partial z_j}{\partial z_m} \prod_{(k,l) \in p} \frac{\partial z_l}{\partial z_k})$ due to distributivity
    - $\frac{\partial z_i}{\partial z_m} = \sum_{p \in P(m,i)} \prod_{(k,l) \in p} \frac{\partial z_l}{\partial z_k}$ by concatenating paths
– Challenge of naively calculating this: With $\sum_{p \in P(j,i)}$, we are summing over an exponential number of paths, leading to a runtime complexity of $O(|P(j,i)|) = O(2^{|E|})$

Forward Pass

Compute output of f by randomly initializing values of input nodes:
1. Initialize node values:
   $z_i = \begin{cases} x_i & \text{if } i \leq n \quad \text{(n input nodes)} \\ 0 & \text{if } i > n \quad \text{(non-input nodes)} \end{cases}$ For $i = n+1, \ldots, M$
   non-input nodes: $z_i = g_i\left(\langle z_{\text{pa}(i)} \rangle\right)$ where g_i denotes the primitive at edge i and $\langle z_{\text{pa}(i)} \rangle$ denotes the ordered set of parent nodes of z_i
2. Return $\{z_1, z_2, \ldots, z_M\}$

Properties:
– Runtime complexity: $O(|E|)$, i.e., linear in the number of edges
– Space complexity: $O(|V|)$, i.e., linear in the number of vertices

Backpropagation

After forward pass, compute the gradient of f with respect to input nodes:

1. Perform forward pass: $z \leftarrow \text{forward propagate}(f, \boldsymbol{x})$
2. Initialize: $\frac{\partial f}{\partial z_i} = \begin{cases} 1 & \text{if } i = M \\ & \text{(gradient of } f \text{ with respect to} \\ & \text{output node} = 1, \text{ since } \partial f / \partial f = 1) \\ 0 & \text{otherwise} \end{cases}$
3. For $i = M-1, \ldots, 1$:
   $\frac{\partial f}{\partial z_i} = \sum_{j:i \in \text{Pa}(i)} \frac{\partial f}{\partial z_j} \frac{\partial z_j}{\partial z_i} = \sum_{j:i \in \text{Pa}(i)} \frac{\partial f}{\partial z_j} \frac{g_j\left(\langle z_{\text{pa}(j)} \rangle\right)}{\partial z_i}$
4. Return $\left[\frac{\partial f}{\partial z_1}, \frac{\partial f}{\partial z_2}, \ldots, \frac{\partial f}{\partial z_M}\right]$

Properties:

- Improvement over Bauer's formula: Partial derivatives that appear on multiple paths are memoized
- Runtime complexity: $O(|E|)$, i.e., the same as forward pass
- Space complexity: $O(|V|)$, i.e., the same as forward pass
- *Cheap gradient principle*: Calculating the gradient has the same complexity as evaluating the function

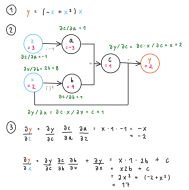
Extension of backpropagation to k^{th} order derivatives:

- Backpropagation on a graph with $|E|$ edges, for inputs $\boldsymbol{x} = (x_1, \ldots, x_n)^\intercal$, for the k^{th} order derivative has runtime $\mathcal{O}(|E| n^{k-1})$
  Proof:
  * For second-order derivative:
    $\nabla_2 f(\boldsymbol{x})(\text{i.e. Hessian}) = \begin{bmatrix} \nabla(\boldsymbol{e}_1^\intercal \nabla f(\boldsymbol{x})(\text{i.e. Jabobian})) \\ \ldots \\ \nabla(\boldsymbol{e}_n^\intercal \nabla f(\boldsymbol{x})) \end{bmatrix}$ because $\boldsymbol{e}_k^\intercal \boldsymbol{A}$ returns k^{th} row of $\boldsymbol{A}$
  * For third-order derivative, similar principle
  * We first differentiate in M edges (∇_1), which means complexity is $1 \times M$
  * We then differentiate by n variables in M edges (∇_2), which means complexity is $n \times M$
  * We then differentiate these n variables by another n variables in M edges (∇_3), which means complexity is $n^2 \times M$
  * ...

Requirements for backpropagation:

- Weights need to be initialized to different values: If they are initialized to the same constant, during backpropagation, all neurons receive the same gradient updates
  * Weights initialized to 0:
    · $\boldsymbol{h} = \varphi\left(\boldsymbol{x} \boldsymbol{B}^{[1]}\right) = 0$ where $\boldsymbol{h}$ corresponds to z, φ is the primitive, and $\boldsymbol{x}$ corresponds to the input nodes
    · $\boldsymbol{y} = \boldsymbol{h} \boldsymbol{B}^{[2]} = 0$ where $\boldsymbol{y}$ corresponds to z'
    · $\frac{\partial L}{\partial \boldsymbol{B}^{[2]}} = \frac{\partial L}{\partial \boldsymbol{y}} \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{B}^{[2]}} = \frac{\partial L}{\partial \boldsymbol{y}} \boldsymbol{h} = 0$, i.e. $\boldsymbol{B}^{[2]}$ is not updated
    · $\frac{\partial L}{\partial \boldsymbol{B}^{[1]}} = \frac{\partial L}{\partial \boldsymbol{y}} \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{B}^{[1]}} = \frac{\partial L}{\partial \boldsymbol{y}} \boldsymbol{B}^{[2]} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{B}^{[1]}} = 0$, i.e. $\boldsymbol{B}^{[1]}$ is not updated
    · Thus, weights will always remain 0 and network will not learn
  * Weights initialized to same constant:
    · $\boldsymbol{B}^{[1]} = \boldsymbol{B}^{[2]}$
    · $\boldsymbol{h}_1 = \boldsymbol{h}_2$
    · $\frac{\partial L}{\partial \boldsymbol{B}^{[1]}} = \frac{\partial L}{\partial \boldsymbol{B}^{[2]}}$
    · Thus, weights will always receive same updates, will remain equal, and network will not learn
- At least one activation must be non-linear so that there is a non-zero gradient

1) Forward pass and backpropagation in graph



15 Convolutional Neural Networks (CNNs)

Formulation

Formulation —

- Model architecture:
  * Input: Composed of channels (e.g. R with 3 channels)
  * *Channel*: Sublayer in input and output, composed of pixels
  * *Convolutional layer*: Composed of feature maps
  * *Feature map*: Sublayer in convolutional layer, composed of neurons, each neuron is generated by applying filter to all receptive fields across all sublayers in lower layer, weights and biases shared across all neurons in feature map
  * *Receptive field*: Group of neurons in lower layer, that single neuron in higher layer is connected to, size $f_h \times f_w$
  * *Filter resp. convolutional kernel*: Weights applied to all receptive fields across all sublayers in lower layer, size $K \times K$
  * *Zero padding*: Padding applied to retain same dimensions in each layer, size $\frac{f_h - 1}{2}$ resp. $\frac{f_w - 1}{2}$
  * *Stride*: By how many neurons receptive field shifts, size $s_h \times s_w$, if stride > 1, spatial dimensions in subsequent layer decrease (*convolution*), if stride < 1, spatial dimensions increase (*deconvolution*)
- Output of neuron in layer n, given previous layer $n-1$: $z_{i,j,k} = b_k + \sum_{f_n} \sum_{f_w} \sum_{f_n'} x_{i',j',k'} \cdot w_{u,v,k',k}$, i.e. sum of element-wise matrix product over all receptive fields and all feature maps, where
  * $z_{i,j,k}$ is the output of neuron in row i and column j on feature map k in layer n
  * f_n and f_w are dimensions of the receptive field in layer $n-1$
  * f_n' is the number of feature maps in layer $n-1$
  * $x_{i',j',k'}$ is the output of neuron in row i' and column j' on feature map k' in layer $n-1$
  * $i' = i \times \text{stride}_h + u - \text{padding}_h$ and $j' = j \times \text{stride}_w + v - \text{padding}_w$
  * $w_{u,v,k',k}$ is the connection weight between any neuron on feature map k in layer n and its input at u, v on feature map k'
  * $u, v \in \Delta_K$ are possible shifts allowed by kernel
- Output of neurons in layer n, given previous layer $n-1$: $\boldsymbol{z}_k = b_k + \sum_{f_n} \sum_{f_w} \sum_{f_n'} \boldsymbol{W}_{k',k} \boldsymbol{X}_{k'}$
- Output size in layer n, given previous layer $n-1$: $H' = \frac{H + 2p - K}{\text{stride}_h} + 1$ and $W' = \frac{W + 2p - K}{\text{stride}_w} + 1$

16 Recurrent Neural Networks (RNN)

Formulation

Formulation —

- Model architecture:
  * Input layer
  * Hidden layer resp. *memory cell*: *Cell state* $\boldsymbol{h}_t$, *cell output* $\boldsymbol{y}_t$
  * Output layer
- Output of neuron in layer n:
  $\boldsymbol{Y}_t = \phi\left(\boldsymbol{X}_t \boldsymbol{W}_x + \boldsymbol{H}_{t-1} \boldsymbol{W}_y + \boldsymbol{b}\right) \boldsymbol{V} = \phi\left(\begin{bmatrix} \boldsymbol{X}_t \\ \boldsymbol{H}_{t-1} \end{bmatrix} \boldsymbol{W} + \boldsymbol{b}\right) \boldsymbol{V}$
  * $\boldsymbol{V}$ is optional: If $\boldsymbol{V} = \boldsymbol{I}$ (identity matrix), $\boldsymbol{H}_t = \boldsymbol{Y}_t$, this is assumed in the following
  * $\boldsymbol{Y}_t$ is an $n_{\text{instances}} \times n_{\text{neurons}}$ matrix containing layer outputs for instances at time t
  * $\boldsymbol{X}_t$ is an $n_{\text{instances}} \times m$ matrix containing encoded inputs for all instances
  * $\boldsymbol{H}_{t-1}$ is an $n_{\text{instances}} \times n_{\text{neurons}}$ matrix containing cell state outputs for instances at time $t-1$
  * $\boldsymbol{W}_x$ is an $m \times n_{\text{neurons}}$ matrix containing connection weights for $\boldsymbol{X}_t$
  * $\boldsymbol{W}_y$ is an $n_{\text{neurons}} \times n_{\text{neurons}}$ matrix containing connection weights for $\boldsymbol{Y}_{t-1}$ (resp. $\boldsymbol{H}_{t-1}$)
  * $\boldsymbol{b}$ is a vector of length n_{neurons} containing the bias term
  * $\boldsymbol{W} = \begin{bmatrix} \boldsymbol{W}_x \\ \boldsymbol{W}_y \end{bmatrix}$
  * ϕ is a non-linear activation function

Optimization

Parameters — Find parameters $\theta = \boldsymbol{W}_x, \boldsymbol{W}_y, \boldsymbol{b}, \boldsymbol{X}_t$, which are shared across time steps

Objective function —

- Maximize log likelihood

Optimization —

- Perform forward pass
- Perform backpropagation
- Gradient:
  * $\boldsymbol{y} = \varphi(\boldsymbol{X}_h \boldsymbol{W}_h)$
  * $\nabla_{\boldsymbol{W}_h} L \propto \sum_{k=1}^{t} \left(\prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}}\right) \frac{\partial h_k}{\partial \boldsymbol{W}_k}$
  * $\frac{\partial h_{i+k}}{\partial h_i} = \prod_{j=0}^{k-1} \frac{\partial h_{i+k-j}}{\partial h_{i+k-j-1}}$
- Perform gradient descent

17 Long-Short-Term Memory (LSTM)

Formulation

Formulation —

- Model architecture:
  * Input layer
  * Hidden layer resp. *memory cell*:
    · *Short-term state* $\boldsymbol{h}_t$, *long-term state* $\boldsymbol{c}_t$
    · Cell output $\boldsymbol{y}_t$
  * Output layer
- *Forget gate*:
  * Sigmoid layer
  * Serves to decide which information to keep from previous cell state, 1 : retain, 0: forget
  * $\boldsymbol{f}_t = \sigma(\boldsymbol{w}_f \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_f$ where $\boldsymbol{w}_f = \begin{bmatrix} \boldsymbol{w}_{xf} & = \text{connection weight for } x_t \\ \boldsymbol{w}_{hf} & = \text{connection weight for } h_{t-1} \end{bmatrix}$
- *Input gate*:
  * Two stages:
    · Sigmoid layer, determines if values are updated, 1 : update values, 0: do not update values
    · Tanh layer, creates vector of new candidate values that could be added to the cell state
  * Serves to decide which information will be stored in the cell state
  * $\boldsymbol{i}_t = \sigma(\boldsymbol{w}_i \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_i$ where $\boldsymbol{w}_i = \begin{bmatrix} \boldsymbol{w}_{xi} & = \text{connection weight for } x_t \\ \boldsymbol{w}_{hi} & = \text{connection weight for } h_{t-1} \end{bmatrix}$
  * $\tilde{\boldsymbol{c}}_t = tanh(\boldsymbol{w}_{\tilde{c}} \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_c$ where $\boldsymbol{w}_{\tilde{c}} = \begin{bmatrix} \boldsymbol{w}_{x\tilde{c}} & = \text{connection weight for } x_t \\ \boldsymbol{w}_{h\tilde{c}} & = \text{connection weight for } h_{t-1} \end{bmatrix}$

- *Cell state*:
  - Two stages:
    - Calculate what is left from previous cell state after forget care
    - Calculate what we need to add to cell state after input gate
  - Serves to update old cell state into new cell state
  - $c_t = f_t \otimes c_{t-1} + i_t \otimes \tilde{c}_t$ where $\otimes$ is element-wise multiplication
- *Output gate*:
  - Three stages:
    - Sigmoid layer, determines which part of cell state to output
    - Tanh layer, activates cell state values
    - Multiply activated cell state and output gate values to get h_t
  - Serves to output h_t for next time step, which is a filtered version of cell state c_t
  - $o_t = \sigma(w_o \cdot [h_{t-1}, x_t] + b_o$ where $w_o = \begin{bmatrix} w_{xo} & = \text{connection weight for } x_t \\ w_{ho} & = \text{connection weight for } h_{t-1} \end{bmatrix}$
  - $h_t = o_t \otimes tanh(c_t)$

18 Attention

Description

- Helps specify which inputs we need to pay attention to when producing a given output
- Can be used:
  - As *cross-attention*: Between encoder and decoder
  - As *self-attention*: Within encoder or decoder

Adjusted method

Steps:

- Generate three sets of re-weighted embeddings:
  - $Q = EW^q$ resp. $q_i = e_i W^q$
    - E $(m \times h)$
    - W_q $(h \times d_k)$
    - Q $(m \times d_k)$
    - q_i is row vector $(1 \times d_k)$
    - e_i is row vector $(1 \times h)$
  - $K = EW^k$ resp. $k_i = e_i W^k$
    - E $(n \times h)$
    - W_k $(h \times d_k)$
    - K $(n \times d_k)$
    - k_i is row vector $(1 \times d_k)$
    - e_i is row vector $(1 \times h)$
  - $V = EW^v$ resp. $v_i = e_i W^v$ where
    - E $(n \times h)$
    - W_v $(h \times d_v)$
    - V $(n \times d_v)$
    - v_i is row vector $(1 \times d_v)$
    - e_i is row vector $(1 \times h)$
- Compute *similarity matrix*: $A = \sigma(\frac{QK^\top}{\sqrt{d_k}})$ in $(m \times n)$ resp. $\alpha_t = \sigma(\frac{q_t K^\top}{\sqrt{d_k}})$ resp. $\alpha_{ti} = \frac{exp(q_t \cdot k_i)}{\sum_{i'} exp(q_t \cdot k_{i'})}$
- Compute *attention-weighted embedding matrix*: $Z = AV$ in $(m \times d_v)$ resp. $z_t = \alpha_t V = \sum_i \alpha_{ti} v_i$
- In cross-attention:
  - Q is decoder input with m
  - V, K are encoder outputs with n
- In self-attention:
  - Q, V, K are all either encoder or decoder inputs with n or m
  - In *masked self-attention*, states with time $\geq m$ in decoder are masked
- In *multi-head attention*:
  - Creates multiple sets (*heads*) of Q, K, V and calculates attention correspondingly
  - Concatenates generated matrices Z
  - $\text{Multi Head Attention}Z = \text{Concat}(Z_{\text{head}_1}, \ldots, Z_{\text{head}_h})W_O + b_O$ where
    - $\text{Concat}(\ldots)$ in $(m \times (n \times n_{heads}))$
    - W_O in $((n_{heads} \times n) \times d_v)$
    - b_O in $1 \times d_v)$

Further proofs

Self-attention without positional encodings is permutation equivariant

- *Permutation equivariance*: $\text{Attention}\Pi Z = \Pi\, \text{Attention}Z$
- The self-attention is given by: $A = ZW_q W_k^\top Z^\top$
- After permutation, self-attention is given by: $A' = (\Pi Z)W_q W_k^\top (\Pi Z)^\top = \Pi Z W_q W_k^\top Z^\top \Pi^\top = \Pi(ZW_q W_k^\top Z^\top)\Pi^\top = \Pi A \Pi^\top$
- Applying softmax: $\text{softmax}(A') = \text{softmax}(\Pi A \Pi^\top) = \Pi\, \text{softmax}(A)\, \Pi^\top$ since permutation matrix simply swaps rows and columns. The softmax operates on a matrix row-wise, i.e. the normalization for each row only depends on entries in that row. For this reason, it does not matter whether the permutation happens before or after applying the softmax
- Final output: $Z' = \text{softmax}(A')(\Pi Z)W_v = \Pi\, \text{softmax}(A)\, \Pi^\top (\Pi Z)W_v = \Pi\, \text{softmax}(A)(\Pi^{-1}\Pi)ZW_v = \Pi\, \text{softmax}(A)ZW_v$ because Π, as a permutation matrix, has exactly one 1 in each row and each column and 0 everywhere else. It is an orthogonal matrix, thus $\Pi^\top = \Pi^{-1}$ and $\Pi\Pi^{-1} = I$

Self-attention with learned Q and without positional encodings is permutation invariant

- *Permutation invariance*: $\text{Attention}\Pi Z = \text{Attention}Z$
- See proof above, but do not decompose Q

Self-attention with positional encodings is not permutation equi- or invariant

19 Positional Embeddings

- Can be absolute or relative
- Attention with *absolute positional encodings*: $A^{\text{absolute}}_{q,k} = \left(Z_q + P_q\right) W_q W_k^\top (Z_k + P_k)^\top = Z_q W_q W_k^\top Z_k^\top + Z_q W_q W_k^\top P_k^\top + P_q W_q W_k^\top Z_k^\top + P_q W_q W_k^\top P_k^\top$
- Attention with *relative positional encodings*, where relative difference $\delta = q - k$: $A^{\text{relative}}_{q,k} := Z_q W_q W_k^\top Z_k^\top + Z_q W_q \widetilde{W}_k r_\delta + u^\top W_k Z_k + v^\top \widetilde{W}_k r_\delta$ where *Gaussian encodings* are given by parameters
  - $W_q = W_k = 0$
  - $\widetilde{W}_k = I$
  - $r_\delta = \begin{pmatrix} \|\delta\|^2 \\ \delta_1 \\ \delta_2 \end{pmatrix}$
  - $v = -\alpha \begin{pmatrix} 1 \\ -2\Delta_1 \\ -2\Delta_2 \end{pmatrix}$
  - v and r_δ are in $(1 \times d_p)$
  - If these parameters are plugged into formula for attention with relative positional encodings, we recover formula for attention with absolute positional encodings
- Relative encodings speed up the calculation of the attention vs. absolute encodings (since d_p is very small), but applying softmax and calculating Z for relative encodings has the same complexity as for absolute encodings, thus diminishing the benefit

20 Encoder Decoder RNNs

Formulation

Formulation — Model architecture:

- Inputs fed into encoder in reverse order
- *Encoder*:
  - Sequence-to-vector
  - Hidden states $h_n^{(e)} = f(W_1^{(e)} h_{n-1}^{(e)} + W_2 w_n)$ where
    - f is activation function
    - w_n is input token embedding at time step n in input sequence
    - $h_{n-1}^{(e)}$ is encoder hidden state from previous time step
- Outputs from encoder to decoder are weighted by attention weights:
  - Context vector $z_m = \sum_{n=1}^N \alpha_{m,n} h_n^{(e)}$ where
    - $h_n^{(e)}$ is the encoder hidden state $(= V)$
    - $\alpha_{m,n}$ is attention weight at decoder time step m for encoder hidden state at time step n, given by $\text{softmax}(h_{m-1}^{(d)} \times [h_1^{(e)}, \ldots, h_N^{(e)}])$ where $h_{m-1}^{(d)}$ is previous decoder hidden state $(= Q)$ and $[h_1^{(e)}, \ldots, h_N^{(e)}]^\top$ are the final encoder hidden states at each time step $(= K)$
- Alongside context vectors, target sequence inputs are fed into decoder with one time step lag during training
- *Decoder*:
  - Vector-to-sequence
  - Hidden states $h_m^{(d)} = f(W_3^{(d)} h_{m-1}^{(d)} + W_2^{(d)} w'_{m-1} + W_1^{(d)} z_m)$ where
    - f is activation function
    - w'_{m-1} is target token embedding at time step $m-1$ in target sequence
    - $h_{m-1}^{(d)}$ is decoder hidden state from previous time step with $h_0^{(d)} = h_N^{(e)}$, i.e. last encoder output is first decoder input
    - z_m is cross-attention

Runtime analysis:

- Let $W^{(e)}h$ be in $((N \times d) \times (d \times d))$ resp. $W^{(d)}h$ in $((M \times d) \times (d \times d))$
- Let number of encoder resp. decoder layers be l_e, l_d
- We perform $z_m = \sum_{n=1}^N \alpha_{m,n} h_n^{(e)}$ for M decoder time steps, summing over N encoder outputs $h_n^{(e)}$ of dimensionality d
- Encoder: $O(l_e N d^2)$ from hidden states
- Decoder: $O(l_d M d^2 + l_d dNM)$
  - $O(l_d M d^2)$ from hidden states
  - $O(l_d dNM)$ from cross-attention

Challenge: Sequential, cannot be parallelized

Solution: Transformers

Optimization

Parameters — Find parameters $\theta = W_1^{(e)}, W_2^{(e)}, W_1^{(d)}, W_2^{(d)}, W_3^{(d)}$

Objective function —

- Maximize log likelihood

Optimization —

- Perform forward pass
- Perform backpropagation
- Gradient with regard to encoder output:
  - $\nabla_{h_1^{(e)}} L = \frac{\partial L}{\partial h_m^{(d)}}) \frac{\partial h_m^{(d)}}{\partial h_n^{(e)}}$
  - $\frac{\partial h_m^{(d)}}{\partial h_n^{(e)}} = \frac{\partial}{\partial h_n^{(e)}} W_3^{(d)} h_{m-1}^{(d)} \times \frac{\partial}{\partial h_n^{(e)}} W_1^{(d)} z_m \times \frac{\partial}{\partial h_n^{(e)}} f(W_3^{(d)} h_{m-1}^{(d)} + W_2^{(d)} w'_{m-1} + W_1^{(d)} z_m)$

* We can further decompose $\frac{\partial}{\partial \boldsymbol{h}_n^{(e)}} \boldsymbol{W}_1^{(d)} \boldsymbol{z}_m$:
  - $= \frac{\partial}{\partial \boldsymbol{h}_n^{(e)}} \alpha_{m,n} \boldsymbol{h}_n^{(e)} + \frac{\partial}{\partial \boldsymbol{h}_n^{(e)}} \sum_{i \neq n}^{N} \alpha_{m,i} \boldsymbol{h}_i^{(e)}$
    $= \alpha_{m,n} + \frac{\partial}{\partial \boldsymbol{h}_n^{(e)}} \alpha_{m,n} \boldsymbol{h}_n^{(e)} + \frac{\partial}{\partial \boldsymbol{h}_n^{(e)}} \sum_{i \neq n}^{N} \alpha_{m,i} \boldsymbol{h}_i^{(e)}$ due to product rule for $\alpha_{m,n} \boldsymbol{h}_n^{(e)}$
    $= \Phi_{m,n} + \Phi'_{m,n} \boldsymbol{h}_n^{(e)} + \sum_{i \neq n}^{N} \left[\Phi_{m,i} \frac{\partial \boldsymbol{h}_i^{(e)}}{\partial \boldsymbol{h}_n^{(e)}} + \Phi'_{m,i} \boldsymbol{h}_i^{(e)} \right]$ due to product rule for $\alpha_{m,i} \boldsymbol{h}_i^{(e)}$ and by replacing $\alpha_{m,n}$ with $\Phi_{m,n}$
* Runtime analysis:
  - $\frac{\partial}{\partial \boldsymbol{h}_n^{(e)}} \boldsymbol{W}_3^{(d)} \boldsymbol{h}_{m-1}^{(d)}$ is in $O(m \times N + N - n)$:
    1) Taking derivatives of m decoder steps, due to attention z_m which is applied over N encoder outputs, takes $O(m \times N)$
    2) Taking derivatives of $N - n$ encoder steps takes $O(N - n)$
  - $\frac{\partial}{\partial \boldsymbol{h}_n^{(e)}} \boldsymbol{W}_1^{(d)} \boldsymbol{z}_m$ is in $O(N)$:
    1) $\Phi_{m,n}$ and $\Phi'_{m,n}$ are in $O(1)$, since they don't contain $\boldsymbol{h}_n^{(e)}$
    2) $\sum_{i \neq n}^{N} \Phi_{m,i} \frac{\partial \boldsymbol{h}_i^{(e)}}{\partial \boldsymbol{h}_n^{(e)}}$ is in $O(N - n)$ if we reuse terms in chain rule by factorizing, since the derivative is only non-null for $N - n$ encoder steps
    3) $\sum_{i \neq n}^{N} \Phi'_{m,i} \boldsymbol{h}_i^{(e)}$ is in $O(N)$, since here we're summing over all N encoder steps
  - $\frac{\partial}{\partial \boldsymbol{h}_n^{(e)}} f(...)$ is in $O(N)$:
    1) $\boldsymbol{W}_3^{(d)} \boldsymbol{h}_{m-1}^{(d)}$ and $\boldsymbol{W}_2^{(d)} w'_{m-1}$ are in $O(1)$, since they don't contain $\boldsymbol{h}_n^{(e)}$
    2) $\boldsymbol{W}_1^{(d)} \boldsymbol{z}_m$ is in $O(N)$, since it represents the attention applied over N encoder outputs
- Perform gradient descent

21 Encoder Decoder Transformers

Formulation

Formulation — Model architecture:
- Inputs fed into encoder in reverse order
- *Encoder*:
  * Sequence-to-vector
  * Takes in input sequence token embeddings (semantic vector, X in $(N \times d_{model})$) and positional embeddings (sinusoidal pointer vector for word position, given that model is not sequential, P in $(N \times d_{model})$) and adds them: $\boldsymbol{H}_0^{(e)} = \boldsymbol{X} + \boldsymbol{P}$
  * Multi-head self-attention, applied to all tokens jointly, where $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$ are tokens in input sequence:
    - $\boldsymbol{Q} = \boldsymbol{H}_{(l-1)}^{(e)} \boldsymbol{W}_q$
    - $\boldsymbol{K} = \boldsymbol{H}_{(l-1)}^{(e)} \boldsymbol{W}_k$
    - $\boldsymbol{V} = \boldsymbol{H}_{(l-1)}^{(e)} \boldsymbol{W}_v$
    - $\text{Attention} \boldsymbol{Z} = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}} \right) \boldsymbol{V}$
    - $\text{Multi Head Attention} \boldsymbol{Z} = \text{Concat}(\boldsymbol{Z}_{\text{head}_1}, \ldots, \boldsymbol{Z}_{\text{head}_h}) \boldsymbol{W}_O + \boldsymbol{b}_O$
  * Addition and normalization: Skip connections (from token + positional embeddings) added back and normalized:
    $\boldsymbol{H}_l^{(e)} = \text{Layer Norm}(\text{Multi Head Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) + \boldsymbol{H}_{(l-1)}^{(e)})$
  * Feed-forward network, parallel for each token:
    $\text{FFN}(\boldsymbol{H}_l^{(e)}) = \text{ReLU}(\boldsymbol{H}_l^{(e)} \boldsymbol{W}_1 + \boldsymbol{b}_1) \boldsymbol{W}_2 + \boldsymbol{b}_2$ where
    - $\boldsymbol{W}_1 \in \mathbb{R}^{(d_v \times r)}$
    - $\boldsymbol{b}_1 \in \mathbb{R}^{(1 \times r)}$
    - $\boldsymbol{W}_2 \in \mathbb{R}^{(r \times d_v)}$
    - $\boldsymbol{b}_2 \in \mathbb{R}^{(1 \times d_v)}$
  * Addition and normalization: Skip connections (from first addition and normalization) added back and normalized:
    $\boldsymbol{H}_l^{(e)} = \text{Layer Norm}(\text{FFN}(\boldsymbol{H}_l^{(e)}) + \boldsymbol{H}_l^{(e)})$
  * Generates hidden states $\boldsymbol{h}_n^{(e)}$
- *Decoder*:
  * Vector-to-sequence
  * Target sequence inputs are fed into decoder with one time step lag (masked self-attention):
    - Takes in target sequence token embeddings (semantic vector, Y in $(M \times d_{model})$) and positional embeddings (sinusoidal pointer vector for word position, given that model is not sequential, P in $(M \times d_{model})$) and adds them:
      $\boldsymbol{H}_0^{(d)} = \boldsymbol{Y} + \boldsymbol{P}$
    - Masked self-attention, applied to all tokens jointly, where $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$ are tokens in target sequence: $\boldsymbol{Q} = \boldsymbol{H}_{(l-1)}^{(d)} \boldsymbol{W}_q$
      $\boldsymbol{K} = \boldsymbol{H}_{(l-1)}^{(d)} \boldsymbol{W}_k$
      $\boldsymbol{V} = \boldsymbol{H}_{(l-1)}^{(d)} \boldsymbol{W}_v$
      $\text{Masked Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}} + \text{mask} \right) \boldsymbol{V}$
      where mask covers tokens in positions $m \geq t$
    - Addition and normalization: Skip connections (from token + positional embeddings) added back and normalized:
      $\boldsymbol{H}_l^{(d)} = \text{Layer Norm}(\text{Masked Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) + \boldsymbol{H}_{(l-1)}^{(d)})$
  * Encoder outputs are fed into decoder with cross-attention:
    - Cross-attention: $\boldsymbol{Q} = \boldsymbol{H}_l^{(d)} \boldsymbol{W}_q$
      $\boldsymbol{K} = \boldsymbol{H}_{(N)}^{(e)} \boldsymbol{W}_k$
      $\boldsymbol{V} = \boldsymbol{H}_{(N)}^{(e)} \boldsymbol{W}_v$
      $\text{Cross Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}} \right) \boldsymbol{V}$
    - Addition and normalization: Skip connections (from first addition and normalization) added back and normalized:
      $\boldsymbol{H}_l^{(d)} = \text{Layer Norm}(\text{Cross Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) + \boldsymbol{H}_l^{(d)})$
  * Feed-forward network, parallel for each token:
    $\text{FFN}(\boldsymbol{H}_l^{(d)}) = \text{ReLU}(\boldsymbol{H}_l^{(d)} \boldsymbol{W}_1 + \boldsymbol{b}_1) \boldsymbol{W}_2 + \boldsymbol{b}_2$ where
    - $\boldsymbol{W}_1 \in \mathbb{R}^{(d_v \times r)}$
    - $\boldsymbol{b}_1 \in \mathbb{R}^{(1 \times r)}$
    - $\boldsymbol{W}_2 \in \mathbb{R}^{(r \times d_v)}$
    - $\boldsymbol{b}_2 \in \mathbb{R}^{(1 \times d_v)}$
  * Addition and normalization: Skip connections (from second addition and normalization) added back and normalized:
    $\boldsymbol{H}_l^{(d)} = \text{Layer Norm}(\text{FFN}(\boldsymbol{H}_l^{(d)}) + \boldsymbol{H}_l^{(d)})$
  * Generates hidden states $\boldsymbol{h}_m^{(d)}$
- Linear layer applied to $\boldsymbol{h}_M^{(d)}$
- Softmax layer applied to select token with highest probability:
  * Neural networks make no independence assumption, i.e. output y_t is conditioned on entire history (non-Markovian structure: $x, y_{<t}$) rather than window of size n (Markovian structure: $x, \langle y_t, \ldots, y_{t-1} \rangle$)
  * This results in runtime of $O(|\Sigma|^n)$ rather than $O(|\Sigma| \times n)$
  * Since it is intractable to search for best sequence overall (explore), we turn to deterministic or stochastic variants (exploit):
    - *Greedy decoding*: Select highest-probability token at each step
    - *Beam search*: Keep n-highest-probability tokens in memory (beam size) and return k-most-likely sequences (top beams)
    - *Nucleus sampling*: Sample tokens from items that cover $p\%$ of PMF

Runtime analysis: Cross-attention:
- Computing Q with $O(m \times h \times d_k)$, K with $O(n \times h \times d_k)$, V with $O(n \times h \times d_v)$
- Assume $m = n$ and $d_k = d_v = d$
- Computing A with $O(m \times d_k \times n)$, computing Z with $O(m \times d_k \times n \times d_v)$
- Assume $m = 1$ (for one specific query) and $d_v = d$
- Total runtime for single layer: $O(n \times h \times d + h \times n \times d)$

Advantages vs. RNNs:
- Relies on attention to obtain a fixed-size representation of a sequence
- Allows to learn longer-range dependencies
- Allows for parallelization

22 Connection CNN and Multi Head Self Attention

Theorem: A multi-head self-attention layer operating on K^2 heads of dimension n and output dimension d_v, employing a relative positional encoding of dimension $d_p \geq 3$, can express any convolutional layer of kernel size $K \times K$ and d_v output channels

Theorem part 1:
- Given a multi-head self-attention layer with $n_{heads} = K^2$ and $n \geq d_v$
- Given a convolutional layer with a $K \times K$ kernel and d_v output channels
- Let $f : [n_{heads}] \to \Delta_K$ be a bijective map between heads and shifts
- Assume $\text{softmax}(A) = \begin{cases} 1 & \text{if } f(h) = \boldsymbol{q} - \boldsymbol{k} = \delta \\ 0 & \text{otherwise} \end{cases}$
- Then, for any convolutional layer, there exists a corresponding weight per head W_v such that the multi-head self-attention equals the convolution
- Proof:
  * Contribution of each head in multi-head self-attention is given by: $W = W_v W_{\text{out}}^{(h)}$ where $W_{\text{out}}^{(h)}$ is the portion of W_{out} associated with head h
  * This means, we can rewrite $\text{Multi Head Attention} \boldsymbol{Z} = \sum_{h \in n_{heads}} \text{softmax}(\boldsymbol{A}^{(h)}) \boldsymbol{Z} \boldsymbol{W}^{(h)} + \boldsymbol{b}_O$
  * This matches $\text{Convolution } \boldsymbol{Z} = \sum_{(u,v) \in \Delta_K} \boldsymbol{X}_{i',j'} \boldsymbol{W}_{u,v} + \boldsymbol{b}$

Theorem part 2:

- It is possible to construct a relative encoding scheme r_δ using parameters W_q, W_k, $\widetilde{W}_k$, and u so that, for every shift $\in \Delta_K$, there exists a vector v that yields the mapping $f : [n_{heads}] \to \Delta_K$
- Assume $A = -\alpha\left(\|\delta - \Delta\|^2 + c\right)$
- Behavior for $\delta = \Delta$ resp. $\delta \neq \Delta$:
  - Softmax is given by: $\text{softmax}(A) = \frac{\exp(-\alpha(\|\delta-\Delta\|^2+c))}{\sum_{k'}\exp(-\alpha(\|\delta'-\Delta\|^2+c))}$
  - In numerator:
    - If $\delta = \Delta$, $\exp(A) = \exp(-\alpha c)$
    - If $\delta \neq \Delta$, $\exp(A) \to 0$ as $\alpha \to \infty$, since entire term inside exponent grows very negative
  - In denominator: $\exp(A) \to \exp(-\alpha c)$ as $\alpha \to \infty$, since only the term corresponding to $\delta = \Delta$ contributes significantly
  - Then,
    - If $\delta = \Delta$, softmax $\to 1$
    - If $\delta \neq \Delta$, softmax $\to 0$
  - This proves assumption in part 1 of theorem
- Constant c is given by $c = \max_{\delta \neq \Delta} \|\delta - \Delta\|^2$:
  - $A = -\alpha\left(\|\delta - \Delta\|^2 + c\right)$
  - To ensure proper softmax behavior $-\alpha c$ must dominate over $-\alpha\|\delta - \Delta\|^2$
  - Then, we require $\|\delta - \Delta\|^2 + c \gg 0$ for $\delta \neq \Delta$

23 Natural Language Processing (NLP) Basics

Formulation

Training data —
- $y = X\beta$
- E.g.: $\begin{bmatrix}\text{score for doc 1}\\ \dots \\ \text{score for doc n}\end{bmatrix}$ $= \begin{bmatrix}\text{feature 1 for doc 1}\dots\text{feature m for doc 1}\\ \dots \\ \text{feature 1 for doc n}\dots\text{feature m for doc n}\end{bmatrix}$ $\times \begin{bmatrix}\text{coefficient for feature 1}\\ \dots \\ \text{coefficient for feature m}\end{bmatrix}$
- Where feature j for doc i = $\boldsymbol{w}^{(i)} \cdot \boldsymbol{a}^{(i)}$ = [weight of word 1 in doc i...weight of word n in doc i] $\times$ $\begin{bmatrix}\text{attribute j of vocab word 1}\\ \dots \\ \text{attribute j of vocab word n}\end{bmatrix}$

Terminology —
- *Corpus*
- *Document*: Contained in corpus, constitutes one of n instances for model
- *Tokens*: Document split into preprocessed words, which are the tokens
- *Vocabulary*: Contains unique tokens in corpus
- *Alphabet*: Contains unique symbols, of which tokens are composed
- Token-level features:
  - *One-hot encoding*: Vector of length of vocabulary, with 1 at index of token and 0 everywhere else
  - *Embeddings*: Measure semantic and syntactic word similarities in higher dimensional space
- Document-level features: Generated by pooling token-level features
- *Pooling methods*:
  - Sum pooling: $e(\boldsymbol{d}) = \sum_{t \in d} e(\boldsymbol{t})$
  - Mean pooling: $e(\boldsymbol{d}) = \frac{1}{|d|}\sum_{t \in d} e(\boldsymbol{t})$ with token weights:
    - *One-hot encoding*: e.g., $[0, 1, 0]$
    - *Bag-of-words*: e.g., $[1, 2, 0]$
    - *TF-IDF*: Bag-of-words counts given by: $\frac{\text{vocab word frequency in document}}{\text{vocab word frequency in corpus}}$
  - Max pooling: $e(\boldsymbol{d}) = \max_{t \in d} e(\boldsymbol{t})$

24 Skip Grams

Description

Task —
- Predict context word given center word
- Generate word embeddings

Formulation

- Vocabulary $\mathcal{V}$ with words w, for which we wish to create embeddings by considering the T preceding and following words of a center word
- Start by processing each document into a set $\mathcal{D}$ of $\mathcal{O}(T \times |\mathcal{V}|)$ pairs of center and context words: $\{(w_i, w_t)\}$ where $w_i \in w$ is center word, $w_t \in w'$ is context word

Optimization

Objective function —
- Bilinear softmax function
- *Bilinear model* is linear, if other variables are held constant, in this case e_{wrd} or e_{ctx}
- Likelihood: $\prod_{(w_i,w_t)\in\mathcal{D}} p(w_t \mid w_i) = \prod_{(w_i,w_t)\in\mathcal{D}} \frac{\exp(e_{\text{wrd}}(w_i)\cdot e_{\text{ctx}}(w_t))}{Z(w_i)} = \prod_{(w_i,w_t)\in\mathcal{D}} \frac{\exp(e_{\text{wrd}}(w_i)\cdot e_{\text{ctx}}(w_t))}{\sum_{w'\in\mathcal{V}}\exp(e_{\text{wrd}}(w_i)\cdot e_{\text{ctx}}(w'))}$ where
  - $e_{\text{wrd}}(w_i) \cdot e_{\text{ctx}}(w_t) = (\boldsymbol{w} \cdot \boldsymbol{e}_{oh}(w_i)) \cdot (\boldsymbol{w} \cdot \boldsymbol{e}_{oh}(w_t))$ where $\boldsymbol{e}_{oh}(w)$ is the one hot encoding
- Log likelihood: $\sum_{(w_i,w_t)\in\mathcal{D}} \log(p(w_t \mid w_i)) = e_{\text{wrd}}(w_i) \cdot e_{\text{ctx}}(w_t) - \log(Z(w_i)) =$
- Challenge: $Z(w_i)$ has $2|\mathcal{V}|$ parameters
- Solution: Use *negative sampling*:
  - For each pair (w_i, w_t), we randomly sample with replacement a set $\mathcal{C}^-$ from $\mathcal{V}$
  - We compute sigmoid, instead of softmax
  - Then we have: $\sum_{(w_i,w_t,\mathcal{C}^-)}(\log(p(w_t \mid w_i)) + \sum_{w^-\in\mathcal{C}^-}\log(1 - p(w^- \mid w_i)))$
  - $= \sum_{(w_i,w_t,\mathcal{C}^-)} \frac{1}{1+\exp(-e_{\text{wrd}}(w_i)\cdot e_{\text{ctx}}(w_t))} + \sum_{w^-\in\mathcal{C}^-}\log(1 - p(w^- \mid w_i))$
  - Here:
    - $\log(p(w_t \mid w_i))$ is computed with sigmoid, instead of softmax
    - $\sum_{w_i^-\in\mathcal{C}^-}\log(1 - p(w^- \mid w_i))$ replaces $-\log(Z(w_i))$

Optimization —
- $argmax_{w,w'}$ log likelihood
- Gradient descent
- We usually use center word embeddings w and throw away context word embeddings w'

25 Language Models

Background

- Alphabet Σ
- String y over an alphabet
- Empty string ε
- Set of all strings given by Kleene closure Σ^*
- BOS: Beginning of sequence token
- EOS: End of sequence token
- Vocabulary $\mathcal{V}$ resp. $\overline{\mathcal{V}}$ if incl. EOS

Description

Task —
- Assign a probability to y, i.e., fit a probability distribution over all Σ^*

Formulation

Globally normalized language model:
- Compute the probability of sentence y and normalize over all $y \in \Sigma^*$
- Challenge: Since Σ^* is infinite, Z is infinite
- Solution: Locally normalized language model

Locally normalized language model:
- Probability of a word y_t in a sentence y is conditioned only on the preceding context
- Compute the probability of sentence y and normalize over all possible words in the vocabulary: $p(\boldsymbol{y}) = p(y_1 \mid \text{BOS}) \times p(y_2 \mid \text{BOS } y_1) \times \dots \times p(y_N \mid \boldsymbol{y}_{<N}) \times p(\text{EOS} \mid \boldsymbol{y})$
- Can be visualized as a *prefix tree*:
  - Probabilities of all children nodes, given their parent node, sum to 1
  - All nodes have EOS as a child to ensure that each of the (possibly infinitely many) paths has a finite length
    Proof:
    - If we forget EOS, $\sum_{w\in\Sigma^*} p(w) = \infty$
    - To see this, let:
    - The probability of a string w of length M be given by: $p(w) = \prod_{m=1}^{M} p(w_m)$ with $\sum_{w\in\Sigma} p(w) = 1$ for unigram model
    - Strings be drawn from $\Sigma^* \bigcup_{M=0}^{\infty} \Sigma^M$
    - We can show that: $\sum_{w\in\Sigma^*} p(w) = \sum_{M=0}^{\infty}\sum_{w\in\Sigma^M} p(w) = \sum_{M=0}^{\infty}(\sum_{w\in\Sigma} p(w))^M = \sum_{M=0}^{\infty} 1^M = \infty$
    - If we include EOS, $\sum_{w\in\Sigma^*} p(\text{EOS}) \times p(w) = p(\text{EOS}) \times \sum_{w\in\Sigma^*} p(w) = 1$ if $\text{EOS} = \frac{1}{\sum_{w\in\Sigma^*} p(w)}$

Tight models:
- If the language model is locally normalized, sums to 1, and has finite length paths
- To enforce tightness: $p(\text{EOS} \mid \cdots) > \delta > 0$
  Proof:
  - Assume we want the probability sentences of less than length n to be $(1 - \delta)$
  - Then, the probability of a sentence of length n is δ
  - The probability over all sentences is then given by: $\sum_{n=1}^{\infty}(1-\delta)^{n-1}\delta = \frac{\delta}{1-(1-\delta)} = 1$ because it forms a geometric series

Other

Using CFGs as Language Models *Prefix probability*:
- To model the probability of a string that exactly matches a sequence, we need a notion of EOS
- However, sometimes it's desirable to model the probability of a string that begins with or equals a sequence (prefix w), but may also continue after the sequence (suffix $\boldsymbol{u}$)
- We can show that $p_{pre}(\mathbf{w}) = \sum_{u\in\Sigma^*} p(\boldsymbol{wu})$
  Proof:
  - $p(\boldsymbol{wu}) = \prod_{n=1}^{N} p(w_n \mid w_0 \cdots w_{n-1}) \times \prod_{n=N+1}^{N+|u|} p(u_{n-N} \mid \boldsymbol{w}\ u_1 \cdots u_{n-N-1})$
  - $\sum_{u\in\Sigma^*} p(\boldsymbol{wu}) = \prod_{n=1}^{N} p(w_n \mid w_0 \cdots w_{n-1}) \times \sum_{u\in\Sigma^*}\prod_{n=N+1}^{N+|u|} p(u_{n-N} \mid \boldsymbol{w}, u_1 \cdots u_{n-N-1}) = \prod_{n=1}^{N} p(w_n \mid w_0 \cdots w_{n-1}) \times 1$ given that model is locally normalized
  - Thus, $\sum_{u\in\Sigma^*} p(wu) = p(\mathbf{w})$
- We can show that $p(w_N | w_1, \dots, w_{N-1}) = \frac{p_{pre}(\mathbf{w})}{p_{pre}(\mathbf{w}_{1:N-1})}$

Proof:
  * $p(\mathbf{w}) = p(w_N|w_1,...,w_{N-1}) \times p(w_1,...,w_{N-1})$
  * Thus, $p(w_N|w_1,...,w_{N-1}) = \frac{p(\mathbf{w})}{p(\mathbf{w}_{1:N-1})}$

Conditional prefix probability:
- $p_{pre}(\mathbf{w}|X) = \sum_{u \in \Sigma^*} p(X \overset{*}{\Rightarrow} \boldsymbol{wu})$

Left-corner probability $p_{\text{lc}}(Y \mid X)$:
- Denotes probability of reaching Y (or sibling leaves, e.g. YZ) along the left corner of a tree rooted at X
- $p_{\text{lc}}(Y \mid X) = \sum_{\alpha \in \mathcal{N}^*} p(X \overset{*}{\Rightarrow} Y\alpha) = \sum_{\alpha \in \mathcal{N}^*} \sum_{t \in \mathcal{T}_X(Y\alpha)} \prod_{X' \to \alpha'} p(X' \to \alpha')$
- Can be calculated using Lehmann's algorithm: Represent left-corner probability as matrix, where each column and row corresponds to a non-terminal X, Y and each entry represents the probability of transitioning between the non-terminals
  1. Set $p_{\text{lc}}[X,Y] = 0$
  2. For each production $X \to Y\beta$, set: $p_{\text{lc}}[X,Y] \mathrel{+}= p(X \to Y\beta)$
  3. Update until convergence: $p_{\text{lc}}[X,Y] = \sum_{X' \to \alpha} p_{\text{lc}}[X,Y] + \sum_Z p_{\text{lc}}[X,Z] \times p_{\text{lc}}[Z,Y]$
- Runtime: $O(|\mathcal{N}|^4)$ where $|\mathcal{N}|$ is number of non-terminals

Inside probability $\beta(i,X,k)$:
- Denotes probability of non-terminal X between position i and k in string
- $\beta(i,X,k) = p(X \overset{*}{\Rightarrow} w_i,...,w_k) = \sum_{t \in \mathcal{T}_X(w_i,...,w_k)} p(t)$

On this basis, we can show that:
- $p_{pre}(w_i,...,w_k|X) = \sum_{j=i}^{k-1} \sum_{Y,Z \in \mathcal{N}} p_{\text{lc}}(YZ|X) \times \beta(i,Y,j) \times p_{pre}(w_{j+1},...,w_k|Z)$

  Proof:
  * Based on prefix probability, we can expand: $p_{pre}(w_i,...,w_k|X) = \sum_{u \in \Sigma^*} p(X \overset{*}{\Rightarrow} w_i,...,w_k\boldsymbol{u})$
  * Due to the CNF, the derivation $X \overset{*}{\Rightarrow} w_i,...,w_k\boldsymbol{u}$ must be of one of the following forms:
    · $X \to YZ$
    · $Y \overset{*}{\Rightarrow} w_i,...,w_j$
    · $Z \overset{*}{\Rightarrow} w_{j+1},...,w_k\boldsymbol{u}$
    · Here, j is a breaking point between $w_i,...,w_k$
  * Then, we can write: $p(X \overset{*}{\Rightarrow} w_i,...,w_k\boldsymbol{u}) = p_{\text{lc}}(YZ|X) \times \beta(i,Y,j) \times p_{pre}(w_{j+1},...,w_k|Z)$
  * We then sum over all possible breaking points j and all pairs of non-terminals Y, Z

On this basis, we can develop a CKY-like algorithm that computes the prefix probabilities for all prefixes of $\boldsymbol{w}$, i.e. w_1, w_1,w_2, ..., $w_1,...,w_N$:
- We can leverage the pre-computed left-corner probabilities via Lehmann's algorithm
  1. We first define a first chart $T[i,j,Y]$ that stores the inside probability $\beta(i,Y,j)$ that non-terminal X generates terminals $w_i,...,w_j$
  2. We initialize $T[i,j,Y] \leftarrow p(Y \to w_i)$
  3. We update: $T[i,j,Y] = \sum_{l=i}^{j-1} \sum_{Y,Z \in \mathcal{N}} p(X \to YZ) \times T[i,l,Y] \times T[l+1,j,Z]$
  4. We define a second chart $P[i,k,X]$ that stores the prefix probability $p_{pre}(w_i,...,w_k|X)$
  5. We initialize $P[i,k,X] \leftarrow T[i,j,Y]$
  6. We update: $P[i,k,X] = \sum_{j=i}^{k-1} \sum_{Y,Z \in \mathcal{N}} p_{\text{lc}}(YZ|X) \times T[i,j,Y] \times P[j+1,k,Z]$
  7. We finally marginalize: $p(w_i,...,w_k) = \sum_{X \in \mathcal{N}} P[1,k,X]$
- Runtime: $O(N^3|\mathcal{N}|^3 + |\mathcal{N}|^4)$ where $|\mathcal{N}|$ is number of non-terminals and N is length of string, $N^3|\mathcal{N}|^3$ is from the CKY-like algorithm for the inside and prefix probabilities and $|\mathcal{N}|^4$ is from Lehmann's algorithm for the left-corner probabilities

1) Prefix tree

26 N Gram Models

Description

Task — Predict the next word in a sequence, based on a locally normalized language model

Formulation

- Naively, probability of a sentence: $p(\boldsymbol{w}) = p(w_1) \times p(w_2 \mid w_1) \times p(w_3 \mid w_1,w_2) \times \cdots \times p(w_N \mid \boldsymbol{w}_{<N}) \times p(\text{EOS} \mid \boldsymbol{w})$
- Challenge: Probabilities for long sequences are likely to be 0, given the low chance of exactly such a sequence appearing in the sentence
- *Markov assumption*: Not all words prior to the target word matter in predicting it, only the last $N-1$ words (history) are considered
- *N-gram*:
  * Sequence of n words as they appear in the sentence
  * Denoted by w_j^i, where i and j refer to words in the sentence
  * If the vocabulary has $|\mathcal{V}|$ words, there are $|\mathcal{V}|$ (distinct) unigrams, $|\mathcal{V}|^2$ bigrams, ..., $|\mathcal{V}|^n$ N-grams, ...
  * If the sentence (or corpus) has M words, there are M (potentially overlapping) unigrams, $M-1$ bigrams, ..., $M-n+1$ N-grams, ...
  * N-gram counts are given by all distinct N-grams and how often they appear in the sentence (or corpus)
  * N-gram probability: $p(w_t \mid w_{t-n+1},\dots,w_{t-1})$
- Under Markov assumption and N-gram, probability of a sentence: $p(\boldsymbol{w}) = \left(\prod_{t=n}^{N} p(w_t \mid w_{t-n+1},\dots,w_{t-1})\right) \times p(\text{EOS} \mid w_{N-n+1},\dots,w_N)$ with appropriate BOS padding for the first $n-1$ conditional probabilities: $w_0 = w_{-1} = w_{-n+2} = \text{BOS}$
  Note: For unigrams, we don't need BOS
- Under this model, we have
  * $|\mathcal{V}|^{n-1} + |\mathcal{V}|^{n-2} + \cdots + |\mathcal{V}| + 1 = \sum_{i=0}^{n-1} |\mathcal{V}|^i$ conditional probabilities for w_t, where
    · $|\mathcal{V}|^{n-1}$ applies if the context is fully occupied with words
    · 1 applies if the context only contains BOS, i.e. w_t is the first token
  * $|\overline{\mathcal{V}}| - 1$ free parameters for each conditional probability, i.e., $|\overline{\mathcal{V}}|^{n-1} \sum_{i=0}^{n-1} |\mathcal{V}|^i$ in total, where $\overline{\mathcal{V}} = \mathcal{V} \cup \{\text{EOS}\}$

Optimization

Via relative frequency counts or via neural nets with log-linear model

Relative frequency counts —
- $p(w_t \mid w_{t-n+1},\dots,w_{t-1}) = \frac{\text{count}(w_{t-n+1},...,w_{t-1},w_t)}{\text{count}(w_{t-n+1},...,w_{t-1})}$
  Note: In denominator, only count N-grams where the correct sequence appears in the correct position 1 to $n-1$
- Corresponds to MLE estimate, if we assume each token is sampled independently from a categorical distribution over the vocabulary TBA
- Dealing with unseen words or word combinations resulting in probability 0:
  * How likely is this?
    · Given a sentence with $M+1$ words and vocabulary $\mathcal{V}$
    · In the sentence, there are M bigrams, but from the vocabulary, we can construct $|\mathcal{V}|^2$ bigrams
    · Probability of a given bigram at index m in the sentence: $\frac{1}{|\mathcal{V}|^2}$
    · Thus, the probability that a given bigram does not appear at index m is: $\left(1 - \frac{1}{|\mathcal{V}|^2}\right)$
    · Thus, the probability that a given bigram does not appear in any of the M spots is: $\left(1 - \frac{1}{|\mathcal{V}|^2}\right)^M$
    · If we seek to bound the likelihood of OOV bigrams with an upper limit ϵ, we have: $\epsilon \geq \left(1 - \frac{1}{|\mathcal{V}|^2}\right)^M$
      $\ln(\epsilon) \geq M \ln\left(1 - \frac{1}{|\mathcal{V}|^2}\right)$
      $\ln(\epsilon) \approx -M \frac{1}{|\mathcal{V}|^2}$ because $\ln(1+\lambda) \approx \lambda$ for $\lambda \approx 0$ for $\lambda \approx 0$
      $M \geq -\ln(\epsilon)|\mathcal{V}|^2 \approx |\mathcal{V}|^2 \ln\left(\frac{1}{\epsilon}\right)$
  * Remedies:
    · Consider creating an OOV category
    · Consider *additive Lidstone smoothing*:
    · Smooth the PMF of the vocabulary by moving weight from seen vocabulary to unseen vocabulary by adding pseudo counts to each word: $p(w_t \mid w_{t-n+1},\dots,w_{t-1}) = \frac{\text{count}(w_{t-n+1},...,w_{t-1},w_t)+\lambda}{\text{count}(w_{t-n+1},...,w_{t-1})+|\mathcal{V}|\lambda}$
    · If $\lambda = 1$, we call this *Laplace smoothing*
    · Corresponds to a MAP estimate if we assume a symmetric Dirichlet distribution with parameter λ as our prior distribution over the weights
    · Consider back-off approach:
    · Revert to lower-order N-grams that do not contain OOVs
    · $\hat{p}(w_t \mid w_{t-n+1},\dots,w_{t-1}) = \begin{cases} p^*(w_t \mid w_{t-n+1},\dots,w_{t-1}) \\ \quad \text{if count}(w_{t-n+1},\dots,w_{t-1}) > 0 \\ \alpha_{n-1}\, p(w_t \mid w_{t-(n-1)+1},\dots,w_{t-1}) \\ \quad \text{otherwise} \end{cases}$
    · Consider interpolation: Combine estimates from different N-gram orders: $p(w_t \mid w_{t-n+1},\dots,w_{t-1}) = \alpha_1 p(w_t) + \alpha_2 p(w_t \mid w_{t-1}) + \ldots + \alpha_n p(w_t \mid w_{t-n+1},\dots,w_{t-1})$

Neural nets —
- Advantage over frequency-count-based N-grams: Parameters are shared between contexts $w_{t-n+1},\dots,w_{t-1}$
- Log-linear model: $p(w_t \mid w_{t-n+1},\dots,w_{t-1}) = \frac{\exp(\mathbf{v}(w_t)\cdot\mathbf{h}_t)}{\sum_{w' \in \overline{\mathcal{V}}} \exp(\mathbf{v}(w')\cdot\mathbf{h}_t)}$
  where:
  * $\mathbf{v}(w)$ are word vectors
  * $\mathbf{h}_t$ are N-gram context vectors of the previous $n-1$ words
- E.g.
  * Inputs: $n-1$ words
  * Transformation: word embedding $\boldsymbol{c}(w)$
  * Concatenated vector of word embeddings: $\boldsymbol{c}(\boldsymbol{w}_{<3}) = [\boldsymbol{c}(w_1)\ \boldsymbol{c}(w_2)\ \boldsymbol{c}(w_3)]$
  * Hidden layer: $\boldsymbol{h}_1 = f(\boldsymbol{c}(\boldsymbol{w}_{<3}))$
  * Activation: Softmax: $p(w_i \mid \boldsymbol{w}_{<3}) = \frac{\exp(\boldsymbol{v}(w_i)\cdot\boldsymbol{h}_3)}{\sum_{w' \in \overline{\mathcal{V}}} \exp(\boldsymbol{v}(w')\cdot\boldsymbol{h}_3)}$
- $\boldsymbol{v}(w_i), \boldsymbol{h_i}, f$, and its parameters θ are learned by maximizing

log-likelihood of training dataset:

$\log p(w^{(i)}) = \sum_{w^{(i)} \in D} \log\left(\prod_{t=1}^{|w^{(i)}|} p(w_t^{(i)} \mid w_{t-1}^{(i)}, \ldots, w_{t-n+1}^{(i)})\right)$

$= \sum_{w^{(i)} \in D} \sum_{t=1}^{|w^{(i)}|} \log\left(p(w_t^{(i)} \mid w_{t-1}^{(i)}, \ldots, w_{t-n+1}^{(i)})\right)$

$= \sum_{w^{(i)} \in D} \sum_{t=1}^{|w^{(i)}|} \boldsymbol{v}(w_t) \cdot \boldsymbol{h_t} - \log(\sum_{w' \in \overline{V}} \exp(\boldsymbol{v}(w') \cdot \boldsymbol{h_t}))$

Evaluation

Intrinsic evaluation: *Perplexity*:

- Perplexity is the normalized inverse probability of test set: Given test sequence $(w_1, \ldots, w_N)$:
  $\text{perplexity}(w_1, \ldots, w_N) = p(w_1, \ldots, w_N)^{-\frac{1}{N}}$ where N is the number of words in the sentence and n is the N-gram order
  Proof:
  - $p(w_1, \ldots, w_N) = \prod_{i=1}^{N} p(w_i \mid w_{i-n+1}, \ldots, w_{i-1})$
  - Normalize by transforming to log and dividing by N: $\frac{\sum_{i=1}^{N} \log(p(w_i \mid \ldots))}{N}$
  - Remove log: $\left(\prod_{i=1}^{N} p(w_i \mid \ldots)\right)^{\frac{1}{N}}$
  - Take inverse: $\frac{1}{\left(\prod_{i=1}^{N} p(w_i \mid \ldots)\right)^{\frac{1}{N}}} = \left(\prod_{i=1}^{N} p(w_i \mid \ldots)\right)^{-\frac{1}{N}} = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{p(w_i \mid w_{i-n+1}, \ldots, w_{i-1})}}$

27 Part of Speech (POS) Tagging with Conditional Random Field (CRF)

Description

Task —

- Tag parts of speech, e.g., adverb, verb, noun, etc.

Formulation

- Input: Sequence of words w of length N
- Output: Sequence of tags $t \in T$ with $t_0 = BOS$ and $t_N = EOS$
- Can represented as *trellis*:
  - DAG, where nodes are divided into vertical slices and each slice is associated with a timestamp
  - Each node corresponds to a distinct state
  - Each arrow represents a transition to a new state in the next slice

Point of departure —

- Log-linear model: $p(t \mid w) = \frac{\exp(\text{score}(t,w))}{Z} = \frac{\exp(\text{score}(t,w))}{\sum_{t'} \exp(\text{score}(t',w))}$
- Challenge: Naively computing Z would take exponential time $\mathcal{O}(|T|^N)$
- Solution: Consider a scoring function that is additively decomposable over tag bigrams:
  $\text{score}(t, w) = \sum_{n=1}^{N} \text{score}(\langle t_{n-1}, t_n \rangle, w)$
- Then, we have: $p(t \mid w) = \frac{\exp(\sum_{n=1}^{N} \text{score}(\langle t_{n-1}, t_n \rangle, w))}{\sum_{t'} \exp(\sum_{n=1}^{N} \text{score}(\langle t'_{n-1}, t'_n \rangle, w))}$
- If we take the denominator:
  - $\sum_{t_1:t_N} \exp(\sum_{n=1}^{N} \ldots) = \sum_{t_1:t_N} \prod_{n=1}^{N} \exp(\ldots) = \sum_{t_1:t_{N-1}} \sum_{t_N} \prod_{n=1}^{N} \exp(\ldots)$ since last tag t_N only depends on t_{N-1}
  - $= \sum_{t_1:t_{N-1}} \prod_{n=1}^{N-1} \exp(\ldots) \sum_{t_N} \prod_{n=N}^{N} \exp(\text{score}(\langle t_{n-1}, t_n \rangle, w))$ based on distributivity of $\otimes$ over $\oplus$
  - $= \sum_{t_1:t_{N-1}} \prod_{n=1}^{N-1} \exp(\ldots) \sum_{t_N} \exp(\text{score}(\langle t_{N-1}, t_N \rangle, w))$
  - $= \sum_{t_1} [\exp(\text{score}(\langle t_0, t_1 \rangle, w)) \times \sum_{t_2} [\exp(\ldots) \times \cdots \times \sum_{t_N} \exp(\text{score}(\langle t_{N-1}, t_N \rangle, w))]]$
- We've moved from an exponential number of terms to a linear number of terms
- We go from having a score for each path in the trellis to a score for each edge

Backward algorithm —

- Algorithm for computing Z
  1. For t_{N-1} (all tags): $\beta(w, t_{N-1}) \leftarrow \exp(\text{score}(\langle t_{N-1}, \text{EOS} \rangle, w))$
     - Handles scores of edges incoming to EOS
     - β is memoized
  2. For $n \in N-2, \ldots, 1$:
     For $t_n \in T$ (all tags):
     $\beta(w, t_n) \leftarrow \sum_{t_{n+1}} \exp(\text{score}(\langle t_n, t_{n+1} \rangle, w)) \times \beta(w, t_{n+1})$
     - Handles scores over normal edges
     - β is memoized
  3. $\beta(w, t_0) \leftarrow \sum_{t_1} \exp(\text{score}(\langle \text{BOS}, t_1 \rangle, w)) \times \beta(w, t_1)$
     - Handles scores of edges outgoing from BOS
  4. Return $\beta(w, t_0)$
- $\beta(w, t_n)$ are *backward variables* that contain the sum of the scores of all paths starting at EOS and ending at tag t_n
- Therefore: $\text{denominator} = \beta(w, t_0) = \beta(w, \text{BOS})$
- Complexity:
  - Time complexity:
    - For scoring function over tag bigrams: $\mathcal{O}(N|T|^2)$, given that we compute $|T|N$ backward variables and for each of them, compute the sum over $|T|$ successor tags
    - For scoring function over tag trigrams: $\mathcal{O}(N|T|^3)$
    - For scoring function over tag N-grams of order n: $\mathcal{O}(N|T|^n)$
  - Space complexity: $\mathcal{O}(N|T|)$, since we have to keep $|T|N$ backward variables in memory
- Alternatively, the same can be achieved with a *forward algorithm*
  - Starting from BOS t_0 and going forward towards EOS t_N
  - Instead of looking at t_{n+1} we look at t_{n-1}
- In addition to probabilities, we can compute entropy of the CRF using the expectation semiring:
  - Instead of $\omega = \exp(\text{score}(\langle t_{n-1}, t_n \rangle, w))$, we do: $\omega = \langle \omega, -\omega \log(\omega) \rangle$
  - Running the backward algorithm with these weights and in the expectation semiring, we can additionally compute unnormalized entropy $H_u = -\sum_t \exp(\text{score}(t, w)) \times \text{score}(t, w)$ in the same runtime complexity:
    - We aim to show that the backward algorithm computes $\langle Z, H_u \rangle$
    - In the base case, $\langle Z, H_u \rangle = \langle \omega, -\omega \log(\omega) \rangle$
    - Let's call the adjusted backward variables β', instead of β
    - $\beta'(w, t_n) \leftarrow \bigoplus_t \langle \omega(t_n, t_{n+1}), -\omega(t_n, t_{n+1}) \log(\omega(t_n, t_{n+1})) \rangle \otimes \beta'(w, t_{n+1})$
    - Based on the definition of $\otimes$ under the expectation semiring, this is: $\beta'(w, t_n) \leftarrow \bigoplus_t \langle Z_{n-1} \times \omega(t_n, t_{n+1}), Z_{n-1} \times -\omega(t_n, t_{n+1}) \log(\omega(t_n, t_{n+1})) + H_{u,n-1} \times \omega(t_n, t_{n+1}) \rangle = \langle Z_{n-1} \times \omega(t_n, t_{n+1}), [Z_{n-1} \times -\log(\omega(t_n, t_{n+1})) + H_{u,n-1}] \times \omega(t_n, t_{n+1}) \rangle$
    - Then:
    - $Z_{n-1} = \sum_t \prod_{i=1}^{n-1} \omega(t_n, t_{n+1}) = \sum_t \exp(\text{score}(t, w))$
    - $H_{u,n-1} = \sum_t [\prod_{i=1}^{n-1} \omega(t_n, t_{n+1})][\sum_{i=1}^{n-1} -\log(\omega(t_n, t_{n+1}))] = -\sum_t [\prod_{i=1}^{n-1} \omega(t_n, t_{n+1})][\log(\prod_{i=1}^{n-1} \omega(t_n, t_{n+1}))] = -\sum_t \exp(\text{score}(t, w)) \times \text{score}(t, w)$
  - We can then compute the normalized entropy H_n:
    - $H_n = -\sum_t p(t) \log(p(t))$
    - If we substitute $p(t) = \frac{1}{Z} \exp(\text{score}(t, w))$, we get: $H_n = -\sum_t \frac{1}{Z} \exp(\text{score}(t, w)) \log(\frac{1}{Z} \exp(\text{score}(t, w)))$
    - We can develop to: $H_n = -\sum_t \frac{1}{Z} \exp(\text{score}(t, w))(\text{score}(t, w) - \log(Z)) = -\sum_t \frac{1}{Z} \exp(\text{score}(t, w)) \text{score}(t, w) - \frac{1}{Z} \exp(\text{score}(t, w)) \log(Z) = \frac{1}{Z} \sum_t -[\exp(\text{score}(t, w)) \text{score}(t, w)] + \log(Z) \sum_t [\frac{1}{Z} \exp(\text{score}(t, w))] = \frac{1}{Z} \sum_t -[\exp(\text{score}(t, w)) \text{score}(t, w)] + \log(Z) \times 1 = Z^{-1} H_u + \log(Z)$

Viterbi algorithm —

- Algorithm for computing the score of the best tag sequence t^* and recovering the sequence itself
- For this, we can ignore the denominator: $p(t \mid w) \propto \exp(\text{score}(t, w))$
  1. For t_{N-1} (all tags):
     - $v(w, t_{N-1}) \leftarrow \exp(\text{score}(\langle t_{N-1}, \text{EOS} \rangle, w))$
     - $b(t_{N-1}) \leftarrow \text{EOS}$ since t_N can only be EOS
  2. For $n \in N-2, \ldots, 1$:
     For $t_n \in T$ (all tags):
     - $v(w, t_n) \leftarrow \max_{t_{n+1}} [\exp(\text{score}(\langle t_n, t_{n+1} \rangle, w)) \times v(w, t_{n+1})]$ for each tag t_n, stores the score of the next best tag t_{n+1}
     - $b(t_n) \leftarrow argmax_{t_{n+1}} [\exp(\text{score}(\langle t_n, t_{n+1} \rangle, w)) \times v(w, t_{n+1})]$ for each tag t_n, stores the tag t_{n+1} that gave the best score
  3. Finally:
     - $v(w, t_0) \leftarrow \max_{t_1} [\exp(\text{score}(\langle t_0, t_1 \rangle, w)) \times v(w, t_1)]$
     - $b(t_0) \leftarrow argmax_{t_1} [\exp(\text{score}(\langle t_0, t_1 \rangle, w)) \times v(w, t_1)]$
  4. For $n = 1, \ldots, N$: Recover the best sequence using backpointers, by always plugging in next t $t_n \leftarrow b(t_{n-1})$
     - $t_1 = b(\text{BOS})$
     - $t_2 = b(t_1)$
     - ...
  5. Return $t_{1:N}$ and $v(w, t_0)$
- $b(t_n)$ are *backpointers* that point to the t_{n+1} tag in t^*
- $v(w, t_n)$ are *Viterbi variables* that contain the score of t^* starting at EOS and ending with tag t_n
- Complexity:
  - For backpointers: $\mathcal{O}(N)$
  - For scoring function over tag N-grams of order n: $\mathcal{O}(N|T|^n)$
  - For higher-order N-grams, we apply restriction that transitions must be valid (e.g. in $t_1 \to t_2, t_3 \to t_4$ t_2 must equal $t3$)
- Alternatively, the same can be achieved with a *forward Viterbi algorithm*
  - Starting from BOS t_0 and going forward towards EOS t_N
  - Instead of looking at t_{n+1} we look at t_{n-1}
  - Backpointers point to the t_{n-1} tag in t^*, i.e. $t_{n-1} \leftarrow b(t_n)$, starting with $t_{N-1} \leftarrow b(\text{EOS}), t_{N-2} \leftarrow b(t_{N-1}), \ldots$

Dijkstra's algorithm —

- Alternative to the Viterbi algorithm:
  - First complete tagging popped from the queue is the score for the best tagging, same as what Viterbi computes
  - If we keep running Dijkstra's algorithm until queue is empty (i.e. if we don't leverage early stopping), Dijkstra's algorithm computes the same values γ as Viterbi
- Runs over time step - tag nodes $\langle n, t \rangle$
- Uses *Popped* to keep track of nodes $\langle n, t \rangle$ that have been

processed
- Uses *PriorityQueue*:
  - Contains node : score pairs $\langle\langle n,t\rangle,\text{score}\rangle$
  - Returns pair with highest score first, regardless of time step or tag, thus, does not necessarily progress over nodes sequentially by time step, but progresses over nodes by descending score
  - If there is no score yet for key $\langle n,t\rangle$ in queue, it's inserted
  - If there is already a score for key $\langle n,t\rangle$ in queue, the higher score is retained: $\max_{\text{score}}\text{score}$
- Uses table γ to store the best score for each node $\langle n,t\rangle$: $\gamma[n,t]=\text{score}$
- Algorithm:
  1. Initialize:
     - Popped $\leftarrow\{\}$
     - Queue $\leftarrow$ Priority Queue()
     - $\gamma\leftarrow-\infty$
  2. Push $\langle\langle 0,\text{BOS}\rangle,0\rangle$ to queue
  3. While $|\text{queue}|>0$:
     (a) Pop $\langle\langle n,t\rangle,\text{score}\rangle$ from queue, add $\langle\langle n,t\rangle,\text{score}\rangle$ to popped, $\gamma[n,t]\leftarrow\text{score}$
     (b) If $n=|w|$ or stop early: Return score
     (c) If $n<|w|$:
     For $t'\in T$:
     If $\langle n+1,t'\rangle$ is not in popped: Push $\langle\langle n+1,t'\rangle,\text{score}(t,t',w)+\gamma[n,t]\rangle$ to queue
  4. Return γ
- Complexity:
  - For scoring function over tag bigrams: $\mathcal{O}(N|T|^2\log(N|T|))$, where $\log(N|T|)$ stems from push and update operations on the queue
  - Dijkstra's is generally slower than Viterbi, unless we leverage early stopping conditions
- Dijkstra's as an alternative to Viterbi requires a semiring which guarantees:
  - Idempotence of $\oplus$ such that algorithm progresses towards the optimal path without revisiting nodes
  - Commutative and associative property of $\oplus$ such that score accumulation always yields same result, even if the nodes are processed in a different order
  - $\mathbb{R}_{\leq 0}$ since scores are log-probabilities, which are always non-positive, since probabilities are in $[0,1]$
- Dijkstra's could also be used as an alternative to the backward algorithm with a different semiring TBA

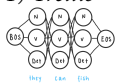
Generalized algorithm —
- A more general formulation of backward, Viterbi, and Dijkstra's algorithms using semirings
- Backward algorithm: Uses inside semiring
- Viterbi algorithm: Uses Viterbi semiring
- Challenge with this original formulation: Multiplying probabilities for long sequences may cause numbers to go to 0, leading to numerical underflow
- Solution: Revert to log-sum-exp semiring (backward algorithm) resp. arctic semiring (Viterbi algorithm)
- Dijkstra's algorithm: Uses arctic semiring

Scoring functions —
- So far, we only imposed the requirement that the scoring function is additively decomposable, but we have not specified it further
- *Hidden Markov Model*: $\text{score}(\langle t_{n-1},t_n\rangle,w)=\text{transition}(t_{n-1},t_n)+\text{emission}(t_n,w_n)=$ transition probability (tag-tag pairs) + emission probability (word-tag pairs) except for $t_N=\text{EOS}$, where we have no emission probability: $\text{score}(\langle t_{N-1},t_N\rangle,w)=\text{transition}(t_{N-1},t_N)+0$
- A more complex example of scoring functions is a neural network with inputs $\langle t_{n-1},t_n\rangle$ and w
- E.g. for the sentence 'The girl drinks':
  - $\text{score}(w=\text{The girl drinks},t=\langle D,N,V\rangle)=\sum_{n=1}^{N}\text{score}(\langle t_{n-1},t_n\rangle,w)$
  - $=\text{score}(\langle\text{BOS},D\rangle,w)+\text{score}(\langle D,N\rangle,w)+\text{score}(\langle N,V\rangle,w)+\text{score}(\langle V,\text{EOS}\rangle,w)$
  - Combination of emission and transition features: $(w_1=\text{The},y_1=D)+(y_1=D,y_0=\text{BOS})+(w_2=\text{girl},y_2=N)+(y_2=N,y_1=D)+(w_3=\text{drinks},y_3=V)+(y_3=V,y_2=N)+(y_4=\text{EOS},y_3=V)$

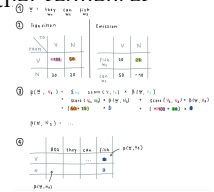
Optimization

Objective function —
- Assume we have a dataset of K data points $(w^{(i)},t^{(i)})$
- Log likelihood:
$$\sum_{k=1}^{K}\log(p(t^{(k)}\mid w^{(k)}))=\sum_{k=1}^{K}\log(\frac{\exp(\text{score}(t^{(k)},w^{(k)}))}{\sum_{t'}\exp(\text{score}(t',w^{(k)}))})=$$
$$\sum_{k=1}^{K}[\text{score}(t^{(k)},w^{(k)})-\log\sum_{t'}\exp(\text{score}(t',w^{(k)}))]$$
- Risk of overflow since sum makes things big and log makes things small
- If we want to use a temperature parameter T, we have:
$$\sum_{k=1}^{K}\log(p(t^{(k)}\mid w^{(k)}))=\sum_{k=1}^{K}\log(\frac{\exp(\text{score}(t^{(k)},w^{(k)})/T)}{\sum_{t'}\exp(\text{score}(t',w^{(k)})/T)})=$$
$$\sum_{k=1}^{K}[\frac{\text{score}(t^{(k)},w^{(k)})}{T}-\log\sum_{t'}\exp(\frac{\text{score}(t',w^{(k)})}{T})]$$
- As $T\to 0$:
  - The numerator $\exp(\text{score}(t^{(k)},w^{(k)})/T)$ becomes very large for high score sequences and very small for low score sequences
  - The denominator $\sum_{t'}\exp(\text{score}(t',w^{(k)})/T)$ becomes dominated by the sequence with the highest score
  - Then, the likelihood approaches 1 for the sequence with the highest score and 0 for all other sequences

1) Trellis



2) Backward algorithm



28 Syntax | Syntactic Parsing

Description

Task — Assign syntactic structure to a string by breaking it down into a hierarchy of constituents that is grammatically correct (similar to dependency parsing)

Terminology —
- *Constituent*: Sequence of words that function as a coherent unit resp. nodes in tree
  - *Terminals*: Words
  - *Non-terminals*: Abstractions over words (e.g. NP, VP, etc.)
- *Grammar*: Set of production rules, according to which strings can be formed from a vocabulary
  - *Context-free grammar (CFG)*: Grammar where rules are applied regardless of context
- *Parse tree*:
  - Represents both syntactic structure of a string and its derivation under grammar
  - Can be considered a bag of production rules and a multiset (set with repeats) since production rule can appear multiple times in the tree

Formulation

Context-free grammar $(\mathcal{N},\mathcal{S},\Sigma,\mathcal{R})$ —
- Set of non-terminals $\mathcal{N}=\{N_1,N_2,\ldots\}$
- Distinguished start non-terminals $\mathcal{S}$
- Set of terminals $\Sigma=\{a_1,a_2,\ldots\}$
- Set of production rules r of the form $N\to\alpha$, where N is non-terminal (incl. $\mathcal{S}$) and $\alpha\in(\mathcal{N}\cup\Sigma)^*$, i.e. α is ordered sequence of terminals or non-terminals
- String s of length M
- Empty string ε
- Set of all strings given by Kleene closure Σ^*
- $T(s)$ is set of trees t that yield s
- E.g.:
  - Language $\{a^ib^jc^k\mid i,j,k\geq 0,i+j=k\}$ is generated by: $\mathcal{S}\to a\mathcal{S}c\mid W$, $W\to bWc\mid\varepsilon$
  - Language $\{a^ib^jc^k\mid i,j,k\geq 0,i+k=j\}$ is generated by: $\mathcal{S}\to KW$, $K\to aKb\mid\varepsilon$, $W\to bWc\mid\varepsilon$
  - Language $\{w\in\{a,b\}^*\mid w\text{ contains min. }3a\}$ is generated by: $\mathcal{S}\to WaWaWaW$, $W\to aW\mid bW\mid\varepsilon$

Probabilistic CFGs — Motivation:
- Often, multiple syntactic structures are grammatically admissible, i.e. the string is ambiguous
- This is the case when a non-terminal can be produced by 2 rules

Probabilistic CFGs $(\mathcal{N},\mathcal{S},\Sigma,\mathcal{R},\mathcal{P})$:
- Additionally define a set of probabilities $\mathcal{P}$ for each production rule
- Probabilities are locally normalized over each transition: $\sum_k p(N\to\alpha_k)=1$ where $N\to\alpha_1,\ldots,N\to\alpha_k$ are expansions of node N
- E.g. $\text{NP}\to\text{VP}(p=0.6)\mid\text{Adj}(p=0.4)$ with total $p=1$
- Then, the probability of a parse tree is the multiplication of the probabilities of the rules used to create the tree: $p(t)=\prod_{r\in t}p(r)=p(S\to S_1S_2)^{M-1}\times p(S\to X)^M\times p(X\to\sigma)^M$ where:
  - $p(S\to S_1S_2)^{M-1}$: repeats the rule $S\to SS$ $M-1$ times to obtain M starting nodes
  - X: non-terminal
  - σ: terminal
- Probability of a string is the sum over all parse trees: $p(s)=p(S\overset{*}{\Rightarrow}s)=\sum_{t\in\mathcal{T}(s)}p(t)$ where $S\overset{*}{\Rightarrow}s$ is sequence of all production rules associated with going from start non-terminal to the string

Chomsky Normal Form (CNF) —
- CFG, where production rules follow specific structure:
  - $N_1\to N_2N_3$ are non-terminal productions
  - $N\to a$ are terminal productions
  - $S\to\varepsilon$ are productions generating an empty string (only if the language contains the empty string)
- Prohibits cyclic rules (e.g. $N\to N$)
- Transform CFG to CNF:
  - Remove null productions $A\to\varepsilon$ by modifying and adding replacements, e.g. if CFG contains $S\to AB|\varepsilon$, $B\to b|\varepsilon$, modify $B\to b$ and add $S\to A$, so that we have $S\to AB|A|\varepsilon$, $B\to b$
  - Remove unit productions $A\to B$ by replacing target with all its productions, e.g. if CFG contains $S\to B$, $B\to bB|B$, replace B with its productions, so that we have $S\to bB|B$, $B\to bB|B$
  - Remove long productions $A\to B_1\ldots B_k$ with $k>2$ by using

intermediate terminals, e.g. if CFG contains $A \to B_1 B_2 B_3$, add intermediate variable, so that we have $A \to B_1 X_1,\ X_1 \to B_2 B_3$
  - Convert terminals in mixed rules $A \to aB$ by using intermediate terminals, e.g. if CFG contains $A \to aB$, add intermediate variable, so that we have $A \to XB,\ X \to a$

Weighted CFGs —
- A more general formulation of PCFGs
- Probabilities are globally normalized over all possible parse trees
- Probability of a parse tree: $p(t) = \frac{\prod_{r \in t} \exp(\text{score}(r))}{Z} = \frac{\prod_{r \in t} \exp(\text{score}(r))}{\sum_{t' \in T} \prod_{r' \in t'} \exp(\text{score}(r'))}$
- Challenge: Z is infinitely large, potentially even larger than Σ^* for ambiguous strings
- Initial solution: Probability of a parse tree, conditioned on string s: $p(t \mid s) = \frac{\prod_{r \in t} \exp(\text{score}(r))}{Z(s)} = \frac{\prod_{r \in t} \exp(\text{score}(r))}{\sum_{t' \in T(s)} \prod_{r' \in t'} \exp(\text{score}(r'))}$
- Challenge: Z is still potentially infinitely large due to cyclic rules
- Final solution: Revert to CNF. Then, size of Z is the number of rooted binary trees, which is the Catalan number $C_{M-1} = \frac{1}{M}\binom{2(M-1)}{M-1}$
  - For $M \leq 2, C_{M-1} = 1$
  - For $M = 3, C_{M-1} = 2$
  - For $M = 4, C_{M-1} = 5$
  - ...
- With CNF, probability of a parse tree (if $s \neq \varepsilon$): $p(t \mid s) = \frac{\prod_{N_i \to N_j N_k, \in t} \exp(\text{score}(N_i \to N_j N_k)) \times \prod_{N_l \to a, \in t} \exp(\text{score}(N_l \to a))}{Z} = \frac{\prod_{N_i \to N_j N_k, \in t} \exp(\text{score}(N_i \to N_j N_k)) \times \prod_{N_l \to a, \in t} \exp(\text{score}(N_l \to a))}{\sum_{t' \in T(s)} \prod_{N_i \to N_j N_k, \in t'} \exp(\text{score}(N_i \to N_j N_k)) \times \prod_{N_l \to a, \in t'} \exp(\text{score}(N_l \to a))}$ where
  - Non-terminal productions: $\prod_{N_i \to N_j N_k, \in t} \exp(\text{score}(N_i \to N_j N_k))$
  - Terminal productions: $\prod_{N_l \to a, \in t} \exp(\text{score}(N_l \to a))$

Span —
- Contiguous segment of a string from word w_i to w_{j-1}: $[i, j]$
- Span is *admissible* if it is possible to construct a parse tree, which covers exactly that span and is a valid constituent according to the grammar, i.e. there exists a non-terminal X such that $X \to w[i:j]$
- For string of length M and parse tree that is organized hierarchically in strictly right-to-left manner (nodes expand along right diagonal of tree only), we have only one admissible span per span size:
  - $[M, M+1)$ for span size $= 1$
  - $[1, M+1)$ for span size $= M$
  - More generally: $[M - \text{span size} + 1, M + 1)$

Optimization

Cocke-Kasami-Younger (CKY) Algorithm —
- Tasks:
  - Compute Z
  - Find best parse and its probability
  - Solve the *recognition problem*: Determine if a given string is admissible by the grammar
- Requires grammar in CNF
- Algorithm to compute Z:
  - Uses inside semiring or log-sum-exp semiring
  - Chart is filled bottom-to-top, left-to-right, exhaustively combining lower-level blocks
  - Chart entry $\text{Chart}[i, j, X]$ stores the probability that the non-terminal X generates the substring $s_i, \ldots, s_j$.
1. $M \leftarrow |s|$ where s is string
   $\text{chart} \leftarrow \overline{0}$
2. For $m = 1, \ldots, M$:
   Terminal productions: For $X \to s_m$, where s_m is terminal at position m in string:
   $\text{Chart}[m, m+1, X] \oplus= \exp(\text{score}(X \to s_m))$ where score is weight assigned to the given rule
3. For span $= 2, \ldots, M$:
   Beginning of span: For $i = 1, \ldots, M - \text{span} + 1$:
   End of span: $k \leftarrow i + \text{span} - 1$
   Breaking point of span: For $j = i+1, \ldots, k-1$:
   Non-terminal productions: For $X \to YZ$: $\text{Chart}[i, k, X] \oplus= \exp(\text{score}(X \to YZ)) \otimes \text{Chart}[i, j, Y] \otimes \text{Chart}[j, k, Z]$ where
   - Score is weight assigned to the given rule
   - Chart entries contain probabilities that Y resp. Z generate left resp. right subtree at i, j resp. j, k
4. Return $\text{Chart}[1, M+1, \mathcal{S}]$, which contains the normalization constant

- Complexity:
  - Runtime complexity: $\mathcal{O}(M^3 |\mathcal{R}|)$
  - Space complexity: $\mathcal{O}(M^2 |\mathcal{R}|)$
  - For parse tree that is organized hierarchically in strictly right-to-left manner, we know $i = M - \text{span size} + 1$ (beginning) and $j = i + 1$ (breaking point), thus, we can omit looping over i and j
  - Then, runtime complexity: $\mathcal{O}(M |\mathcal{R}|)$
- Algorithm to find best parse and its probability: Run above algorithm with Viterbi or arctic semiring and implement backpointers
- Algorithm to find whether string is admissible by grammar: Run above algorithm with Boolean semiring

Other

Pumping lemma —
- In any CFG, long enough strings have some kind of repeating structure
- Then, there exists a k such that a string s with $|s| > k$ can be written as $s = uxyzv$, where:
  - u, y, v are fixed parts, whereas x and z are pumpable parts
  - $ux^n yz^n v$ is also in the CFG
  - Not both x and z are not empty
  - $|xyz| < k$

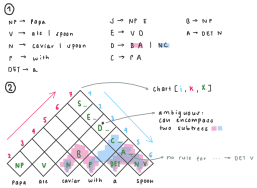
  Proof:
  - Consider a parse tree with height $|\mathcal{N}| + 1 = N$. This tree uses all non-terminals in $|\mathcal{N}|$. Thus, any tree with height $> N$ must include some repetition of a non-terminal symbol A
  - Let $k >$ length of any string yielded by a tree with maximum height N
  - If $|s| \geq k$, then s must be yielded by a tree with height $> N$, meaning s must include repetition of at least one non-terminal symbol A
  - From this, properties follow:
    - If T is the entire parse tree, T_A is the parse tree with root upper A, and T_{AA} is the parse tree with root lower A
    - T_A has height $\leq N$, since if it were $> N$, there would need to be another recurring A between T_A and T_{AA} which is a contradiction. Then, $|xyz| < k$, since tree with maximum height N can only yield strings with length $< k$
    - If both x and z were empty (i.e. we replaced T_A with T_{AA}), we would have a tree with height $\leq N$ that yields s with length $> k$, which is a contradiction. Then, x and z cannot be empty
    - We can extend the proof to any N
- Pumping lemma shows that languages that require strict equality between counts symbols cannot be generated by a CFG: String $a^k b^k c^k = uxyzv$ must fulfill $|xyz| \leq k$. Then, if a is in the string, the string cannot contain a c, and vice versa. Then, the string does not fulfill the requirement of strict equality of counts between symbols

Encoding a CRF for POS tagging as CFG —
- E.g., given CGF:
  - Start symbol expands to arbitrary non-terminals: $S \to AS \mid BS$
  - Start symbol can also directly expand to terminals: $S \to a \mid b$
  - Non-terminal A maps to terminal a: $A \to a$
  - Non-terminal B maps to terminal b: $B \to b$
- Transform grammar to encode CRF for POS tagging:
  - Start of sequence transition: Start symbol expands to non-terminals for each POS tag: $S \to B_{t_1} \mid B_{t_2} \mid \cdots \mid B_{t_{|\mathcal{T}|}}$
  - N-gram transitions: Tag associated with B_{t_i} is associated with a word via A_{t_i} and followed by another tag N-gram B_{t_j}: $B_{t_i} \to A_{t_i} B_{t_j} \quad \forall t \in \mathcal{T}$
  - Termination rule: For last tag in string: $B_{t_i} \to A_{t_i}, \quad \forall t \in \mathcal{T}$
  - Emissions: Maps tags to words: $A_{t_i} \to w_n, \quad \forall t \in \mathcal{T}, \forall w_n \in \mathcal{W}$
  - In total, $O(|T|^2 + |T||W|)$ rules, $|T|^N$ for transitions between N-grams of order N and $|T||W|$ for emissions
- In CRF, probability of tree: $p(t_1, \ldots, t_N \mid s_1, \ldots, s_N) \propto \prod_{n=1}^{N} \psi(t_n, t_{n-1}) \times \phi(t_n, s_n)$
- In CGF, probability of tree: $p(t \mid s_1, \ldots, s_N) \propto \prod_{r \in t} \text{score}(r)$ where
  - Start of sequence transition: $\text{score}(S \to B_{t_i, t_j, \ldots}) = \psi(\text{BOS}, t_i)$
  - N-gram transitions: $\text{score}(B_{t_i} \to A_{t_i} B_{t_j}) = \psi(t_i, t_j)$
  - Termination rule: $\text{score}(B_{t_i} \to A_{t_i}) = 1$
  - Emissions: $\text{score}(A_{t_i} \to s_n) = \phi(t_i, s_n)$

1) Syntactic parsing

2) Pumping lemma

3) CKY algorithm



29 Syntax | Dependency Parsing

Description

Task — Assign syntactic structure to a string by breaking it down into a hierarchy of dependencies (similar to syntactic parsing)

Formulation

- String w of length N
- *Spanning tree*:
  - Connects N nodes via $N - 1$ edges
  - Contains no cycles
- For dependency parsing, we also require:
  - The tree has exactly one root node, with only one outgoing edge
  - All non-root nodes have exactly one incoming edge
  - Edges are directed
  - Edges are labeled
- We can turn syntactic parsing into dependency parsing based

on rules:
  * Head of a production rule in syntactic parsing is the head of the dependency relation in dependency parsing
  * Then, an arrow goes from the head to the sibling
  * E.g. in $NP \to AdjN$, N is the head, so there is an arrow from N to Adj
- Types of dependency trees:
  * *Projective trees*:
    · No crossing arcs
    · Closely related to constituents (syntactic parsing)
    · Note: In syntactic parsing, dependency relationships are always nested within constituents, meaning that syntactic parsing trees are always projective
  * *Non-projective trees*:
    · Crossing arcs
    · In focus in the following

Probability of spanning tree —
- $p(t \mid w) = \frac{\exp(\text{score}(t,w))}{Z} = \frac{\exp(\text{score}(t,w))}{\sum_{t'} \exp(\text{score}(t',w))}$
- Challenge: Naively, if $|w| = N$:
  * It takes $\mathcal{O}(N^N)$ time to compute Z, if we don't impose a spanning tree structure and every node can depend on any other node, including itself, since there are N^N trees in graph
  * *Cayley's formula*: $\mathcal{O}(N^{N-2})$, if we impose a spanning tree structure constraint
  * $\mathcal{O}((N-1)^{N-2})$, if we also impose the constraint that there is exactly one root node
- Solution:
  * *Edge factored assumption*:
    $p(t \mid w) = \frac{\prod_{(i\to j)\in t} \exp(\text{score}(i,j,w)) \exp(\text{score}(r,w))}{Z} = \frac{\prod_{(i\to j)\in t} \exp(\text{score}(i,j,w)) \exp(\text{score}(r,w))}{\sum_{t'} \prod_{(i\to j)\in t'} \exp(\text{score}(i,j,w)) \exp(\text{score}(r,w))}$ where $(i \to j)$ is an edge in the tree, r is the root
  * *Kirchhoff's Matrix-Tree Theorem*: Method for counting the number of undirected spanning trees in $\mathcal{O}(N^3)$ time
  * *Tutte's Matrix-Tree Theorem*: Generalizes Kirchhoff's Matrix-Tree Theorem to directed spanning trees:
    · Take adjacency matrix and root vector:
    · *Adjacency matrix*: One entry for each node i to node j, if they were connected via an edge $i \to j$
    $A_{ij} = \exp(\text{score}(i,j,w))$ with 0 on diagonal and not-null on off-diagonal
    · *Root vector*: One entry for each node j, if it were the root node $\rho_j = \exp(\text{score}(j,w))$
    · Construct *Laplacian matrix*: Not accounting for constraint that there is only one root node:
    $$L_{ij} = \begin{cases} -A_{ij} & \text{if } i \neq j \\ \rho_j + \sum_{k\neq i} A_{kj} & \text{otherwise} \end{cases}$$
    Accounting for constraint that there is only one root node:
    $$L_{ij} = \begin{cases} \rho_j & \text{if } i = 1 \\ \sum_{i'=1,i'\neq j}^{n} A_{i'j} & \text{if } i = j \\ -A_{ij} & \text{otherwise} \end{cases} \quad \text{i.e.}$$
    $$\begin{cases} \text{- first row of L contains root scores} \\ \text{- diagonal of L (except } L_{1,1}\text{) contains} \\ \text{sum within each column of } A \text{ (except } A_{i,i}\text{)} \\ \text{- off-diagonal of L (except first row) contains} \\ \text{elements of A multiplied with } -1 \end{cases}$$
    · According to matrix tree theorem: $|L| = \det(L) = Z =$ number of trees in graph
    · According to Cayley's formula: number of trees in graph = N^{N-2}
  * Then, we can compute Z under the edge factored assumption in $\mathcal{O}(N^3)$

Optimization

Construct tree:
- N nodes plus 1 extra root node
- $(N+1)^{(N+1)}$ edges without spanning tree constraint

Decode tree:
- Challenge: To perform decoding, greedy *Kruskal's algorithm* does not work, since this selects the highest-scoring edge at each steps, which may be suboptimal in the directed case
- Solution: *Chu-Liu/Edmonds algorithm*

Chu-Liu/Edmonds algorithm:
- Algorithm:
  1. Greedy algorithm selects best incoming edge for each node, except the root
  2. This can cause cycles, which we need to contract:
     * Cycle treated as one node
     * Edges between nodes in the cycle are dead
     * Edges between nodes fully outside the cycle are external
     * Edges exiting the cycle are exits
     * Edges entering the cycle are enters
  3. Break cycle: For each enter edge, break cycle by removing edges that are also incoming at the node where the enter edge is incoming
  4. Re-weight: To enter edge, add weights of remaining edges that are strictly on (not in) the cycle
  5. Now, greedy algorithm selects tree without cycles
  6. Then, we can re-expand: Pick edge of highest-scoring enter edge in contracted form, pick edges that were used to re-weight when breaking cycle on that enter edge
- Without root constraint: Can be run in $\mathcal{O}(N^2)$ time
- With root constraint:
  * Naively: Run above algorithm N times, fixing each edge as the only one emanating from the root, adds factor N to runtime
  * Solution: Adjusted algorithm
- Adjusted algorithm:
  1. Run steps $1-4$ (contract cycle, break cycle, re-weight edges) as above
  2. If there are multiple edges emanating from the root: For each root edge, calculate cost of deleting this edge: Cost = Weight of root edge - weight of next-best incoming edge to target node (treating any cycles as single nodes)
  3. Preliminarily remove edge with lowest cost, but keep target node intact
  4. Preliminarily repeat step 2 (calculate root edge cost, remove lowest-cost root edge) here as needed
  5. Re-run greedy algorithm in contracted form
  6. If this leads to a cycle: Undo removal of edge with lowest cost, contract (treating any cycles as single nodes), then re-expand
     Otherwise: Re-expand

Other

First vs. second-order dependency parsing —
- Order of dependency parsing:
- *Grandparent* g, *parent* h, *sibling* s, *trailing sibling* t, *modifier* m, where root can be a grandparent or parent
  * *First-order dependency parsing*:
    · Considers only direct parent-modifier relationships
    · $h \to m$
  * *Second-order dependency parsing*: Extends to include:
    · Parent-sibling relationships: $h \to s; h \to m$
    · Grandparent-parent-modifier relationships: $g \to h \to m$
  * *Third-order dependency parsing*: Further extends to include:
    · Grandparent-parent relationships: $g \to h; h \to s; h \to m$
    · Parent-trailing sibling relationships: $h \to t; h \to s; h \to m$
- Edge scores:
  * k is grandparent, i is parent, j is modifier, s is sibling
  * Probability of spanning tree: $p(t \mid w) = \frac{1}{Z} \exp(\text{score}(t,w))$
  * First-order dependency parsing: Score given by:
    $\Psi(\boldsymbol{y}, \boldsymbol{w}; \theta) = \sum_{i \xrightarrow{r} j \in \boldsymbol{y}} \left[\psi_{\text{parent}}(i \xrightarrow{r} j, \boldsymbol{w}; \theta)\right]$
  * First-order dependency parsing with N scores
  * Second-order dependency parsing: Score given by:
    $\Psi(\boldsymbol{y}, \boldsymbol{w}; \theta) = \sum_{i\to j\in\boldsymbol{y}} [\psi_{\text{parent}}(i \xrightarrow{r} j, \boldsymbol{w}; \theta)$
    $+ \sum_{k\to i\in\boldsymbol{y}} \psi_{\text{grandparent}}(i \xrightarrow{r} j, k, r', \boldsymbol{w}; \theta)$
    $+ \sum_{i\to s\in\boldsymbol{y}, s\neq j} \psi_{\text{sibling}}(i \xrightarrow{r} j, s, r', \boldsymbol{w}; \theta)]$
  * In worst case (flat parse, root has a child, which is a parent to all other $N-1$ nodes, that end up being siblings with the other $N-2$ nodes), second-order dependency parsing with
    · N first-order scores
    · $N-1$ second-order grandparent scores
    · $(N-1)\times(N-2)$ second-order sibling scores
    · In total:
    $N + (N-1) + (N-1)\times(N-2) = 2N - 1 + N^2 - 3N + 2 = N^2 - N + 1$ scores

Deep Dive: Cayley's Formula and Matrix Tree Theorem for Unweighted Graphs — 1) Aim: Prove that $N^{(N-2)} = \det(L)$
Number of spanning trees in a undirected complete graph:
- Cayley's formula: Number of spanning trees in a undirected complete graph is N^{N-2}
- Undirected complete graph: Undirected edge exists between every pair of nodes
- Assume N nodes in G
- Assume adjacency matrix A with 1 on off-diagonal and 0 on diagonal
- Laplacian matrix L without root constraint is then given by -1 on off-diagonal and $N-1$ on diagonal
- Minor Laplacian matrix $\hat{L}_i$, which results from Laplacian matrix if i^{th} row and column is removed, has the same structure
- Assume $\hat{L}_i$ has $p = q = N-1$ rows and columns
- For node pairs $p, q \in \{1, \ldots, N-1\}$ assume k^{th} element of vector v is $v_k = \begin{cases} 1 & \text{if } k = p \\ -1 & \text{if } k = q \\ 0 & \text{otherwise} \end{cases}$
- If we apply $\hat{L}_i v_k$, we get:
  * Effect on row $k = p$:
    · The diagonal term contributes $(N-1)\times v_p = (N-1)\times 1 = N-1$
    · From the off-diagonal terms, only q contributes $-1 \times v_q = -1 \times -1 = 1$
    · Thus: $[\hat{L}_i v]_p = (N-1) + 1 = N$
  * Effect on row $k = q$:
    · The diagonal term contributes $(N-1)\times v_q = (N-1)\times -1 = -N+1$
    · From the off-diagonal terms, only p contributes $-1 \times v_p = -1 \times 1 = -1$

  - Thus: $[\hat{L}_i v]_q = -N + 1 - 1 = -N$
- Effect on rows $k \neq p, q$:
  - For any $k \neq p, q$, the vector v has zero entries
  - Thus: $[\hat{L}_i v]_k = 0$
- Then, we can see that v_k is an eigenvector of $\hat{L}_i$ with eigenvalue N: $\hat{L}_i v_k = N \times v_k$
- Now assume two sets:
  - $S_1 = \left\{ x \mid x = \sum_{p,q \in \{1,\dots,N-1\}, p \neq q} a v \right\}$ contains linear combinations x of v
  - $S_2 = \left\{ x \mid \mathbf{1}^\top x = \sum_{k=1}^{N-1} x_k = 0 \right\}$ requires that the sum of all components in x is 0
- We can show that $S_1 = S_2$:
  - $S_1 \subseteq S_2$: Elements in S_1 fulfill requirement in S_2 due to structure of v: $\sum_{k=1}^{N-1} v_k = 1 + (-1) + 0 + \dots + 0 = 0 \Rightarrow \sum_{k=1}^{N-1} \sum_{p,q \in \{1,\dots,N-1\}, p \neq q} a v_k = 0$
  - $S_2 \subseteq S_1$: S_2 lies in a subspace of dimension $N-2$, since we have $N-1$ components that must sum to 0. v span this subspace
  - This shows that there are $N-2$ linearly independent eigenvectors v
- Since $\hat{L}_i$ is a diagonal matrix, it's determinant is given by the product of the eigenvalues for all linearly independent eigenvectors. We have shown that. The eigenvalue is N and there are $N-2$ linearly independent eigenvectors. Thus, $\det(\hat{L}_i) = N^{(N-2)}$

Number of spanning trees in a directed complete graph:
- Number of spanning trees in a directed complete graph is N^{N-1}
- If an undirected graph G with N nodes has N_U spanning trees, then there are $N_D = N_U \times N$ spanning trees in its bidirected counterpart G', where for each undirected edge $i - j$ in G there are 2 directed edges $i \to j, j \to i$ in G', since we can form N rooted trees in G' for every tree in G
- If $N_U = N^{(N-2)}$ as proven above, then $N_D = N_U \times N = N^{(N-2)} \times N = N^{(N-1)}$

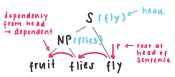
2) Aim: Prove that $\det(L)$ = number of trees in graph
- Let Laplacian matrix be L_G, minor Laplacian matrix $\hat{L}_i$
- We define L_e as the Laplacian matrix for a graph with N nodes, but only a single edge between i, j. This means that L_e contains 0 everywhere, except for the diagonal entries $L_{e,ii}, L_{e,jj}$, which are 1, and the off-diagonal entries $L_{e,ij}, L_{e,ji}$ which are -1
- Then, Laplacian matrix can be decomposed as: $L_G = \sum_{e \in E} L_e$
- We can show that $L_e = (e_i - e_j)(e_i - e_j)^\top$ where e is the standard basis vector:
  - *Incidence vector* for an edge given by $e_i - e_j = [0, \dots, 1, \dots, 0, \dots, -1, \dots, 0]^\top$ with 1 in position i (source node) and -1 in position j (target node)
  - The same goes for e_j
  - In the matrix $\left[(e_i - e_j)(e_i - e_j)^\top\right]$, each entry is given by:
$$\left[(e_i - e_j)(e_i - e_j)^\top\right]_{kl} = \begin{cases} 1 & \text{if } k = l \in \{i, j\} \\ -1 & \text{if } (k, l) \in \{i, j\} \\ 0 & \text{if } (k, l) \notin \{i, j\} \end{cases}$$
  - This matches L_e
- We can show that L_e is psd:
  - $L_G = \sum_{e \in E} L_e = \sum_{e \in E} (e_i - e_j)(e_i - e_j)^\top$
  - $x^\top L_G x = \sum_{(i,j) \in E} x^\top (e_i - e_j)(e_i - e_j)^\top x = \sum_{(i,j) \in E} \left[x^\top (e_i - e_j)\right]^2 \geq 0$ for all x
- *Incidence matrix* for all edges given by $B = [\mathbf{e}_{i_1} - \mathbf{e}_{j_1}, \mathbf{e}_{i_2} - \mathbf{e}_{j_2}, \dots, \mathbf{e}_{i_M} - \mathbf{e}_{j_M}] \in \mathbb{R}^{N \times M}$ where N is the number of nodes and M is the number of edges in G
- The k^{th} column $\mathbf{e}_{i_k} - \mathbf{e}_{j_k}$ of B represents the incidence vector of the edge e_k
- $L_G = \sum_{e_k \in E} (\mathbf{e}_{i_k} - \mathbf{e}_{j_k})(\mathbf{e}_{i_k} - \mathbf{e}_{j_k})^\top = \sum_{k=1}^{M} B_k B_k^\top = BB^\top$
- Then, $\hat{L}_i = B[i]B[i]^\top$ where $B[i]$ is B with the i^{th} row removed
- *Cauchy-Binet formula*:
  - Let $A \in \mathbb{R}^{N \times M}$, $B \in \mathbb{R}^{M \times N}$ with $M \geq N$
  - For an index set $S \subseteq [M]$, define the submatrices $A_S, B_S \in \mathbb{R}^{N \times N}$ by taking the indexed columns (of A) and rows (of B)
  - Then $\det(AB) = \sum_S \det(A_S)\det(B_S)$ where we sum over all $\binom{M}{N}$ index sets such that $|S| = N$
- Lemma:
  - For $|S| = N - 1$, the submatrix $B[i]_S$ corresponds to selecting $N-1$ edges E_S in G:
$$\det(B[i]_S) = \begin{cases} 1 & \text{if } E_S \text{ forms a spanning tree of } G \\ 0 & \text{otherwise} \end{cases}$$
- We now use the Cauchy-Binet formula and the lemma to show that $\det(\hat{L}_i) = Z$ = number of trees in graph:
  - According to Cauchy-Binet formula, $\det(\hat{L}_i) = \sum_S \det(B[i]_S)\det(B[i]_S^\top) = \sum_S \det(B[i]_S)^2$
  - According to lemma,
$$\det(B[i]_S) = \begin{cases} 1 & \text{if } E_S \text{ forms a spanning tree of } G \\ 0 & \text{otherwise} \end{cases}$$
  - Then, $\det(\hat{L}_i) = \sum_S \det(B[i]_S)^2$ counts the number of trees in graph

Proof of Lemma:
- $\det(B[i]_S) = 0$ if E_S is not a spanning tree:
  - If E_S is not a spanning tree, it must contain a cycle $C \subseteq E_S$
  - For a cycle C, the sum of the incidence vectors of the edges in C is zero: $\sum_{e \in C} (\mathbf{e}_i - \mathbf{e}_j) = 0$
  - This means, that $B[i]_C$ has linearly dependent columns
  - Then, its determinant is 0
- $\det(B[i]_S) = 1$ if E_S is a spanning tree:
  - Base case: For a single edge E_S, the matrix $B[i]_S$ is a 1×1 matrix, containing ± 1. The determinant for a matrix with just one element is the element itself: $\det(B[i]_S) = \pm 1$
  - Inductive hypothesis: Assume that for any spanning tree with $k = N - 2$ edges: $\det(B[i]_S) = \pm 1$
  - Inductive step: For a spanning tree with $k + 1 = N - 1$ edges:
    - We can swap rows and columns of $B[i]_S$, such that
    - The column corresponding to the edge $i \to j$ (where j is a leaf node) is the last column
    - The row corresponding to the leaf node j is the bottom row
    - Then, $B[i]_S = \begin{bmatrix} M & \mathbf{v} \\ \mathbf{0}^\top & \pm 1 \end{bmatrix}$,
    - Since a leaf node j is connected to the rest of the tree by exactly one edge (degree = 1), in this case $i \to j$, the incidence vector will have: $[0, 0, \dots, \pm 1]^\top$
    - Using the block matrix determinant formula: $\det(B[i]_S) = \det(M) \times (\pm 1) = \pm \det(M)$
    - By the inductive hypothesis, $\det(M) = \pm 1$
    - Therefore: $\det(B[i]_S) = \pm 1$

Deep Dive: Cayley's Formula and Matrix Tree Theorem for Weighted Graphs — Aim: Prove that $\sum_{\text{spanning tree } \tau} \prod_{e=(i,j) \in \tau} w_{ij} = \det(L)$
- Laplacian matrix L without root constraint and with weights is given by $-w_{ij}$ on off-diagonal and $\sum_j w_{ij}$ on diagonal
- For this, we need to replace the unweighted incidence matrix B with a weighted incidence matrix $\tilde{B}$ where the k^{th} column represents the weighted incidence vector of the edge e_k, with $\sqrt{w_{ij}}$ at position i (source node), $-\sqrt{w_{ij}}$ at position j (target node), and 0 elsewhere
- This allows us to get $L = \tilde{B}\tilde{B}^\top$
- For a tree τ, the determinant of the weighted incidence submatrix $\tilde{B}[i]_S$ contributes: $\prod_{e \in \tau} w_{ij}$

1) Dependency parse

dependency parsing



2) Syntactic parsing →

30 Semantics | Semantic Parsing

Description

Task — Assign semantic structure, i.e. logical form, to a sentence

Description —
- *Principle of compositionality*:
  - The meaning of a complex expression is a function of the meanings of that expression's constituent parts
  - Thus, we can syntactically parse a sentence and then construct the semantic representation bottom-up
- Applying the principle of compositionality:
  - Syntactic rule, e.g. S → NP VP
  - Induces semantic rule, e.g. s.sem → VP.sem(NP.sem): Semantic representation of sentence s is a function of the semantic representation of the verb phrase, applied to the semantic representation of the noun phrase

Lambda Calculus

Basics —
- Basic components:
  - Logical constants: Represent objects (e.g. Boston) and relations (e.g. likes)
  - Variables:
    - Undetermined logical constants
    - Objects are represented in lowercase as $x, y, z, \dots$ and are input to $\lambda x.f(x)$
    - Relations are represented in uppercase as $P, Q, R, \dots$ and are input to $\lambda P.P(\dots)$
    - *Free variables*: Do not occur in the scope of any abstraction that holds its name
    - Otherwise, *bound variables*: Bound by the abstraction with the smallest scope
    - E.g. $((\lambda x[.\lambda y[.(x((\lambda x.[x]x)y))]]\lambda x.[x])z)$ where
    - z is unbound
    - Other variables are bound by same-colored abstractions
    - Scopes of abstractions are indicated by square brackets
  - Every relation has an *arity* that determines the number of objects it relates (e.g. $P(x, y)$ has arity 2)
  - *Literals*: Formed by applying relations to objects (e.g. likes(Alex, y), $P(x, y)$)
  - Logical connectives and quantifiers: $\exists, \forall, \neg, \wedge, \vee$
- New terms can be constructed using 2 recursive rules:
  - 1) Abstraction:
    - If M is a term, N is a term, and x is a variable:
    - λx is an *abstraction*
    - $\lambda x.M$ is a function with input x and *scope of abstraction* M.

It replaces every free occurrence of x in M with whatever the function is applied to (e.g. N)
  - $\lambda x.MN$ is a function applied to N
  - We denote the output of $\lambda x.MN$ as $M[x := N]$
  - Note:
  - In the output of $\lambda x.MN$, only $M[x := N]$ remains, and $\lambda x.$ and N disappear
  - In $M[x := N]$ for relations $P(x)$, the relation P is exchanged, but the arguments x are left untouched
  - Work from outside in
  - Determine what is applied to what based on what is the argument in the CCG expression $X \mid Y$ (e.g. if Y is the argument in X/Y, then the lambda expression is applied to Y: $\lambda x.MY$)
  - When forming abstraction (i.e. $M \to \lambda x.M$), already bound occurrences of x in M remain bound by inner λ, free occurrences of x become bound by outer λ
* 2) Application: If M and N are terms, (MN) is a term
  - When forming application (i.e. $M \to MN$), already bound occurrences of x in M remain bound by inner λ, free occurrences of x remain free

Alpha conversion —
- Process of renaming a variable in a lambda term
- We can rename a variable in an abstraction together with all its occurrences in the scope of the abstraction
- If the renamed variable remains bound to the same abstraction and the remaining variables remain free resp. bound as before, the renaming is valid
- E.g. $\lambda x.\lambda y.(x((\lambda x.xx)y)) \to \lambda z.\lambda y.(z((\lambda x.xz)y))$ is valid
  $\lambda x.(xy) \to \lambda y.(yy)$ is not valid, since y was free before and is now bound

Beta reduction —
- Process of applying one lambda term to another: $(\lambda x.M)N$
- We can apply one lambda term to another if the free variables in N remain free in $M[x := N]$
- If this is not the case, first apply alpha-conversion to M
- Termination issue: Repeatedly applying beta reductions may not terminate
- E.g. $\lambda y.(z((\lambda x.\mathbf{x}z)y)) \to \lambda y.(z(\mathbf{z}y))$
  $(\lambda x.\lambda y.(\mathbf{x}((\lambda x.x\mathbf{x})y))z) \to \lambda y.(\mathbf{z}((\lambda x.xz)y))$ where
  * *blue* corresponds to M
  * *orange* corresponds to N
  * *red* corresponds to $M[x := N]$
  * ***bold*** corresponds to free variables in M

Equivalence — Two lambda terms are equivalent if they can be obtained from each other via a series of α- and β-conversions

Linear Indexed Grammar resp. Combinatory Categorial Grammars (CCG)

Features —
- LIG, CCG: Mildly context sensitive
- LIG, CCG: Support languages with complex or free word order (e.g. cross-serial dependencies, long-distance dependencies, coordination)
- CCG: *Compositionality*: Semantic representation of a sentence is built in tandem with its syntactic derivation
- The latter two are advantages over CFGs

Linear Indexed Grammar $\langle \mathcal{N}, S, \mathcal{I}, \Sigma, \mathcal{R} \rangle$ —
- $\mathcal{N}$: Set of non-terminals $N_1, N_2, N_3, \ldots$
- S: Start non-terminal
- $\mathcal{I}$: Finite set of indices $f, g, h, \ldots$
- Σ: Alphabet of terminals $a_1, a_2, a_3, \ldots$
- $\mathcal{R}$: Set of production rules of the following forms:
  1. $N[\sigma] \to \alpha M[\sigma]\beta$
  2. $N[\sigma] \to \alpha M[f\sigma]\beta$
  3. $N[f\sigma] \to \alpha M[\sigma]\beta$
  * Where
    - N and M are non-terminals
    - α and β are sequences of terminals and non-terminals
    - σ is associated with the non-terminal on the LHS and passed to exactly one non-terminal on RHS
    - For a stack, base symbol σ is usually combined with a marker f, where markers encode a count and can be popped from (rule 3) or pushed to (rule 2) the stack
- Can generate languages, which cannot be generated by CFG.
  E.g. language $\{a^n b^n c^n d^n : n \geq 0\}$ via rules:
  * $S[\sigma] \to aS[\sigma f]d$
  * $S[\sigma] \to T[\sigma]$
  * $T[\sigma f] \to bT[\sigma]c$
  * $T[\varepsilon] \to \varepsilon$

Combinatory Categorial Grammar $\langle \mathcal{V}_T, \mathcal{V}_N, S, f, \mathcal{R} \rangle$ —
- $\mathcal{V}_N$: *Atomic categories*: Set of non-terminals
- S: Start non-terminal
- $\mathcal{V}_T$: Set of terminals
- $\mathcal{R}$: Set of production rules
- $f : \mathcal{V}_T \to \mathcal{C}(\mathcal{V}_N)$: *Lexicon*: Function that maps elements of $\mathcal{V}_T \cup \{\varepsilon\}$ to finite subsets of $\mathcal{C}(\mathcal{V}_N)$
- $\mathcal{C}(\mathcal{V}_N)$: Set of *categories*, including *atomic* and *complex* categories

Deep-dive on categories, rules, and lexicon:
- Categories:
  * *Atomic categories*:
    - Complete constituents
    - Terminals
  * *Complex categories*:
    - Incomplete constituents
    - Built recursively from atomic categories via operators
    - Function that specifies the type of result and type and direction of its arguments with pattern: $X \mid Y \quad Y \to X$, where
    - $X \mid Y$ is function, with $\mid$ being an operator, Y being argument, and X being output
    - Applying function to Y yields X
    - *Operators*:
    - Backward slash: $X\backslash Y$: "Give me argument Y to my left, and I return an X"
    - Forward slash: X/Y: "Give me argument Y to my right, and I return an X"
    - Operators need to be read from outside in, e.g. (S\NP)/NP: Needs NP to its right, then needs NP to its left, then produces sentence S
    - Complex categories have an *arity* that determines the number of arguments it has (e.g. $X/Y\backslash Z$ has arity 2)
  * Every category can be written in the form
    $X = A \mid_m X_m \cdots \mid_1 X_1$ where
    - A: Atomic category, output of X
    - $X_1, \ldots, X_m$: Arbitrary categories, arguments of X
    - $\mid$ = / or \, whether X expects argument on right or left side
    - m = Arity of X, ≥ 0
  * Set of categories $\mathcal{C}(\mathcal{V}_N)$ is the smallest set such that:
    - If $C \in \mathcal{V}_N$, then $C \in \mathcal{C}(\mathcal{V}_N)$
    - If $C_1, C_2 \in \mathcal{V}_N$, then $(C_1/C_2) \in \mathcal{C}(\mathcal{V}_N)$ and $(C_1\backslash C_2) \in \mathcal{C}(\mathcal{V}_N)$
- Rules:
  * Specify how categories can be combined into other categories
  * *Function application*:
    - *Forward application* (>): If a category expects an argument to the right of (X/Y) and argument Y is available, they combine to form X: $X/Y \quad Y \to X$
    - *Backward application* (<): If a category expects an argument to the left of $(X\backslash Y)$ and argument Y is available, they combine to form X: $Y \quad X\backslash Y \to X$
    - Have *primary input* (function part in rule) and *secondary input* (argument part in rule)
    - *AB grammar resp. basic categorical grammar resp. pure categorical grammar*: CCGs that only have application rules, have power of CFG: $A \to a$ in CCG $\equiv A \to a$ in CFG
      $A \to (A/C)C$ in CCG $\equiv A \to BC$ in CFG
  * *Function composition*:
    - Two functions combine to form a single function: $f, g \to f \circ g$
    - *Forward composition* $((B_>)$: $(X/Y) \quad (Y/Z) \to (X/Z)$
    - *Backward composition* $((B_<)$: $(Y\backslash Z) \quad (X\backslash Y) \to (X\backslash Z)$
  * *Higher-order rules*:
    - *Forward* $(>^n)$: $X/Y \; Y \mid_n Y_n \ldots \mid_1 Y_1 \to X \mid_n Y_n \ldots \mid_1 Y_1$
    - *Backward* $(<^n)$: $Y \mid_n Y_n \ldots \mid_1 Y_1 \; X\backslash Y \to X \mid_n Y_n \ldots \mid_1 Y_1$
    - n is the *degree* of the rule
    - If $n = 0$, corresponds to application rule
  * *Type raising*:
    - Turn an atomic category into a complex category so it can participate in higher-order functions
    - *Forward type raising* $(T_>)$: $X \to T/(T\backslash X)$
    - *Backward type raising* $(T_<)$: $X \to T\backslash(T/X)$
    - E.g. Intransitive S\NP (e.g. walked)
      Transitive (S\NP)/NP (e.g. respected)
      Ditransitive ((S\NP)/NP)/NP (e.g. gave)
  * Rule instance is obtained by substituting $X, Y, Z, \ldots$ by concrete categories, e.g. S, NP, VP, etc.
  * CCGs have finite set of rules ($2^{|\mathcal{V}_N|}$ forward, $2^{|\mathcal{V}_N|}$ backward rules) but infinitely many rule instances
- Lexicon: Associates terminals with categories, encodes the structure (vs. CFG, where rules encodes the structure)
  * E.g. entry in lexicon for atomic category: Harry := NP
  * E.g. entry in lexicon for complex category: walks := $(S\backslash NP)$

CCG parsing:
- General rule of inference: $\frac{A_1 \quad \ldots \quad A_k}{B}$ where B is a consequence of $A_1, \ldots, A_k$
- *Derivation trees*:
  * Consist of unary and binary branches
  * *Unary branches*: Rules for atomic categories
  * *Binary branches*: Rules for complex categories

CCG parsing | CKY-style parsing algorithm:
- Let w be the sentence of length n to be parsed, w_i be a token in the sentence, and $w[i, j]$ be the substring from $w_{i+1} \ldots w_j$, and $w_{ii} = \varepsilon$
- Aim: Construct item $[S, 0, n]$, i.e. we can construct sentence based on words between positions 1 and n
- Derivation tree leading to this outcome has internal nodes $[X, i, j]$ i.e. we can construct category based on words between positions $i + 1$ and j
- *Axioms*: $[X, i, i + 1]$ where $w_{i+1} = X$ is a lexicon entry
- *Inference rules*:
  * $\frac{[X/Y, i, j] \quad [Y\beta, j, k]}{[X\beta, i, k]} \quad X/Y \quad Y\beta \to X\beta$
    resp.
    $\frac{[Y\beta, i, j] \quad [X\backslash Y, j, k]}{[X\beta, i, k]} \quad Y\beta \quad X\backslash Y \to X\beta$
- Challenge: With the composition rule, we can grow the arity of the primary input categories and get exponentially many primary input categories

- Solution: Restrict the arity of the categories

CCG parsing | Polynomial time algorithm:
- Arity of categories is bounded by *grammar constant* C_g: $[X,i,j]$ where $\text{ar}(X) \leq C_g$
- Challenge: Categories with arity $> C_g$ can no longer be derived
- Solution:
  - Introduce new rules that decompose longer derivations into smaller pieces
  - *Derivation context* c:
    - Configuration in derivation, where there are two subtrees t' and c, which together form subtree t
    - t parses the segment $w_{[i,j]}$ in the sentence, t' parses $w_{[i,i']}$ thereby deriving the function $\cdots X \mid Y$, and c parses $w_{[j',j]}$ thereby deriving the corresponding argument $Y \cdots$
    - t' and c combine to form $\cdots X \cdots$ which represents the entire derivation t for $w_{[i,j]}$
    - None of the combinatory rules used in c touches X, i.e. c only operates on Y
    - For this reason, c is a derivation context and can be re-used across multiple derivations for X
  - New inference rules:
    - Next to normal inference rules above
    - 1) If the composition of two categories $(X \mid Y \quad Y\beta)$ would result in a category with arity $\text{ar}(X\beta) > C_g$:

      Context item is derived: $\frac{[X/Y,i,j] \quad [Y\beta,j,k]}{[/Y,\beta,i,i,j,k]}$

      resp.

      $\frac{[Y\beta,i,j] \quad [X\backslash Y,j,k]}{[\backslash Y,\beta,i,i,j,k]}$???

      where β are all non-terminals and edge operators, except for Y
    - 2.1) If the further composition of a context item with another category $(X \mid Y \quad Y\beta)$ would result in a category with arity $\text{ar}(X\beta) \leq C_g$:

      Context item can be recombined with the original derivation: $\frac{[X/Y,i',j'] \quad [/Y,\beta,i,i',j',j]}{[X\beta,i,j]}$

      resp.

      $\frac{[\backslash Y,\beta,i,i',j',j] \quad [X\backslash Y,i',j']}{[X\beta,i,j]}$?
    - 2.2.1) If the further composition of a context item with another category $(\beta \mid Z \quad Z_\gamma)$ would result in a category with arity $\text{ar}(Y\beta_\gamma) \leq C_g$:

      Context item can be extended normally, similar to derivation trees: $\frac{[|Y,\beta/Z,i,i',j',j] \quad [Z_\gamma,j,k]}{[|Y,\beta_\gamma,i,i',j',k]}$

      resp.

      $\frac{[Z_\gamma,i,j] \quad [|Y,\beta\backslash Z,j,j',k',k]}{[|Y,\beta_\gamma,i,j',k',k]}$?

      where β_γ refers to β without $/Z$ (can be ε)
    - 2.2.2) If the further composition of a context item with another category would result in a category with arity $\text{ar}(Y\beta_\gamma) > C_g$:

      A new context is derived from the previous context: $\frac{[|Y,\beta/Z,i,i',j',j] \quad [Z_\gamma,j,k]}{[/Z,\gamma,i,i,j,k]}$

      resp.

      $\frac{[Z_\gamma,i,j] \quad [|Y,\beta\backslash Z,j,j',k',k]}{[\backslash Z,\gamma,i,i,j,k]}$???
    - 3.1) When the arity is small enough, the new context item can be recombined with the previous context: $\frac{[|Y,\beta/Z,i'',i',j',j''] \quad [/Z,\varepsilon,i,i'',j'',j]}{[|Y,\beta,i,i',j',j]}$

      resp.

      $\frac{[\backslash Z,\varepsilon,i'',i',j',j''] \quad [|Y,\beta\backslash Z,i,i'',j'',j]}{[|Y,\beta,i,i',j',j]}$???
- Grammar constant C_g is at least as large as maximal Y (determined by largest arity a in lexicon) and β (determined by the maximum degree n of composition rules), since we need to restrict the size of $Y\beta$
- Together: $C_g \geq \max\{\ell, a+n\}$

CCG combine syntactic and semantic information when paired with lambda calculus:
- E.g.

  Mary := NP : MARY(x)

  likes := (S\NP)/NP : $\lambda x.\ \lambda y.\ \text{likes}(x,y)$

  natural language := syntax : semantics
- If we parse a sentence syntactically, we can derive semantics bottom-up in the same order, working from smallest constituents to highest constituents

SK-Calculus resp. Combinatory Logic

- Alternative to lambda calculus
- Basic terms:
  - *Variables*: $x, y, z, \ldots$
  - *Primitive functions resp. combinators*:
    - S *combinator*: $Sxyz = (xz(yz)) = ((xz)(yz))$
    - K *combinator*: Constant function: $Kxy = ((Kx)y) = x$
    - I *combinator*: Identity function: $Ix = x$
    - B *combinator*: Composition function: $Bxyz = (x(yz))$
    - C *combinator*: $Cxyz = ((xz)y)$
    - T *combinator*: Type-raising function: $Txy = yx$
- Terms are recursively constructed via application: If M and N are terms, (MN) is a term
- Parentheses are left-associative, e.g. $(Kxyz) = (((Kx)y)z)$
- *SK calculus*: Only leverages S and K operator
  - SKK and I are extensionally equivalent: $S(KK)x = SKKx = Kx(Kx) = x$
  - Therefore, the SK basis is complete, and we don't need I operator

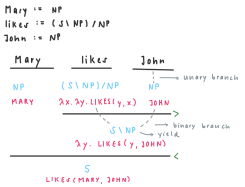
Transforming Lambda Calculus into SK-Calculus

Apply recursively defined transformation T:
1. $T(x) = x$ for every variable x
2. $T[(E_1 E_2)] = (T[E_1]T[E_2])$ for all lambda terms E_1, E_2
3. $T[\lambda x.E] = (KT[E])$ for every lambda term E where x is either bound or absent within the term
4. $T[\lambda x.x] = (SKK) = I$
5. $T[\lambda x.\lambda y.E] = T[\lambda x.T[\lambda y.E]]$ for every lambda term E where x is free within the term
6. $T[\lambda x.(E_1 E_2)] = (ST[\lambda x.E_1]T[\lambda x.E_2])$ for all lambda terms E_1, E_2 where x is free within at least one of the two terms

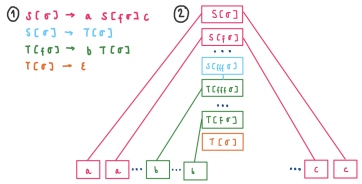
1) Semantic and syntactic parse

2) Derivation context

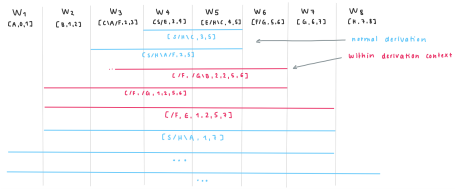
3) CCG parsing: Derivation trees



4) Linear indexed grammar: Example for language, which can be generated by LIG but not CFG



5) Polynomial time algorithm



31 Weighted Finite State Automata (WFSA)

Task — Determine whether a string is an element of a given language

Background —
- String s
- Empty string ε
- Set of all strings given by Kleene closure Σ^*
- String language L, subset of Σ^*

(W)FSA | Finite State Automata (FSA)

Formalization —
- $\mathcal{A}$ is a 5-tuple $(\Sigma, Q, I, F, \delta)$
  - Σ is an alphabet, consisting of letters $a, b, c, \ldots$
  - Q is a finite set of states
  - $I \subseteq Q$ is the set of initial states, usually only one
  - $F \subseteq Q$ is the set of final or accepting states, can be multiple
  - $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$ resp. $q \xrightarrow{a \in \Sigma \cup \{\varepsilon\}} q'$ is a finite multiset of transitions from one state in Q to another state in Q via a symbol in Σ
- Can be represented as a directed, possibly cyclical graph
- Sequentially reads individual symbols of an input string s and transitions from state q to state q' upon reading a symbol a iff $(q, a, q') \in \delta$
- If the automaton, after reading the last symbol of s, ends up in a state $q_f \in F$, the automaton *accepts* the string

(W)FSA | Weighted Finite State Automata (WFSA)

- Generalization of FSA, where transitions are weighted with a semiring: Unweighted FSA corresponds to WFSA weighted with Boolean semiring
- $\mathcal{A}$ is a 7-tuple $(\Sigma, Q, I, F, \delta, \lambda, \rho)$ over a semiring $\mathcal{W} = (\mathbb{K}, \oplus, \otimes, 0, 1)$ which, vs. FSA, defines:
  - $I = \{q \in Q \mid \lambda(q) \neq 0\} \subseteq Q$
  - $F = \{q \in Q \mid \rho(q) \neq 0\} \subseteq Q$
  - $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times \mathbb{K} \times Q$ resp. $q \xrightarrow{a \in \Sigma \cup \{\varepsilon\}/w} q'$ is a finite multiset of weighted transitions from one state in Q to another state in Q via a symbol in Σ
  - $\lambda : Q \to \mathbb{K}$ is an initial weighting function over Q, is $\overline{0}$ for non-initial states
  - $\rho : Q \to \mathbb{K}$ is a final weighting function over Q, is $\overline{0}$ for non-final states

(W)FSA | (W)FSA Terminology

- *Paths*:
  - A path π is an element of δ^* with consecutive transitions: $q_1 \xrightarrow{a_1/w_1} q_2, q_2 \xrightarrow{a_2/w_2} q_3, \cdots, q_{N-1} \xrightarrow{a_N/w_N} q_N$
  - $\text{p}(\pi) = q_1$ is the beginning state of the path

* $q(\pi) = q_N$ is the ending state of the path
* *Length* of path $|\pi|$ is the number of transitions
* *Yield* of path $s(\pi)$ is the concatenation of symbols on the path
* Path sets: Denoted by capital Π
  - $\Pi(\mathcal{A})$: Set of all paths in the automaton
  - $\Pi(s)$: Set of all paths with yield s
  - $\Pi(q)$: Set of all paths starting at q
  - $\Pi(q, q')$: Set of all paths from q to q'
  - $\Pi(q, s, q')$: Set of all paths from q to q' with yield s
  - $\Pi(\mathcal{Q}) = \bigcup_{q \in Q} \Pi(q)$
  - $\Pi(\mathcal{Q}, \mathcal{Q}') = \bigcup_{q \in Q, q' \in Q'} \Pi(q, q')$
  - $\Pi(\mathcal{Q}, s, \mathcal{Q}') = \bigcup_{q \in Q, q' \in Q'} \Pi(q, s, q')$
* *Inner path weight* $w_I(\pi)$ of a path $\pi = q_0 \xrightarrow{a_1/w_1} q_1 \cdots q_{N-1} \xrightarrow{a_N/w_N} q_N$: $w_I(\pi) = \bigotimes_{n=1}^{N} w_n$
* *Path weight* $w(\pi)$ of a path: $w(\pi) = \lambda(\mathrm{p}(\pi)) \otimes w_I(\pi) \otimes \rho(\mathrm{q}(\pi))$
* A path is *accepting resp. successful* iff $w(\pi) \neq 0$

– *Cycles*:
* A path is a cycle if starting and finishing states are the same
* A path contains a cycle if the same state appears multiple times

– Transitions:
* Outgoing arcs from q: $E_{\mathcal{A}}(q) = \{a, t, w \mid (q, a, w, t) \in \delta\}$
* Incoming arcs to q: $E_{\mathcal{A}}^{-1}(q) = \{a, s, w \mid (s, w, a, q) \in \delta\}$

– States:
* State q is *accessible* iff $q \in I$ or there exists a path whose weight is not 0 from I to q
* State q is *co-accessible* iff $q \in F$ or there exists a path whose weight is not 0 from q to F
* States that are either not accessible or not co-accessible are called *useless*

– (W)FSA is *unambiguous* iff for every string s there is at most one accepting path

WFST | Weighted Finite State Transducers (WFST)

Task — Transforms input string in a given language to output string in a given language, e.g. Muse → $\mu o \upsilon \sigma \alpha$

Background —
– x: Input string
– y: Transliteration of input string

Formalization —
– $\mathcal{T}$ is an 8-tuple $(\Sigma, \Omega, Q, I, F, \delta, \lambda, \rho)$ over a semiring $\mathcal{W} = (\mathbb{K}, \oplus, \otimes, 0, 1)$ which, vs. WFSA, defines:
* Σ is the input alphabet
* Ω is the output alphabet
* $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times (\Omega \cup \{\varepsilon\}) \times \mathbb{K} \times Q$ resp. $q \xrightarrow{a \in \Sigma \cup \{\varepsilon\} : b \in \Omega \cup \{\varepsilon\} / w} q'$ is a finite multiset of weighted transitions from one state in Q to another state in Q via a symbol in Σ and in Ω

WFST | Compositions

– Operation to combine two or more WFSTs
– $\mathcal{T}_1 \circ \mathcal{T}_2$ is the composition of $\mathcal{T}_1 = (\Sigma, \Omega, Q_1, I_1, F_1, \delta_1, \lambda_1, \rho_1)$ and $\mathcal{T}_2 = (\Omega, \Gamma, Q_2, I_2, F_2, \delta_2, \lambda_2, \rho_2)$ and is given by: $\mathcal{T} = (\Sigma, \Gamma, Q, I, F, \delta, \lambda, \rho)$ such that: $\mathcal{T}(x, y) = \bigoplus_{z \in \Omega^*} \mathcal{T}_1(x, z) \otimes \mathcal{T}_2(z, y)$ where $\mathcal{T}(i, j)$ is the weight assigned by the transducer to a mapping from an input string i to an output string j
– Naive algorithm to compute composition:
1. $\mathcal{T} \leftarrow (\Sigma, \Omega, Q, I, F, \delta, \lambda, \rho)$ to create a new WFST
2. Create initial state $(q_1^i, q_2^i) \in I_1 \times I_2$ and final state $(q_1^f, q_2^f) \in F_1 \times F_2$
3. For $q_1, q_2 \in Q_1 \times Q_2$:
   For $(q_1 \xrightarrow{a:b/w_1} q_1'), (q_2 \xrightarrow{c:d/w_2} q_2') \in E_{\mathcal{T}_1}(q_1) \times E_{\mathcal{T}_2}(q_2)$:
   If
   * $b = c$
   * (q_1, q_2) and (q_1', q_2') are both accessible (reachable from (q_1^i, q_2^i)) in the composed transducer
   * (q_1, q_2) and (q_1', q_2') are both co-accessible (can reach (q_1^f, q_2^f)) in the composed transducer
   * Add states (q_1, q_2), (q_1', q_2') to $\mathcal{T}$ if they have not yet been added
   * Add arc $(q_1, q_2) \xrightarrow{a:d/w_1 \otimes w_2} (q_1', q_2')$ to $\mathcal{T}$
4. For $(q_1^i, q_2^i) \in I_1 \times I_2$: $\lambda_{\mathcal{T}} = \lambda_1(q_1^i) \otimes \lambda_2(q_2^i)$
5. For $(q_1^f, q_2^f) \in F_1 \times F_2$: $\rho_{\mathcal{T}} = \rho_1(q_1^f) \otimes \rho_2(q_2^f)$
6. Return $\mathcal{T}$

– Challenge: Runs through many useless states, has runtime complexity $\mathcal{O}(|Q_1||Q_2|)$
– Solution: *Accessible algorithm*
– *Accessible algorithm*:
* Intuition: Construct all possible pairs of initial states and then expand outwards (depth-first search), adding accessible states
* Note: Produced states may still be non-co-accessible
1. stack $\leftarrow [(q_1, q_2) \mid q_1 \in I_1, q_2 \in I_2]$ (ordered list, allows duplicates)
   visited $\leftarrow \{(q_1, q_2) \mid q_1 \in I_1, q_2 \in I_2\}$ (unordered set, does not allow duplicates)
2. $\mathcal{T} \leftarrow (\Sigma, \Omega, Q, I, F, \delta, \lambda, \rho)$ to create a new WFST
3. While $|\text{stack}| > 0$:
   (a) $q_1, q_2 \leftarrow$ stack.pop()
   (b) For:
   $(q_1 \xrightarrow{a:b/w_1} q_1'), \quad (q_2 \xrightarrow{c:d/w_2} q_2') \in E_{\mathcal{T}_1}(q_1) \times E_{\mathcal{T}_2}(q_2)$:
   If $b = c$:
   i. Add states (q_1, q_2), (q_1', q_2') to $\mathcal{T}$ if they have not yet been added
   ii. Add arc $(q_1, q_2) \xrightarrow{a:d/w_1 \otimes w_2} (q_1', q_2')$ to $\mathcal{T}$
   iii. If (q_1', q_2') not in visited: Push (q_1', q_2') to stack and visited
4. For $(q_1^i, q_2^i) \in I_1 \times I_2$: $\lambda_{\mathcal{T}} = \lambda_1(q_1^i) \otimes \lambda_2(q_2^i)$
5. For $(q_1^f, q_2^f) \in F_1 \times F_2$: $\rho_{\mathcal{T}} = \rho_1(q_1^f) \otimes \rho_2(q_2^f)$

(W)FSA | Pathsum and Lehmann's algorithm

Pathsum —
– *Pathsum* of $\mathcal{A}$: $Z(\mathcal{A}) = \bigoplus_{\pi \in \Pi(\mathcal{A})} w(\pi)$
– Pathsum in different semirings:
* Pathsum is equal to the shortest path weight in the tropical semiring where $\otimes = +$ and $\oplus = \min$
* Pathsum is equal to the sum of all paths in the inside semiring where $\otimes = \times$ and $\oplus = +$
* In a $\overline{0}$-closed semiring:
  - Pathsum depends only on paths of length $N - 1$, since paths of length $\geq N$ contain cycles, but cycles do not improve the pathsum in $\overline{0}$-closed semirings, since $\overline{1} \oplus a = \overline{1}$
  - Then, in Lehmann's algorithm, $R^* = \bigoplus_{n=0}^{N-1} R^{\otimes n}$
  - This has time complexity of $O((N-1) \times (N^3 + N^2))$, since we need to multiply over matrix of size $N \times N$ ($O(N^3)$), sum over matrix of size $N \times N$ ($O(N^2)$) for $N - 1$ iterations
  - For idempotent semirings, where we can leverage binary decomposition, has time complexity of $O(\log_2(N-1) \times N^3 + (N-1) \times N^2)$, since we can pre-compute the $\log_2(N-1)$ required powers R^{2^k} for matrix multiplication

Pathsum in acyclic WFSA:
– In acyclic WFSA, the number of accepting paths is finite, thus $Z(\mathcal{A})$ is finite
– To compute pathsum $Z(\mathcal{A})$, we need a topological sort of the nodes (starts with initial state and progresses towards final state)
– Backward algorithm:
* Computes $Z(\mathcal{A})$
1. For $q \in \text{Rev-Top}(\mathcal{A})$ (starts with final state and progresses towards initial state):
   (a) If $q \in F$: $\beta(q) = \rho(q)$
   (b) Else: $\beta(q) = \bigoplus_{q \xrightarrow{a/w} q'} w \otimes \beta(q')$
2. Return: $\bigoplus_{q^i \in I} \lambda(q^i) \otimes \beta(q^i)$
* Runs in $\mathcal{O}(|\delta|)$ time and $\mathcal{O}(|Q|)$ space
* Could alternatively also be computed as forward algorithm:
1. For $q \in \text{Top}(\mathcal{A})$:
   (a) If $q \in I$: $\alpha(q) = \lambda(q)$
   (b) Else: $\alpha(q) = \bigoplus_{q' \xrightarrow{a/w} q} w \otimes \alpha(q')$
2. Return: $\bigoplus_{q^f \in F} \rho(q^f) \otimes \beta(q^f)$

Pathsum in cyclical WFSA:
– In cyclical WFSA, the number of accepting paths is infinite, thus $Z(\mathcal{A})$ is infinite
– Lehmann's algorithm can nonetheless calculate $Z(\mathcal{A})$
– To use this algorithm, we need to represent WFSA as a matrix:
* Define $|\Sigma|$ *adjacency matrices*, one for each transition symbol $a \in \Sigma$: $\boldsymbol{W}^{(a)} \in \mathbb{R}^{(|Q| \times |Q|)}$: Contains the from-states q_n in the rows and the to-states q_m in the columns, entries are given by the weight w for the transition $q_n \xrightarrow{a/w} q_m$ if it exists, otherwise $\overline{0}$
* Since in the pathsum all paths are summed, we can collapse all transitions from q_n to q_m into a single transition without a label, whose weight is the $\oplus$-sum of all original transitions: $\boldsymbol{W} = \bigoplus_{a \in \Sigma \cup \{\varepsilon\}} \boldsymbol{W}^{(a)}$
* Pathsum for paths between node q_n to node q_m of length exactly l can be calculated via matrix multiplication of the adjacency matrix: $\boldsymbol{W}^l = \boldsymbol{W} \otimes \boldsymbol{W} \otimes \ldots$
  Proof: Matrix multiplication given by $(\boldsymbol{W} \otimes \boldsymbol{W})_{nm} = \bigoplus_{i=1}^{|Q|} \boldsymbol{W}_{ni} \otimes \boldsymbol{W}_{im}$. Thus, we compute the $\otimes$-product of the weights along any given path between q_n and q_m. Then, we sum the computed weights across all paths between q_n and q_m

Lehmann's algorithm:
– Computes $Z(\mathcal{A})$ by computing Kleene closure of matrix R over closed semiring
– Algorithm: Computing $Z(\mathcal{A})$

1. Lehmann algorithm computes R with $|Q|^2$ entries. Each entry represents the $\oplus$-sum of the weights over the path between two nodes q_i, q_k: $R_{ik} = \bigoplus_{\pi \in \Pi(q_i,q_k)} w_I(\pi)$ where cyclical terms (either single nodes, or node combinations) are denoted with $*$
2. From R, we can compute $Z(\mathcal{A})$:
   $Z(\mathcal{A}) = \bigoplus_{i,k=1}^{|Q|} \lambda(q_i) \otimes R_{ik} \otimes \rho(q_k)$ where only pairs (i,k) are retained where i is initial state and k is final state, since for all other pairs either $\lambda(q_i) = \overline{0}$ or $\rho(q_k) = \overline{0}$

- Algorithm for step 1: Computing R:
  1. $R^{(0)} \leftarrow W$
  2. For $j \leftarrow 1$ up to $|Q|$:
     For $i \leftarrow 1$ up to $|Q|$:
     For $k \leftarrow 1$ up to $|Q|$:
     $R_{ik}^{\leq j} \leftarrow R_{ik}^{\leq j-1} \oplus \left(R_{ij}^{\leq j-1} \otimes (R_{jj}^{\leq j-1})^* \otimes R_{jk}^{\leq j-1}\right)$
  3. Return: $I \bigoplus R^{(|Q|)}$ where I is the matrix with $\overline{1}$ on the diagonal and $\overline{0}$ everywhere else
- Intuition for step 1: Computing R:
  - 1) For paths that do not pass through intermediary nodes (i.e. for direct connections): $R^{\leq 0} = W$
  - 2) For paths that do pass through intermediary nodes: Let $\Pi^{\leq j}(q_i, q_k)$ be paths between q_i and q_k that do not cross nodes with indices $> j$ and $R_{ik}^{\leq j}$ be the pathsum for these paths
  - We can partition the paths in $\Pi^{\leq j}(q_i, q_k)$ into two subsets: Paths that do not cross j and paths that do
  - $\Rightarrow$ For paths that do not cross j:
    - $\pi \in \Pi^{\leq j-1}(q_i, q_k)$
    - In that case, the pathsum is: $R_{ik}^{\leq j-1}$
  - $\Rightarrow$ For paths that do cross j:
    - π can be decomposed into cycle that starts and ends in j, part before cycle, and part after cycle: $\pi_{ij}\pi_{jj}\pi_{jk}$
    - In that case, we can also decompose the pathsum into partial pathsums: $R_{ij}^{\leq j-1} \otimes (R_{jj}^{\leq j-1})^* \otimes R_{jk}^{\leq j-1}$
  - $\Rightarrow$ Combining these subsets, we get $R_{ik}^{\leq j} \rightarrow R_{ik}^{\leq j-1} \oplus R_{ij}^{\leq j-1} \otimes (R_{jj}^{\leq j-1})^* \otimes R_{jk}^{\leq j-1}$
  - Clearly $R_{ik}^{\leq |Q|} = R_{ik}$
  - This defines a natural procedure to build up the pathsum from the partial pathsums
- Runs in $\mathcal{O}(|Q|^3)$ time
- Lehmann algorithm encompasses Floyd-Warshall algorithm and Gauss-Jordan algorithm

Transliteration

Motivation for using WFSAs resp. WFSTs — Why can't we use a CRF?

- In CRFs, we can only align sequences of the same length, e.g., $|x| = |y|$
- In WFSTs resp. WFSAs, we can align sequences of different lengths

Aim and approach — Aim: Compute log-likelihood

$p(y|x) = \frac{\exp(\text{score}(x,y))}{Z}$ where $\text{score}(x,y)$ is the pathsum $\log \sum_{\pi \in \Pi(y)} w(\pi)$

Approach:

- We encode source sequence $x \in \Sigma^*$ as WFST $\mathcal{T}_x$: This prepares x for alignment
- We encode target sequence $y \in \Omega^*$ as WFST $\mathcal{T}_y$: This prepares y for alignment
- We need a WFST $\mathcal{T}$ that maps string over Σ to string over Ω: Each path in $\mathcal{T}$ represents one alignment of some string in Σ^* to some string in some string in Ω^*
- We can generate the composition $\mathcal{T}_x \circ \mathcal{T}$: Each path in $\mathcal{T}_x \circ \mathcal{T}$ represents one alignment of x to some string in Ω^*
- We can generate the composition $\mathcal{T}_x \circ \mathcal{T} \circ \mathcal{T}_y$: Each path in $\mathcal{T}_x \circ \mathcal{T} \circ \mathcal{T}_y$ represents one alignment of x to y
- $p(y \mid x)$ is then given by the probability of all paths in $\mathcal{T}_x \circ \mathcal{T}$ that align x to y: $= \frac{\text{pathsum of } \mathcal{T}_x \circ \mathcal{T} \circ \mathcal{T}_y}{\text{pathsum of } \mathcal{T}_x \circ \mathcal{T}}$
- Training: To compute $p(y \mid x)$, we can use Lehmann's algorithm with the log-sum-exp semiring
- Inference: To compute highest scoring y of composition $\mathcal{T}_x \circ \mathcal{T}$, we can use e.g. Dijkstra's algorithm or Floyd-Warshall algorithm with the arctic semiring and backpointers
- If we condition on x in WFST $\mathcal{T}_x \circ \mathcal{T}$, this WFST becomes a WFSA

Training: Lehmann's Algorithm —

- $\mathcal{A}_x$ is the WFSA with:
  - λ is a vector of start weights for starting states: $\lambda_n = \lambda(q_n)$
  - ρ is a vector of end weights for ending states: $\rho_n = \rho(q_n)$
  - Set of adjacency matrices, one for each output symbol: $W^{(\omega)}$ for any $\omega \in \Omega \cup \{\varepsilon\}$: $W_{nm}^{(\omega)}$ is the weight for $q_n \xrightarrow{\omega} q_m$
  - $W^{(\omega)}$ can be collapsed into a single matrix: $W = \sum_{\omega \in \Omega \cup \{\varepsilon\}} W^{(\omega)}$

Computation of Z:

- Z is given by: $Z(x) = \sum_{y' \in \Omega^*} \exp(\text{score}(x, y'))$
  $= \sum_{y' \in \Omega^*} \exp\left(\log \sum_{\pi \in \Pi(y')} w(\pi)\right)$
  $= \sum_{y' \in \Sigma^*} \sum_{\pi \in \Pi(y')} \lambda(\text{p}(\pi)) \otimes w_I(\pi) \otimes \rho(\text{q}(\pi))$
  $= \sum_{y' \in \Sigma^*} \sum_{\pi \in \Pi(y')} \lambda(\text{p}(\pi)) \otimes \prod_{n=1}^{|\pi|} w_n(\pi) \otimes \rho(\text{q}(\pi))$ which is the pathsum
  $= \sum_{q_n, q_m \in Q} \left(\sum_{\pi \in \Pi(q_n, q_m)} w(\pi)\right)$
  $= \sum_{q_n, q_m \in Q} \sum_{\pi \in \Pi(q_n, q_m)} \lambda(\text{p}(\pi)) \otimes w_I(\pi) \otimes \rho(\text{q}(\pi))$
  $= \sum_{q_n, q_m \in Q} \sum_{\pi \in \Pi(q_n, q_m)} \lambda(q_n) \otimes w_I(\pi) \otimes \rho(q_m)$
  $= \sum_{q_n, q_m \in Q} \lambda(q_n) \otimes (\sum_{\pi \in \Pi(q_n, q_m)} w_I(\pi)) \otimes \rho(q_m)$
  $= \sum_{q_n, q_m \in Q} \lambda(q_n) \otimes (W^*)_{nm} \otimes \rho(q_m)$
  $= \lambda^\top W^* \rho$ where W^* can be computed via Lehmann's algorithm
- In this way, we can compute log likelihood for a training instance
- To go from a single training instance to the entire training dataset:
  $\sum_{n=1}^{N} \log p(y^{(n)} \mid x^{(n)}) = \sum_{n=1}^{N} \left[\text{score}_\theta(y^{(n)}, x^{(n)}) - \log Z_\theta(x^{(n)})\right]$

Inference: Floyd-Warshall algorithm —

- Aim: Find shortest distances between all pairs of nodes in a graph, where the weights of the edges in the path are additive
- Corresponds to Lehmann algorithm weighted over the tropical semiring
- Requirement: No negative cycles in the graph, since if there were negative cycles, we could always loop over the cycle to lower the pathsum
- Based on that requirement, we can drop $(R_{jj}^{\leq j-1})^*$ in Lehmann algorithm

Lehmann's applications | Gauss-Jordan algorithm

- Aim: For matrix $M^{D \times D}$, find M^{-1}
- We know that, for matrix M Lehmann's algorithm computes M^*
- We know that $M^* = (I - M)^{-1}$
  Proof:
  - $\mathbf{M^* = I + MM^*}$
  - $\mathbf{M^* - MM^* = I}$
  - $\mathbf{M^*(I - M) = I}$
  - $\mathbf{M^* = (I - M)^{-1}}$
- Conversely, $(I - M)^* = M^{-1}$
  Proof:
  - $\mathbf{(I - M)^* = I + (I - M)(I - M)^*}$
  - $\mathbf{(I - M)^* - (I - M)(I - M)^* = I}$
  - $\mathbf{(I - M)^*(I - (I - M)) = I}$
  - $\mathbf{(I - M)^* = M^{-1}}$
- Thus, we can get $\mathbf{M^{-1}}$ by running Lehmann algorithm on $\mathbf{(I - M)}$ weighted over the inside field (instead of semiring, since a field gives us subtraction and division)

(W)FSA applications | N-gram, CRF, HMM

N-gram models —

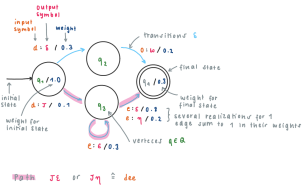
- Let $y = (y_1, \ldots, y_L)$ be a string with $y_1 = \text{BOS}$, $y_L = \text{EOS}$
- Let N-gram be of order N
- We want to model $p(y_n \mid y_{n-N+1}, \ldots, y_{n-1})$
- If we represent this with WFSA:
  - $\Sigma = V \cup \{\langle\text{BOS}\rangle, \langle\text{EOS}\rangle\}$
  - States:
    - Represent all N-grams
    - $Q = \bigcup_{n=0}^{N} \{\{\langle\text{BOS}\rangle\}^{N-n} \times V^{n-1} \times (V \cup \{\langle\text{EOS}\rangle\})\}$
      E.g.: If $V = \{\text{foo, bar}\}$:
      For bi-gram, we have the states BOS, foo, bar, EOS
      For tri-gram, we have the states BOSBOS, BOSfoo, BOSbar, foofoo, barbar, foobar, barfoo, fooEOS, barEOS, BOSEOS
    - $q_{ij\ldots}$ represents the state where the N-gram's most recent word $y_n = i$, the second-most recent word $y_{n-1} = j$, etc.
  - $I = \{\langle\text{BOS}, \ldots, \text{BOS}\rangle\}$ (N times) is a subset of Q at $n = 0$
  - $F = Q$
  - Transitions:
    - $\delta = \Big\{(y_{-N} \ldots y_{-1}, y_0, p(y_0 \mid y_{-N+1}, \ldots, y_{-1}), y_{-N+1} \ldots y_0)$ for $(y_{-N+1} \ldots y_{-1}) \in \bigcup_{n=0}^{N-1} \{\langle\text{BOS}\rangle\}^{N-1-n} \times V^n\Big\}$
    - $\delta(q_{ij\ldots}, y_n, 1, q_{ki\ldots}) =$
      $\begin{cases} \log(p(y_n = k \mid y_{n-1} = i, y_{n-2} = j, \ldots, y_{n-N+1} = \ldots)) \\ \quad \text{if } y_n = k \\ -\infty \text{ otherwise} \end{cases}$
    - Transitions are only defined if N-gram order ≥ 2
    - Only transitions between N-grams which can follow each other are allowed and receive weights $> \overline{0}$
    - The weight represents the probability of $p(y_n \mid y_{n-N+1}, \ldots, y_{n-1})$
  - Initial state:
    - $\lambda = \langle\text{BOS}\rangle \ldots \langle\text{BOS}\rangle \rightarrow \overline{1}$ (N times) i.e. state corresponding to full padding of BOS receives weight $\overline{1}$
    - $\lambda(q_{ij\ldots}) = \log(p(y_n = i \mid y_{n-1} = \text{BOS}, y_{n-2} = \text{BOS}, \ldots, y_{n-N+1} = \text{BOS}))$
  - Final state:

  - $\rho = y_{n-N+1} \dots y_1 \langle \text{EOS} \rangle \to \overline{1}$ i.e. state ending with EOS receives weight $\overline{1}$
  - $\rho(q_{ij\dots}) = \log(p(y_n = \text{EOS} \mid y_{n-1} = i, y_{n-2} = j, \dots, y_{n-N+1} = \dots))$
- We can compute $p(\boldsymbol{y}) = \lambda(q_I) + \sum_{n=1}^{L} \delta(q_{y_{n-N},\dots,y_{n-1}}, y_n, q_{y_{n-N+1},\dots,y_n}) + \rho(q_F)$
- For a vocabulary V, there are $x + |V|^{N-1} + z$ states:
  * $|V|^{N-1}$ intermediate states with full N-gram history
  * $x = \begin{cases} |V|^{N-2} & \text{for N-grams with } N \geq 2 \\ 0 & \text{for unigrams} \end{cases}$ start states with BOS padding
  * $z = \begin{cases} 1 & \text{for N-grams with } N \geq 2 \\ 0 & \text{for unigrams} \end{cases}$ end states

CRFs —
- Let $|\boldsymbol{x}| = L$ and $|\boldsymbol{y}| = M$
- We want to model $p(\boldsymbol{y} \mid \boldsymbol{x}) = \frac{\exp(\text{score}(\boldsymbol{y},\boldsymbol{x}))}{Z(\boldsymbol{x})} = \frac{\prod_{n=2}^{M} \exp(\boldsymbol{w} \cdot \boldsymbol{f}(y_n, y_{n-1}, \boldsymbol{x}, n))}{\sum_{\boldsymbol{y}'} \exp(\text{score}(\boldsymbol{y}', \boldsymbol{x})) = \sum_{\boldsymbol{y}'} \prod_{n=2}^{M} \exp(\boldsymbol{w} \cdot \boldsymbol{f}(y'_n, y'_{n-1}, \boldsymbol{x}, n))}$ where
  * $\text{score}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{w} \cdot \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y})$
  * We assume $\boldsymbol{f}$ decomposes additively over bi-grams: $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{n=2}^{M} f(y_n, y_{n-1}, \boldsymbol{x}, n)$
- If we represent this with WFSA:
  * $\Sigma = \{\varepsilon\}$
  * States $Q = V^M$ represent all hidden states y
  * $I = Q$
  * $F = Q$
  * Transitions $\delta = \{(y_n, \varepsilon, \exp(\boldsymbol{w} \cdot f(y_n, y_m, \boldsymbol{x})), y_m)\}$ with weights representing CRF scores assigned to hidden state y_m after hidden state y_n, given observable input sequence $\boldsymbol{x}$
  * $\lambda = y_n \to \exp(\boldsymbol{w} \cdot f(y_n, \langle \text{BOS} \rangle, \boldsymbol{x}))$
  * $\rho = y_n \to \exp(\boldsymbol{w} \cdot f(\langle \text{EOS} \rangle, y_n, \boldsymbol{x}))$

HMMs — TBA

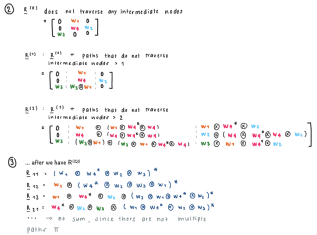
1) WFSA



2) WFST Composition

form: Adjacency matrix

3) Represent WFSA in matrix
4) Lehmann's algorithm: Compute pathsum



32 Reinforcement Learning for NLP

Formulation

S: States
- $s = \mathcal{X} \times \mathcal{Y} = \{\boldsymbol{x}, \tilde{\boldsymbol{y}}\}$, where $\boldsymbol{x}$ is input and $\tilde{\boldsymbol{y}}$ is partial output

A: Actions
- Elementary modifications, e.g., adding a single edge to a dependency tree

T: Transition function
- Specifies which state the agent will end up in if it commits a certain action in a certain state
- Transition corresponds to replacing the current partial output $\tilde{\boldsymbol{y}}$ with transformed partial output $a(\tilde{\boldsymbol{y}})$ after applying action a: $T((\boldsymbol{x}, \tilde{\boldsymbol{y}}), a) = (\boldsymbol{x}, a(\tilde{\boldsymbol{y}}))$

r: Reward function
- Specifies the reward the agent obtains by committing a certain action in a certain state: $r(s = (\boldsymbol{x}, a(\tilde{\boldsymbol{y}}))) = -\big(\Delta(a(\tilde{\boldsymbol{y}}), \boldsymbol{y}) - \Delta(\tilde{\boldsymbol{y}}, \boldsymbol{y})\big)$ i.e. -(loss after action - loss before action) where:
  * $\Delta(a(\tilde{\boldsymbol{y}}), \boldsymbol{y})$ is the loss between the true output $\boldsymbol{y}$ and the new prediction after applying action a to $\tilde{\boldsymbol{y}}$
  * $\Delta(\tilde{\boldsymbol{y}}, \boldsymbol{y})$ is the loss between the true output $\boldsymbol{y}$ and the current prediction $\tilde{\boldsymbol{y}}$
- Intuitively:
  * If loss after action > loss before action, r is negative
  * Otherwise, r is positive

π: Policy
- Set of rules that tell the agent which action to take in a certain state
- Policy score: Distinguishes good from bad policies: $\eta(\pi, s) = \sum_{t=1}^{T} r(s_t, \pi(s_t)) \mid \pi, s_1 = s$ where:
  * Total reward for policy starting at $s = s_1$ is calculated by summing the reward at each time step
  * $\pi(s_t)$ maps action a_t to state s_t
  * $r(s_t, \pi(s_t))$ is the reward of taking action a_t in state s_t
  * T is the maximum number of steps allowed to transition from s_1 to a final valid output

Optimization

Optimal policy search: $\pi^* = argmax_{\theta \in \Theta} \mathbb{E}_{s \sim \mathcal{D}}[\eta(\pi_\theta, s)]$