Regression

Terminology

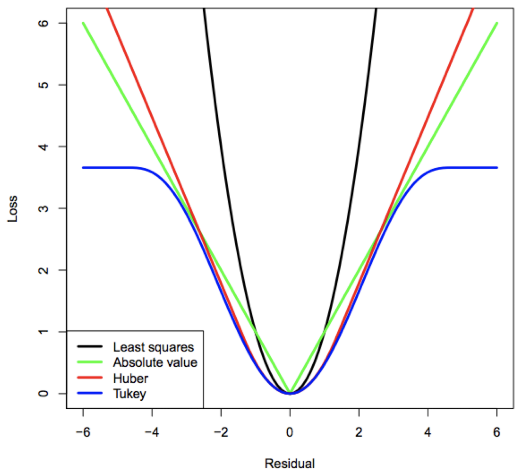
- Data consists of **pairs** $(\mathbf{x}_n, y_n)$, where y_n is the n'th output and x_n is a vector of D inputs. The number of pairs N is the data-size and D is the dimensionality.
- Two goals of regression: **prediction** and **interpretation**
- The regression function: $y_n \approx f_w(\mathbf{x}_n)\ \forall n$
- Regression finds correlation not a causal relationship.
- **Input variables** a.k.a. covariates, independent variables, explanatory variables, exogenous variables, predictors, regressors.
- **Output variables** a.k.a. target, label, response, outcome, dependent variable, endogenous variables, measured variable, regressands.

Linear Regression

- Assumes linear relationship between inputs and output.
- $y_n \approx f(\mathbf{x}_n) := w_0 + w_1 x_{n1} + ... + w_D x_{nD}$ $:= \tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}$ contain the additional offset term (a.k.a. bias).
- Given data we learn the weights $\mathbf{w}$ (a.k.a. estimate or fit the model)
- Overparameterisation $D > N$ eg. univariate linear regression with a single data point $y_1 \approx w_0 + w_1 x_{11}$. This makes the task under-determined (no unique solution).

Loss Functions $\mathcal{L}$

- A loss function (a.k.a. energy, cost, training objective) quantifies how well the model does (how costly its mistakes are).
- $y \in \mathbb{R} \Rightarrow$ desirable for cost to be symmetric around 0 since $\pm$ errors should be penalized equally.
- Cost function should penalize "large" mistakes and "very large" mistakes similarly to be robust to outliers.
- Mean Squared Error:
$\mathsf{MSE}(\mathbf{w}) := \frac{1}{N}\sum_{n=1}^{N}[y_n - f_{\mathrm{w}}(\mathbf{x}_n)]^2$
not robust to outliers.
- Mean Absolute Error:
$\mathsf{MAE}(\mathbf{w}) := \frac{1}{N}\sum_{n=1}^{N}|y_n - f_{\mathrm{w}}(\mathbf{x}_n)|$
- Convexity: a function is convex iff a line segment between two points on the function's graph always lies above the function.
- Convexity: a function $h(\mathbf{u}), \mathbf{u} \in \mathbb{R}^D$ is convex if $\forall\ \mathbf{u}, \mathbf{v} \in \mathbb{R}^D, 0 \le \lambda \le 1$:
$h(\lambda\mathbf{u} + (1-\lambda)\mathbf{v}) \le \lambda h(\mathbf{u}) + (1-\lambda)h(\mathbf{v})$
Strictly convex if $\le \Rightarrow <$
- Convexity, a desired computational property: A strictly convex function has a unique global minimum $\mathbf{w}^*$. For convex functions, every local minimum is a global minimum.
- Sums of convex functions are also convex $\Rightarrow$ MSE combined with a linear model is convex in $\mathbf{w}$.
- Proof of convexity for MAE:
$\mathsf{MAE}(\mathbf{w}) := \frac{1}{N}\sum_{n=1}^{N}\mathcal{L}_n(\mathbf{w}), \mathcal{L}_n(\mathbf{w}) = |y_n - f_{\mathrm{w}}(\mathbf{x}_n)|$
$\mathcal{L}_n(\lambda w_1 + (1-\lambda)w_2) \le \lambda\mathcal{L}_n(w_1) + (1-\lambda)\mathcal{L}_n(w_2)$
$|y_n - x_n^T(\lambda w_1 + (1-\lambda)w_2)| \le \lambda|y_n - x_n^T w_1| + (1-\lambda)|y_n - x_n^T w_2|$
$(1-\lambda) \ge 0 \Rightarrow (1-\lambda)|y_n - x_n^T w_2| = |(1-\lambda)y_n - (1-\lambda)x_n^T w_2|$
$a = \lambda y_n - \lambda x_n^T w_1, b = (1-\lambda)y_n - (1-\lambda)x_n^T w_2$
$a + b = y_n - x_n^T(\lambda w_1 + (1-\lambda)w_2)$
$|a+b| \le |a| + |b| \Rightarrow \mathcal{L}_n(\mathbf{w})$ convex $\Rightarrow \mathsf{MAE}(\mathbf{w})$ convex
- Huber loss:
$\mathcal{H}uber(e) := \begin{cases} \frac{1}{2}e^2 & ,\text{if } |e| \le \delta \\ \delta|e| - \frac{1}{2}\delta^2 & ,\text{if } |e| > \delta \end{cases}$ convex, differentiable, and robust to outliers but setting δ is not easy.
- Tukey's bisquare loss:
$\frac{\partial\mathcal{L}}{\partial e} := \begin{cases} e\{1 - e^2/\delta^2\}^2 & ,\text{if } |e| \le \delta \\ 0 & ,\text{if } |e| > \delta \end{cases}$ non-convex, but robust to outliers.



Optimisation

- Given $\mathcal{L}(\mathbf{w})$ we want $\mathbf{w}^* \in \mathbb{R}^D$ which minimises the cost: $\min_{\mathbf{w}} \mathcal{L}(\mathbf{w}) \rightarrow$ formulated as an optimisation problem
- Local minimum $\mathbf{w}^* \Rightarrow \exists\epsilon > 0$ s.t. $\mathcal{L}(\mathbf{w}^*) \le \mathcal{L}(\mathbf{w})\ \forall\mathbf{w}$ with $\|\mathbf{w} - \mathbf{w}^*\| < \epsilon$
- Global minimum $\mathbf{w}^*$, $\mathcal{L}(\mathbf{w}^*) \le \mathcal{L}(\mathbf{w})\ \forall\mathbf{w} \in \mathbb{R}^D$

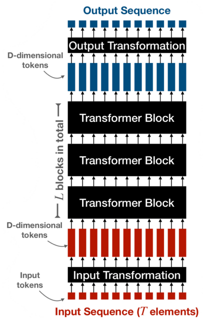
Smooth Optimisation

Transformers

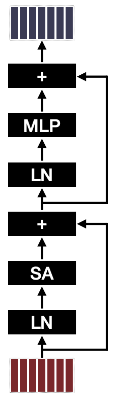
- A transformer is a neural network that iteratively transforms a sequence to another sequence and mixes the information between the sequence elements via self-attention.

Architecture

- Self-Attention (SA): mixes information between tokens
- Multi-Layer Perceptron (MLP): mixes information within each token
- Skip connections are widely used
- Layer normalization (LN) is usually placed at the start of a residual branch



Text Token Embeddings

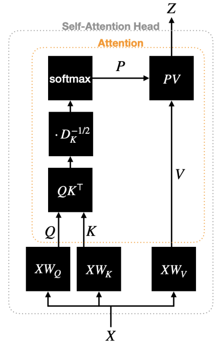


- Tokenization: split the input text into a sequence of input tokens (typically word fragments + some special symbols) according to some predefined tokenizer procedure:
- Convert each token ID $i \in \{1, ..., N_{vocab}\}$ into a real-valued vector $\mathbf{w}_i \in \mathbb{R}^D$
- This can be seen as a matrix multiplication $\mathbf{W} \cdot \mathbf{e}_i = \mathbf{W}_{:,i} = \mathbf{w}_i$ (with $\mathbf{W} \in \mathbb{R}^{\mathrm{D} \times \mathrm{N_{vocab}}}$)
- $\mathbf{W}$ is learned via backpropagation, along with all other transformer parameters (however, the tokenizer procedure is typically fixed in advance and not learned)
- The whole input sequence of T tokens leads to an input matrix $X \in \mathbb{R}^{T \times D}$

Attention

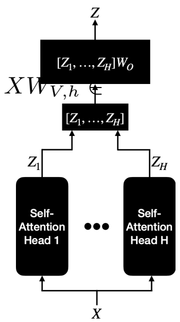
- Attention is a function that transforms a sequence of tokens to a new sequence of tokens using a learned input-dependent weighted average
- Input tokens : $V \in \mathbb{R}^{T_{in} \times D}$
- Output tokens : $Z \in \mathbb{R}^{T_{out} \times D}$
- Output tokens are simply a weighted average of the input tokens: $z_i = \sum_{j=1}^{T_i} p_{ij} v_j$ i.e. $Z = PV$
- Weighting coefficients $\mathcal{P} \in [0,1]^{T_{out} \times T_{in}}$ form valid probability distributions over the input tokens $\sum_{j=1}^{T_{in}} p_{ij} = 1$
- Query tokens : $Q \in \mathbb{R}^{T_{out} \times D_K}$
- Key tokens : $K \in \mathbb{R}^{T_{in} \times D_K}$
- Determine weight $p_{i,j}$ based on how simmilar q_i and k_j are.
- Use inner product to obtain raw similarity scores.
- Normalize with softmax (scaled the temperature by $\sqrt{D_K}$) to obtain a probability distribution.
- $P = \text{softmax}\left(\frac{QK^\top}{\sqrt{D_K}}\right)$ The softmax is applied on each row independently. Scaling ensures uniformity at initialization and faster convergence

Self-Attention



- V, K, Q are all derived from the same input token sequence $X \in \mathbb{R}^{T \times D}$
- Values : $V = XW_V \in \mathbb{R}^{T \times D}$, $W_V \in \mathbb{R}^{D \times D}$
- Keys : $K = XW_K \in \mathbb{R}^{T \times D_K}$, $W_K \in \mathbb{R}^{D \times D_K}$
- Queries : $Q = XW_Q \in \mathbb{R}^{T \times D_K}, W_Q \in \mathbb{R}^{D \times D_K}$
- W_Q, W_V, W_K are learned parameters.
- $Z = \text{softmax}\left(\frac{XW_Q W_K^T X^T}{\sqrt{D_K}}\right) XW_V$

Multi-Head Self-Attention



- Run H Self-Attention "heads" in parallel
- $Z_h = \text{softmax}\left(\frac{XW_{Q,h}W_{K,h}^\top X^\top}{\sqrt{D_K}}\right) XW_{V,h} \in \mathbb{R}^{T \times D_V}$
- $W_{V,h} \in \mathbb{R}^{D \times D_V}$, $W_{K,h} \in \mathbb{R}^{D \times D_K}$, $W_{Q,h} \in \mathbb{R}^{D \times D_K}$
- The final output is obtained by concatenating head-outputs and applying a linear transformation $Z = [Z_1, \ldots, Z_H]W_o$ where $W_O \in \mathbb{R}^{HD_V \times D}$ is learned via backpropagation

Positional Information

- Attention by itself does not account for the order of input
- incorporate a positional encoding in the network which is a function from the position to a feature vector $pos : \{1, ..., T\} \rightarrow \mathbb{R}^D$
- The most basic choice is to add a positional embedding W_{pos} corresponding to each token's position t to the input embedding. $W_{pos} \in \mathbb{R}^{D \times T}$ is learned via backpropagation along with the other parameters

MLP

- Mixing Information within Tokens
- Apply the same transformation to each token independently: $MLP(X) = \varphi(XW_1)W_2$
- $W_1, W_2 \in \mathbb{R}^{D \times D}$ learned via backprop

Output Transformations

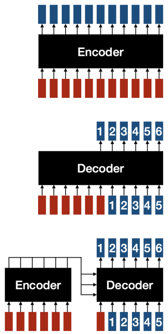
- typically simple: linear transformation or a small MLP

- dependent on the task: Single output (e.g.,sequence-level classification): apply an output transformation to a special taskspecific input token or to the average of all tokens. Multiple outputs (e.g., per-token classification): apply an output transformation to each token independently

Vision Transformer Architecture

- Self-attention is more general than convolution and can potentially express it
- The receptive field is the whole image after just one self-attention layer
- ViTs require more data than CNNs due to their reduced inductive bias in extracting local features
- In many cases, the model attends to image regions that are semantically relevant for classification

Encoders & Decoders



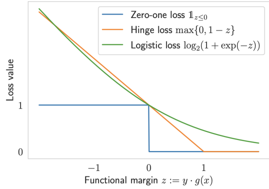
- Encoders (e.g., classification): They produce a fixed output size and process all inputs simultaneously
- Decoders (e.g., ChatGPT): Auto-regressively sample the next token as $x_{t+1} \sim sofimax(f(x_1, ..., x_t))$ and use it as new input token. Capable of generating responses of arbitrary length.
- Encoder-decoder (e.g., translation): First encode the whole input (e.g., in one language) and then decode to token by token (e.g., in a different language)

Adversarial ML

- We don't understand how NN models generalize and react to shifts in the distribution of data (i.e., distribution shifts)
- Classification problem: $(X, Y) \sim \mathcal{D}$, Y with range $\{-1, 1\}$
- Standard risk: average zero-one loss over X: $R(f) = \mathbb{E}_{\mathcal{D}}\left[1_{f(X) \neq Y}\right] = \mathbb{P}_{\mathcal{D}}\left[f(X) \neq Y\right]$ i.e. minimise proba of wrong prediction.
- Adversarial risk: average zero-one loss over small, worst-case perturbations of X: $R_\varepsilon(f) = \mathbb{E}_{\mathcal{D}}\left[\max_{\hat{x}, \|\hat{x}-X\| \leq \varepsilon} 1_{f(\hat{x}) \neq Y}\right]$

Generating adversarial examples

- Task: given an input (x, y) and a model $f : \mathcal{X} \to \{-1, 1\}$ find an input $\hat{x}$ s.t.: a) $\|\hat{x} - x\| \leq \varepsilon$ b) the model f makes a mistake on it.
- Trivial case: x already missclassified → no action required
- General case: find $\hat{x}$ such that at $f(\hat{x}) \neq y$ and $||\hat{x} - x|| \leq \varepsilon$ i.e. $\hat{x} \in B_x(\varepsilon) \cap \{x' f(x') = -y\}$
- Optimization problem with respect to the inputs
- Problem: optimizing the indicator function is difficult: 1) The indicator function 1 is not continuous 2) The NN prediction f outputs discrete class values $\{-1, 1\}$
- Replace the difficult problem involving the indicator with a smooth problem $\max_{\hat{x}, \|\hat{x}-X\| \leq \varepsilon} 1_{f(\hat{x}) \neq Y} \to \max_{\hat{x}, \|\hat{x}-X\| \leq \varepsilon} \ell(yg(\hat{x}))$
- decreasing, margin-based (i.e., dependent on $y * g(x)$) classification losses



White-Box attacks

- Solve $\max_{\hat{x}, \|\hat{x}-X\| \leq \varepsilon} \ell(yg(\hat{x}))$ knowing g
- $\nabla_x \ell(yg(x)) = y\ell'(yg(x)) \nabla_x g(x)$, with $y\ell'(yg(x)) \leq 0$ since classification losses are decreasing.
- Move in direction of $\propto -y \nabla_x g(x)$
- Interpretation $f(x) = \text{sign}(g(x))$: If $y = 1$ we want to decrease $g(x)$ and follow $-\nabla_x g(x)$. If $y = -1$ we want to decrease $g(x)$ and follow $\nabla_x g(x)$
- By using ℓ and not directly $yg(\hat{x})$ it will extend to multi-class classification and robust training.
- linearize the loss $\tilde{\ell}(x) := \ell(yg(x))$

$\max_{\|\hat{x}-x\| \leq \varepsilon} \tilde{\ell}(x)$
$\approx \max_{\|\hat{x}-x\| \leq \varepsilon} \tilde{\ell}(x) + \nabla_x \tilde{\ell}(x)^T (\hat{x} - x)$
$= \tilde{\ell}(x) + \max_{\|\hat{x}-x\| \leq \varepsilon} \nabla_x \tilde{\ell}(x)^T (\hat{x} - x)$
$= \tilde{\ell}(x) + \max_{\|\delta\| \leq \varepsilon} \nabla_x \tilde{\ell}(x)^T \delta$

- We need to maximize the inner product under a norm constraint, i.e. find the optimal local update
- This is a simple problem for which we can get a closed-form solution depending on the norm used to measure the perturbation size $||\delta||$

One-step attack

- Solution for the ℓ_2 norm:

$\delta_2^\star = \varepsilon \cdot \frac{\nabla_x \tilde{\ell}(x)}{||\nabla_x \tilde{\ell}(x)||_2} = -\varepsilon y * \frac{\nabla_x g(x)}{||\nabla_x g(x)||_2} \Rightarrow$
$\hat{x} = x - \varepsilon y \cdot \frac{\nabla_x g(x)}{\|\nabla_x g(x)\|_2}$

- Solution for the ℓ_∞ norm called **Fast Gradient Sign Method**:

$\delta_\infty^\star = \varepsilon \cdot \text{sign}(\nabla_x \tilde{\ell}(x)) = -\varepsilon y \cdot \text{sign}(\nabla_x g(x)) \Rightarrow$
$\hat{x} = x - \varepsilon y \cdot \text{sign}(\nabla_x g(x))$

Multi-step attack

- These updates can be done iteratively and combined with a projection Π on the feasible set (i.e., balls ℓ_2 / ℓ_∞ here)
- Projected Gradient Descent (PGD attack)
- ℓ_2 norm:

$\delta^{t+1} = \Pi_{B_2(e)}[\delta^t + \alpha \cdot \frac{\nabla \tilde{\ell}(x+\delta^t)}{\|\nabla \tilde{\ell}(x+\delta^t)\|_2}]$

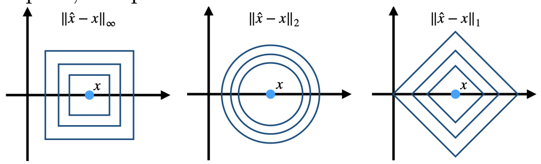
$$\Pi_{B_2(\varepsilon)}(\delta) = \begin{cases} \varepsilon \cdot \delta / \|\delta\|_2, & \text{if } \|\delta\|_2 \geq \varepsilon \\ \delta \text{ otherwise} \end{cases}$$

- ℓ_∞ norm:

$$\delta^{t+1} = \Pi_{B_\infty(\varepsilon)}\left[\delta^t + \alpha \cdot \text{sign}(\nabla \tilde{\ell}(x + \delta^t))\right],$$

$$\Pi_{B_\infty(\varepsilon)}(\delta)_i = \begin{cases} \varepsilon \cdot \text{sign}(\delta_i), & \text{if } |\delta_i| \geq \varepsilon \\ \delta_i \text{ otherwise} \end{cases}$$

- the gradients are computed by backprop w.r.t. inputs, not parameters!



Black-box attacks

- We don't know $g(x)$
- Obtaining a surrogate model can be costly and there is no guarantee of success
- Query-based methods often require a lot of queries (10k-100k), easy to restrict access for the attacker!

Query-based gradient estimation

- Score-based: we can query the continuous model scores $g(x) \in \mathbb{R}$. We can approximate the gradient by using the finite difference formula: $\nabla_x g(x) \approx \sum_{i=1}^d \frac{g(x+\alpha e_i) - g(x)}{\alpha} e_i$
- Decision-based: we can query only the predicted class $f(x) \in \{-1, 1\}$, similar techniques can be adapted for the decision-based case.

Transfer Attacks

- Train a similar surrogate model $\hat{f} \approx f$ on similar data
- Model stealing (query f given some unlabeled inputs $\{x_n, f(x_n)\}_{n=1}^N$) can facilitate transfer attacks.

Adversarial training

- Adversarial training: the goal is to minimize the adversarial risk:

$\min_\theta R_\varepsilon(f_\theta) = \mathbb{E}_{\mathcal{D}}\left[\max_{\hat{x}, \|\hat{x}-X\| \leq \varepsilon} 1_{f(\hat{x}) \neq Y}\right]$

- $\mathcal{D}$ unknown → approximate it with a sample average + classification loss is non-continuous → use a smooth loss ⇒

$\min_\theta \frac{1}{N} \sum_{n=1}^N \max_{\hat{x}_n, \|x_n - \hat{x}_n\| \leq \varepsilon} \ell(y_n g_\theta(\hat{x}_n))$

1) $\forall x_n$, $\hat{x}_n^\star \approx \arg\max_{||x_n - \hat{x}_n| \leq \varepsilon} \ell(y_n g_\theta(\hat{x}_n))$
2) GD step w.r.t. θ using $\frac{1}{N} \sum_{n=1}^N \nabla_\theta \ell(y_n g_\theta(\hat{x}_n^\star))$

Advantages

- state-of-the-art approach for robust classification
- more interpretable gradients
- fully compatible with SGD

Disadvantages

- Increased computational time: proportional to the number of PGD steps
- Robustness-accuracy tradeoff: using too large ε leads to worse standard accuracy

Adversarial Example

$x \in \mathbb{R}^d, y \sim Bernoulli(\{-1, 1\}), Z_i \sim \mathcal{N}(0, 1)$

- Robust features: $x_1 = y + Z_1$
- Non-robust features: $x_i = y\sqrt{\frac{\log d}{d-1}} + Z_i$, $\forall i \in \{-1, 1\}$
- $d \to \infty \Rightarrow \uparrow$ adversarial risk and $\downarrow$ standard risk
- using the robust feature x_1:

MLE: $\arg\max_{\hat{y} \in \{\pm 1\}} p(\hat{y} \mid x_1) =$
$\arg\max_{\hat{y} \in \{\pm 1\}} \frac{p(x_1|\hat{y})p(\hat{y})}{p(x_1)} = \arg\max_{\hat{y} \in \{\pm 1\}} p(x_1 \mid \hat{y})$ assuming $p(y = 1) = p(y = -1)$

- Standard Risk: $\int_0^\infty \frac{1}{\sqrt{2\pi}} e^{-0.5(x+1)^2} dx \approx 0.16$ good but not perfect!
- using both robust and non-robust features:

MLE for all features $x_i = ya_i + Z_i$
$\arg\max_{\hat{y} \in \{\pm 1\}} p(\hat{y} \mid x)$
$= \arg\max_{\hat{y} \in \{\pm 1\}} \prod_{i=1}^d p(x_i \mid \hat{y})$
$= \arg\max_{\hat{y} \in \{\pm 1\}} \sum_{i=1}^d \log p(x_i \mid \hat{y})$
$= \arg\max_{\hat{y} \in \{\pm 1\}} \sum_{i=1}^d \log \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x_i - \hat{y}a_i)^2}$
$= \arg\min_{\hat{y} \in \{\pm 1\}} \sum_{i=1}^d (x_i - \hat{y}a_i)^2$
$= \arg\min_{\hat{y} \in \{\pm 1\}} \sum_{i=1}^d (x_i^2 - 2x_i\hat{y}a_i + \hat{y}^2 a_i^2)$
$= \arg\max_{\hat{y} \in \{\pm 1\}} \hat{y} \sum_{i=1}^d x_i a_i$
$\hat{y} \sum_{i=1}^d x_i a_i = \hat{y}y(\sum_{i=1}^d a_i^2) + \hat{y} \sum_{i=1}^d a_i Z_i = \hat{y}y(1 + \log(d)) + \hat{y}Z$ where $Z := \sum_{i=1}^d a_i Z_i \sim \mathcal{N}(0, 1 + \log d)$

Scaling by $1/(1 + \log d)$ the MLE results in: $y\hat{y} + \hat{y}Z$ with $Z \sim \mathcal{N}(0, 1/(1 + \log d))$
$d \to \infty, \hat{y}Z \to 0 \Rightarrow$ standard risk $R(f) \to 0$

- using the non-robust features improves standard risk!
- Adversarial risk:

The adversary can use tiny ℓ_∞ perturbations:
$\varepsilon = 2\sqrt{\frac{\log d}{d-1}}$ ($\to 0$ when) $d \to \infty$)
$\hat{x}_1 = \left(1 - 2\sqrt{\frac{\log d}{d-1}}\right) y + Z_1$, almost unaffected
$\hat{x}_i = -\sqrt{\frac{\log d}{d-1}} y + Z_i$, completely flipped
$R_\varepsilon(f) \approx 1 \Rightarrow$ tradeoff between accuracy and robustness.

Matrix Factorization

Given items (movies) $d = 1, 2, ..., D$ and users $n = 1, 2, ..., N$, we define X to be the $D \times N$ matrix containing all rating entries. That is, x_{dn} is the rating of n-th user for d-th item. Note that most ratings x_{dn} are missing, and our task is to predict them accurately.

Algorithm

$X \approx WZ^T$, $W \in \mathbb{R}^{D\times K}$, $Z \in \mathbb{R}^{N\times K}$ tall matrices $K << N, D$

$\min_{W,Z} \mathcal{L}(W,Z) := \frac{1}{2}\sum_{(d,n)\in\mathbb{Q}}[x_{dn} - (WZ^T)_{dn}]^2$

- We hope to "explain" each rating x_{dn} by a numerical representation of the corresponding item and user - in fact by the inner product of an item feature vector with the user feature vector.
- The set $\Omega \subseteq [D] \times [N]$ collects the indices of the observed ratings of the input matrix X.
- This cost is not jointly convex w.r.t. W and Z, nor identifiable as $(w^*, z^*) \Leftrightarrow (\beta w^*, \beta^{-1} z^*)$

Choosing K

- $\uparrow K \Rightarrow$ overfitting ($\Leftrightarrow \downarrow K \Rightarrow$ underfitting). For $K >> N, D \Rightarrow (W^*, {Z^*}^T) = (X, I) = (I, X)$

Regularization

$\frac{1}{2}\sum_{(d,n)\in\Omega}[x_{dn} - (WZ^T)_{dn}]^2 + \frac{\lambda_w}{2}||W||^2_{\text{Frob}} + \frac{\lambda_z}{2}||Z||^2_{\text{Frob}}$ $\lambda_w, \lambda_z \in \mathbb{R} > 0$

Stochastic Gradient Descent

$\mathcal{L} = \frac{1}{|\Omega|}\sum_{(d,n)\in\Omega} \underbrace{\frac{1}{2}[x_{dn} - (\mathbf{W}\mathbf{Z}^\mathsf{T})_{dn}]^2}_{f_{d,n}}$

For one fixed element (d, n) of the sum, we derive the gradient entry (d', k) for W:

$\frac{\partial}{\partial w_{d',k}} f_{d,n}(W,Z) \in \mathbb{R}^{D\times K} =$
$\begin{cases} -\left[x_{dn} - (\mathbf{W}\mathbf{Z}^T)_{dn}\right] z_{n,k} & \text{if } d' = d \\ 0 \text{ otherwise} \end{cases}$

$\frac{\partial}{\partial z_{n',k}} f_{d,n}(W,Z) \in \mathbb{R}^{N\times K} =$
$\begin{cases} -\left[x_{dn} - (\mathbf{W}\mathbf{Z}^T)_{dn}\right] w_{d,k} & \text{if } n' = n \\ 0 \text{ otherwise} \end{cases}$

- cost: $\Theta(K)$ which is cheap!

Alternating Least Squares

- No missing entries:
$\frac{1}{2}\sum_{d=1}^{D}\sum_{n=1}^{N}\left[x_{dn} - \left(W\mathbf{Z}^\mathsf{T}\right)_{dn}\right]^2$
$= \frac{1}{2}\|\mathbf{X} - \mathbf{W}\mathbf{Z}^\mathsf{T}\|^2_{Frob}$

- We first minimize w.r.t. Z for fixed W and then minimize W given Z (closed form solutions):
$Z^\mathsf{T} := (\mathsf{W}^\mathsf{T}\mathsf{W} + \lambda_z \mathsf{I}_K)^{-1}\mathsf{W}^\mathsf{T}\mathsf{X}$
$\mathbf{W}^\mathsf{T} := (\mathbf{Z}^\mathsf{T}\mathbf{Z} + \lambda_w \mathbf{I}_K)^{-1}\mathbf{Z}^\mathsf{T}\mathbf{X}^\mathsf{T}$
- Cost: need to invert a $K \times K$ matrix
- With missing entries: Can you derive the ALS updates for the more general setting, when only the ratings $(d,n) \in \Omega$ contribute to the cost, i.e.
$\frac{1}{2}\sum_{(d,n)\in\Omega}\left[x_{dn} - (\mathsf{W}\mathsf{Z}^\mathsf{T})_{dn}\right]^2$
Compute the gradient with respect to each group of variables, and set to zero.

Text Representation

-Finding numerical representations for words is fundamental for all machine learning methods dealing with text data.

-Goal: For each word, find mapping (embedding) $w_i \mapsto \mathbf{w}_i \in \mathbb{R}^K$

Co-Occurence Matrix

-A big corpus of un-labeled text can be represented as the co-occurrence counts. n_{ij} := #contexts where word w_i occurs together with word w_j.
- Needs definition of Context e.g. document, paragraph, sentence, window and Vocabulary $\mathcal{V} := \{w_1, \ldots, w_D\}$
- For words $w_d = 1, 2, \ldots, D$ and context words $w_n = 1, 2, \ldots, N$, the co-occurence counts n_{ij} form a very sparse $D \times N$ matrix.

Learning Word-Representations

- Find a factorization of the cooccurence matrix!
- Typically uses log of the actual counts, i.e. $x_{dn} := \log(n_{dn})$.
- Aim to find $\mathbf{W}, \mathbf{Z}$ s.t. $\mathbf{X} \approx \mathbf{W}\mathbf{Z}^\top$
$\min_{\mathbf{W},\mathbf{Z}} \mathcal{L}(\mathbf{W},\mathbf{Z}) :=$
$\frac{1}{2}\sum_{(d,n)\in\Omega} f_{dn}\left[x_{dn} - \left(\mathbf{W}\mathbf{Z}^\top\right)dn\right]^2$
where $\mathbf{W} \in \mathbb{R}^{D\times K}$, $\mathbf{Z} \in \mathbb{R}^{N\times K}$, $K \ll D, N$, $\Omega \subseteq [D] \times [N]$ indices of non-zeros of the count matrix $\mathbf{X}$, f_{dn} are weights to each entry.

GloVe

A variant of word2vec.
$f_{dn} := \min\{1, (n_{dn}/n_{\max})^\alpha\}$, $\alpha \in [0;1]$ (e.g. $\alpha = \frac{3}{4}$)

Note: Choosing K; K e.g. 50, 100, 500

Training

- Stochastic Gradient Descent (SGD) ($\Theta(K)$ per step $\rightarrow$ easily parralelizable)
- Alternating Least-Squares (ALS)

Skip-Gram Model

- Uses binary classification (logistic regression objective), to separate real word pairs (w_d, w_n) from fake word pairs. Same inner product score = matrix factorization.
- Given w_d, a context word w_n is:
real = appearing together in a context window of size 5
fake = any word $w_{n'}$ sampled randomly: Negative sampling (also: Noise Contrastive Estimation)

Learning Representations of Sentences & Documents

- Supervised: For a supervised task (e.g. predicting the emotion of a tweet), we can use matrix factorization or CNNs. - Unsupervised:
Adding or averaging (fixed, given) word vectors,
Training word vectors such that adding/averaging works well
Direct unsupervised training for sentences (appearing together with context sentences) instead of words

Fast Text

Matrix factorization to learn document/sentence representations (supervised).
Given a sentence $s_n = (w_1, w_2, \ldots, w_m)$, let $\mathbf{x}_n \in \mathbb{R}^{|\mathcal{V}|}$ be the bag-of-words representation of the sentence.

$$\min_{\mathbf{W},\mathbf{Z}} \mathcal{L}(\mathbf{W},\mathbf{Z}) := \sum_{s_n \text{ a sentence}} f\left(y_n \mathbf{W}\mathbf{Z}^\top \mathbf{x}_n\right)$$

where $\mathbf{W} \in \mathbb{R}^{1\times K}, \mathbf{Z} \in \mathbb{R}^{|\mathcal{V}|\times K}$ are the variables, and the vector $\mathbf{x}_n \in \mathbb{R}^{|\mathcal{V}|}$ represents our n-th training sentence.
Here f is a linear classifier loss function, and $y_n \in \{\pm 1\}$ is the classification label for sentence $\mathbf{x}_n$.

Language Models

Selfsupervised training:

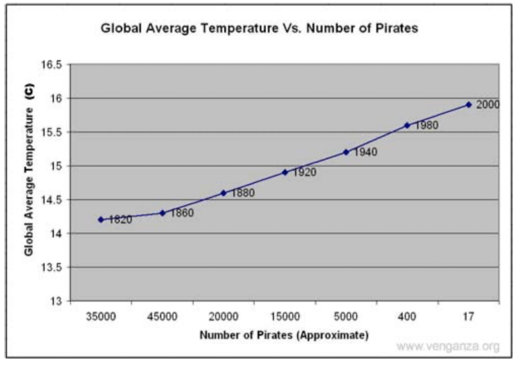
- Can a model generate text? train classifier to predict the continuation (next word) of given text
- Multi-class: Use soft-max loss function with a large number of classes D = vocabulary size
- Binary classification: Predict if next word is real or fake (i.e. as in word2vec)
- Impressive recent progress using large models, such as transformers

---

1 This is RGB red text.

For $x \in [r_{i-1}, r_i]$
$r(x) = \tilde{a}_1 x + \tilde{b}_1 + \sum_{j=2}^{m} \tilde{a}_j (x - \tilde{b}_j)_+$

For $x \in [r_{i-1}, r_i]$
$r(x) = \tilde{a}_1 x + \tilde{b}_1 + \sum_{j=2}^{m} \tilde{a}_j (x - \tilde{b}_j)_+$



Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

. . . .

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.
