Linear Regression

- $h(x) = \theta^T x = \theta_0 + \sum_{j=1}^{d} \theta_j x_j$
- $J(\theta) = \frac{1}{2n} \sum_{i=1}^{n} \left[h(x^{(i)}) - y^{(i)} \right]^2$
- $\frac{\partial J}{\partial \theta} = \frac{1}{n} X^T X \theta - X^T y$

GD

- $\theta \leftarrow \theta - \alpha \frac{1}{n} \sum_{i=1}^{n} (h(x^{(i)}) - y^{(i)}) x$
- $\theta \leftarrow \theta - \alpha \nabla J(\theta)$

SGD

- $\theta \leftarrow \theta - \alpha \nabla \ell(x^i, y^i, \theta)$ s.t. $\ell = (h-y)^2$
- $\theta \leftarrow \theta - \alpha \left[h(x^i) - y^i \right] x^i$

MBGD

- $\theta \leftarrow \theta - \alpha \frac{1}{b} \sum_{i \in B} \nabla \ell(x^i, y^i, \theta)$

Norm

- $\hat{\theta} = (X^T X)^{-1} X^T y$

Linear Basis Fcn.

- $h(x) = \theta^T \phi(x) = \sum_j \theta_j \phi_j(x)$
- Loss = MSE (same)

GD

- $\theta \leftarrow \theta - \alpha \frac{1}{n} \sum_i^n \left[h(x^i) - y^i \right] \phi(x^i)$
- $\hat{\theta} = \left[\phi(x)^T \phi(x) \right]^{-1} \phi(x) \vec{y}$

Generalization

K-fold CV

- $K = n \Rightarrow$ leave 1 out
- choose best

Ridge Regression

$J(\theta) = \frac{1}{2n} \sum_i^n \left[h(x^i) - y^i \right]^2 + \frac{\lambda}{2} \sum_{j=1}^{d} \theta_j^2$

$\sum_{j=1}^{d} \theta_j^2 = \| \theta_{1:d} \|_2^2$

GD L_2 regularized

- $\theta_0 \leftarrow \theta_0 - \alpha \frac{1}{n} \sum_i^n \left[h(x^i) - y^i \right]$
- $\theta_j \leftarrow \theta_j - \alpha \frac{1}{n} \sum_i^n \left[h(x^i) - y^i \right] x_j^i - \alpha \lambda \theta_j$
- $\bar{\theta} = \left[X^T X + \lambda I_{1:d} \right]^{-1} X^T y$

Perceptrons

- $h(x) = \text{sign}(\theta^T x) = \begin{cases} +1 & z > 0 \\ -1 & z \le 0 \end{cases}$

Surrogate Loss

- $J(\theta) = \frac{1}{n} \sum_i^n \max(0, -y^i \theta^T x^i)$

$h(x)^{-1}$ \ y	-1	1
1	0	-2
-1	2	0

- $\theta \leftarrow \theta - \frac{\alpha}{2} \left[h(x^i) - y^i \right] x^i$

Logistic Regression

- $h(x) = \sigma(\theta^T x)$ s.t. $\sigma(z) = \frac{1}{1+e^{-z}}$
- $\sigma'(z) = \sigma(z)(1 - \sigma(z))$

Iteration Loss

- $\ell(y^i, x^i, \theta) = \begin{cases} -\log\left[h(x^i) \right] & y^i = 1 \\ -\log\left[1 - h(x^i) \right] & y^i = 0 \end{cases}$

Candidate Loss

- $J(\theta) = -\sum_i^n y^i \log\left[h(x^i) \right] + (1 - y^i) \log\left[1 - h(x^i) \right]$

GD

- $\theta_0 \leftarrow \theta_0 - \alpha \sum_i^n \left(h(x^i) - y^i \right)$
- $\theta_j \leftarrow \theta_j - \alpha \sum_i^n \left(h(x^i) - y^i \right) x_j^i + \lambda \theta_j$

Matrices

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

$A^{-1}A = A^{-1}A = I \Rightarrow$ if 2×2 $A^{-1} = \frac{1}{|A|} A$

$(AB)^T = B^T A^T$ & $(ABC)^T = C^T B^T A^T$

Multi-Class Logistic Regression

- For 2 classes:

$$h_\theta(x) = \frac{1}{1+\exp(-\theta^T x)} = \frac{\exp(\theta^T x)}{1+\exp(\theta^T x)}$$

weight assigned to $y = 0$ | weight assigned to $y = 1$

- For C classes:

$$h^{(c)}_{\theta_{1:C}}(x) = p_{\theta_{1:C}}(y = c \mid x) = \frac{\exp(\theta_c^T x)}{\sum_{k=1}^{C} \exp(\theta_k^T x)}$$

- Here $\theta_c \in \mathbb{R}^{d+1}$ is a parameter vector for class $c \in \{1, \ldots, C\}$
- Hypothesis also called the **softmax** function
- Note that sum of class probabilities equals 1

18

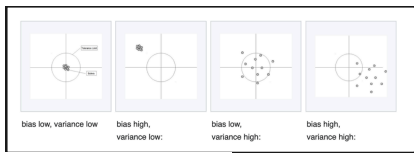
Multi-Class Logistic Regression

- The hypothesis for class c

$$h^{(c)}_{\theta_{1:C}}(x) = \frac{\exp(\theta_c^T x)}{\sum_{k=1}^{C} \exp(\theta_k^T x)} \Rightarrow C(d+1) = |\theta|$$

- Gradient descent simultaneously updates all parameters for c
  - Same derivative as before, just with the above $h^{(c)}_{\theta_{1:C}}(x)$
- Predict class label as the most probable label

$$\hat{y} = \arg\max_{c \in \{1, \ldots, C\}} h^{(c)}_{\theta_{1:C}}(x)$$



K-Nearest Neighbors

$h(x_i) = \text{majority}\left\{ y_{nn_1(x_i)}, \ldots, y_{nn_K(x_i)} \right\}$

$nn(\vec{x}) = \underset{i \in [n]}{\text{argmin}} \sum_{j=1}^{d} (x_j - x_j^{(i)})^2 = \underset{i \in [n]}{\text{argmin}} \| \vec{x} - \vec{x}^{(i)} \|_2^2$

Decision Trees

$I(A,Y) = H(Y) - H(A|Y)$

- Symmetric, ≥ 0

$H(Y) = -\sum_k P(A = a_k) \log P(A = a_k)$ } prob of pred e[pred]

$H(Y|A) = \sum_k P(A = a_k) H(Y | A = a_k)$ } prob of features given pred e[pred]

Choose highest IG

```
function BuildTree(D, A, Y)                    // D: training set, S: input attributes, Y: class attribute
    if D is empty
        return leaf node with failure
    else if all samples in D have same label label for Y
        return leaf node with label
    else if A is empty or all D have same feature values
        return leaf node with majority vote of values of Y in D
    else
        A ← "best" decision attribute among S using D
        tree ← new tree with root node assigned attribute A
        for each value a_k of A
            D_k ← examples from D with value a_k for attribute A
            subtree ← BuildTree(D_k, S − A, Y)
            add subtree as child of tree with branch labeled A = a_k
        return tree
```

Representer Theorem

$\hat{\theta} = \sum_i^n \beta_i \phi(\vec{x}^i)$

$\beta_i = -\frac{1}{2\lambda n} L'\left(\theta^T \phi(x^i), y^i \right)$

Kernel function

$K(\vec{x}^i, \vec{x}^j) = \phi(\vec{x}^i)^T \phi(\vec{x}^j)$

$$\tilde{J}(\beta) = \frac{1}{n} \sum_{i=1}^{n} L\left(\sum_{j=1}^{n} \beta_j k(x^{(i)}, x^{(j)}), y^{(i)} \right) + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \beta_i \beta_j k(x^{(i)}, x^{(j)})$$

- Standard objective
  $J(\theta) = \frac{1}{2n} \sum_{i=1}^{n} \left(y^{(i)} - \theta^T \phi(x^{(i)}) \right)^2 + \lambda \|\theta\|_2^2$
- Gradient Update
  $\theta \leftarrow \theta - \alpha \left(\frac{1}{n} \sum_{i=1}^{n} (\theta^T \phi(x^{(i)}) - y^{(i)}) \phi(x^{(i)}) + \lambda \theta \right)$
- Kernalized objective
  $\tilde{J}(\beta) = \frac{1}{2n} \sum_{i=1}^{n} \left(y^{(i)} - \sum_{j=1}^{n} \beta_j K_{ij} \right)^2 + \lambda \beta^T K \beta$
- Kernalized Gradient Update
  $\beta \leftarrow \beta - \alpha \left(\frac{1}{n} \sum_{i=1}^{n} \left(\sum_{j=1}^{n} \beta_j K_{ij} - y^{(i)} \right) K^{(i)} + \lambda \sum_{i=1}^{n} \beta_i K^{(i)} \right)$
  where $K^{(i)} = K_{i,:}^T$ (transpose of i-th row of K)

Kernalized Hypothesis

$h(\vec{x}) = \sum_i^n \beta_i K(\vec{x}^i, \vec{x})$

$\|\phi(\vec{u})\|_2^2 = \phi(\vec{u})^T \phi(\vec{u}) = K(\vec{u}, \vec{u})$ ↓

$\|\phi(\vec{u}) - \phi(\vec{v})\|_2^2 = \|\phi(\vec{u})\|_2^2 + \|\phi(\vec{v})\|_2^2 - 2\phi(\vec{u})^T \phi(\vec{v})$

$\|\phi(\vec{u}) + \phi(\vec{v})\|_2^2 = \|\phi(\vec{u})\|_2^2 + \|\phi(\vec{v})\|_2^2 + 2\phi(\vec{u})^T \phi(\vec{v})$

$= K(u,u) + K(v,v) \pm 2K(u,v)$

- **Mercer's Theorem**: A kernel function k is valid **if and only if** over for *any* set of $m > 0$ points, the corresponding kernel matrix K is **symmetric** and **positive semi-definite (PSD)**
- *Note:* The m points are not necessarily the training set

- **Symmetric:** $K_{ij} = K_{ji}$ for all i, j
  - Equivalently, $K = K^T$
- **Positive semi-definite:** For all $z \in \mathbb{R}^m$, $z^T K z \ge 0$
  - Equivalently, all eigenvalues of K are real and positive

- **Composition Rules:** If $k_1(u, v)$ and $k_2(u, v)$ are valid kernels, then the following are also valid kernels:
  - $k_1(u, v) + k_2(u, v)$ is a valid kernel
  - $k_1(u, v) k_2(u, v)$ is a valid kernel
  - $c k_1(u, v)$ for any positive real $c > 0$
  - $g(u) k_1(u, v) g(v)$ for any real-valued function g
  - $\exp k_1(u, v)$

- Hypothesis for linear models
  $h_\theta(x) = g(\theta^T x)$
- ℓ_2 regularized loss for linear models:
  $J(\theta) = \frac{1}{n} \sum_{i=1}^{n} L(\theta^T x^{(i)}, y^{(i)}) + \lambda \|\theta_{1:d}\|_2^2$

where

- **Linear Regression:** $g(z) = z$ and $L(\theta^T x^{(i)}, y^{(i)}) = \frac{1}{2} (y^{(i)} - \theta^T x^{(i)})^2$
- **Perceptron:** $g(z) = \text{sign}(z)$ and $L(\theta^T x^{(i)}, y^{(i)}) = \max(0, -y^{(i)} \theta^T x^{(i)})$
- **Logistic Regression:** $g(z) = \text{sigmoid}(z)$ and $L(\theta^T x^{(i)}, y^{(i)}) = -y^{(i)} \log\left(\text{sigmoid}(\theta^T x^{(i)}) \right) - (1 - y^{(i)}) \log\left(1 - \text{sigmoid}(\theta^T x^{(i)}) \right)$

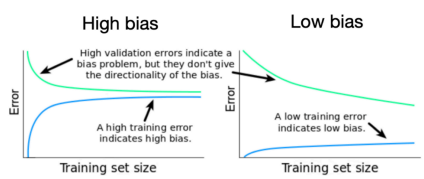
- **Step 1** (Pre-processing): Pre-compute Kernel Matrix $K \in \mathbb{R}^{n \times n}$ for n training points
- **Step 2** (Kernalized Training): Update β by gradient descent over reparameterized risk

$\beta^T K \beta$

$$\tilde{J}(\beta) = \frac{1}{n} \sum_{i=1}^{n} L\left(\sum_{j=1}^{n} \beta_j K_{ij}, y^{(i)} \right) + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \beta_i \beta_j K_{ij}$$

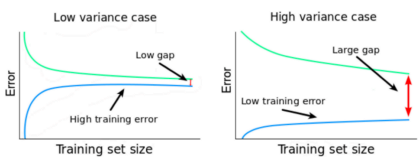
- **Step 3** (Kernalized Evaluation): Make predictions on a test point x by computing its **weighted** similarity with all training points :

$$h_\theta(x) = g(\theta^T \phi(x)) = g\left(\sum_{i=1}^{n} \beta_i k(x^{(i)}, x) \right)$$



Assumption: Irreducible error is not very high

Learning Curves: Assessing Variance



Derivatives of Logarithmic Functions

$$\frac{d}{dx} \ln(x) = \frac{1}{x} \qquad \frac{d}{dx} |x| = \frac{x}{|x|} \frac{d}{dx} x$$

$$\frac{d}{dx} \ln\left[f(x) \right] = \frac{1}{f(x)} f'(x)$$

$$\frac{d}{dx} \log_a(x) = \frac{1}{x \ln a}$$

$$\frac{d}{dx} \log_a\left[f(x) \right] = \frac{1}{f(x) \ln a} f'(x)$$

RBF Kernel:

$$k(\vec{x}^{(i)}, \vec{x}^{(j)}) = \exp\left[-\frac{\| \vec{x}^{(i)} - \vec{x}^{(j)} \|_2^2}{2\sigma^2} \right]$$

bandwidth hyper determines L2 significance

Regression

Given training instances, generate a hypothesis function $h(x) \approx y$.

1. Hypothesis class $\mathcal{H}$
2. Loss function $J(\theta)$
3. Optimize loss

Linear Regression

1. Parameterized linear functions:
   $h_\theta(x) = \theta_0 + \sum_{i=1}^{d} \theta_i x_i$
2. Least squares: $J(\theta) = \frac{1}{2n} \sum_{i=1}^{n} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$
   (Vectorized: $J(\theta) = \frac{1}{2n}(X\theta - y)^t(X\theta - y)$)
3. Solve $\min_\theta J(\theta)$:
   (a) Gradient Descent: $\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{k=0}^{d} \theta_k x_k^{(i)} - y^{(i)} \right) x_j^{(i)}$$

   (b) Stochastic Gradient Descent:
   $J(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell\left(x^{(i),y^{(i),\theta}}\right)$
   $\theta = \theta - \alpha \nabla_\theta \ell\left(x^{(i)}, y^{(i)}, \theta\right)$
   (c) Mini-batch Gradient Descent
   (d) Normal Equations: Closed form solution:

$$\theta = (X^t X)^{-1} X^t y$$

Generalization

Generalize the hypothesis class to: $h_\theta(x) = \theta^t \phi(x)$, where ϕ is a k-dimensional basis, eg.
$\phi(x) = \begin{bmatrix} 1 & x & x^2 & \dots \end{bmatrix}$ (polynomial regression).
New normal equation: $\hat{\theta} = (\phi(X)^t \phi(X))^{-1} \phi(X)^t y$

Regularization

New loss function:
$J(\theta) = \frac{1}{2n} \sum_{i=1}^{n} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2} \|\theta\|_2^2$

Representor Theorem

If $\lambda > 0$, then the optimal solution $\hat{\theta}$ is a linear combination of the training examples:

$$\hat{\theta} = \sum_{i=1}^{n} \alpha_i \phi\left(x^{(i)}\right)$$

Kernel Trick

1. Kernel function: $K(x, z) = \phi(x)^t \phi(z)$
2. Kernelized hypothesis: $h_\theta(x) = \sum_{i=1}^{n} \alpha_i K(x^{(i)}, x)$
3. Kernelized loss:
   $J(\alpha) = \frac{1}{n} \sum_{i=1}^{n} L\left(\sum_{j=1}^{n} \alpha_j K, y^{(i)}\right) + \frac{\lambda}{2} \alpha^t K \alpha$

A kernel is valid if it can be induced via a feature map ϕ.

Mercer's Theorem

A function $K(x, z)$ is a valid kernel iff it is symmetric and positive semi-definite.

Kernel Composition Rules

1. $K(x, z) = aK_1(x, z) + bK_2(x, z)$
2. $K(x, z) = K_1(x, z) K_2(x, z)$
3. $K(x, z) = cK_1(x, z)$
4. $K(x, z) = g(x) K_1(x, z) g(z)$, for g real-valued.
5. $K(x, z) = \exp K_1(x, z)$

Classification

Given labelled training examples, generate a hypothesis function $h(x) \approx y$.

Perceptron

1. $h_\theta(x) = \text{sign}(\theta^t x)$
2. 0/1 loss: $J_{0/1}(\theta) \frac{1}{n} \sum_{i=1}^{n} \ell_{0/1}(x^{(i)}, y^{(i)}, \theta)$ where

$$\ell_{0/1}(x^{(i)}, y^{(i)}, \theta) = \begin{cases} 0 & h_\theta(x^{(i)}) = y^{(i)} \\ 1 & \text{otherwise} \end{cases}$$

   Surrogate loss:

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} \max(0, -y^{(i)} \theta^t x^{(i)})$$

3. $\theta = \theta + \alpha y^{(i)} x^{(i)}$ if $y^{(i)} \theta^t x^{(i)} \leq 0$.

Convergence is guaranteed if the data is linearly separable.

Logistic Regression

1. $h_\theta(x) = \frac{1}{1 + e^{-\theta^t x}}$
2. Logistic loss: $J(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell_{log}(x^{(i)}, y^{(i)}, \theta)$
   where

$$\ell_{log}(x^{(i)}, y^{(i)}, \theta) = \begin{cases} -\log(h_\theta(x^{(i)})) & y^{(i)} = 1 \\ -\log(1 - h_\theta(x^{(i)})) & y^{(i)} = 0 \end{cases}$$

3. $\theta = \theta - \alpha \sum_{i=1}^{n} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)$

Multi-Class Logistic Regression

1. $h_\theta(x) = \frac{\exp(\theta^t x)}{\sum_{j=1}^{k} \exp(\theta_j^t x)}$
2. Multiclass logistic loss:
   $J(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell_{log}(x^{(i)}, y^{(i)}, \theta)$ where

$$\ell_{log}(x^{(i)}, y^{(i)}, \theta) = -\log\left(\frac{e^{\theta_{y^{(i)}}^t x^{(i)}}}{\sum_{j=1}^{k} e^{\theta_j^t x^{(i)}}}\right)$$

3. $\theta_j = \theta_j - \alpha \sum_{i=1}^{n} \left(1\{y^{(i)} = j\} - h_\theta\left(x^{(i)}\right)\right) x^{(i)}$

k-Nearest Neighbors

Nearest neighbor: $nn(x) = \text{argmin}_{x^{(i)} \in S} d(x, x^{(i)})$, where d is a distance metric.
k-nearest neighbors is the set of k nearest neighbors of x.
Then we take the majority vote of the labels of the k-nearest neighbors.
$h(x) = \text{majority } knn(x)$
Takes $O(nd)$ time. Memory intensive. No parameters.

Decision Trees

Given a set of labelled training examples, we want to learn a tree. Useful for interpretability.
Use Occam's razor: prefer simpler trees.

Iterative Dichotomiser (ID3)

1. Choose the best attribute to split on.
2. Recurse on the subsets.

How to choose the best attribute? Use maximum information gain.
Entropy: $H(S) = -\sum_{y \in Y} p(y) \log p(y)$
Information gain: $I(A, Y) = H(Y) - H(Y|A)$

Pre-Pruning

Common options: minimum leaf size, maximum depth, maximum number of nodes.

Post-Pruning

Reduced error pruning: remove a node if it doesn't decrease the error on the validation set.

Other

Bias-Variance Tradeoff

Bias: $\mathbb{E}[\hat{f}(x)] - f(x)$
Variance: $\mathbb{E}[(\hat{f}(x) - \mathbb{E}[\hat{f}(x)])^2]$

SVMs

Optimizes ntd+1 vars

Margin:

$$\gamma^{(i)} = y^{(i)} \frac{\vec{\theta}^T \vec{x}^{(i)}}{\|\vec{\theta}_{1:d}\|_2}$$

$$\gamma = \min_{i \in [n]} \gamma^{(i)}$$

Hard Margin:

$$\min_{\theta} \frac{1}{2}\|\vec{\theta}\|_2^2 \quad \text{given}$$

$$y^{(i)} \vec{\theta}^T \vec{x}^{(i)} \geq 1 \quad \forall i$$

Soft Margin:

$$\min_{\theta} \frac{1}{2}\|\vec{\theta}\|_2^2 + C \sum_{i=1}^{n} \epsilon_i \quad \text{given}$$

$$y^{(i)} \vec{\theta}^T \vec{x}^{(i)} \geq 1 - \epsilon_i \quad \forall i$$

$$\epsilon_i \geq 0 \quad \forall i \in [1, n]$$

Hinge Loss:

$$h_\theta(x) = g(\vec{\theta}^T \vec{x}) = \text{Sign}(\vec{\theta}^T \vec{x})$$

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} L(\vec{\theta}^T \vec{x}^{(i)}, y^{(i)}) + \lambda \|\vec{\theta}_{1:d}\|_2^2$$

$$L = \max(0, 1 - y^{(i)} \vec{\theta}^T \vec{x}^{(i)})$$

- Recall Regularized Linear models

$$h_\theta(x) = g(\theta^T x)$$

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} L(\theta^T x^{(i)}, y^{(i)}) + \lambda \|\theta_{1:d}\|_2^2$$

where
- **Linear Regression:** $g(z) = z$ and $L(\theta^T x^{(i)}, y^{(i)}) = (y^{(i)} - \theta^T x^{(i)})^2$
- **Perceptron:** $g(z) = \text{sign}(z)$ and $L(\theta^T x^{(i)}, y^{(i)}) = \max(0, -y^{(i)} \theta^T x^{(i)})$
- **Logistic Regression:** $g(z) = \text{sigmoid}(z)$ and $L(\theta^T x^{(i)}, y^{(i)}) = -y^{(i)} \log\left(\text{sigmoid}(\theta^T x^{(i)})\right) - (1 - y^{(i)}) \log\left(1 - \text{sigmoid}(\theta^T x^{(i)})\right)$

- For **SVMs**, $g(z) = \text{sign}(z)$. Can we find an $L(\cdot)$ to fit SVMs in this template?

Ensembles

Bootstrap: Create random train sets of n from total n w/ repetition

Bagging: Create many bootstrap replicas and train many estimate classifier on out-of-bootstrap data average all classifiers for predictions

Random Forests: bagging w/ training instances & features; make each node a feature bootstrap; don't prune and estimate on test; average output of all subtrees for preds

AdaBoost:

1) Init uniform weight vector $\vec{w_i}$ for n instances s.t. $|\vec{w_i}| = n$

for $t = 1, \ldots, T$:

2) Train model $h_{t=1}$ on X, y w/ $\vec{w_i}$

3) Measure error ϵ as misclassification

4) Choose $\beta_t = \frac{1}{2} \ln\left[\frac{1 - \epsilon_t}{\epsilon_t}\right]$

5) Update weights: $w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\vec{x_i})\right) \quad \forall i$

6) Normalize w_{t+1} to be dist. $[0, 1]$ s.t. $\|w_{t+1}\| = 1$

$$w_{t+1,i} = \frac{w_{t+1,i}}{\sum_j^n w_{t+1,j}} \quad \forall i$$

end for

7) $H(\vec{x}) = \text{sign}\left(\sum_{t=1}^{T} \beta_t h_t(\vec{x})\right)$

weights of correct: $\cdot = \exp(-\beta_t) \leq 1$

weights of incorrect: $\cdot = \exp(\beta_t) \geq 1$

- if $\epsilon \leq 0.5 \Rightarrow \beta_t \geq 0$ (better than random)

Essentially, weigh the cost of misclassification differently for each instance

$$J_{\text{reg}}(\theta) = -\sum_{i=1}^{n} w_{t,i} [y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))] + \frac{\lambda}{2} \|\theta_{1:d}\|_2^2$$

- Works best on weak learners h_t → incorrect are weight inc. logarithmically

K-Means Clustering

Inputs: Training set of n datapoints $\{x^{(i)}\}$, number of clusters k
- **Step 1:** Initialize cluster centroids $\mu_1, \mu_2, \cdots, \mu_k$
- **Step 2:**

Loop until convergence: {

For every point i, assign it to a cluster

$$c_i := \arg\min_j \|x^{(i)} - \mu_j\|_2^2$$

For every cluster j, re-estimate its centroid

$$\mu_j := \frac{\sum_{i=1}^{n} 1\{c_i = j\} x^{(i)}}{\sum_{i=1}^{n} 1\{c_i = j\}}$$

}

Output: clusters for training points $c_{1:n}$, cluster centers for test points $\mu_{1:k}$

- k-means optimizes the following objective function:

$$\arg\min_S \sum_{i=1}^{k} \sum_{x \in S_i} \|x - \mu_i\|_2^2$$

where $S = \{S_1, \cdots, S_k\}$ is a partitioning over $X = \{x^{(1)}, \cdots, x^{(n)}\}$ such that $X = \cup_{i=1}^{k} S_i$ and $\mu_i = \text{mean}(S_i)$

Inertia: Sum of squared distances to cluster centers

- we want low inertia & low k
  - but, high k → low inertia → choose k when Δinertia >

PCA

- Consider an encoding-decoding mechanism for dimensionality reduction:
  - Encoding (**Principal Component Analysis or PCA**): $y^{(i)} = W x^{(i)}$ where $W \in \mathbb{R}^{m \times d}$ and $y^{(i)} \in \mathbb{R}^m$
  - Decoding (**Reverse PCA**): $\tilde{x}^{(i)} = U y^{(i)}$ where $U \in \mathbb{R}^{d \times m}$ and $\tilde{x}^{(i)} \in \mathbb{R}^d$
- To learn PCA encoding-decoding, we minimize reconstruction loss:

$$\min_{U,W} \sum_{i=1}^{n} \|x^{(i)} - \tilde{x}^{(i)}\|_2^2 = \sum_{i=1}^{n} \|x^{(i)} - UW x^{(i)}\|_2^2$$

$$= \|X - XW^T U^T\|_F^2 \text{ (via vectorization)}$$

(where $\|M\|_F^2 = \sum_i \sum_j M_{i,j}^2$ denotes the squared Frobenius norm)

$$\min_{U,W} \sum_{i}^{n} \|\vec{x}^{(i)} - \hat{x}^{(i)}\|_2^2 = \|X - X W^T U^T\|_F^2$$

- **Lemma 1 (without proof):** For the PCA objective, the optimal solution for U is such that the columns of U are orthogonal i.e. $U^T U = I_m$ and $W = U^T$
- **Implication:** Can rewrite PCA objective as:

$$\min_U \sum_{i=1}^{n} \|x^{(i)} - UU^T x^{(i)}\|_2^2$$

s.t. $UU^T = I_d$

Constrained optimization problem

- **Lemma 2 (without proof):** The solution to the PCA learning objective is given by the first m eigenvectors of the empirical covariance matrix
  - Let $A = \frac{1}{(n-1)} \sum_{i=1}^{n} (x^{(i)} - \mu)(x^{(i)} - \mu)^T$ be the empirical covariance matrix. Here, $\mu = \frac{1}{n} \sum_{i=1}^{n} x^{(i)}$ is the mean vector.
  - Let $v_i \in \mathbb{R}^d$ denote the eigenvector for the i-th **largest** eigenvalue of A. Let $V = [v_1, v_2, \ldots, v_d]$ be the matrix of eigenvectors.
  - The solution to the PCA learning objective is given by the first m columns of V i.e., $U = [v_1, v_2, \ldots, v_m]$
- **Implication:** Can solve for PCA on X by performing an eigendecomposition of A
  - $v_1, v_2, \ldots, v_m$ are also known as the **principal components directions**
  - $v_1^T x, v_2^T x, \ldots, v_m^T x$ are also known as the **principal components scores**

- **Step 1:** Compute covariance matrix $A \in \mathbb{R}^{d \times d}$

$$A = \frac{1}{n-1}(X - \mu)^T (X - \mu) = \frac{1}{(n-1)} \sum_{i=1}^{n} (x^{(i)} - \mu)(x^{(i)} - \mu)^T$$

- **Step 2:** Compute the eigenvectors of A
  - V, Λ = numpy.linalg.eig(A)
  - $\{v_i, \lambda_i\}_{i=1,\ldots,d}$ are the eigenvectors/eigenvalues of A sorted in descending order of eigenvalues
- **Step 3:** Return the top m eigenvectors $U = [v_1, v_2, \ldots, v_m]$ as the principal component directions
- For any point x, its reduced representation is $y = Wx = U^T x$

To recover: $\vec{y} = U^T \vec{x} \Rightarrow U^T \vec{y} = UU^T \vec{x}$

↳ $UU^T = I_d \quad \therefore \vec{x} = U^T \vec{y}$

Explained Variance measures relative importance of each eigenvector for each principal component i:

$$EV_i = \frac{\lambda_i}{\lambda_1 + \ldots + \lambda_d} \Rightarrow \text{ignores } d\text{-}m \text{ components w/ least significance i.e. small } \lambda_{m:d}$$

Principal Component Direction:
- Order of m-largest eigenvecs show largest direction of variance
- each is orthogonal to another: $UU^T = I$
- eigenvectors are unit vectors s.t. $\gamma_i = \pm \|\gamma_i \vec{v_i}\|$

Neural Networks

Inputs: $x \in \mathbb{R}^d, y \in \mathbb{R}^C$

Output: $f_\theta(x) = \text{softmax}(s)$

- **Logistic Regression**

$$s = Wx + b \leftarrow \text{scores}$$

  - Parameters: $W \in \mathbb{R}^{C \times d}, b \in \mathbb{R}^C$
- **1-hidden layer neural net**

$$s = W_2 \max(0, W_1 x + b_1) + b_2$$

  - Parameters: $W_1 \in \mathbb{R}^{H \times d}, W_2 \in \mathbb{R}^{C \times H}, b_1 \in \mathbb{R}^H, b_2 \in \mathbb{R}^C$
- hidden state is the activated scores s.t. $\vec{h} = \max(0, W_1 \vec{x} + \vec{b_1}) = \text{ReLU}(\text{scores})$

Scores of next $W_2 = \vec{s} = W_2 \vec{h} + \vec{b_2}$

Dense Net ≡ Multi-Layer Perceptron

Activation Functions:

Sigmoid: $\sigma(x) = \frac{1}{1 + e^{-x}}$

tanh: $\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$

ReLU: $\max(0, x)$

Leaky ReLU: $\max(0.2x, x)$

Softplus: $\log(1 + \exp(x))$

ELU: $f(x) = \begin{cases} x & x > 0 \\ \alpha(\exp(x) - 1) & x \leq 0 \end{cases}$

- 2-HL NN is a universal approximator like kNN

Forward Pass:

$$h = \max(0, W_1 x + b_1)$$
$$s = W_2 h + b_2$$
$$f_\theta(x) = \text{softmax}(s)$$

$$J(\theta) = -\sum_{i=1}^{n} \sum_{c=1}^{C} y_c^{(i)} \log f_\theta(x^{(i)})$$

$$\text{softmax}(x_j) = \frac{e^{x_j}}{\sum_c^K e^{x_c}}$$

Cross-Entropy Loss:

$$L = -\sum_c^K y_c^{(i)} \log[\text{softmax}(x_c)]$$

$$\frac{\partial}{\partial x_j^{(i)}} L(\vec{x}^{(i)}, \vec{y}^{(i)}) = S_j^{(i)} - y_j^{(i)} \Rightarrow \frac{1}{N}(S - Y) \leftarrow \text{are not vecs}$$

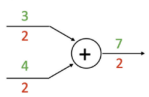
$\times \equiv f$ in this eq.

Backprop

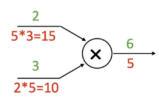
$$\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial f} \frac{\partial f}{\partial \theta} \quad \text{s.t.} \quad \frac{\partial L}{\partial f} = \nabla \quad \frac{\partial f}{\partial \theta} \text{ given by back pass}$$

Common Gate reduction:

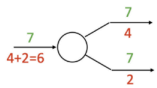
add gate: gradient distributor



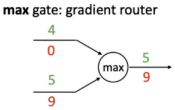
mul gate: "swap multiplier"



copy gate: gradient adder



max gate: gradient router



$$\nabla_W L(f, y) = \nabla_f L(f, y)^T \vec{H} + \nabla_W \hat{R} = \frac{1}{N}(S - Y)^T H + \lambda W$$

$\vec{H} = \text{ReLU}(\vec{X}W + b)$ ← activated scores for that layer

$$\nabla_b L(f, y) = \frac{1}{N}(S - Y)^T \vec{1}_{0:N}$$

Vector Derivatives

$x \in \mathbb{R}, y \in \mathbb{R}$	$x \in \mathbb{R}^N, y \in \mathbb{R}$	$x \in \mathbb{R}^N, y \in \mathbb{R}^M$
Regular derivative:	Derivative is **Gradient**:	Derivative is **Jacobian**:
$\frac{\partial y}{\partial x} \in \mathbb{R}$	$\frac{\partial y}{\partial x} \in \mathbb{R}^N$, $\left(\frac{\partial y}{\partial x}\right)_i = \frac{\partial y}{\partial x_i}$	$\frac{\partial y}{\partial x} \in \mathbb{R}^{N \times M}$ $\left(\frac{\partial y}{\partial x}\right)_{i,j} = \frac{\partial y_j}{\partial x_i}$
If x changes by a small amount, how much		

Matrix Multiply $y = xw$

$$y_{i,j} = \sum_k x_{i,k} w_{k,j}$$

Matrix Mults ⇒ implicit Jacobian

$$\frac{\partial L}{\partial x_0} = \left(\frac{\partial x_1}{\partial x_0}\right)\left(\frac{\partial x_2}{\partial x_1}\right)\left(\frac{\partial x_3}{\partial x_2}\right)\left(\frac{\partial L}{\partial x_3}\right)$$

$[D_0 \times D_1]$ $[D_1 \times D_2]$ $[D_2 \times D_3]$ $[D_3]$

dL/dx = (dL/dy) w^T

[N x D] [N x M] [M x D]

dL/dw = x^T (dL/dy)

[D x M] [D x N] [N x M]

Convolution NNs

Input: $N \times C_{in} \times H \times W$

Filters: $C_{out} \times C_{in} \times K_h \times K_w$

Bias: $1 \times C_{out}$

Out: $N \times C_{out} \times H' \times W'$ s.t. $H' = H - K_h + 1$, $W' = W - K_w + 1$

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

n_{in}: number of input features

n_{out}: number of output features

k: convolution kernel size

p: convolution padding size

s: convolution stride size

- applies weight matrix called filter s.t. $S = \vec{w}\vec{x} + b$
- slide filter across image → activation maps → stack

Dropout: drop nodes w/ some probability in training ↳ Stops backprop on those nodes → still used for test

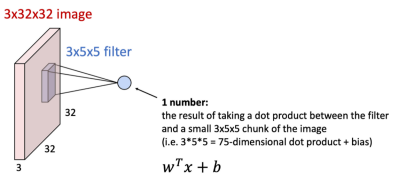
Momentum: moving averages to smooth function S_t

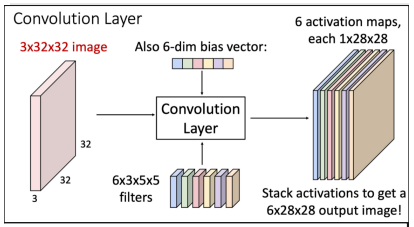
$V_0 = 0$ and $\beta \in [0, 1]$

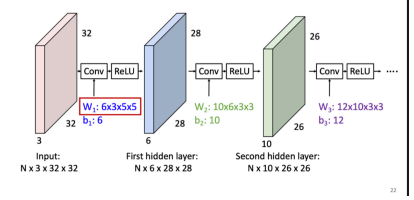
$$V_t = \beta V_{t-1} + (1 - \beta) S_t \Rightarrow S_t = \nabla_\theta J(\vec{\theta})$$

$$\vec{\theta}_{t+1} \leftarrow \vec{\theta}_t - \alpha \vec{V}_t \quad : \text{smooths grad. descent}$$

Convolutional NNs Visuals





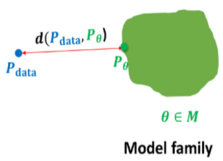


Generative Models

Statistical

- Learn a prob. dist. $p_\theta(x) \sim p_{data}(x)$
- $p_\theta(x)$ is high when x is dog (e.g.)

$x_i \sim P_{data}$
$i = 1, 2, \ldots, n$



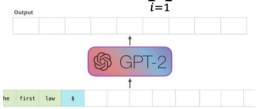
Model family

Autoregressive Models

- **Key idea:** Decompose the joint distribution as a product of 1d probability conditionals using chain rule of probability

$$p_\theta(x) = \prod_{i=1}^{d} p_\theta(x_i | x_1, x_2, \cdots, x_{i-1}) = \prod_{i=1}^{d} p_\theta(x_i | x_{<i})$$

$$p(x_{1:d}) = \prod_{i=1}^{d} p(x_i | x_{<i}) \qquad p(x_{1:d}) = \prod_{i=1}^{d} p(x_i | x_{<i})$$



Representing Prob. Dists.

- **Distribution Type:** Depends on whether x_i is discrete or continuous
  - Continuous x_i: p_θ can be a Gaussian distribution
  - Discrete x_i: p_θ can be a categorical (multinomial) distribution
- The parameters $\boldsymbol{\theta}$ signify a mapping to the sufficient statistics of the chosen distribution for p_θ
  - Gaussian: $\boldsymbol{\theta}$ are parameters of any function that map to mean, variance of p_θ
  - Categorical: $\boldsymbol{\theta}$ are parameters of any function that map to probabilities of each category
- Parameters $\boldsymbol{\theta}$ can be shared or separate for each $p_\theta(x_i | x_{<i})$
  - Shared: one function with d*k outputs where k is number of sufficient stats for a distribution type (e.g., k=2 for Gaussian, C for categorical)
  - Separate: d functions, each with k outputs

- RNNs & Transformers try to parametrize conditionals
- Inference requires sequential sampling

$$\widehat{x_1} \sim p_\theta(x_1)$$
$$\widehat{x_2} \sim p_\theta(x_2 | \widehat{x_1})$$
$$\widehat{x_3} \sim p_\theta(x_3 | \widehat{x_1}, \widehat{x_2})$$
$$\ldots$$
$$\widehat{x_d} \sim p_\theta(x_d | \widehat{x_{<d}})$$

Eigen Decomposition

$A\vec{v} = \lambda\vec{v}$ → all $c\vec{v}$ are eigenvecs

$(A - \lambda I)\vec{v} = \vec{0}$ → eigenvecs

$|A - \lambda I| = 0$ → eigenvecs

Matrix Properties

U is orth. $\Rightarrow UU^T = U^T U = I$

$(AB)^T = B^T A^T$

$(ABC)^T = C^T B^T A^T$

$AB = BA = I \Rightarrow$ invertible: $B = A^{-1}$

Max partial

$$\frac{\partial \max(0, x)}{\partial x} = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$$

MAE partial

$$\frac{\partial}{\partial \theta_j} |h_\theta(x) - y| = \underbrace{\frac{h(x) - y}{|h(x) - y|}}_{\text{Sign}(h(x)-y)} \underbrace{\frac{\partial}{\partial \theta_j}(h(x) - y)}_{x_j} = x_j \cdot \text{Sign}(h(x) - y)$$

Example Decision Tree IG

#	Input Attribute: *Electric*	*Price*	*Condition*	*Color*	Label: *Buy*
1	Yes	L	N	R	Yes
2	No	M	N	G	No
3	No	M	U	G	No
4	Yes	H	U	R	Yes
5	Yes	H	N	G	Yes
6	No	L	N	G	No
7	No	H	N	R	Yes
8	Yes	M	U	G	Yes
9	Yes	L	U	G	No
10	No	L	N	G	No

ations: L: Low, M: Medium, H: High, N: New, U: Used, R: Red, G: Green.

$$H(Buy) = \underbrace{-\frac{5}{10}\log_2\frac{5}{10}}_{\text{No}} \underbrace{-\frac{5}{10}\log_2\frac{5}{10}}_{\text{Yes}} = 1$$

$$H(Buy | Price = \text{Low}) = -\frac{1}{4}\log_2\frac{1}{4} - \frac{3}{4}\log_2\frac{3}{4} = 0.8$$

$$H(Buy | Price = \text{Medium}) = -\frac{1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3} = 1.6 - \frac{2}{3} = 0.93$$

$$H(Buy | Price = \text{High}) = 0$$

$$H(Buy | Price) = \frac{4}{10} \times 0.8 + \frac{3}{10} \times 0.93 + \frac{3}{10} \times 0 = 0.6$$

$$I(Buy, Price) = 0.4$$

Midterm Kernel Proofs

(a) **[2 points]** Let $k(\mathbf{u}, \mathbf{v})$ be a valid kernel function for $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$. Prove or disprove that $ck(\mathbf{u}, \mathbf{v})$ is a valid kernel function for $c > 0$.

Answer: Yes, this is a valid kernel (1pt). The proof can be done in 2 ways:
(1) We can prove it by first principles. Let $\phi(\mathbf{u}, \mathbf{v})$ be the feature map for $k(\mathbf{u}, \mathbf{v})$. Then $\sqrt{c}\phi(\mathbf{u}, \mathbf{v})$ is the feature map for $ck(\mathbf{u}, \mathbf{v})$ (1 pt).
(2) Alternatively, we can prove it by Mercer's Theorem. Symmetricity (0.5pts): Since $k(\mathbf{u}, \mathbf{v})$ is a valid kernel map, $ck(\mathbf{u}, \mathbf{v}) = ck(\mathbf{v}, \mathbf{u})$. PSD (0.5pts): Since $k(\mathbf{u}, \mathbf{v})$ is a valid kernel, $\mathbf{z}^T K \mathbf{z} \geq 0$ for all $\mathbf{z}$. Hence, $c\mathbf{z}^T K \mathbf{z} \geq 0$ whenever $c > 0$.

(b) **[3 points]** Let $k(\mathbf{u}, \mathbf{v})$ be a valid kernel function for $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$ and $f : \mathbb{R}^d \to \mathbb{R}$ be any real-valued function. Prove that $f(\mathbf{u})k(\mathbf{u}, \mathbf{v})f(\mathbf{v})$ is a valid kernel function.

Answer: Yes, this is a valid kernel. Since $k(\mathbf{u}, \mathbf{v})$ is a valid kernel, it can be expressed as $k(\mathbf{u}, \mathbf{v}) = \phi(\mathbf{u})^T \phi(\mathbf{v})$. Now, consider the feature map $\phi'(\mathbf{u}) = f(\mathbf{u})\phi(\mathbf{u})$ and definition of a valid kernel to finish the proof of the claim.

(c) **[5 points]** Prove or disprove that $\exp(-\frac{1}{2}||\mathbf{u} - \mathbf{v}||_2^2)$ is a valid kernel function.

[Hints: (1) You can directly use the result from part b above. (2) Recall from lecture the following composition rule (which can be used without proof): for any valid kernel function $k(\mathbf{u}, \mathbf{v})$, $\exp(k(\mathbf{u}, \mathbf{v}))$ is a valid kernel function.]

Answer: Yes, this is a valid kernel. We write $\exp(-\frac{1}{2}||\mathbf{u} - \mathbf{v}||_2^2) = \exp(-\frac{1}{2}(\mathbf{u} - \mathbf{v})^T(\mathbf{u} - \mathbf{v})) = \exp(-\frac{1}{2}\mathbf{u}^T\mathbf{u} + \mathbf{u}^T\mathbf{v} - \frac{1}{2}\mathbf{v}^T\mathbf{v}) = \exp(-\frac{1}{2}\mathbf{u}^T\mathbf{u})\exp(\mathbf{u}^T\mathbf{v})\exp(-\frac{1}{2}\mathbf{v}^T\mathbf{v})$. We then note that $\exp(\mathbf{u}^T\mathbf{v})$ is a valid kernel since $\mathbf{u}^T\mathbf{v}$ is a valid kernel (take $\phi(\mathbf{u}) = \mathbf{u}$) and $\exp(k(\mathbf{u}, \mathbf{v}))$ is a valid kernel for any valid kernel $k(\mathbf{u}, \mathbf{v})$. We then apply the previous part with $f(\mathbf{u}) = \exp(-\frac{1}{2}\mathbf{u}^T\mathbf{u})$ and $k(\mathbf{u}, \mathbf{v}) = \exp(\mathbf{u}^T\mathbf{v})$. **[If a student gives a correct alternate solution, give full credit.]**

NN additional Grads

$$R = \frac{\lambda}{2}||W||_2^2 \quad \text{s.t.}$$

$$\hat{R} = \frac{\lambda}{2}||W_1 + \ldots + W_n||_2^2$$

$$\frac{\partial \hat{R}}{\partial W_1} = \lambda W_1$$

$$S_j = \text{Softmax}(x_j) = \frac{e^{x_j}}{\sum_c^K e^{x_c}}$$

$$\frac{\partial S_j}{\partial x_j} = S_c\left[\vec{1}\{c = j\} - S_j\right]$$

$$L(\vec{x}, \vec{y}) = -\sum_c^K y_c \log S_c$$

$$L(X, Y) = -\frac{1}{N}\sum_c^K Y_c \log S_c$$

$$\frac{\partial L}{\partial x_j} = S_j - Y_j$$