CS2109S
AY24/25 SEM 1

Artificial Intelligence

Uninformed Search

Type	Breath-First	Uniform-Cost	Depth-First
Frontier	Queue	Priority Queue (Path Cost)	Stack
Time	$O(b^d)$	$O(b^{C^*/\epsilon})$	$O(b^m)$
Space	$O(b^d)$	$O(b^{C^*/\epsilon})$	$O(bm)$
Complete	Yes if b is finite	Yes if $\epsilon > 0$ and C^* is finite	No
Optimal	Yes (step cost same)	Yes if $\epsilon > 0$	No

- C^* cost of optimal solution, ϵ minimum edge cost
- In uniform-cost search, $\epsilon = 0$ may cause zero cost cycle.

Type	Depth-Limited	Iterative Deepening
Frontier	Stack	Stack
Time	$O(b^l)$	$O(b^d)$
Space	$O(bl)$	$O(bd)$
Complete	No	Yes
Optimal	No	Yes (step cost same)

Informed Search

Type	Greedy Best First	A*
Frontier	Priority Queue	Priority Queue
$f(n)$	$h(n)$	$g(n) + h(n)$
Time	$O(b^m)$	$O(b^m)$
Space	$O(b^m)$	$O(b^m)$
Complete	No	Yes
Optimal	No	**Depends on the heuristic**

Heuristics

- **Admissible** $\rightarrow h(n) \leq h^*(n)$ for every node n where $h^*(n)$ is the true cost.
  - **Thorem:** If $h(n)$ is admissible, then A* using **tree search** is optimal.
- **Consistent** $\rightarrow h(n) \leq c(n, a, n') + h(n')$ and $h(G) = 0$
- **Dominance** $\rightarrow$ If $h_2(n) \geq h_1(n)$ for all n, then h_2 **dominates** h_1.
  - If h_2 is admissible, then it is closer to the true cost function (better for search)

Machine Learning

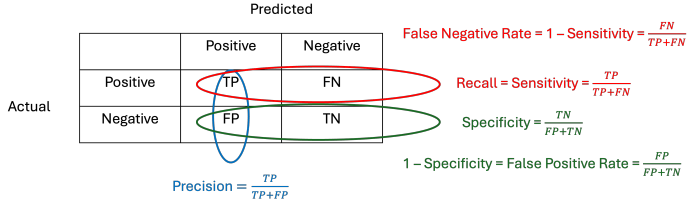
Performance Measures

Regression

- Absolute Error = $| \hat{y} - y |$
- Squared Error = $(\hat{y} - y)^2$
- Mean Absolute Error (MAE) = $\frac{1}{N} \sum_{i=1}^{N} | \hat{y} - y |$ for N examples
- Mean Squared Error (MSE) = $\frac{1}{N} \sum_{i=1}^{N} (\hat{y} - y)^2$ for N examples

Classification

- Accuracy = $\frac{1}{N} \sum_{N}^{i=1} 1_{\hat{y}_i = y_i}$



- Accuracy = $\frac{TP+TN}{TP+FP+TN+FN}$
- Precision = $\frac{TP}{TP+FP}$
- Recall = $\frac{TP}{TP+FN}$
- F1 Score = $\frac{2}{\frac{1}{Precision} + \frac{1}{Recall}}$

Decision Trees

- **Entropy** $\rightarrow$ measure of randomness in data
  - Let $v_1, v_2, \ldots v_n$ be n possible values (labels) we are predicting.
  - $I(P(v_1), P(v_2), \ldots, P(v_n)) = -\sum_{i=1}^{n} P(v_i) \log_2 P(v_i)$ where $P(v_i)$ is the proportion of examples with the output label v_i
- **Information Gain** $\rightarrow$ Reduction in entropy
  - A chosen attribute A divides the training set E into subsets $E_1, \ldots, E_v$ according to their values for A, where A has v distinct values.
    - $remainder(A) = \sum_{i=1}^{v} \frac{|E_i|}{|E|} \cdot I(E_i)$
    - $IG(A) = Entropy(A) - remainder(A)$

Usage:

- $\text{Entropy}(p, n) = I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$
  - where p is the number of positive labels and n is the number of negative labels
- This table is for two outcomes and for two branches.
- To read this table, p is the row headers (left vertical) and n is the column headers (top horizontal).

p\n	0	1	2	3	4	5	6	7	8	9	10
0	0										
1	0	1									
2	0	0.9183	1								
3	0	0.8113	0.971	1							
4	0	0.7219	0.9183	0.9852	1						
5	0	0.65	0.8631	0.9544	0.9911	1					
6	0	0.5917	0.8113	0.9183	0.971	0.994	1				
7	0	0.5436	0.7642	0.8813	0.9457	0.9799	0.9957	1			
8	0	0.5033	0.7219	0.8454	0.9183	0.9612	0.9852	0.9968	1		
9	0	0.469	0.684	0.8113	0.8905	0.9403	0.971	0.9887	0.9975	1	
10	0	0.4395	0.65	0.7793	0.8631	0.9183	0.9544	0.9774	0.9911	0.998	1

- $\log_2 \frac{x}{y} = \log_2 x - \log_2 y$
- $\log_2 1 = 0, \log_2 2 = 1, \log_2 3 = 1.5849, \log_2 4 = 2, \log_2 5 = 2.3219$
- $\log_2 6 = 2.5849, \log_2 7 = 2.8073, \log_2 8 = 3, \log_2 9 = 3.1699, \log_2 10 = 3.3219$

Linear Regression

- Notations
  - n = number of features
  - $x^{(i)}$ = input features of the i-th training example
  - $x_j^{(i)}$ = value of feature j in the i-th training example
- **Hypothesis**: $h_w(x) = w_0 x_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n = \sum_{j=0}^{n} w_j x_j$
- $= \begin{bmatrix} w_0 \\ w_1 \\ \cdots \\ w_n \end{bmatrix}^T \begin{bmatrix} x_0 \\ x_1 \\ \cdots \\ x_n \end{bmatrix} = w^T x$ where x_0 is the bias.
- **Loss Function** $\rightarrow J_{MSE}(w) = \frac{1}{2m} \sum_{m}^{i=1} (h_w(x^{(i)}) - y^{(i)})^2$
- Our goal is to find w or the weights w_0 to w_n that **fits the data well**.
- **Gradient of Loss Function:** $\frac{\delta J_{MSE}(w)}{\delta w_j} = \frac{1}{m} \sum_{i=1}^{m} (h_w(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$
- **Weight Updates**: $w_j \leftarrow w_j - \gamma \frac{\delta J_{MSE}(w_0, w_1, \ldots, w_n)}{\delta w_j}$
- $w_j \leftarrow w_j - \gamma \frac{1}{m} \sum_{i=1}^{m} (h_w(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$

Gradient Descent $\rightarrow$ An optimization algorithm for finding a local minimum of a differentiable function.

1. Start at some w
2. Pick a nearby w that reduce $J(w)$
   - $w_j \leftarrow w_j - \gamma \frac{\delta J(w_0, w_1, \ldots)}{\delta w_j}$ where γ is the learning rate.
3. Repeat until minimum is reached.

- To find the global minimum, the loss function has to be a **convex function**, which is a function with **one minimum which is the global minimum.**
- Theorem: The MSE loss function is convex for linear regression.
- When γ is just nice, as $J_{MSE}(w)$ gets closer to a minimum, the **gradient** becomes smaller and the **steps** become smaller.

Variants of Gradient Descent

- **Batch Gradient Descent** $\rightarrow$ Consider **all** training examples
- **Mini-batch Gradient Descent** $\rightarrow$ Consider a **subset** of training examples.
- **Stochastic Gradient Descent** $\rightarrow$ Select one random data point at a time.
  - Cheaper per iteration. More randomness, may escape local minima.

Dealing with Features of Different Scales

- **Mean Normalization**: $x_j = \frac{x_j - \mu_j}{\sigma_j}$
- **Min-Max scaling** and **robust scaling**.
- **Polynomial Regression**: $h_w(x) = w_0 + w_1 f_1 + w_2 f_2 + \cdots + w_n f_n$, where f_1 to f_n are transformed features. For example: $f_1 = x$, $f_2 = x^2$

Normal Equation

- $X = \begin{bmatrix} 1 & X_1^{(1)} & \cdots & X_n^{(1)} \\ 1 & X_1^{(2)} & & X_n^{(2)} \\ 1 & \vdots & & \vdots \\ 1 & X_1^{(m)} & & X_n^{(m)} \end{bmatrix}, w = \begin{bmatrix} w_0 \\ w_1 \\ \cdots \\ w_n \end{bmatrix}, y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \cdots \\ y^{(m)} \end{bmatrix}$
- $h_w(X) = X \cdot w$
- $w = (X^T X)^{-1} X^T Y$

Logistic Regression

Logistic Function (Sigmoid)

- $\sigma(z) = \frac{1}{1+e^{-z}}$ (Treat output as probability between 0 and 1)

Cross-entropy Loss

- **Cross-entropy for C classes:** $CE(y, \hat{y}) = \sum_{i=1}^{C} -y_i \log_e \hat{y}_i$
- **Binary cross-entropy:** $BCE(y, \hat{y}) = -y \log_e \hat{y} - (1-y) \log(1-\hat{y})$
- Binary Cross Entropy (BCE) is a **convex function** for logistic regression.

Logistic Regression with Gradient Descent

- **Hypothesis (for n features):**
  $h_w(x) = \sigma(w_0 x_0 + w_1 x_1 + \cdots + w_n x_n) = \sigma(\sum_{j=0}^{n} w_j x_j) = \sigma(w^T x)$
- **Loss Function:**
  - $J_{BCE}(w) = \frac{1}{m} \sum_{i=1}^{m} BCE(y^{(i)}, h_w(x^{(i)}))$
  - $\frac{\delta J_{BCE}(w)}{\delta w_j} = \frac{\delta}{\delta w_j} \frac{1}{m} \sum_{i=1}^{m} BCE(y^{(i)}, h_w(x^{(i)}))$
  - $\frac{\delta J_{BCE}(w)}{\delta w_j} = \frac{1}{m} \sum_{i=1}^{m} (h_w(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$
  - The gradient of BCE is the same as MSE.
- **Weight Update (for n features):** $w_j \leftarrow w_j - \gamma \frac{\delta J_{BCE}(w_0, w_1, \cdots, w_n)}{\delta w_j}$
  - $w_j \leftarrow w_j - \gamma \frac{1}{m} \sum_{i=1}^{m} (h_w(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$
- The weight update is also the same as linear regression.
- To deal with a **Non-Linear Decision Boundary** we apply the **logistic function** on the polynomial regression equation.
- $h_w(x) = \sigma(w_0 + w_1 f_1 + w_2 f_2 + \cdots + w_n f_n)$
- f_n are the transformed features, which can be x_1 to x_n as n-degree polynomials. (Eg: $f_1 = x_1, f_2 = x_2, f_3 = x_1^2, f_4 = x_2^2$)

Multi-class Classification

One vs All

- Fit one classifer per class, fit against all other classes
- Pick **highest probability**
- Linear time in the number of classes.

One vs One

- Fit one classifer per pair
- Pick **most wins**
- $\binom{n}{2}$ classifications for n classes

Receiver Operator Characteristic (ROC) Curve

- ROC curve is a plot of the **True Positive Rate** against the **False Positive Rate**
- The model is more accurate than random chance if its **ROC curve** is above the diagonal random line.

Area Under Curve of ROC

- $AUC > 0.5$ means the model is better than chance.
- $AUC \approx 1$ means the model is very accurate.

Linear Regression with Regularization

- Hypothesis: $h_w(x) = w^T x$
- Cost function: $J(w) = \frac{1}{2m}[\sum_{i=1}^{m}(h_w(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} w_j^2]$
  $= \frac{1}{2m}\sum_{i=1}^{m}(h_w(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2$
- Gradient Descent: $w_j \leftarrow w_j - \gamma \frac{1}{m}\sum_{i=1}^{m}(h_w(x^{(i)}) - y^{(i)})x_j^{(i)} - \gamma\frac{\lambda}{m}w_j$

Effect of Regularization

- When $\lambda = 0$, reduces to normal linear regression.
- When $\lambda > 0$ grows larger, the shrinkage effect of the weights gets bigger.

Logistic Regression with Regularization

- Hypothesis: $h_w(x) = \frac{1}{1+e^{-w^T x}}$
- Cost function: $J(w) =$
  $-\frac{1}{m}\sum_{i=1}^{m} y^{(i)}\log(h_w(x^{(i)})) + (1-y^{(i)})\log(1-h_w(x^{(i)})) + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2$
- Gradient Descent: $w_j \leftarrow w_j - \gamma \frac{1}{m}\sum_{i=1}^{m}(h_w(x^{(1)}) - y^{(i)})x_j^{(i)} - \gamma\frac{\lambda}{m}w_j$

Hard-margin Support Vector Machine

- **Goal:** Find a **decision boundary** that **maximises the margin** between two classes of data points.

Decision Rule

- Suppose we have two classes of data points ($+$ and $-$)
- Let the decision rule be $w \cdot x + b \geq 0$ then classify that point as a $+$.
- w represent the weights, x represents the features and b represents the offset.
- We can think of this decision rule as if we are looking at the upper margin line or how are we going to classify a $+$ data point.

Formulation

- Let $b = -c$
- $w \cdot x \geq c$ then $+ \implies w \cdot x + b \geq 0$ then $+$
- Assume $y^{(i)} = +1$ for $+$ samples and $y^{(i)} = -1$ for $-$ samples
- The constraints for the existing data:
  - $w \cdot x^+ + b \geq 1$
  - $w \cdot x^- + b \leq -1$
- $y^{(i)}(w \cdot x^{(i)} + b) \geq 1$ (from the constraints above)
- $y^{(i)}(w \cdot x^{(i)} + b) - 1 \geq 0$
- $y^{(i)}(w \cdot x^{(i)} + b) - 1 = 0$ for all $x^{(i)}$ on the margin
- $(w \cdot x^{(i)} + b) = \frac{1}{y^{(i)}}$
- $w \cdot x^+ + b = +1$ and $w \cdot x^- + b = -1$
- $w \cdot x^+ = 1 - b$ and $w \cdot x^- = -1 - b$

Margin

- Margin = $(x^+ - x^-) \cdot \frac{w}{||w||} = \frac{w \cdot x^+ - w \cdot x^-}{||w||} = \frac{(1-b)-(-1-b)}{||w||} = \frac{2}{||w||}$

Objective

- We want to maximize the margin and also classify the data points correctly.
- The objective function can be formulated as:

$$\max \frac{2}{||w||} \text{ s.t. } y^{(i)}(w \cdot x^{(i)} + b) - 1 \geq 0$$

- We can simplify this formulation:
  - Maximizing the margin
    $\max \frac{2}{||w||} \rightarrow \frac{1}{||w||} \rightarrow \min ||\, w \,|| \rightarrow \min \frac{1}{2} ||\, w \,||^2$
  - Classify the data points correctly (assume $x^{(i)}$ is on the margin)
    $y^{(i)}(w \cdot x^{(i)} + b) - 1 \geq 0 \implies b = y^{(i)} - w \cdot x^{(i)}$
    If $x^{(i)}$ is correctly classified and on the margin, then $y^{(i)} = w \cdot x^{(i)} + b$
- The objective function is formulated as a Lagrange function, where α is the Lagrangian multiplier.
  $L(w, b, \alpha) = \frac{1}{2} ||\, w \,||^2 - \sum_i \alpha^{(i)}[y^{(i)}(w \cdot x^{(i)} + b) - 1], \forall_i \alpha^{(i)} \geq 0$
- We want to maximize α and minimize w and b, so we need to find the stationary points of L where the partial derivatives are 0.
  $\max_\alpha \min_{w,b} L(w, b, \alpha)$
  $\frac{\delta L(w,b,\alpha)}{\delta w} = w - \sum_i \alpha^{(i)} y^{(i)} x^{(i)} = 0 \implies w = \sum_i a^{(i)} y^{(i)} x^{(i)}$
  $\frac{\delta L(w,b,\alpha)}{\delta b} = \sum_i \alpha^{(i)} y^{(i)} = 0$
- Plug back w and b into the Lagrange function to obtain the "dual" objective.
  $L(a) = \sum_i \alpha^{(i)} - \frac{1}{2}\sum_i \sum_j \alpha^{(i)}\alpha^{(j)} y^{(i)} y^{(j)} x^{(i)} \cdot x^{(j)}$
- Finally, we have the constrained optimization problem
  - Primal: $\min \frac{1}{2} ||\, w \,||^2$ s.t. $y^{(i)}(w \cdot x^{(i)} + b) - 1 \geq 0$
  - Dual: $\max_{\alpha \geq 0} \sum_i \alpha^{(i)} - \frac{1}{2}\sum_i \sum_j \alpha^{(i)}\alpha^{(j)} y^{(i)} y^{(j)} x^{(i)} \cdot x^{(j)}$

Soft-margin Support Vector Machines

- Introduce **slack variables** and allow for some misclassifications.
- Let $\xi^{(i)}$ be a slack variable for the i-th data point.
- The objective is to **maximize the margin** and **minimize the slacks**.

Decision Rule

- $w \cdot x^+ + b \geq 1 - \xi^{(i)}$
- $w \cdot x^- + b \geq -1 + \xi^{(i)}$

Objective

- Let C be a constant.
- Objective Function: $\min \frac{1}{2} ||\, w \,||^2 + C\sum_i \xi^{(i)}$ s.t.
  $y^{(i)}(w \cdot x^{(i)} + b) - 1 \geq 0 - \xi^{(i)}, \forall_i \xi^{(i)} \geq 0$
- $\xi^{(i)} \geq 1 - y^{(i)}(w \cdot x^{(i)} + b)$
- Objective (Unconstrained):
  $J(w, b) = \frac{1}{2} ||\, w \,||^2 + C\sum_i \max\{0, 1 - y^{(i)}(w \cdot x^{(i)} + b)\}$
- If C is larger, the effect of penalizing the slack variable $\xi^{(i)}$ is greater, resulting in less slack.
- If C is smaller, the effect of penalizing the slack variable $\xi^{(i)}$ reduces, resulting in more slack.

Hinge Loss

- The unconstrained objective function is also called **hinge loss**.
- We recover the standard notation: $b = w_0$.
- Let $\text{cost}_1 = \max\{0, 1 - z\}$ and $\text{cost}_{-1} = \max\{0, 1 + z\}$
- $J(w) = \frac{1}{2}\sum_{j=1}^{n} w_j^2 + C\sum_i \max\{0, 1 - y^{(i)}(w^T x^{(i)})\}$
- $= C\sum_i \max\{0, 1 - y^{(i)}(w^T x^{(i)})\} + \frac{1}{2}\sum_{j=1}^{n} w_j^2$
- $= C\sum_{i=1}^{m}(\frac{(1+y^{(i)})}{2}\text{cost}_1(w^T x^{(i)}) + \frac{1-y^{(i)}}{2}\text{cost}_{-1}(w^T x^{(i)})) + \frac{1}{2}\sum_{j=1}^{n} w_j^2$

Kernel Methods & Kernel Trick

- **Kernel Method** $\rightarrow$ Uses kernel functions to map the feature space to a higher dimension d using a non-linear kernel.
  - Decision boundary becomes a hyperplane of degree $(d-1)$ in the transformed space for $d \geq 2$.
- Using a feature mapping function ϕ, the decision boundary is now $w^T\phi(x) + b \geq 0$.
- Example mappings of $\phi(x)$ could include $[x_1 x_2, x_1^2, x_2^2]$ for data with two features x_1 and x_2.
- However, unlike a kernel function, the mapping function $\phi(x)$ can be computationally expensive or infeasible for high-dimensional feature spaces.
- **Kernel Trick** $\rightarrow$ Allow computation of dot products in the transformed feature space directly using a kernel function $K(u, v) = \phi(u) \cdot \phi(v)$ without explicitly defining or computing $\phi(u)$.
  - Reduces computational cost and handles infinite-dimensional feature spaces, such as those used in Gaussian kernels.

Polynomial Kernel

- Polynomial degree 1:

$$K(u, v) = \phi(u) \cdot \phi(v) = [u_1, u_2]^T \cdot [v_1, v_2]^T = u_1 v_1 + u_2 v_2 = u \cdot v$$

- Polynomial degree 2:

$$K(u, v) = (\phi(u) \cdot \phi(v)) = (u \cdot v)^2$$

- Polynomial degree d:

$$K(u, v) = (u \cdot v)^d$$

Gaussian Kernel (Radial Basis Function)

- The Gaussian kernel is defined as:

$$K(u, v) := e^{-\frac{\|u-v\|^2}{2\sigma^2}}$$

- $\phi(u)$ maps data to an **infinite-dimensional** feature space.
- The transformed features are never explicitly computed, as the kernel trick is used to calculate $K(u, v)$ directly.

Perceptron

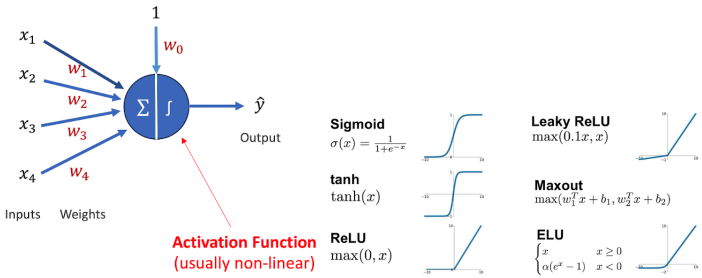
$$\hat{y} = h_w(x) = g(\sum_{i=0}^{n} w_i x_i) \quad (1)$$

- A perceptron multiplies each input with its corresponding weight and passes this value through an activation function $g(z)$ to produce the final output.
- Activation function (sign function): $g(z) = 1$ if $z \geq 0$ otherwise -1.

Perceptron Learning Algorithm

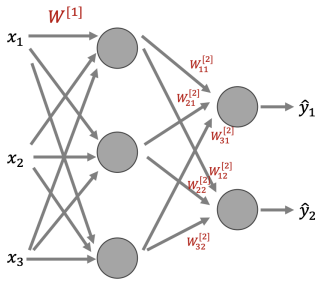
- Initialize $\forall_i w_i$
- Loop (until convergence or max steps exceeded)
  - For each instance $(x^{(i)}, y^{(i)})$, classify $\hat{y}^{(i)} = h_w(x^{(i)})$
  - Select one **misclassified** instance $(x^{(j)}, y^{(j)})$
  - Update weights: $w \leftarrow w + \gamma(y^{(j)} - \hat{y}^{(j)})x^{(j)}$ where γ is the learning rate.

Neural Networks



- In neural networks, we can use different activation functions.

Forward Propogation: Matrix Multiplication



- $x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$
- $W^{[1]} = \begin{bmatrix} W^{[1]}_{11} & W^{[1]}_{12} & W^{[1]}_{13} \\ W^{[1]}_{21} & W^{[1]}_{22} & W^{[1]}_{23} \\ W^{[1]}_{31} & W^{[1]}_{32} & W^{[1]}_{33} \end{bmatrix}$
- $W^{[2]} = \begin{bmatrix} W^{[2]}_{11} & W^{[2]}_{12} \\ W^{[2]}_{21} & W^{[2]}_{22} \\ W^{[2]}_{31} & W^{[2]}_{32} \end{bmatrix}$

- $\hat{y} = g^{[2]}(W^{[2]^T} g^{[1]}(W^{[1]^T} x))$
- $\hat{y} = g^{[2]}(\begin{bmatrix} W^{[2]}_{11} & W^{[2]}_{12} \\ W^{[2]}_{21} & W^{[2]}_{22} \\ W^{[2]}_{31} & W^{[2]}_{32} \end{bmatrix}^T g^{[1]}(\begin{bmatrix} W^{[1]}_{11} & W^{[1]}_{12} & W^{[1]}_{13} \\ W^{[1]}_{21} & W^{[1]}_{22} & W^{[1]}_{23} \\ W^{[1]}_{31} & W^{[1]}_{32} & W^{[1]}_{33} \end{bmatrix}^T \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix})) = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \end{bmatrix}$

General Case $\hat{y} = g^{[L]}(W^{[L]^T} \dots g^{[2]}(W^{[2]^T} g^{[1]}(W^{[1]^T} x))) = \begin{bmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_c \end{bmatrix}$

Neural Network: Tasks

Task	Output	Activation Function	Range
Regression	1	Linear / No Activation $g(x) = x$	$\hat{y} \in \mathbb{R}$
Binary Classification	1	Sigmoid $g(x) = \frac{1}{1+e^{-x}}$	$\hat{y} \in [0, 1]$
Multi Class Classification	C	Softmax $g(z) = \frac{e^{z_i}}{\sum_C^{j=1} e^{z_j}}$	$\hat{y} \in [0, 1]$ $\hat{y}_i \in [0, 1]$ $\hat{y}_c \in [0, 1]$

Neural Networks vs Other Models

Model	Feature Mapping	Activation	Decision Boundary
Logistic Regression	None	Sigmoid	Linear, non-robust
Logistic Regression	Handcrafted	Sigmoid	Non-linear, non-robust
Support Vector Machines	Handcrafted	Sign	Non-linear, robust
Neural Networks	Learned	Depends	Non-linear, non-robust

- **Non-robust Decision Boundary**: Prone to misclassification since the decision boundary can be too close to the data points.
- **Robust Decision Boundary**: Decision boundary is guranteed to be far from the data points.

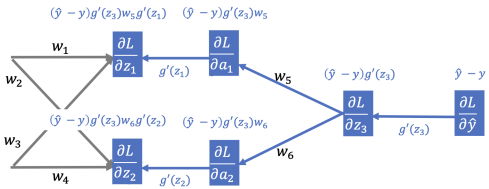
Backpropogation

Background: Chain Rule

- Single variable
  - Let $a = f(x)$ and $z = g(a)$.
  - $\Delta x \to \delta A \to \Delta z$
  - $\frac{\delta z}{\delta x} = \frac{\delta z}{\delta a} \frac{\delta a}{\delta x}$
- Multi variable
  - Let $a = f(x)$, $b = g(z)$ and $z = h(a, b)$.
  - $\frac{\delta z}{\delta x} = \frac{\delta z}{\delta a} \frac{\delta a}{\delta x} + \frac{\delta z}{\delta b} \frac{\delta b}{\delta x}$

Backpropogation: General Steps

- Assume this neural network uses a **non-linear** activation $g(x)$ and that $g(x)$ is differentiable



1. Do **forward propogation**: Express each neuron as a weighted sum of its weights and inputs and the activation function.
2. Start by finding $\frac{\delta L}{\delta \hat{y}}$ the rate of change of the loss function w.r.t $\hat{y}$
3. Suppose z_3 is the weight sum of the neuron in layer 3 before being put into the activation function. Then find $\frac{\delta \hat{y}}{\delta z_3}$ which is the rate of change of $\hat{y}$ w.r.t z_3.
4. From there we can find the rate of change with respect to the weights and the prior layers.

Convolution Neural Networks

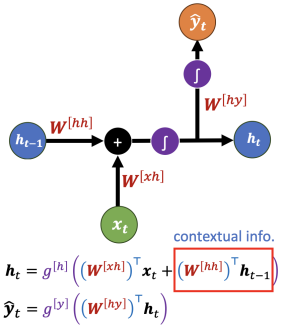
Convolution Layer

- Let image X be a 2D image of width w and height h.
- Let k be the kernel dimension, p be the padding dimension and s be the stride.
- Output Row = $\lfloor \frac{w-k+2p}{s} \rfloor + 1$
- Output Height = $\lfloor \frac{h-k+2p}{s} \rfloor + 1$
- Number of trainable parameters = $(\text{input channels} \times k \times k + 1) \times \text{output channels}$

Pooling Layer

- The pooling layer **downsamples** feature maps (output from convolution layer).
- The pooling layer steps are similar to the convolution layer steps, except that **aggregation methods** are performed on the $k \times k$ window instead.
- **Aggregation Methods:** Max-pool, Average-pool, Sum-pool
- No trainable parameters

Recurrent Neural Networks

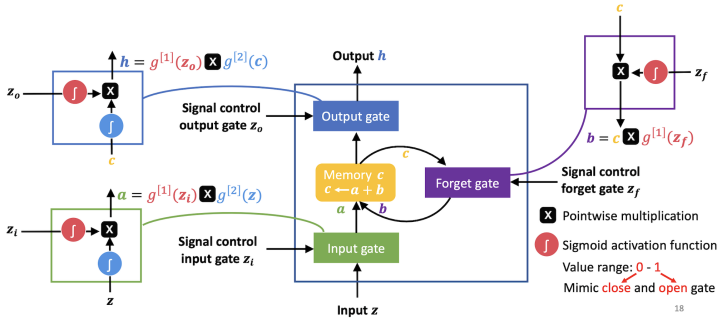


Bidirectional RNN

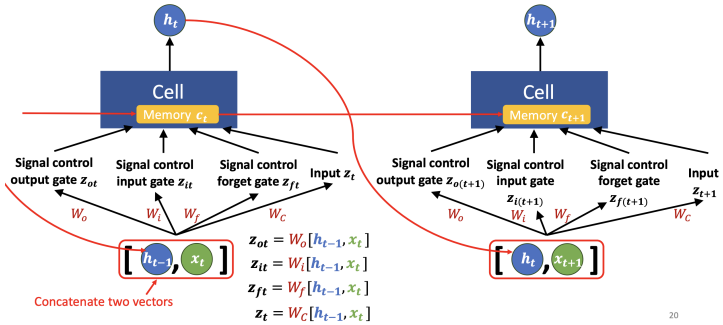
- Used to process data in both the forwards and backwards timestamp.
- It concatenates both the information before x_t and the information after x_t to predict the output.

Long Short-Term Memory (LSTM)

Motivation: Maintain relevant context over time, i.e. *forget* and *remember*.



- The concatenated vector $[h_{t-1}, x_t]$ multiplied by the respective weight matrix for each signal or input is used to generate the signal or input for one LSTM cell.



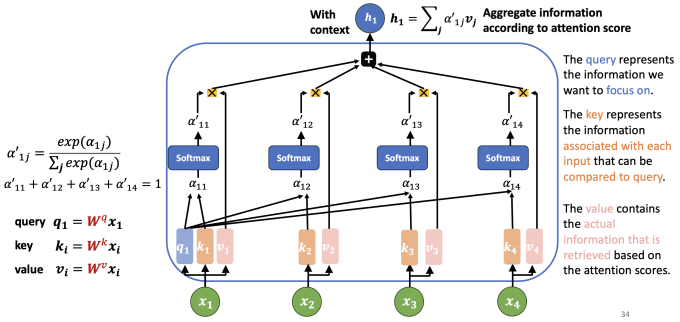
RNN Types and Applications

Type	Example
One to One $T_x = T_y = 1$	Traditional Neural Network
One to Many $T_x = 1, T_y > 1$	Music Generation
Many to One $T_x > 1, T_y = 1$	Sentiment Analysis (Classification)
Many to Many $T_x = T_y$	Name Entity Recognition
Many to Many $T_x \neq T_y$	Machine Translation Speech Recognition

Attention Score

- **Query** $\to \mathbf{q}_i = W^q \mathbf{x}_i$ represents the current token we want focus on.
- **Key** $\to \mathbf{k}_j = W^k \mathbf{x}_j$ represents the information associated with each input to be compared with the query. (denoted by k)
- **Attention Score** $\to \mathbf{v}_j = W^v \mathbf{x}_j$ represents the information that each token contributes based on the attention score (scalar value).
- All weight matrices W are learned during training.

Self Attention Mechanism



Goal: Measure the relevance/importance of one token x_i *w.r.t* all other tokens in sequence.

1. **Attention Scores Computation**: $\alpha_{ij} = \mathbf{q}_i \cdot \mathbf{k}_j$, where $\alpha_{ij} \in \mathbb{R}$.
2. **Normalise Scores**: Use softmax to convert attention scores to probabilities, i.e. $\alpha'_{ij} = \frac{exp(\alpha_{ij})}{\sum_j exp(\alpha_{ij})}$ and $\sum_j \alpha'_{ij} = 1$.
3. **Weighted Sum**: $h_i = \sum_j \alpha'_{ij} \mathbf{v}_j$.

Note: Each h_i can be calculated in parallel.

- The attention score matrix A is a $n \times n$ matrix for n inputs.
- $h_i = \sum_j \alpha'_{ij} v_j$

Transformers

Definition: Deep (layers of encoders and decoders) based on self-attention mechanism.

- **Encoder**: Encodes input sequence into a series of representations.
- **Decoder**: Generates the output sequence.
- **Encoder-Decoder Attention**: Enable decoder to utilise rich contextual information **provided by encoder**.
  - **Query**: Generated based on previous decoder's block output.
  - **Key, Value**: Generated based on encoder's output.

Dropout/Early Stopping

Goal: Perform regularisation to prevent overfitting.

- Networks with high capacity can easily overfit to the training data.
- **Randomly** set a subset of activations in a layer to 0 during forward pass.
- Force network to learn more robust and generalized features by introducing randomness.
- **Early Stopping**: Halting training process when $J_{D_{val}}$ stops improving.

Vanishing/Exploding Gradient

- **Vanishing Gradient**: **Small** $\frac{\partial L}{\partial w_i} \to 0$ as it gets repeatedly multiplied during back-propagation. (Solution: non-saturating $g(x)$, e.g. ReLU).
- **Exploding Gradient**: **Large** $\frac{\partial L}{\partial w_i} \to e^x$ as it gets repeatedly multiplied during back-propagation. (Solution: gradient clipping, i.e. set range of min,max values for gradient).

Unsupervised Learning

Clustering $\to$ Given a set of m data points $\{x^{(1)}, \ldots, x^{(m)}\}$, identify $k \geq 2$ clusters in the data. ($x^{(i)}$ can be high dimensional, i.e. many features)
Centroid $\to$ **Mean position** of all data points $\mathbf{x}^{(i)}$ for $i = 1, \ldots, m_1$ within a cluster j, used as a reference point to determine a given data point's cluster.

$$\mu_j = \frac{1}{m_1} \sum_{i=1}^{m_1} \mathbf{x}^{(i)}$$

K-Means Algorithm

Goal: Find K clusters.

1. **Randomly** initialize K centroids, $\mu_1, \ldots, \mu_K \in \mathbb{R}^n$.
2. Repeat until convergence:
   2.1. **Cluster Assignment**: For $i = 1, \ldots, m$, $c^{(i)} = \arg\min_j ||x^{(i)} - \mu_j||^2$
   2.2. **Update Centroids**: For $k = 1, \ldots, K$, $\mu_k = \frac{1}{|c_k|} \sum_{x^{(i)} \in c_k} x^{(i)}$

K-Means Convergence

- **Local Optima**: K-Means may be stuck depending on **initial assignment** of centroids.
- **Distortion/Loss Function**: Average distance of each sample to its centroid. For K clusters and m features:

$$J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K) = \frac{1}{m} \sum_{i=1}^{m} ||x^{(i)} - \mu_{c^{(i)}}||^2$$

- **Number of Clusters**: Elbow method where distortion as K increases. (Data may not have an elbow/have multiple elbows)

K-Means Variants

- Pick K initial centroids randomly from the points in the data.
- **K-Medoids**: Pick the data points that are **closest** to the centroids, and use them as the centroids.
  - Good for scenarios where low dimensional data points exist in high dimensional vector space.

Hierarchical Clustering

Motivation: When we cannot decide on a fixed number of clusters.

Agglomerative Clustering (Bottom-Up)

1. Start with each data point as its own cluster.
2. Iteratively merge the **closest** clusters until all data points belong to a single cluster. (Find a pair of clusters that is the "nearest" to merge them together)

Note: High space and time complexity $\Rightarrow$ Impractical for large datasets.

Clustering Combination Methods

Method	Description
Centroid Method	Combining clusters with minimum distance between the centroids of the two clusters
Single Linkage	Minimum distance between the closest elements in clusters $D(c_1, c_2) = \min D(x_1, x_2)$
Complete Linkage	Maximum distance between elements in clusters $D(c_1, c_2) = \max D(x_1, x_2)$
Average Linkage	Average of the distances of all pairs $D(c_1, c_2) = \frac{1}{\lvert c_1 \rvert} \frac{1}{\lvert c_2 \rvert} \sum \sum D(x_1, x_2)$

Dimensionality Reduction

Definition: Find a lower-dimensional representation of the data, i.e. given $x^{(i)} \in \mathbb{R}^r$, find $\tilde{x}^{(i)} \in \mathbb{R}^r$ s.t. $r < n$.

- Number of samples increases exponentially with the number of features.
- **Reduction**: Identify "most important" (variation in data) features. Project data into lower dimension by removing "non-important" components.
- **Reconstruction**: Project data into higher dimension by adding "non-important" components.
- **Change of Basis**: Remove dependence between components back.

Preliminaries: Linear Algebra

- **Orthonormal Sets**: Let $S = \{\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_n}\} \subseteq \mathbb{R}^n$. Set S is orthonormal if $\forall\, i, j \in |S|$:

$$\mathbf{v}_i \cdot \mathbf{v}_j = \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{otherwise} \end{cases}$$

- $\mathbf{v}_i \cdot \mathbf{v}_i = 1 \Rightarrow ||\mathbf{v}_i|| = \sqrt{\mathbf{v}_i \cdot \mathbf{v}_i} = 1$
- **Orthonormal Basis**: $\forall\, \mathbf{u} \in \mathbb{R}^n$ can be written as a unique linear combination of $\mathbf{v} \in S$.

Compact SVD

Intuition: Decompose matrix X to its most significant patterns using only top k singular values and corresponding singular vectors.
Fact: All $n \times m$ matrices X can be factorised as $X = U\Sigma V^\top$.

- Each column vector $x_i \in X, i \in 1 \ldots m$ represents each data point.
- Each element in x_i (row element) represents each of the m features.
- **Left Singular Vectors**: U is $n \times m$ and has m **orthonormal** columns. (Mapping from the original feature space to the principal component space)
- **Singular Values**: Σ is $m \times m$ and is diagonal with $\sigma_j \geq 0$. **Ordered by importance**, i.e. largest to smallest. (Importance of each principal component)
- **Right Singular Vectors**: V is $m \times m$ and has m **orthonormal** rows and columns. (Define each principal component)

Reducing Dimensions using SVD

Key Idea: Set all singular values except first r to 0, $\sigma_{j>r} = 0$.

- **Reduction**: $X \approx \widetilde{U}\widetilde{\Sigma}\widetilde{V}^\top \Rightarrow Z := \widetilde{U}^\top X$ where Z is $r \times m$ **reduced data**.
- **Reconstruction**: $\widetilde{X} := \widetilde{U}Z$, where $\widetilde{X}$ is $n \times m$. It can be shown that $\widetilde{X} \approx X$ (best approximation) by $\widetilde{U}^\top \widetilde{U} = I_r$ due to orthonormality of columns in $\widetilde{U}$.
- **Encoder-Decoder Structure**: r-SVD defines structure for reduction and reconstruction of high dimensional data.

PCA

Definition: Capture components that maximises the *statistical variations* of the data, i.e. directions of basis that has highest variance.
Standard Identities

- $Var(X) = E[(x - \overline{x})^2]$
- $Cov(x, y) = E[(x - \overline{x}) - (y - \overline{y})]$

Covariance Matrix

Note: In many cases, random variables are **not independent** in data matrix $X = (\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}) \Rightarrow$ non-zero off-diagonals in covariance matrix.

1. Compute **mean-centered data**: $\hat{\mathbf{x}}^{(i)} = \mathbf{x}^{(i)} - \overline{x}$.
2. Let $\hat{X} = (\hat{\mathbf{x}}^{(1)}, \ldots, \hat{\mathbf{x}}^{(m)})$
3. Compute covariance matrix: $Cov(X) = \frac{1}{m} \hat{X} \hat{X}^\top$

Choosing Number of Components

Goal: Find suitable r s.t. at least $p\%$ of variance is retained.
Suppose we were given a data matrix $X = (x^{(1)}, \ldots, x^{(m)})$

1. Compute covariance matrix $Cov(X) = \frac{1}{m} \hat{X} \hat{X}^\top$.
2. Compute SVD, i.e. $Cov(X) = U\Sigma V^\top$, to obtain U (new basis)
3. Choose **minimum** r s.t. $\frac{\sum_{i=1}^{r} \sigma_i^2}{\sum_{i=1}^{m} \sigma_i^2} \geq \frac{p}{100}$.
4. Reduce to r components to obtain $\widetilde{U}$.