# CS2109S CheatSheet AY24/25 —— @Jin Hang

## Basic Concepts

**PEAS Framework:**
- **Performance measure:** define "goodness" of a solution
- **Environment:** define what the agent can and cannot do
- **Actuators:** outputs
- **Sensors:** inputs

**Properties of Task Environment**

**Fully observable:** (vs. **Partially** observable) An agent's sensors give it access to the **complete** state of env. at each point in time.
**Deterministic:** (vs. **Stochastic**) Next state of env. is completely determined by current state and action executed by the agent.
**Strategic:** env. is **deterministic** except for actions of other agents.
**Episodic:** (vs. **Sequential**) Agent's experience is divided into atomic "episodes". Each episode consists of agent perceiving and then performing a single action. The choice of action in each episode depends **only** on the episode itself.
**Static:** (vs. **Dynamic**) The env. is unchanged while an agent is deliberating.
**Semi-dynamic:** env. itself does not change with passage of time, but the agent's performance score does.
**Discrete:** (vs. **Continuous**) A limited number of distinct, clearly defined percepts and actions.
**Single agent:** (vs. Multi.) An agent operating by itself in an env.

**Agents**
- The agent function maps from percept histories to actions.
- An agent is completely specified by the agent function.
- A rational agent will choose actions that maximize 'P'
- **Exploration:** Learn more about the world
- **Exploitation:** Maximize gain based on current knowledge

**Uninformed Search**

**Trick:** Assume we know correct max depth for IDS/DLS
- Terminate (have sol): BFS/DLS/IDS
- Terminate (No sol): DLS
- Find Answer: BFS/DLS/IDS

**Graph Search:** Additional $O(b^m)$ memory to store all visited nodes to avoid revisiting states.

**BFS**: Queue
- $T(n) = 1 + b + b^2 + \cdots + b^d = O(b^d)$ (# nodes generated)
- $S(n) = O(b^d)$
- **Worst case:** Expand the last child in a branch
- **Complete:** Yes, if B is finite. If a solution **exists**, then the depth of the shallowest node s must be finite, so BFS must eventually search this depth. Hence, it's complete.
  - Apply goal-test when **PUSHING** a successor state to the frontier to preserve completeness.
- **Optimal:** Generally not optimal because it simply does not take costs into consideration when determining which node to replace on the frontier.
  - Optimal is if all edge costs are equivalent.

**DFS**: Stack
- $T(n) = O(b^m) \Rightarrow O(b^l)$ for DLS
- $S(n) = O(bm) \Rightarrow O(bl)$ for DLS
- **Complete:** No, when depth is infinite or can go back. There exists the possibility that DFS will faithfully yet tragically get "stuck" searching for the deepest node in an infinite-sized search tree $\Rightarrow$ **may never find a solution**.
- **Optimal:** No, DFS simply finds the "**leftmost**" solution in the search tree without regard for path costs.

**Iterative Deepening Search (IDS)**
- $T(n) = b^0 + (b^0 + b^1) + \cdots + \left(b^0 + \cdots + b^d\right) = (d+1)b^0 + db^1 + (d-1)b^2 + \cdots + 2b^{d-1} + b^d = O(b^d)$
- $S(n) = O(bd)$
- Complete: Yes
- Optimal: Yes, if step cost is the same everywhere
- Overhead = (#IDS - #DLS)/#DLS

**Uniform-cost Search (UCS)**: Priority Queue (path cost)
- $T(n) = O(b^{C^*/\epsilon})$
- $S(n) = O(b^{C^*/\epsilon})$
- **Complete:** Yes, if $\epsilon > 0$ and $C^*$ finite. If a goal state **exists**, it must have some finite length shortest path; $\Rightarrow$ must find shortest path.
- **Optimal:** Yes, if $\epsilon > 0$
  - $C^*$ cost of optimal solution
  - $\epsilon$ minimum edge cost. $\epsilon = 0$ may cause zero cost cycle

**Bidirectional Search:** Forward (from start) and Backward (from goal): $\Rightarrow 2 \times O\left(b^{d/2}\right) < O(b^d)$

**Greedy Best-first Search:** Priority Queue ($f(n) = h(n)$)
- $T(n), S(n) = O(b^m)$, good heuristic gives improvement
- Complete and Optimal: **No**. Particularly when useing a bad heuristic function. It acts unpredictably from scenario to scenario, and can range from going straight to a goal state to acting like a badly-guided DFS and exploring all wrong areas.

**A* Search:** Priority Queue ($f(n) = g(n) + h(n)$)
- $T(n), S(n) = O(b^m)$, good heuristic gives improvement
- Complete: Yes
- Optimal: Yes
  - If $h(n)$ **admissible** and using **tree search**.
  - if $h(n)$ is **consistent** and using **graph search**.
  - Work with negative edge weights.
  - A* **DO NOT** work with negative heuristics even admissible

**Admissible:** A heuristic $h(n)$ is admissible if $\forall n,\ h(n) \leq h*(n)$
- $h*(n)$ is the true cost to reach the goal state from $n$.
- **Conservative:** An admissible heuristic **never** overestimates the cost to reach the goal

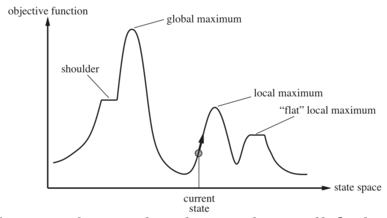
**Manhattan Distance:** $\text{MD}(x_1, y_1, x_2, y_2) = |x_1 - x_2| + |y_1 - y_2|$
Let $p_i$ be the current location
- $\frac{1}{k}\sum_{i=1}^{k} \text{MD}(p_i, g) \leq \max\{\text{MD}(p_i, g)\} \leq h^*(n)$
- $\frac{1}{2}(\max\{\text{MD}(p_i, g)\} + \min\{\text{MD}(p_i, g)\}) \leq \max\{\text{MD}\} \leq h^*(n)$
- $\max\{\min\{\text{MD}(p_i, g)\}\} \leq h^*(n)$

**Consistent:** $\forall n$, every successor $n'$ of $n$ generated by any action $a$, $h(n) \leq c(n, a, n') + h(n')$, and $h(G) = 0$.$h$ is consistent $\Rightarrow$ $f(n') = g(n') + h(n') = g(n) + c(n, a, n') + h(n') \geq g(n) + h(n) = f(n)$ $\Rightarrow$ $f(n)$ is **non-decreasing** along any path
- **Theorem:** If $h(G) = 0$, then $h(n)$ is consistent $\Rightarrow$ admissible
- Admissibility **does NOT** imply consistency.

**Dominance:** $\forall n, h_2(n) \geq h_1(n)$, then $h_2$ dominates $h_1$ $\Leftrightarrow$ $h_2$ is better for search if admissible.
- No dominance relationship if $h(n)$ is not admissible.

**Local Search**
- **Goal:** minimizes the number of conflict violations
- Almost constant time, but **incomplete and sub-optimal**
- We are not interested in obtaining the solution path, but rather, reaching the goal state.
- When the search space is huge, using informed search could take a very long time.

**Hill climbing algorithm**
- The algorithm **does not** maintain a search tree but only the states and the corresponding values of the objective.
- May be trapped in **local maxima**
- **Simulated Annealing:**
  $\mathbf{P} = e^{\frac{\text{value(next)} - \text{value(curr)}}{T}}$



**Theorem:** if T decreases slowly enough, simulated annealing will find a global optimum with high probability.

**Mini-Max**
- $T(n) = O(b^m), S(n) = O(bm)$, with depth first exploration
- **Complete:** Yes, if tree is finite
- **Optimal:** Yes, against optimal opponent

**Alpha-beta Pruning:** Won't change the decision
- Good move order improves effectiveness of pruning $\Rightarrow O\left(b^{\frac{m}{2}}\right)$

**Evaluation Functions:** There is **no** notion of admissibility and consistency in local search and adversarial search.

## Supervised Learning
- **Regression:** predict **continuous** output
- **Classification:** predict **discrete** output

**Formalism** Assume that $y$ is generated by $f: x \to y$. We want to find a hypothesis $h$:$x \to \widehat{y}$ (from a hypothesis class $H$) s.t. $h \approx f$ given a training set $\{(x_1, f(x_1)), ..., (x_N, f(x_N))\}$ We use a **learning algorithm** to find this hypothesis

**Error:** If the output of the hypothesis is a continuous value
- MSE$= \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$, where $\hat{y}_i = h(x_i)$ and $y_i = f(x_i)$
- MAE$= \frac{1}{N}\sum_{i=1}^{N}\|\widehat{y}_i - y_i\|$

## Decision Tree
- **Continues-valued Attributes:** Define a discrete-valued input attribute to partition the values into a discrete set of intervals.
- **Missing Values:**
  - Assign the most common value of the attribute
  - Assign the most common value of attribute with same output
  - Assign probability to each possible value and sample
  - Drop the attribute
  - Drop the rows
- **Accuracy:** $\frac{\text{TP+TN}}{\text{TP+FN+FP+TN}}$
- **Precision:** $P = TP/(TP + FP)$
- **Recall:** $R = TP/(TP + FN)$
- **F1 Score:** $F1 = \frac{2}{\frac{1}{P} + \frac{1}{R}}$
- **Maximize** Precision/Recall $\Leftrightarrow$ **Minimize** FP/FN (Depends on context)

| | | Actual Label | |
|---|---|---|---|
| | | Positive | Negative |
| Predicted Label | Positive | TP | FP |
| | Negative | FN | TN |

- **Entropy:** $I(P(v_1), ..., P(v_n)) = -\sum_{i=1}^{n} P(v_i)\log_2 P(v_i)$
- **Remainder:** Entropy of children nodes
  remainder(A)$=\sum_{i=1}^{v}\frac{p_i+n_i}{p+n}I(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i})$
- **Information Gain:** $IG(A) = I(\frac{p}{p+n}, \frac{n}{p+n}) - \text{remainder}(A)$

**Overfitting:** DT performance is perfect on training data, but worse on test data. DT captures data perfectly, including the noise.
**Occam's Razor:** Prefer short/simple hypotheses. In favor:
- Short/simple hypothesis that fits the data is unlikely to be coincidence
- Long/complex hypothesis that fits the data may be coincidence

Against:
- Many ways to define small sets of hypotheses (e.g., trees with prime number of nodes that uses attribute beginning with "Z")
- Different hypotheses representations may be used instead

**Linear Regression**

**MSE:** $J_{MSE}(w) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_w(x^{(i)}) - y^{(i)}\right)^2$
- MSE loss function is **convex** for linear regression.

**Gradient Descent** $w_j \leftarrow w_j - \gamma\frac{\partial J(w_0, w_1, ..., w_n)}{\partial w_j}$
- $a = \frac{\partial J(w_0, w_1)}{\partial w_0}$, $b = \frac{\partial J(w_0, w_1)}{\partial w_1}$
- $w_0 = w_0 - \gamma a$, $w_1 = w_1 - \gamma b$

**Batch GD:** Consider all training examples. Much likely to get stuck in local minima
**Mini-batch GD:** Consider a subset of training examples at a time; Cheaper (Faster) per iteration; **Randomness**, **may escape** local minima
**Stochastic GD:** Select **one** random data point at a time; Cheapest (**Fastest**) per iteration; **More randomness**, may escape local minima

**Normalization:** $x_j \leftarrow \frac{x_j - \mu_j}{\sigma_j}$ deal with features of diff. scales.
- Preserves the relationships between the data points

**Normal Equation** Set $\frac{\partial J_{MSE}(w)}{\partial w} = 0 \Rightarrow w = (X^TX)^{-1}X^TY$

$$X = \begin{bmatrix} 1 & x_1^{(1)} & \dots & x_n^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(m)} & \dots & x_n^{(m)} \end{bmatrix} \quad Y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \dots \\ y^{(m)} \end{bmatrix} \quad w = \begin{bmatrix} w_0 \\ w_1 \\ \dots \\ w_n \end{bmatrix}$$

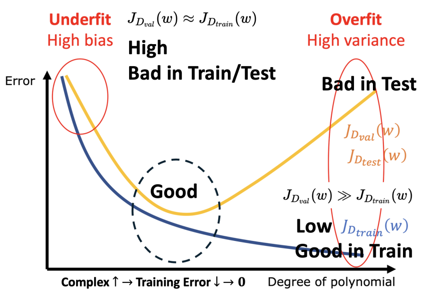
| | Gradient Descent | Normal Equation |
|---|---|---|
| $\gamma$ | Need to choose | No Need |
| Iteration(s) | Many | None |
| large # of features | No Problem | Slow $(X^TX)^{-1} \to O(n^3)$ |
| Feature scaling | May be necessary | Not necessary |
| Constraints | | $X^TX$ should be invertible |

**Logistic Regression**

**Logistic function (Sigmoid)** $\sigma(z) = \frac{1}{1+e^{-z}}$
- $\sigma(z)' = \sigma(z)(1 - \sigma(z)) \in [0, 0.25]$
- $\Pr(y = 1) = p = \sigma(h_w(x)) = \frac{1}{1+e^{-h_w(x)}}$
- $\frac{\partial \log(p)}{\partial w_i} = -\left(\frac{1}{1+e^{-h_w(x)}}\frac{\partial}{\partial w_i}(1 + e^{-h_w(x)})\right) = -p(-\frac{\partial h_w(x)}{\partial w_i})e^{-h_w(x)} = p(\frac{\partial h_w(x)}{\partial w_i})(\frac{1}{p} - 1) = (1 - p)\frac{\partial h_w(x)}{\partial w_i}$
- $\frac{\partial \log(1-p)}{\partial w_i} = -\frac{\partial h_w(x)}{\partial w_i} + \frac{\partial \log(p)}{\partial w_i} = -p(\frac{\partial h_w(x)}{\partial w_i})$
- When $h_w(x) = w^Tx = \sum_{i=1}^{n} w_ix_i$, we have $\frac{\partial h_w(x)}{\partial w_i} = x_i$

**Binary cross-entropy (BCE)** Convex for logistic regression
- $\text{BCE}(y,\hat{y}) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$
- $\frac{\partial \text{BCE}(y, h_w(x))}{\partial w_i} = \frac{\partial h_w(x)}{\partial w_i}(h_w(x) - y)$
- **BCE Loss** $J_{BCE}(w) = \frac{1}{m}\sum_{i=1}^m BCE\left(y^{(i)}, h_w(x^{(i)})\right)$
- Single data with incorrect prediction $\rightarrow \infty$ loss $\Rightarrow$ Not suitable
- Often used for classification tasks

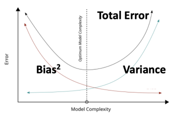
**Multi-class Classification:** Fit one classifier per class, fit against all other classes. Pick **highest probability**.
- Sensitivity: TPR=Recall=TP/(TP+FN)
- 1-Specificity: FPR=FP/(FP+TN)
- Choose threshold that have high TPR and low FPR.
- AUC: Area under ROC curve and above x-axis $> 0.5 \Rightarrow$ better than chance; $\simeq 1 \Rightarrow$ very accurate.
- Model is more accurate than random chance if ROC curve is above the diagonal random line.



**Reduce Bais**
- Complexity ↑
- More Feature

**Reduce Variance**
- Regularization
- Use More Data

**Bais-Variance Trade-off**



<u>**Evaluating the model**</u> Accuracy and AUC-ROC metrics

**Loss Function:** Assess proximity of predict to actual

<u>**Address Over-fitting**</u>
- Reduce the number of features $\Rightarrow$ High poly degree $\rightarrow$ Low
- Regularization: Keep all features but reduce magnitude $w_i$

**Regularization:** Parameter $\lambda$. For $h_w(x) := w^T x$, we have
- **(L2:)** $J(w) = \frac{1}{2m}\left[\sum_{i=1}^m (h_w(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{i=1}^n w_j^2\right]$
- **GD:** $w_n \leftarrow w_n - \gamma\frac{1}{m}[\sum_{i=1}^m (h_w(x^{(i)}) - y^{(i)})x_n^{(i)} + \lambda w_n]$
- **Normal Equation** Works even if $X^TX$ non-invertible if $\lambda > 0$!

$$\lambda\sum_{j=1}^n w_j^2 \rightarrow \lambda\begin{bmatrix}0&0&0&0\\0&1&0&0\\0&0&\ddots&0\\0&0&0&1\end{bmatrix} \rightarrow w = \left(X^T\hat{X} + \lambda\begin{bmatrix}0&0&0&0\\0&1&0&0\\0&0&\ddots&0\\0&0&0&1\end{bmatrix}\right)^{-1} X^TY$$

$X: m\times(n+1)$
**Triangle:** $(n+1)\times(n+1)$
$Y: m$

- Heavily penalises larger para. Attempts to pull **all** para. small

**L1:** Use $\lambda\sum_{i=1}^n |w_i| \Rightarrow$ Reduce # of features(set less important to 0) [**Feature Selection**] $\rightarrow$ Compexity$\downarrow$ $\rightarrow$ prevent over-fitting

<u>**SVM**</u>
- Decision Rule: $w\cdot x + b \geq 0$ then $+$
- $y^{(i)}(w\cdot x^{(i)} + b) - 1 = 0$ for all $x^{(i)}$ on **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:** $\max\frac{2}{\|w\|}$ s.t. $y^{(i)}(w\cdot x^{(i)} + b) - 1 \geq 0$ [Classify correctly] $\Leftrightarrow \min\frac{1}{2}\|w\|^2$

**Soft-Margin:** Allow misclassifications [Higher $C \rightarrow$ less slack]
- $\min\frac{1}{2}\|w\|^2 + C\sum_i \xi^{(i)}$ s.t. $y^{(i)}(w\cdot x^{(i)} + b) - 1 \geq 0 - \xi^{(i)}$ $\Rightarrow \xi^{(i)} \geq 1 - y^{(i)}(w\cdot x^{(i)} + b) \geq 0$
- $J(w,b) = \frac{1}{2}\|w\|^2 + C\sum\max\{0, 1 - y^{(i)}(w\cdot x^{(i)} + b)\}$

**Non-linear decision boundary:** $w^T\phi(x) \geq 0$ then $+ \rightarrow \phi$ can produce a huge number of features [Not scalable]

$\Rightarrow$ SVM: Let $w = \sum_i \alpha^{(i)}y^{(i)}\phi(x^{(i)})$, we have part $\phi(x^{(i)})\phi(x)$
- 1D: $\phi(x) = [x, x^2]^T \Rightarrow K(u,v) = \phi(u)\phi(v) = u\cdot v$
- 2D: $\phi(x) = [x_1, x_2, x_1x_2, x_1^2, x_2^2, ...]^T \Rightarrow K(u,v) = (u\cdot v)^2$
- $n^d$ Terms: $K(u,v) = (u\cdot v)^n$ [**Kernel**]

**Kernel Trick:** map training set into a different space $\rightarrow$ linear

<u>**Neural Network**</u>

**Sign Function:** $g(z) = \begin{cases}+1 & \text{if } z \geq 0\\ -1 & \text{if } z < 0\end{cases}$

**ReLU** $g(x) = \max(0, x) \Rightarrow g'(x) = 1_{x>0}$
- Single-layer perceptron is a **linear** classifier
- Multi-layer perceptron can learn non-linear decision boundaries

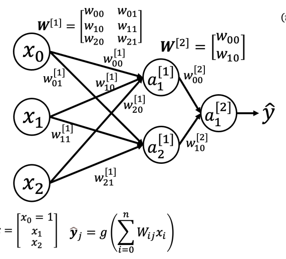
**Perceptron Learning Algorithm**
- Initialize $\forall_i w_i$
- Loop (until convergence or max steps reached)
  - For each $(x^{(i)}, y^{(i)})$, classify $\hat{y}^{(i)} = h_w(x^{(i)}) = g(w^Tx)$
  - Select one **misclassified** instance
  - Update weights: $w \leftarrow w + \gamma(y^{(j)} - \hat{y}^{(j)})x^{(j)}$

**Vanishing gradient:** $\because \sigma'(z) \in [0, 0.25] \Rightarrow$ if we $\times$ many small number together $\Rightarrow w$ update small $\Rightarrow$ **slow** convergence.

**Exploding gradient:** large gradients got multiplied again and again until it overflows. **Mitigation:**
- Using Non-saturating Activation Functions, e.g. ReLU
- Gradient Clipping

**Gradient Descent:** For signal layer $w_i \leftarrow w_i - \gamma\frac{dL}{dw_i}$

**For Multi-Layer:**



$$\underset{(m+1)\times 1}{a^{[p]}} = W^{[p]T}x = \underset{(n+1)\times(m+1)}{\begin{bmatrix}W_{00} & \cdots & W_{n0}\\ \vdots & \ddots & \vdots\\ W_{0m} & \cdots & W_{nm}\end{bmatrix}}\begin{bmatrix}x_0\\ \vdots\\ x_n\end{bmatrix} = \begin{bmatrix}\sum_{i=0}^n W_{i0}x_i\\ \vdots\\ \sum_{i=0}^n W_{im}x_i\end{bmatrix}$$

$$L = \frac{1}{2}(\hat{\boldsymbol{y}} - \boldsymbol{y})^2$$

$$\frac{\partial L}{\partial w_{21}^{[1]}} = \frac{\partial L}{\partial\hat{y}}\frac{\partial\hat{y}}{\partial z_1^{[2]}}\frac{\partial z_1^{[2]}}{\partial a_2^{[1]}}\frac{\partial a_2^{[1]}}{\partial z_2^{[1]}}\frac{\partial z_2^{[1]}}{\partial w_{21}^{[1]}}$$
$$= (\hat{y} - y)\cdot g'\left(z_1^{[2]}\right)\cdot w_{10}^{[2]}\cdot g'\left(z_2^{[1]}\right)\cdot x_2$$

$$\boldsymbol{x} = \begin{bmatrix}x_0 = 1\\ x_1\\ x_2\end{bmatrix} \quad \hat{\boldsymbol{y}}_j = g\left(\sum_{i=0}^n W_{ij}x_i\right)$$

$$\sigma'\left(z_1^{[2]}\right) = \sigma\left(z_1^{[2]}\right)\left(1 - \sigma\left(z_1^{[2]}\right)\right) = \hat{Y}(1 - \hat{Y})$$

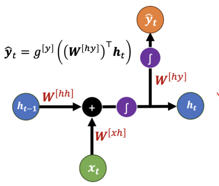
$$\hat{\boldsymbol{y}} = g^{[L]}\left(W^{[L]^T}\dots g^{[L-1]}\left(W^{[L-1]^T}\dots g^{[l]}\left(W^{[l]^T}\dots g^{[2]}\left(W^{[2]^T}g^{[1]}(W^{[1]^T}\boldsymbol{x})\right)\right)\right)\right) = \begin{bmatrix}\hat{y}_1\\ \dots\\ \hat{y}_C\end{bmatrix}$$

- Without non-linear activations $\Rightarrow$ collapses to linear ($\hat{\mathbf{y}} = A\mathbf{x}$)
- Reorder data points help model converge faster
- Reorder may direct the model converge to a different weight (model)

**CNN**
- **Convolution:** Multiply the sliding input window with **Kernel/ Filter** then sum (**Not** matrix multiplication)
  **Trainable Parameter Size:** $(W_{\text{in}} \times H_{\text{in}} \times C_{\text{in}} + 1) \times C_{\text{out}}$
  **Feature Map Size:** $\left\lfloor\frac{H-K+2P}{S}\right\rfloor + 1$
- **Max Pooling** Reduces dim, no($\mathbf{0}$) parameters to be learned
- **Fully Connected Layer**
  (input units $\times$ output units) + output units
- **Batch Normalization Layer:** # of features $\times$ 2
- **Dropout** applied later to force model pay attention to all of the abstract features of the example.

**Recurrent NN:** Handling sequential data (e.g. sentence)
- One-to-many: $T_x = 1, T_y > 1$
- Many-to-one: $T_x > 1, T_y = 1$
- Many-to-many: $T_x = T_y$ or $T_x \neq T_y$

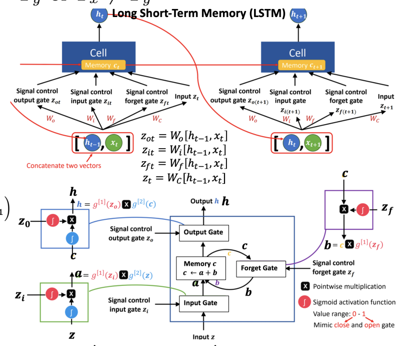
$\hat{\boldsymbol{y}}_t = g^{[y]}\left(\left(\boldsymbol{W}^{[hy]}\right)^{\mathsf{T}}\boldsymbol{h}_t\right)$



$\boldsymbol{h}_t = g^{[h]}\left(\left(\boldsymbol{W}^{[xh]}\right)^{\mathsf{T}}\boldsymbol{x}_t + \left(\boldsymbol{W}^{[hh]}\right)^{\mathsf{T}}\boldsymbol{h}_{t-1}\right)$

**Same weights are applied at each time step**

$\boldsymbol{h} = g^{[1]}(\mathbf{z}_0)\times g^{[2]}(\mathbf{c})$
$\boldsymbol{a} = g^{[1]}(\mathbf{z}_i)\times g^{[2]}(\boldsymbol{z})$
$\boldsymbol{b} = \boldsymbol{c}\times g^{[1]}(\boldsymbol{z_f})$



Long Short-Term Memory (LSTM)
$h_t$
$h_{t+1}$
Cell
Memory $c_t$
Cell
Memory $c_{t+1}$
Signal control output gate $z_{ot}$
Signal control input gate $z_{it}$
Signal control forget gate $z_{ft}$
Input $z_t$
Signal control output gate $z_{o(t+1)}$
Signal control input gate $z_{i(t+1)}$
Signal control forget gate $z_{f(t+1)}$
Input $z_{t+1}$
$W_o$
$W_i$
$W_f$
$W_c$
$[h_{t-1}, x_t]$
$[h_t, x_{t+1}]$
Concatenate two vectors
$z_{ot} = W_o[h_{t-1}, x_t]$
$z_{it} = W_i[h_{t-1}, x_t]$
$z_{ft} = W_f[h_{t-1}, x_t]$
$z_t = W_C[h_{t-1}, x_t]$
$h$
$h = g^{[1]}(z_o)$ ⊠ $g^{[2]}(c)$
$z_0$
$c$
$a = g^{[1]}(z_i)$ ⊠ $g^{[2]}(z)$
$z_i$
$z$
Output $h$ $h$
Signal control output gate $z_o$
Output Gate
Memory $c$
$c \leftarrow a + b$
Forget Gate
Input Gate
Signal control input gate $z_i$
Input $z$
$c$
$a$
$b$
$z_f$
$b = c$ ⊠ $g^{[1]}(z_f)$
Signal control forget gate $z_f$
Pointwise multiplication
Sigmoid activation function
Value range: 0 - 1
Mimic close and open gate

**Self-Attention** $K^TQ = A \rightarrow A' \rightarrow H = VA'$
- **Query:** information we want to focus on.
- **Key:** info. associated with each input can be compared to query.
- **Value:** actual info. retrieved based on attention scores.

**Transformer**: Encoder-Decoder Attention
- **Query:** Generated based on previous decoder block's output
- **Key, Value:** Generated based on encoder's output
- Decoder utilize rich contextual info. provided by the encoder

**Hyperparameter Tuning** $\Rightarrow$ Finding the best model
- Learning rate
- Epochs
- #layers [↑]
- #nodes/layer [↑]
- Activation func
- Architecture
- $C$
- Kernel
- Gamma
- ↑ hidden layers is more efficient than ↑ nodes

**Unsupervised Learning:** Given a set of data points $\{x^{(1)}, ..., x^{(m)}\}$, learn patterns in the data (No label).
- **Clustering:** identify clusters in the data
- **Dimensionality reduction:** find a lower-dimensional representation of the data

<u>**K-Means**</u>
- **Randomly** initialize $K$ centroids: $\mu_1, ..., \mu_K$
  $\Rightarrow$ Multiple restarts of K-means give different solutions
- Repeat until convergence:
  - For $i = 1, ..., m$ : $c^{(i)} \leftarrow$ index of cluster centroid $(u_1 - u_v)$ closest to $x^{(i)}$
  - For $k = 1, ..., K$ : $\mu_k \leftarrow$centroid of data points $x^{(i)}$ assigned to cluster $k$
- No more change: converged!

It can be shown that each step in the K-Means algorithm never increases a certain loss function (the "distortion").

$$J\left(c^{(1)}, c^{(2)}, \cdots, c^{(m)}, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \cdots, \boldsymbol{\mu}_K\right) = \frac{1}{m}\sum_{i=1}^m\left\|\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_{c^{(i)}}\right\|^2$$

**Elbow Method:** Pick $K$ at elbow point. $\Rightarrow$ Heuristic method, data may not have an elbow or have multiple elbows.

<u>**K-Medoids**</u> Pick the **data points** that are closest to the centroids, and use them as the centroids.

<u>**Hierarchical Clustering**</u> Cluster Number: $N, N-1, \dots 1$
- Every data point is a cluster
- **Loop** (until all points are in one cluster): Find a pair of cluster that is "nearest", merge them together

**High** space and time complexity: impractical for large datasets.

<u>**Dimensionality Reduction**</u> SVD and PCA

<u>**Curse of dimensionality:**</u> # of samples to learn a hypothesis class increases **exponentially** with the # of features.

**Idea:** Change the basis of the vector to remove dependence between components (dimensions)

**Singular Value Decomposition:** For any $n\times m$ real-valued matrix $\boldsymbol{X}$, there exists a factorization $\boldsymbol{X} = \boldsymbol{U\Sigma V}^T$ called SVD, such that
- $\boldsymbol{U}$ is $n\times m$ and has $m$ **orthonormal** col. [**New basis**]
- $\boldsymbol{\Sigma}$ is $m\times m$ and is **diagonal** with $\sigma_j \geq 0$ [**Basis Importance**]
- $\boldsymbol{V}$ is $m\times m$ and has $m$ **orth.** col. and rows [**Combiner**]

**SVD Dim Reduction** Set all singular values except first $r$ to 0. Thus $(n\times r)(r\times r)(r\times m) = (n\times m)$ [encoder-decoder structure]
- **Data Matrix:** $X = U\Sigma V^T \approx \widetilde{U}\widetilde{\Sigma}\widetilde{V}^T$
- **Reduced data:** $Z := \widetilde{U}^TX$
- **Reconstructed data:** $\widetilde{X} := \widetilde{U}Z$

$\widetilde{X} = \widetilde{U}\boldsymbol{Z} = \widetilde{U}\widetilde{U}^T\boldsymbol{X} = \widetilde{U}\widetilde{U}^T\boldsymbol{U\Sigma V}^T \approx \widetilde{U}\widetilde{U}^T\widetilde{U}\widetilde{\Sigma}\widetilde{V}^T = \widetilde{U}\widetilde{\Sigma}\widetilde{V}^T \approx \boldsymbol{U\Sigma V}^T = \boldsymbol{X}$

**Principal Component Analysis** By setting $r$, we only care about the $r$ RVs with largest variance. Capture components that **maximize** the statistical variations of the data.
- Create the covariance matrix of the data: $Cov(\boldsymbol{X}) = \frac{1}{m}\widehat{\boldsymbol{X}}\widehat{\boldsymbol{X}}^T$
- Compute SVD on $Cov(\boldsymbol{X})$ to obtain the $U$ matrix (new basis)
- Reduce $r$ components to obtain $\widetilde{\boldsymbol{U}}$

Note that X's singular values^2 are related to the singular values (variances) in Cov:

$$\boldsymbol{XX}^T = \boldsymbol{U\Sigma V}^T\left(\boldsymbol{U\Sigma V}^T\right)^T = \boldsymbol{U\Sigma V}^T\boldsymbol{V\Sigma U}^T = \boldsymbol{U\Sigma\Sigma U}^T = \boldsymbol{U}\begin{bmatrix}\sigma_1^2 & & \\ & \sigma_2^2 & \\ & & \ddots\ \sigma_m^2\end{bmatrix}\boldsymbol{U}^T$$

**Task:** Retain at least 99% of variance in the data:
- Choose minimum $r$, such that $\frac{\sum_{i=1}^r\sigma_i^2}{\sum_{i=1}^m\sigma_i^2} \geq 0.99$
- Closeness of original and reconstructed data points $\frac{\sum_{i=1}^m\|\widehat{x}^{(i)} - \widetilde{x}_i\|^2}{\sum_{i=1}^m\|\widehat{x}^{(i)}\|^2} \leq 0.01$

<u>**Appendix**</u>

**Magic Entropy Number**
- $I(\frac{1}{2}, \frac{1}{2}) = 1$
- $I(\frac{1}{4}, \frac{3}{4}) = 0.811$
- $I(\frac{1}{5}, \frac{4}{5}) = 0.722$
- $I(\frac{1}{6}, \frac{5}{6}) = 0.650$
- $I(\frac{1}{7}, \frac{6}{7}) = 0.592$
- $I(\frac{3}{7}, \frac{4}{7}) = 0.985$
- $I(\frac{3}{10}, \frac{7}{10}) = 0.881$
- $I(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}) = 1.585$
- $I(\frac{1}{6}, \frac{1}{3}, \frac{1}{2}) = 1.459$
- Math: $\log_2 N = \lg N/\lg 2$
- $I(1, 0) = 0$
- $I(\frac{1}{3}, \frac{2}{3}) = 0.918$
- $I(\frac{2}{5}, \frac{3}{5}) = 0.971$
- $I(\frac{2}{7}, \frac{5}{7}) = 0.863$
- $I(\frac{3}{8}, \frac{5}{8}) = 0.954$
- $I(\frac{2}{12}, \frac{5}{12}, \frac{5}{12}) = 1.483$
- $I(\frac{1}{7}, \frac{2}{7}, \frac{4}{7}) = 1.379$