Chain rule: $p(x,y) = p(x)p(y|x) = p(y)p(x|y)$

Extended chain rule: $p(x,y,z,w) = p(x)p(y|x)p(z|x,y)p(w|x,y,z)$

Conditional probability as a distribution: $\sum_x p(X|Y=y) = 1$

Independence: If $p(X|Y) = p(X)$, then X and Y are independent for all values of X

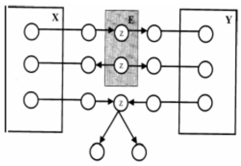
Conditional independence: $X \perp Y|Z \leftrightarrow p(X,Y|Z) = p(X|Z)p(Y|Z)$

Factored joint distribution of a Bayes net: $p(x_1, \dots, x_n) = \prod p(x_i|parents(x_i))$

Local Markov property: $X_i \perp NonDescendants(X_i)|Parents(X_i)$

Trail blocking: Evidence in **cascade**, evidence in **common cause**, or no evidence in **V structure** (including descendants)

D-separation: when all trails between 2 sets of nodes are blocked given evidence set. D-separation implies conditional independence



Cascade X -> **E** -> Y
Common cause X <- **E** -> Y
V structure X -> ~~E(and descendents)~~ <- Y

Bayes net stores joint probabilities of nodes and immediate parents. **Sum product variable elimination** exploits conditional independences in a Bayes net to compute joint probabilities without a full joint distribution table.

Loss function: A means to measure accuracy of a classifier function. Must be differentiable so as to apply gradient descent. Loss over an epoch/minibatch is taken as the average loss over the each of the individual predictions

Softmax loss/Cross entropy: Given output values $S_1, S_2, \dots, S_k \in \mathbb{R}$, take probabilities as $p_n = \frac{e^{S_n}}{\sum e^{S_i}}$. Loss is computed as $\frac{1}{N}\sum_{i=1}^{N}(-\log p_i)$. Useful for probabilities and classification tasks

Mean Square Error (MSE) loss: Given output values $S_1, S_2, \dots, S_k \in \mathbb{R}$, take loss as $\frac{1}{N}\sum_{i=1}^{N}(S_i - y_i)^2$. Useful for predicting numerical values/continuous variables

Neuron input: For m inputs $x_1, \dots, x_m$ into a neuron, each neuron computes $z = \sum_{i=1}^{m} w_i x_i + b$, where w_i is the individual weight learned and b is the bias. Each neuron thus has m+1 trainable parameters.

Activation functions: Sigmoid $f(x) = \frac{1}{1+e^{-x}}$, tanh $f(x) = \tanh(x)$, ReLu (rectified linear unit) $f(x) = \max(0,x)$ – nonlinear functions applied to values computed by neurons to improve usefulness of the model

ANN Hyperparameters: Hidden layers, nodes in each layer, activation function, output nodes, initial weights & bias, learning rate. optimization algos, batch size, epochs

CNN: Special type of ANN that extracts features from images, and can be highly generalizable for different tasks

Filter: $(N \times N \times D\ )$ set of values that slides over the image spatially and computes dot products. N is a hyperparameter, D is the depth of the input volume. Each filter can only produce an activation map of depth = 1

Activation maps: Each filter tries to interpret certain features, and activates in areas where that feature is found

Stride: How many rows/columns the filter is shifted by when convolving over the image

Padding: Adding zeros uniformly to the image on all sides to avoid fractional outputs for activation maps

Convolutional layer summary: Given input volume $(W_1 \times H_1 \times D_1)$, with K filters, filter size F (assume square), stride S, and padding P, produces an output volume ($W_2 = \frac{W_1 - F + 2P}{S} + 1$, $H_2 = \frac{H_1 - F + 2P}{S} + 1, D_2 = K$). Trainable parameters = $K(F \times F \times D_1 + 1)$

Max pooling: To make activation map smaller and more tractable, we have a filter that simply takes the max value of all those under that filter's view. No training parameters needed

Dropout: Randomly drop out neurons during training to force every neuron to learn something useful.

Batch normalization: Normalise a layer's input by subtracting the mini-batch mean and dividing it by standard deviation, to ensure that those inputs have mean = 0 and s.d. = 1. Scale and shift the normalized value
$y = \gamma \cdot \hat{x} + \beta$

Dependency parsing: Relations between words can be represented using a dependency tree

Named entity recognition: Label each noun with the concepts they represent (entity, org., person)

Coreference resolution: Find all expressions that refer to the same entity

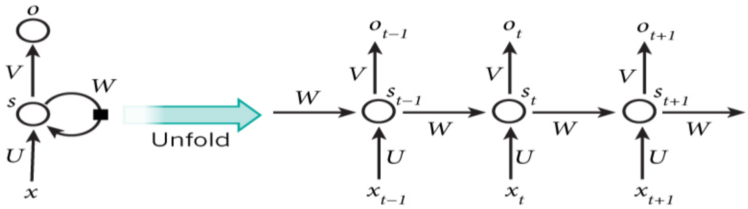
Co-occurrence matrix: N x N, symmetric matrix storing frequencies of words occurring within a certain window of each other within a corpus of text.

X=USV^T: SVD process applied on co-occurrence matrix

Continuous bag of words: Predict word given context. Training data = context, prediction = word

Skip-gram: Predict context given word. Training data = word, prediction = context. Often performs better due to additional training data w Naïve Bayes assumption, better for rare words

Negative sampling: Reduce complexity by updating small subset of weights



Recurrent NN:
$s_t = f(Ux_t + Ws_{t-1})$ where f is an activation function, $o_t = g(Vs_t)$ where g is output activation

Search problem: Comprises *state space, operators, costs, objective*

Search	Informed?	Metric	Queue replace?	Explore replace?
BFS	No	Breadth, node ordering	No	No
DFS	No	Depth, node ordering	No	No
BFS opt	No	Path cost $g(n)$	Yes	No
Best-first	Yes	Heuristic $h(n)$	Yes	No
A*	Yes	Path and heuristic $g(n) + h(n)$	Yes	Yes

Admissible heuristic: ∀ states $n, h(n) \leq c(s, g)$, i.e. heuristic not exceed min cost

Consistent heuristic: ∀ states n, p, $h(n) \leq c(n, p) + h(p)$ (triangle inequality)

MDP terms: *Decision epoch*-finite/infinite horizon, *absorbing state*-transition outside is impossible, *markov assumption*-**past⊥future|present**, *transition function* must be normalized

EU defined for every state and action: $Q(s,a) = \sum_{s`} P(s`|s,a)[R(s,a,s`) + V(s`)]$
MEU defined for every state: $V = \max_a Q(s,a)$

Policy can be *stationary/non-stationary*, *deterministic/stochastic*

Value iteration: Set $V^0(s) = 0$ ∀ states s, each time step t, update ∀ s:
$Q^t(s,a) = \sum_{s`} P(s`|s,a)\,[R(s,a,s`) + \gamma V^{t-1}(s`)]$, where discount factor $\gamma < 1$
$V^t(s) = \max_a Q^t(s,a)$
Convergence when $\max_s |V^{t+1}(s) - V^t(s)| < \epsilon$
Optimal policy guarantee: $\pi^*(s) = \arg\max_a Q^{(t+1)}(s,a)$

Multi-armed bandit: Given tries t, action count $k_t(a)$, action rewards $R_1(a), \dots, R_{k_t}(a)$,
expected payoff $Q_t(a) = \frac{R_1(a)+\cdots+R_{k_t}(a)}{k_t(a)}$

Greedy-pick $\max_a Q_t(a)$	**ϵ-greedy**-$\max_a Q_t(a)$ (1-ϵ) of the time else random
Softmax-$P(a) = \frac{e^{Q_t(a)}}{\sum_b e^{Q_t(b)}}$	**Upper confidence bound**-$\max_a(Q_t(a) + c\sqrt{\frac{\log(t)}{k_t(a)}})$, with constant c

Q-learning: World is a set of *discrete and finite* states and actions. An **experience** is (s_t, a_t, r_t, s_{t+1}) where s_t is **state** at time step t, a_t is **action**, r_t is **reward** and s_{t+1} is **new state**. An **episode** is a sequence of experiences from start to a **terminal state**

Tabular Q-learning: Store Q(s,a) for each state and action in 2D array (rows are states and columns are actions). On new experience, compute $Q^{new}(s_t, a_t) = Q(s_t, a_t) + \alpha\left[r_t + \gamma \cdot \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right]$, where α is step size/learning rate, and γ is discount factor. Repeat for a fixed number of experiences, or upon some convergence condition (among $Q(s,a)$ or $V(s)$)

Feature-based learning: Instead of states, learn specific features. Represent Q value according to weighted combination of these features ⇒ $Q(s,a) = w_1 \cdot f_1(s,a) + \cdots + w_n \cdot f_n(s,a)$, where w_i are weights and f_i are features. Saves on memory as weights can be stored as a single vector independent of state, learning becomes more generalizable.