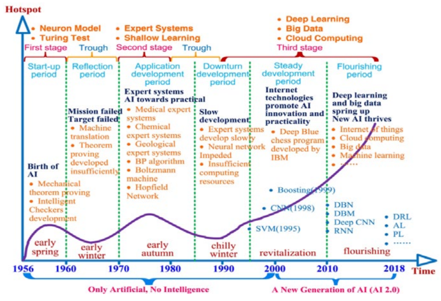
CSCI3230 / ESTR3108 – Fundamentals of Artificial Intelligence

Artificial Intelligence
Computer systems capable of performing complex tasks that mimic human intelligence, such as reasoning, making decisions, or solving problems.
Machine Learning
Algorithms whose performance improve as they are exposed to more data over time.
Deep Learning
A specific group of machine learning models in form of multilayered neural networks, that can be learnt from massive data.



AI in Healthcare: Applied to clinical practice such as disease diagnosis, treatment protocol development, drug design, bioinformatics, personalized medicine, patient monitoring.
AI in Autonomous Driving: Help autonomous vehicles understand the surroundings and perform path planning.
AI in Robotics: Robot learning studies techniques for robot to acquire novel skills or adapt to environment. Surgical robots, assembling robots.
AI in Science: Prediction of chemical structure: AlphaFold2, Financial growth predictions, Weather forecasts
AI in Languages: NLP, ChatGPT; **AI in Games:** AlphaGo, NPCs
How to Learn from Data
(i) Learning algorithm; (ii) Model; (iii) Data
Data Split
Training set: sample data used in training phase to fit parameters.
Validation set: tunes hyperparameters of the model.
Test set: provides unbiased evaluation of the final model.
Cross Validation: (i) shuffle the dataset randomly and split the dataset into k groups. (ii) for each unique group: take the group as holdout set, take the remaining groups as training dataset. (iii) summarize the skill of the model using all model evaluation scores.
Performance: Compare with previous models, Groundtruth measures.

Accuracy: $\frac{TP+TN}{TP+FP+TN+FN}$, Precision: $\frac{TP}{TP+FP}$, Recall (sensitivity): $\frac{TP}{TP+FN}$
Confusion Matrix: visualizing the predictions performed by model.
Supervised Learning: Learns a function that maps an input to an output based on example input-output pairs. (i) Labeled data, (ii) Predict outcome/future, (iii) Direct feedback.
Unsupervised Learning: Learns patterns from unlabeled data. (i) Unlabeled data, (ii) Find hidden structure, (iii) No feedback.
Reinforcement Learning: Learns to make decisions in an environment to maximize the cumulative award. (i) Learn series, (ii) Reward system, (iii) Decision process.
Regression: A statistical process for estimating the relation between dependent variables (outcoms) and independent variables (features).
A function is **<u>linear</u>** iff $f(u+v) = f(u) + f(v), f(cu) = cf(u)$.
Goal: utilize the training data to find the general mapping $\hat{f}: X \to Y$ which is as close to f as possible.
Multivariate Linear Function:

$\hat{y} = \hat{f}_{\Theta,\theta_0}(X) = X^\top\Theta + \theta_0$, where $\Theta = (\theta_1, \dots, \theta_n)^\top, X = (x_1, \dots, x_n)^\top$.
m samples: $\hat{Y} = \mathbf{X}\Theta,\ \mathbf{X} = \left(\mathbf{1}, X^{(1)^\top}, \dots, X^{(m)^\top}\right)^\top, \hat{Y} = (\hat{y}^{(1)}, \dots, \hat{y}^{(m)})^\top$.
Objective function: Residual Sum of Squares (RSS)

$J(\Theta) = \sum_{i=1}^m \left(\hat{f}_\Theta(X^{(i)}) - y^{(i)}\right)^2 = \left\|\hat{f}_\Theta(\mathbf{X}) - Y\right\|_2^2$, which is convex.

$\frac{\partial J(\Theta)}{\partial \Theta} = 2\mathbf{X}^\top\mathbf{X}\Theta - \mathbf{X}^\top Y - \mathbf{X}^\top Y = 0, \frac{\partial^2 J(\Theta)}{\partial \Theta^2} = 2\mathbf{X}^\top\mathbf{X} > 0, \Theta^* = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top Y.$
Shrinkage Method: Aims to shrink the parameters by imposing a penalty on coefficients
Training data are always noisy $y = f(X) + \epsilon, \mathrm{E}(\epsilon) = 0, \mathrm{Var}(\epsilon) = \sigma^2$.
For every fixed X with label y, the expected prediction error at X is
$\mathrm{EPE}(X) = \mathrm{E}\left(\left(y - \hat{f}(X)\right)^2\right) = \mathrm{Bias}\left(\hat{f}(\mathrm{X})\right)^2 + \mathrm{Var}\left(\hat{f}(X)\right) + \sigma^2$,
$\mathrm{Bias}\left(\hat{f}(X)\right) = E\left(\hat{f}(X)\right) - f(X), \mathrm{Var}\left(\hat{f}(X)\right) = \mathrm{E}(\left(\hat{f}(X) - \mathrm{E}(\hat{f}(X))\right)^2)$.
Low variance will cause high bias (underfitting, low model complexity), while low bias will result in high variance (overfitting, high model complexity).
Problem of Ordinary Least Squares: overfitting
<u>Lasso</u> regression: L1 norm $\lambda\|\Theta\|_1$, promotes the sparsity within the model, provides automatic feature selection. Suitable for high-dimensional datasets.
<u>Ridge</u> regression: squared-L2 norm $\lambda\|\Theta\|_2^2$, shrinks the magnitude, avoid taking large coefficients that result in high variance when correlated. As λ increases, high-value coefficients shrink at a greater rate than low-value coefficients.
As λ increases, the bias increases, the variance decreases. Suitable for highly correlated predictors.
Classification: A process of categorization such that ideas and objects are recognized, differentiated and understood.
Goal: utilize the training data and recognize from a set of categories that it belongs to.
Regression v.s. Classification: Output is discrete (classification), continuous (regression).
Decision Boundary: a union of curves or surfaces that partition the feature space.
Logistic Regression: models probabilities for classification problems with two outcomes. Hard to use linear regression $(-\infty, +\infty)$ due to sensitivity to outliers.

(i) $\mathrm{odds}(p) = \frac{p}{1-p} \in (0, +\infty)$ (ii) $\mathrm{logit}(p) = \ln \mathrm{odds}(p) = \ln \frac{p}{1-p}$.

Apply **sigmoid**, $\mathrm{logit}^{-1} = \frac{1}{1+e^{-z}}$ to the values z to probability in $(0,1)$.

$P(\hat{y} = 1|X) = \sigma(X^\top\Theta) = \mathrm{logit}^{-1}(X^\top\Theta) = \frac{1}{1+e^{-(X^\top\Theta)}}$. Hyperplane: $z = X^\top\Theta$
Advantages: exactly outputs the probability of $y = 1$ conditioned on the input X; changes significantly around $X^\top\Theta = 0$, which is a good characteristic to distinguish samples with different labels; robust to outliers.
The probability that the label $y^{(i)}$ is correctly predicted is
$p_i = P(\hat{y}^{(i)} = 1|X^{(i)})^{y^{(i)}} \left(1 - P(\hat{y}^{(i)} = 1|X^{(i)})\right)^{1-y^{(i)}}$
Joint probability => maximize the likelihood $p(\Theta) = \Pi_{i=1}^m p_i$
Solve minimization of $-\ln p(\Theta)$ by gradient descent.
Gradient descent: $\Theta \leftarrow \Theta - \alpha\nabla f(\Theta)$, where α is the learning rate.
Small α results in slow learning process; Large α may lead to divergence.
Cross-entropy error $-y \ln P(y) - (1-y)\ln(1 - P(y))$.

$$\text{Logistic Regression: } \nabla f(\Theta) = \sum_{i=1}^m \left(\frac{1}{1+e^{-X^{(i)\top}\Theta}} - y^{(i)}\right) X^{(i)}.$$

Support Vector Machines: Find a separating hyperplane that has the maximum distance to the closest point. More noise tolerance and more robust.

Let one such hyperplane be $\mathbf{w}^\top\mathbf{x} + b = 0$, then the distance between $\mathbf{x}_i$ and the hyperplane is $\gamma = \frac{|\mathbf{w}^\top\mathbf{x}+b|}{\|\mathbf{w}\|}$.
Objective:

$$\max \frac{2}{\|\mathbf{w}\|} \Rightarrow \min \frac{1}{2}\|\mathbf{w}\| \Rightarrow \min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2$$

$$\mathrm{s.t.}\ 1 - y_i(\mathbf{w}^\top\mathbf{x}_i + b) \le 0.$$

Primal:

$$\min_{\mathbf{w},b} \max_{\boldsymbol{\alpha}} \frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^m \alpha_i\left(1 - y_i(\mathbf{w}^\top\mathbf{x}_i + b)\right)$$

Dual:

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^m \alpha_i - \frac{1}{2}\sum_{i=1}^m \sum_{i=1}^m \alpha_i\alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j$$

$$\text{such that } \sum_{i=1}^m \alpha_i y_i = 0, \alpha_i \ge 0, i = 1,2, \dots, m.$$

If $y_i(\mathbf{w}^\top\mathbf{x}_i + b) - 1 > 0$, sample point is farther than the support vectors, α_i must be 0. If $y_i(\mathbf{w}^\top\mathbf{x}_i + b) - 1 = 0$, sample point is a support vectors, $\alpha_i \ge 0$.

$$\mathbf{w}^* = \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i, b^* = \frac{1}{y_s} - \mathbf{w}^\top\mathbf{x}_s.$$

Kernel SVM: $f(\mathbf{x}) = \mathbf{w}^\top\phi(\mathbf{x}) + b$,
Kernel function $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^\top\phi(\mathbf{x}_j)$
Optimal hyperplane:

$$f(\mathbf{x}) = \sum_{i=1}^m \alpha_i y_i \kappa(\mathbf{x}, \mathbf{x}_i) + b.$$

Polynomial Kernel $\kappa(\mathbf{x}_i, \mathbf{x}_j) = (x_i^\top x_j)^d$
Sigmoid Kernel $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \mathrm{tahn}(\beta\mathbf{x}_i^\top\mathbf{x}_j + \theta)$
Gaussian Kernel $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma^2)$
Soft Margin SVM: Give up some noisy samples

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^m l_{0/1}(y_i(\mathbf{w}^\top\mathbf{x}_i + b) - 1)$$

Use hinge loss to replace $l_{0/1}$.
Clustering
Goal: group a set of objects such that objects in the same group (cluster) are more similar to each other than tot those in other groups. High intra-cluster (compactness) similarity, low inter-cluster similarity (isolation).
Similarity Measure
Euclidean distance: $\mathrm{dist}(\mathbf{a}, \mathbf{b}) = \sqrt{\sum_{i=1}^n (a_i - b_i)^2}$

Manhattan distance: $\mathrm{dist}(\mathbf{a}, \mathbf{b}) = \sqrt{\sum_{i=1}^n |a_i - b_i|}$

Chebyshev distance: $\mathrm{dist}(\mathbf{a}, \mathbf{b}) = \max_i |a_i - b_i|$

Minkowski distance: $\mathrm{dist}(\mathbf{a}, \mathbf{b}) = \left(\sum_{i=1}^n |a_i - b_i|^p\right)^{\frac{1}{p}}$

Cosine similarity: $\mathrm{dist}(\mathbf{a}, \mathbf{b}) = \frac{a \cdot b}{\|a\| \cdot \|b\|}$

Partitional Clustering Algorithms
Centroid-based: Data points close to the central vectors are assigned to the respective clusters, K-Means
Density-based: Higher density areas belong to one cluster. Objects in sparse areas are considered to be noise or border points, DBSCAN
Gaussian Mixture Models
Spectral Clustering
Hierarchical Clustering Algorithms: a hierarchy of clusters.
Agglomerative (bottom up), Divisive (top-down)
K-Means
Choose K (random) data points to be the initial centroids.
  Assign each data point to the closest centroid.
  Recompute the centroids using the current cluster memberships.
Repeat the assignments and updates if the convergence criterion is not met. Stopping criteria: (i) No change of centroids; (ii) No reassignment of data points to clusters; (iii) Sum of squared error is satisfactory.
Let M be the number of data points, K be the number of clusters, I be the number of iterations, N be the number of attributes.
Time complexity: $O(IKMN)$. Space complexity: $O((M+K)N)$.
Advantage: (i) easy to understand and implement, (ii) linear complexity when I, K are small, (iii) Scalable to large data sets, (iv) Guarantee convergence, (v) Visualizable clusters, (vi) tigher clusters than hierarchical clustering
Disadvantage: (i) Manual hyperparameter K, (ii) Empty cluster, (iii) Sensitive to outliers, (iv) Non-globular shapes.
DBSCAN (Density-based spatial clustering of applications with noise**)**
Clusters are dense regions in the data space, clusters as maximal sets of density-connected points. Noise points lie in regions of low density.
ϵ: maximum radius of the neighborhood; MinPts: minimum number of points required for core point within ϵ.
ϵ-neighbour: data points within a radius of ϵ from a data point (including itself)
Core point: has more than or equal to MinPts within ϵ.
Border point: has fewer than MinPts within ϵ, but is in the neighborhood of a core point.
Noise point (outlier): any point that is neither a core nor a border.
An object q is *directly density-reachable* from object p if p is a core object and q is in p's ϵ-neighborhood. Asymmetric relationship.
A point p is *indirectly density-reachable* (density-connected) from q if there is p_1 directly density-reachable from p_2, p_2 directly density-reachable from p_3, ..., p_2 directly density-reachable from q.
Algorithm $O(n \log n)$ average, $O(n^2)$ worst
Label data points into core, border and noise.
Eliminate noise points.
For every core point p that has not been assigned to a cluster, create a new cluster with the point p and all the points that are density-connected to p.
Assign border points to the cluster of the closest core point.
Determine the "knee" of the curve. The idea is that, for points in a cluster, their k-th nearest neighbors are at roughly the same distance. Plot sorted distance of every point to its k-th neighbor.
Advantage: discover clusters of arbitrary shapes and sizes, number of clusters is determined automatically, can separate clusters from noise, robust to outliers.
Disadvantage: sensitive to outliers – hard to determine optimal parameters, has problem identifying clusters of varying densities, may not work well with high-dimensional data, only a graph of density-connected points.

Agglomerative Clustering:
Let each data point be a cluster.
Repeat until a single cluster remains: merge two closest clusters and update the proximity matrix.
Single Linkage (Min distance)
Cluster distance = distance of two closest elements in each cluster.
Complete Linkage (Max distance)
Cluster distance = distance of two farthest elements in each cluster.
Average Linkage (Group average distance)
Cluster distance = average distance of each pair in the two clusters.
Centroid Linkage (Distance between centroids)
Cluster distance = distance of two cluster centroids.

Method	Linkage criteria	Time complexity	Comment
Single Link	Smallest distance between two points	$O(n^2)$	Chaining effect
Complete Link	Longest Distance between two points	$O(n^2 logn)$	Sensitive to noise
Average Link	Average Distance of all the data points	$O(n^2 logn)$	Best choice for most applications
Centroid	Distance between Centroid	$O(n^2 logn)$	Inversion can occur

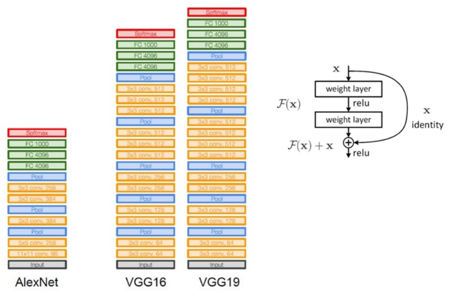
Neural Network
Neuron as a logistic unit, connects input to neuron with wires.
Activation Functions

Sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$.
Advantages: continuous, limited output, easy to optimize, can be used as output. Disadvantages: vanishing gradient at $x \to \pm\infty$, high computational complexity, output mean is not centered at 0, compromising convergence speed.

Tanh: $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} = 2\sigma(x) - 1$.
Advantage: output $\in [-1,1]$, converging faster than sigmoid.
Disadvantage: vanishing gradient at $x \to \pm\infty$, high computational complexity.
ReLU (Rectified Linear Unit): $f(x) = \max(0, x)$.
Advantage: Gradient is not saturated and the convergence speed is fast, no gradient disappearance, no exponential calculation, fast computation and low complexity. Disadvantage: neuron death $x < 0$. Output mean is greater than zero, compromising convergence.
$\Theta_{mn}^{(j)} =$ weight controlling function mapping from unit m in layer j to unit n in layer $j+1$.
Training an NN: (i) Network architecture; (ii) Parameters: e.g. initialization; (iii) Backpropagation algorithms, optimizers.
Softmax function: Normalizes the output of a network η to a probability distribution ϕ over predicted output classes.

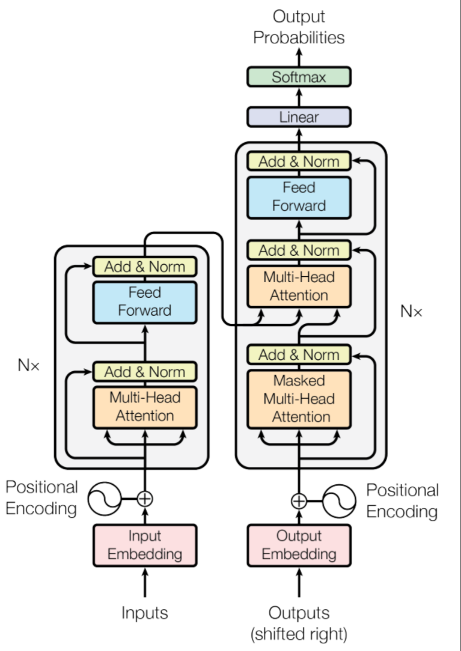
$$\phi_i = \frac{e^{\eta_i}}{\sum_{i=1}^k e^{\eta_i}}.$$

Loss Function: maps an event or values of one or more variables onto a real number intuitively representing some "cost" associated with the event.
Mean Square Error: average squared difference between the estimated values and the actual values. $\ell_{\mathrm{MSE}} = (y_i - \hat{y}_i)^2$.
Cross Entropy Error: $\ell_{\mathrm{CE}} = -y_i \log \hat{y}_i$.
Learning rate determines step size at each iteration while moving toward minimum of a loss function. Training loss can significantly decrease after learning rate decay.
Linear decay $\alpha_t = \alpha_0(1 - t/T)$.
Cosine decay $\alpha_t = 1/2 \cdot \alpha_0(1 + \cos t\pi/T)$.
Inverse sqrt decay $\alpha_t = \alpha_0/\sqrt{t}$.
Linearly increasing learning rate
Convolutional Layers:
Convolutional layers are responsible for learning and extracting features from the input data. Each layer consists of multiple filters.
Kernel: a filter that convolves with the image by sliding over the image spatially and computing the dot products, and produces feature maps (with specific pattern or features in input data, such as edges, textures, shapes).
Channel (D): the output dimension, each feature map corresponds to a specific channel representing the activation of the corresponding filter.
Convolutional layer receives a 3D volume of size $W_1 \times H_1 \times D_1$ and requires four hyper-parameters:
number of filters K
filter spatial extent F
the convolution stride S
the amount of zero padding P
Produces a volume of size $W_2 \times H_2 \times D_2$ with:
$W_2 = (W_1 + 2P - F)/S + 1, H_2 = (H_1 + 2P - F)/S + 1, D_2 = K$
It introduces $F \times F \times D_1$ weights in each filter, in total, for a convolutional layer, it has $(F \times F \times D_1) \times K$ weights and K biases.
In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.
Pooling layer: make representations smaller and more manageable, operate over each activation map independently. Downsamples the volume spatially, independently in each depth slice of the input volume. Needs two hyperparameters, pooling spatial extent F and pooling stride S. Receives a 3D volume of size $W_1 \times H_1 \times D$.
Produces an output of $W_2 \times H_2 \times D$, $W_2 = (W_1 - F)/S + 1, H_2 = (H_1 - F)/S + 1$. Number of parameters = 0.
In practice, a fully connected layer is attached at the end of the CNN model, providing the model with ability to mix signals, connecting every single neuron to the neurons in the next layer. Can be adapted to obtain a probabilistic output.
LeNet: Convolve input, Non-linearity: ReLU, Pooling
AlexNet: 227*227*3 images, 650000 neurons, 60M parameters, 630M connections, final feature layer: 4096 dimensional.
VGG: small convolution filters through the net, 3*3 kernels, paddings, 64-512 channels, 2*2 stride 2 max-pooling, ReLU.
ResNet: An overly deep model have higher training errors (vanishing gradients), add extra layers set as identity. Let $H(x) = F(x) + x$, residual block.

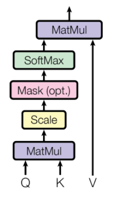


CSCI3230 / ESTR3108 – Fundamentals of Artificial Intelligence

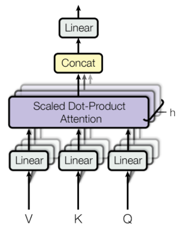
ImageNet dataset for Large Scale Visual Recognition Challenge: deep ResNets can be trained without difficulties, low training error low test error.
CNN Tasks: classification, localization, object detection, segmentation.
Data Augmentation: adding more training data into models, alleviating data scarcity for learning better models, reduces data overfitting and creates variability, resolve class imbalance issues, reducing costs of collecting and labeling data, increases generalization ability. Example: crop, resize, flip, color distort, rotate, add gaussian noise/blur, cutout.
Dropout: keeping a neuron active with only a probability of p. Prevents all neurons from synchronously optimizing their weights. Prevents co-adaptation, a neuron must perform well in itself with different contexts provided by other neurons, improves generalization capability.
Recurrence Neural Network: recurrently processes the input, struggles with catastrophic forgetting, precludes parallelization.
Transformers: Sequential modeling in natural language processing, rely entirely on an attention mechanism based on **query, key, value.**



Scaled Dot-Product Attention



Multi-Head Attention



The input sentence should be first converted into a sequence of **tokens**, $d_{\text{model}} = 512$, embeddings are learned to convert tokens to vectors.
Positional encodings are injected with sine and cosine functions.
Repeated transformer **blocks**: multi-head attention, feed forward, residual addition and normalization.
Scaled Dot-Product Attention: Give each token a KEY, VALUE, QUERY representation. Use its QUERY to find the most relevant token in the sequence based on the KEYS: calculate similarity with dot product, calculate softmax distribution based on dot product values, combine it with VALUE. (performed in parallel)

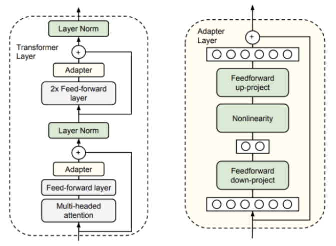
$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V.$$

Dot products are scaled due to large values of dot products and small softmax gradients.
Multi-Head Attention: h different learned projection weights are concatenated and once again projected.

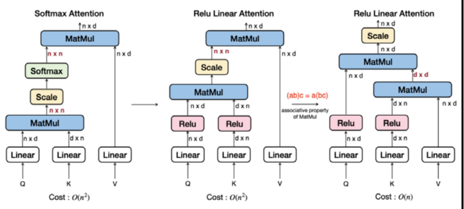
$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1, \dots, \text{head}_\text{h})W^O$$

$$\text{head}_i = \text{Attention}(QW_i^q, KW_i^k, VW_i^V)$$

the total computational cost is similar to that of single-head attention with full dimensionality.
Feed Forward: $\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$, adds non-linearities and mixes channel values.
Vision Transformers: Spatial attention, interpolating images. Each patch will correspond to a token. The initial dimension of token could be the simple concatenation of all the elements in the patch.
Use several linear layers to project the initial tokens to the dimension of Transformer layers, generates patch embedding.
Input dimension of the linear layer: $H \times W \times C/P^2$
Positional embeddings in ViT ensure that spatially closer tokens have higher dot-product similarities.
Modern Transformers
Pretrain a foundation model on large unlabeled data
Finetune with small labeled data on a specific downstream task
Parameter-efficient fine-tuning: Insert an Adapter composed of linear layers and nonlinear activation functions
ViT for Dense Prediction: quadratic computation complexity to input image size, cost of self-attention: $4hwC^2 + 2(hw)^2C$, patch encoding reduces the resolution of the image with a scale of patch size. The information of original resolution is hard to recover for pixel-wise prediction.



Only green modules are trained in the adaptation



EfficientViT: replace the non-linear Softmax in the attention with linear MatMul. The non-linearity is added by ReLU before the attention calculation, which won't affect the associate property. Apply the associate property, and get linear attention with complexity of $O(n)$, good at capturing global context information but bad at capturing local information.
Multimodal Transformer: CLIP (Contrastive Language-Image Pre-Training) models have two transformer branches: a vision branch and a text branch.
CLIP is trained with text supervision by contrastive learning
The training dataset consists of image-text pairs, with each image accompanied by a descriptive text.
The training brings the encoded image embedding closer to its paired text embedding, and pushing it away from other text embeddings (with the cross-entropy loss)
CLIP enables zero-shot classification without any finetuning
Prepare a text description for each class
Do classification by matching the image to be classified with the text of all classes
Zero-shot CLIP is competitive with a fully supervised baseline.
Search Problem:
Single state: complete world knowledge, complete action knowledge.
Multiple state: incomplete world knowledge, incomplete action knowledge.
Contingency problem: not possible to define a complete sequence of actions that constitute a solution in advance because information about the intermediary states is unknown.
Exploration problem: state space and effects of actions are unknown.
Components:
state space: indicate which state that the agent starts in.
successor function: Given a particular state x, $S(x)$ returns the set of states reachable from x by any single action.
start state: the set of all states reachable from the initial state by any sequence of actions.
goal test: the agent can apply the goal test to a single state to determine if it is a goal state.
Path: a path in the state space is simply any sequence of actions leading from one state to another. **Operator set:** the set of possible actions available. **Path cost function:** the sum of the costs of the individual actions along the path.
Search tree is superimposed over the state space. The root is a search node corresponding to the initial state. Leaf nodes correspond to states that do not have successors in the tree, either unexpanded but generated the empty set. **State space graph**: nodes are instances of states, arc represent successors (action results), goal test is a set of goal nodes, each state occur only once.
General tree search: fringe – collection of nodes that are waiting to be expanded, expansion – pick one out of the fringe and expand, strategy: determine the order of the nodes to be expaned.
Evaluation of Search Algorithm:
Completeness: guaranteed for a solution?
Optimality: is the solution optimal?
Time Complexity: time taken to find a solution?
Space Complexity: memory needed?
Uninformed Search: no additional information about states beyond that provided in the problem definition.
Breadth-First Search: all the nodes at depth d in the search tree are expanded, before the nodes at depth $d+1$. Optimality is guaranteed if the path cost is a non-decreasing function of the depth of the node. Time complexity: $O(b^d)$ when the depth is d and every state has b successors.
Uniform-Cost Search: Expands the node with the lowest path cost. Optimal provided that every step cost is greater than or equal to some small positive constant ϵ. Time complexity: $O(b^{C^*/\epsilon})$ where C^* is the optimal path cost.
Depth-First Search: Expands the deepest node in the current fringe of the search tree. Only when the search hits a dead end (a non-goal node with no expansion) does the search go back, and expand nodes at shallower level, and so on. Not complete nor optimal (due to cycles, infinite depth). Time complexity: $O(b^m)$, where m is the maximum depth. Space complexity: $O(bm)$.
Depth-Limited Search: Uses depth-first search, nodes at depth ℓ are treated as if they have no successors. Incomplete $\ell < d$, non-optimal $\ell > d$. Time complexity: $O(b^\ell)$, where b is the number of successors for each state, depth limit is ℓ. Space complexity: $O(b\ell)$.
Iterative-Deepening Search: Gradually increase the depth limit. Complete when branching factor is finite. Optimal when path cost is a non-decreasing function of the depth of the node. Time complexity: $O(b^d)$ when the depth is d and every state has b successors.
Informed Search: Uses problem-specific knowledge beyond the definition of the problem itself.
Heuristic Function: A function that estimates how close a state is to a goal. Admissible if: $0 \le h(n) \le h^*(n)$, where $h^*(n)$ is the true cost to reach a goal. Combining heuristic functions: take max of the both.
Greedy Best-First Search: expands the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly. Not complete: susceptible to false starts. Not guaranteed to be optimal. Time complexity: $O(b^m)$, where m is the maximum depth. Space complexity: $O(b^m)$, retains all nodes.
A* Search: combines the cost to reach the node, and the cost to get from the node to the goal. UCS orders by path cost, Greedy search orders by goal proximity. Complete on locally finite graphs, optimal. Time complexity: $O(b^m)$, where m is the maximum depth. Space complexity: $O(b^m)$, retains all nodes.
Comparison
Greedy expands the nodes whose estimated distance to the goal is the smallest
Uniform-cost expands equally in all "directions"
A$*$ expands mainly toward the goal, but does hedge its bets to