**Types** Supervised Learning: Learn from labeled training data, make predictions about unseen/future data; classification (discrete class labels, binary/multiclass), regression (continuous output value) Reinforcement Learning: The system (aka agent) improves its performance based on interactions

with an environment; trial and error; agent receives feedback (reward) from the environment. Unsupervised Learning: Unlabeled data/data of unknown structure, explores the structure of data. Preprocessing: Feature selection, extraction & scaling, Dimensionality reduction, Sampling. Learning Process: Model selection, Cross-validation, Performance metric, Hyperparameter optimization. Evaluation & Prediction: use the test dataset to estimate how well it performs on unseen data.

**Artificial Neuron** z = dot(w, x); (x0 = 1, w0 = -theta, called the bias unit) phi(z) = 1 ? z >= 0 : -1; **Perceptron Learning Rule** for each training sample x[i], compute output y\_hat[i], update weights w[j] += eta \* (y[i] – y\_hat[i]) \* x[i][j]; the convergence of the perceptron is only guaranteed if the two classes are linearly separable & the learning rate is sufficiently small; for not linearly separable samples, set a max epoch count. **Adaptive Linear Neuron** Improvement on Perceptron algorithm, weights are updated based on a linear activation function: phi(z) = z; use threshold function to make the final prediction; cost function J(w) = (1/2) \* sum((y[i] – phi(z[i]))^2); differentiable & convex, can be minimized with gradient descent. Weight update: w[j] += eta \* sum((y[i] – phi(z[i])) \* x[i][j]); Batch gradient descent: weight update is calculated based on all samples in the training set. Feature Scaling: Standardization: for feature j, x[j] = (x[j] – average(x[,j])) / standard\_deviation(x[,j]); Stochastic Gradient Descent: Instead of updating the weights based on the sum of the accumulated errors over all samples, update all the weights incrementally for each training sample; Typically reaches convergences faster because of the more frequent weight updates, Can escape shallow local minima more readily for nonlinear cost functions, Can be used for online learning (model is trained on the fly as new training data arrives); shuffle the training set for every epoch. Mini-Batch Learning: A compromise between batch gradient descent and stochastic gradient descent, apply batch gradient descent to smaller subsets of the training data; converge faster than batch gradient descent.

**Multiclass Classification** One-versus-Rest: Train one classifier per class, where the class is treated as positive & all other classes are considered negative,

to classify a new data sample, use all classifiers, assign the class label with the highest confidence to the particular sample. **Logistic Regression**: Logit function: defined as the logarithm of odds, domain = (0, 1), range = real numbers, logit(p) = ln(p) – ln(1 - p); logit(P(y = 1|x)) = z = dot(w, x); Activation function phi(z) = logit\_inverse(z). Threshold function y\_hat = 1 ? phi(z) >= 0.5 : 0; Loss function J(phi(z), y) = -y \* ln(phi(z)) – (1 - y) \* ln(1 – phi(z)); **Overfitting & Underfitting** model may perform well on training data but does not generalize well to test data; overfitting (high variance), underfitting (high bias); Regularization: introduce additional information (bias) to penalize extreme parameter (weight) values; useful method to handle collinearity (high correlation among features), filter out noise from data & eventually prevent overfitting; requires feature scaling e.g. standardization; L2 Regularization: add (lambda / 2) \* sum(w[j] ^ 2) to the cost function; lambda is the regularization parameter, larger means more regularization strength.