$W_{LS} = (X^T X)^{-1} X^t Y$

When $X^T X$ Is not invertible, multicollinearity or N<d

$W_{RLS} = (X^T X + \lambda I)^{-1} X^t Y$

Soft margin SVM:

$$\min \frac{1}{n} \sum^{n} Max\{0, 1 - y < w, \theta >\} + \lambda ||W||^2_2$$

Hard Margin SVM:

$$W^T x + b = 0$$

Find a^{LS}, 1. Substitute the loss function, 2. Take integral, 3. Differentiate for a, 4. Solve a

Find MLE: 1. Log of probabilities. 2. Differentiate for lambda. 3. Solving for lambda.

The decision boundary for NN is not linear.

An increase in λ can help with overfitting.

One-vs-All: N number of classifiers

One-vs-One (**All pairs**): Number of classifiers, $N * (N-1)/2$

End-to-End: Neural Network classifier.

Precision: $\frac{TP}{TP+FP}$

Accuracy: $\frac{TP+TN}{TP+FP+TN+FN}$

Recall: $\frac{TP}{TP+FN}$

Linear programming is an efficient way to find a linearly separable solution.

Universal approximation theorem: In a feed forward network with a single hidden layer containing a finite number of neurons it can approximate continuous functions.

Hinge loss is used by SVM to maximize the margin and penalize misclassifications. (Output is -1 or 1)

$$L(y, \hat{y}) = \max(0, 1 - y * \hat{y})$$

Cross entropy loss: Outputs the predicted probabilities (Between 0 and 1)

$L(Y, P) = -[y \log(P) + (1-Y) \log(1-P)]$

The **perceptron** cannot find a solution if the data is not linearly separable.

Hard Margin cannot find a solution if the data is not linearly separable.

A **Sigmond Neural Network** can be approximated following the universal approximation theorem.

Gradient Decent: Requires an initial guess, $w^{t+1} = w^t - a\, \Delta_w (E(w))$, is an optimization algorithm is used to minimize the cost function. Finds **global** or **local** minima.

Stochastic Gradient Decent: Randomly select a subset as a batch, it has the advantage of randomness. Gradient decent estimate:
$\hat{g} \leftarrow \frac{i}{m} \sum_i L(f(x^i; \theta), y^i)$,
Because of randomness it has the possibility to escape local minima due to the **noisy** estimate of the gradient.

Batch Gradient Decent: Divides the training data into a number of batches, $\Delta^{j}_{w} = \sum_{x,y} \Delta_w (l(f_w(x), y))$, this helps with memory and has the option for parallelization. Has the same speed of convergence as GD.

Back propagation is used by both GD and SGD

ResNET skips connections or uses shortcuts to allow the back propagation to reduce the **vanishing gradient**. **ReLu** does also reduce **vanishing gradient**.

Normal **back propagation** does not deal with the issue of **vanishing gradient**.

Early stopping: Can be used to mitigate overfitting.

Dropout: For each iteration of SGD drop a note with 1-p probability.

Weight sharing: To help with computational efficiency which makes the weights be shared so the number of weights to be trains in smaller.

Calculate the size of **CNN:**
$\left\lfloor \frac{Size - Kernel}{Stride} \right\rfloor + 1$

Random forests: Using random subspace and bagging, this is good with categorical features. Deeper trees can overfit.

Decision trees: Selects tree nodes randomly, labels are based on votes. Final class is done by voting between trees.

Boosting: Reduce bias, create stronger classifiers out of weaker classifiers. Picking a base classifier, and one by one address the short comings. In practice very good with test error.

Bagging: Using non overlapping training subsets, to create truly independent/diverse classifiers (**I.I.D.).** wasteful on small trainset.

Bootstrap sampling: Resampling technique drawing samples from source data to estimate a population parameter.

Vapnik-Chrvonenkis dimensions (VC): d+1 linear classifiers. VC(H) is the max number of points that H can shatter. Richer classes have higher VC dimensions.

Shatter: How many classes can possibly be made any set of linear classifiers can't shatter 4 points.

Shallow / Deep neural network: Deep networks experience vanishing gradient.

Threshold (Sign): $h(x) = \begin{cases} 1, x > 0 \\ o, O.W. \end{cases}$

Sigmond: $h(x) = \frac{1}{1+e^{-x}}$

ReLu (Rectified Linear): $h(x) = \max\{0, x\}$, Helps with vanishing gradient.

Tanh: $h(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

Fundamental Theorem of Statistical Learning: The optimal number of samples for **PAC** to learn H: $\theta(\frac{VC(h) + \log(\frac{1}{\delta})}{\epsilon^2})$

Pac learning (Probability approximately correct): Learns H using samples for every distribution. The loss of A(S) is less than the optimal plus some error.

Aggregation: Helps with variance and overfitting. Works good on base learners that are accurate and diverse.

Ensample learning: Train multiple classifiers and aggregate the decision (Voting). Reduce bias and variance.

N-Gram: Sequence of characters, words or tokens. Not very scalable.

Discriminator: Binary classifier, input is a real or fake image, and it assigns a true or false label.

Freezing some layers: This is done because some layers might do some elementary tasks, such as define a color. This is done during finetuning.

Fine tuning: Updates using target after layers have been frozen.

Max pooling (CNN): Takes the highest number in an area to transfer into another image.

NP Hard: A class of problems that are at least as hard as the hardest problem.

Empirical Risk minimum (EMR): Selecting the model with the smallest average error over the training set.

Exploiting locality: Sparse connectivity, better for computation?

Parameter sharing: If extracting one element in an image is useful, it will be useful in other parts of the image as well.