linear algebra

- **Scalars**: a single number
- **Vectors**: an array of numbers arranged in order
- **Matrices**: a 2-D array of numbers, $A \in R^{m*n}$ means A is a matrix with height of m and width of n.
- **Tensors**: an array of numbers arranged on a regular grid with a variable number of axes
- Product: $C_{i,j} = \sum_k A_{i,k} B_{k,j}$
- element-wise product: $A \odot B$
- Identity matrix: $A_{-1} A = I_n$
- **linear combination**: over a set of vectors $v^{(1)}, \ldots, v^{(n)}$, is $\sum_i c_i v^{(i)}$, the **span** is the set of all points obtained by linear combination. A set of vectors is **linearly independent** if no vector in the set is a linear combination of the other vectors.
- A square matrix with linearly dependent columns is known as **singular**
- **Norm**: $||x||_p = (\sum_i |x_i|^p)^{\frac{1}{p}}$, when $p = 2$ known as **Euclidean norm**
- **symmetric**: $A = A^T$. **unit vector**: $||x||_2 = 1$. **orthogonal**: $x^T y = 0$
- An **eigenvector** of a square matrix A is a non-zero vector v such that multi- plication by A alters only the scale of v: $Av = \lambda v$

Perceptron

- **Training set** ($X = [x^1, \ldots, x^n]$, $y = [y_1, \ldots, y_n]$)
  - $x^i \in R^d$: instance with d features
  - $y_i \in -1, 1$
- **Linear threshold function**: $y_i = sign(< w, x^i > + b)$, where $w \in R^d$ is **separating hyperplane**, $b \in R$ is offset or bias
- **perceptron** is an algorithm for **supervised learning** of **binary classifier**
- Single layer perceptrons are only capable of learning linearly separable patterns

```
repeat
    select some column a of training set A:
        if <a, w> ≤ s then
            w = w + a
            t = t + 1
until convergence
```

- If the training set is linearly separable, then the perceptron is guaranteed to converge. Suppose the input vectors can be separated by a hyperplane with a margin γ, i.e. there exists a weight vector w ,$||w|| = 1$, and a bias b such that $w \cdot x_j > \gamma$ for all $j : d_j = 1$ and $w \cdot x_j < \gamma$ for all $j : d_j = 0$. And let R be the maximum norm of an input vector. Novikoff (1962) proved that in this case the perceptron algorithm converges after making $O(R^2/\gamma^2)$ updates. The

idea of the proof is that the weight vector is always adjusted by a bounded amount in a direction with which it has a negative dot product, and thus can be bounded above by $O(\sqrt{(t)})$ where t is the number of changes to the weight vector. But it can also be bounded below by O(t) because if there exists an (unknown) satisfactory weight vector, then every change makes progress in this (unknown) direction by a positive amount that depends only on the input vector.

- The larger the margin, the faster the perceptron converges. But perceptron stops at an arbitrary linear separator
- If non-separable:
  - Find a better feature representation
  - Use a deeper model
  - Soft margin
- Multiclass Perceptron: One vs all / One vs One

Multilayer perceptron

- Each layer:
  - Input x
  - z = Ux + c, where U is matrix
  - h = f(z), f is an activation function (**nonlinear transform**)
  - output: $\hat{y} =< h, w > +b$
- Activation function:
  - **Sigmoid** $f(t) = \sigma(t) = \frac{1}{1+e^{-t}}$
  - **Tanh** $f(t) = tanh(t) = \frac{e^t - e^{-t}}{e^t + e^{-t}}$
  - **Rectified Linear** $f(t) = t_+ := max(t, 0)$
- **Gradient Descent**: $\min_\Theta L(\Theta) := \frac{1}{n} \sum_{i=1}^{n} [q(x_i; \Theta), y_i]$
  - $\Theta_{t+1} = \Theta_t - \eta \nabla(\Theta_t)$
- **Backpropogation: chain rule** $f(x) = g[h(x)] \rightarrow f'(x) = g'[h(x)] * h'(x)$

Linear Regression

- **linear regression** is a linear approach for modeling the relationship between a scalar dependent variable y and one or more explanatory variables (independent) X
- Given a pari (X, Y), find function such that $f(X) \approx Y$
- Minimize expected loss (risk): $\min_{f: X \rightarrow y} E[L(f(X))]$
- Least squares: $min_f E||f(X) - Y||_2^2$
- $min_{A,b} \frac{1}{n} \sum_{i=1}^{n} ||X_i A + b - Y_i||_2^2$ = $min_W \frac{1}{n} \sum_{i=1}^{n} ||X_i W - Y||_2^2$ = $min_{W \in R^{(d+1)*m}} ||XW - Y||_F^2$
- Solving least squares: $min_{W \in R^{(d+1)*m}} ||XW - Y||_F^2 = W^T (X^T X) W - 2W^T X^T Y + Y^T$ => $X^T XW = X^T Y$ (Normal Equation)

KNN (K-nearest neighbors)

- Distance d:$X, X \to R_+$ such that
  - **symmetry**: d(x, x') = d(x, x')
  - **definite**: d(x, x') = 0 iff x = x'
  - **triangle inequality**: $d(a,b) <= d(a,c) + d(c,b)$
- Eg L_p distance: $d_p(x,x') = ||x - x'||_p$ Euclidean if p=2, Manhattan if p =1, Chebyshev if p=inf
- **Complexity** of 1-NN. (n: number of training samples, d: number of features)
  - **Training** O(1) but O(nd) space
  - **Testing** O(nd) for each query point
  - alternative method: Voronoi diagram. query cost O(log n) in 2D. $O(n^d)$ space and O(dlogn) query time for learger d.
- **k-NN**
  - Store training set (X, y)
  - For query x', find k nearest points $x_1, x_2, \ldots, x_k$ in X, predict $y' = mode(y(x_1, \ldots, x_k))$
  - Test complexity: O(ndk)
- **Bayes error**: $P^* = \min_{f:X \to \{\pm 1\}} P(f(X) \neq Y)$
- **Bayes rule**: $P(f(X) \neq Y) = P(f(X) = 1, Y = -1) + P(f(X) = -1, Y = 1)$ = $E[P(f(X) = 1, Y = -1|X|) + P(f(X) = -1, Y = 1|X)]$ = $E[1_{f(X)=1} P(Y = -1|X) + 1_{f(X)=-1} P(Y = 1|X)]$ = $E[1_{f(X)=1}(1 - \eta(X)) + 1_{f(X)=-1}\eta(X)]$ = $E[\eta(X) + 1_{f(X)=1}(1 - 2\eta(X))]$ where $\eta(X) = P(Y = 1|X)$
- Multi-class $f^*(X) = argmax_{m=1,\ldots,c} P(Y = m|X)$, $P^* = 1 - max_{m=1,\ldots,c} P(Y = m|X)$

Hard-margin SVM

- Training set is linearly separable and exists a halfspace (w, b), such that $y_i = sign(< w, x_i > +b)$ for all i. Rewritten as $\forall i \in [m], y(< w, x_i > +b) > 0$
- **The distance between a point x and a hyperplane defined by (w, b) where $||w|| = 1$ is** $| < w, x > +b|$
  - Proof: distance between a point x and the hyperplane is $min\{||x - v|| :< w, b > +b = 0\}$ Taking $v = x - (< w, x > +b)w$ we have $< w, v > +b =< w, x > -(< w, x > +b)||w||^2 + b = 0$ and $||x - v|| = | < w, x > +b| * ||w|| = | < w, x > +b|$. We take any other point u on the hyperplane, thus $< w, u > +b = 0$ we have $||x - u||^2 = ||x - v + v - i||^2$ = $||x - v||^2 + ||v - u||^2 + 2 < x - v, v - u > \geq ||x - v||^2 + 2 < x - v, v - u >$ = $||x - v||^2 + 2(< w, x > +b) < w, v - u >= ||x - v||^2$ the distance between x and u is at least the distance between x and v
- The closest point in the traiing set to separating hyperplane is $\min_{i \in [m]} | < w, x_i > +b|$ therefore the **Hard-SVM rule** is $argmax_{(w,b):||w||=1} min_{i \in [m]} | < w, x_i > +b|$ s.t. $\forall i, y_i(< w, x_i > +b) > 0$.

- $max_{w,b} \frac{1}{||w||_2}$ or $min_{w,b} \frac{1}{2}||w||_2^2$ s.t. $\forall i, y_i(w^T x_i + b) \geq 1$
- hard-SVM searches for **w** of minimal norm among all the vectors that separate the data and for which $|< w, x_i > + b| \geq 1$ for all i. We enforce the margin to be 1, finding the largest margin halfspace boins down to finding w whose norm is minimal
- Proof:
  - Let (w^*, b^*) be a solution of hard margin SVM and define the margin be $\gamma^* = min_{i \in [m]} y_i(< w^*, x_i > + b^*)$. Therefore, $\forall i \; y_i(< w^*, x_i > + b^*) \geq \gamma^*$ or equivalently $y_i(< \frac{w^*}{\gamma^*}, x_i > + \frac{b^*}{\gamma^*}) \geq 1$. Hence the pair $(\frac{w^*}{\gamma^*}, \frac{b^*}{\gamma^*})$ satisfies the conditions of the quadratic optimization problem. Therefore $||w_0|| \leq ||\frac{w^*}{\gamma^*}|| = \frac{1}{\gamma^*}$. Then $\forall i$, $y_i(< \hat{w}, x_i > + \hat{b}) = \frac{1}{||w_0||} y_i(< w_0, x_i > + b_0) \geq \frac{1}{||w_0||} \geq \gamma^*$ Since $||\hat{w}|| = 1$ than $(\hat{w}, \hat{b})$ is optimal.
- **Lagrangian dual**
  - **Primal:** $\min_{w,b} \frac{1}{2}||w||_2^2$ s.t. $\forall i, y_i(w^T x_i + b) \geq 1$
  - **Lagrangian Dual**: $\min_{w,b} max_{\alpha \geq 0} \frac{1}{2}||w||_2^2 - \sum_i \alpha_i [y_i(w^T x_i + b) - 1]$
    - equation equals 0 if $\forall i, y_i < w, x_i > \geq 1$ and equals to ∞ otherwise
    - once α is fixed, the optimization problem with repect to **w** is unconstrained and the objective is differentiable
  - **Deriving the dual**: set the objective's gradient equals to zero. Plugging the result into the dual equation. Rearranging yields the dual problem.

Soft-SVM

- viewed as a relaxation of the Hard-SVM that can be applied even if the training set is not linearly separable.
- introducing **nonnegative slack variables** $\xi_1, \ldots, \xi_m$ and replacing each constraint by $y_i(< w, x_i > + b) \geq 1 - \xi_i$. ξ_i measures by how much the constraint is being violated.
- **Rule**: $\min_{w,b,\xi} \frac{1}{2}||w||_2^2 + C \sum_{i=1}^n \xi_i$ s.t. $\forall i, (1 - y_i \hat{y}_i)_+ \leq \xi_i$ where C is a parameter controls the tradeoff between the two terms.
- **the hinge loss**: $(1 - y\hat{y})_+ = max\{1 - y\hat{y}, 0\}$
- **Lagrangian**
  - $\min_{w,b,\xi} \max_{\alpha \geq 0, \beta \leq 0} \frac{1}{2}||w||_2^2 + \sum_i C\xi_i + \alpha(1 - y_i \hat{y}_i - \xi_i) + \beta_i \xi_i$
    - $\frac{\partial}{\partial b} = \sum_i \alpha_i y_i$
    - $\frac{\partial}{\partial \xi_i} = C - \alpha_i + \beta_i = 0$
    - $\frac{\partial}{\partial w} = w - \sum_i \alpha_i y_i x_i = 0$
    - plug in, get: $\max_{C \geq \alpha \geq 0} \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j x_i^T x_j$
- **Gradient Descent**: $\min_{w,b} L(w) := \frac{C}{n} \sum_{i=1}^n l(y_i \hat{y}_i) + \frac{1}{2}||w||_2^2$

Kernel

- **lifting**: map the original instance space into another space to make the class of halfspaces more expressive.
- **feature space** $\psi : R^d \rightarrow R^h$
- basic paradigm:
  - Given domain set X and a learning task, choose a mapping ψ
  - Given a sequence of labeled examples $S = (x_i, y_i), \ldots, (X_m, y_m)$, create the image sequence $\hat{S} = (\psi(x_1), y_1), \ldots, (\psi(x_m), y_m)$
  - Train a linear predictor h over $\hat{S}$
  - Predict the label of a test point x to be $h(\psi(x))$
- Embedding the input space into high dimensional feature space makes halfspace learning more expressive. However, the **computational complexity** of learning is a serious problem.
- **Kernels**: inner products in the feature space, we define the kernel function $K(x, x') =< \psi(x), \psi(x') >$
- Think of K as specifying similarity between instances and of the embedding ψ as mapping the domain set X into a space where these similarities are realized as inner products.
- Examples:
  - Polynomial kernel $k(x, x') = (x^T x')^p$ or $k(x, x') = (x^T x' + 1)^p$
  - Gaussian Kernel (Square exponential kernel) $k(x, x') = exp(-||x - x'||_2^2/\sigma)$
  - Laplace Kernel $k(x, x') = exp(-||x - x'||_2/\sigma)$
  - Matern Kernel ...
- **Kernel matrix**: $K_{ij} = k(x_i, x_j)$ is symmetric and positive semidefinite
- **positive semidefinite (PSD)** $\forall \alpha \in R^n, \alpha^T K \alpha = \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j K_{ij} \geq 0$
- **Kernel calculus**
  - If k is a kernel, so is λk for any $\lambda \geq 0$
  - If k_1 and k_2 are kernels, so is $k_1 + k_2$
  - If k_1 and k_2 are kernels, so is $k_1 k_2$
- Instance of the general problem $\min_w (f(< w, \psi(x_i) >, \ldots, < w, \psi(x_m) >) + R(||w||))$ (denote Equation 1)
  - Soft-SVM let $R(a) = \lambda a^2$ and $f(a_1, \ldots, a_m) = \frac{1}{m} \sum_i \{0, 1 - y_i a_i\}$
  - Hard_SVM let $R(A) = a^2$ and let $f(a_1, \ldots, a_m)$ be 0 if there exists b such that $y_i(a_i + b) \geq 1$ for all i or ∞ otherwise
- **Representer Theorem** Assume that ψ is a mapping from X to a Hilbert space. Then there exists a vector $\alpha \in R^m$ such that $w = \sum_{i=1}^m \alpha_i \psi(x_i)$ is an optimal solution of Equation 1
  - Proof: Let w^* be an optimal solution of Equation 1. we can rewrite w^* as $w^* = \sum_{i=1}^m \alpha_i \psi(x_i) + u$. Set $w = w^* - u$. Clearly, $||w^*||^2 = ||w||^2 + ||u||^2$, thus $||w|| \leq ||w^*|$. Since R is nondecreasing, obtain $R(||w||) \leq R(||w^*||)$. For all i we get $< w, \psi(x_i) >=< w^* - u, \psi(x_i) >=< w^*, \psi(x_m) >$. Hence $f(< w, \psi(x_1) >, \ldots, < w, \psi(x_m) >) = f(< w^*, \psi(x_1 >, \ldots, < w^*, \psi(x_m) >)$. The

objective of Equation 1 at w canot be larger than at w^*. Therefore w is also an optimal slution.

- Writing $w = \sum_{j=1}^{m} \alpha_j \psi(x_j)$ for all i, we get
  $< w, \psi(x_i) >=< \sum_j \alpha_j \psi(x_j), \psi(x_i) >= \sum_{j=1}^{m} \alpha_J < \psi(x_j), \psi(x_i) >$ Let
  $K(x, x') =< \psi(x), \psi(x') >$ be a function implements the kernel function. Instead of solving Equation 1, we can solfe the equivalent problem
  $\min_{\alpha \in R^m} f(\sum_{j=1}^{m} \alpha_j K(x_j, x_1), \ldots, \sum_{j=1}^{m} \alpha_j K(x_j, x_m)) + R(\sqrt{\sum_{i,j=1}^{m} \alpha_i \alpha_j K(x_j, x_i)})$,
  denote this equation to Equation 2
- To solve the Equation 2, we do not need direc access to elements in the feature space, the only thing we should know is **how to calculate inner products in the feature space** or the **m*m matrix called** Gram matrix (or **Kernel matrix**).
- Advantage: dimension of the feature space may be extremely large while implementing the kernel function is very simple.
- To predict the new element, computer the inner product of it and the training data, and predict with the weight we trained.

Gaussian Process

- **Gaussian distribution**: $p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{[-\frac{(x-\mu)^2}{2\sigma^2}]}$
- Important facts
  - $\begin{matrix} X \sim N_d(\mu, \Sigma) \\ A \in R^{p*d} \end{matrix} \rightarrow AX \sim N_p(A\mu, A\Sigma A^T)$
  - $\begin{pmatrix} X_1 \\ X_2 \end{pmatrix} \sim N\left(\begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}, \begin{bmatrix} \Sigma_{11}\Sigma 12 \\ \Sigma 21 \Sigma 22 \end{bmatrix}\right)$
  - joint = marginal * conditinal
  - $X \sim N(\mu_1, \Sigma_{11})$, $X_2 | X_1 \sim N(\mu, \Sigma)$ where $\mu = \mu_2 + \Sigma_{21}\Sigma_{11}^{-1}(X_1 - \mu_1)$, $\Sigma = \Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12}$
- In aupervised learning, we assume $y_i = f(x_i)$ for some unknown function f possibly corrupted by noise.Optimal approach is to infer a **distribution over function** $p(f|X, y)$, and then to use this to make predictions $p(y_*|x_*, X, y) = \int p(y_*|f, x_*)p(f|X, y)df$
- Do not attempt to identify best-fit model. Instead, compute a posterior predictive distributions over models.
- **Multivariate Gaussians**
  - write $x \sim N(\mu, \Sigma)$ if a vector-valued random variable $x \in R^n$ have a **multivariate normal distribution** with mean μ and covariance matrix $\Sigma \in S_{++}^n$, where S_{++}^n refers to the space of symmetric positive definite n*n matrices.

- Consider a random vector $x \in R^n$ with $x \sim N(\mu, \Sigma)$. Suppose that variables in x have been partitioned into two sets $x_A = [x_1 \dots x_r] \in R^r$ and $x_B = [x_{r+1} \dots x_n] \in R^{n-r}$. such that $x = \begin{bmatrix} x_A \\ x_B \end{bmatrix}$, $\mu = \begin{bmatrix} \mu_A \\ \mu_B \end{bmatrix}$ and $\Sigma = \begin{bmatrix} \Sigma_{AA} \Sigma_{AB} \\ \Sigma_{BA} \Sigma_{BB} \end{bmatrix}$. Then the following properties hold:
  - **Normalization**: $\int_x p(x; \mu, \Sigma) dx = 1$
  - **Marginalization**: $p(x_A) = \int_{x_B} p(x_A, x_B; \mu, \Sigma) dx_B$
    $p(x_B) = \int_{x_A} p(x_A, x_B; \mu, \Sigma) dx_A$
  - **Conditioning**: $p(x_A | x_B) = \frac{p(x_A, x_B; \mu, \Sigma)}{\int_{x_A} p(x_A, x_B; \mu, \Sigma) dx_A}$ $p(x_B | x_A) = \frac{p(x_A, x_B; \mu, \Sigma)}{\int_{x_B} p(x_A, x_B; \mu, \Sigma) dx_B}$
  - **Summation**: Given $y \sim N(\mu, \Sigma)$ and $z \sim (\mu', \Sigma')$, then $y + z \sim N(\mu + \mu', \Sigma + \Sigma')$
- **Gaussian processes** are the extension of multivariate Gaussians to infinite-sized collections of real-valued variables.
- A **Gaussian processes (GP)** defines a prior over functions, which can be converted into a posterior over functions once we have seen some data. Only need to define a distribution over the function's values at a finite, but arbitrary, set of points, say $x_1, \dots, x_n$. A GP assumes $p(f(x_1), \dots, f(x_n))$ is jointly Gaussian with some mean $\mu(x)$ and covariance $\Sigma(x)$ given by $\Sigma_{ij} = K(x_1, x_j)$
- A collection of Gaussian random variables $\{Z_t : t \in T\}$ such that for any finite N, $\{Z_T : t \in N\}$ is jointly Gaussian. A Gaussian process is a function of two variables $Z(t, w)$
  - For any finite N, $\{Z_t := Z(t, w) | t \in N\}$ is a Gaussian random vector
  - For any w, $Z_w := Z(t, w) : T \to R$ is a function of one variable t
- Let $X = \{x_1, \dots, x_m\}$ be any finite set of elements. Consider the set $H = \{f_0, f_1, \dots\}$ of all possible functions mapping from X to R. Since $f(\cdot) \in H$ has onely m elements, can represent $f(\cdot)$ compactly as an m-dimensional vector $\vec{f} = [f(x_1), f(x_2), \dots, f(x_m)]^T$. Then associate some **probability density** with each function in H. This shows the probability distributions over functions with finite domains can be represented using a finite-dimensional multivariate Gaussian distribution over function outputs $f(x_1), \dots, f(x_m)$ at finite number of input points $x_1, \dots, x_m$
- A **Gaussian process** is a stochastic process such that any finite subcollection of random variables has a multivariate Gaussian distribution.
- A collection of random variables $\{f(x) : x \in X\}$ is drawn from a Gaussian process with **mean function** $m(\cdot)$ and **covariance function** $k(\cdot, \cdot)$ if any finite set of elements $x_1, \dots, x_m \in X$, the associated finite set of random variables $f(x_1), \dots, f(x_m)$ have distribution
$$\begin{bmatrix} f(x_1) \\ \vdots \\ f(x_m) \end{bmatrix} \sim N\left(\begin{bmatrix} m(x_1) \\ \vdots \\ m(x_m) \end{bmatrix} \begin{bmatrix} k(x_1, x_2) \dots k(x_1, x_2) \\ \vdots \ddots \ddots \ddots \vdots \\ k(x_m, x_1) \dots \ k(x_m, x_m) \end{bmatrix} \right).$$

Mixture of Gaussians

- **Mixture models**: $p(x|\theta) = \sum_{k=1}^K p(z = k) p(x | z = k, \theta)$

- **k-mean clustering algorithm**
  - given a training set $\{x^{(1)}, \ldots, x^{(m)}\}$ and to group the data into a few clusters
  - Algorithm:
    1. Intialize cluster centroids $\mu_1, \mu_2, \ldots, \mu_k \in R^n$ randomly
    2. Repeat until converge:
       1. For every i, set $c^{(i)} = argmin_j ||x(i) - \mu_j||^2$
       2. For every j, set $\mu_j = \frac{\sum_{i=1}^m 1\{c^{(i)}=j\}x^{(i)}}{\sum_{i=1}^m 1\{c^{(i)}=j\}}$
  - In the loop, assign each training example to the closest cluster centroid and moving each cluster centroid to the mean of the points assigned to it.
  - The k-means algorithm guaranteed to converge.
- **Mixtures of Gaussians and the EM algorithm**
  - Model the data by specifying a joint distribution $p(x^{(i)}, z^{(i)}) = p(x^{(i)}, |z^{(i)})p(z^{(i)})$, where $z^{(i)} \sim Multinomial(\phi)$.
  - We model each $x^{(i)}$ was generated by randomly choosing $z^{(i)}$ from $\{1, \ldots, k\}$, and then $x^{(i)}$ was drawn from one of k Gaussians depending on $z^{(i)}$.
  - **EM algorithm**
    1. (E-step) for each i,j, set $w_j^{(i)} = p(z^{(i)} = j|x^{(i)}; \phi, \mu, \Sigma)$
    2. (M-step)
       1. $\phi_j = \frac{1}{m}\sum_{i=1}^m w_j^{(i)}$
       2. $\mu_j = \frac{\sum_{i=1}^m w_j^{(i)} x^{(i)}}{\sum_{i=1}^m w_j^{(i)}}$
       3. $\Sigma = \frac{\sum_{i=1}^m w_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)}{\sum_{i=1}^m w_j^{(i)}}$
  - E-step tries to guess the values of the $z^{(i)}$'s. In E-step, calculate the posterior probability of parameters of the $z^{(i)}$'s given $x^{(i)}$.
  - M-step update the parameters of mode based on guesses.

Deep Learning

- **Neural Networks**
  - **Input layer**: m input features $x_1, \ldots, x_m$
  - **Hidden layer**: hidden units. These units are called "hidden" because we do not have the truth/training value for the hidden unites. It is possible to have multiple hidden layers.
  - **Output layer**: output neuron
  - **Loss function**:
    - Sigmoid: $g(z) = \frac{1}{1+e^{-z}}$
    - ReLU: $g(z) = max(z, 0)$

    - tanh: $g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$
- **Bias vs Variance**: is tradeoff of simultaneously minimizing two sources of error taht prevent supervised learning algorithms from generalizing beyond their training set
- **Regularization**: reduce generalization but not its training error
- **Injecting noise in outputs**
- **Early stopping**
- **Ensembles**: training multiple modelsand have them vote
- **Dropout**: turn of an activation randomly to prevent overfiting.
- **Momentum**: adds a velocity term as learning rate, initially large and decay over time
- How to choose optimizer: lec12 p12.
- **Convolutional Neural Networks (CNNs)**
    - similar to ordinary Neural Networks made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity.
    - A simple ConvNet is a sequence of layers, each layer transforms one volume of activations to another. Three main types of layers: **Convolutional Layer**, **Pooling Layer**, **Fully-Connected Layer**
    - **(N - F + 2 * P) / S + 1**
    - **Summary**:
        - Accepted volumn of size: $W_1 * H_1 * D_1$
        - Hyperparameters:
            - Number of filters K
            - Filter size F
            - Stride S
            - amount of zero padding P
        - Produces a volume of size $W_2 * H_2 * D_2$
            - $W_2 = (W_1 - F + 2P)/S + 1$
            - $H_2 = (H_1 - F + 2P)/S + 1$
            - $D_2 = K$
    - **Pooling**
        - Benefits: Some translation invariance, No params, easy to backprop, less computations, Increased receptive field
        - Bad： loss information(Though these information are usually useless)
        - N = (N - F) / S + 1
    - **Receptive field**: How much a value can see in the original data.
    - x layers, n*n filter, D depth, $xn^2 D_1 D_2$ numebr of params

Recurrent Neural Networks (RNN)

- Motivation: Introducing bias for modeling sequences. Multiple inputs into a model
- **LSTM**
  - f: **Forget gate**
  - i: **Input gate**
  - g: **Gate gate**
  - o: **Output gate**

Generative Adversarial Networks (GAN)

- Goal: Given training data p_{data} and generate new samples from p_{model}
- A **generator** proposes samples, a **discriminator** examines samples. The two network play a minimax game on the error rate of the generator.
- A **discriminator** is a classifier given an image and ouput a image between 0 and 1.
- A **generator** is usualy a convlutional neural network. Given a random noise and it generate some image from the noise.
- At beginning the generator is making just random noise. And then the generator gets negative reward from the discriminator. Eventually, the generator will create image indistinctable from the original image and fools discriminator.
- $\min_G \max_D E_{x\sim p_{data}}[logD(x)] + E_{x\sim p_{model}}[log(1-D(x))]$
  - $D(X)$: probability of x coming from same distribution as training data
  - Discriminator D: maximize probability of training data and minimize probability of generated sample
  - Generator G: maximize probability of generated sample to fool discriminator

Decision Tree

- How to choose attributes:
  $(j^*, t^*) = argmin_{j=1,..,d} \min_{t\in T_j} l(\{(x_1, y_i), : x_{ij} \leq t\} + l(\{(x_i, y_i) : x_{ij} > t\}))$
- Misclassification error: $l(D) = 1 - \hat{p_{\hat{y}}}$
- Entropy: $l(D) = -\sum_{c=1}^{C} \hat{p}_c log\hat{p}_c$

Bagging and Boosting

- **Boostrap aggregating - bagging**
  - Train each learner on different learner using same learning algorithm
  - Create m bags of data each is a subset of training data, add data randomly with replacement from the training data to each bags. Each bag is used to train different model and we get different models. To prodict, we predict using all mdoels and take the average of the output.
- **Boosting: Ada Boost**

- Train each model with a bag of data randomly selected from training data as usual way.
- Than test using the whole training data. To find the data that are not well predicted (with siginificant error). When building the next bag, the data with siginicant error are more likely to be picked and trained.