- Supervised learning: labeled with correct behavior
- Reinforcement: learning interacts with the world, max scalar reward
- Unsupervised: no label example - looking for interesting pattern
- KNN: $\|x^{(a)} - x^{(b)}\|_2 = \sqrt{\sum_{j=1}^{D} (x_j^{(a)} - x_j^{(b)})^2}$ j - dimension $x \in \mathbb{R}^D$
- Algo: 1. Find k examples $\{x^{(i)}, t^{(i)}\}$ closest to the test instance x.
  2. classification output is major class.

$$y^* = \underset{t^{(z)} \in \text{class label}}{\text{argmax}} \sum_{i=1}^{k} \mathbb{I}(t^{(z)} = t^{(i)})$$

- small k: may overfit, be sensitive to random idosyncrasies in training data
- Large k: may underfit, fail to capture important regularities.
- balance: depends on n, $k < \sqrt{n}$, choose use validation set.
- measure generalization error using test set. each point is $O(\varepsilon^d)$
- Curse of dimensionality: The volume of a single ball of radius ε around ↑
  The total volume of $[0,1]^d$ is 1. $O((\frac{1}{\varepsilon})^d)$ points are needed to cover the ↑ volume. In high dimension, most points are approx. same distance.
- Normalization: Sensitive to different ranges of different features (units)
  $\tilde{x}_j = \frac{x_j - \mu_j}{\sigma_j}$ simple fix: normalize each dimension to be zero mean & unit variance.
- Computation cost: compute D-dimension Euc dis with N points $O(ND)$
  Don't sort dis $O(N \log N)$ · Store entire data set · Done for each query
- Entropy: High Entropy: flat hist, uncertainty↑, unpredictable

$$P(y|x) = \frac{P(x,y)}{P(x)} \quad P(x) = \sum_y P(x,y) \quad H(Y) = -\sum_{y \in Y} P(y) \log_2 P(y)$$

Joint Entropy: $H(X,Y) = -\sum_{x \in X} \sum_{y \in Y} P(x,y) \log_2 P(x,y)$

Conditional Entropy: $H(Y|X=x) = -\sum_{y \in Y} P(y|x) \log_2 P(y|x)$

Expected: $H(Y|X) = E_x[H[Y|X]] = \sum_{x \in X} P(x) H(Y|X=x)$

$$= -\sum_{x \in X} \sum_{y \in Y} P(x,y) \log_2 P(y|x)$$

- $H(X,Y) = H(X|Y) + H(Y) = H(Y|X) + H(X)$
- X, Y independent: $H(Y|X) = H(Y)$ $H(Y|Y) = 0$ $H(Y|X) \le H(Y)$
- $IG(Y|X) = H(Y) - H(Y|X)$ X completely uninformative $IG(Y|X) = 0$, informative $IG(Y|X) = H(Y)$
- Algo: Decision Tree
  - Pick a feature to split at a non terminal node
  - Split examples into groups based on feature value
  - for each group: if no example, return majority from parents
    else if all example in same class - return class
    else loop to step 1
  - Terminal when all leaves contain same class or empty.
- not too small: need to handle important but possible subtle distinction in data; not too big: computation efficiency, avoid overfitting
  Human Intepretability ⇒ desire small tree with infor nodes near root.
- KNN { fewer hyperparameter (k); incorporate interesting distance measure
  DT { simple to deal with discrete features, missing values, and poorly scaled data · Fast at test time · more intepretable

## Bias-Variance Decomposition:

- Generalization: Ability of a model to correctly classify/predict drawn from unseen example (from the same dist, that the training data was)
- Square Error loss: $L(y,t) = \frac{1}{2}(y-t)^2$ $p(t|x)$: t random, choose y

$y^* = E[t|x]$ prove: $E[(y-t)^2|x] = E[y^2 - 2yt + t^2 | x] = y^2 - 2yE[t|x] + E[t^2|x]$

$= y^2 - 2yE[t|x] + E[t|x]^2 + Var[t|x]$ $Var[x] = E[x^2] - E[x]^2$

$= y^2 - 2yy^* + y^{*2} + Var[t|x] = (y-y^*)^2 + Var[t|x]$ Bayes Error

Bayes Optimal: Best we hope to with any learning algo.

Treat y as random: $E[(y-t)^2] = E[(y-y^*)^2] + Var[t]$

$= E[y^2 - 2y^*y + y^{*2}] + Var[t] = (y^{*2} - 2y^*E[y] + E[y]^2) + Var[y] + Var[t]$

Bayes Optimality: $E[(y-t)^2] = (y^* - E[y])^2 + Var(y) + Var(t)$ (batch)

Bias: How wrong the (use boosting) Variance (use bagging) Bayes error
prediction is corresponds to underfitting.

Variance: the amount of variability in predictions (overfitting)

Bayes error: the inherant unpredictability of the targets.

- prediction y at query x is a random variable (from the choice of dataset)
- $y^*$ is the optimal deterministic prediction (choice of label)
- t is a random target sample from true conditional $p(t|x)$ (from ↑)

- Bagging: sample m independent training set from Psample.
  Idea { compute $y_i$ using each training set.
  compute average prediction $y = \frac{1}{m} \sum_{i=1}^{m} y_i$

Bias unchanged $E[y] = E[\frac{1}{m} \sum_{i=1}^{m} y_i] = \frac{1}{m} \cdot m \cdot E[y_i] = E[y_i]$

Variance ↓: $Var[y] = V[\frac{1}{m} \sum_{i=1}^{m} y_i] = \frac{1}{m^2} m \cdot Var[y_i] = \frac{1}{m} Var[y_i]$

Algo: Given training set D, use the empirical distribution $P_D$ as a proxy for Psample ("bootstrap aggregation or bagging")

- Take a dataset D with n examples · Generate m new dataset ("resample or bootstrap sample"). Each dataset has n examples sampled from D with replacement · Average the predictions of models trained on the m dataset. As $|D| \to \infty$, $P_D \to$ Psample

- Random forest: reduce correlation between bagged decision tree. Var↓ (for each node, chose a random subset of features)
- Bagging: · reduce overfitting by averaging predictions · A small ensamble often better than a single great model. limitation: · bias unchanged in case of square error. · Correlation in classifiers mean less Var↓ ⇒ Add more randomness in RF. · Weighted members equally may not be the best ⇒ weighted ensambling often leads to better results if member are very different. ($y_{bagged} = \mathbb{I}(\frac{1}{m} \sum_{i=1}^{m} y_i > 0.5)$ Binary)

- Linear regression: $y = f(x) = \sum_j w_j x_j + b = w^T x + b$ (w, b are parameter)

Loss function: L(y,t) defines how badly the algo's prediction y fits target t for some example x. Square error loss func: $L(y,t) = \frac{1}{2}(y-t)^2$ → residual y-t

Cost function: loss function averaged overall training example
empirical or average "loss".

$$J(w,b) = \frac{1}{2N} \sum_{i=1}^{N} (y^{(i)} - t^{(i)})^2 = \frac{1}{2N} \sum_{i=1}^{N} (w^T x^{(i)} + b - t^{(i)})^2$$ dataset.

Design matrix & target vector ⇒ compute prediction for the whole

$$Xw + b = y \quad \begin{pmatrix} x_1^{(1)} & x_2^{(1)} & \cdots & x_D^{(1)} \\ & & & \\ x_1^{(N)} & x_2^{(N)} & \cdots & x_D^{(N)} \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_D \end{pmatrix} + b \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} = \begin{pmatrix} y^{(1)} \\ \vdots \\ y^{(N)} \end{pmatrix} \text{ vs } \begin{pmatrix} t^{(1)} \\ \vdots \\ t^{(N)} \end{pmatrix}$$

$X^{(p)}_{N \times D}$

$$y = Xw \quad X = \begin{bmatrix} 1 & [x^{(1)}]^T \\ 1 & [x^{(2)}]^T \\ \vdots & \end{bmatrix} \in \mathbb{R}^{N \times D+1} \text{ and } w = \begin{bmatrix} b \\ w_1 \\ \vdots \end{bmatrix} \in \mathbb{R}^{D+1}$$

$\begin{pmatrix} w_1 w_2 & w_1 x_2 - x_1 x_2 - x_1 x_2 \\ x_2 x_1 & w_1 x_2 x_2 w_2 \end{pmatrix}$

Squared error cost: $\vec{y} = X\vec{w} + b\vec{1}$, $J = \frac{1}{2N} \|y - t\|^2$, $J = \frac{1}{2} \|y - t\|^2$

Optimization: minimize cost function $J(\vec{w})$

$\nabla_w J = \frac{\partial J}{\partial w} = \begin{pmatrix} \partial J / w_1 \\ \partial J / w_D \end{pmatrix}$ ① direct: Take gradient wrt w, set it = 0
② iterative solution Sum:

①: $J(w) = \frac{1}{2} \|X\vec{w} - \vec{t}\|_2 = \frac{1}{2} (X\vec{w} - \vec{t})^T (X\vec{w} - \vec{t}) = \frac{1}{2} (\vec{t} - X\vec{w})^T \mathbb{I} (\vec{t} - X\vec{w})$

$\frac{\partial J(w)}{\partial w} = \frac{1}{2} [-2X^T (\vec{t} - X\vec{w})] = X^T X\vec{w} - X^T\vec{t}$ set $\frac{\partial J}{\partial w} = 0 \Rightarrow \vec{w} = (X^T X)^{-1} X^T \vec{t}$

② Initialize the weights to something reasonable (eg. all zeros)
and repeatedly adjust them in the direction of steepest descent.
$\partial J / \partial w_j > 0 \Rightarrow$ decreasing J requires decreasing $w_j$ ↙
$\partial J / \partial w_j < 0 \Rightarrow$ decreasing J requires increasing $w_j$ ↘ $w_j \leftarrow w_j - \alpha \frac{\partial J}{\partial w_j}$
learning rate: α (α↑, change fast) If min J total loss ⇒ $\alpha' = \alpha / N$ 0.001~0.1

$\vec{w} \leftarrow \vec{w} - \alpha \frac{\partial J}{\partial \vec{w}}$ (for: Linear Re: $\vec{w} \leftarrow \vec{w} - \frac{\alpha}{N} \sum_{i=1}^{N} (y^{(i)} - t^{(i)}) x^{(i)}$)

Gradient descent: update $\vec{w}$ in the direction of fastest decrease.

avg: $\frac{\partial J(w)}{\partial w} = \frac{1}{N} X^T (X\vec{w} - \vec{t})$ $[\vec{x}^{(1)} \cdots \vec{x}^{(N)}]_{D \times N} \times [\vec{y} - t]_{N \times 1}$

$J(w) = \frac{1}{2N} \|X\vec{w} - \vec{t}\|_2 = \frac{1}{N} \sum_{i=1}^{N} \vec{x}^{(i)} \cdot (y^{(i)} - t^{(i)}) \Rightarrow \sum \vec{x}^{(i)}_{D \times 1} \times [y^{(i)} - t^{(i)}]_{1 \times 1}$

(sum of d 1/T d dimensional input vector) $d \times 1$

Why? · more efficient in high dimensional space, matrix inversion $O(D^3)$
· Gradient descent update $O(ND)$ or less with stochastic ··
· easier to implement · applicable to a much broader set of models

Feature mapping: (model nonlinear relationship)
map input features to another space $\psi(x): \mathbb{R}^D \to \mathbb{R}^d$ linear
Polynomial Feature mapping: $y = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{i=0}^{M} w_i x^i$
Feature mapping: $\psi(x) = [1, x, x^2, \ldots, x^M]^T$, $y = \psi^T(x)\vec{w}$ is linear in $w_0$..
M - too small - too simple - underfit too large - too complex - overfit

Regularization: control the model complexity, improve generalization
$L^2$ penalty: as our regularizer $R(w) = \frac{1}{2} \|\vec{w}\|_2^2 = \frac{1}{2} \sum_j w_j^2$ fit data
$J_{reg}(\vec{w}) = J(\vec{w}) + \lambda R(\vec{w}) = J(\vec{w}) + \frac{\lambda}{2} \sum_j w_j^2$ tradeoff ← norm of ws
λ: large λ penalize weight values more Tune λ with validation set

$w^{Ridge} = \underset{\vec{w}}{\text{argmin}}\, J_{reg}(\vec{w}) = \underset{\vec{w}}{\text{argmin}} \frac{1}{2N} \|X\vec{w} - \vec{t}\|_2^2 + \frac{\lambda}{2} \|w\|_2^2 = (X^T X + \lambda N I)^{-1} X^T \vec{t}$

weight decay: $\vec{w} \leftarrow \vec{w} - \alpha \frac{\partial}{\partial \vec{w}} (J + \lambda R) = \vec{w} - \alpha (\frac{\partial J}{\partial w} + \lambda \frac{\partial R}{\partial w})$
$= \vec{w} - \alpha (\frac{\partial J}{\partial w} + \lambda \vec{w}) = (1 - \alpha\lambda) \vec{w} - \alpha \frac{\partial J}{\partial \vec{w}}$

∩ concave ( )
∪ convex ) (

Convex: if a function is convex, then every critical point is global optimum
a set S is convex if any line segment connecting two points in S lies entirely within S
$x_1, x_2 \in S \Rightarrow \lambda x_1 + (1-\lambda) x_2 \in S$ for $0 \le \lambda \le 1$ $\Rightarrow \lambda x_1 + \ldots + \lambda_N x_N \in S$ $(\lambda_1 + \ldots + \lambda_N = 1)$
weighted averages, convex combinations of points in S lies within $x_1, \ldots, x_N \in S$

convex function: $f((1-\lambda)x_0 + \lambda x_1) \le (1-\lambda) f(x_0) + \lambda f(x_1)$ bowl shape ∀

$Z = w^T x + b$, if loss funct is a convex func of Z ⇒ convex of w, b

J
large α, overshoot, oscillations, missing optimum, instability
small α, slow convergence
good convergence
# iteration → validation set

choose α: · Grid search
· Training curve
· changing algo (eg. line search)

Stochastic Gradient descent (batch: sum over entire dataset)
updates the parameters based on the gradient for one training example
Repeat { (1) choose example i uniformly at random
(2) perform update $\theta \leftarrow \theta - \alpha \frac{\partial L^{(i)}}{\partial \theta}$

Pros: · cost of each update is independent of N · make sign progress before seeing all the data · stocastic gradient descent is an unbiased estimate of the batch gradient given sampling each example uniformly at random

$\frac{\partial J(w)}{\partial \theta} = E_x[\frac{\partial L^{(i)}}{\partial \theta}] = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial L^{(i)}}{\partial \theta} = \frac{\partial J}{\partial \theta}$ Cons: High variance, can't exploit efficient vectorized operations

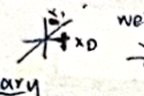
Mini-Batch Gradient descent: randomly choosen medium-sized subset of training examples M. In theory, sample examples independently and uniformly with replacement. In practice, permute the training set and then go through it sequentially. Each path over the data is called an epoch.
{ large size: more computation time; estimate accurate, exploit vectorization
small size: faster update; estimate noiser; cannot exploit vectorization
|M| - hyperparameter, |M| = 100 reasonable. α - influence noise in parameter
⇒ start with large α to get closer to optimum, gradually decrease the learning rate to ↓ fluctuation

**Binary linear classification** $z = w^T x + b$ $y = \begin{cases} 1, & \text{if } z \geq r \\ 0, & \text{if } z < r \end{cases}$

predict a discrete-valued target given a D-dimension input $x \in \mathbb{R}^D$

Simplified: $z = w^T x$, $y = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$ ($w^T x + b \geq r$ b-r add $x_0 = 1$ $w_0 = b$)

NOT: $x_0$ $x_1$ $t$ : 1 0 1; 1 1 0 — $z = w_0 x_0 + w_1 x_1$ $\begin{cases} w_0 \geq 0 \\ w_0 + w_1 < 0 \end{cases}$




The line $\{x: w^T x = 0\}$ defines the decision boundary

A line in 2D, or a hyperplane in high dimensions. pass through origin.

Data is linearly separable if a linear decision rule can perfectly seperate the training examples.

Weight space: each training example specifies a half space that w must lie in to guarantee correct classification

Feasible region satisfy all the constraints.

The problem is feasible if the feasible region is non empty.

Data space: points are training examples, a line represents a set of weights.

Weight space: points are weights, a half space represents a training example constraining the weights

Learn weights for linearly separable dataset { ① Linear programming ② perceptron algorithm

**Logistic Regression**: if data not linearly seperable • define loss func • find weights that minimize the average loss over the training examples.

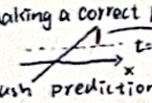
**0-1 Loss**: $L_{0-1}(y,t) = \begin{cases} 0, & \text{if } y = t \\ 1, & \text{if } y \neq t \end{cases} = \mathbb{I}[y \neq t]$ $\text{cost} J = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}[y^{(i)} \neq t^{(i)}]$

cons: • to mini loss, we need critical point, but $\nabla$ is zero almost everywhere $\Rightarrow \Delta w$ has no effect on the loss • 0-1 loss is discontinuous at $z=0$, $\nabla$ is undefined.

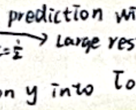
**Squared loss**: $L_{SE}(z,t) = \frac{1}{2}(z-t)^2$. Treat binary targets as continuous values

• Make final prediction y by thresholding z at $\frac{1}{2}$.

cons: if $t=1$, $L(z=10) > L(z=0) \Rightarrow$ Making a correct prediction with high confidence should be good, but incur a large loss.




Large residual




**Logistic Activation function**: squash prediction y into $[0,1]$

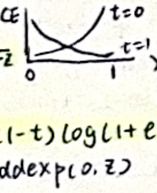
$z = w^T x$ $y = \sigma(z)$ $L_{SE}(y,t) = \frac{1}{2}(y-t)^2$ $\sigma(z) = \frac{1}{1+e^{-z}}$ (activation func)

cons: suppose $t=1$ and z is very negative $\Rightarrow$ $y \approx 0$ is very wrong, but the weights appears to be at the critical point: $\frac{\partial L}{\partial w_j} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial w_j} = 0 \cdot \frac{\partial z}{\partial w_j} = 0$

**Cross Entropy loss**: intepret $y \in [0,1]$ as the prob. $t=1$

• Heavily penalize the extreme mis-classification cases

$L_{CE} = -t \log y - (1-t) \log(1-y)$ $z = w^T x$ $y = \sigma(z) = \frac{1}{1+e^{-z}}$




assume $t=1$

Numerical Instability: $t=1$, $z \ll 0$, $y \to 0$ $\log y \to \log 0$

**Logistic Cross Entropy**: $L_{LCE}(z,t) = t \log(1+e^{-z}) + (1-t)\log(1+e^{-z})$

$E = t^* * \text{np.logaddexp}(0, -z) + (1-t) * \text{np.logaddexp}(0, z)$

**Gradient of logistic loss**: $\frac{\partial L_{CE}}{\partial w_j} = \frac{\partial L_{CE}}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{w_j} = (-\frac{t}{y} + \frac{1-t}{1-y}) y(1-y) x_j = (y-t) x_j$

$w_j \leftarrow w_j - \alpha \frac{\partial J}{\partial w_j} = w_j - \frac{\alpha}{N} \sum_{i=1}^{N} (y^{(i)} - t^{(i)}) x_j^{(i)}$ generalized linear model

**Gradient Checking with finite difference**

$$\frac{\partial}{\partial x_i} f(x_1, \dots, x_N) = \lim_{h \to 0} \frac{f(x_1, x_2, \dots, x_i + h, \dots, x_N) - f(x_1, x_2, \dots, x_i - h, \dots, x_N)}{2h}$$

eg. $x_1 = 3$ $x_2 = 7$ $h = 0.01$: $\frac{\partial f}{\partial x_2} \approx \frac{f(3, 7.01) - f(3, 6.99)}{0.02}$

• Run gradient checks on small, randomly chosen inputs.
• use double precision floats • plug in small value of h eg $10^{-10}$
• compute relative error: $\frac{|a-b|}{(|a|+|b|)}$ (should be very small $10^{-6}$) a: finite difference b: derivitive by your implement

**Hypothesis**: is a function $f: X \to T$ from the input to target space

**Hypothesis space** H as a set of hypothesis (eg a set of linear functions)

**Inductive bias**: preference for some eg of H over others better than others.

**No free lunch**: if dataset / problem were not naturally biased $\Rightarrow$ no ML would be better than others.

Parametric Algo = H is defined using a finite set of parameters eg DT

non parametric = hypo space H is defined in terms of the data eg KNN. no parameters defined in terms of data.

eg linearity, continuity simplicity.

**Multi-class classification** • Targets form a discrete set $\{1, \dots, K\}$

• represent targets as one-hot vectors or one of k encoding. $\vec{t} = (0, \dots, 0, 1, \dots, 0) \in \mathbb{R}^K$

Vectorized form: $\vec{z} = W\vec{x} + \vec{b}$ or $z = W\vec{x}$ (with $x_0 = 1$)

Non vectorized form: $z_k = \sum_{j=1}^{D} w_{kj} x_j + b_k$ for $k = 1, 2, \dots, K$

$W: K \times D$ $x: D \times 1$ $b: K \times 1$ $z: K \times 1$

eg. $D=2$ $K=4$ $W = \begin{bmatrix} -w_1- \\ -w_2- \\ -w_3- \\ -w_4- \end{bmatrix}_{4 \times 2}$ $\vec{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$ $\vec{z} = W\vec{x} + \vec{b}$ = 

the believe that model has class k

$z_k$: how much the model prefers the kth prediction. $w_i^T x + b_i$

$y_i = \begin{cases} 1, & \text{if } i = \arg\max_k z_k \\ 0, & \text{otherwise} \end{cases}$ ↓

**Softmax Regression** $y_k = \text{softmax}(z_1, \dots, z_K)_k = \frac{e^{z_k}}{\sum_{k'} e^{z_{k'}}}$

• $\sum y_k = 1$ • $z_k$ are called logits • $y_k$ as probability • If $z_k$ is much larger than the others, then $\text{softmax}(\vec{z})_k \approx 1$ and it behaves like arg max

**Cross Entropy Loss** $L_{CE}(y,t) = -\sum_{k=1}^{K} t_k \log y_k = -\vec{t}^T (\log \vec{y})$ to loss.

$t^T = [0\ 1\ 0\ 0]$ $\vec{y}^T = [0.01\ 0.9\ 0.05\ 0.04]$ only one output contribute

**Gradient descent for softmax** $\frac{\partial L_{CE}}{\partial w_k} = \frac{\partial L_{CE}}{\partial z_k} \cdot \frac{\partial z_k}{\partial w_k} = (y_k - t_k)\vec{x}$

$w_k \leftarrow w_k - \alpha \cdot \frac{1}{N} \cdot \sum_{i=1}^{N} (y_k^{(i)} - t_k^{(i)}) \vec{x}^{(i)}$

**model performance**: • **Progress** = training curve (# of iteration vs error)

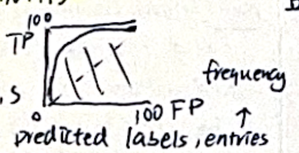
• **Accuracy** (average 0-1 loss, error rate) $\text{Acc} = \frac{TP+TN}{P+N}$

highly sensitive to class imbalance =

| pred. \ true class | 0 | 1 |
|---|---|---|
| 0 | TN | FN II |
| 1 | FP I | TP |

**Sensitivity** = $TP/(TP+FN)$ **Specificity** = $TN/(TN+FP)$

**Receiver Operating Characteristic (ROC) Curve**:




frequency ↑

Area under curve can quantify if a binary classifier achieves a good tradeoff between s.s

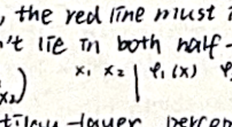
Confusion matrix: rows are true label colare predicted labels, entries

**Limits of linear classification**: some dataset are not linearly seperable.

(Prove seperable): each training point → linear rule → inequality constraint in w space → feasible region. (Prove not seperable): Proof by contradiction

Half space are convex: if two points lie in the same space, the line segment connecting them also lie in the same space. Half. Suppose the problem is feasible If positive examples are in the positive half-space, then the green line segment must be as well. Similarly, the red line must in the negative half-space But the intersection can't lie in both half-space ⊖ contradiction! ✗

**Feature map**: $\varphi(x) = \begin{pmatrix} x_1 \\ x_2 \\ x_1 x_2 \end{pmatrix}$



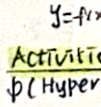

**Neural Network**: multi-layer perceptrons: consists fully connected layer

**Fully connected layer**: all input units are connected to all output units

Every layer i: weight matrix $N_i \times N_{i-1}$, connect $N_{i-1}$ input $N_i$ output units.

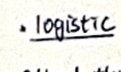
$y = f(x) = \phi(Wx + b)$ $\phi$: applied component wise (non linearity)

**Activation function** (Hyper para) = **Identity** $y = z$ 

, **Rectified Linear Unit (ReLU)** $y = \max(0, z)$ 

• **Soft ReLU** $y = \log(1 + e^z)$ 

• **logistic** $y = 1/(1+e^{-z})$ 

• **Hyperbolic** $y = \frac{e^z - e^{-z}}{e^z + e^{-z}}$ 

• **Hard threshold** $y = \begin{cases} 1, & \text{if } z > 0 \\ 0, & \text{if } z \leq 0 \end{cases}$ 

**Compute of each layer**: last layer < regression $y = f^{(L)}(h^{(L-1)}) = (W^{(L)} h^{(L-1)} + b^{(L)})$

binary $y = f^{(L)}(h^{(L-1)}) = \sigma(W^{(L)} h^{(L-1)} + b^{(L)})$

$y = f^{(L)} \circ f \circ \dots \circ f^{(1)}(x)$ composition

**Expressivity**: two model A, B. (HA HB) => A is more expressive can represent any fun in $H_B$

Note: **Linear** NN $y = W^{(3)} W^{(2)} W^{(1)} \vec{x}$ regression.

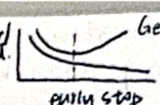
can only represent linear functions, no more expressive than Linear

**Non Linear NN** with non linear activation functions. **Universal func Approximator**: can approximate any function arbitrarily well

For any $f: X \to T$, there is a sequence of $f_i \in H$ with $f_i \to f$.

**Binary Input & Targets**: $2^D$ hidden units, each of which responds to one particular input configuration, require only 1 hidden layer

**Early stopping** to avoid overfitting don't add explicit $R(\theta)$ to cost

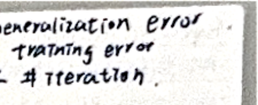



$z = f(g(x))$ $\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x}$ **univariate chain rule.**

**Error Signal** $\bar{y} = dL/dy$, just a values our prog computing

**Multivariate chain rule** $\frac{d}{dt} f(x(t), y(t)) = \frac{\partial f}{\partial x} \frac{\partial x}{\partial t} + \frac{\partial f}{\partial y} \frac{\partial y}{\partial t}$

$\bar{t} = \bar{x} \frac{\partial x}{\partial t} + \bar{y} \frac{\partial y}{\partial t}$




(parents before child)

**Back Propagation** =

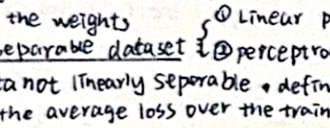
let $V_1, \dots, V_N$ be a topological ordering of the computation graph

↳ node, we try to compute gradients.

• **Forward pass**: For $i = 1, \dots, N$, compute $V_i$ as a function of Parent($V_i$)

• **Backward Pass**: For $i = N-1, \dots, 1$ $\bar{V}_i = \sum_{j \in \text{children}(V_i)} \bar{V}_j \frac{\partial V_j}{\partial V_i}$

**Regularized logistic least squares**:




Forward: $z = wx + b$, $y = \sigma(z)$, $L = \frac{1}{2}(y-t)^2$, $R = \frac{1}{2}w^2$, $L_{reg} = L + \lambda R$

Backward: $\bar{L}_{reg} = 1$, $\bar{R} = \bar{L}_{reg} \frac{dL_{reg}}{dR} = \bar{L}_{reg} \cdot \lambda$, $\bar{L} = \bar{L}_{reg} \frac{\partial L_{reg}}{\partial L} = \bar{L}_{reg}$, $\bar{y} = \bar{L} \cdot \frac{dL}{dy} = \bar{L}(y-t)$, $\bar{z} = \bar{y} \frac{dy}{dz} = \bar{y}\sigma'(z)$, $\bar{w} = \bar{z} \frac{\partial z}{\partial w} + \bar{R} \frac{dR}{dw} = \bar{z} \cdot x + \bar{R} w$, $\bar{b} = \bar{z} \frac{dz}{db} = \bar{z}$

**BP for two layer NN**:




Forward: $z_i = \sum_j w_{ij}^{(1)} x_j + b_i^{(1)}$, $h_i = \delta(z_i)$, $y_k = \sum_i w_{ki}^{(2)} h_i + b_k^{(2)}$, $L = \frac{1}{2} \sum_k (y_k - t_k)^2$

Back: $\bar{L} = 1$, $\bar{y}_k = \bar{L}(y_k - t_k)$, $\bar{W}_{ki}^{(2)} = \bar{y}_k h_i$, $\bar{b}_k^{(2)} = \bar{y}_k$, $\bar{h}_i = \sum_k \bar{y}_k W_{ki}^{(2)}$, $\bar{z}_i = \bar{h}_i \sigma'(z_i)$, $\bar{W}_{ij}^{(1)} = \bar{z}_i x_j$, $\bar{b}_i^{(1)} = \bar{z}_i$

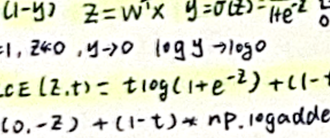
Forward: one add-multiply op per weight

back: two add-multiply op per weight

• linear in # of layer

• quadratic in units per layer

**vectorized** =




Forward: $z = W^{(1)} x + b^{(1)}$, $h = \sigma(z)$, $y = W^{(2)} h + b^{(2)}$, $L = \frac{1}{2}\|t - y\|^2$

$\bar{L} = 1$, $\bar{y} = \bar{L}(y-t)$, $\bar{W}^{(2)}_{2\times K} = \bar{y}_{2\times D} h^T_{D\times K}$, $\bar{b}^{(2)} = \bar{y}$, $\bar{h} = W^{(2)T} \bar{y}$, $\bar{z} = \bar{h} \circ \sigma'(z)$, $\bar{W}^{(1)} = \bar{z} x^T$, $\bar{b}^{(1)} = \bar{z}$

should not set all $W = 0$ initial!

neurally implausible.

**Invariant**: image transformed slightly $\Rightarrow$ classification unchange

convolution 1D: $(x * w)[t] = \sum_{\tau=0}^{K-1} x[t-\tau] w[\tau]$

• commutativity • linearly

2D: $(X * w)[i, j] = \sum_s \sum_t X[i-s, j-t] * w[s, t]$

**max pooling**: $y_i = \max_{j \text{ in pooling group}} z_j$ reduce size

$(A+B)^T = A^T + B^T$

$(rA)^T = rA^T$

$(AB)^T = B^T A^T$

$I^T = I$

$AA^{-1} = A^{-1}A = I$

$(rA)^{-1} = \frac{1}{r} A^{-1}$

$(A^T)^{-1} = (A^{-1})^T$

$(A^{-1})^{-1} = A$

$(AB)^{-1} = B^{-1}A^{-1}$

$I^{-1} = I$

$f(w) = w^T A w + b^T w + \gamma$

$\nabla f = \frac{1}{2}(A^T + A)w + b$

$= Aw + b$ (if A sym)

$f = w^T A w$ $\nabla^2 f = A + A^T = 2A$

$\frac{\partial x^T a}{\partial x} = \frac{\partial a^T x}{\partial x} = a$ $\frac{\partial a^T X b}{\partial X} = ab^T$ $\frac{\partial x^T B x}{\partial x} = (B + B^T)x$

$\frac{\partial (\vec{x} - As)^T W (x - As)}{\partial s} = -2A^T W(x - As)$

$\frac{\partial}{\partial x} = 2w(x - As)$

$\frac{\partial}{\partial A} = -2w(x - As)s^T$