General Knowledge

- **Supervised Learning:**
  - Classification (Labeled Dataset)
- **Unsupervised Learning:**
  - Clustering (Unlabeled Dataset)
  - Only given set of measurements, no classes
- **Data Cleaning:**
  - Preprocess data to reduce noise
  - handle missing values
- **Relevance Analysis:** Remove irrelevant or redundant attributes
- **Data Transformation:** Generalize and/or normalize data

Evaluating Classification Methods

- Prediction Accuracy
- Speed (training speed), scalability (inference speed)
- Robustness: ability to handle noise and missing values
- Interpretability
- Goodness of rules: compactness of classification rules (e.g. DT size)

Decision Tree (DT)

Two Stages

1. Tree construction
2. Tree pruning (rm branch that reflects outliers and noise)

Tree Construction

ID3/C4.5: **information gain**

- Attributes assumed categorical
- Can be modified for continuous-valued attributes

IBM IntelligentMiner: **gini index**

- Attributes assumed continuous
- Need other tools to get possible split values
- Can be modified for categorical attributes

Information Gain

Definition:

- How much did information entropy **decrease by**?
- Information Entropy $\equiv$ impurity
- $\text{Gain}(D, a)$: Information gain if we were to split samples D by attribute a into V subsets $\{D_1, D_2, ..., D_V\}$.

Equations

Information : $I(x_i) = -\log_2(p(x_i))$

$$\begin{aligned}\text{Information Entropy} : \text{Ent}(X) &= E(I(X)) \\ &= \textstyle\sum_{i=1}^{n} p(x_i) I(x_i) \\ &= \textstyle\sum_{i=1}^{n} p(x_i)(-\log_2(p(x_i))) \\ &= -\textstyle\sum_{i=1}^{n} p(x_i)(\log_2(p(x_i)))\end{aligned}$$

Information Gain : $\text{Gain}(D, a) = \text{Ent}(D) - \sum_{v=1}^{V} \frac{|D_v|}{|D|} \text{Ent}(D_v)$

Impurity After Split : $-\sum_{v=1}^{V} \frac{|D_v|}{|D|} \sum_{x_i \in D_v} p(x_i) \log_2 p(x_i)$

Everything Together:

$$\begin{aligned}&\text{Gain}(D, a) = \\ &= \text{Ent}(D) - \sum_{v=1}^{V} \frac{|D_v|}{|D|} \text{Ent}(D_v) \\ &= -\left(\sum_{i=1}^{n} p(x_i) \log p(x_i) - \sum_{v=1}^{V} \frac{|D_v|}{|D|} \sum_{x_i \in D_v} p(x_i) \log p(x_i)\right)\end{aligned}$$

- For categorical values V **is the number of distinct values of** a
- $|D_v|$ is the size v'th split. $|D|$ is the size of the entire dataset

Tree Pruning & Enhancements

- Pre-pruning:
  - Halt tree construction early
  - Halt before *goodness measure* falls below certain threshold.
  - (hard to choose appropriate threshold)
- Post-pruning:
  - Removes branches from a "fully grown" tree.
  - Progressively remove branches.
  - Use testing set to decide the best pruned tree.

Other Enhancements

Continuous-valued Attributes:

- Treat each unique continuous attribute value in the dataset as discrete and identify all possible split points between them.
- e.g. [21.6, 22.0, 23.0]. Try splits at: [21.8, 22.5]
- Time Complexity: $O(N)$ after sort. $O(N \log N)$ in total.

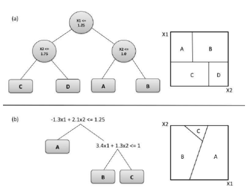
Missing Values:

- Method 1: Assign most common value of that attribute
- Method 2: Assign probability to each of the possible values

Attribute Construction:

- Create new attribute on existing ones (e.g., spare ones)
- Purpose: Reduce fragmentation, repetition, replication

Non-axis Parallel Split



Divide & Conquer

Internal Decision Nodes:

- Univariate: Use single attribute a_i
  - Numeric: Binary split
  - Discrete n-way split, for all n possible values
- Multivariate: use all attributes $a_i \in A$

Leaves:

- Classification: class labels
- Regression: Numeric - avg of leaf nodes / local fit (fit a more sophisticated model locally, e.g., linear regression)

Pseudocode

```
def GenerateTree(⍰):
  if NodeEntropy(⍰) < θ_I /* (See I_m on slide 29)
      Create leaf labelled by majority class in ⍰
      return
  i ← SplitAttribute(⍰)
  for each branch of x[i]:
      Find ⍰[i] falling in branch
      GenerateTree(⍰[i])

def SplitAttribute(⍰):
    MinEnt ← MAX
    for all attributes i = 1, ..., d:
        if x[i] is discrete with n values:
            Split ⍰ into ⍰[1], ..., ⍰[n] by x[i]
            e ← SplitEntropy(⍰[1], ..., ⍰[n])
            if e < MinEnt:
                MinEnt ← e
                bestf ← i
        else: # x[i] is numeric
            for all possible splits
                Split ⍰ into ⍰[1], ⍰[2] on x[i]
                e ← SplitEntropy(⍰[1], ⍰[2])
                if e < MinEnt:
                    MinEnt ← e
                    bestf ← i
    return bestf
```

Exam Tips (Rule Extraction Format)

```
R1:  IF (age > 38.5) AND (years-in-job > 2.5) THEN y = 0.8
R2:  IF (age > 38.5) AND (years-in-job ≤ 2.5) THEN y = 0.6
R3:  IF (age ≤ 38.5) AND (job-type = 'A') THEN y = 0.4
R4:  IF (age ≤ 38.5) AND (job-type = 'B') THEN y = 0.3
R5:  IF (age ≤ 38.5) AND (job-type = 'C') THEN y = 0.2
```

KNN (K-Nearest Neighbors)

Instance-Based Methods *(a.k.a memory-based)*

- Instance-based classification: Store training examples, delay processing until a new instance must be classified **("Lazy Evaluation")**

Examples:

- k-nearest neighbors: instances as points in Euclidean space.
- Case-based reasoning: symbolic representation and knowledge-based inference

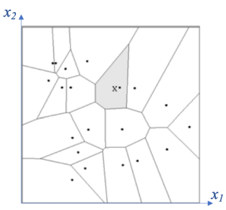
Procedure

- Time Complexity: $O(n \log k)$ via min-heap of size k

1. Compute distance to all other training records.
2. Sort the distance, and find k nearest neighbors
3. Use the k nearest neighbors to determine the class label of the unknown sample (e.g., through majority voting)

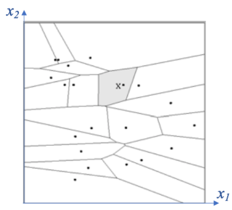
- Increasing k: Less sensitive to noise
- Decreasing k: Capture finer structure

Voronoi Tessellation



$\left(a_{x_1} - b_{x_1}\right)^2 + \left(a_{x_2} - b_{x_2}\right)^2$

No weighting on either x_1 or x_2. They have equal effect on the decision boundaries.



$\left(a_{x_1} - b_{x_1}\right)^2 + 3\left(a_{x_2} - b_{x_2}\right)^2$

Put more weighting on x_2, thus resulting decision boundary to be more sensitive towards x_2

Distance Metrics

- Continuous Attributes: Euclidean Distance
  - **normalize** each dimension by standard deviation
- Discrete Data: use hamming distance

Curse of Dimensionality

Definition: Prediction accuracy degrade quickly when number of attributes grow, because

- When many irrelevant attributes involved, relevant attributes are shadowed, and distance measurements are unreliable

Solution:

- Remove irrelevant attributes in pre-processing (e.g., dimensionality reduction such as PCA)

Neural Network

Perceptrons

Step Activation: $y = \begin{cases} 1 \text{ if } z > 0 \\ 0 \text{ otherwise} \end{cases}$

Update Rule:

$$w \leftarrow w - \alpha \frac{\partial L}{\partial w_i}$$

$$\frac{\partial L}{\partial \mathbf{w}} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial \mathbf{w}}$$

$$\because \frac{\partial z}{\partial \mathbf{w}} = \mathbf{x},$$

$$\therefore \frac{\partial L}{\partial \mathbf{w}} = \left(\frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial z}\right) \cdot \mathbf{x} = \delta \cdot \mathbf{x}$$

Where:

- δ is the error term
- L (Loss): Loss function
- α: Learning rate
- $\frac{\partial L}{\partial y}$: sensitivity of loss w.r.t. the activation output
- $\frac{\partial y}{\partial z}$: derivative of the activation function
- $\frac{\partial z}{\partial \mathbf{w}}$: since $z = \mathbf{w}^T\mathbf{x} + b$, the derivative w.r.t. to w is input vector $\mathbf{x}$

Linear Activation: *(w/ MSE Loss)*

$$\hat{y} = z = \mathbf{w}^T\mathbf{x} + b; \;\; L(y, \hat{y}) = (y - \hat{y})^2$$

$$\delta = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} = \frac{\partial L}{\partial \hat{y}} \cdot 1 = 2(y - \hat{y})$$

Sigmoid Activation *(w/ Binary Cross Entropy Loss)*:

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}$$

$$L(y, \hat{y}) = -(y \cdot \log \hat{y} + (1 - y)\log(1 - \hat{y}))$$

Derivatives:

$$\text{Loss} : \frac{\partial L}{\partial \hat{y}} = -\left(\frac{y}{\hat{y}} - \frac{1 - y}{1 - \hat{y}}\right) = \frac{\hat{y} - y}{\hat{y}(1 - \hat{y})}$$

$$\text{Activation} : \frac{\partial \hat{y}}{\partial z} = \hat{y}(1 - \hat{y}) \text{ or } \sigma(z)(1 - \sigma(z))$$

$$\text{Error Term} : \delta = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} = \frac{\hat{y} - y}{\hat{y}(1 - \hat{y})} \cdot \hat{y}(1 - \hat{y})$$

$$= \hat{y} - y \text{ or } \sigma(z) - y$$

Weight Update *(everything together)*:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha(\sigma(z) - y) \cdot \mathbf{x}$$

Softmax Activation *(w/ Cross Entropy Loss)*:

There are C inputs (logits): $\{z_1, z_2, ..., z_C\}$
with K output probability (activations): $\{y_1, y_2, ..., y_C\}$

$$\hat{y}_i = \sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}, \quad \forall i \in \{1, 2, ..., C\}$$

$$L(\hat{y}_i, y_i) = \begin{cases} -\log \hat{y}_i & \text{if } y_i = \text{True} \\ 0 & \text{if } y_i = \text{False} \end{cases}$$

Derivatives:

$$\text{Loss} : \frac{\partial L}{\partial \hat{y}_i} = -\frac{y_i}{\hat{y}_i}$$

$$\text{Activation} : \frac{\partial \hat{y}_i}{\partial z_j} = \hat{y}_i(\Delta_{ij} - \hat{y}_j)$$

$$\text{where} \quad \Delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

$$\begin{aligned}\text{Error Term} : \delta_j &= \frac{\partial L}{\partial z_j} \\ &= \sum_{i=1}^{C} \frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial y_i}{\partial z_j} \\ &= \sum_{i=1}^{C} \left(-\frac{y_i}{\hat{y}_i}\right)(\hat{y}_i(\Delta_{ij} - \hat{y}_j)) \\ &= \sum_{i=1}^{C} -y_i \cdot (\Delta_{ij} - \hat{y}_j) \\ &= -\sum_{i=1}^{C} y_i \cdot (\Delta_{ij} - \hat{y}_j) \\ &= -\sum_{i=1}^{C} y_i \Delta_{ij} + \sum_{i=1}^{C} y_i \hat{y}_j\end{aligned}$$

$$\text{1st Term} : \sum_{i=1}^{C} y_i \Delta_{ij} = y_j$$

$$\text{2nd Term} : \sum_{i=1}^{C} y_i \hat{y}_j = \hat{y}_j \sum_{i=1}^{C} y_i = \hat{y}_j,$$

y is one-hot so sum to 1

$$\therefore \boldsymbol{\delta_j = \hat{y}_j - y_j}$$

Weight Update *(everything together)*:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha(\hat{y}_j - y_j) \cdot \mathbf{x}$$

Linear Separability

Structure	Types of Decision Regions	Exclusive-OR Problem	Classes with Meshed regions	Most General Region Shapes
Single-Layer	Half Plane Bounded By Hyperplane	A B B A	B A	
Two-Layer	Convex Open Or Closed Regions	A B B A	B A	
Three-Layer	Abitrary (Complexity Limited by No. of Nodes)	A B B A	B A	

Training

- Momentum: $\Delta w^t = -\alpha \frac{\partial L}{\partial w} + \beta \Delta w^{t-1}$, where β is momentum rate
- Adaptive learning rate: Increasing α if L decreasing for a long time, else lower learning rate α

- Number of weights: $h(d+1) + (h+1)k$
- Space Complexity: $O(h \cdot (d+k))$
- Time complexity: $O(e \cdot h \cdot (d+k))$

Where,

- d is number of inputs
- h is number hidden units
- k is number of outputs
- e is number of training steps

Iteration Over Data

- Batch Gradient Descent:
  - Entire dataset each step
  - avg error to determine gradient
- Stochastic Gradient Descent (SGD)
  - Pick just one sample at every step
  - Update gradient only based on that single record
- Mini-batch Gradient Descent
  - Pick batch size of $k \ll N$,
  - where N is the total size of the dataset

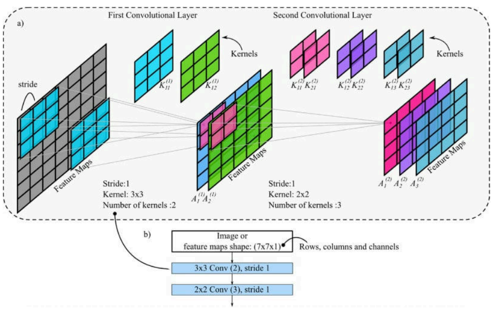
CNN (Convolution Neural Network)

Motivations

Less learnable parameters. Better to learn small models.

- Sparse connection
- Parameter sharing

Design



- Output: Binary, Multinomial, Continuous, Count
- Input: fixed size, can use padding to make image same size
- Architecture: choice requires experimentation
- Optimization: Backprop

Layers

- Convolution Layer
- Pooling Layer
- Fully-connected (FC) Layer
- Input and output

Activation

ReLU (Rectified Linear Unit):

$$\mathrm{ReLU}(x) = \max(0, x)$$

Motivations:

- Outputs 0 for negative values, introducing sparse representation
- Does not face vanishing gradient as with sigmoid or tanh
- Fast: does not need e^x computation

Conv Layer

$$n_{\text{out}} = \left\lfloor \frac{n_{\text{in}} + 2p - k}{s} \right\rfloor + 1$$
$$= \left\lceil \frac{n_{\text{in}} + 2p - k + 1}{s} \right\rceil$$

where,

- n_{in}: number of input features
- n_{out}: number of output features
- k: convolution kernel size
- p: padding size (on one side)
- s: convolution stride size

Training

Supervised Pre-training

- Use labeled data, work bottom up:
  - train hidden layer 1. Fix param
  - train hidden layer 2. Fix param
  - ...
- Use labeled data, supervised finetune
  - Train entire network
  - Finetune to final task

Training w/ Max-pooling

Max-pooling

$$y_{11} = \max(x_{11}, x_{12}, x_{21}, x_{22})$$

Derivative

For values of input feature map $x_i \in \{x_1, x_2, ..., x_n\}$, where x_i is amongst the inputs of $y_j \in \{y_1, y_2, ..., y_m\}$

In simpler terms:

- x is input before max-pooling,
- y is output after max-pooling,
- $m < n$ for obvious reasons
- x_i can be inputs of multiple y

Then we have,

$$\frac{\partial L}{\partial x_i} = \sum_{y_j \in \substack{\text{all max-pooling} \\ \text{windows convering } x_i}} \frac{\partial L}{\partial y_j} \cdot \frac{\partial y_j}{\partial x_i}$$

The core idea is that,

$$\frac{\partial L}{\partial x_i} = \begin{cases} \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial x_i} & \text{if } x_i = y \\ 0 & \text{otherwise} \end{cases}$$

Regularization

Generalization Error: $\mathcal{L}(h) = E_{(x,y)\sim P(x,y)}[f(h(x), y)]$, where $h(x)$ is the learned model. It shows the expected error (using error function f) on the true distribution ($E_{(x,y)\sim P(x,y)}$)

Bias-Variance Tradeoff

The following shows that the **expected** MSE error of our predictor $\hat{f}$ on a single data point x can be explained by bias, variance, and irreducible terms.

$$E\left[\left(Y - \hat{f}(x)\right)^2\right] = E\left[\left(f(x) + \varepsilon - \hat{f}(x)\right)^2\right]$$
$$= E\left[\left(f(x) - \hat{f}(x)\right)^2\right] + 2E\left[\left(f(x) - \hat{f}(x)\right)\varepsilon\right] + E\left[\varepsilon^2\right]$$
$$= \underbrace{\mathrm{Bias}\left[\hat{f}(x)\right]^2}_{\mathrm{Bias}^2} + \underbrace{\mathrm{Var}\left[\hat{f}(x)\right]}_{\text{Variance}} + \underbrace{\sigma^2}_{\substack{\text{Irreducible Error} \\ \text{(Inherent Noise in data)}}}$$

Detailed breakdown of each term:

$$2E\left[\left(f(x) - \hat{f}(x)\right)\varepsilon\right] = 2\left(f(x) - E\left[\hat{f}(x)\right]\right)E[\varepsilon] = 0$$

$E[\varepsilon^2] = \sigma^2$ (**Note**: $E(\varepsilon) = 0$, $\mathrm{Var}(\varepsilon) = \sigma^2$, where σ is SD of noise)

$$E\left[\left(f(x) - \hat{f}(x)\right)^2\right] = E\left[\left(f(x) - E\left[\hat{f}(x)\right] + E\left[\hat{f}(x)\right] - \hat{f}(x)\right)^2\right]$$
$$= E\left[\left(f(x) - E\left[\hat{f}(x)\right]\right)^2\right] +$$
$$2E\left[\left(f(x) - E\left[\hat{f}(x)\right]\right)\left(E\left[\hat{f}(x)\right] - \hat{f}(x)\right)\right] +$$
$$E\left[\left(E\left[\hat{f}(x)\right] - \hat{f}(x)\right)^2\right]$$

The **1st** term can be turn into **Bias**. Since $f(x)$ is not a random variable, but a fixed deterministic function of x, hence $E[f(x)] = f(x)$.

$$E\left[\left(f(x) - E\left[\hat{f}(x)\right]\right)^2\right] = E[f(x)^2] - 2E\left[f(x)E\left[\hat{f}(x)\right]\right] + E\left[E\left[\hat{f}(x)\right]^2\right]$$
$$= f(x)^2 - 2f(x)E\left[\hat{f}(x)\right] + E\left[\hat{f}(x)\right]^2$$
$$= \left(f(x) - E\left[\hat{f}(x)\right]\right)^2$$
$$= \mathrm{Bias}\left[\hat{f}(x)\right]^2$$

The **2nd** term can be reduced to 0. Remember that $f(x)$ is a constant and can be taken out of $E[...]$

$$2E\left[\left(f(x) - E\left[\hat{f}(x)\right]\right)\left(E\left[\hat{f}(x)\right] - \hat{f}(x)\right)\right]$$
$$= E\left[f(x)E\left[\hat{f}(x)\right] - f(x)\hat{f}(x) - E\left[\hat{f}(x)\right]^2 + E\left[\hat{f}(x)\right]\hat{f}(x)\right]$$
$$= f(x)E\left[\hat{f}(x)\right] - f(x)E\left[\hat{f}(x)\right] - E\left[\hat{f}(x)\right]^2 + E\left[\hat{f}(x)\right]^2 = 0$$

And the **3rd** term is directly **Variance**:

$$E\left[\left(E\left[\hat{f}(x)\right] - \hat{f}(x)\right)^2\right] = \mathrm{Var}\left[\hat{f}(x)\right]$$

- Underfitting: high bias, low variance. High train & test error
- Overfitting: low bias, high variance. Low train error. High test error

Regularization

Norm

Add regularization term to the original loss function $\lambda \sum \|w\|^2$. This encourages smaller model weights.

- L_1 norm: sum of absolute weights $\|w\|^1 = \sum_i |w_i|$
- L_2 norm: sum of squared weights $\|w\|^2 = \sqrt{\sum_i |w_i|^2}$
- L_p norm: $\|w\|^p = \sqrt[p]{\sum_i |w_i|^p}$

- Squared weight penalize large values more
- Absolute weight penalize smaller values more
- In general, smaller values of p (< 2) encourages sparser vectors. Larger values of p discourage large weights.

Dropout

Has a p value. (Suggested: 0.5 for hidden, and 0.8 for input). The number of ways to drop out for n nodes: $\binom{n}{1} + \binom{n}{2} + ... + \binom{n}{n}$

Conceptually:

1. Seen as **bagged ensemble** of exponentially many neural networks.
   - The NN is the entire ensemble.
   - Each sub-network formed by dropout is a base model
2. Simulates **sparse activation**, encouraging spare representation

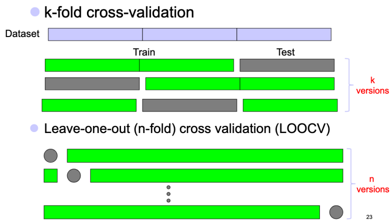
Early Stop

Stop training before overfitting occurs

Data Augmentation

Artificially create more training data (e.g., rotation, trim, masking)

Cross Validation



Notes

Three kinds of error

- Inherent: unavoidable
- Bias: due to over-simplifications
- Var: inability to perfectly estimate parameters from limited data

How to reduce variance?

- Choose a simpler classifier (like regularizer)
- Get more training data (like data augmentation)
- Cross-validate the parameters

Ensemble Methods

Advantages: Improved accuracy; other types of classifiers can be directly included; Easy to implement; Not too much parameter tuning
Disadvantage: Black box; Not compact representation.

Who Minimize Variance:

- Bagging
- Random Forest

Who Minimize Bias:

- Boosting
- Ensemble Selection

Proof

Consider binary classification problem $y \in \{-1, +1\}$, with ground truth function f, base classifier h_i, and base classifier error rate ε:

- Base Classifier Error Rate: $P\left(h_{i(x)} \neq f(x)\right) = \varepsilon$
- Ensemble Classifier: $H(x) = \mathrm{sign}\left(\sum_i^T h_{i(x)}\right)$

Assume independence of h_i error rates. From *Hoeffding Inequality*:

$$P(H(x) \neq f(x)) = \sum_{k=0}^{\lfloor T/2 \rfloor} \binom{T}{k} (1-\varepsilon)^k \varepsilon^{T-k}$$
$$\leq \exp\left(-\frac{1}{2}T(1-2\varepsilon)^2\right)$$

Therefore, as the number of base classifiers T increase, the probability of misclassification decrease exponentially

Bagging

Bagging = **B**ootstrap **Agg**regation

Traditional Bagging

1. Takes original dataset D with N training examples
2. Creates M **different** copies $\{D_m\}_{m=1}^M$ via
   - Sampling from D with replacement
   - Each D_m has same number of examples as D
3. Train base classifiers $h_1, ..., h_M$ using $D_1, ..., D_m$
4. Average / Vote model as the final ensemble $h = \frac{1}{M}\sum_{m=1}^M h_m$

Random Forest

1. Takes original dataset D with N training examples
2. Creates M **different** copies $\{D_m\}_{m=1}^M$ via
   - Sampling from D with replacement
   - Each D_m has same number of examples as D
3. Train **decision trees** $t_1, ..., t_M$ using $D_1, ..., D_m$
   - suppose there are k features
   - **randomly sample** k' ($\leq k$) **features** at each split
   - when $k' = k$ its indifferent to traditional DTs
   - usually choose $k' = \log_2 k$ (or $\sqrt{k}$ according to Korris)
4. Average / Vote model as the final ensemble $h = \frac{1}{M}\sum_{m=1}^M h_m$

Intuition: By randomizing not just samples (step 2), but also features (step 3), we introduce even greater variety compared to *Bagging*, reducing correlation. (The entire premise of ensemble method is independence between classifiers)

Boosting

Basic Idea

$$H(x)=\sum_{t=1}^{T}\alpha_t h_t(x)$$

- Train T weak learners (through T iterations)
- Each weak learner:
  - **only** have to be slightly better than random
  - Focuses on difficult data points

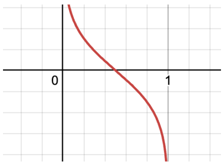
Procedure

- Initialize **weights** for the N samples.
  - $D_1=\{w_{11},\ldots w_{1N}\}$ is uniform distribution ($\forall w_{1i}=1/N$)
  - D_t means the weighted dataset at t iteration
- **For $t=1,2,\ldots,T$ do:**
  1. Train h_t on weighted dataset D_t, and compute error of ε_t

$$\varepsilon_t=\sum_{i=1}^{N}P_{x\sim D_t}(h_t(x_i)\neq y_i)=\sum_{h_t(x_i)\neq y_i}w_{ti}$$

  2. Use h_t's error rate ε_t to determine its weight α_t in the ensemble

$$\alpha_t=\frac{1}{2}\log\frac{1-\varepsilon_t}{\varepsilon_t}$$



  - if $\varepsilon_t>0.5$, then $\alpha_t<0$ and exponentially decrease.
  - if $\varepsilon_t<0.5$, then $\alpha_t>0$ and exponentially increases.

  3. Use h_t's weight α_t to determine the **next** weighted dataset D_{t+1}

$$D_{t+1}=\{w_{t+1,1},\ldots,w_{t+1,N}\}$$

$$w_{t+1,i}=\begin{cases}w_{ti}\times\exp(\alpha_t) & \text{if } h_{t(x_i)}\neq y_i \text{ (incorrect prediction)}\\ w_{ti}\times\exp(-\alpha_t) & \text{if } h_{t(x_i)}=y_i \text{ (correct prediction)}\end{cases}$$

  - Wrong $\longrightarrow w_{t+1,i}\uparrow$; correct $\longrightarrow w_{t+1,i}\downarrow$
  - Change magnitude $\propto\exp(\alpha_t)$
  - because high h_t importance α_t requires:
    - … much higher weights for misclassified samples
    - … much lower weights for correct samples
    - to balance error impact

  4. Normalize D_{t+1} by so that it sums to 1

$$w_{t+1,i}=\frac{w_{t+1,i}}{\sum_{j=1}^{N}w_{t+1,j}}$$

- Output "boosted" ensemble $H(x)=\operatorname{sign}\left(\sum_{t=1}^{T}\alpha_t h_t(x)\right)$

XGBoost

- All of the advantages of gradient boosting
- CPU parallelism by default
- Low runtime, high model performance

Boosting vs Bagging

- Bagging computationally efficient
- Both reduce variance
- Bagging can't reduce bias, but boosting can
- Bagging is better when we don't have high bias, and only want to reduce variance. (e.g., when overfitting)

Natural Language Processing (NLP)

Text Similarity

- Stop List: *irrelevant* words, despite high frequency (a, the, of, for…)
- Word Stem: syntactic variants of words (drug, drugs, drugged)
- Term freq table: T[i,j] = num. occurence of word t_i in document d_j
- Cosine distance:
  - $S(v_1,v_2)=\frac{v_1\cdot v_2}{\|v_1\|\cdot\|v_2\|}$
  - sum of element wise multiplication / product of their lengths

Bag of Words Model

- Bag of Words (BoW): unordered set of words. no positional info
- Term Frequency (tf): # of occurence of term t in document d
- Document Frequency (df): # of **documents** that contains term t
- Collection Frequency (cf): # of occur. of term t across **all** documents
- Inverse Document Frequency (idf): **Significance** of a term.
  - $\mathrm{idf}_t=\log\frac{N}{\mathrm{df}_t}$; If term t only appears in small fraction of documents, $\mathrm{idf}_t\uparrow$. If a lot of documents have term t, $\mathrm{idf}_t\downarrow$
- $\text{tf-idf}_{t,d}=\mathrm{tf}_{t,d}\times\mathrm{idf}_t$. Greatest when term t **occurs many times** within a **small fraction** of documents, v.v. ($\therefore$ low for stop words)

Vector-Space Model

- Each doc is a vector. Each dimension is **tf-idf** of a term in dictionary.
- Issue: too many terms, too many dimensions of tf-idf

Dimensionality Reduction

Latent Semantic Analysis (LSA):

- Use SVD, decompose **term-document matrix $\rightarrow$ term-topic matrix $\times$ topic-document matrix**
- Low rank approximation of document-term matrix that is **loseless**
- As 3-layer FC-NN: [Layer 1 Terms] [Layer 2 Topics] [Layer 3 Docs]

Word2Vec

- Word Similarity = Word Vector (Embedding) Similarity
- Continuos BoW: Predict mid word from surrounding (fixed window)
- Skip-gram: Predict surrounding from mid word. (Difficult but scales)

Clustering

Distance Metrics

- Continuous Variable: $L_1, L_2, \ldots, L_p$ norm
- Binary Variable: $|0-1|+|1-0|+|1-1|\ldots$
- Categorical: 1. One-hot encoding, 2. Simple matching: (total features − matched features)/ total features

K-Means

Strength: 1. efficient $O(tkn)$, t iterations, k clusters, n samples
Weakness: local optimum. Not good for non-convex shapes, categorical features, and noisy data (outliers).

$$\text{Loss}=\sum_{i=1}^{k}\sum_{x_j\in C_i}(x_j-m_i)^2$$

$$\frac{\partial\,\text{Loss}}{\partial m_i}=\sum_{i=1}^{k}\sum_{x_j\in C_i}\frac{\partial}{\partial m_i}(x_j-m_i)^2=\sum_{x_j\in c_i}=-2(x_j-m_i)$$

$$\text{Let }\frac{\partial\,\text{Loss}}{\partial m_i}=0,\quad\text{then } m_i=\frac{1}{|C_i|}\sum_{x_j\in C_i}x_j\quad\text{(the mean operation)}$$

Hierarchical Clustering *(Agglomerative - ANGES)*

1. Initialize: 1 sample per cluster $\rightarrow N$ clusters
2. **REPEAT**:
   - Merge a pair of clusters with **least distance**
   - Decrement # of clusters by one
   - **UNTIL** only 1 cluster left.

Sinle linkeage clustering

- aka. nearest neighbor (1NN) technique
- Distance measured by **closest pair** of sample from each group

Complete Linkage Clustering

- aka. furthest neighbor technique
- Distance measured by **furthest pair** of samples from each group

Centroid Linkage Clustering

- Distance measured by **mean** (centroids) of each cluster $\frac{1}{n}\sum_{i=1}^{n}x_i$

Dimensionality Reduction (DR)

Principal Component Analysis (PCA)

1. Compute covariance matrix Σ
2. Calculate eigenvalues of eigenvectors of Σ
   - Eigenvectors w/ largest eigenvalue λ_1 is 1^{st} principal component
   - Eigenvectors with k^{th} largest eigenvalue λ_k is k^{th} PC
   - Proportion of variance captured by k^{th} PC = $\lambda_k/\left(\sum_i\lambda_i\right)$

NOTES:

- PCA can be seen as noise reduction.
- Fail when data consists of multiple clusters
- Direction of max variance may not be most informative

Non-linear DR

- ISOMAP: Isometric Feature Mapping
- t-SNE: t-Sochastic Neighbor Embedding

Reinforcement Learning (RL)

Value Iteration

$V^\pi(s_0)=\mathbb{E}_\pi[\tau|s_0]$: expected reward if start at state s_0, and following policy π, forming a trajectory τ.

$$\begin{aligned}V^\pi(s_0)&=R(s_0)+\mathbb{E}[\gamma R(s_1)+\gamma^2R(s_2)+\ldots|s_0=s,\pi]\\&=R(s_0)+\mathbb{E}[\gamma V^\pi(s_1)|s_0=s,\pi]\\&=R(s_0)+\gamma\sum_{a_0\in A}\sum_{s'\in S}\pi(a|s)p(s'|s_0,a_0)V^\pi(s')\\&=R(s_0)+\gamma\sum_{s'\in S}p_{a_0\sim\pi(s_0)}(s'|s_0,a_0)V^\pi(s')\end{aligned}$$

$V^*(s)$: Optimal value function; $\pi^*(s)$: Optimal policy function

$$V^*(s)=\max_\pi V^\pi(s)=R(s)+\max_{a\in A}\gamma\sum_{s'\in S}P_{a\sim\pi^*(s)}(s'|s,a)V^*(s')$$

$$\pi^*(s)=\arg\max_a[R(s)+\gamma\cdot p(s'|s,a)V^*(s')]$$

assert $|S|<\infty, |A|<\infty$
for all $s\in S$, initialize $V(s)\leftarrow 0$
while (not converge) do {
for all $s\in S$:

$$V(s)\leftarrow R(s)+\max_{a\in A}\gamma\sum_{s'\in S}p_{s,a}(s')\,V(s')$$

}
return policy:

$$\pi(s)\leftarrow\arg\max_{a\in A}\left[R(s)+\gamma\sum_{s'\in S}p_{s,a}(s')V(s')\right]$$

Q-Function

$Q^\pi(s_0,a)=\mathbb{E}[\tau|s_0,a]$: Similar to $V^\pi(s_0)$, but stricter – It not only starts from s_0, but also select action a immediately thereafter

$$\because Q^\pi(s,a)=\sum_{s'\in S}p(s'|s,a)V^\pi(s')$$

$$\begin{aligned}\therefore V^\pi(s)&=R(s)+\gamma\sum_{a\in A}\pi(a|s)\sum_{s'\in S}p(s'|s,a)V^\pi(s')\\&=R(s)+\gamma\sum_{a\in A}\pi(a|s)\,Q^\pi(s,a)\end{aligned}$$

Exploration-Exploitation

$$\pi_{k+1}(a|s)=\begin{cases}\frac{\varepsilon}{|A|}+1-\varepsilon & a=\arg\max Q(s,a)\\ \frac{\varepsilon}{|A|} & \text{otherwise.}\end{cases}$$

- ε decays $1\rightarrow 0$. $\therefore\pi$ would start w/ uniform distribution
- as $\varepsilon\rightarrow 0$, the $\frac{\varepsilon}{|A|}$ across all actions no longer sum to 1.
- We give the remaining part $(1-\varepsilon)$ to best a (exploit)

Korris: $\pi(a|s)\propto e^{Q(s,a)/T}$. prob. of exploit exponentially $\uparrow$ as $T\downarrow$

Semi-supervised Learning

- **small set** of labelled training data
- **large set** of unlabeled training data

Cluster-based Methods

The Yarowsky Algorithm

1. Train classifier on labeled data (e.g., DT / probabilistic model)
2. Unlabeled data with **high confidence** by classifier is **labeled**
3. Repeat step 1-2 on expanded labeled set.
4. Stop when converged

Seeded K-Mean Clustering

- labeled data used for initialization (choosing cluster centers)
- labeled data not used in subsequent steps
- init: $C_h=\frac{1}{|S_h|}\sum_{x\in S_h}x$, for $h=1\ldots k$ (i.e. # of clusters)

Constrained K-Mean Clustering

- labeled data used for initialization (choosing cluster centers)
- Original labeled data unchanged. Only unlabeled are changed.

COP-Kmeans

- Weaker than partial labeling
- More compatible with data exploration

1. Given constraints: **must-link** and **cannot-link** data points
2. Cluster centers chosen randomly **without** violating constraints
3. Each step, data points reassigned to nearest cluster without violating constraints.

COP-KMEANS(data set D, must-link constraints $Con_= \subseteq D\times D$, cannot-link constraints $Con_{\neq}\subseteq D\times D$)

1. Let $C_1\ldots C_k$ be the initial cluster centers.
2. For each point d_i in D, assign it to the closest cluster C_j **such that** VIOLATE-CONSTRAINTS(d_i, C_j, $Con_=$, $Con_{\neq}$) **is false. If no such cluster exists, fail (return {})**.
3. For each cluster C_i, update its center by averaging all of the points d_j that have been assigned to it.
4. Iterate between (2) and (3) until convergence.
5. Return $\{C_1\ldots C_k\}$.

VIOLATE-CONSTRAINTS(data point d, cluster C, must-link constraints $Con_=\subseteq D\times D$, cannot-link constraints $Con_{\neq}\subseteq D\times D$)

1. For each $(d,d_=)\in Con_=$: If $d_=\notin C$, return true.
2. For each $(d,d_{\neq})\in Con_{\neq}$: If $d_{\neq}\in C$, return true.
3. Otherwise, return false.

Data-based Methods

Manifold Assumption

- Geometry affects the answer
- Assumption: Data is distributed on low-dimension manifold
- **Unlabeled Data** is used to **estimate the manifold geometry**!

Smoothness Assumption

- Decision boundary pass through regions of low data density

Appendix

Derivatives

Basic Rules

$$\frac{\partial c}{\partial x}=0$$

where c is a constant

$$\frac{\partial x^n}{\partial x}=nx^{n-1}$$

$$\frac{\partial k*f(x)}{\partial x}=k\frac{\partial f(x)}{\partial x}$$

$$\frac{\partial f(x)+g(x)}{\partial x}=\frac{\partial f(x)}{\partial x}+\frac{\partial g(x)}{\partial x}$$

$$\frac{\partial f(x)-g(x)}{\partial x}=\frac{\partial f(x)}{\partial x}-\frac{\partial g(x)}{\partial x}$$

$$\frac{\partial f(x)g(x)}{\partial x}=f(x)\frac{\partial g(x)}{\partial x}+g(x)\frac{\partial f(x)}{\partial x}$$

$$\frac{\partial\frac{f(x)}{g(x)}}{\partial x}=\frac{g(x)\frac{\partial f(x)}{\partial x}-f(x)\frac{\partial g(x)}{\partial x}}{(g(x))^2}$$

$$\frac{\partial f(g(x))}{\partial x}=\frac{\partial f(u)}{\partial u}\cdot\frac{\partial g(x)}{\partial x}$$

where u = g(x)

Derivatives of Elementary Functions

Exponential Functions

$$\frac{\partial e^x}{\partial x}=e^x$$

$$\frac{\partial a^x}{\partial x}=a^x\cdot\ln(a)$$

Logarithmic Functions

$$\frac{\partial \ln(x)}{\partial x} = \frac{1}{x}$$
$$\frac{\partial \log_{a(x)}}{\partial x} = \frac{1}{x \cdot \ln(a)}$$

Hyperbolic Functions

$$\frac{\partial \sinh(x)}{\partial x} = \cosh(x)$$
$$\frac{\partial \cosh(x)}{\partial x} = \sinh(x)$$
$$\frac{\partial \tanh(x)}{\partial x} = \operatorname{sech}^2(x)$$
$$\frac{\partial \coth(x)}{\partial x} = -\operatorname{csch}^2(x)$$
$$\frac{\partial \operatorname{sech}(x)}{\partial x} = -\operatorname{sech}(x) * \tanh(x)$$
$$\frac{\partial \operatorname{csch}(x)}{\partial x} = -\operatorname{csch}(x) * \coth(x)$$

Special Rules

$$\frac{\partial f(g(h(x)))}{\partial x} = \frac{\partial f(u)}{\partial u} * \frac{\partial g(v)}{\partial v} * \frac{\partial h(x)}{\partial x}$$

where u = g(v), v = h(x)

$$\text{if } y = f^{-1}(x), \text{then } \frac{\partial y}{\partial x} = \frac{1}{\frac{\partial f(y)}{\partial y}}$$

Derivatives of *Loss Function* ($\frac{\partial L}{\partial \hat{y}}$)

- No need to consider their batched version. For example, only need to consider $(y-\hat{y})^2$ for MSE. No need for $\frac{1}{n}\sum_{i=1}^{n}(y-\hat{y})^2$
- Because n samples $\rightarrow n$ different $\hat{y} \rightarrow n$ different gradients for each weight down the chain rule.
- For any L in the form of $\frac{1}{n}\sum f(\hat{y}_i, y_i)$, the derivative yield by each sample is $\frac{1}{n}f'(\hat{y}_i, y_i)$. Try
- So as the gradients accumulate for each weight w, they are automatically batch averaged $(1/n)$.

Mean Absolute Error (MAE)

$$L_{\text{MAE}}(y,\hat{y}) = |y-\hat{y}|$$
$$\frac{\partial L}{\partial \hat{y}} = \frac{\partial}{\partial \hat{y}}|y-\hat{y}| = \begin{cases} 1 & \text{if } y-\hat{y} > 0 \\ -1 & \text{if } y-\hat{y} < 0 \\ \text{undefined} & \text{if } y-\hat{y} = 0 \end{cases}$$

Mean Squared Error (MSE)

$$L_{\text{MSE}}(y,\hat{y}) = (y-\hat{y})^2 \qquad \frac{\partial L}{\partial \hat{y}} = 2(\hat{y}-y)$$

Root Mean Squared Error (RMSE)

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y-\hat{y})^2}$$
$$\begin{aligned}\frac{\partial\ \text{RMSE}}{\partial \hat{y}_j} &= \frac{1}{2}\left(\frac{1}{n}\sum_{i=1}^{n}(y-\hat{y})^2\right)^{-1/2} \cdot \frac{\partial}{\partial \hat{y}_j}\left(\frac{1}{n}\sum_{i=1}^{n}(y-\hat{y})^2\right) \\ &= \frac{1}{2}\left(\frac{1}{n}\sum_{i=1}^{n}(y-\hat{y})^2\right)^{-1/2} \cdot \frac{2(y_j-\hat{y}_j)}{n} \\ &= \frac{1}{2}\frac{1}{\sqrt{\text{MSE}}} \cdot \frac{2(y_j-\hat{y}_j)}{n} \\ &= \frac{\hat{y}_j - y_j}{n \cdot \text{RMSE}}\end{aligned}$$

Huber Loss (HL)

Let $e = y - \hat{y}$:

$$L_{\text{HL}}(y,\hat{y}) = \begin{cases} \frac{1}{2}e^2 & \text{if } |e| \le \delta \\ \delta \cdot \left(|e| - \frac{1}{2}\delta\right) & \text{otherwise} \end{cases}$$

- When $|e| \le \delta : \frac{\partial L}{\partial \hat{y}} = e \cdot \frac{\partial e}{\partial \hat{y}} = e \cdot (-1) = -e$
- When $|e| > \delta : \frac{\partial L}{\partial \hat{y}} = \delta \cdot \frac{\partial}{\partial \hat{y}}|e|$

From MAE:

$$\frac{\partial |e|}{\partial \hat{y}} = \operatorname{sign}(e) = \begin{cases} 1 & \text{if } e > 0 \\ -1 & \text{if } e < 0 \\ \text{undefined} & \text{if } e = 0 \end{cases}$$

Finally:

$$\frac{\partial L_{\text{HL}}}{\partial \hat{y}} = \begin{cases} y-\hat{y} & \text{if } |y-\hat{y}| \le \delta \\ \delta \cdot \operatorname{sign}(y-\hat{y}) & \text{if } |y-\hat{y}| > \delta \end{cases}$$

Log Cosh Loss

$$L(y,\hat{y}) = \log(\cosh(\hat{y}-y))$$

Let $z = \hat{y} - y$, then $L = \log(\cosh(z))$

$$\frac{\partial L}{\partial \hat{y}} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial \hat{y}}$$
$$\frac{\partial L}{\partial z} = \frac{\partial}{\partial z}\log(\cosh(z)) = \frac{1}{\cosh(z)} \cdot \sinh(z) = \tanh(z)$$
$$\frac{\partial z}{\partial \hat{y}} = \frac{\partial}{\partial \hat{y}}(\hat{y}-y) = 1$$
$$\therefore \frac{\partial L}{\partial \hat{y}} = \tanh(\hat{y}-y)$$

Binary Cross Entropy (BCE)

$$L(y,\hat{y}) = -[y\log\hat{y} + (1-y)\log(1-\hat{y})]$$

Derivative:

$$\begin{aligned}\frac{\partial L}{\partial \hat{y}} &= \frac{\partial}{\partial \hat{y}}[-y\log\hat{y}] + \frac{\partial}{\partial \hat{y}}[-(1-y)\log(1-\hat{y})] \\ &= -\frac{y}{\hat{y}} - (1-y)\cdot\frac{-1}{1-\hat{y}} = \frac{1-y}{1-\hat{y}} \\ \therefore \frac{\partial L}{\partial \hat{y}} &= -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}\end{aligned}$$

Cross Entropy

$$L(y,\hat{y}) = -\sum_{k=1}^{K} y_k \log\hat{y_k}$$
$$L(y_k,\hat{y}_k) = \begin{cases} -\log\hat{y_k} & \text{if } y_k \text{ is ground truth label} \\ 0 & \text{otherwise} \end{cases}$$

Derivatives:

$$\frac{\partial L}{\partial \hat{y_i}} = -\frac{y_i}{\hat{y_i}}$$

Hinge Loss

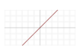
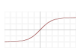
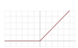
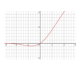
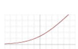
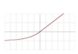
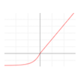
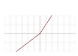
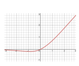
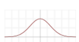
$$L(y,\hat{y}) = \max(0, 1 - y\cdot\hat{y})$$
$$\frac{\partial L}{\partial \hat{y}} = \begin{cases} -y & \text{if } 1 - y\hat{y} > 0 \\ 0 & \text{otherwise} \end{cases}$$

Kullback-Leibler (KL) Divergence

$$L(P \parallel Q) = \sum_{k=1}^{K} P(k)\log\left(\frac{P(k)}{Q(k)}\right)$$

- P is true distribution
- Q is approximated distribution
- K is the number of classes

Derivative of *Activation Function* ($\frac{\partial \hat{y}}{\partial z}$)

Name	Plot	Function	Derivative
identify		x	1
Binary Step		$\begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x \ge 0 \end{cases}$	0
Sigmoid		$\sigma(x) = \frac{1}{1+e^{-x}}$	$\sigma(x)(1-\sigma(x))$
Tanh		$\frac{e^x - e^{-x}}{e^x + e^{-x}}$	$1 - \tanh^2(x)$
ReLU		$\begin{cases} 0 \text{ if } x \le 0 \\ x \text{ if } x > 0 \end{cases}$ $= \max(0, x)$	$\begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x > 0 \end{cases}$
GeLU		$\frac{x\left(1+\operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right)}{2}$ $= x\Phi(x)$	$\Phi(x) + x\phi(x)$
Softplus		$\ln(1+e^x)$	$\frac{1}{1+e^{-x}}$
ELU		$\begin{cases} \alpha(e^x - 1) \text{ if } x < 0 \\ x \text{ if } x \ge 0 \end{cases}$	$\begin{cases} \alpha e^x \text{ if } x < 0 \\ 1 \text{ if } x \ge 0 \end{cases}$
SELU		$\lambda \cdot \text{ELU}$, $\lambda = 1.0507$, $\alpha = 1.67326$	$\lambda\begin{cases} \alpha e^x \text{ if } x < 0 \\ 1 \text{ if } x \ge 0 \end{cases}$
PReLU		$\begin{cases} \alpha x \text{ if } x < 0 \\ x \text{ if } x \ge 0 \end{cases}$	$\begin{cases} \alpha \text{ if } x < 0 \\ 1 \text{ if } \ge 0 \end{cases}$
ELiSH		$\begin{cases} \frac{e^x - 1}{1+e^{-x}} \text{ if } x < 0 \\ \frac{x}{1+e^{-x}} \text{ if } x \ge 0 \end{cases}$	$\begin{cases} \frac{2e^{2x}+e^{3x}-e^x}{e^{2x}+2e^x+1} \\ \frac{xe^x+e^{2x}+e^x}{e^{2x}+2e^x+1} \end{cases}$
Gaussian		e^{-x^2}	$-2xe^{-x^2}$
Softmax		$\sigma(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^{C} e^{x_j}}$ for $i = 1, \ldots, J$	$\sigma(\vec{x})_i \cdot (\Delta_{ij} - \sigma(\vec{x})_j)$

Softmax Derivative

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{C}} e^{z_j}$$

subsititute, $S = \sum_{j=1}^{C} e^{z_j}$

$$\sigma(z)_i = \frac{e^{z_i}}{S}$$
$$\because \frac{\partial}{\partial x}\left(\frac{u}{v}\right) = \frac{vu' - uv'}{v^2}$$

Find u' and v' for $\sigma(z)_j$ where $u = e^{z_i}, v = S$:

$$u' = \frac{\partial e^{z_i}}{\partial z_j} = \begin{cases} e^{z_i} & \text{if } i = j \\ 0 & \text{if } i \ne j \end{cases}$$
$$v' = \frac{\partial S}{\partial z_j} = \frac{\partial}{\partial z_j}\left(\sum_{j=1}^{C} e^{z_j}\right) = e^{z_j}$$

Differentiate (*note*: $\Delta_{ij} = \begin{cases} 1 \text{ if } i=j \\ 0 \text{ if } i \ne j \end{cases}$):

$$\begin{aligned}\therefore \frac{\partial \sigma_i}{\partial z_j} &= \frac{S \cdot \left[e^{z_i} \cdot \Delta_{ij}\right] - e^{z_i} \cdot e^{z_j}}{S^2} \\ &= e_{z_i} \cdot \frac{S \cdot \Delta_{ij} - e^{z_j}}{S^2} \\ &= \frac{e^{z_i}}{S} \cdot \frac{S \cdot \Delta_{ij} - e^{z_j}}{S} \\ &= \sigma(z)_i \cdot \frac{S \cdot \Delta_{ij} - e^{z_j}}{S} \\ &= \sigma(z)_i \cdot \left(\Delta_{ij} - \frac{e^{z_j}}{S}\right) \\ &= \sigma(z)_i \cdot \left(\Delta_{ij} - \sigma(z)_j\right)\end{aligned}$$

***Tanh* Derivative**

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Let $u = e^x - e^{-x}$ and $v = e^x + e^{-x}$, find u' and v':

$$u' = e^x + e^{-x}$$
$$v' = e^x - e^{-x}$$

Differentiate:

$$\begin{aligned}\frac{\partial}{\partial x}\tanh(x) &= \frac{(e^x+e^{-x})(e^x+e^{-x}) - (e^x-e^{-x})(e^x-e^{-x})}{(e^x+e^{-x})^2} \\ &= \frac{(e^x+e^{-x})^2 - (e^x-e^{-x})^2}{(e^x+e^{-x})^2} \\ &= 1 - \frac{(e^x-e^{-x})^2}{(e^x+e^{-x})^2} \\ &= 1 - \tanh^2(x)\end{aligned}$$

Equivalently:

$$\begin{aligned}\frac{\partial}{\partial x}\tanh(x) &= \frac{\partial}{\partial x}\left(\sinh\frac{x}{\cosh}x\right) \\ &= \frac{\cosh x\cosh x - \sinh x \sinh x}{\cosh^2 x} \\ &= \frac{\cosh^2 x - \sinh^2 x}{\cosh^2 x} \\ &= \frac{1}{\cosh^2 x} = \operatorname{sech}^2 x = 1 - \tanh^2(x)\end{aligned}$$

Sigmoid Derivative

$$\sigma(x) = \frac{1}{1+e^{-x}}$$
$$\because \frac{\partial}{\partial u}\left(\frac{1}{u}\right) = \left(-\frac{1}{u^2}\right)$$

Let $u = 1 + e^{-x}$:

$$\begin{aligned}\frac{\partial \sigma}{\partial x} &= \frac{\partial \sigma}{\partial u} \cdot \frac{\partial u}{\partial x} \\ &= \frac{\partial}{\partial u}\left(\frac{1}{u}\right) \cdot \frac{\partial}{\partial x}(1+e^{-x}) \\ &= -\frac{1}{u^2} \cdot -e^{-x} \\ &= \frac{e^{-x}}{u^2} = \frac{e^{-x}}{(1+e^{-x})^2} \\ &= \frac{1}{1+e^{-x}} \cdot \frac{e^{-x}}{1+e^{-x}} \\ &= \frac{1}{1+e^{-x}} \cdot \frac{(1+e^{-x}) - 1}{1+e^{-x}} \\ &= \frac{1}{1+e^{-x}} \cdot \left(\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}}\right) \\ &= \frac{1}{1+e^{-x}} \cdot \left(1 - \frac{1}{1+e^{-x}}\right) \\ &= \sigma(x) \cdot (1 - \sigma(x))\end{aligned}$$