Classification

2019-Fall

Prof. Hyunjung Shim

Slide credits: Bhiksha Raj (CMU), Fei-Fie Li (Stanford), Alexander Amini (MIT)

Supervised vs. Unsupervised Learning

Supervised learning (classification)

Training data are accompanied by labels indicating the class of the observations.

A new tuple in testing data is classified based on the model learned from training set.

Unsupervised learning (clustering)

Class labels of training data are unknown.

Given a set of measurements, establish the existence of classes (or clusters) in the data.

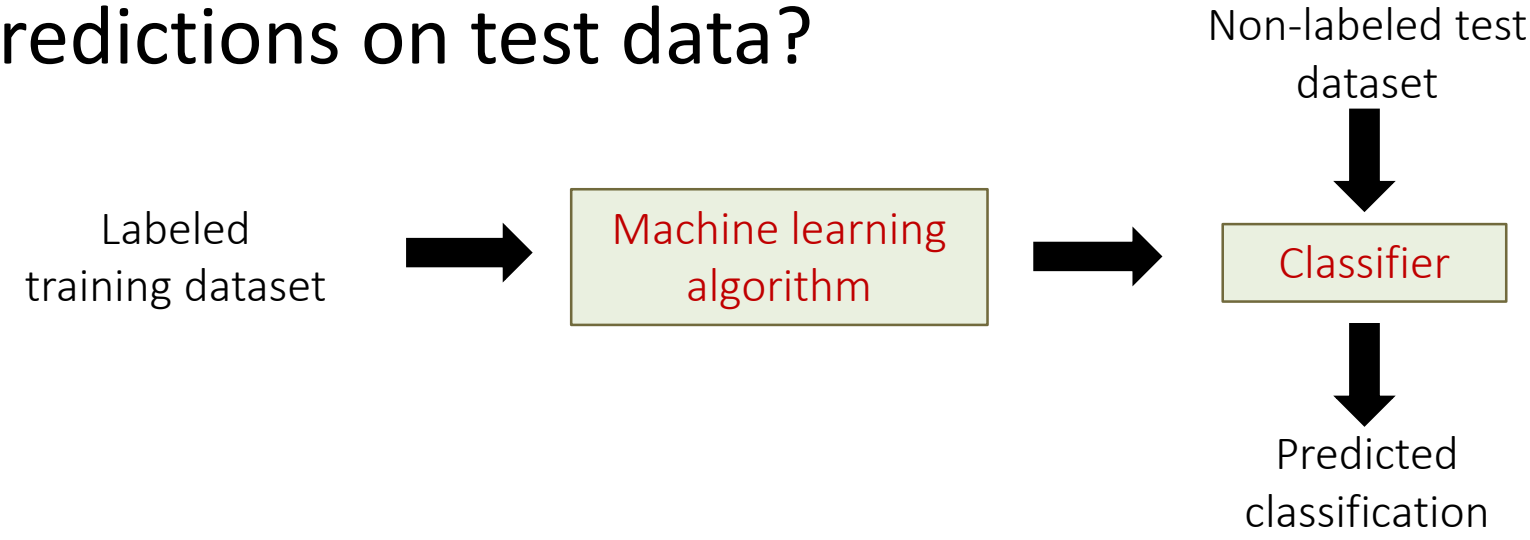
Classification

Classification problem

Learn a model from training data (numeric or categorical) with class labels.

Predict the label of a new tuple using the learned model.

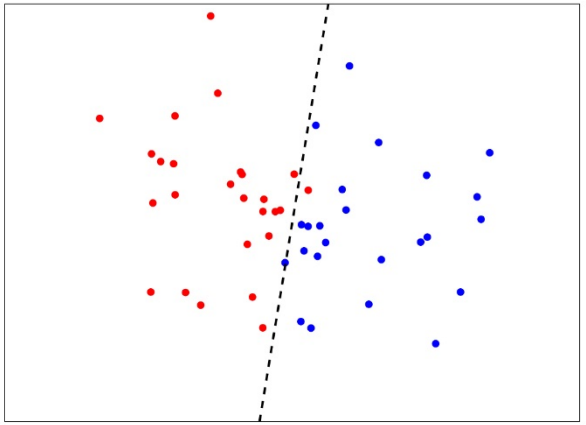
How to build a model for highly accurate predictions on test data?



Process for Classification

A model is built from a training set with predetermined classes

Each sample is assumed to belong to a predefined class, as determined by the class label attribute.



The set of tuples used for model construction is training set.

The model is represented as classification rules, decision trees, or mathematical formula.

Process for Classification

The model is used for classifying future or unknown tuples with non-labels.

Estimate the accuracy of the model.

The known label of test sample is compared with the classified result from the model.

A test set should be independent of a training set.

Example of Classification Problem

Text categorization (e.g., spam filtering)

Fraud detection

Optical character recognition

Market segmentation (e.g., predict if a customer will respond to promotion)

Bioinformatics (e.g., classify proteins according to their functions)

Example of Classification Problem

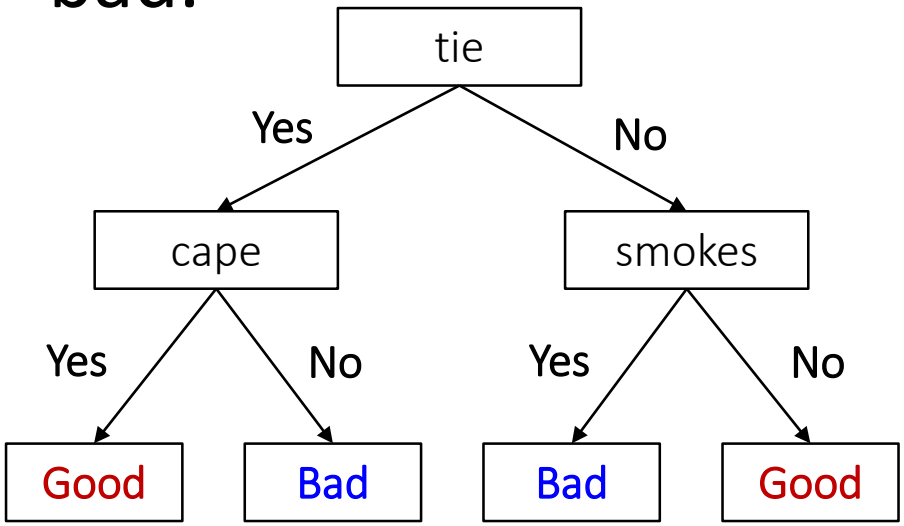
Build a classification model to classify people as good or bad from their appearance.

Training data

	gender	*mask*	*cape*	*tie*	*ears*	*smokes*	*Label*
batman	male	yes	yes	no	yes	no	Good
robin	male	yes	yes	no	no	no	Good
alfred	male	no	no	yes	no	no	Good
penguin	male	no	no	yes	no	yes	Bad
catwoman	female	yes	no	no	yes	no	Bad
joker	male	no	no	no	no	no	Bad

Example of Classification Problem

Predict new people (unlabeled data) as good or bad.

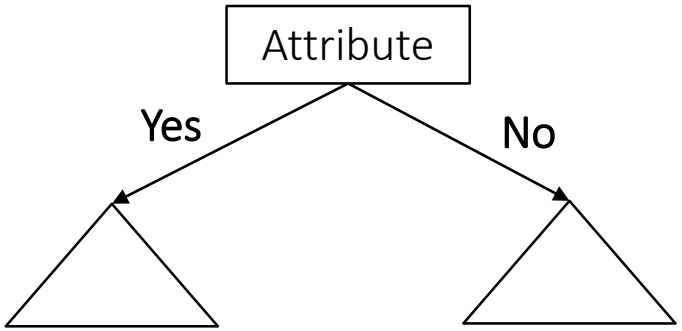


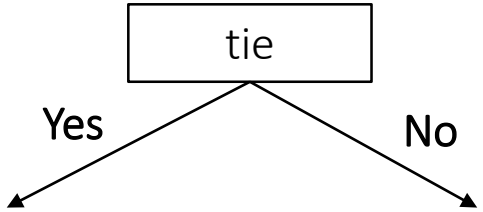
Testing data

	gender	*mask*	*cape*	*tie*	*ears*	*smokes*	*Label*
batgirl	female	yes	yes	no	yes	no	??
riddler	male	yes	no	no	no	no	??

Key Idea of Decision Tree

Find a decision rule to split data into disjoint subsets.

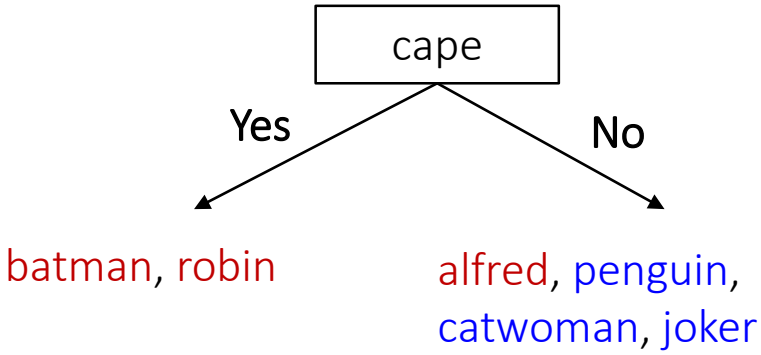


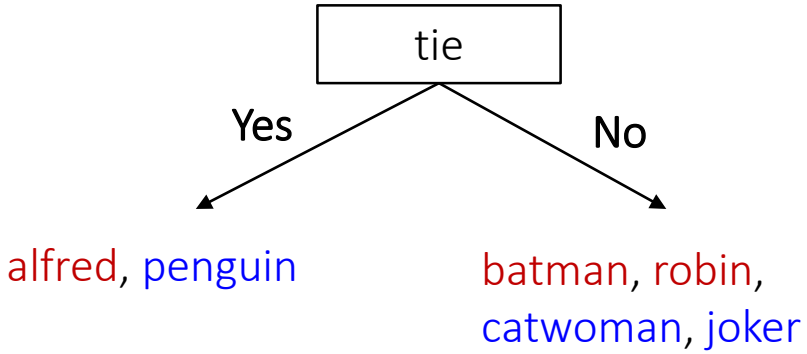


	gender	*mask*	*cape*	*tie*	*ears*	*smokes*	*Label*
batman	male	yes	yes	no	yes	no	Good
robin	male	yes	yes	no	no	no	Good
alfred	male	no	no	yes	no	no	Good
penguin	male	no	no	yes	no	yes	Bad
catwoman	female	yes	no	no	yes	no	Bad
joker	male	no	no	no	no	no	Bad

Key Idea of Decision Tree

Q: Which rule is better to split data into disjoint subsets?





How to Measure Purity

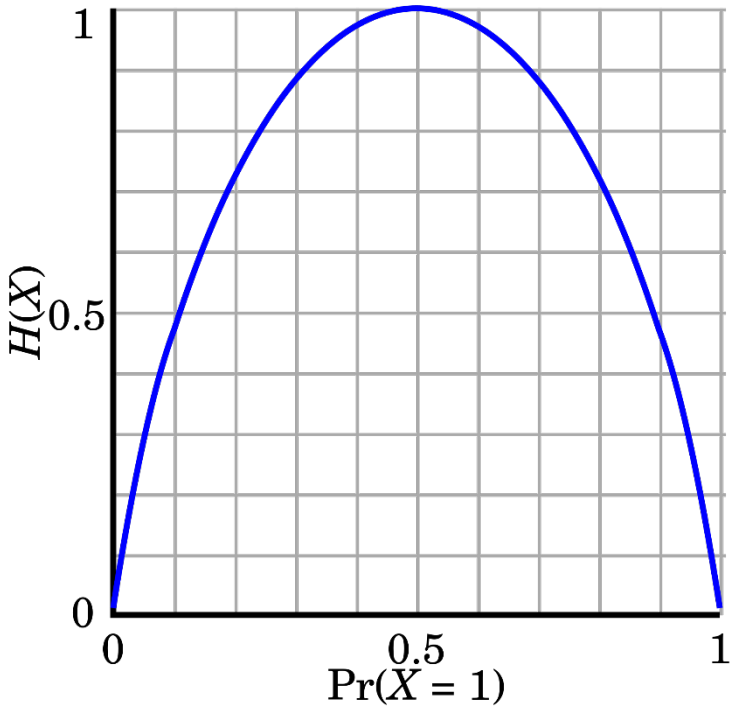
Information entropy

Given an arbitrary data set *D*, it is partitioned into *k* subsets.

$$H(D) = -\sum_{i=1}^{k} p_i \log_2 p_i$$

Zero if only one class exists.

One if all classes are divided equally.



How to Measure Purity

Assume that there are two classes *P* and *N*.

Let the data set *D* contain *p* elements of class *P* and *n* elements of class *N*.

For dataset D, the information entropy is defined as:

$$H(D) = I(p,n) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

Example

Batman, robin, alfred, penguin, catwoman, joker

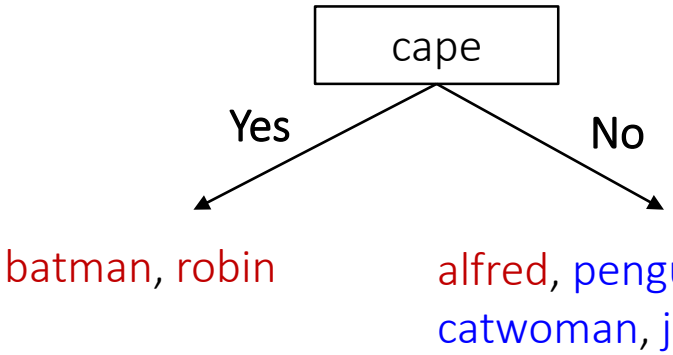
Information Gain

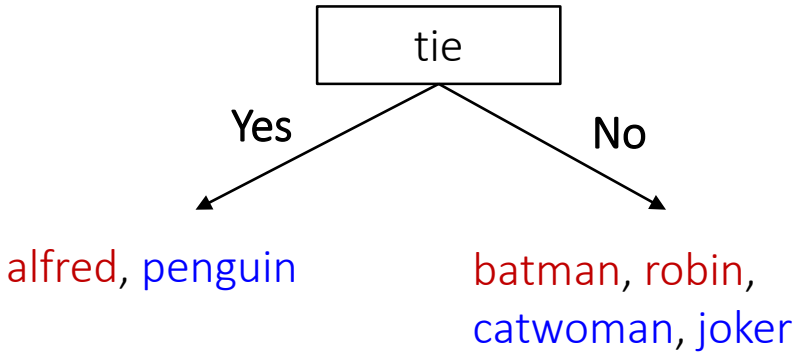
Suppose that an attribute *A* is chosen.

A data set *D* is partitioned into disjoint subsets $\{D_1, D_2, \ldots, D_v\}$.

If D_i contains p_i examples of *P* and n_i examples of *N*, the expected information entropy needed to classify samples is calculated as:

$$I_A(D) = \sum_{i=1}^{v} \frac{|D_i|}{|D|} \times H(D_i)$$





$$I_{cape}(D) = \frac{2}{6} \times I(2,0) + \frac{4}{6} \times I(1,3)$$

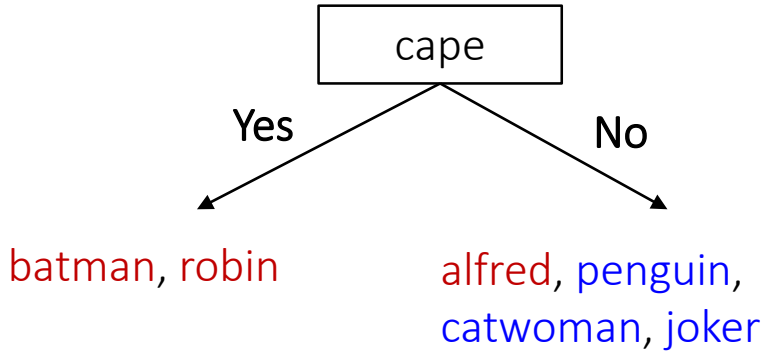
$$I_{tie}(D) = \frac{2}{6} \times I(1,1) + \frac{4}{6} \times I(2,2)$$

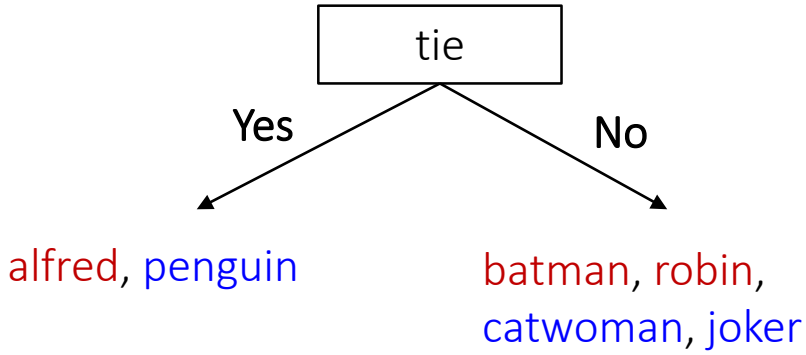
Information Gain

Information gain by branching on attribute A

Entropy before branching– Entropy after branching

$$Gain(A) = H(D) - I_A(D) = -\sum_{i=1}^{k} p_i \log_2 p_i - I_A(D)$$





$$Gain(cape) = H(D) - I_{cape}(D)$$
$$= I(3,3) - \left(\frac{2}{6} \times I(2,0) + \frac{4}{6} \times I(1,3)\right)$$

$$Gain(tie) = H(D) - I_{tie}(D)$$
$$= I(3,3) - \left(\frac{2}{6} \times I(1,1) + \frac{4}{6} \times I(2,2)\right)$$

Algorithm for Decision Tree

Basic algorithm (a greedy algorithm)

Tree is constructed in a top-down recursive divide-and-conquer manner.

At start, all the training examples are at the root.

Attributes are categorical (if continuous-valued, they are discretized in advance).

Attributes are chosen on the basis of a heuristic or statistical measure (e.g., information gain).

Examples are partitioned recursively based on selected attributes.

Algorithm for Decision Tree

Conditions for stopping partitioning

All samples for a given node belong to the same class.

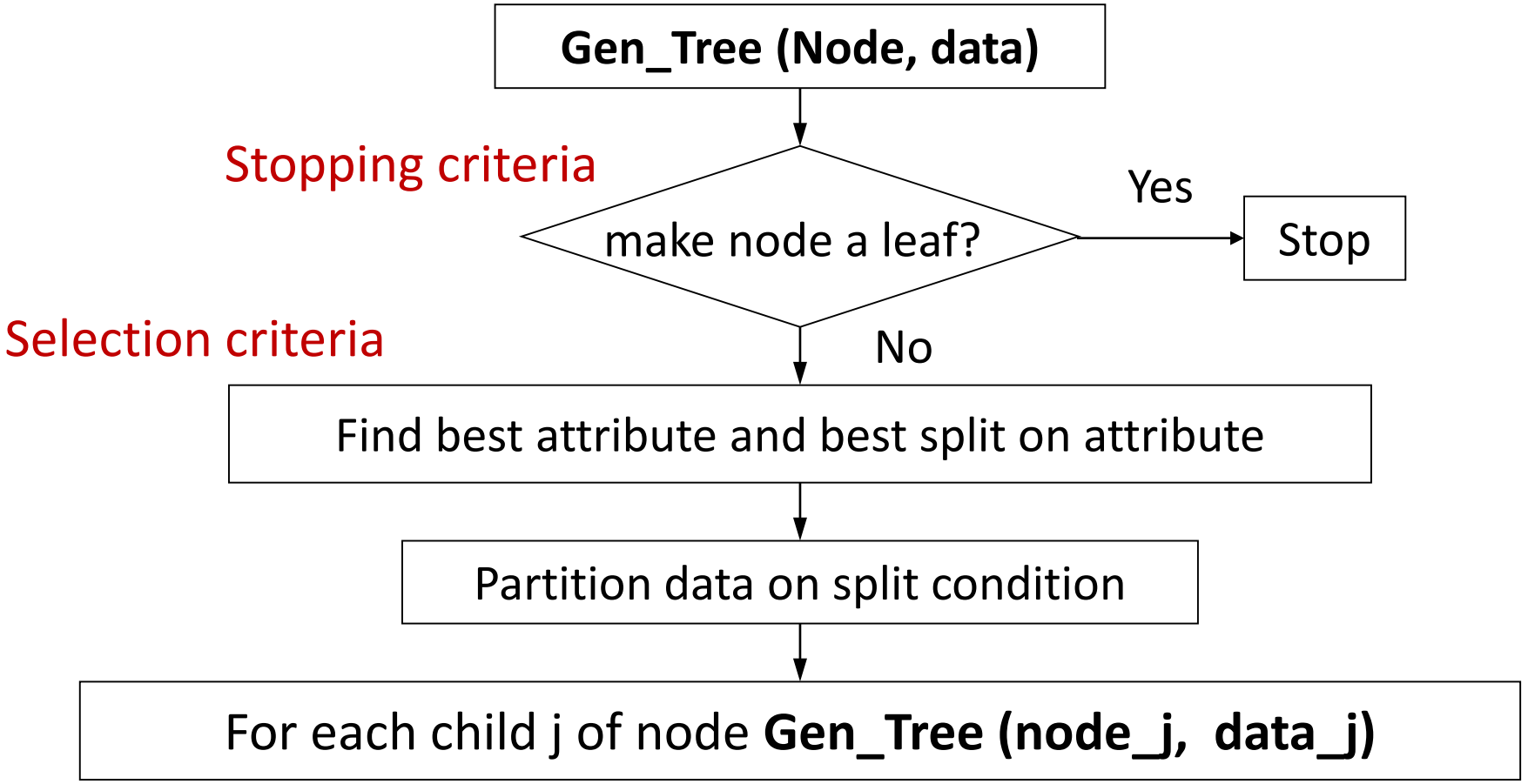
There are no remaining attributes for further partitioning.

The majority voting is employed for classifying the leaf.

There are no samples left.

Algorithm for Decision Tree

Top-down construction



How to Work the Decision Tree

For D, select the attribute the highest information gain.

$$Gain(A) = H(D) - I_A(D) = -\sum_{i=1}^{k} p_i \log_2 p_i - I_A(D)$$

Training data

	gender	*mask*	*cape*	*tie*	*ears*	*smokes*	*Label*
batman	male	yes	yes	no	yes	no	Good
robin	male	yes	yes	no	no	no	Good
alfred	male	no	no	yes	no	no	Good
penguin	male	no	no	yes	no	yes	Bad
catwoman	female	yes	no	no	yes	no	Bad
joker	male	no	no	no	no	no	Bad

How to Work the Decision Tree (ID3)

For D, select the attribute the highest information gain.

$H(D) =$

$I_{gender}(D) =$

$I_{mask}(D) =$

$I_{cape}(D) =$

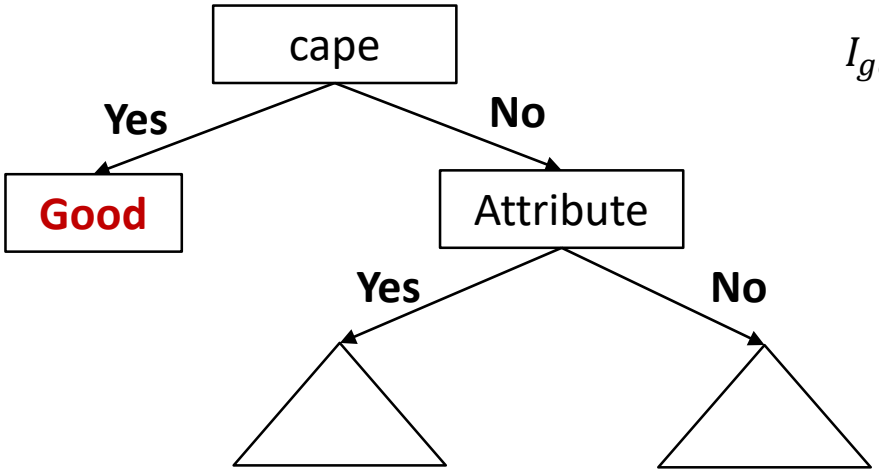
$I_{tie}(D) =$

$I_{ears}(D) =$

$I_{smokes}(D) =$

How to Work the Decision Tree (ID3)

For a subset (cape = no) , select the attribute the highest information gain.



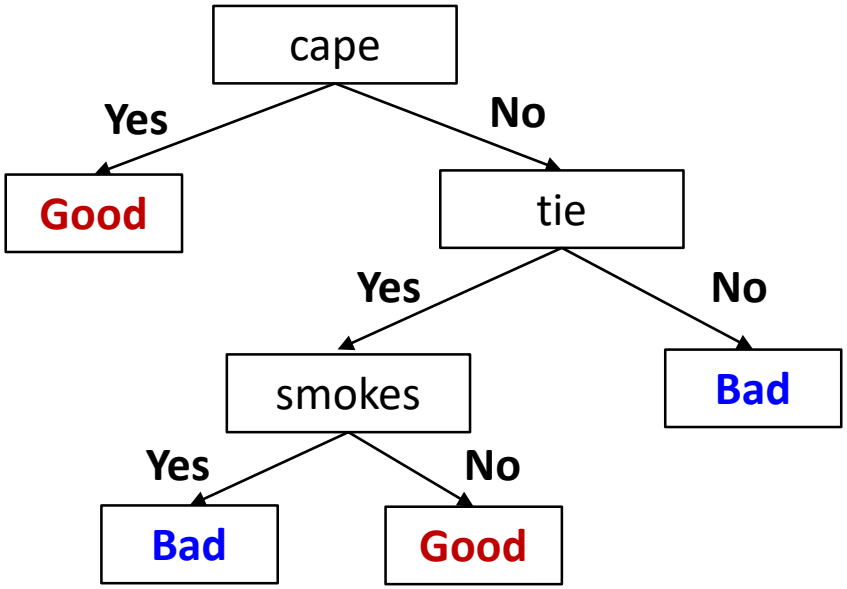
$I_{gender}(D) =$

$I_{tie}(D) =$

	gender	*mask*	*tie*	*ears*	*smokes*	*Label*
alfred	male	no	yes	no	no	Good
penguin	male	no	yes	no	yes	Bad
catwoman	female	yes	no	yes	no	Bad
joker	male	no	no	no	no	Bad

How to Work the Decision Tree (ID3)

Predicting the labels of tuples on test data



Testing data

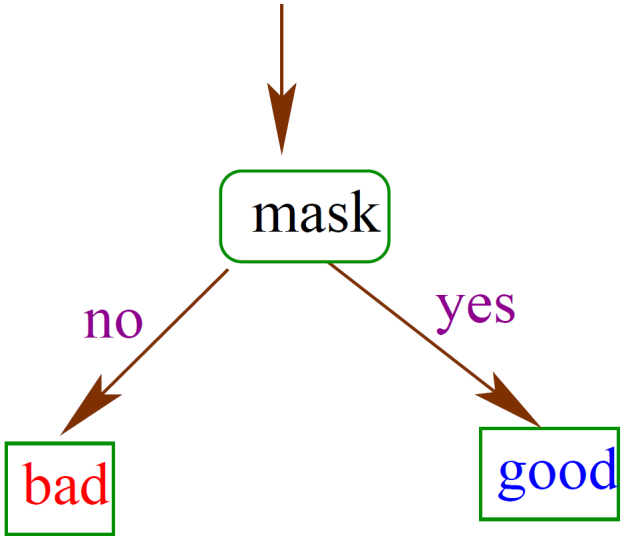
	gender	*mask*	*cape*	*tie*	*ears*	*smokes*	*Label*
batgirl	female	yes	yes	no	yes	no	??
riddler	male	yes	no	no	no	no	??

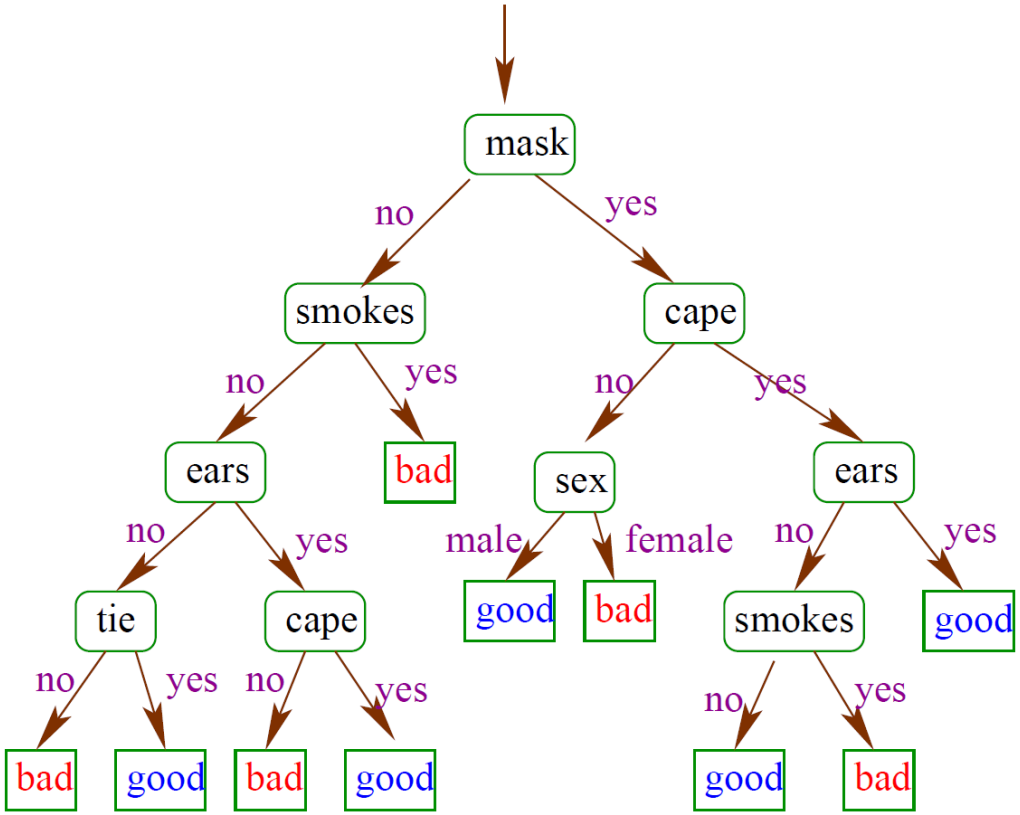
Possible Decision Trees

Underfitting vs. overfitting

Underfitting: overly simple, does not even fit available data

Overfitting: too compltex, perfectly classifies training data





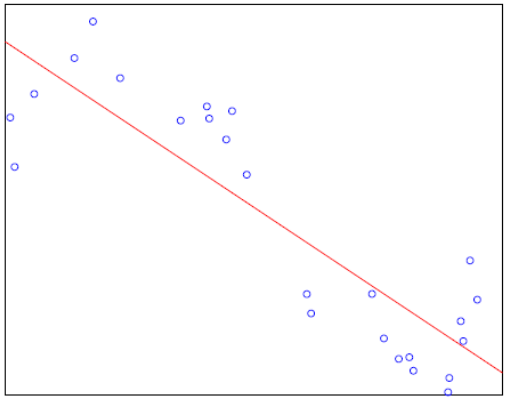
Overfitting and Tree Pruning

Overfitting

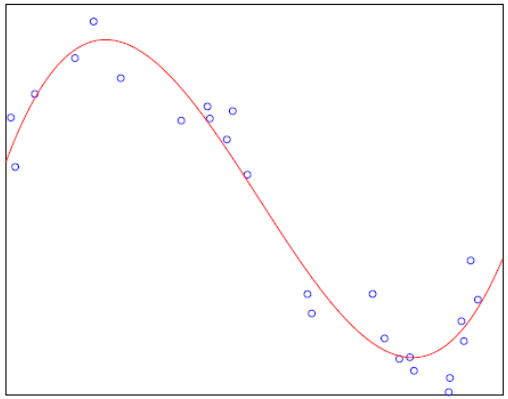
An induced tree may overfit the training data.

Too many branches, some may reflect anomalies due to noise or outliers.

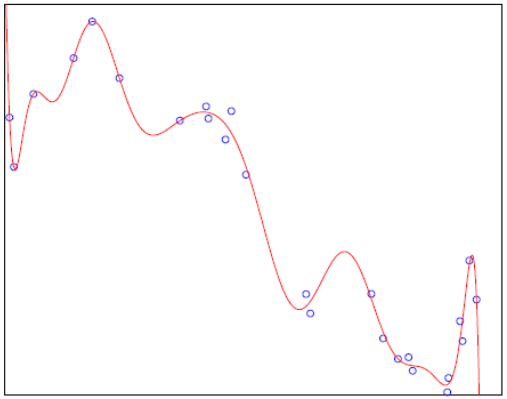
Poor accuracy for unseen samples.



underfit
(degree = 1)



ideal fit
(degree = 3)



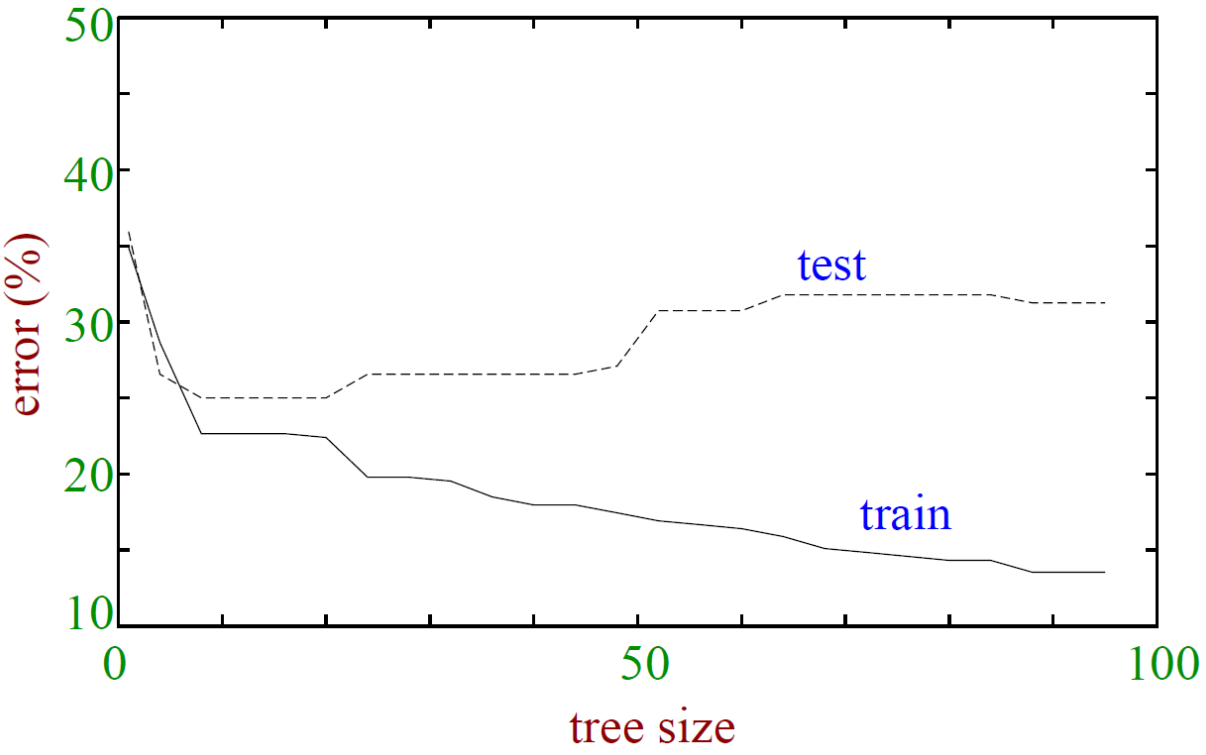
overfit
(degree = 20)

Tree Size vs. Accuracy

Trees must be big enough to fit training data

But, tree that are too big may overfit.

It is difficult to tell best tree sizes from training error.



Two Approaches to Avoid Overfitting

Prepruning

Do not split a node if this would result in the goodness measure falling below a threshold.

Difficult to choose an appropriate threshold.

Postpruning

Remove branches from a "fully grown" tree.

Use a set of data different from the training data to decide which is the "best pruned tree".

Supervised Deep Learning

Logistic Regression ~ Classification

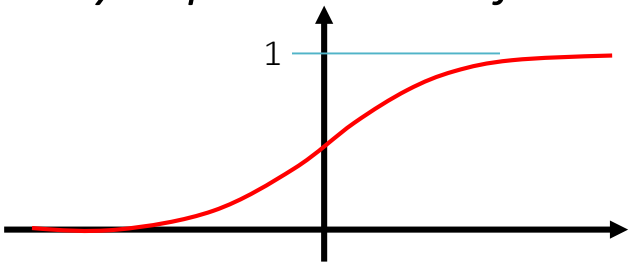
Simple binary classification algorithm

Start with a function of the form:

$$f(x;\theta) \equiv \sigma(\theta^\top x) = \frac{1}{1+\exp(-\theta^\top x)}$$

Interpretation: f(x) is probability that y = 1.

Sigmoid "nonlinearity" squashes linear function to [0,1].



Find choice of θ that minimizes loss function.

Loss Function

Loss function measures gap between prediction and ground truth.

For regression: Mean squared error (MSE)

For classification: Cross entropy loss

Mean squared Error

$$MSE = \frac{1}{N}\sum(t_i - s_i)^2$$

Prediction

Ground Truth

Cross entropy loss

$$CE = -\sum_{i}^{C} t_i log(s_i)$$

Classes

Prediction

Ground Truth {0,1}

Loss Function: Cross Entropy

Cross entropy measures the difference between two distributions.

$$H(p,q) = -\sum_{i=1}^{k} p_i \log_2 q_i$$

Interpretation: Expectation of negative log likelihood.

Example: p(x) = [1 0], q(x) = [0 1]

$$-p(x)\log q(x) = -[1\ 0]\begin{bmatrix}0\\1\end{bmatrix} = -(-\infty + 0) = \infty$$

$$-p(x)\log q(x) = -[1\ 0]\begin{bmatrix}1\\0\end{bmatrix} = -(0 + 0) = 0$$

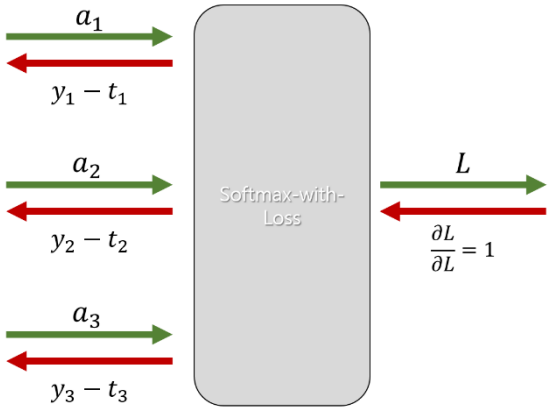
Loss Function: Cross Entropy

Computing the gradient of CE

Output → **Softmax** → Cross entropy

Suppose $g(x) = e^{o_i}, f(x) = \sum_{k=1}^{K} e^{o_k}$

$p_i = \frac{g(x)}{f(x)}$, compute $\frac{dp_i}{do_j}$ where $\frac{g(x)}{f(x)} = \frac{g'(x)f(x) - g(x)f'(x)}{f(x)^2}$



1) $i = j$, $\frac{e^{o_i} \sum_{k=1}^{K} e^{o_k} - e^{o_i} e^{o_i}}{\left(\sum_{k=1}^{K} e^{o_k}\right)^2} = \frac{e^{o_i}}{\sum_{k=1}^{K} e^{o_k}} \times \frac{\sum_{k=1}^{K} e^{o_k} - e^{o_i}}{\sum_{k=1}^{K} e^{o_k}} = ?$

2) $i \neq j$, $\frac{0 \sum_{k=1}^{K} e^{o_k} - e^{o_j} e^{o_i}}{\left(\sum_{k=1}^{K} e^{o_k}\right)^2} = \frac{-e^{o_j}}{\sum_{k=1}^{K} e^{o_k}} \times \frac{e^{o_i}}{\sum_{k=1}^{K} e^{o_k}} = ?$

Loss Function: Cross Entropy

Computing the gradient of CE

Output → Softmax → **Cross entropy**

$L = -\sum_i y_i \log p_i$, compute $\frac{dL}{do_j}$.

$$\therefore \frac{dL}{do_j} = -\sum_k y_k \frac{d \log p_k}{do_j} = -\sum_k y_k \frac{d \log p_k}{dp_k} \frac{dp_k}{do_j}$$

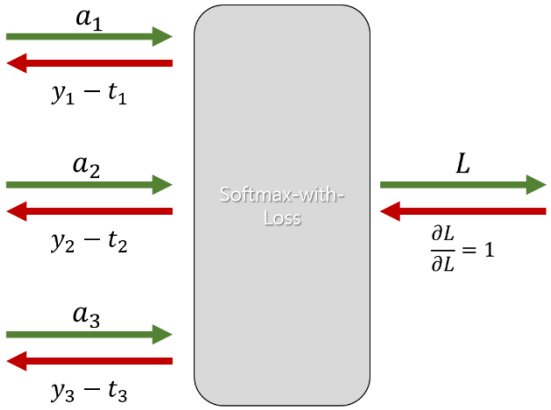
$$= -\sum_k y_k \frac{1}{p_k} \times \frac{dp_k}{do_j} .$$

Utilizing the derivative of softmax $\frac{dp_k}{do_j}$,

$$\frac{dL}{do_j} = -y_j \frac{1}{p_j} \times \frac{dp_j}{do_j} - \sum_{k \neq j} y_k \frac{1}{p_k} \times \frac{dp_k}{do_j}$$

$$= -y_j \frac{p_j(1-p_j)}{p_j} - \sum_{k \neq j} y_k \frac{1}{p_k} \times (-p_k p_j)$$

$$= -y_j(1-p_j) + \sum_{k \neq j} y_k p_j = ?$$



What is the advantage of cross entropy?

Optimization

How do we tune θ to minimize $\mathcal{L}(\theta)$?

One algorithm: gradient descent

Compute gradient:

$$\nabla_\theta \mathcal{L}(\theta) = \sum_i^m x^{(i)} \cdot (y^{(i)} - f(x^{(i)}; \theta))$$

Follow gradient "downhill":

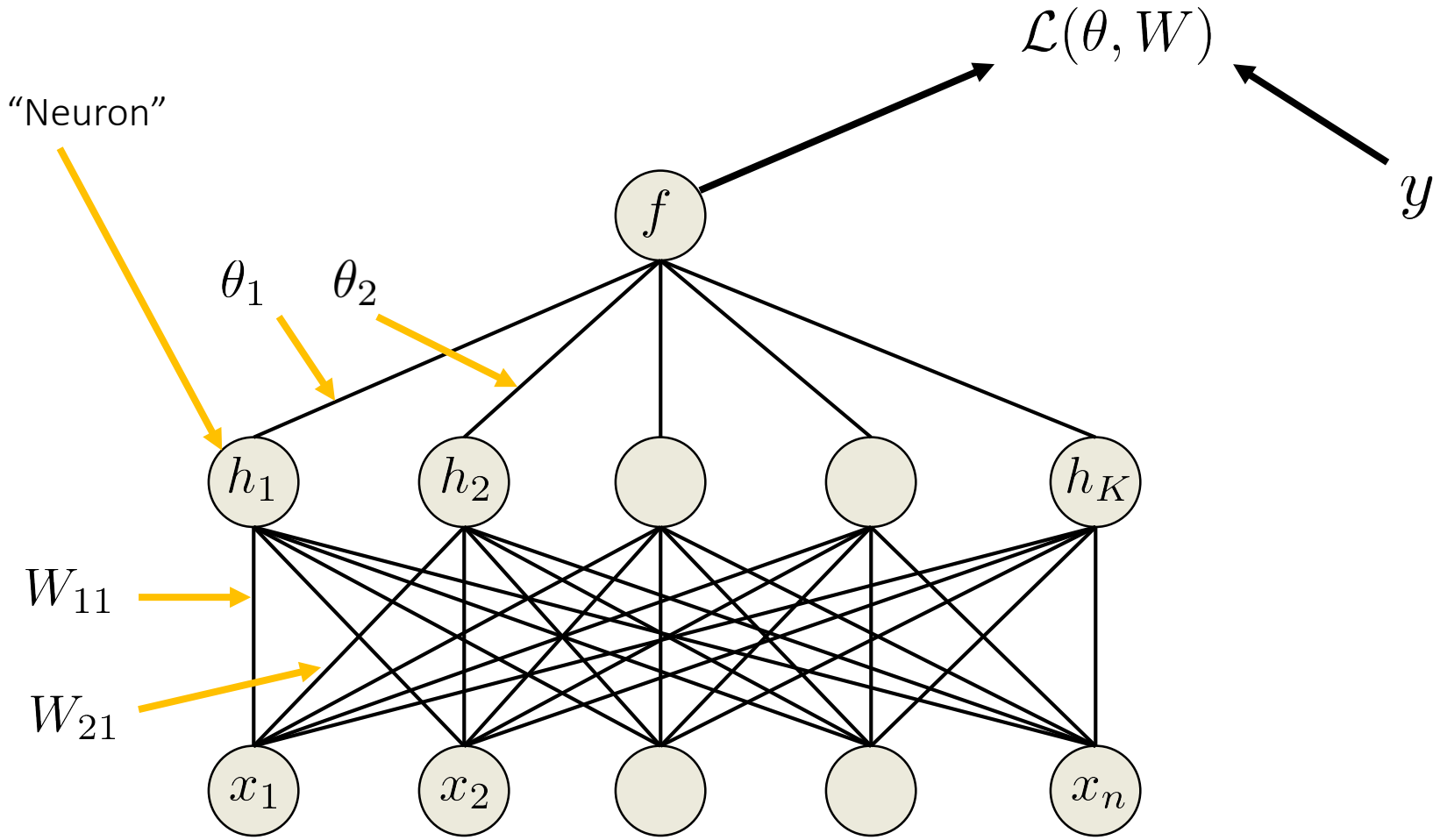
$$\theta := \theta - \eta \nabla_\theta \mathcal{L}(\theta)$$

Stochastic Gradient Descent (SGD): take step using gradient from only small batch of examples.

Scales to larger datasets. [Bottou & LeCun, 2005]

Neural network

This model is a sigmoid “neural network”:



Training Procedure

Collect labeled training data

For SGD: Randomly shuffle after each epoch!

$$\mathcal{X} = \{(x^{(i)}, y^{(i)}) : i = 1, \ldots, m\}$$

For a batch of examples:

Compute gradient w.r.t. all parameters in network.

$$\Delta_\theta := \nabla_\theta \mathcal{L}(\theta, W)$$

$$\Delta_W := \nabla_W \mathcal{L}(\theta, W)$$

Make a small update to parameters.

$$\theta := \theta - \eta_\theta \Delta_\theta$$

$$W := W - \eta_W \Delta_W$$

Repeat until convergence.

Training Procedure

Historically, this has not worked so easily.

Non-convex: Local minima; convergence criteria.

Optimization becomes difficult with many stages.

"Vanishing gradient problem"

Hard to diagnose and debug malfunctions.

Many things turn out to matter:

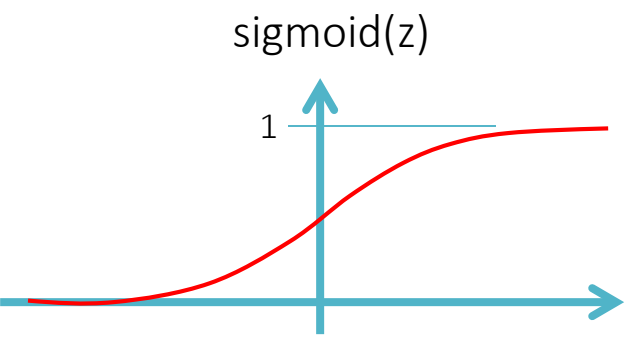
Choice of nonlinearities.

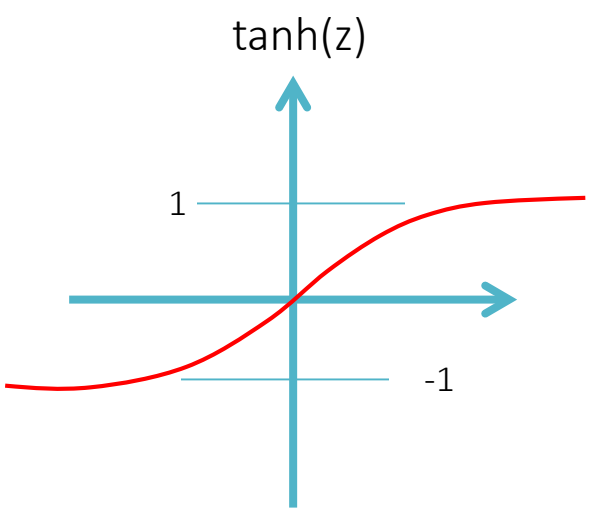
Optimizer parameters: momentum, learning rate, minibatches

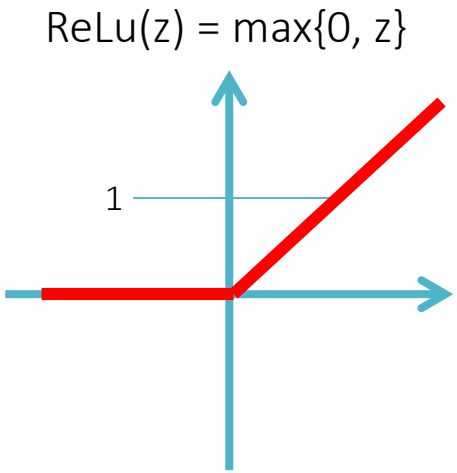
Nonlinearities

Choice of functions inside network matters.

Activations tell us whether this neuron should be fired or not.







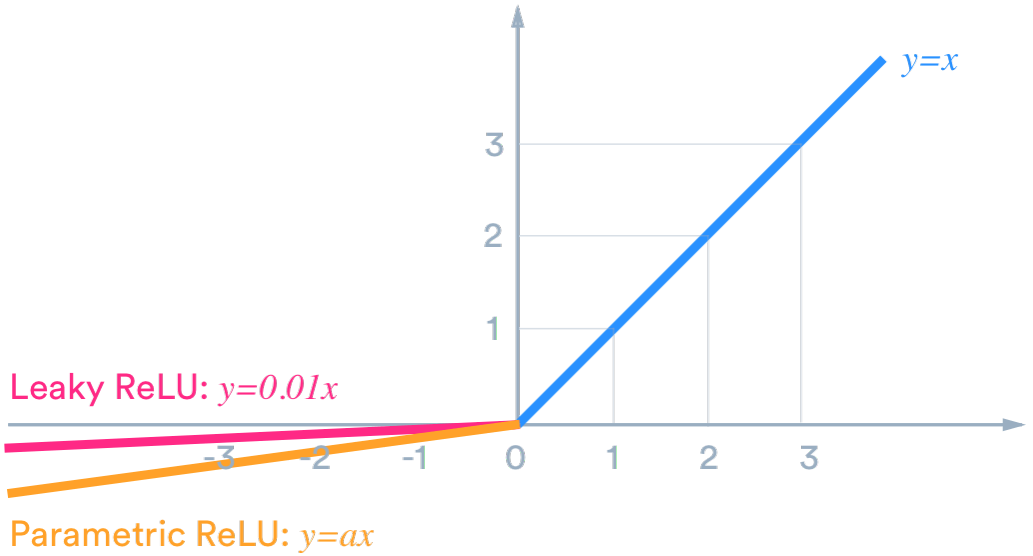
"Rectified Linear Unit"
→ Increasingly popular.

[Nair & Hinton, 2010]

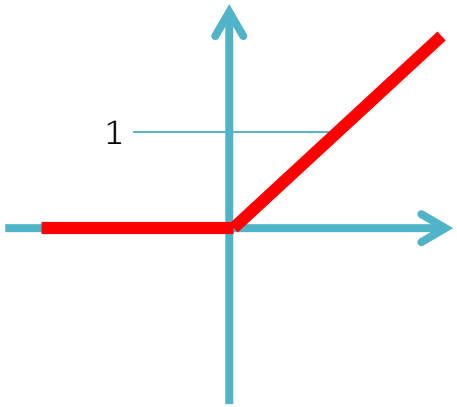
Nonlinearities

Choice of functions inside network matters.

Mostly, ReLu and Leaky ReLu are popular.



ReLu(z) = max{0, z}



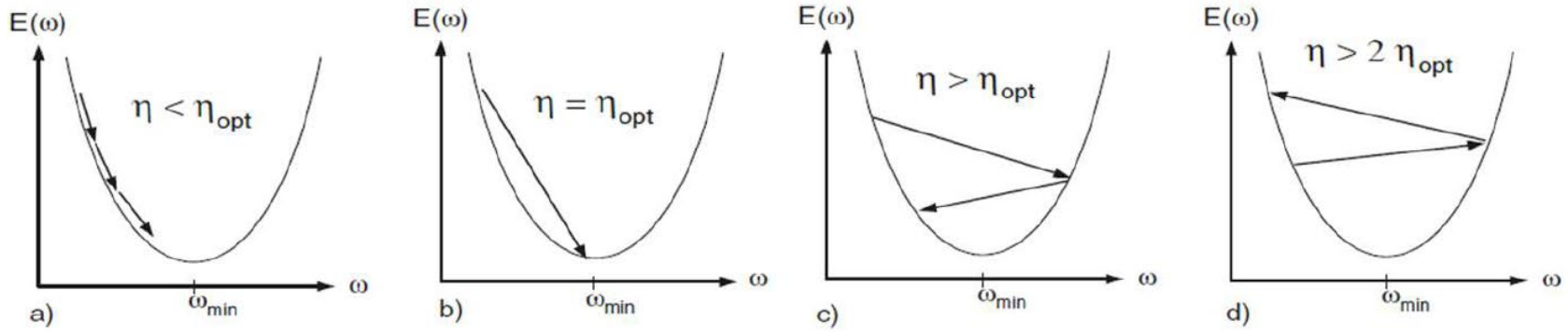
"Rectified Linear Unit"

→ Increasingly popular.

[Nair & Hinton, 2010]

Optimization: Learning Rate

Large learning rate hinders the convergence of training.



Larger learning rate: cannot converge, but helps to escape the local minima.

Smaller learning rate: falling into local minima, but converging

Optimization: Learning Rate

Decaying learning rate

Start with a large learning rate

Gradually reduce it with iterations

Typical decay schedules

- Linear decay: $\eta_k = \frac{\eta_0}{k+1}$
- Quadratic decay: $\eta_k = \frac{\eta_0}{(k+1)^2}$
- Exponential decay: $\eta_k = \eta_0 e^{-\beta k}$, where $\beta > 0$

Optimization: Momentum

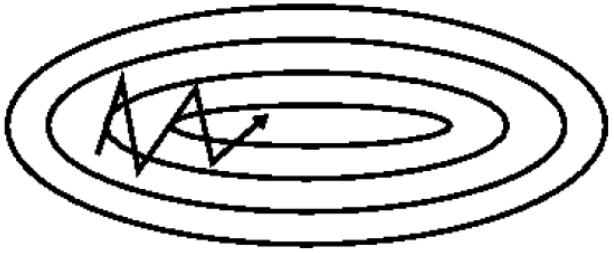
"Smooth" estimate of gradient from several steps of SGD:

$$v := \mu v + \epsilon_t \nabla_\theta \mathcal{L}(\theta)$$

$$\theta := \theta + v$$

Plain gradient update

With momentum



- The momentum method maintains a running average of all gradients until the *current* step

$$\Delta W^{(k)} = \beta \Delta W^{(k-1)} - \eta \nabla_W Err\left(W^{(k-1)}\right)$$

$$W^{(k)} = W^{(k-1)} + \Delta W^{(k)}$$

  - Typical β value is 0.9

Bengio, 2012: "Practical Recommendations for Gradient-Based Training of Deep Architectures"
Hinton, 2010: "A Practical Guide to Training Restricted Boltzmann Machines"

Optimization: Minibatches

Mini-Batch size: Number of training instances the network evaluates per weight update step.

Larger batch size = faster

Smaller batch size = (empirically) better generalization

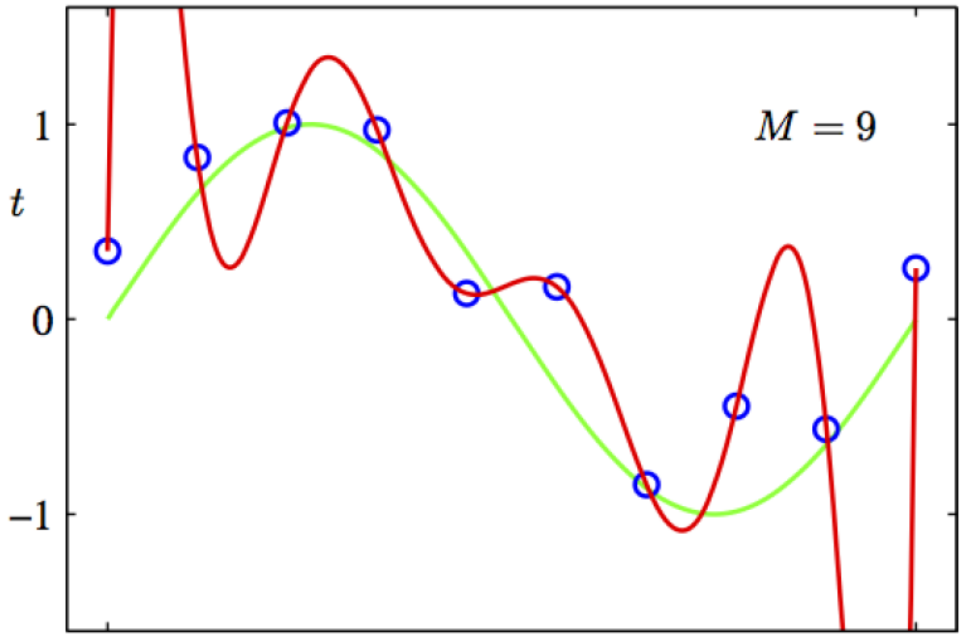
"Training with large minibatches is bad for your health. More importantly, it's bad for your test error. Friends don't let friends use minibatches larger than 32."
- Yann LeCun
Revisiting Small Batch Training for Deep Neural Networks (2018)

Overfitting and Regularization

Help the network **generalize** to data it hasn't seen.

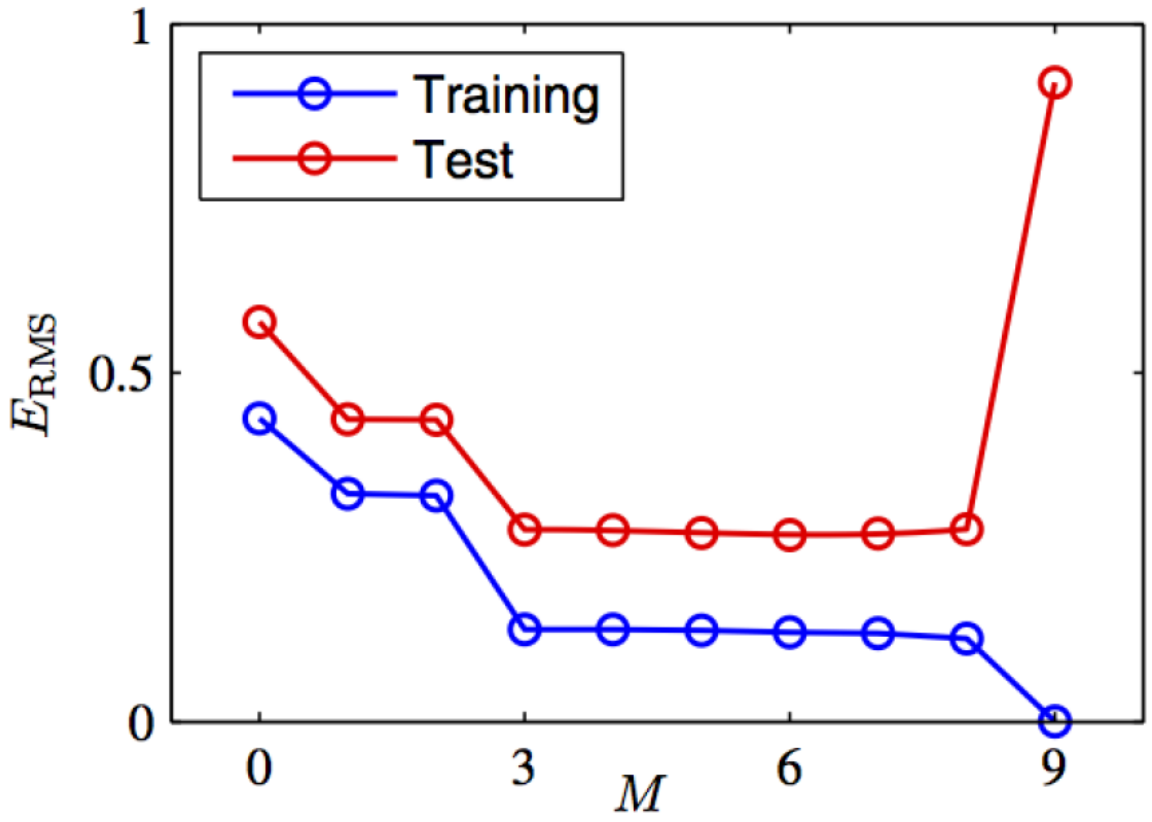
Big problem for small datasets.

Overfitting example (a sine curve vs 9-degree polynomial):



Overfitting and Regularization

Overfitting: The error decreases in the training set but increases in the test set.



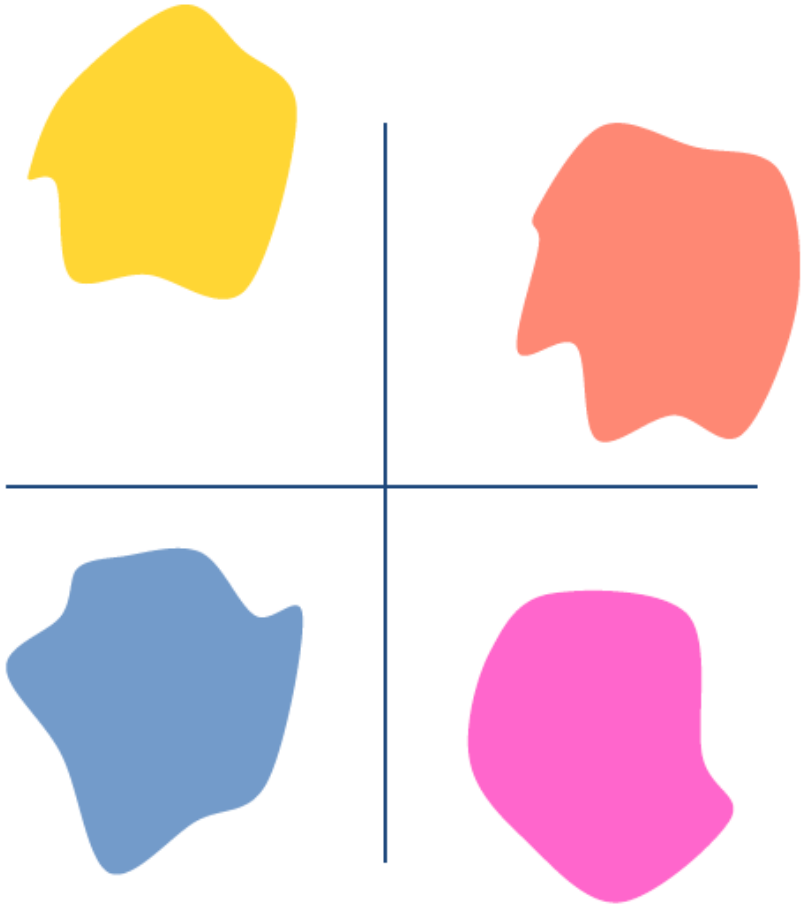
Regularization: Batch norm.

We deal with minibatches.

We assume that training data have similar distributions.

That means, minibatches are similar.

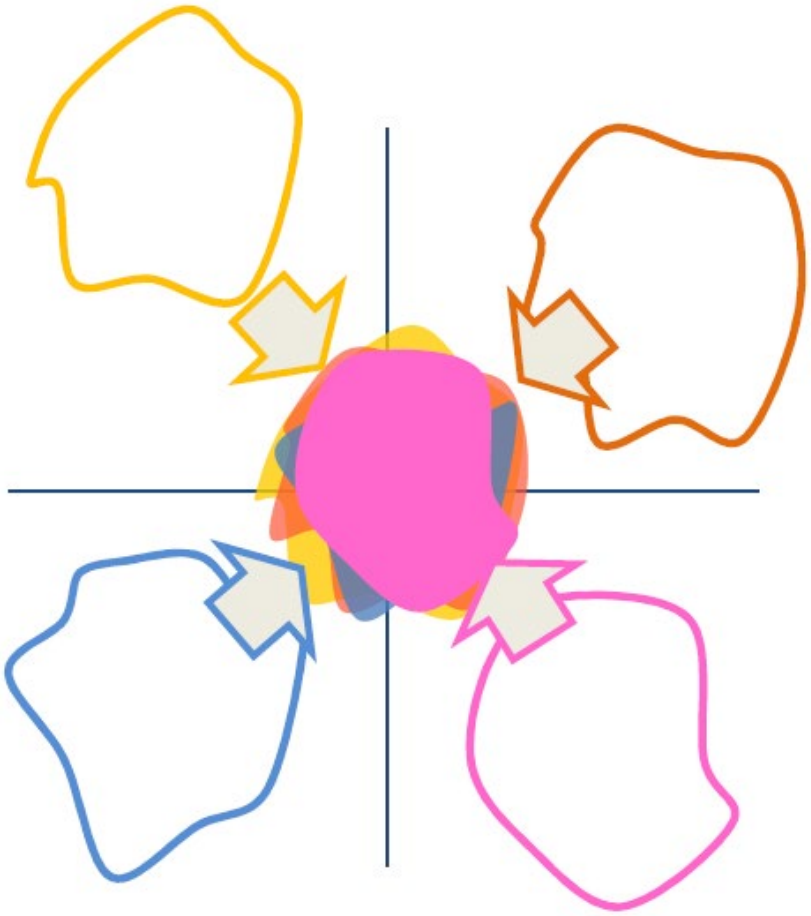
In reality, no! they are not similar.



Regularization: Batch norm.

We want to move all batches to locate at the “standard” location.

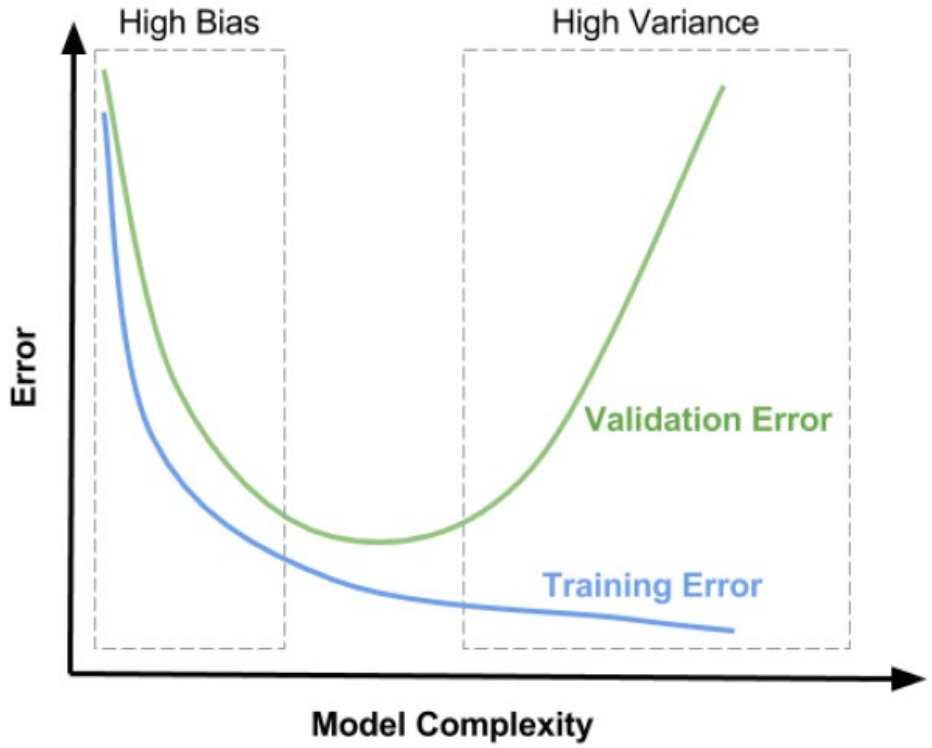
All batches to have a mean of 0 and unit standard deviation.



Regularization: Early Stopping

Stop training when performance on the validation set decreases.

We can create validation set as a subset of training set.

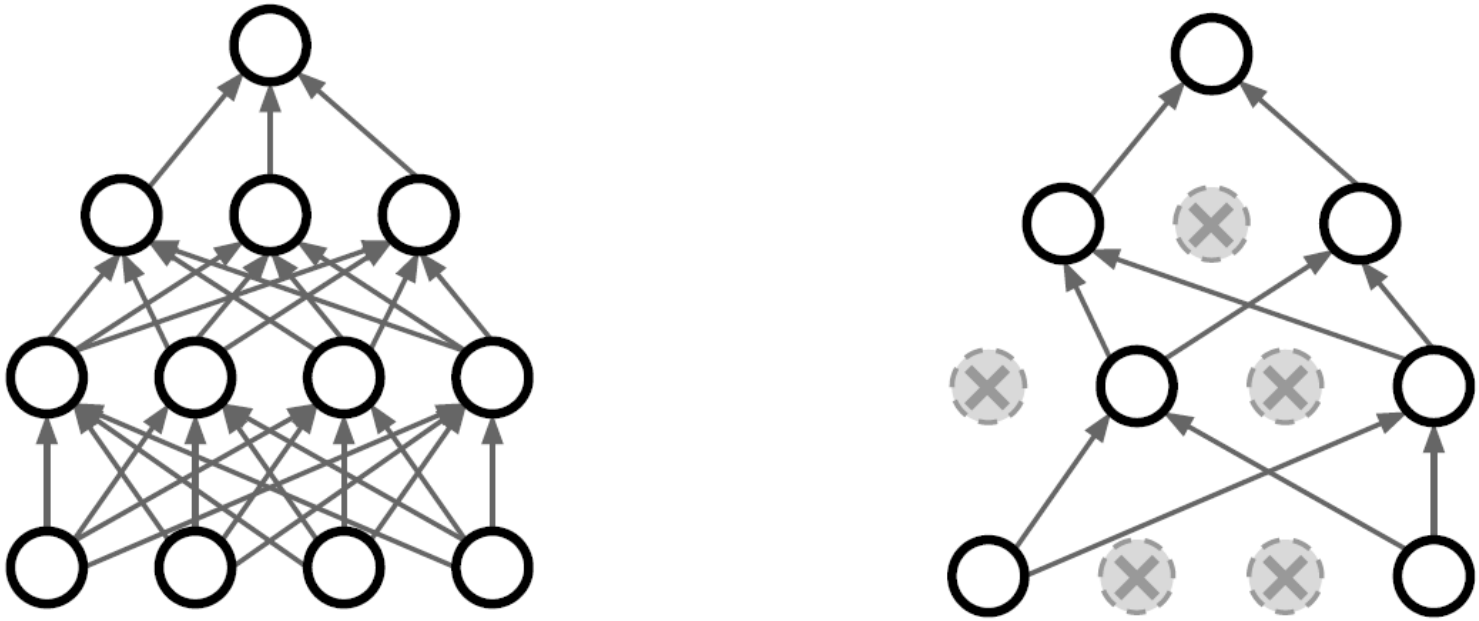


Regularization: Dropout

Randomly remove some nodes of the network during training, but not testing.

Drop rate

Compensation during testing



Regularization: Dropout

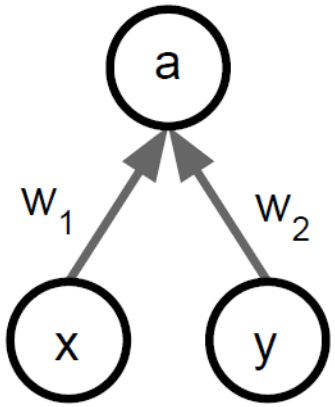
Why is it a good idea?

It forces the network to have a redundant representation.

In this way, we prevent to rely too much on specific features.

Dropout: Test

At testing phase, we do not drop the node. Instead, we adjust the value of output features by drop rate. (consistent with training phase)



Consider a single neuron.

At test time we have: $E[a] = w_1 x + w_2 y$

During training we have:

$$E[a] = \frac{1}{4}(w_1 x + w_2 y) + \frac{1}{4}(w_1 x + 0y) + \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2 y) = \frac{1}{2}(w_1 x + w_2 y)$$

At test time, **multiply** by dropout probability

Lab assignment & homework

Implement the network.

Your homework is to implement the mixup. TA will announce the detail instructions.