INT305 note

(Machine Learning)

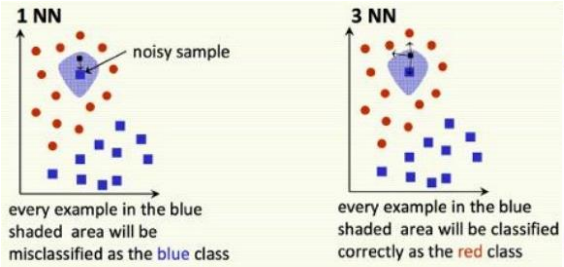
1 Introduction

1.1 Supervised learning (much of this course)

Task	Inputs	Labels
object recognition	image	object category
image captioning	image	caption
document classification	text	document category
speech-to-text	audio waveform	text
⋮	⋮	⋮

1.1.1 KNN

- Nearest neighbours sensitive to noise or mis-labelled data ("class noise").
- Smooth by having k nearest neighbours vote.

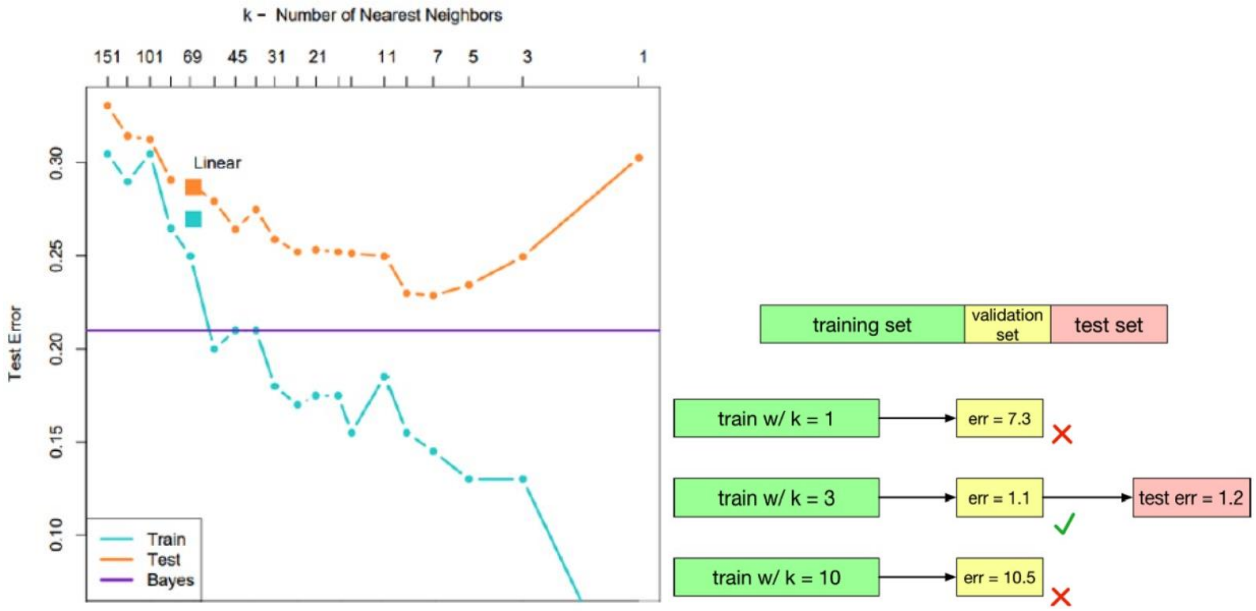


Algorithm (kNN):

1. Find k examples $\{\mathbf{x}^{(i)}, t^{(i)}\}$ closest to the test instance $\mathbf{x}$
2. Classification output is majority class

$$y = arg\max_{t^{(z)}} \sum_{i=1}^{k} \mathbb{I}(t^{(z)} = t^{(i)})$$

- Balancing hyperparameter k
  - ➢ Optimal choice of k depends on number of data points n.
  - ➢ Nice theoretical properties if $k \to \infty$ and $k/n \to 0$.
  - ➢ Rule of thumb: choose $k < \sqrt{n}$.
  - ➢ We can choose k using validation set.



2 Linear Methods for Regression, Optimization

Linear regression exemplifies recurring themes of this course:

- Choose a model and a loss function
- Formulate an optimization problem
- Solve the minimization problem using one of two strategies

- ➢ Direct solution (set derivatives to zero)
- ➢ Gradient descent

- Vectorize the algorithm, i.e. represents in terms of linear algebra
- Make a linear model more powerful using features
- Improve the generalization by adding a regularizer

2.1 Supervised Learning Setup

In supervised learning:

- There is input $\mathbf{x} \in \mathcal{X}$, typically a vector of features (or covariates)
- There is target $t \in \mathcal{T}$, (also called response, outcome, output, class)
- Objective is to learn a function $f: \mathcal{X} \to \mathcal{T}$ such that $t \approx y = f(\mathbf{x})$ based on some data $\mathcal{D} = \{(\mathbf{x}^{(i)}, t^{(i)}) \; for \; i = 1, 2, \cdots, N\}$

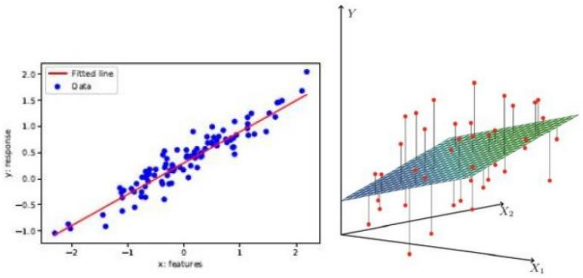
2.2 Linear Regression

2.2.1 Linear Regression Model

Model: In linear regression, we use a linear function of the features $\mathbf{x} = x_1, \cdots, x_D \in \mathbb{R}^D$ to make predictions y of the target value $t \in \mathbb{R}$:

$$y = f(\mathbf{x}) = \sum_j w_j x_j + b$$

- y is the prediction
- $\mathbf{w}$ is the weights
- b is the bias (or intercept)
- $\mathbf{w}$ and b together are the parameters
- We hope that our prediction is close to the target: $y \approx t$.



- If we have only 1 feature: $y = wx + b$ where $w, x, b \in \mathbb{R}$.
- y is linear in x.
- If we have only D features: $y = \mathbf{w}^\top \mathbf{x} + b$ where $\mathbf{w}, \mathbf{x} \in \mathbb{R}^D, b \in \mathbb{R}$
- y is linear in $\mathbf{x}$.

2.2.2 Linear Regression workflow

We have a dataset $\mathcal{D} = \{(\mathbf{x}^{(i)}, t^{(i)}) \; for \; i = 1, 2, \cdots, N\}$:

- $\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, \cdots x_D^{(i)})^\top \in \mathbb{R}^D$ are the inputs (e.g. age, height)
- $t^{(i)} \in \mathbb{R}$ is the target or response (e.g. income)
- Predict $t^{(i)}$ with a linear function of $\mathbf{x}^{(i)}$:
  - $t^{(i)} \approx y^{(i)} = \mathbf{w}^\top \mathbf{x}^{(i)} + b$
  - Different $\mathbf{w}$, b define different lines.
  - We want the "best" line $\mathbf{w}$, b .

2.2.3 Linear Regression Loss Function

- A loss function $\mathcal{L}(y,t)$ defines how bad it is if, for some example $\mathbf{x}$, the algorithm predicts y, but the target is actually t.
- Squared error loss function:

$$\mathcal{L}(y,t) = \frac{1}{2}(y-t)^2$$

- $y - t$ is the residual, and we want to make this small in magnitude.
- The 1/2 factor is just to make the calculations convenient.

• Cost function: loss function averaged over all training examples.

$$\mathcal{J}(\mathbf{w},b)=\frac{1}{2N}\sum_{i=1}^{N}(y^{(i)}-t^{(i)})^2=\frac{1}{2N}\sum_{i=1}^{N}(\mathbf{w}^\intercal\mathbf{x}^{(i)}+b-t^{(i)})^2$$

2.2.3 Linear Regression Vectorization

But if we expand $y^{(i)}$, it will get messy:

$$\frac{1}{2N}\sum_{i=1}^{N}(\sum_{j=1}^{D}\left(w_jx_j^{(i)}\right)+b-t^{(i)})^2$$

Vectorize algorithms by expressing them in terms of vectors and matrices:

$$\mathbf{w}=(w_1,\cdots w_D)^\intercal \qquad \mathbf{x}=(x,\cdots x_D)^\intercal$$
$$y=\mathbf{w}^\intercal\mathbf{x}+b$$

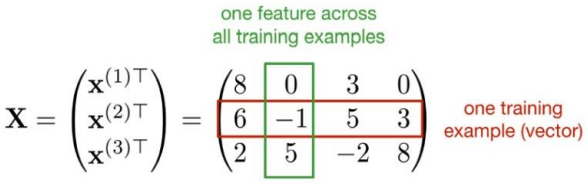
Python code:

```
y = b
for j in range(M):
    y += w[j] * x[j]
```

=

```
y = np.dot(w, x) + b
```

Organize all the training examples into a design matrix $\mathbf{X}$ with one row per training example, and all the targets into the target vector $\mathbf{t}$:



$$\mathbf{X}=\begin{pmatrix}\mathbf{x}^{(1)\top}\\ \mathbf{x}^{(2)\top}\\ \mathbf{x}^{(3)\top}\end{pmatrix}=\begin{pmatrix}8&0&3&0\\ 6&-1&5&3\\ 2&5&-2&8\end{pmatrix}$$

Computing the predictions for the whole dataset:

$$\mathbf{X}\mathbf{w}+b\mathbf{1}=\begin{pmatrix}\mathbf{w}^T\mathbf{x}^{(1)}+b\\ \vdots\\ \mathbf{w}^T\mathbf{x}^{(N)}+b\end{pmatrix}=\begin{pmatrix}y^{(1)}\\ \vdots\\ y^{(N)}\end{pmatrix}=\mathbf{y}$$

Computing the squared error cost across the whole dataset:

$$\mathbf{y}=\mathbf{X}\mathbf{w}+b\mathbf{1}$$
$$\mathcal{J}=\frac{1}{2N}\|\mathbf{y}-\mathbf{t}\|^2$$

We can also add a column of 1's to design matrix, combine the bias and the weights, and conveniently write:

$$\mathbf{X}=\begin{bmatrix}1&[\mathbf{x}^{(1)}]^\intercal\\ 1&[\mathbf{x}^{(2)}]^\intercal\\ 1&\vdots\end{bmatrix}\in\mathbb{R}^{N\times(D+1)}\quad\text{and}\quad\mathbf{w}=\begin{bmatrix}b\\ w_1\\ w_2\\ \vdots\end{bmatrix}\in\mathbb{R}^{D+1}$$

Then, our predictions reduce to $\mathbf{y}=\mathbf{X}\mathbf{w}$.

2.3 Direct Solution •

2.3.1 Linear Algebra •

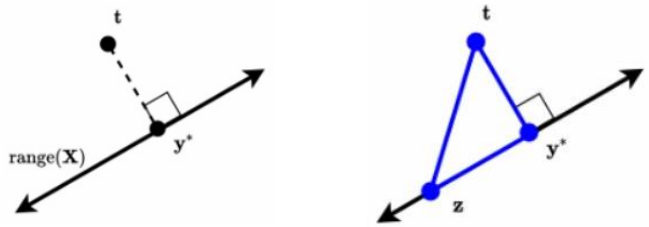
• We seek $\mathbf{w}$ to minimize $||\mathbf{X}\mathbf{w}-\mathbf{t}||^2$, or equivalently $||\mathbf{X}\mathbf{w}-\mathbf{t}||$

• range $\mathbf{X}=\{\mathbf{X}\mathbf{w}|\mathbf{w}\in\mathbb{R}^D\}$ is a D-dimensional subspace of $\mathbb{R}^N$

• Recall that the closest point $\mathbf{y}^*=\mathbf{X}\mathbf{w}^*$ in subspace range($\mathbf{X}$) of $\mathbb{R}^N$ to arbitrary point $\mathbf{t}\in\mathbb{R}^N$ is found by orthogonal projection.

• We have $(\mathbf{y}^*-\mathbf{t})\perp\mathbf{X}\mathbf{w},\forall\mathbf{w}\in\mathbb{R}^D$

• $\mathbf{y}^*$ is the closest point to $\mathbf{t}$



2.3.2 Calculus •

• Partial derivative: derivative of a multivariate function with respect to one of its arguments.

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

• To compute, take the single variable derivative, pretending the other arguments are constant.

• Example: partial derivatives of the prediction y.

$$\frac{\partial y}{\partial w_j} = \frac{\partial}{\partial w_j}\left[\sum_{j'} w_{j'} x_{j'} + b\right] \qquad \frac{\partial y}{\partial b} = \frac{\partial}{\partial b}\left[\sum_{j'} w_{j'} x_{j'} + b\right]$$
$$= x_j \qquad = 1$$

• For loss derivatives, apply the chain rule:

$$\frac{\partial \mathcal{L}}{\partial w_j} = \frac{d\mathcal{L}}{dy}\frac{\partial y}{\partial w_j} \qquad \frac{\partial \mathcal{L}}{\partial b} = \frac{d\mathcal{L}}{dy}\frac{\partial y}{\partial b}$$
$$= \frac{d}{dy}\left[\frac{1}{2}(y-t)^2\right] \cdot x_j \qquad = y - t$$
$$= (y-t)x_j$$

• For cost derivatives, use linearity and average over data points:

$$\frac{\partial \mathcal{J}}{\partial w_j} = \frac{1}{N}\sum_{i=1}^{N} (y^{(i)} - t^{(i)}) x_j^{(i)} \qquad \frac{\partial \mathcal{J}}{\partial b} = \frac{1}{N}\sum_{i=1}^{N} y^{(i)} - t^{(i)}$$

• Minimum must occur at a point where partial derivative are zero.

$$\frac{\partial \mathcal{J}}{\partial w_j} = 0 \ (\forall j), \qquad \frac{\partial \mathcal{J}}{\partial b} = 0.$$

(If $\partial \mathcal{J} / \partial w_j \neq 0$, you could reduce the cost by changing w_j)

• We call the vector of partial derivatives the gradient.

• Thus, the "gradient of f: $\mathbb{R}^D \to \mathbb{R}$", denoted $\nabla f(\mathbf{w})$, is:

$$\left(\frac{\partial}{\partial w_1} f(\mathbf{w}), \cdots, \frac{\partial}{\partial w_D} f(\mathbf{w})\right)^\intercal$$

• The gradient points in the direction of the greatest rate of increase.

• Analogue of second derivative (the "Hessian" matrix):

$$\nabla^2 f(\mathbf{w}) \in \mathbb{R}^{D \times D} \text{ is a matrix with} [\nabla^2 f(\mathbf{w})]_{ij} = \frac{\partial^2}{\partial w_i \partial w_j} f(\mathbf{w})$$

• We seek $\mathbf{w}$ to minimize $\mathcal{J}(\mathbf{w}) = ||\mathbf{X}\mathbf{w} - \mathbf{t}||^2 / 2$

• Taking the gradient with respect to $\mathbf{w}$ we get:

$$\nabla_{\mathbf{w}} \mathcal{J}(w) = \mathbf{X}^\intercal \mathbf{X}\mathbf{w} - \mathbf{X}^\intercal \mathbf{t} = 0$$

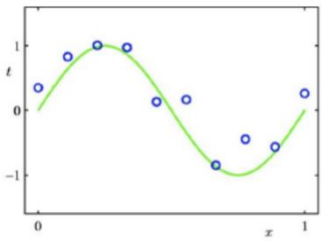
$$\mathbf{w}^* = (\mathbf{X}^\intercal \mathbf{X})^{-1} \mathbf{X}^\intercal \mathbf{t}$$

• Linear regression is one of only a handful of models in this course that permit direct solution.

2.4 Polynomial Feature Mapping

2.4.1 Introduction

The relation between the input and output may not be linear. But we can still use linear regression by mapping the input features to another space using feature mapping (or basis expansion). $\varphi(\mathbf{x}) : \mathbb{R}^D \to \mathbb{R}^d$ and treat the mapped feature (in $\mathbb{R}^d$) as the input of a linear regression procedure.



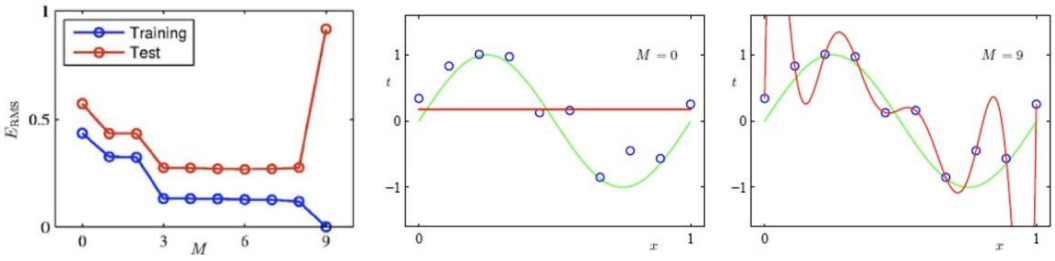
Find the data using a degree-M polynomial function of the form:

$$y = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{i=0}^{M} w_i x^i$$

- Here the feature mapping is $\varphi(x) = [1, x, x^2, \ldots, x^M]^\top$.
- We can use linear regression to find $\mathbf{w}$, since $y = \varphi(x)^\top \mathbf{w}$ is linear with $w_0, w_1, \ldots, w_M$

2.4.2 Model Complexity and Generalization

- Underfitting (M=0): model is too simple – does not fit the data.
- Overfitting (M=9): model is too complex – fits perfectly.



2.4.3 L^2 Regularization

Regularizer: a function that quantifies how much we prefer one hypothesis VS another.

- We can encourage the weights to be small by choosing as our regularizer the L^2 penalty.

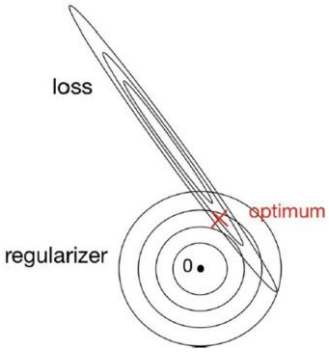
$$\mathcal{R}(\mathbf{w}) = \frac{1}{2}\sum_j w_j^2$$

(To be precise, the L^2 norm is Euclidean distance, we're regularizing the squared L^2 norm)

- The regularized cost function makes a tradeoff between fit to the data and the norm of the weights.

$$\mathcal{J}_{reg}(\mathbf{w}) = \mathcal{J}(\mathbf{w}) + \lambda \mathcal{R}(\mathbf{w}) = \mathcal{J}(\mathbf{w}) + \frac{\lambda}{2}\sum_j w_j^2$$

- If you fit training data poorly, $\mathcal{J}$ is large. If your optimal weights have high values, $\mathcal{R}$ is large.
- Large λ penalize wight values more.
- Like M, λ is a hyperparameter we can tune with a validation set.



2.4.4 L^2 Regularized Least Squares: Ridge regression

For the least squares problem, we have $\mathcal{J}(\mathbf{w}) = \frac{1}{2N}\|\mathbf{X}\mathbf{w} - \mathbf{t}\|^2$

- When $\lambda > 0$ (with regularization), regularized cost gives:

$$\mathbf{w}_\lambda^{\text{Ridge}} = \underset{\mathbf{w}}{\operatorname{argmin}}\, \mathcal{J}_{\text{reg}}(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}}\, \frac{1}{2N}\|\mathbf{X}\mathbf{w} - \mathbf{t}\|_2^2 + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$
$$= (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{t}$$

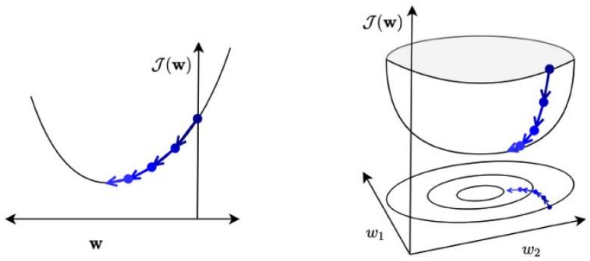
- The case $\lambda = 0$ (no regularization) reduces to least squares solution!

2.5 Gradient Descent

2.5.1 Concepts

- Many times, we do not have a direct solution: Taking derivatives of $\mathcal{J}$ w.r.t $\mathbf{w}$ and setting them to 0 doesn't have an explicit solution.

- Gradient descent is an iterative algorithm, which means we apply an update repeatedly until some criterion is met.
- We initialize the weights to something reasonable (e.g., all zeros) and repeatedly adjust them in the direction of steepest descent.



(就是等到斜率为 0 即为最优解)

- Observe:
  - ➢ If $\partial\mathcal{J}/\partial w_j > 0$, then increasing w_j increases $\mathcal{J}$.
  - ➢ If $\partial\mathcal{J}/\partial w_j < 0$, then increasing w_j decreases $\mathcal{J}$
- The following update always decreases the cost function for small enough α (unless $\partial\mathcal{J}/\partial w_j = 0$):
- $\alpha > 0$ is a learning rate (or step size). The larger it is, the faster $\mathbf{w}$ changes (but values are typically small).
- This gets its name from the gradient:

$$\nabla_{\mathbf{w}}\mathcal{J} = \frac{\partial\mathcal{J}}{\partial\mathbf{w}} = \begin{pmatrix} \frac{\partial\mathcal{J}}{\partial w_1} \\ \vdots \\ \frac{\partial\mathcal{J}}{\partial w_D} \end{pmatrix}$$

- Update rule in vector form:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha\frac{\partial\mathcal{J}}{\partial\mathbf{w}}$$

- And for linear regression we have:

$$\mathbf{w} \leftarrow \mathbf{w} - \frac{\alpha}{N}\sum_{i=1}^{N}\left(y^{(i)} - t^{(i)}\right)\mathbf{x}^{(i)}$$

- So gradient descent updates $\mathbf{w}$ in the direction of fastest decrease.
- Observe that once it converges, we get a critical point. i.e. $\frac{\partial\mathcal{J}}{\partial\mathbf{w}} = \mathbf{0}$

2.5.2 Gradient Descent for Linear Regression

- Why gradient descent, if we can find the optimum directly?
  - ➢ gradient descent can be applied to a much broader set of models
  - ➢ gradient descent can be easier to implement than direct solutions
  - ➢ For regression in high-dimensional space, gradient descent is more efficient than direct solution

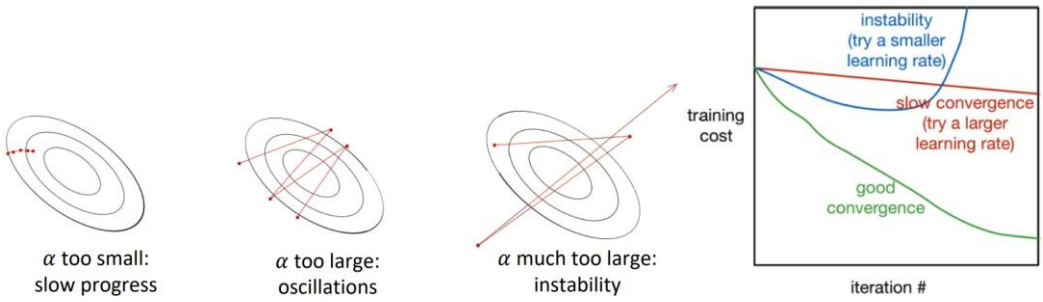
2.5.3 Gradient Descent under the L^2 Regularization

- The gradient descent update to minimize the L^2 regularized cost $\mathcal{J} + \lambda\mathcal{R}$ results in weight decay:

$$\begin{aligned}\mathbf{w} &\leftarrow \mathbf{w} - \alpha\frac{\partial}{\partial\mathbf{w}}(\mathcal{J} + \lambda\mathcal{R}) \\ &= \mathbf{w} - \alpha\left(\frac{\partial\mathcal{J}}{\partial\mathbf{w}} + \lambda\frac{\partial\mathcal{R}}{\partial\mathbf{w}}\right) \\ &= \mathbf{w} - \alpha\left(\frac{\partial\mathcal{J}}{\partial\mathbf{w}} + \lambda\mathbf{w}\right) \\ &= (1 - \alpha\lambda)\mathbf{w} - \alpha\frac{\partial\mathcal{J}}{\partial\mathbf{w}}\end{aligned}$$

2.5.4 Learning Rate (Step Size)

- In gradient descent, the learning rate α is a hyperparameter we need to tune.
- Good values are typically between 0.001 and 0.1.



- To diagnose optimization problems, it's useful to look at training curves: plot the training cost as a function of iteration.
- Warning: in general, it's very hard to tell from the training curves whether an optimizer has converged. They can reveal major problems, but they can't guarantee convergence.

2.5.5 Stochastic Gradient Descent

Stochastic gradient descent (SGD): update the parameters based on the gradient for a single training example

1- Choose i uniformly at random

$$2\text{-}\ \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \frac{\partial \mathcal{L}^{(i)}}{\partial \boldsymbol{\theta}}$$

- Cost of each SGD update is independent of N!
- SGD can make significant progress before even seeing all the data!
- Mathematical justification: if you sample a training example uniformly at random, the stochastic gradient is an unbiased estimate of the batch gradient:

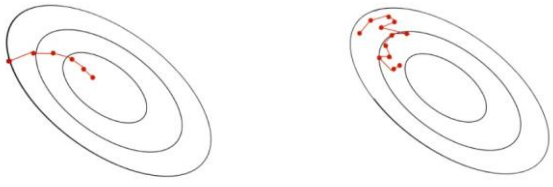
$$\mathbb{E}\left[\frac{\partial \mathcal{L}^{(i)}}{\partial \boldsymbol{\theta}}\right] = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial \mathcal{L}^{(i)}}{\partial \boldsymbol{\theta}} = \frac{\partial \mathcal{J}}{\partial \boldsymbol{\theta}}$$

2.5.6 Mini-batch Stochastic Gradient Descent

- Problems with using single training example to estimate gradient:
  - ➢ Variance in the estimate may be high
  - ➢ We can't exploit efficient vectorized operations
- Compromise approach:
  - ➢ Compute the gradients on a randomly chosen medium-sized set of training examples $\mathcal{M} \subset \{1, \cdots, N\}$ called a mini-batch.
- Stochastic gradients computed on larger mini-batches have smaller variance.
- The mini-batch size $|\mathcal{M}|$ is a hyperparameter that needs to be set.
  - ➢ Too large: requires more compute: e.g., it takes more memory to store the activations, and longer to compute each gradient update
  - ➢ Too small: can't exploit vectorization, has high variance
  - ➢ A reasonable value might be $|\mathcal{M}| = 100$.

2.5.7 Comparation

- Batch gradient descent moves directly downhill (locally speaking).
- SGD takes steps in a noisy direction, but moves downhill on average.



batch gradient descent　　stochastic gradient descent

▲ Batch Gradient Descent，全批量梯度下降，是最原始的形式，它是指在每一次迭代时使用所有样本来进行梯度的更新。优点是全局最优解，易于并行实现；缺点是当样本数目很大时，训练过程会很慢。

▲ Stochastic Gradient Descent，随机梯度下降，是指在每一次迭代时使用一个样本来进行参数的更新。优点是训

练速度快；缺点是准确度下降，并且可能无法收敛或者在最小值附近震荡。
▲ Mini-Batch Gradient Descent，小批量梯度下降，是对上述两种方法的一个折中办法。它是指在每一次迭代时使用一部分样本来进行参数的更新。这种方法兼顾了计算速度和准确度。

3 Linear Classifiers, Logistic Regression, Multiclass Classification

3.1 Binary linear classification

- Classification: given a D-dimensional input $\mathbf{x} \in \mathbb{R}^D$ predict a discrete-valued target
- Binary: predict a binary target $t \in \{0,1\}$
  - ➢ Training examples with $t = 1$ are called positive examples, and training examples with $t = 0$ are called negative examples.
  - ➢ $t \in \{0,1\}$ or $t \in \{-1, +1\}$ is for computational convenience.
- Linear: model prediction y is a linear function of $\mathbf{x}$, followed by a threshold r:

$$z = \mathbf{w}^\intercal \mathbf{x} + b$$

$$y = \begin{cases} 1 & if\ z \geq r \\ 0 & if\ z < r \end{cases}$$

- Eliminating the threshold: We can assume without loss of generality (WLOG) that the threshold $r = 0$

$$\mathbf{w}^\intercal \mathbf{x} + b \geq r \iff \mathbf{w}^\intercal \mathbf{x} + \underbrace{b - r}_{\triangleq w_0} \geq 0$$

- Eliminating the bias: Add a dummy feature x_0 which always takes the value 1. the weight $w_0 = b$ is equivalent to a bias (same as linear regression)

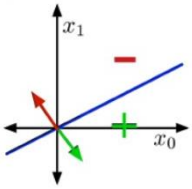
- Simplified model: receive input $\mathbf{x} \in \mathbb{R}^{D+1}$ with $x_0 = 1$

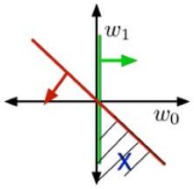
$$z = \mathbf{w}^\intercal \mathbf{x}$$

- Example:

x_0	x_1	t
1	0	1
1	1	0

  - Suppose this is our training set, with the dummy feature $x0$ included.
  - Which conditions on $w0$, $w1$ guarantee perfect classification?
    - ➢ When $x1 = 0$, need: $z = w0x0 + w1x1 \geq 0 \longleftrightarrow w0 \geq 0$
    - ➢ When $x1 = 1$, need: $z = w0x0 + w1x1 < 0 \longleftrightarrow w0 + w1 < 0$
    - ➢ Example solution: $w0 = 1$, $w1 = -2$



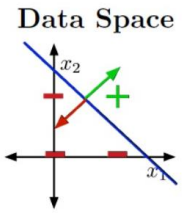


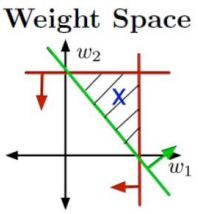
$w_0 \geq 0$

$w_0 + w_1 < 0$

  - Training examples are points
  - Weights (hypotheses) $\mathbf{w}$ can be represented by half-spaces: $H+ = \{\mathbf{x}: \mathbf{w}^\intercal \mathbf{x} \geq 0\}$, $H- = \{\mathbf{x}: \mathbf{w}^\intercal \mathbf{x} < 0\}$
  - The boundary is the decision boundary: $\{\mathbf{x}: \mathbf{w}^\intercal \mathbf{x} = 0\}$
  - If the training examples can be perfectly separated by a linear decision rule, we say data is linearly separable.
  - Weights (hypotheses) $\mathbf{w}$ are points
  - Each training example $\mathbf{x}$ specifies a half-space $\mathbf{w}$ must lie in to be correctly classified: $\mathbf{w}^\intercal \mathbf{x} \geq 0$ if $t = 1$
    - ➢ $x0 = 1$, $x1 = 0$, $t = 1 \longrightarrow (w0, w1) \in \mathbf{w}: w0 \geq 0$
    - ➢ $x0 = 1$, $x1 = 1$, $t = 0 \longrightarrow (w0, w1) \in \mathbf{w}: w0 + w1 < 0$
  - The region satisfying all the constraints is the feasible region; if this region is nonempty, the problem is

feasible, otherwise it is infeasible.





- Slice for $x_0 = 1$ and
- Example sol: $w_0 = -1.5, w_1 = 1, w_2 = 1$
- Decision boundary:

$$w_0 x_0 + w_1 x_1 + w_2 x_2 = 0$$
$$\Longrightarrow -1.5 + x_1 + x_2 = 0$$

- Slice for $w_0 = -1.5$ for the constraints
- $w_0 < 0$
- $w_0 + w_2 < 0$
- $w_0 + w_1 < 0$
- $w_0 + w_1 + w_2 \geq 0$

3.2 Towards Logistic Regression

Define loss function then try to minimize the resulting cost function

Attempt 1: 0-1 loss

$$\mathcal{L}_{0-1}(y,t) = \begin{cases} 0 & if\ y = t \\ 1 & if\ y \neq t \end{cases}$$
$$= \mathbb{I}[y \neq t]$$

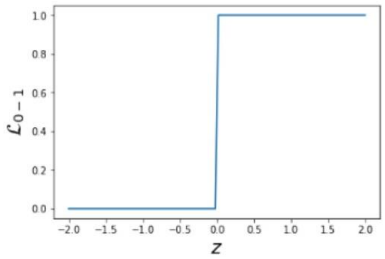
• Usually, the cost $\mathcal{J}$ is the averaged loss over training examples; for 0-1 loss, this is the misclassification rate:

$$\mathcal{J} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}\left[y^{(i)} \neq t^{(i)}\right]$$

• Minimum of a function will be at its critical points, use Chain rule to find the critical point of 0-1 loss

$$\frac{\partial \mathcal{L}_{0-1}}{\partial w_j} = \frac{\partial \mathcal{L}_{0-1}}{\partial z}\frac{\partial z}{\partial w_j}$$

• $\partial \mathcal{L}_{0-1}/\partial_z$ is zero everywhere it's defined:



➢ $\partial \mathcal{L}_{0-1}/\partial w_j$ = 0 means that changing the weights by a very small amount probably has no effect on the loss.

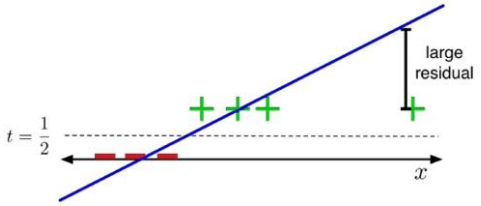
➢ Almost any point has 0 gradient!

Attempt 2: Linear Regression

$$z = \mathbf{w}^{\mathrm{T}}\mathbf{x}$$
$$\mathcal{L}_{\mathrm{SE}}(z,t) = \frac{1}{2}(z-t)^2$$

• Doesn't matter that the targets are actually binary. Treat them as continuous values.

• For this loss function, it makes sense to make final predictions by thresholding z at 1/2



• The loss function hates when you make correct predictions with high confidence!

• It $t = 1$, it's more unhappy about $z = 10$ than $z = 0$.

Attempt 3: Logistic Activation Function

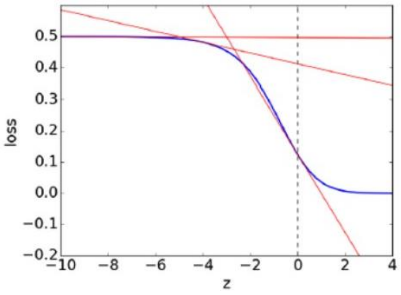
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

• $\sigma^{-1}(y) = \log(y/(1 - y))$ is called the logit.

• A linear model with a logistic nonlinearity is known as log-linear:

$$z = \mathbf{w}^\intercal \mathbf{x}$$
$$y = \sigma(z)$$
$$\mathcal{L}_{\text{SE}}(y, t) = \frac{1}{2}(y - t)^2$$

• Used in this way, σ is called an activation function.



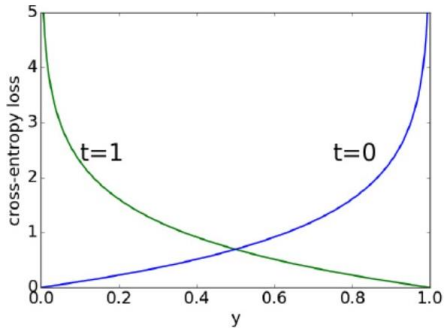
$$\frac{\partial \mathcal{L}}{\partial w_j} = \frac{\partial \mathcal{L}}{\partial z}\frac{\partial z}{\partial w_j}$$

(plot of $\mathcal{L}_{\text{SE}}$ as a function of z, assuming $t = 1$)

- For $z \ll 0$, we have $\sigma(z) \approx 0$.
- $\partial\mathcal{L}/\partial z \approx 0$ (check) ➔$\partial\mathcal{L}/\partial w_j \approx 0$ ➔ derivative w.r.t. w_j is small ➔ w_j is like a critical point.
- If the prediction is really wrong, you should be far from a critical point (which is your candidate solution).

• Because $y \in 0, 1$, we can interpret it as the estimated probability that $t = 1$. If $t = 0$, then we want to heavily penalize $y \approx 1$.

• Cross-entropy loss (aka log loss) captures this intuition:



$$\mathcal{L}_{\text{CE}}(y, t) = \begin{cases} -\log y & \text{if } t = 1 \\ -\log(1 - y) & \text{if } t = 0 \end{cases}$$
$$= -t\log y - (1 - t)\log(1 - y)$$

• The logistic loss is a convex function in $\mathbf{w}$, so let's consider the gradient descent method.

➢ Recall: we initialize the weights to something reasonable and repeatedly adjust them in the direction of steepest descent.

➢ A standard initialization is $\mathbf{w} = 0$.

$$\mathcal{L}_{\text{CE}}(y, t) = -t\log y - (1 - t)\log(1 - y)$$
$$y = 1/(1 + e^{-z}) \text{ and } z = \mathbf{w}^\intercal \mathbf{x}$$
$$\frac{\partial \mathcal{L}_{\text{CE}}}{\partial w_j} = \frac{\partial \mathcal{L}_{CE}}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial w_j} = \left(-\frac{t}{y} + \frac{1 - t}{1 - y}\right) \cdot y(1 - y) \cdot x_j$$
$$= (y - t)x_j$$

Gradient descent (coordinate-wise) update to find the weights of logistic regression:

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

Gradient descent updates for Linear regression and Logistic regression (both examples of generalized linear models):

$$\mathbf{w} \leftarrow \mathbf{w} - \frac{\alpha}{N}\sum_{i=1}^{N}(y^{(i)} - t^{(i)})\mathbf{x}^{(i)}$$

3.3 Multiclass Classification and Softmax Regression

3.3.1 Multiclass Classification

- Classification tasks with more than two categories
- Targets form a discrete set $\{1, \cdots, K\}$.
- It's often more convenient to represent them as one-hot vectors, or a one-of-K encoding:

$$\mathbf{t} = \underbrace{(0, \cdots, 0, 1, 0 \cdots, 0)}_{\text{entry } k \text{ is 1}} \in \mathbb{R}^K$$

- We can start with a linear function of the inputs.

$$z_k = \sum_{j=1}^{D} w_{kj}x_j + b_k \quad \text{for} \quad k = 1,2,\cdots,K$$

- Now there are D input dimensions and K output dimensions, so we need $K \times D$ weights, which we arrange as a weight matrix $\mathbf{W}$.
- Also, we have a K-dimensional vector $\mathbf{b}$ of biases. Then eliminate the bias $\mathbf{b}$ by taking $\mathbf{W} \in \mathbb{R}^{K\times(D+1)}$ and adding a dummy variable $x_0 = 1$.

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b} \text{ or with dummy } x_0 = 1 \ \mathbf{z} = \mathbf{W}\mathbf{x}$$

- We can interpret the magnitude of z_k as a measure of how much the model prefers $k$as its prediction to turn this linear prediction into a one-hot prediction.

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

3.3.2 Softmax Regression

- We need to soften our predictions for the sake of optimization.
- We want soft predictions that are like probabilities, i.e., $0 \le y_k \le 1$ and $\sigma k\ y_k = 1$.
- A natural activation function to use is the softmax function, a multivariable generalization of the logistic function:

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

  - ➢ Outputs can be interpreted as probabilities (positive and sum to 1)
  - ➢ If zk is much larger than the others, then $\text{softmax}(\mathbf{z})_k \approx 1$ and it behaves like argmax.
- If a model outputs a vector of class probabilities, we can use cross-entropy as the loss function:

$$\mathcal{L}_{\text{CE}}(\mathbf{y}, \mathbf{t}) = -\sum_{k=1}^{K} t_k \log y_k$$
$$= -\mathbf{t}^{\top}(\log \mathbf{y})$$

- Just like with logistic regression, we typically combine the softmax and cross-entropy into a softmax-cross-entropy function.
- Softmax regression (with dummy $x0 = 1$):

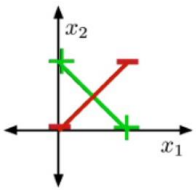
$$\mathbf{z} = \mathbf{W}\mathbf{x}$$
$$\mathbf{y} = \text{softmax}(\mathbf{z})$$
$$\mathcal{L}_{\text{CE}} = -\mathbf{t}^{\top}(\log \mathbf{y})$$

- Gradient descent updates can be derived for each row of $\mathbf{W}$:

$$\frac{\partial \mathcal{L}_{\text{CE}}}{\partial \mathbf{w}_k} = \frac{\partial \mathcal{L}_{\text{CE}}}{\partial z_k} \cdot \frac{\partial z_k}{\partial \mathbf{w}_k} = (y_k - t_k) \cdot \mathbf{x}$$

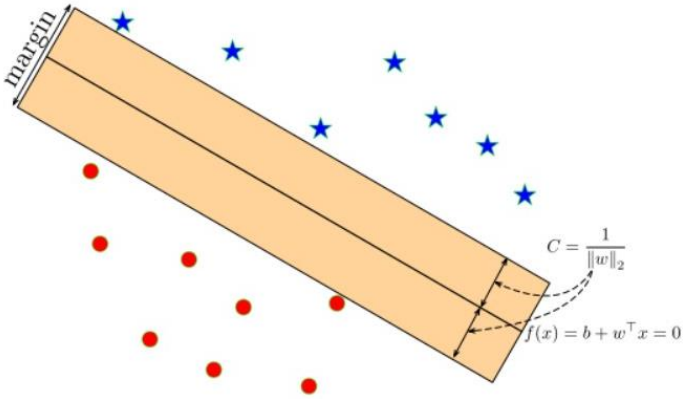
$$\mathbf{w}_k \leftarrow \mathbf{w}_k - \alpha \frac{1}{N} \sum_{i=1}^{N} \left(y_k^{(i)} - t_k^{(i)}\right) \mathbf{x}^{(i)}$$

- Similar to linear/logistic reg (no coincidence) (verify the update)
- Sometimes we can overcome the nonlinear problem with feature maps:

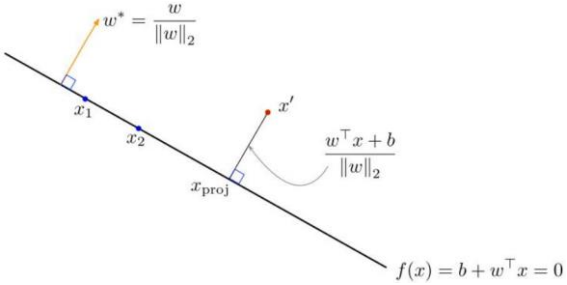


4 SVM, SVM Loss and Softmax Loss

4.1 Optimal Separating Hyperplane



- Concept: A hyperplane that separates two classes and maximizes the distance to the closest point from either class, i.e., maximize the margin of the classifier.
- Intuitively, ensuring that a classifier is not too close to any data points leads to better generalization on the test data.



4.1.1 Geometry of Points and Planes

- Recall that the decision hyperplane is orthogonal (perpendicular) to **w**.
- The vector **w*** = $\frac{\mathbf{w}}{||\mathbf{w}||2}$ is a unit vector pointing in the same direction as **w**.
- The same hyperplane could equivalently be defined in terms of **w***
- The (signed) distance of a point x' to the hyperplane is:

$$\frac{\mathbf{w}^\top \mathbf{x}' + b}{\|\mathbf{w}\|_2}$$

4.1.2 Maximizing Margin as an Optimization Problem

- Recall: the classification for the i-th data point is correct when:

$$\text{sign}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) = t^{(i)}$$

- This can be rewritten as:

$$t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) > 0$$

- Enforcing a margin of C:

$$t^{(i)} \cdot \underbrace{\frac{(\mathbf{w}^\top \mathbf{x}^{(i)} + b)}{\|\mathbf{w}\|_2}}_{\text{signed distance}} \geq C$$

- Max-margin objective:

$$\max_{\mathbf{w},b} C$$
$$\text{s.t.} \quad \frac{t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b)}{\|\mathbf{w}\|_2} \geq C \qquad i = 1, \ldots, N$$

- Plug in C = 1/||w||$_2$ and simplify:

$$\underbrace{\frac{t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b)}{\|\mathbf{w}\|_2} \geq \frac{1}{\|\mathbf{w}\|_2}}_{\text{geometric margin constraint}} \iff \underbrace{t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) \geq 1}_{\text{algebraic margin constraint}}$$

- Equivalent optimization objective:

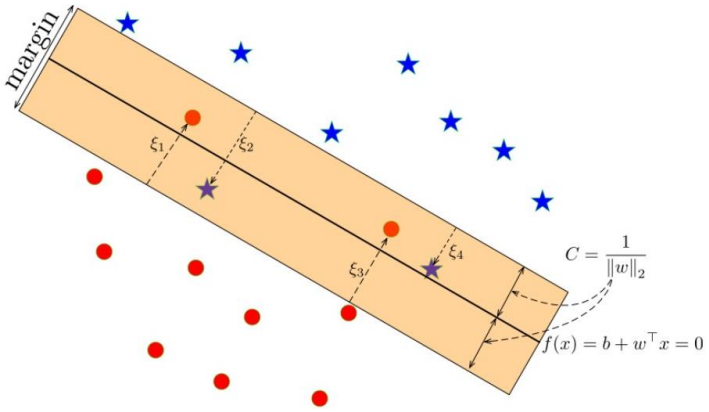
$$\min \|\mathbf{w}\|_2^2$$
$$\text{s.t.} \quad t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) \geq 1 \qquad i = 1, \ldots, N$$

4.2 Support Vector Machine

4.2.1 Concepts

- Observe: if the margin constraint is not tight for $\mathbf{x}^{(i)}$, we could remove it from the training set and the optimal $\mathbf{w}$ would be the same.
- The important training examples are the ones with algebraic margin 1, and are called support vectors
- Hence, this algorithm is called the (hard) Support Vector Machine (SVM) (or Support Vector Classifier).
- SVM-like algorithms are often called max-margin or large-margin.

4.2.2 Maximizing Margin for Non-Separable Data Points



- Main idea:
  - Allow some points to be within the margin or even be misclassified; were present this with slack variables ξ_i.
  - But constrain or penalize the total amount of slack.
- Soft margin constraint:

For $\xi_i \geqslant 0$: $$\frac{t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b)}{\|\mathbf{w}\|_2} \geq C(1 - \xi_i)$$

- Reduce ξ_i
- Soft-margin SVM objective:

$$\min_{\mathbf{w},b,\xi} \frac{1}{2}\|\mathbf{w}\|_2^2 + \gamma \sum_{i=1}^{N} \xi_i$$
$$\text{s.t.} \quad t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) \geq 1 - \xi_i \qquad i = 1, \ldots, N$$
$$\xi_i \geq 0 \qquad i = 1, \ldots, N$$

- γ is a hyperparameter that trades off the margin with the amount of slack
  - For $\gamma = 0$, we'll get $\mathbf{w} = 0$.
  - As γ -> ∞ we get the hard-margin objective.
- Note: it is also possible to constrain $\sum_i \xi^i$ instead of penalizing it.

4.2.3 From Margin Violation to Hinge Loss

Let's simplify the soft margin constraint by eliminating ξ_i. Recall:

$$t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) \geq 1 - \xi_i \qquad i = 1, \ldots, N$$
$$\xi_i \geq 0 \qquad i = 1, \ldots, N$$

- Rewrite as $\xi_i \geq 1 - t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b)$.
- **Case 1:** $1 - t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) \leq 0$
  - The smallest non-negative ξ_i that satisfies the constraint is $\xi_i = 0$.
- **Case 2:** $1 - t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) > 0$
  - The smallest ξ_i that satisfies the constraint is $\xi_i = 1 - t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b)$.
- Hence, $\xi_i = \max\{0, 1 - t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b)\}$.
- Therefore, the slack penalty can be written as

$$\sum_{i=1}^{N} \xi_i = \sum_{i=1}^{N} \max\{0, 1 - t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b)\}.$$

If we write $y^{(i)}(\mathbf{w}, b) = \mathbf{w}^\top \mathbf{x} + b$, then the optimization problem can be written as

$$\min_{\mathbf{w}, b, \xi} \sum_{i=1}^{N} \max\{0, 1 - t^{(i)} y^{(i)}(\mathbf{w}, b)\} + \frac{1}{2\gamma} \|\mathbf{w}\|_2^2$$

- The loss function $\mathcal{L}_{\mathrm{H}}(y, t) = \max\{0, 1 - ty\}$ is called the **hinge** loss.
- The second term is the L_2-norm of the weights.
- Hence, the soft-margin SVM can be seen as a linear classifier with hinge loss and an L_2 regularizer.

4.2.4 Multiclass SVM Loss

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	10.9

Multiclass SVM loss:

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$

= max(0, 2.2 - (-3.1) + 1)
+max(0, 2.5 - (-3.1) + 1)
= max(0, 5.3) + max(0, 5.6)
= 5.3 + 5.6
= 10.9

Multiclass SVM loss:

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:
$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$

and the full training loss is the mean over all examples in the training data:

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i$$

L = (2.9 + 0 + 10.9)/3
= **4.6**

4.3 Softmax

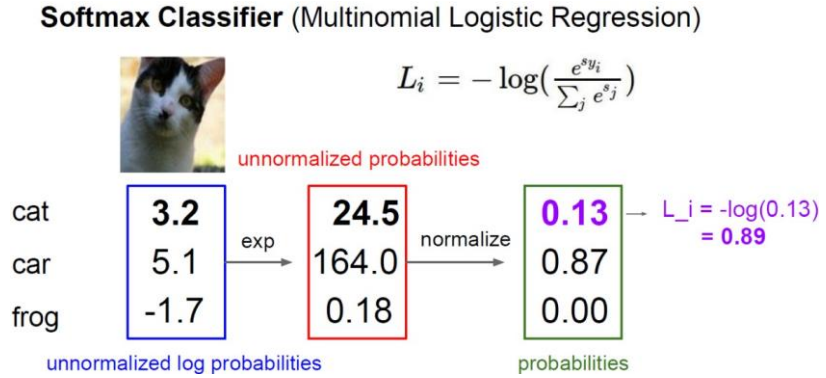
scores = unnormalized log probabilities of the classes.

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \text{where} \quad s = f(x_i; W)$$

Softmax function

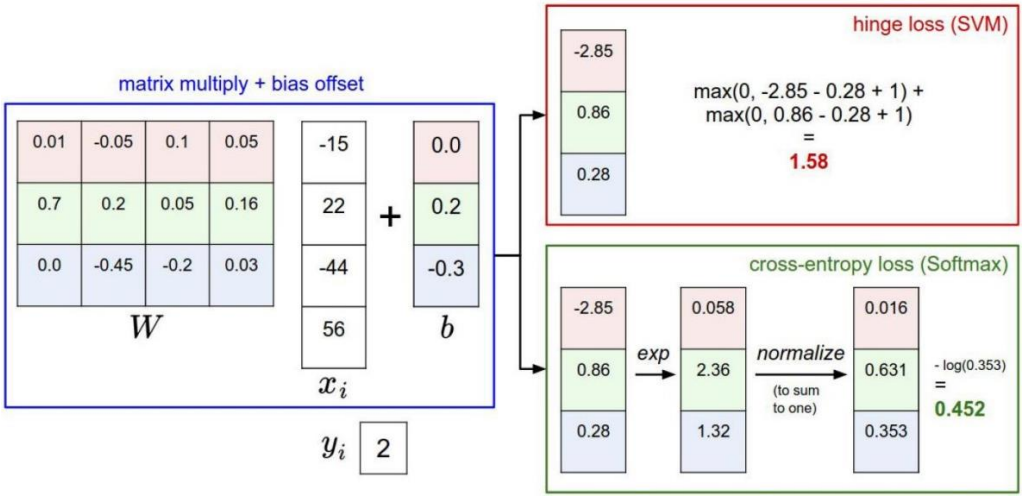
Want to maximize the log likelihood, or (for a loss function) to minimize the negative log likelihood of the correct class:

$$L_i = -\log P(Y = y_i | X = x_i)$$



4.4 SVM & Softmax

【深度学习 CV】SVM, Softmax 损失函数_svm 评估函数-CSDN 博客



Softmax vs. SVM

$L_i = -\log(\frac{e^{sy_i}}{\sum_j e^{s_j}})$ $L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$

assume scores:
[10, -2, 3]
[10, 9, 9]
[10, -100, -100]
and $y_i = 0$

Q: Suppose I take a datapoint and I jiggle a bit (changing its score slightly). What happens to the loss in both cases?

5 Neural Network and Back Propagation

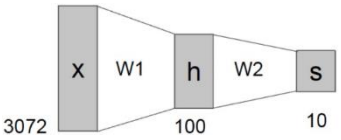
5.1 Neural Network

Neural Network: without the brain stuff

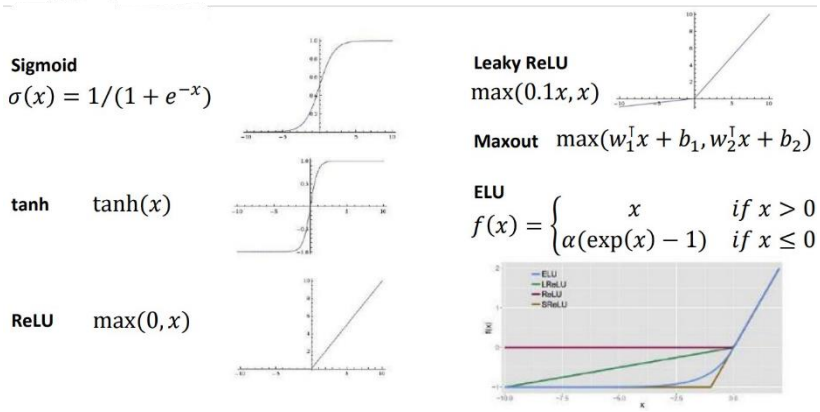
(Before) Linear score function: $f = Wx$

(Now) 2-layer Neural Network: $f = W_2\max(0, W_1x)$

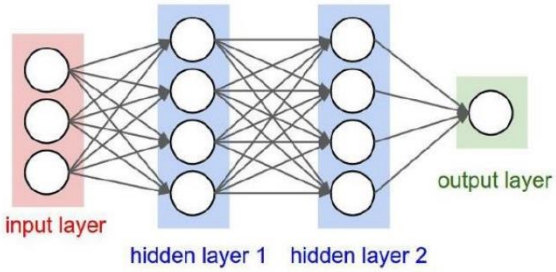
or 3-layer Neural Network: $f = W_3\max(0, W_2\max(0, W_1x))$



5.1.1 Activation Functions



Example Feed-forward computation of a Neural Network




```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

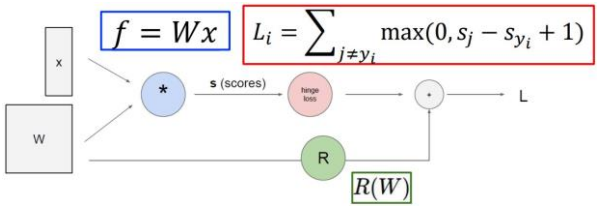
5.1.2 Gradient Descent

$s = f(x; W) = Wx$ scores function

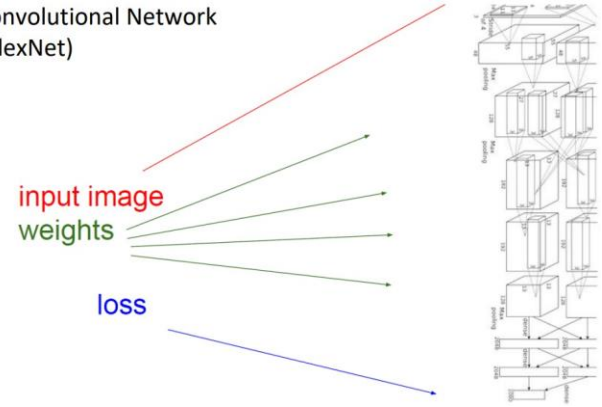
$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$ SVM loss

$L = \frac{1}{N}\sum_{i=1}^{N} L_i + \sum_k W_k^2$ data loss + regularization

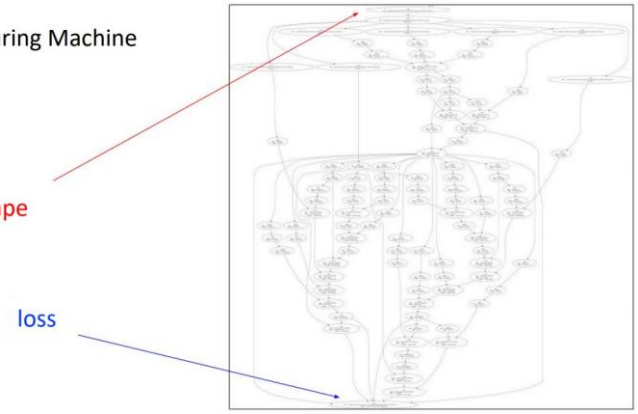
want $\nabla_W L$



Convolutional Network (AlexNet)



Neural Turing Machine



Example 1:

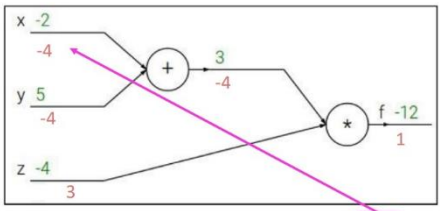
$f(x, y, z) = (x + y)z$

e.g. x = -2, y = 5, z = -4

$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$

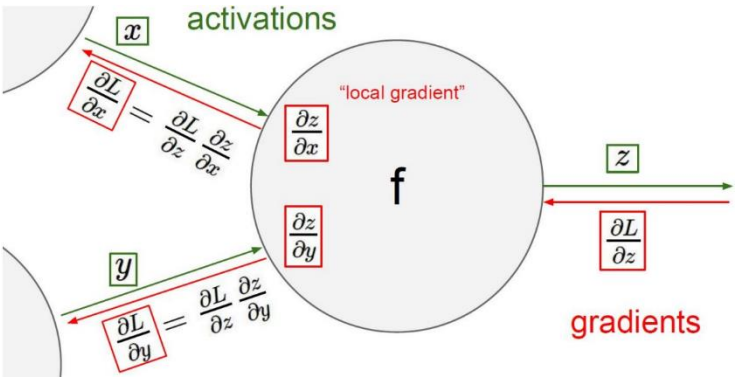
$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



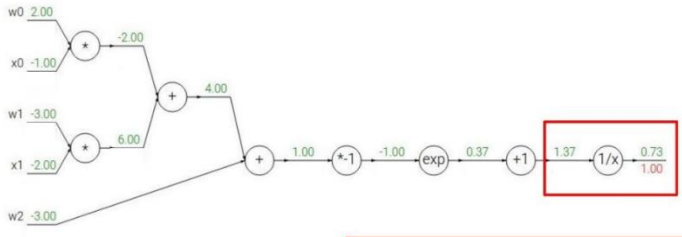
Chain rule:

$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y}$



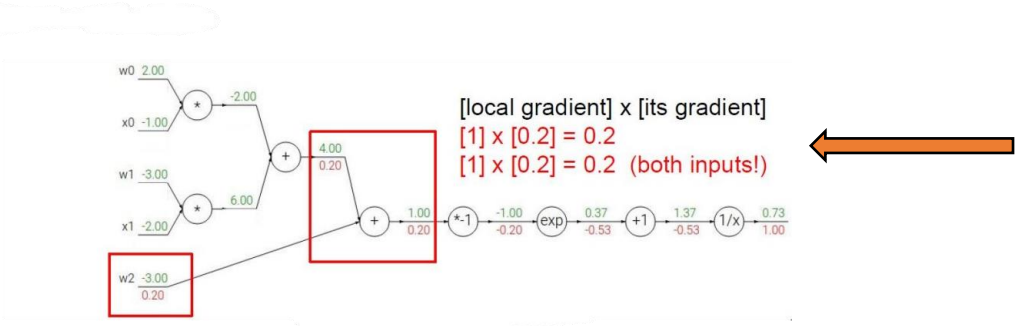
Example 2:

$f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$



$f(x) = e^x$	$\rightarrow$	$\frac{df}{dx} = e^x$	$f(x) = \frac{1}{x}$	$\rightarrow$	$\frac{df}{dx} = -1/x^2$
$f_a(x) = ax$	$\rightarrow$	$\frac{df}{dx} = a$	$f_c(x) = c + x$	$\rightarrow$	$\frac{df}{dx} = 1$

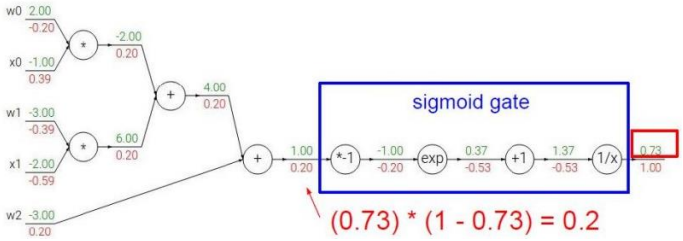
求导



Sigmoid feature:

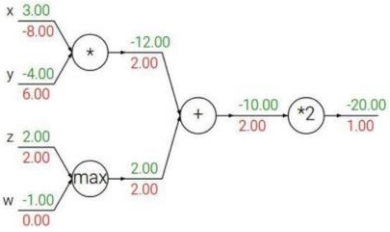
$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}} \qquad \boxed{\sigma(x) = \frac{1}{1+e^{-x}}} \quad \text{sigmoid function}$$

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\sigma(x)$$



Patterns in backward flow

add gate: gradient distributor
max gate: gradient router
mul gate: gradient... "switcher"?

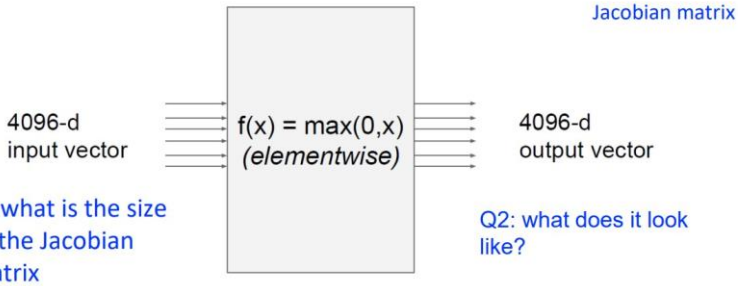


5.1.3 Gradients for vector

Vectorized operation

$$\frac{\partial L}{\partial x} = \boxed{\frac{\partial f}{\partial x}}\frac{\partial L}{\partial f}$$

Jacobian matrix



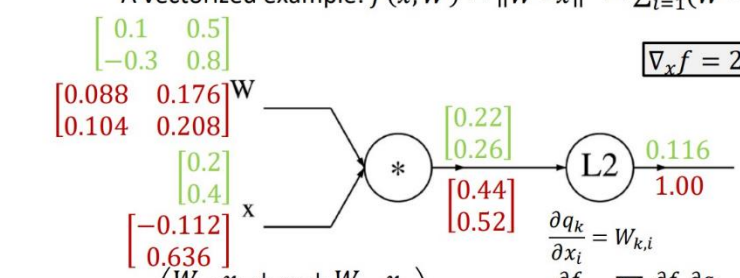
Q: what is the size of the Jacobian matrix [4096 x 4096!]

Q2: what does it look like?

Example:

A vectorized example: $f(x,W) = \|W \cdot x\|^2 = \sum_{i=1}^{n}(W \cdot x)_i^2$

$$\boxed{\nabla_x f = 2W^T \cdot q}$$



$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2$$

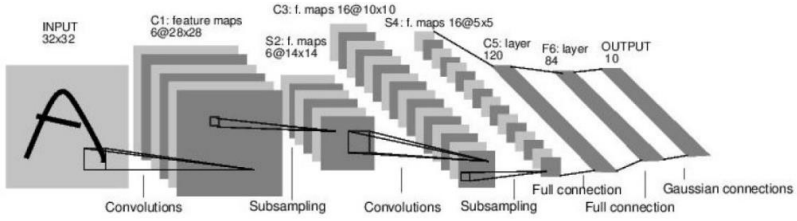
$$\frac{\partial q_k}{\partial x_i} = W_{k,i}$$

$$\frac{\partial f}{\partial x_i} = \sum_k \frac{\partial f}{\partial q_k}\frac{\partial q_k}{\partial x_i} = \sum_k 2q_k W_{k,i}$$

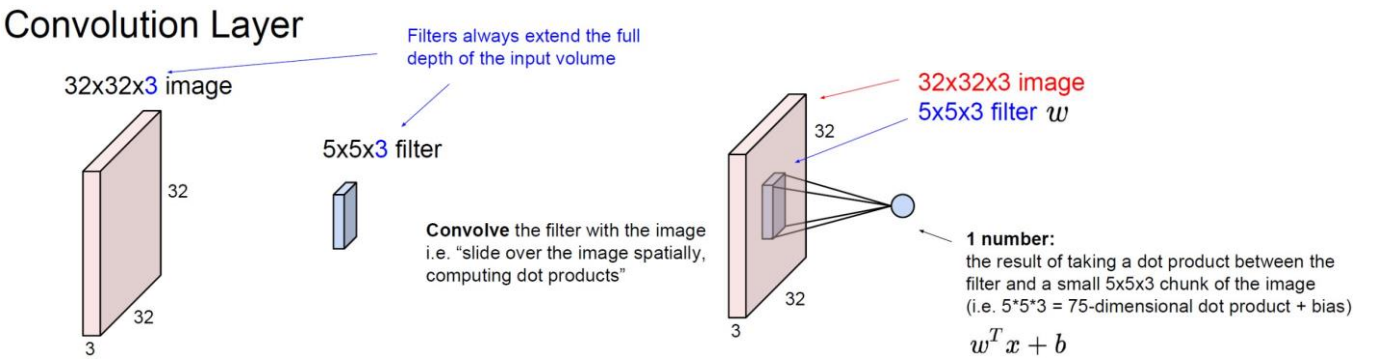
Always check: The gradient with respect to a variable should have the same shape as the variable

6 Convolutional Neural Network

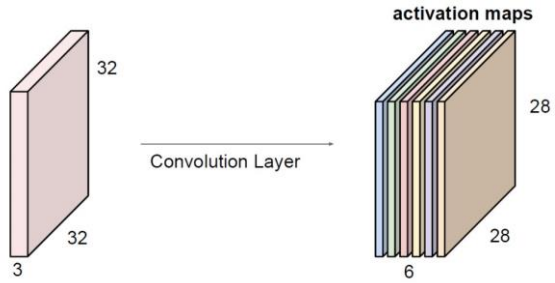
6.1 Basic concepts



[LeNet-5, LeCun 1980]

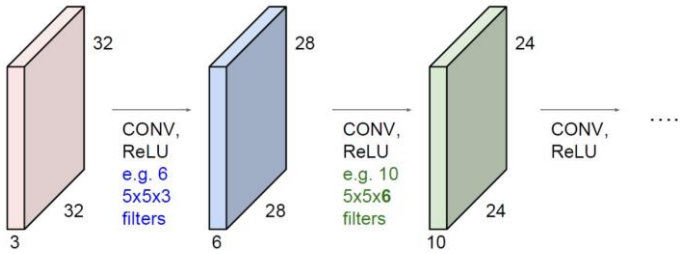


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

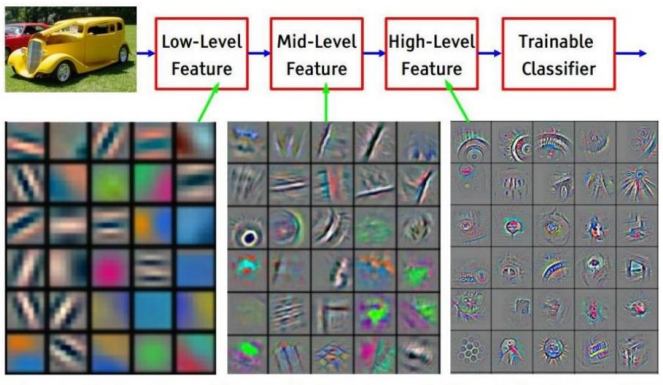


We stack these up to get a "new image" of size 28x28x6!

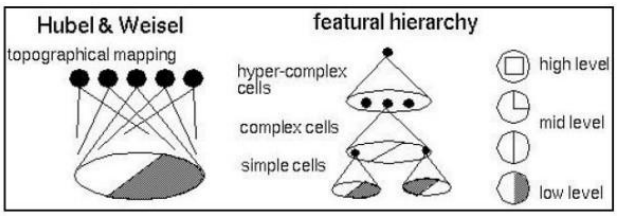
Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions

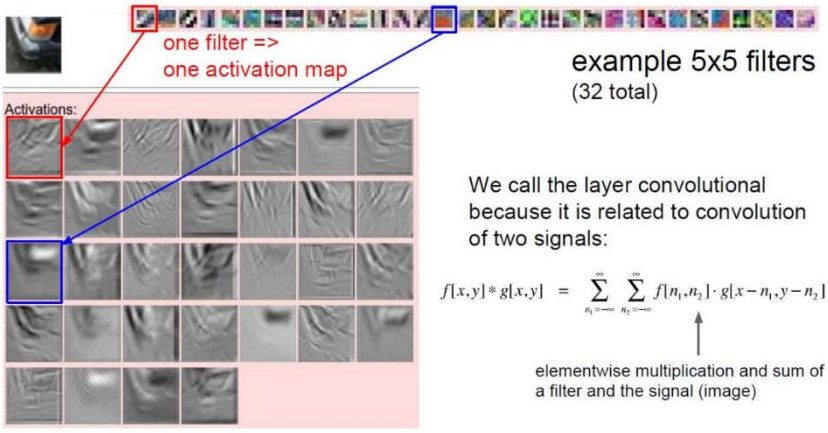


Preview:



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]





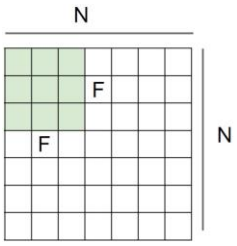
We call the layer convolutional because it is related to convolution of two signals:

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

elementwise multiplication and sum of a filter and the signal (image)

6.2 Convolutional layer

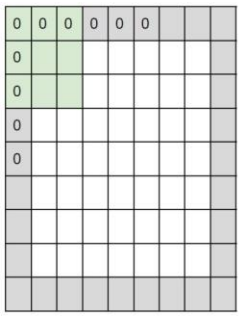
- Output size:



Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33

- Common to zero pad the border:



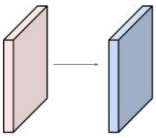
e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

- Calculation example:

Examples time:

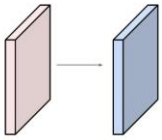
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2



Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10

Examples time:

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2



Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

- Conclusion:

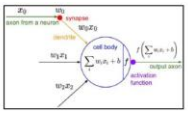
Common settings:

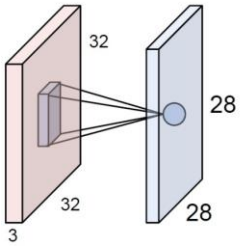
K = (powers of 2, e.g. 32, 64, 128, 512)
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

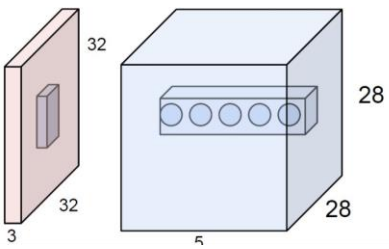
The brain/neuron view of CONV Layer





An activation map is a 28x28 sheet of neuron outputs:
1. Each is connected to a small region in the input
2. All of them share parameters

"5x5 filter" -> "5x5 receptive field for each neuron"

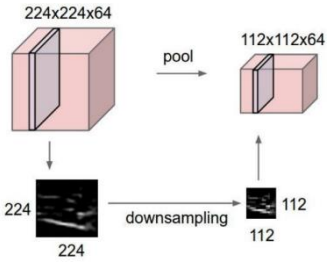


E.g. with 5 filters,
CONV layer consists of neurons arranged in a 3D grid (28x28x5)

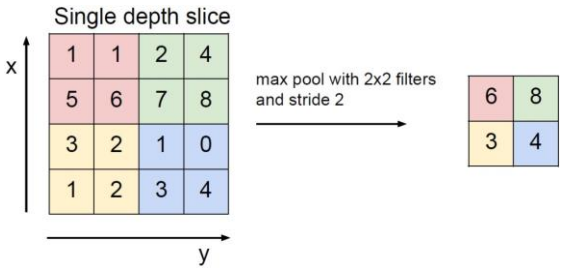
There will be 5 different neurons all looking at the same region in the input volume

6.3 Pooling layer

- Makes the representations smaller and more manageable
- Operates over each activation map independently



Max pooling:



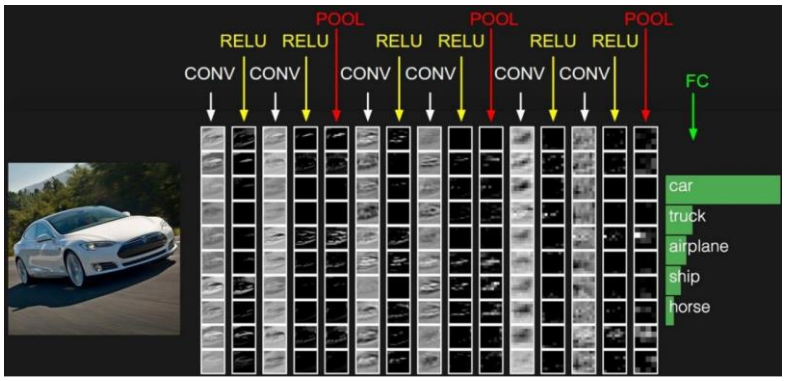
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

Common settings:

F = 2, S = 2
F = 3, S = 2

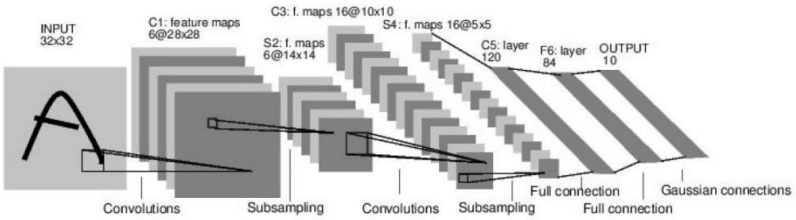
6.4 Fully connected layer

• Contains neurons that connect to the entire input volume, as in ordinary Neural Network

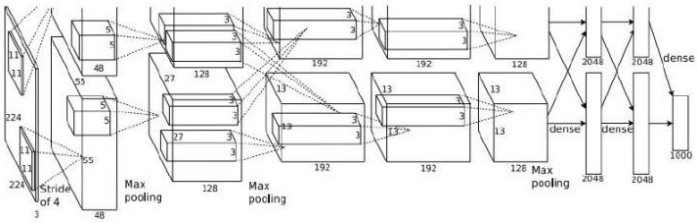


6.5 Case study: Models

6.5.1 LeNet-5



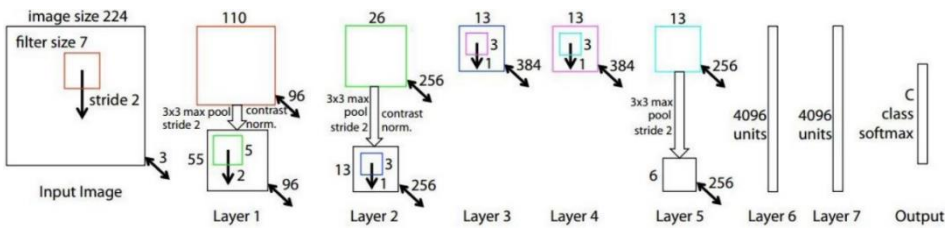
6.5.2 AlexNet



Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

6.5.3 ZFNet



AlexNet but:
CONV1: change from (11x11 stride 4) to (7x7 stride 2)
CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

ImageNet top 5 error: 15.4% -> 14.8%

6.5.4 VCGNet

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

best model

11.2% top 5 error in ILSVRC 2013
->
7.3% top 5 error

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = **102,760,448**
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

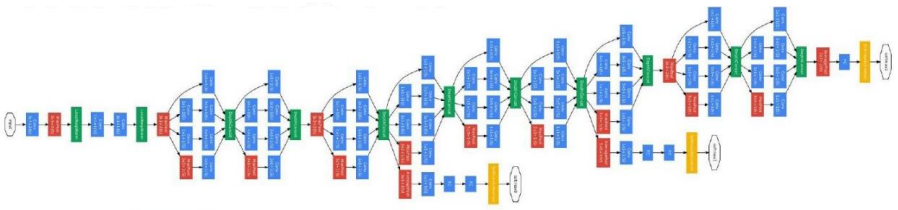
Note:

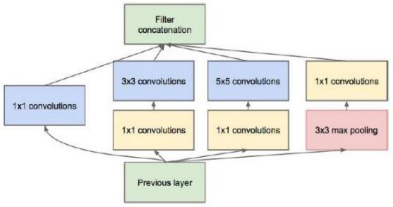
Most memory is in early CONV

Most params are in late FC

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

6.5.5 GoogleNet





Inception module

ILSVRC 2014 winner (6.7% top 5 error)

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

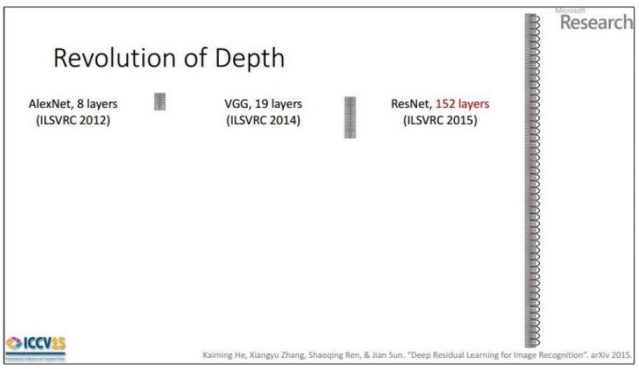
Fun features:

- Only 5 million params! (Removes FC layers completely)

Compared to AlexNet:
- 12X less params
- 2x more compute
- 6.67% (vs. 16.4%)

6.5.6 ResNet

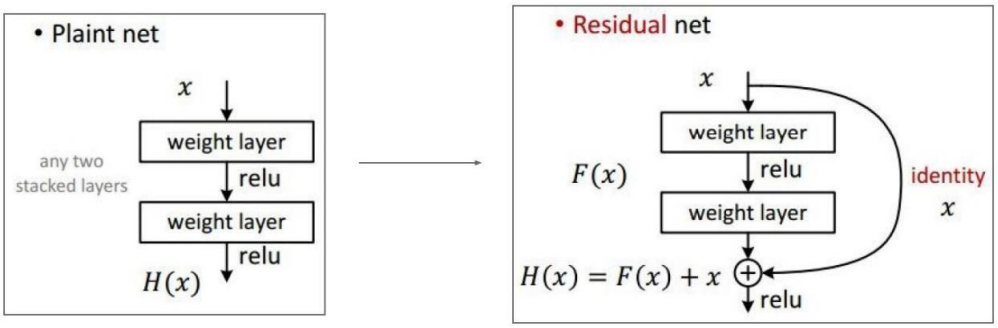
ILSVRC 2015 winner (3.6% top 5 error)



2-3 weeks of training on 8 GPU machine

at runtime: faster than a VGGNet! (even though it has 8x more layers)

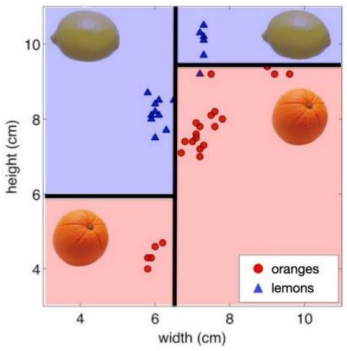
Residual block

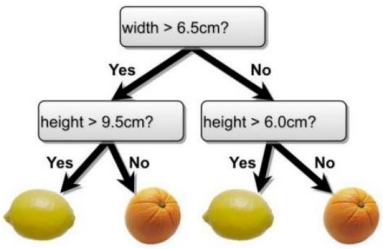


7 Decision Trees & Bias-Variance Decomposition

7.1 Decision Trees

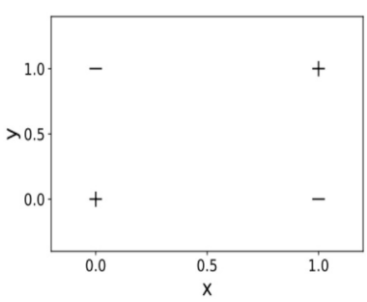
- Make predictions by splitting on features according to a tree structure.
- Split continuous features by checking whether that feature is greater than or less than some threshold.
- Decision boundary is made up of axis-aligned planes.

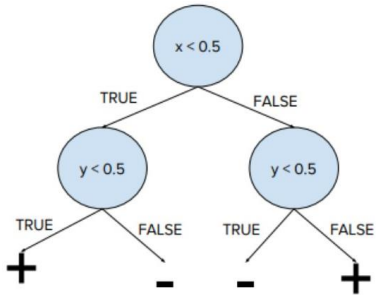




- Each path from root to a leaf defines a region R_m of input space
- Let $\{(x^{(m_1)}, t^{(m_1)}), \ldots, (x^{(m_k)}, t^{(m_k)})\}$ be the training examples that fall into R_m
- Classification tree (we will focus on this):
  - discrete output
  - leaf value y^m typically set to the most common value in $\{t^{(m_1)}, \ldots, t^{(m_k)}\}$
- Regression tree:
  - continuous output
  - leaf value y^m typically set to the mean value in $\{t^{(m_1)}, \ldots, t^{(m_k)}\}$

7.1.1 Discrete Features





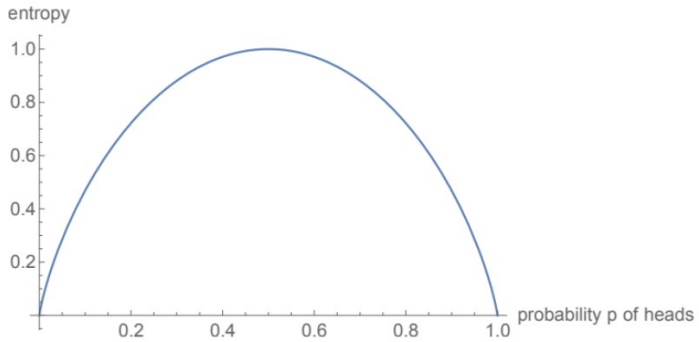
- For any training set we can construct a decision tree that has exactly the one leaf for every training point, but it probably won't generalize.
  - Decision trees are universal function approximators.
- But, finding the smallest decision tree that correctly classifies a training set is NP complete.
- Resort to a greedy heuristic:
  - Start with the whole training set and an empty decision tree.
  - Pick a feature and candidate split that would most reduce the loss.
  - Split on that feature and recurse on subpartitions.
- Which loss should we use?
  - Let's see if misclassification rate is a good loss.
- How can we quantify uncertainty in prediction for a given leaf node?
  - If all examples in leaf have same class: good, low uncertainty
  - If each class has same amount of examples in leaf: bad, high uncertainty
- **Idea:** Use counts at leaves to define probability distributions; use a probabilistic notion of uncertainty to decide splits.
- The entropy of a discrete random variable is a number that quantifies the uncertainty inherent in its possible outcomes.
- The entropy of a loaded coin with probability p of heads is given by

$$-p\log_2(p) - (1-p)\log_2(1-p)$$

$$-\frac{8}{9}\log_2\frac{8}{9} - \frac{1}{9}\log_2\frac{1}{9} \approx \frac{1}{2} \qquad -\frac{4}{9}\log_2\frac{4}{9} - \frac{5}{9}\log_2\frac{5}{9} \approx 0.99$$

- Notice: the coin whose outcomes are more certain has a lower entropy.
- In the extreme case $p = 0$ or $p = 1$, we were certain of the outcome before observing. So, we gained no certainty by observing it, i.e., entropy is 0.

- Can also think of entropy as the expected information content of a random draw from a probability distribution.



- Claude Shannon showed: you cannot store the outcome of a random draw using fewer expected bits than the entropy without losing information.
- So units of entropy are bits; a fair coin flip has 1 bit of entropy.
- More generally, the entropy of a discrete random variable Y is given by

$$H(Y) = -\sum_{y \in Y} p(y) \log_2 p(y)$$

- **"High Entropy"**:
  - Variable has a uniform like distribution over many outcomes
  - Flat histogram
  - Values sampled from it are less predictable
- **"Low Entropy"**
  - Distribution is concentrated on only a few outcomes
  - Histogram is concentrated in a few areas
  - Values sampled from it are more predictable

7.1.2 Entropy of a Joint Distribution

- Example: $X = \{\text{Raining, Not raining}\}$, $Y = \{\text{Cloudy, Not cloudy}\}$

	Cloudy	Not Cloudy
Raining	24/100	1/100
Not Raining	25/100	50/100

$$\begin{aligned} H(X,Y) &= -\sum_{x \in X} \sum_{y \in Y} p(x,y) \log_2 p(x,y) \\ &= -\frac{24}{100} \log_2 \frac{24}{100} - \frac{1}{100} \log_2 \frac{1}{100} - \frac{25}{100} \log_2 \frac{25}{100} - \frac{50}{100} \log_2 \frac{50}{100} \\ &\approx 1.56\text{bits} \end{aligned}$$

- What is the entropy of cloudiness Y, **given that it is raining**?

$$\begin{aligned} H(Y|X=x) &= -\sum_{y \in Y} p(y|x) \log_2 p(y|x) \\ &= -\frac{24}{25} \log_2 \frac{24}{25} - \frac{1}{25} \log_2 \frac{1}{25} \\ &\approx 0.24\text{bits} \end{aligned}$$

- We used: $p(y|x) = \frac{p(x,y)}{p(x)}$, and $p(x) = \sum_y p(x,y)$ (sum in a row)

- What is the entropy of cloudiness, given the knowledge of whether or not it is raining?

$$\begin{aligned} H(Y|X) &= \sum_{x \in X} p(x) H(Y|X=x) \\ &= \frac{1}{4} H(\text{cloudy}|\text{is raining}) + \frac{3}{4} H(\text{cloudy}|\text{not raining}) \\ &\approx 0.75 \text{ bits} \end{aligned}$$

- Some useful properties:
  - H is always non-negative
  - Chain rule: $H(X,Y) = H(X|Y) + H(Y) = H(Y|X) + H(X)$
  - If X and Y independent, then X does not affect our uncertainty about Y: $H(Y|X) = H(Y)$
  - But knowing Y makes our knowledge of Y certain: $H(Y|Y) = 0$
  - By knowing X, we can only decrease uncertainty about Y: $H(Y|X) \leq H(Y)$
- How much more certain am I about whether it's cloudy if I'm told whether it is raining? My uncertainty in Y minus my expected uncertainty that would remain in Y after seeing X.
- This is called the information gain $IG(Y|X)$ in Y due to X, or the mutual information of Y and X

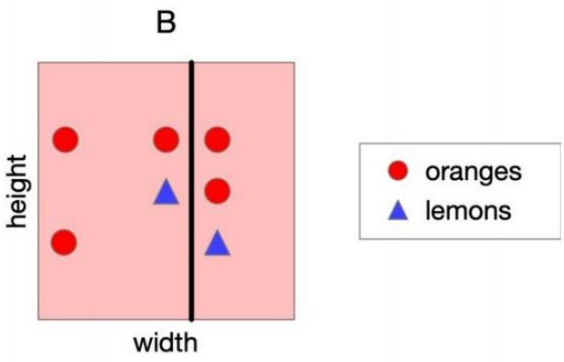
$$IG(Y|X) = H(Y) - H(Y|X)$$

- If X is completely uninformative about Y: $IG(Y|X) = 0$
- If X is completely informative about Y: $IG(Y|X) = H(Y)$
- The information gain of a split: how much information (over the training set) about the class label Y is gained by knowing which side of a split you're on.

information gain

7.1.3 Example

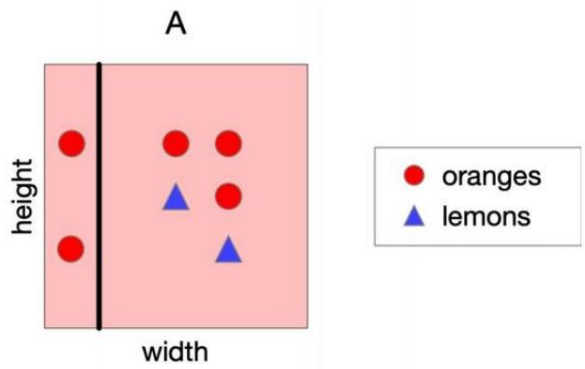
- What is the information gain of split B? Not terribly informative...



- Root entropy of class outcome: $H(Y) = -\frac{2}{7}\log_2(\frac{2}{7}) - \frac{5}{7}\log_2(\frac{5}{7}) \approx 0.86$
- Leaf conditional entropy of class outcome: $H(Y|left) \approx 0.81$, $H(Y|right) \approx 0.92$
- $IG(split) \approx 0.86 - (\frac{4}{7} \cdot 0.81 + \frac{3}{7} \cdot 0.92) \approx 0.006$

```
HY <- -2/7*log2(2/7)-5/7*log2(5/7)
HYL <- -3/4*log2(3/4)-1/4*log2(1/4)
HYR <- -2/3*log2(2/3)-1/3*log2(1/3)
IGS <- HY-(4/7*HYL+3/7*HYR)
```

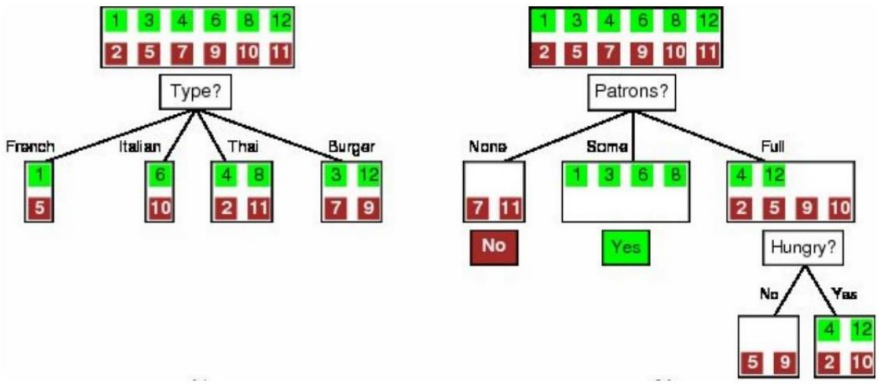
- What is the information gain of split A? Very informative!



- Root entropy of class outcome: $H(Y) = -\frac{2}{7}\log_2(\frac{2}{7}) - \frac{5}{7}\log_2(\frac{5}{7}) \approx 0.86$
- Leaf conditional entropy of class outcome: $H(Y|left) = 0$, $H(Y|right) \approx 0.97$
- $IG(split) \approx 0.86 - (\frac{2}{7} \cdot 0 + \frac{5}{7} \cdot 0.97) \approx 0.17$!!

7.1.4 Decision Trees Construction Algorithm

- Simple, greedy, recursive approach, builds up tree node-by-node
  1. pick a feature to split at a non-terminal node
  2. split examples into groups based on feature value
  3. for each group:
     - if no examples – return majority from parent
     - else if all examples in same class – return class
     - else loop to step 1
- Terminates when all leaves contain only examples in the same class or are empty.



$$IG(Y) = H(Y) - H(Y|X)$$

$$IG(type) = 1 - \left[\frac{2}{12}H(Y|\text{Fr.}) + \frac{2}{12}H(Y|\text{It.}) + \frac{4}{12}H(Y|\text{Thai}) + \frac{4}{12}H(Y|\text{Bur.})\right] = 0$$

$$IG(Patrons) = 1 - \left[\frac{2}{12}H(0,1) + \frac{4}{12}H(1,0) + \frac{6}{12}H(\frac{2}{6},\frac{4}{6})\right] \approx 0.541$$

- Problems:
  - You have exponentially less data at lower levels
  - Too big of a tree can overfit the data
  - Greedy algorithms don't necessarily yield the global optimum
- Handling continuous attributes
  - Split based on a threshold, chosen to maximize information gain

7.1.5 Decision Trees comparison to some other classifiers

Advantages of decision trees over KNNs and neural nets

- Simple to deal with discrete features, missing values, and poorly scaled data
- Fast at test time
- More interpretable

Advantages of KNNs over decision trees

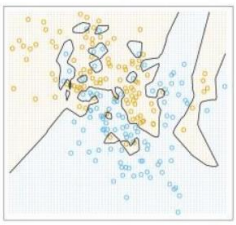
- Few hyperparameters
- Can incorporate interesting distance measures (e.g. shape contexts)

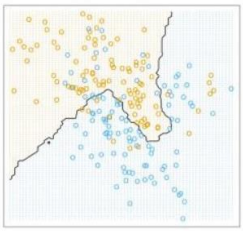
Advantages of neural nets over decision trees

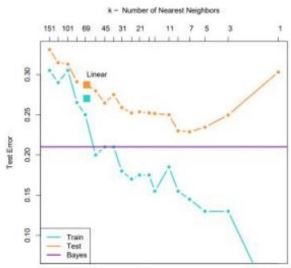
- Able to handle attributes/features that interact in very complex ways (e.g. pixels)
- We can combine multiple classifiers into an ensemble, which is a set of predictors whose individual decisions are combined in some way to classify new examples
  - E.g., (possibly weighted) majority vote
- For this to be nontrivial, the classifiers must differ somehow, e.g.
  - Different algorithm
  - Different choice of hyperparameters
  - Trained on different data
  - Trained with different weighting of the training examples
- Today, we deepen our understanding of generalization through a bias-variance decomposition.
  - This will help us understand ensembling methods.

7.2 Bias-Variance Decomposition

- Recall that overly simple models underfit the data, and overly complex models overfit.







- We can quantify this effect in terms of the bias/variance decomposition.

7.2.1 Basic Setup

- Recap of basic setup:
  - Fix a query point $\mathbf{x}$.
  - Repeat:
    - Sample a random training dataset $\mathcal{D}$ i.i.d. from the data generating distribution p_{sample}.
    - Run the learning algorithm on $\mathcal{D}$ to get a prediction y at $\mathbf{x}$.
    - Sample the (true) target from the conditional distribution $p(t|\mathbf{x})$.
    - Compute the loss $L(y, t)$.
- Notice: y is independent of t.
- This gives a distribution over the loss at $\mathbf{x}$, with expectation $\mathbb{E}[L(y, t) \mid \mathbf{x}]$.
- For each query point $\mathbf{x}$, the expected loss is different. We are interested in minimizing the expectation of this with respect to $\mathbf{x} \sim p_{\text{sample}}$.

7.2.2 Bayes Optimality

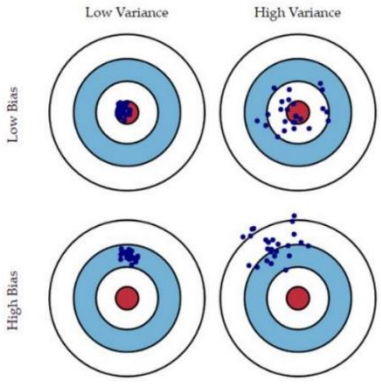
- For now, focus on squared error loss, $L(y,t) = \frac{1}{2}(y-t)^2$.
- A first step: suppose we knew the conditional distribution $p(t \mid \mathbf{x})$. What value y should we predict?
  - Here, we are treating t as a random variable and choosing y.
- **Claim:** $y_* = \mathbb{E}[t \mid \mathbf{x}]$ is the best possible prediction.
- **Proof:**

$$\begin{aligned}\mathbb{E}[(y-t)^2 \mid \mathbf{x}] &= \mathbb{E}[y^2 - 2yt + t^2 \mid \mathbf{x}] \\ &= y^2 - 2y\mathbb{E}[t \mid \mathbf{x}] + \mathbb{E}[t^2 \mid \mathbf{x}] \\ &= y^2 - 2y\mathbb{E}[t \mid \mathbf{x}] + \mathbb{E}[t \mid \mathbf{x}]^2 + \mathrm{Var}[t \mid \mathbf{x}] \\ &= y^2 - 2yy_* + y_*^2 + \mathrm{Var}[t \mid \mathbf{x}] \\ &= (y-y_*)^2 + \mathrm{Var}[t \mid \mathbf{x}]\end{aligned}$$

- The first term is nonnegative, and can be made 0 by setting $y = y_*$.
- The second term corresponds to the inherent unpredictability, or noise, of the targets, and is called the Bayes error.
  - This is the best we can ever hope to do with any learning algorithm. An algorithm that achieves it is Bayes optimal.
  - Notice that this term doesn't depend on y.
- This process of choosing a single value y_* based on $p(t \mid \mathbf{x})$ is an example of decision theory.
- Now return to treating y as a random variable (where the randomness comes from the choice of dataset).
- We can decompose out the expected loss (suppressing the conditioning on $\mathbf{x}$ for clarity):

$$\begin{aligned}\mathbb{E}[(y-t)^2] &= \mathbb{E}[(y-y_\star)^2] + \mathrm{Var}(t) \\ &= \mathbb{E}[y_\star^2 - 2y_\star y + y^2] + \mathrm{Var}(t) \\ &= y_\star^2 - 2y_\star\mathbb{E}[y] + \mathbb{E}[y^2] + \mathrm{Var}(t) \\ &= y_\star^2 - 2y_\star\mathbb{E}[y] + \mathbb{E}[y]^2 + \mathrm{Var}(y) + \mathrm{Var}(t) \\ &= \underbrace{(y_\star - \mathbb{E}[y])^2}_{\text{bias}} + \underbrace{\mathrm{Var}(y)}_{\text{variance}} + \underbrace{\mathrm{Var}(t)}_{\text{Bayes error}}\end{aligned}$$

- We just split the expected loss into three terms:
  - bias: how wrong the expected prediction is (corresponds to underfitting)
  - variance: the amount of variability in the predictions (corresponds to overfitting)
  - Bayes error: the inherent unpredictability of the targets
- Even though this analysis only applies to squared error, we often loosely use "bias" and "variance" as synonyms for "underfitting" and "overfitting".
- Throwing darts = predictions for each draw of a dataset



8 Bagging & Boosting

8.1 Bias/Variance Decomposition

We treat predictions y at a query **x** as a random variable (where the randomness comes from the choice of dataset), y_* is the optimal deterministic prediction, t is a random target sampled from the true conditional p(t|**x**).

$$\mathbb{E}[(y-t)^2] = \underbrace{(y_\star - \mathbb{E}[y])^2}_{\text{bias}} + \underbrace{\text{Var}(y)}_{\text{variance}} + \underbrace{\text{Var}(t)}_{\text{Bayes error}}$$

➢bias: how wrong the expected prediction is (corresponds to underfitting)

➢variance: the amount of variability in the predictions (corresponds to overfitting)

➢Bayes error: the inherent unpredictability of the targets

8.2 Bagging

8.2.1 Motivation

Suppose we could somehow sample m independent training sets from p_{sample}. We could then compute the prediction y_i based on each one, and take the average $y = \frac{1}{m}\sum_{i=1}^{m} y_i$.

➢Bayes error: unchanged, since we have no control over it

➢Bias: unchanged, since the averaged prediction has the same expectation

$$\mathbb{E}[y] = \mathbb{E}\left[\frac{1}{m}\sum_{i=1}^{m} y_i\right] = \mathbb{E}[y_i]$$

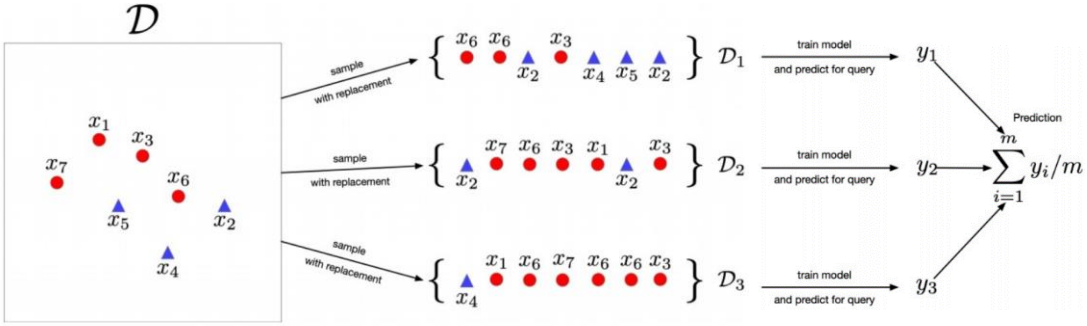
➢Variance: reduced, since we're averaging over independent samples

$$\text{Var}[y] = \text{Var}\left[\frac{1}{m}\sum_{i=1}^{m} y_i\right] = \frac{1}{m^2}\sum_{i=1}^{m}\text{Var}[y_i] = \frac{1}{m}\text{Var}[y_i]$$

8.2.2 Idea

Solution: given training set D, use the empirical distribution p_D as a proxy for p_{sample}. This is called bootstrap aggregation, or bagging.

- Take a single dataset D with n examples.
- Generate D new datasets ("resamples" or "bootstrap samples"), each by sampling n training examples from D, with replacement.
- Average the predictions of models trained on each of these datasets.
- Intuition: As $|D| \to \infty$, we have $p_D \to p_{sample}$.

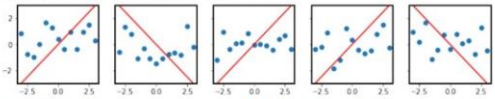


In this example $n = 7, m = 3$

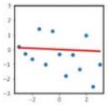
8.2.3 Effect on Hypothesis Space

Bagging can change the hypothesis space / inductive bias:

- $x \sim \mathcal{U}(-3,3), t \sim \mathcal{N}(0,1)$
- $\mathcal{H} = \{wx | w \in \{-1,1\}\}$
- Sampled datasets & fitted hypotheses:



- Ensembled hypotheses (mean over 1000 samples):



➢ The ensembled hypothesis is not in the original hypothesis space!

8.2.4 Bagging for Binary Classification

- If our classifiers output real-valued probabilities $z_i \in [0, 1]$, then we can average the predictions before thresholding:

$$y_{\text{bagged}} = \mathbb{I}(z_{\text{bagged}} > 0.5) = \mathbb{I}\left(\sum_{i=1}^{m} \frac{z_i}{m} > 0.5\right)$$

- If our classifiers output binary decisions $y_i \in \{0, 1\}$, then we can still average the predictions before thresholding:

$$y_{\text{bagged}} = \mathbb{I}\left(\sum_{i=1}^{m} \frac{y_i}{m} > 0.5\right)$$

- A bagged classifier can be stronger than the average underlying model

8.2.5 Some issues

- Problem: the datasets are not independent, so we don't get the $1/m$ variance reduction.
  - ➢Possible to show that if the sampled predictions have variance σ^2, and correlation ρ, then:

$$\text{Var}\left(\frac{1}{m}\sum_{i=1}^{m} y_i\right) = \frac{1}{m}(1-\rho)\sigma^2 + \rho\sigma^2$$

- Solution: Random forests = bagged decision trees, with one extra trick to decorrelate the predictions
  - ➢When choosing each node of the decision tree, choose a random set of d input features, and only consider splits on those features

8.3 Boosting

To focus on specific examples, boosting uses a weighted training set.

- Train classifiers sequentially, each time focusing on training examples that the previous ones got wrong.
- The shifting focus strongly decorrelates their predictions

8.3.1 Weighted Training set

- Key idea: we can learn a classifier using different costs (aka weights) for examples.
- Change cost function:

$$\sum_{n=1}^{N} \frac{1}{N} \mathbb{I}\left[h(x^{(n)}) \neq t^{(n)}\right] \quad \text{becomes} \quad \sum_{n=1}^{N} w^{(n)} \mathbb{I}\left[h(x^{(n)}) \neq t^{(n)}\right]$$

- Usually require each $w^{(n)}$ and $\sum_{n=1}^{N} w^{(n)} = 1$.

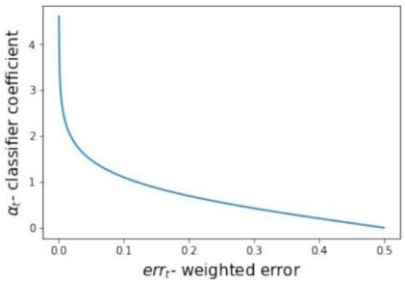
8.3.2 AdaBoost (Adaptive Boosting)

- Given a base classifier, the key steps of AdaBoost are:
  1. At each iteration, re-weight the training samples by assigning larger weights to samples (i.e., data points) that were classified incorrectly.
  2. Train a new base classifier based on the re-weighted samples.
  3. Add it to the ensemble of classifiers with an appropriate weight.
  4. Repeat the process many times.
- Requirements for base classifier:
  - ➢Needs to minimize weighted error.
  - ➢Ensemble may get very large, so base classifier must be fast. It turns out that any so-called weak learner / classifier suffices.
- Individually, weak learners may have high bias (underfit). By making each classifier focus on previous mistakes, AdaBoost reduces bias
- Weak learner is a learning algorithm that outputs a hypothesis (e.g., a classifier) that performs slightly better than chance, e.g., it predicts the correct label with probability 0.51 in binary label case.
- Decision Stump: A decision tree with a single split

8.3.2 AdaBoost Algorithm

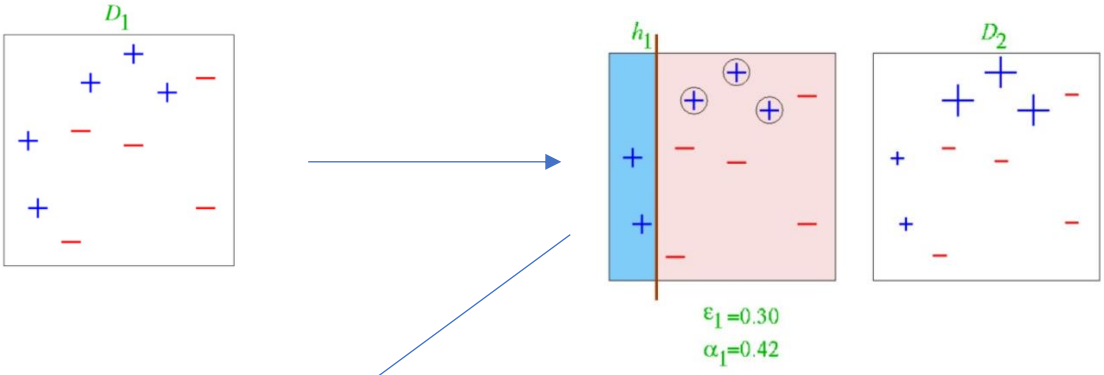
- Input: Data D_N, weak classifier WeakLearn (a classification procedure that returns a classifier *h*, e.g. best decision stump, from a set of classifiers $\mathcal{H}$, e.g., all possible decision stumps), number of iterations T
- Output: Classifier $H_{(x)}$

- Initialize sample weights: $w^{\{n\}} = \frac{1}{N}$ for $n = 1, \cdots, N$
- For $t = 1, \cdots, T$
  - Fit a classifier to weighted data ($h_t \leftarrow \text{WeakLearn}(\mathcal{D}_N, \mathbf{w})$), e.g.,
    $$h_t \leftarrow \underset{h \in \mathcal{H}}{\operatorname{argmin}} \sum_{n=1}^{N} w^{(n)} \mathbb{I}\{h(\mathbf{x}^{(n)}) \neq t^{(n)}\}$$
  - Compute weighted error $\text{err}_t = \frac{\sum_{n=1}^{N} w^{(n)} \mathbb{I}\{h_t(x^{(n)}) \neq t^{(n)}\}}{\sum_{n=1}^{N} w^{(n)}}$
  - Compute classifier coefficient $\alpha_t = \frac{1}{2} \log \frac{1-\text{err}_t}{\text{err}_t} (\in (0, \infty))$
  - Update data weights
    $$w^{(n)} \leftarrow w^{(n)} \exp\left(-\alpha_t t^{(n)} h_t(\mathbf{x}^{(n)})\right) \left[\equiv w^{(n)} \exp\left(2\alpha_t \mathbb{I}\{h_t(\mathbf{x}^{(n)}) \neq t^{(n)}\}\right)\right]$$



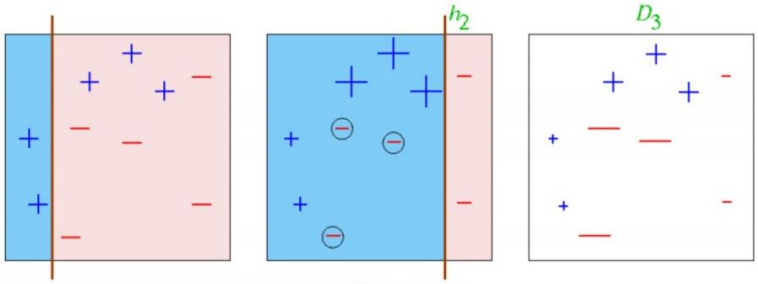
- Return $H_{(x)} = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$
- Weak classifiers which get lower weighted error get more weight in the final classifier

  Also: $w^{(n)} \leftarrow w^{(n)} \exp\left(2\alpha_t \mathbb{I}\{h_t(\mathbf{x}^{(n)}) \neq t^{(n)}\}\right)$
  - If $\text{err}_t \approx 0, \alpha_t$ high so misclassified examples get more attention
  - If $\text{err}_t \approx 0.5, \alpha_t$ low so misclassified examples are not emphasized



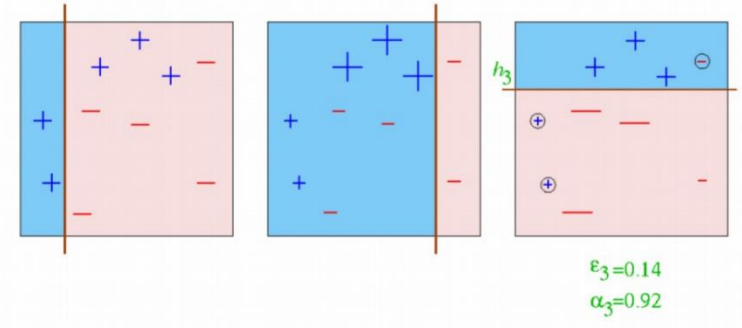
$$\mathbf{w} = \left(\frac{1}{10}, \ldots, \frac{1}{10}\right) \Rightarrow \text{Train a classifier (using } \mathbf{w}) \Rightarrow \text{err}_1 = \frac{\sum_{i=1}^{10} w_i \mathbb{I}\{h_1(\mathbf{x}^{(i)}) \neq t^{(i)}\}}{\sum_{i=1}^{N} w_i} = \frac{3}{10}$$

$$\Rightarrow \alpha_1 = \frac{1}{2} \log \frac{1 - \text{err}_1}{\text{err}_1} = \frac{1}{2} \log\left(\frac{1}{0.3} - 1\right) \approx 0.42 \Rightarrow H(\mathbf{x}) = \text{sign}\left(\alpha_1 h_1(\mathbf{x})\right)$$



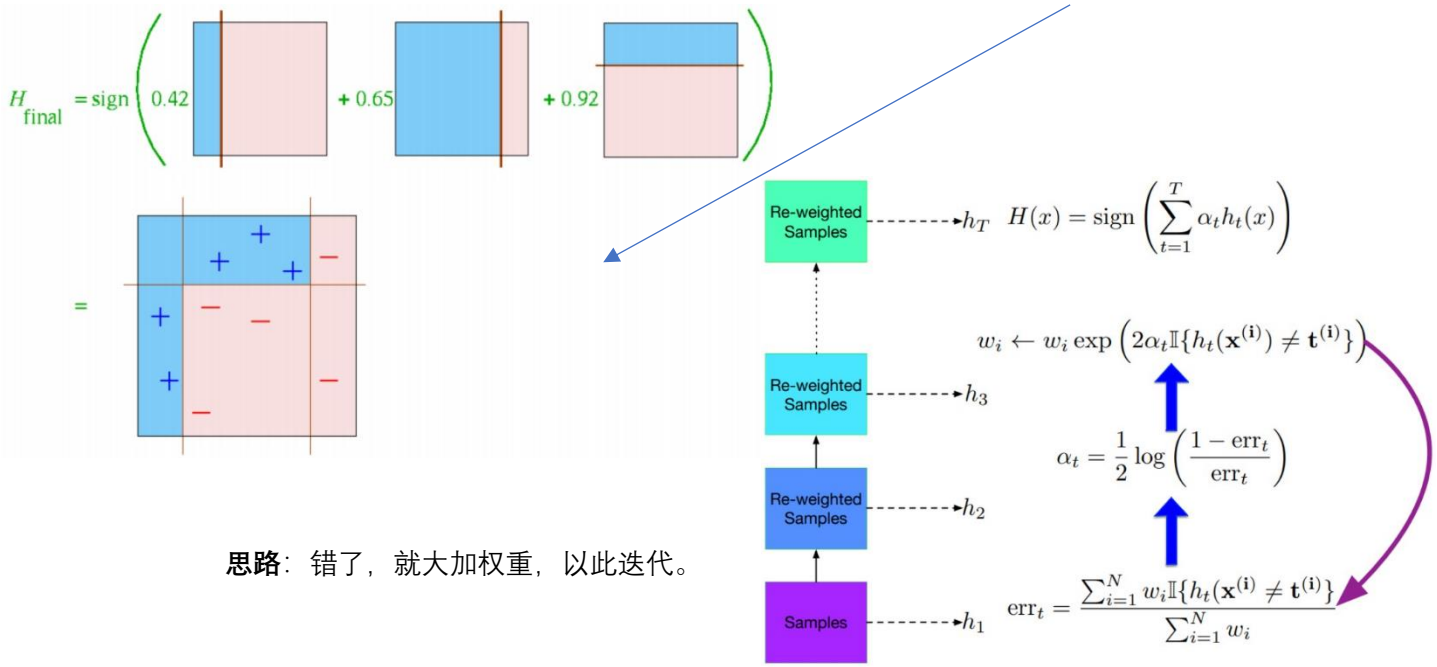
$$\mathbf{w} = \text{updated weights} \Rightarrow \text{Train a classifier (using } \mathbf{w}) \Rightarrow \text{err}_2 = \frac{\sum_{i=1}^{10} w_i \mathbb{I}\{h_2(\mathbf{x}^{(i)}) \neq t^{(i)}\}}{\sum_{i=1}^{N} w_i} = 0.21$$

$$\Rightarrow \alpha_2 = \frac{1}{2} \log \frac{1 - \text{err}_3}{\text{err}_3} = \frac{1}{2} \log\left(\frac{1}{0.21} - 1\right) \approx 0.66 \Rightarrow H(\mathbf{x}) = \text{sign}\left(\alpha_1 h_1(\mathbf{x}) + \alpha_2 h_2(\mathbf{x})\right)$$



$$\mathbf{w} = \text{updated weights} \Rightarrow \text{Train a classifier (using } \mathbf{w}) \Rightarrow \text{err}_3 = \frac{\sum_{i=1}^{10} w_i \mathbb{I}\{h_3(\mathbf{x}^{(i)}) \neq t^{(i)}\}}{\sum_{i=1}^{N} w_i} = 0.14$$

$$\Rightarrow \alpha_3 = \frac{1}{2} \log \frac{1 - \text{err}_3}{\text{err}_3} = \frac{1}{2} \log\left(\frac{1}{0.14} - 1\right) \approx 0.91 \Rightarrow H(\mathbf{x}) = \text{sign}\left(\alpha_1 h_1(\mathbf{x}) + \alpha_2 h_2(\mathbf{x}) + \alpha_3 h_3(\mathbf{x})\right)$$



思路：错了，就大加权重，以此迭代。

8.3.3 AdaBoost Minimizes the Training Error

Theorem

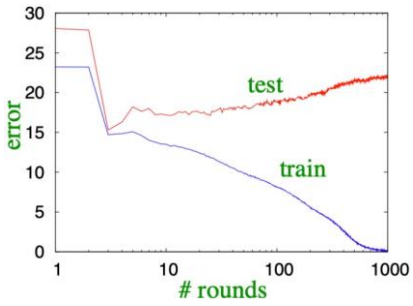
Assume that at each iteration of AdaBoost the WeakLearn returns a hypothesis with error $\text{err}_t \leq \frac{1}{2} - \gamma$ for all $t = 1, \ldots, T$ with $\gamma > 0$. The training error of the output hypothesis $H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right)$ is at most

$$L_N(H) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}\{H(\mathbf{x}^{(i)}) \neq t^{(i)})\} \leq \exp\left(-2\gamma^2 T\right).$$

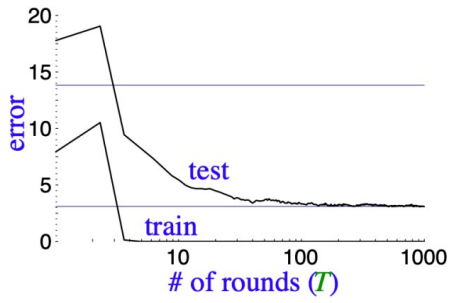
- This is under the simplifying assumption that each weak learner is γ-better than a random predictor.
- This is called geometric convergence. It is fast!

8.3.4 Generalization Error of AdaBoost

- As we add more weak classifiers, the overall classifier H becomes more "complex".
- We expect more complex classifiers overfit, if one runs AdaBoost long enough, it can in fact overfit.
- But often it does not! Sometimes the test error decreases even after the training error is zero!



<Expect>



<True situation>

8.3.5 Additive Models

An additive model with m terms is given by:

$$H_m(x) = \sum_{i=1}^{m} \alpha_i h_i(\mathbf{x})$$

Where $(\alpha_1, \cdots, \alpha_m) \in \mathbb{R}^m$.

A greedy approach to fitting additive models, known as stagewise training:

1. Initialize $H_0(x) = 0$
2. For $m = 1$ to T:
   - Compute the m-th hypothesis $H_m = H_{m-1} + \alpha_m h_m$, i.e. h_m and α_m, assuming previous additive model H_{m-1} is fixed:

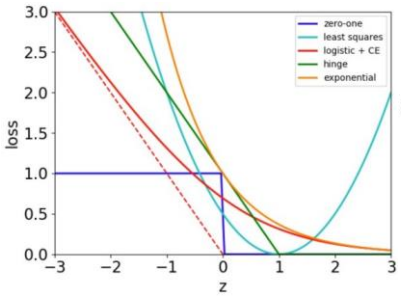
$$(h_m, \alpha_m) \leftarrow \underset{h \in \mathcal{H}, \alpha}{\text{argmin}} \sum_{i=1}^{N} \mathcal{L}\left(H_{m-1}(\mathbf{x}^{(i)}) + \alpha h(\mathbf{x}^{(i)}),\ t^{(i)}\right)$$

   - Add it to the additive model

$$H_m = H_{m-1} + \alpha_m h_m$$

We want to see how the stagewise training of additive models can be done. So, we should consider the exponential loss:

$$\mathcal{L}_{\mathrm{E}}(z,t) = \exp(-tz)$$



$$\begin{aligned}(h_m, \alpha_m) &\leftarrow \operatorname*{argmin}_{h\in\mathcal{H},\alpha} \sum_{i=1}^{N} \exp\left(-\left[H_{m-1}(\mathbf{x}^{(i)}) + \alpha h(\mathbf{x}^{(i)})\right] t^{(i)}\right) \\ &= \sum_{i=1}^{N} \exp\left(-H_{m-1}(\mathbf{x}^{(i)}) t^{(i)}\right) \exp\left(-\alpha h(\mathbf{x}^{(i)}) t^{(i)}\right) \\ &= \sum_{i=1}^{N} w_i^{(m)} \exp\left(-\alpha h(\mathbf{x}^{(i)}) t^{(i)}\right).\end{aligned}$$

Here we defined $w_i^{(m)} \triangleq \exp\left(-H_{m-1}(\mathbf{x}^{(i)}) t^{(i)}\right)$ (doesn't depend on h, α)

Obtain the additive model $H_m(x) = \sum_{i=1}^{m} \alpha_i h_i(\mathbf{x})$ with:

$$\begin{aligned}h_m &\leftarrow \operatorname*{argmin}_{h\in\mathcal{H}} \sum_{i=1}^{N} w_i^{(m)} \mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t^{(i)}\}, \\ \alpha &= \frac{1}{2}\log\left(\frac{1-\mathrm{err}_m}{\mathrm{err}_m}\right), \qquad \text{where } \mathrm{err}_m = \frac{\sum_{i=1}^{N} w_i^{(m)} \mathbb{I}\{h_m(\mathbf{x}^{(i)}) \neq t^{(i)}\}}{\sum_{i=1}^{N} w_i^{(m)}}, \\ w_i^{(m+1)} &= w_i^{(m)} \exp\left(-\alpha_m h_m(\mathbf{x}^{(i)}) t^{(i)}\right).\end{aligned}$$

8.3.6 Boosting Summary

- Boosting reduces bias by generating an ensemble of weak classifiers.
- Each classifier is trained to reduce errors of previous ensemble.
- It is quite resilient to overfitting, though it can overfit.

9 Probabilistic Models

9.1 Model

- Start with a simple biased coin example:
  - You flip a coin $N = 100$ times and get outcomes $\{x_1, \ldots, x_N\}$ where $x_i \in \{0, 1\}$ and $x_i = 1$ is interpreted as heads H.
  - Suppose you had $N_H = 55$ heads and $N_T = 45$ tails.
  - What is the probability it will come up heads if we flip again? Let's design a model for this scenario, fit the model. We can use the fit model to predict the next outcome.
- The coin is possibly loaded. So, we can assume that one coin flip outcome x is a Bernoulli random variable for some currently unknown parameter $\theta \in [0, 1]$.
- It's sensible to assume that$\{x_1, \ldots, x_N"\}$are independent and identically distributed Bernoullis.
- Thus the joint probability of the outcome $\{x_1, \ldots, x_N"\}$ is:

$$p(x_1, \ldots, x_N|\theta) = \prod_{i=1}^{N} \theta^{x_i}(1-\theta)^{1-x_i}$$

And we can also call it the likelihood function:

$$L(\theta) = \prod_{i=1}^{N} \theta^{x_i}(1-\theta)^{1-x_i}$$

We usually work with log-likelihoods:

$$\ell(\theta) = \sum_{i=1}^{N} x_i \log\theta + (1-x_i)\log(1-\theta)$$

- Good values of θ should assign high probability to the observed data. This motivates the maximum likelihood criterion, that we should pick the parameters that maximize the likelihood:

$$\hat{\theta}_{\mathrm{ML}} = \max_{\theta\in[0,1]} \ell(\theta)$$

- Setting this to zero gives the maximum likelihood estimate:

$$\begin{aligned}\frac{\mathrm{d}\ell}{\mathrm{d}\theta} &= \frac{\mathrm{d}}{\mathrm{d}\theta}\left(\sum_{i=1}^{N} x_i \log\theta + (1-x_i)\log(1-\theta)\right) \\ &= \frac{\mathrm{d}}{\mathrm{d}\theta}\left(N_H \log\theta + N_T \log(1-\theta)\right) \\ &= \frac{N_H}{\theta} - \frac{N_T}{1-\theta}\end{aligned} \longrightarrow \hat{\theta}_{\mathrm{ML}} = \frac{N_H}{N_H + N_T}$$

where $N_H = \sum_i x_i$ and $N_T = N - \sum_i x_i$.

- By finding the maximum likelihood we actually minimizing cross-entropies too:

$$\begin{aligned}\hat{\theta}_{\mathrm{ML}} &= \max_{\theta\in[0,1]} \ell(\theta) \\ &= \min_{\theta\in[0,1]} -\ell(\theta) \\ &= \min_{\theta\in[0,1]} \sum_{i=1}^{N} -x_i \log\theta - (1-x_i)\log(1-\theta)\end{aligned}$$

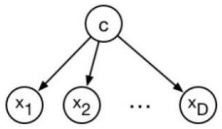
- There are two approaches to classification:
  - Discriminative approach: estimate parameters of decision boundary/class separator directly from labeled examples.
    - Model $p(t|\mathbf{x})$ directly (logistic regression models)
    - Learn mappings from inputs to classes (linear/logistic regression, decision trees etc)
    - Tries to solve: How do I separate the classes?
  - Generative approach: model the distribution of inputs characteristic of the class (Bayes classifier).
    - Model $p(\mathbf{x}|t)$
    - Apply Bayes Rule to derive $p(t|\mathbf{x})$
    - Tries to solve: What does each class "look" like?
  - Key difference: is there a distributional assumption over inputs?

9.2 Two approaches

9.2.1 A Generative Model: Bayes Classifier

$$\underbrace{p(c|\mathbf{x})}_{\text{Pr. class given words}} = \frac{p(\mathbf{x},c)}{p(\mathbf{x})} = \frac{\overbrace{p(\mathbf{x}|c)}^{\text{Pr. words given class}}\ p(c)}{p(\mathbf{x})}$$

- Problem: specifying a joint distribution over $D+1$ binary variables require $2^{D+1}-1$ entries. This is computationally prohibitive and would require an absurd amount of data to fit.
- We can represent model using an directed graphical model, or Bayesian network:



  - In a Bayesian network, each node represents a random variable and the arrows represent the dependencies between the variables. If there is an arrow pointing from node A to node B, then we can say "node B depends on node A". This dependency is usually expressed in terms of conditional probabilities.
  - The parameters can be learned efficiently because the log-likelihood decomposes into independent terms for each feature. Each of these log-likelihood terms depends on different sets of parameters, so they can be optimized independently.
- We predict the category by performing inference in the model, apply Bayes' Rule:

$$p(c\,|\,\mathbf{x}) = \frac{p(c)p(\mathbf{x}\,|\,c)}{\sum_{c'} p(c')p(\mathbf{x}\,|\,c')} = \frac{p(c)\prod_{j=1}^{D} p(x_j\,|\,c)}{\sum_{c'} p(c')\prod_{j=1}^{D} p(x_j\,|\,c')}$$

$$p(c\,|\,\mathbf{x}) \propto p(c)\prod_{j=1}^{D} p(x_j\,|\,c)$$

- For input x, predict by comparing the values of $p(c)\prod_{j=1}^{D} p(x_j\,|\,c)$ for different c (e.g. choose the largest).

- Demerit and Merit:
  - Naive Bayes is an amazingly cheap learning algorithm!
  - Training time: estimate parameters using maximum likelihood
    - ➢ Compute co-occurrence counts of each feature with the labels.
    - ➢ Requires only one pass through the data!
  - Test time: apply Bayes' Rule
    - ➢ Cheap because of the model structure. (For more general models, Bayesian inference can be very expensive and/or complicated.)
  - We covered the Bernoulli case for simplicity. But our analysis easily extends to other probability distributions.
  - Unfortunately, it's usually less accurate in practice compared to discriminative models due to its "naive" independence assumption.

9.2.2 MLE issue: Data Sparsity

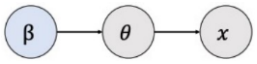
- Maximum likelihood has a pitfall: if you have too little data, it can overfit.
- E.g., what if you flip the coin twice and get H both times?

$$\theta_{\mathrm{ML}} = \frac{N_H}{N_H + N_T} = \frac{2}{2+0} = 1$$

- Because it never observed T, it assigns this outcome probability 0. This problem is known as data sparsity.

9.2.3 Bayesian Parameter Estimation

- In maximum likelihood, the observations are treated as random variables, but the parameters are not.
- The Bayesian approach treats the parameters as random variables as well. β is the set of parameters in the prior distribution of θ.



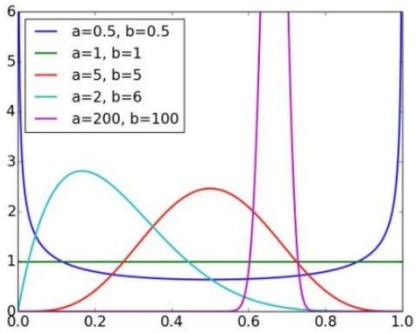
- To define a Bayesian model, we need to specify two distributions:
  - ➢The prior distribution $p(\theta)$, which encodes our beliefs about the parameters before we observe the data.
  - ➢The likelihood $p(D|\theta)$, same as in maximum likelihood.
- When we update our beliefs based on the observations, we compute the posterior distribution using Bayes' Rule:

$$p(\boldsymbol{\theta} \mid \mathcal{D}) = \frac{p(\boldsymbol{\theta}) p(\mathcal{D} \mid \boldsymbol{\theta})}{\int p(\boldsymbol{\theta}') p(\mathcal{D} \mid \boldsymbol{\theta}') \, \mathrm{d}\boldsymbol{\theta}'}$$

For the coin example, our experience tells us 0.5 is more likely than 0.99. One particularly useful prior that lets us specify this is the beta distribution (beta distribution 是指一种连续概率分布，它定义在区间 [0, 1] 上，用两个正参数 α 和 β 来控制分布的形状。它可以作为先验分布来表示对概率参数 θ 的不确定性):

$$p(\theta; a, b) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \theta^{a-1}(1-\theta)^{b-1}$$

$$p(\theta; a, b) \propto \theta^{a-1}(1-\theta)^{b-1}$$



- ➢The expectation $E[\theta] = a/(a+b)$ (easy to derive).
- ➢The distribution gets more peaked when a and b are large.
- ➢The uniform distribution is the special case where $a = b = 1$.

- Computing the posterior distribution:

$$\begin{aligned} p(\boldsymbol{\theta} \mid \mathcal{D}) &\propto p(\boldsymbol{\theta}) p(\mathcal{D} \mid \boldsymbol{\theta}) \\ &\propto \left[\theta^{a-1}(1-\theta)^{b-1}\right] \left[\theta^{N_H}(1-\theta)^{N_T}\right] \\ &= \theta^{a-1+N_H}(1-\theta)^{b-1+N_T}. \end{aligned}$$

(This is just a beta distribution with parameters $N_H + a$ and $N_T + b$)

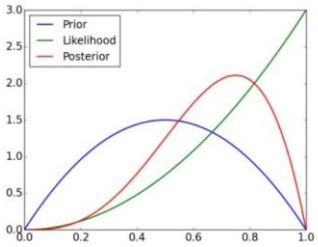
- The posterior expectation of θ is:

$$\mathbb{E}[\theta \mid \mathcal{D}] = \frac{N_H + a}{N_H + N_T + a + b}$$

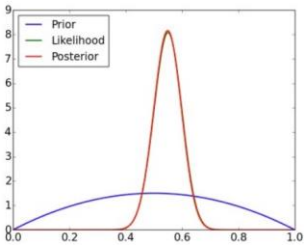
(The parameters a and b of the prior can be thought of as pseudo-counts)

The reason this works is that the prior and likelihood have the same functional form. This phenomenon is known as conjugacy (conjugate priors), and it's very useful.

Small data setting
$N_H = 2,\ N_T = 0$

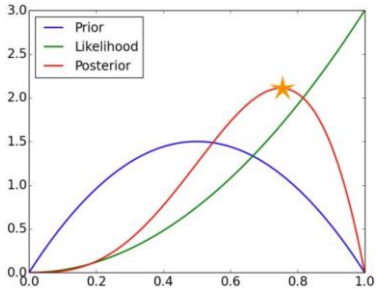


Large data setting
$N_H = 55,\ N_T = 45$



9.2.4 Maximum A-Posteriori Estimation

Maximum a-posteriori (MAP) estimation: find the most likely parameter settings under the posterior



$$\begin{aligned}\hat{\boldsymbol{\theta}}_{\text{MAP}} &= \arg\max_{\boldsymbol{\theta}}\ p(\boldsymbol{\theta}\,|\,\mathcal{D}) \\ &= \arg\max_{\boldsymbol{\theta}}\ p(\boldsymbol{\theta},\mathcal{D}) \\ &= \arg\max_{\boldsymbol{\theta}}\ p(\boldsymbol{\theta})\,p(\mathcal{D}\,|\,\boldsymbol{\theta}) \\ &= \arg\max_{\boldsymbol{\theta}}\ \log p(\boldsymbol{\theta}) + \log p(\mathcal{D}\,|\,\boldsymbol{\theta})\end{aligned}$$

Joint probability in the coin flip example:

$$\begin{aligned}\log p(\theta,\mathcal{D}) &= \log p(\theta) + \log p(\mathcal{D}\,|\,\theta) \\ &= \text{Const} + (a-1)\log\theta + (b-1)\log(1-\theta) + N_H\log\theta + N_T\log(1-\theta) \\ &= \text{Const} + (N_H + a - 1)\log\theta + (N_T + b - 1)\log(1-\theta)\end{aligned}$$

Maximize by finding a critical point:

$$0 = \frac{\mathrm{d}}{\mathrm{d}\theta}\log p(\theta,\mathcal{D}) = \frac{N_H + a - 1}{\theta} - \frac{N_T + b - 1}{1-\theta}$$

$$\hat{\theta}_{\text{MAP}} = \frac{N_H + a - 1}{N_H + N_T + a + b - 2}$$

Comparison of estimates in the coin flip example:

	Formula	$N_H = 2, N_T = 0$	$N_H = 55, N_T = 45$
$\hat{\theta}_{\text{ML}}$	$\frac{N_H}{N_H+N_T}$	1	$\frac{55}{100} = 0.55$
$\mathbb{E}[\theta\|\mathcal{D}]$	$\frac{N_H+a}{N_H+N_T+a+b}$	$\frac{4}{6} \approx 0.67$	$\frac{57}{104} \approx 0.548$
$\hat{\theta}_{\text{MAP}}$	$\frac{N_H+a-1}{N_H+N_T+a+b-2}$	$\frac{3}{4} = 0.75$	$\frac{56}{102} \approx 0.549$

$\hat{\theta}_{\text{MAP}}$ assigns nonzero probabilities as long as $a, b > 1$

10 k-Means and EM Algorithm

10.1 K-means

10.1.1 K-means Objective

Find cluster centers $\{m_k\}_{k=1}^K$ and assignments $\{r^{(n)}\}_{n=1}^N$ to minimize the sum of squared distances of data points $\{x^{(n)}\}$ to their assigned centers.

- Data sample $n = 1,\cdots,N$: $\mathbf{x}^{(n)} \in \mathbb{R}^D$ (observed),
- Cluster center $k = 1,\cdots,K$: $\mathbf{m}_k \in \mathbb{R}^D$ (not observed),
- Responsibilities: Cluster assignment for sample n: $\mathbf{r}^{(n)} \in \mathbb{R}^K$ 1-of-K encoding (not observed)

$$\min_{\{\mathbf{m}_k\},\{\mathbf{r}^{(n)}\}} J(\{\mathbf{m}_k\},\{\mathbf{r}^{(n)}\}) = \min_{\{\mathbf{m}_k\},\{\mathbf{r}^{(n)}\}} \sum_{n=1}^{N}\sum_{k=1}^{K} r_k^{(n)}||\mathbf{m}_k - \mathbf{x}^{(n)}||^2$$

where $r_k^{(n)} = \mathbb{I}[\mathbf{x}^{(n)} \text{ is assigned to cluster } k]$, i.e., $\mathbf{r}^{(n)} = [0,..,1,..,0]^\top$

(NP-hard problem)

So, the Optimization problem is:

$$\min_{\{\mathbf{m}_k\},\{\mathbf{r}^{(n)}\}} \sum_{n=1}^{N} \underbrace{\sum_{k=1}^{K} r_k^{(n)} ||\mathbf{m}_k - \mathbf{x}^{(n)}||^2}_{\text{distance between } x^{(n)} \text{ and its assigned cluster center}}$$

- Since $r_k^{(n)} = \mathbb{I}[\mathbf{x}^{(n)}$is assigned to cluster $k]$, i.e., $\mathbf{r}^{(n)} = [0, \cdots, 1, \cdots, 0]^T$
- inner sum is over K terms but only one of them is non-zero.
- E.g. say sample $\mathbf{x}^{(n)}$ is assigned to cluster $k = 3$, then

$$\mathbf{r}^n = [0, 0, 1, 0, \ldots]$$

$$\sum_{k=1}^{K} r_k^{(n)} ||\mathbf{m}_k - \mathbf{x}^{(n)}||^2 = ||\mathbf{m}_3 - \mathbf{x}^{(n)}||^2$$

Problem is hard when minimizing jointly over the parameters $\{\boldsymbol{m}_k\}$, $\{\boldsymbol{r}^{(n)}\}$. But if we fix one and minimize over the other, then it becomes easy. Also, it doesn't guarantee the same solution.

$$\min_{\mathbf{r}^{(n)}} \sum_{k=1}^{K} r_k^{(n)} ||\mathbf{m}_k - \mathbf{x}^{(n)}||^2$$

Assign each point to the cluster with the nearest center.

$$r_k^{(n)} = \begin{cases} 1 & \text{if } k = \arg\min_j \|\mathbf{x}^{(n)} - \mathbf{m}_j\|^2 \\ 0 & \text{otherwise} \end{cases}$$

E.g. if $\boldsymbol{x}^{(n)}$ is assigned to cluster $\hat{k}$.

$$\mathbf{r}^{(n)} = \underbrace{[0, 0, \ldots, 1, \ldots, 0]^\top}_{\text{Only } \hat{k}\text{-th entry is 1}}$$

10.1.2 Alternating Minimization

- Likewise, if we fix the assignments $\{\mathbf{r}^{(n)}\}$ then can easily find optimal centers $\{\mathbf{m}_k\}$

➢ Set each cluster's center to the average of its assigned data points: For $l = 1, 2, \ldots, K$

$$\begin{aligned} 0 &= \frac{\partial}{\partial \mathbf{m}_l} \sum_{n=1}^{N} \sum_{k=1}^{K} r_k^{(n)} ||\mathbf{m}_k - \mathbf{x}^{(n)}||^2 \\ &= 2 \sum_{n=1}^{N} r_l^{(n)} (\mathbf{m}_l - \mathbf{x}^{(n)}) \quad \Longrightarrow \quad \mathbf{m}_l = \frac{\sum_n r_l^{(n)} \mathbf{x}^{(n)}}{\sum_n r_l^{(n)}} \end{aligned}$$

- Let's alternate between minimizing $J(\{\mathbf{m}_k\}, \{\mathbf{r}^{(n)}\})$ with respect to $\{\mathbf{m}_k\}$ and $\{\mathbf{r}^{(n)}\}$
- This is called alternating minimization

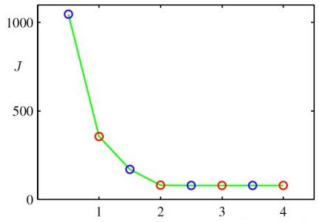
10.1.3 K-means Algorithm

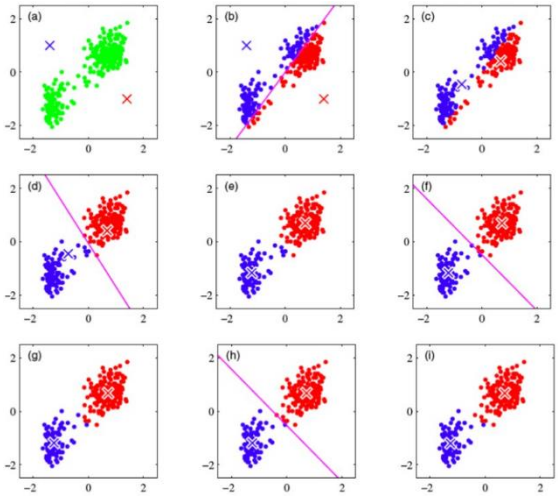
- Initialization: Set K cluster means **m**1, …, **m**K to random values
- Repeat until convergence (until assignments do not change):
  - ➢Assignment: Optimize J w.r.t. {**r**}: Each data point x(n) assigned to nearest center and Responsibilities (1-hot or 1-of-K encoding)

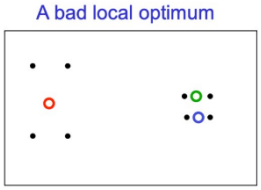
$$\hat{k}^{(n)} = \arg\min_k ||\mathbf{m}_k - \mathbf{x}^{(n)}||^2 \qquad r_k^{(n)} = \mathbb{I}[\hat{k}^{(n)} = k] \quad \text{for} \quad k = 1, .., K$$

  - ➢Refitting: Optimize J w.r.t. {**m**}: Each center is set to mean of data assigned to it

$$\mathbf{m}_k = \frac{\sum_n r_k^{(n)} \mathbf{x}^{(n)}}{\sum_n r_k^{(n)}}$$







Q: How can we reduce the local minima?

A: By try many random starting points.

10.1.4 Soft K-means Algorithm

• Instead of making hard assignments of data points to clusters, we can make soft assignments. One cluster may have a responsibility of $.7$ for a datapoint and another may have a responsibility of $.3$.

- Initialization: Set K means $\{\mathbf{m}_k\}$ to random values
- Repeat until convergence (measured by how much J changes):
  - ➢Assignment: Each data point n given soft "degree of assignment" to each cluster mean k, based on responsibilities

$$r_k^{(n)} = \frac{\exp[-\beta\|\mathbf{m}_k - \mathbf{x}^{(n)}\|^2]}{\sum_j \exp[-\beta\|\mathbf{m}_j - \mathbf{x}^{(n)}\|^2]}$$

$$\Longrightarrow \mathbf{r}^{(n)} = \text{softmax}(-\beta\{\|\mathbf{m}_k - \mathbf{x}^{(n)}\|^2\}_{k=1}^K)$$

  - ➢Refitting: Model parameters, means, are adjusted to match sample means of datapoints they are responsible for

$$\mathbf{m}_k = \frac{\sum_n r_k^{(n)}\mathbf{x}^{(n)}}{\sum_n r_k^{(n)}}$$

10.2 Multivariate Data

- Multiple measurements (sensors)
- D inputs / features / attributes
- N instances / observations / examples

$$\mathbf{X} = \begin{bmatrix} [\mathbf{x}^{(1)}]^\top \\ [\mathbf{x}^{(2)}]^\top \\ \vdots \\ [\mathbf{x}^{(N)}]^\top \end{bmatrix} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \cdots & x_D^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \cdots & x_D^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(N)} & x_2^{(N)} & \cdots & x_D^{(N)} \end{bmatrix}$$

10.2.1 Multivariate Mean and Covariance

- Mean

$$\boldsymbol{\mu} = \mathbb{E}[\mathbf{x}] = \begin{pmatrix} \mu_1 \\ \vdots \\ \mu_d \end{pmatrix}$$

- Covariance

$$\boldsymbol{\Sigma} = \text{Cov}(\mathbf{x}) = \mathbb{E}[(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^\top] = \begin{pmatrix} \sigma_1^2 & \sigma_{12} & \cdots & \sigma_{1D} \\ \sigma_{12} & \sigma_2^2 & \cdots & \sigma_{2D} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{D1} & \sigma_{D2} & \cdots & \sigma_D^2 \end{pmatrix}$$

- The statistics ($\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$) uniquely define a multivariate Gaussian (or multivariate Normal) distribution, denoted $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ or $\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$

10.2.2 Multivariate Gaussian Distribution

- Normally distributed variable $x \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ has distribution:

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right]$$

- We'll be working with the following generative model for data $\mathcal{D}$
- Assume a datapoint $\mathbf{x}$ is generated as follows:
  - Choose a cluster z from $\{1,\ldots,K\}$ such that $p(z = k) = \pi_k$
  - Given z, sample $\mathbf{x}$ from a Gaussian distribution $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_z, \mathbf{I})$
- Can also be written:

$$p(z = k) = \pi_k$$

$$p(\mathbf{x}|z = k) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \mathbf{I})$$

10.2.3 Fitting GMMs: Maximum Likelihood

Maximum likelihood objective:

$$\log p(\mathcal{D}) = \sum_{n=1}^{N} \log p(\mathbf{x}^{(n)}) = \sum_{n=1}^{N} \log\left(\sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}^{(n)}|\boldsymbol{\mu}_k, \mathbf{I})\right)$$

- **Observation: if we knew** $z^{(n)}$ for every $\mathbf{x}^{(n)}$(i.e. our dataset was $\mathcal{D}_{\text{complete}} = \{(z^{(n)}, \mathbf{x}^{(n)})\}_{n=1}^{N}$) the maximum likelihood problem is easy:

$$\begin{aligned}
\log p(\mathcal{D}_{\text{complete}}) &= \sum_{n=1}^{N} \log p(z^{(n)}, \mathbf{x}^{(n)}) \\
&= \sum_{n=1}^{N} \log p(\mathbf{x}^{(n)}|z^{(n)}) + \log p(z^{(n)}) \\
&= \sum_{n=1}^{N}\sum_{k=1}^{K} \mathbb{I}[z^{(n)} = k]\left(\log \mathcal{N}(\mathbf{x}^{(n)}|\boldsymbol{\mu}_k, \mathbf{I}) + \log \pi_k\right)
\end{aligned}$$

- By maximizing $\log p(\mathcal{D}_{\text{complete}})$, we would get this:

$$\begin{aligned}
\hat{\boldsymbol{\mu}}_k &= \frac{\sum_{n=1}^{N} \mathbb{I}[z^{(n)} = k]\,\mathbf{x}^{(n)}}{\sum_{n=1}^{N} \mathbb{I}[z^{(n)} = k]} = \text{class means} \\
\hat{\pi}_k &= \frac{1}{N}\sum_{n=1}^{N} \mathbb{I}[z^{(n)} = k] = \text{class proportions}
\end{aligned}$$

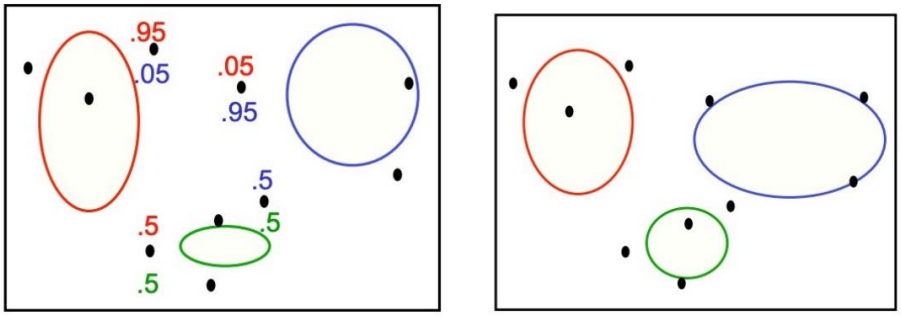
- We haven't observed the cluster assignments $z^{(n)}$, but we can compute $p(z^{(n)}|\mathbf{x}^{(n)})$ using Bayes rule
- Conditional probability (using Bayes rule) of z given $\mathbf{x}$

$$\begin{aligned}
p(z = k|\mathbf{x}) &= \frac{p(z = k)p(\mathbf{x}|z = k)}{p(\mathbf{x})} \\
&= \frac{p(z = k)p(\mathbf{x}|z = k)}{\sum_{j=1}^{K} p(z = j)p(\mathbf{x}|z = j)} \\
&= \frac{\pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \mathbf{I})}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_j, \mathbf{I})}
\end{aligned}$$

10.2 EM Algorithm

10.2.1 EM Algorithm

- This motivates the Expectation-Maximization algorithm, which alternates between two steps:
  1. E-step: Compute the posterior probabilities $r_k^{(n)} = p(z^{(n)} = k|\mathbf{x}^{(n)})$ our current model - i.e. how much do we think a cluster is responsible for generating a datapoint.
  2. M-step: Use the equations on the last slide to update the parameters, assuming $r_k^{(n)}$ are held fixed- change the parameters of each Gaussian to maximize the probability that it would generate the data it is currently responsible for.



- Initialize the means $\hat{\boldsymbol{\mu}}_k$ and mixing coefficients $\hat{\pi}_k$
- Iterate until convergence:
  - E-step: Evaluate the responsibilities $r_k^{(n)}$ given current parameters

$$r_k^{(n)} = p(z^{(n)}=k|\mathbf{x}^{(n)}) = \frac{\hat{\pi}_k \mathcal{N}(\mathbf{x}^{(n)}|\hat{\boldsymbol{\mu}}_k, \mathbf{I})}{\sum_{j=1}^K \hat{\pi}_j \mathcal{N}(\mathbf{x}^{(n)}|\hat{\boldsymbol{\mu}}_j, \mathbf{I})} = \frac{\hat{\pi}_k \exp\{-\frac{1}{2}\|\mathbf{x}^{(n)} - \hat{\boldsymbol{\mu}}_k\|^2\}}{\sum_{j=1}^K \hat{\pi}_j \exp\{-\frac{1}{2}\|\mathbf{x}^{(n)} - \hat{\boldsymbol{\mu}}_j\|^2\}}$$

  - M-step: Re-estimate the parameters given current responsibilities

$$\hat{\boldsymbol{\mu}}_k = \frac{1}{N_k}\sum_{n=1}^N r_k^{(n)}\mathbf{x}^{(n)}$$

$$\hat{\pi}_k = \frac{N_k}{N} \quad \text{with} \quad N_k = \sum_{n=1}^N r_k^{(n)}$$

  - Evaluate log likelihood and check for convergence

$$\log p(\mathcal{D}) = \sum_{n=1}^N \log\left(\sum_{k=1}^K \hat{\pi}_k \mathcal{N}(\mathbf{x}^{(n)}|\hat{\boldsymbol{\mu}}_k, \mathbf{I})\right)$$

10.2.2 Review

- The maximum likelihood objective $\sum_{n=1}^N \log p(\mathbf{x}^{(n)})$ was hard to optimize
- The complete data likelihood objective was easy to optimize:

$$\sum_{n=1}^N \log p(z^{(n)}, \mathbf{x}^{(n)}) = \sum_{n=1}^N \sum_{k=1}^K \mathbb{I}[z^{(n)} = k](\log \mathcal{N}(\mathbf{x}^{(n)}|\boldsymbol{\mu}_k, \mathbf{I}) + \log \pi_k)$$

- We don't know $z^{(n)}$'s (they are latent), so we replaced $\mathbb{I}[z^{(n)} = k]$ with responsibilities $r_k^{(n)} = p(z^{(n)} = k|\mathbf{x}^{(n)})$
- That is: we replaced $\mathbb{I}[z^{(n)} = k]$ with its expectation under $p(z^{(n)}|\mathbf{x}^{(n)})$ (E-step).
- We ended up with the expected complete data log-likelihood:

$$\sum_{n=1}^N \mathbb{E}_{p(z^{(n)}|\mathbf{x}^{(n)})}[\log p(z^{(n)}, \mathbf{x}^{(n)})] = \sum_{n=1}^N \sum_{k=1}^K r_k^{(n)}(\log \mathcal{N}(\mathbf{x}^{(n)}|\boldsymbol{\mu}_k, \mathbf{I}) + \log \pi_k)$$

  which we maximized over parameters $\{\pi_k, \boldsymbol{\mu}_k\}_k$(M-step)

- The EM algorithm alternates between
  - The E-step: computing the $r_k^{(n)} = p(z^{(n)} = k|\mathbf{x}^{(n)})$ (i.e., expectation $\mathbb{E}[\mathbb{I}[z^{(n)} = k]|\mathbf{x}^{(n)}]$) given the current model parameter $\pi_k, \boldsymbol{\mu}_k$
  - The M-step: update the model parameters $\pi_k, \boldsymbol{\mu}_k$ to optimize the expected complete data log-likelihood

- The K-Means Algorithm:
    1. Assignment step: Assign each data point to the closest cluster.
    2. Refitting step: Move each cluster center to the average of the data assigned to it.
- The EM Algorithm:
    1. E-step: Compute the posterior probability over z given our current model.
    2. M-step: Maximize the probability that it would generate the data it is currently responsible for.