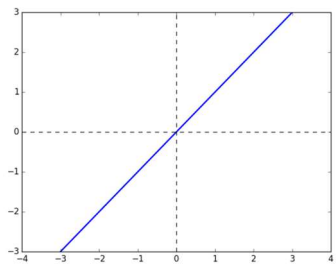
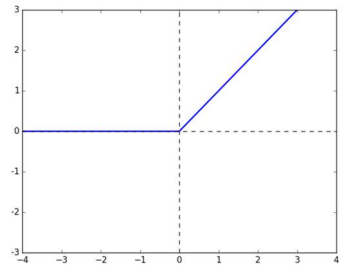
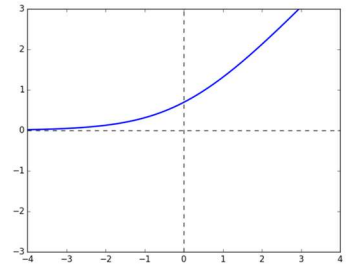
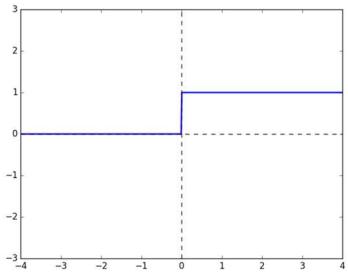
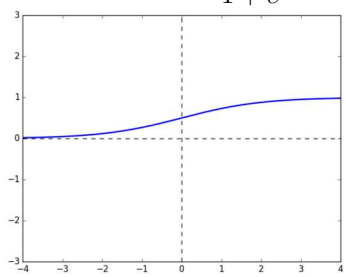
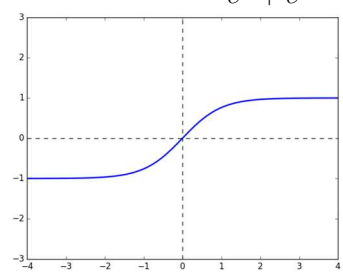
Neural Networks

Neuron/Unit: A model of a real neuron, which activates after a threshold of voltage is passed. Like logistic regression.

$$y(\mathbf{x}) = \phi\big(f(\mathbf{x})\big) = \phi(\mathbf{w} \cdot \mathbf{x} + b)$$

Activation Function (ϕ): The threshold that controls the activation of a neuron.

Identity	Rectified Linear Unit (ReLU)	Soft ReLU
$\phi(x) = x$	$\phi(x) = \max(0, x)$	$\phi(x) = \log(1 + e^x)$
		
Hard Threshold	**Logistic**	**Hyperbolic Tangent**
$\phi(x) = \mathbb{I}(x > 0)$	$\phi(x) = \sigma(x) = \dfrac{1}{1 + e^{-x}}$	$\phi(x) = \tanh x = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$
		

Black Box: In computing, a program where users don't care about inner workings, just using its inputs/outputs.

Neural Network: A directed graph of neurons – has input layer of all $\mathbf{x}_i$, hidden layers, and output layer of all $y_i(\mathbf{x})$

- **Recurrent:** A neural network that is a directed graph with cycles
- **Feed-Forward:** A neural network that is a directed acyclic graph
- **Fully-Connected Layer:** A neural network where all units in one layer connect to all units in the next layer
- **Bottleneck Layer:** A layer with much smaller dimensions than surrounding layers. "Loses" info
- **Multilayer Perceptron:** A multilayer neural network of fully-connected layers.

Assume for a layer, N input neurons ($\mathbf{x} \in \mathbb{R}^N$), M output neurons ($h(\mathbf{x}) \in \mathbb{R}^M$). Also, ϕ is applied component-wise.

$$h(\mathbf{x}) = \phi\big(f(\mathbf{x})\big) = \phi(W\mathbf{x} + \mathbf{b}) = \phi\left(\begin{bmatrix} (w_1)_1 & \cdots & (w_1)_N \\ \vdots & \ddots & \vdots \\ (w_M)_1 & \cdots & (w_M)_N \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} + \begin{bmatrix} b \\ \vdots \\ b \end{bmatrix}\right)$$

A neural network is a composition of layer functions that can be implemented as a black box:

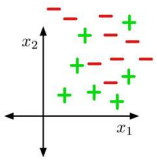
$$\begin{aligned} h_1(\mathbf{x}) &= \phi(W_1\mathbf{x} + \mathbf{b}_1) \\ (h_2 \circ h_1)(\mathbf{x}) &= \phi(W_2 h_1(\mathbf{x}) + \mathbf{b}_2) \\ (h_3 \circ h_2 \circ h_1)(\mathbf{x}) &= \phi(W_3(h_2 \circ h_1)(\mathbf{x}) + \mathbf{b}_3) \\ &\vdots \\ y(\mathbf{x}) = (h_K \circ \ldots \circ h_1)(\mathbf{x}) &= \phi^*(W_K(h_{K-1} \circ \ldots \circ h_1)(\mathbf{x}) + \mathbf{b}_K) \end{aligned}$$

Regression:
Choose $\phi^*(x) = x \in \mathbb{R}$

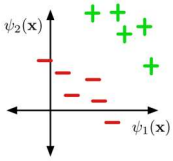
Binary Classification:
Choose $\phi^*(x) = \sigma(x) \in [0,1]$

For regression/binary classification, $W_K = \mathbf{w}_k$ (one output)

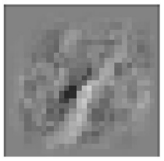
All h_k use weights W_k to transform



into something linearly classifiable,



When classifying digits, if we turn $\mathbf{w}_i$ to an image, we often see **oriented edges**, an example of **feature detection** in computer vision. Usually, these are more common in W_1 (ie. weights of layer 1)



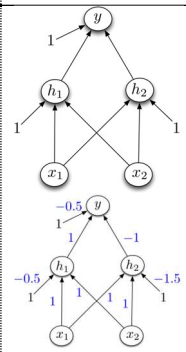
XOR, Binary Linear Classification, Feature Maps

$$f(\mathbf{x}) = \mathbf{w} \cdot \psi(\mathbf{x})$$
$$\psi(\mathbf{x}) = (x_0, x_1, x_2, x_1 x_2)$$

$\mathbf{x}^{\mathrm{T}}$			$\psi(\mathbf{x})$				t
x_0	x_1	x_2	$\psi_0(\mathbf{x})$	$\psi_1(\mathbf{x})$	$\psi_2(\mathbf{x})$	$\psi_3(\mathbf{x})$	
1	0	0	1	0	0	0	0
1	0	1	1	0	1	0	1
1	1	0	1	1	0	0	1
1	1	1	1	1	1	1	0

$$f(\mathbf{x}_1) = w_0(1) + w_1(0) + w_2(0) + w_1 w_2(0) < 0$$
$$f(\mathbf{x}_2) = w_0(1) + w_1(0) + w_2(1) + w_1 w_2(0) \geq 0$$
$$f(\mathbf{x}_3) = w_0(1) + w_1(1) + w_2(0) + w_1 w_2(0) \geq 0$$
$$f(\mathbf{x}_4) = w_0(1) + w_1(1) + w_2(1) + w_1 w_2(1) \geq 0$$

XOR, Neural Network



We set $\phi(x) = \mathbb{I}(x > 0)$.
Each edge is a w_i we must find.
Each 1 represents a x_0 with weight w_0 (AKA b)

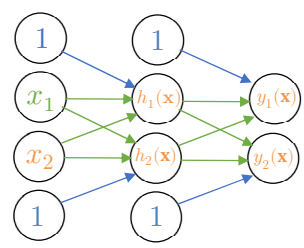
$$h_1(\mathbf{x}) = \mathbb{I}(x_1(1) \quad + x_2(1) \quad + 1(-0.5\ ) > 0)$$
$$h_2(\mathbf{x}) = \mathbb{I}(x_1(1) \quad + x_2(1) \quad + 1(-1.5\ ) > 0)$$
$$y(\mathbf{x}) = \mathbb{I}(h_1(\vec{x})(1) + h_2(\vec{x})(-1) + 1(-0.5\ ) > 0)$$

Notice the connection to logic:

$$h_1(\mathbf{x}) = \mathbb{I}(x_1 + x_2 > 0.5) \equiv x_1 \vee x_2$$
$$h_2(\mathbf{x}) = \mathbb{I}(x_1 + x_2 > 1.5) \equiv x_1 \wedge x_2$$
$$y(\mathbf{x}) = \mathbb{I}(h_1(\mathbf{x}) + (1 - h_2(\mathbf{x})) > 1.5)$$
$$\equiv h_1(\mathbf{x}) \wedge (\neg h_2(\mathbf{x})) \equiv x_1 \text{ XOR } x_2$$

For 1-layer XOR on n inputs, use 2^n neurons for all T/F combinations.

- Binary linear classification can't classify XOR without feature maps, which are hard to design, especially for very complex functions. Neural networks help find these feature maps for us.
- The hard threshold/0-1 loss $\phi(x) = \mathbb{I}(x > 0)$ is not ideal for gradient descent as the derivative is almost all 0
- Use the smooth logistic function $\phi(x) = \sigma(nx)$ (where n is large), which approaches the hard threshold.



Depth: Number of layers in a neural network (**shallow** means little layers, **deep** means much layers, **deep learning** refers to training high-depth neural networks)
Width: Number of units in a layer
Input Space: Set $\mathcal{X}$ of all possible inputs
Target Space: Set $\mathcal{T}$ of all possible targets
Hypothesis: A function $f: \mathcal{X} \to \mathcal{T}$
Hypothesis Space: The set $\mathcal{H}$ of all possible hypotheses f a given model can have

- *Parametric Algorithms:* $\mathcal{H}$ is a set of all parameter combinations
- *Non-Parametric Algorithms:* $\mathcal{H}$ is defined in terms of the training data
- **Inductive Bias:** Of an algorithm, its preference to choose some hypotheses $f \in \mathcal{H}$ over others.
  - eg. For linear regression, $\mathcal{H}$ is the set of linear functions
  - eg. Ridge regression results in an inductive bias for smaller weights
- **Expressivity:** The variety of functions able to be represented by $\mathcal{H}$
  - $\mathcal{H}_B \subseteq \mathcal{H}_A$ means model A is more expressive; it can represent any function $f \in \mathcal{H}_B$
  - eg. Neural networks where $\phi(x) = x$ are <u>as expressive as linear regression</u>. Every layer $h_k(x)$ is a linear function, which is equivalent to a single linear layer.
  - eg. Multilayer feed-forward neural networks with nonlinear ϕ are **universal function approximators**: they can approximate any function arbitrarily well. $\forall f: \mathcal{X} \to \mathcal{T}, \exists f_i \in \mathcal{H}, f_i \to f$
    - This is not necessarily a good thing – **overfitting** is a concern
    - **Early Stopping:** Stopping training just as generalization error begins to increase

No Free Lunch Theorem: If datasets were not naturally biased, no ML algorithm would be better than any other.

Let $\star_n$ represent the n-th component of function $\star$, and $\star^{(n)}$ represent a value in the n-th layer.

$$h_1(\mathbf{x}) = \phi\left(W_1^{(1)} \cdot \mathbf{x}\right) = \phi\left[(w_1)_0^{(1)} + (w_1)_1^{(1)} x_1 + (w_1)_2^{(1)} x_2\right]$$
$$h_2(\mathbf{x}) = \phi\left(W_2^{(1)} \cdot \mathbf{x}\right) = \phi\left[(w_2)_0^{(1)} + (w_2)_1^{(1)} x_1 + (w_2)_2^{(1)} x_2\right]$$
$$y_1(\mathbf{x}) = \phi\left(W_1^{(2)} \cdot h(\mathbf{x})\right) = \phi\left[(w_1)_0^{(2)} + (w_1)_1^{(2)} h_1(\mathbf{x}) + (w_1)_2^{(2)} h_2(\mathbf{x})\right]$$
$$y_2(\mathbf{x}) = \phi\left(W_2^{(2)} \cdot h(\mathbf{x})\right) = \phi\left[(w_2)_0^{(2)} + (w_2)_1^{(2)} h_1(\mathbf{x}) + (w_2)_2^{(2)} h_2(\mathbf{x})\right]$$

Note: Use $\mathbf{x} = (1, x_1, \dots, x_n)$ and $h(\mathbf{x}) = (1, h_1(\mathbf{x}), \dots, h_n(\mathbf{x}))$ since we squish b into $W = (W_1, W_2)$

$$h(\mathbf{x}) = \phi\left(\begin{bmatrix} (w_1)_0 & (w_1)_1 & (w_2)_1 \\ (w_2)_0 & (w_2)_1 & (w_2)_2 \end{bmatrix}^{(1)} \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}\right)$$
$$= \phi(W^{(1)}\mathbf{x})$$
$$y(\mathbf{x}) = \phi\left(\begin{bmatrix} (w_1)_0 & (w_1)_1 & (w_2)_1 \\ (w_2)_0 & (w_2)_1 & (w_2)_2 \end{bmatrix}^{(2)} \begin{bmatrix} 1 \\ h_1(\mathbf{x}) \\ h_2(\mathbf{x}) \end{bmatrix}\right)$$
$$= \phi\left(W^{(2)} h(\mathbf{x})\right)$$

Calculating Univariate $\frac{\partial \mathcal{L}}{\partial \mathbf{w}}$ the Normal Way

Suppose we have the network

$$f = f(\mathbf{x}) = \mathbf{w}\cdot\mathbf{x} + b$$
$$y = y(\mathbf{x}) = \sigma(f(\mathbf{x}))$$
$$\mathcal{L} = \mathcal{L}(y(\mathbf{x}), t) = \frac{1}{2}(y(\mathbf{x}) - t)^2$$

The derivation for $\sigma'(f(\mathbf{x}))$ is the exact same as for binary classification, logistic regression.

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathbf{w}} &= \frac{\partial \mathcal{L}}{\partial y}\cdot\frac{\partial y}{\partial f}\cdot\frac{\partial f}{\partial \mathbf{w}} \\
&= \frac{\partial}{\partial y}\left[\frac{1}{2}(y-t)^2\right]\frac{\partial}{\partial f}[\sigma(f)]\frac{\partial}{\partial \mathbf{w}}[(\mathbf{w}\cdot\mathbf{x}+b)] \\
&= \left[(y-t)\frac{\partial}{\partial y}(y-t)\right]\sigma'(f)\mathbf{x} \\
&= (y-t)y(1-y)\mathbf{x} \\
\frac{\partial \mathcal{L}}{\partial b} &= \frac{\partial \mathcal{L}}{\partial y}\cdot\frac{\partial y}{\partial f}\cdot\frac{\partial f}{\partial b} = (y-t)y(1-y)
\end{aligned}$$

Calculating Univariate $\frac{\partial \mathcal{L}}{\partial \mathbf{w}}$ the Efficient Way

$$\begin{aligned}
\overline{y} = \frac{\partial \mathcal{L}}{\partial y} &= \frac{\partial}{\partial y}\left[\frac{1}{2}(y-t)^2\right] \\
&= y - t \\
\overline{f} = \frac{\partial \mathcal{L}}{\partial f} &= \frac{\partial \mathcal{L}}{\partial y}\cdot\frac{\partial y}{\partial f} \\
&= \overline{y}(y(1-y))
\end{aligned}$$

$$\begin{aligned}
\overline{\mathbf{w}} = \frac{\partial \mathcal{L}}{\partial \mathbf{w}} &= \frac{\partial \mathcal{L}}{\partial f}\cdot\frac{\partial f}{\partial \mathbf{w}} \\
&= \overline{f}\mathbf{x} \\
\overline{b} = \frac{\partial \mathcal{L}}{\partial b} &= \frac{\partial \mathcal{L}}{\partial f}\cdot\frac{\partial f}{\partial b} \\
&= \overline{f}
\end{aligned}$$

We found the loss function in the order $f \to y \to \mathcal{L}$

We found derivatives in reverse order, $y \to f \to (\mathbf{w}, b)$.

The notation $\overline{*} = \frac{\partial \mathcal{L}}{\partial *}$ is called **error signal**.

$\otimes$ and $\ominus$ are element-wise $\times$ and $-$

Computational Graph: A directed graph where a function's most-direct dependencies point to it.

$$\begin{tikzcd}
\mathbf{x} \arrow[dr] & & t \arrow[dr] & \\
b \arrow[r] & f(\mathbf{w}) \arrow[r] & y(\mathbf{x}) \arrow[r] & \mathcal{L}(y(\mathbf{x}), t) \\
\mathbf{w} \arrow[ur] & & &
\end{tikzcd}$$

Topological Ordering: Of a directed graph G, a linear ordering L where $\forall u, v \in G$,

$$u \text{ points to } v \text{ in } G \Leftrightarrow u \text{ precedes } v \text{ in } L$$

Fan-out: The max number of children a node can have in a graph.

Backpropagation: A computationally efficient way of finding $\frac{\partial \mathcal{L}}{\partial \mathbf{w}}$ to update the weights of a neural network.

<u>Algorithm:</u>

Let $v_1, \dots, v_N$ be a topological ordering of the computation graph (ie. parents are before children).

1) **Forward Pass:** A loop $1 \to N$ down the computational graph AKA forward in the ordering.

   For each v_{child}, express it in terms of $\text{ParentFunc}(v_{\text{child}})$

2) **Backward Pass:** A loop $N \to 1$ up the computational graph AKA backward in the ordering.

   For each v_{parent}, calculate $\overline{v_i} = \sum_{\text{child}\in\text{ChildrenFunc}(v_i)} \overline{v_{\text{child}}}\frac{\partial v_{\text{child}}}{\partial v_{\text{parent}}}$

Backpropagation, Regularized Linear Regression $\frac{\partial \mathcal{L}}{\partial \mathbf{w}}$

$$\begin{tikzcd}
\mathbf{x} \arrow[dr] & & t \arrow[dr] & & \\
b \arrow[r] & f(\mathbf{x}) \arrow[r] & y(\mathbf{x}) \arrow[r] & \mathcal{L}(y(\mathbf{x}), t) \arrow[r] & \mathcal{L}_{\text{reg}}(y(\mathbf{x}), t) \\
\mathbf{w} \arrow[urr] \arrow[rrr] & & & \mathcal{R}(\mathbf{w}) \arrow[ur] &
\end{tikzcd}$$

Forward Pass

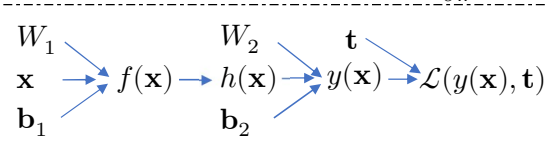
$$\begin{aligned}
f = f(\mathbf{x}) &= \mathbf{w}\cdot\mathbf{x} + b \\
y = y(\mathbf{x}) &= \sigma(f(\mathbf{x})) \\
\mathcal{L} = \mathcal{L}(y(\mathbf{x}), t) &= \frac{1}{2}(y(\mathbf{x}) - t)^2 \\
\mathcal{R} = \mathcal{R}(\mathbf{w}) &= \frac{1}{2}\|\mathbf{w}\|^2 \\
\mathcal{L}_{\text{reg}} = \mathcal{L}_{\text{reg}}(y(\mathbf{x}), t) &= \mathcal{L}(y(\mathbf{x}), t) + \lambda\mathcal{R}(\mathbf{w})
\end{aligned}$$

Backward Pass

$$\overline{\mathcal{L}_{\text{reg}}} = \frac{\partial \mathcal{L}_{\text{reg}}}{\partial \mathcal{L}_{\text{reg}}} = 1$$

$$\begin{aligned}
\overline{\mathcal{R}} = \frac{\partial \mathcal{L}_{\text{reg}}}{\partial \mathcal{R}} &= \overline{\mathcal{L}_{\text{reg}}}\frac{\partial \mathcal{L}_{\text{reg}}}{\partial \mathcal{R}} &&= \overline{\mathcal{L}_{\text{reg}}}\frac{\partial}{\partial \mathcal{R}}[\mathcal{L} + \lambda\mathcal{R}] = \boxed{\overline{\mathcal{L}_{\text{reg}}}\lambda} \\
\overline{\mathcal{L}} = \frac{\partial \mathcal{L}_{\text{reg}}}{\partial \mathcal{L}} &= \overline{\mathcal{L}_{\text{reg}}}\frac{\partial \mathcal{L}_{\text{reg}}}{\partial \mathcal{L}} &&= \overline{\mathcal{L}_{\text{reg}}}\frac{\partial}{\partial \mathcal{L}}[\mathcal{L} + \lambda\mathcal{R}] = \boxed{\overline{\mathcal{L}_{\text{reg}}}} \\
\overline{y} = \frac{\partial \mathcal{L}_{\text{reg}}}{\partial y} &= \overline{\mathcal{L}}\frac{\partial \mathcal{L}}{\partial y} &&= \overline{\mathcal{L}}\frac{\partial}{\partial y}\left[\frac{1}{2}(y-t)^2\right] = \boxed{\overline{\mathcal{L}}(y-t)} \\
\overline{f} = \frac{\partial \mathcal{L}_{\text{reg}}}{\partial f} &= \overline{y}\frac{\partial y}{\partial f} &&= \overline{y}\frac{\partial}{\partial f}[\sigma(f)] = \boxed{\overline{y}\sigma'(f)} \\
\overline{\mathbf{w}} = \frac{\partial \mathcal{L}_{\text{reg}}}{\partial \mathbf{w}} &= \overline{f}\frac{\partial f}{\partial \mathbf{w}} + \overline{\mathcal{R}}\frac{\partial \mathcal{R}}{\partial \mathbf{w}} &&= \overline{f}\frac{\partial}{\partial \mathbf{w}}[\mathbf{w}\cdot\mathbf{x} + b] + \overline{\mathcal{R}}\frac{\partial}{\partial \mathbf{w}}\left[\frac{1}{2}\|\mathbf{w}\|^2\right] = \boxed{\overline{f}\mathbf{x} + \overline{\mathcal{R}}\mathbf{w}} \\
\overline{b} = \frac{\partial \mathcal{L}_{\text{reg}}}{\partial b} &= \overline{f}\frac{\partial f}{\partial b} &&= \overline{f}\frac{\partial}{\partial b}[\mathbf{w}\cdot\mathbf{x} + b] = \boxed{\overline{f}}
\end{aligned}$$

Backpropagation, Simple Neural Network $\frac{\partial \mathcal{L}}{\partial \mathbf{w}}$



Forward Pass

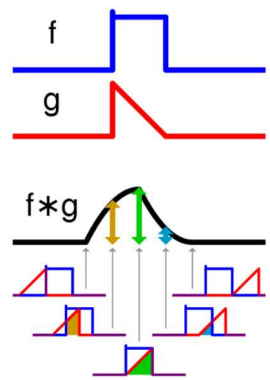
$$\mathbf{f} = f(\mathbf{x}) = W_1\mathbf{x} + \mathbf{b}_1$$
$$\mathbf{h} = h(\mathbf{x}) = \sigma(f(\mathbf{x})) \text{ (element-wise)}$$
$$\mathbf{y} = y(\mathbf{x}) = W_2 h(\mathbf{x}) + \mathbf{b}_2$$
$$\mathcal{L} = \mathcal{L}(y(\mathbf{x}), \mathbf{t}) = \frac{1}{2}\|y(\mathbf{x}) - \mathbf{t}\|^2$$

For backward passes, keep track of the error signals' dimensions. Change the order of matrix multiplication if necessary.

Backward Pass

$$\overline{\mathcal{L}} = \frac{\partial \mathcal{L}}{\partial \mathcal{L}} = 1$$
$$\overline{\mathbf{y}} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}} = \overline{\mathcal{L}}\frac{\partial \mathcal{L}}{\partial \mathbf{y}} = \overline{\mathcal{L}}\frac{\partial}{\partial y}\left[\frac{1}{2}\|\mathbf{y} - \mathbf{t}\|^2\right] = \boxed{\overline{\mathcal{L}}(\mathbf{y} - \mathbf{t})}$$
$$\overline{W_2} = \frac{\partial \mathcal{L}}{\partial W_2} = \overline{\mathbf{y}}\frac{\partial y}{\partial W_2} = \overline{\mathbf{y}}\frac{\partial}{\partial W_2}[W_2\mathbf{h} + \mathbf{b}_2] = \boxed{\overline{\mathbf{y}}\mathbf{h}^{\mathrm{T}}}$$
$$\overline{\mathbf{b}_2} = \frac{\partial \mathcal{L}}{\partial \mathbf{b}_2} = \overline{\mathbf{y}}\frac{\partial \mathbf{y}}{\partial \mathbf{b}_2} = \overline{\mathbf{y}}\frac{\partial}{\partial \mathbf{b}_2}[W_2\mathbf{h} + \mathbf{b}_2] = \boxed{\overline{\mathbf{y}}}$$
$$\overline{\mathbf{h}} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}} = \overline{\mathbf{y}}\frac{\partial \mathbf{y}}{\partial \mathbf{h}} = \overline{\mathbf{y}}\frac{\partial}{\partial \mathbf{h}}[W_2\mathbf{h} + \mathbf{b}_2] = \boxed{W_2^{\mathrm{T}}\overline{\mathbf{y}}}$$
$$\overline{\mathbf{f}} = \frac{\partial \mathcal{L}}{\partial \mathbf{f}} = \overline{\mathbf{h}}\frac{\partial \mathbf{h}}{\partial \mathbf{f}} = \overline{\mathbf{h}}\frac{\partial}{\partial \mathbf{f}}[\sigma(\mathbf{f})] = \boxed{\overline{\mathbf{h}} \otimes \sigma'(\mathbf{f})} \text{ (elementwise } \times)$$
$$\overline{W_1} = \frac{\partial \mathcal{L}}{\partial W_1} = \overline{\mathbf{f}}\frac{\partial \mathbf{f}}{\partial W_1} = \overline{\mathbf{f}}\frac{\partial}{\partial W_1}[W_1\mathbf{x} + \mathbf{b}_1] = \boxed{\overline{\mathbf{f}}\mathbf{x}^{\mathrm{T}}}$$
$$\overline{\mathbf{b}_1} = \frac{\partial \mathcal{L}}{\partial \mathbf{b}_1} = \overline{\mathbf{f}}\frac{\partial \mathbf{f}}{\partial \mathbf{b}_1} = \overline{\mathbf{f}}\frac{\partial}{\partial \mathbf{b}_1}[W_1\mathbf{x} + \mathbf{b}_1] = \boxed{\overline{\mathbf{f}}}$$

Convolution ($*$): In functional analysis, a mathematical operator on functions (sometimes called **signals** for the application in signal processing). In the discrete case:



For $f, g\colon \mathbb{Z} \to \mathbb{R}$,	*For $f, g\colon \mathbb{Z}^2 \to \mathbb{R}$,*
$(f * g)(x) = \sum_{i=-\infty}^{\infty} f(i)\, g(x - i)$	$(f * g)(x, y) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f(i, j)\, g[x - i, y - j]$

Computationally, think of signals f and g like an array. We find the sum of all $f[i] \times g.\mathrm{reverse}()[i]$

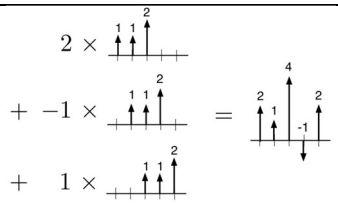
Properties of Convolution

$$(f * g)(x) = (g * f)(x)$$
$$(f * (\alpha g + \beta h))(x) = \alpha(f * g)(x) + \beta(f * h)(x)$$

The continuous case replaces sum with an integral. Visually, think of it like reflecting g (left) and sliding it left to right across f. The area intersecting f and g in any given moment is $f * g$.

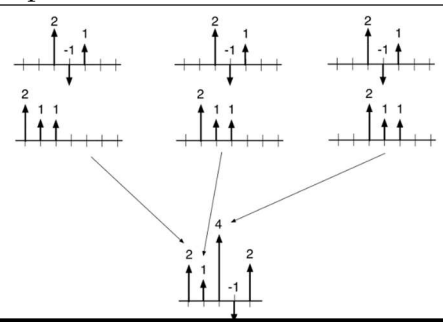
See this video for better visual explanation. Visually, for the discrete case, consider the problem $[2, -1, 1] * [1, 1, 2]$:

Translate-And-Scale



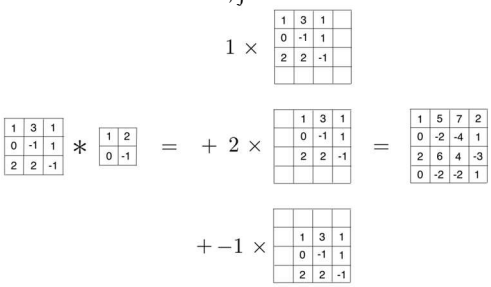
Zero Padding: Treating finite signals as infinitely large by adding 0s where the domain's undefined

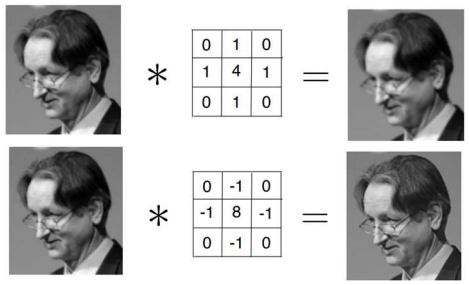
Flip-And-Filter



$(f * g)(i)$ is equal to the sum of all possible $f(i) \times g(i)$ obtained by sliding $g(i)$ rightwards.

For the 2D case, just consider the translate-and-scale method:

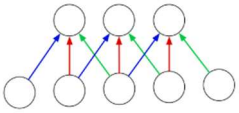




For any cell in the kernel matrix, consider the image formed by translating every pixel in the original image by 1 *(in the same direction as from the matrix's center cell to the given cell)*. Multiply the new image by the cell. Do this for every cell, and add the results.

Convolutional Network: A regularized type of multilayer perceptron specialized for image recognition

- **Locally-Connected Layers:** When neurons in one layer do not connect to every neuron in the next layer
- **Convolution Layer:** Layers where each set of neurons look at one image region, and weights are shared between all image locations. Can be stacked.



    - Consists of filters, functions which, when convolved with the image, creates a **feature map/channel**
    - Shared weights allow the network to detect features in many locations of the image
- Note that W is linear; if the system is completely linear, no number of layers is more powerful than 1 layer.
    - **Pooling Layer:** Layers that deliberately lose some information to reduce computational load and add non-linearity and invariance
        - **Max-Pooling:** When neurons take the max of neurons from a **pooling group** in the previous layer.
    - **Linear Rectification:** A non-linear activation function-based transformation that adds a non-linearity, like ReLU, $y_n(\vec{x}) = \max(0, h_n(\mathbf{x}))$
- **Equivariance:** A property of convolution layers – translating inputs translates outputs by the same amount
- **Invariance:** A property of convolution layers – translating inputs doesn't affect outputs

Probabilistic Modelling

Likelihood Function (L): Given observed data $\mathcal{D} = \{x_1, \dots, x_N\}$, a probability function $L(\dots)$ *(parameters vary)* that retroactively describes the probability distribution of the "real" underlying data *(where order of data matters)*.

- eg. $L(\theta) = p_{X|\theta}(\mathcal{D}|\theta) = p_{X|\theta}(x_1, \dots, x_N|\theta)$, where X is the "real" probability distribution of data, θ is a parameter. This reads as "the probability that the data would be $x_1, \dots, x_N$, given we know θ"
- Notation can be inconsistent. $p(\theta) = p_\theta(\theta) = p_\theta(k)$. I use k only if to differentiate between possible θ values.

Log-Likelihood Function ($\ell = \ln(L)$): Used in practice as it gives reasonably-sized numbers and simplifies math.

eg. Flip a coin $N = 100$ times. Let $\mathcal{D} = \{x_1, \dots, x_N\}$, where $x_i \in \{0,1\}$ $(1 = \text{Heads}, 0 = \text{Tails})$

Since coin flips are i.i.d., we model them as $X \sim \text{Bernoulli}(\theta)$ where $\theta \in [0,1]$ is the chance of heads.

$$L(\theta) = p_{X|\theta}(\mathcal{D}, \theta) = \prod_{n=1}^{N} \begin{cases} \theta & x_n = 1 \\ 1-\theta & x_n = 0 \end{cases} = \prod_{n=1}^{N} \theta^{x_n}(1-\theta)^{1-x_n}$$

To make the math easier, we use log-likelihood:

$$\ell(\theta) = \ln \prod_{n=1}^{N} \theta^{x_n}(1-\theta)^{1-x_n} = \sum_{n=1}^{N} [x_n \ln\theta + (1-x_n)\ln(1-\theta)]$$

Note the similarity to binary cross-entropy!

$$\mathcal{L}(y(\vec{x}), t) = -t \ln y(\vec{x}) - (1-t)\ln\big(1 - y(\vec{x})\big)$$

Maximum Likelihood Criterion: Maximize L (set $\nabla\ell = 0$) and plug in observed data $\mathcal{D}$ to find ℓ's parameters.

- As $\ln x$ is convex, $\text{argmax}(L) = \text{argmax}(\ell)$, and ℓ is easier to work with. Solve directly or gradient descent!

eg. Flip a coin $N = 100$ times. Suppose we observe $N_\text{H} = 55$, $N_\text{T} = 45$. What are the odds of heads/tails?

The likelihood function is the probability mass function of the number of heads/tails after $N = 100$ coin flips.

$$\ell(\theta) = \ln \prod_{n=1}^{N} \theta^{x_n}(1-\theta)^{1-x_n} = \sum_{n=1}^{N} [x_n \ln\theta + (1-x_n)\ln(1-\theta)]$$

Maximum likelihood criterion says to obtain the most accurate θ, maximize $\ell(\theta)$.

$$\begin{aligned} \frac{d\ell}{d\theta} &= \frac{d}{d\theta}\left(\sum_{n=1}^{N} [x_n \ln\theta + (1-x_n)\ln(1-\theta)]\right) \\ &= \sum_{n=1}^{N}\left(\frac{x_n}{\theta} + \frac{1-x_n}{1-\theta}\right) \\ &= \sum_{n=1}^{N}\frac{x_n}{\theta} + \sum_{n=1}^{N}\frac{1-x_n}{1-\theta} \\ &= \frac{N_\text{H}}{\theta} - \frac{N_\text{T}}{1-\theta} = 0 \end{aligned}$$

$$\therefore \theta^* = \frac{N_\text{H}}{N_\text{H} + N_\text{T}} = 0.55$$

eg. Suppose we get a temperature distribution $\mathcal{D} = \{-2.5, -9.9, -12.1, -8.9, -6.0, -4.8, 2.4\} = \{x_1, \dots, x_N\}$ where $N = 7$. Choose $X \sim \text{Normal}(\mu, \sigma^2)$ for simplicity. Need to find μ and σ.

$$\begin{aligned} \ell(\mu, \sigma^2) &= \ln p_{X|\mu,\sigma^2}(\mathcal{D}|\mu, \sigma^2) \\ &= \ln \prod_{n=1}^{N}\left[\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x_n-\mu)^2}{2\sigma^2}}\right] \\ &= \sum_{n=1}^{N} \ln\left[\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x_n-\mu)^2}{2\sigma^2}}\right] \\ &= \sum_{n=1}^{N}\left[\ln\left(\frac{1}{\sigma\sqrt{2\pi}}\right) + \ln\left(e^{-\frac{(x_n-\mu)^2}{2\sigma^2}}\right)\right] \\ &= -\sum_{n=1}^{N}\left[\ln(\sigma) + \ln\sqrt{2\pi} + \frac{(x_n-\mu)^2}{2\sigma^2}\right] \end{aligned}$$

To find optimal σ, μ values, we differentiate with respect to both variables and set them to 0.

$$\begin{aligned} \frac{d\ell}{d\mu} &= -\sum_{n=1}^{N}\left[\frac{2(x_n-\mu)(-1)}{2\sigma^2}\right] \\ &= \frac{1}{\sigma^2}\sum_{n=1}^{N}(x_n - \mu) \\ &= \frac{1}{\sigma^2}\left[\sum_{n=1}^{N}(x_n) - N\mu\right] = 0 \end{aligned}$$

Therefore, set $\mu = \frac{1}{N}\sum_{n=1}^{N} x_n = -5.97$.

$$\begin{aligned} \frac{d\ell}{d\sigma} &= -\sum_{n=1}^{N}\left[\frac{1}{\sigma} - \frac{(x_n-\mu)^2}{\sigma^3}\right] \\ &= -\sum_{n=1}^{N}\left[\frac{\sigma^2 - (x_n-\mu)^2}{\sigma^3}\right] = 0 \end{aligned}$$

Therefore, set $N\sigma^2 = \sum_{n=1}^{N}(x_n - \mu)^2$, so $\sigma = 4.55$.

Discriminative: A classification approach based on finding targets directly via training set data.

- Try to "learn" $p(t|\vec{x})$ directly, finds a mapping $y(\vec{x}) \approx t$. Includes every model we've covered so far.
- $p(t|\vec{x})$ is informal, reads as "the probability that an input's target is t, given the input is $\vec{x}$"

Bayesian/Generative: A classification approach based on modelling the distribution of training set data in a target.

- Try to "learn" $p(\vec{x}|t)$, calculate $p(t|\vec{x})$ via Bayes Rule
- $p(\vec{x}|t)$ is informal, reads as "the probability that the input looks like $\vec{x}$, given its target is t"

Naïve Bayes: Generative classifier that assumes all x_i in $\vec{x}$ are conditionally independent given t (ie. covariance is 0)

eg. An email spam classifier, with N emails in training set, where $t \in \{0,1\}$, $\vec{x} \in \{0,1\}^D$, and $x_d \in \{0,1\}$ represents whether the d-th word (of D total words from all emails) appears inside an email.

Finding log-likelihood,

$$\begin{aligned} L(\theta) &= p_{X|\theta}(\mathcal{D}|\theta) \\ &= \prod_{n=1}^{N} p_{X|\theta}((x_n)_1, \dots (x_n)_D, t_n|\theta) \\ \ell(\theta) &= \sum_{n=1}^{N} \ln p(\overrightarrow{x_n}, t_n) \end{aligned}$$

For now, I omitted θ for visual simplicity.

How do we expand $p(\overrightarrow{x_n}, t_n)$?

$$\begin{aligned} p(\overrightarrow{x_n}, t_n) = p(t_n) &\times p((x_n)_1|t_n) \\ &\times p((x_n)_2|(x_n)_1, t_n) \\ &\times p((x_n)_3|(x_n)_1, (x_n)_2, t_n) \\ &\times \dots \\ &\times p((x_n)_D|(x_n)_1, \dots, (x_n)_{D-1}, t_n) \end{aligned}$$

This is computationally infeasible, so in Naïve Bayes, we assume

$$p(\overrightarrow{x_n}, t_n) = p(t_n) \times \prod_{d=1}^{D} p((x_n)_d|t_n)$$

That is, $p(\text{word } i \text{ appears}|t_n)$ doesn't affect $p(\text{word } j \text{ appears}|t_n)$.

Although the Naïve Bayes assumption is terrible, it simplifies our math.

$$\begin{aligned} \ell(\theta) &= \sum_{n=1}^{N} \ln p(\overrightarrow{x_n}, t_n) \\ &= \sum_{n=1}^{N} \ln[p(t_n) \cdot p(\overrightarrow{x_n}|t_n)] \\ &= \sum_{n=1}^{N} \ln[p(t_n) \cdot p((x_n)_1|t_n) \cdot \dots \cdot p((x_n)_D|t_n)] \end{aligned}$$

$$\begin{aligned} &= \sum_{n=1}^{N} \ln\left[p(t_n) \cdot \prod_{d=1}^{D} p((x_n)_d|t_n)\right] \\ &= \sum_{n=1}^{N} \left(\ln p(t_n) + \sum_{d=1}^{D} \ln p((x_n)_d|t_n)\right) \\ &= \sum_{n=1}^{N} \ln p(t_n) + \sum_{d=1}^{D}\sum_{n=1}^{N} \ln p((x_n)_d|t_n) \end{aligned}$$

Optimizing Left Term

Let's define $\phi = p(t_n = 1)$.

Recall that $t_n \in \{0,1\}$,

$$\begin{aligned} p(t_n) &= \begin{cases} \phi & t_n = 1 \\ 1-\phi & t_n = 0 \end{cases} \\ &= \phi^{t_n}(1-\phi)^{1-t_n} \end{aligned}$$

$$\begin{aligned} \therefore \ell_1(\phi) &= \sum_{n=1}^{N} \ln p(t_n) \\ &= \sum_{n=1}^{N} \ln[\phi^{t_n}(1-\phi)^{1-t_n}] \\ &= \sum_{n=1}^{N} [t_n \ln\phi + (1-t_n)\ln(1-\phi)] \end{aligned}$$

$$\begin{aligned} \frac{d\ell_1}{d\phi} &= \sum_{n=1}^{N} \frac{t_n}{\phi} + \sum_{n=1}^{N} \frac{1-t_n}{1-\phi} \\ &= \frac{N_{\text{spam}}}{\phi} + \frac{N_{\text{not spam}}}{1-\phi} = 0 \\ \therefore \phi^* &= \frac{N_{\text{spam}}}{N_{\text{spam}} + N_{\text{not spam}}} \end{aligned}$$

That is, the percentage of spam emails.

Optimizing Right Term

Let's define $\phi_k = p((x_n)_d = 1|t_n = k)$ where $k \in \{0,1\}$.

$$\begin{aligned} p((x_n)_d|t_n) &= \begin{cases} \phi_k & (x_n)_d = 1 \\ 1-\phi_k & (x_n)_d = 0 \end{cases} \\ &= \phi_k^{(x_n)_d}(1-\phi_k)^{1-(x_n)_d} \end{aligned}$$

$$\begin{aligned} \therefore \ell_2(\phi_k) &= \ln \prod_{n=1}^{N} p((x_n)_d|t_n) \\ &= \sum_{n=1}^{N} [(x_n)_d \ln\phi_k + (1-(x_n)_d)\ln(1-\phi_k)] \\ &= \sum_{n=1}^{N} (1-t_n)[(x_n)_d \ln\phi_0 + (1-(x_n)_d)\ln(1-\phi_0)] + \sum_{n=1}^{N} t_n[(x_n)_d \ln\phi_1 + (1-(x_n)_d)\ln(1-\phi_1)] \end{aligned}$$

Differentiating with respect to ϕ_0 and ϕ_1,

$$\frac{d\ell_2}{d\phi_0} = \sum_{n=1}^{N}(1-t_n)\left[\frac{(x_n)_d}{\phi_0} - \frac{1-(x_n)_d}{1-\phi_0}\right] = 0$$
$$\sum_{n=1}^{N}(1-t_n)[(x_n)_d(1-\phi_0) - (1-(x_n)_d)\phi_0] = 0$$
$$\sum_{n=1}^{N}(1-t_n)[(x_n)_d - \phi_0] = 0$$
$$\sum_{n=1}^{N}(1-t_n)(x_n)_d = \phi_0 \sum_{n=1}^{N}(1-t_n)$$
$$\phi_0^* = \frac{\sum_{n=1}^{N}(1-t_n)(x_n)_d}{\sum_{n=1}^{N}(1-t_n)} = \frac{N_{\text{dth word,no spam}}}{N_{\text{not spam}}}$$

"The percentage of non-spams containing the d-th word"

$$\frac{d\ell_2}{d\phi_1} = \sum_{n=1}^{N}t_n\left[\frac{(x_n)_d}{\phi_1} - \frac{1-(x_n)_d}{1-\phi_1}\right] = 0$$
$$\sum_{n=1}^{N}t_n[(x_n)_d(1-\phi_1) - (1-(x_n)_d)\phi_1] = 0$$
$$\sum_{n=1}^{N}t_n[(x_n)_d - \phi_1] = 0$$
$$\sum_{n=1}^{N}t_n(x_n)_d = \phi_1 \sum_{n=1}^{N}t_n$$
$$\phi_1^* = \frac{\sum_{n=1}^{N}t_n(x_n)_d}{\sum_{n=1}^{N}t_n} = \frac{N_{\text{dth word,spam}}}{N_{\text{spam}}}$$

"The percentage of spams containing the d-th word"

Now that we know optimal $p(t_n)$ and $p((x_n)_d|t_n)$ values, at test time, we apply Bayes' Rule to find $p(t|\vec{x})$.

$$p(t=k|\vec{x}) = \frac{p(t=k)p(\vec{x}|t=k)}{p(\vec{x})} = \frac{p(t=k)p(\vec{x}|t=k)}{\sum_{k'=0}^{1}p(t=k')p(\vec{x}|t=k')} = \frac{p(t=k)\prod_{d=1}^{D}p(x_d|t=k)}{\sum_{k'=0}^{1}p(t=k')\prod_{d=1}^{D}p(x_d|t=k')}$$

Recall that the ideal $p(t=k)$ is $\frac{N_{\text{spam}}}{N_{\text{spam}}+N_{\text{not spam}}}$, and the ideal $p(x_d=1|t=k)$ is $\frac{N_{\text{dth word,spam/no spam}}}{N_{\text{spam/no spam}}}$.

Predict the $t \in \{0,1\}$ that results in the highest $p(t|\vec{x})$; the denominator is constant and can be omitted.

For computational speed, use the $x = e^{\ln x}$ rule to turn the product into a summation: $p(t=k)e^{\sum_{d=1}^{D}\ln p(x_d|t=k)}$

- Works with any probability distribution
- **Efficient** – 1 pass of data at train-time, $\mathcal{O}(D)$ at test-time – but **inaccurate**, due to the independence assumption

Data Sparsity: Lack of data. Causes log to fail, overfitting in maximum likelihood. Solve with prior distributions.

- eg. If $p(x_d|t=k)=0$ in our dataset, then $\ln p(x_d|t=k) = -\infty$.
- eg. Flip a coin twice, get H twice, then maximum likelihood says $\theta = \frac{2}{2+0} = 1$

Prior Distribution ($p(\theta)$)	**Likelihood ($L(\theta) = p(\mathcal{D}\|\theta)$)**	**Posterior Distribution ($p(\theta\|\mathcal{D})$)**
Our beliefs about parameters before observing data. **Uninformative Prior:** A prior distribution that assumes the least possible. • Usually, it's something computationally convenient *(uniform distribution, Bernoulli)*	The likelihood of observing a given configuration of data, given parameters.	Our updated beliefs about parameters after observing a configuration of data. $p(\theta\|\mathcal{D}) = \frac{p(\theta)p(\mathcal{D}\|\theta)}{\int_{-\infty}^{\infty}p(\theta')p(\mathcal{D}\|\theta')\,d\theta'}$ $\propto p(\theta)p(\mathcal{D}\|\theta)$ We rarely compute the denominator.

Maximum A-Posteriori (MAP) Estimation: Maximizing the posterior *(rather than the likelihood in maximum likelihood)*. Usually apply log first: it simplifies computations and doesn't affect the argmax (as $\ln x$ is monotonic).

$$\underset{\theta\in[0,1]}{\operatorname{argmax}}\, p(\theta|\mathcal{D}) = \underset{\theta\in[0,1]}{\operatorname{argmax}}\, \ln p(\theta|\mathcal{D})$$

Normalization: Multiplying a $f(x)$ by a constant to make it a probability density function (ie. $\int_{-\infty}^{\infty}f(x)\,dx = 1$).

Beta Distribution: Continuous distribution $X \sim \text{Beta}(a,b)$ common in machine learning. See this for visualizations.

$$f_X(x) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}x^{a-1}(1-x)^{b-1} \quad \text{for } x \in [0,1]$$
$$\propto x^{a-1}(1-x)^{b-1}$$

$\Gamma(x) = \int_0^{\infty}t^{x-1}e^{-t}dt = (x-1)!$ is the normalizer (ignore it)

- $f_X(x)$ is only defined on $x \in [0,1]$, 0 elsewhere
- $\mathbb{E}[X] = \frac{a}{a+b}$
- $a = b = 1$ is the uniform distribution
- High values of a, b mean larger, narrower peaks.

eg. Flip a coin. We don't know how biased the coin is, so θ can be anything from 0 to 1.

Let the prior $p(\theta)$ follow the beta distribution,

$$\theta \sim \text{Beta}(a, b)$$
$$p(\theta) \propto \theta^{a-1}(1-\theta)^{b-1}$$

From before, we know the likelihood $L(\theta)$ is

$$L(\theta) = p(\mathcal{D}|\theta) = \prod_{n=1}^{N} \theta^{x_n}(1-\theta)^{1-x_n} = \theta^{N_\text{H}}(1-\theta)^{N_\text{T}}$$

Therefore, calculating the posterior,

$$\begin{aligned} \therefore p(\theta|\mathcal{D}) &\propto p(\theta) f(\mathcal{D}|\theta) \\ &\propto [\theta^{a-1}(1-\theta)^{b-1}][\theta^{N_\text{H}}(1-\theta)^{N_\text{T}}] \\ &= \theta^{N_\text{H}+a-1}(1-\theta)^{N_\text{T}+b-1} \end{aligned}$$

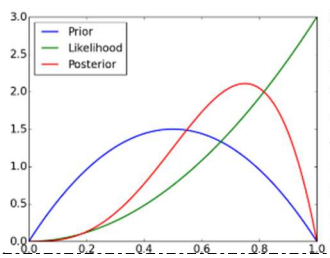
If you **normalize** this, this is equivalent to $\theta|\mathcal{D} \sim \text{Beta}(N_\text{H}+a, N_\text{T}+b)$.

$$\therefore \mathbb{E}[\theta|\mathcal{D}] = \frac{N_\text{H}+a}{N_\text{H}+N_\text{T}+a+b}$$

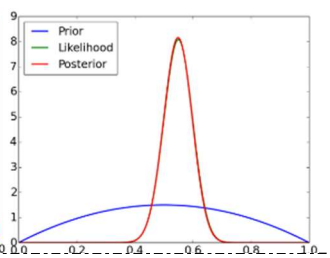
Pseudo-Count: Numbers added to observations as to change a generated probability. In this case, it's a, b

Conjugates: Expressions represented by the same functional form. In this case, it's $p(\theta)$ and $p(\theta|\mathcal{D})$.

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



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



Finding the MAP estimation of θ,

$$\begin{aligned} \frac{d}{d\theta}\ln p(\theta|\mathcal{D}) &= \frac{d}{d\theta}\ln[\theta^{N_\text{H}+a-1}(1-\theta)^{N_\text{T}+b-1}] \\ &= \frac{d}{d\theta}[(N_\text{H}+a-1)\ln\theta \\ &\quad + (N_\text{T}+b-1)\ln(1-\theta)] \\ &= \frac{N_\text{H}+a-1}{\theta} - \frac{N_\text{T}+b-1}{1-\theta} = 0 \end{aligned}$$

$$\therefore \theta = \frac{N_\text{H}+a-1}{N_\text{H}+N_\text{T}+a+b-2}$$

Multivariate Random Variables	*Linear Algebra Formulas*
$\mathbb{E}[X] = \begin{bmatrix} E[X_1] \\ \vdots \\ E[X_d] \end{bmatrix}$ $\text{Var}[X] = \mathbb{E}[(X-\mathbb{E}[X])(X-\mathbb{E}[X])^\text{T}]$ $= \mathbb{E}[XX^\text{T}] - \mathbb{E}[X]\mathbb{E}[X]^\text{T}$ $\text{Cov}[X,Y] = \mathbb{E}[(X-\mathbb{E}[X])(Y-\mathbb{E}[Y])^\text{T}]$ $= \mathbb{E}[XY^\text{T}] - \mathbb{E}[X]\mathbb{E}[Y]^\text{T}$	$\frac{\partial}{\partial\vec{x}}[\vec{x}^\text{T}A^{-1}\vec{x}] = 2A^{-1}\vec{x}$ $\frac{\partial}{\partial A}[\vec{x}^\text{T}A^{-1}\vec{x}] = -A^{-1}\vec{x}\vec{x}^\text{T}A^{-1}$ $\frac{\partial \det(A)}{\partial A} = \det(A)\,(A^{-1})^\text{T}$

Normal/Gaussian Distribution: A continuous distribution. Popular in machine learning for simple calculations

- **Central Limit Theorem:** The sum of many independent random variables is roughly Gaussian

Univariate	*Multivariate*
$X \sim \text{Normal}(\mu, \sigma^2)$	$X \sim \text{Normal}(\vec{\mu}, \Sigma)$
$\mu = \mathbb{E}[x]$ $\sigma^2 = \text{Var}[x]$	$\vec{\mu} = \mathbb{E}[\vec{x}] = \begin{bmatrix} \mu_1 \\ \vdots \\ \mu_d \end{bmatrix}$ $\Sigma = \text{Cov}(\vec{x}, \vec{x}) = \mathbb{E}[(\vec{x}-\vec{\mu})(\vec{x}-\vec{\mu})^\text{T}] = \begin{bmatrix} \sigma_{1,1}^2 & \sigma_{1,2} & \cdots & \sigma_{1,d} \\ \sigma_{2,1} & \sigma_{2,2}^2 & \cdots & \sigma_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{d,1} & \sigma_{d,2} & \cdots & \sigma_{d,d}^2 \end{bmatrix}$
$f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$	$f_X(\vec{x}) = \frac{1}{\det(\Sigma)^{\frac{1}{2}}(2\pi)^{\frac{d}{2}}} e^{-\frac{1}{2}(\vec{x}-\vec{\mu})^\text{T}\Sigma^{-1}(\vec{x}-\vec{\mu})}$
Translate $N(0,1)$ by μ units right, stretch by factor of σ to get $N(\mu, \sigma^2)$	Translate $N(0, I)$ in direction $\vec{\mu}$, scale by $\Sigma^{\frac{1}{2}}$ (ie. $\Sigma^{\frac{1}{2}}\Sigma^{\frac{1}{2}} = \Sigma$) to get $N(\vec{\mu}, \Sigma)$

- Σ is a symmetric and positive semi-definite matrix *(see next section)*
- $\overrightarrow{x_n}$ is uncorrelated with $\overrightarrow{x_m}$ if covariance matrix Σ is diagonal (ie. all non-diagonal values are 0)
  - Gaussian Naïve Bayes assumes that Σ is diagonal

eg. Let $\mathcal{D} = \{(-2.5,-7.5),(-9.9,-14.9),(-12.1,-17.5),(-8.9,-13.9),(-6.0,-11.1)\} = \{x_1,\dots,x_N\}$ where $N=5$, a temperature distribution of highs/lows. Choose X~Normal$(\vec{\mu},\Sigma)$ for simplicity. Find $\vec{\mu},\Sigma$.

$$\begin{aligned}
\ell(\vec{\mu},\Sigma) &= \ln \prod_{n=1}^{N} p_{X|\vec{\mu},\Sigma}(\mathcal{D}|\vec{\mu},\Sigma) \\
&= \ln \prod_{n=1}^{N} \left[\frac{1}{\det(\Sigma)^{\frac{1}{2}}(2\pi)^{\frac{d}{2}}} e^{-\frac{1}{2}(\overrightarrow{x_n}-\vec{\mu})^{\mathrm{T}}\Sigma^{-1}(\overrightarrow{x_n}-\vec{\mu})}\right] \\
&= \sum_{n=1}^{N} \ln \left[\frac{1}{\det(\Sigma)^{\frac{1}{2}}(2\pi)^{\frac{d}{2}}} e^{-\frac{1}{2}(\overrightarrow{x_n}-\vec{\mu})^{\mathrm{T}}\Sigma^{-1}(\overrightarrow{x_n}-\vec{\mu})}\right] \\
&= \sum_{n=1}^{N} \left[-\ln\left(\det(\Sigma)^{\frac{1}{2}}(2\pi)^{\frac{d}{2}}\right) + \ln e^{-\frac{1}{2}(\overrightarrow{x_n}-\vec{\mu})^{\mathrm{T}}\Sigma^{-1}(\overrightarrow{x_n}-\vec{\mu})}\right] \\
&= \sum_{n=1}^{N} \left[-\ln\det(\Sigma)^{\frac{1}{2}} - \ln(2\pi)^{\frac{d}{2}} - \frac{1}{2}(\overrightarrow{x_n}-\vec{\mu})^{\mathrm{T}}\Sigma^{-1}(\overrightarrow{x_n}-\vec{\mu})\right] \\
&= -\frac{1}{2}\sum_{n=1}^{N}[\ln\det(\Sigma) + d\ln 2\pi + (\overrightarrow{x_n}-\vec{\mu})^{\mathrm{T}}\Sigma^{-1}(\overrightarrow{x_n}-\vec{\mu})]
\end{aligned}$$

Taking the derivative with respect to $\vec{\mu}$,

$$\begin{aligned}
\frac{\partial \ell}{\partial \vec{\mu}} &= -\frac{1}{2}\sum_{n=1}^{N}\frac{\partial}{\partial\vec{\mu}}[(\overrightarrow{x_n}-\vec{\mu})^{\mathrm{T}}\Sigma^{-1}(\overrightarrow{x_n}-\vec{\mu})] \\
&= -\sum_{n=1}^{N}\Sigma^{-1}(\overrightarrow{x_n}-\vec{\mu}) = 0
\end{aligned}$$

$$\boxed{\therefore \vec{\mu} = \frac{1}{N}\sum_{n=1}^{N}\overrightarrow{x_n}}$$

$$\approx (-1.576, -2.596)$$

This formula is called **empirical mean**, where mean is based off past results rather than a theoretical distribution. The formula below is **empirical covariance**.

Taking the derivative with respect to Σ,

$$\begin{aligned}
\frac{\partial \ell}{\partial \Sigma} &= -\frac{1}{2}\sum_{n=1}^{N}\frac{\partial}{\partial\Sigma}[\ln\det(\Sigma) + (\overrightarrow{x_n}-\vec{\mu})^{\mathrm{T}}\Sigma^{-1}(\overrightarrow{x_n}-\vec{\mu})] \\
&= -\frac{1}{2}\sum_{n=1}^{N}\left[\frac{\partial\ln\det(\Sigma)}{\partial\det(\Sigma)}\cdot\frac{\partial\det(\Sigma)}{\partial\Sigma} + \frac{\partial}{\partial\Sigma}(\overrightarrow{x_n}-\vec{\mu})^{\mathrm{T}}\Sigma^{-1}(\overrightarrow{x_n}-\vec{\mu})\right] \\
&= -\frac{1}{2}\sum_{n=1}^{N}\left[\frac{1}{\det(\Sigma)}\cdot\det(\Sigma)\,(\Sigma^{-1})^{\mathrm{T}} - \Sigma^{-1}(\overrightarrow{x_n}-\vec{\mu})(\overrightarrow{x_n}-\vec{\mu})^{\mathrm{T}}\Sigma^{-1}\right] \\
&= -\frac{1}{2}\sum_{n=1}^{N}[\Sigma^{-1} - \Sigma^{-1}(\overrightarrow{x_n}-\vec{\mu})(\overrightarrow{x_n}-\vec{\mu})^{\mathrm{T}}\Sigma^{-1}] \\
&= -\frac{1}{2}\sum_{n=1}^{N}\Sigma^{-1}[1 - (\overrightarrow{x_n}-\vec{\mu})(\overrightarrow{x_n}-\vec{\mu})^{\mathrm{T}}\Sigma^{-1}] = 0 \\
&\sum_{n=1}^{N}[1 - (\overrightarrow{x_n}-\vec{\mu})(\overrightarrow{x_n}-\vec{\mu})^{\mathrm{T}}\Sigma^{-1}] = 0 \\
&\sum_{n=1}^{N}[1 - (\overrightarrow{x_n}-\vec{\mu})(\overrightarrow{x_n}-\vec{\mu})^{\mathrm{T}}\Sigma^{-1}]\Sigma = 0 \\
&\sum_{n=1}^{N}[\Sigma - (\overrightarrow{x_n}-\vec{\mu})(\overrightarrow{x_n}-\vec{\mu})^{\mathrm{T}}] = 0
\end{aligned}$$

$$\boxed{\therefore \Sigma = \frac{1}{N}\sum_{n=1}^{N}(\overrightarrow{x_n}-\vec{\mu})(\overrightarrow{x_n}-\vec{\mu})^{\mathrm{T}}}$$

Linear regression is just $t|\vec{x}$~Normal$(\vec{w}\cdot\vec{x},1)$ with some extra constants (which don't affect argmax)

$$\begin{aligned}
\ln p(t|\vec{x}) &= \ln\prod_{n=1}^{N}\left(\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(t_n-\vec{w}\cdot\vec{x})^2}{2(1)^2}}\right) \\
&= -\sum_{n=1}^{N}\left(\ln\sigma + \frac{1}{2}\ln 2\pi + \frac{(\vec{w}\cdot\vec{x}-t_n)^2}{2}\right) \\
&= \text{Constant} - \frac{1}{2}\sum_{n=1}^{N}(\vec{w}\cdot\vec{x}-t_n)^2
\end{aligned}$$

$\boldsymbol{L_2}$ **regularization** is just $\vec{w}$~Normal$(0,\eta I)$ with extra constants (where $\eta\in\mathbb{R}$ is like $\frac{1}{\lambda}$ from ridge regression)

$$\begin{aligned}
\ln p(\vec{w}) &= \ln\left(\frac{1}{\det(\eta I)^{\frac{1}{2}}(2\pi)^{\frac{d}{2}}}e^{-\frac{1}{2}(\vec{w}-0)^{\mathrm{T}}(\eta I)^{-1}(\vec{w}-0)}\right) \\
&= -\frac{1}{2}\ln(\det\eta I) - \frac{d}{2}\ln 2\pi - \frac{1}{2}\vec{w}^{\mathrm{T}}(\eta I)^{-1}\vec{w} \\
&= \text{Constant} - \frac{1}{2\eta}\|\vec{w}\|^2
\end{aligned}$$

- Finding the argmax of $-x$ (here) is equivalent to finding the argmin of x (linear regression)

Gaussian Discriminant Analysis: Classification algorithm that finds the best Gaussian fit to data $p(\vec{x}|t)$

$$p(\vec{x}|t) = f_X(\vec{x})$$

Each target has a separate Σ_k and $\overrightarrow{\mu_k}$, which are the parameters to find using maximum likelihood.

eg. Multivariate Gaussian discriminant analysis for binary classification. $t \in \{0,1\}$

Let ϕ_k be percentage of inputs with target $k \in \{0,1\}$.

$$\phi_k = \frac{1}{N}\sum_{n=1}^{N} \mathbb{I}(t_n = k)$$

Define $\overrightarrow{\mu_k}$, Σ_k as empirical mean/covariance of all inputs with target k. Obtainable via max-likelihood.

$$\overrightarrow{\mu_k} = \frac{\sum_{n=1}^{N} \mathbb{I}(t_n = k)\overrightarrow{x_n}}{\sum_{n=1}^{N} \mathbb{I}(t_n = k)}$$

$$\Sigma_k = \frac{\sum_{n=1}^{N} \mathbb{I}(t_n = k)(\overrightarrow{x_n} - \overrightarrow{\mu_k})(\overrightarrow{x_n} - \overrightarrow{\mu_k})^{\mathrm{T}}}{\sum_{n=1}^{N} \mathbb{I}(t_n = k)}$$

At test-time, for any new query point,

$$\begin{aligned} p(t = k) &= \begin{cases} \phi_1 & k = 1 \\ 1 - \phi_1 & k = 0 \end{cases} \\ &= \begin{cases} \phi_0 & k = 0 \\ 1 - \phi_0 & k = 1 \end{cases} \\ &= \phi_k^k (1 - \phi_k)^{1-k} \end{aligned}$$

We also know that

$$p(\vec{x}|t) = f_X(x) = \frac{1}{\det(\Sigma)^{\frac{1}{2}}\,(2\pi)^{\frac{d}{2}}} e^{-\frac{1}{2}(\vec{x}-\vec{\mu})^{\mathrm{T}}\Sigma^{-1}(\vec{x}-\vec{\mu})}$$

Using Bayes' rule (and logarithms on both sides),

$$\begin{aligned} \ln p(t = k|\vec{x}) &= \ln \frac{p(t = k)p(\vec{x}|t = k)}{p(\vec{x})} \\ &= \ln p(t = k) + \left[-\frac{1}{2}\ln\det\Sigma_k - \frac{d}{2}\ln 2\pi - \frac{1}{2}(\vec{x} - \overrightarrow{\mu_k})^{\mathrm{T}}\Sigma_k^{-1}(\vec{x} - \overrightarrow{\mu_k}) + k\ln\phi_k + (1-k)\ln(1-\phi_k)\right] - \ln p(\vec{x}) \end{aligned}$$

In probabilistic modelling, the **decision boundary** is the solution to $p(t = 0|\vec{x}) = 0.5 = p(t = 1|\vec{x})$.

$$\ln p(t = 0|\vec{x}) = \ln p(t = 1|\vec{x})$$

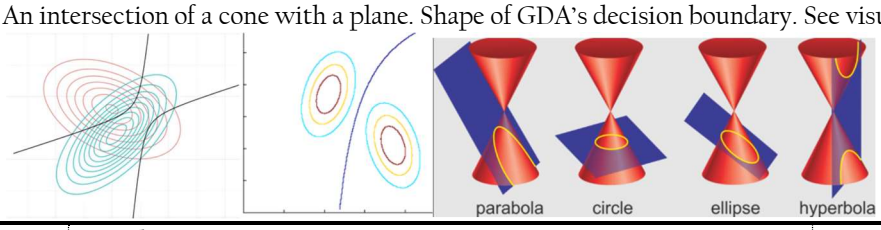
$$-\frac{1}{2}\ln\det\Sigma_0 - \frac{1}{2}(\vec{x} - \overrightarrow{\mu_0})^{\mathrm{T}}\Sigma_0^{-1}(\vec{x} - \overrightarrow{\mu_0}) + \ln(1 - \phi_0) = -\frac{1}{2}\ln\det\Sigma_1 - \frac{1}{2}(\vec{x} - \overrightarrow{\mu_1})^{\mathrm{T}}\Sigma_1^{-1}(\vec{x} - \overrightarrow{\mu_1}) + \ln\phi_1$$

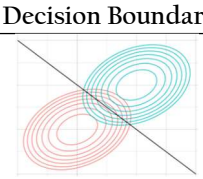
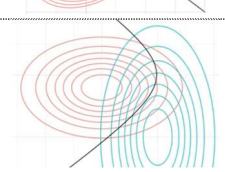
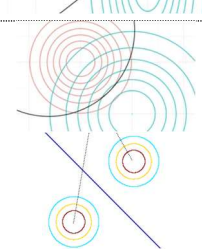
The terms without $\vec{x}$ will be constants, so we simplify the equation to

$$(\vec{x} - \overrightarrow{\mu_0})^{\mathrm{T}}\Sigma_0^{-1}(\vec{x} - \overrightarrow{\mu_0}) = (\vec{x} - \overrightarrow{\mu_1})^{\mathrm{T}}\Sigma_1^{-1}(\vec{x} - \overrightarrow{\mu_1}) + \text{Constant}$$

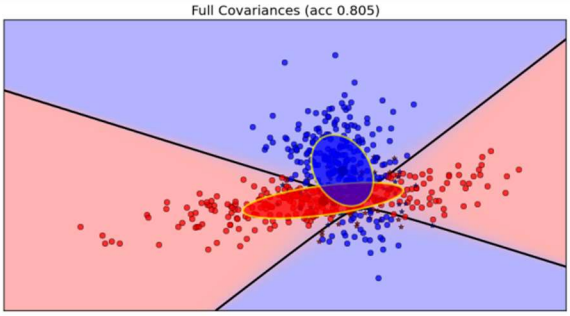
$$\vec{x}^{\mathrm{T}}\Sigma_0^{-1}\vec{x} - 2\overrightarrow{\mu_0}^{\mathrm{T}}\Sigma_0^{-1}\vec{x} = \vec{x}^{\mathrm{T}}\Sigma_1^{-1}\vec{x} - 2\overrightarrow{\mu_1}^{\mathrm{T}}\Sigma_1^{-1}\vec{x} + \text{Constant}$$

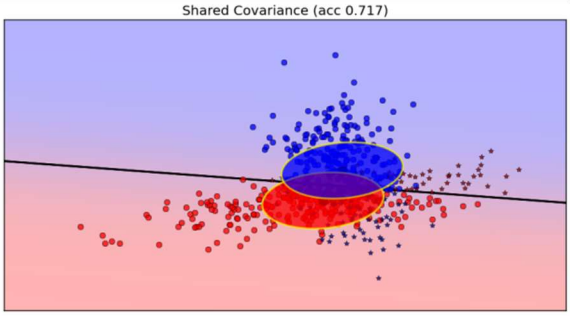
Conic Section: An intersection of a cone with a plane. Shape of GDA's decision boundary. See visualizations here.

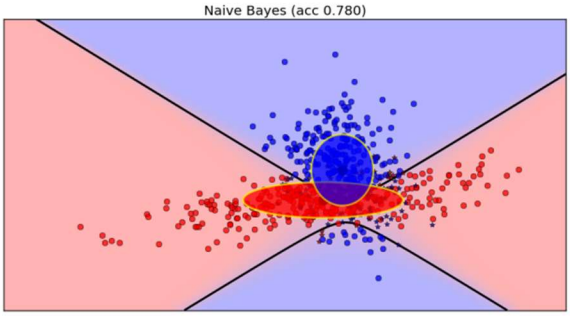


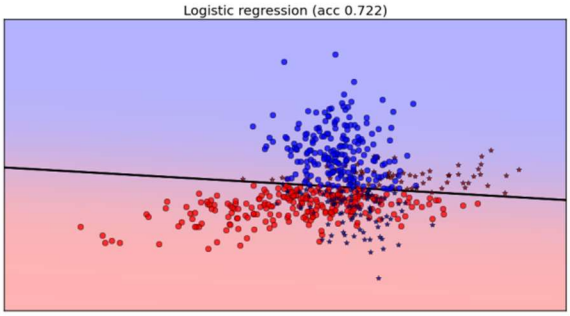
Assumption	Result	Decision Boundary
Shared Covariances (ie. $\Sigma_0 = \Sigma_1 = \Sigma$)	$\vec{x}^{\mathrm{T}}\Sigma^{-1}\vec{x} - 2\overrightarrow{\mu_0}^{\mathrm{T}}\Sigma^{-1}\vec{x} = \vec{x}^{\mathrm{T}}\Sigma^{-1}\vec{x} - 2\overrightarrow{\mu_1}^{\mathrm{T}}\Sigma^{-1}\vec{x} + \text{Constant}$ $\overrightarrow{\mu_0}^{\mathrm{T}}\Sigma^{-1}\vec{x} = \overrightarrow{\mu_1}^{\mathrm{T}}\Sigma^{-1}\vec{x} + \text{Constant}$ $(\overrightarrow{\mu_0} - \overrightarrow{\mu_1})^{\mathrm{T}}\Sigma^{-1}\vec{x} = \text{Constant}$ This is a **linear** decision boundary! Logistic regression!	
Naïve Bayes (ie. Diagonal Σ)	Contours only vary via scaling, cannot be rotated. Still produces **conic section** decision boundaries. For high-dimensional $\vec{x} \in \mathbb{R}^D$, reduces size of Σ_k from D^2K parameters to D parameters.	
Spherical/Isotropic Covariances (ie. $\Sigma = \sigma^2 I$)	Contours will be spheres, and decision boundaries are **spherical**. If spheres are the same size (ie. $\sigma_0 = \sigma_1$), then decision boundary is a **line** bisecting $\overrightarrow{\mu_0}$ and $\overrightarrow{\mu_1}$. $C = (\vec{x} - \overrightarrow{\mu_0})^{\mathrm{T}}\Sigma^{-1}(\vec{x} - \overrightarrow{\mu_0}) - (\vec{x} - \overrightarrow{\mu_1})^{\mathrm{T}}\Sigma^{-1}(\vec{x} - \overrightarrow{\mu_1})$ $= (\vec{x} - \overrightarrow{\mu_0})^{\mathrm{T}}(\sigma^2 I)^{-1}(\vec{x} - \overrightarrow{\mu_0}) - (\vec{x} - \overrightarrow{\mu_1})^{\mathrm{T}}(\sigma^2 I)^{-1}(\vec{x} - \overrightarrow{\mu_1})$ $= \frac{1}{\sigma^2}[\lVert\vec{x} - \overrightarrow{\mu_0}\rVert^2 - \lVert\vec{x} - \overrightarrow{\mu_1}\rVert^2]$	

Gaussian Discriminant Analysis, Shared Covariances	*Logistic Regression*
➢ Stronger modelling assumption that class-conditional data is multivariate Gaussian o If true, GDA is the most efficient model o If not, then poorer prediction accuracy ➢ Handles missing features better	➢ Usually better than GDA for non-Gaussian distributions (ie. vast majority of distributions)









Principal Component Analysis

Field: A set $\mathbb{F}$ where the addition, subtraction, multiplication, and division of any elements of $\mathbb{F}$ are in $\mathbb{F}$ (eg. $\mathbb{R}, \mathbb{N}, \mathbb{C}$)

Vector Space: Over a field $\mathbb{F}$, the set V defined by two operations, $\vec{u}, \vec{v} \in V, \alpha \in \mathbb{F} \Rightarrow \vec{u} + \vec{v} \in V, \alpha\vec{v} \in V$, where:

- $(\alpha + \beta)\vec{v} = \alpha\vec{v} + \beta\vec{v}$
- $(\alpha\beta)\vec{v} = \alpha(\beta\vec{v})$
- $(\vec{u} + \vec{v}) + \vec{w} = \vec{u} + (\vec{v} + \vec{w})$
- $1\vec{v} = \vec{v}$
- $\vec{0} \in V$
- $-\vec{v} \in V$

Affine Space: Informally, a vector space without any concept of origin.

Projection: Of $\vec{x} \in \mathbb{R}^D$ on subspace $\mathcal{S}$ (K-dimensional, in $\mathbb{R}^D$, where $K < D$), the nearest point of $\mathcal{S}$ to $\vec{x}$.

- For $\mathcal{S} = \text{span}(\vec{u})$, $\quad \text{proj}_{\mathcal{S}}(\vec{x}) = \text{vcomp}_{\vec{u}}(\vec{x}) = (\vec{x} \cdot \vec{u})\vec{u}$
- For $\mathcal{S} = \text{span}(\overrightarrow{u_1}, \dots, \overrightarrow{u_K})$, $\quad \text{proj}_{\mathcal{S}}(\vec{x}) = \sum_{k=1}^{K} \text{vcomp}_{\overrightarrow{u_k}}(\vec{x}) = \sum_{k=1}^{K} (\vec{x} \cdot \overrightarrow{u_k})\overrightarrow{u_k}$
  - In vector form, we have $\quad \text{proj}_{\mathcal{S}}(\vec{x}) = UU^{\mathrm{T}}\vec{x}$ $\quad$ where $U = [\overrightarrow{u_1} \quad \cdots \quad \overrightarrow{u_K}]$ is $D \times K$
- For $\mathcal{S} =$ an affine space, $\quad \text{proj}_{\mathcal{S}}(\vec{x}) = \sum_{k=1}^{K} ((\vec{x} - \vec{\mu}) \cdot \overrightarrow{u_k})\overrightarrow{u_k} + \vec{\mu}$ $\quad$ where $\vec{\mu} \in \mathcal{S}$ is an arbitrary origin
  - In vector form, we have $\quad \text{proj}_{\mathcal{S}}(\vec{x}) = UU^{\mathrm{T}}(\vec{x} - \vec{\mu}) + \vec{\mu}$ $\quad$ where $U = [\overrightarrow{u_1} \quad \cdots \quad \overrightarrow{u_K}]$ is $D \times K$

Reconstruction: The term $\underline{\vec{x}} = \text{proj}_{\mathcal{S}}(\vec{x}) \in \mathcal{S}$

- Note that $\vec{x}$ and $\underline{\vec{x}}$ have the same means

Representation/Code: A $K \times 1$ vector $\vec{z}$ representing a mapping of data to a space that's easier to manipulate/visualize

- In the above case, $\vec{z} = U^{\mathrm{T}}\vec{x}$ for vector spaces, $\vec{z} = U^{\mathrm{T}}(\vec{x} - \vec{\mu})$ for affine spaces
- If $K \ll D$, it's computationally cheaper to work with the representation than $\vec{x}$

Representation Learning: When a learning algorithm tries to learn a representation

Dimensionality Reduction: Mapping data to a lower-dimensional space

- Saves computation/memory, reduces overfitting, and can allow visualization (if reduced to 3D)

Orthonormal: Two vectors $\vec{u}, \vec{v}$ if both are unit vectors and orthogonal (ie. $\vec{u} \cdot \vec{v} = 0$)

Orthogonal Matrix: Matrix Q if $Q^{\mathrm{T}}Q = QQ^{\mathrm{T}} = I$, AKA $Q^{\mathrm{T}} = Q^{-1}$

- Q is orthogonal $\Leftrightarrow$ Q's columns are orthogonal

Eigenvector: Of square matrix A, vector $\vec{x}$ such that $A\vec{x} = \lambda\vec{x}$ for some λ, the corresponding **eigenvalue**.

- Matrices of size $D \times D$ have at most D unique eigenvalues
- Symmetric matrices have D linearly independent eigenvectors $\overrightarrow{x_d}$ (where $\lambda_d \in \mathbb{R}$) which form a basis for $\mathbb{R}^D$ *(ie. their span is $\mathbb{R}^D$)* and can be chosen to be orthonormal (by changing the magnitude of the $\overrightarrow{x_d}$).

Spectral Decomposition: AKA eigendecomposition, a way to factor a symmetric $D \times D$ matrix A

$$A = Q\Lambda Q^{\mathrm{T}} = Q\Lambda Q^{-1}$$

- $Q = [\overrightarrow{q_1} \quad \dots \quad \overrightarrow{q_D}]$ is an orthogonal matrix ($\overrightarrow{q_d}$ are A's eigenvectors)
- $\Lambda = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \lambda_D \end{bmatrix}$ is a diagonal matrix (λ_d are A's eigenvalues)

Let q_d be A's eigenvectors,

$$A\overrightarrow{q_d} = \lambda_d\overrightarrow{q_d}$$

Therefore

$$AQ = Q\Lambda$$
$$A = Q\Lambda Q^{-1}$$

We can convert between $\mathcal{E}$ and eigenvector coordinates Q:

$$[\vec{x}]_{\mathcal{E}} = Q^{\mathrm{T}}[\vec{x}]_Q$$
$$[\vec{x}]_Q = Q[\vec{x}]_{\mathcal{E}}$$

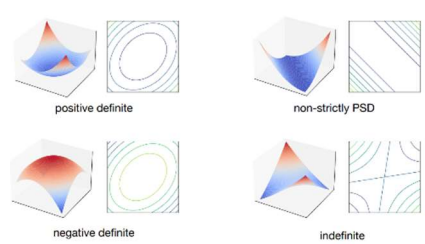
The transformation A rescales individual dimensions by a factor λ_d:

$$[\vec{x}]_{\mathcal{E}} = [x_1]_Q\overrightarrow{q_1} + \dots + [x_D]_Q\overrightarrow{q_D}$$
$$A[\vec{x}]_{\mathcal{E}} = [x_1]_Q A\overrightarrow{q_1} + \dots + [x_D]_Q A\overrightarrow{q_D}$$
$$= [x_1]_Q\lambda_1\overrightarrow{q_1} + \dots + [x_D]_Q\lambda_D\overrightarrow{q_D}$$

All symmetric matrices are diagonal under the right coordinate system.

Quadratic Form: A function $f(\vec{x}) = \vec{x}^{\mathrm{T}}A\vec{x}$ where A is symmetric

- **Positive Definite:** $\quad A \succ 0, \quad$ if $\vec{x}^{\mathrm{T}}A\vec{x} > 0$ for all $\vec{x} \neq \vec{0}$
- **Positive Semi-Definite:** $\quad A \succcurlyeq 0, \quad$ if $\vec{x}^{\mathrm{T}}A\vec{x} \geq 0$ for all $\vec{x} \neq \vec{0}$
- **Negative Semi-Definite:** $\quad A \preccurlyeq 0, \quad$ if $\vec{x}^{\mathrm{T}}A\vec{x} \leq 0$ for all $\vec{x} \neq \vec{0}$
- **Negative Definite:** $\quad A \prec 0, \quad$ if $\vec{x}^{\mathrm{T}}A\vec{x} < 0$ for all $\vec{x} \neq \vec{0}$
- **Indefinite:** $\quad$ If $\vec{x}^{\mathrm{T}}A\vec{x}$ can be positive or negative



Singular/Degenerate: A non-invertible matrix.

Matrix Square Root: Of a positive semi-definite (PSD) matrix A, the matrix $A^{\frac{1}{2}}$ where $A^{\frac{1}{2}}A^{\frac{1}{2}} = A$

- AA^{T} is PSD
- A is PSD $\Rightarrow \mathbb{E}[A]$ is PSD
- A is PD $\Leftrightarrow \lambda_d > 0$
- A is PSD $\Leftrightarrow \lambda_d \geq 0$

For symmetric matrices,

$$A^k = Q\Lambda^k Q^{\mathrm{T}}$$
$$A^{-1} = Q\Lambda^{-1}Q^{\mathrm{T}}$$
$$A^{\frac{1}{2}} = Q\Lambda^{\frac{1}{2}}Q^{\mathrm{T}} \text{ (if } A \text{ is PSD)}$$
$$\det A = \prod_{d=1}^{D} \lambda_d$$

- Non-negative linear combinations of PSD matrices are PSD
- A is PD $\Leftrightarrow$ contour lines of $\vec{x}^{\mathrm{T}}A\vec{x}$ is elliptical
  (principal axes of ellipses aligned with eigenvectors)
- A is PD and diagonal $\Leftrightarrow$ contours lines of $\vec{x}^{\mathrm{T}}A\vec{x}$ is elliptical
  (principle axes of ellipses aligned with axes)

Determinant Properties

$\det AB = \det A \det B$ $\qquad$ $\det A^{\mathrm{T}} = \det A$ (as A is square)

$\det A = 0 \Leftrightarrow A$ is singular $\qquad$ A is orthogonal $\Rightarrow \det A = \pm 1$

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

Frobenius Norm: Norm generalized to a matrix, $\|A\|_F = \sqrt{\sum_{m=1}^{M}\sum_{n=1}^{N} A_{n,m}^2}$

Principal Component Analysis (PCA): Unsupervised learning algorithm that performs linear dimensionality reduction to find the most "important" dimensions (ie. principle components) in the data.

- Set $\vec{\mu} \in \mathbb{R}^D$ to the empirical mean of the $\overrightarrow{x_n}$ to center our data (it simplifies computations).
- Choose the most important matrix U (size $D \times K$) with orthonormal columns with two equivalent criteria:
  - Minimizing **reconstruction error** $\qquad \operatorname{argmin}_U \frac{1}{N}\sum_{n=1}^{N}\|\overrightarrow{x_n} - \overrightarrow{\underline{x_n}}\|^2$
  - Maximizing **reconstruction variance** $\qquad \operatorname{argmax}_U \frac{1}{N}\sum_{n=1}^{N}\|\overrightarrow{\underline{x_n}} - \vec{\mu}\|^2$
- The goal is finding an accurate low-dimensional ($\mathbb{R}^K$) representation of data ($\mathbb{R}^D$).

Let X be a $N \times D$ matrix representing all data as vectors $\overrightarrow{x_n} \in \mathbb{R}^D$	$X = (\overrightarrow{x_1}^{\mathrm{T}}, \dots, \overrightarrow{x_N}^{\mathrm{T}})$	The reconstruction error can be vectorized into $\frac{1}{N}\sum_{n=1}^{N}\|\overrightarrow{x_n} - \overrightarrow{\underline{x_n}}\|^2 = \frac{1}{N}\sum_{n=1}^{N}\|\overrightarrow{x_n} - U\overrightarrow{z_n}\|^2 = \frac{1}{N}\|X - ZU^{\mathrm{T}}\|_F^2$
Let Z be a $N \times K$ matrix representing simplified data as vectors $\overrightarrow{z_n} \in \mathbb{R}^K$	$Z = (\overrightarrow{z_1}^{\mathrm{T}}, \dots, \overrightarrow{z_N}^{\mathrm{T}})$	
Let U be a $D \times K$ matrix	$U = [\overrightarrow{u_1} \quad \cdots \quad \overrightarrow{u_K}]$	So $X \approx ZU^{\mathrm{T}}$ & Z is a **rank-K** approximation of X.

- K is a hyperparameter. For $K = 1$, where $U = \vec{u}$,
  - We can show $\frac{1}{N}\sum_{n=1}^{N}\|\overrightarrow{\underline{x_n}} - \vec{\mu}\|^2 = \vec{a}^{\mathrm{T}}\Lambda\vec{a}$ where $\vec{a} = Q^{\mathrm{T}}\vec{u}$
  - We can show the optimal choice is $\vec{a} = \overrightarrow{e_1}$, where $\{\overrightarrow{e_1}, \dots, \overrightarrow{e_D}\}$ is standard basis, AKA $\vec{u} = \overrightarrow{q_1}$
    - We can show the i-th principle component is the eigenvector associated with the i-th largest λ_d (ie. The i-th principal component is the i-th eigenvalue of Σ)
  - Components of $\vec{z}$ are uncorrelated with each other (ie. the covariance matrix is diagonal)

$$\begin{aligned}\mathrm{Cov}(\vec{z}) &= \mathrm{Cov}\big(U^{\mathrm{T}}(\vec{x} - \vec{\mu})\big)\\ &= U^{\mathrm{T}}\mathrm{Cov}(\vec{x})U \text{ (as } \mathrm{Cov}(A\vec{x}) = A\mathrm{Cov}(\vec{x})A^{\mathrm{T}} \text{ and } \vec{\mu} \text{ is constant)}\\ &= U^{\mathrm{T}}\Sigma U\\ &= U^{\mathrm{T}}Q\Lambda Q^{\mathrm{T}}U \text{ (by spectral decomposition)}\\ &= [I \quad 0]\Lambda[I \quad 0]^{\mathrm{T}} \text{ (since columns of } U, Q \text{ orthonormal and can be rearranged)}\\ &= \text{top left } K \times K \text{ block of } \Lambda\end{aligned}$$

Singular-Value Decomposition (SVD): Generalization of spectral decomposition to any real $N \times D$ matrix X.

$$X = A\Sigma B^{\mathrm{T}}$$

A	$N \times N$	Orthonormal columns of **left singular vectors**	**Left Singular Vectors:** Eigenvectors of XX^{T}
Σ	$N \times D$	Diagonal matrix of **singular values** conventionally sorted in descending order	**Singular Values:** Square roots of eigenvalues of XX^{T} and $X^{\mathrm{T}}X$
B	$D \times D$	Orthonormal columns of **right singular vectors**	**Right Singular Vectors:** Eigenvectors of $X^{\mathrm{T}}X$

- Since Σ is not guaranteed to be square, set values in the "extra" rows/columns to 0s.
- It is possible to show the first n singular vectors are the first n principal components (ie. $Z = A\Sigma$)

Derivation – Equivalence of minimizing reconstruction error, maximizing reconstructing variance

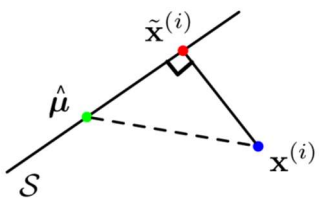
Recall that

$$\begin{aligned}\vec{z} &= U^{\mathrm{T}}(\vec{x}-\vec{\mu})\\ \underline{\vec{x}} &= U\vec{z}+\vec{\mu}\\ &= UU^{\mathrm{T}}(\vec{x}-\vec{\mu})+\vec{\mu}\end{aligned}$$

Remember that U has orthonormal columns, so $U^{\mathrm{T}}U = I$

$\underline{\vec{x}}$ is similar to $\vec{x}$, but because $UU^{\mathrm{T}} = I_K$, $\underline{\vec{x}}$ has 0 in some components.

Lecture uses $\tilde{\mathbf{x}}$ for $\underline{\vec{x}}$.



We will show $\underline{\vec{x}}-\vec{\mu}$ and $\underline{\vec{x}}-\vec{x}$ are orthogonal.

$$\begin{aligned}(\underline{\vec{x}}-\vec{\mu})^{\mathrm{T}}(\underline{\vec{x}}-\vec{x}) &= (U\vec{z})^{\mathrm{T}}(U\vec{z}+\vec{\mu}-\vec{x})\\ &= \vec{z}^{\mathrm{T}}U^{\mathrm{T}}\big(U\vec{z}-(\vec{x}-\vec{\mu})\big)\\ &= \vec{z}^{\mathrm{T}}U^{\mathrm{T}}U\vec{z}-\vec{z}^{\mathrm{T}}U^{\mathrm{T}}(\vec{x}-\vec{\mu})\\ &= \vec{z}^{\mathrm{T}}\vec{z}-(U\vec{z})^{\mathrm{T}}(\vec{x}-\vec{\mu})\\ &= \Big(U^{\mathrm{T}}(\vec{x}-\vec{\mu})\Big)^{\mathrm{T}}U^{\mathrm{T}}(\vec{x}-\vec{\mu})-\Big(UU^{\mathrm{T}}(\vec{x}-\vec{\mu})\Big)^{\mathrm{T}}(\vec{x}-\vec{\mu})\\ &= (\vec{x}-\vec{\mu})^{\mathrm{T}}UU^{\mathrm{T}}(\vec{x}-\vec{\mu})-(\vec{x}-\vec{\mu})^{\mathrm{T}}(\vec{x}-\vec{\mu})\\ &= (\vec{x}-\vec{\mu})^{\mathrm{T}}(\vec{x}-\vec{\mu})-(\vec{x}-\vec{\mu})^{\mathrm{T}}(\vec{x}-\vec{\mu})\\ &= 0\end{aligned}$$

Thus there's a right triangle between $\underline{\vec{x}}$, $\vec{x}$, and $\vec{\mu}$, so by Pythagorean theorem,

$$\|\underline{\vec{x}}-\vec{\mu}\|^2+\|\vec{x}-\underline{\vec{x}}\|^2 = \|\vec{x}-\vec{\mu}\|^2$$

$$\frac{1}{N}\sum_{n=1}^{N}\|\underline{\overrightarrow{x_n}}-\vec{\mu}\|^2+\frac{1}{N}\sum_{n=1}^{N}\|\overrightarrow{x_n}-\underline{\overrightarrow{x_n}}\|^2 = \frac{1}{N}\sum_{n=1}^{N}\|\overrightarrow{x_n}-\vec{\mu}\|^2$$

LS can be expressed in terms of U, while RS can't, so RS is a constant. Thus

$$\frac{1}{N}\sum_{n=1}^{N}\|\underline{\overrightarrow{x_n}}-\vec{\mu}\|^2 = \text{Constant}-\frac{1}{N}\sum_{n=1}^{N}\|\overrightarrow{x_n}-\underline{\overrightarrow{x_n}}\|^2$$

$$\text{Reconstruction Variance} = \text{Constant} - \text{Reconstruction Error}$$

Thus, maximizing LS is the same as minimizing RS.

Derivation – Optimization for principal component analysis, $K = 1$

Remember that

$$\begin{aligned}\vec{\tilde{x}}, \vec{\mu} &\in \mathcal{S}\\ \vec{x} &\in \mathbb{R}^D\\ \vec{z} = U^{\mathrm{T}}\vec{x} &\in \mathbb{R}^K\\ U &\text{ is } D\times K\end{aligned}$$

First, we note that

$$\begin{aligned}\|\underline{\vec{x}}-\vec{\mu}\|^2 &= \|U\vec{z}\|^2\\ &= (U\vec{z})^{\mathrm{T}}U\vec{z}\\ &= \vec{z}^{\mathrm{T}}U^{\mathrm{T}}U\vec{z}\\ &= \vec{z}^{\mathrm{T}}\vec{z}\\ &= \|\vec{z}\|^2\end{aligned}$$

In the case $K = 1$, the matrix U of orthogonal columns is a unit vector $\vec{u}$ and z is a scalar,

$$\begin{aligned}z &= (\vec{x}-\vec{\mu})\cdot\vec{u}\\ z^2 &= \|\underline{\vec{x}}-\vec{\mu}\|^2\end{aligned}$$

Let $\{\overrightarrow{e_1},\dots,\overrightarrow{e_D}\}$ be the standard basis.

We will then simplify reconstruction variance and maximize it,

$$\begin{aligned}\frac{1}{N}\sum_{n=1}^{N}\|\underline{\overrightarrow{x_n}}-\vec{\mu}\|^2 &= \frac{1}{N}\sum_{n=1}^{N}z_n^2\\ &= \frac{1}{N}\sum_{n=1}^{N}\big((\overrightarrow{x_n}-\vec{\mu})\cdot\vec{u}\big)^2\\ &= \frac{1}{N}\sum_{n=1}^{N}\vec{u}^{\mathrm{T}}(\overrightarrow{x_n}-\vec{\mu})(\overrightarrow{x_n}-\vec{\mu})^{\mathrm{T}}\vec{u} \quad (\text{as } (\vec{u}\cdot\vec{v})^2 = \vec{u}^{\mathrm{T}}\vec{v}\vec{v}^{\mathrm{T}}\vec{u})\\ &= \vec{u}^{\mathrm{T}}\left[\frac{1}{N}\sum_{n=1}^{N}(\overrightarrow{x_n}-\vec{\mu})(\overrightarrow{x_n}-\vec{\mu})^{\mathrm{T}}\right]\vec{u}\\ &= \vec{u}^{\mathrm{T}}\Sigma\vec{u} \quad (\text{where } \Sigma \text{ is } D\times D)\\ &= \vec{u}^{\mathrm{T}}Q\Lambda Q^{\mathrm{T}}\vec{u} \quad (\text{by spectral decomposition, } \Sigma \text{ is symmetric})\\ &= \vec{a}^{\mathrm{T}}\Lambda\vec{a} \quad (\text{where } \vec{a} = Q^{\mathrm{T}}\vec{u}\in\mathbb{R}^D)\\ &= [a_1 \quad \cdots \quad a_D]\begin{bmatrix}\lambda_1 a_1\\ \vdots\\ \lambda_D a_D\end{bmatrix}\\ &= \sum_{d=1}^{D}\lambda_d a_d^2\end{aligned}$$

When factoring Σ into $Q\Lambda Q^{\mathrm{T}}$, we arrange Λ such that the λ_d are ordered:

$$\begin{aligned}\Lambda_{1,1} &\geq \cdots \geq \Lambda_{D,D}\\ \lambda_1 &\geq \cdots \geq \lambda_D\end{aligned}$$

Note that $\vec{a}$ is a unit vector:

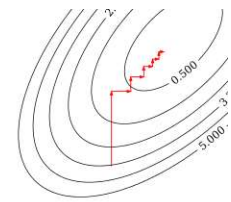
$$\begin{aligned}\|\vec{a}\|^2 &= \vec{a}^{\mathrm{T}}\vec{a}\\ &= \vec{u}^{\mathrm{T}}QQ^{\mathrm{T}}\vec{u}\\ &= \vec{u}^{\mathrm{T}}\vec{u}\\ &= \|\vec{u}\|^2 = 1\end{aligned}$$

To maximize variance, set $\vec{a} = \overrightarrow{e_1}$, and $\vec{a} = Q^{\mathrm{T}}\vec{u} \Rightarrow \vec{u} = Q\vec{a} = \overrightarrow{q_1}$.

The ith principal component is the eigenvector with the ith largest eigenvalue of Σ.

Sparse Matrix: A matrix in which the majority of values are 0.
Partially Observed Matrix: A matrix in which not all values are known (eg. matrix of users to movie ratings).
Matrix Completion: The task of filling in missing entries of a partially observed matrix

Observed: A variable that is known (eg. $\vec{x}$). It is **partially observed** if only some components are known.
Latent Factor/Hidden Variable: An unobserved variable $\vec{z} \in \mathbb{R}^K$ where $\vec{z} = f(\vec{x})$ for some $f\colon \mathbb{R}^D \to \mathbb{R}^K$ with $K < D$
Latent Factor Models: Models that tries to find K latent factors of all $x \in \mathbb{R}^D$ and predict outputs as a function of those K latent factors (K is a hyperparameter).



- Formally, we find representation $\vec{z} \in \mathbb{R}^K$ of $\vec{x} \in \mathbb{R}^D$

Block Coordinate Descent/Alternating Minimization: An iterative algorithm that optimizes two variables x, y where on alternating steps, one of x, y is set constant while the other is optimized.
Alternating Least Squares: Alternating minimization using least squares distance

eg. Matrix R of N users to D movie ratings, partially observed as most users don't watch the majority of available movies. We want to predict missing entries; predicted ratings of never seen-before movies for a user.

Let $\overrightarrow{u_n} \in \mathbb{R}^K$ be the latent factors of the n-th user
Let $\overrightarrow{z_d} \in \mathbb{R}^K$ be the latent factors of the d-th movie
The n-th user's rating for the d-th movie is $R_{n,d} \approx \overrightarrow{u_n} \cdot \overrightarrow{z_d}$
In matrix form, $R \approx UZ^{\mathrm{T}}$ where

$$U = \begin{bmatrix} \overrightarrow{u_1}^{\mathrm{T}} \\ \vdots \\ \overrightarrow{u_N}^{\mathrm{T}} \end{bmatrix} \text{ is } N \times K$$

$$Z = \begin{bmatrix} \overrightarrow{z_1}^{\mathrm{T}} \\ \vdots \\ \overrightarrow{z_D}^{\mathrm{T}} \end{bmatrix} \text{ is } D \times K$$

We can write reconstruction error as

$$\frac{1}{N}\|R - ZU^{\mathrm{T}}\|_F^2 = \frac{1}{N}\sum_{n=1}^{N}\sum_{d=1}^{D}(R_{n,d} - \overrightarrow{u_n} \cdot \overrightarrow{z_d})^2$$

But most values of $R_{n,d}$ are blank! Thus let

$$R' = \{(n,d)\colon R_{n,d} \text{ is observed}\}$$

Then we minimize error on R' with

$$\operatorname{argmin}_{U,Z}\left(\frac{1}{2}\sum_{(n,d)\in R'}(R'_{n,d} - \overrightarrow{u_n} \cdot \overrightarrow{z_d})^2\right)$$

And use the best U, Z to extrapolate to $R \setminus R'$.

- Previously, $\vec{z} = U^{\mathrm{T}}(\vec{x} - \vec{\mu})$, so $\vec{z}$ depended on U and $\operatorname{argmin}_{U,Z} = \operatorname{argmin}_U$. Now, we find $\operatorname{argmin}_{U,Z}$.
- Unfortunately, $\frac{1}{2}\sum_{(n,d)\in R'}(R'_{n,d} - \overrightarrow{u_n} \cdot \overrightarrow{z_d})^2$ is non-convex

We can re-write the objective function as

$$\frac{1}{2}\sum_{(n,d)\in R'}(R'_{n,d} - \overrightarrow{u_n} \cdot \overrightarrow{z_d})^2 = \frac{1}{2}\sum_{n=1}^{N}\sum_{d\in\{d:(n,d)\in R'\}}(R'_{n,d} - \overrightarrow{u_n} \cdot \overrightarrow{z_d})^2 = \frac{1}{2}\sum_{d=1}^{D}\sum_{n\in\{n:(n,d)\in R'\}}(R'_{n,d} - \overrightarrow{u_n} \cdot \overrightarrow{z_d})^2$$

For simplicity, define $D_n = \{d\colon (n,d) \in R'\}$

$$A = \sum_{d\in D}\left(R'_{n,d} - \overrightarrow{u_n}^{\mathrm{T}}\overrightarrow{z_d}\right)^2$$

This term depends only on $\overrightarrow{u_n}$. Taking the derivative,

$$\begin{aligned}\frac{\partial}{\partial\overrightarrow{u_n}}A &= -\sum_{d\in D_n}(R'_{n,d} - \overrightarrow{u_n}^{\mathrm{T}}\overrightarrow{z_d})(\overrightarrow{z_d}) \\ &= 0 \\ \sum_{d\in D_n}R'_{n,d}\overrightarrow{z_d} &= \sum_{d\in D_n}(\overrightarrow{u_n}^{\mathrm{T}}\overrightarrow{z_d})\overrightarrow{z_d}^{\mathrm{T}} \\ &= \overrightarrow{u_n}^{\mathrm{T}}\sum_{d\in D_n}\overrightarrow{z_d}\overrightarrow{z_d}^{\mathrm{T}}\end{aligned}$$

$$\overrightarrow{u_n}^{\mathrm{T}} = \sum_{d\in D_n}R'_{n,d}\overrightarrow{z_d}\left(\sum_{d\in D_n}\overrightarrow{z_d}\overrightarrow{z_d}^{\mathrm{T}}\right)^{-1}$$

$$\overrightarrow{u_n} = \left(\sum_{d\in D_n}\overrightarrow{z_d}\overrightarrow{z_d}^{\mathrm{T}}\right)^{-1}\sum_{d\in D_n}R'_{n,d}\overrightarrow{z_d}$$

For simplicity, define $N_d = \{n\colon (n,d) \in R'\}$

$$B = \sum_{n\in\{n:(n,d)\in R'\}}\left(R'_{n,d} - \overrightarrow{z_d}^{\mathrm{T}}\overrightarrow{u_n}\right)^2$$

This term depends only on $\overrightarrow{z_n}$. Taking the derivative,

$$\begin{aligned}\frac{\partial}{\partial\overrightarrow{u_n}}B &= -\sum_{n\in N_d}(R'_{n,d} - \overrightarrow{z_d}^{\mathrm{T}}\overrightarrow{u_n})(\overrightarrow{u_n}) \\ &= 0 \\ \sum_{n\in N_d}R'_{n,d}\overrightarrow{u_n} &= \sum_{n\in N_d}(\overrightarrow{z_d}^{\mathrm{T}}\overrightarrow{u_n})\overrightarrow{u_n}^{\mathrm{T}} \\ &= \overrightarrow{z_d}^{\mathrm{T}}\sum_{n\in N_d}\overrightarrow{u_n}\overrightarrow{u_n}^{\mathrm{T}}\end{aligned}$$

$$\overrightarrow{z_d}^{\mathrm{T}} = \sum_{n\in N_d}R'_{n,d}\overrightarrow{u_n}\left(\sum_{n\in N_d}\overrightarrow{u_n}\overrightarrow{u_n}^{\mathrm{T}}\right)^{-1}$$

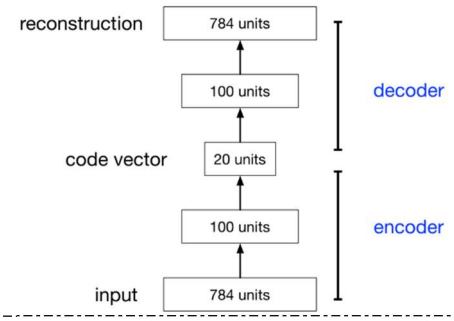
$$\overrightarrow{z_d} = \left(\sum_{n\in N_d}\overrightarrow{u_n}\overrightarrow{u_n}^{\mathrm{T}}\right)^{-1}\sum_{n\in N_d}R'_{n,d}\overrightarrow{u_n}$$

An algorithm for this would look something like:

```
while not converged:
    for n in range(1, N+1):    u_n ← (Σ_{d∈D_n} z_d z_d^T)^{-1} Σ_{d∈D_n} R'_{n,d} z_d
    for d in range(1, D+1):    z_d ← (Σ_{n∈N_d} u_n u_n^T)^{-1} Σ_{n∈N_d} R'_{n,d} u_n
```

Autoencoder: A feed-forward neural network that outputs a $\mathbb{R}^K$ approximation of an input $\vec{x} \in \mathbb{R}^D$

- We want a lower-dimensional representation, so $K < D$
- Uses a bottleneck layer of K neurons (hyperparameter)
- Mapping to 2D allows visualization
- Helps learn abstract features in unlabelled data that can be applied to a supervised task



Linear Autoencoder: Autoencoder made up of linear functions

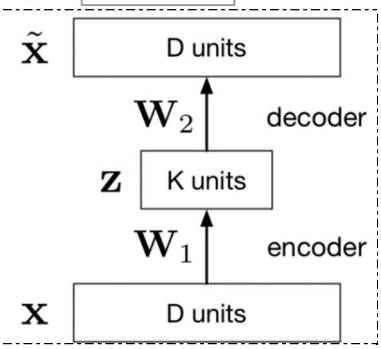
eg. Consider one hidden layer, $\phi(x) = x$, and $\mathcal{L}(\vec{x}, \underline{\vec{x}}) = \|x - \underline{\vec{x}}\|^2$

The reconstruction is a linear function, $\underline{\vec{x}} = W_2 W_1 \vec{x}$.

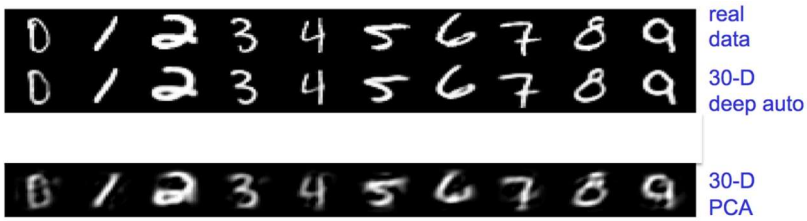
We know $\underline{\vec{x}} \in \text{span}(W_2 \text{ columns})$ as $\underline{\vec{x}}$ is the result of a product with W_2

The optimal K-dimensional linear subspace, reconstruction error-wise, is the subspace from PCA whose transformation matrix is U.

Optimal weights are just principal components! Set $W_1 = U^{\mathrm{T}}, W_2 = U$



Nonlinear Autoencoder: Autoencoders that project data onto a nonlinear manifold (image of decoder). Can learn much more powerful codes compared to linear autoencoders



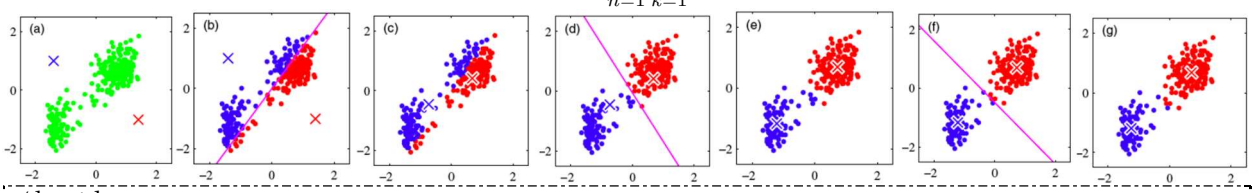
Clustering

Multimodal: A distribution with multiple modes (local regions of high probability mass/density)

k-Means Clustering: Unsupervised learning technique to fit N unlabelled data points into spherical clusters

- There are K clusters (hyperparameter), each with a center (the mean of all cluster points)
- Goal is to assign data points ($\overrightarrow{x_n} \in \mathbb{R}^D$) to clusters that minimize distances to cluster centers ($\overrightarrow{m_k} \in \mathbb{R}^D$)
  - Let $\overrightarrow{r_n} \in \mathbb{R}^K$ be a one-hot vector where $(\overrightarrow{r_n})_k = \mathbb{I}(\overrightarrow{x_n} \text{ assigned to cluster } k)$.
  - Let $M = [\overrightarrow{m_1} \quad \cdots \quad \overrightarrow{m_K}]^{\mathrm{T}}, R = [\overrightarrow{r_1} \quad \cdots \quad \overrightarrow{r_N}]^{\mathrm{T}}$

$$\min_{M,R} \mathcal{J}(M,R) = \min_{M,R} \sum_{n=1}^{N} \sum_{k=1}^{K} (r_n)_k \|\overrightarrow{m_k} - \overrightarrow{x_n}\|^2$$



<u>Algorithm</u>

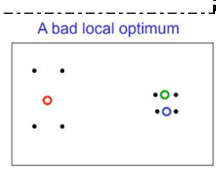
1) Randomly initialize cluster centers $\overrightarrow{m_k}$
2) Alternate between two steps until convergence (ie. data point assignments don't change):
   a. **Assignment:** Assign data points to their closest clusters – minimize with respect to R

$$\text{argmin}_{\overrightarrow{r_n}} \sum_{k=1}^{K} (r_n)_k \|\overrightarrow{m_k} - \overrightarrow{x_n}\|^2 = \begin{bmatrix} \mathbb{I}\left(\text{argmin}_{k\in\{1,\dots,K\}} \|\overrightarrow{m_k} - \overrightarrow{x_n}\|^2 = 1\right) \\ \vdots \\ \mathbb{I}\left(\text{argmin}_{k\in\{1,\dots,K\}} \|\overrightarrow{m_k} - \overrightarrow{x_n}\|^2 = K\right) \end{bmatrix} \quad \text{(closest cluster center to } \overrightarrow{x_n}\text{)}$$

   b. **Refitting:** Move cluster centers to mean of its members – minimize with respect to M

$$\text{argmin}_{\overrightarrow{m_{k^*}}} \sum_{n=1}^{N} \sum_{k=1}^{K} (r_n)_k \|\overrightarrow{m_k} - \overrightarrow{x_n}\|^2 = \frac{\sum_{n=1}^{N} (r_n)_{k^*} \overrightarrow{x_n}}{\sum_{n=1}^{N} (r_n)_{k^*}} \quad \text{(mean of all } \overrightarrow{x_n} \text{ in class } k^*\text{)}$$

- K-means will converge as cost is guaranteed to reduce each iteration, and number of cluster assignments is finite
- Unstable, $\mathcal{J}$ is non-convex, no guarantee of global minimum.
- Used in **vector quantization**, technique for compressing data
- Used in **image segmentation**, technique for finding superpixels ("perceptual pixels")



Optimization – Derivation of Iterative Solution, K-Means Clustering

$$\frac{\partial}{\partial \overrightarrow{m_{k^*}}} \sum_{n=1}^{N} \sum_{k=1}^{K} (r_n)_k \|\overrightarrow{m_k} - \overrightarrow{x_n}\|^2 = \frac{\partial}{\partial \overrightarrow{m_{k^*}}} \sum_{n=1}^{N} (r_n)_{k^*} \|\overrightarrow{m_{k^*}} - \overrightarrow{x_n}\|^2 \quad \text{(since } \overrightarrow{r_n} \text{ is one-hot)}$$

$$= 2 \sum_{n=1}^{N} (r_n)_{k^*} (\overrightarrow{m_{k^*}} - \overrightarrow{x_n}) = 0$$

$$\overrightarrow{m_{k^*}} \sum_{n=1}^{N} (r_n)_{k^*} = \sum_{n=1}^{N} (r_n)_{k^*} \overrightarrow{x_n}$$

$$\therefore \overrightarrow{m_{k^*}} = \frac{\sum_{n=1}^{N} (r_n)_{k^*} \overrightarrow{x_n}}{\sum_{n=1}^{N} (r_n)_{k^*}}$$

Soft k-Means Clustering: A variant of k-means that makes soft assignments (between 0 to 1) of data to clusters

- Introduce hyperparameter β where as $\beta \to \infty$, soft k-means becomes k-means

<u>Algorithm</u>

1) Randomly initialize cluster centers $\overrightarrow{m_k}$
2) Alternate between two steps until convergence (ie. data point assignments don't change):
   a. **Assignment:** Assign data points to their closest clusters – optimize for R

$$\overrightarrow{r_n} = \text{softmax}\left(-\beta \begin{bmatrix} \|\overrightarrow{m_1} - \overrightarrow{x_n}\|^2 \\ \vdots \\ \|\overrightarrow{m_K} - \overrightarrow{x_n}\|^2 \end{bmatrix}\right)$$

   b. **Refitting:** Move cluster centers to mean of its members – optimize for M

$$\overrightarrow{m_{k^*}} = \frac{\sum_{n=1}^{N} (r_n)_{k^*} \overrightarrow{x_n}}{\sum_{n=1}^{N} (r_n)_{k^*}}$$

Identifiable: A probability model that given infinitely much data, can theoretically get true model parameters.

Latent Variable Model: A probabilistic interpretation of latent factors Z,

$$p(\vec{x}) = \sum_{z \in Z} p_{\mathcal{D},Z}(\vec{x}, z) = \sum_{z \in Z} p_{\mathcal{D}|Z}(\vec{x}|z) p_Z(z)$$

Mixture Model: A latent variable model where $p_Z(z)$ is a categorical distribution (ie. $Z = \{z_1, \dots, z_N\}$ is finite).

- Assumes data was generated by
  - Choosing a mixture component randomly (ie. $p_Z(z)$)
  - Sampling from the distribution associated with that component (ie. $p_{\mathcal{D}|Z}(\vec{x}|z)$)
- Has two main parts:
  - **Mixing Proportions/Coefficients:** Vector $\vec{\eta}$ with $\eta_1 + \dots + \eta_K = 1$ and $\eta_k \geq 0$.
  - Parameters associated with component distributions

Gaussian Mixture Model (GMM): The mixture model

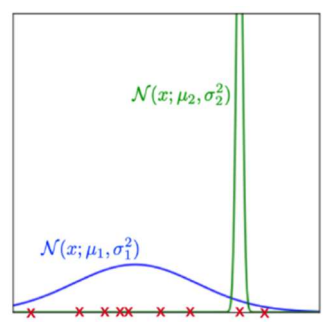
$$z_n \sim \text{Categorical}(\vec{\eta}) \quad \text{(for some categorical distribution)}$$
$$\overrightarrow{x_n} | z_n \sim \text{Normal}(\mu_{z_n}, \Sigma_{z_n})$$

$$p_{X|\theta}(\vec{x}|\theta) = \sum_{k=1}^{K} \frac{\eta_k}{\det(\Sigma_k)^{\frac{1}{2}} (2\pi)^{\frac{D}{2}}} e^{-\frac{1}{2}(\vec{x}-\overrightarrow{\mu_k})^{\mathrm{T}} \Sigma_k^{-1} (\vec{x}-\overrightarrow{\mu_k})} \quad \left(\text{where } \theta = (\vec{\mu}, \vec{\eta}, \Sigma), \begin{array}{r} \vec{\mu} = \{\overrightarrow{\mu_1}, \dots, \overrightarrow{\mu_K}\} \\ \vec{\eta} = \{\overrightarrow{\eta_1}, \dots, \overrightarrow{\eta_K}\} \\ \Sigma = \{\Sigma_1, \dots, \Sigma_K\} \end{array}\right)$$

- GMMs, and even diagonal GMMs (where the Σ_k are diagonal) are **universal density approximators**
- GMMs are **not identifiable**, infinitely many ways to fit data
  - eg. You can fit noise as just noise or as a rapidly varying function with no noise
  - eg. $\theta_1 = (\overrightarrow{\mu_1}, \overrightarrow{\eta_1}, \Sigma), \theta_2 = (\overrightarrow{\mu_2}, \overrightarrow{\eta_2}, \Sigma)$ is the same as $\theta_1 = (\overrightarrow{\mu_2}, \overrightarrow{\eta_2}, \Sigma), \theta_2 = (\overrightarrow{\mu_1}, \overrightarrow{\eta_1}, \Sigma)$
- We can attempt to find parameters with maximum likelihood:
  - Let $X = [\overrightarrow{x_1} \quad \cdots \quad \overrightarrow{x_N}]^{\mathrm{T}}, \vec{z} = [z_1 \quad \cdots \quad z_N]^{\mathrm{T}}$

$$\begin{aligned} \ln p(X|\theta) &= \ln \prod_{n=1}^{N} p_{X_n|\theta}(\overrightarrow{x_n}|\theta) \\ &= \sum_{n=1}^{N} \ln \left(\sum_{k=1}^{K} \frac{\eta_k}{\det(\Sigma_k)^{\frac{1}{2}} (2\pi)^{\frac{D}{2}}} e^{-\frac{1}{2}(\vec{x}-\overrightarrow{\mu_k})^{\mathrm{T}} \Sigma_k^{-1} (\vec{x}-\overrightarrow{\mu_k})} \right) \end{aligned}$$

  - This isn't simplifiable, so the derivative is very complicated. No closed-form solution.
  - Difficult to optimize with gradient descent due to non-convexity, enforcing non-negativity constraint on π_k, PSD constraint on Σ_k, and $\frac{\partial}{\partial \Sigma_k}$ is computationally slow



  - **Singularity:** A tiny-variance Gaussian component fitted to one training example. Can create arbitrarily high training likelihoods *(think of it like K-means, where you give one point its own class).*

Assuming we know θ, we can find $p(z_k = 1|\overrightarrow{x_n})$ using Bayes' Rule (similar to posterior):

$$p(z_k = 1|\overrightarrow{x_n}) = \frac{p(z_k = 1)p(\vec{x}|z_k = 1)}{\sum_{i=0}^{1} p(z_k = i)p(\vec{x}|z_k = i)}$$

Assuming we know $\vec{z}$, we can find the optimal θ by maximizing joint log-likelihood,

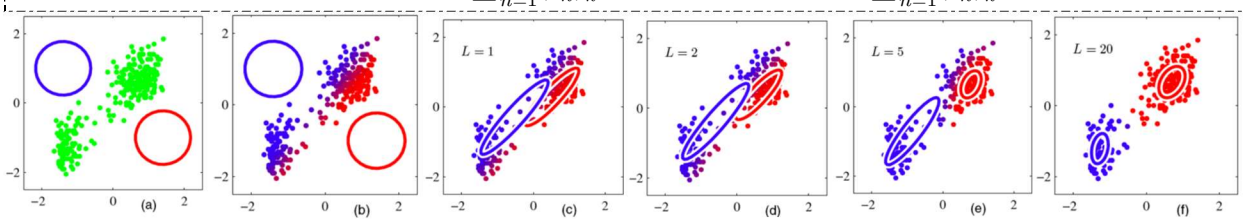
$$\ln p(X, \vec{z}|\theta) = \sum_{n=1}^{N} \ln p_{X_n, \vec{z}|\theta}(\overrightarrow{x_n}, z_n|\theta)$$
$$= \sum_{n=1}^{N} \left[\ln p_{\vec{z}|\theta}(z_n|\theta) + \ln p_{X_n|\theta}(\overrightarrow{x_n}|\theta)\right]$$

The first term is found computationally.
Substitute the formula for GMM into the second term.

Expectation-Maximization (E-M): Alternating minimization algorithm for $\vec{z}$ and θ

<u>Algorithm</u>

1) Randomly initialize $\vec{z}, \theta$
2) Alternate between two steps until convergence
   a. **Expectation Step:** Assign each $\overrightarrow{x_n}$ a soft probability of coming from each distribution – find $\vec{z}$
   $$(r_n)_k = p(z_k = 1|\overrightarrow{x_n}, \theta)$$
   b. **Maximization Step:** Find the best Gaussian fits to their members – optimize for θ
   $$\eta_k = \frac{1}{N}\sum_{n=1}^{N}(r_n)_k\,, \qquad \overrightarrow{\mu_k} = \frac{\sum_{n=1}^{N}(r_n)_k\,\overrightarrow{x_n}}{\sum_{n=1}^{N}(r_n)_k}, \qquad \Sigma_k = \frac{\sum_{n=1}^{N}(r_n)_k\,(\overrightarrow{x_n} - \overrightarrow{\mu_k})(\overrightarrow{x_n} - \overrightarrow{\mu_k})^{\mathrm{T}}}{\sum_{n=1}^{N}(r_n)_k}$$



- If mixture coefficients are $\eta_k = \frac{1}{K}$, covariances are diagonal, $\Sigma_k = \sigma I$ for any $\sigma > 0$, E-M is soft k-means.

Reinforcement Learning

Agent: The learning model that is controlling an entity in some program.
Sequential Decision Making: When agents choose from many actions which affect future possibilities.
Action (A_t): Set of everything the agent can do at time t.
State (S_t): Set of all possible descriptions of the program at time t. *(assume this is fully observable, AKA knowable)*
Action Space ($\mathcal{A}$): Set of all possible action. Can be finite or infinite *(assume it's finite for now)*
State Space ($\mathcal{S}$): Set of all possible states. Can be discrete or continuous.
Transition Probability ($\mathcal{P}$): The probability $\mathcal{P}: \mathcal{S}|\mathcal{S}, \mathcal{A} \to [0,1]$ of transforming to a new state given an action. Describes the **dynamics** of the environment. If known, we can **plan**; otherwise, we do **trial-and-error learning**
Policy (π): A function π describing how agents choose actions, given the state.

- **Deterministic Policy:** The function $\pi: \mathcal{S} \to \mathcal{A}$ *"Given a state, output an action"*
- **Stochastic Policy:** The function $\pi: \mathcal{S} \times \mathcal{A} \to [0,1]$ *"Given a state & action, output the prob. of choosing the action"*
  - Can be interpreted as $\pi: \mathcal{S} \to [0,1]^{|\mathcal{A}|}$ *"Given a state, output a vector of probabilities for each action"*
  - Can be interpreted as a probability function $\pi: \mathcal{A}|\mathcal{S} \to [0,1]$
  - The probability of taking actions $a_1, \dots, a_T$, given corresponding states $s_1, \dots, s_t$, is
    $p(s_1, a_1, \dots, s_T, a_T) = p(s_1)[\pi(a_1|s_1)\mathcal{P}(s_2|s_1, a_1)] \cdots [\pi(a_{T-1}|s_{T-1})\mathcal{P}(s_T|s_{T-1}, a_{T-1})]\pi(a_T|s_T)$

Reward Signal (R_t): A function outputting a value that determines how "good" a decision is at time t.

- Stochastic rewards are $R_t \sim \mathcal{R}(\mathcal{S}, \mathcal{A}|S_t, A_t)$ for some $\mathcal{R}: \mathcal{S} \times \mathcal{A} \to \mathbb{R}$
- Deterministic rewards $R_t = \mathcal{R}(S_t, A_t)$ for some $\mathcal{R}: \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ *(assume this for now)*

Return (G_t): Reward signal, but accounting for long-term reward. We should maximize $\mathbb{E}[G_t]$.

- **Undiscounted:** Return that places no weight on future reward, $G_t = R_t + R_{t+1} + R_{t+2} + \cdots$
- **Discounted:** Return that places differing weights on future reward, $G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots$
  - **Discount Factor:** Hyperparameter $0 \le \gamma < 1$ affecting weights of rewards now vs. later

Markov Assumption: The assumption that S_{t+1} depends on S_t but nothing before it.
Markov Decision Process (MDP): Mathematical framework for reinforcement learning, tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$

- $\mathcal{S}$: State space
- $\mathcal{A}$: Action space
- $\mathcal{P}$: Transition probabilities
- $\mathcal{R}$: Immediate reward functions
- γ: Discount factor

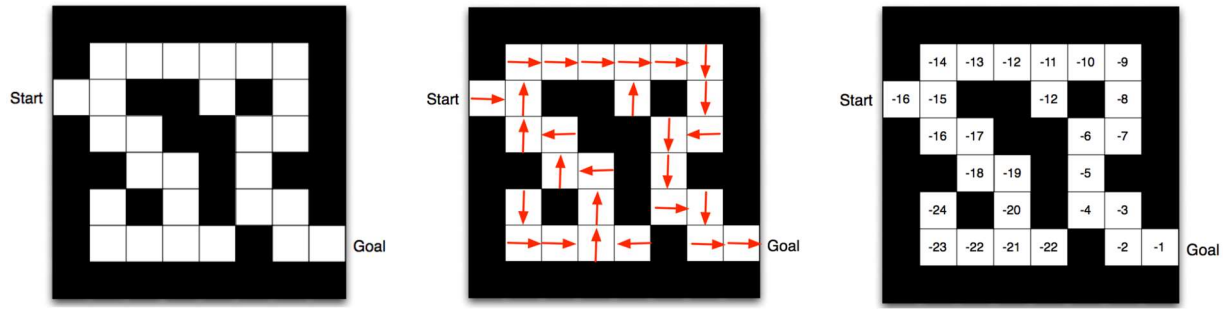
Value Function (V^π): For policy π at state $s^* \in \mathcal{S}$, function $V^\pi: \mathcal{S} \to \mathbb{R}$ measuring the desirability of being in state s^*. Equal to expected gain based on state,

$$\begin{aligned}
V^\pi(s^*) &= \mathbb{E}_\pi[G_t|S_t = s^*] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k} \,|S_t = s^*\right] \\
&= \sum_{a \in A_t} \pi(a|s^*)\left[\mathcal{R}(s^*, a) + \gamma \sum_{s \in S_{t+1}} \mathcal{P}(s|s^*, a)\mathbb{E}_\pi[G_{t+1}|\mathcal{S}_{t+1} = s]\right] \\
&= \sum_{a \in A_t} \underbrace{\pi(a|s^*)}_{\substack{\text{prob. of} \\ \text{action } a}} \left[\underbrace{\mathcal{R}(s^*, a)}_{\text{initial gain}} + \underbrace{\gamma \sum_{s \in S_{t+1}} \mathcal{P}(s|s^*, a)V^\pi(s)}_{\text{future gain}}\right] \quad (\pi(a|s^*) \text{ is one-hot if deterministic})
\end{aligned}$$

State-Action Value Function/Q-Function (Q^π): For policy π at state $s^* \in \mathcal{S}$, given action $a^* \in \mathcal{A}$, function $Q^\pi: \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ measuring the desirability of an action, given a state.

$$\begin{aligned}
Q^\pi(s^*, a^*) &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k} \,|S_t = s^*, A_t = a^*\right] \\
&= \underbrace{\mathcal{R}(s^*, a^*)}_{\text{initial gain}} + \underbrace{\gamma \sum_{s \in S_{t+1}} \mathcal{P}(s|a^*, s^*) \sum_{a \in A_{t+1}} \pi(a|s)Q^\pi(s, a)}_{\text{future gain}} \quad (\pi(a|s) \text{ is one-hot if deterministic})
\end{aligned}$$

eg. Consider a maze, with $\mathcal{A} = \{\text{Up}, \text{Down}, \text{Left}, \text{Right}\}, \mathcal{S} = \{\text{agent's locations}\}, \mathcal{R} = -1 \text{ per step}, \gamma = 0$



Arrows are $\pi(s)$ for all $s \in \mathcal{S}$ Numbers are $V^\pi(s)$ for all $s \in \mathcal{S}$

Bellman Equation: A type of equation that returns the "value" of action a at time t as the sum of the action's immediate reward and the "value" of future actions as a result of a.

- Includes V^π and Q^π
- V^π and Q^π can be expressed in terms of each other. It's easier to find V^π from Q^π *(as you iterate through a, not s)*

$$V^\pi(s^*) = \sum_{a \in A_t} \pi(a|s^*) Q^\pi(s^*, a) \qquad Q^\pi(s^*, a^*) = \mathcal{R}(s^* a^*) + \gamma \sum_{s \in S_{t+1}} \mathcal{P}(s|s^*, a^*) V^\pi(s)$$

Bellman Backup/Update Operator: On vectors of all states/actions $s^* \in \mathcal{S}, a^* \in \mathcal{A}$, the operator T^π where

$$(T^\pi V)(s^*) = V^\pi\big(V(s^*)\big)$$
$$(T^\pi Q)(s^*, a^*) = Q^\pi\big(Q(s^*, a^*)\big)$$

Where V, Q are analogous to V^π, Q^π, and all functions are applied element-wise.

Fixed Point: A point that does not change under a transformation; that is, a point x such that $f(x) = x$

Contraction Map: Function $f: \mathbb{R}^n \to \mathbb{R}^n$ if $\|f(\overrightarrow{x_1}) - f(\overrightarrow{x_2})\| \le \alpha \|\overrightarrow{x_1} - \overrightarrow{x_2}\|$ for some $0 \le \alpha < 1$

- For some point $\overrightarrow{x_1}$, contraction map f, $(f \circ \ldots \circ f)(\overrightarrow{x_1}) \to \overrightarrow{x_2}$ for some unique fixed point $\overrightarrow{x_2}$
  - This is because $\|(\underbrace{f \circ \ldots \circ f}_{k \text{ times}})(\overrightarrow{x_1}) - \overrightarrow{x_2}\| \le \alpha^k \|\overrightarrow{x_1} - \overrightarrow{x_2}\| \to 0$ as $k \to \infty$

Dynamic Programming: An optimization and programming method based on breaking problems into nested sub-problems and recursively solving them. Requires its problems to fit Bellman equations.

Assuming a deterministic policy, we choose the action that maximizes the desirability of the resulting state (ie. maximize the Q-function).

$$\pi^*(s) = \text{argmax}_{a \in A_t} Q^\pi(s, a)$$
$$\therefore Q^{\pi^*}(s^*, a^*) = \mathcal{R}(s, a) + \gamma \sum_{s \in S_{t+1}} \mathcal{P}(s|a^*, s^*) Q^{\pi^*}\big(s, \pi^*(s)\big)$$
$$= \mathcal{R}(s, a) + \gamma \sum_{s \in S_{t+1}} \mathcal{P}(s|a^*, s^*) \max_{a \in A_{t+1}} Q^{\pi^*}(s, a)$$
$$\therefore (T^{\pi^*} Q)(s^*, a^*) = Q^{\pi^*}\big(Q(s^*, a^*)\big)$$

$T^{\pi^*} Q$ happens to be a **contraction map** (see proof below).

Value Iteration: The idea of applying T^{π^*} iteratively until the result converges to an ideal fixed point:

$$Q^{\pi^*}\big(Q(s^*, a^*)\big) = Q(s^*, a^*)$$

Use the pseudocode below:

```
initialize Q randomly
while not converged:
    Q ← T^{π*} Q
```

Planning: Creating a MDP where the dynamics (ie. $\mathcal{P}$) are known

- Value iteration requires knowing $\mathcal{P}$ and explicitly representing Q as a vector
  - $|\mathcal{S}|$ can be extremely large or infinite
  - $|\mathcal{A}|$ can also be continuous and/or infinite

Learning: Creating a MDP where the dynamics (ie. $\mathcal{P}$) are unknown

- Learn a model of the environment, and do planning in this model *(ie. model-based reinforcement learning)*
- Learn a value function
- Learn a policy directly
- Deal with large $|\mathcal{S}|$ using function approximations!

Optimization – Proof that T^{π^*} is a Contraction Map

$$
\begin{aligned}
|(T^{\pi^*}Q_1)(s^*,a^*) - (T^{\pi^*}Q_2)(s^*,a^*)| &= \left|\left[\mathcal{R}(s^*,a^*) + \gamma \textstyle\sum_{s\in S_{t+1}} \mathcal{P}(s|s^*,a^*) \max_{a\in A_{t+1}} Q_1(s^*,a^*)\right] - \left[\mathcal{R}(s^*,a^*) + \gamma \textstyle\sum_{s\in S_{t+1}} \mathcal{P}(s|s^*,a^*) \max_{a\in A_{t+1}} Q_2(s^*,a^*)\right]\right| \\
&= \gamma \left| \sum_{s\in S_{t+1}} \mathcal{P}(s|s^*,a^*) \left[\max_{a\in A_{t+1}} Q_1(s^*,a^*) - \max_{a\in A_{t+1}} Q_2(s^*,a^*)\right]\right| \\
&\le \gamma \sum_{s\in S_{t+1}} \mathcal{P}(s|s^*,a^*) \left[\max_{a\in A_{t+1}} |Q_1(s^*,a^*) - Q_2(s^*,a^*)|\right] \text{ (by triangle inequality?)} \\
&= \gamma \max_{a\in A_{t+1}} |Q_1(s^*,a^*) - Q_2(s^*,a^*)| \sum_{s\in S_{t+1}} \mathcal{P}(s|s^*,a^*) \\
&= \gamma \max_{a\in A_t+1} |Q_1(s^*,a^*) - Q_2(s^*,a^*)| \text{ (transition probabilities add to 1)} \\
&= \gamma \|Q_1(s^*,a^*) - Q_2(s^*,a^*)\|_\infty \text{ (the } \infty\text{-norm is defined as } \max\{x_1,\dots,x_n\})
\end{aligned}
$$

Since our s^*, a^* was arbitrary, this holds for any s^*, a^*, so $\max_{s\in S_t, a\in A_t} |T^{\pi^*}Q_1 - T^{\pi^*}Q_2| \le \gamma \max_{a\in A_t+1} |Q_1 - Q_2|$

$$\therefore \|T^{\pi^*}Q_1 - T^{\pi^*}Q_2\|_\infty \le \gamma \|Q_1 - Q_2\|_\infty$$

Monte Carlo: An estimating technique based on taking random samples

- To estimate $\mu = \mathbb{E}[X]$, one can repeatedly sample X and update $\mu \leftarrow \mu + \alpha(X - \mu)$ with learning rate α

Exploration-Exploitation Tradeoff: The tradeoff between choosing what seems to be the best path versus trying out new paths, which occurs when we don't know $\mathcal{P}$ and must find the best choices ourselves.

- **ϵ-Greedy Policy:** Stochastic policy $\pi(a,s) = \begin{cases} \operatorname*{argmax}_{a\in A_t} Q^\pi(s,a) & \text{with probability } 1-\epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases}$ (usually $\epsilon = \frac{1}{20}$)

Q-Learning: The process of finding the best Q without knowledge of $\mathcal{P}$

- Initialize $Q(s,a)$ for all $(s,a) \in \mathcal{S} \times \mathcal{A}$
- Agent starts at state S_0
- For each time step $t \in \mathbb{N}$,
  - Choose A_t according to ϵ-greedy policy
  - Observe S_{t+1} and R_t, update Q based on results as if it were a Monte Carlo sample:

$$Q(S_{t+1}, A_{t+1}) \leftarrow Q(S_t, A_t) + \alpha\left[R_t + \gamma \max_{a\in\mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t)\right]$$

    - Note that the policy π isn't in this formula! π only determines what states we visit.
    - **Temporal Difference Learning:** Updating prior predictions to match later predictions

Off-Policy: An algorithm that doesn't rely on π to converge to the optimal Q-function – we can pick any π we want

On-Policy: An algorithm that relies on π to converge to the optimal Q-function (eg. policy gradient, not in CSC311)

Tabular Representation: Representation of data as a table/matrix.

- We've stored Q as a $|\mathcal{S}| \times |\mathcal{A}|$ matrix – impractical, doesn't share structure between related states

Parameterized Function: A lower-dimensional approximation function $\mathbb{R}^n \to \mathbb{R}^m$ with $n < m$

- Can be done with a neural network; called **deep Q-learning**

EXTRA NOTES AFTER STUDYING FOR EXAMS

- Dropout, sparsity, capacity, momentum, cosine distance, saturation – concepts taught in previous offerings of CSC311
- Though validation set is used for hyperparameters, it's okay to train optimization hyperparameters (learning rate, batch size) on the training set – they determine convergence rather than overfitting.
- Testing set is used for generalization, so that's why we can't use it for anything else