Feature Learning

Anita Faul

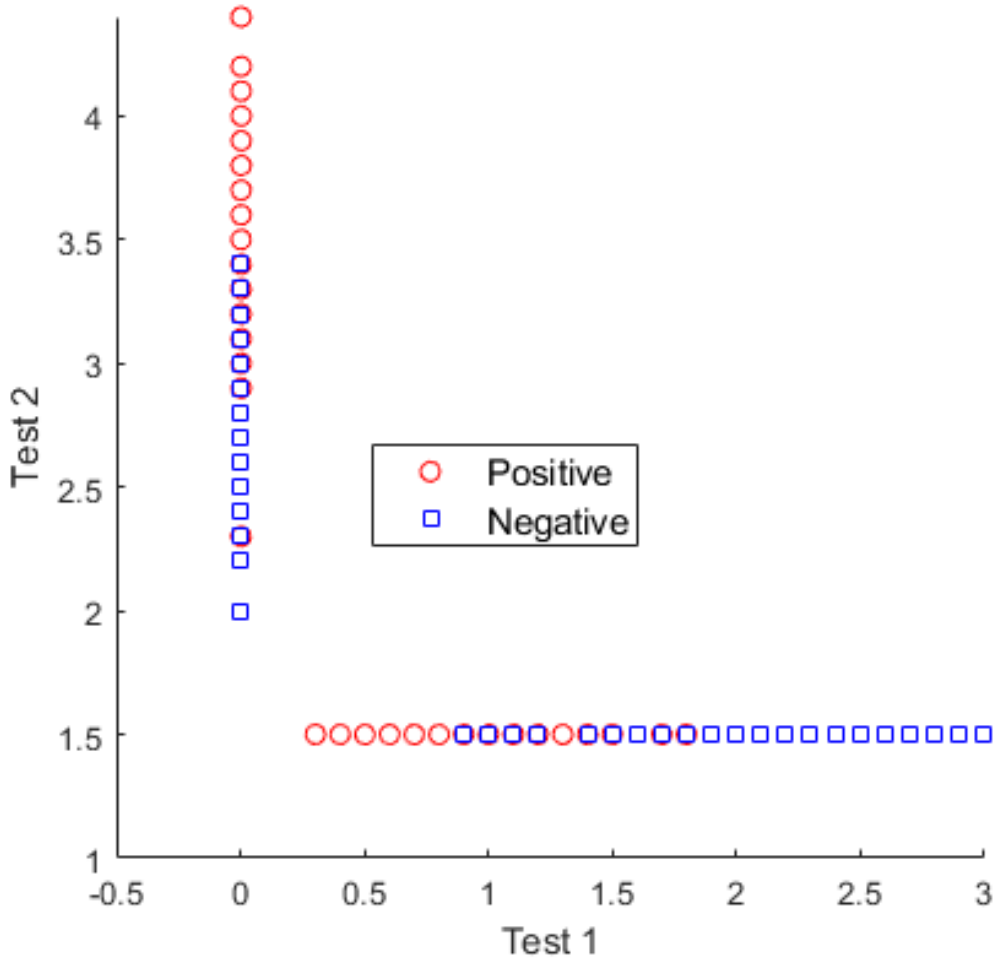
Laboratory for Scientific Computing, University of Cambridge

Binary Classification

Binary Classification:

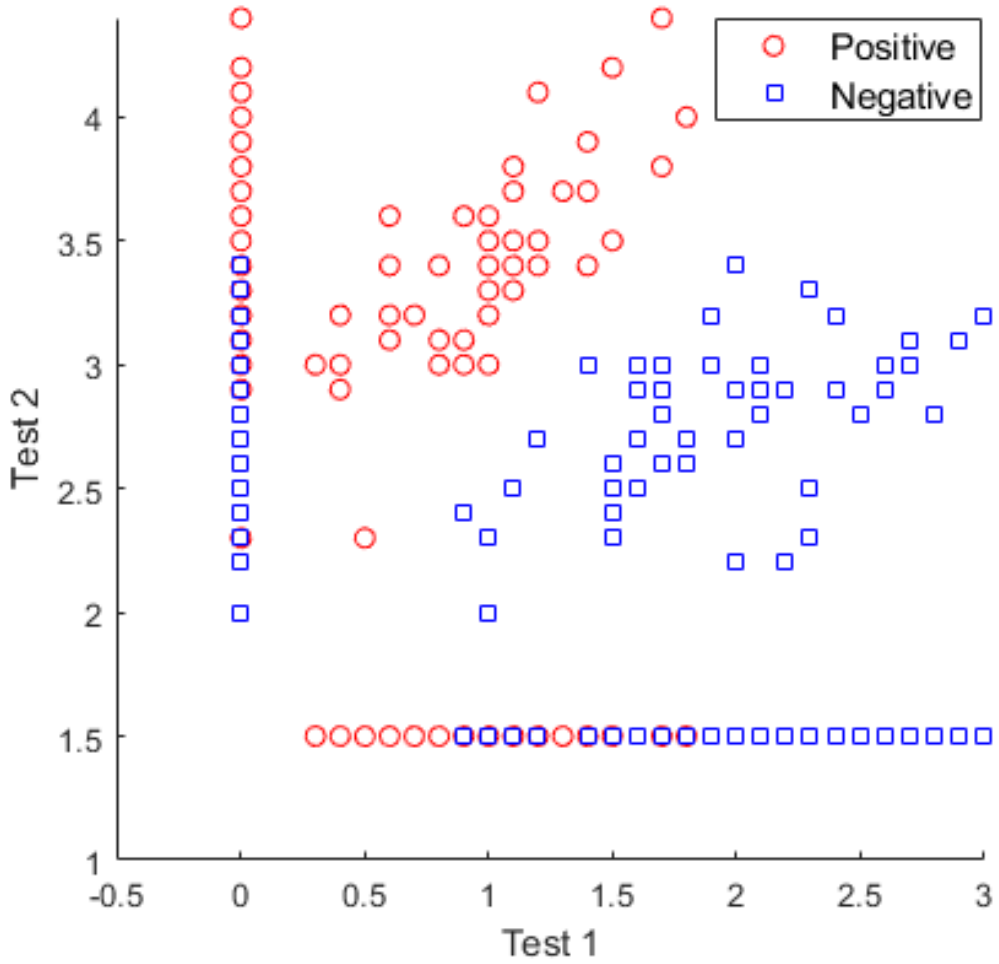
- N negative samples,
- P positive samples,
- altogether $N + P$ samples.
- Two tests.

Combination of tests

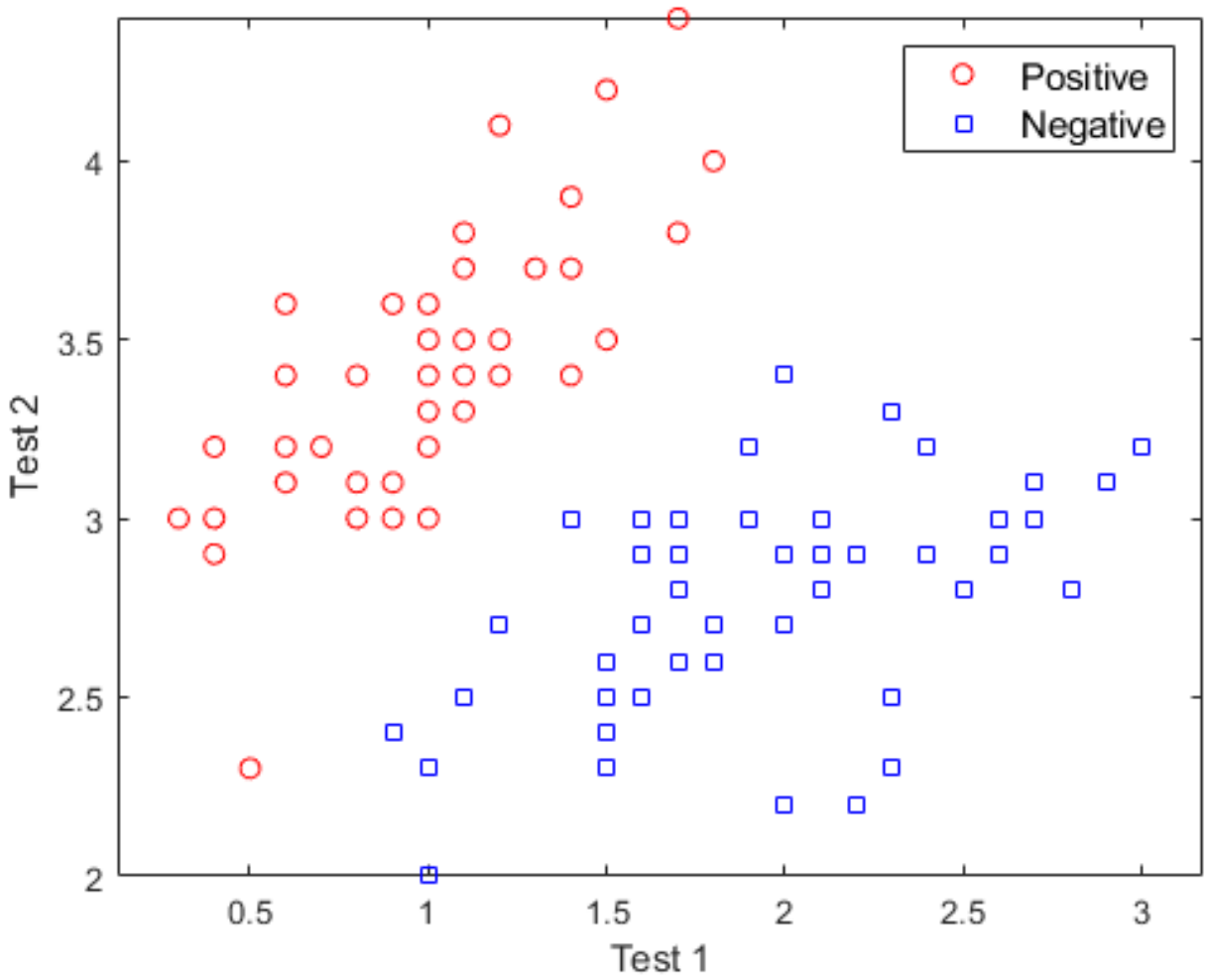


Each Test on its own is meaningless.

Combination of tests



Visual representation of the data



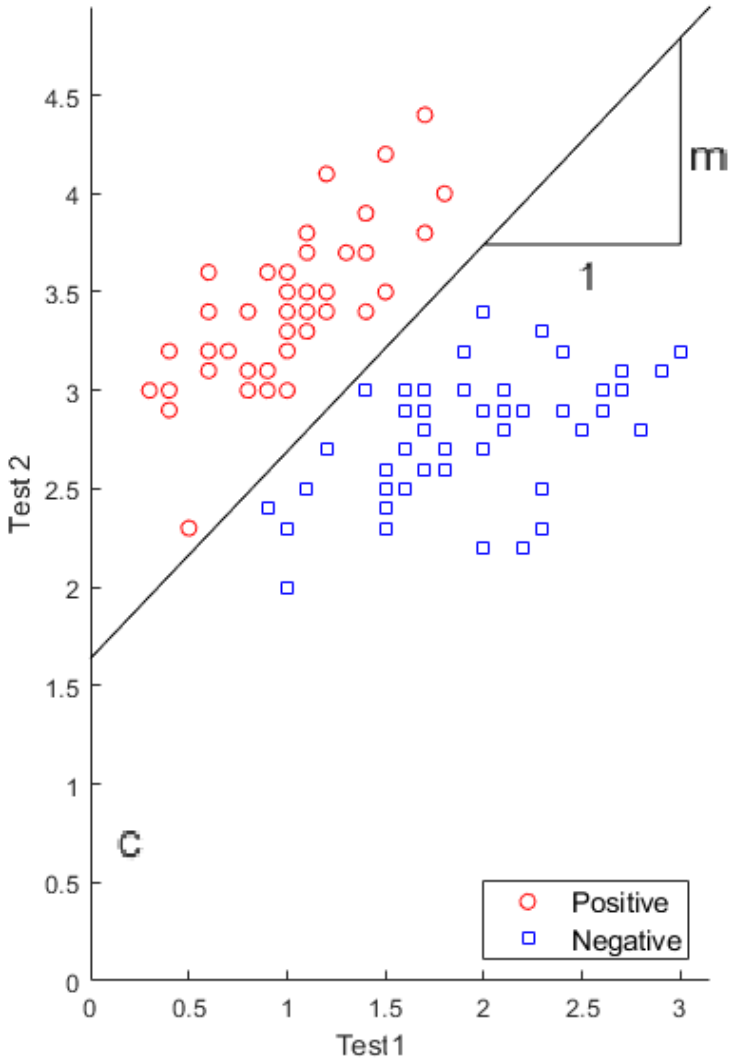
Line

$$y = mx + c.$$

or

$$y - mx - c = 0.$$

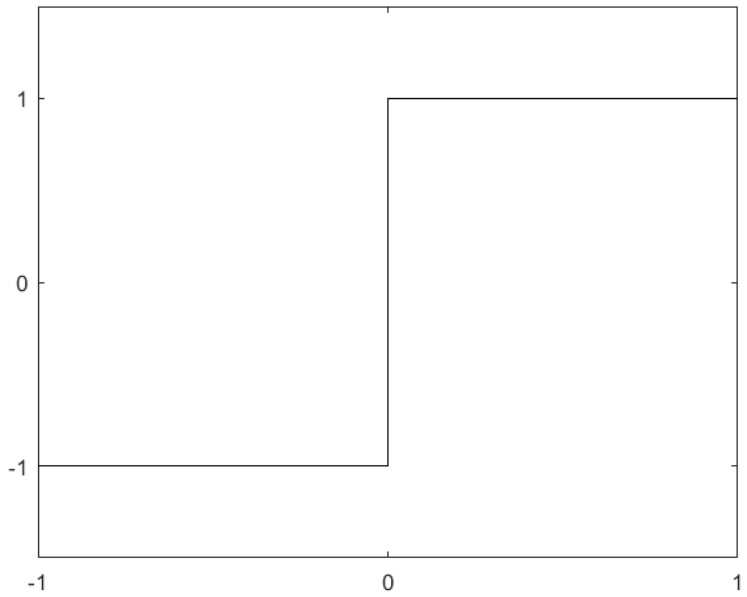
Line



Perceptron

Define the step function

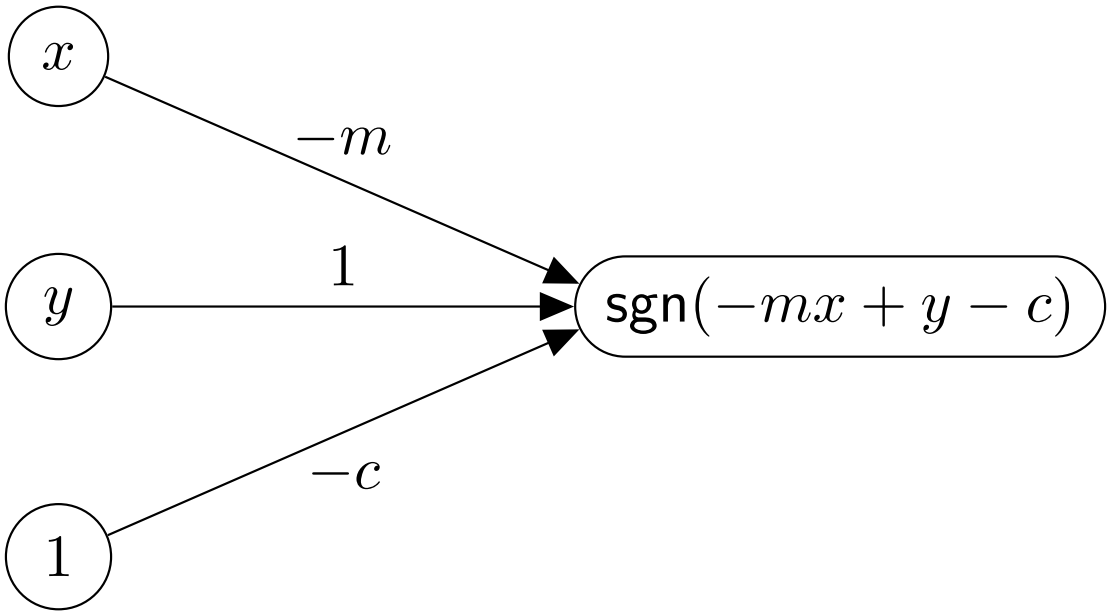
$$\text{sgn}(t) = \begin{cases} -1 & \text{if} \quad t < 0 \\ 0 & \text{if} \quad t = 0 \\ 1 & \text{if} \quad t > 0 \end{cases}$$



Then for sample $\mathbf{v}$

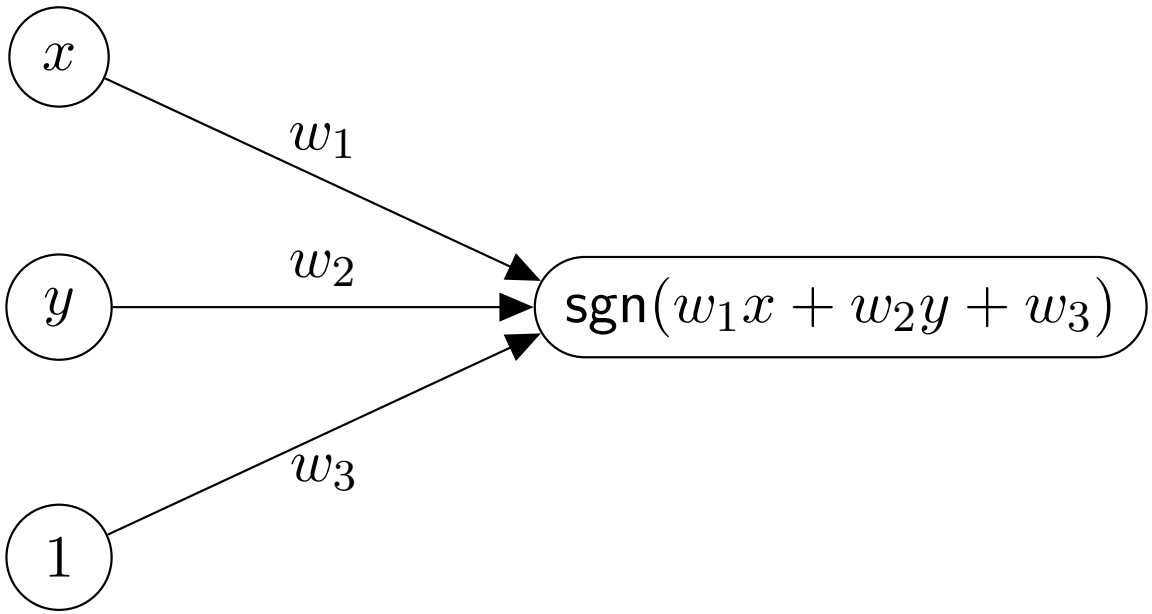
- $\text{sgn}(y - mx - c) = -1 \Rightarrow$ negative class,
- $\text{sgn}(y - mx - c) = 1 \quad \Rightarrow$ positive class.

Neural Network



Neural Network

More generally:

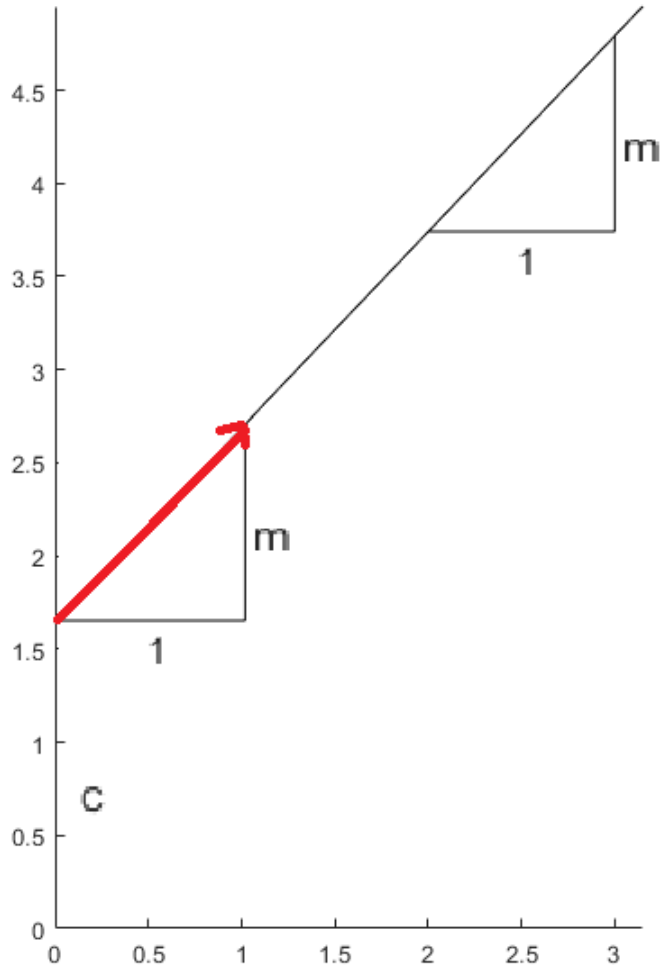


since scaling of the direction $\vec{d}$ and its perpendicular $\vec{d}^{\perp}$ does not matter (and it's inconvenient to keep a value fixed at 1).

Line

Geometric interpretation:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x \\ mx + c \end{pmatrix} = \begin{pmatrix} 1 \\ m \end{pmatrix} x + \begin{pmatrix} 0 \\ c \end{pmatrix}.$$



Line

General equation of a line: point and direction

$$\vec{p} + x\vec{d},$$

$$x \in (-\infty, +\infty).$$

Independent of dimension!

However, the line itself is not needed, we want to test on which side of the line a point lies.

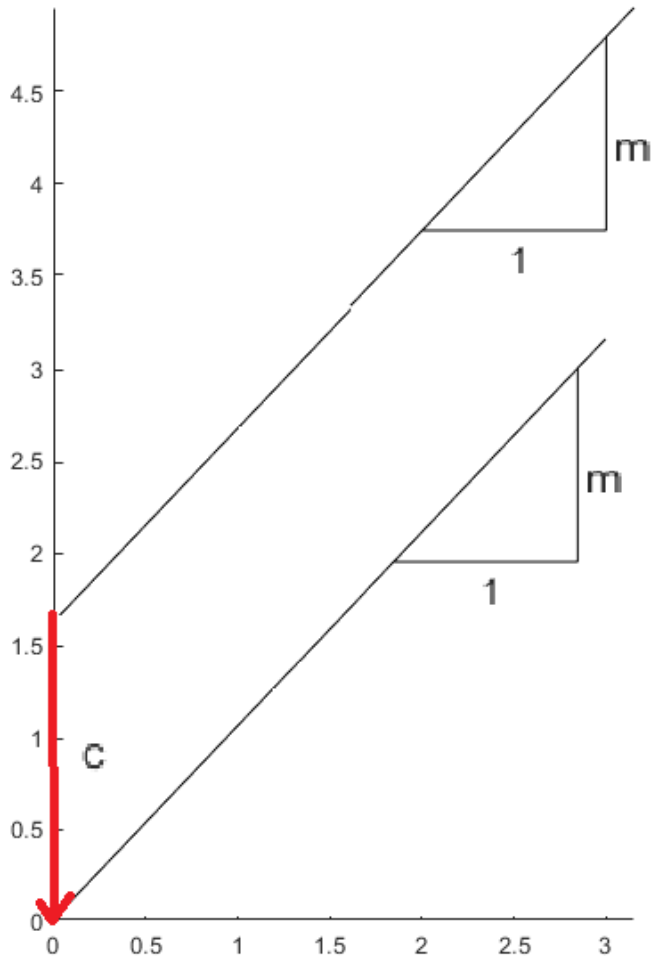
Line

$$y - mx - c = 0.$$

$$\begin{pmatrix} -m & 1 \end{pmatrix} \begin{pmatrix} x - 0 \\ y - c \end{pmatrix} = 0.$$

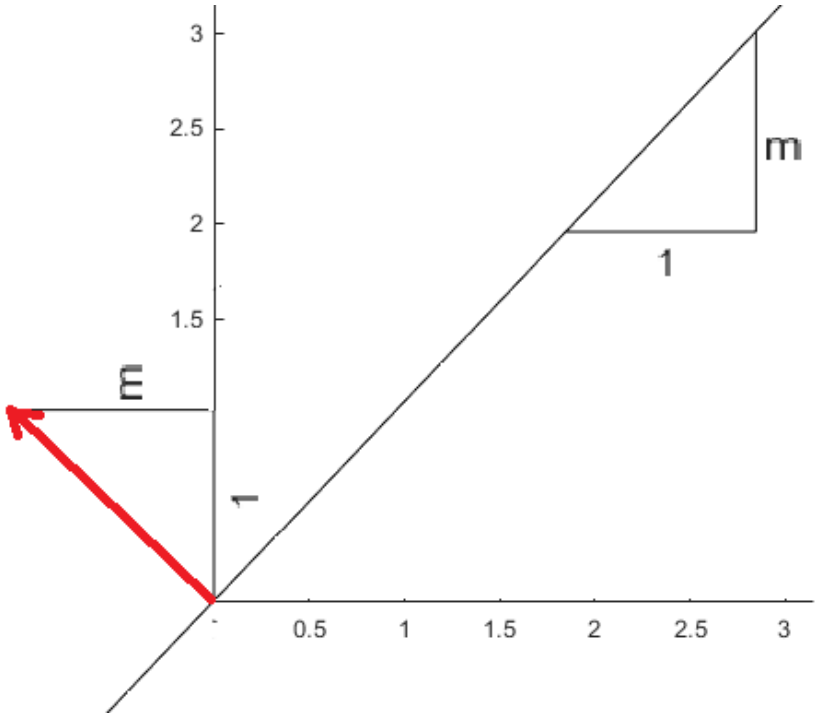
Line

$$\begin{pmatrix} x-0 \\ y-c \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} - \vec{p}$$



Line

$$\begin{pmatrix} -m & 1 \end{pmatrix} \begin{pmatrix} x-0 \\ y-c \end{pmatrix} = \vec{d}^{\perp} \left[\begin{pmatrix} x \\ y \end{pmatrix} - \vec{p} \right] = 0.$$



Perceptron

- The Perceptron initializes $w_1 = w_2 = w_3 = 0$ and updates them with each new training sample (x_n, y_n) by

$$\begin{aligned} w_1^{\text{new}} &= w_1^{\text{old}} + \frac{\alpha}{2}\left(c_n - \text{sgn}(w_1 x_n + w_2 y_n + w_3)\right) x_n, \\ w_2^{\text{new}} &= w_2^{\text{old}} + \frac{\alpha}{2}\left(c_n - \text{sgn}(w_1 x_n + w_2 y_n + w_3)\right) y_n, \\ w_3^{\text{new}} &= w_3^{\text{old}} + \frac{\alpha}{2}\left(c_n - \text{sgn}(w_1 x_n + w_2 y_n + w_3)\right). \end{aligned}$$

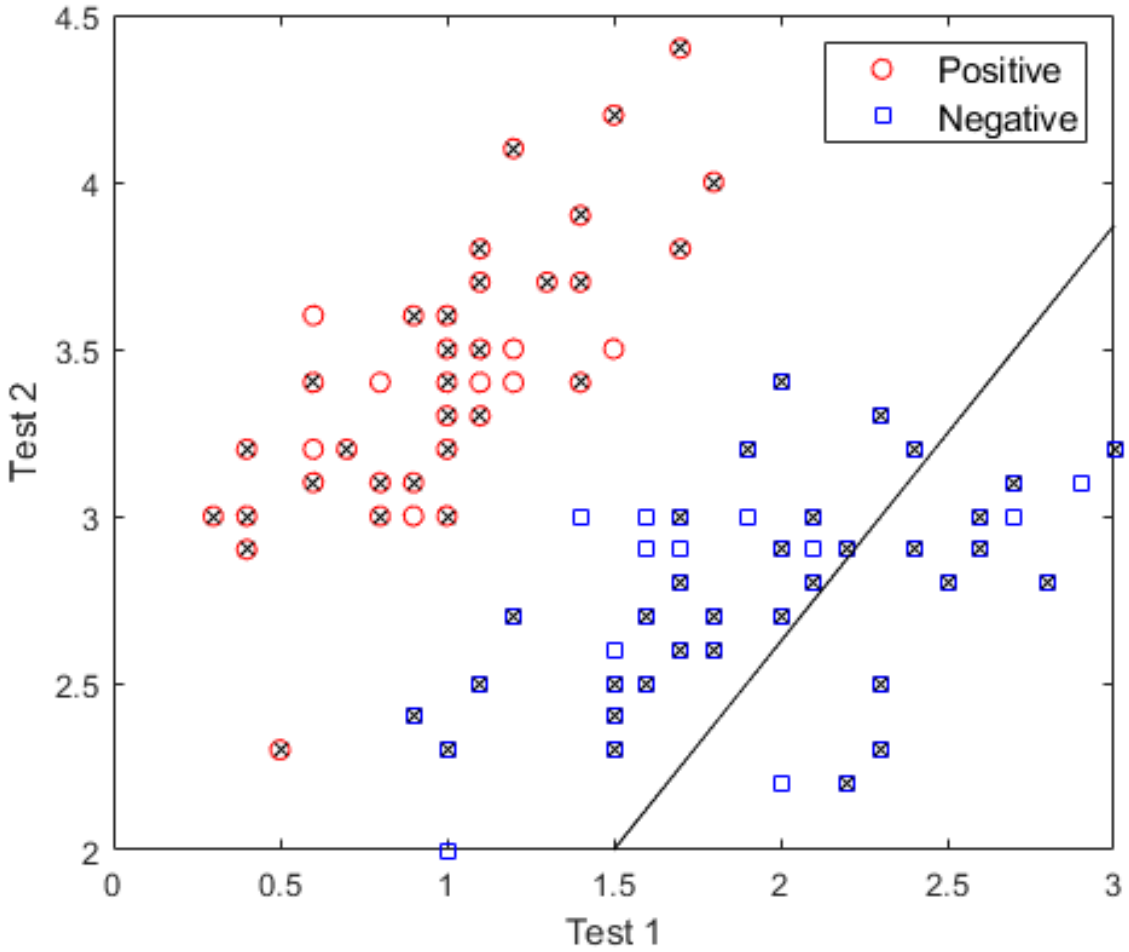
  where c_n is the class label of (x_n, y_n), that is $c_n = -1$, if (x_n, y_n) is a negative sample, and $c_n = 1$, if (x_n, y_n) is a positive sample.
- $0 < \alpha \leq 1$ is the learning rate. If the learning rate is chosen too big any changes are too radical and oscillations occur.

Perceptron

The geometric effect is:

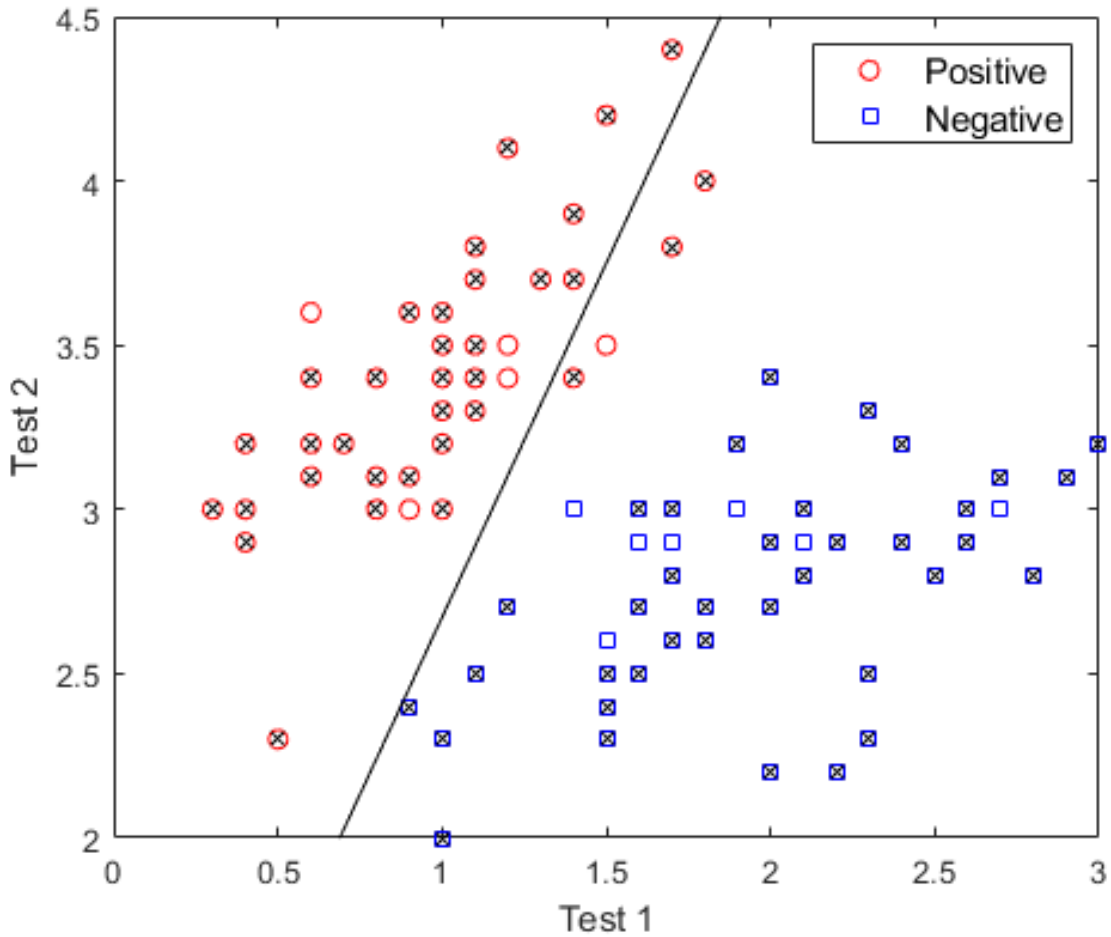
- Moving the separation line.
- In particular, moving $\vec{p}$ and $\vec{d}$.
- Or equivalently, moving up and down the y-axis and pulling the handle $\vec{d}^{\perp}$ clockwise and anti-clockwise.

Perceptron



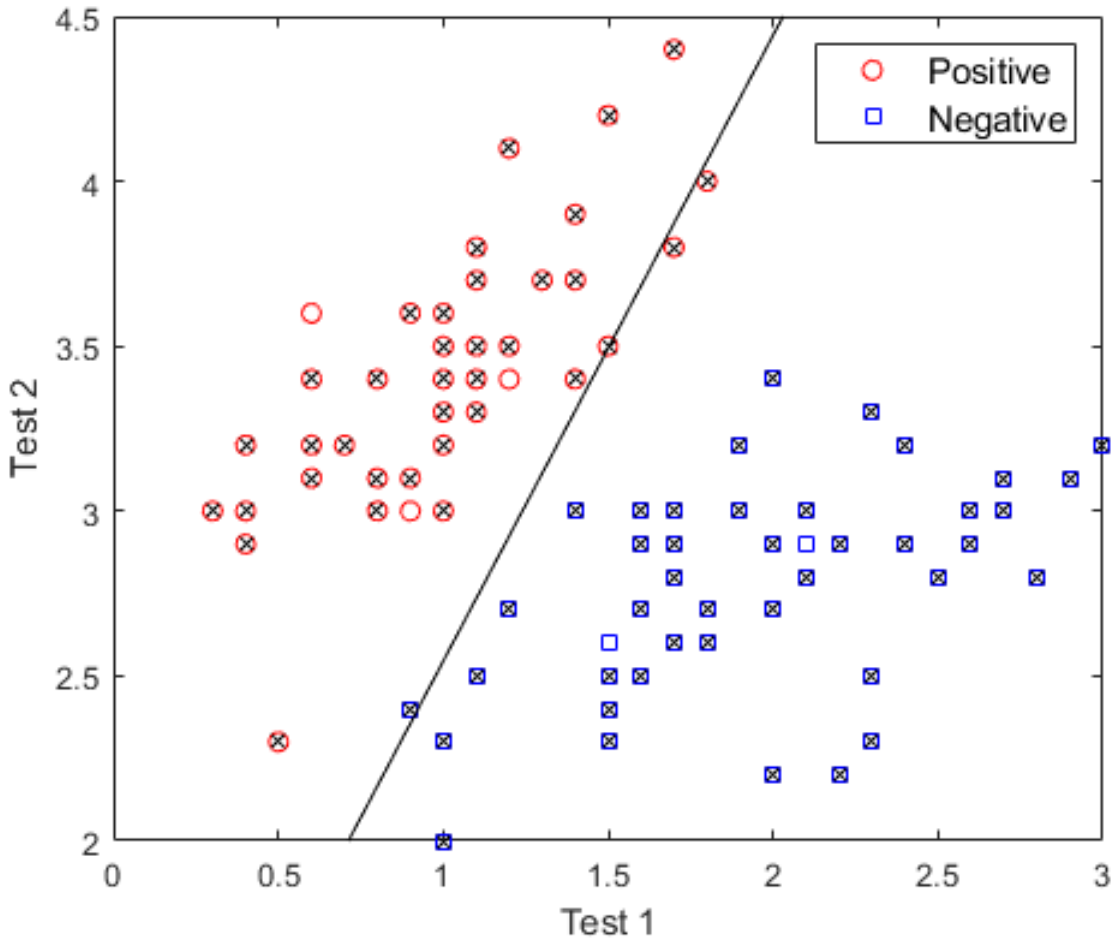
70% of data used

Perceptron



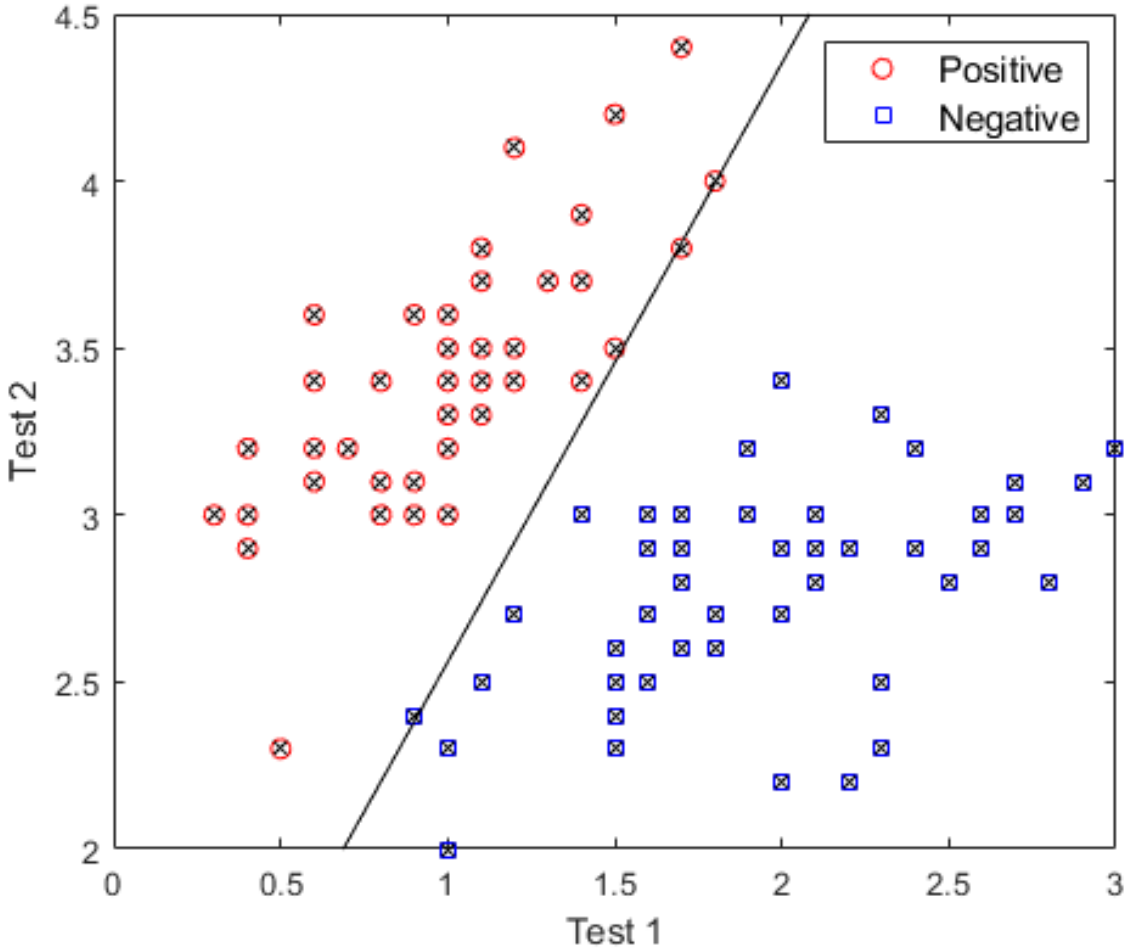
80% of data used

Perceptron



90% of data used

Perceptron

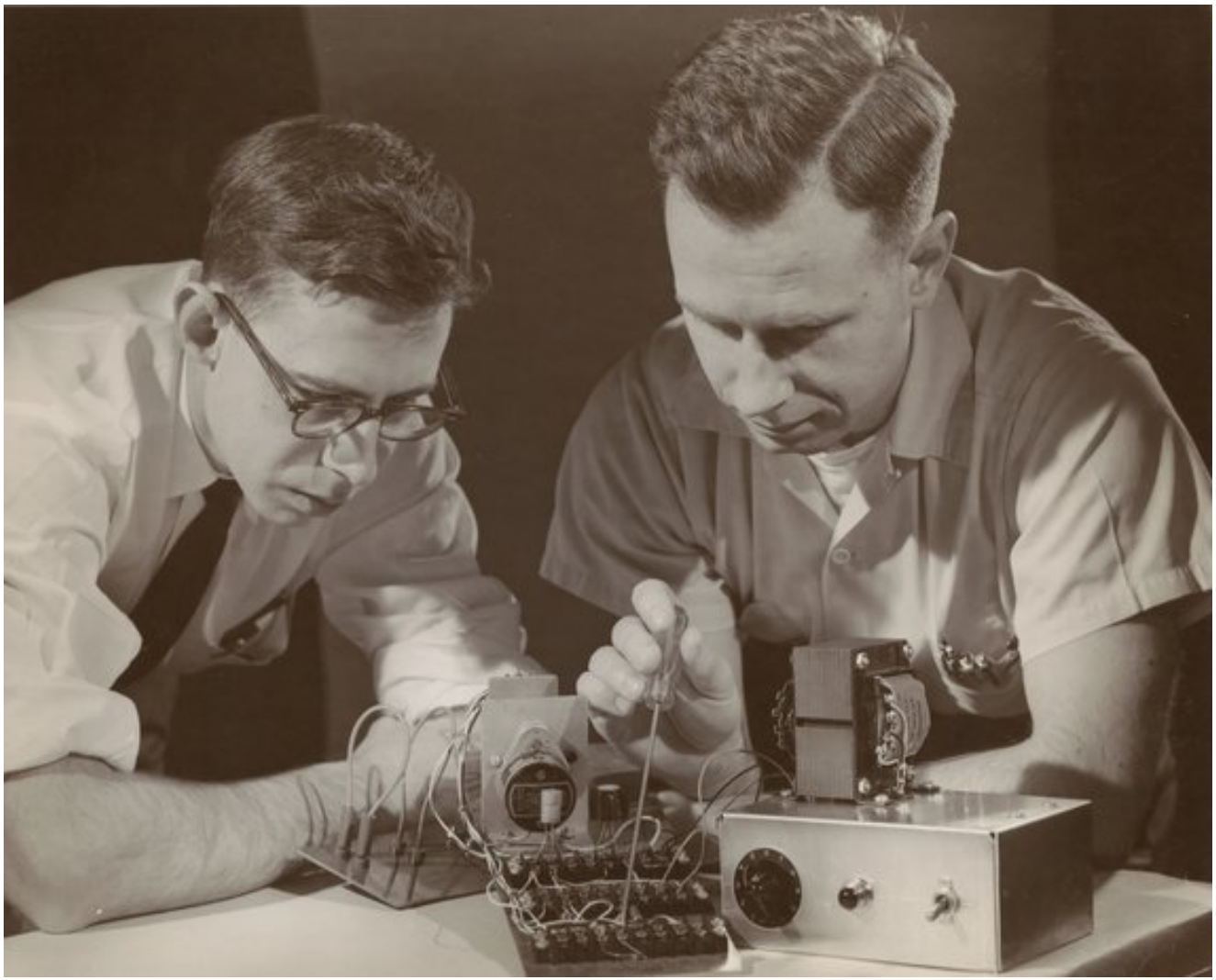


complete data used

Perceptron

- invented 1957 by Rosenblatt,
- built in 1958 as a physical machine for image recognition,
- 400 photo cells,
- weights as potentiometers which were updated by electric motors.

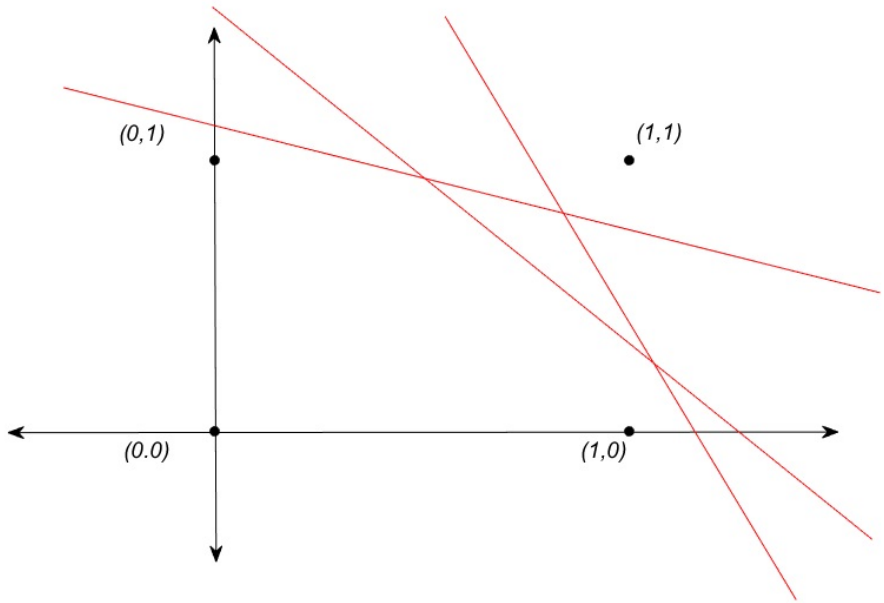
Perceptron



Neural Networks - AND

The perceptron is capable of implementing the logical AND.

x	y	AND
0	0	0
0	1	0
1	0	0
1	1	1



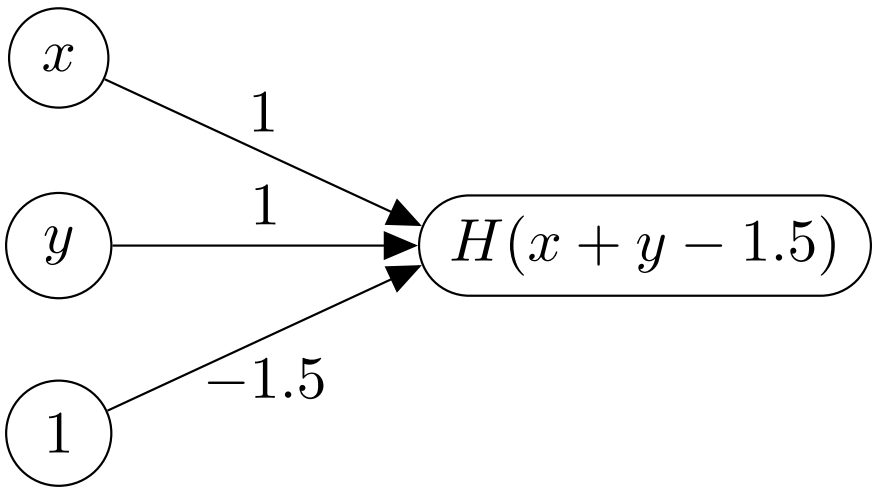
Each line is described by $w_1 x + w_2 y + w_3 = 0$. For any of the lines, points to the right should result in output 1.

Neural Networks - AND

Using the Heaviside step function defined by

$$H(x) = \begin{cases} 0 & \text{if} \quad x \leq 0 \\ 1 & \text{if} \quad x > 0 \end{cases},$$

the resulting neural network might be:

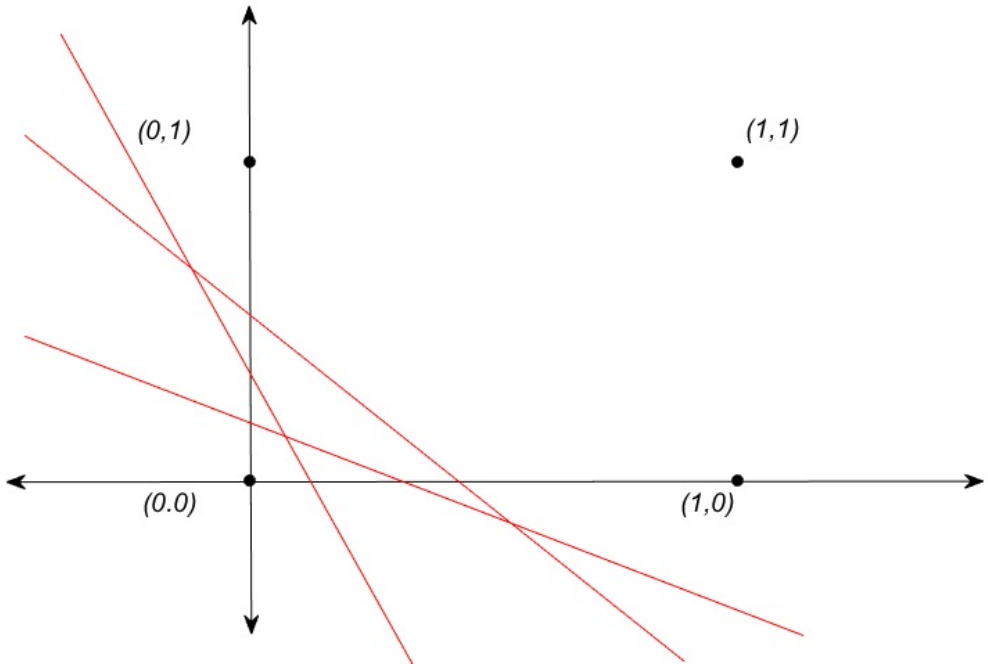


Note, the learning process might have arrived at different choices for w_1, w_2 and w_3.

Neural Networks - OR

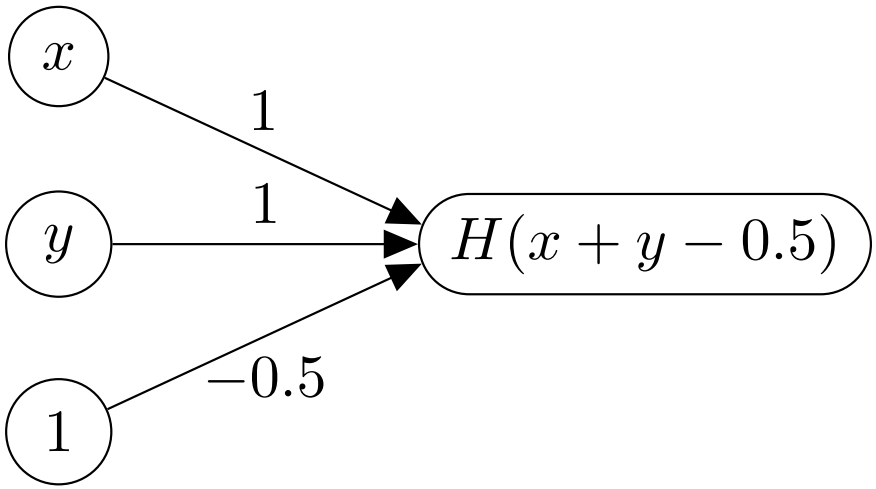
The perceptron is capable of implementing the logical OR.

x	y	OR
0	0	0
0	1	1
1	0	1
1	1	1



Neural Networks - OR

Using the Heaviside step function again the resultant neural network might be

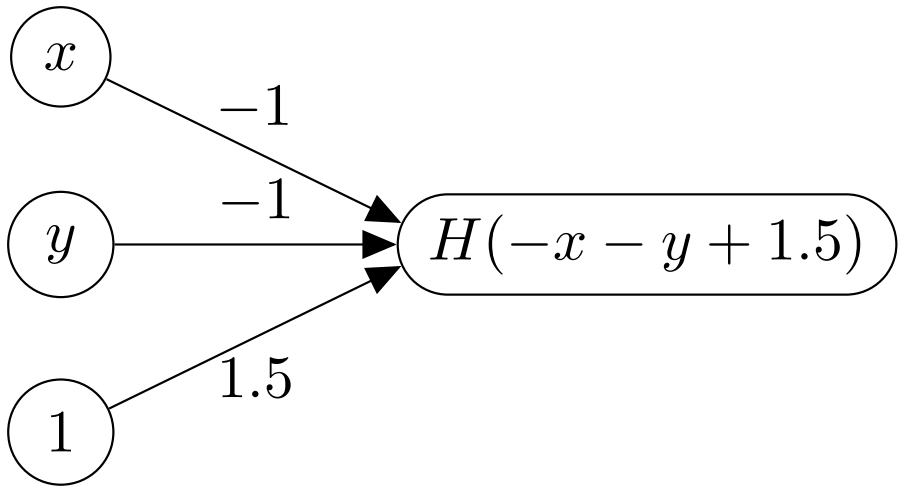


Note, the learning process might have arrived at different choices for w_1, w_2 and w_3.

Neural Networks - NAND

The perceptron is capable of implementing the logical AND NOT = NAND, for example:

x	y	NAND
0	0	1
0	1	1
1	0	1
1	1	0

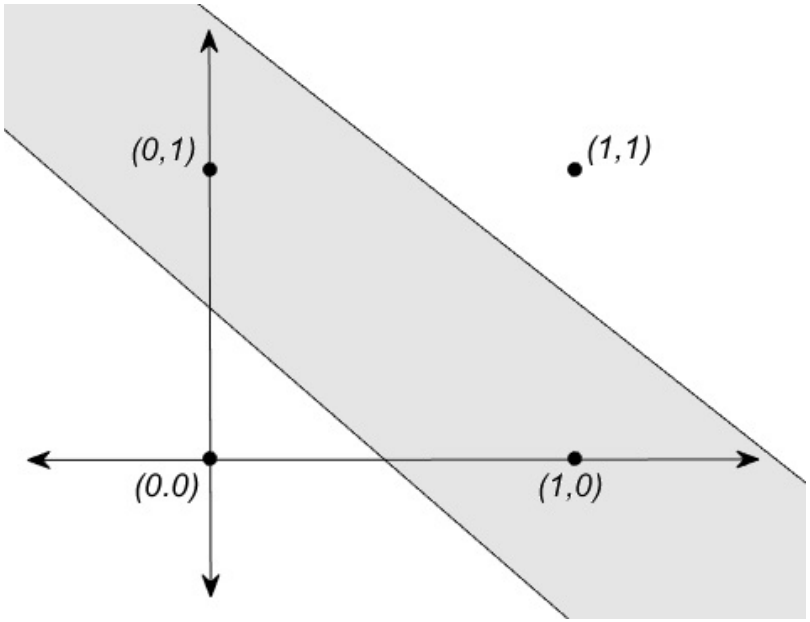


Note, the learning process might have arrived at different choices for w_1, w_2 and w_3.

Neural Networks - XOR

The perceptron is *not* capable of implementing the exclusive OR = XOR (one or the other, but not both).

x	y	XOR
0	0	0
0	1	1
1	0	1
1	1	0

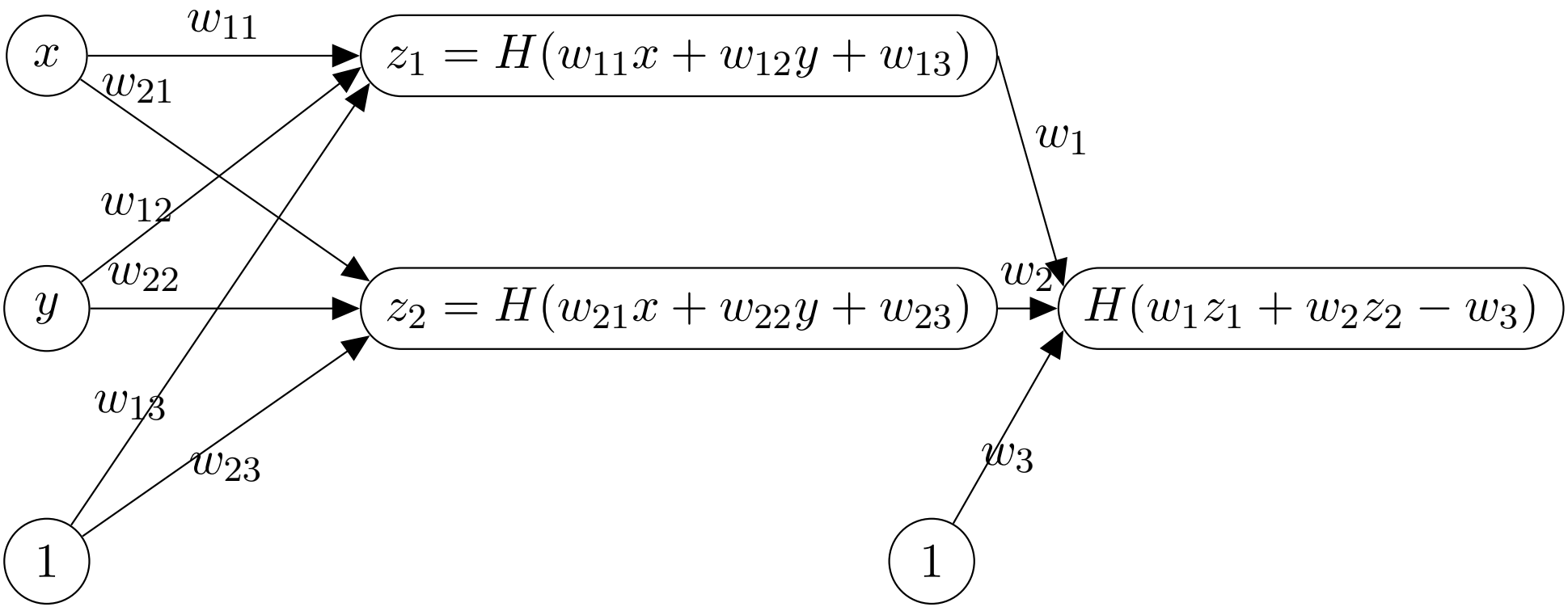


XOR can be implemented by introducing a *hidden* set of neurons.

Neural Networks - XOR

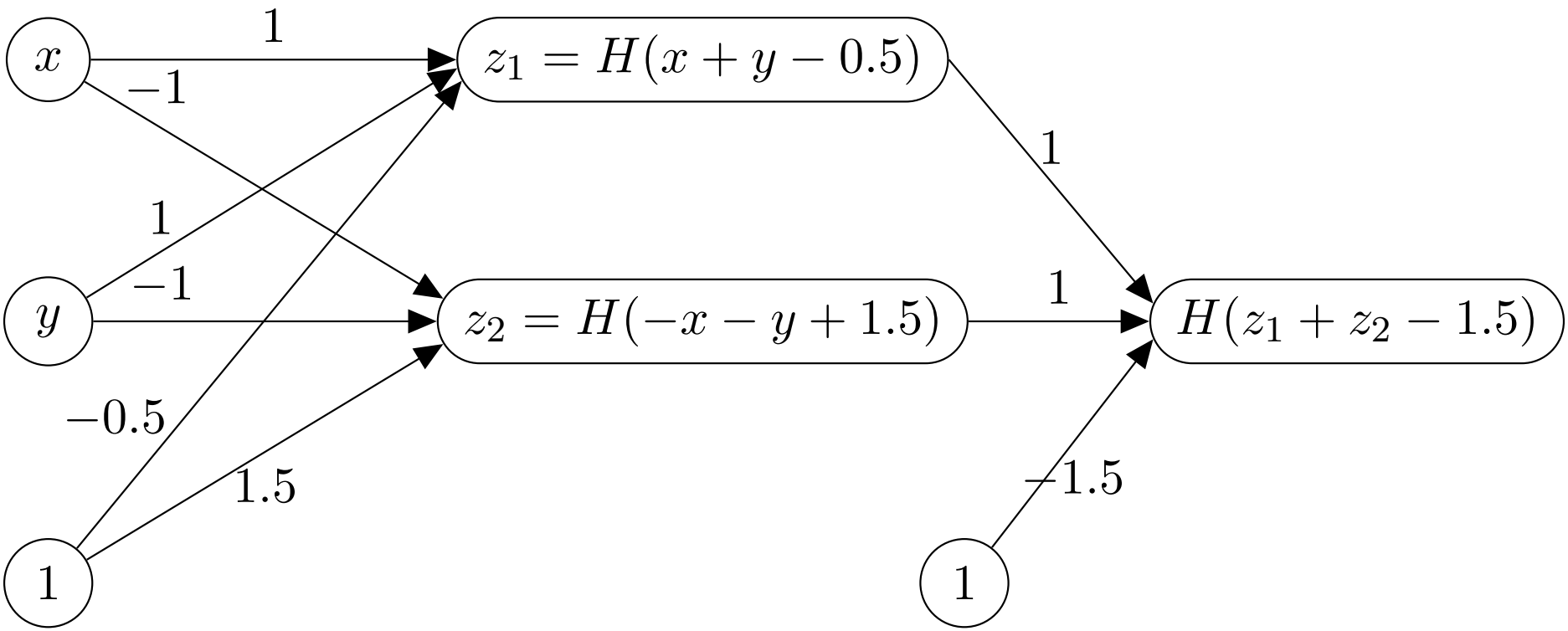
- Two lines are necessary. Points to the right of one line and to the left of the other line shall output 1.
- Or in other words: Points to the right of one line AND NOT to the right of the other line shall output 1.

Neural Networks - XOR



Neural Networks - XOR

For example:



Neural Networks

- Any region can be described by hyperplanes.
- Any function can be built.

4 V's of Big Data

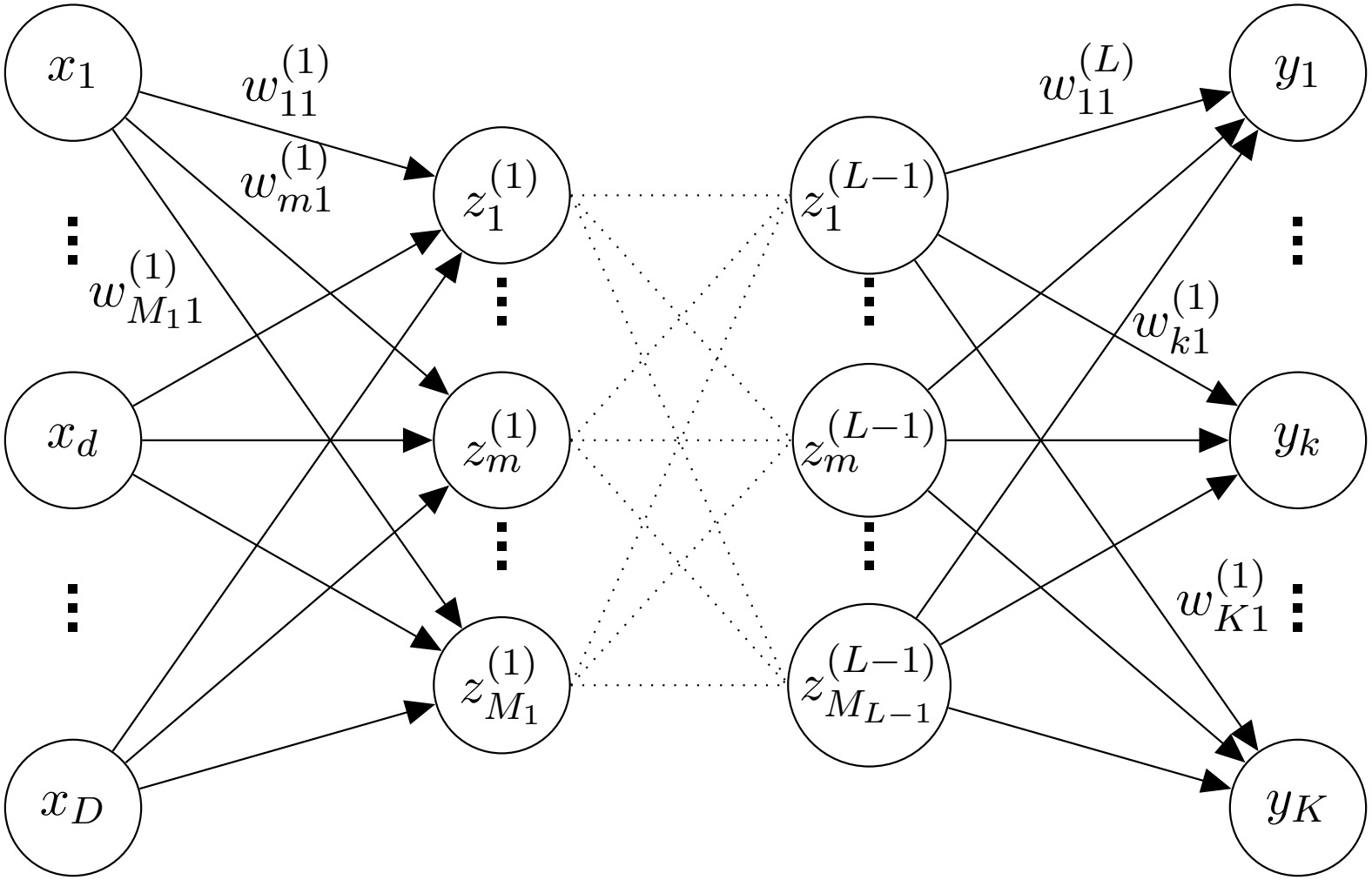
4 V's of Big Data

- *Veracity*: Sufficiently many samples, where the features have reliable values.
- *Volume*: Large volume of data necessary to cope with missing or corrupted parts of the data, but a challenge in itself.
- *Velocity*: New data arrives at extraordinary velocity.
- *Variety*: Many different sources.

Neural Networks

- Given data pairs $(\mathbf{x}_1, \mathbf{t}_1), \ldots, (\mathbf{x}_N, \mathbf{t}_N)$.
- $\mathbf{x}_n \in \mathbb{R}^D$ and each component of $\mathbf{x}_n$ is a feature.
- $\mathbf{t}_n \in \mathbb{R}^K$.
- Each component of $\mathbf{x}_n$ is passed to one input neuron.
- The results of the output neurons are compared to $\mathbf{t}_n$.
- Several sets of hidden neurons with latent variables $\mathbf{z}$.
- Synapses connecting one set of neurons to another are a layer.

Neural Networks



Propagation and Activation

- The *propagation function* gives the *activation*

$$a_j^{(l)} = \sum_{i=1}^{M_{l-1}} w_{ji}^{(l)} z_i^{(l-1)} = \mathbf{w}_j^{(l)^T} \mathbf{z}^{(l-1)}.$$

- The *activation function* (*transfer function*) h_l generates the latent variable

$$z_j^{(l)} = h_l(a_j^{(l)}) = h_l\left(\mathbf{w}_j^{(l)^T} \mathbf{z}^{(l-1)}\right).$$

- In the first layer $a_j^{(1)} = \mathbf{w}_j^{(1)^T} \mathbf{x}$ and $z_j^{(1)} = h_1\left(\mathbf{w}_j^{(1)^T} \mathbf{x}\right)$.
- In the last layer $y_k = h_L(a_k^{(L)}) = h_L\left(\mathbf{w}_k^{(L)^T} \mathbf{z}^{(L-1)}\right)$.

Activation Functions

The task determines the activation function in the output layer.

- For *regression* problems, it is the identity, i.e. the *linear* activation function

$$h_L(x) = x.$$

- For *binary classification*, $K = 1$ and t_n is either 0 or 1 depending on whether the sample with features $\mathbf{x}_n$ belongs to the negative or positive class. The activation function is the *logistic sigmoid*

$$h_L(x) = (1 + \exp(-x))^{-1}.$$

Activation Functions

- If there are K *separate, binary classification* tasks for the same data, then $\mathbf{t}_n$ is a vector of zeros and ones. The k^{th} component indicates the class membership of the k^{th} task. There are K output neurons, each using the logistic sigmoid function.
- For *multiple classes*, $\mathbf{t}_n$ is a *1-of-K representation*. The output activation function is the *softmax* function. It maps the K-dimensional vector of activations $\mathbf{a} = (a_1, \ldots, a_K)^T$ to a K-dimensional vector $\boldsymbol{\sigma}(\mathbf{a})$ with the j^{th} entry of $\boldsymbol{\sigma}(\mathbf{a})$ being

$$\boldsymbol{\sigma}(\mathbf{a})_j = \frac{\exp(a_j)}{\sum_{k=1}^{K} \exp(a_k)}.$$

Neural Networks

- The neural network is a function mapping the input $\mathbf{x} = (x_1, \ldots, x_D)^T$ to the output $\mathbf{y} = (y_1, \ldots, y_K)^T$.
- The input passing through the network is known as *forward propagation*.
- Cycles are not allowed.
- It is strictly *feed-forward*.

Synapses

- Not all possible synapses need to be present, e.g convolutional neural networks.
- Different to the weight of a synapse becoming zero during the learning process.
- A *skip-layer* connection stretchs over two layers.
- The larger the number of synapses, the more training data is necessary.

Training

- Training is adjusting the weights until the output is deemed good enough.
- Let $\mathbf{w}$ denote all weights of the network.
- The output $\mathbf{y}$ depends on the input $\mathbf{x}$ and $\mathbf{w}$: $\mathbf{y}(\mathbf{x}, \mathbf{w})$.
- For *regression* $\mathbf{t}_n$ are drawn from a normal distribution with the mean $\mathbf{y}(\mathbf{x}, \mathbf{w})$ and variance $\sigma^2\mathbf{I}$.
- The data likelihood is

$$\mathcal{L} = \prod_{n=1}^{N} (2\pi\sigma^2)^{-K/2} \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{t}_n - \mathbf{y}(\mathbf{x}_n, \mathbf{w})\|^2\right).$$

Error functions

- Minimizing the negative logarithm

$$-\log \mathcal{L} = \frac{NK}{2} \log \sigma^2 + \frac{NK}{2} \log(2\pi) + \frac{1}{2\sigma^2} \sum_{n=1}^{N} \|\mathbf{t}_n - \mathbf{y}(\mathbf{x}_n, \mathbf{w})\|^2.$$

- Minimizing the *error sum of squares (ESS)*

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \|\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n\|^2.$$

Error functions

- For *binary classification*, $y(\mathbf{x}_n, \mathbf{w})$ is interpreted as the probability of belonging to the positive class.
- The likelihood is

$$\mathcal{L} = \prod_{n=1}^{N} y(\mathbf{x}_n, \mathbf{w})^{t_n} \left(1 - y(\mathbf{x}_n, \mathbf{w})\right)^{1-t_n} .$$

- The negative logarithm is the *cross-entropy* error function

$$E(\mathbf{w}) = -\sum_{n=1}^{N} t_n \log y(\mathbf{x}_n, \mathbf{w}) + (1 - t_n) \log \left(1 - y(\mathbf{x}_n, \mathbf{w})\right) .$$

- If the network tackles K *separate, binary classification* tasks on the same data, then the error function is

$$E(\mathbf{w}) = -\sum_{k=1}^{K} \sum_{n=1}^{N} t_{n,k} \log y_k(\mathbf{x}_n, \mathbf{w}) + (1 - t_{n,k}) \log \left(1 - y_k(\mathbf{x}_n, \mathbf{w})\right) .$$

Error functions

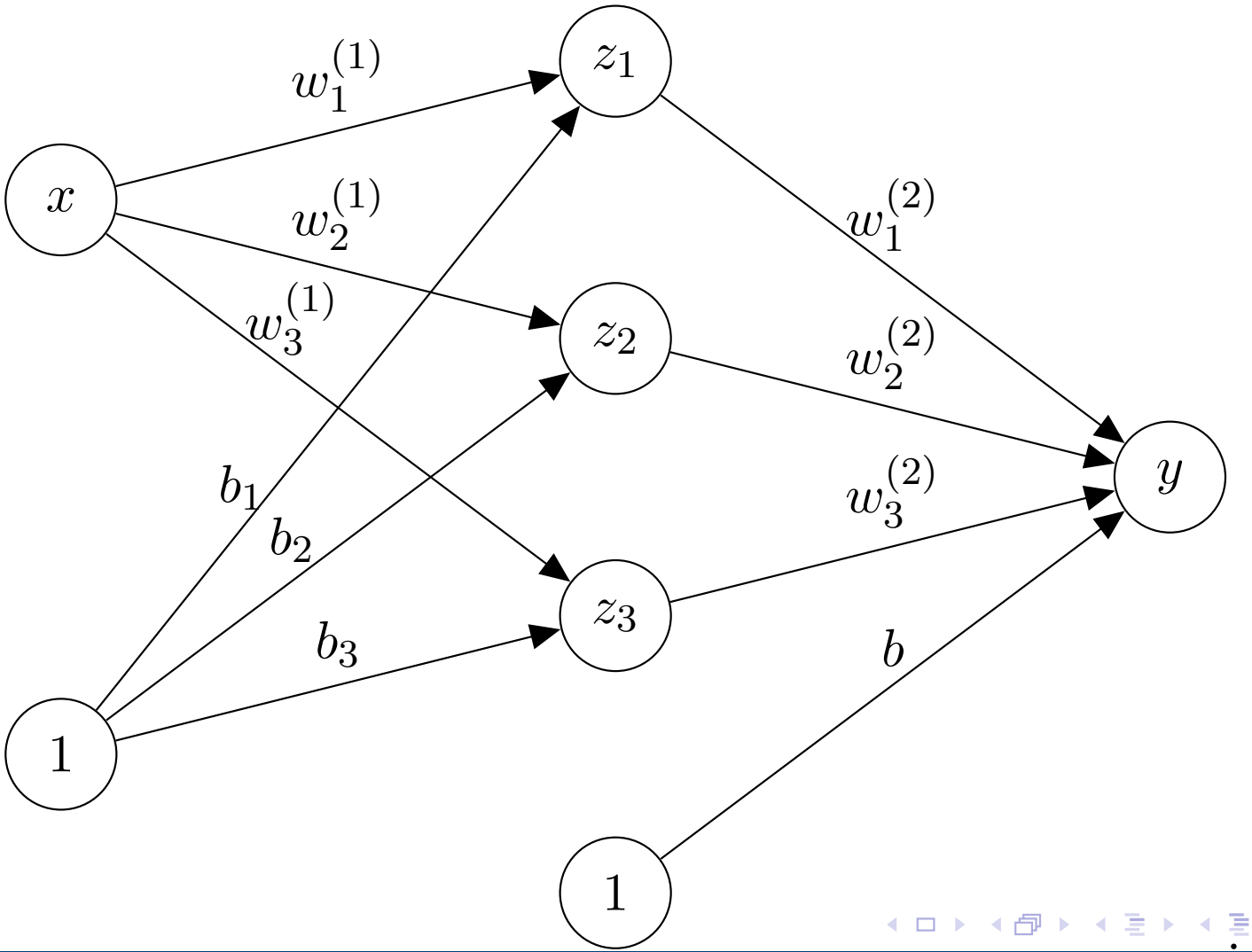
- For K *multiple classes*, $y_k(\mathbf{x}_n, \mathbf{w})$ is taken as the probability of belonging to the k^{th} class and $\mathbf{t}_n$ is a 1-of-K representation.
- The likelihood is

$$\mathcal{L} = \prod_{n=1}^{N} \prod_{k=1}^{K} y_k(\mathbf{x}_n, \mathbf{w})^{t_{n,k}}.$$

- The negative logarithm is

$$E(\mathbf{w}) = -\sum_{k=1}^{K} \sum_{n=1}^{N} t_{n,k} \log y_k(\mathbf{x}_n, \mathbf{w}).$$

Example



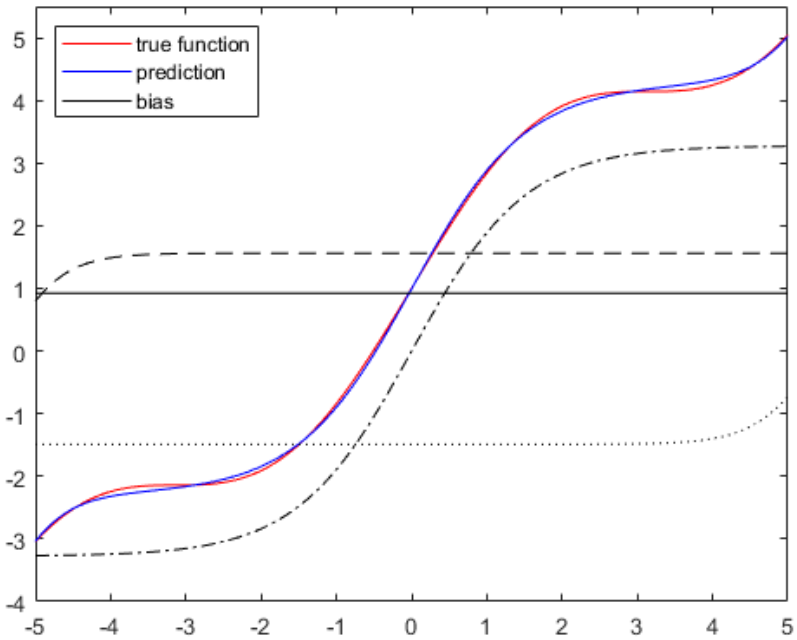
Example

- The activation function in the first layer is the *tan-sigmoid*

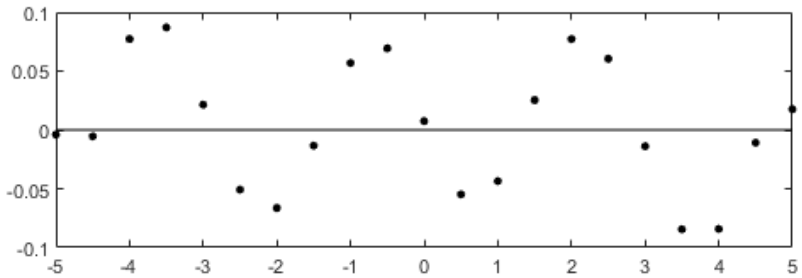
$$h_1(x) = \frac{2}{1 + \exp(-2x)} - 1,$$

- In the second layer, it is the linear activation function.
- Dummy neurons, which always take the value one, are used for a shift, known as the *bias*.
- Do not underestimate the bias.

Example

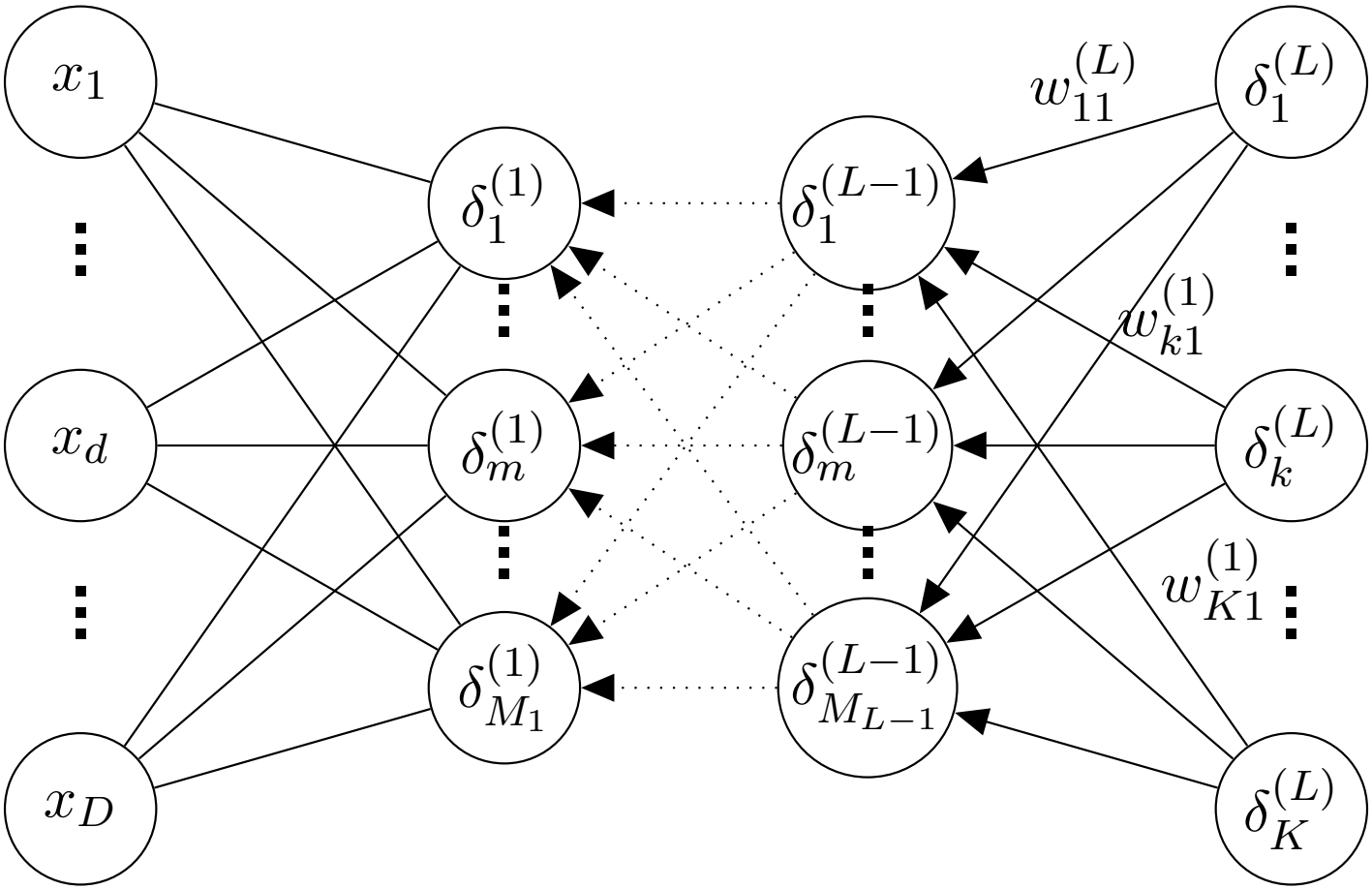


True function, prediction, bias and constructed basis functions.



Error at training points.

Error Backpropagation



$$\delta_k^{(L)} = y_k(\mathbf{x}_n, \mathbf{w}) - t_{nk},$$

Error Backpropagation

- A necessary condition for a minimum is that the *gradient* $\nabla E(\mathbf{w})$ vanishes, where the gradient is the vector of the derivatives of $E(\mathbf{w})$ with respect to each $w_{ji}^{(l)}$.
- Gradient evaluated for a particular $\mathbf{w}$ points in the direction of the steepest ascent.
- Iterative minimization:

$$\mathbf{w}^{\text{new}} = \mathbf{w} - \eta \nabla E(\mathbf{w}),$$

where the step size $\eta > 0$ is called the *learning rate*.

Error Backpropagation

- *Steepest descent*, *conjugate gradients* or *quasi-Newton methods*.
- All error functions have the form

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w}).$$

- Since the derivative of a sum is the sum of derivatives

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N} \nabla E_n(\mathbf{w}).$$

Error Backpropagation

Update possibilities:

- Calculating $\nabla E_n(\mathbf{w})$ for all n and using $\nabla E(\mathbf{w})$ in the chosen optimization method, are known as *batch methods*.
- When $\mathbf{w}$ is updated using only a subset $\mathcal{S}$ of training data, that is $\sum_{\mathcal{S}} \nabla E(\mathbf{w})$ is used in the optimization method, then these are *mini-batch methods*
- If the optimization method is gradient descent, this is known as *mini-batch gradient descent*.
- If only one $\nabla E_n(\mathbf{w})$ is used, when updating $\mathbf{w}$, the method is *online*, *sequential* or *stochastic*. The algorithm either cycles through the training data or selects training samples randomly with replacement.

Chain Rule

Leibniz 1675

- In regression, the outputs are the same as the activations:

$$E_n(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{K}(y_k(\mathbf{x}_n,\mathbf{w}) - t_{nk})^2 = \frac{1}{2}\sum_{k=1}^{K}\left(a_k^{(L)}(\mathbf{x}_n,\mathbf{w}) - t_{nk}\right)^2,$$

- The derivative with respect to $w_{ji}^{(l)}$ is

$$\begin{aligned}\frac{\partial}{\partial w_{ji}^{(l)}}E_n(\mathbf{w}) &= \sum_{k=1}^{K}\left(a_k^{(L)}(\mathbf{x}_n,\mathbf{w}) - t_{nk}\right)\frac{\partial}{\partial w_{ji}^{(l)}}a_k^{(L)}(\mathbf{x}_n,\mathbf{w}) \\ &= \sum_{k=1}^{K}(y_k(\mathbf{x}_n,\mathbf{w}) - t_{nk})\frac{\partial}{\partial w_{ji}^{(l)}}a_k^{(L)}(\mathbf{x}_n,\mathbf{w}).\end{aligned}$$

Chain Rule

For binary classification

$$
\begin{aligned}
E_n(\mathbf{w}) &= -t_n \log \frac{1}{1+\exp(-a^{(L)}(\mathbf{x}_n, \mathbf{w}))} \\
&\quad -(1-t_n)\log\left(1 - \frac{1}{1+\exp(-a^{(L)}(\mathbf{x}_n, \mathbf{w}))}\right) \\
&= t_n \log\left(1+\exp(-a^{(L)}(\mathbf{x}_n, \mathbf{w}))\right) \\
&\quad -(1-t_n)\log\frac{\exp(-a^{(L)}(\mathbf{x}_n, \mathbf{w}))}{1+\exp(-a^{(L)}(\mathbf{x}_n, \mathbf{w}))} \\
&= \log\left(1+\exp(-a^{(L)}(\mathbf{x}_n, \mathbf{w}))\right) + (1-t_n)a^{(L)}(\mathbf{x}_n, \mathbf{w}).
\end{aligned}
$$

Chain Rule

Differentiating with respect to $w_{ji}^{(l)}$ gives

$$\begin{aligned}
\frac{\partial}{\partial w_{ji}^{(l)}} E_n(\mathbf{w}) &= \frac{\exp(-a^{(L)}(\mathbf{x}_n, \mathbf{w}))}{1+\exp(-a^{(L)}(\mathbf{x}_n, \mathbf{w}))} \frac{\partial}{\partial w_{ji}^{(l)}} (-a^{(L)}(\mathbf{x}_n, \mathbf{w})) \\
&\quad + (1-t_n) \frac{\partial}{\partial w_{ji}^{(l)}} a^{(L)}(\mathbf{x}_n, \mathbf{w}) \\
&= \left(\frac{1}{1+\exp(-a^{(L)}(\mathbf{x}_n, \mathbf{w}))} - 1 \right) \frac{\partial}{\partial w_{ji}^{(l)}} a^{(L)}(\mathbf{x}_n, \mathbf{w}) \\
&\quad + (1-t_n) \frac{\partial}{\partial w_{ji}^{(l)}} a^{(L)}(\mathbf{x}_n, \mathbf{w}) \\
&= (y(\mathbf{x}_n, \mathbf{w}) - t_n) \frac{\partial}{\partial w_{ji}^{(l)}} a^{(L)}(\mathbf{x}_n, \mathbf{w})
\end{aligned}$$

Chain Rule

Classifying K classes, let k_n be the index for which $t_{n,k} = 1$.

$$E_n(\mathbf{w}) = -\log \frac{\exp(a_{k_n}^{(L)})}{\sum_{s=1}^{K} \exp(a_s^{(L)})} = \log \left(\sum_{s=1}^{K} \exp(a_s^{(L)}) \right) - a_{k_n}^{(L)}.$$

The derivative with respect to $w_{ji}^{(l)}$ is

$$\begin{aligned}
\frac{\partial}{\partial w_{ji}^{(l)}} E_n(\mathbf{w}) &= \frac{\sum_{r=1}^{K} \exp(a_r^{(L)}) \frac{\partial}{\partial w_{ji}^{(l)}} a_r^{(L)}}{\sum_{s=1}^{K} \exp(a_s^{(L)})} - \frac{\partial}{\partial w_{ji}^{(l)}} a_{k_n}^{(L)} \\
&= \sum_{r=1}^{K} y_r(\mathbf{x}_n, \mathbf{w}) \frac{\partial}{\partial w_{ji}^{(l)}} a_r^{(L)} - t_{n,r} \frac{\partial}{\partial w_{ji}^{(l)}} a_r^{(L)} \\
&= \sum_{r=1}^{K} \left(y_r(\mathbf{x}_n, \mathbf{w}) - t_{n,r} \right) \frac{\partial}{\partial w_{ji}^{(l)}} a_r^{(L)},
\end{aligned}$$

Canonical Link

That the derivatives of the n^{th} contribution of all error functions result in

$$\frac{\partial}{\partial w_{ji}^{(l)}} E_n(\mathbf{w}) = \sum_{k=1}^{K} \left(y_k(\mathbf{x}_n, \mathbf{w}) - t_{nk}\right) \frac{\partial}{\partial w_{ji}^{(l)}} a_k^{(L)}(\mathbf{x}_n, \mathbf{w}).$$

is known as the *canonical link* between the output activation functions and the error function.

Chain rule

Since

$$a_m^{(s)}(\mathbf{x}_n, \mathbf{w}) = \sum_{r=1}^{M_{s-1}} w_{mr}^{(s)} z_r^{(s-1)} = \sum_{r=1}^{M_{s-1}} w_{mr}^{(s)} h_{s-1}\left(a_r^{(s-1)}(\mathbf{x}_n, \mathbf{w})\right),$$

the derivative with respect to $w_{ji}^{(l)}$ is

$$\frac{\partial}{\partial w_{ji}^{(l)}} a_m^{(s)}(\mathbf{x}_n, \mathbf{w}) = \sum_{r=1}^{M_{s-1}} w_{mr}^{(s)} h'_{s-1}(a_r^{(s-1)}) \frac{\partial}{\partial w_{ji}^{(l)}} a_r^{(s-1)}(\mathbf{x}_n, \mathbf{w}),$$

where h'_{s-1} is the derivative of h_{s-1}.

Chain rule

$$\begin{aligned}
\frac{\partial}{\partial w_{ji}^{(l)}} E_n(\mathbf{w}) &= \sum_{m=1}^{M_s} \delta_m^{(s)} \sum_{r=1}^{M_{s-1}} w_{mr}^{(s)} h'_{s-1}(a_r^{(s-1)}) \frac{\partial}{\partial w_{ji}^{(l)}} a_r^{(s-1)}(\mathbf{x}_n, \mathbf{w}) \\
&= \sum_{r=1}^{M_{s-1}} \underbrace{h'_{s-1}(a_r^{(s-1)}) \sum_{m=1}^{M_s} \delta_m^{(s)} w_{mr}^{(s)}}_{\delta_r^{(s-1)}} \frac{\partial}{\partial w_{ji}^{(l)}} a_r^{(s-1)}(\mathbf{x}_n, \mathbf{w}),
\end{aligned}$$

Chain rule

- Once we reach the l^{th} activations $a_m^{(l)}$, it is directly dependent on $w_{ji}^{(l)}$ via

$$a_m^{(l)} = \sum_{s=1}^{M_{l-1}} w_{ms}^{(l)} z_s^{(l-1)}.$$

- Differentiation is straightforward and we arrive at

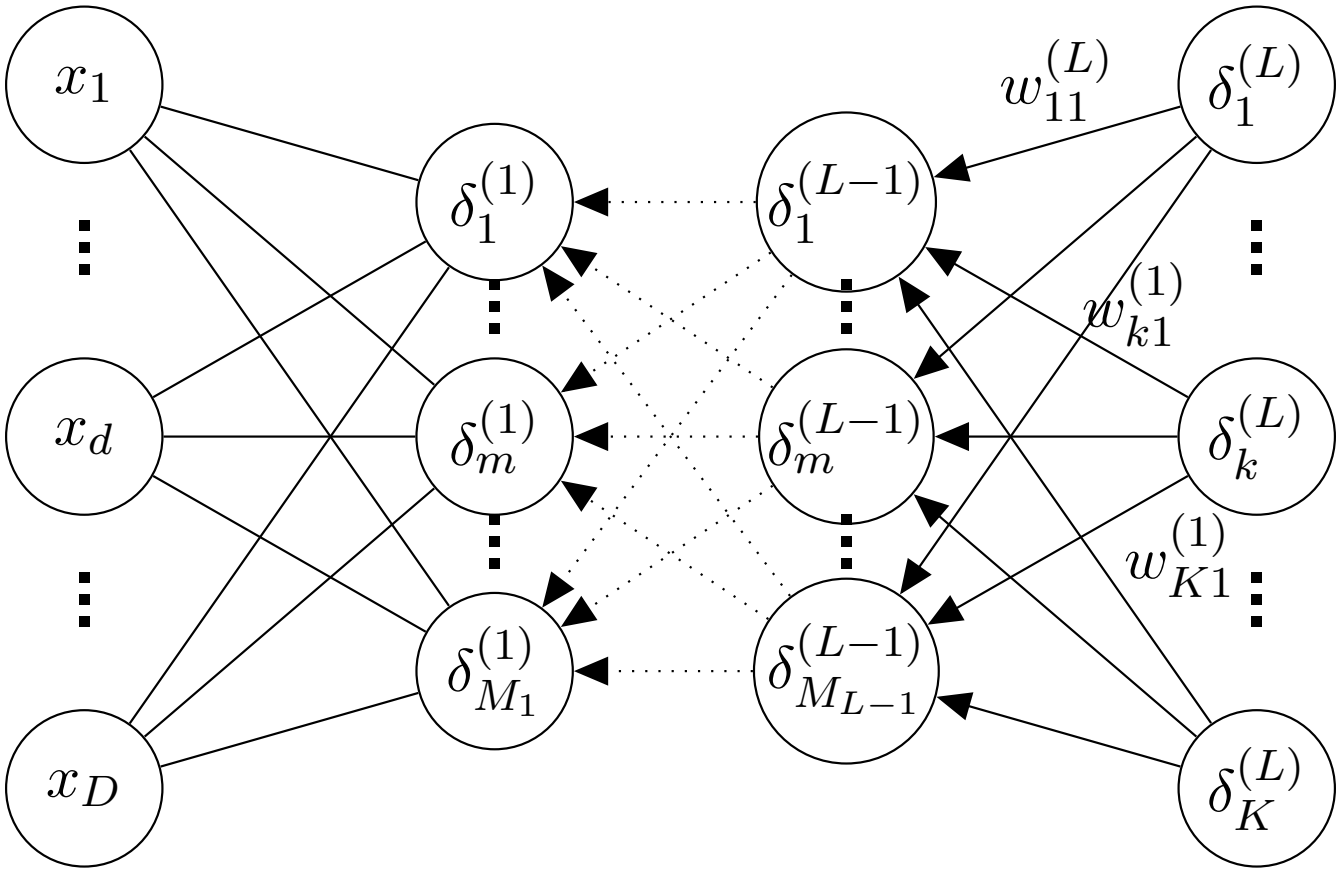
$$\frac{\partial}{\partial w_{ji}^{(l)}} a_m^{(l)}(\mathbf{x}_n, \mathbf{w}) = \begin{cases} 0 & \text{for} \quad m \neq j \\ z_i^{(L-1)} & \text{for} \quad m = j \end{cases}.$$

- The derivative of the n^{th} contribution to the error becomes

$$\frac{\partial}{\partial w_{ji}^{(l)}} E_n(\mathbf{w}) = \delta_j^{(l)} z_i^{(l-1)}.$$

- Note that, if $l = 1$, instead of $z_i^{(l-1)}$, $x_{n,i}$ is used.

Error Backpropagation



$$\delta_k^{(L)} = y_k(\mathbf{x}_n, \mathbf{w}) - t_{nk}, \qquad \delta_r^{(s-1)} = h'_{s-1}(a_r^{(s-1)}) \sum_{m=1}^{M_s} \delta_m^{(s)} w_{mr}^{(s)}$$

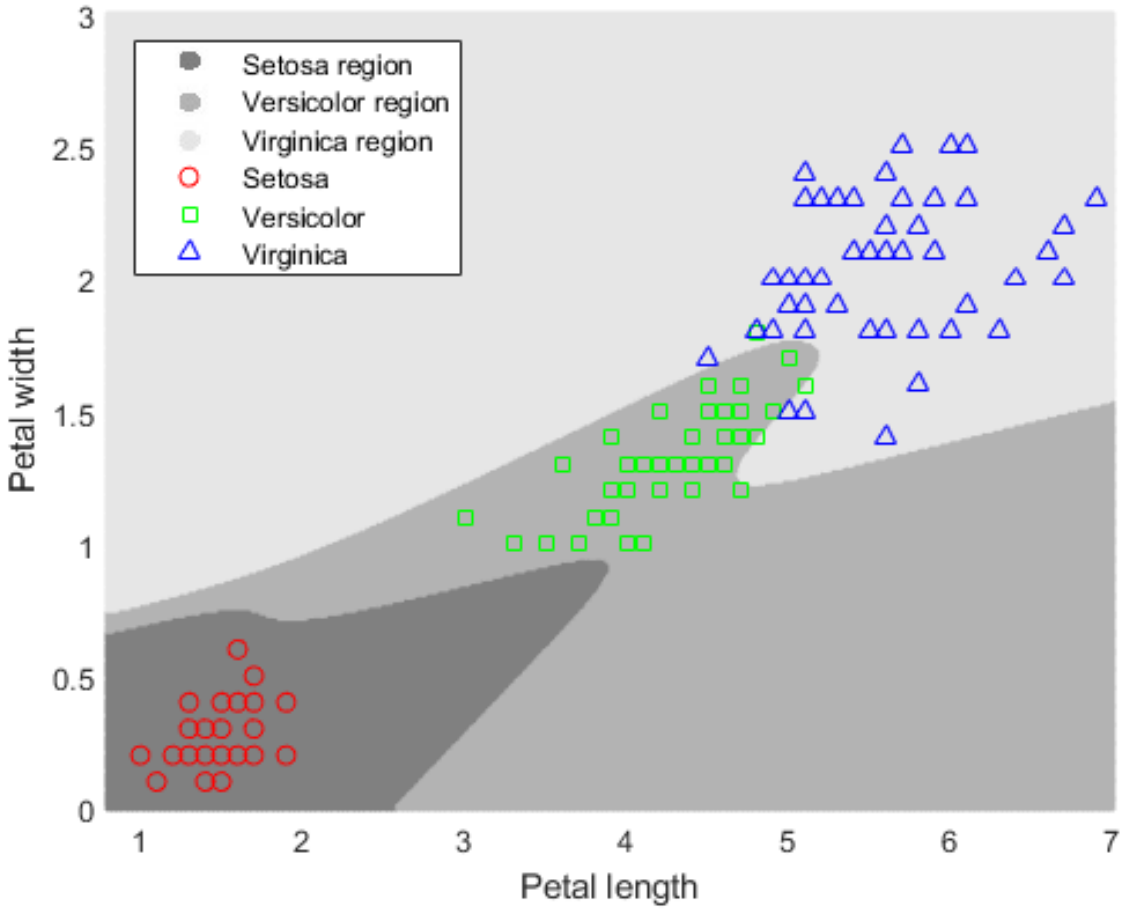
Intuition

- In binary classification $\frac{\partial}{\partial w_i^{(L)}} E_n(\mathbf{w}) = (y(\mathbf{x}_n, \mathbf{w}) - t_n) z_i^{(L-1)}$

	$y(\mathbf{x}_n, \mathbf{w}) - t_n \in$
incorrectly classified as positive	$[-1, -0.5)$
correctly classified as positive	$(-0.5, 0]$
correctly classified as negative	$[0, 0.5)$
incorrectly classified as negative	$(0.5, 1]$

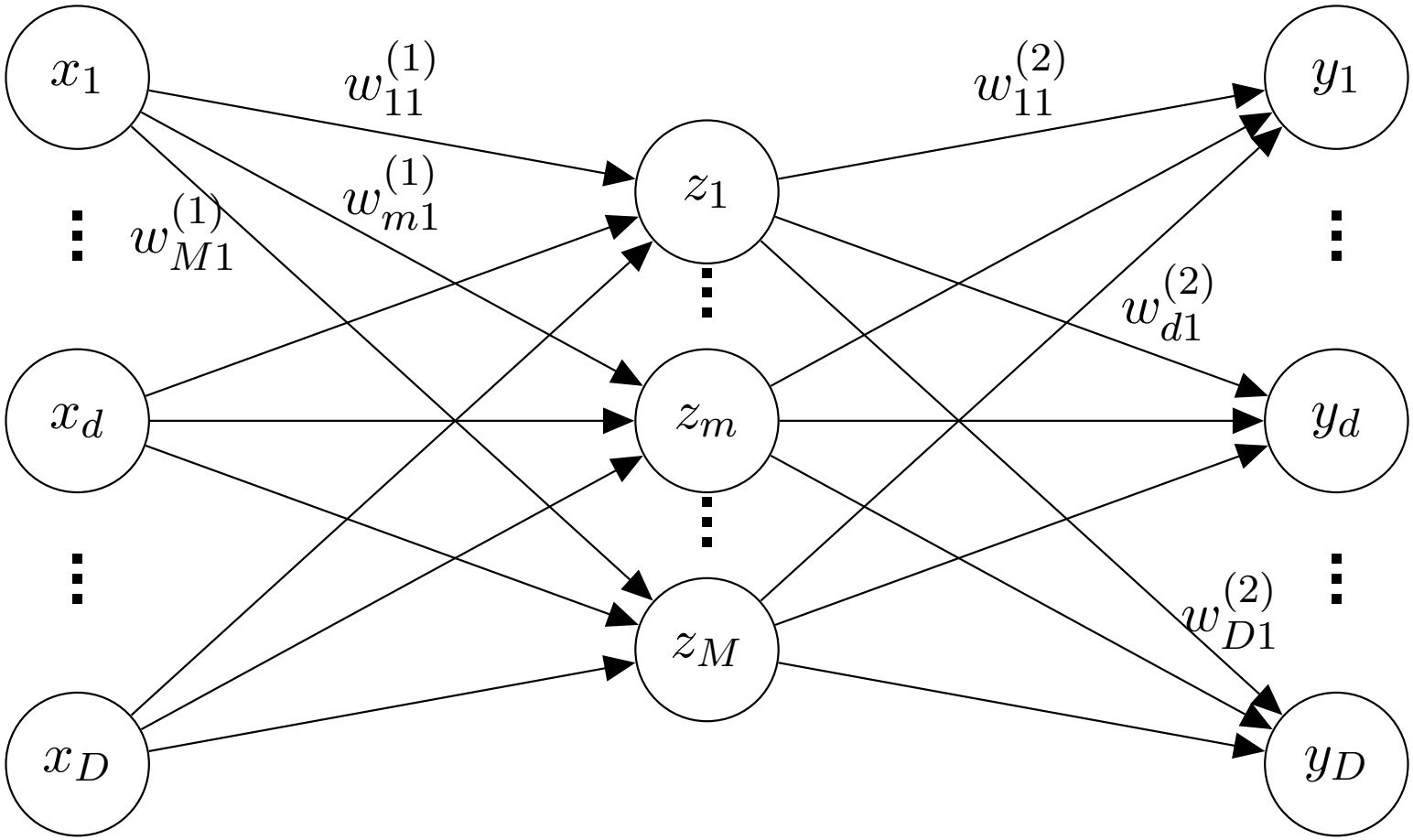
- When the weights get updated by this component of the gradient vector, they are adjusted in a positive direction, if the sample is in the negative class, and in a negative direction, if it is in the positive class.
- Adjustments are weighted by $z_i^{(L-1)} \geq 0$ and the learning rate $\eta > 0$.

Example



Two layer neural network classification with a set of three hidden neurons using the logistic sigmoid activation function.

Autoencoders



Goal: Output shall resemble input.

Autoencoders

- Purpose: Reconstruction of the input data, i.e. $\mathbf{y} \approx \mathbf{x}$.
- Therefore $K = D$.
- Even number of layers L.
- Therefore number of sets of hidden neurons is odd.
- First $L/2$ layers encode the data.
- Last $L/2$ layers decode it.
- *Encoder* function ϕ maps the input $\mathbf{x} = (x_1, \ldots, x_D)^T$ to the latent variables $\mathbf{z}^{(L/2)} = (z_1^{(L/2)}, \ldots, z_{M_{L/2}}^{(L/2)})^T$.
- *Decoder* function ψ maps the latent variables $\mathbf{z}^{(L/2)}$ back to $\mathbf{x}$ (hopefully).

Autoencoders

- Bias generally not implemented as dummy neuron, but explicitly as vector $\mathbf{b}^{(l)} = (b_1, \ldots, b_{M_l})^T$.
- Weights of each layer are stored in the weight matrix $\mathbf{W}^{(l)}$, the (j, i) entry being $w_{ji}^{(l)}$.
- Vector of activations $\mathbf{a}^{(l)} = (a_1, \ldots, a_{M_l})^T$ is then

$$\mathbf{a}^{(l)} = \mathbf{W}^{(l)}\mathbf{z}^{(l-1)} + \mathbf{b}^{(l)},$$

  where $\mathbf{z}^{(0)} = \mathbf{x}$.
- The notation $h^{(l)}(\mathbf{a}^{(l)})$ is used to denote the element-wise application of the activation function $h^{(l)}$ to the entries of $\mathbf{a}^{(l)}$.
- Error function is typically the *mean squared error*,

$$E(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \|\mathbf{x}_n - \psi(\phi(\mathbf{x}_n))\|^2.$$

Denoising Autoencoder (DAE)

Denoising Autoencoder (DAE)

- Training samples $\mathbf{x}_1, \ldots, \mathbf{x}_n$ first undergo a probabilistic corruption process giving corrupted training samples $\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_n$.
- Error function $E(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} \|\mathbf{x}_n - \psi(\phi(\tilde{\mathbf{x}}_n))\|^2$.
- The corrupted data is encoded and decoded and the results compared to the uncorrupted data.

k-Sparse Autoencoders (k-SAE)

k-Sparse Autoencoders (SAE)

- The *k-sparse autoencoder* keeps only the k largest latent variables in a particular set of hidden neurons and set the others to zero.
- Backpropagation only passes through the hidden neurons, whose latent variables are non-zero.
- Similar to quantification in the JPEG standard in 1992.

Sparse Autoencoders (SAE)

- *Average activation* of the m^{th} neuron in the l^{th} set of hidden neurons is defined as

$$p_m^{(l)} = \frac{1}{N} \sum_{n=1}^{N} z_m^{(l)}(\mathbf{x}_n).$$

- Aim: $z_m^{(l)}$ has non-negligible values for only a subset of inputs identifying a subset with common features.

Sparse Autoencoders (SAE)

- Error function includes the *sparsity regularization* penalty term

$$\begin{aligned}\Omega_s &= \sum_{l=1}^{L-1}\sum_{m=1}^{M_l} D_{KL}(p\|p_m^{(l)}) \\ &= \sum_{l=1}^{L-1}\sum_{m=1}^{M_l}\left(p\log\frac{p}{p_m^{(l)}} + (1-p)\log\frac{1-p}{1-p_m^{(l)}}\right),\end{aligned}$$

  weighted by a parameter β.

- $D_{KL}(p\|p_m^{(l)})$ is the *Kullback–Leibler divergence* between $p_m^{(l)}$ and p, which is the predefined *sparsity proportion*.

Sparse Autoencoders (SAE)

- Latent variables might become small, while the weights in the subsequent layer become large counteracting the effect.
- The *L_2 regularization* term weighted by a parameter λ acts on the weights,

$$\Omega_w = \frac{1}{2} \sum_{l=1}^{L} \sum_{j=1}^{M_{l-1}} \sum_{i=1}^{M_l} w_{ji}^{(l)},$$

where $M_0 = M_L = D$.

Contractive Autoencoders (CAE)

- Let the encoder function be $\phi(\mathbf{x}) = (\phi_1(\mathbf{x}), \ldots, \phi_{M_{L/2}}(\mathbf{x}))^T$. Its *Jacobian matrix* is

$$J_\phi(\mathbf{x}) = \begin{pmatrix} \frac{\partial \phi_1}{\partial x_1}(\mathbf{x}) & \cdots & \frac{\partial \phi_1}{\partial x_D}(\mathbf{x}) \\ \vdots & \ddots & \vdots \\ \frac{\partial \phi_{M_{L/2}}}{\partial x_1}(\mathbf{x}) & \cdots & \frac{\partial \phi_{M_{L/2}}}{\partial x_D}(\mathbf{x}) \end{pmatrix}.$$

- If $M_{L/2} = D$, then the modulus of its determinant is the factor by which volumes in $\mathbb{R}^D$ under the transformation by ϕ shrink or expand.

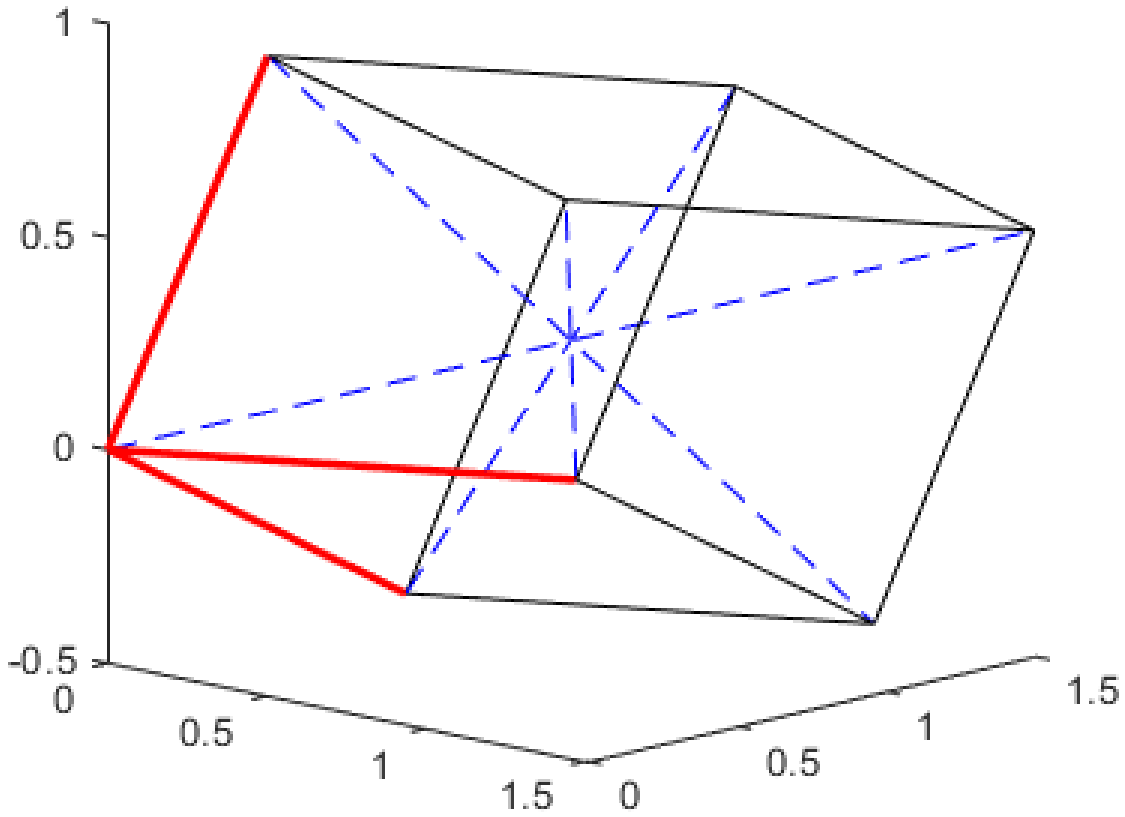
Contractive Autoencoders (CAE)

- If $M_{L/2} < D$, the *Frobenius norm* is a measure of change.
- The penalty term

$$\begin{aligned}\Omega_F &= \frac{1}{2}\sum_{n=1}^{N}\|J_\phi(\mathbf{x}_n)\|_F^2 = \sum_{n=1}^{N}\mathrm{tr}(J_\phi(\mathbf{x}_n)^T J_\phi(\mathbf{x}_n)) \\ &= \sum_{n=1}^{N}\sum_{m=1}^{M_{L/2}}\sum_{d=1}^{D}[\frac{\partial \phi_m}{\partial x_d}(\mathbf{x}_n)]^2\end{aligned}$$

is added to the error function, weighted by λ.

Contractive Autoencoders (CAE)



The squared Frobenius norm is the average of the squared lengths of the space diagonals of the parallelepiped spanned by the columns of a 3×3 matrix $\mathbf{A}$.

Contractive Autoencoders (CAE)

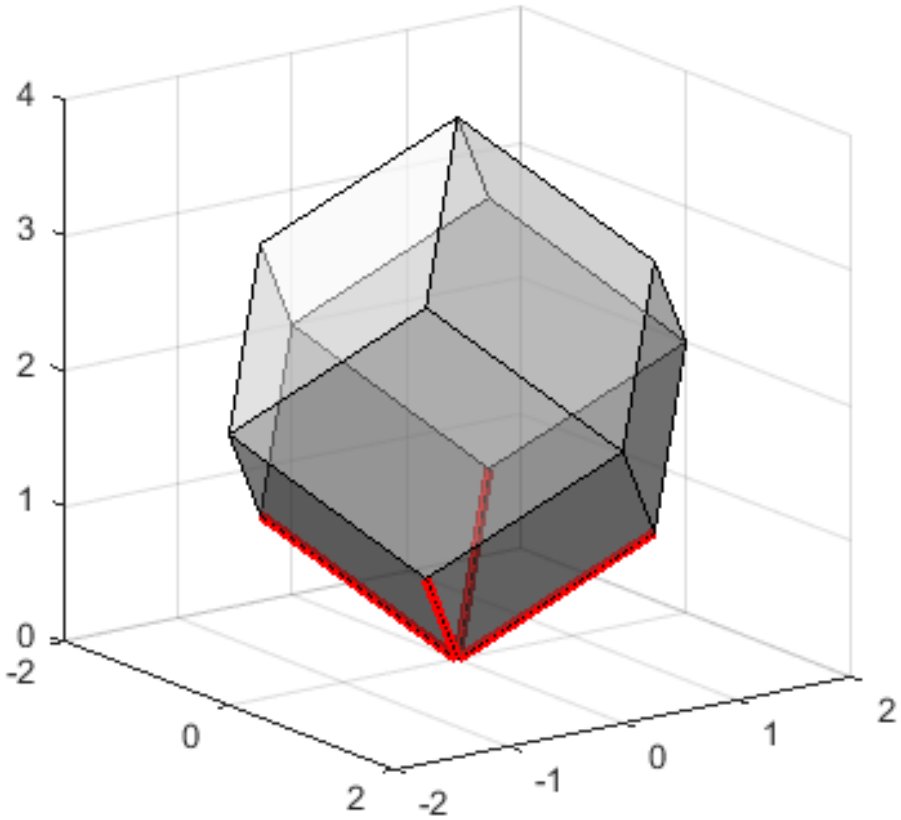


Image of the four dimensional unit hypercube under the action of a 3×4 matrix $\mathbf{A}$..

Variational Autoencoders (VAE)

- Assumption: Data are sampled from some distribution $p(\mathbf{x})$, where

$$p(\mathbf{x}) = \int \underbrace{p\left(\mathbf{x}|\mathbf{z}^{(L/2)}\right)}_{\mathcal{N}(\psi(\mathbf{z}^{(L/2)}), \sigma^2\mathbf{I})} \underbrace{p\left(\mathbf{z}^{(L/2)}\right)}_{\text{prior: } \mathcal{N}(0,\mathbf{I})} d\mathbf{z}^{(L/2)}.$$

- Estimated by by sampling many values $\mathbf{z}_1^{(L/2)}, \ldots, \mathbf{z}_K^{(L/2)}$ and calculating

$$\begin{aligned} p(\mathbf{x}) &= \frac{1}{K}\sum_{k=1}^{K} p(\mathbf{x}|\mathbf{z}_K^{(L/2)}) \\ &= \frac{1}{K}\sum_{k=1}^{K} (2\pi\sigma^2)^{-D/2} \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{x} - \psi(\mathbf{z}^{(L/2)})\|^2\right). \end{aligned}$$

Variational Autoencoders (VAE)

Complete data likelihood estimated as:

$$\prod_{n=1}^{N} p(\mathbf{x}_n) = \prod_{n=1}^{N} \frac{1}{K} \sum_{k=1}^{K} (2\pi\sigma^2)^{-D/2} \exp\left(-\frac{1}{2\sigma^2} \|\mathbf{x}_n - \psi(\mathbf{z}^{(L/2)})\|^2\right).$$

Disadvantages:

- No convenient sum of squares.
- K needs to be large (especially in high dimensions) to obtain a good estimate.
- Sampling is not a continuous operation and thus not differentiable.

Variational Autoencoders (VAE)

- Reduce K, by choosing $p(\mathbf{z}^{(L/2)})$ such that only values of $\mathbf{z}^{(L/2)}$ are sampled which are likely to produce samples of $\mathbf{x}$.
- Encoder gives $p_\phi(\mathbf{z}^{(L/2)}|\mathbf{x})$ as the mean and covariance matrix

$$(\boldsymbol{\mu}(\mathbf{x}), \boldsymbol{\Sigma}(\mathbf{x})) = \phi(\mathbf{x}).$$

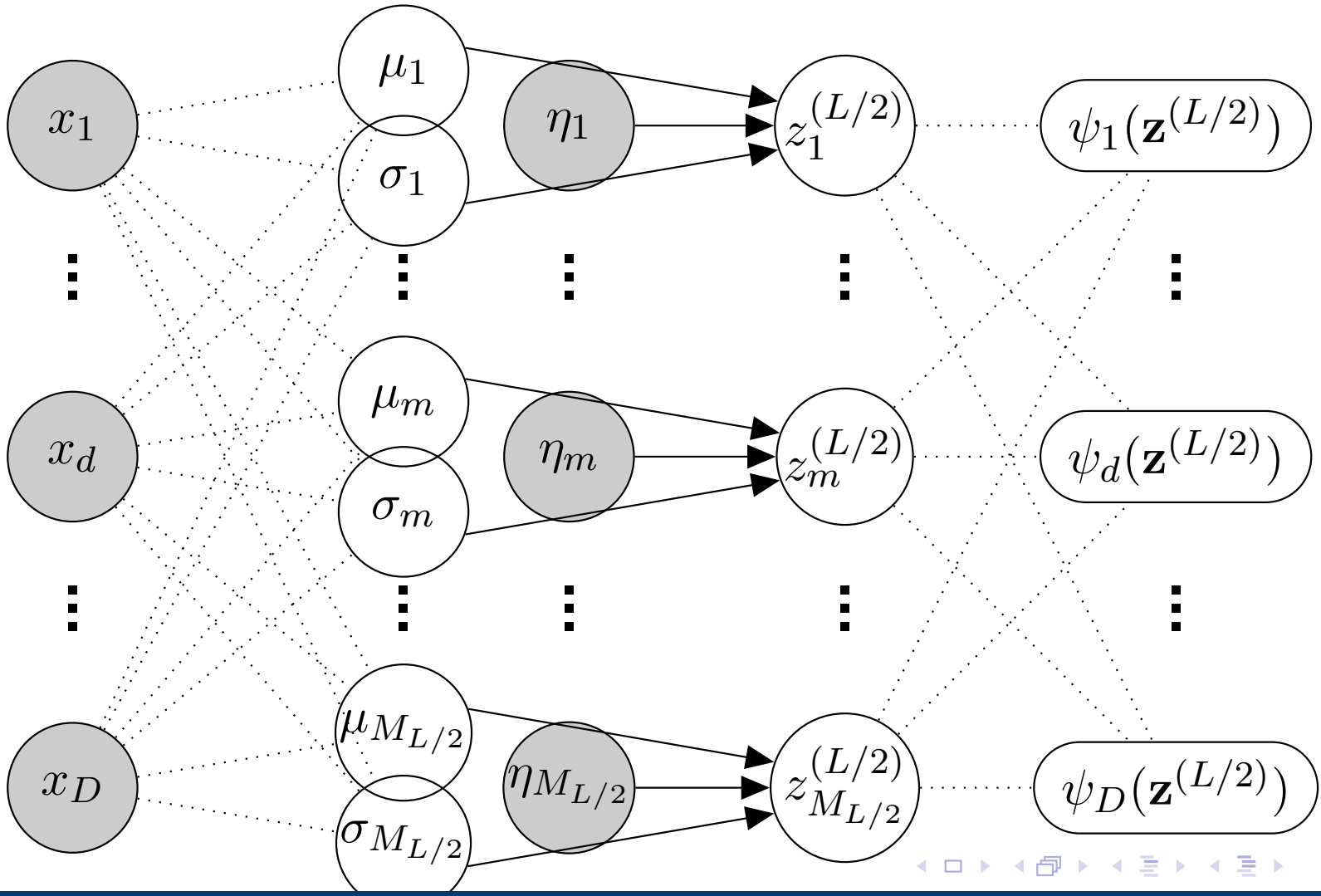
- $\boldsymbol{\Sigma}(\mathbf{x})$ is restricted to be diagonal, that is $\boldsymbol{\Sigma}(\mathbf{x}) = \mathrm{diag}(\sigma_1^2, \ldots \sigma_{M_{L/2}}^2)$.
- $\Rightarrow 2M_{L/2}$ latent variables.

Variational Autoencoders (VAE)

- To avoid sampling in the forward propagation, it is moved to additional input neurons.
- Vector $\boldsymbol{\eta} = (\eta_1, \ldots \eta_{M_{L/2}})^T$ is sampled from the standard normal distribution in $M_{L/2}$ dimensions.

$$\mathbf{z}^{(L/2)} = \boldsymbol{\mu}(\mathbf{x}) + \boldsymbol{\Sigma}^{1/2}(\mathbf{x})\boldsymbol{\eta} = \begin{pmatrix} \mu_1(\mathbf{x}) + \sigma_1(\mathbf{x})\eta_1 \\ \vdots \\ \mu_{M_{L/2}}(\mathbf{x}) + \sigma_{M_{L/2}}(\mathbf{x})\eta_{M_{L/2}} \end{pmatrix}.$$

Variational Autoencoders (VAE)



Variational Autoencoders (VAE)

Error function is the sum over all training data $\mathbf{x}_n$ of $-\log p(\mathbf{x}_n)$ and the Kullback-Leibler divergence between $p_\phi(\mathbf{z}^{(L/2)}|\mathbf{x}_n)$ and $p(\mathbf{z}^{(L/2)}|\mathbf{x}_n)$.

$$
\begin{aligned}
E_n &= -\log p(\mathbf{x}_n) + D_{KL}(p_\phi(\mathbf{z}^{(L/2)}|\mathbf{x}_n)||p(\mathbf{z}^{(L/2)}|\mathbf{x}_n)) \\
&= \int p_\phi(\mathbf{z}^{(L/2)}|\mathbf{x}_n) \left[-\log p(\mathbf{x}_n) - \log \frac{p(\mathbf{z}^{(L/2)}|\mathbf{x}_n)}{p_\phi(\mathbf{z}^{(L/2)}|\mathbf{x}_n)} \right] d\mathbf{z}^{(L/2)}.
\end{aligned}
$$

Using Bayes rule

$$p(\mathbf{z}^{(L/2)}|\mathbf{x}_n) = \frac{p(\mathbf{x}_n|\mathbf{z}^{(L/2)})p(\mathbf{z}^{(L/2)})}{p(\mathbf{x}_n)},$$

this simplifies to

$$\begin{aligned} E_n &= \int p_\phi(\mathbf{z}^{(L/2)}|\mathbf{x}_n) \left[-\log \frac{p(\mathbf{z}^{(L/2)})}{p_\phi(\mathbf{z}^{(L/2)}|\mathbf{x}_n)} - \log p(\mathbf{x}_n|\mathbf{z}^{(L/2)}) \right] d\mathbf{z}^{(L/2)} \\ &= D_{KL}(p_\phi(\mathbf{z}^{(L/2)}|\mathbf{x}_n) \| p(\mathbf{z}^{(L/2)})) \\ &\quad - \int p_\phi(\mathbf{z}^{(L/2)}|\mathbf{x}_n) \log p(\mathbf{x}_n|\mathbf{z}^{(L/2)}) d\mathbf{z}^{(L/2)}. \end{aligned}$$

The first term is the Kullback–Leibler divergence between the normal distribution with mean $\boldsymbol{\mu}(\mathbf{x}_n)$ and variance $\boldsymbol{\Sigma}(\mathbf{x}_n)$ and the standard normal distribution which is the prior of $\mathbf{z}^{(L/2)}$. This is given by

$$D_{KL}(p_\phi(\mathbf{z}^{(L/2)}|\mathbf{x}_n)\|p(\mathbf{z}^{(L/2)})) =$$

$$\frac{1}{2}\left[\mathrm{tr}\boldsymbol{\Sigma}(\mathbf{x}_n) + \boldsymbol{\mu}(\mathbf{x}_n)^T\boldsymbol{\mu}(\mathbf{x}_n) - M_{L/2} - \log|\boldsymbol{\Sigma}(\mathbf{x}_n)|\right] =$$

$$\frac{1}{2}\sum_{m=1}^{M_{L/2}}\left[\sigma_m^2(\mathbf{x}_n) + \mu_m^2(\mathbf{x}_n) - \log\sigma_m^2(\mathbf{x}_n)\right] - \frac{M_{L/2}}{2},$$

Variational Autoencoders (VAE)

- The second term is the negative expectation of $\log p(\mathbf{x}_n|\mathbf{z}^{(L/2)})$ with respect to $\mathbf{z}^{(L/2)}$.
- It is estimated by summing over samples generated by $\mathbf{z}^{(L/2)} = \boldsymbol{\mu}(\mathbf{x}_n) + \boldsymbol{\Sigma}^{1/2}(\mathbf{x}_n)\boldsymbol{\eta}$, where $\boldsymbol{\eta}$ follows the standard normal distribution.
- It is customary to enrich the training samples with samples drawn from the standard normal distribution to arrive at new training samples $\mathbf{x}_{nk} = (\mathbf{x}_n, \boldsymbol{\eta}_k)^T$ for $n = 1, \ldots, N$ and $k = 1, \ldots, K$ for some suitably chosen K.

Variational Autoencoders (VAE)

- The second term is estimated as

$$-\log p(\mathbf{x}_n|\boldsymbol{\mu}(\mathbf{x}_n) + \boldsymbol{\Sigma}^{1/2}(\mathbf{x}_n)\boldsymbol{\eta}_k) =$$

$$\frac{D}{2}\log(2\pi\sigma^2) + \frac{1}{2\sigma^2}\|\mathbf{x}_n - \psi(\boldsymbol{\mu}(\mathbf{x}_n) + \boldsymbol{\Sigma}^{1/2}(\mathbf{x}_n)\boldsymbol{\eta}_k)\|^2.$$

- Therefore there are NK terms in the error function of the form:

$$\begin{aligned} E_{nk} &= \frac{1}{2}\sum_{m=1}^{M_{L/2}}\left[\sigma_m^2(\mathbf{x}_n) + \mu_m^2(\mathbf{x}_n) - \log\sigma_m^2(\mathbf{x}_n)\right] \\ &\quad + \frac{1}{2}\|\mathbf{x}_n - \psi(\boldsymbol{\mu}(\mathbf{x}_n) + \boldsymbol{\Sigma}^{1/2}(\mathbf{x}_n)\boldsymbol{\eta}_k)\|^2. \end{aligned}$$

Handwritten Digits

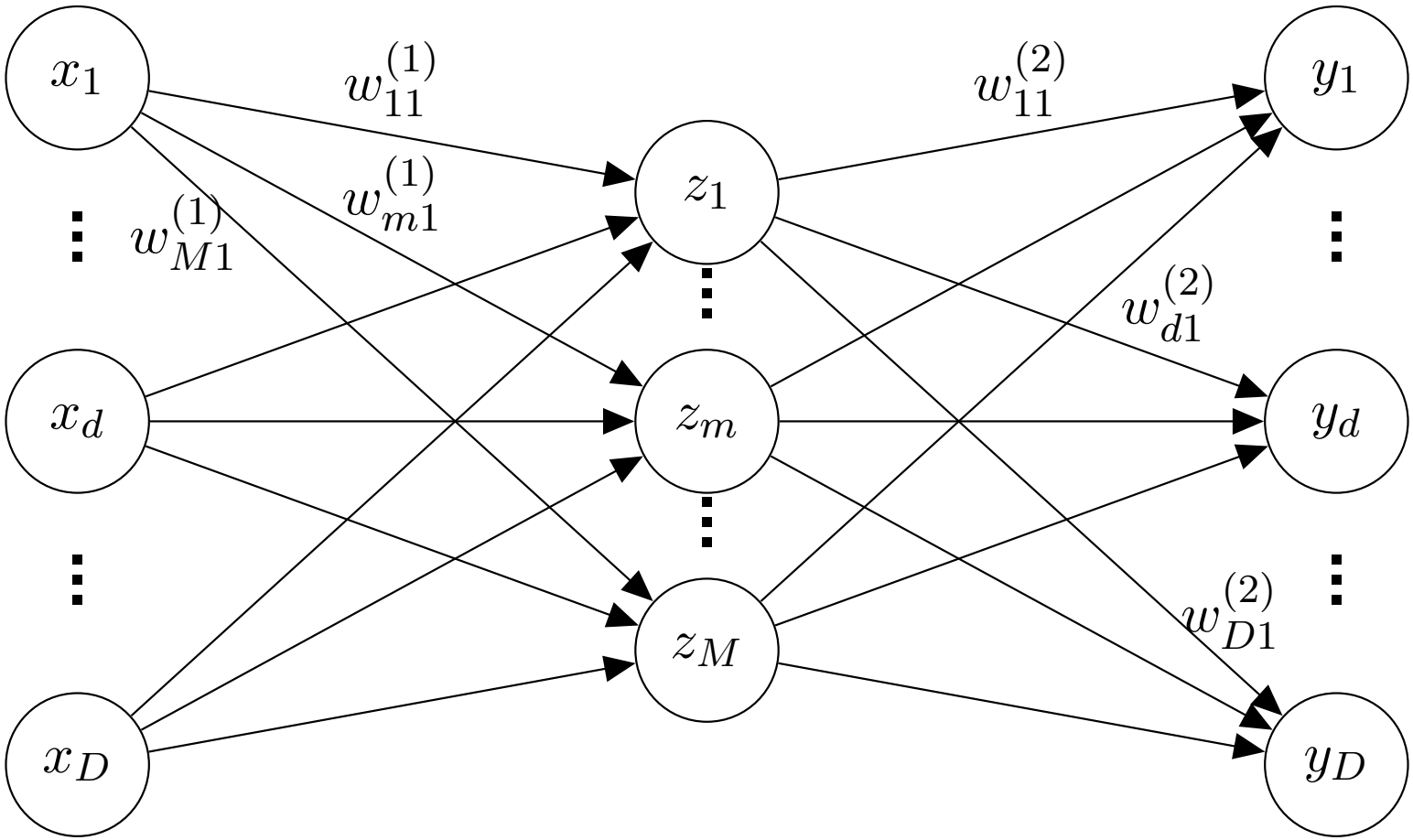
Handwritten Digits

28×28 pixels.

$\Rightarrow 784$ dimensions.

One image is one point in the 784-dimensional space.

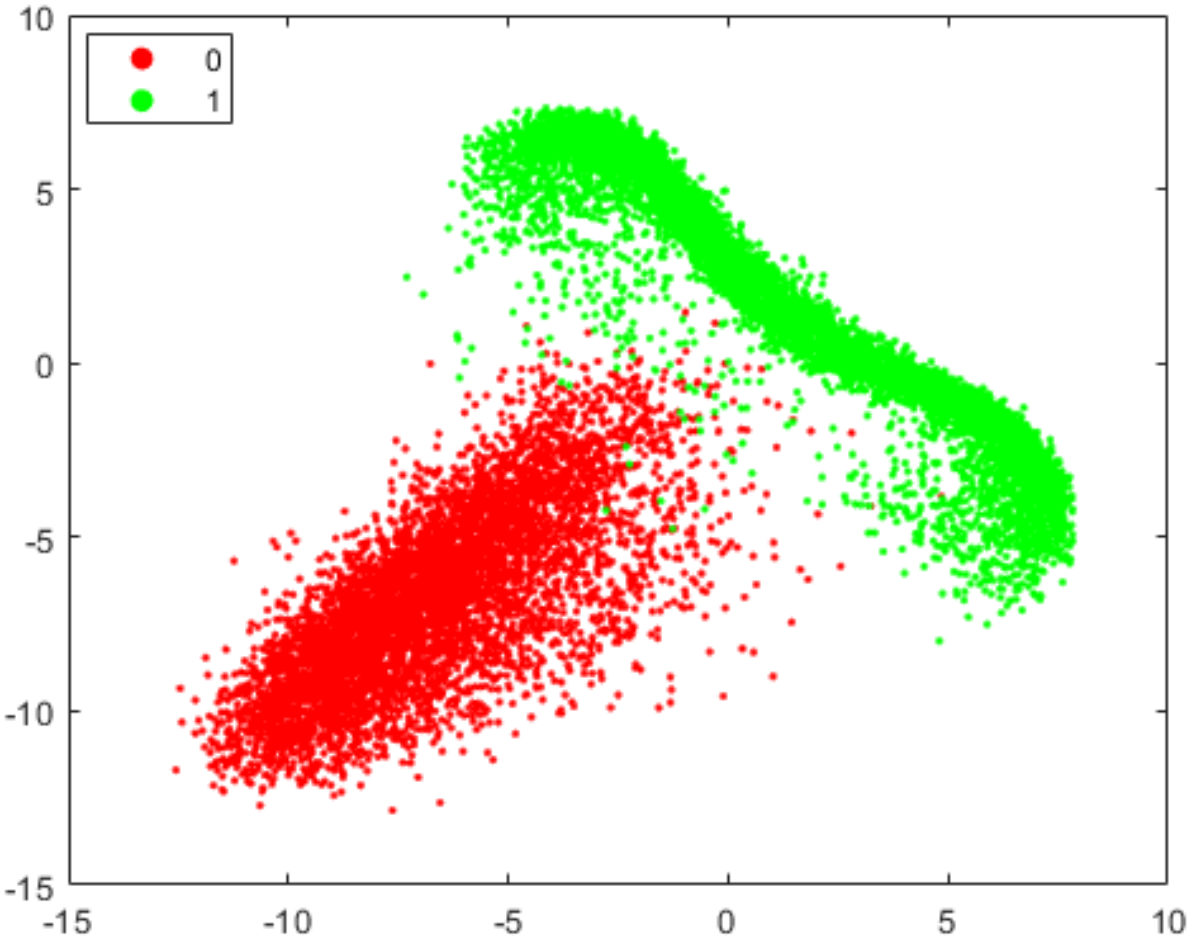
Autoencoder Example



Zero and One, 2 Hidden Neurons

- Starting simple: Two digits, *zero* and *one*.
- Only two hidden neurons, $M = 2$.
- This means, we look along two directions in the space with 784 dimensions.

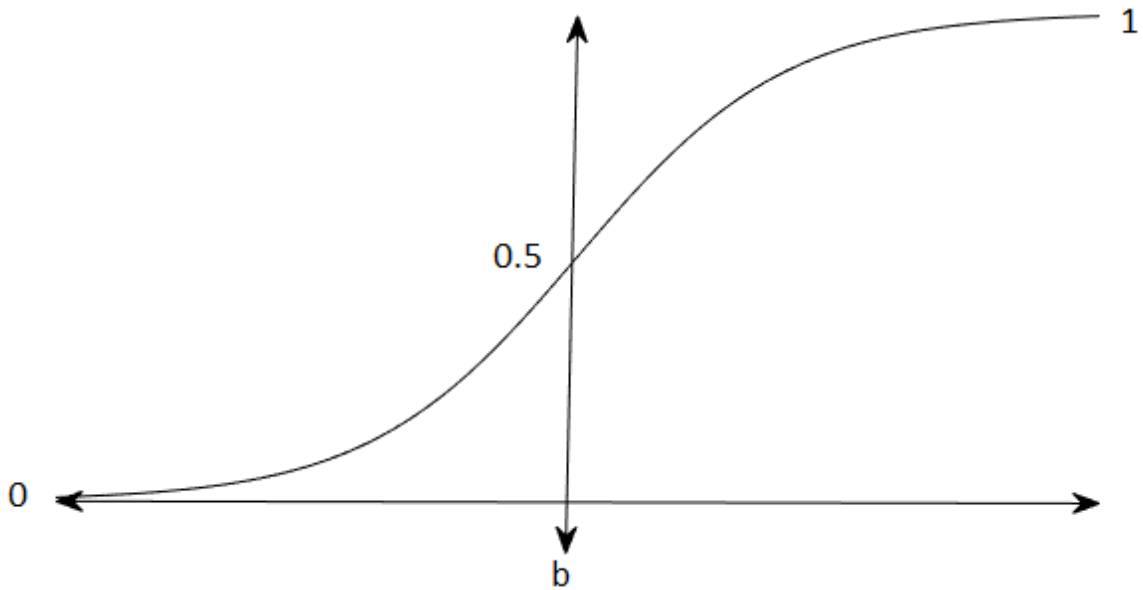
Zero and One, 2 Hidden Neurons



Logistic Sigmoid

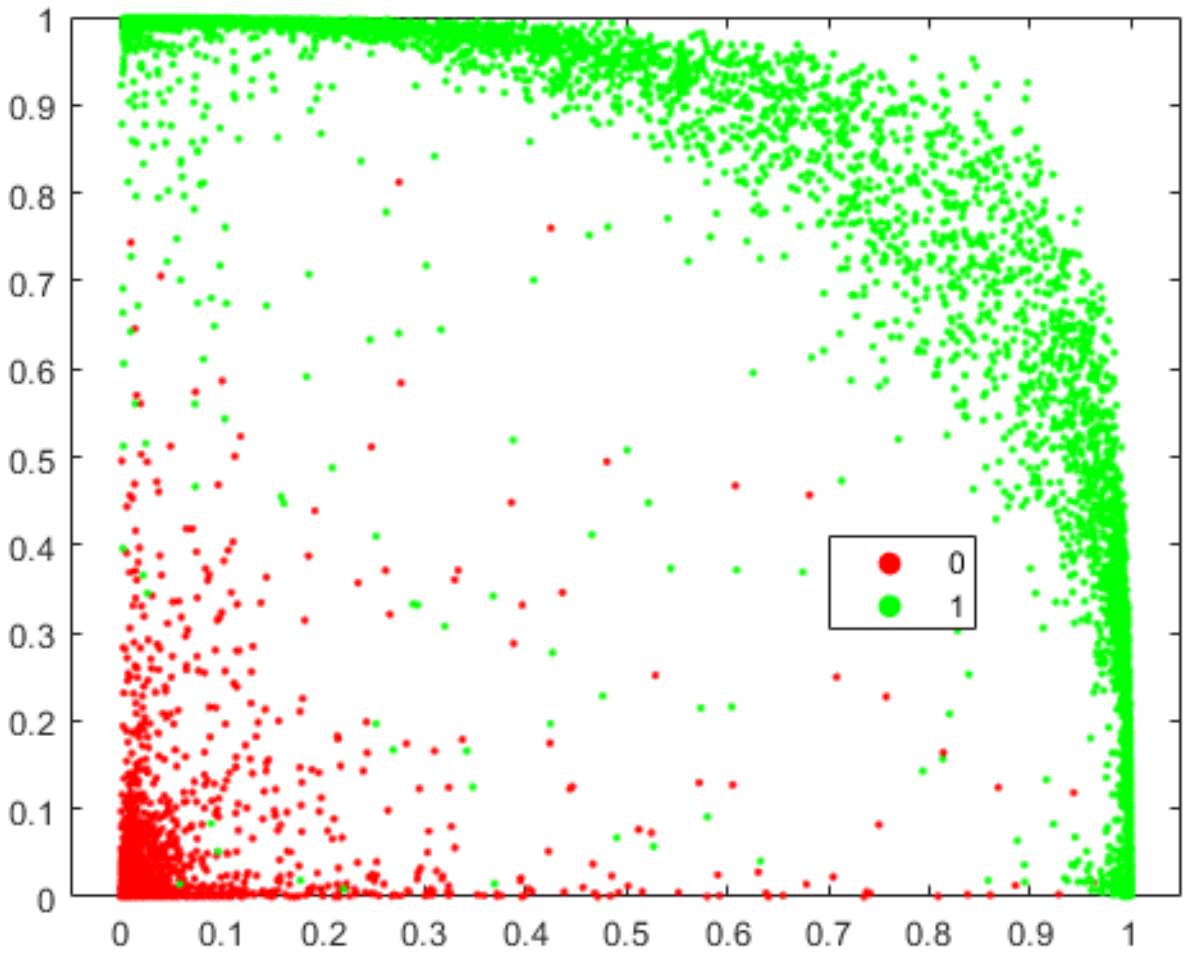
The logistic sigmoid function

$$\frac{1}{1+\exp(-a(x-b))}$$



can be used to pull the points further apart.

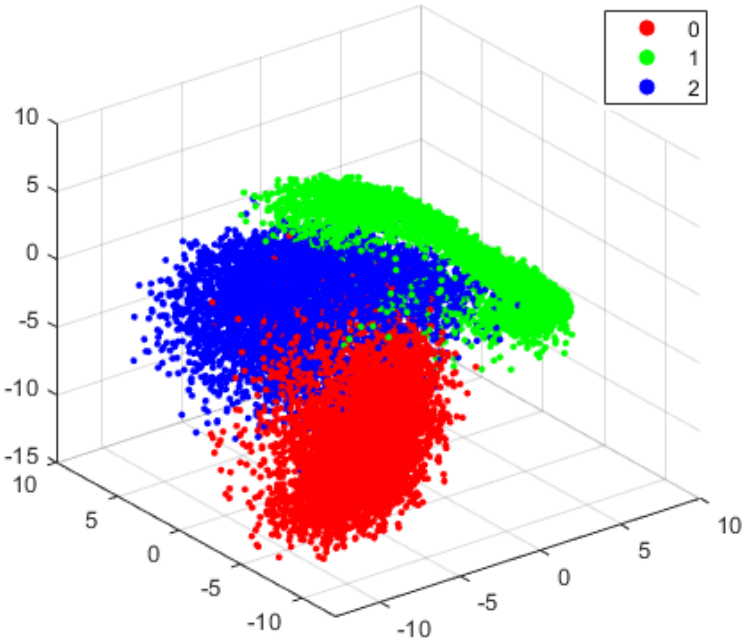
Zero and One, 2 Hidden Neurons

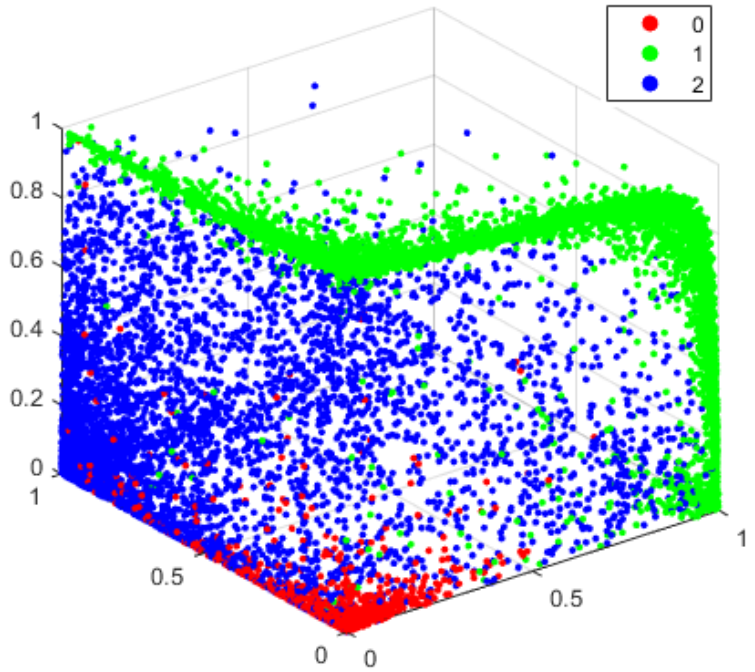


Zero and One

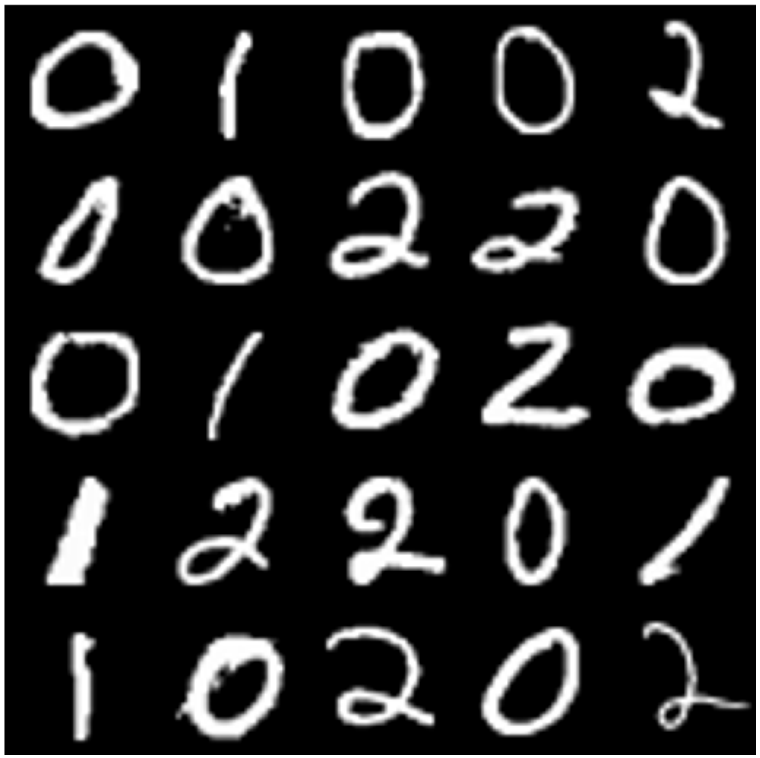
Zero, One and Two, 3 Hidden Neurons

Three digits, *zero*, *one* and *two*. Only three hidden neurons, $M = 3$.

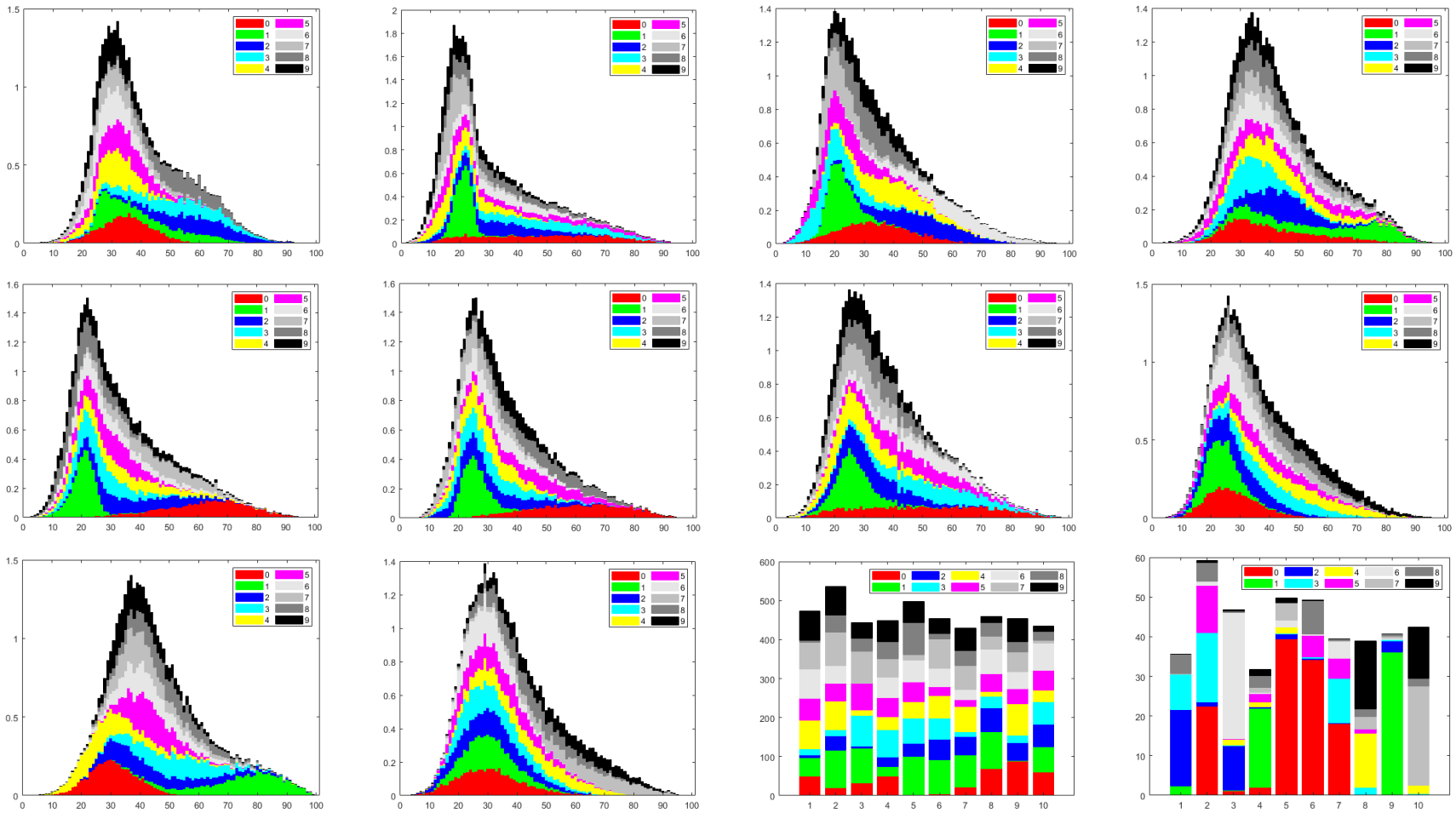




Zero, One and Two, 3 Hidden Neurons



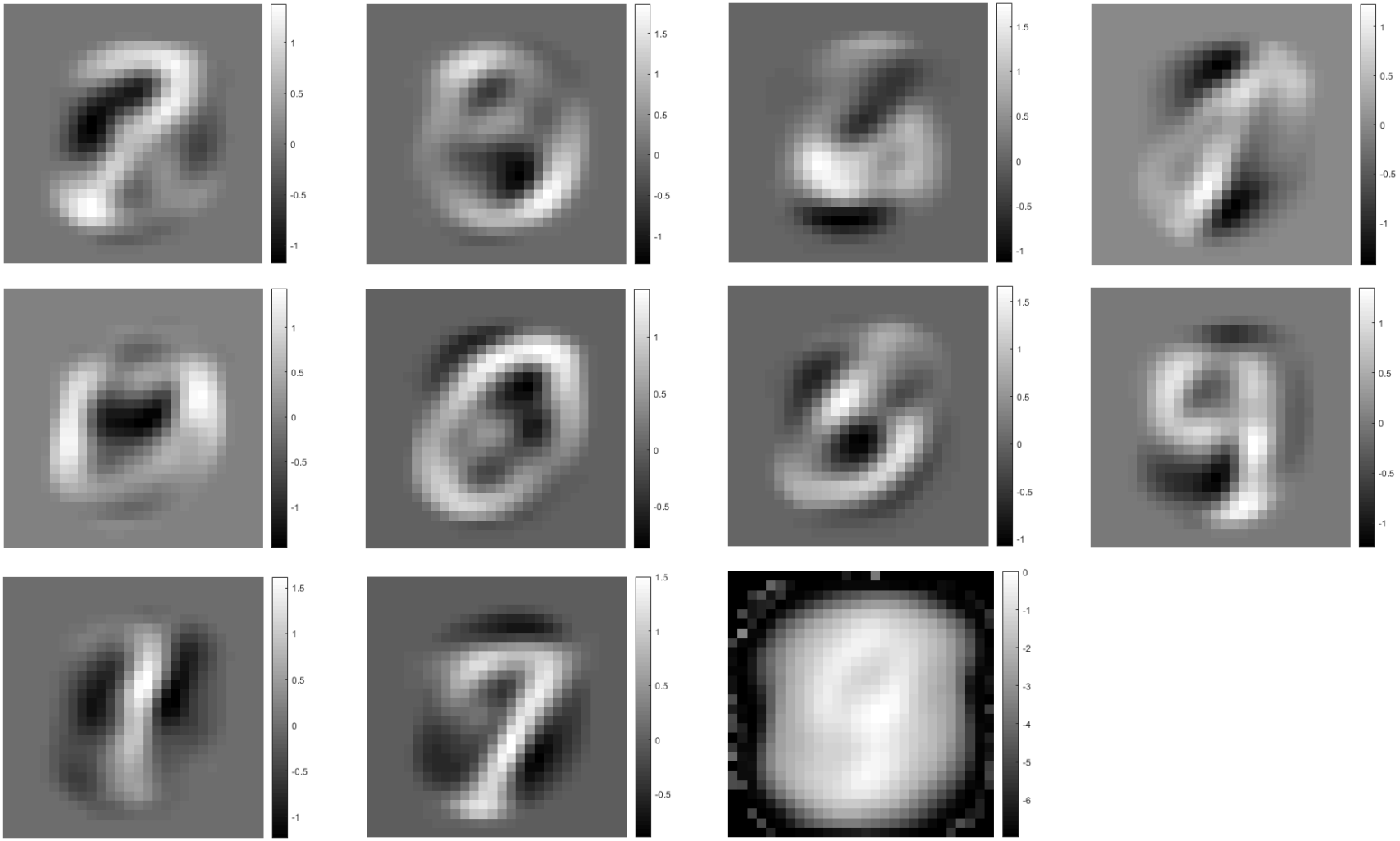
10 Digits, 10 Hidden Neurons



10 Digits, 10 Hidden Neurons

Neuron \ Digit	0	1	2	3	4	5	6	7	8	9
1			×	×					×	
2	×			×		×			×	
3			×				×			
4		×							×	
5	×							×		
6	×					×			×	
7	×			×		×	×			
8					×					×
9		×	×							
10								×		×

10 Digits, 10 Hidden Neurons



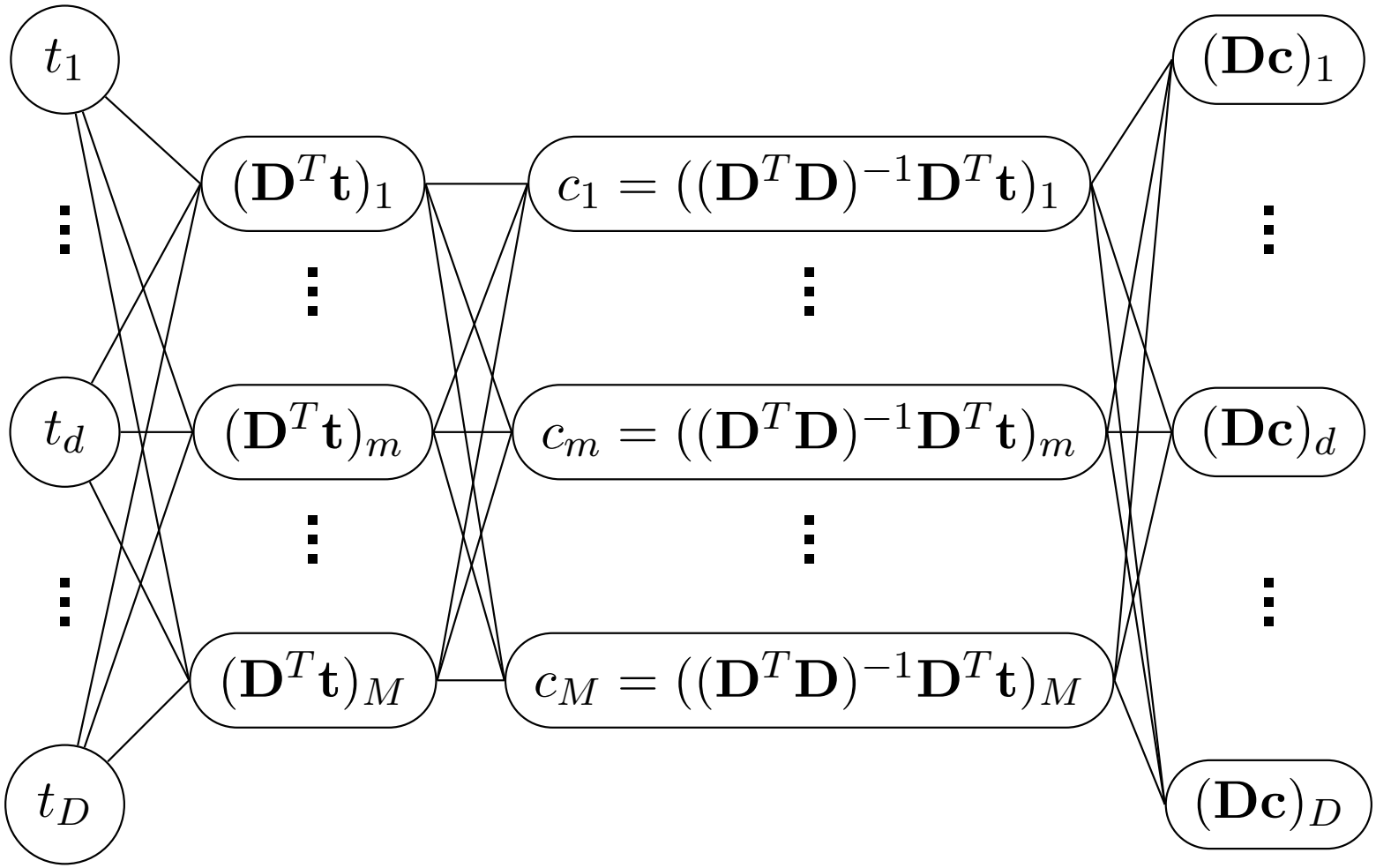
10 Digits, 10 Hidden Neurons

Ten digits, ten hidden neurons. ☹

$\Rightarrow$ more hidden neurons 😄

Relationship to other techniques

Ordinary Least Squares: $\mathbf{Dc} = \mathbf{D}(\mathbf{D}^T\mathbf{D})^{-1}\mathbf{D}^T\mathbf{t}$.



Ordinary Least Squares

- Error sum of squares

$$E(\mathbf{D}) = \sum_{n=1}^{N} \left(\mathbf{t}_n^T \mathbf{t}_n - \mathbf{t}_n^T \mathbf{D} (\mathbf{D}^T \mathbf{D})^{-1} \mathbf{D}^T \mathbf{t}_n \right),$$

  since it is the ordinary least squares solution.

- The derivative with regards to the entry D_{ij} of $\mathbf{D}$ is

$$\frac{\partial}{\partial D_{ij}} E(\mathbf{D}) = \sum_{n=1}^{N} \left((\mathbf{D}^T \mathbf{D})^{-1} \mathbf{D}^T \mathbf{t}_n \right)_j \left(\mathbf{D} (\mathbf{D}^T \mathbf{D})^{-1} \mathbf{D}^T \mathbf{t}_n - \mathbf{t}_n \right)_i,$$

- The design matrix can be inferred from the data, if there are enough data samples $\mathbf{t}_n$.

Ordinary Least Squares

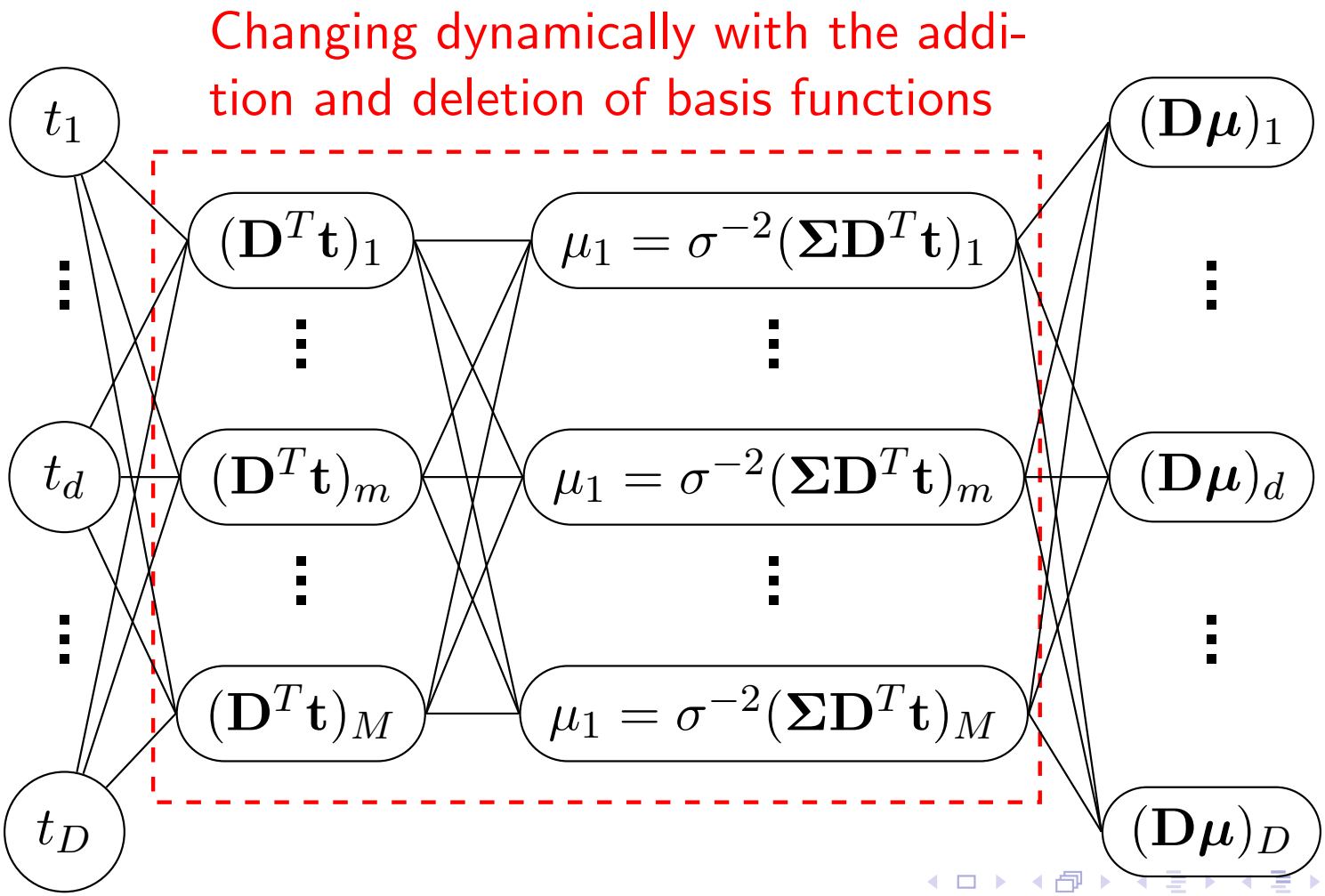
- $\mathbf{D}\mathbf{D}^T$ is symmetric and positive semi-definite.
- It can be diagonalized by an orthogonal matrix $\mathbf{Q}$ such that $\mathbf{D}\mathbf{D}^T = \mathbf{Q}\mathbf{P}\mathbf{Q}^T$, where $\mathbf{P}$ is a diagonal matrix with non-negative entries.

$$\mathbf{D}(\mathbf{D}^T\mathbf{D})^{-1}\mathbf{D}^T\mathbf{t} = (\mathbf{D}\mathbf{Q}\mathbf{P}^{-1/2})(\mathbf{D}\mathbf{Q}\mathbf{P}^{-1/2})^T\mathbf{t}.$$

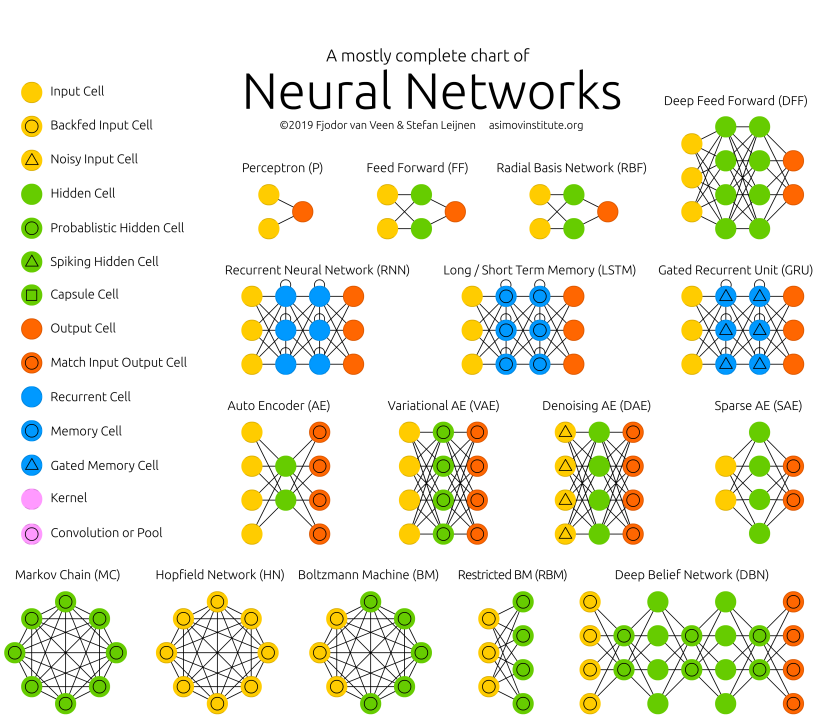
- $\Rightarrow$ tied two-layer autoencoder.
- $\mathbf{P}^{-1/2}$ are different scalings of the hidden variables.
- $\mathbf{Q}$ is either a rotation or a reflection.

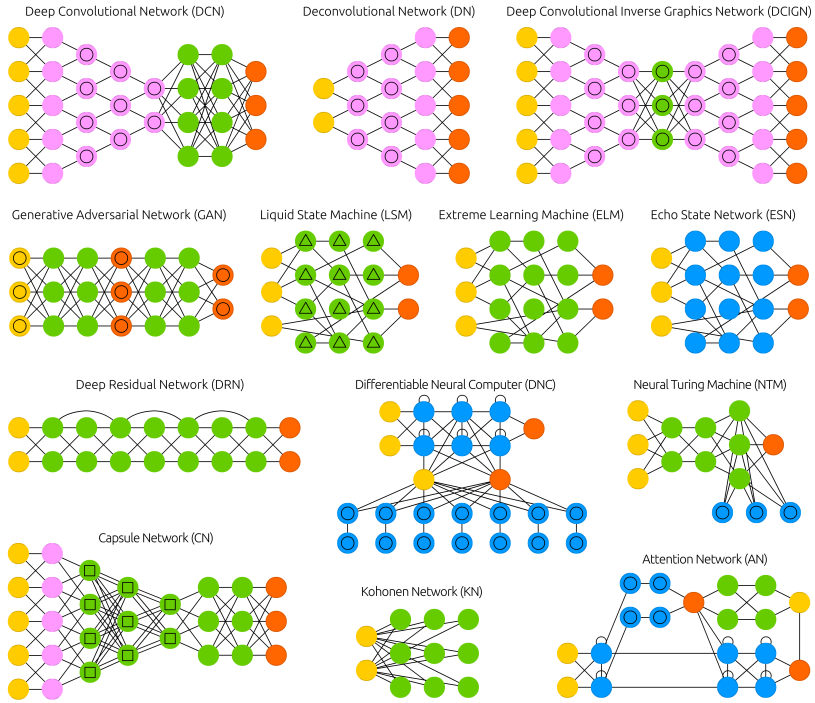
Bayesian Learning

Bayesian Learning: $\mathbf{D}\boldsymbol{\mu} = \sigma^{-2}\mathbf{D}\boldsymbol{\Sigma}\mathbf{D}^T\mathbf{t}$.



Neural Network Zoo





http://www.asimovinstitute.org/neural-network-zoo/

Indian Buffet Process

- Dimension M of the latent space unknown.
- $\mathbf{t}_1, \ldots, \mathbf{t}_N$ set of data samples in a D dimensional space.
- Assumption: For $n = 1, \ldots, N$

$$\mathbf{t}_n = \mathbf{D}\mathbf{c}_n + \boldsymbol{\epsilon}_n,$$

where $\boldsymbol{\epsilon}_n$ is additive Gaussian noise with zero mean and variance $\sigma^2\mathbf{I}$ specific to this data sample, the columns of the $D \times M$ matrix $\mathbf{D}$ represent latent features, and $\mathbf{c}_n$ is the sample specific vector of coefficients.

Indian Buffet Process

$$\underbrace{\mathbf{T}}_{N\times D} = \underbrace{\mathbf{C}}_{N\times M}\underbrace{\mathbf{D}^T}_{M\times D} + \underbrace{\boldsymbol{E}}_{N\times D}.$$

- Each entry of $\mathbf{E}$ is drawn from a normal distribution with zero mean and variance σ^2.
- $\mathbf{D}$ is the *factor loading matrix* and its columns are the *factor loadings*.
- $\mathbf{C}$ is the *coordinate matrix*.

Indian Buffet Process

- Number of columns in $\mathbf{C}$ and number of rows in $\mathbf{D}^T$, M, can be infinite.
- Constraint: only a finite, small number of entries in each row of $\mathbf{C}$ are non-zero.
- The amount of data is finite and each data sample is generated as a finite linear combination.
- $\mathbf{C}$ is written in terms of a *binary indicator matrix* $\mathbf{Z}$ and a *weight matrix* $\mathbf{W}$:

$$\mathbf{C} = \mathbf{Z} \odot \mathbf{W},$$

where $\odot$ denotes the element-wise, *Hadamard product*.

Indian Buffet Process

Indian Buffet Process

- Customers are indexed by $1, \ldots, n, \ldots, N$.
- Infinitely many dishes, but not all have been chosen yet (perhaps not even prepared yet).
- M is the number of different dishes chosen so far.

Indian Buffet Process

- The number of dishes i_1 the first customer chooses is drawn from a *Poisson distribution* with rate λ, which is the expected number of dishes each customer chooses.
- In matrix $\mathbf{Z}$, the first i_1 entries of the first row are set to one.
- $M = i_1$.
- The amount he takes from each dish is recorded in the first row of matrix $\mathbf{W}$.

Indian Buffet Process

- The n^{th} customer chooses from the current M dishes with probability j_m/n, if dish m was chosen j_m times.
- j_m is the number of non-zero entries in column m above row n in matrix $\mathbf{Z}$.
- If he chooses a particular dish m, then a one is placed in the m^{th} column of the n^{th} row of $\mathbf{Z}$.
- He also tries i_n new dishes, where i_n is drawn from a Poisson distribution with rate λ/n.
- Ones in columns with numbers $M+1, \ldots, M+i_n$ in the n^{th} row.
- M is updated to $M+i_n$.
- The amount of each dish is recorded in the n^{th} row of $\mathbf{W}$.

Indian Buffet Process

- Since λ/n decreases with every new customer, the probability that he will choose even just one new dish decreases, but does not become zero.
- If our model space does not have enough latent features to explain the data, there is a non-zero probability to create a feature which will help explain the data.

Indian Buffet Process

In each row, the expected number of non-zero entries is λ.

- True for the first row, since the expectation of a Poisson distribution is given by the rate λ.
- Assume that the expected number of non-zero entries for rows $1, \ldots, n-1$ is λ.
- The expected number of nonzero entries in the n^{th} row is

$$\begin{aligned}
\mathbb{E}[\sum_{m=1}^{M} \frac{j_m}{n}] + \frac{\lambda}{n} &= \frac{1}{n}\mathbb{E}[\sum_{m=1}^{M}\sum_{k=1}^{n-1} Z_{km}] + \frac{\lambda}{n} \\
&= \frac{1}{n}\sum_{k=1}^{n-1}\mathbb{E}[\sum_{m=1}^{M} Z_{km}] + \frac{\lambda}{n} \\
&= \frac{(n-1)\lambda}{n} + \frac{\lambda}{n} = \lambda,
\end{aligned}$$

where Z_{km} denotes the (k, m) entry in matrix $\mathbf{Z}$.

Indian Buffet Process

- *Rich-get-richer*, since the probability of choosing a dish depends on the number of times it has been the choice of any of the previous customers.
- Dishes chosen by many customers relate to *global features*.
- Dishes chosen only by a few customers indicate *local features*.

Indian Buffet Process

- A different order of customers means multiplying $\mathbf{Z}$ by a permutation matrix from the left which changes the row order.
- Swap columns so that the number of leading zeros in each column is increasing by a permutation matrix from the right.
- The same permutations are performed on $\mathbf{W}$.

Left-ordering of binary matrices

Left-ordering of binary matrices

- Given an order of customers, each column is interpreted as a binary number with the most significant bit in the first row and the least significant one in the last.
- Columns are ordered in decreasing magnitude.
- Two columns representing the same binary number means two dishes were chosen by the same set of customers.

Left-ordering of binary matrices

- Let K be the number of distinct dishes.
- Let m_k be the number of occurrences of the distinct column k for $k = 1, 2, \ldots, K$.
- Possible reordering of same columns: $\prod_{k=1}^{K} m_k!$.
- Number of different ways the same data can have been generated:

$$\frac{M!}{\prod_{k=1}^{K} m_k!}.$$

Indian Buffet Process

- This specifies the prior distribution $p(\mathbf{Z})$.
- For $\mathbf{W}$, a prior distribution is specified for each element, e.g. the normal distribution or the Laplace distribution.
- For $\mathbf{D}$, a prior distribution is specified for each column.

Reverse

- With $\mathbf{D}$ fixed, determine the posterior distribution of each row of $\mathbf{Z}$ and $\mathbf{W}$ for each data sample, given the other data samples.
- The dependence on the other samples is encoded in $\mathbf{D}$.
- Draw $\mathbf{z}_n$ and $\mathbf{w}_n$ from this distribution.
- With $\mathbf{Z}$ and $\mathbf{W}$ fixed, determine the posterior distribution for each column of $\mathbf{D}$.
- Draw $\mathbf{d}_m$ form this distribution.
- Repeat.