Recall that a basic approach for constructing non-linear function classes in using basis functions.

$$f(x, a) = \sum_{\ell=1}^{m} a_\ell \phi_\ell(x) = a^T \phi(x).$$

where $\phi(x) = [\phi_1(x), \dots, \phi_m(x)]^T$ is a set of basis functions that is believed to capture important information regarding the input, and $a = [a_1, \dots, a_m]^T$ is a coefficient vector applied on $\phi(x)$.

For example, in polynomial regression, we assume $\phi_\ell(x) = x^{\ell-1}$ and hence $f(x, a) = \sum_{\ell=0}^{m-1} a_\ell x^\ell$.

Neural Networks

Approach to non-linear Regression:

Fixed basis function:

$$f(x;w) = \sum_i w_i \phi_i(x)$$

e.g. polynomial regression: $f(x,w) = w_0 + w_1 x + w_2 x^2 + \dots$

Adaptive basis functions:

* Kernel method: $f(x,w) = \sum_{i \in data} w_i k(x, x^{(i)})$

* Neural Network: $f(x; w, v) = \sum_i w_i \phi(x; v_i)$

$$\phi_i(x) = \phi(x, v_i)$$

i^{th} neuron

coefficient of the basis functions (weight of neuron i)

Objective: $\min_{w,v} E_D\left[(y - f(x;w,v))^2\right]$.

Basis function (neuron) form: Each neuron ϕ_i is assumed to have the following form:

$$\phi_i(x) = \sigma\left(\sum_{l=1}^{d} v_l x_l + v_0\right), \quad \text{where } x = \begin{bmatrix} x^{(1)} \\ \vdots \\ x^{(d)} \end{bmatrix}, \quad v = \begin{bmatrix} v_1 \\ \vdots \\ v_d \end{bmatrix}$$

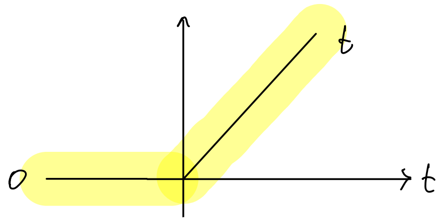
basis function aka neuron

activation function (non-linear).

each neuron ϕ_i has an activation function, each which is composed of summation of $v_l x_l + v_0$, $l = 1$ to d.

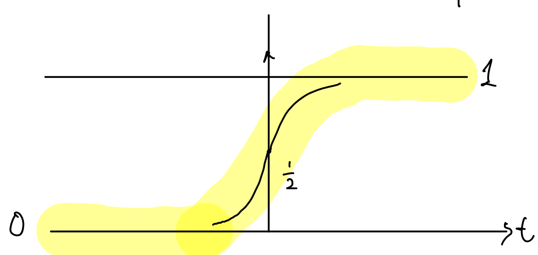
Examples of activation function:

1) ReLU (Rectified Linear Unit): $\sigma(t) = \max(0, t)$.



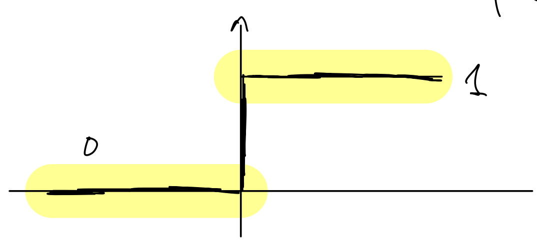
$$\sigma(t) = \begin{cases} 0, & t \geq 0 \\ t, & t > 0. \end{cases}$$

2) Sigmoid: $\sigma(t) = \dfrac{\exp(t)}{1 + \exp(t)} = \dfrac{1}{1 + \exp(-t)}$



3) Indicator:

$$\sigma(t) = \mathbb{I}(x > 0) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$$



neuron $\in \{1, \dots, m\}$

linear coefficient $\in \{1, \dots, m\}$ d = data point weight l of d for neuron i: $v_{il} \in \{0, \dots, d\}$

$m \sim$ complexity of NN.

$$\underbrace{f(x; w, v)}_{\text{neural network}} = \underbrace{\sum_{i=1}^{M}}_{\text{summing all neurons from 1 to } m.} w_i \, \sigma \Big(\underbrace{\sum_{l=1}^{d} v_{il} x_l + v_{i0}}_{\text{each neuron has its own components, from 1 to d.}} \Big),$$

where $v_i = [v_{i0}, v_{i1}, \dots, v_{id}]^T$ // vector

$V = [v_1, \dots, v_m]$ // matrix

<u>Loss function</u> : $\min_{w,v} E_D\left[\left(y - f(x; w, v)\right)^2\right]$.

use GD or SGD.

Learning a neural network is a nonconvex optimization problem! No closed form, can only find local minima.

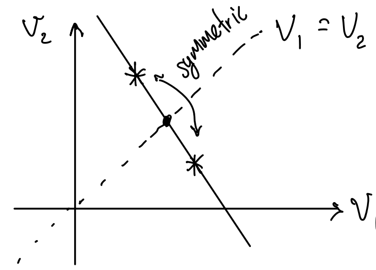
Example: A NN with 2 neurons:

$$f(x, v) = \sigma(x - v_1) + \sigma(x - v_2)$$

$$L(v) = E_D\left[\left(y - \sigma(x - v_1) - \sigma(x - v_2)\right)^2\right].$$

Assume $[v_1^*, v_2^*]$ is optimal.

Then $[v_2^*, v_1^*]$ is also optimal.



If $L(v)$ is convex,

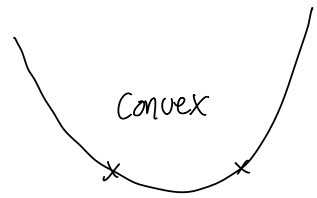
then $[\bar{v}^*, \bar{v}^*]$ is also optimal,

where $\bar{v}^* =$ average of the 2.

$$= \frac{v_1^* + v_2^*}{2}$$

mostly

usually not



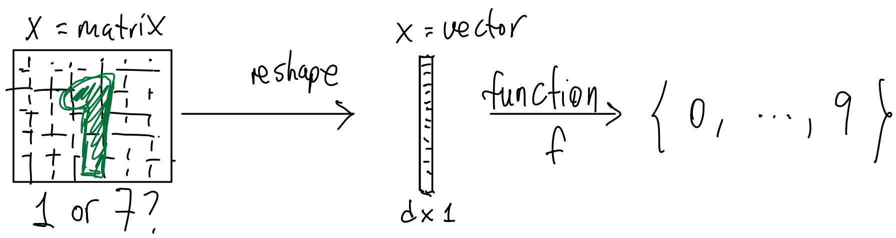
Gradient Descent: To solve the optimization.

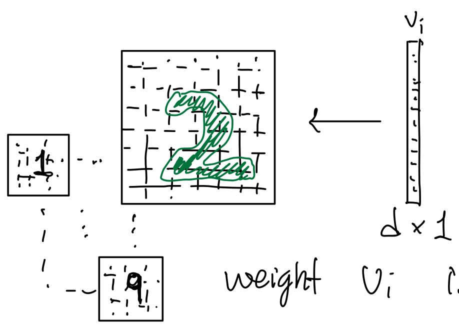
$$\min_{w,v} L(w,v) : \begin{cases} w^{new} = w^{old} - \epsilon \nabla_w L(w,v) \\ v^{new} = v^{old} - \epsilon \nabla_v L(w,v) \end{cases}$$

- Initialization is critical. Because where you initialize decides which local min you end up with.

- Initialize V to be "random small values".
- Initialize the neurons differently.
- Be careful with initialization of neurons and step sizes.

Example: Digit Classification:





$$\sum_{i=1}^{m} w_i \sigma \left(v_i^T x - v_0\right)$$

This depends on the activation function:

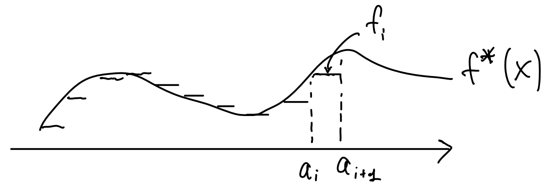
weight v_i is estimated by minimizing the loss.

Take inner product of X (image) with v_i (weights) and see which one has the least loss. That is the closest template.

If $\sigma = \mathbb{I}$: $\sum_{i=1}^{m} w_i \mathbb{I}(v_i^T x \geq v_0)$ // if inner product $> V_0$, then neuron fires.

Theoretical Results

If you have infinite neurons, $f(x; w, v)$ can approximate arbitrary nonlinear functions:
If m is large enough, then the NN $f(x, w, v)$ can approximate any continous function.



can approximate using step (indicator) function:

$$\sum_{i=1}^{m} f_i \mathbb{I}(a_i \leq x \leq a_{i+1})$$

$$= \sum_{i=1}^{m} f_i \left(\mathbb{I}(x \geq a_i) - \mathbb{I}(x \geq a_{i+1})\right)$$

$$= \sum_{i=1}^{m} (f_i - f_{i-1}) \mathbb{I}(x \geq a_i)$$

$\Rightarrow$ There must exist some parameter that can approximate a continous function arbitrarily well.

using graph reprosentation is easier.

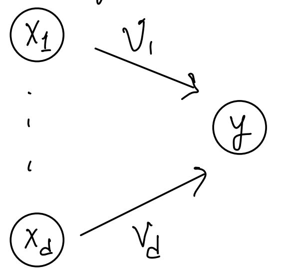
Graphical Representation of NN:

Similar to Gaussian Markov Field.

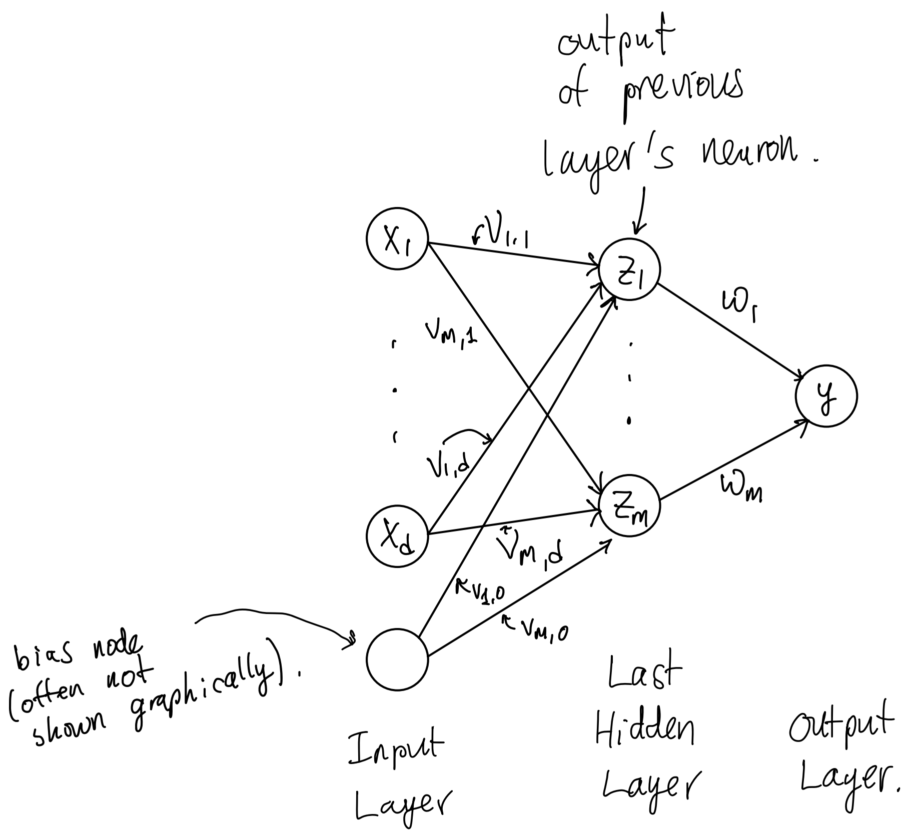
we can illustrate this using linear regression:

data points

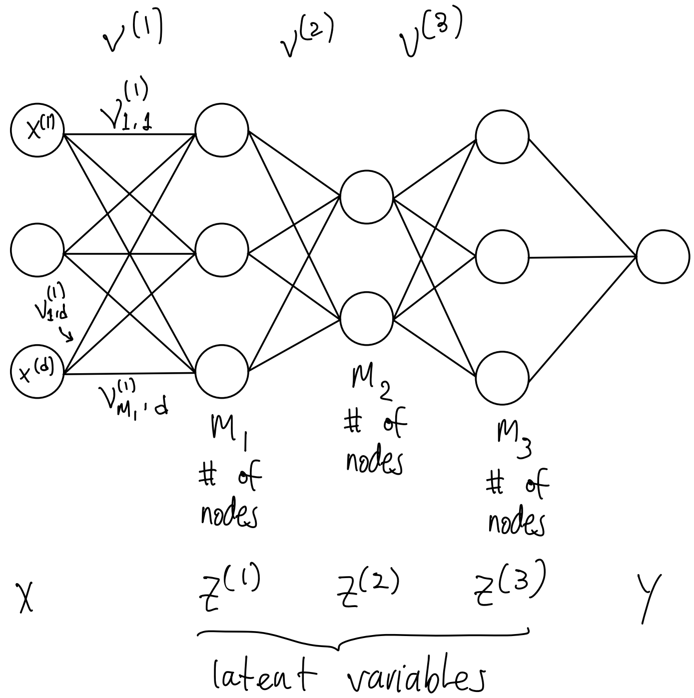
$$y = \sum_{l=1}^{d} v_l x_l$$



$$y = \sum_{i=1}^{M} w_i \, \underbrace{\sigma\left(\sum_{l=1}^{d} v_{i,l} \, x_l + v_0 \right)}_{\triangleq \; z_i \; = \; \text{output of each neuron.}}$$



If multiple hidden layers : Deep neural network.



$$Z^{(1)} = \Phi(X, V^{(1)}) = \begin{bmatrix} \sigma((v_1^{(1)})^T x) \\ \vdots \\ \sigma((v_{M_1}^{(1)})^T x) \end{bmatrix}$$

← first neuron in $Z^{(1)}$

size: $M_1 \times d$.

← last neuron in $Z^{(1)}$.

$\vdots$

$$Z^{(2)} = \Phi(Z^{(1)}, V^{(2)}) = \begin{bmatrix} \sigma((v_1^{(2)})^T Z^{(1)}) \\ \vdots \\ \sigma(v_{M_2}^{(2)})^T Z^{(1)} \end{bmatrix}$$

← first neuron in $Z^{(2)}$.

size : $M_2 \times M_1$.

← last neuron in $Z^{(2)}$.

$$Z^{(3)} = I(Z^{(2)}, V^{(3)}) = \cdots$$

$$y = \sum_{i=1}^{M_3} w_i Z_i^{(3)}$$

<u>Backpropagation</u> = Calculation of gradients of DNNs.

Backpropagating the error from output to inputs.

Consider an example DNN where each layer has 1 neuron:

$$x \xrightarrow[v^{(1)}]{} z^{(1)} \xrightarrow[v^{(2)}]{} z^{(2)} \cdots \xrightarrow[v^{(m)}]{} y$$

$$L(v) = E_D\left[\left(y - f(x, v^{(1)} \ldots v^{(m)})\right)^2\right]$$

$$\nabla_{v^{(1)}} L(v) = 2 E_D\left[\left(f(x, v) - y\right) \frac{df(x, v)}{dv^{(1)}}\right] \quad \text{// chain rule.}$$

where

$$\frac{df(x, v)}{dv^{(1)}} = \frac{dy}{dv^{(1)}} = \left(\frac{dy}{dz^{m}}\right)\left(\frac{dz^{m}}{dz^{m-1}}\right) \cdots \left(\frac{dz^{3}}{dz^{2}}\right)\left(\frac{dz^{2}}{dz^{1}}\right)\left(\frac{dz^{1}}{dv^{1}}\right) \quad \text{// chain rule.}$$