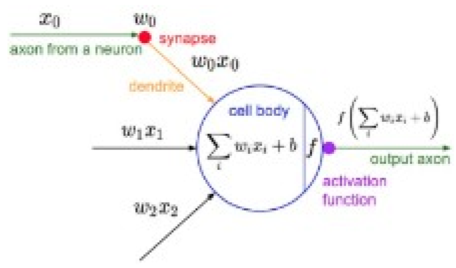
11. Artificial Neural Networks

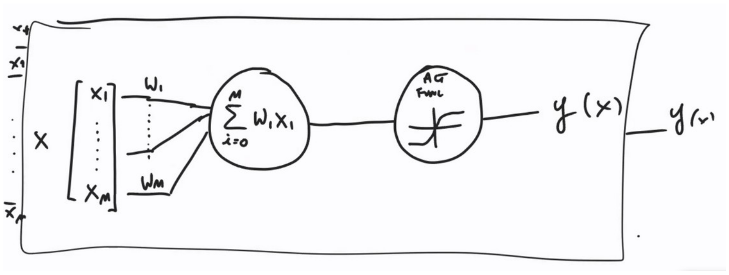
Approximate functions that take vectors as input and retrieve vectors as output

D $=\{(x_n,t_n)^N_{n=1}\}$ such that $t_n \approx f(x_n)$

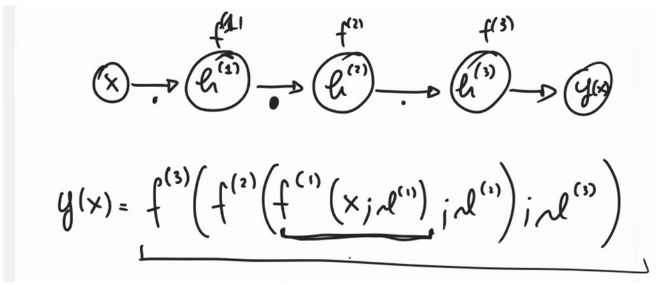
Define $y = \hat{f}(x;\theta)$ and learn parameters θ so that ˆf approximates f.

11.1 FeedForward Networks

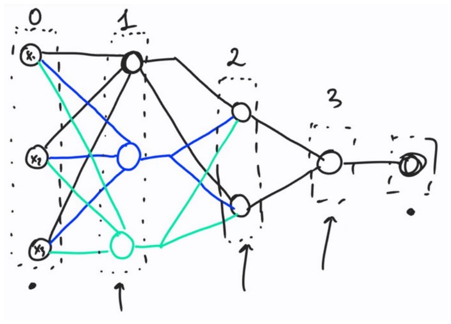




Layers have to take as input only the output of the previous layer. Intermediate layers are called hidden layers.



x
$f^{(1)}$
$h^{(1)}$
$f^{(2)}$
$h^{(2)}$
$f^{(3)}$
$h^{(3)}$
$y(x)$
$y(x) = f^{(3)}(f^{(2)}(f^{(1)}(x;\theta^{(1)});\theta^{(2)});\theta^{(3)})$



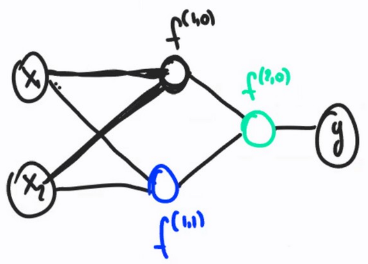
FNNs are chain structures.

The length of the chain is the depth of the network.

Final layer also called output layer.

Deep learning follows from the use of networks with a large number oflayers (large depth).

Why to use FNN → to model non linear models, the FNN learning manages complex combination of many parametric functions.



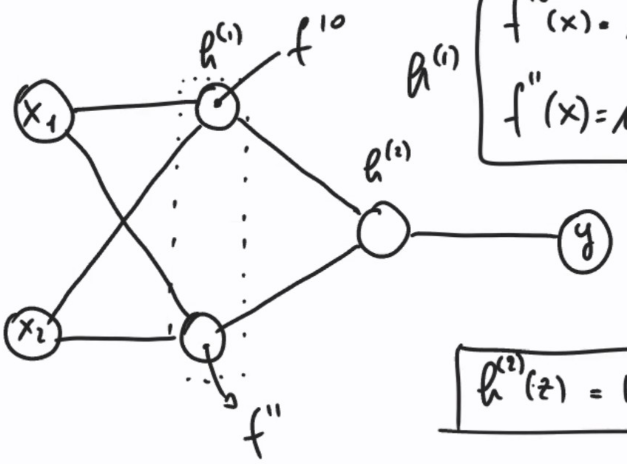
$$f^{(10)}(x_1,x_2) = f\left(w_1^{10}x_1 + w_2^{10}x_2 + w_0^{10}\right)$$

$$f^{(11)}(x_1,x_2) = f\left(w_1^{11}x_1 + w_2^{11}x_2 + w_0^{11}\right)$$

11.2 XOR Example

Not modelable with linear model → linear regression retrieve not optimal results.

Dataset:D $=\{((0,0)^T, 0), ((0,1)^T, 1), ((1,0)^T, 1), ((1,1)^T, 0)\}$



$h^{(1)}$
$$\boxed{\begin{aligned} f^{10}(x) &= \max\left(0; w_1^{10}x_1 + w_2^{10}x_2 + w_0^{10}\right) \\ f^{11}(x) &= \max\left(0; w_1^{11}x_1 + w_2^{11}x_2 + w_0^{11}\right) \end{aligned}}$$

$$\boxed{h^{(2)}(z) = w_1^2(z_1) + w_2^2(z_2) + w_0^2}$$

$$W = \begin{bmatrix} w_1^{10} & w_1^{11} \\ w_2^{10} & w_2^{11} \end{bmatrix} \qquad h^{(1)}(x) = g\left(W^T x + c\right) \qquad c = \begin{bmatrix} w_0^{10} \\ w_0^{11} \end{bmatrix}$$

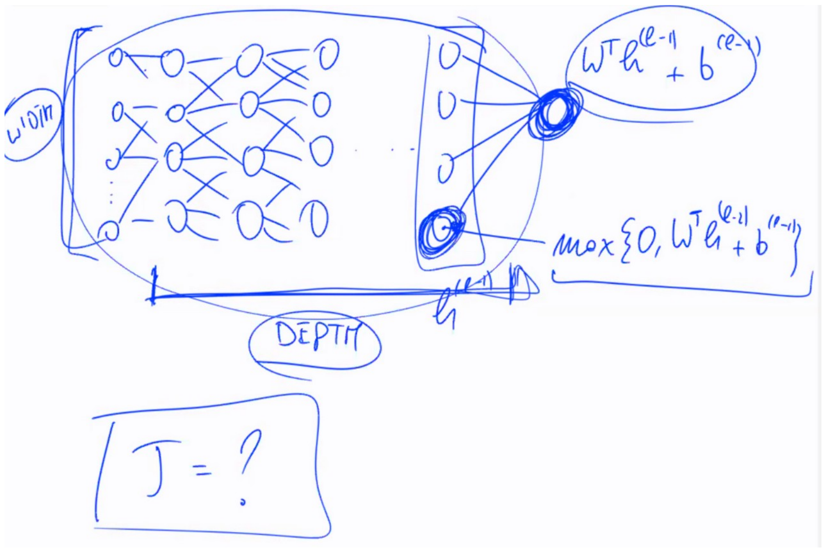
$$g(x) = \max(0; x)$$

$$h^{(2)}(x) = w^T x + b$$

$$y(x) = h^{(2)}\left(h^{(1)}(x)\right) = \underbrace{\left(w^T\left(\max\left(0; W^T x + c\right)\right) + b\right)}_{y(x;\theta)}$$

$$\boxed{\theta = \langle w, W, c, b \rangle}$$

Mean squared error (MSE) loss function:


WIDTH
$W^T h^{(l-1)} + b^{(l-1)}$
$\max\{0, W^T h^{(l-1)} + b^{(l-1)}\}$
$h^{(l-1)}$
DEPTH
$J = ?$

$$J(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^{N}(t_n - y(\mathbf{x}_n))^2$$

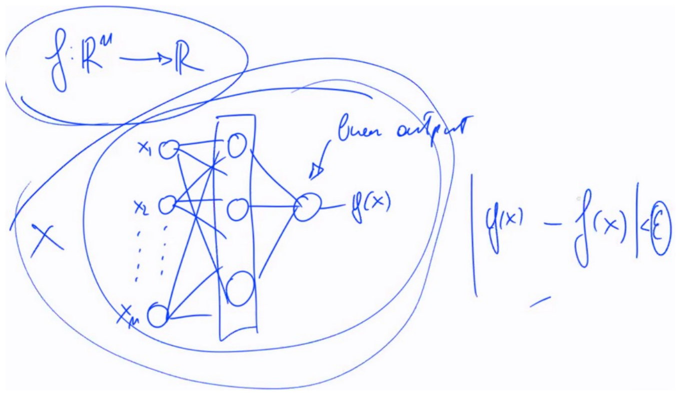
Solution:

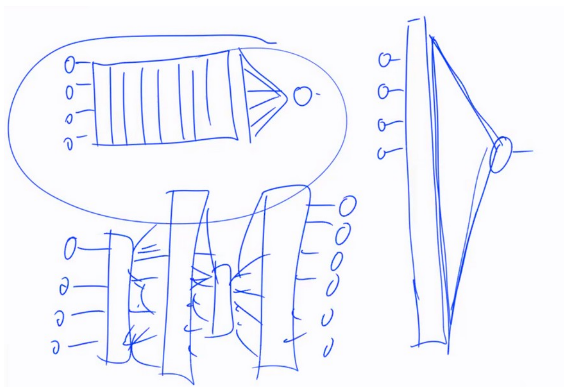
$$\mathbf{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \mathbf{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \mathbf{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, b = 0$$

11.3 Architectural design

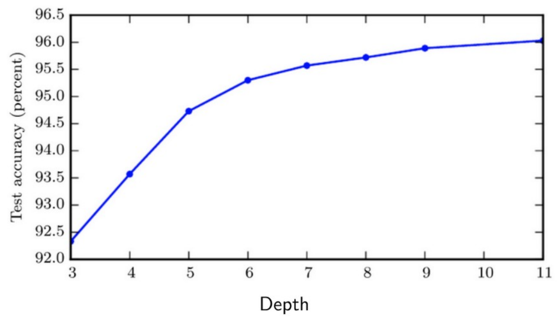
11.3.1 Depth

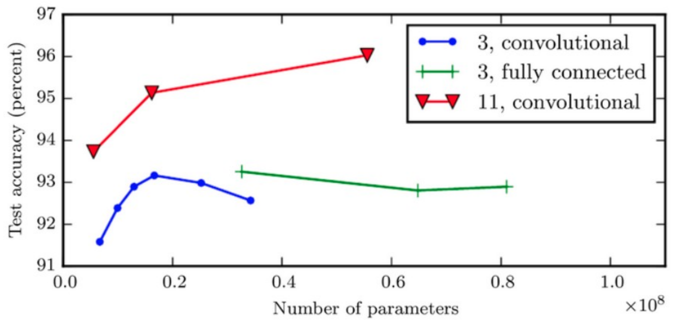
Universal approximation theorem: a FFN with a linear output layer (returns linear combination of hidden layer) and at least one hidden layer with any “squashing” activation function (e.g.,sigmoid) can approximate any Borel measurable function with any desired amount of error, provided that enough hidden units are used.It works also for other activation functions (e.g., ReLU)





Note: It’s better increase numer of layer than the number of component for each layer.





11.3.2 Cost function

Model implicitly defines a conditional distributionp(t|x,θ) (Probability that an input from output instance space, in the output provides by the network correspond with the one provided by the function).

Cost function: Maximum likelihood principle (cross-entropy)

$$J(\boldsymbol{\theta}) = E_{\mathbf{x},\mathbf{t}\sim\mathcal{D}}\left[\underbrace{-\ln(p(\mathbf{t}|\mathbf{x},\boldsymbol{\theta}))}_{\text{Minimize it.}}\right]$$

Example:
Assuming additive Gaussian noise we have

$$p(\mathbf{t}|\mathbf{x},\boldsymbol{\theta}) = \mathcal{N}(\mathbf{t}|\overset{\text{Centered.}}{\underbrace{f(\mathbf{x};\boldsymbol{\theta})}_{\text{function you are learning.}}}, \beta^{-1}I)$$

Mean of the normal distribution.

and hence

$$J(\boldsymbol{\theta}) = E_{\mathbf{x},\mathbf{t}\sim\mathcal{D}}\left[\frac{1}{2}||\mathbf{t} - f(\mathbf{x};\boldsymbol{\theta})||^2\right]$$

Maximum likelihood estimation with Gaussian noise corresponds to mean squared error minimization.

11.4 Output units activation functions

In case of:

1. Regression

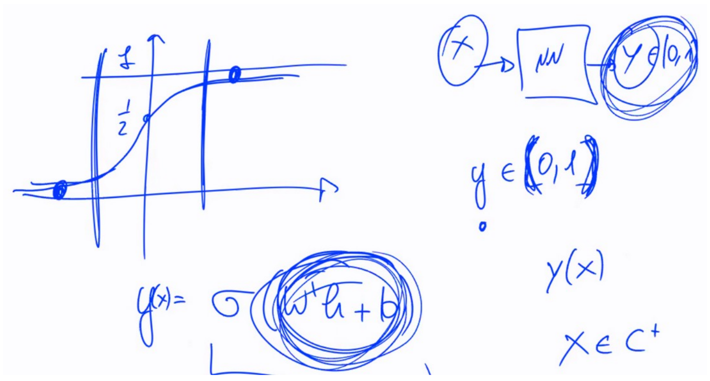
Output is the linear cobination of the inputs

Likelihood → Gaussian

$y = W^t h + b$, using Gaussian → $p(t|x) = N(t|y,B^{-1})$

Cost function: maximum likelihood (cross-entropy) that is equivalent to minimizing mean squared error.

2. Binary classification



Sigmoid units: $y=\sigma(w^T h+b)$

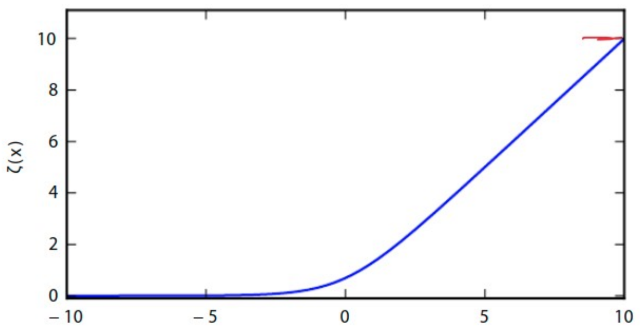
x is hidden in h

Likelihood → Bernoulli

$J(\theta) = E_{x,t\sim D}[-\ln p(t|x)]$

$-\ln p(t|x) = -\ln \sigma(\alpha)^t (1-\sigma(\alpha))^{1-t} = -\ln \sigma((2t-1)\alpha) = \text{softplus}((1-2t)\alpha)$

with $\alpha = wTh+b$.



The softplus function

3. Multi-class classification

Softmax units: $y_i = \text{softmax}(\alpha_i) = \exp(\alpha_i)/\sum_j \exp(\alpha_j)$

Likelihood –> Multinomial

$J_i(\theta) = E_{x,t\sim D}[-\ln \text{softmax}(\alpha_i)]$ with $\alpha_i = w^T_i h + b_i$.

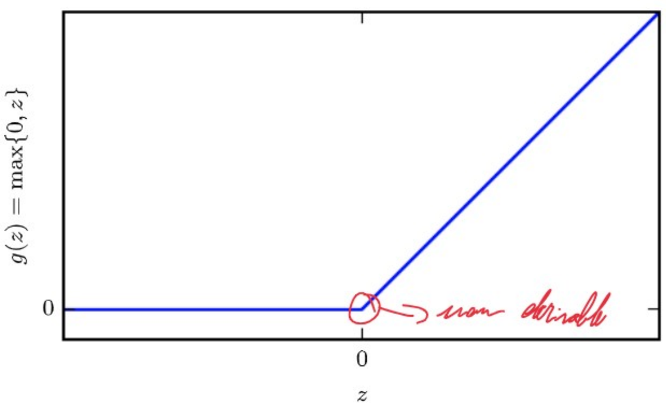
4.(bonus) Hidden unit activation functions

Since predicting which activation function will work best is usually impossible, we use:

Rectified Linear Units(ReLU):

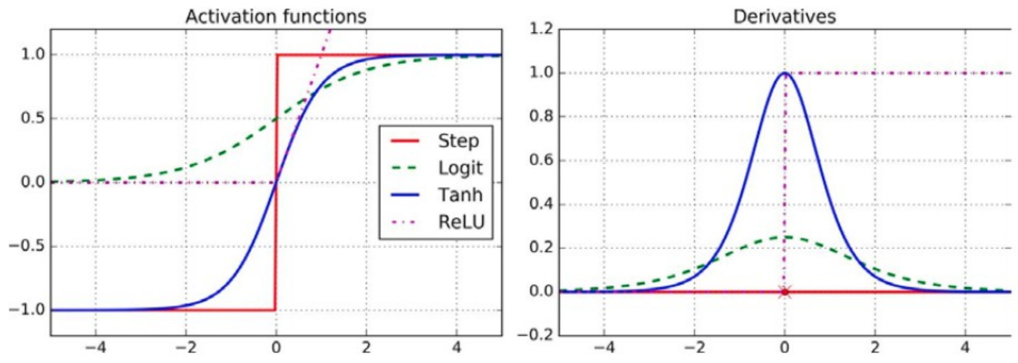
$g(\alpha) = \max(0,\alpha)$. ($\alpha$ is the linear comb. of the inputs).

Easy to optimize, not differentiable at 0

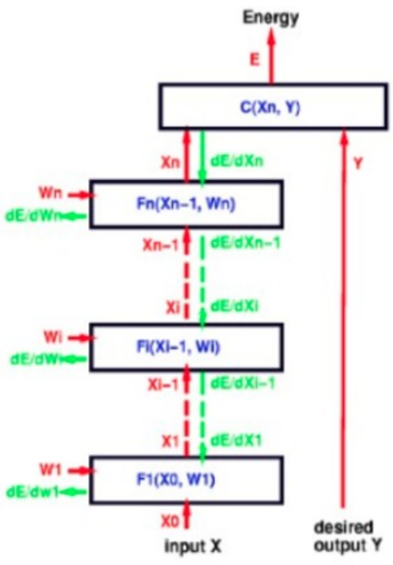


Sigmod and hyperbolic tangent:

g(α) =σ(α) and g(α) = tanh(α) Closely related astanh(α) = 2σ(2α)−1.



11.5 Gradient computation



To train the network we need tocompute the gradients with respectto the network parameters **θ**.

Back-propagation or backprop algorithm is used to propagate gradient computation from the cost through the whole network.

back-propagation is only used to compute the gradients

back-propagation is not a training algorithm

back-propagation is not specific to FNNs

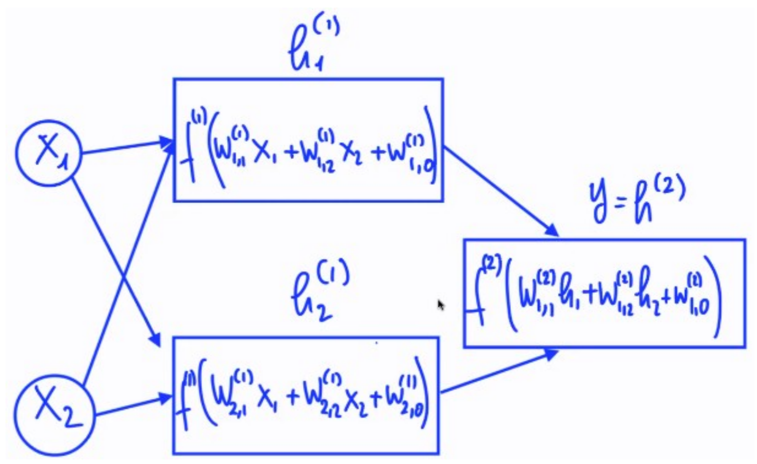
Output depends on the parameters of the network (weights).

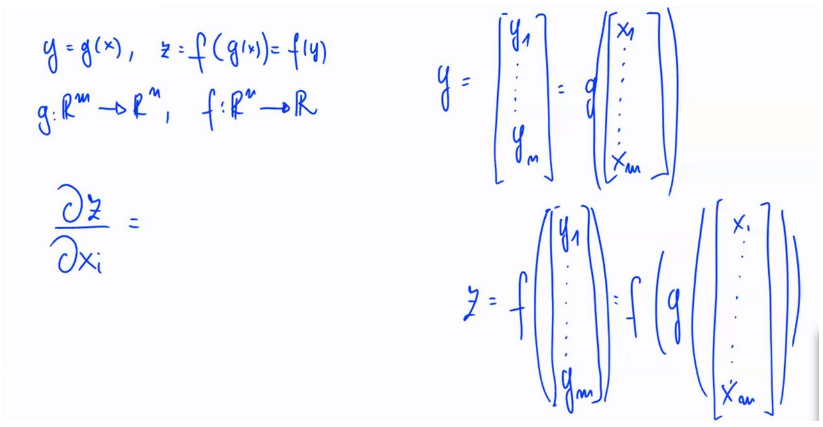
Goal: find the weights to minimize difference from wanted and obtained

Backpropagation is just a preliminary step to compute the weights.

You are learning the network when you minimize lost function.

Example of back propagation:


$h_1^{(1)}$
X_1
X_2
$f^{(1)}(W_{1,1}^{(1)}x_1 + W_{1,2}^{(1)}x_2 + W_{1,0}^{(1)})$
$h_2^{(1)}$
$f^{(1)}(W_{2,1}^{(1)}x_1 + W_{2,2}^{(1)}x_2 + W_{2,0}^{(1)})$
$y = h^{(2)}$
$f^{(2)}(W_{1,1}^{(2)}h_1 + W_{1,2}^{(2)}h_2 + W_{1,0}^{(2)})$


$y = g(x), \quad z = f(g(x)) = f(y)$
$g: \mathbb{R}^m \to \mathbb{R}^n, \quad f: \mathbb{R}^n \to \mathbb{R}$
$\frac{\partial z}{\partial x_i} =$
$y = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} = g\left(\begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}\right)$
$z = f\left(\begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}\right) = f\left(g\left(\begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}\right)\right)$

Compute the gradient of the cost function w.r.t. the parameters $\nabla_\theta J(\theta)$

Chain rule

Let: $y = g(x)$ and $z = f(g(x)) = f(y)$
Applying the chain rule we have:

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

For vector functions, $g : \mathbb{R}^m \mapsto \mathbb{R}^n$ and $f : \mathbb{R}^n \mapsto \mathbb{R}$ we have:

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i},$$

equivalently in vector notation:

$$\nabla_{\mathbf{x}} z = \left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}}\right)^T \nabla_{\mathbf{y}} z,$$

with $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ the $n \times m$ Jacobian matrix of g.

Gradient is the genralization of partial derivative from unidimensional to multidimensional.

It tells us how the components of the function grows when you get far from the selected point in term of distance.

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \qquad f : \mathbb{R}^2 \to \mathbb{R} \qquad y : \mathbb{R}^m \to \mathbb{R}^n$$

$$\nabla_x f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \\ \frac{\partial f}{\partial x_2} \end{bmatrix} \qquad \frac{dy}{dx} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_m} \\ \vdots & \ddots & \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_m} \end{bmatrix}$$

Think of it as in unidimensional.

back-prop can be used either to produce an expression for the output of the function or one for the gradient.

Algorithm

```
Forward step
Require: Network depth $l$
Require: $W^{(i)}, i \in \{1, \dots, l\}$ weight matrices
Require: $\mathbf{b}^{(i)}, i \in \{1, \dots, l\}$ bias parameters
Require: $\mathbf{x}$ input value
Require: $\mathbf{t}$ target value
  $h^{(0)} = \mathbf{x}$
  for $k = 1, \dots, l$ do
    $\boldsymbol{\alpha}^{(k)} = \mathbf{b}^{(k)} + W^{(k)}\mathbf{h}^{(k-1)}$
    $\mathbf{h}^{(k)} = f(\boldsymbol{\alpha}^{(k)})$
  end for
  $\mathbf{y} = \mathbf{h}^{(l)}$
  $J = L(\mathbf{t}, \mathbf{y})$
```

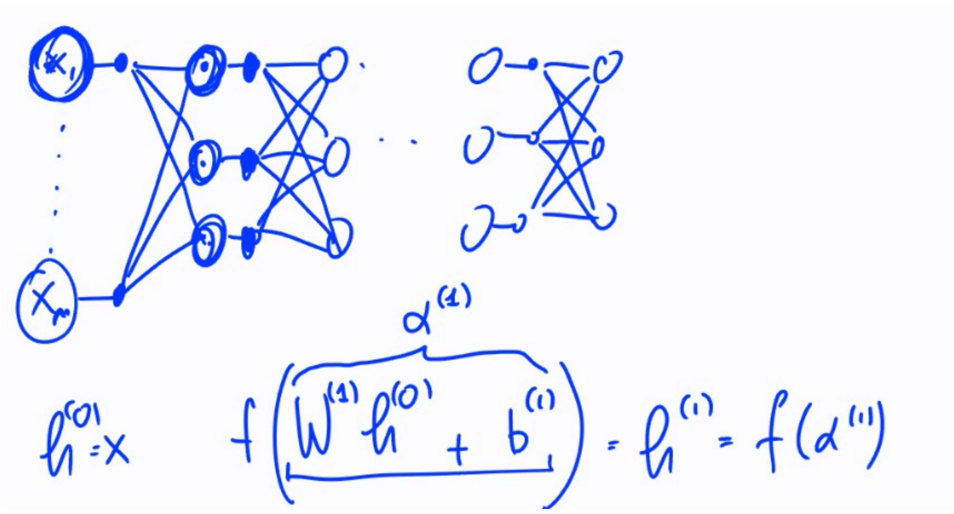
(This version of backprop is specific for fully connected MLPs.)

```
Backward step
  $\mathbf{g} \leftarrow \nabla_{\mathbf{y}} J = \nabla_{\mathbf{y}} L(\mathbf{t}, \mathbf{y})$
  for $k = l, l-1, \dots, 1$ do
    Propagate gradients to the pre-nonlinearity activations:
    $\mathbf{g} \leftarrow \nabla_{\boldsymbol{\alpha}^{(k)}} J = \mathbf{g} \odot f'(\boldsymbol{\alpha}^{(k)})$ {$\odot$ denotes elementwise product}
    $\nabla_{\mathbf{b}^{(k)}} J = \mathbf{g}$
    $\nabla_{W^{(k)}} J = \mathbf{g}(\mathbf{h}^{(k-1)})^T$
    Propagate gradients to the next lower-level hidden layer:
    $\mathbf{g} \leftarrow \nabla_{\mathbf{h}^{(k-1)}} J = (W^{(k)})^T \mathbf{g}$
  end for
```

More general versions for acyclic graphs exist.

Dynamic programming is used to avoid doing the same computations multiple times.

Gradients can be computed either in symbolic or numerical form.



$$h^{(0)} = x \qquad f\Big(\overbrace{\underbrace{W^{(1)} h^{(0)} + b^{(1)}}}^{\alpha^{(1)}}\Big) = h^{(1)} = f(\alpha^{(1)})$$

and soon in chain b$^{(1)}$ bias

$$\alpha^{(k)} = W^{(k)} h^{(k-1)} + b^{(k)} \qquad h^{(k)} = f(\alpha^{(k)})$$

f may change among layers

t,y referred to only one pair instance output.

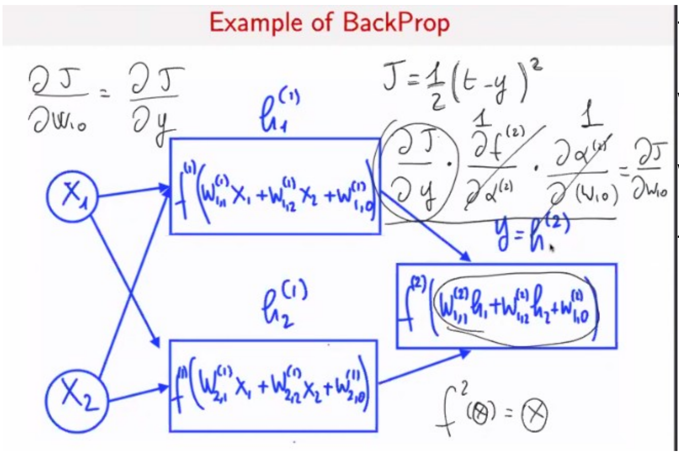
Our task is to compute all the partial derivative with respect to all these parameters because error change based on these parameters.

we proceed backward:

1. compute partial derivative referred to weights

2. apply product rule

for this specific network:


Example of BackProp
$\frac{\partial J}{\partial w_{10}} = \frac{\partial J}{\partial y}$
$J = \frac{1}{2}(t-y)^2$
$h_1^{(1)}$
$h_2^{(1)}$
x_1
x_2
$f^{(1)}(W_{1,1}^{(1)}x_1 + W_{1,2}^{(1)}x_2 + W_{1,0}^{(1)})$
$f^{(1)}(W_{2,1}^{(1)}x_1 + W_{2,2}^{(1)}x_2 + W_{2,0}^{(1)})$
$\frac{\partial J}{\partial y} \cdot \frac{\partial f^{(2)}}{\partial \alpha^{(2)}} \cdot \frac{\partial \alpha^{(2)}}{\partial (W_{10})} = \frac{\partial J}{\partial w_{10}}$
1
1
$y = h^{(2)}$
$f^{(2)}(W_{1,1}^{(2)}h_1 + W_{1,2}^{(2)}h_2 + W_{1,0}^{(2)})$
$f^2(x) = x$

f is linear, so → x

we know alfa2 that is the internal of f2

whenever you derive y you always get 1

then derivate j wrt of y

BackProp $J = \frac{1}{2}(t-y)^2$

$(t-y)(-1) = (y-t)$

new example

$$\frac{\partial J}{\partial w_{12}} = \frac{\partial J}{\partial y} \cdot \frac{\overset{1}{\cancel{\partial y}}}{\cancel{\partial \alpha^{(2)}}} \cdot \frac{\cancel{\partial \alpha^{(2)}}}{\cancel{\partial w_{1,2}}} \, \overset{h_2}{\underset{h_1}{h^{(1)}}} = (y - t) \cdot \overline{h_2}$$

it’s called backpropagation because we compute from the end to the beginning

Continue with the example…

$$\alpha_i^{(1)} = w_{i,0}^{(1)} + w_{i,1}^{(1)} x_1 + w_{i,2}^{(1)} x_2 \quad i = 1, 2$$

$$h_i^{(1)} = f^{(1)}(\alpha_i^{(1)}) = \texttt{ReLU}(\alpha_i^{(1)}) \quad i = 1, 2$$

$$\alpha^{(2)} = w_{1,0}^{(2)} + w_{1,1}^{(1)} h_1^{(1)} + w_{1,2}^{(1)} h_2^{(1)}$$

$$h^{(2)} = f^{(2)}(\alpha^{(2)}) = \alpha^{(2)}$$

$$y = h^{(2)}$$

Loss function MSE

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

$$\boldsymbol{\theta} = \langle w_{1,0}^{(1)}, w_{1,1}^{(1)}, w_{1,2}^{(1)}, w_{2,0}^{(1)}, w_{2,1}^{(1)}, w_{2,2}^{(1)}, w_{1,0}^{(2)}, w_{1,1}^{(2)}, w_{1,2}^{(2)} \rangle$$

Forward step

Given $x_1, x_2, w_{i,j}^{(k)}, t$

compute $\alpha_1^{(1)}, \alpha_2^{(1)}, \alpha^{(2)}, h_1^{(1)}, h_2^{(1)}, h^{(2)}, y, J = L(t, y)$

Backward step

Given $x_1, x_2, w_{i,j}^{(k)}, t, \alpha_1^{(1)}, \alpha_2^{(1)}, \alpha^{(2)}, h_1^{(1)}, h_2^{(1)}, h^{(2)}, y, J = L(t, y)$

compute $\frac{\partial J}{\partial w_{i,j}^{(k)}}$

Gradient computation

$$\frac{\partial J(\boldsymbol{\theta})}{\partial y} = \frac{\partial}{\partial y} \frac{1}{2}(t - y)^2 = y - t$$

$$\frac{\partial J(\boldsymbol{\theta})}{\partial w_{i,j}^{(2)}} = \frac{\partial J(\boldsymbol{\theta})}{\partial y} \frac{\partial y}{\partial w_{i,j}^{(2)}} \quad \text{with} \quad \frac{\partial y}{\partial w_{1,0}^{(2)}} = 1, \quad \frac{\partial y}{\partial w_{1,1}^{(2)}} = h_1^{(1)} \quad \frac{\partial y}{\partial w_{1,2}^{(2)}} = h_2^{(1)}$$

$$\frac{\partial J(\boldsymbol{\theta})}{\partial h_i^{(1)}} = \frac{\partial J(\boldsymbol{\theta})}{\partial y} \frac{\partial y}{\partial h_i^{(1)}} \quad \text{with} \quad \frac{\partial y}{\partial h_1^{(1)}} = w_{1,1}^{(2)} \quad \frac{\partial y}{\partial h_2^{(1)}} = w_{1,2}^{(2)}$$

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \alpha_i^{(1)}} = \frac{\partial J(\boldsymbol{\theta})}{\partial h_i^{(1)}} \frac{\partial h_i^{(1)}}{\partial \alpha_i^{(1)}} = \frac{\partial J(\boldsymbol{\theta})}{\partial h_i} \texttt{step}(\alpha_i^{(1)})$$

$$\frac{\partial J(\boldsymbol{\theta})}{\partial w_{i,j}^{(1)}} = \frac{\partial J(\boldsymbol{\theta})}{\partial \alpha_i^{(1)}} \frac{\partial \alpha_i^{(1)}}{\partial w_{i,j}^{(1)}} \quad \text{with} \quad \frac{\partial \alpha_i^{(1)}}{\partial w_{i,0}^{(1)}} = 1, \quad \frac{\partial \alpha_i^{(1)}}{\partial w_{i,1}^{(1)}} = x_1 \quad \frac{\partial y}{\partial w_{i,2}^{(1)}} = x_2$$

Gradient computation (in vector notation)

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$\mathbf{W}^{(1)} = \begin{bmatrix} w_{1,1}^{(1)} w_{1,2}^{(1)} \\ w_{2,1}^{(1)} w_{2,2}^{(1)} \end{bmatrix}, \mathbf{b}^{(1)} = \begin{bmatrix} w_{1,0}^{(1)} \\ w_{2,0}^{(1)} \end{bmatrix}$$

$$\mathbf{W}^{(2)} = \begin{bmatrix} w_{1,1}^{(2)} w_{1,2}^{(2)} \end{bmatrix}, \mathbf{b}^{(2)} = \begin{bmatrix} w_{1,0}^{(2)} \end{bmatrix}$$

$$f^{(1)}(z) = \texttt{ReLU}(z),\ f^{(2)}(z) = z$$

$$\mathbf{h}^{(0)} = \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$\boldsymbol{\alpha}^{(1)} = \begin{bmatrix} \alpha_1^{(1)} \\ \alpha_2^{(1)} \end{bmatrix} = \mathbf{W}^{(1)}\mathbf{h}^{(0)} + \mathbf{b}^{(1)}$$

$$\mathbf{h}^{(1)} = \begin{bmatrix} h_1^{(1)} \\ h_2^{(1)} \end{bmatrix} = f^{(1)}(\boldsymbol{\alpha}^{(1)}) = \begin{bmatrix} f^{(1)}(\alpha_1^{(1)}) \\ f^{(1)}(\alpha_2^{(2)}) \end{bmatrix} = \begin{bmatrix} \texttt{ReLu}(\alpha_1^{(1)}) \\ \texttt{ReLu}(\alpha_2^{(2)}) \end{bmatrix}$$

$$\boldsymbol{\alpha}^{(2)} = \left[\alpha^{(2)}\right] = \mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}$$

$$\mathbf{h}^{(2)} = \left[h^{(2)}\right] = f^{(2)}(\boldsymbol{\alpha}^{(2)}) = \left[f^{(2)}(\alpha^{(2)})\right] = \left[\alpha^{(2)}\right] = \boldsymbol{\alpha}^{(2)}$$

$$\mathbf{y} = \mathbf{h}^{(2)} = \boldsymbol{\alpha}^{(2)}$$

$$\mathbf{g} \leftarrow \nabla_{\mathbf{h}^{(2)}} J = \nabla_y J = \nabla_y \tfrac{1}{2}(t-y)^2 = \tfrac{\partial(\frac{1}{2}(t-y)^2)}{\partial y} = y - t$$

$$\mathbf{g} \leftarrow \nabla_{\alpha^{(2)}} J = \mathbf{g} \odot f^{(2)\prime}(\alpha^{(2)}) = \mathbf{g} \odot \tfrac{\partial \alpha^{(2)} - t}{\partial \alpha^{(2)}} = \mathbf{g} \odot 1 = \mathbf{g}$$

$$\nabla_{\mathbf{b}^{(2)}} J \leftarrow \tfrac{\partial J}{\partial w_{1,0}^{(2)}} = \mathbf{g}$$

$$\nabla_{\mathbf{W}^{(2)}} J \leftarrow \begin{bmatrix} \frac{\partial J}{\partial w_{1,1}^{(2)}} & \frac{\partial J}{\partial w_{1,2}^{(2)}} \end{bmatrix} = \mathbf{g} \cdot (\mathbf{h}^{(1)})^T$$

$$\mathbf{g} \leftarrow \nabla_{\mathbf{h}^{(1)}} J = \begin{bmatrix} \frac{\partial J}{\partial h_1^{(1)}} \\ \frac{\partial J}{\partial h_2^{(1)}} \end{bmatrix} = (\mathbf{W}^{(2)})^T \cdot \mathbf{g}$$

$$\mathbf{g} \leftarrow \nabla_{\boldsymbol{\alpha}^{(1)}} J = \mathbf{g} \odot f^{(1)'}(\boldsymbol{\alpha}^{(1)}) = \mathbf{g} \odot \begin{bmatrix} \frac{\partial \texttt{ReLU}(\alpha_1^{(1)})}{\partial \alpha_1^{(1)}} \\ \frac{\partial \texttt{ReLU}(\alpha_2^{(1)})}{\partial \alpha_2^{(1)}} \end{bmatrix} = \mathbf{g} \odot \begin{bmatrix} \texttt{step}(\alpha_1^{(1)}) \\ \texttt{step}(\alpha_2^{(1)}) \end{bmatrix}$$

$$\nabla_{\mathbf{b}^{(1)}} J \leftarrow \begin{bmatrix} \frac{\partial J}{\partial w_{1,0}^{(1)}} \\ \frac{\partial J}{\partial w_{2,0}^{(2)}} \end{bmatrix} = \mathbf{g}$$

$$\nabla_{\mathbf{W}^{(1)}} J \leftarrow \begin{bmatrix} \frac{\partial J}{\partial w_{1,1}^{(1)}} & \frac{\partial J}{\partial w_{1,2}^{(1)}} \\ \frac{\partial J}{\partial w_{2,1}^{(1)}} & \frac{\partial J}{\partial w_{2,2}^{(1)}} \end{bmatrix} = \mathbf{g} \cdot (\mathbf{h}^{(1)})^T$$

11.6 Stochastic Gradient Descent

Now we want to train the network use stochastic

```
Require: Learning rate $\eta \geq 0$
Require: Initial values of $\boldsymbol{\theta}^{(1)}$
  $k \leftarrow 1$
  while stopping criterion not met do
    Sample a subset (minibatch) $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$ of $m$ examples from
    the dataset $D$
    Compute gradient estimate: $\mathbf{g} = \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}^{(k)}), \mathbf{t}^{(i)})$
    Apply update: $\boldsymbol{\theta}^{(k+1)} \leftarrow \boldsymbol{\theta}^{(k)} - \eta \mathbf{g}$
    $k \leftarrow k + 1$
  end while
```

Observe: $\nabla_{\boldsymbol{\theta}} L(f(\mathbf{x}; \boldsymbol{\theta}), \mathbf{t})$ obtained with backprop

We start with some randome values of the parameters (typically small).

Training rate is not static, reduce it during the execution.

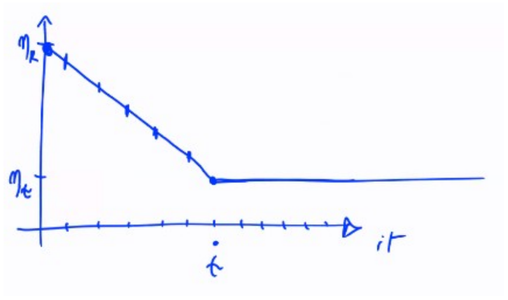
η usually changes according to some rule through the iterations

Until iteration τ ($k \leq \tau$):

$$\eta^{(k)} = \left(1 - \frac{k}{\tau}\right)\eta^{(k)} + \frac{k}{\tau}\eta^{(\tau)}$$

After iteration τ ($k > \tau$):

$$\eta^{(k)} = \eta^{(\tau)}$$



Momentum can accelerate learning
Motivation: Stochastic gradient can largely vary through the iterations

```
Require: Learning rate $\eta \geq 0$
Require: Momentum $\mu \geq 0$
Require: Initial values of $\boldsymbol{\theta}^{(1)}$
  $k \leftarrow 1$
  $\mathbf{v}^{(1)} \leftarrow 0$
  while stopping criterion not met do
    Sample a subset (minibatch) $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$ of $m$ examples from
    the dataset $D$
    Compute gradient estimate: $\mathbf{g} = \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}^{(k)}), \mathbf{t}^{(i)})$
    Compute velocity: $\mathbf{v}^{(k+1)} \leftarrow \mu\mathbf{v}^{(k)} - \eta\mathbf{g}$, with $\mu \in [0, 1)$
    Apply update: $\boldsymbol{\theta}^{(k+1)} \leftarrow \boldsymbol{\theta}^{(k)} + \mathbf{v}^{(k+1)}$
    $k \leftarrow k + 1$
  end while
```

Only change on velocity parameter → as you learn you keep track of the rate of change.

We apply a momentum to our gradient.

Idea: you want to make gradient reduce smoothly.

velocity → change rate (called momentum)→ get it from the next iteration.

Momentumμmight also increase according to some rule through theiterations.

11.6.1 SGD with Nesterov momentum

Momentum is applied before computing the gradient

$$\tilde{\boldsymbol{\theta}} = \boldsymbol{\theta}^{(k)} + \mu\mathbf{v}^{(k)}$$

Sometimes it improves convergence rate.
Based on analysis of the gradient of the loss function it

$$\mathbf{g} = \frac{1}{m}\nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\mathbf{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \mathbf{t}^{(i)})$$

is possible todetermine, at any step of the algorithm, whether the learning rate shouldbe increased or decreased.

11.7 Regularization

overfitting → model perform better on the data than on the train set.

we have to reduce it because it may be that adding a new instance it will not be classified well. to reduce overfit, add regularization term,

Add a regularization term E_{reg} to the cost function

$$E_{\text{reg}}(\boldsymbol{\theta}) = \sum_j |\theta_j|^q.$$

Resulting cost function:

$$\bar{J}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \lambda E_{\text{reg}}(\boldsymbol{\theta}).$$

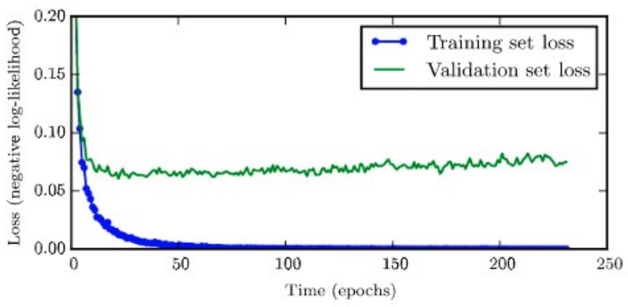
or augment dataset:

1. Data transforming (e.g. image rotation, scaling, varying, illumination conditions) by doing this you have a much larger dataset and it is more difficult to recognize it, so the model will be more “complete”.

2. Add noise

11.7.1 Early stopping

stop earlier to avoid overfitting



When to stop? Use cross-validation to determine best values.

11.7.2 Dropout

Randomly remove network units with some probability alfa

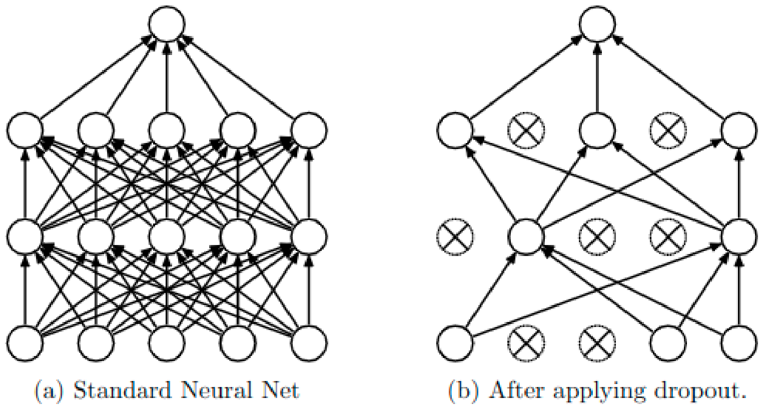
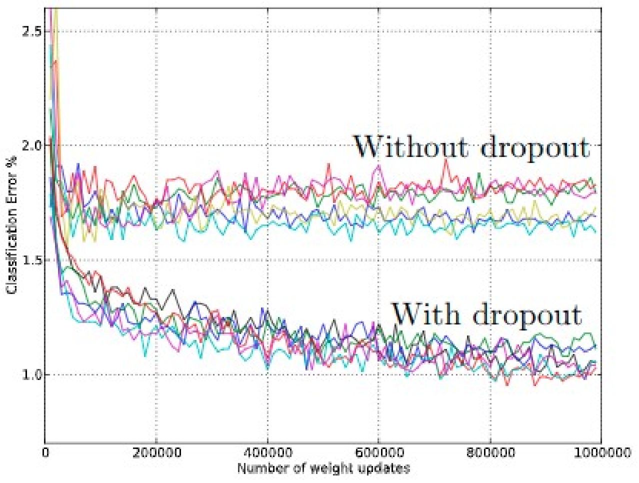


Image from Srivastava *et al.*. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting"



FNN → cascade model

in NN the central feature, they are able to learn transformation of the input (the precedents approaches not).