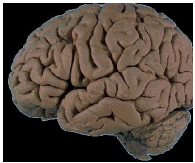
Topic 16: Neural networks

Mathews Jacob

December 9, 2020

Learn from the brain



Our brain is extremely good at classification

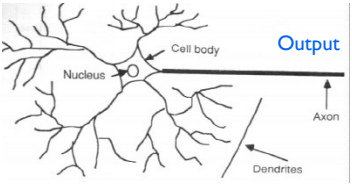
Human brain contains around 10 billion neurons

Each neuron is connected to 10,000 synapses

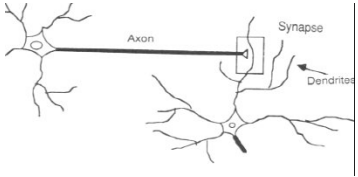
	processing elements	processing speed	style of computation	learns	intelligent, conscious
	10^{14} synapses	100 Hz	parallel, distributed	yes	yes
	10^{8} transistors	10^{9} Hz	serial, centralized	a little	not yet

How do neurons process data

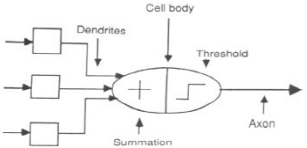
Neuron as an input output system



Massively connected

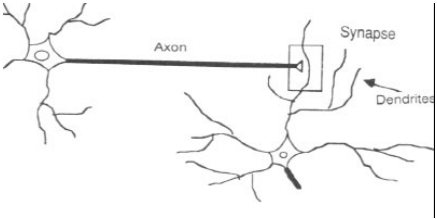


Model neurons on a computer



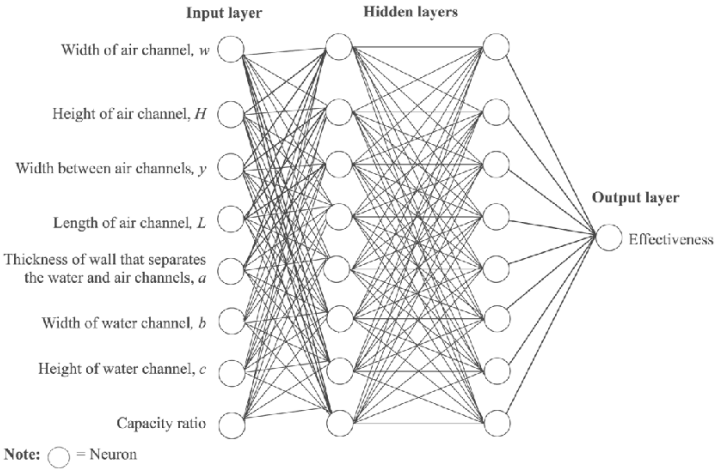
How does the brain learn ?

Synaptic plasticity

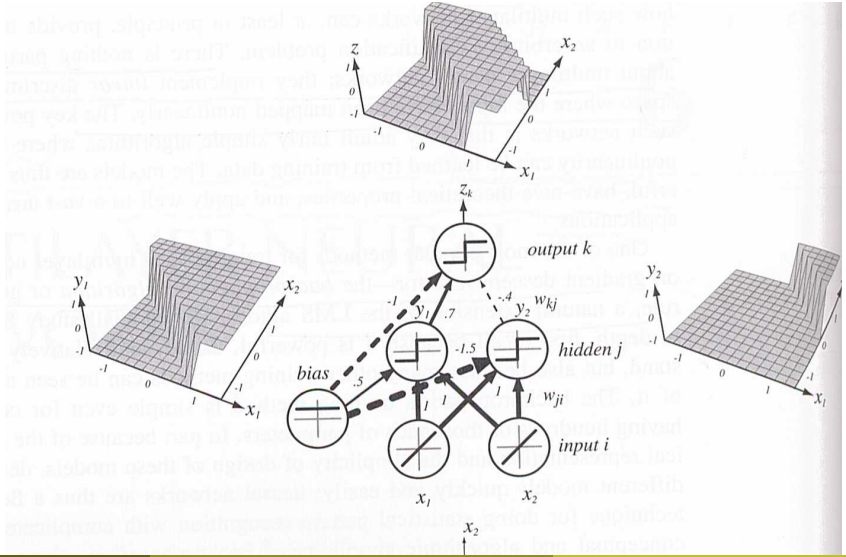


Change the strength of the connection based on activity

Multilayer neural networks

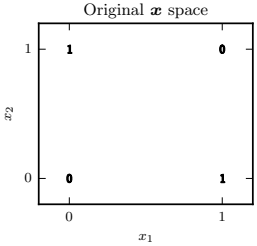


How does a neural network classify ?



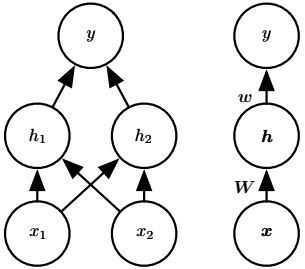
Feed forward neural network: first layer

XOR is not linearly separable



Two layer network

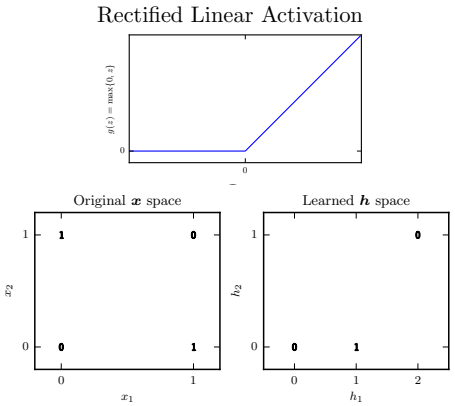
Network Diagrams



Two layer network

Original data is not linearly separable

$$
\begin{aligned}
f(\mathbf{x}, \mathbf{W}, \mathbf{c}, \mathbf{w}, b) &= \mathbf{w}^T \max\left(0, \mathbf{W}^T \mathbf{x} + \mathbf{c}\right) + b \\
\mathbf{W} &= \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \\
\mathbf{c} &= \begin{bmatrix} 0.5 \\ -1.5 \end{bmatrix} \\
\mathbf{w} &= \begin{bmatrix} 0.7 \\ -0.4 \end{bmatrix} \\
\mathbf{b} &= -1
\end{aligned}
$$

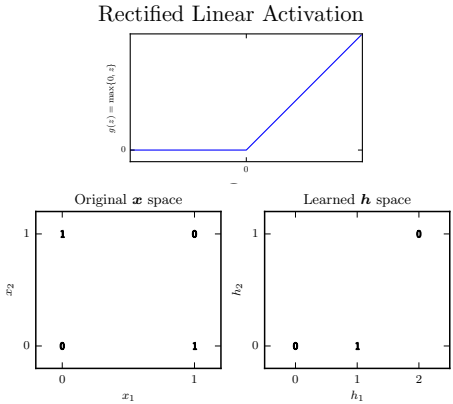


Two layer network

Hidden later data is linearly separable

$$\mathbf{h} = \max\left(0, \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0.5 \\ -1.5 \end{bmatrix}\right)$$

$$\begin{aligned} \mathbf{h}([0,0]^T) &= [0.5, -1.5]^T \\ \mathbf{h}([0,1]^T) &= [1.5, 0]^T \\ \mathbf{h}([1,0]^T) &= [1.5, 0]^T \\ \mathbf{h}([1,1]^T) &= [2.5, 0.5]^T \end{aligned}$$



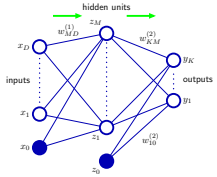
Two layer network

Feature mapping in XOR learning

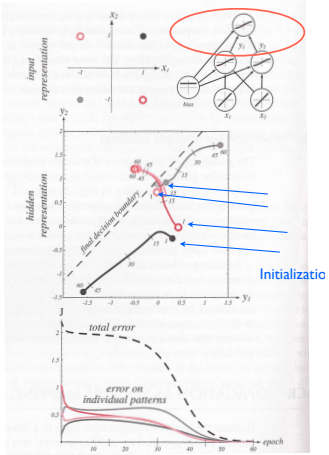
Linear discriminant

Original features are not linearly separable

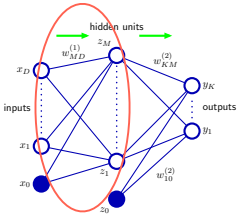
Non-linear mapping by hidden layer



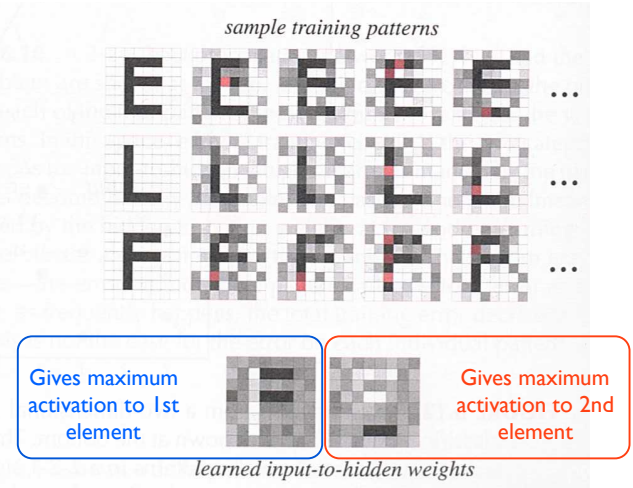
2-2-1 network



Weights capture important features



64-2-3 network



Feed forward neural network: first layer

- First trainable layer: hidden layer

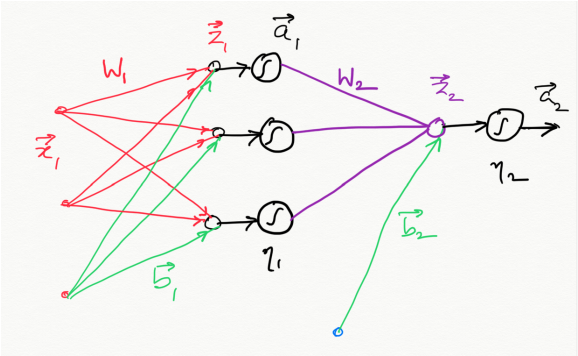
$$z_1(j) = \sum_{i=1}^{D} w_1(j,i)x(i) + b_1(j)$$

- Non-linear activation

$$a_1(j) = \eta_1\left(z_1(j)\right); \quad j = 1, .., M$$

- Matrix form

$$\mathbf{z}_1 = \eta_1\left(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1\right)$$



Two layer network

Feed forward neural network: second layer

- second trainable layer

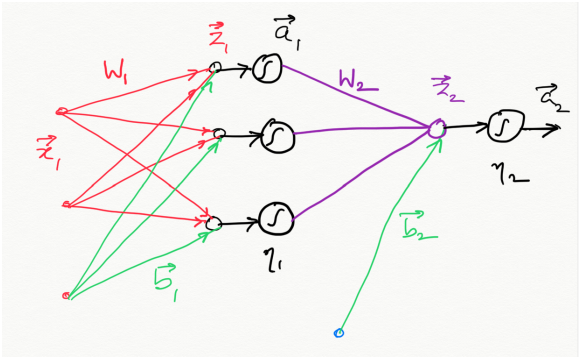
$$z_2(j) = \sum_{i=1}^{D} w_2(j,i)a_1(i) + b_2(j)$$

- Non-linear activation

$$a_2(j) = \eta_2\left(z_2(j)\right); \quad j = 1,..,M$$

- Matrix form

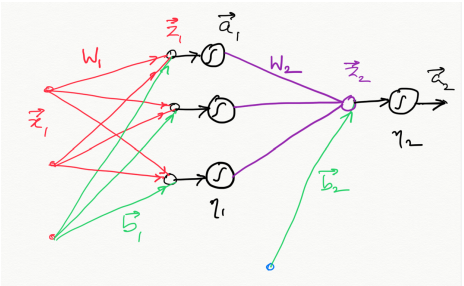
$$\mathbf{z}_2 = \eta_1\left(\mathbf{W}_2\mathbf{z}_1 + \mathbf{b}_2\right)$$



Two layer network

What if activation function is linear: i.e, $\eta_i(x) = x$

- First layer
$$\mathbf{z}_1 = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1$$
- Second layer
$$\mathbf{z}_2 = \mathbf{W}_2\mathbf{z}_1 + \mathbf{b}_2$$
- Simplifying
$$\begin{aligned} \mathbf{z}_2 &= \mathbf{W}_2(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2 \\ &= \mathbf{W}_2\mathbf{W}_1\mathbf{x} + \underbrace{\mathbf{W}_2\mathbf{b}_1 + \mathbf{b}_2}_{\mathbf{c}_2} \end{aligned}$$
- Equivalent to single layer network



Two layer network

Common activation functions

- Absolute function

$$\eta(x) = |x|$$

- Sigmoid function: linear close to origin

$$\eta(x) = \sigma(x) = \frac{1}{1 + \exp(-x)}$$

- Hyperbolic tangent: linear close to origin

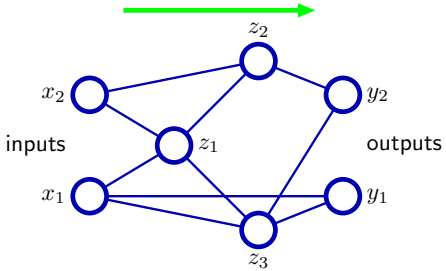
$$\eta(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0. \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0. \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}. \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^z)$	Multi-layer Neural Networks	

Activation with linear response close to origin

Logistic (sigmoid)	$\phi(z) = \frac{1}{1+e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$	Multi-layer Neural Networks	

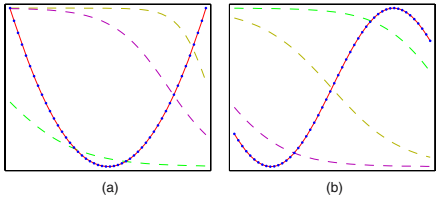
- Small inputs $\rightarrow$ linear response
- Single layer network: convex
- Initialization with small weights



- Skip layer connections
- Small weights & sigmoid activation

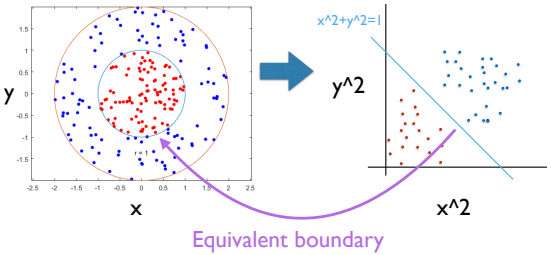
NN: Universal function approximators

- Two layer networks with linear output layer can approximate any function with arbitrary accuracy
- May require large number of neurons in hidden layer !!
- Existence proof: no guarantee on finding these parameters !!

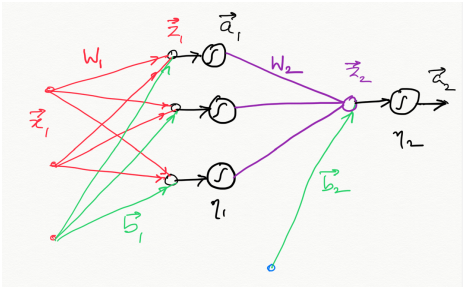


3 layer model with *tanh* activation

Intuition: Learned kernel & perceptron

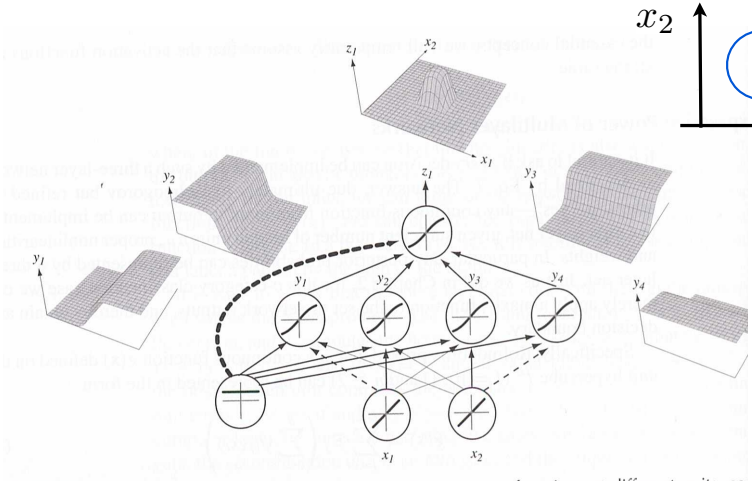


Kernel methods: fixed non-linear mapping



NN: First layer maps the input to a feature space $\mathbf{a}_1$

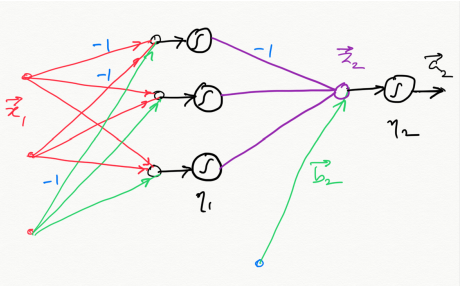
Intuition: intersection of half planes



Weight space symmetries

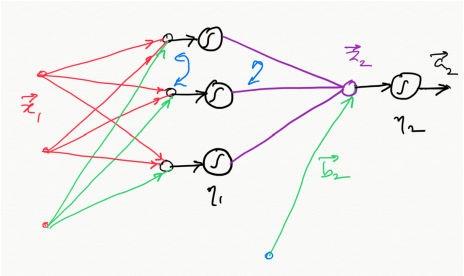
- Tanh activation: Equivalent output
  - Multiply input weights by -1
  - Multiply output weights by -1
  - 2^M possibilities

- Multiple equivalent good solutions !!
- Find one good solution !!

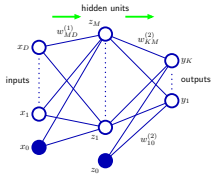


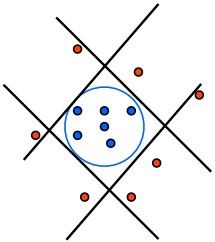
Weight space symmetries

- Swap hidden nodes
  - Equivalent output
  - $M!$ possibilities
- Multiple global minima with same cost !!
- Find one good solution !!



Neural network training





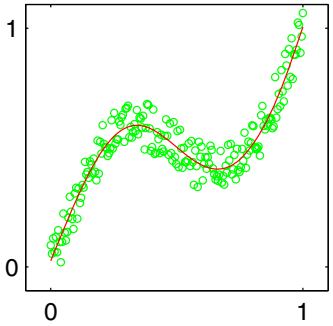
How to train your Neural Network

Regression: Cost function

- Cost function for N training data points

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

  - $\mathbf{t}_n$: expected output
  - Negative log likelihood: Gauss. probability



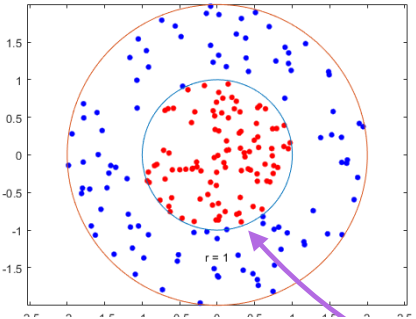
Classification: cost function

- Cost function for N training data points

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \left(t_n \ln y_n + (1 - t_n) \ln(1 - y_n) \right)$$

  - Cross entropy penalty: negative log likelihood
  - Sigmoid output justified: logistic regression
- Multiclass classification: softmax function

$$y_k(\mathbf{x}, \mathbf{w}) = \frac{\exp(b_k)}{\sum_k \exp(b_k)}$$

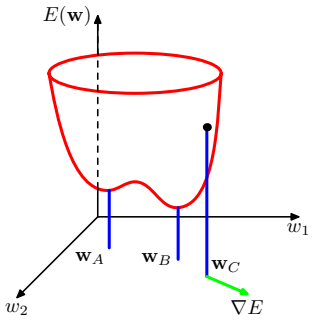


Optimization

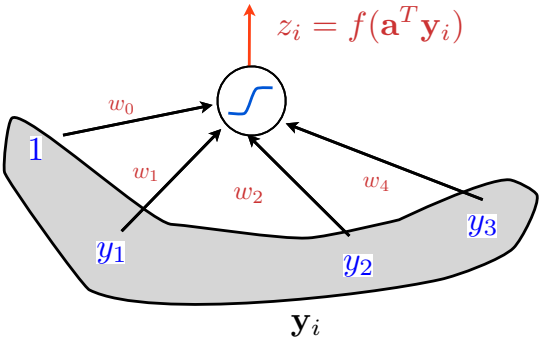
- Steepest descent

$$\mathbf{w}^{(n+1)} = \mathbf{w}^{(n)} - \gamma \nabla_{\mathbf{w}} E$$

  - Can only guarantee convergence to a local minimum
- How you compute the gradient ?
  - Finite differences: computationally expensive
  - Error back propagation !!



Revisit: logistic regression



Minimize errors

$$\mathcal{J}(\mathbf{a}) = \sum_{i=1}^{N} \|f(\mathbf{a}^T \mathbf{y}_i) - t_i\|^2$$

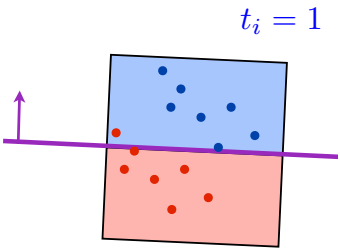
Logistic regression: optimization

Choose weights that minimize the error

$$\mathbf{a} = \arg\min_{\mathbf{w}} \mathcal{J}(\mathbf{a})$$

Criterion

$$\mathcal{J}(\mathbf{a}) = \sum_{i=1}^{N} \|f(\mathbf{a}^T \mathbf{y}_i) - t_i\|^2$$



Steepest descend update

$$\mathbf{a}_{n+1} = a_n - \gamma \, \nabla \mathcal{J}(\mathbf{a})$$

Gradient calculation: backpropagation

Criterion

$$\mathcal{J}(\mathbf{a}) = \sum_{i=1}^{N} \|f(\mathbf{a}^T \mathbf{y}_i) - t_i\|^2$$

Gradient

$$\nabla \mathcal{J} = \sum_{i=1}^{N} (z_i - t_i)\ f'(\mathbf{a}^T \mathbf{y}_i)\ \mathbf{y}_i$$

Properly classified points: $z_i = t_i$

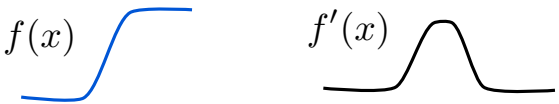
$$\nabla \mathcal{J} = \sum_{\mathbf{y}_i \in \mathcal{Y}} \underbrace{(z_i - t_i)}_{e_i}\ f'(\mathbf{a}^T \mathbf{y}_i)\ \mathbf{y}_i$$

Gradient: intuition

Gradient

$$\nabla \mathcal{J} = \sum_{\mathbf{y}_i \in \mathcal{Y}} f'(\mathbf{a}^T \mathbf{y}_i)\ e_i \quad \mathbf{y}_i$$

Gradient of the function is localized



Give importance to points near boundary

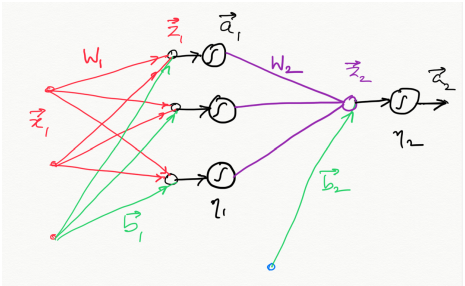
Forward propagation

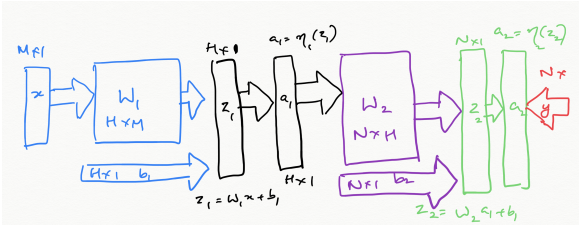
- First layer

$$\mathbf{a}_1 = \eta_1 \left(\underbrace{\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1}_{\mathbf{z}_1} \right)$$

- Second layer

$$\mathbf{a}_2 = \eta_1 \left(\underbrace{\mathbf{W}_2 \mathbf{a}_1 + \mathbf{b}_2}_{\mathbf{z}_2} \right)$$





Computation of the gradient terms: layer 2

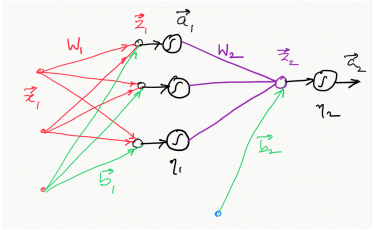
- Cost function for a training data point $\mathbf{y}$

$$\mathcal{E}\left(\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2\right) = \left(\frac{1}{2}\right) \sum_{n=1}^{N} \underbrace{\|y(n) - a_2(n)\|^2}_{E(n)}$$

- Gradient wrt $\mathbf{W}_2$ using chain rule

$$\frac{\partial \mathcal{E}}{\partial W_2(i)} = \underbrace{\frac{\partial \mathcal{E}}{\partial a_2}}_{\delta = (a_2 - y)} \underbrace{\frac{\partial a_2}{\partial z_2}}_{\eta_2'(z_2)} \underbrace{\frac{\partial z_2}{\partial W_2(i)}}_{a_1(i)}$$

- $\delta = a_2 - y$: Error



$$z_2 = \eta_2 \left(\sum_{i=1}^{H} W_2(i) a_1(i) + b_2 \right) \quad (1)$$

Layer 2 gradients: matrix form

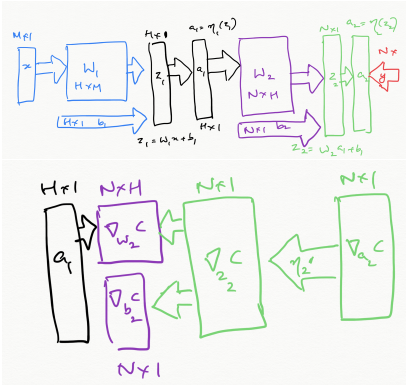
- Gradient wrt $\mathbf{W}_2$ using chain rule

$$\frac{\partial \mathcal{E}}{\partial W_2(i)} = \underbrace{\frac{\partial \mathcal{E}}{\partial a_2}}_{\delta=(a_2-y)} \underbrace{\frac{\partial a_2}{\partial z_2}}_{\eta_2'(z_2)} \underbrace{\frac{\partial z_2}{\partial W_2(i)}}_{a_1(i)}$$

- Matrix of size $H \times 1$: $\nabla_{\mathbf{W}_2} E$

$$\underbrace{\nabla_{\mathbf{W}_2}\mathcal{E}}_{1\times H} = \underbrace{\nabla_{\mathbf{z}_2}\mathcal{E}}_{1\times 1} \underbrace{\mathbf{a}_1^H}_{1\times H}$$

- $\underbrace{\nabla_{\mathbf{z}_2}\mathcal{E}}_{1\times 1} = \underbrace{\sigma'(\mathbf{z}_2)}_{1\times 1} \odot \underbrace{\nabla_{\mathbf{a}_2}\mathcal{E}}_{1\times 1}$



General case with n outputs

- Gradient wrt $\mathbf{W}_2$ using chain rule

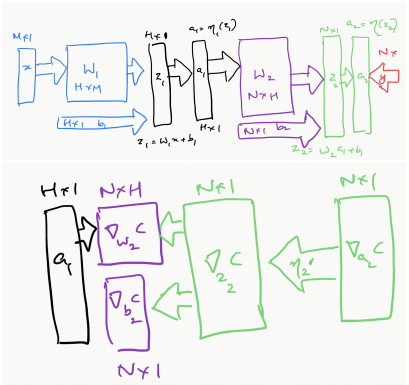
$$\frac{\partial \mathcal{E}}{\partial W_2(n,i)} = \underbrace{\frac{\partial \mathcal{E}}{\partial a_2(n)}}_{\delta(n)=\left(a_2(n)-y(n)\right)} \underbrace{\frac{\partial a_2(n)}{\partial z_2(n)}}_{\eta_2'(z(n))} \underbrace{\frac{\partial z_2(n)}{\partial W_2(n,i)}}_{a_1(i)}$$

- Matrix of size $N \times H$: $\nabla_{\mathbf{W}_2} E$

$$\underbrace{\nabla_{\mathbf{W}_2}\mathcal{E}}_{N\times H} = \underbrace{\nabla_{\mathbf{a}} E}_{N\times 1}\ \underbrace{\mathbf{a}_1^T}_{1\times H}$$

- Point by point multiplication:

$$\underbrace{\nabla_{\mathbf{a}} E}_{1\times K} = \underbrace{\sigma'(\mathbf{a})}_{1\times K} \odot \underbrace{\nabla_{\mathbf{a}} E}_{1\times K}$$



Bias term gradients

- Gradient wrt $\mathbf{b}_2$ using chain rule

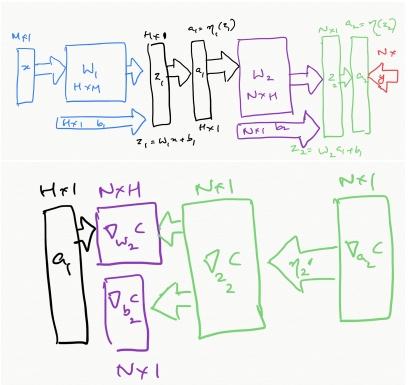
$$\frac{\partial \mathcal{E}}{\partial b_2(n)} = \underbrace{\frac{\partial \mathcal{E}}{\partial a_2(n)}}_{\delta(n)=\left(a_2(n)-y(n)\right)} \underbrace{\frac{\partial a_2(n)}{\partial z_2(n)}}_{\eta_2'(z(n))} \underbrace{\frac{\partial z_2(n)}{\partial b_2}}_{1}$$

- Matrix of size $1 \times N$: $\nabla_{\mathbf{a}_2} E$

$$\underbrace{\nabla_{\mathbf{b}_2}\mathcal{E}}_{1\times N} = \underbrace{\nabla_{\mathbf{z}_2}\mathcal{E}}_{1\times N}$$

- Point by point multiplication:

$$\underbrace{\nabla_{\mathbf{z}_2}\mathcal{E}}_{1\times K} = \underbrace{\sigma'(\mathbf{a}_2)}_{1\times K} \odot \underbrace{\nabla_{\mathbf{a}_2} E}_{1\times K}$$

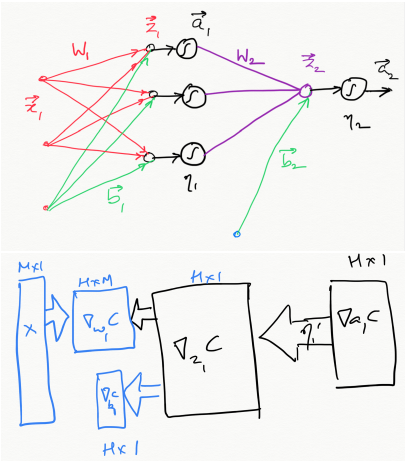


Computation of the gradient terms: layer 1

- Gradient wrt $\mathbf{W}^{(1)}$ using chain rule

$$\frac{\partial E}{\partial W_1(i,j)} = \underbrace{\frac{\partial E}{\partial a_1(i)}}_{\delta(i)} \underbrace{\frac{\partial a_1(i)}{\partial z_1(i)}}_{\eta_1'(z_1(i))} \underbrace{\frac{\partial z_1(i)}{\partial W_1(i,j)}}_{x(j)}$$

- δ_i: perturbation of cost wrt hidden node
  - Commonly referred to as error at that node
  - Evaluated using chain rule



Layer 1 gradients: matrix form

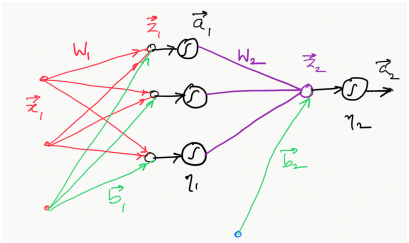
- Gradient wrt $\mathbf{W}^{(1)}$ using chain rule

$$\frac{\partial E}{\partial W_1(i,j)} = \underbrace{\frac{\partial E}{\partial a_1(i)}}_{\delta(i)} \underbrace{\frac{\partial a_1(i)}{\partial z_1(i)}}_{\eta_1'(z_1(i))} \underbrace{\frac{\partial z_1(i)}{\partial W_1(i,j)}}_{x(j)}$$

- Matrix of size $H \times M$: $\nabla_{\mathbf{W}_1} E$

$$\underbrace{\nabla_{\mathbf{W}_1} E}_{H \times M} = \underbrace{\nabla_{\mathbf{a}_1} E}_{H \times 1} \underbrace{\mathbf{x}^T}_{1 \times M}$$

- $\underbrace{\nabla_{\mathbf{a}_1} E}_{1 \times H} = \underbrace{\eta_1'(\mathbf{z}_1)}_{1 \times H} \odot \underbrace{\nabla_{\mathbf{a}_1} E}_{1 \times H}$



Errors δ_i at hidden nodes

- Gradient expression

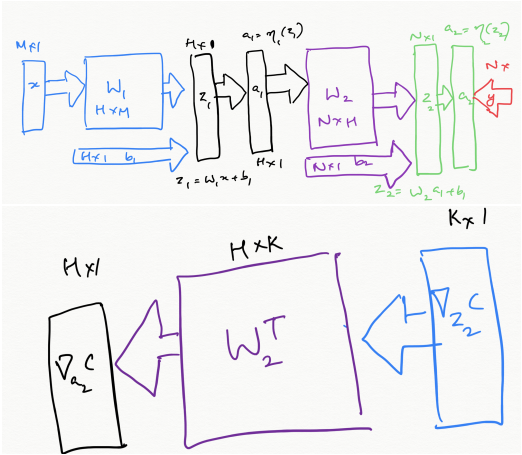
$$\frac{\partial E}{\partial a_i} = \frac{\partial \mathcal{E}}{\partial z_2} W_2(i)$$

- Matrix of size $H \times 1$

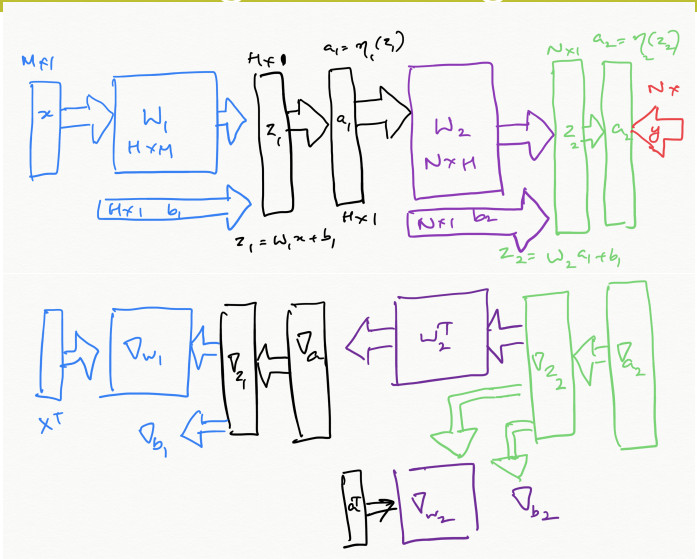
$$\underbrace{\nabla_{\mathbf{a}} \mathcal{E}}_{H \times 1} = \underbrace{\mathbf{W}_2{}^T}_{H \times 1} \underbrace{\nabla_{z_2} \mathcal{E}}_{1 \times 1}$$

- General case with k- outputs $H \times 1$

$$\underbrace{\nabla_{\mathbf{a}} E}_{H \times 1} = \underbrace{\mathbf{W}_2{}^T}_{H \times K} \underbrace{\nabla_{z_2} E}_{K \times 1}$$

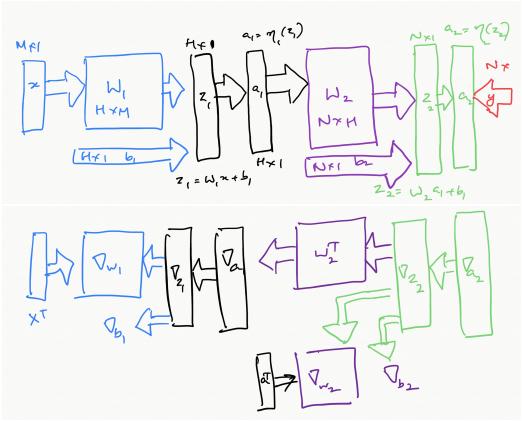


Putting all of it together



Gradient computation: summary

- For each training data
  - Perform forward propagation to compute $\mathbf{z}_1, \mathbf{a}_1, \mathbf{z}_2, \mathbf{a}_2$
  - Evaluate the errors: $\nabla_{\mathbf{a}_1}\mathcal{E}$ and $\nabla_{\mathbf{a}_1}\mathcal{E}$
  - Evaluate the gradients: $\nabla_{\mathbf{W}_1}\mathcal{E}$ and $\nabla_{W_2}\mathcal{E}$
- Sum the gradients for all training points
- Computational complexity $\mathcal{O}(W)$
- Computational complexity of finite difference approach $\mathcal{O}(W^2)$
  - Perturb each weight term
  - Perform forward pass and evaluate difference in error



Gradient descent

Criterion

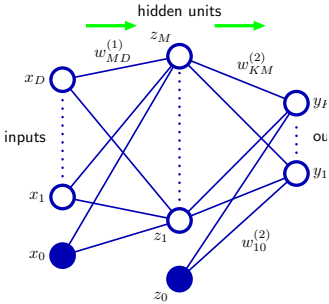
$$\mathcal{E}(\mathbf{w}) = \sum_{n=1}^{N} \underbrace{\|\mathbf{y}(n) - \mathbf{t}(n)\|^2}_{\mathcal{E}_n}$$

Gradient with respect to weight vector

$$\nabla_{\mathbf{w}} J$$

Update

$$\mathbf{w}_{n+1} = \mathbf{w}_{n+1} - \gamma \, \nabla_{\mathbf{w_n}} J$$



Optimization: pseudocode

Given N training points

for n = 1: MaxIterations

$\nabla_{\mathbf{w}} J = 0$

for i = 1: N

Forward propagation

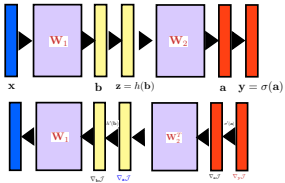
Backward propagation

$\nabla_{\mathbf{w}} J = \nabla_{\mathbf{w}} J + \nabla_{\mathbf{w}} J(i)$

end

Update $\mathbf{w}_{n+1} = \mathbf{w}_{n+1} - \gamma \nabla_{\mathbf{w_n}} J$

end



Computationally expensive

Batch learning/ Stochastic gradient descent (SGD)

Given N training points

for n = 1: MaxIterations

$\nabla_{\mathbf{w}} J = 0$

for i in Subset(n)

Forward propagation

Backward propagation

$\nabla_{\mathbf{w}} J = \nabla_{\mathbf{w}} J + \nabla_{\mathbf{w}} J(i)$

end

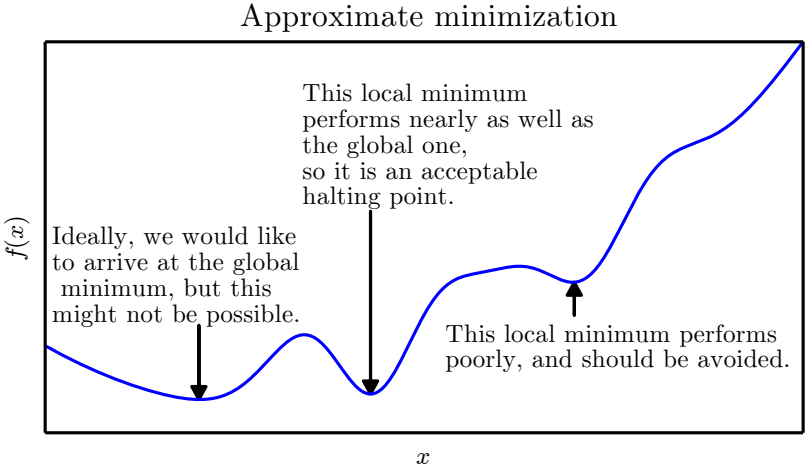
Update $\mathbf{w}_{n+1} = \mathbf{w}_{n+1} - \gamma \nabla_{\mathbf{w_n}} J$

end

Use different subsets at each iteration

Independent random subsets: SGD

Optimization: our goal



Steepest descent

Goal: minimize $\mathcal{E}(\mathbf{w})$

Taylor series approximation of cost

$$\mathcal{E}(\mathbf{w}) = \mathcal{E}(\mathbf{w}_n) + (\mathbf{w} - \mathbf{w}_n)^T \nabla_{\mathbf{w}} \mathcal{E} + ...$$

Steepest descent: linear approximation

Slow convergence: O(iterations)

Error not guaranteed to monotonically decrease

Choose step size appropriately

Tradeoffs with stepsize

Steepest descent update

$$w_{n+1} = w_{n+1} - \eta w_n$$

Convergence dependent on η