University of Manchester AI ML Society - Introduction to ML

Workshop 4
20th of November 2019

By Botond Maros

Intro to ML timetable

Workshop 1 - Introduction to Machine Learning

Workshop 2 - Data preprocessing

Workshop 3 - Fundamental Algorithms I

Workshop 4-5 - Neural Networks Part I

Workshop 6 - Neural Networks Part II

Today’s session

- Revise logistic regression, gradient descent
- Neural network architecture
- Backpropagation
- Activation functions + batches

What is modeling?

- Come up with a model that describes data
- Measure how it describes the data
- Calculate error = difference of real data vs model
- Find model with lowest error

⇒ This is <u>exactly</u> what we are doing with neural networks

Logistic regression and gradient descent

Recall logistic regression and gradient descent

- Binary classification - linear model + sigmoid function
- Formalizing the problem using negative log likelihood
- 2nd line is the error we are trying to minimize

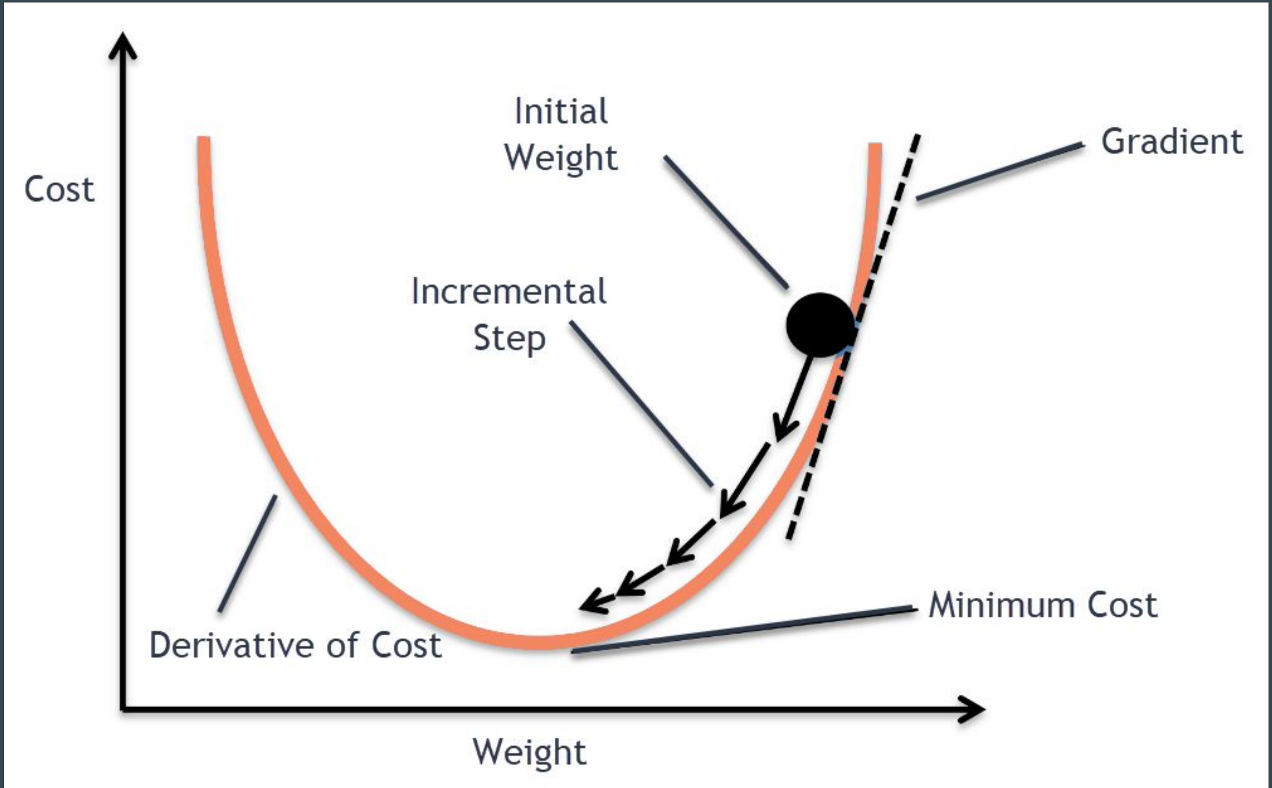
$$NLL(w) = -\sum_{i=1}^{N} log[\mu_i^{I(y_i=1)} * (1-\mu_i)^{I(y_i=0)}]$$

$$= -\sum_{i=1}^{N}[y_i log\ \mu_i + (1-y_i)log(1-\mu_i)]$$

where

$$\mu_i = sigm(w^T x_i)$$

Gradient descent



Gradient descent for logistic regression

Gradient:

$$g(w) = \frac{\partial(Err(w))}{\partial w} = -\sum_{i=1}^{N} x_i(y_i - \sigma(w^T x_i))$$

Gradient descent:

Given initial vector w_0

do for k=1, 2, ...

$$w_{k+1} = w_k - \alpha_k * g(w_k)$$

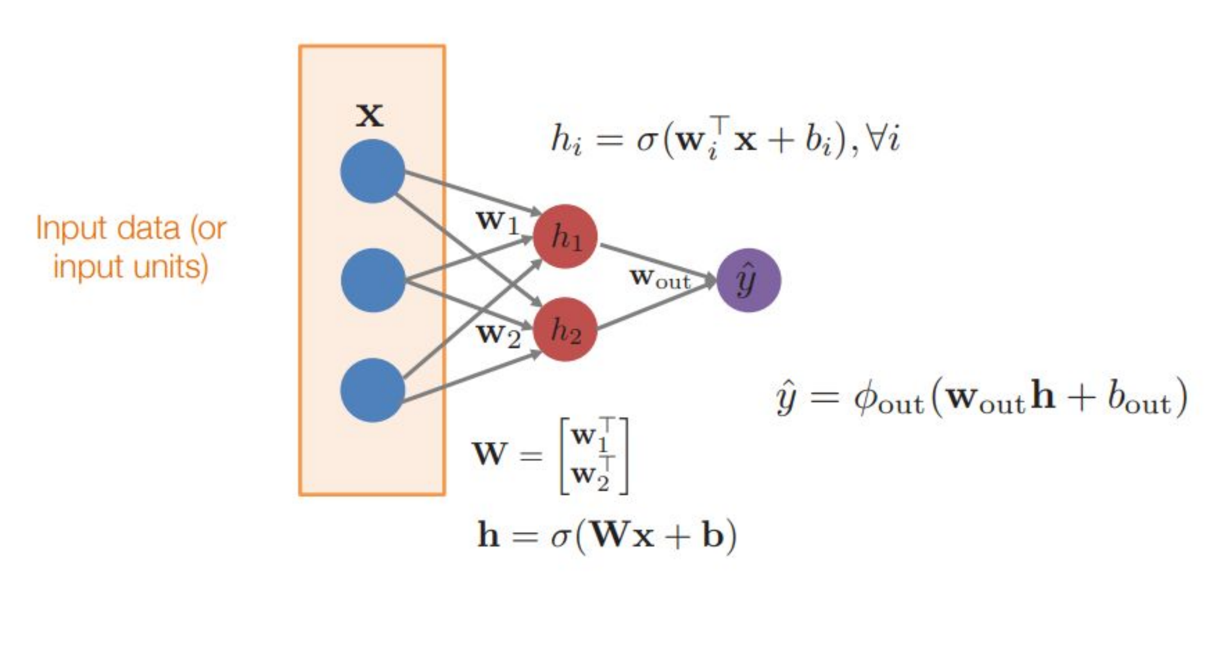
stop if :

$$|w_{k+1} - w_k| < \epsilon$$

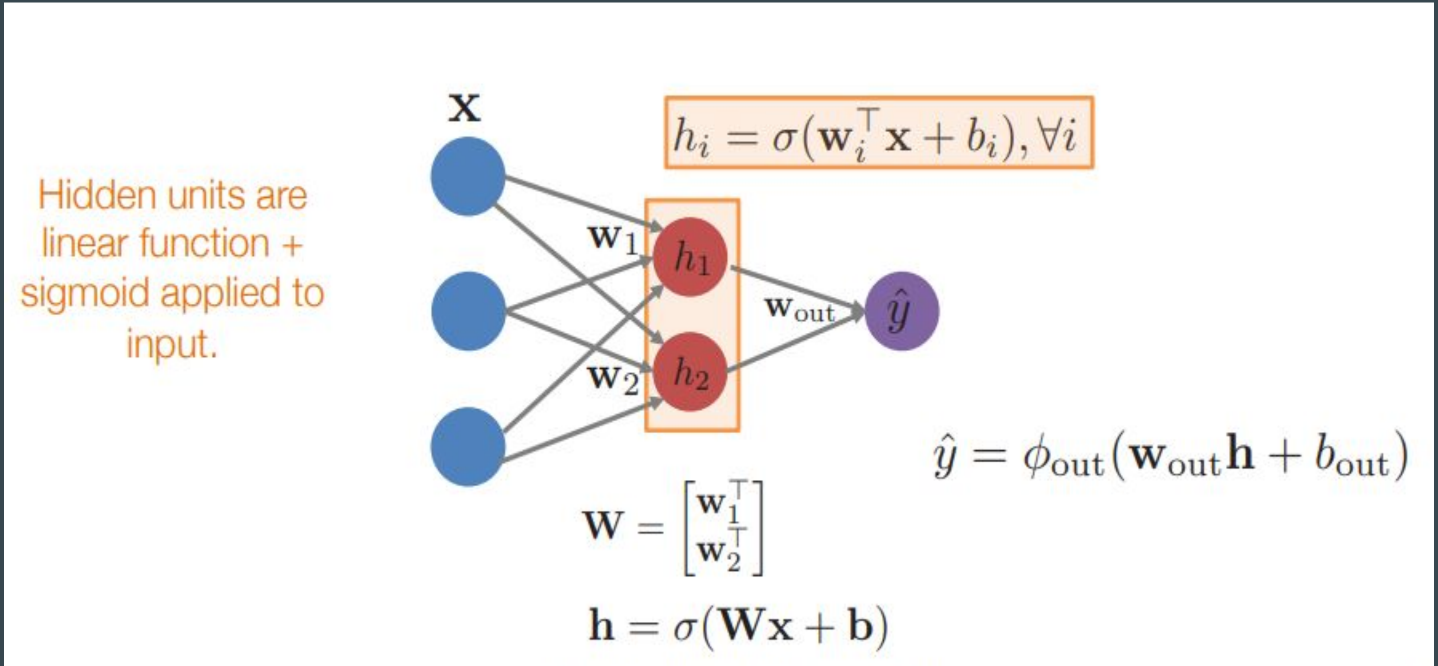
Artificial neural networks

Artificial neural networks

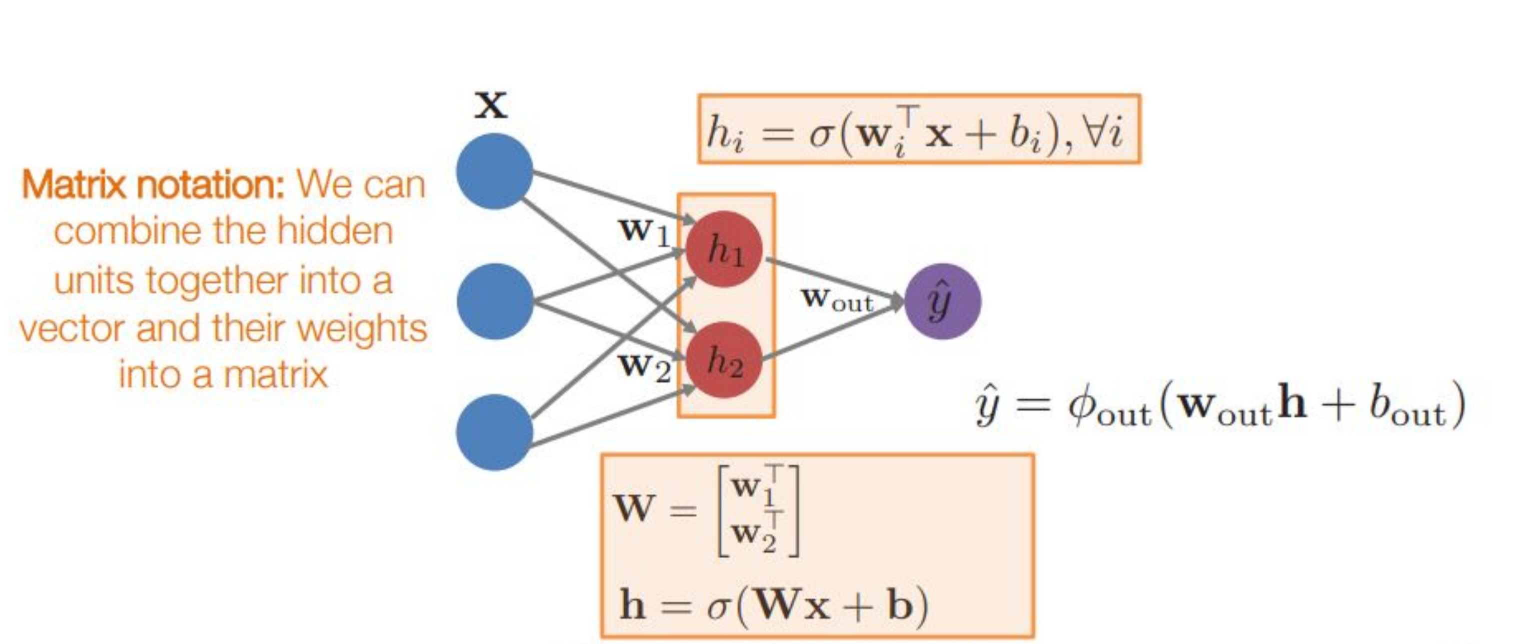
= stacking multiple logistic regressions and training simultaneously



ANN - hidden layers



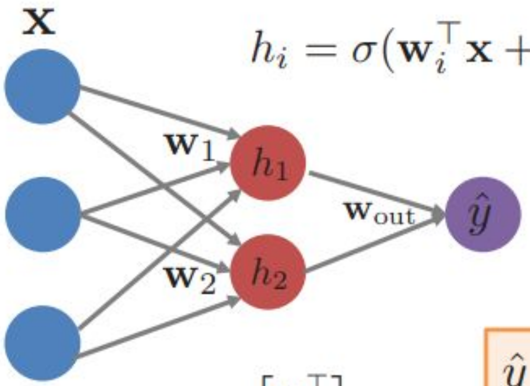
ANN - hidden layers



ANN- output layer

Output unit: Linear function of the hidden units followed by an "activation function", ϕ_{out}.

$$h_i = \sigma(\mathbf{w}_i^\top \mathbf{x} + b_i), \forall i$$



$$\boxed{\hat{y} = \phi_{\text{out}}(\mathbf{w}_{\text{out}}\mathbf{h} + b_{\text{out}})}$$

$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \end{bmatrix}$$

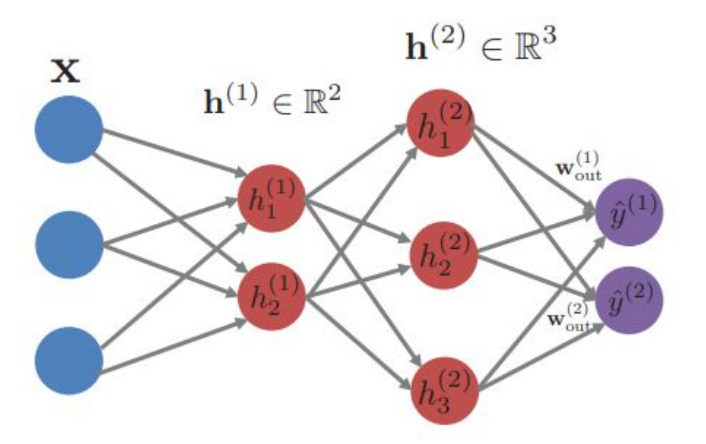
$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

Output's activation funciton

- We can use ANNs for linear regression out(z) = z
- Or for binary classification out(z) = sigmoid(z)
- Or for multi label classification with multiple output nodes

Architecture properties

- We can have multiple hidden layers
- No cross connection between nodes in a layer
- No backward connection
- Usually fully connected



Multiple layers

$$\mathbf{h}^0 = \mathbf{x}$$

Initialize

for i=1...H:

$$\mathbf{h}^{(i)} = \sigma(\mathbf{W}^{(i)}\mathbf{h}^{(i-1)} + \mathbf{b}^{(i)})$$

Compute each hidden layer sequentially

$$\hat{\mathbf{y}} = \phi_{\text{out}}(\mathbf{W}_{\text{out}}\mathbf{h}^{(H)} + \mathbf{b}_{\text{out}})$$

Compute the output

- So now we only left one thing to do and that finding the optimal weights
- Optimal weights = lowest error
- -> use gradient descent

Backpropagation

Recall Gradient descent

- Calculates error, takes the derivative
- Modifies the weight by the gradient times a number
- ANN: each layer depends on the previous one
- If we change something, it will have a domino effect
- Want to calculate each edges share to the error
- How can we do this fast?

Given initial vector w_0

do for k=1, 2, ...

$$w_{k+1} = w_k - \alpha_k * g(w_k)$$

stop if :

$$|w_{k+1} - w_k| < \epsilon$$

$\mathbf{h}^0 = \mathbf{x}$ Initialize

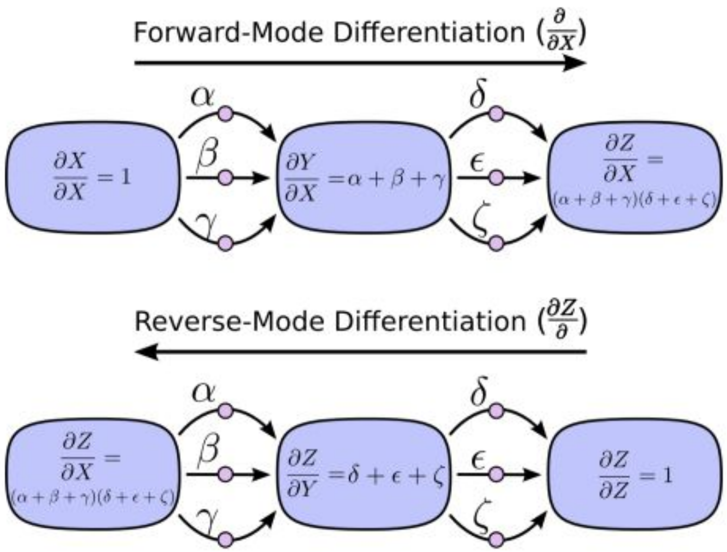
for i=1...H: Compute each hidden layer sequentially

$$\mathbf{h}^{(i)} = \sigma(\mathbf{W}^{(i)}\mathbf{h}^{(i-1)} + \mathbf{b}^{(i)})$$

$\hat{\mathbf{y}} = \phi_{\text{out}}(\mathbf{W}_{\text{out}}\mathbf{h}^{(H)} + \mathbf{b}_{\text{out}})$ Compute the output

Backpropagation

- **Forward mode**
  - Goes from source(s) to sink.
  - At each node, sum all the incoming edges/derivatives.
- **Reverse mode:**
  - Goes from sink to source(s).
  - At each node, sum all the outgoing edges/derivatives.
- Both only touch each edge once!



Derivatives

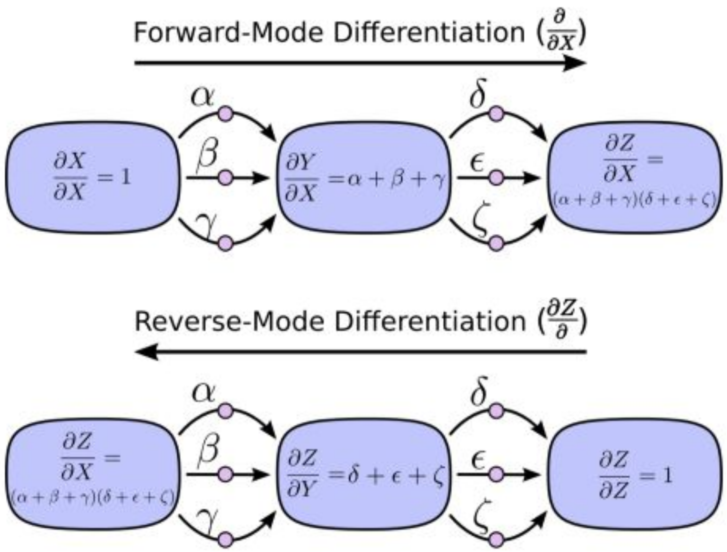
Calculate X's effect on Z:

$$\frac{\partial Z}{\partial X} = \alpha\delta + \alpha\epsilon + \alpha\zeta + \beta\delta + \beta\epsilon + \beta\zeta + \gamma\delta + \gamma\epsilon + \gamma\zeta$$

$$\frac{\partial Z}{\partial X} = (\alpha + \beta + \gamma)(\delta + \epsilon + \zeta)$$

Backpropagation

- **Forward mode**
  - Goes from source(s) to sink.
  - At each node, sum all the incoming edges/derivatives.
- **Reverse mode:**
  - Goes from sink to source(s).
  - At each node, sum all the outgoing edges/derivatives.
- Both only touch each edge once!



Putting everything together

What we have so far

- Multiple logistic regressions
- Lot of nodes and layers
- Backpropagation
- We can use backpropagation for calculating every derivative in one go

We just put everything together and use gradient descent for the optimization

Putting everything together

- Initialize all weights to small random numbers.

Initialization

- Repeat until convergence:
  - Pick a training example, $\mathbf{x}$.
  - Feed example through network to compute output $\mathbf{y}$.

Forward pass

  - For the output unit, compute the correction:

$$\frac{\partial J}{\partial \mathbf{w}_{\text{out}}} = \delta_{\text{out}}\mathbf{x}$$

  - For each hidden unit *j*, compute its share of the correction:

$$\frac{\partial J}{\partial \mathbf{w}_j} = \delta_{\text{out}} w_{out,j} \sigma(\mathbf{w}_j^\top \mathbf{x} + b)(1 - \sigma(\mathbf{w}_j^\top \mathbf{x} + b))\mathbf{x}$$

Backpro-
pagation

  - Update each network weight:

$$\mathbf{w}_j = \mathbf{w}_j - \alpha \frac{\partial J}{\partial \mathbf{w}_j} \;\; \forall j, \qquad \mathbf{w}_{out} = \mathbf{w}_{out} - \alpha \frac{\partial J}{\partial \mathbf{w}_{out}}$$

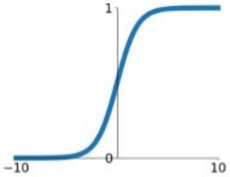
Gradient
descent

Activation functions

Activation Functions

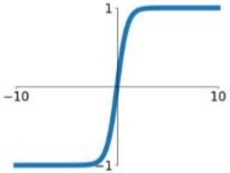
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



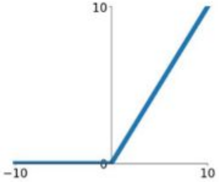
tanh

$\tanh(x)$



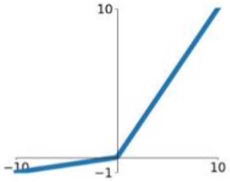
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

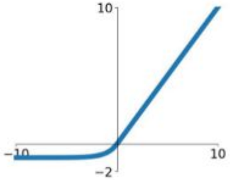


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$



Batches

- How often we update the weights?
- Every training example?
- After the whole data-set
- In between : mini-batch

Questions?

- Multiple logistic regression
- Trained together
- By gradient descent
- Using backpropagation

- Batches
- Activation functions

Thank you!

Resources:

- http://colah.github.io/posts/2015-08-Backprop/
- https://www.cs.mcgill.ca/~wlh/comp551/