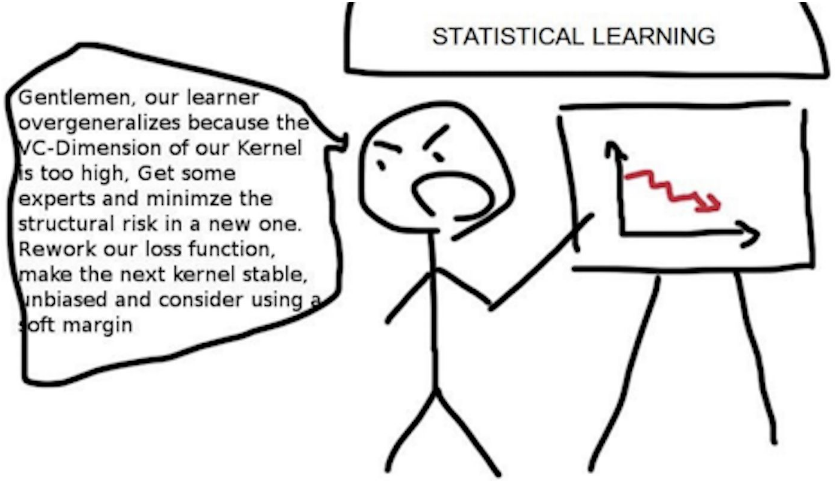
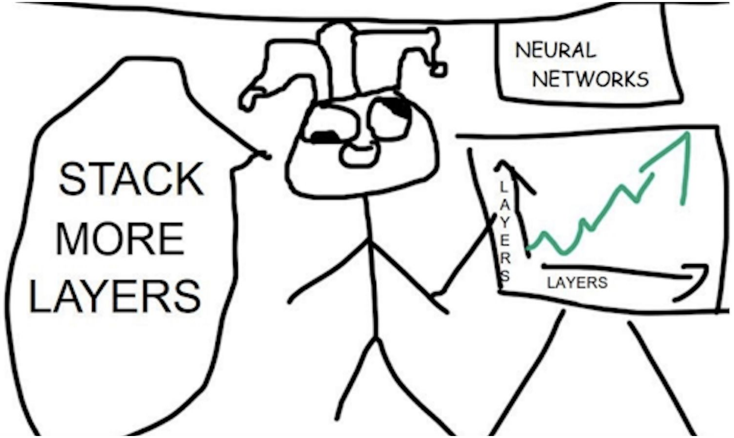
Deep learning

Mannu Malhotra

Until Now:



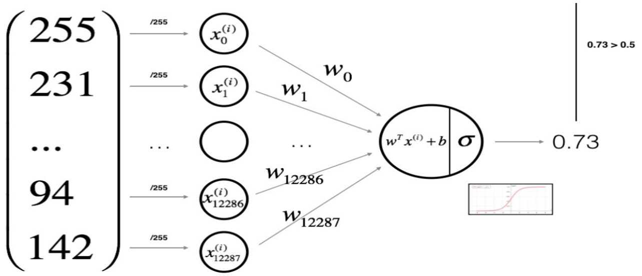
Now:



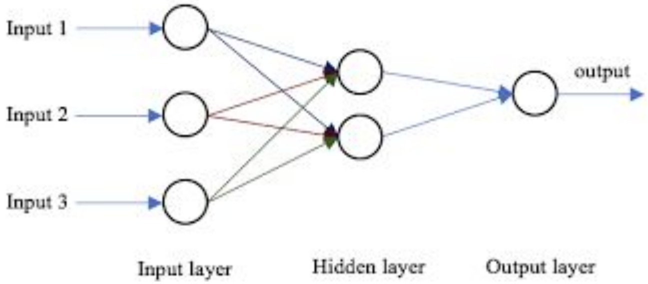
Agenda

- Neural Networks 101:
  - What are NN
  - Why do we need NN
  - What are computation graphs
  - Derivation on computation graphs
  - **Hands on: Implement logistic regression in NN style**
  - NN representation
- Advanced topics:
  - Activation functions
  - Gradient descent and Backpropagation
  - Initialization of weights
- Practical Approach
  - Building blocks of DNN, forward and back prop
  - Softmax and **hands on: Softmax regression using Keras**
  - Explain homework: Building a NN in python using numpy.

What are Neural Networks

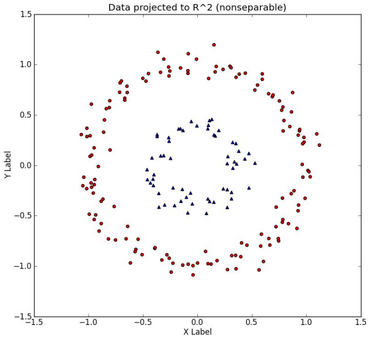


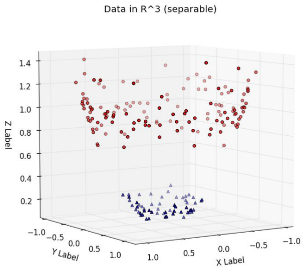
- We learned linear regression
  - y = wx + b
- Logistic regression : linear regression with sigmoid
  - For binary classification.
  - Uses sigmoid activation.
  - Get the value between 0 and 1 to determine the class.
- Logistic regression : single layer, single unit neural network.



Why do we need Neural Networks?

- Linearly non-separable data
    - Linear models like logistic regression can not learn on such data.

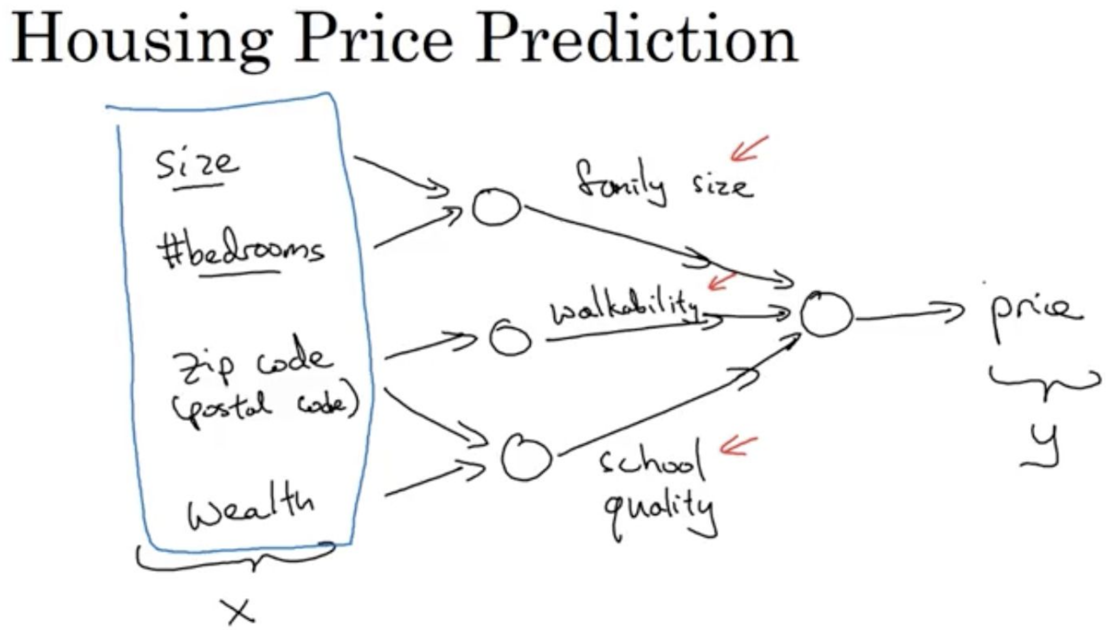




    - Use polynomial features
    - Use kernels to do features transformation to new dimensions.
    - Use stacking of logistic regression to build non linear learner.
        - NN : y = w2*sigmoid(w1*x+b1) + b2

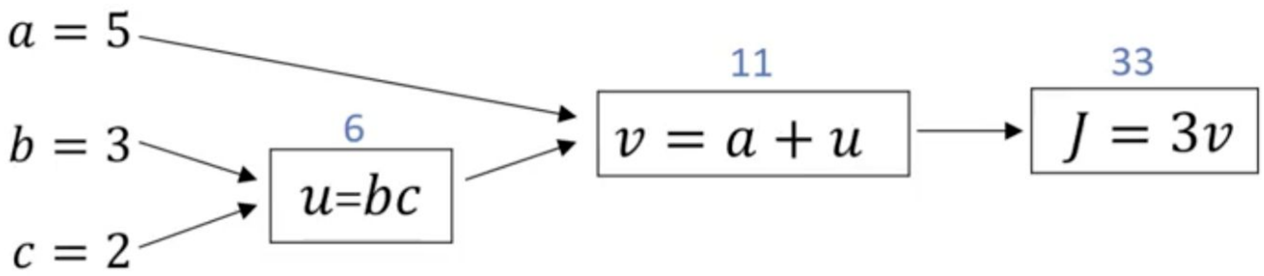
Why do we need Neural Networks?

- Feature learning/end-to-end learning!

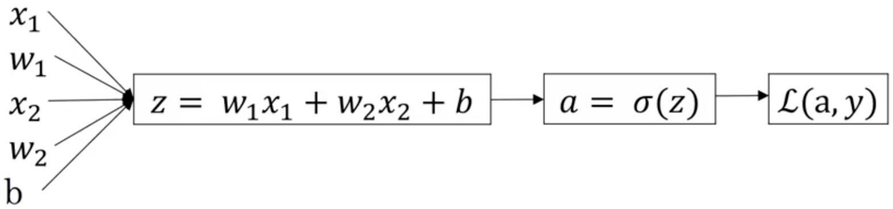


Computation graphs & chain rule for first derivative

- A NN is nothing but a computation graph.
- Computation of neural network = forward propagation (output of the neural network) + backward propagation (for optimization like gradient descent).
- Illustration on chain rule : how would I calculate dJ/da (viz. da).



Logistic regression derivatives



$$z^{(i)} = w^T x^{(i)} + b$$
$$\hat{y}^{(i)} = a^{(i)} = sigmoid(z^{(i)})$$
$$\mathcal{L}(a^{(i)}, y^{(i)}) = -y^{(i)} \log(a^{(i)}) - (1 - y^{(i)}) \log(1 - a^{(i)})$$

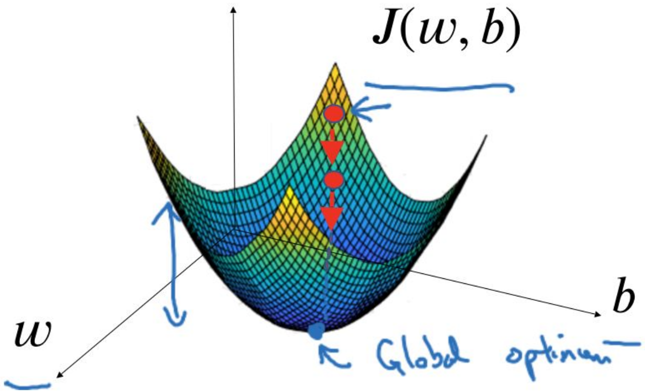
- Objective: Modify w & b to min(L)
- Calculate derivative of Loss function w.r.t. to w and b.
  - i.e. calculate dw & db
- da = -y/a + (1-y)/(1-a)
- da/dz = a(1-a)
- dz = a - y
- db = dz, dw = xdz

Gradient descent: One last time!

Recap: $\hat{y} = \sigma(w^T x + b),\ \sigma(z) = \frac{1}{1+e^{-z}}$

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)}\log\hat{y}^{(i)} + (1-y^{(i)})\log(1-\hat{y}^{(i)})$$

- Objective: Find w,b to minimize J (cost).



Gradient descent algorithm

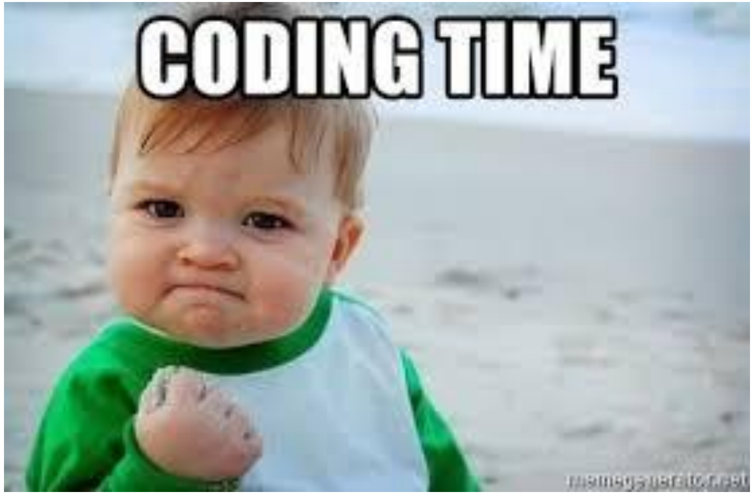
repeat until convergence {

$$\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta_0,\theta_1)$$

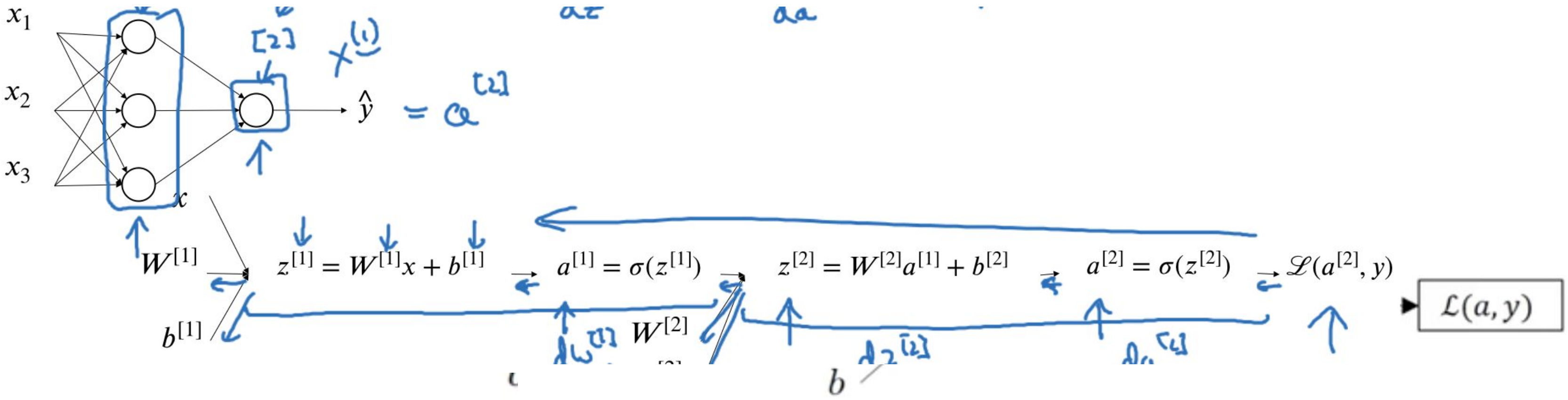
(for $j = 1$ and $j = 0$)

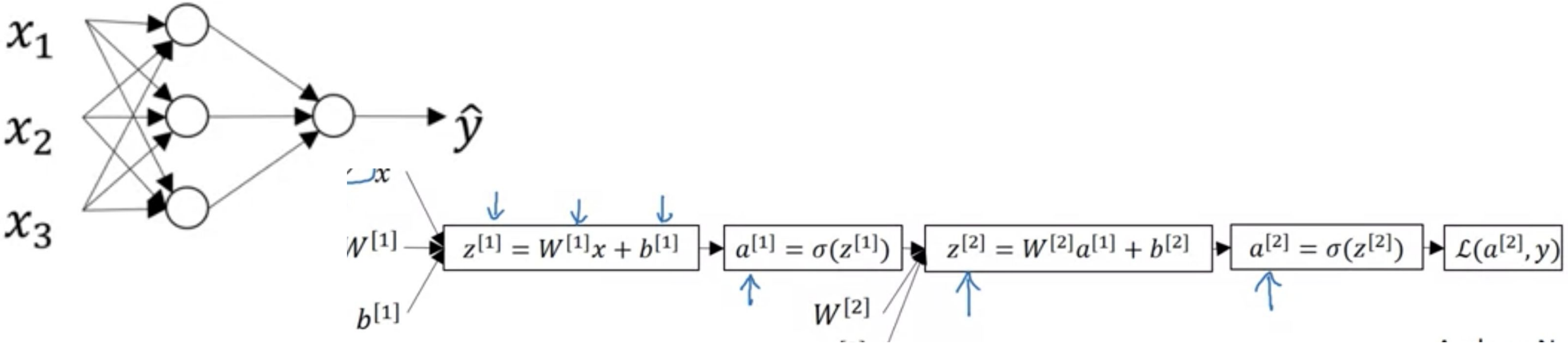
}

Hands on: Implement logistic regression in NN style



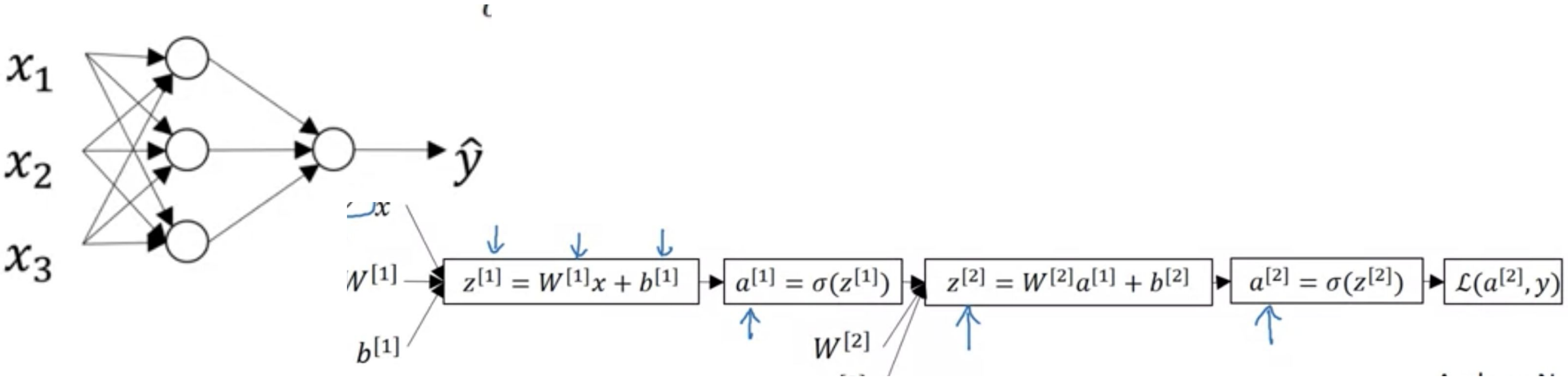
NN representation : logistic regression vs NN



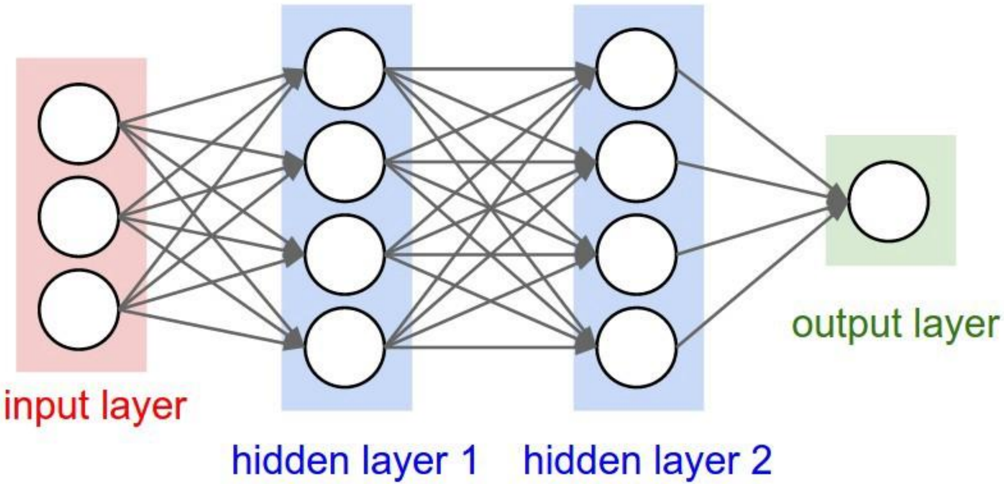


NN representation

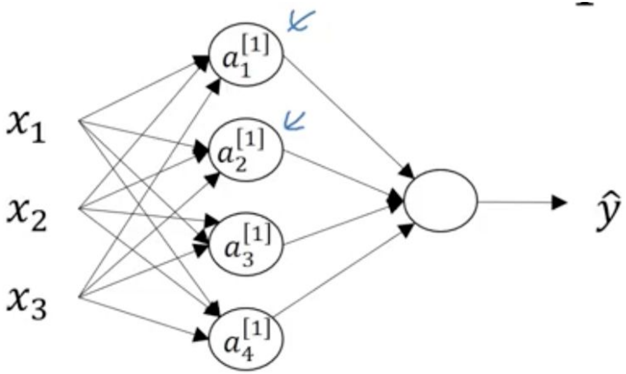
- Forward prop:
  - cal $z^{[1]}$,$a^{[1]}$; $w^{[2]} = a^{[1]}$; cal $z^{[2]}$,$a^{[2]}$; pred = $a^{[2]}$; cal L(pred,y)
- Backward prop:
  - Cal $da^{[2]}$,$dz^{[2]}$,$dw^{[2]}$,$db^{[2]}$,$da^{[1]}$,$dz^{[1]}$,$dw^{[1]}$,$db^{[2]}$



NN representation : layers



NN representation :



$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]},\ a_1^{[1]} = \sigma(z_1^{[1]})$$

$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]},\ a_2^{[1]} = \sigma(z_2^{[1]})$$

$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]},\ a_3^{[1]} = \sigma(z_3^{[1]})$$

$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]},\ a_4^{[1]} = \sigma(z_4^{[1]})$$

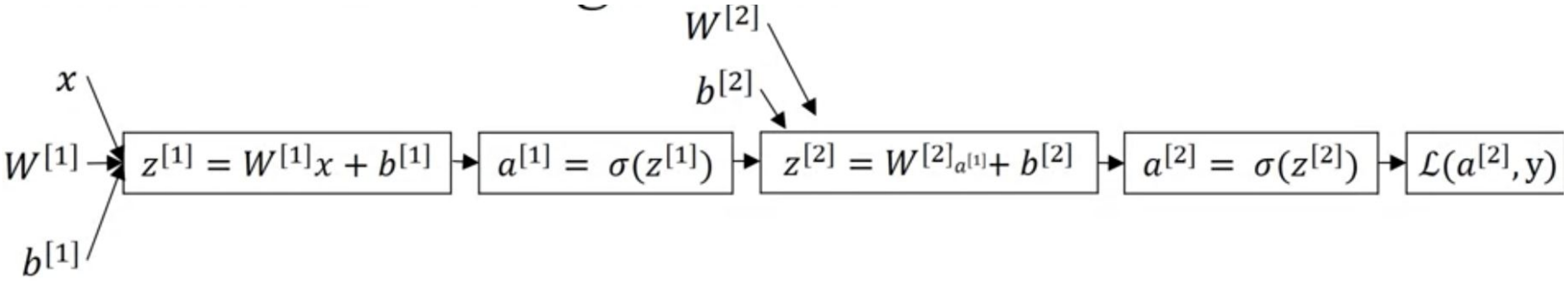
Back prop:

$z^{[1]} = w^{[1]} X + b^{[1]}$
$a^{[1]} = g^{[1]}(z^{[1]})$

$z^{[2]} = w^{[2]}A^{[1]} + b^{[2]}$
$A^{[2]} = g^{[2]}(z^{[2]})$

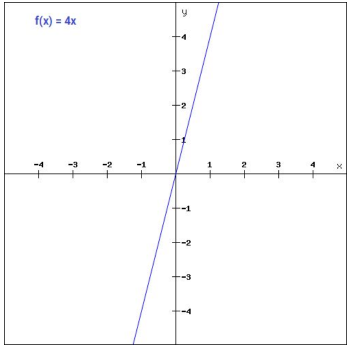
$dz^{[2]} = A^{[2]} - Y$
$dw^{[2]} = dz^{[2]}*A^{[1]}/m$ $(X \to A^{[1]})$
$db^{[2]} = \sum dz^{[2]}/m$

$dz^{[1]} = w^{[2]}*dz^{[2]} * g^{[1]'}(z^{[1]})$
$dw^{[1]} = dz^{[1]}\ XT$
$db^{[2]} = \sum dz^{[1]}/m$

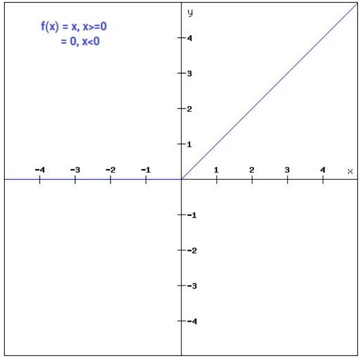


Activation functions: Which Activation functions to use?

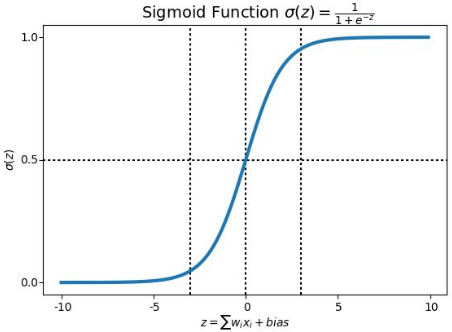
1. Linear:



2. Relu:



3. Sigmoid:



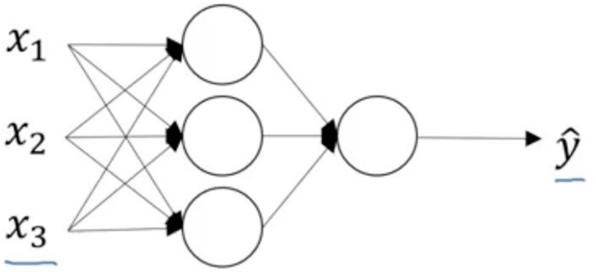
4. Softmax

Activation functions

- Default choice for hidden units.
- Prevent from vanishing gradients.

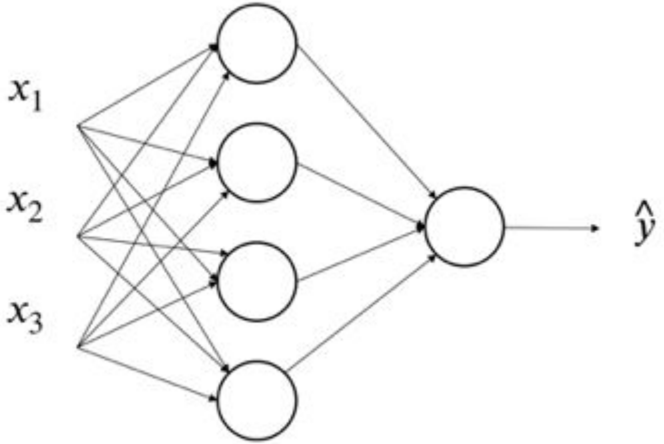
Activation functions: Why do we need activation functions?

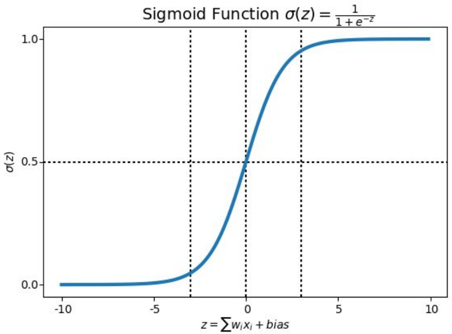
- Why not linear activation function(no activation)?
- ~~a = g(z)~~ , a = z.
- Y_pred = linear function to input parameters.
- z1 = w1.x + b
- a1 = g(z1) = z1
- a2 = w(a1)+b = w2.z1 + b2 = w2(w1.x + b1) + b2
- Y_pred = A2 = w2.w1.x + w2.b1 + b2



Initialization of weights

- All hidden units learning the same function : symmetric breaking.
- Initialise with small values of weights to make learning faster:
  - Large value of weights ->
  - High value of z ->
  - Flat part for sigmoid
  - Tangent value to be small.





Multiclass classification? : Softmax

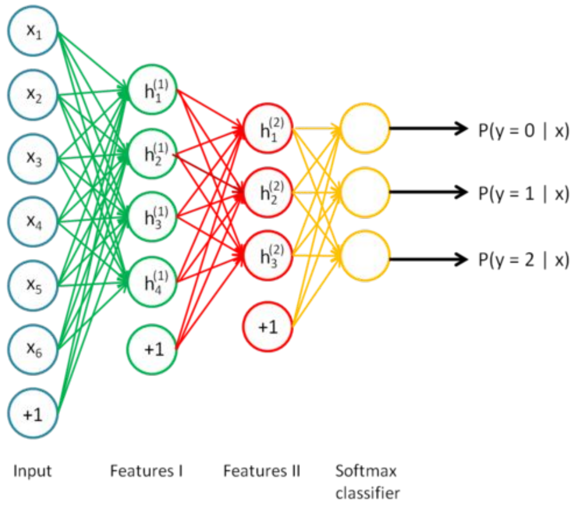
- What if we have multiple classes in the target?
- We can have:
  - One vs all (One vs rest)
    - # of models trained: N (Number of classes)
    - Problems: Class imbalance, can not capture the interdependencies of the classes.
  - One vs One (One vs One)
    - # of models trained: N(N-1)/2
    - Problem: High time complexity, can not capture the interdependencies of the classes.
- Softmax layer to the rescue!

Multiclass classification? : Softmax

- N units in the output layer.
- Sigmoid vs Softmax:
  - Just a different activation function.

$$\frac{1}{1+e^{-(w^T x+b)}} \qquad \frac{e^{y_i}}{\sum_j e^{y_j}}$$

- Just a normalization of results.



Which deep learning framework to use?

Keras

mxnet

TensorFlow

The Microsoft Cognitive Toolkit

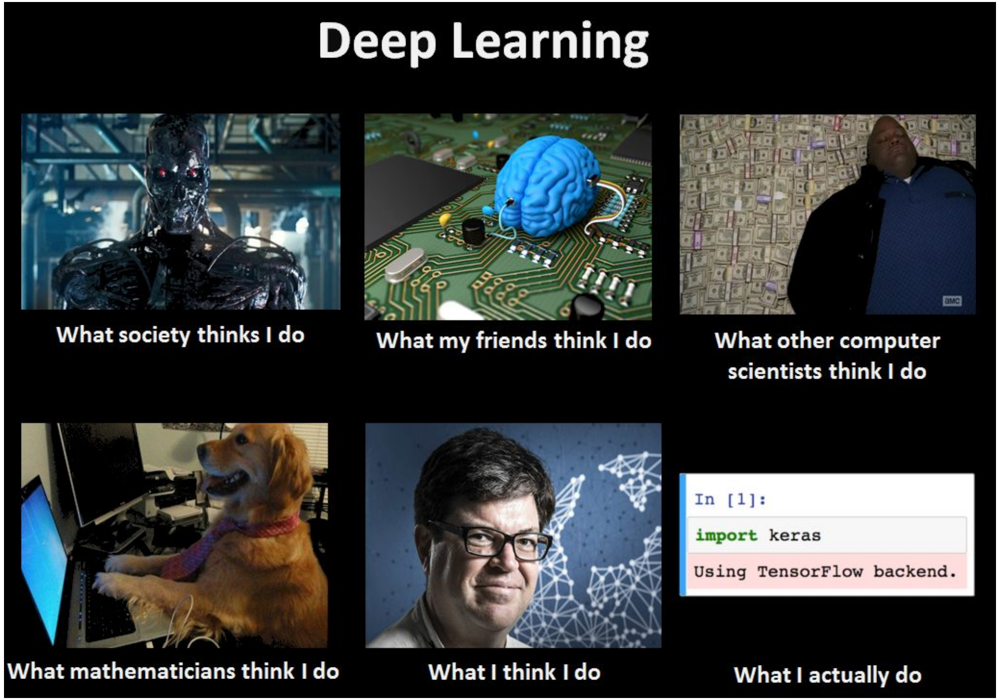
PYTORCH

Caffe

Chainer

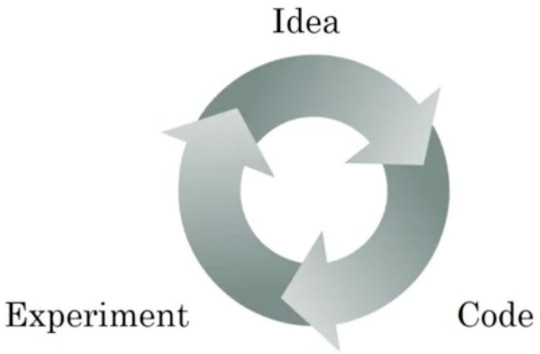
Which deep learning framework to use?

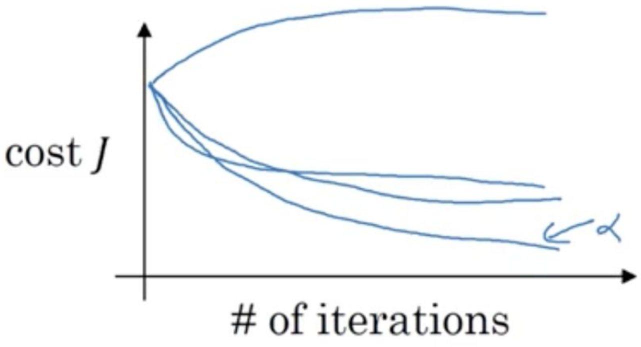
Hands on: Softmax regression using Keras

- 

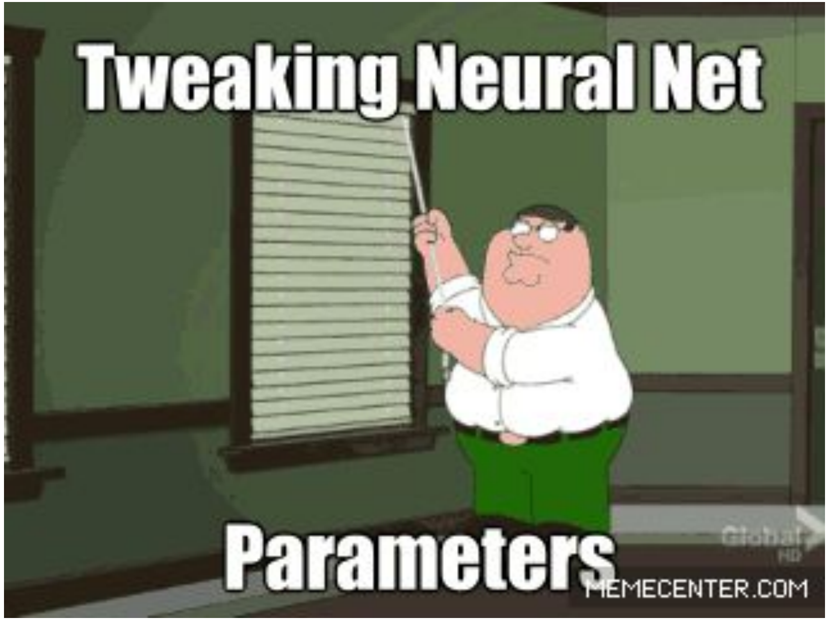
Hyperparameters!

- Parameters which can not be learned.
  - E.g. learning rate, number of layers, number of units, epochs (iterations), choice of activation functions.
- How to find the optimal values?
  - Trial and error and plot the cost function.





Hyperparameters!



Terima kasih!

Explain homework!