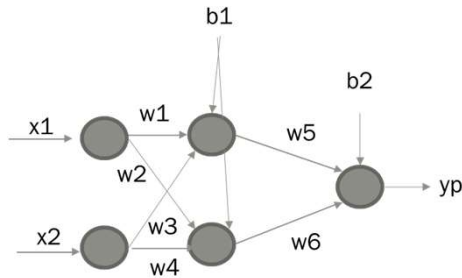


Training a NNet

- Choose
 - Architecture (number of hidden layers and neurons in each layer)
 - Activation function
 - Cost function
 - Optimization method
 - Performance metric
- Initialize
 - Initial guess of weights and Bias
- Predict values for all samples-> forward pass (1)
- Estimate cost function→ (2)
- Reduce cost function→ backward pass (3)
- Evaluate performance on train and holdout
- Repeat 1-3 till cutoff/convergence

forward Propagation

	g	it1
x1	1	
x2	2	
Ya (actual)	3	
w1	0.1	
w2	0.2	
w3	0.1	
w4	0.2	
b1	1	
w3	0.3	
w4	0.4	
b2	0.5	
yp		
CF		



$$Cf = 1/100 * \sum \frac{1}{2} (Y_a - Y_p)^2$$

$$HN1i \rightarrow w1x1 + w3x2 + b1$$

$$HN2i \rightarrow w2x1 + w4x2 + b1$$

$$ONi \rightarrow w5HN1o + w6HN2o + b2$$

$$Yp \rightarrow \frac{1}{1 + e^{-ONi}}$$

$$W1(new) = w1(old) - \alpha \frac{dCf}{dw1}$$

$$W2(new) = w2(old) - \alpha \frac{dCf}{dw2}$$

$$W3(new) = w3(old) - \alpha \frac{dCf}{dw}$$

$$HN1o \rightarrow \frac{1}{1 + e^{-HN}}$$

$$HN2o \rightarrow \frac{1}{1 + e^{-HN2i}}$$

$$Yp = x1 * w1 * w5 + x1 * w3 * w6 + x2 * w2 * w5 + x2 * w4 * w6 + 2b1 + b2$$

$$Yp = x1(w1 * w5 + w3 * w6) + x2(w2 * w5 + w4 * w6) + 2b1 + b2$$

Back propagation

Let's consider updating w_5

Objective is to evaluate $\frac{dCf}{dw_5}$ to calculate $W_5(\text{new}) = w_5(\text{current}) - \alpha \frac{dCf}{dw_5}$

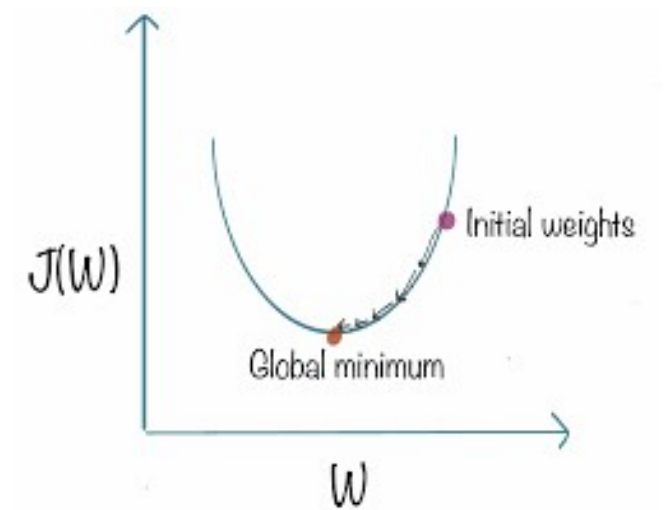
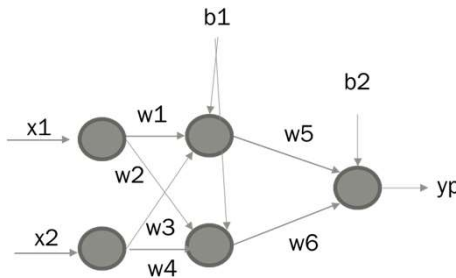
$$Cf = \frac{1}{2}(Y_a - Y_p)^2 \quad Y_p \rightarrow \frac{1}{1 + e^{-ON_i}} \quad ON_i \rightarrow w_5 HN_{1o} + w_6 HN_{2o} + b_2$$

$$\frac{dCf}{dw_5} = \frac{\partial Cf}{\partial Y_p} \times \frac{\partial Y_p}{\partial ON_i} \times \frac{\partial ON_i}{\partial w_5}$$

$$\frac{\partial Cf}{\partial Y_p} = (Y_a - Y_p)$$

$$\frac{\partial Y_p}{\partial ON_i} = \frac{1}{(1 + e^{-ON_i})^2}$$

$$\frac{\partial ON_i}{\partial w_5} = HN_{1o}$$



Gradient descent

- Gradient Descent
- 10,000 records
 - Batch gradient descent → 1 update
 - Stochastic gradient descent (specify epochs) → 10000 updates
 - Mini-Batch gradient descent (100 samples for each stochastic evaluation) → 100
- Momentum-Based Gradient Descent
 - Slow convergence
 - Stuck in local minima
- Nesterov momentum update

Without momentum:

$$w_{t+1} = w_t - \eta \nabla w_t$$

$$b_{t+1} = b_t - \eta \nabla b_t$$

With momentum:

$$update_t^w = \gamma \cdot update_{t-1}^w + \eta \nabla w_t$$

$$w_{t+1} = w_t - update_t^w$$

$$update_t^b = \gamma \cdot update_{t-1}^b + \eta \nabla b_t$$

$$b_{t+1} = b_t - update_t^b$$

Learning rate

- Same learning rate for all parameters do not work
- adaptive learning rate methods
 - Adagrad
 - RMSprop
 - Adadelata
 - Adam

Activation function

- Nonlinear transformation of the inputs
- This contributes significantly to the learning ability of NNets

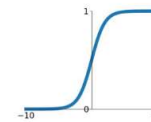
<https://engmrk.com/activation-function-for-dnn/>

https://en.wikipedia.org/wiki/Activation_function

Activation Functions

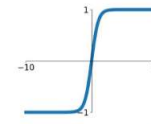
Sigmoid

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



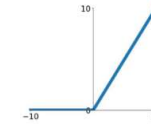
tanh

$$\tanh(x)$$



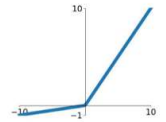
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

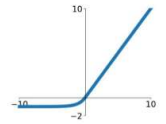


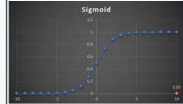
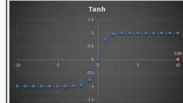
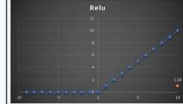
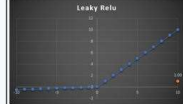
Maxout

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

ELU

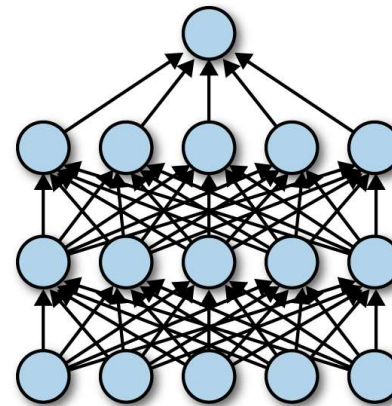
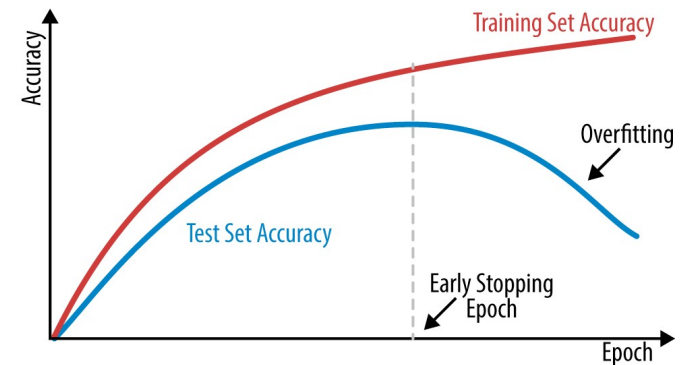
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



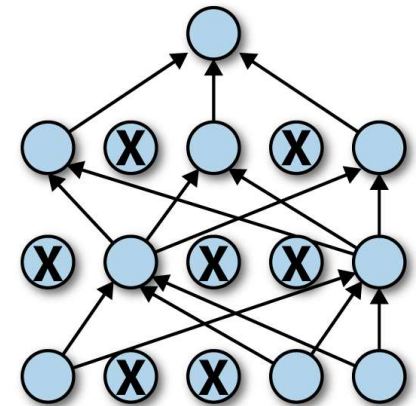
Name	Plot	Equation	Derivative
Sigmoid		$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Tanh		$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$f'(x) = 1 - f(x)^2$
Rectified Linear Unit (relu)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Leaky Rectified Linear Unit (Leaky relu)		$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0.01 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$

Avoiding overfitting in NNets

- Early stopping
- Drop out
- Batch Normalization
- L1 and L2 regularizations



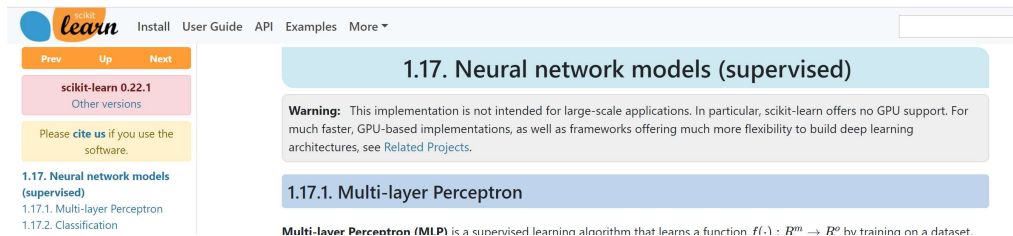
(a) Standard Neural Net



(b) After applying dropout

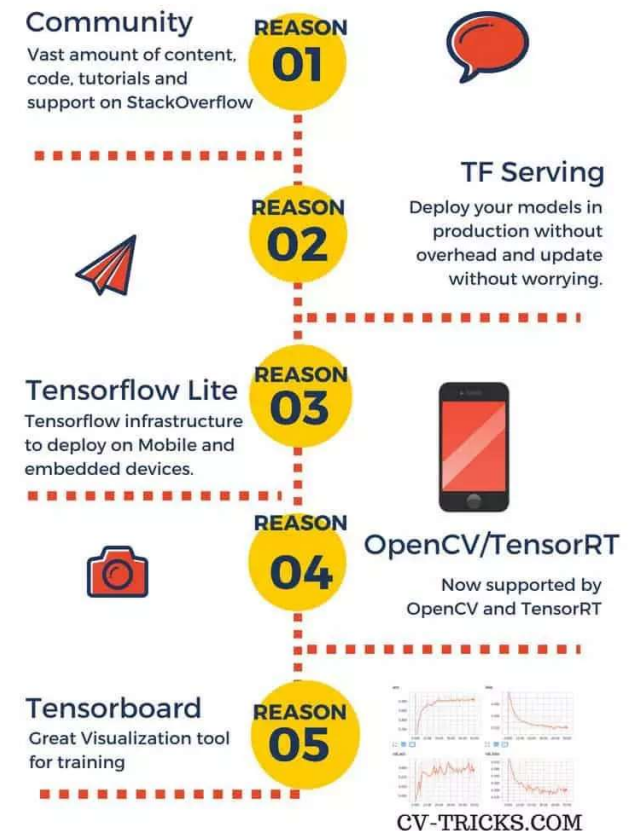
TensorFlow

- Why not Scikit-learn



- Other popular frame work
- Pytorch Vs TensorFlow
- **Tensorflow** is the most famous library used in production for deep learning models
- **TensorFlow** is not that **easy** to use.
- **Keras** is a high level API built on **TensorFlow**. It is more user-friendly and **easy** to use as compared to TF

5 Reasons to choose Tensorflow



References

- <https://dzone.com/articles/feature-engineering-for-deep-learning>
- <https://benanne.github.io/2014/08/05/spotify-cnns.html>
- <https://towardsdatascience.com/why-automated-feature-engineering-will-change-the-way-you-do-machine-learning-5c15bf188b96>

Grayscale, RGB, HSV, CMYK

Week-2

Revisit forward and back propagation

Activation functions

Loss functions

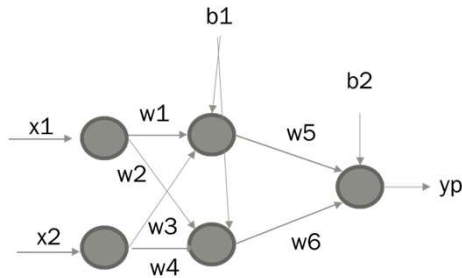
Weight initialization

Training steps

Regression Hands-on:

forward Propagation

	g	it1
x1	1	
x2	2	
Ya (actual)	3	
w1	0.1	
w2	0.2	
w3	0.1	
w4	0.2	
b1	1	
w3	0.3	
w4	0.4	
b2	0.5	
yp		
CF		



$$Cf = 1/100 * \sum \frac{1}{2} (Y_a - Y_p)^2$$

$$HN1i \rightarrow w1x1 + w3x2 + b1 \rightarrow \text{pre-activation}$$

$$HN2i \rightarrow w2x1 + w4x2 + b1$$

$$ONi \rightarrow w5HN1o + w6HN2o + b2$$

$$Yp \rightarrow \frac{1}{1 + e^{-ONi}}$$

$$W1(\text{new}) = w1(\text{old}) - \alpha \frac{dCf}{dw1}$$

$$W2(\text{new}) = w2(\text{old}) - \alpha \frac{dCf}{dw2}$$

$$W3(\text{new}) = w3(\text{old}) - \alpha \frac{dCf}{dw3}$$

$$HN1o \rightarrow \frac{1}{1 + e^{-HN1i}}$$

$$HN2o \rightarrow \frac{1}{1 + e^{-HN2i}}$$

$$Yp = x1 * w1 * w5 + x1 * w3 * w6 + x2 * w2 * w5 + x2 * w4 * w6 + 2b1 + b2$$

$$Yp = x1(w1 * w5 + w3 * w6) + x2(w2 * w5 + w4 * w6) + 2b1 + b2$$

Back propagation

Let's consider updating w_5

Objective is to evaluate $\frac{dCf}{dw_5}$ to calculate $W_5(\text{new}) = w_5(\text{current}) - \alpha \frac{dCf}{dw_5}$

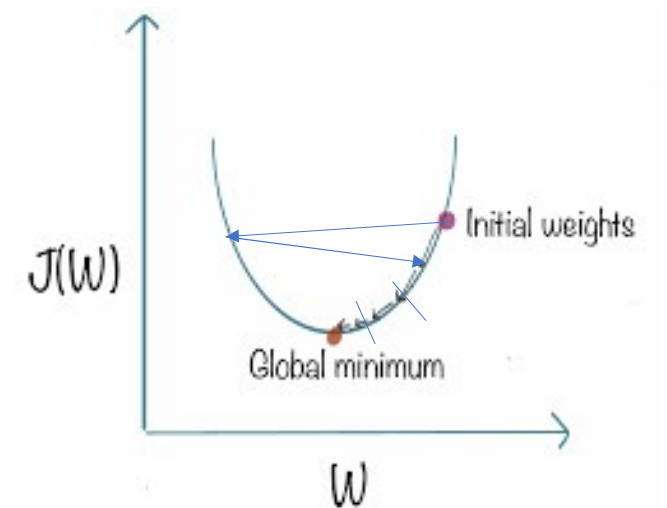
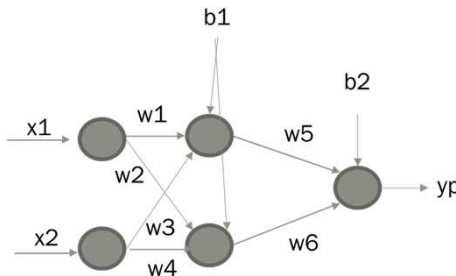
$$Cf = \frac{1}{2}(Y_a - Y_p)^2 \quad Y_p \rightarrow \frac{1}{1 + e^{-ON_i}} \quad ON_i \rightarrow w_5 HN_{1o} + w_6 HN_{2o} + b_2$$

$$\frac{dCf}{dw_5} = \frac{\partial Cf}{\partial Y_p} \times \frac{\partial Y_p}{\partial ON_i} \times \frac{\partial ON_i}{\partial w_5}$$

$$\frac{\partial Cf}{\partial Y_p} = (Y_a - Y_p)$$

$$\frac{\partial Y_p}{\partial ON_i} = \frac{1}{(1 + e^{-ON})}$$

$$\frac{\partial ON_i}{\partial w_5} = HN_{1o}$$



Activation function

- Nonlinear transformation of the inputs
- This contributes significantly to the learning ability of NNets

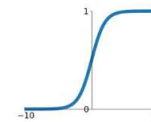
<https://engmrk.com/activation-function-for-dnn/>

https://en.wikipedia.org/wiki/Activation_function

Activation Functions

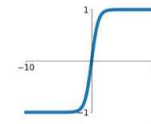
Sigmoid

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



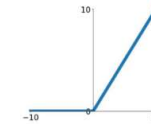
tanh

$$\tanh(x)$$



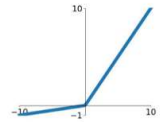
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

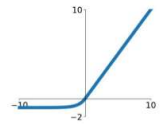


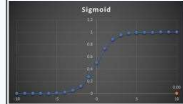
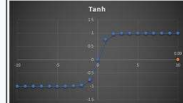
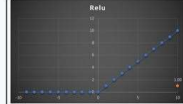
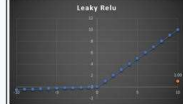
Maxout

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

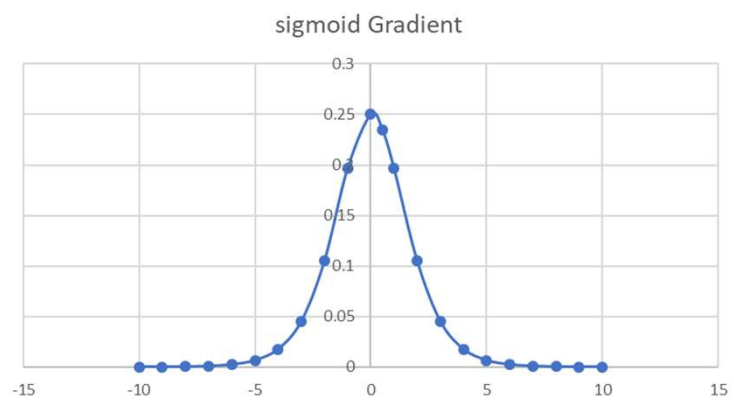
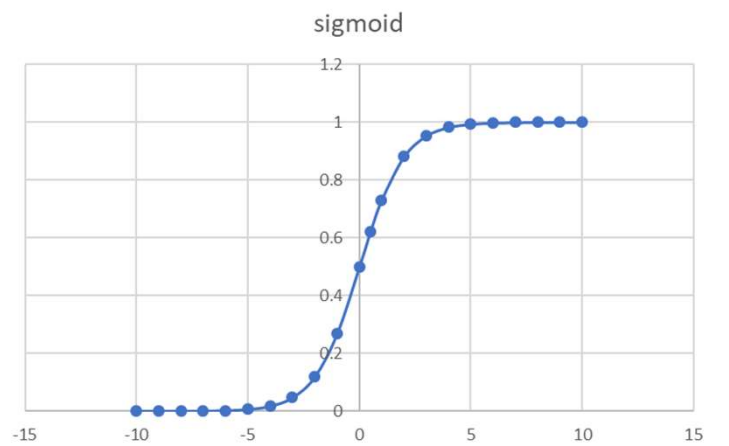
ELU

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

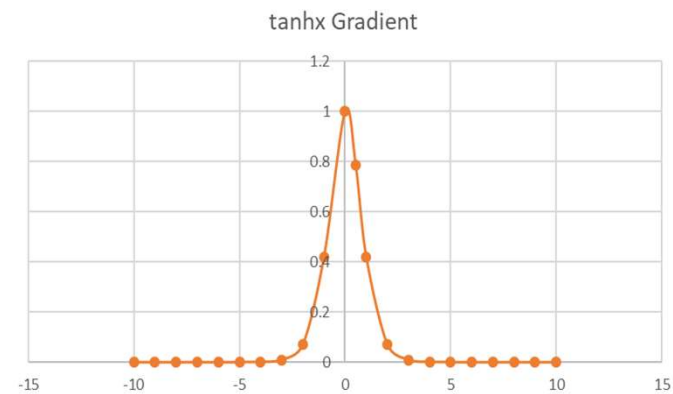
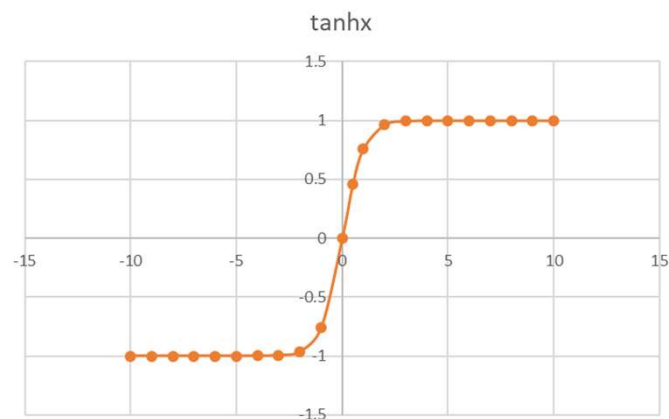
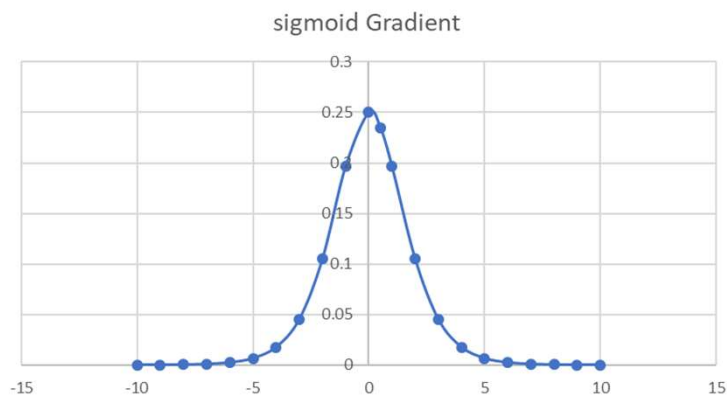
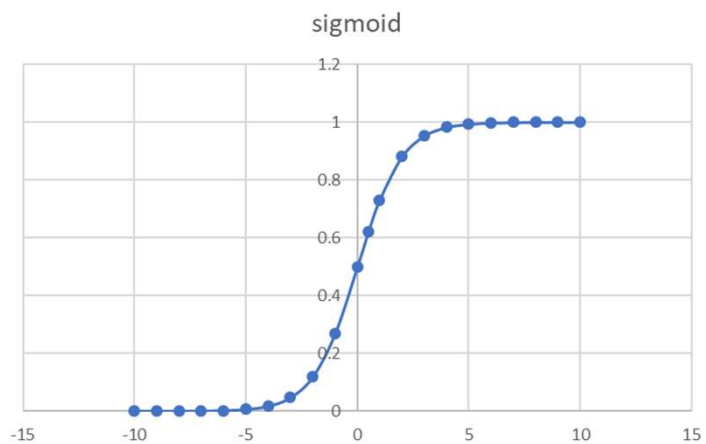


Name	Plot	Equation	Derivative
Sigmoid		$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Tanh		$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$f'(x) = 1 - f(x)^2$
Rectified Linear Unit (relu)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Leaky Rectified Linear Unit (Leaky relu)		$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0.01 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$

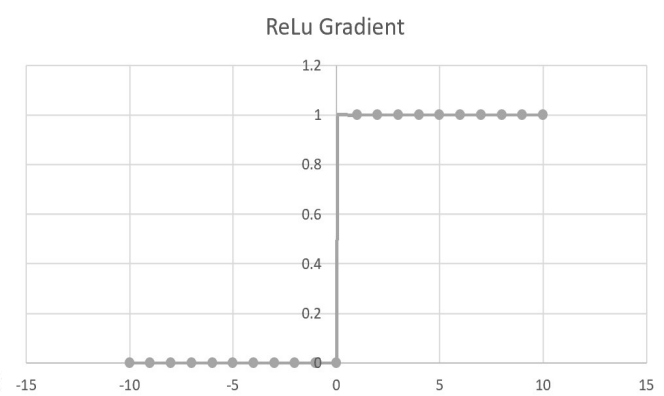
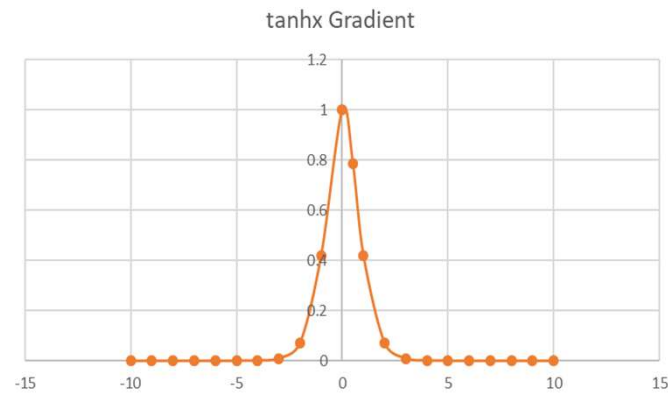
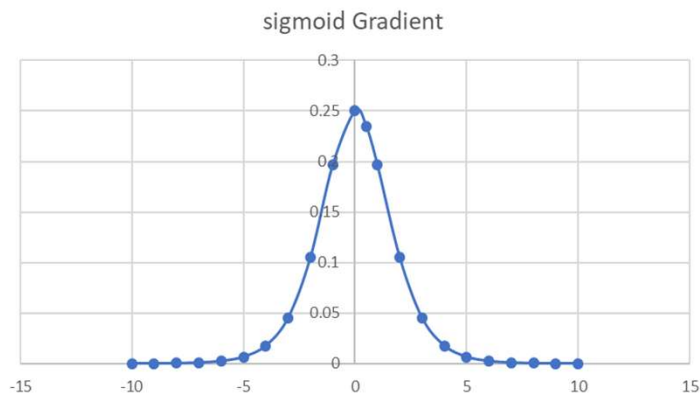
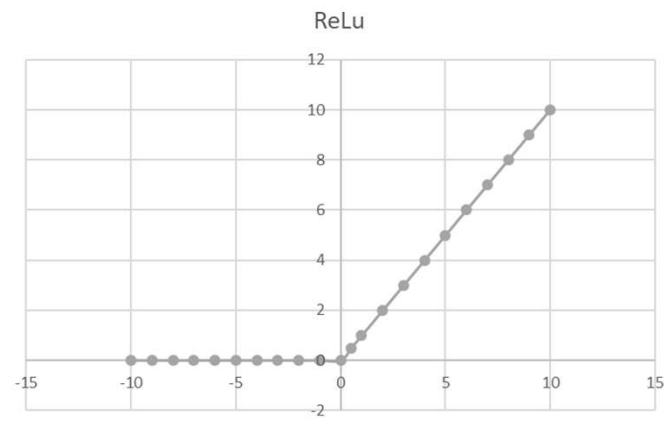
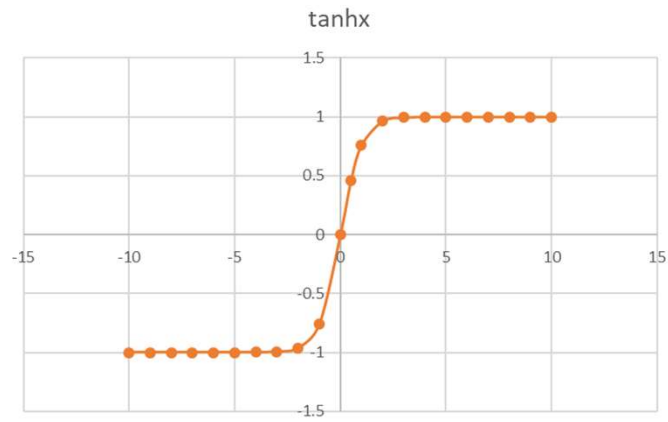
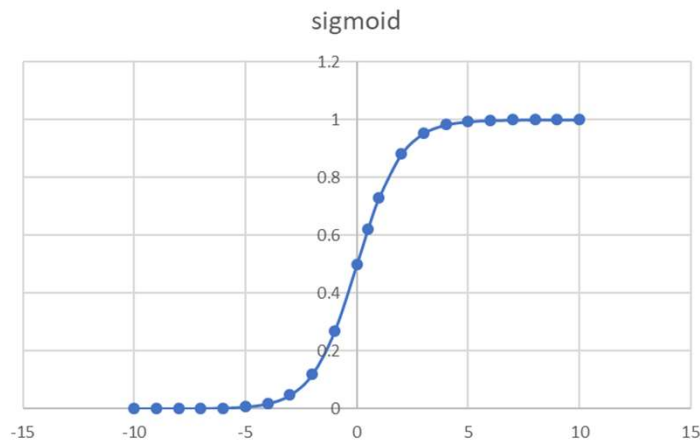
Sigmoid



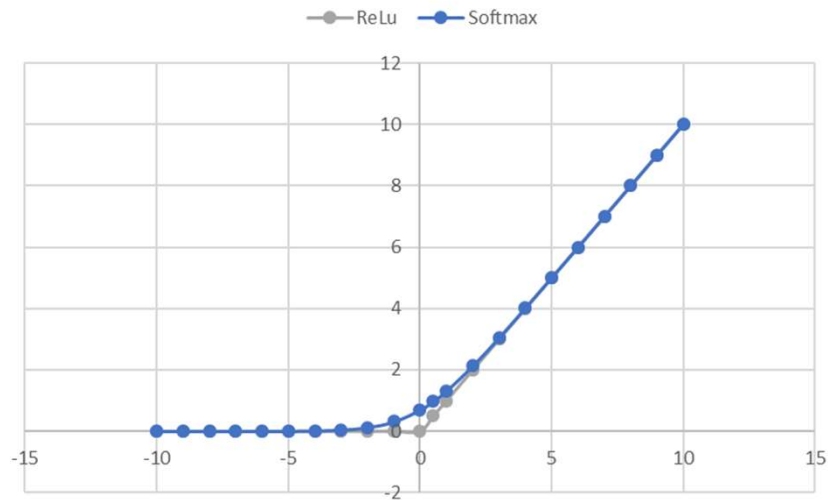
Sigmoid and tanh



Sigmoid, tanh, ReLu



ReLu Vs softplus



No specific advantage of softplus over ReLu
Larger Computational cost for softplus

$$f(x)=\ln(1+\exp(x))$$

Regression exercise:

Use ReLu for hidden layers

No activation function for output layer

Loss function is MSE

For Classification

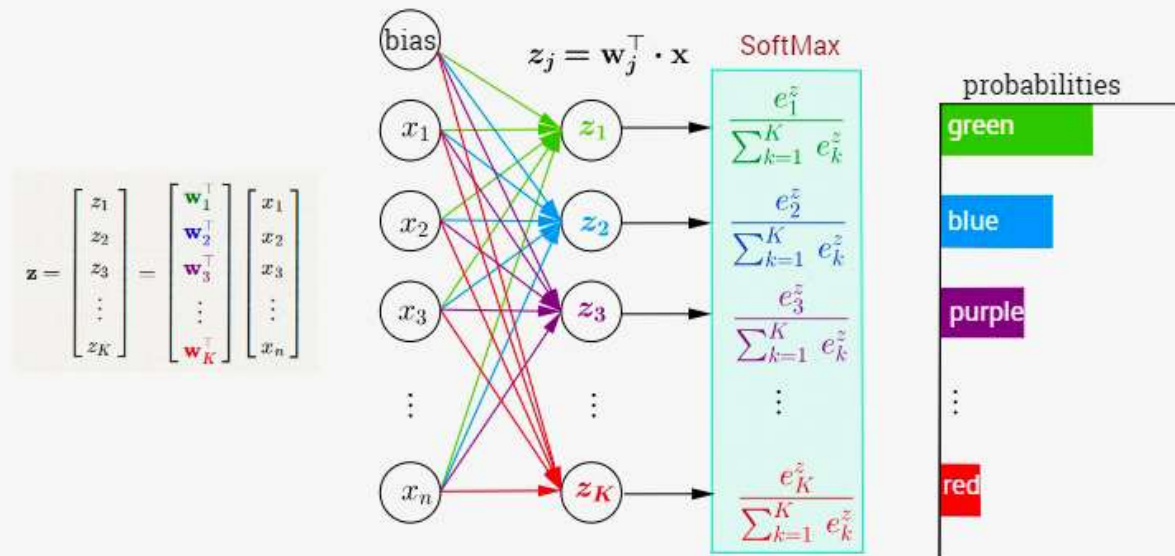
Use ReLu for hiddenlayers

Use Softmax for outputlayer

Loss function is cross entropy

SoftMax

Multi-Class Classification with NN and SoftMax Function



$$\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}$$

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

Why use softMax why not sigmoid?

Cost function

- Activation function is for non linear transformation
- Cost function is for updating the weights during learning
- Regression → cost function ?? MSqError
- Classification → Binary cross entropy /multi-class cross entropy

Cost function for multi-label classification

- **Cross entropy/categorical cross entropy (CCE)**
 - Requires one-hot encoding for the class labels
 - Classification of a car, bus, bike or truck → $[[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]]$
 - For very large dataset not memory or computation efficient
- **Sparse Categorical cross entropy (SCCE)**
 - Requires just label encoding for class labels
 - Classification of a car, bus, bike or truck → $[0,1,2,3]$
 - Computation and memory efficient
- **Note:** the output layer is same in both and the activation function used is also same. We get the probability distribution from the output layer

<https://www.machinecurve.com/index.php/2019/10/06/how-to-use-sparse-categorical-crossentropy-in-keras/>

What to focus for Training a Neural Net

1. **Hyper tune the architecture**
2. **Focus on activation function**
3. **Focus on cost function**
4. **Focus on optimization method→**
 - Using a single learning rate is not a good idea
 - Using simple gradient descent method is not a good idea
 - Optimization method called **ADAM** which addresses the above two limitations
 - Uses different learning rates for each parameter
 - Uses Momentum based gradient descent for weight update
5. **Focus on weight initialization→ Xavier initialization, Kaiming He Initialization**
6. **Regularization→ batch normalization and Drop out**

How Many Layers and Nodes to Use?

- No straight forward answer
- Use systematic experimentation to figure out the optimum L and L_n
- Fit models on a smaller subset of the training dataset to speed up the search
- *countless heuristics on how to estimate the number of layers and either the total number of neurons or the number of neurons per layer \rightarrow not useful beyond the examples*
- Use transfer learning wherever possible

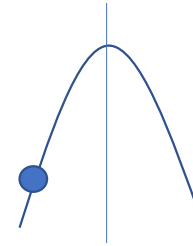
Weight initialization

- Wrong weight initialization leads to Vanishing or exploding gradient problems
- Scale the input to mean=0 and std.dev=1. initializing the weights also using random normal/uniform leads to the EG or VG problems
- Xavier initialization sets a layer's weights to values chosen from a random normal distribution that's bounded between

$$\pm \frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}}$$

- n_i is the number of incoming network connections, or “fan-in,” to the layer,
- n_{i+1} is the number of outgoing network connections from that layer, also known as the “fan-out.”

Weight initialization



- Kaiming He Initialization
- Random numbers from standard normal distribution
- Multiply each randomly chosen number by $\sqrt{2/n}$ where n is the number of incoming connections coming into a given layer from the previous layer's output ("fan-in")
- [-0.05, 0.04, 0.08,.....] mean=0, std dev=1
- Bias tensors are initialized to zero.