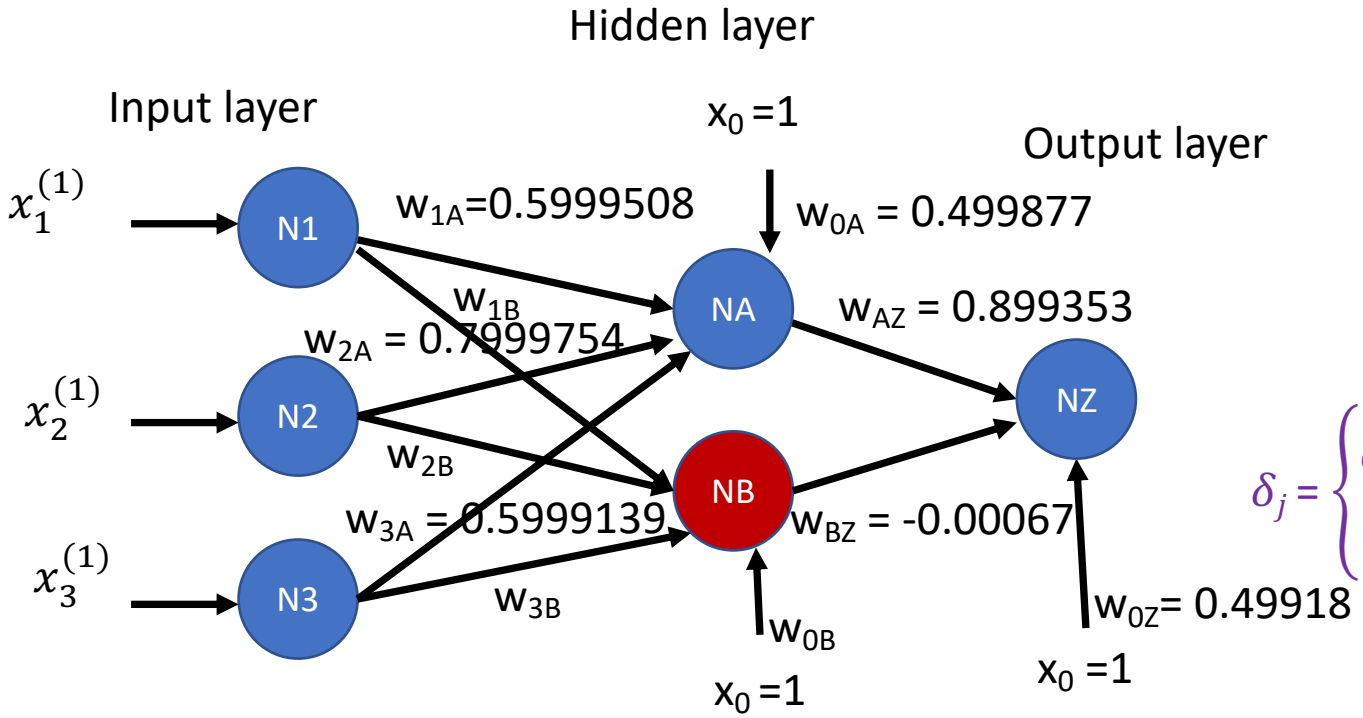
ECS 171 Machine Learning

Lecture8: DNN (hyperparameter tuning, Momentum, mini-batch GD, L1,L2 regularization, Dropout)

Instructor: Dr. Setareh Rafatirad

MLP: Backpropagation Example



Source: Discovering Knowledge in Data D. Larose

actual$_z$= 0.8 → residual error = 0.8 – 0.875 = -0.075

N_1 =0.4	N_A =0.7892
N_2 =0.2	N_B =0.8176
N_3 =0.7	N_z =0.875

residual error = 0.8 – 0.875 = -0.075

$learning\ rate\ ; 0 \leq \eta \leq 1$

$Assume: \eta = 0.1$

For output layer node

$$\delta_j = \begin{cases} output_j(1 - output_j)(actual_j - output_j) \\ output_j(1 - output_j) \sum w_{jk}\delta_j \end{cases}$$

For hidden layer nodes

Weighted sum of the error responsibilities for the nodes downstream from the particular hidden layer node.

The only node downstream from N_B is N_Z.

$\delta_B = N_B(1 - N_B)(w_{BZ}\delta_Z) =$
0.8176(1-0.8176) (0.9)(-0.0082) = -0.0011

Update Rule:

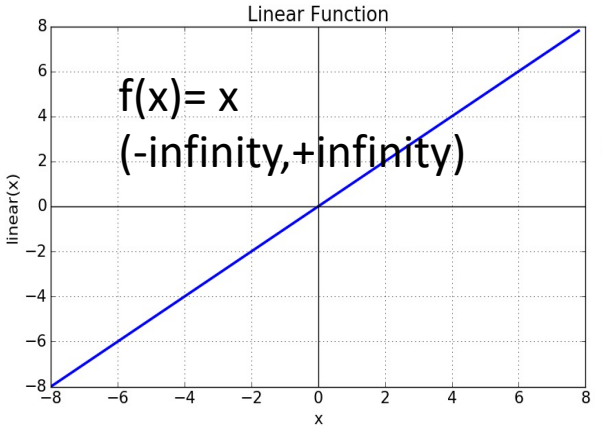
$$w_{0B,new} = w_{0B,current} + \Delta w_{0B}$$

$$\eta\delta_B x_j$$

Which weights to update?
Do this on your own as a practice at home.

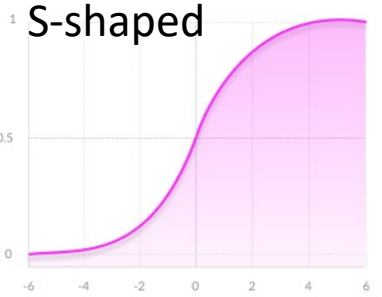
Selection of an Activation Function

- Activation functions determine the output of a NN (determine whether a neuron should ”fire”.
- They can normalize the output of each neuron.
- Computationally efficient
- Increasingly use non-linear functions to learn complex data and provide accurate predictions.
- Linear function doesn’t help with the complexity of data
- Non-Linear activation functions help the model to generalize or adapt with the variety of data
- Examples:
  - Linear function (linear line)
  - Binary step function
  - Non-linear activation functions
  - Sigmoid/logistic, Relu, Parametric Relu, Tanh/hyperbolic tangent , softmax, Swish
  - Swish outperforms Relu in terms of classification accuracy.

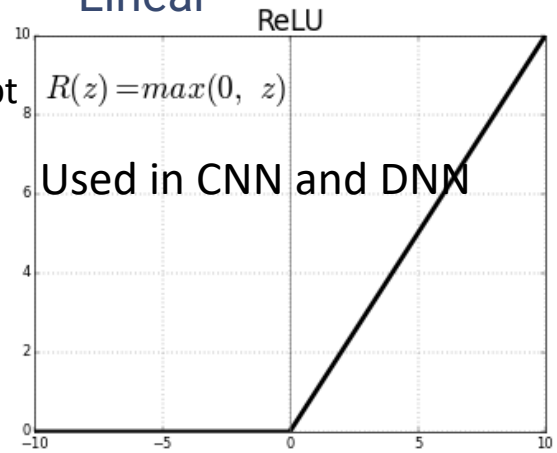


Linear

Output range: [0,1]
S-shaped

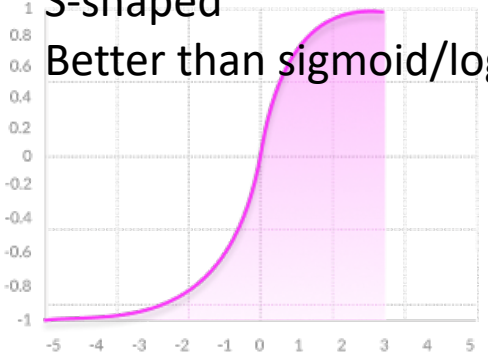


Sigmoid / Logistic



ReLU

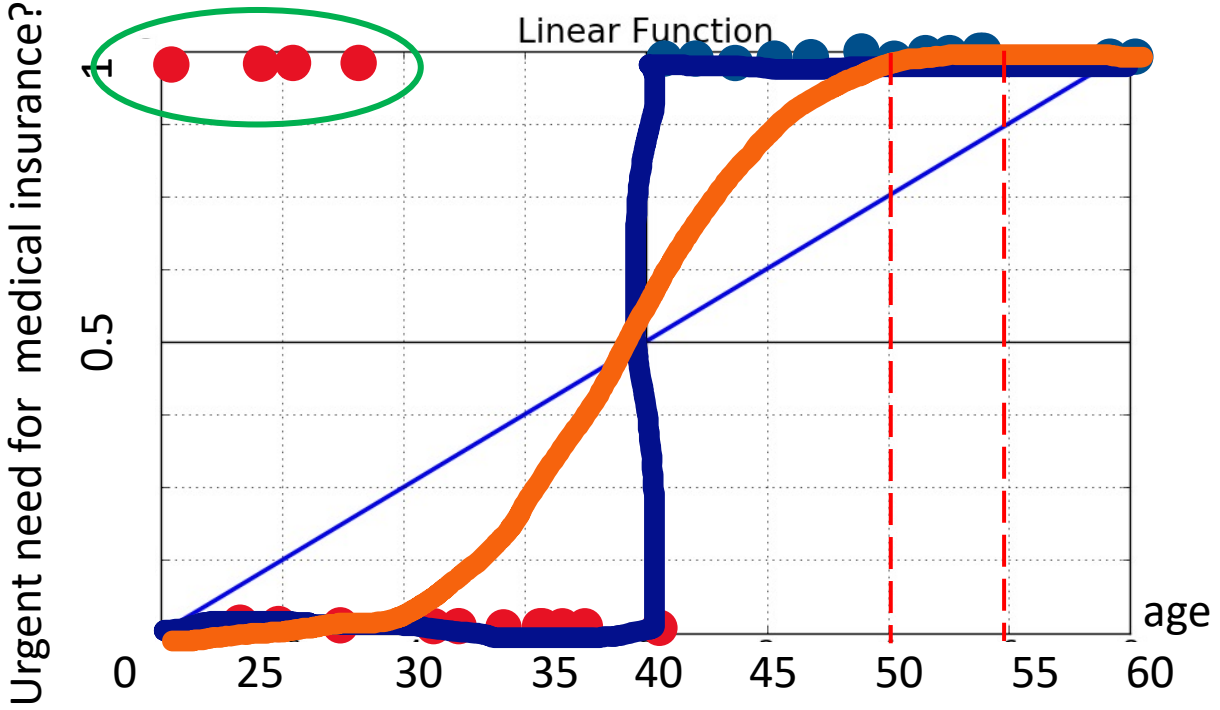
Output range: [-1,1]
S-shaped
Better than sigmoid/logistic

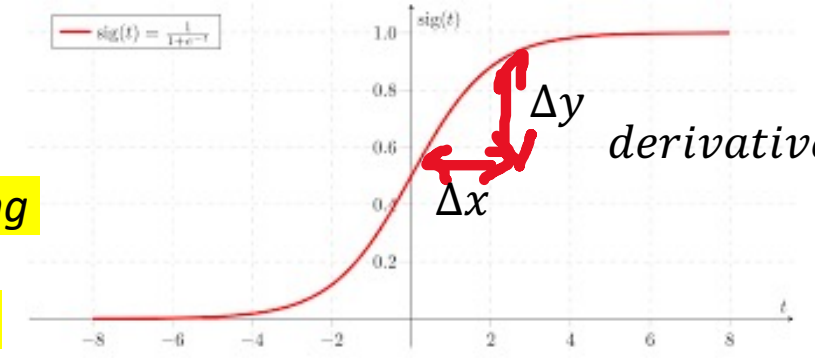


TanH / Hyperbolic Tangent

Application of Activation Functions

- Linear Function :linear problems
- Binary Step Function (output is 0 or 1): Binary classification, not good for multi-class classification such as classifying number images into 0-9 labels.
- Sigmoid and Tanh: good for multiclass classification (see the next slide)
  - You can use sigmoid in the output layer as a general guideline. In general, using tanh instead of sigmoid is better. Why? tanh calculates a mean of 0 so it will center your data
  - there are issues with these functions such as slowing down the learning process aka “Vanishing Gradients” problem. Therefore, computationally not efficient for hidden layers.

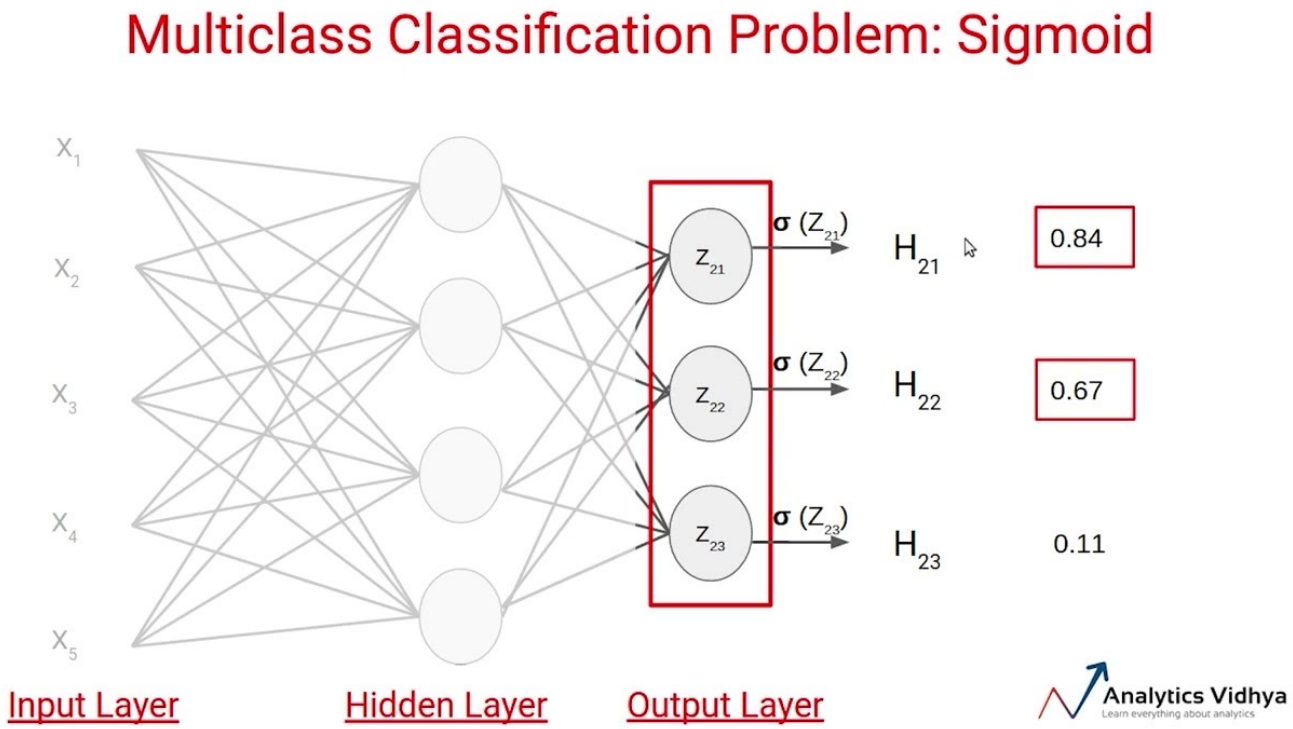




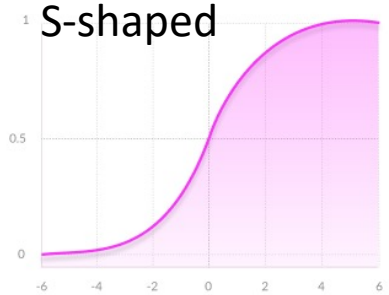
$$derivative = \frac{\Delta y}{\Delta x}$$

Example : how much the likelihood of being in urgent need for medical insurance changes based on how much the age of a person changes.

Application of Activation Functions cont.

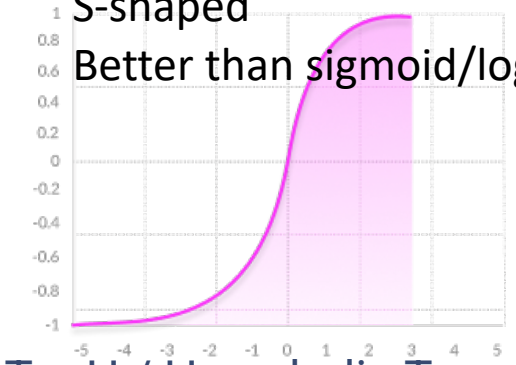


Output range: [0,1]
S-shaped



Sigmoid / Logistic

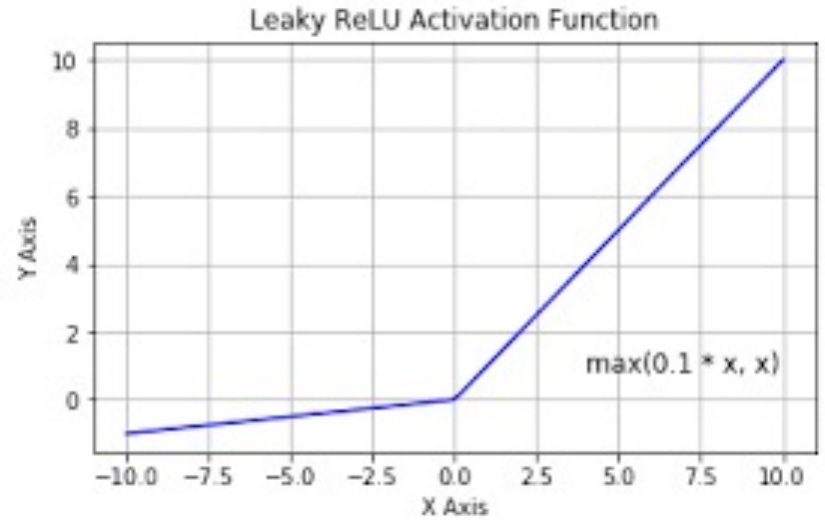
Output range: [-1,1]
S-shaped
Better than sigmoid/logistic

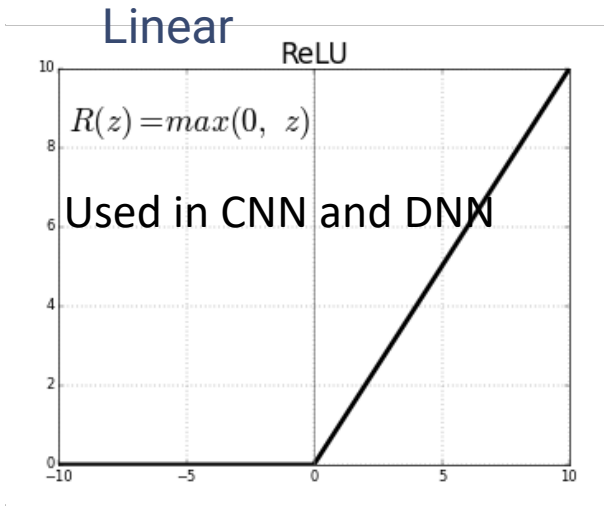


TanH / Hyperbolic Tangent

Application of Activation Functions cont.

- ReLU: very light-weight function , default choice for hidden layers. ReLU also has “Vanishing Gradient” problem.

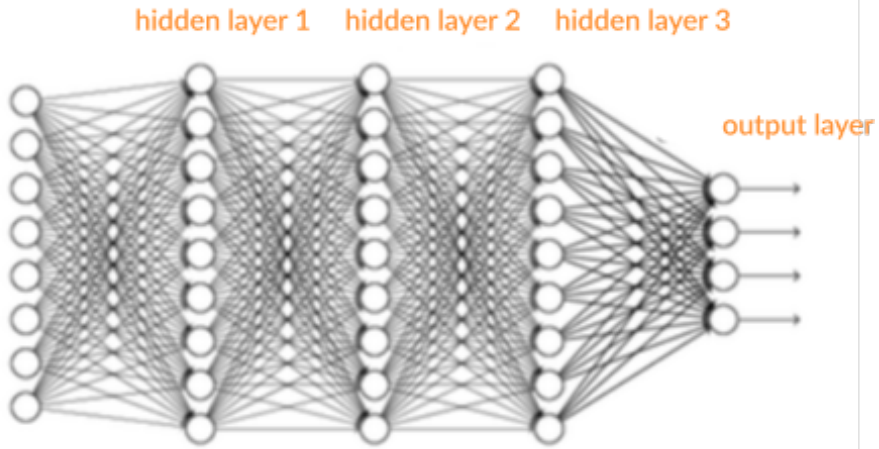




ReLU

In general, choosing an activation function in a NN is based on trial and error.

Deep Neural Network (DNN)

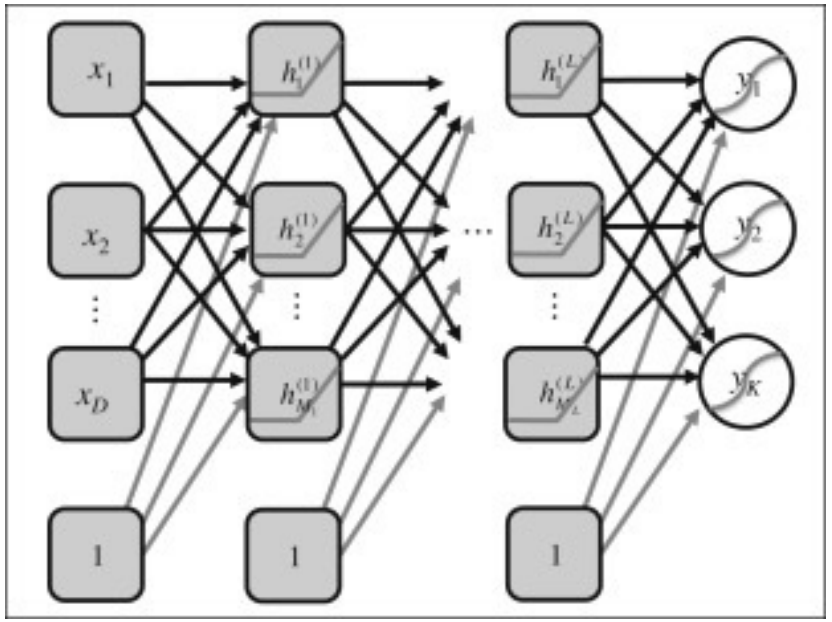


- Deep neural network is a deep layered network that compose computations performed by many layers. It has between 2-8 additional layers of neurons (hidden layers).
- ***h*$^{(l)}$*(x)*** *: hidden layer activation function (at layer l) such as sigmoid, tanh, etc.*
- *The computation for a network with L hidden layers:*

$$f(x) = f\left[a^{(L+1)}\left(h^{(L)}\left(a^{(L)}\left(\ldots\left(h^{(2)}\left(a^{(2)}\left(h^{(1)}\left(a^{(1)}(x)\right)\right)\right)\right)\right)\right)\right)\right]$$

$$a^{(l)}(x) = W^{(l)}x + b^{(l)}$$
$$a^{(l)}(\hat{x}) = \theta^{(l)}\hat{x}\,, l = 1$$
$$a^{(l)}\left(\hat{h}^{(l-1)}\right) = \theta^{(l)}\hat{h}^{(l-1)}\,, l \geq 1$$

$\hat{x}$ indicates that 1 is appended to vector x by convention.



Deep NN are in speech recognition, computer vision, and in competitive challenges such as the image net large scale visual recognition challenge.

Deep Neural Network Hyperparameters

- DNN can be learned using stochastic gradient descent optimization algorithm.
- Hyperparameters are variables that determine the structure of a DNN (such as the number of hidden layers, units (i.e., neurons – many can increase accuracy, while smaller number can cause underfitting) and variables that control how the network is trained (such as the learning rate, number of epochs) .
  - Hyperparameters have a crucial role in a machine learning model's performance.
  - Hyperparameters are not the model's weights.
- Hyperparameter Examples in DNN
  - Learning rate: very important parameter for configuring the network and model performance.
  - Number of epochs (1 Epoch means when the whole training data is passed forward and backward through the neural network only once)
  - Batch size
  - Number of layers
  - Number of neurons at each layer
  - Momentum term value
- How to find the best hyperparameters? Hyperparameter optimization is the task of finding the best hyperparameters for a learning algorithm.

Methods for Hyperparameter Tuning (Optimization)

- Manual Search: Through controlled experiments where each experiment is one combination of hyperparameter values.
  a) Keep all hyperparameters constant except one.
  b) Analyze the effect and make decision about which hyperparameter to change next.
  c) Repeat.
  - Drawback: each experiment requires training the model end-to-end, and can be very slow for Deep Learning algos (e.g., CNN)

Methods for Hyperparameter Tuning (Optimization) cont.

- Other model hyperparameter optimization techniques: offer ways to automatically find the best possible combination of hyperparameter values for a machine learning model. Optimization is a non-trivial task.
  - Grid search
    - Provided by **GridSearchCV** class in Scikit-learn
    - The **GridSearchCV** process constructs and evaluates one model for each combination of parameters.
    - Cross validation is used to evaluate each model (default k=3,this can be overridden by specifying the **cv** argument to the **GridSearchCV** constructor).
  - Random search
  - Bayesian optimization
  - Hyperband

Hyperparameter Sweeps

- Given a dataset, hyperparameter sweep offers efficient ways to automatically find the best possible combination of hyperparameter values for a machine learning model.
- Learning rate has the highest priority in hyperparameter tuning for a neural network
- Benefits of Huperparameter Sweeps
  - Set up hyperparameter searches through declarative configurations
    - Read https://blog.floydhub.com/training-neural-nets-a-hackers-perspective/#introduction-to-declarative-configuration
  - Experiment with the hyperparameter tuning methods listed in the previous slide

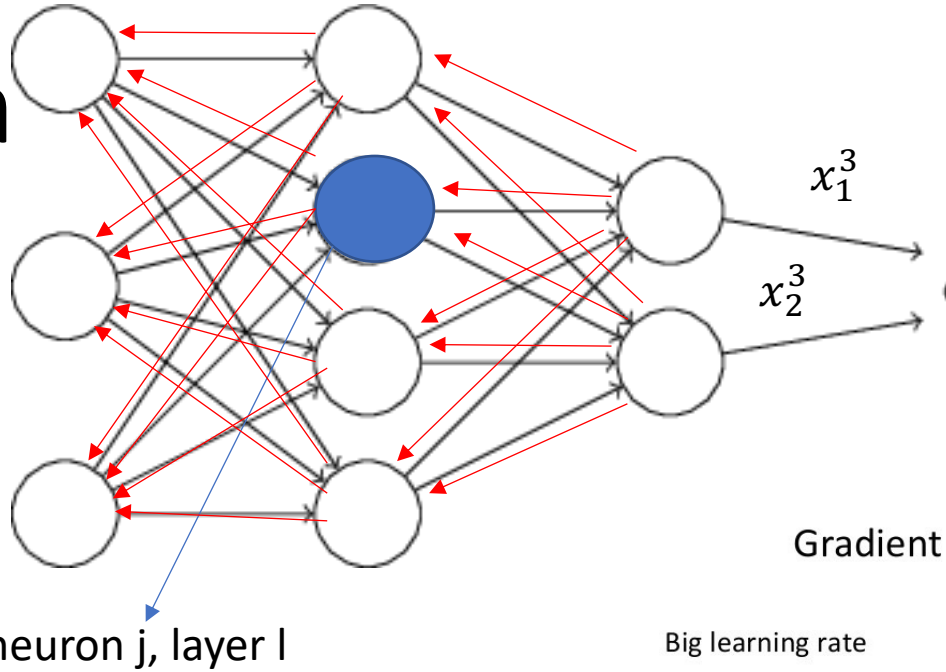
See this documentation for more hands-on perspective : https://docs.wandb.com/sweeps

Backpropagation

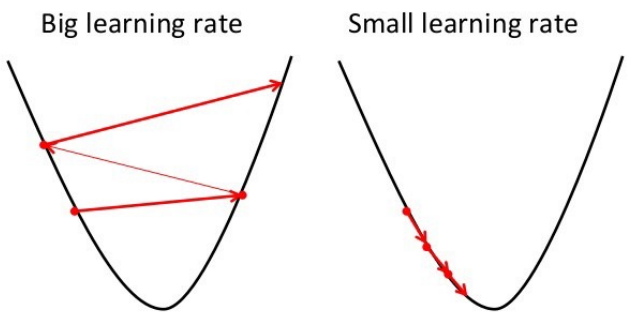
$$E_n = \frac{1}{2}\sum_k (\hat{y}_{nk} - y_{nk})^2$$

Gradient of the error function

$$\frac{\partial E_n}{\partial w_{jk}} = (\hat{y}_{nj} - y_{nj})x_{nj}$$



Gradient Descent



$$\frac{\partial E_n}{\partial w_{jk}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{jk}} \qquad a_j^l = \sigma(\sum_k w_{jk}^l z_k^{l-1} + b_j^l)$$

Update Rule:

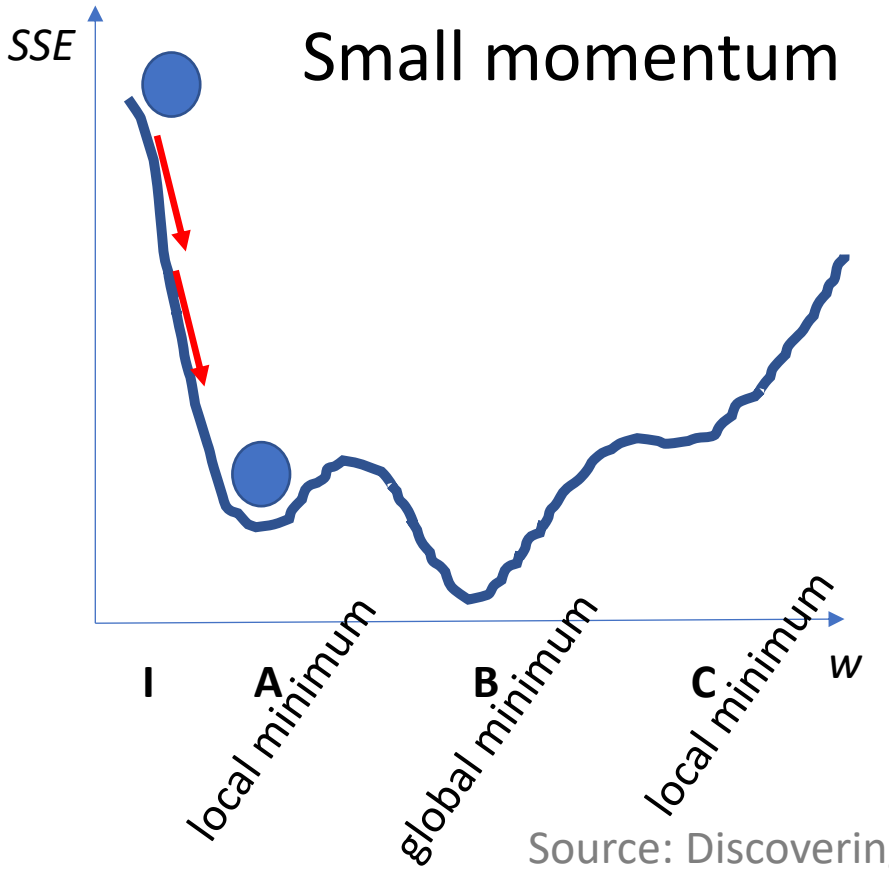
$$w_{new} = w_{current} + \Delta w_{current}$$

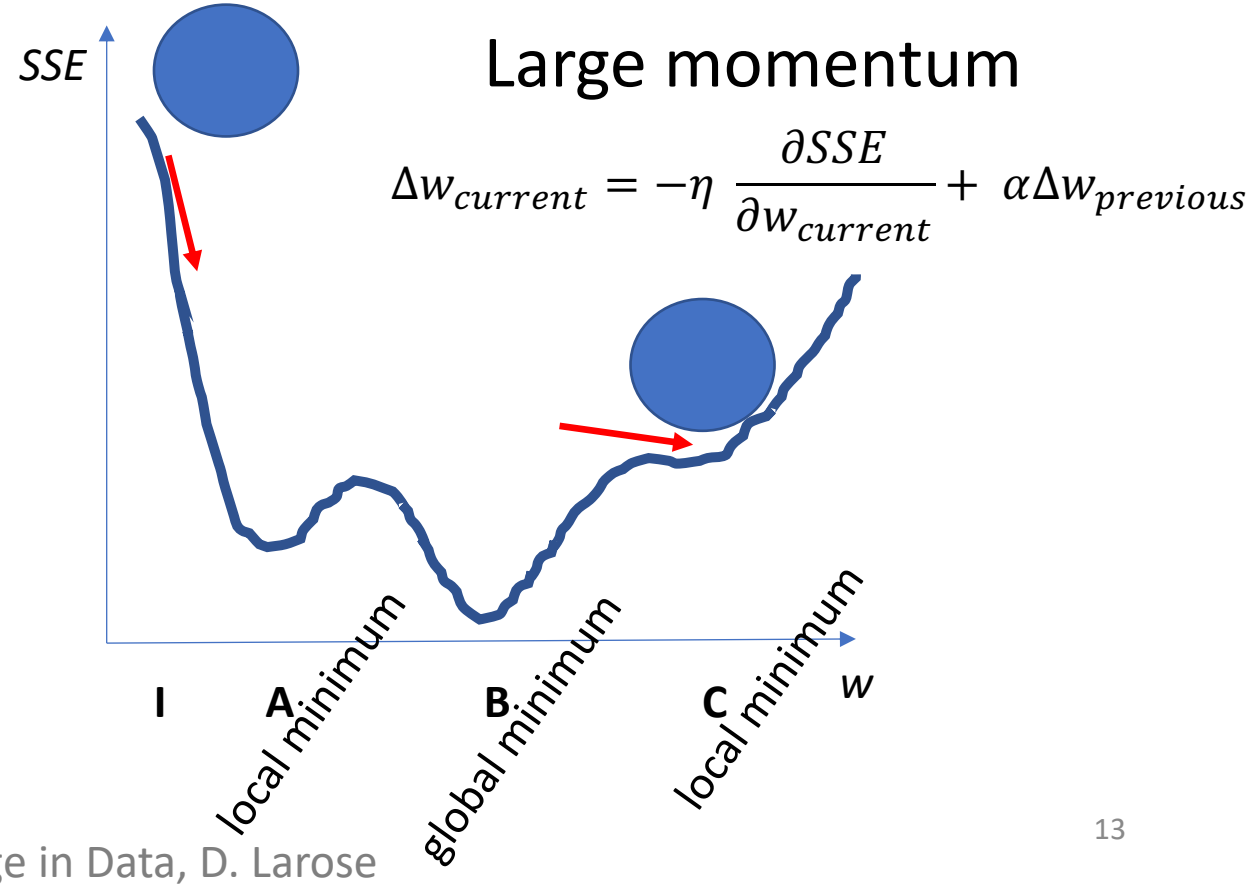
$$\eta\delta_j z_j \qquad \eta : learning\ rate\ ; 0 \leq \eta \leq 1$$

Momentum Term Influence

Backpropagation is made more powerful with the addition of a momentum term α.
The momentum term represents inertia.

- "momentum" helps to know the direction of the next step with the knowledge of the previous steps.
- It helps to prevent oscillations.
- Momentum can also be implemented with mini-batch GD.





Source: Discovering Knowledge in Data, D. Larose

Momentum with Mini-Batch GD

Backpropagation is made more powerful with the addition of a momentum term α. The momentum term represents inertia.

$\eta : learning\ rate\ ; 0 \leq \eta \leq 1 \qquad 0 \leq \alpha < 1$

$$\Delta w_{current} = -\eta \frac{\partial SSE}{\partial w_{current}} + \alpha \Delta w_{previous}$$

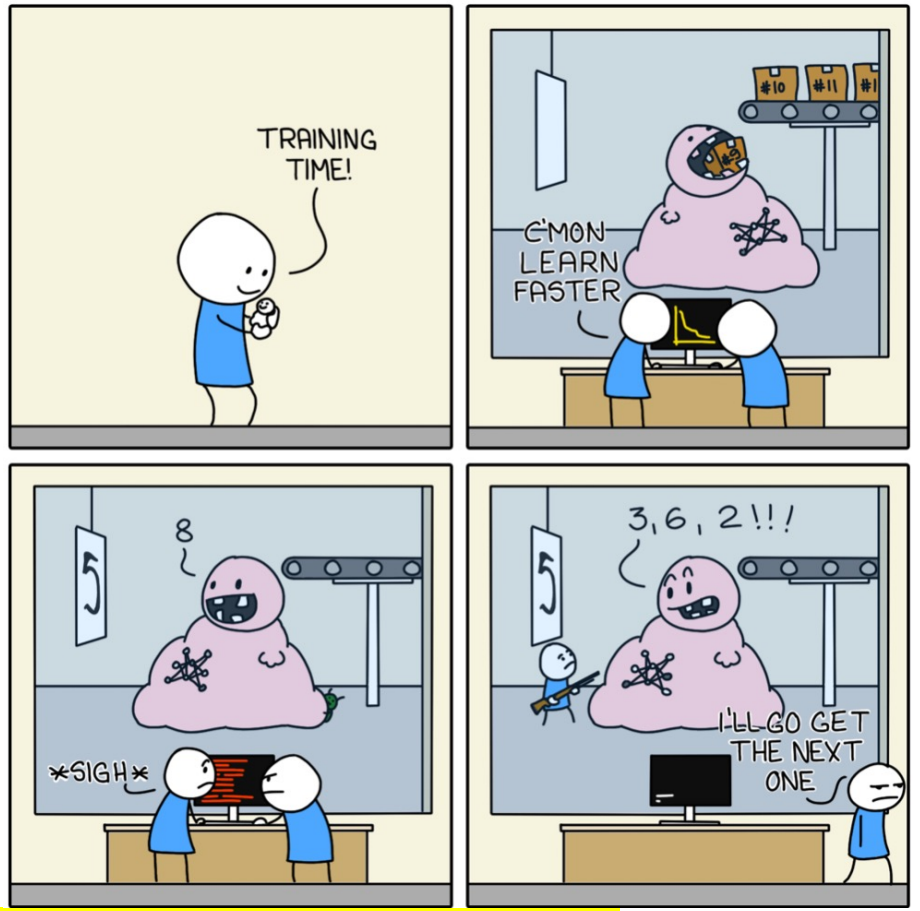
Previous weight adjustment

(Reed & Marks) prove that including momentum in backpropagation algorithm results in the adjustment becoming exponential average of all pervious adjustments:

$$\Delta w_{current} = -\eta \sum_{k=0}^{\infty} \alpha^k \frac{\partial SSE}{\partial w_{current-k}}$$

large values of α allows the algorithm to "remember" more terms in the adjustment history.

small values of α reduce the inertial effects as well as the influence of the recent adjustments, until α=0.



Momentum with Mini-Batch Gradient Descent

mini-batches contain 2 to several hundred samples
For large models, the choice of the mini-batch size is constrained by computational resources.
Batch size impacts the stability and speed of learning.

$if\ \nabla_\theta L(\theta) = gradient\ of\ the\ loss,$
$then\ momentum\ is\ implemented\ by\ computing\ a\ moving\ average:$
$\Delta\theta = -\eta\nabla_\theta L(\theta) + \alpha\Delta\theta^{old}\ ;\ \alpha \in [0,1] \quad \equiv \quad \Delta w_{current} = -\eta\ \frac{\partial SSE}{\partial w_{current}} + \ \alpha\Delta w_{previous}$

α: momentum term

- *Commonly, momentum is first initialized to 0.9 , but it is then tuned similar to learning rate, during the training process. A typical choice of momentum is between 0.5 and 0.9.*
- *Learning rate may be adopted over epochs t to give η_t . In the first few epochs, a fixed learning rate is often used. Then a decaying schedule is followed:*

$$\eta_t = \frac{\eta_0}{1+\varepsilon t}, or\ \eta_t = \frac{\eta_0}{t^\varepsilon}, (0.5 < \varepsilon \leq 1)$$

η_t : learning rate at epoch t, it can be different at each epoch

Mini-Batch Gradient Descent

- Deep learning models are often optimized using mini-batch gradient descent.
- Uses a small subset of the data and updates the weights based on the average gradient of the subset. The rest is the same (initialize the parameters, enter a parameter update loop, terminate by monitoring a test/validation set). In contrast with stochastic GD , the main loop iterates over the mini-batches.
- The mini-batches depending on the batch size, are obtained from the training set. Batches are randomly selected non-overlapping subsets of training set.
- Weights (parameters) are updated after processing each batch.
- Pros
  - more efficient than stochastic gradient descent since the batching allows the efficiency of not having all training data in memory and algorithm implementations.
  - More accurate than batch gradient descent
- Cons
  - Requires an additional hyper-parameter called mini-batch size
  - Error information must be accumulated across mini-batches of training data (similar to batch GD)

Regularization in DNN

Objective: use optimized regularization coefficient and obtain a generalized model.

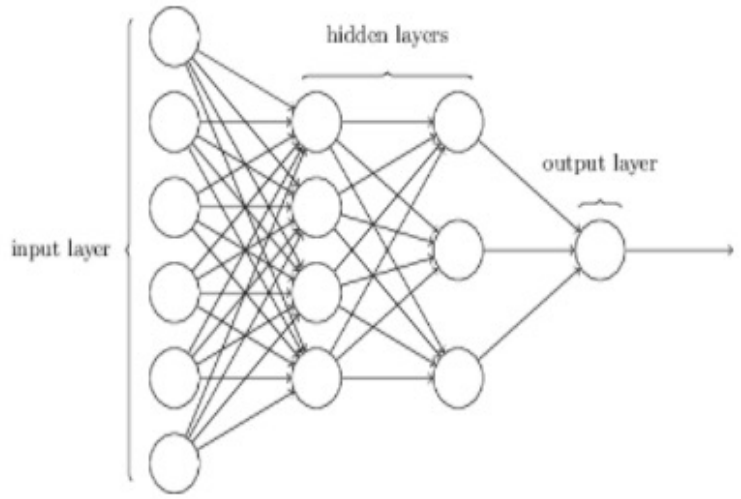
- Regularization Techniques in Deep Learning
  - L2 and L1 regularization
  - Dropout
  - Data augmentation
  - Early stopping

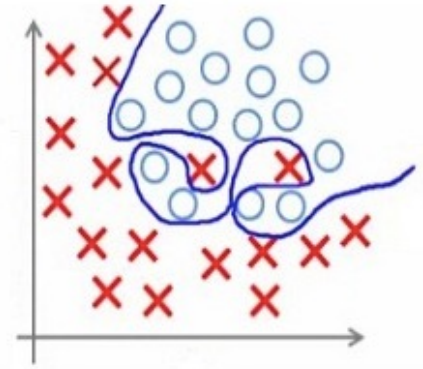
Cost function = $L(\theta)$ + Regularization term

update rule:

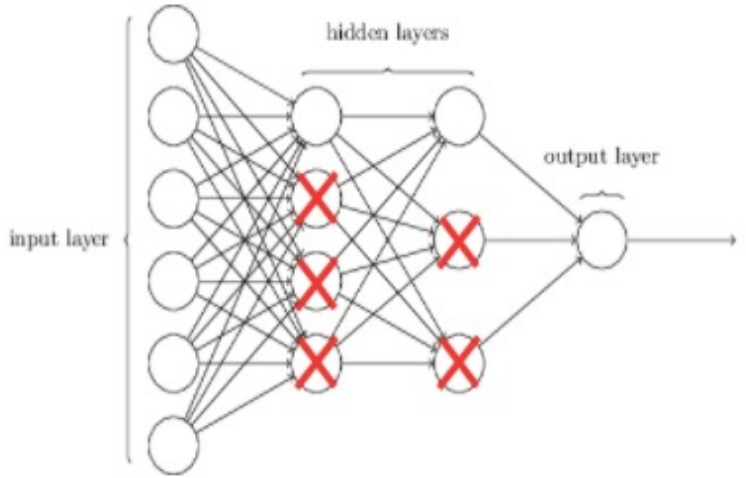
$$\theta_{new} = \theta - \eta_t \left[\frac{1}{B_k}\sum_{i\in I}\left[\frac{\partial L(f(x_i;\theta),y_i)}{\partial\theta}\right] + \frac{B_k}{N}\,\lambda\,\frac{\partial R(\theta)}{\partial\theta}\right]$$

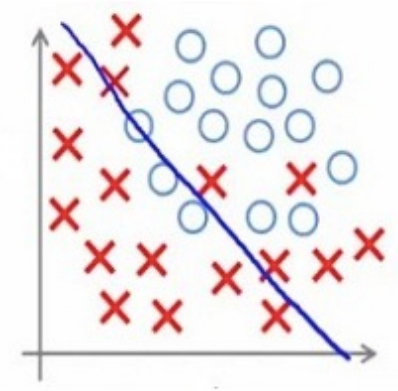
B_k: Batch size
θ: model's weight
L: Loss function
$R(\theta)$: Regularizer with weight λ





Over-fitting





Under-fitting

When the regularization coefficients are too high, some of the weights matrices are nearly equal to zero. Can you explain why?

L2 & L1 Regularization

- The regularization term differs in L1 and L2.
  - https://keras.io/api/layers/regularizers/
- L2 regularization is also known as *weight decay* as it forces the weights to decay towards zero (but not exactly zero).

$$Cost\,function = Loss + \frac{\lambda}{2m} * \sum ||w||^2$$

λ= regularization parameter

- L1 regularization may reduce the weights to zero.

Regularization term

$$Cost\,function = Loss + \frac{\lambda}{2m} * \sum ||w||$$

m = training examples with n features

lambda is the regularization parameter.

Weight Regularization in Keras

- L1: Sum of the absolute weights.
- L2 :Sum of the squared weights.
- L1L2: Sum of the absolute and the squared weights.

A weight regularizer can be added to each hidden layer:

```
keras.regularizers.l1(0.01)
keras.regularizers.l2(0.01)
keras.regularizers.l1_l2(l1=0.01, l2=0.01)
```

Keyword arguments

- kernel_regularizer: Regularizer to apply a penalty on the layer's kernel
- bias_regularizer: Regularizer to apply a penalty on the layer's bias
- activity_regularizer: Regularizer to apply a penalty on the layer's output

https://keras.io/api/layers/regularizers/

Sample code to apply L2 regularization (Directly calling a regularizer)

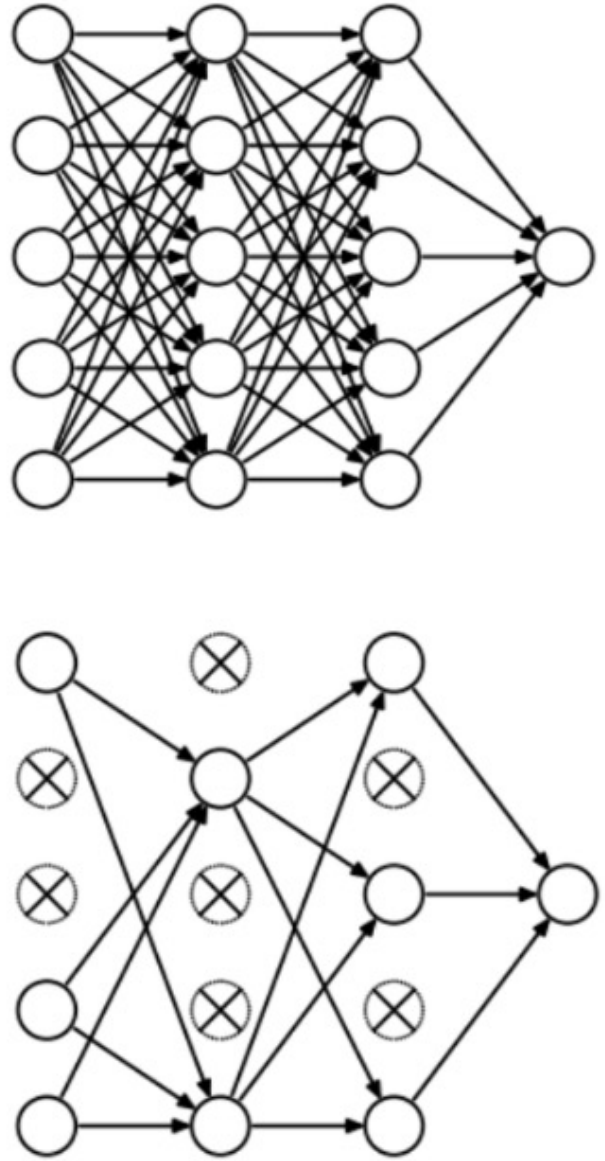
```
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense
from keras.regularizers import l2

def create_model():
    # Create model
    model = Sequential(name=name)
    model.add(Dense(nodes, input_dim=input_dim, activation='relu'))
    model.add(Dense(output_dim=output_dim, activation='relu'),
       kernel_regularizer=regularizers.l2(0.01))
    model.add(Dense(n output_dim=output_dim, activation='relu'),
       kernel_regularizer=regularizers.l2(0.01))
    model.add(Dense(output_dim, activation='sigmoid') )

    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])
    return model
```

Dropout Regularization

- Frequently used in Deep Learning.
- **At every iteration, some nodes are randomly selected and removed along with all of their incoming and outgoing connections.**
- So each round (iteration) has a different set of nodes and this results in a different set of outputs.
- Dropout outperforms a normal network model. In this sense, it can be thought of as *ensemble model*.
- **The probability of choosing the number of nodes to be dropped is the hyperparameter of the dropout function.**
- One can define the probability of dropping to have a value (such as 0.25) and tune it in further for better results using grid search method.
- In keras, one can implement dropout using keras core layer:
  https://keras.io/api/layers/core_layers/#dropout
  https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/



Dropout Regularization in keras used in DNN

```
# dropout in the input layer with weight constraint
def create_model():
        # create model
        model = Sequential()
        model.add(Dropout(0.2, input_shape=(60,)))
        model.add(Dense(60, activation='relu', kernel_constraint=maxnorm(3)))
        model.add(Dense(30, activation='relu', kernel_constraint=maxnorm(3)))
        model.add(Dense(1, activation='sigmoid'))
        # Compile model
        sgd = SGD(lr=0.1, momentum=0.9)
        model.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['accuracy'])
        return model
```