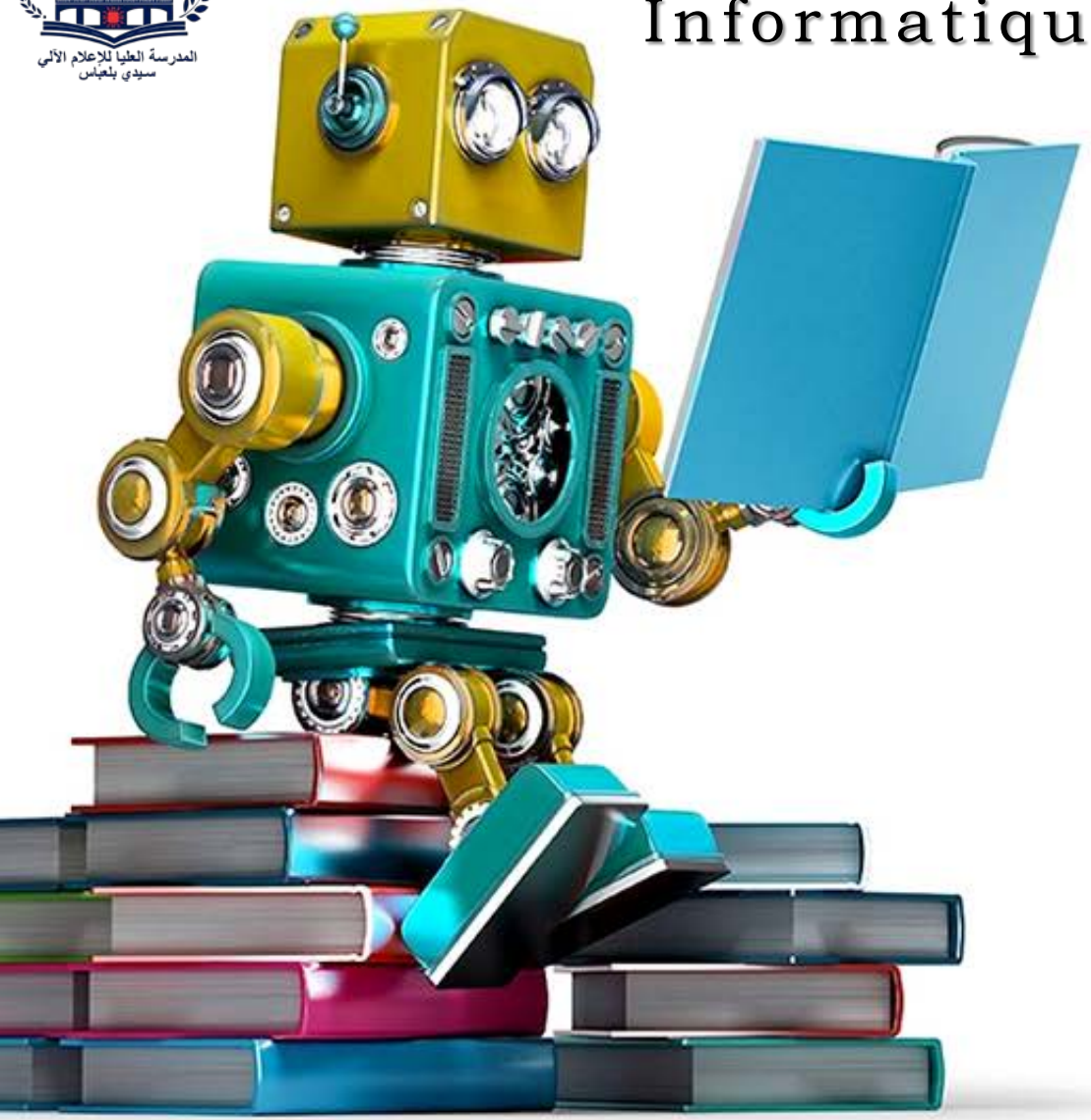
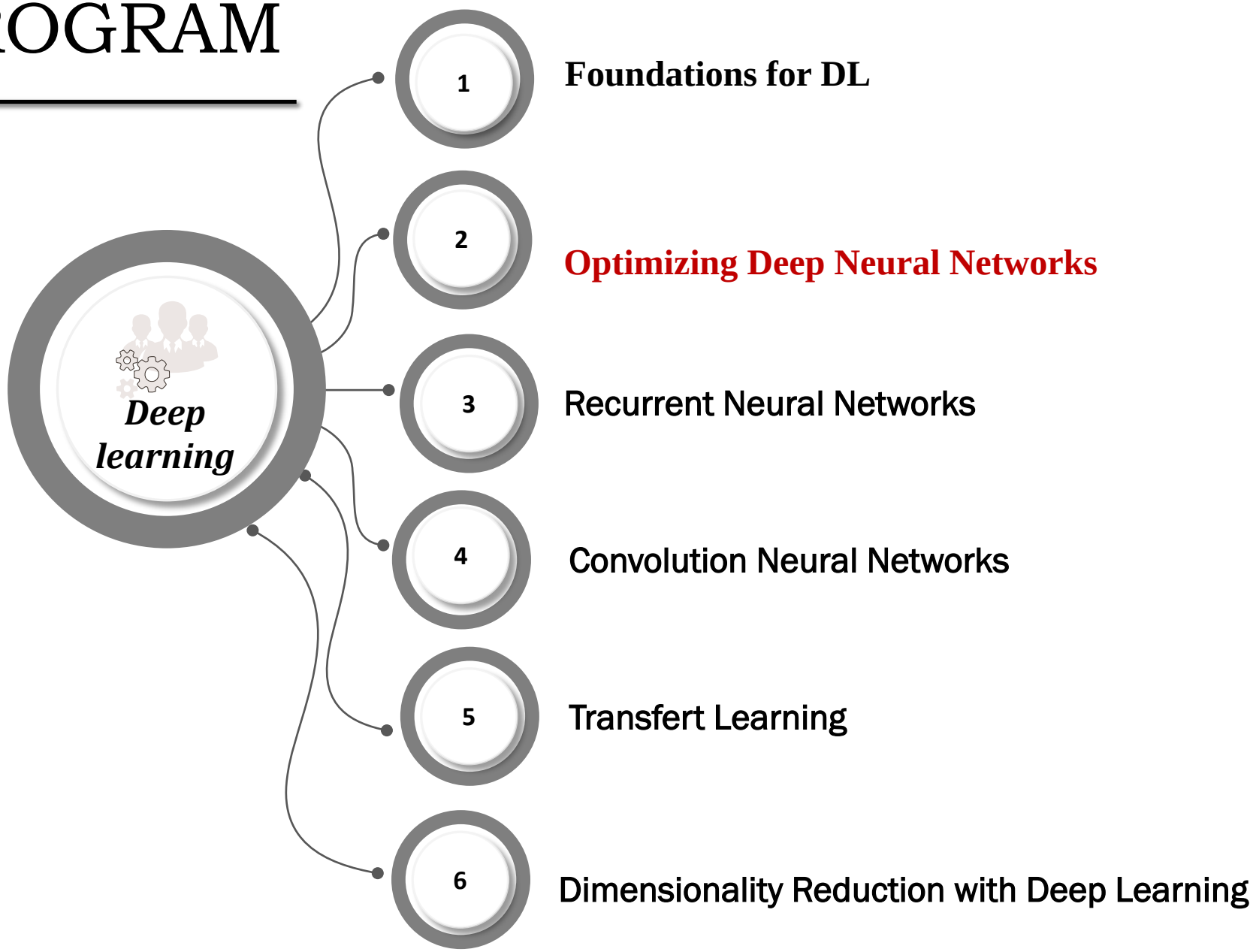
ESI-SBA - École Supérieure en Informatique 08-MAI-1945



# DEEP LEARNING

DR N. DIF

# PROGRAM




1. **Foundations for DL**
2. **Optimizing Deep Neural Networks**
3. Recurrent Neural Networks
4. Convolution Neural Networks
5. Transfert Learning
6. Dimensionality Reduction with Deep Learning

# REMINDER

The difference between Batch an Epoch

**Batch size** ($n_{batch}$) is a hyperparameter of gradient descent that controls the number of training samples to work through before the model's internal parameters are updated.
**Number of epochs** ($n_{epoch}$) is a hyperparameter of gradient descent that controls the number of complete passes through the training dataset.
**Number of iterations :** $n_{epoch} \times \frac{N}{n_{batch}}$ , N is the number of samples. $\frac{N}{n_{batch}}$ is the number of batches in each epoch.

**How to choose the Batch size ?**
**How about small and large values?**
**Why randomizing data?**

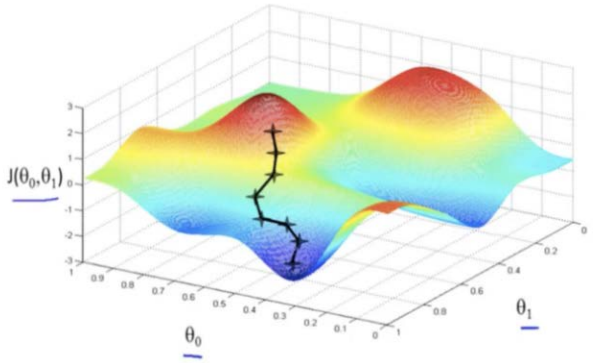
# 1. Optimization Algorithms

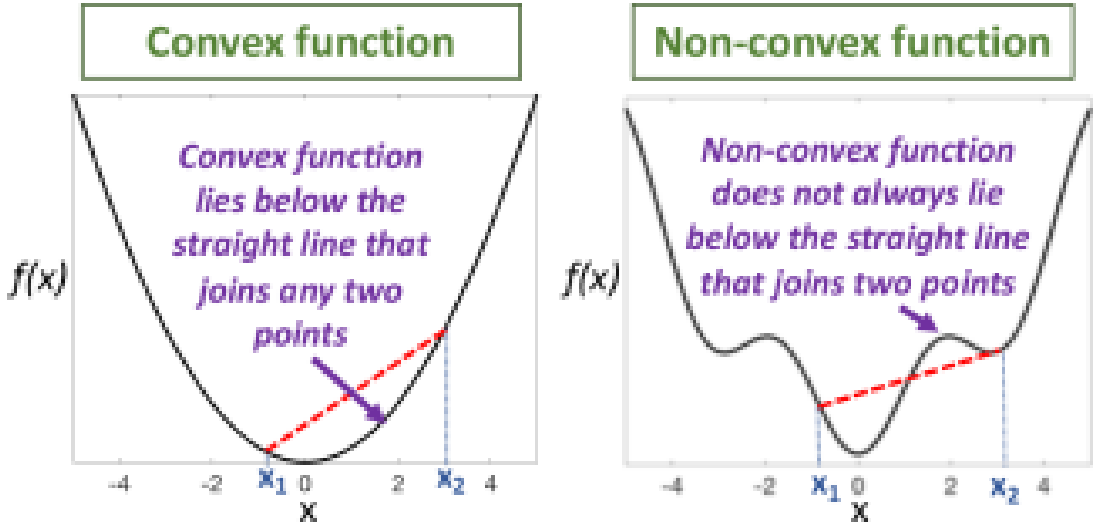
## *1.1.* Gradient descent (SGD)

**Optimization**

- The task of minimizing the cost/loss function *J(w)* parameterized by the model's parameters $W \epsilon R^d$
- Find the global minimum of the objective function. This is feasible if the objective function is **convex**

- Generally, in deep learning, the cost function is non convex. The minimum can be local or global.




Source :https://qiml.radiology.wisc.edu/wpcontent/uploads/sites/760/2021/03/lecture_A1_04_Convexity1.pdf

# 1. Optimization Algorithms

## *1.1.* Gradient descent (SGD)

Gradient descent is an iterative stochastic optimization algorithm that helps to find the minimum of a function.

### Batch Gradient Descent

- Type of gradient descent which processes all the training examples for each iteration of gradient descent ($n_{batch}$ = N).
- Parameters update on all dataset.

### Stochastic Gradient Descent

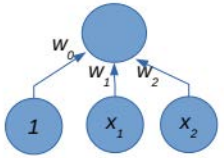
- Type of gradient descent which processes one training example for each iteration of gradient descent ($n_{iterations} = n_{epoch} \times N$ ).
- Parameters update for each training example.

### Mini-Batch Gradient Descent

- Type of gradient descent which processes $n_{batch}$ training example for each iteration of gradient descent ($n_{iterations} = n_{epoch} \times \frac{N}{n_{batch}}$ ).
- Parameters update for each training $n_{batch}$ example.

# 1. Optimization Algorithms

## *1.1.* Gradient descent (SGD) : Exercise




Consider the following, “neural network”, where the activation of the output unit is just the dot product between the input and the weight vector: $h(x) = w^t$x, $w_0 = 0, w_1 = 1, w_2 = 0.5$. We have the following set of example (D.train) :

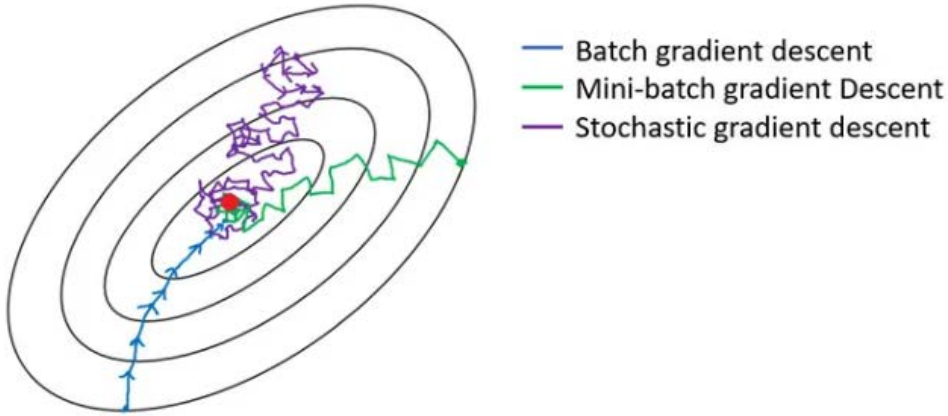
$$X = \begin{bmatrix} 1 & 2 \\ -2 & 5 \\ 0 & 1 \end{bmatrix}, y = \begin{bmatrix} 1 \\ 6 \\ 1 \end{bmatrix}$$

a. What output will this network produce for x = $[2 \quad 3]^T$?
b. Calculate $L(w) = \frac{1}{2}\sum_{(x_i,y_i)\in D}(y_i - w^T x_i)^2$ for this dataset.
c. Perform one round of **batch gradient descent** updates using the data set D.train and η = 0.01.
d. Recalculate the error with the new weights to confirm it went down.
e. Repeat the previous question using one round of stochastic gradient descent. Process the points in the order they appear in the data set. Is the final loss lower or higher in this case?

# 1. Optimization Algorithms

## *1.1.* Gradient descent (SGD)

| | Advantages | Disadvantages |
|---|---|---|
| Batch | Has straight trajectory towards the minimum and it is guaranteed to converge in theory to the global minimum if the loss function is convex.<br>unbiased estimate of gradients | Slow for larger datasets.<br>Memory capacity issues |
| Mini Batch | - Faster than Batch version<br>- Adds noise to the learning process that helps improving generalization error. | Due to the noise, the learning steps have more oscillations and requires adding learning-decay |
| Stoc-astic | Same as mini-batch | - The high number of iterations would increase the run time.<br>- Add more noise compared to mini-batch. |



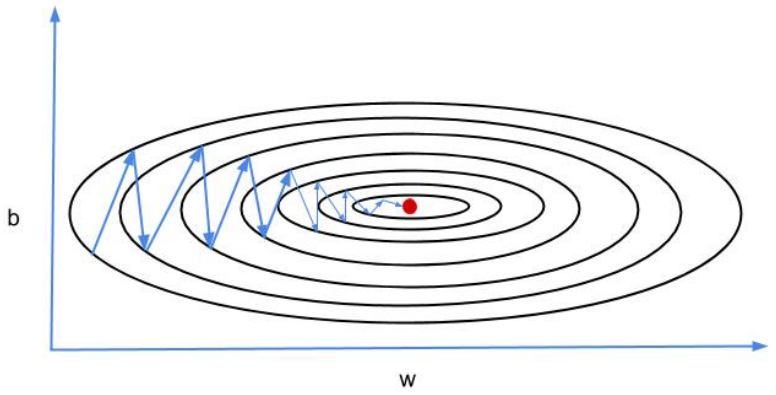

Gradient descent variants' trajectory towards minimum

Source :https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3

# 1. Optimization Algorithms

## *1.1.* Gradient descent (SGD)

During optimization, the search may progress downhill towards the minima, but during this progression, it may move in another direction, even uphill, **depending on the gradient of specific points (sets of parameters)** encountered during the search.




It oscillates too much on vertical axis (as show in above image). This oscillations slows down the convergence to minimum and prevent using large learning rate.

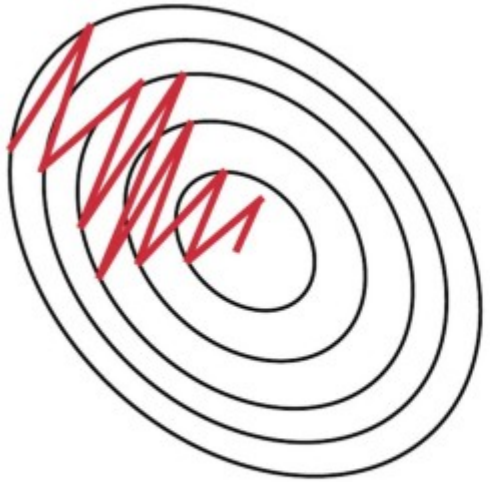
What we want is less oscillation on vertical axis and faster march on horizontal axis towards the minimum. Consequently this would result into faster learning and convergence to the minimum.

# 1. Optimization Algorithms

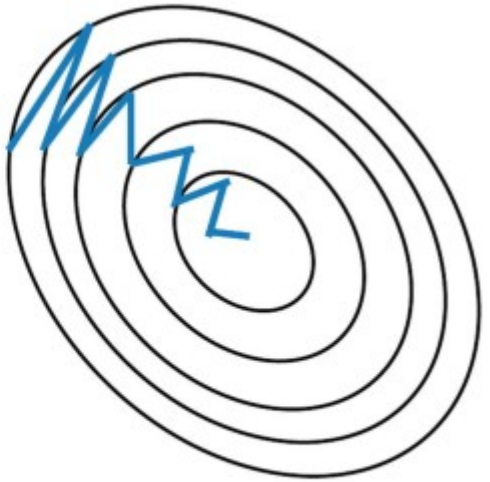
## *1.2.* Gradient descent (SGD) with momentum

**Momentum** is an extension to the gradient descent optimization algorithm that allows the search to build inertia in a direction in the search space and overcome the oscillations of noisy gradients.

- Try to accelerate convergence by damping oscillation using velocity (the speed of "movement" from the previous updates).
- Move in the opposite direction of the gradient and also in the averaged direction of the last few updates.
- The basic idea of Gradient Descent with momentum is to calculate the exponentially weighted average of your gradients and then use that gradient instead to update your weights



Stochastic Gradient Descent **without** Momentum



Stochastic Gradient Descent **with** Momentum

$$W_{t+1} = W_t - \eta\, V_t$$

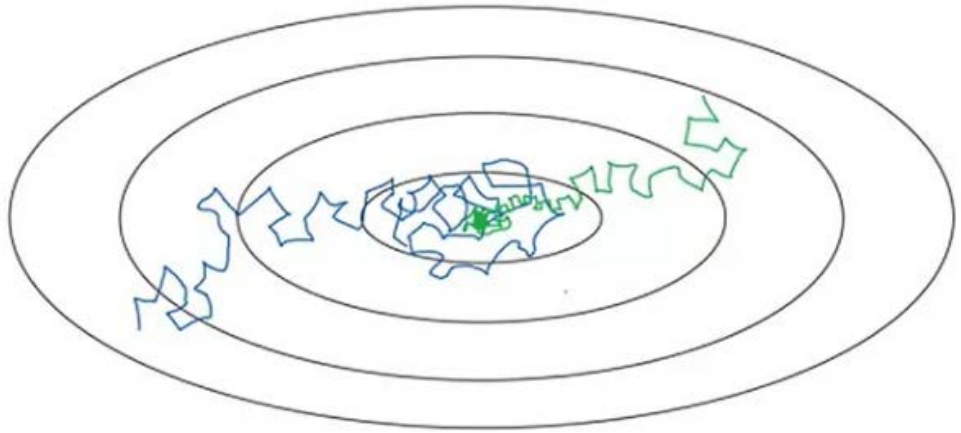
$$V_t = \beta\, V_{t-1} + (1-\beta)\Delta W_t$$

β presents a decaying factor because it is defining the speed of past velocity (**usually set to 0.9**).

# 1. Optimization Algorithms

## *1.3.* Learning rate decay

Learning rate decay is a technique for training modern neural networks. It starts training the network with a large learning rate and then slowly reducing/decaying it until local minima is obtained :**time-based decay**, **Step-based decay,** and **exponential decay**



**Time-based decay**

$$Lr = Lr0 \times \frac{1}{1 + decay\_rate \times epoach\_num}$$

**Keras's implementation**

$$Lr = Lr0 \times \frac{1}{1 + decay\_rate \times iteration\_num}$$

**Step-based decay**

$$Lr = Lr0 \times Drop\_rate^{\frac{Epoach\_num}{Epoach\_drop}}$$

`Drop_rate` specifies the amount that learning rate is modified, and the `epochs_drop` specifies how frequent the modification is.
Lr0 is the initial learning rate.

**Exponential decay**

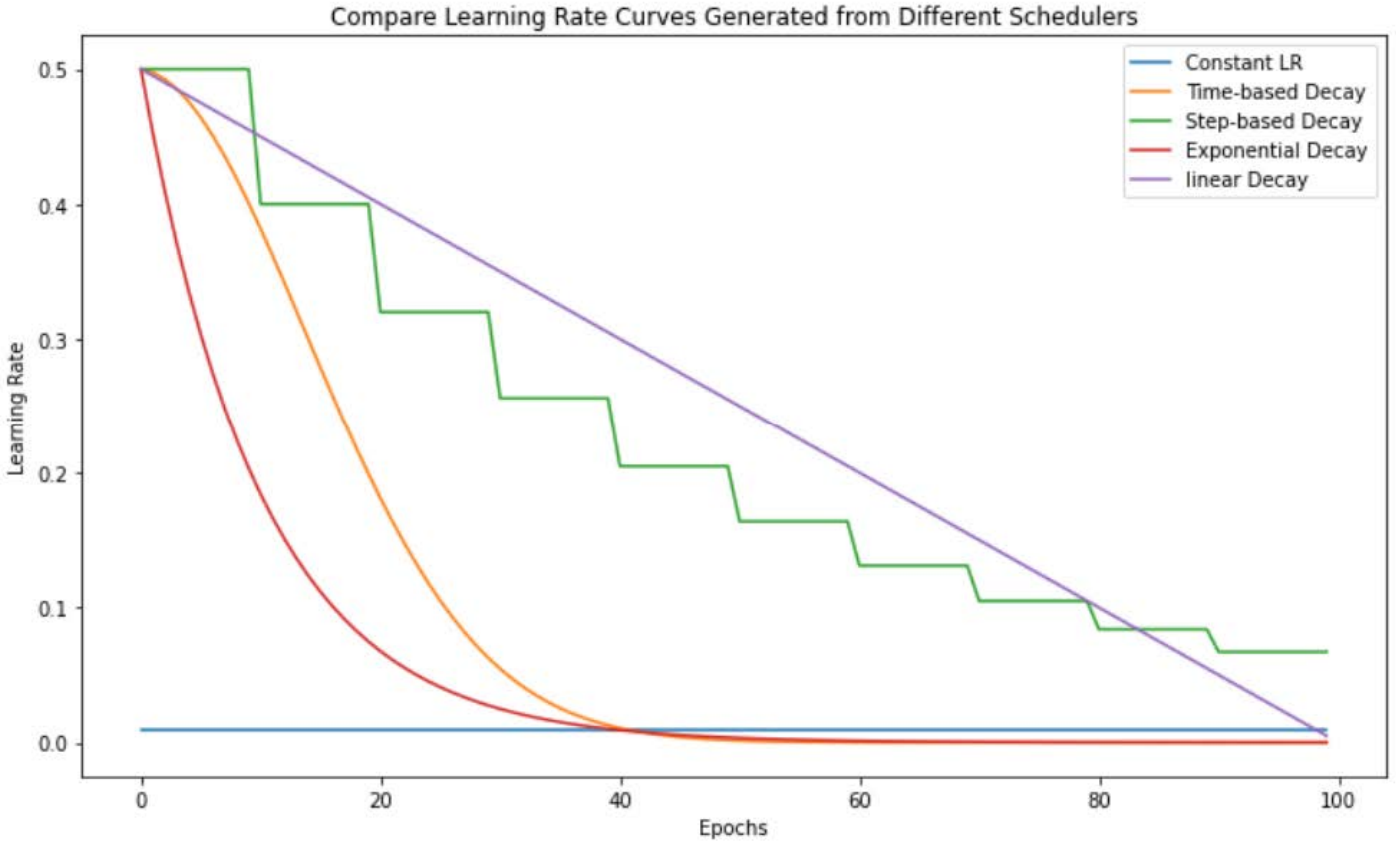
$$Lr = Lr0 \times e^{-kt}$$

K : decay rate
T : epoch number

Source https://medium.com/analytics-vidhya/learning-rate-decay-and-methods-in-deep-learning-2cee564f910b

# 1. Optimization Algorithms

## *1.3.* Learning rate decay




Source https://medium.com/analytics-vidhya/learning-rate-decay-and-methods-in-deep-learning-2cee564f910b

# 1. Optimization Algorithms

## *1.4.* RmsProp Optimizer (Root Mean Squared Propagation)

**-Adaptive gradient descent algorithms** such RMSprop and Adam provide an alternative to classical SGD. These per-parameter learning rate methods provide **heuristic approach** without requiring expensive work in tuning hyperparameters for the learning rate.

- RMSprop deals with oscillation the above issue by using a moving average of squared gradients to normalize the gradient

$$V_t = \beta\, V_{t-1} + (1-\beta)\Delta W_t{}^2$$

$$W_{t+1} = W_t - \eta \frac{\Delta W_t}{\sqrt{V_t+\epsilon}}$$

$\epsilon$ = A small positive constant ($10^{-8}$)

$\beta = 0.999$

# 1. Optimization Algorithms

## *1.5. Adaptative learning rate methods :* Adam Optimizer

-Adam optimizer involves a combination of two gradient descent methodologies: **Momentum** and **RMSPROP**.
- It controls the rate of gradient descent in such a way that there is minimum oscillation when it reaches the global minimum while taking big enough steps (step-size) so as to pass the local minima hurdles along the way.

$$W_{t+1} = W_t - \eta \frac{V_t}{\sqrt{S_t + \epsilon}}$$

$$S_t = \beta_2 \, S_{t-1} + (1 - \beta_2) \Delta W_t{}^2$$

$$V_t = \beta \, V_{t-1} + (1 - \beta) \Delta W_t$$

# 1. Optimization Algorithms

## *1.5. Adaptative learning rate methods :* Adam Optimizer


Comparing Performances of Different Learning Rate Schedules and Adaptive Learning Algorithms

Source https://towardsdatascience.com/

# 2. Overfitting, Regularization and Gradient Checking

## 2.1. Overfitting, Underfitting, and Fitting

### Overfitting

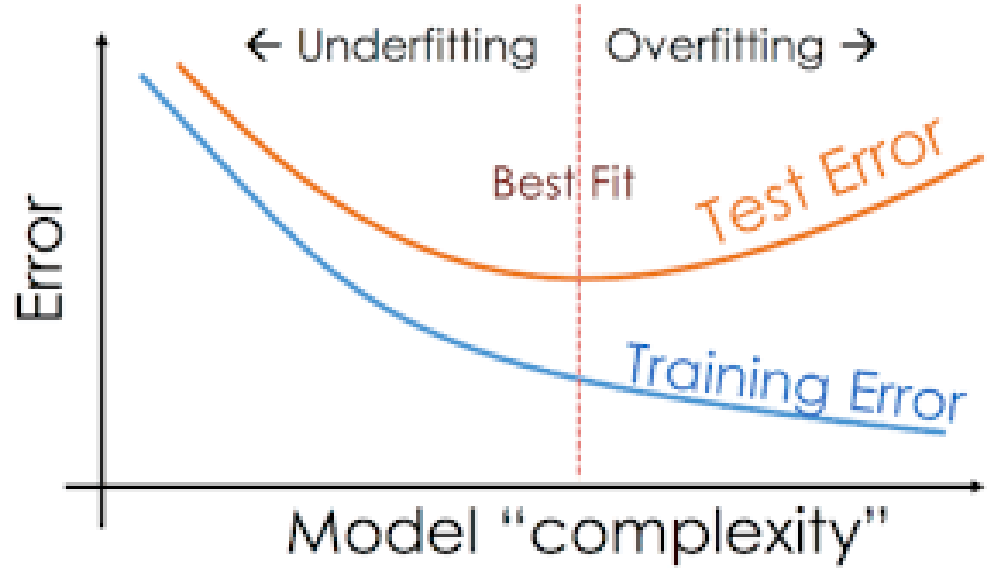
A model that learns the training dataset too well, performing well on the training dataset but does not perform well on the test set.

### Underfitting

A model that fails to sufficiently learn the problem and performs poorly on a training dataset and does not perform well on the test set.

### Fitting

A model that suitably learns the training dataset and generalizes well to the test set.




**How To Prevent Overfitting?**

# 2. Overfitting, Regularization and Gradient Checking

## 2.2. Regularization : L1 and L2 Norm

**Regularization** refers to techniques that are used to calibrate deep learning models in order to minimize the adjusted loss function and prevent overfitting or underfitting.

**Weight decay** works by adding a penalty term to the cost function of a neural network which has the effect of shrinking (decaying) the weights during backpropagation.

### L2 Norm (Weight decay)

Adds the sum of the squared weights to the error term E multiplied by a hyperparameter lambda.

$$C(w,b) = \frac{1}{m}\sum_{i=1}^{n} L(\widehat{y_i}, y_i) + \frac{\lambda}{2m}\sum_{l=1}^{L}\left\|W^{[l]}\right\|^2$$

$$\left\|W^{[l]}\right\|^2 = \sum_{i=1}^{n^{[l+1]}}\sum_{j=1}^{n^{[l]}}\left(w_{ij}^{[l]}\right)^2$$

### L1 Norm

- Adds sum of the weights to the error term E multiplied by a hyperparameter lambda.
- L1 norm can completely eliminate parameters by setting the weights equal to zero.

$$C(w,b) = \frac{1}{m}\sum_{i=1}^{n} L(\widehat{y_i}, y_i) + \frac{\lambda}{2m}\sum_{l=1}^{L}\left\|W^{[l]}\right\|$$

$\lambda$ : Regularization parameter (values : 0.01, 0.02, …,0.4).

**Why Does Shrinking the Weights Help Prevent Overfitting?**

# 2. Overfitting, Regularization and Gradient Checking

## 2.2. Regularization : L1 and L2 Norm

$$C(w,b) = \frac{1}{n}\sum_{i=1}^{n} L(\widehat{y_i}, y_i) + \frac{\lambda}{2m}\sum_{l=1}^{L} \left\| W^{[l]} \right\|^2$$

IF we have one layer

$$\frac{\partial C}{\partial w} = \frac{\partial E}{\partial w} + \frac{\partial}{\partial w}\left(\frac{\lambda}{2m}\|W\|^2\right)$$

$$W = W - \alpha\left[\frac{\partial J}{\partial w} + \frac{\lambda}{m}W\right] = W - \frac{\alpha\lambda}{m}W - \alpha\frac{\partial J}{\partial w}$$

Shrinking the weights has the practical effect of effectively deactivating some of the neurons by shrinking their weights close to zero. The larger you set the regularization parameter lambda, the smaller the weights will become.

When using L2 regularization, the neurons technically are still active, but their impact on the overall learning process will be very small.

# 2. Overfitting, Regularization and Gradient Checking

## 2.2. Regularization : Dropout

**Dropout** works by randomly disabling neurons and their corresponding connections. This prevents the network from relying too much on single neurons and forces all neurons to learn to generalize better.

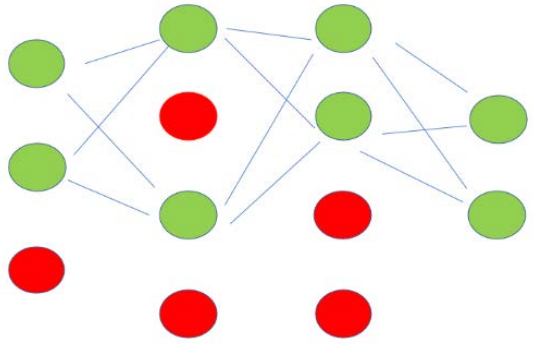
### Why dropout works?

- DNN are prone to overfitting due to the large number of parameters.
- Reducing parameters by removing different neurons helps to reduce overfitting and improve generalization.
- Removing different neurons on every pass through the network generate different architectures that their results is averaged➔ presents alternative to ensemble learning techniques

### Is Dropout Used During Testing?

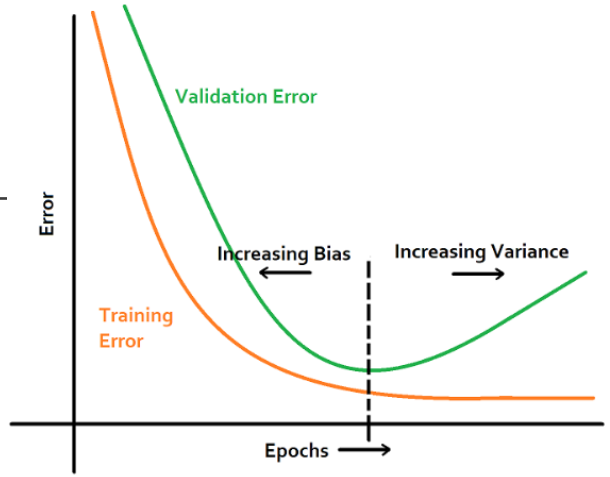
### How Dropout Works ?

Set a dropout rate  for each layer (generally last hidden layers).
According to the speciated dropout rate, in each pass, n neurons will be disactivated randomly.



# 2. Overfitting, Regularization and Gradient Checking

## 2.2. Regularization : Early stopping




- When training a large network, there will be a point during training when the model will **stop generalizing** and start learning the **statistical noise** in the training dataset. This result in an increase in generalization error ➔ **Use early stopping.**
- There are three elements to using early stopping:

### ❶ Monitoring model performance

- The choice of a **dataset** (validation data) and a **metric** (loss or accuracy) to evaluate the model.
- Evaluating the model on the validation set at the end of each epoch adds an **additional computational cost** during training.

### ❷ Trigger to stop

- Comparing the performance on the validation dataset at the prior and current training epochs, and stop training if the loss increase.
- If validation loss go up and down many times ➔ A decrease in performance is observed over a given number of epochs.

### ❸ The choice of model to use

Save the model weights if the performance of the model on a holdout dataset is better than at the previous epoch.

# 2. Overfitting, Regularization and Gradient Checking

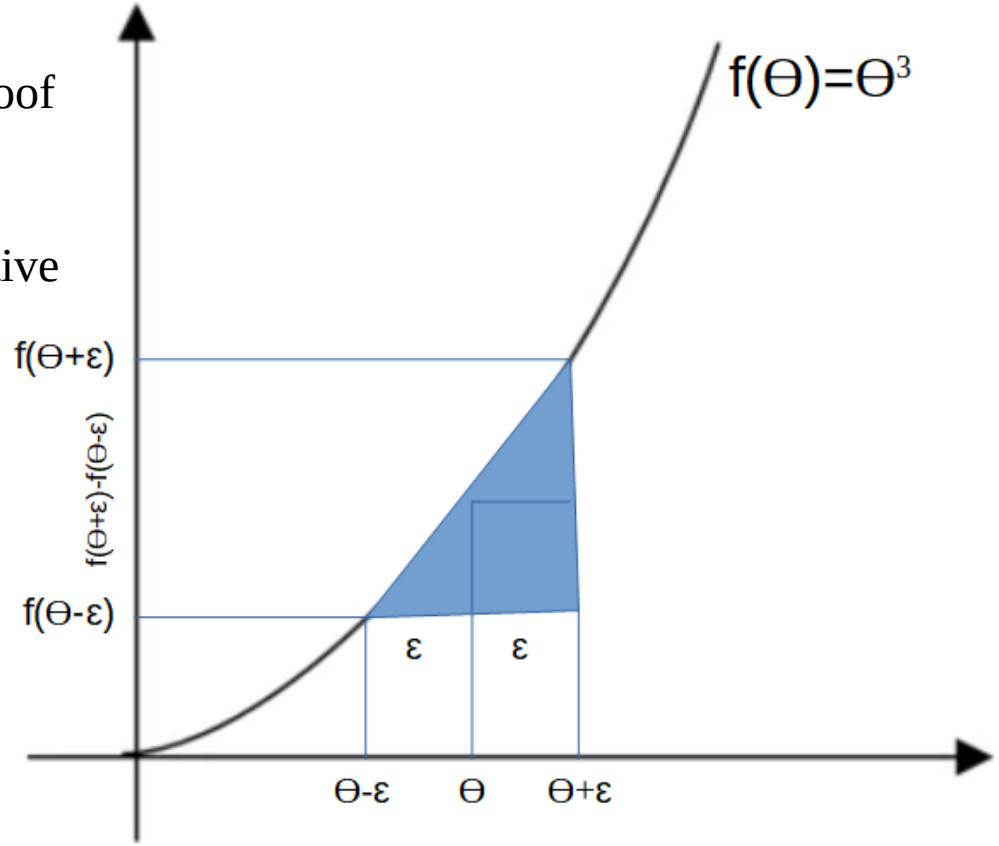
## 2.3. Gradient Checking

Backpropagation is quite challenging to implement, and sometimes has bugs Gradient checking is used to give a proof that the backpropagation is actually working.

For each parameter $\theta_i$, numerically approximate the derivative as follows :

$$\frac{dJ}{d\theta} \approx \frac{dJ}{d\theta}(aproxi) = \frac{J(\theta+\epsilon)-J(\theta-\epsilon)}{2\epsilon}$$

In practice, $\epsilon$ is set to a small constant around $10^{-4}$



# 2. Overfitting, Regularization and Gradient Checking

## 2.4. Regularization : Batch Normalization

**Normalization** is a data pre-processing tool used to bring the numerical data to a common scale without distorting its shape.

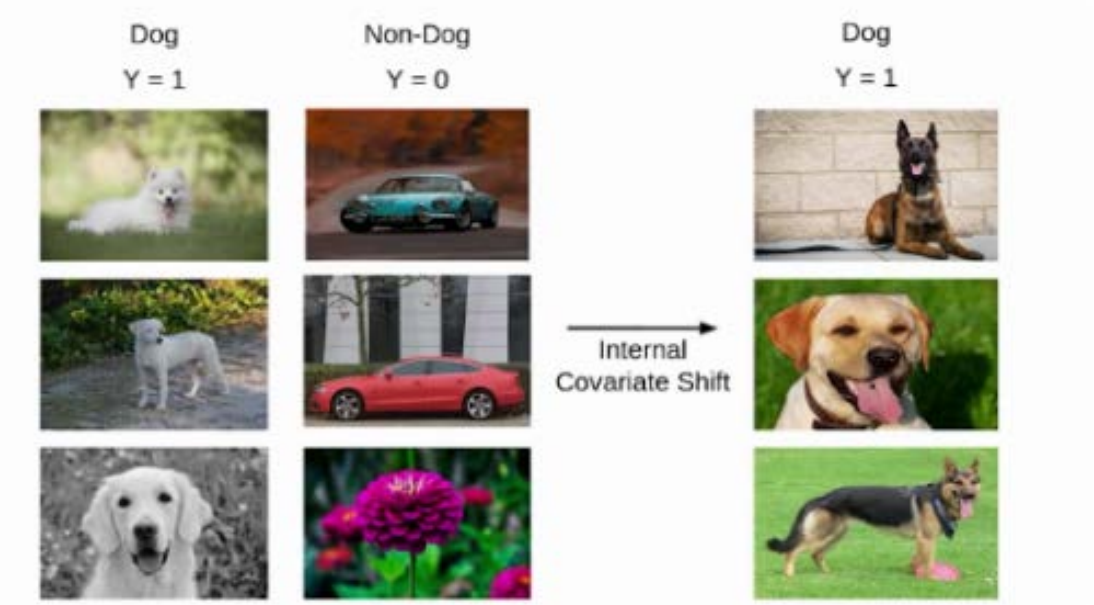
**Batch normalization** extra layers in a deep neural network. The new layer performs the standardizing and normalizing operations on the input of a layer coming from a previous layer.
For the input layer, batch normalization takes place in **batches**, not as a single input to make neural networks faster and more stable.

### Advantages of Batch Normalization

- Speed Up the Training
- Handles internal covariate shift

# NOTE : INTERNAL COVARIATE SHIFT

If we get a new set of images, consisting of non-white dogs. These new images will have a slightly different distribution from the previous images. Now the model will change its parameters according to these new images. Hence the distribution of the hidden activation will also change. This change in hidden activation is known as an internal covariate shift.



# 2. Overfitting, Regularization and Gradient Checking

## 2.4. Regularization : Batch Normalization

**Input:** Values of $x$ over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: $\gamma, \beta$
**Output:** $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

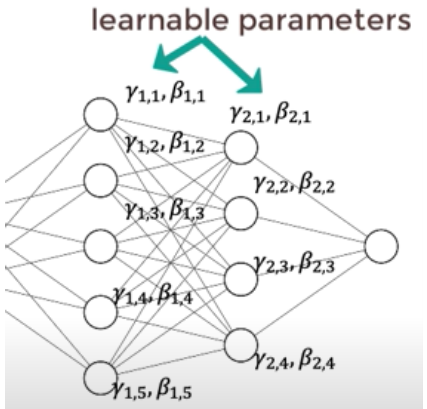
$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

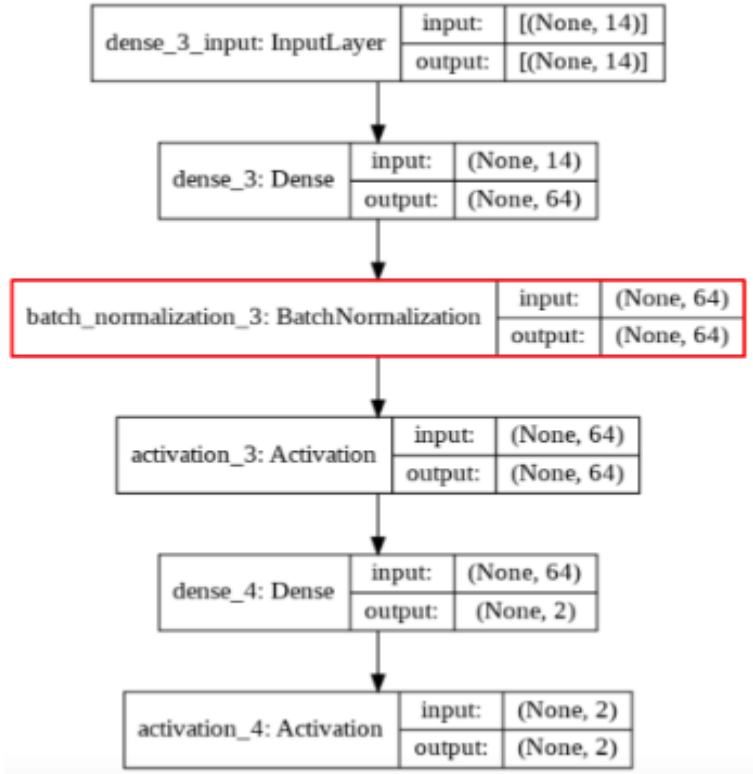
**Algorithm 1:** Batch Normalizing Transform, applied to activation $x$ over a mini-batch.

Source [1]

Mean and variance and dependent on the samples of the batch, we calibrate them by introducing two learnable parameters for each neuron

β and γ are learnable parameters that are updated during training based on SGD.




A DNN with a batch normalization layer.

# 3. Hyperparameter Tuning

## 3.1. Hyperparameter Tuning

Deep learning models are full of hyper-parameters and finding the best configuration for these parameters in such a high dimensional space is not a trivial challenge. to provide best result, we need to find the optimal value of these **hyper-parameters.**

### ❶ Hyperparameters related to Network structure

- Number of Hidden Layers and units.
- Dropout rate (0.2, 0.4, 0.5).
- Network Weight Initialization (Mostly uniform distribution).
- Activation function.

### ❷ Hyperparameters related to Training Algorithm

- Learning Rate (0.1, 0.01, 0.001.
- Momentum (decaying factor).
- Number of epochs (20, 50, 100).
- Batch size ( 32, 64, 128, 256,…).
- Optimizer (Mini batch SGD, with momentum, RmsProp, Adam).

### Methods used to find out Hyperparameters

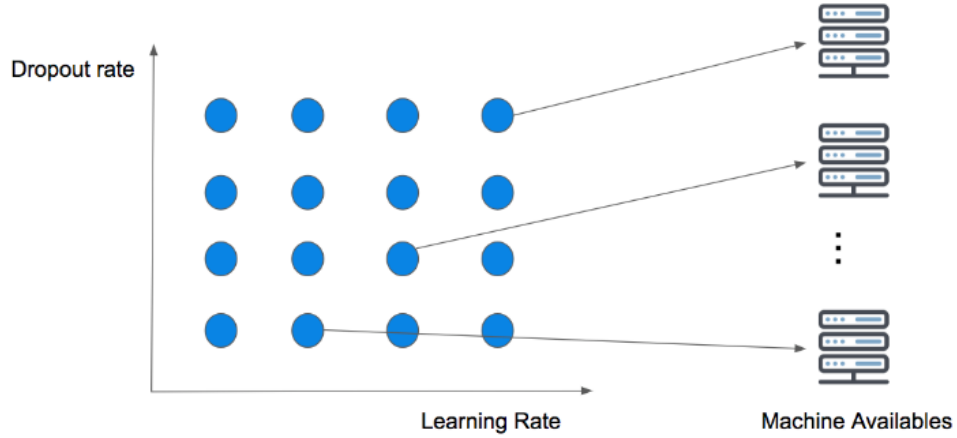
- Manual Search (Babysitting).
- Grid Search.
- Random Search.
- Bayesian Optimization.

(Model Design + Hyperparameters) → Model Parameters

# 3. Hyperparameter Tuning

## 3.2. Hyperparameter Tuning : Grid Search

1. Define a grid on *n* dimensions, where each of these maps for an hyperparameter. e.g. *n* = (learning_rate, dropout_rate, batch_size)
2. For each dimension, define the range of possible values: e.g. batch_size = [4, 8, 16, 32, 64, 128, 256]
3. Search for all the possible configurations and wait for the results to establish the best one: e.g. *C1* = (0.1, 0.3, 4) -> acc = 92%, *C2* = (0.1, 0.35, 4) -> acc = 92.3%, etc..




**Disadvantages**

- Curse of dimensionality.
- Computationally expensive.

**Advantages**

- Guarantee to find the best hyper-parameters.

# 3. Hyperparameter Tuning

## 3.2. Hyperparameter Tuning : Random Search

The only real difference between Grid Search and Random Search is on the *step 1* of the strategy cycle – Random Search picks the point randomly from the configuration space

**Disadvantages**

- It does not guarantee to find the best hyper-parameters.

**Advantages**

- Good on high spaces.
- Give good results in less iterations.

Unfortunately, both Grid and Random Search share the common downside: ***"Each new guess is independent from the previous run!***

**Try Bayesian Optimization**

# REFERENCES

[1] Ioffe, S., & Szegedy, C. (2015, June). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning* (pp. 448-456). pmlr.