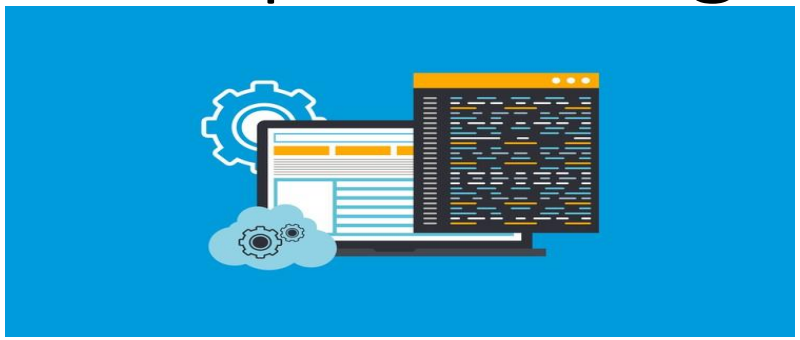
Deep Learning by TensorFlow 2.0 : Basic to Advance with Python



Training Content (Hands on marked in green color)

Part 1: Foundation

Basic

i. Installation and Folder structure
ii. Why Deep Learning is emerging
iii. What is Wide & Deep Learning
iv. Deep Learning Package
v. What is TensorFlow
vi. Deep Learning: Working steps
vii. Important terms of Deep Learning
viii. Activation functions
ix. Comparison of optimizers

i. Basics of TensorFlow
ii. Rank of tensors
iii. Various Reduction's operation
iv. Few important operations
v. TensorFlow inbuilt constants
vi. Tensor segmentation
vii. Various sequence utilities
viii. Tensor slicing and joining

Part 2: Regression and Classifications

i. Regression with Premade Estimators
ii. Tensor Board
iii. Regression by using Keras (tf.keras)
iv. Classifications using Premade Estimators
v. Multiclass classification using Keras

Hands on marked in green color

Training Content (Hands on marked in green color)

Part 3: CNN

i. How to Teach Machines
ii. What is Convolutional Neural Networks (CNN)
iii. CNN: Showcase of multiplications
iv. CNN details
    i. Single Array (after flattening)
    ii. Image dimension
    iii. Various Strides
    iv. Padding applications
    v. Pooling applications
    vi. Dimensions of the layers and flow

- Text Classification Using CNN (https://medium.com/jatana/report-on-text-classification-using-cnn-rnn-han-f0e887214d5f)

Part 4: RNN and LSTM

i. RNN applications
ii. What is Recurrent Neural Networks
iii. Why RNN
iv. The Vanishing and Exploding Gradient
v. The Long Short Term Memory (LSTM)
vi. LSTM and GRU Architecture
vii. LSTM vs GRU
viii. Which models to choose
ix. Library of RNN
x. Text Classification Using RNN (https://medium.com/jatana/report-on-text-classification-using-cnn-rnn-han-f0e887214d5f)
xi. Language modelling using RNN (https://medium.com/praemineo/language-modelling-using-recurrent-neural-networks-part-2-c13f5e07b6e)

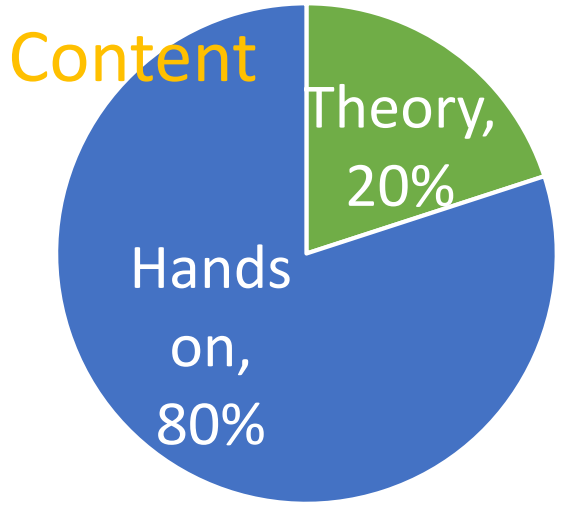
Prerequisite

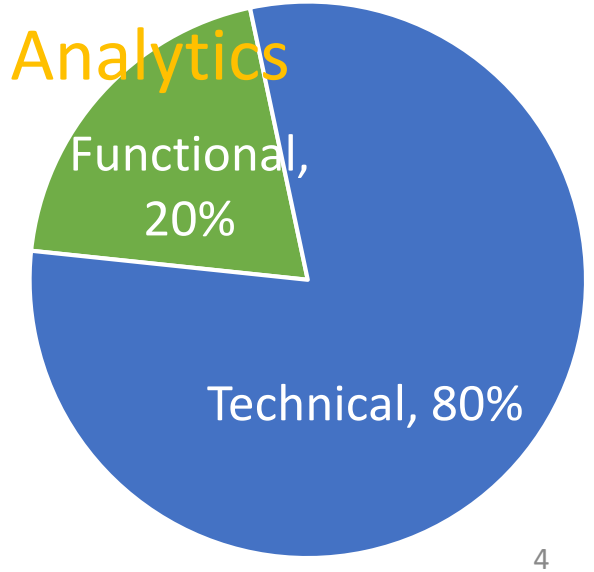
- Expert in Machine Learning Concepts
- Expert in Python 3.6.8 or above

https://www.udemy.com/course/data-scientist-for-ai-and-machine-learning-using-python/?referralCode=1B0FC13B7511890C9B3F

- Aware of Spyder (Anaconda 3 (64-bit))
- Programming awareness
- Participant count : Max 15 per batch
- Lab details will be shared separately

Methodology





Installations , Technology , Folder structure and 1.x vs 2.x

- software_list.txt – Please follow the steps

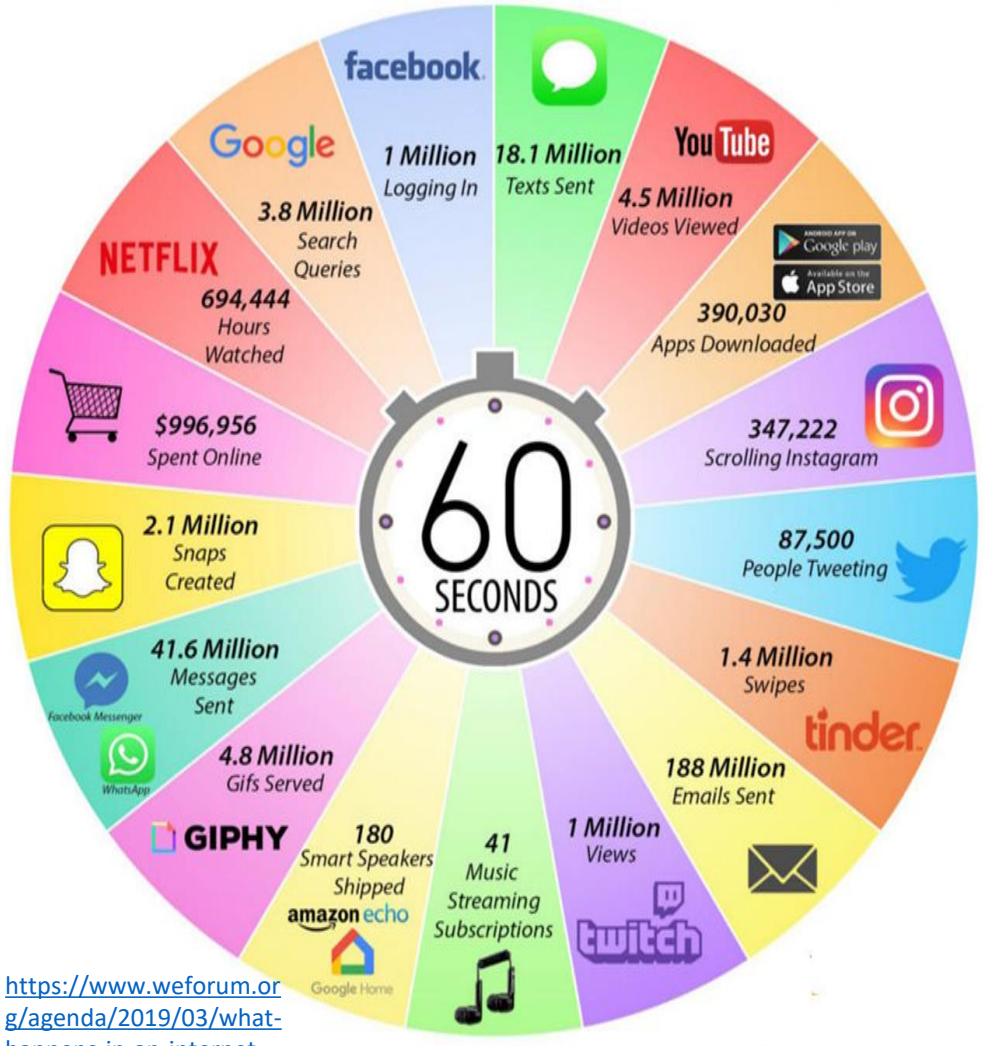
- Anaconda (https://www.anaconda.com/download/) or any Python IDE

- Presentations and Code are uploaded in section (1 or 2)

- TensorFlow 1.x vs 2.x

Why Deep Learning is emerging

DATA



https://www.weforum.org/agenda/2019/03/what-happens-in-an-internet-minute-in-2019

ALGORITHMS

- Logistic regression — 1958
- Hidden Markov Model — 1960
- Stochastic gradient descent — 1960
- Support Vector Machine — 1963
- k-nearest neighbors — 1967
- Artificial Neural Networks — 1975
- Expectation Maximization — 1977
- Decision tree — 1986
- Q-learning — 1989
- Random forest — 1995
- Deep Learning - 2006

INFRASTRUCTURE

Unstructured data is the information that doesn't reside in a traditional relational database

What is Wide & Deep Learning

- DL is a ML technique
- Teaches machines to do work like humans
- Learn by example.

- Usages of DL
- Driverless cars
- Object recognition
- Speech recognition
- ……..

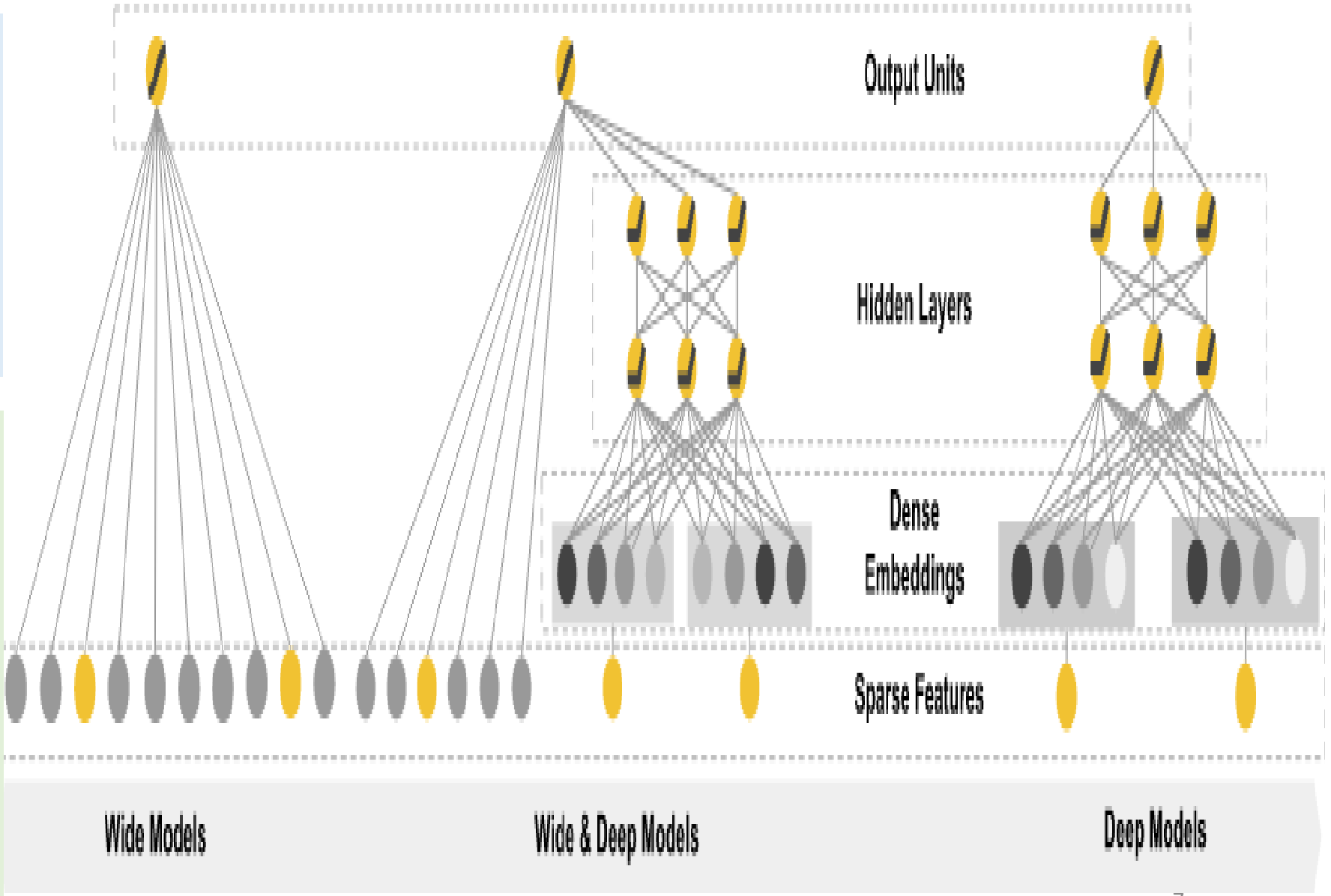


Image source link is at reference section

Deep Learning Package

- Tensorflow
- Theano (academic project at Université de Montréal)
- Keras
- Mxnet
- Torch
- Caffe
- CuDNN
- Etc.

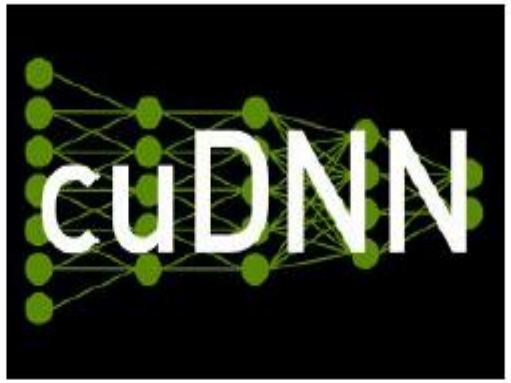


Image source link is at reference section

What is TensorFlow

- “An open-source software library for Machine Intelligence”
- Google
- Helps in Deep learning models
- A Python library (Stable and widely used)

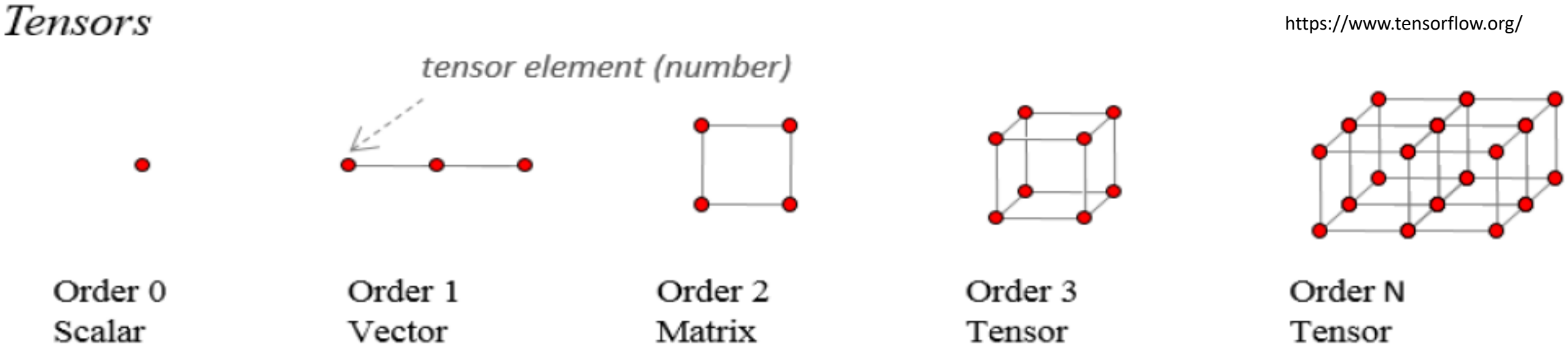
https://www.tensorflow.org/

- A highly efficient C++ backend to do its computation
- R interface to TensorFlow using the high-level Keras and Estimator APIs

What is Tensor

- A tensor consists of a set of primitive values shaped into an array of any number of dimensions (known as tensor's rank).
- Here are some examples of tensors:

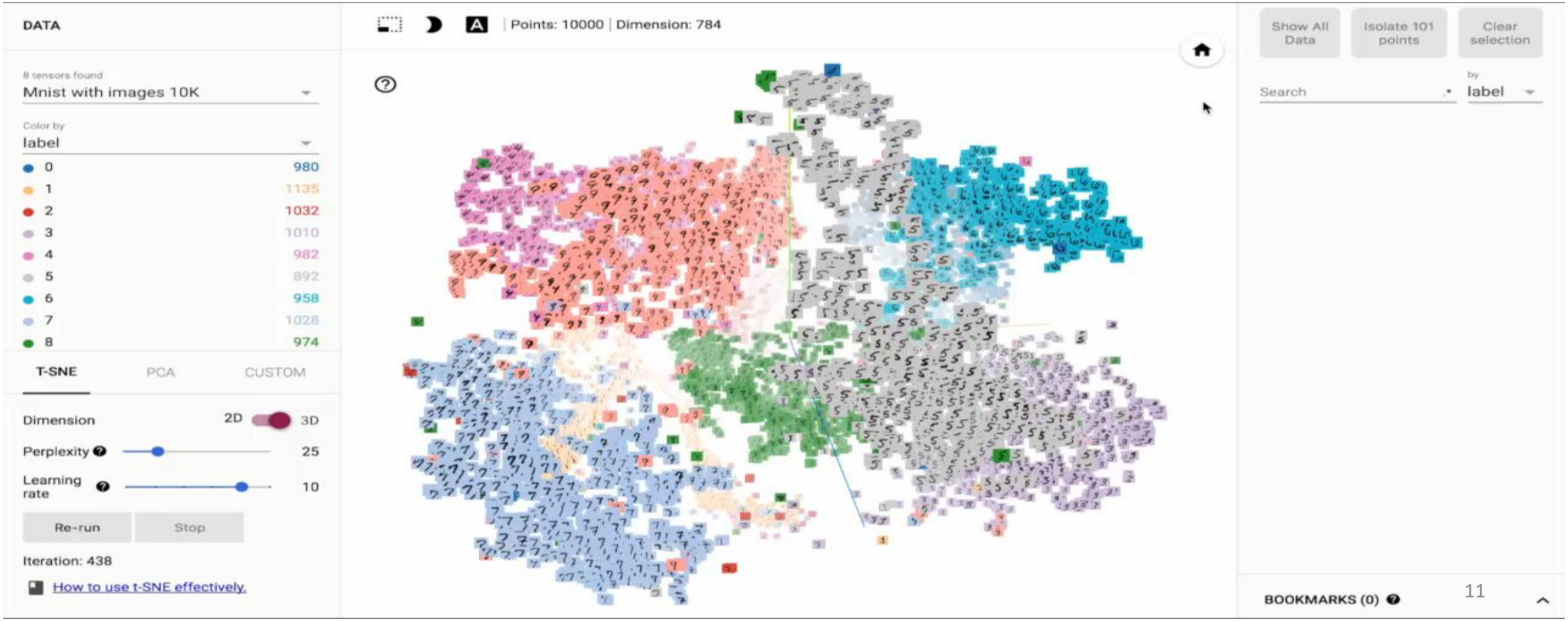
https://www.tensorflow.org/



- ✓ 3 # a rank 0 tensor; this is a scalar with shape []
- ✓ [1. ,2., 3.] # a rank 1 tensor; this is a vector with shape [3]
- ✓ [[1., 2., 3.], [4., 5., 6.]] # a rank 2 tensor; a matrix with shape [2, 3]
- ✓ [[[1., 2., 3.]], [[7., 8., 9.]]] # a rank 3 tensor with shape [2, 1, 3]

TensorBoard: Visualizing Learning

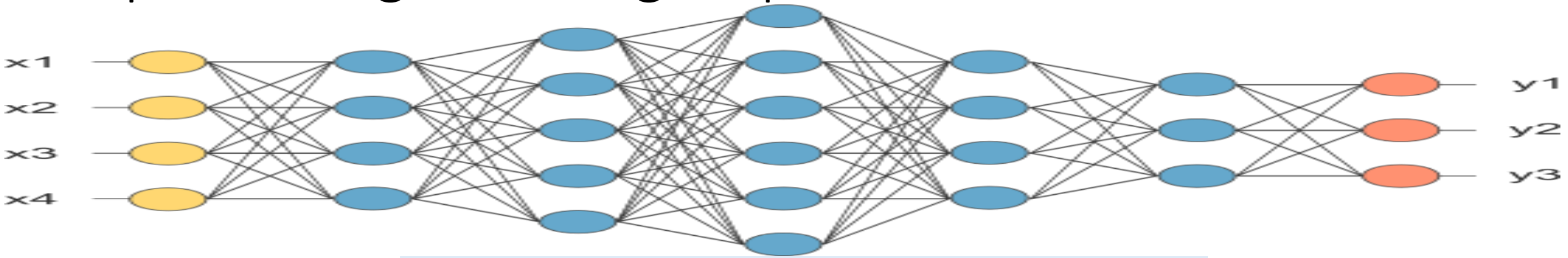
- It is visualization tools to understand, debug, and optimize TF programs
- To visualize your TensorFlow graph
- Plot quantitative metrics about the execution of graph



TensorFlow : Two major ways (keras, Estimators)

- A high-level TensorFlow library that simplifies the machine learning as following:
  - running training loops
  - running evaluation loops
  - managing data sets
  - managing feeding
- It defines many common models.
- Premade Estimators
  - LinearClassifier and LinearRegressor (Also known as wide)
  - DNNClassifier and DNNRegressor (Also known as deep)
  - DNNLinearCombinedClassifier and DNNLinearCombinedRegressor (Also known as wide and deep)
- Internal TensorFlow Keras is an API specification available as tf.keras

Deep Learning: Working steps



DNNs learn a hierarchy of features.

The lower-level features typically catch local patterns and generic features. These patterns are very sensitive to changes in the raw feature.

The higher-level features are more specific task oriented, abstract and invariant to the variations in the raw feature

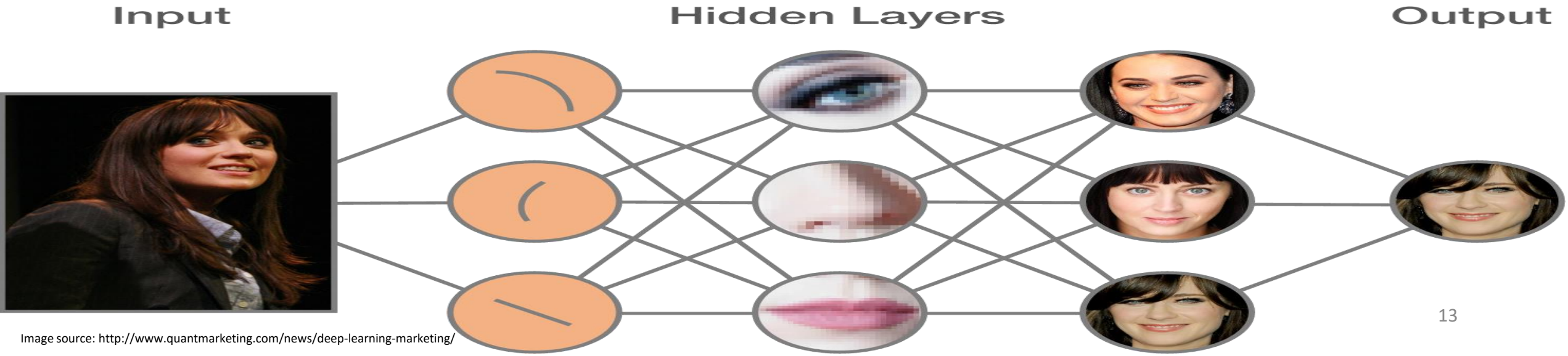
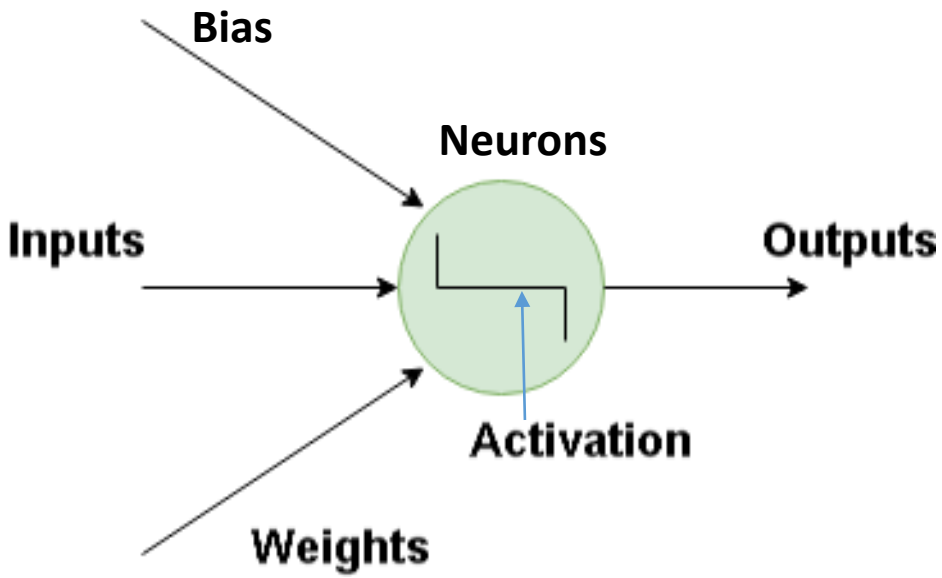
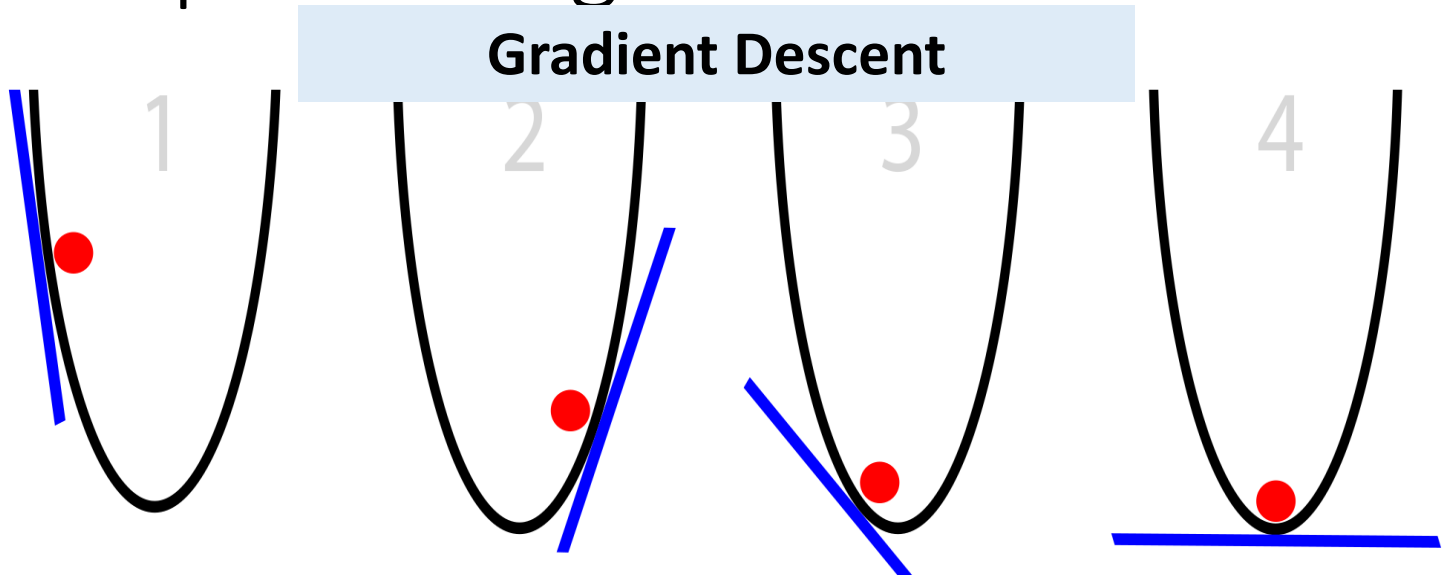
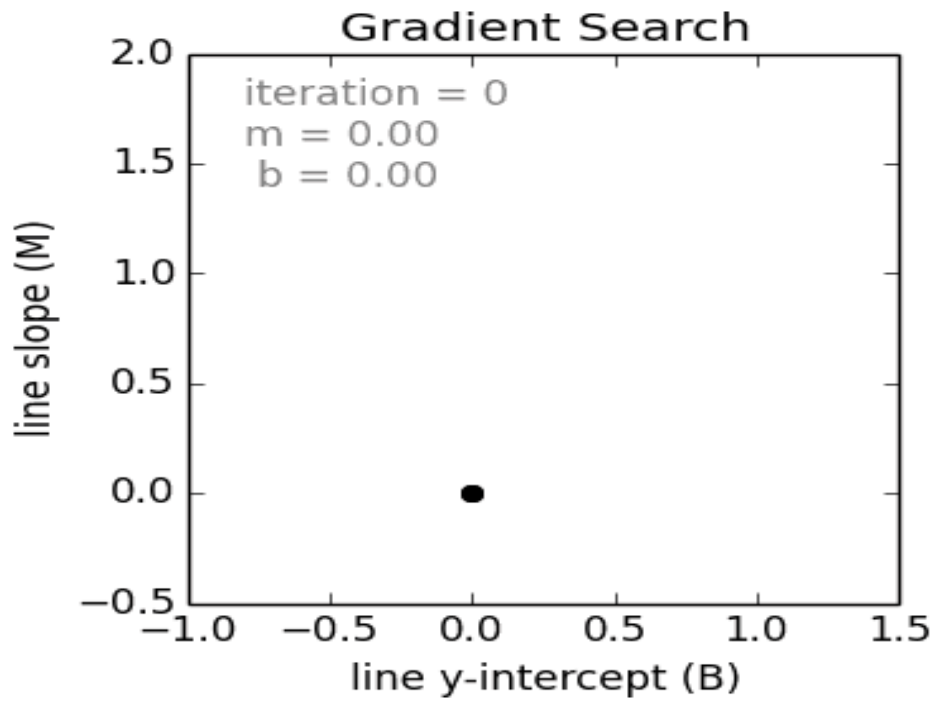


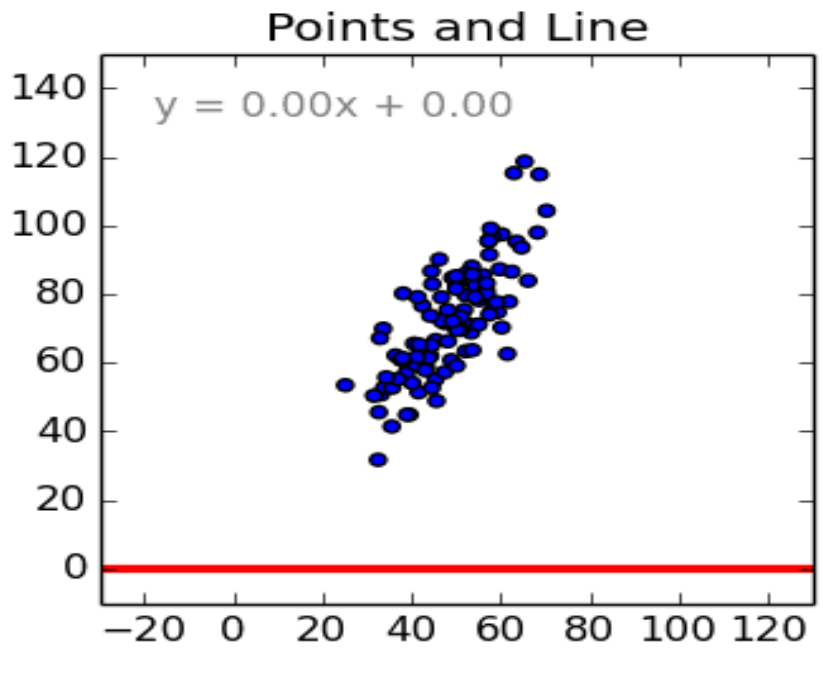
Image source: http://www.quantmarketing.com/news/deep-learning-marketing/

Few Important terms of Deep Learning





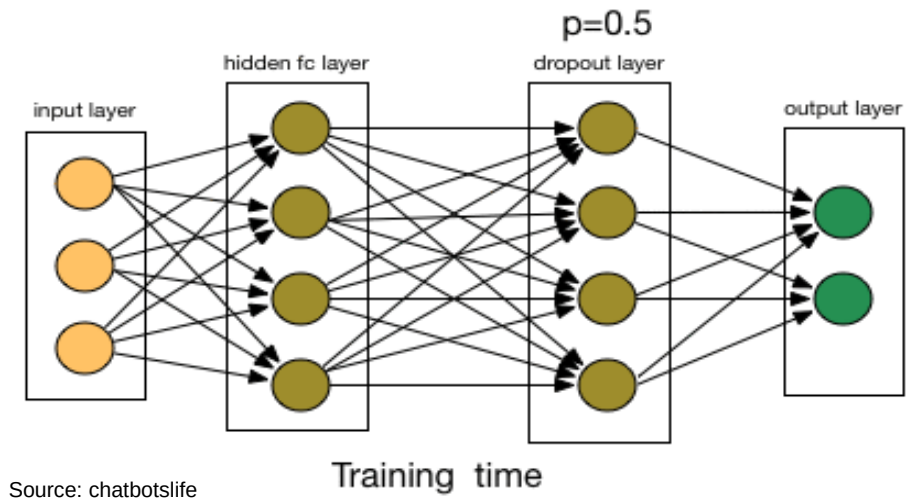




- Loss function
- Epochs
- Batch

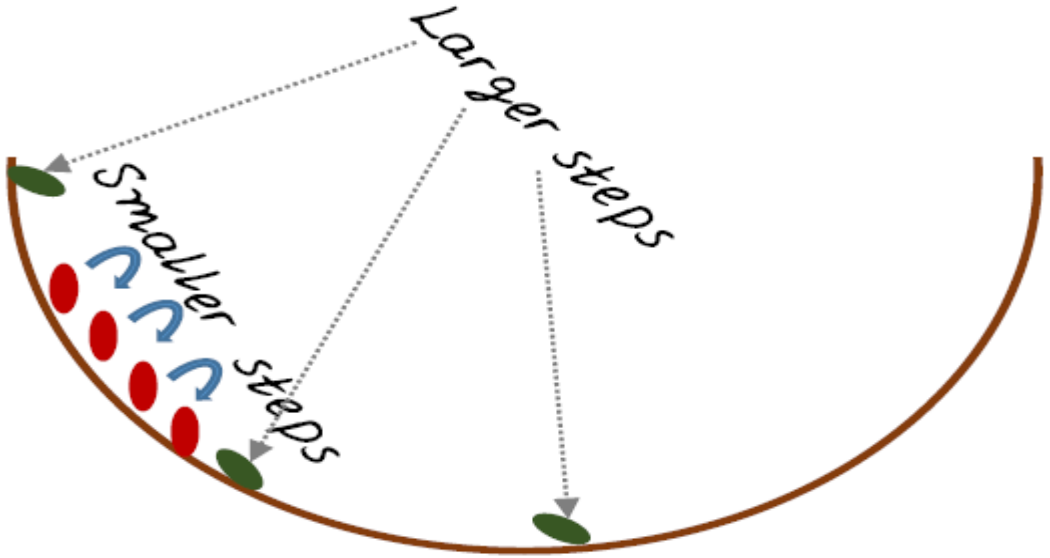
Few Important terms of Deep Learning cont

Dropout

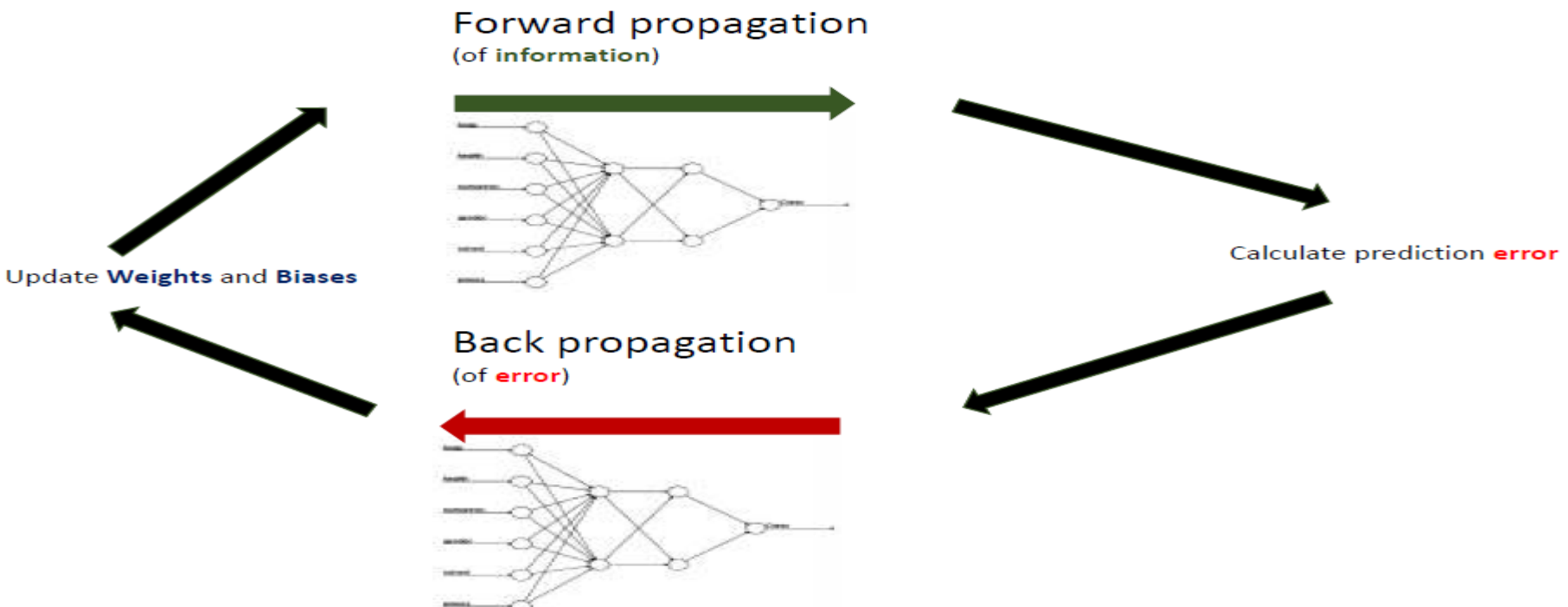


Source: chatbotslife

Learning Rate

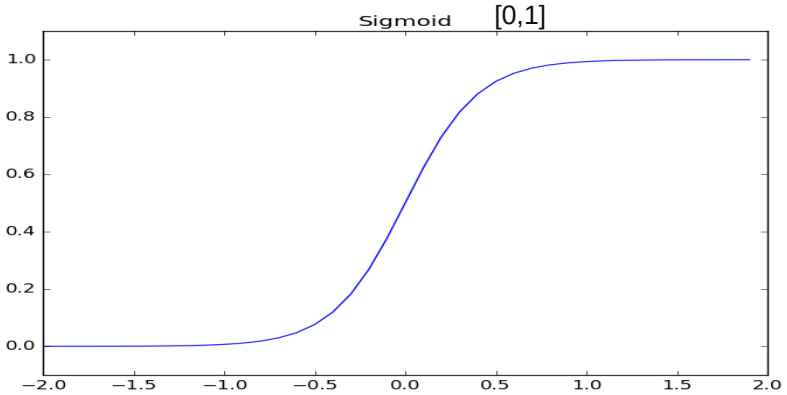


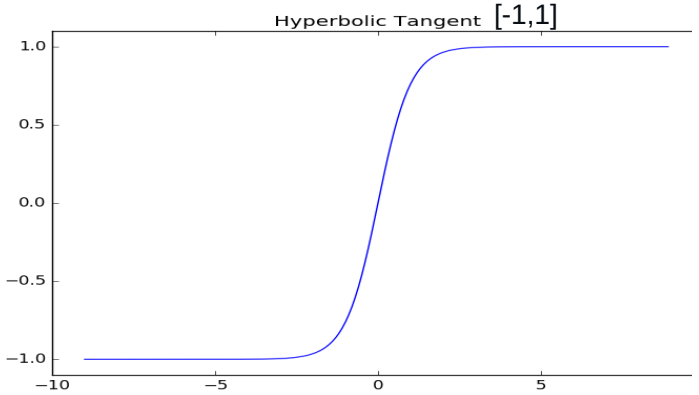
Back Propagation

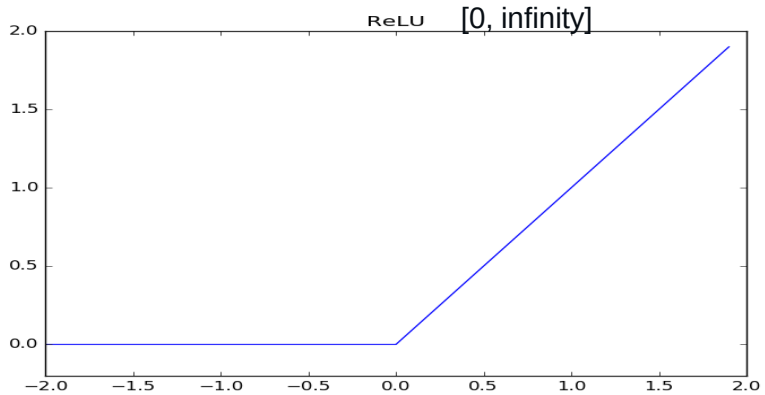


Activation functions

https://en.wikipedia.org/wiki/Activation_function http://playground.tensorflow.org/





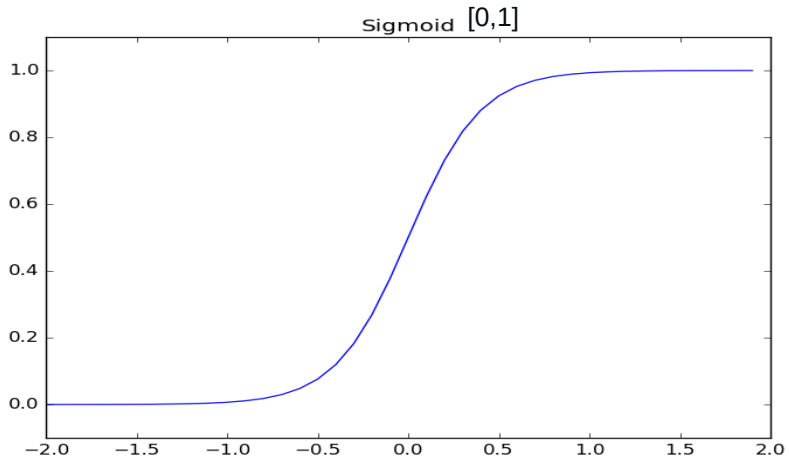


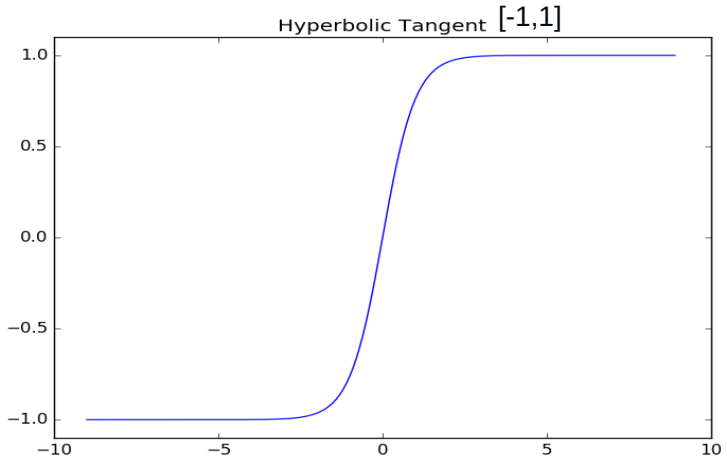
- Softmax or sigmoid has the advantage of "no blowing up activation", because their output is limited to < 1.
- Suffers from the gradient vanishing problem
- Slow convergence rate
- Not a zero-centric function

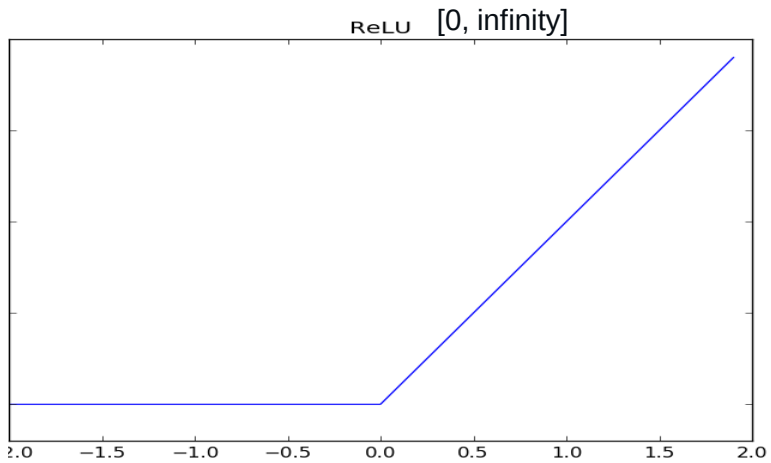
- Tanh can map any real number ([-Inf, Inf]) to a number between [-1 1]
- It is symmetric regarding the 0
- Most of time tanh is quickly converge than sigmoid and performs better accuracy.
- It maybe slower than ReLU but produces more natural looking fits for the data using only linear inputs.

- ReLU is non linear function and trains six times fast than tanh
- No gradient vanishing problem, as Relu's gradient is constant = 1
- Sparsity: When W*x<0, Relu gives 0, which means sparsity
- The usefulness of ReLU is that it prevents gradients from saturating (detail in next slide)

Activation functions cont







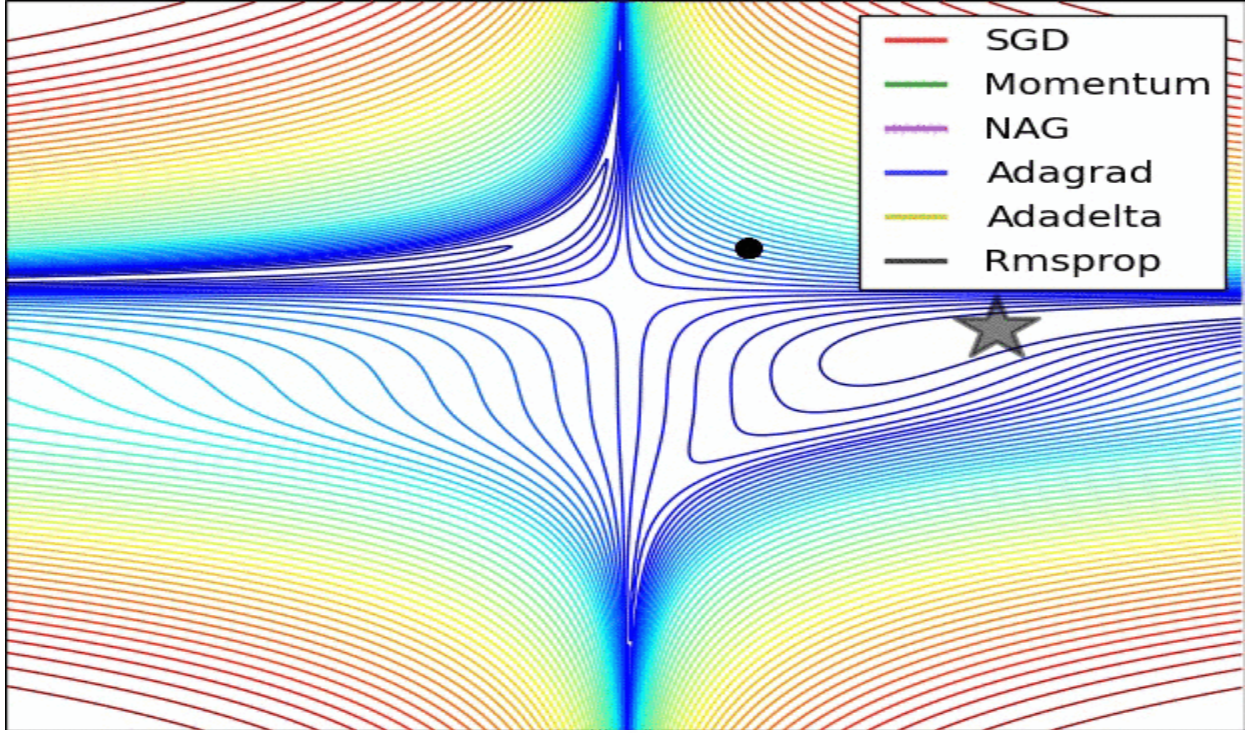
Comparing the graphs of sigmoid or tanh or ReLU, you can see that the previous two "flatten out" far from zero.

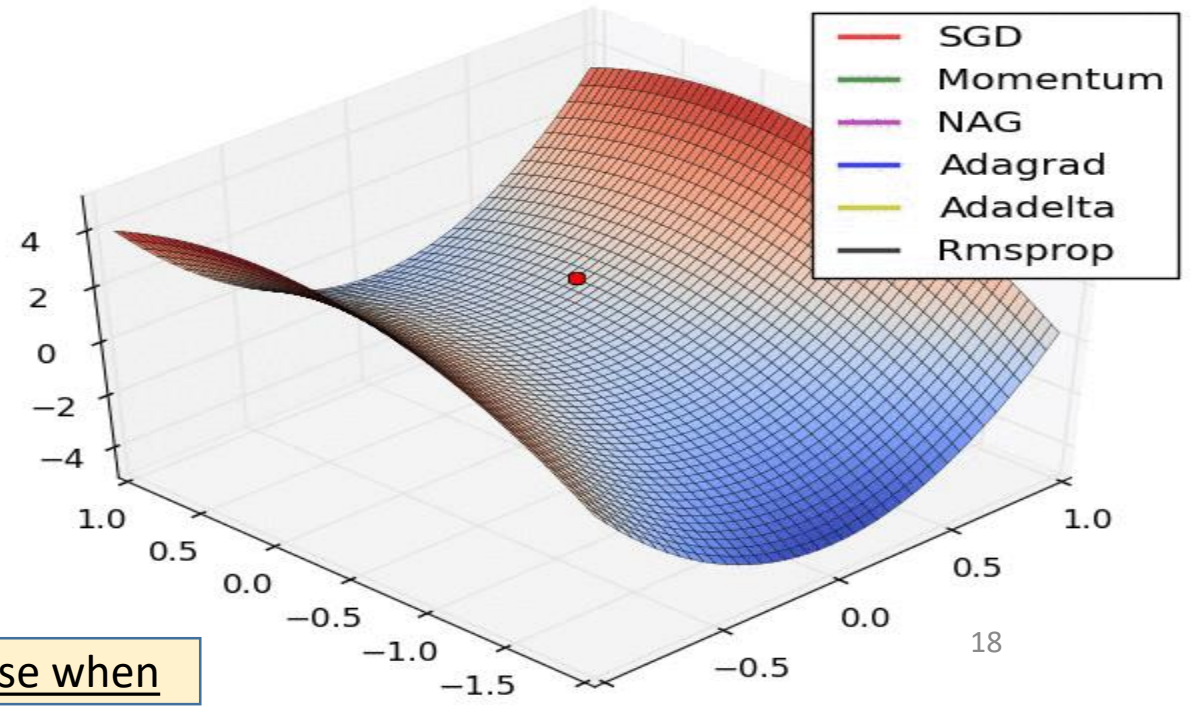
- Gradients saturating: The tanh and sigmoid functions will zero out the gradient if the activation is high. So, say the output of a layer is 5 - if you apply the tanh function to it, the derivative of tanh at x=5 is flat, and thus the network can't learn.
- Notice all this took place within a single layer activation. That means that it can happen for a deep network as well as for a one-layer neural network. ReLU prevents this issue from happening by providing a nonlinear function that doesn't saturate (the derivative is always just 1 for x > 0, meaning it acts as an identity function in back propagation)
- ReLU for the win

Comparison of optimizers

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

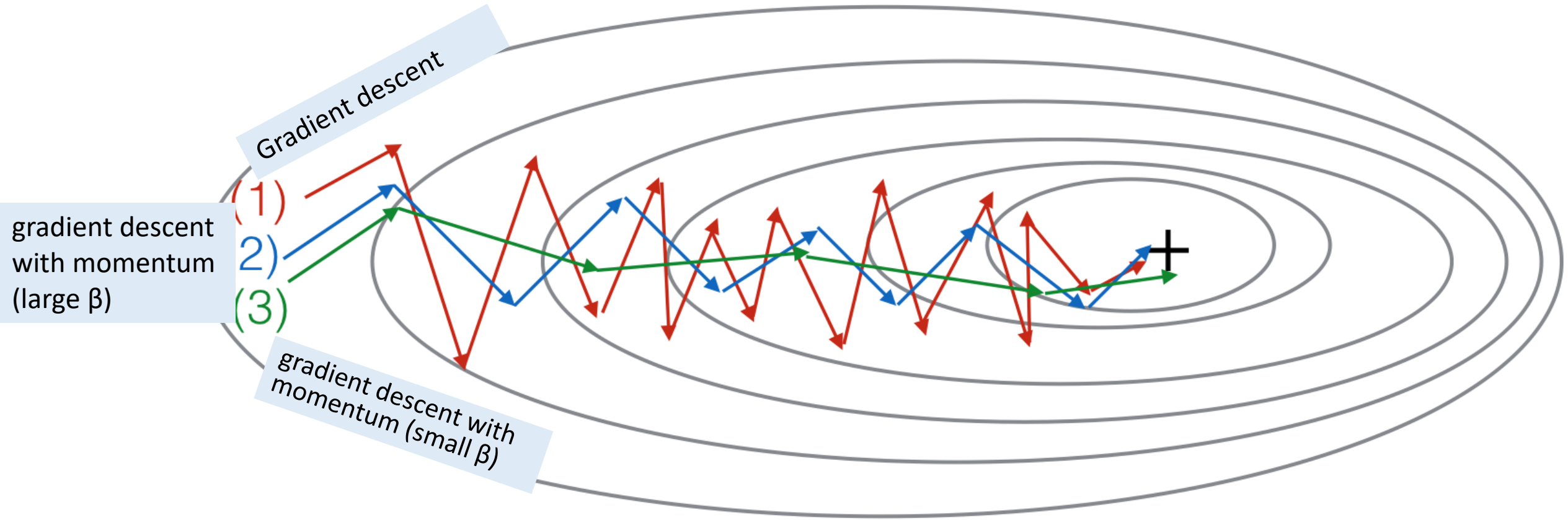
- **SGD** is incremental gradient descent, a stochastic approximation of the gradient descent optimization method for minimizing an objective function that is written as a sum of differentiable functions. In other words, SGD tries to find minima or maxima by iteration.
- **Adam** or adaptive momentum is an algorithm similar to AdaDelta. But in addition to storing learning rates for each of the parameters it also stores momentum changes for each of them separately. The more sophisticated than the descent optimizer SGD





Which one to use when

Summary with Visualization



gradient descent with momentum (large β)

Foundation of TensorFlow: Hands on using Python Scripts

i. Basics of TensorFlow

ii. Rank of tensors

iii. Various Reduction's operation

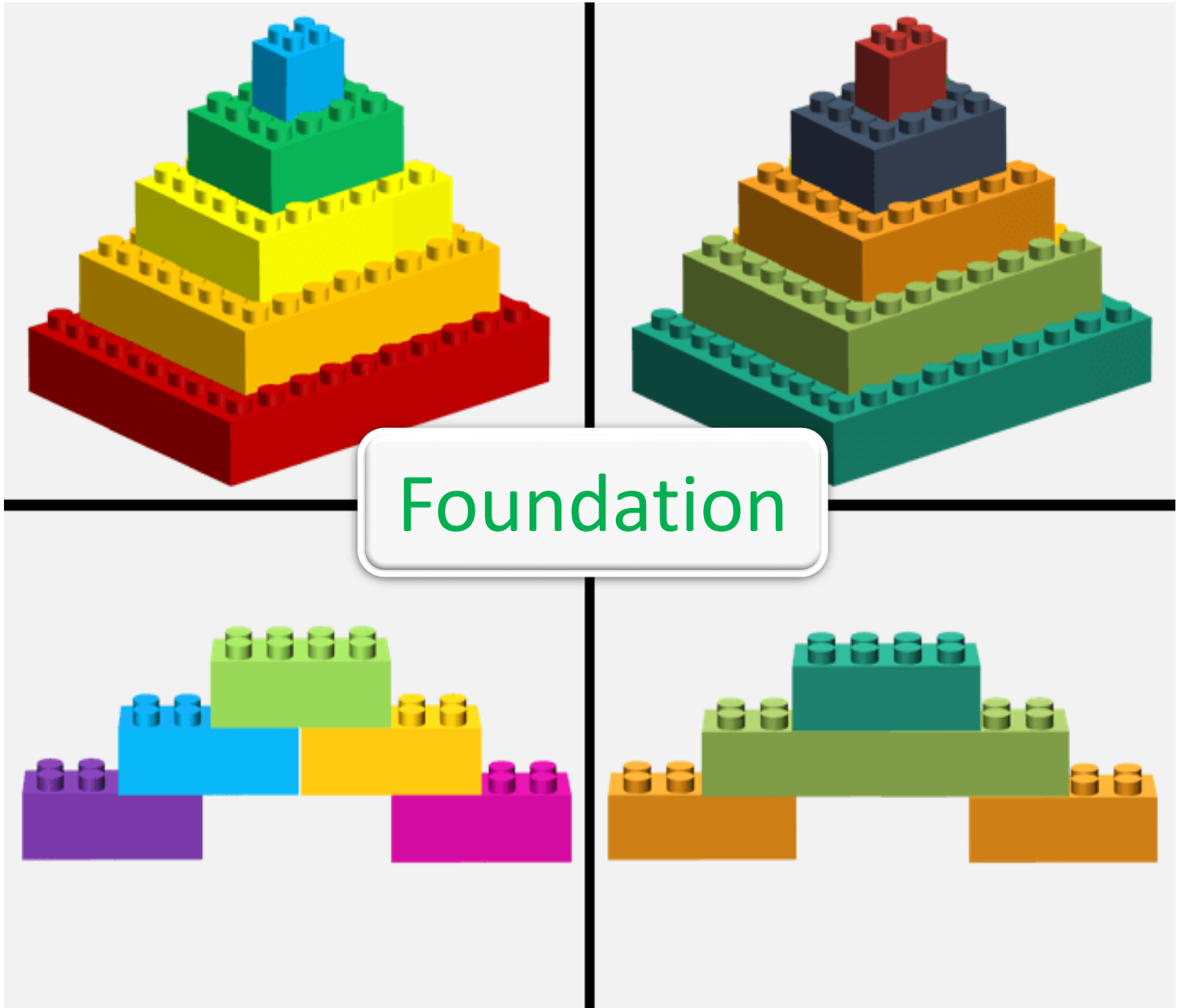
iv. Few important operations

v. TensorFlow inbuilt constants

vi. Tensor segmentation

vii. Various sequence utilities

viii. Tensor slicing and joining



Regression and Classifications

Must be known before proceeding ahead with DL

- What is Regression and Classifications
- Why do you need Regression and Classifications
- Which are proven ML techniques for Regression and Classifications
- How to measure error metrics for Regression and Classifications
- How to test model capability for Regression and Classifications

Scripts (Hands on)

Regression

i. Regression with Premade Estimators
ii. TensorBoard
iii. Regression by using Keras (tf.keras)

Classifications

iv. Classifications using Premade Estimators
v. Multiclass classification using Keras

Few more Classifications

2.1.dl_tf_intermediate_classifications.py

- Binary classification on Kaggle data using TensorFlow Multi level
- Explore few more ways to better classification

How to Teach Machines?

In order for a program to be capable of learning something, it must first be capable of being told it. - John McCarthy







Image source link is at eference section

Let us dive into details of Deep Learning

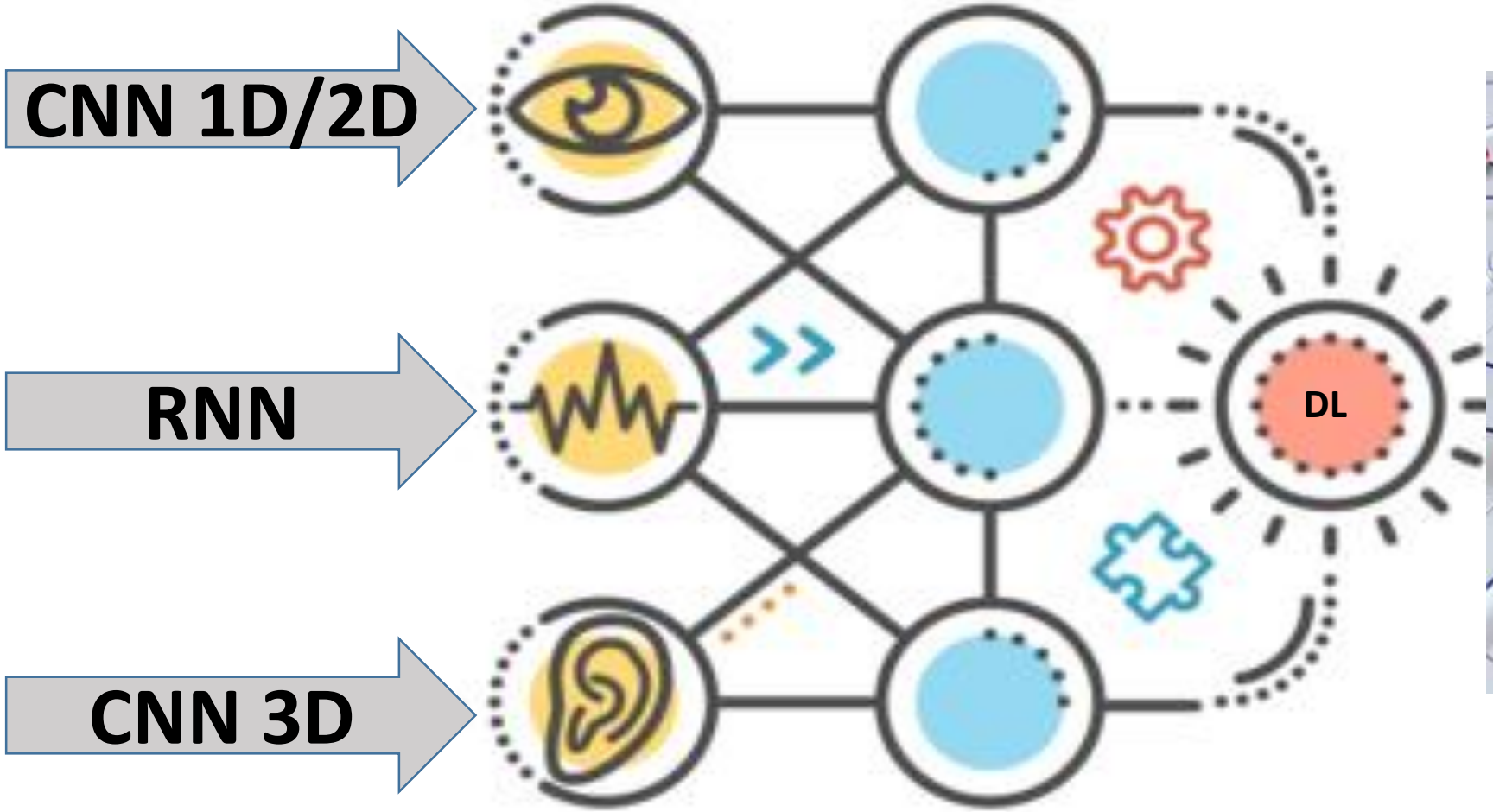
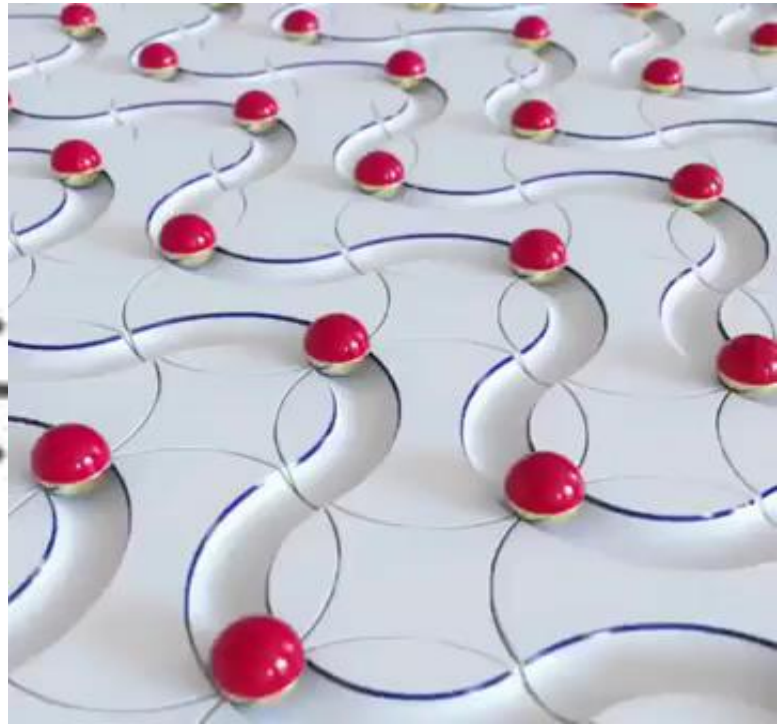


Image source link is at reference section



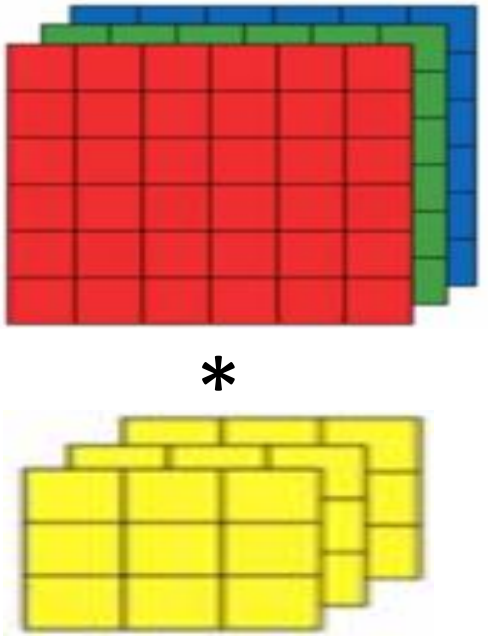
https://www.linkedin.com/feed/update/activity:6458507109741973504/

Not exhaustive

Convolutional Neural Networks (CNN)

Definition and Applications

- Convolution is a mathematical operation that is performed on two functions to produce a third function.
- CNN is a class of deep and feed-forward artificial neural network
- Make the explicit assumption that the inputs are images
- CNNs use relatively little pre-processing compared to other image classification algorithms.
- Synthesize their own feature extractor
- Applications – Image, video recognition, recommender systems and NLP



Steps

- Applying a CNN filter essentially means: Center a kernel on a pixel

i. Multiply the pixels under that kernel by the values in the kernel

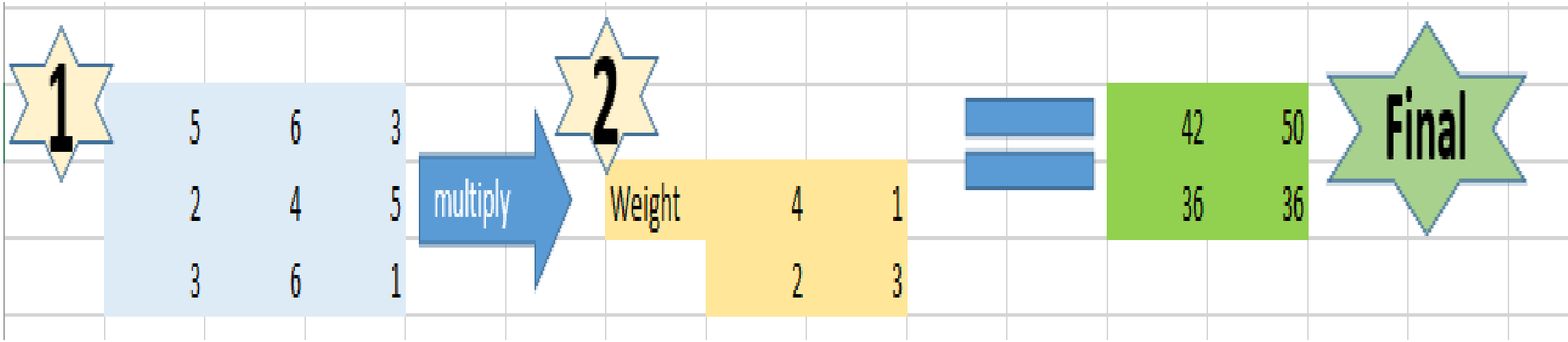
ii. Sum all the those results

iii. Replace the top left pixel with the summed result

- This process is known as convolution.

CNN: Showcase of multiplications

Open file 'Miscelaneous.xlsx' -> tab CNN



- We extracted features from an image by using the spatial arrangement of the images.
- It is what exactly a **Convolutional Neural Network** does.
- The convolved image with extracted specific features did not lose the information about its spatial arrangement.
- Another benefit - it reduces the number of parameters from the image and hence reduces the number of parameters we need to train for the network.

Stride

- For Stride 1 – It moves across the entire image moving **one** pixel at a time
- For Stride 2 – It moves across the entire image moving **two** pixel at a time
- Define it, like a hyper parameter, as to how we would want the weight matrix to move across the image.

INPUT IMAGE

18	54	51	239	244	188
55	121	75	78	95	88
35	24	204	113	109	221
3	154	104	235	25	130
15	253	225	159	78	233
68	85	180	214	245	0

WEIGHT

1	0	1
0	1	0
1	0	1

429

INPUT IMAGE

18	54	51	239	244
55	121	75	78	95
35	24	204	113	109
3	154	104	235	25
15	253	225	159	78

WEIGHT

1	0	1
0	1	0
1	0	1

429

Image source link is at reference section

The size of image keeps on reducing as we increase the stride value

Padding

- Padding the input image with zeros across helps retaining image sizes
- More than one layer of zeros can be added around the image in case of higher stride values.
- Initial shape of the image is retained after padding - known as **same padding** since the output image has the same size as the input.

0	0	0	0	0	0	0	0
0	18	54	51	239	244	188	0
0	55	121	75	78	95	88	0
0	35	24	204	113	109	221	0
0	3	154	104	235	25	130	0
0	15	253	225	159	78	233	0
0	68	85	180	214	245	0	0
0	0	0	0	0	0	0	0

WEIGHT

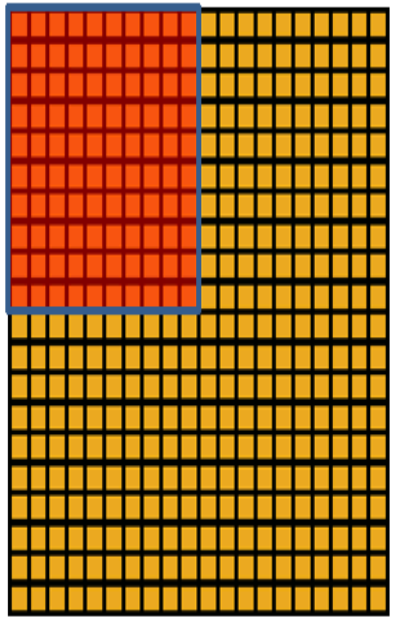
1	0	1
0	1	0
1	0	1

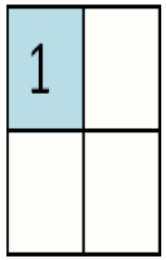
139

Image source link is at reference section

Pooling

Pooling is also called subsampling or down sampling, it reduce the feature map dimension and keep the important information. There were some different types of polling, like Max, Average, Sum and etc. For example, max pooling take the largest element from the feature map within the window, following shows an example of Max Pooling with a 2×2 window.





Convolved feature

Pooled feature

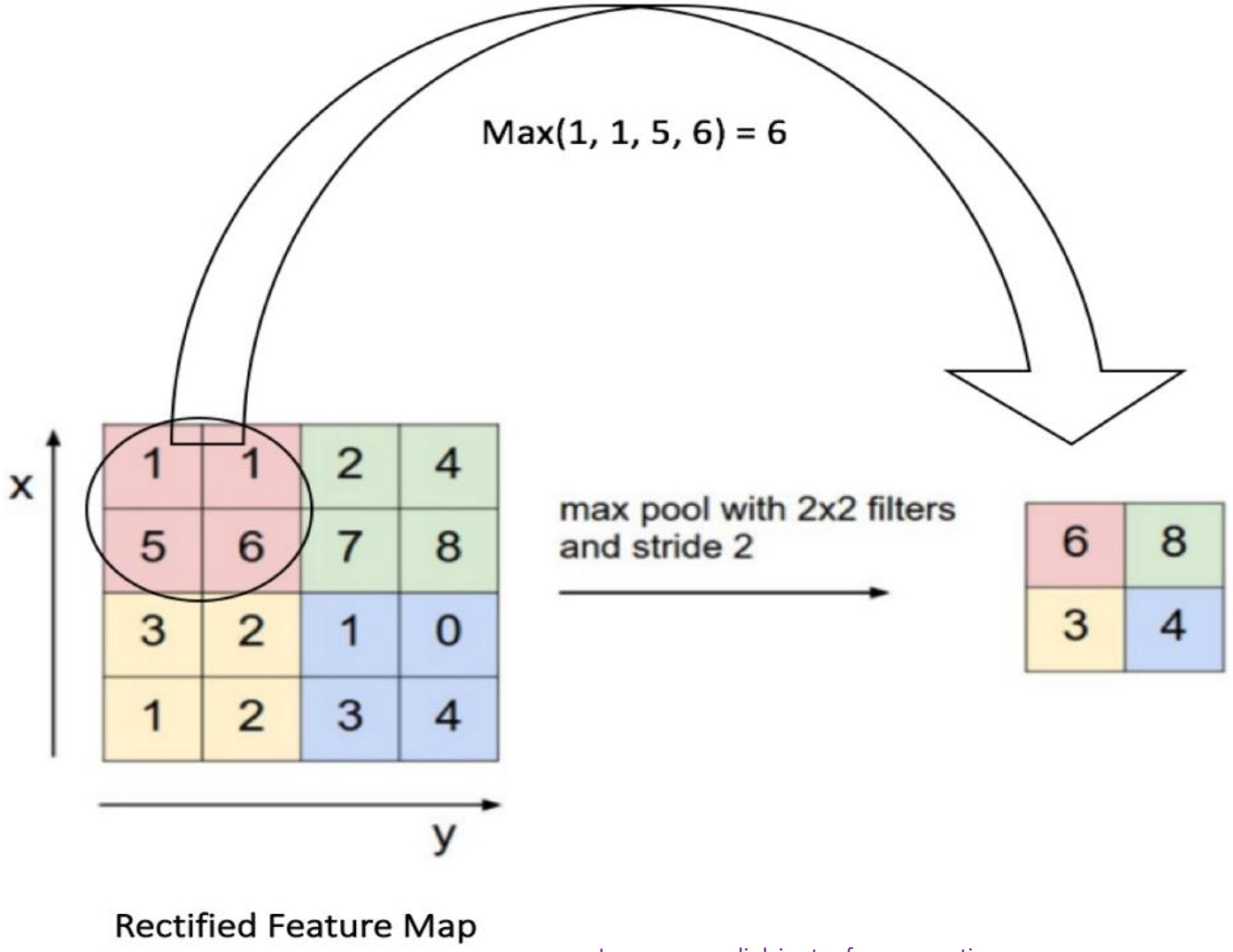


Image source link is at reference section

Input and Output dimensions

Three hyper parameters control the size of output
1.The number of filters – the depth of the output volume will be equal to the number of filter applied.
2.Stride
3.Zero padding

The spatial size of the output image can be calculated as**([W-F+2P]/S)+1**. Here, W is the input volume size, F is the size of the filter, P is the number of padding applied and S is the number of strides.

Here W=32, F=5, P=0 and S=1. The output depth will be equal to the number of filters applied i.e. 10. The size of the output volume will be ([32-5+0]/1)+1 = 28. Therefore the output volume will be 28*28*10.

CNN: Dimensions of the layers and flow (Putting it all together)

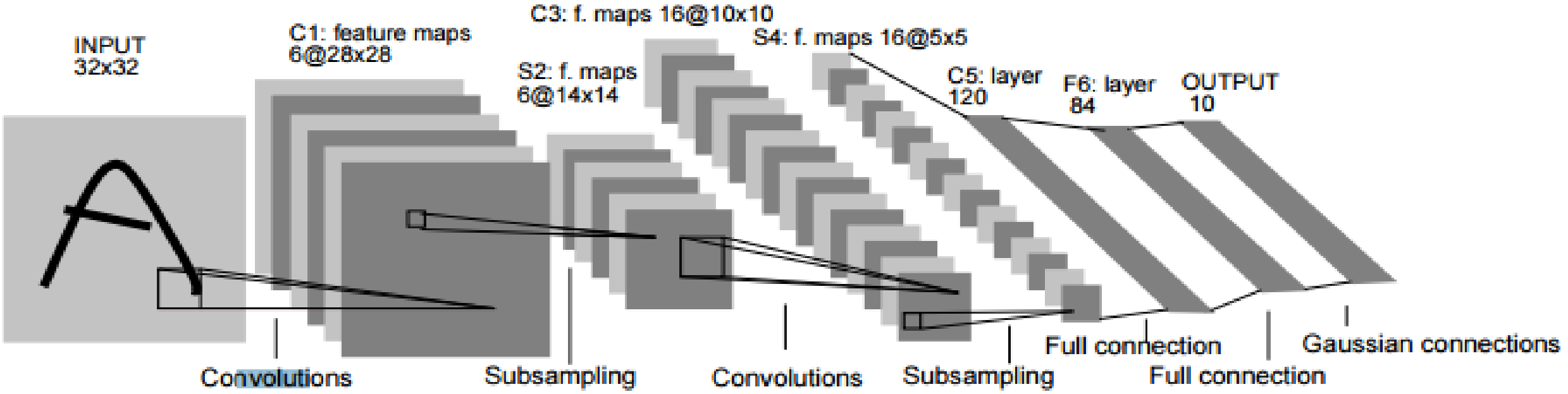
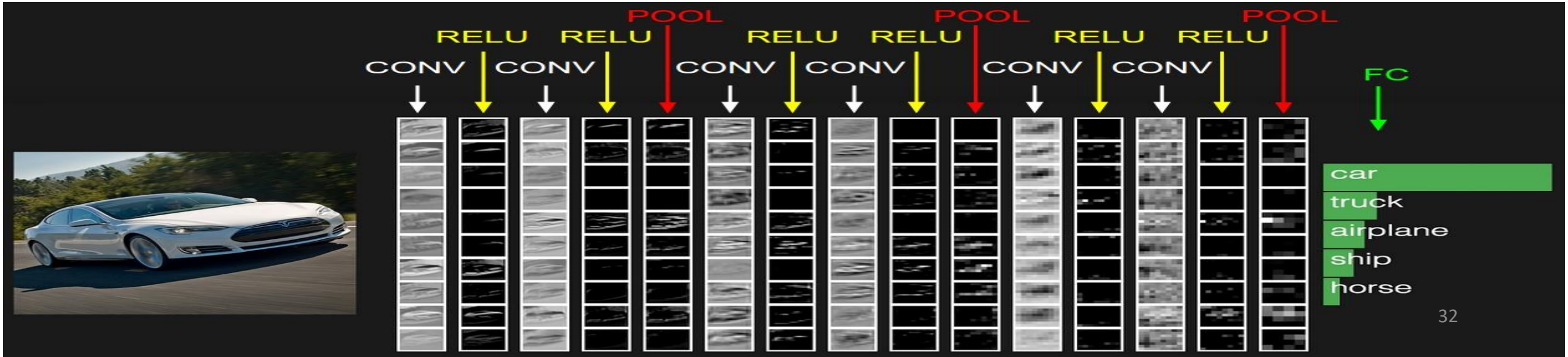


Image source link is at reference section



CNN in TensorFlow

- tf.nn.conv2d(input, filter, strides, padding, use_cudnn_on_gpu, data_format)

➢input: tensor with definite format of four dimensions

[batch, in_height, in_width, in_channels] (order is called NHWC or NCWH)

Example, a single 28×28 pixel color image will have the following shape: [1,28,28,3]

➢filter: This is a tensor representing a kernel or filter. It has a very generic method:

[filter_height, filter_width, in_channels, out_channels]

➢strides: This is a list of four int tensor datatypes, which indicate the sliding windows for each dimension.

➢Padding: This can be SAME or VALID. SAME will try to conserve the initial tensor dimension, but VALID will allow it to grow in case the output size and padding are computed.

➢use_cudnn_on_gpu: This indicates whether or not to use the CUDA GPU CNN

library to accelerate calculations.

➢data_format: This specifies the order in which data is organized (NHWC or NCWH).

Script (Hands on): The classifications using Convolutional Neural Networks (CNNs)

RNN: Let us start with Example

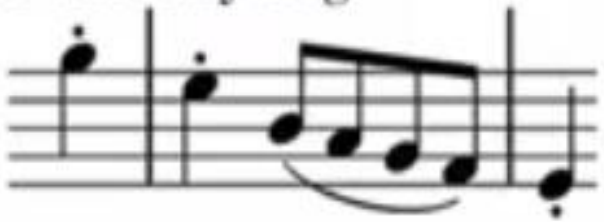
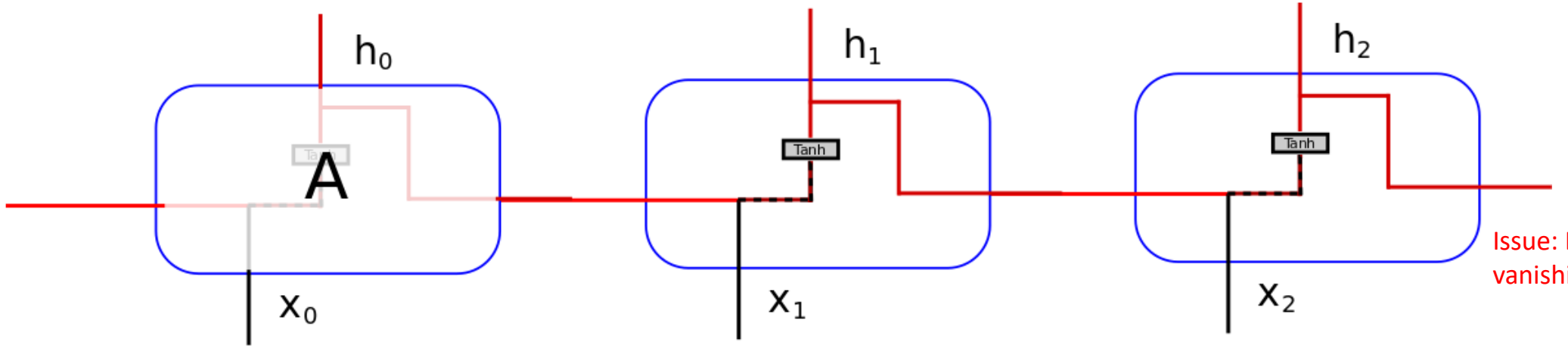
Speech recognition		→	"The quick brown fox jumped over the lazy dog."
Music generation	∅	→	
Sentiment classification	"There is nothing to like in this movie."	→	★☆☆☆☆
DNA sequence analysis	AGCCCCTGTGAGGAACTAG	→	AGCCCCTGTGAGGAACTAG
Machine translation	Voulez-vous chanter avec moi?	→	Do you want to sing with me?
Video activity recognition		→	Running
Name entity recognition	Yesterday, Harry Potter met Hermione Granger.	→	Yesterday, Harry Potter met Hermione Granger.

Image source link is at reference section

Recurrent Neural Networks

RNN is a sequential model of neural networks, which have the property of reusing information already given in previous data. Instead of neurons, LSTM networks have memory blocks that are connected through layers.



Issue: Exploding and vanishing gradients

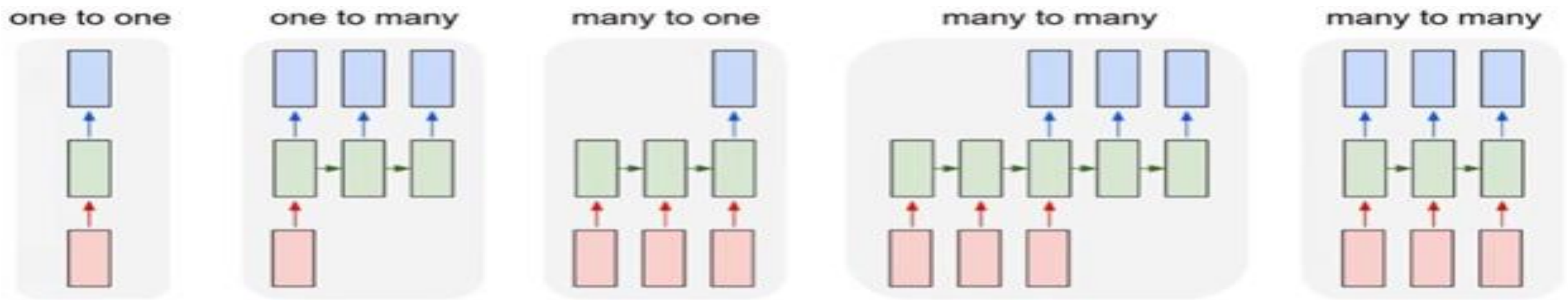


Image source link is at reference section

Why do we need RNN or why simple a neural net cannot do well on NLP problems

- Simple neural network will need a fixed length of input sequence, which is not true for texts. This is a minor challenge as we can pad all sequences by 0s but such intervention increases redundancy.
- Simple neural network will have enormous number of parameters to train. For instance, if our individual sequence has 20 elements and our vocabulary is of 10k words, our input feature list becomes 200k long.
- Simple neural network does not have shared weights over sequences. If you have read about CNN, you will know how shared weights in the convolution layer helps us to compensate for spatial movement of the subject. Similarly, if our words of key interest move in the sequence, it should have minimal impact to our model.
- Additionally, our model should try to understand the meaning of a sentence as a whole instead of individual words. This is exactly what LSTM and GRU, which are types of RNN, do.

The Vanishing and Exploding Gradient

- Some deep neural networks, the gradient tends to get smaller as we move backward through the hidden layers. This means that neurons in the earlier layers learn much more slowly than neurons in later layers. This is known as the *vanishing gradient problem*
- Sometimes the gradient gets much larger in earlier layers! This is the *exploding gradient*
- This instability is a fundamental problem for gradient-based learning in DNN.

Vanishing Gradient

Exploding Gradient

How to recognize

- *The Training process takes too long*
- *Accuracy decreases*

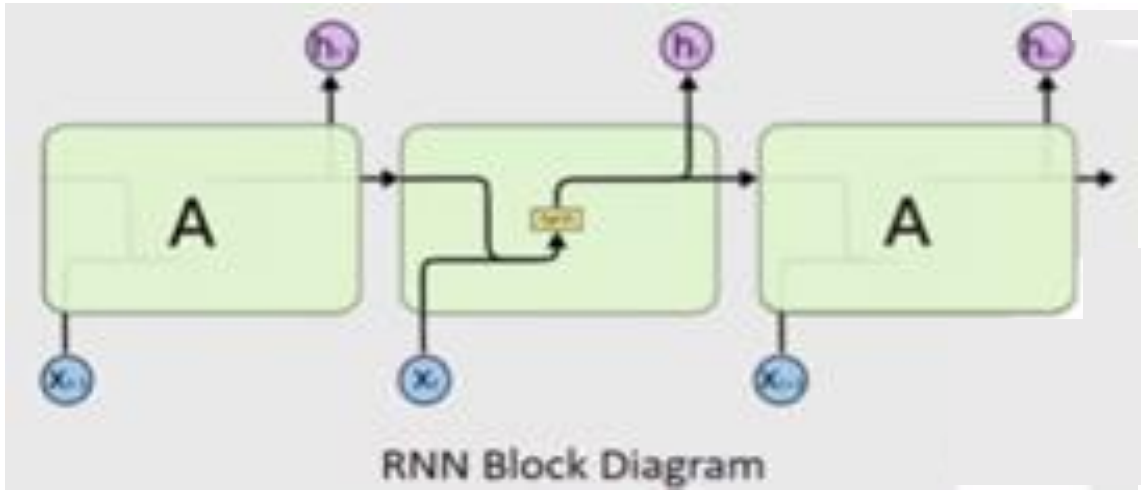
- The model weights are getting very large.
- The model weights go to NaN.
- The model is unstable, resulting in large changes in loss

Probable Solution

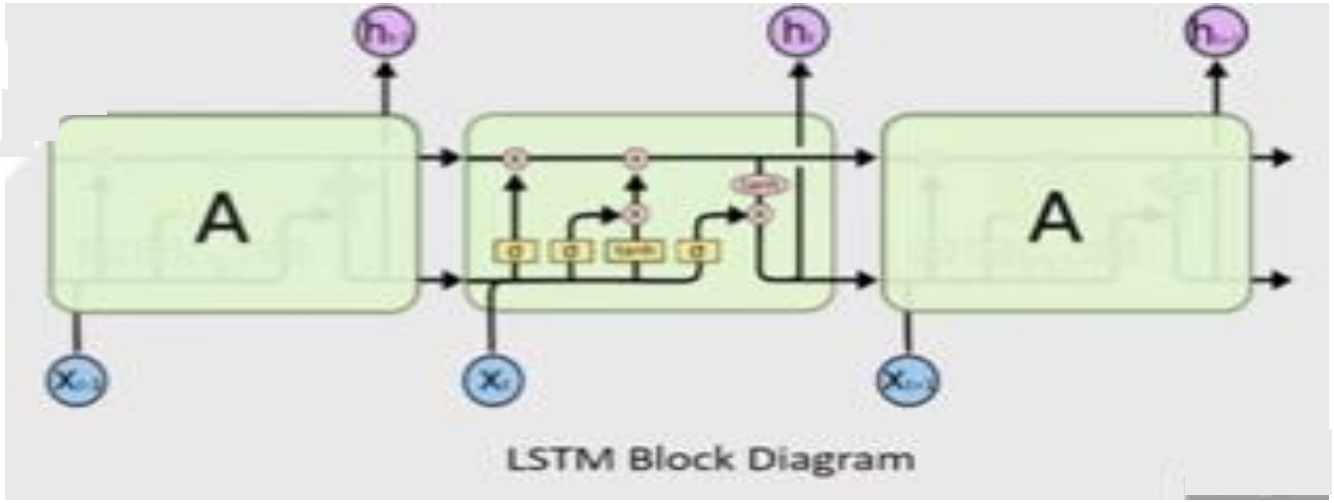
- Use activation function Relu
- Review design (Residual network)
- Use LSTM
- Initialize weights appropriately
- Use upper limit for weights

The Long Short Term Memory (LSTM)

- The Long Short Term Memory (LSTM) is a specific RNN architecture to represent long term dependencies. Moreover, they are specifically designed to remember information patterns and information over long periods of time.
- It's operation allow both discard (not important) low state data, and incorporate (important) new data to the state.



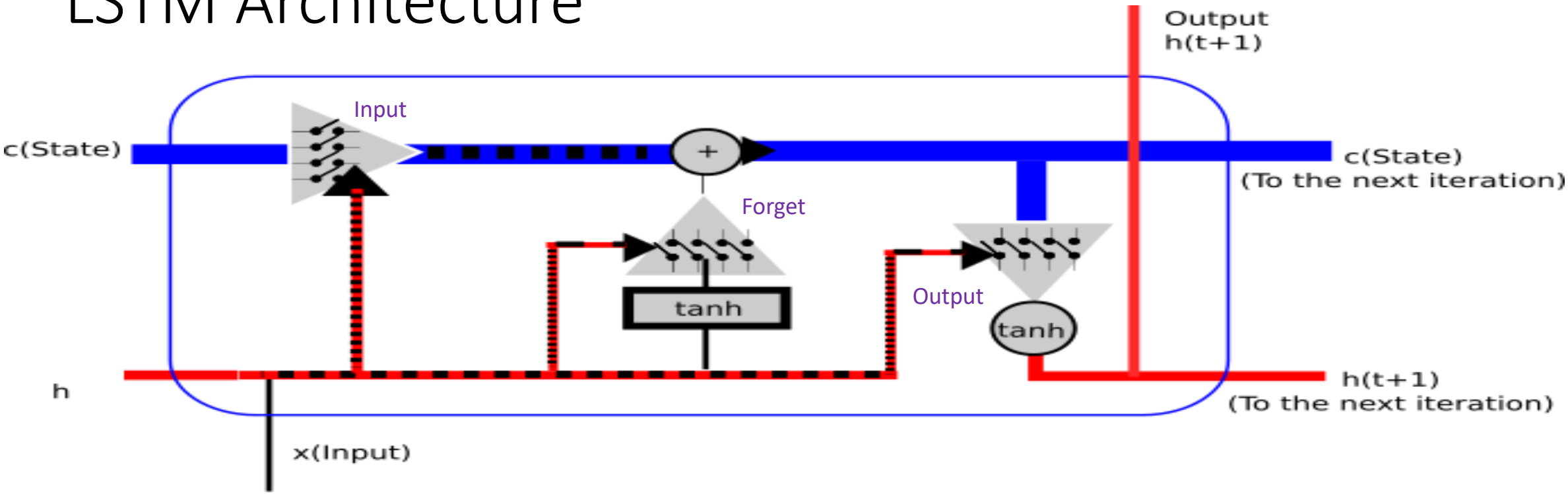
RNN Block Diagram



LSTM Block Diagram

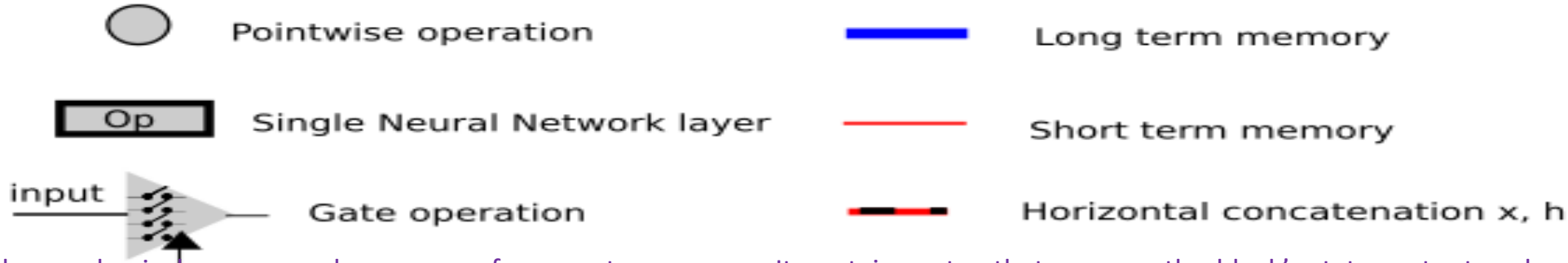
Image source link is at reference section

LSTM Architecture



Benefit - learn long sequences, a one-shot multi-step forecast

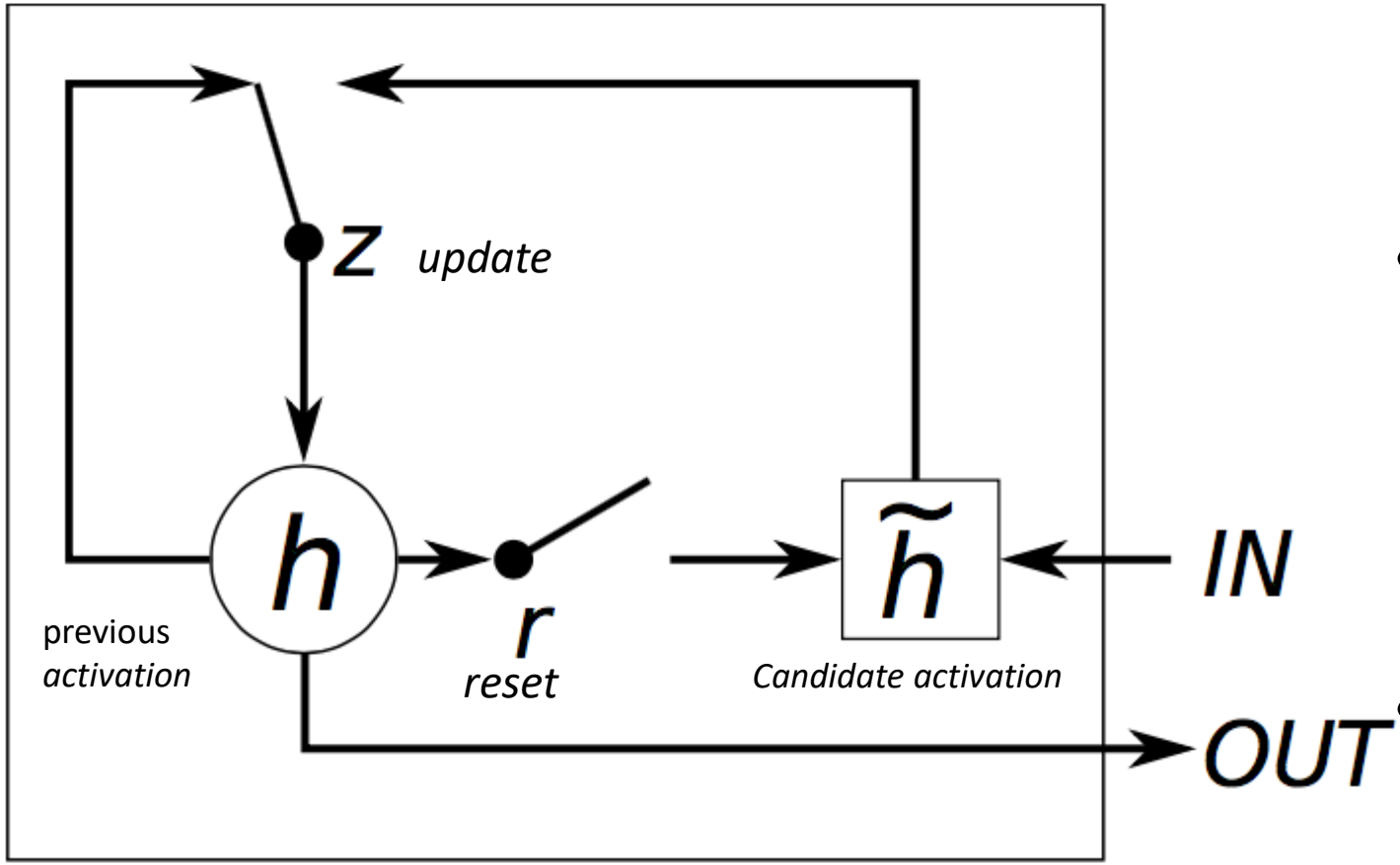
tricky to configure



Block is smarter than a classical neuron and a memory for recent sequences. It contains gates that manage the block's state, output and uses the sigmoid activation units to control whether they are triggered or not, making the change of state and addition of information flowing through the block conditional. Three types - Input , Output and Forget Gate. Each unit is like a mini-state machine where the gates of the units have weights that are learned during the training procedure.

Image source link is at reference section

Gated recurrent neural networks : GRU Architecture



- It has two gates - *reset* and *update* gates
- The reset gate sits between the previous activation and the next candidate activation to forget previous state
- The update gate decides how much of the candidate activation to use in updating the cell state.

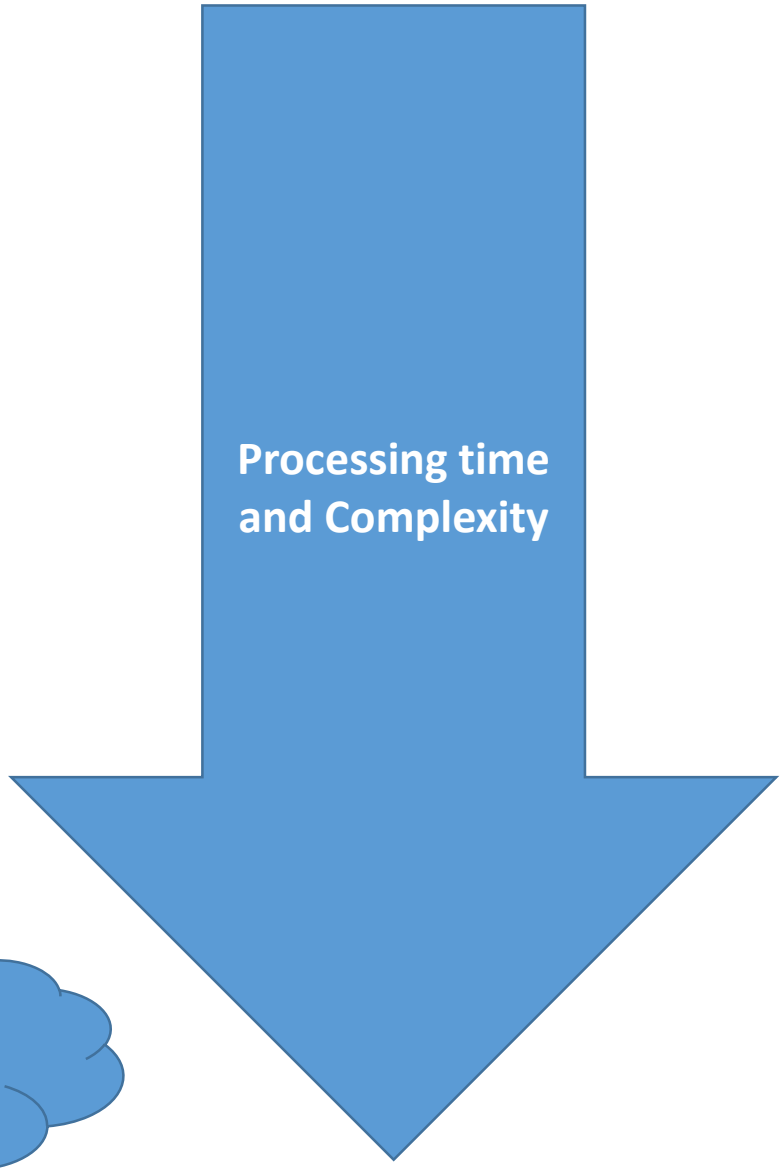
GRU Gating. Chung, Junyoung, et al. “Empirical evaluation of gated recurrent neural networks on sequence modeling.” (2014)

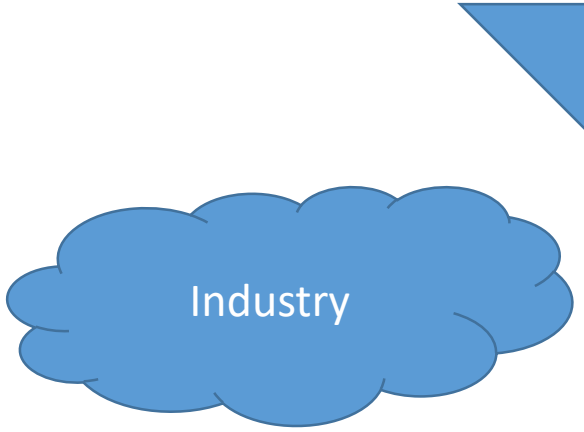
LSTM vs GRU

- LSTM maintains two different memories – Cell State and Hidden state. Cell state is the long term memory and hidden state is the short term memory, very similar to human brain.
- Forget gate is an important difference between LSTM and GRU. This gate regulates how much previous information needs to be sent to the next cell, whereas GRU exposes its entire memory to the next cell.
- Hidden state is used to evaluate all the gates in LSTM. Hence controlling the hidden state that is moving forward to the next cell can give us complete control of all gates in the next cell. The difference is not a strong one when it comes to practical applications of LSTM vs. GRU.
- The cell state acts as a shortcut for the back propagation to travel and hence avoids the problem of diminishing gradient.
- GRU is new and hence not as reliable as LSTM. I, personally, not convinced of it
- GRU runs a lot faster because of the lower number of parameters.
- LSTM works better in sentences with long dependencies. Heard conflict and hence validate
- GRU is relatively new. Experiment with it

LSTM and GRU models: Which one to choose

- Bidirectional LSTM
- Bidirectional GRU
- LSTM
- GRU





Library of RNNCells

- BasicRNNCell
- BasicLSTMCell
- GRUCell
- LSTMCell
- LayerNormBasicLSTMCell
- CoupledInputForgetGateLSTMCell
- TimeFreqLSTMCell
- GridLSTMCell
- **New:** NASCell

- MultiRNNCell
- DropoutWrapper
- DeviceWrapper
- ResidualWrapper
- AttentionCellWrapper
- CompiledWrapper

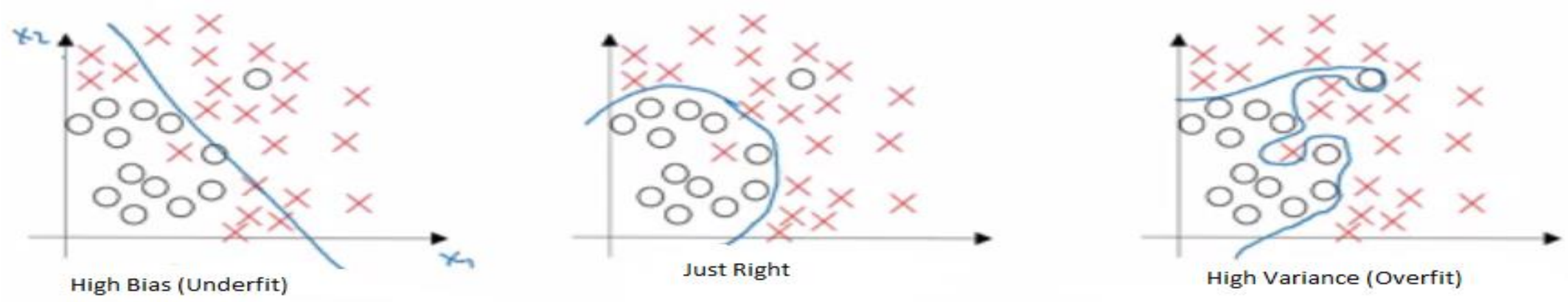
Script (Hands on): The Time Series using LSTM

i. Univariate Time Series using LSTM
   i. With train –test mode
   ii. With train mode only

ii. Multivariate Time Series using LSTM

How to know models are good enough: Bias vs Variance

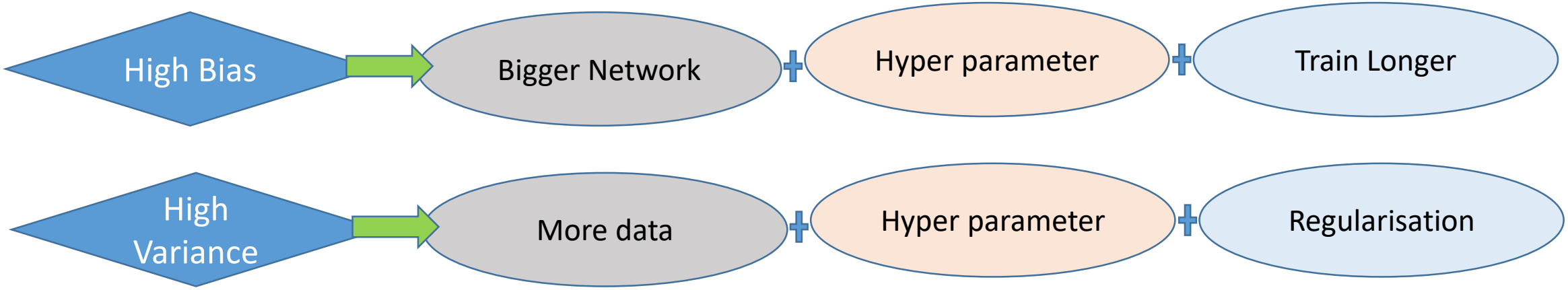
Definition



Error symptom

	High Bias	High Variance	High Bias & High Variance	Low Bias & Low Variance
Train	16%	2%	16%	1%
Dev/Test	17%	12%	30%	1.5%

Solution



How to avoid Overfitting

- Definition: Training error vs. Test error. If you have close to 0 training error and much larger test error, it typically means that your model is overfitting.
- Dropout - A 0.5 to 0.75 dropout rate between fully connected layers is going to work quite well and a 0.1–0.3 dropout rate would typically do better for the input layer. But as a rule of thumb: the more data you have, the less dropout you actually need.
- Minimize model complexity
- Total number of weights in your hidden layers should be of maximum 1/2 of your training samples.
- Default: Try getting more data vertically (samples) or horizontally (features).
- Use architecture that generalizes well
- Add regularization $Cost\,function = Loss + \frac{\lambda}{2m} * \sum \|w\|$ $Cost\,function = Loss + \frac{\lambda}{2m} * \sum \|w\|^2$ **lambda** is the regularization parameter

Various sources: Please see the references

Script (Hands on): How to avoid Overfitting

- Weight regularization: L2 (weight decay)
- Weight regularization: L1
- Dropout
- Batch Normalization
- Early Stopping

Hyper parameters tuning for Deep Learning Models

- Grid search (random is preferred) is a hyper parameter optimization technique.
- Important because the neural networks are difficult to configure and there are a lot of parameters that need to be set.

i. Learning rate and Momentum
ii. Epochs
iii. Batch size (Particularly for LSTM RNN and CNN)
iv. Number of Layers
v. Batch Normalisation
vi. Number of neurons
vii. Neuron Activation Function
viii. Network Weight
ix. Dropout rate
x. Learning rate decay

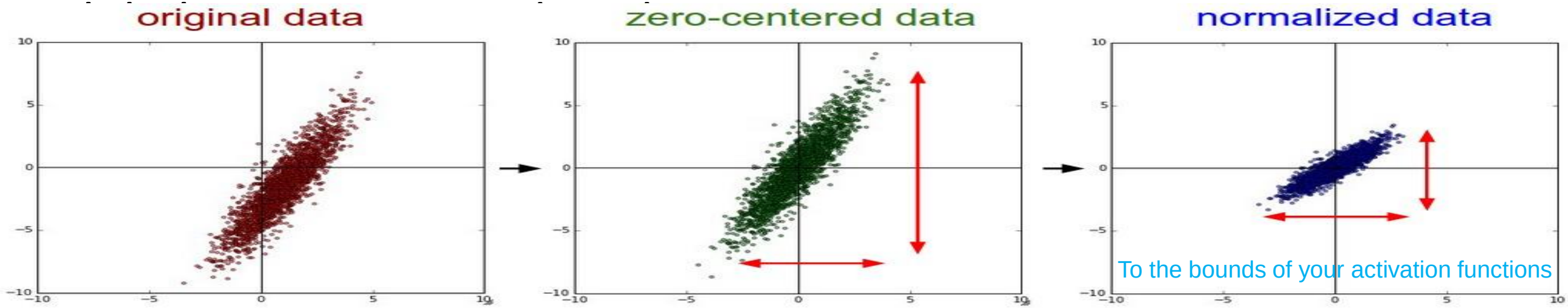
Priority 1, **Priority 2**, **Priority 3**

Batch Normalisation can be considered for tuning

- Each mini batch is scaled by mean and standard deviation calculated on just that mini batch
- It works just before activation in one of the following way
  - model.add(Dense(10, input_dim=(2,), batch_norm = True, activation = 'relu'))
  - model.add(Dense(10))
  - model.add(BatchNormalization())
  - model.add(Activation('relu'))
- It adds some noise within mini-batch. Similar to dropout, it adds some noise to each hidden layer's activation
- It works later in process rather than early as happen in input data
- BN reduce the chance of fluctuation and hence future layer can learn better
- It has slight regularization effect

Best Practices for DL

1. Normalization can improve learning speed because the path to the



2. More data more accuracy
3. Choosing a good activation function allows training better and efficiently
4. Choose large initial learning rate if it will not oscillate or diverge so as to find a better global minimum.
5. Shuffling and augmentation of data

References

- https://www.tensorflow.org/tutorials/wide_and_deep
- https://www.tensorflow.org
- https://en.wikipedia.org/wiki/TensorFlow
- https://stackoverflow.com/questions/36162180/gradient-descent-vs-adagrad-vs-momentum-in-tensorflow
- http://int8.io/comparison-of-optimization-techniques-stochastic-gradient-descent-momentum-adagrad-and-adadelta/
- http://yann.lecun.com/exdb/lenet/
- Book: Building Machine Learning Projects with TensorFlow. Author: Rodolfo Bonnin
- http://ufldl.stanford.edu/tutorial/supervised/Pooling/
- https://www.analyticsvidhya.com/blog/2017/06/architecture-of-convolutional-neural-networks-simplified-demystified/#comment-137567
- http://www.yuthon.com/images/non-convex_and_convex_function.png
- https://iamtrask.github.io/img/sgd_optimal.png
- https://github.com/jalajthanaki/NLPython/blob/master/ch9/gradientdescentexample/gradient_descent_example.gif
- https://www.analyticsvidhya.com/blog/2017/08/10-advanced-deep-learning-architectures-data-scientists/?utm_source=feedburner&utm_medium=email&utm_campaign=Feed%3A+AnalyticsVidhya+%28Analytics+Vidhya%29
- Coursera Deep Learning course
- https://machinelearningmastery.com/improve-deep-learning-performance/
- https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/
- https://www.coursera.org/specializations/deep-learning by Andrew Ng et.al
- How to avoid overfitting – regularization, Page 198, Hands-On Transfer Learning with Python.pdf
- https://media.licdn.com/dms/document/C4D1FAQHLns6sxgJ2Aw/feedshare-document-pdf-analyzed/0?e=1569063600&v=beta&t=N8HTLP4mu7yfQGdHWc3wjqOHeFkiIOeS5WWzrplnx60

Annexures

Miscellaneous

- 13 Common Mistakes Amateur Data Scientists Make and How to Avoid Them?
- https://semanti.ca/blog/?glossary-of-machine-learning-terms

Few examples (Ranking)

Back

S.N	Scenario	SGD	AdaGrad	AdaDelta	Adam
1	**Network with linear layer and softmax output**	3	1	4	2
2	**Network with sigmoid layer (100 neurons), linear layer and softmax output**	1	4	2	3
3	**Network with tanh layer (100 neurons), linear layer and softmax output**	1	4	2	3
4	**Network with sigmoid layer (300 neurons), ReLU layer (100 neurons), sigmoid layer (50 neurons) again, linear layer and softmax output**	2	4	1	3

- SGD seem to be a good fit to any problem of the above – produce leading results often.
- AdaGrad is the best with the shallow problem (softmax classifier)
- AdaDelta is somewhere between all others – never leads, never fails badly too.
- Adam has his big moment minimizing error of network.

Adam (adaptive moment estimation)

Back

- The Adam optimization algorithm is an extension to stochastic gradient descent

- Adam is an optimization algorithm that can used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data.

- When introducing the algorithm, the authors list the attractive benefits of using Adam on non-convex optimization problems, as follows:

    - Straightforward to implement.
    - Computationally efficient.
    - Little memory requirements.
    - Invariant to diagonal rescale of the gradients.
    - Well suited for problems that are large in terms of data and/or parameters.
    - Appropriate for non-stationary objectives.
    - Appropriate for problems with very noisy/or sparse gradients.
    - Hyper-parameters have intuitive interpretation and typically require little tuning.

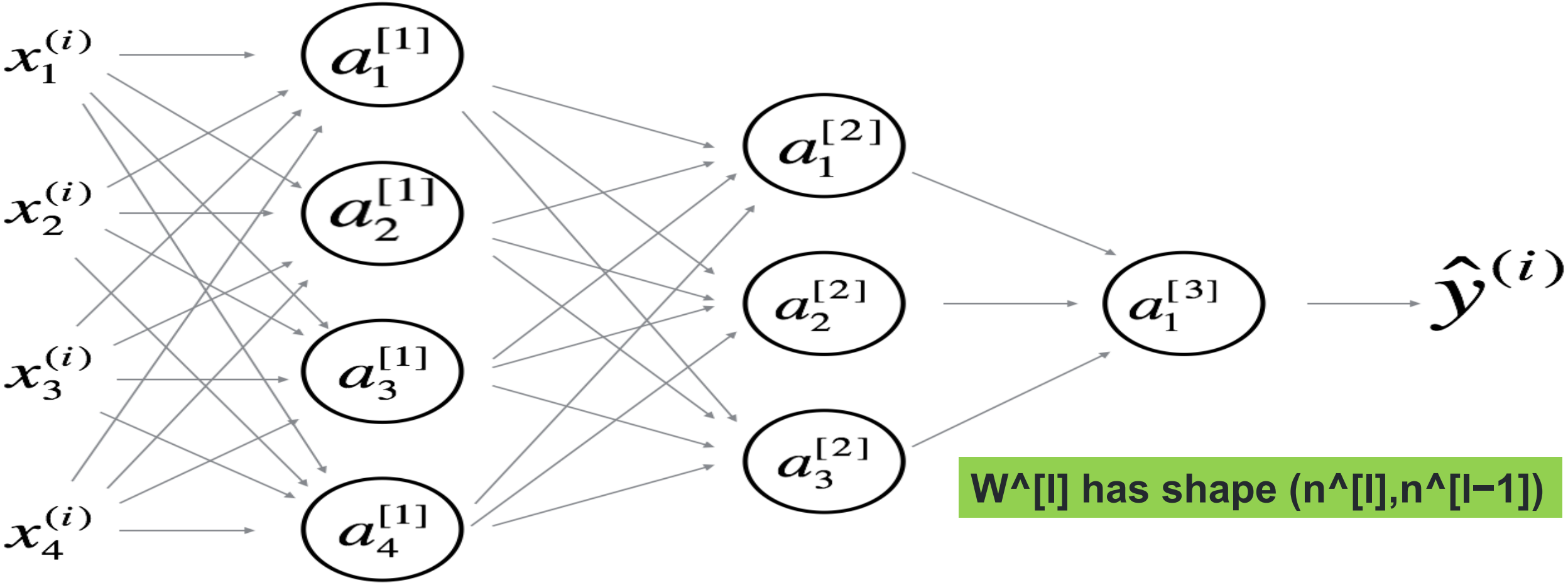
- Stochastic gradient descent maintains a single learning rate (termed alpha) for all weight updates and the learning rate does not change during training.

- The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients.

Numpy to TensorFlow

Numpy	TensorFlow
`a = np.zeros((2,2)); b = np.ones((2,2))`	`a = tf.zeros((2,2)), b = tf.ones((2,2))`
`np.sum(b, axis=1)`	`tf.reduce_sum(a,reduction_indices=[1])`
`a.shape`	`a.get_shape()`
`np.reshape(a, (1,4))`	`tf.reshape(a, (1,4))`
`b * 5 + 1`	`b * 5 + 1`
`np.dot(a,b)`	`tf.matmul(a, b)`
`a[0,0], a[:,0], a[0,:]`	`a[0,0], a[:,0], a[0,:]`

CW: What is shape of W and b at each levels (1,2,3)



W^[l] has shape (n^[l],n^[l−1])

W^[1] will have shape (4, 4)
b^[1] will have shape (4, 1)

W^[2] will have shape (3, 4)
b^[2] will have shape (3, 1)

W^[3] will have shape (1, 3)
b^[3] will have shape (1, 1)

GPU vs CPU

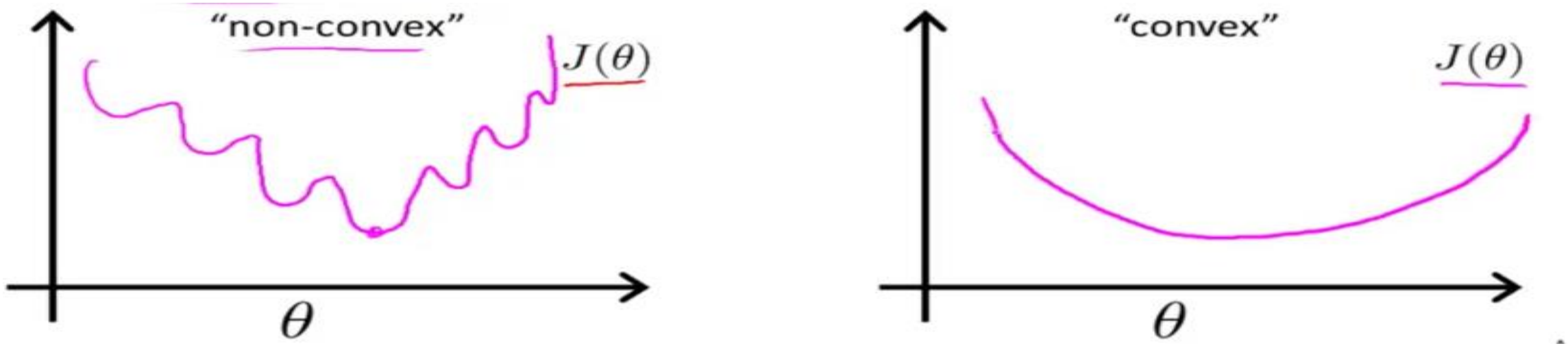
GPU

- hundreds of simpler cores
- thousand of concurrent hardware threads
- maximize floating-point throughput
- most die surface for integer and fp units

CPU

- few very complex cores
- single-thread performance optimization
- transistor space dedicated to complex ILP
- few die surface for integer and fp units

Non-convex and convex curve



We mostly use functions that are able to provide a convex curve because then we can use the gradient descent algorithm to minimize the error function and reach a global minimum. As you can see , a non-convex curve has many local minima, so to reach a global minimum is very challenging and time-consuming because you need to apply second order or nth order optimization to reach a global minimum, whereas in a convex curve, you can reach a global minimum certainly and quickly.