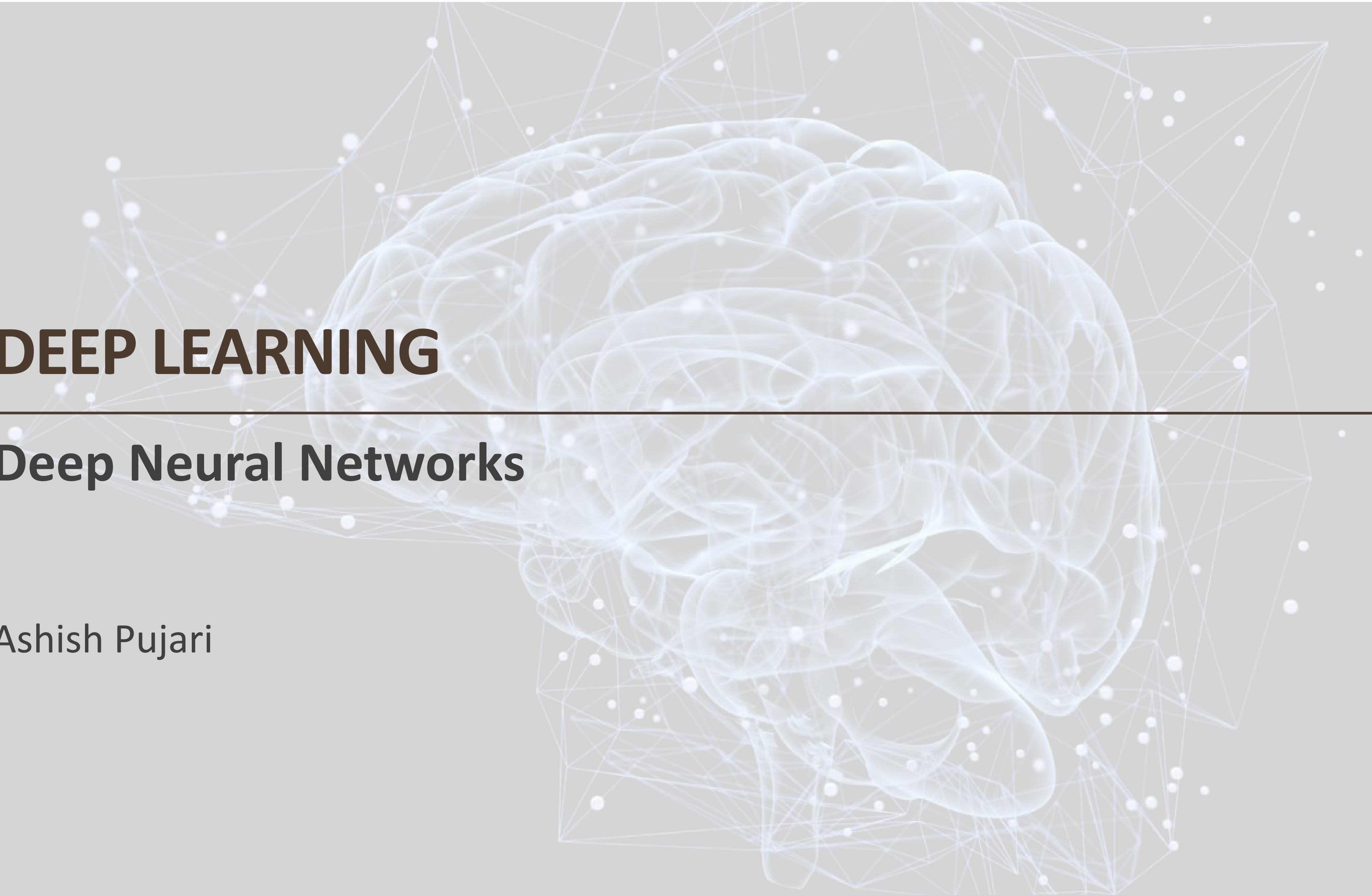


DEEP LEARNING

Deep Neural Networks

Ashish Pujari

Lecture Outline

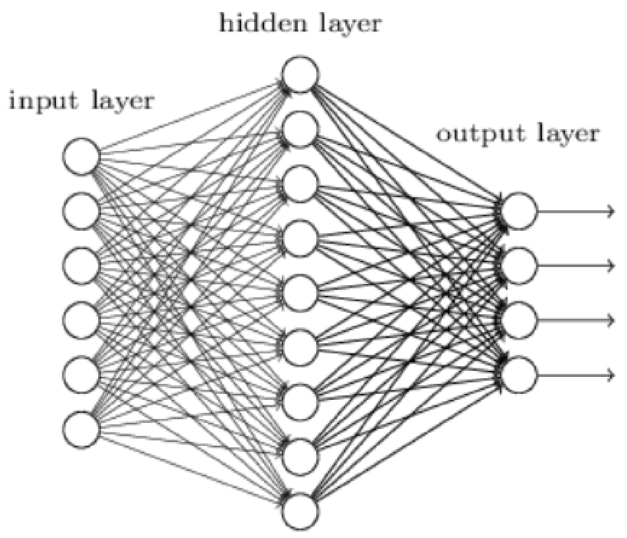
1. DNN
2. DNN Design
3. TensorFlow

DNN

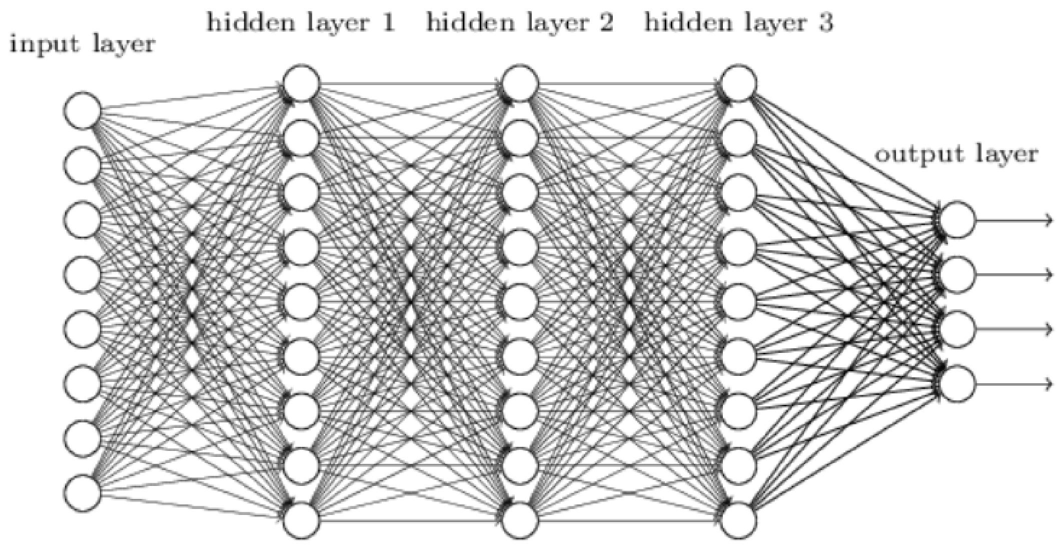
Deep Neural Networks

Deep Neural Networks (DNNs)

- DNNs contain many more hidden layers or neurons in different configurations
- Known to generalize better on high dimensional data and cognitive problems



Artificial Neural Network



Deep Neural Network

DNN Architectures

- FFN: Feed forward Network
- CNN : Convolutional Neural Networks
- RNN: Recurrent Neural Networks
- LSTM: Long Short-Term Memory
- DBN: Deep Belief Networks

Benefits of Depth

- Highly nonlinear functions are even better function approximators
- Learning representations makes them highly flexible to various problems and domains
- Training can be parallelized using powerful multicore CPUs and GPUs

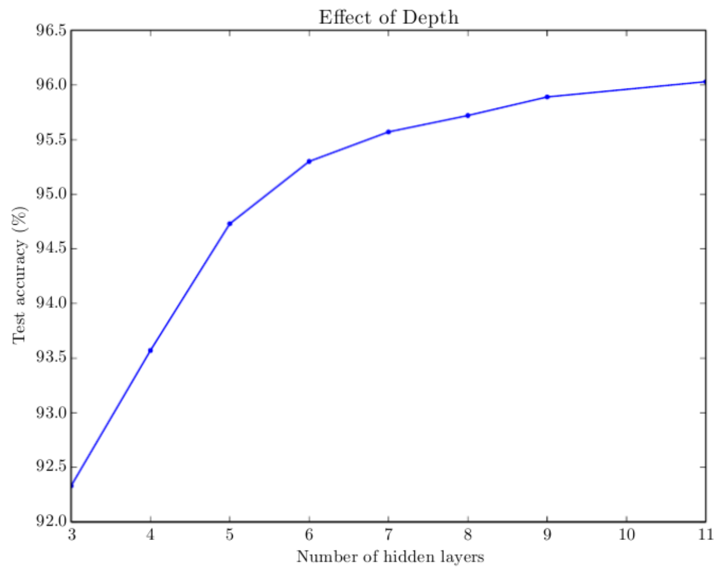
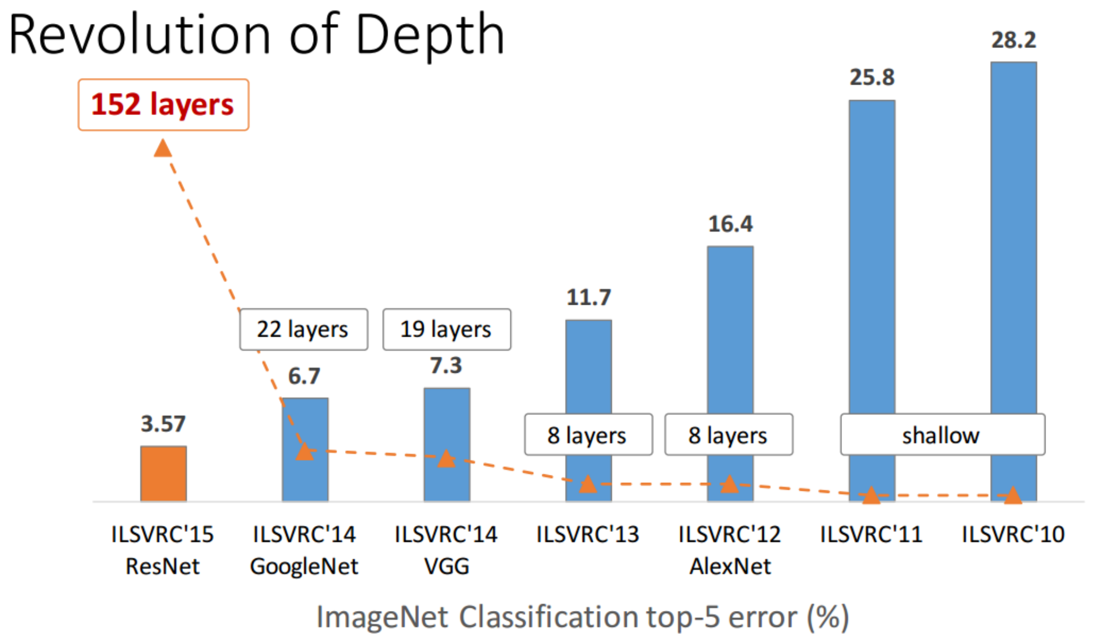
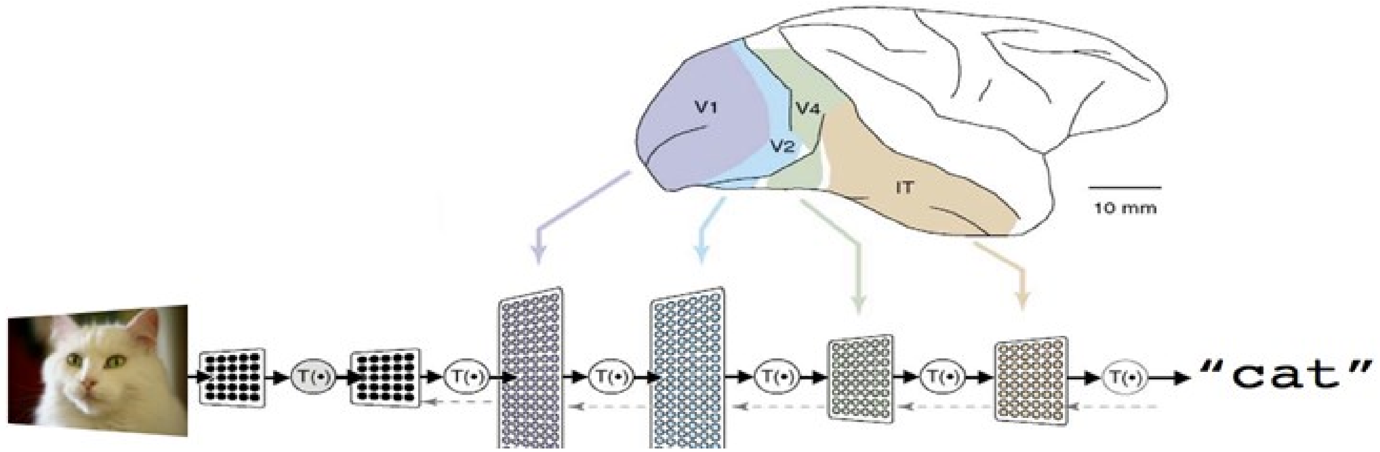


Figure 6.6: Empirical results showing that deeper networks generalize better when used to transcribe multi-digit numbers from photographs of addresses. Data from Goodfellow *et al.* (2014d). The test set accuracy consistently increases with increasing depth. See Fig. 6.7 for a control experiment demonstrating that other increases to the model size do not yield the same effect.



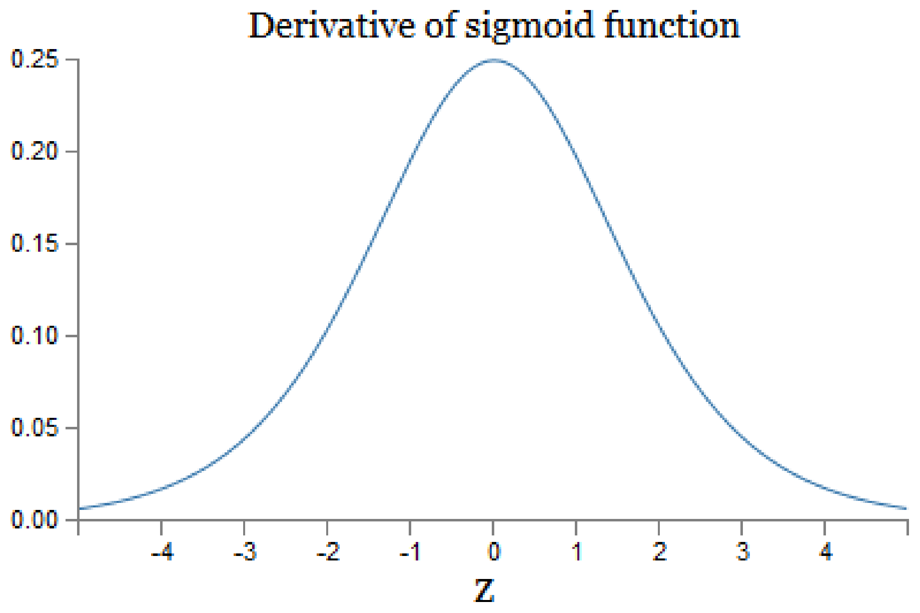
DNN Challenges

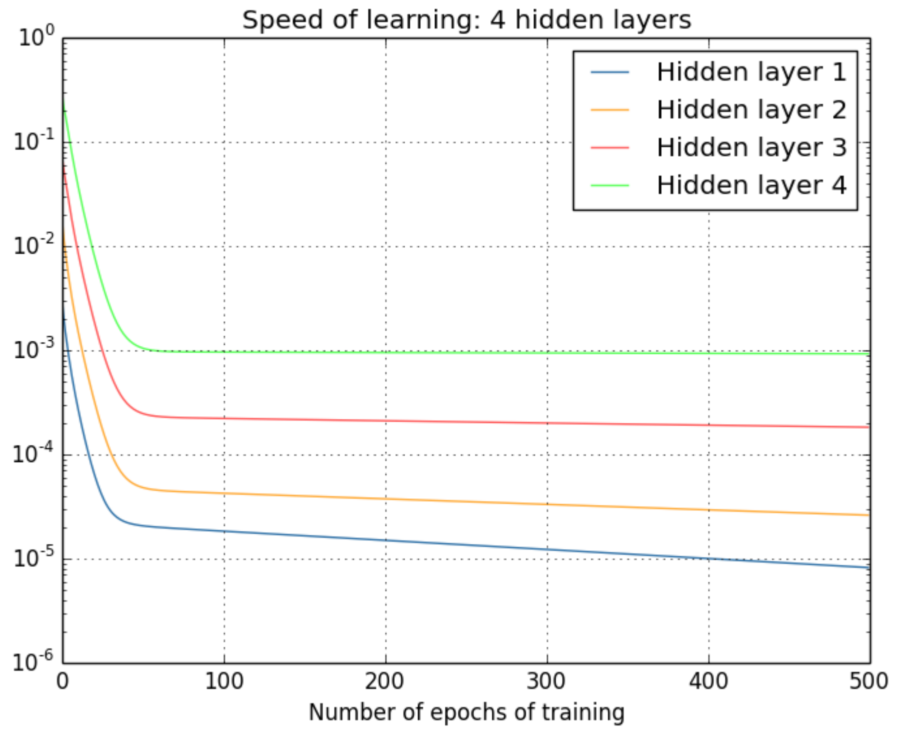
- Significant compute and infrastructure requirements
- Vanishing and exploding gradients
- Long train and test times
- Requirement for big labelled datasets
- Black-box/opaque approach



Vanishing and Exploding Gradients

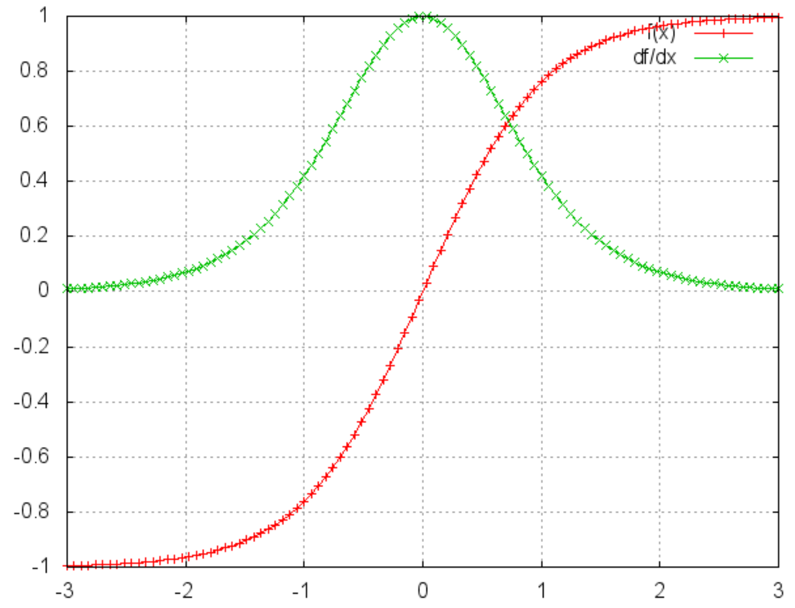
- Saturating nonlinearities (like tanh or sigmoid) can not be used for deep networks as they tend to get stuck in the saturation region as the network grows deeper

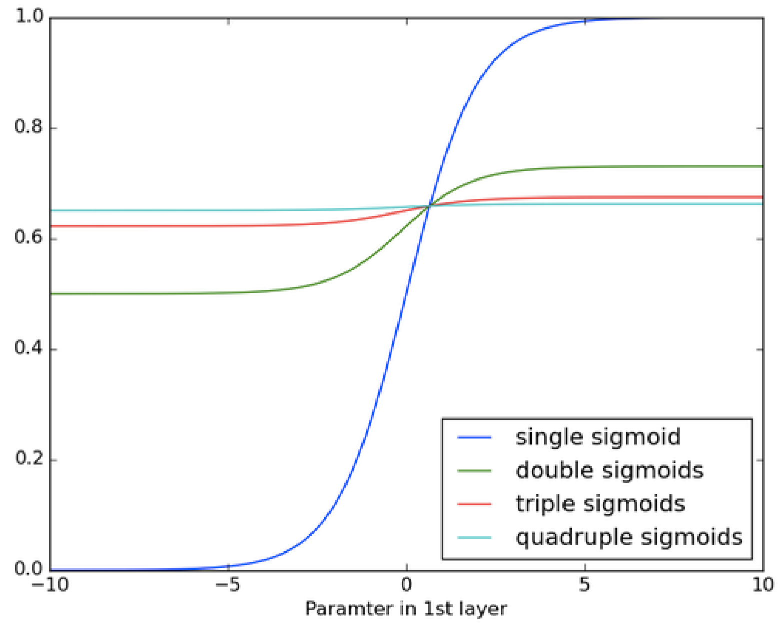




Vanishing Gradients

- Gradients for the lower layers (closer to the input) can become very small. With small values in the matrix and multiple matrix multiplications, the gradient values shrink exponentially fast, eventually vanishing completely
- The ReLU activation function can help prevent vanishing gradients

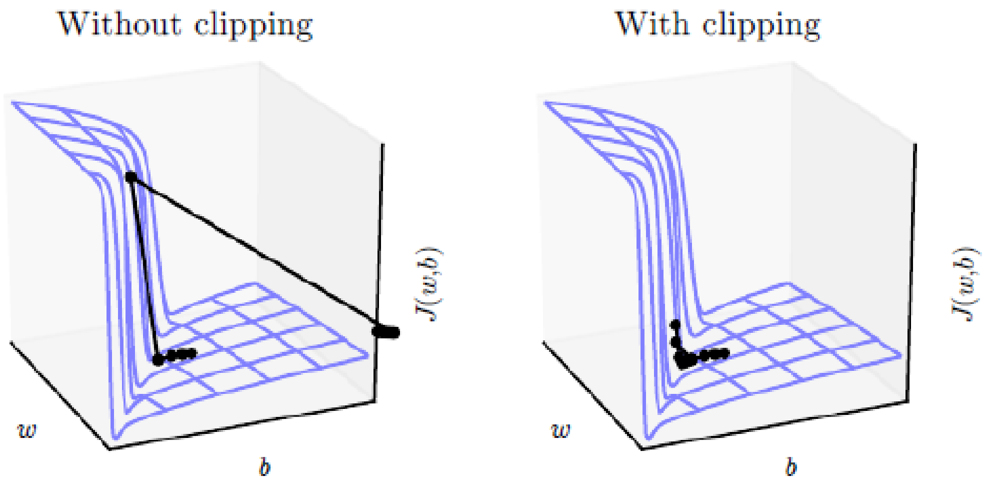




https://deeplearning4j.org/lstm.html

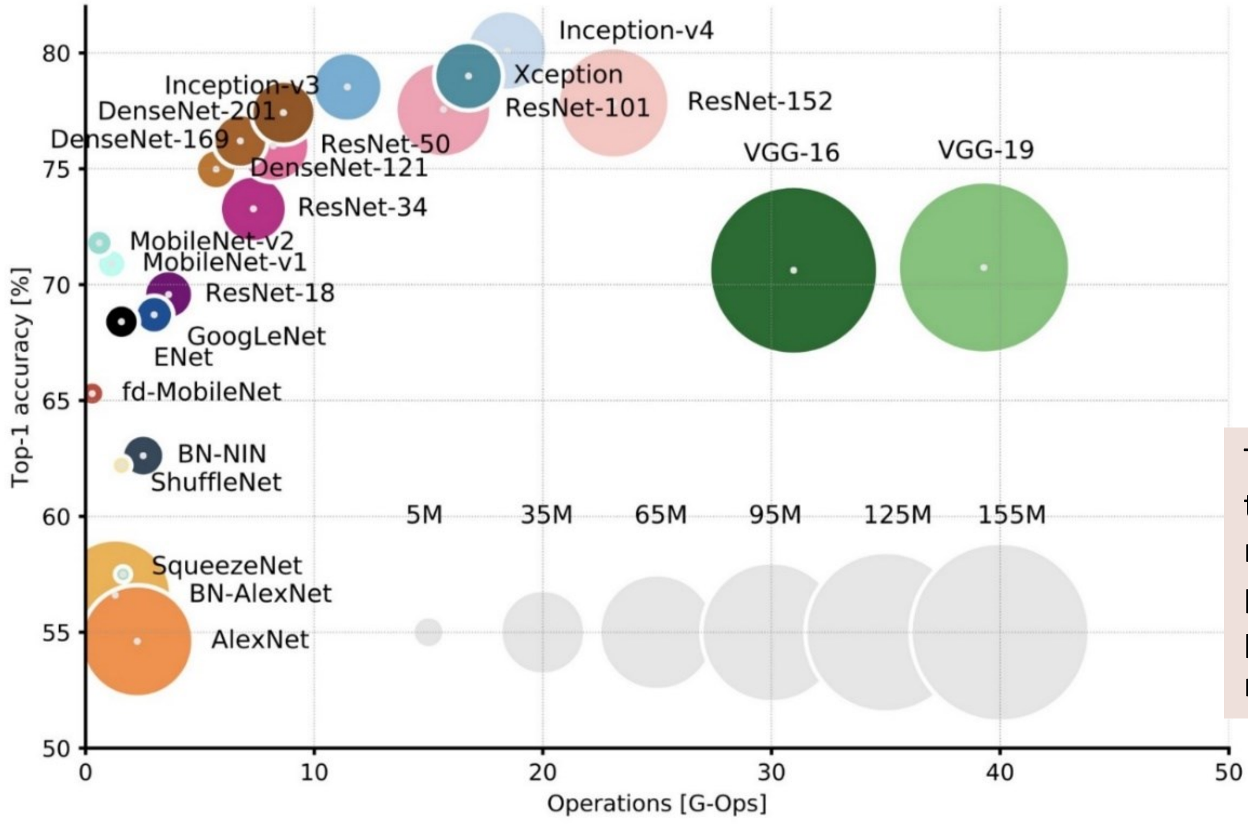
Exploding Gradient

- Solved relatively easily because gradients can be truncated or squashed:
  - Gradient Clipping
  - Weight Regularization



Overparameterization

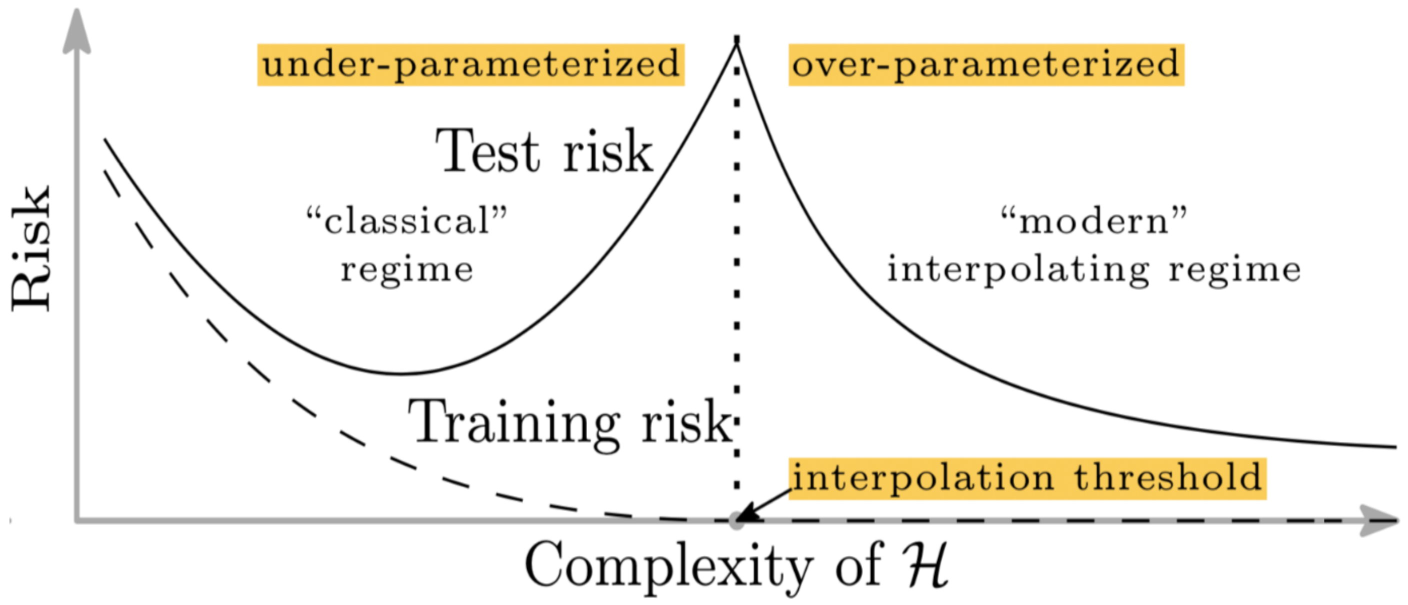
- Models where the number of parameters is significantly larger than the sample size
- Despite massive sizes, DNNs have exhibited remarkable generalization performance



Top-1 one-crop accuracy versus the number of operations required for a single forward pass. The size of the blobs is proportional to the number of network parameters

https://arxiv.org/abs/1605.07678

Double Descent



"Double descent" risk curve that extends the traditional U-shaped bias-variance curve beyond the point of interpolation

https://arxiv.org/abs/1812.11118

https://openai.com/research/deep-double-descent

DNN DESIGN

Preprocessing, Regularization,Hyperparamters

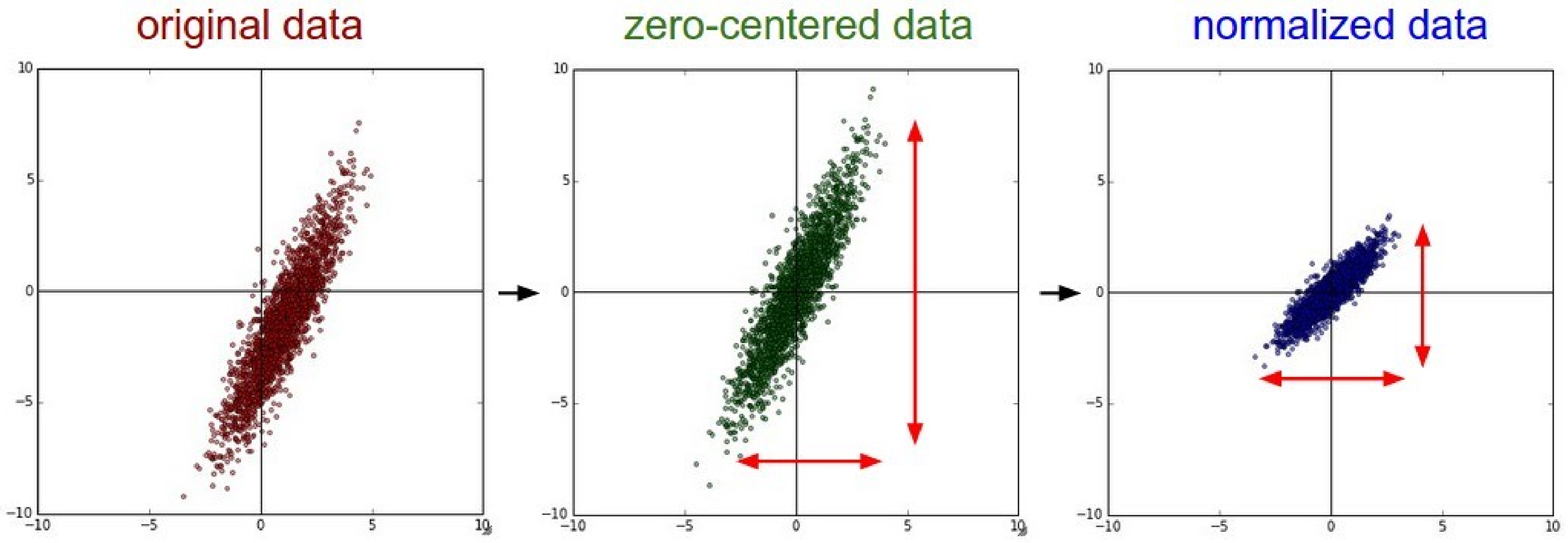
Data Preprocessing: Normalization

- Mean subtraction - centering the data around origin

```
X -= np.mean(X, axis = 0)
```

- Normalization - dimensions approximately at same scale

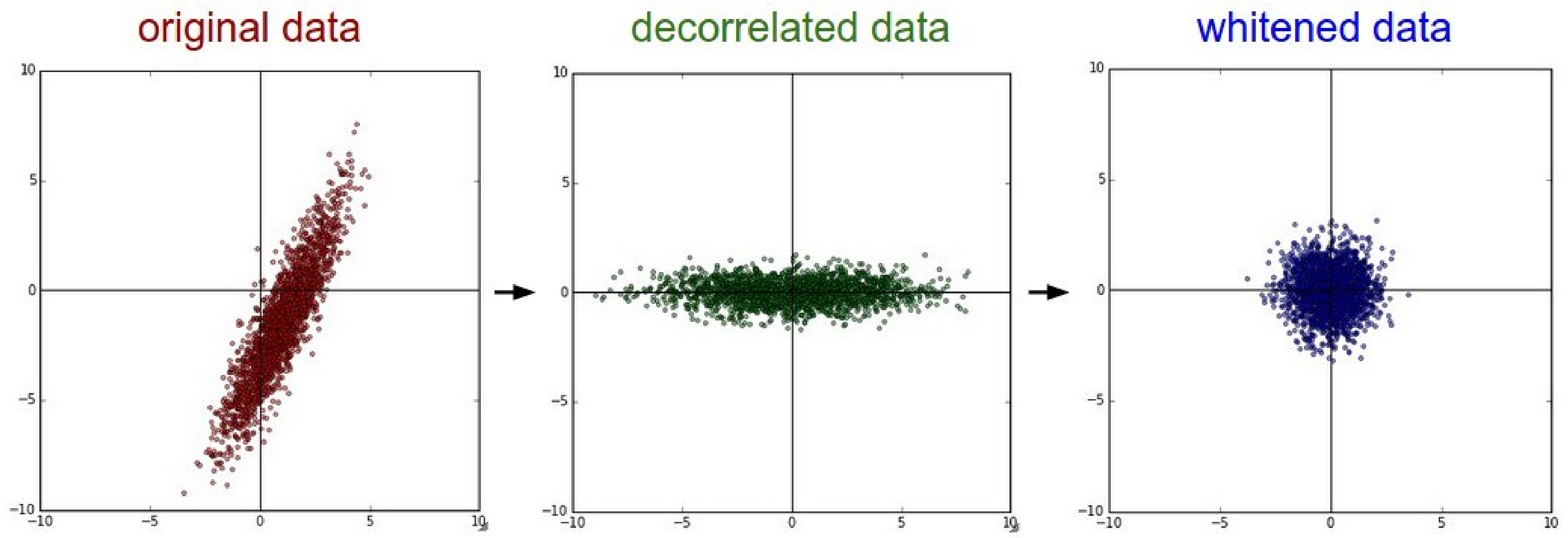
```
X /= np.std(X, axis = 0)
```



http://cs231n.github.io/neural-networks-2/

Data Preprocessing: PCA, Whitening

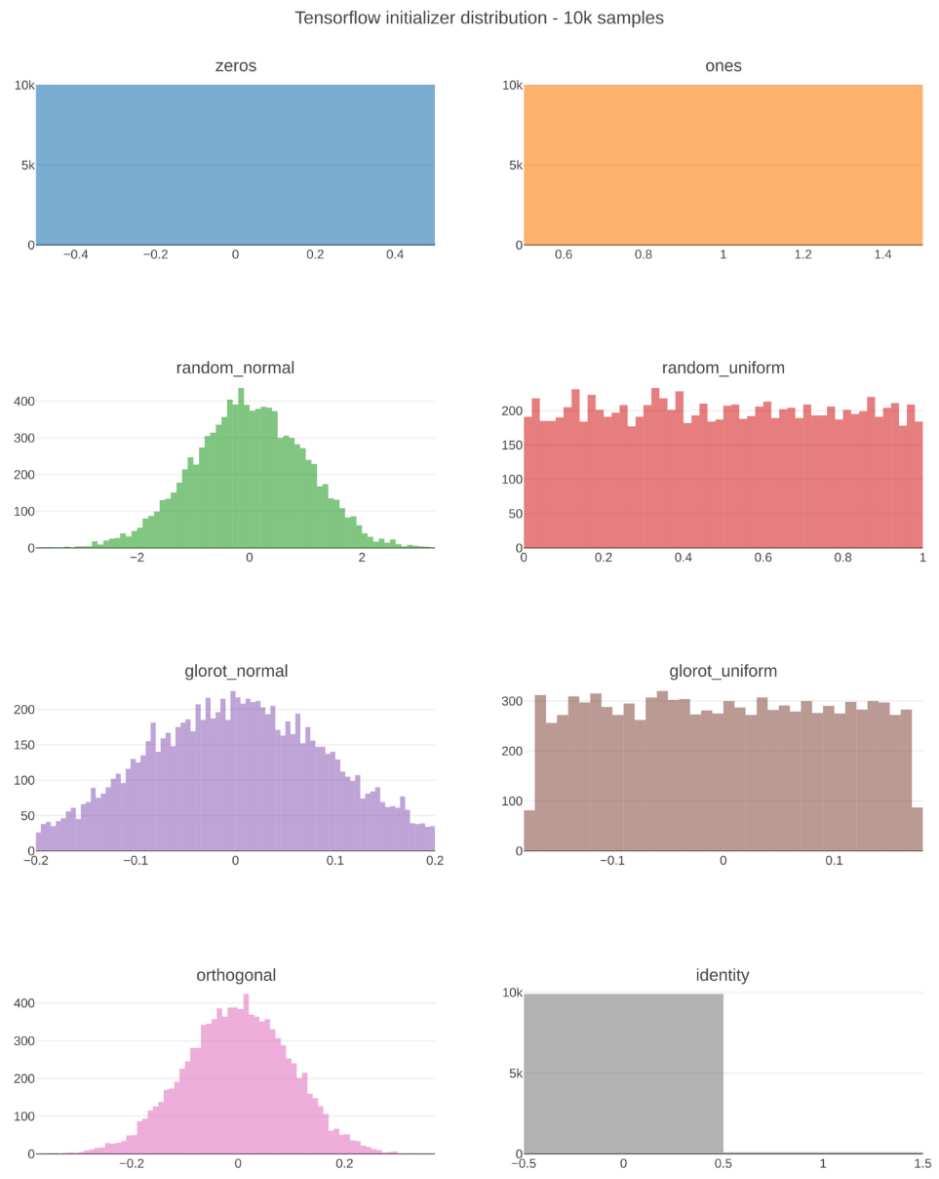
1. Center data , calculate co-variance matrix
2. SVD factorization, decorrelation
3. PCA - dimensionality reduction
4. Whitening - stretching data into an isotropic gaussian blob to normalize the scale



http://cs231n.github.io/neural-networks-2/

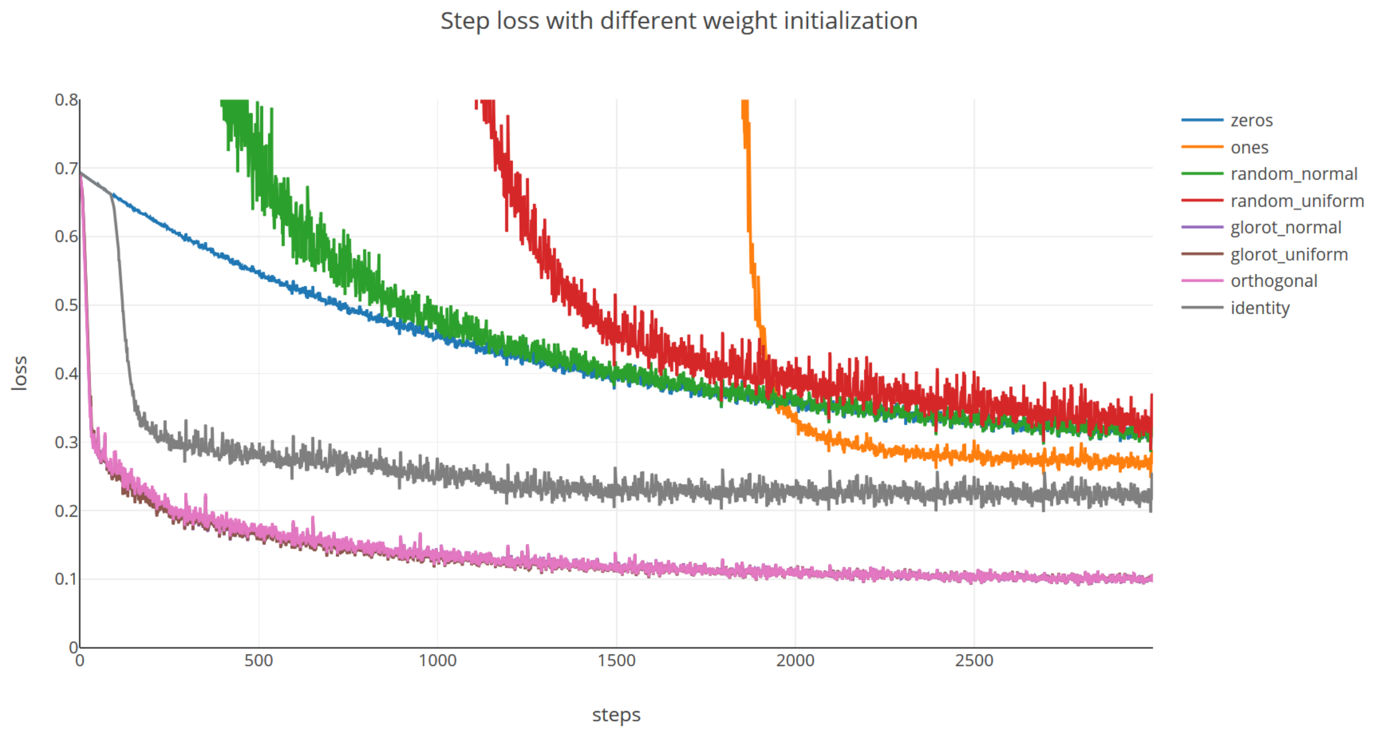
Weight Initialization

- Normalized Data
  - Weights expected to be centered at 0, small positive/negative
  - Asymmetry between neurons facilitates learning
  - Same valued or 0 weights very hard to learn.
- Random Initialization
  - From Gaussian or Uniform distribution



Weight Initialization: Glorot

- Allows for scaling the weight distribution on a layer-by-layer basis.
- Draws from a normal or uniform distribution with centered mean and standard deviation scaled to the layer's number of input and output neurons



https://becominghuman.ai/priming-neural-networks-with-an-appropriate-initializer-7b163990ead

Regularization

- Prevents overfitting by modifying the loss function by adding additional terms that penalize large weights
- L2 regularization is usually preferred over L1 due to empirical performance results

L1: $R(\theta) = \|\theta\|_1 = \sum_{i=1}^{n} |\theta_i|$

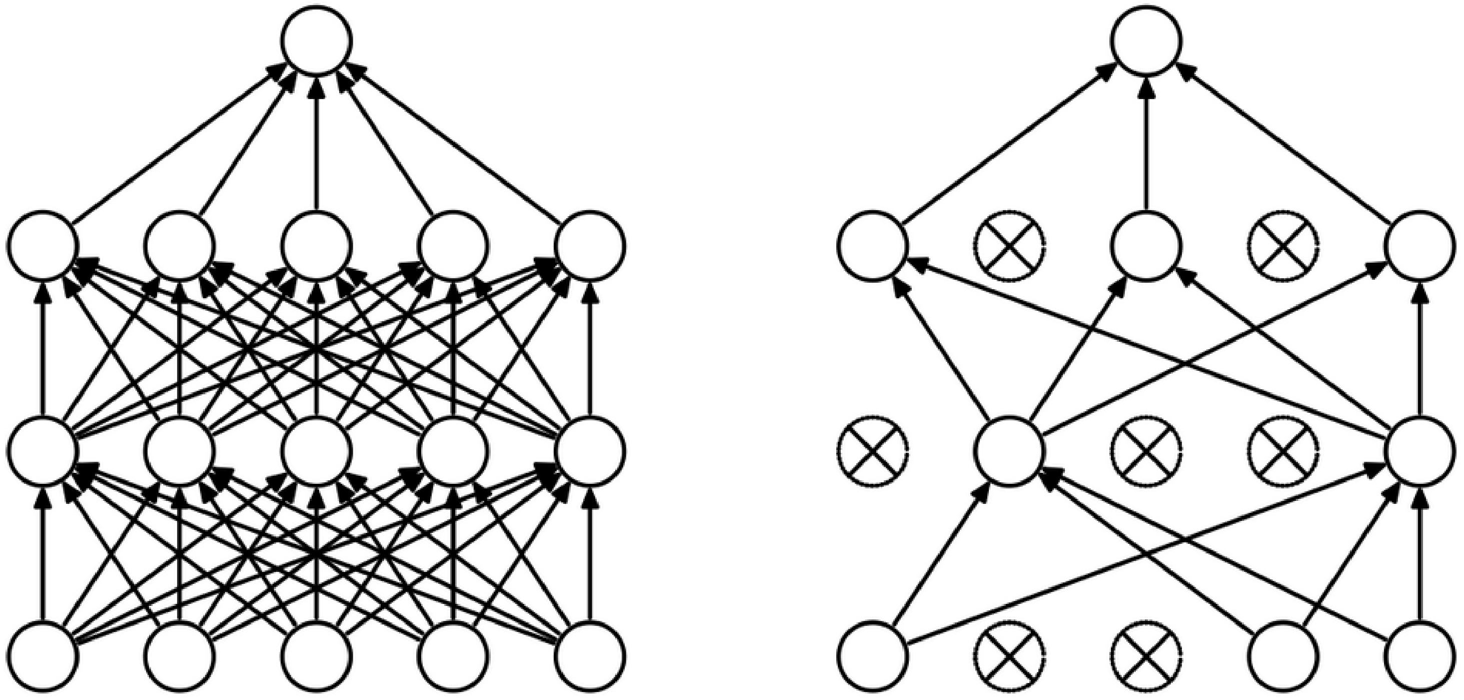
Error function + magnitude of all weights in the neural network

L2: $R(\theta) = \|\theta\|_2^2 = \sum_{i=1}^{n} \theta_i^2$

Error function + squared magnitude of all weights in the neural network

Dropout

- Randomly drop weights from a layer such that at each layer neurons are forced to learn the multiple characteristics of the network

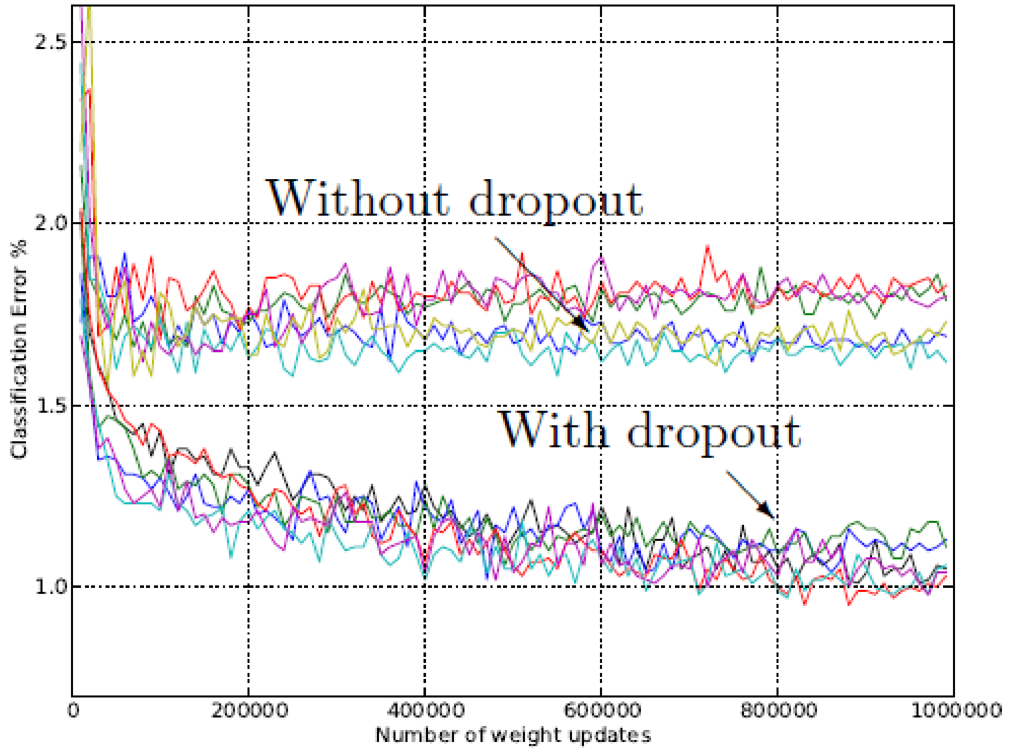


(a) Standard Neural Net (b) After applying dropout.

https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

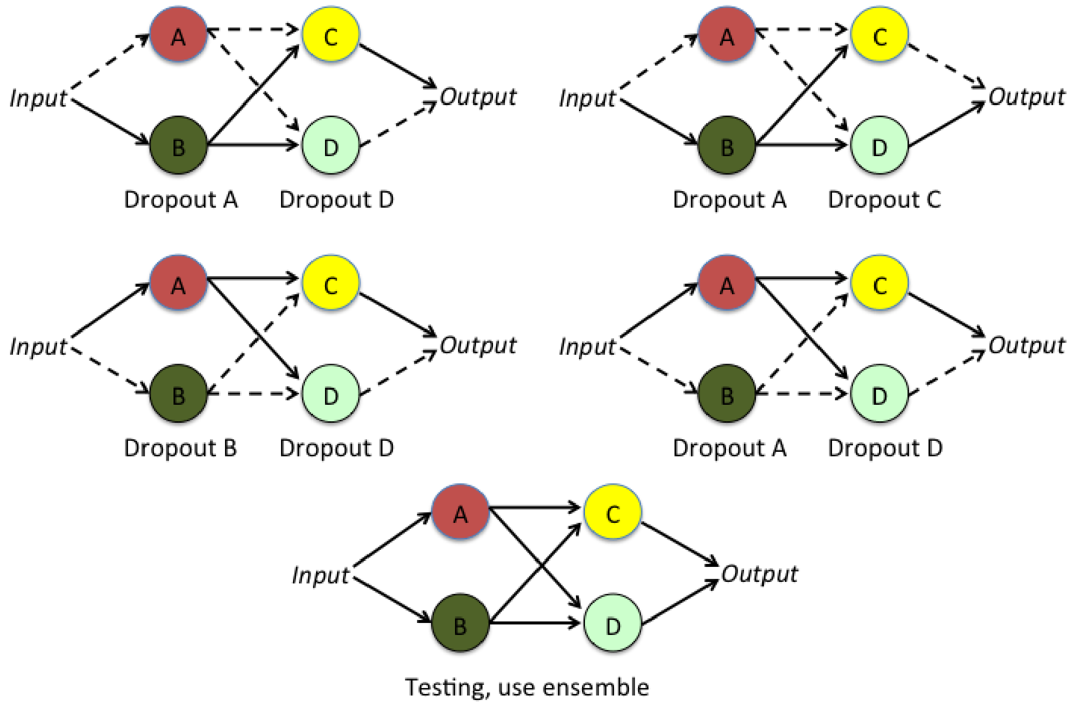
Dropout

- Prevents overfitting by combining exponentially many different ANN architectures efficiently



Dropout: Ensemble Effect

- Effect of taking ensemble over $({}^{N}C_{\frac{N}{2}})^{\wedge}h$ models.



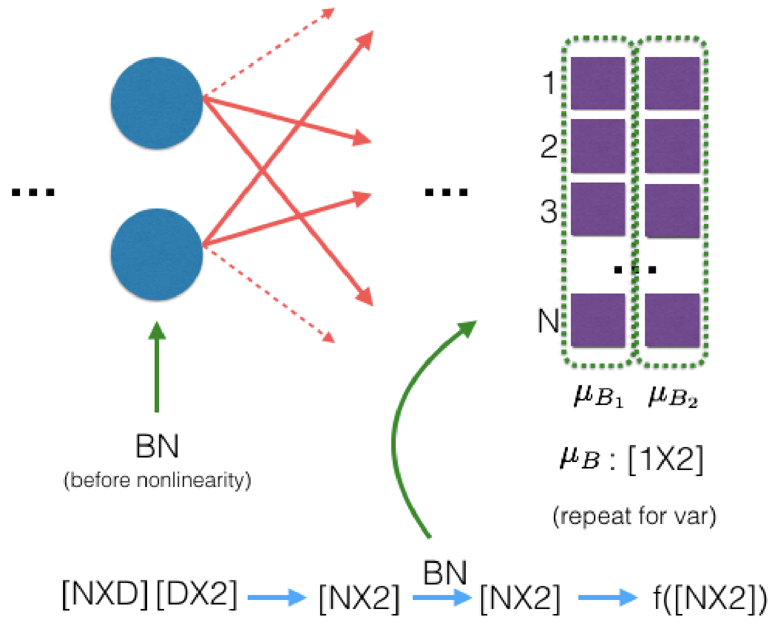
2 layers and 4 neurons in each layer, $^4C_2$ X $^4C_2$ = 36 ; takes average over 36 models.

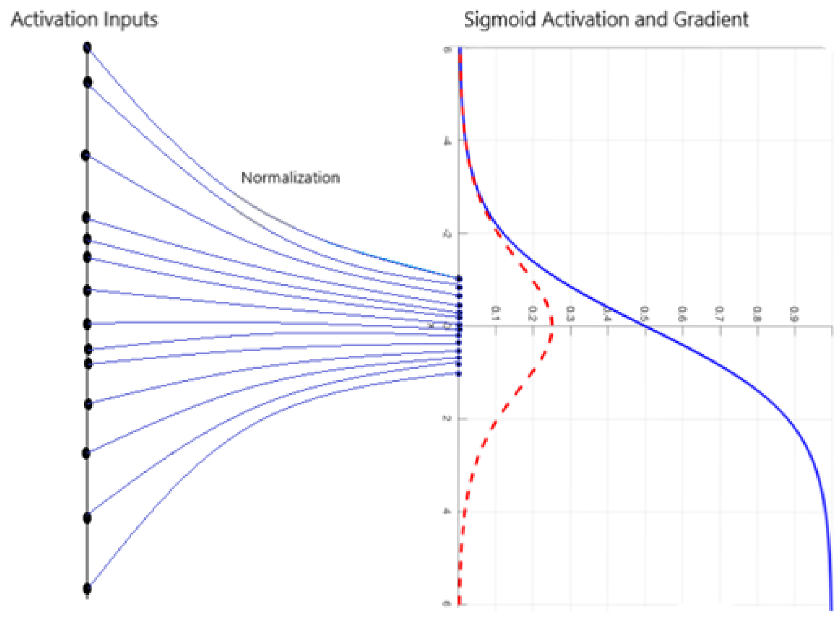
2 layers with 100 neurons in each layer, takes average over 24502500 possible models

https://medium.com/@vivek.yadav/why-dropouts-prevent-overfitting-in-deep-neural-networks-937e2543a701

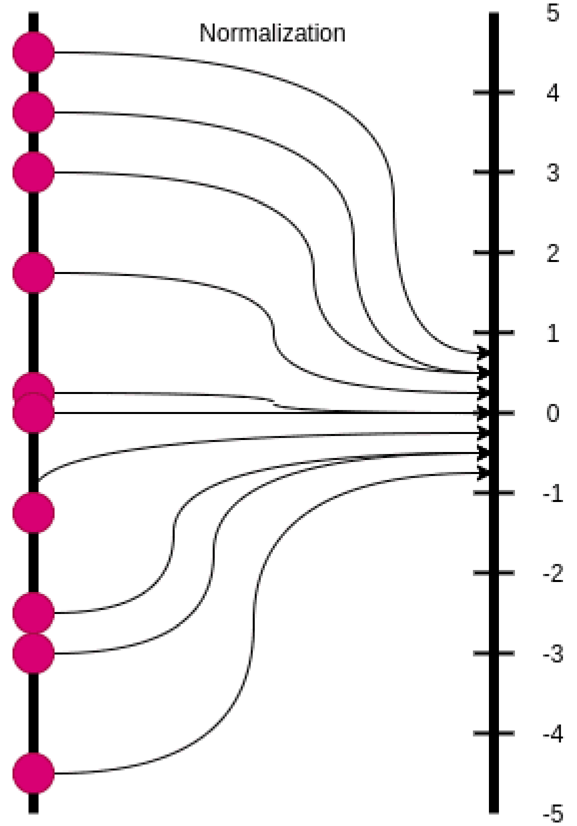
Batch Normalization

- Method to normalize the inputs of each layer, in order to fight the internal covariate shift problem.
- Reduces vanishing gradient during training and can decrease training time and result in better performance





Batch Normalization



Input

$$\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i$$ Mini-batch mean

$$\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2$$ Mini-batch variance

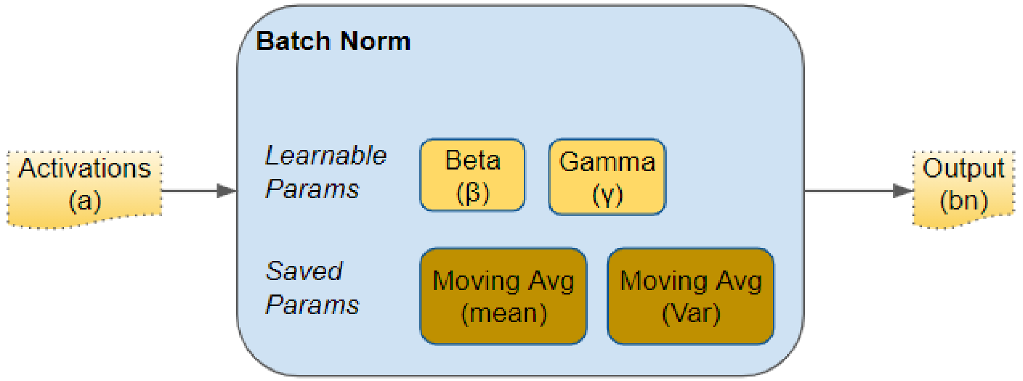
$$\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$ Normalize

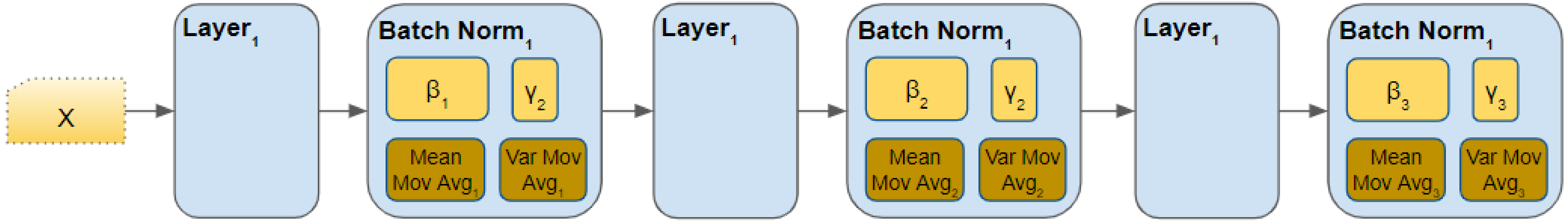
$$y_i = \gamma \hat{x}_i + \beta = \mathrm{BN}_{\gamma,\beta}(x_i)$$ Scale and Shift

Output

Batch Normalization: Parameters

- BN layer has parameters of its own:
  - Two learnable parameters beta and gamma.
  - Two non-learnable parameters (Mean Moving Average and Variance Moving Average) are saved as part of the 'state' of the Batch Norm layer.

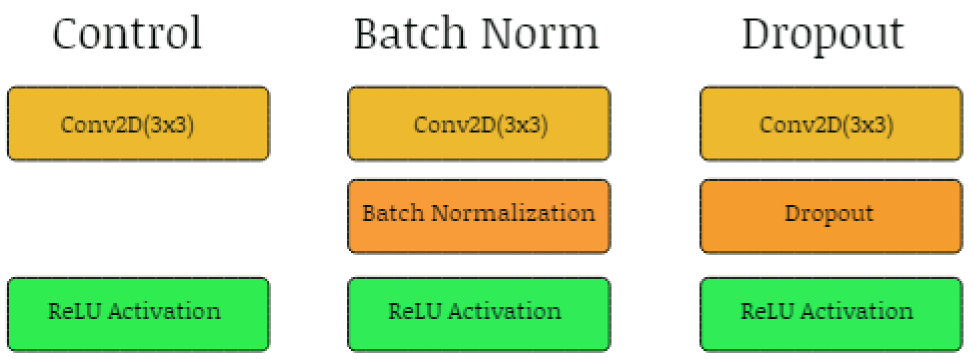


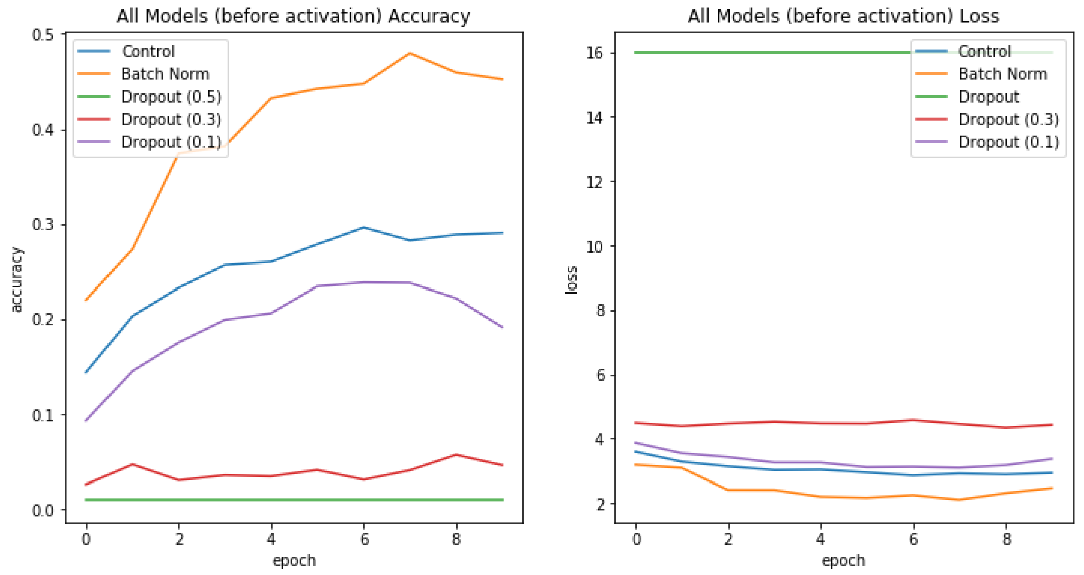


3 hidden layers and 3 BN layers in a network, would have three learnable beta and gamma parameters for the three layers

https://towardsdatascience.com/batch-norm-explained-visually-how-it-works-and-why-neural-networks-need-it-b18919692739

Dropout vs Batch Normalization





https://towardsdatascience.com/dont-use-dropout-in-convolutional-networks-81486c823c16

Hyperparameter Tuning

Architecture

- Architecture
- No of hidden layers
- No of hidden units in each layer
- Activation function
- Batch Normalization
- Dropout

Optimization

- Weights Initialization
- Learning Rate
- Loss Function
- Batch size
- Momentum
- Number of epochs

Hyperparameters - Optimizers, Learning Rate, Gradient

Optimizers

- `tf.train.GradientDescentOptimizer`
- `tf.train.AdadeltaOptimizer`
- `tf.train.AdagradOptimizer`
- `tf.train.AdagradDAOptimizer`
- `tf.train.MomentumOptimizer`
- `tf.train.AdamOptimizer`
- `tf.train.FtrlOptimizer`
- `tf.train.ProximalGradientDescentOptimizer`
- `tf.train.ProximalAdagradOptimizer`
- `tf.train.RMSPropOptimizer`

Decaying the learning rate

- `tf.train.exponential_decay`
- `tf.train.inverse_time_decay`
- `tf.train.natural_exp_decay`
- `tf.train.piecewise_constant`
- `tf.train.polynomial_decay`

Gradient Clipping

- `tf.clip_by_value`
- `tf.clip_by_norm`
- `tf.clip_by_average_norm`
- `tf.clip_by_global_norm`
- `tf.global_norm`

https://www.tensorflow.org/api_guides/python/train#Optimizers

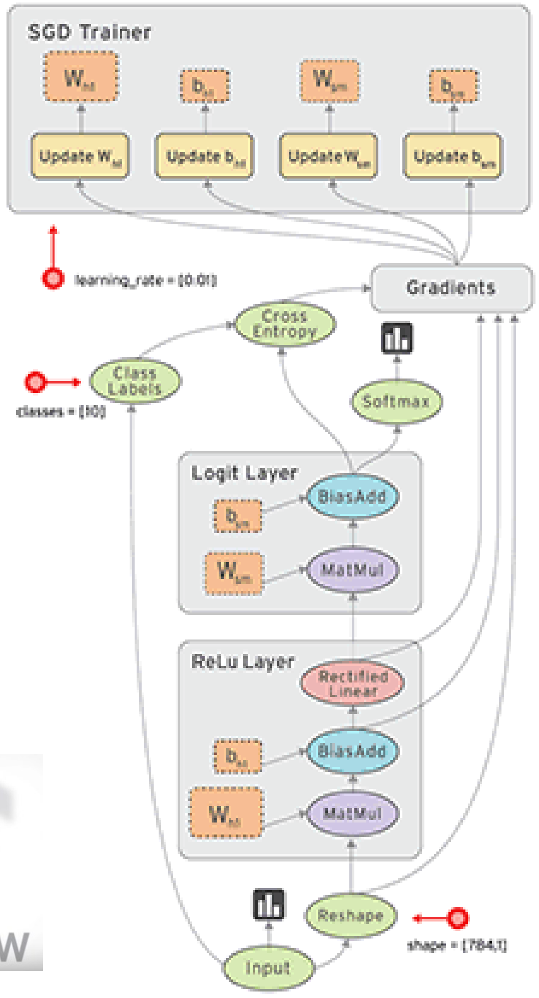
Best Practices

- Normalize all data and/or use batch normalization
- Use small initial random weights
- Reduce learning rate when weights oscillate
- Keep derivatives from going to zero by choosing non-saturating activations
- Use momentum to speed learning
- Use mini-batches for stabilizing gradients
- Avoid overfitting - L2 regularization, Dropouts, Early stopping of training
- Use neural network ensembles

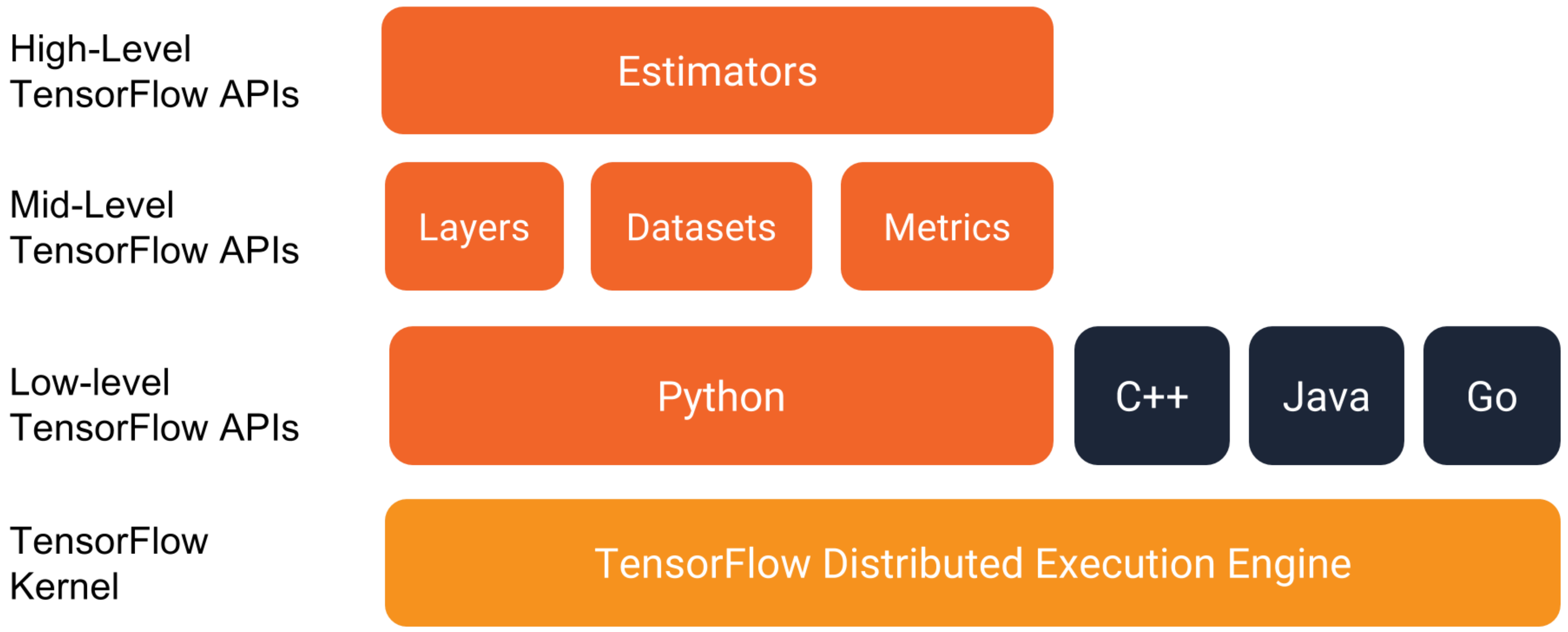
TensorFlow

TensorFlow

- TensorFlow expresses a numeric computation as a directed graph
- Tensor is the central unit of data which consists of a set of primitive values shaped into a multi-dimensional array
- A tensor's rank is its number of dimensions, while its shape is a tuple of integers specifying the array's length along each dimension.

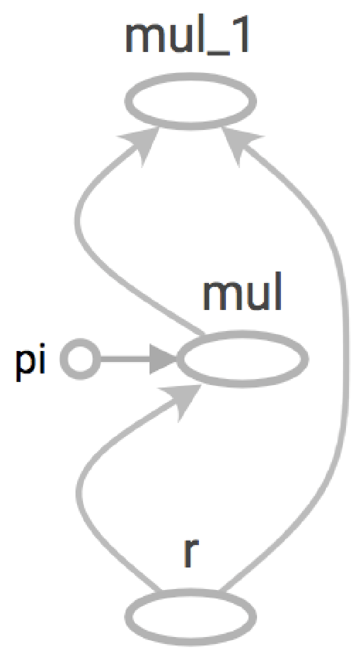


TensorFlow Architecture


High-Level TensorFlow APIs
Estimators
Mid-Level TensorFlow APIs
Layers
Datasets
Metrics
Low-level TensorFlow APIs
Python
C++
Java
Go
TensorFlow Kernel
TensorFlow Distributed Execution Engine

Graph

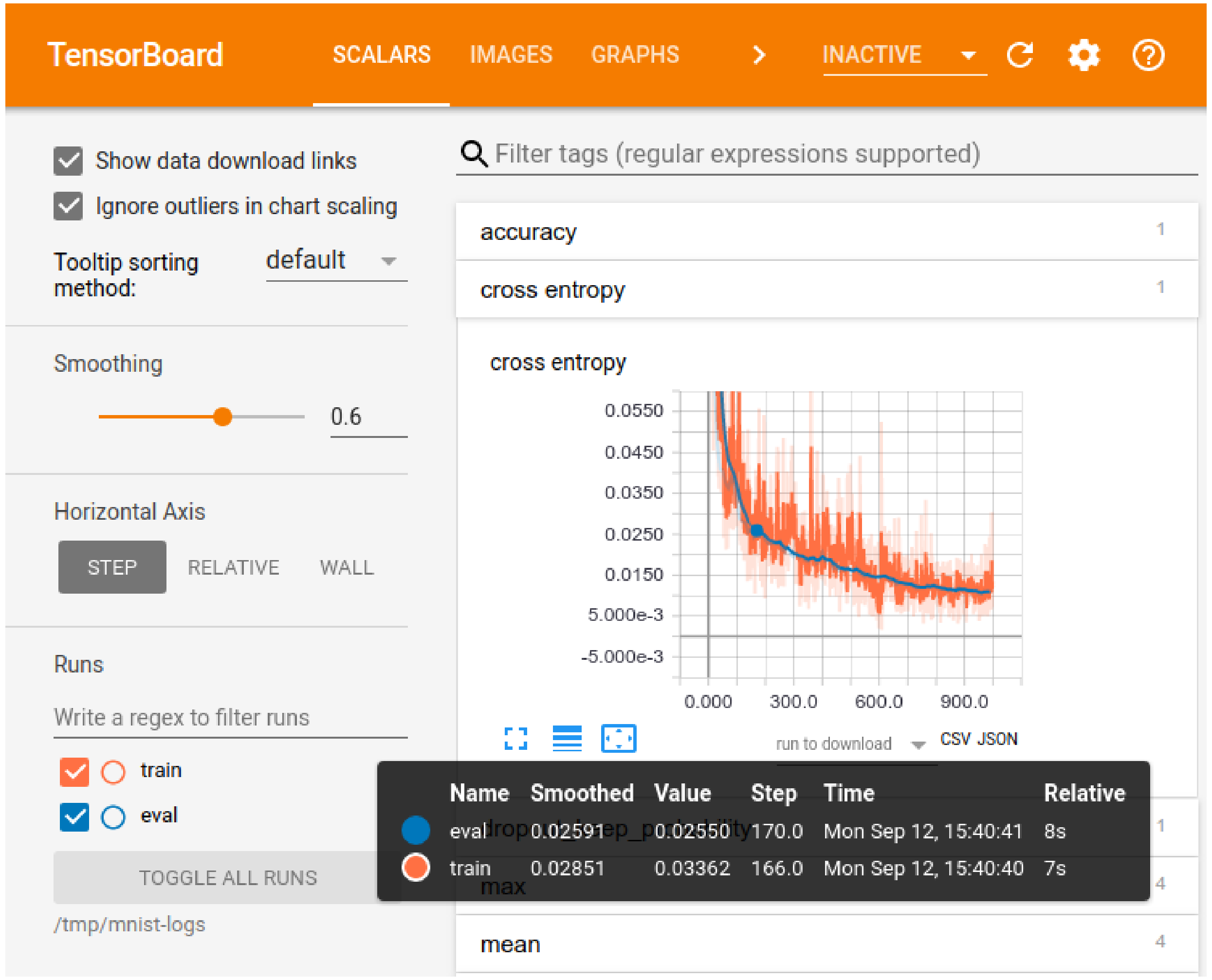
- A computational graph is a series of TensorFlow operations arranged into a graph.
- The graph is composed of two types of objects:
  - Operations (or "ops"): The nodes of the graph. Operations describe calculations that consume and produce tensors.
  - Tensors : The edges in the graph. These represent the values that will flow through the graph. Most TensorFlow functions return tf.Tensors. tf.Tensors do not have values, they are handles to elements in the computation graph.



Advantages of Graphs

- Parallelism
- Distributed execution
- Compilation
- Portability

TensorBoard



- [✓] Show data download links
- [✓] Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing: 0.6

Horizontal Axis: [✓ STEP] RELATIVE WALL

Runs

Write a regex to filter runs

- [✓] train
- [✓] eval

TOGGLE ALL RUNS

/tmp/mnist-logs

Filter tags (regular expressions supported)

accuracy	1
cross entropy	1

cross entropy

run to download CSV JSON

Name	Smoothed	Value	Step	Time	Relative
eval	0.02591	0.02550	170.0	Mon Sep 12, 15:40:41	8s
train	0.02851	0.03362	166.0	Mon Sep 12, 15:40:40	7s

max	4
mean	4