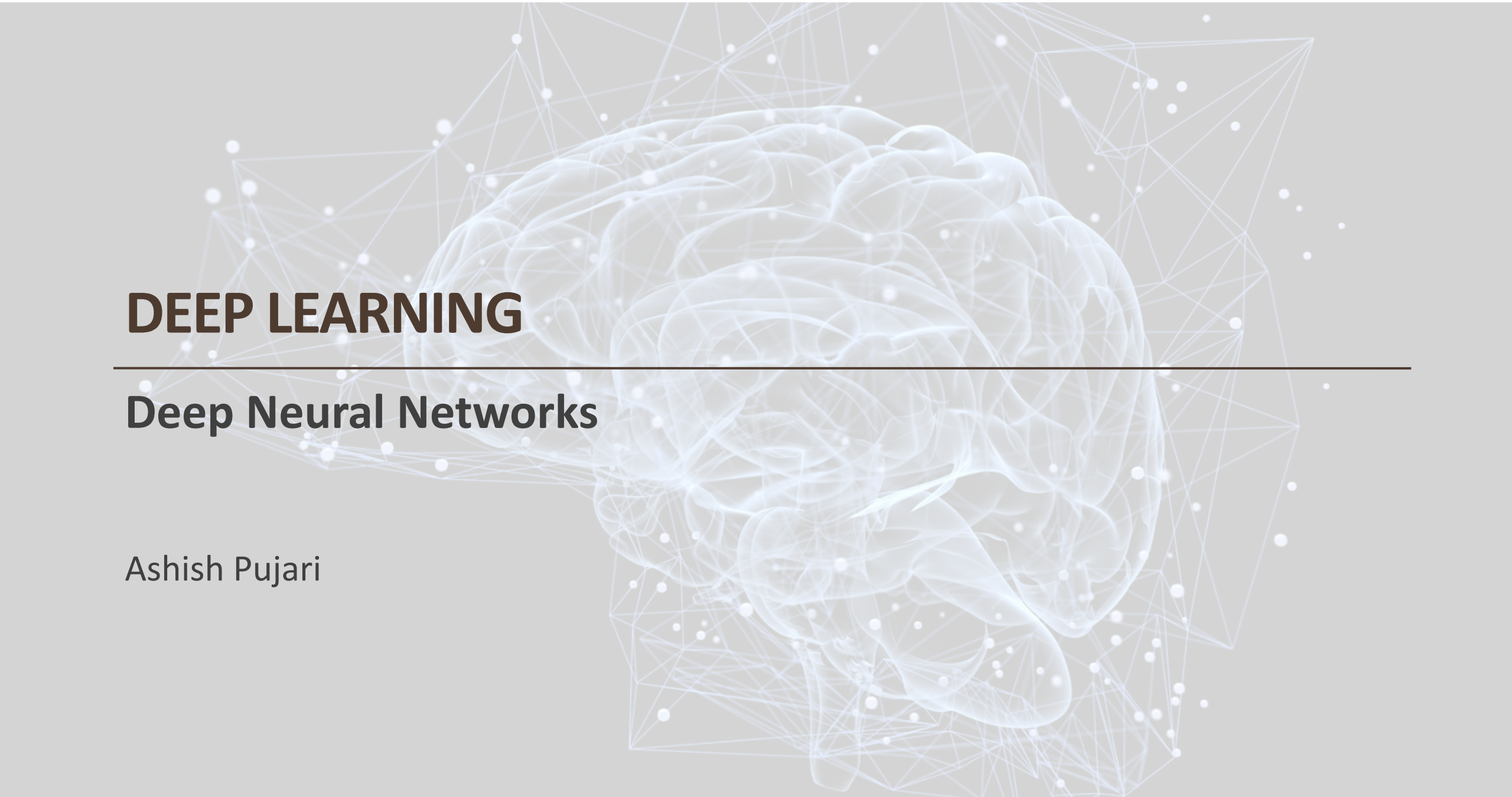


DEEP LEARNING

Deep Neural Networks

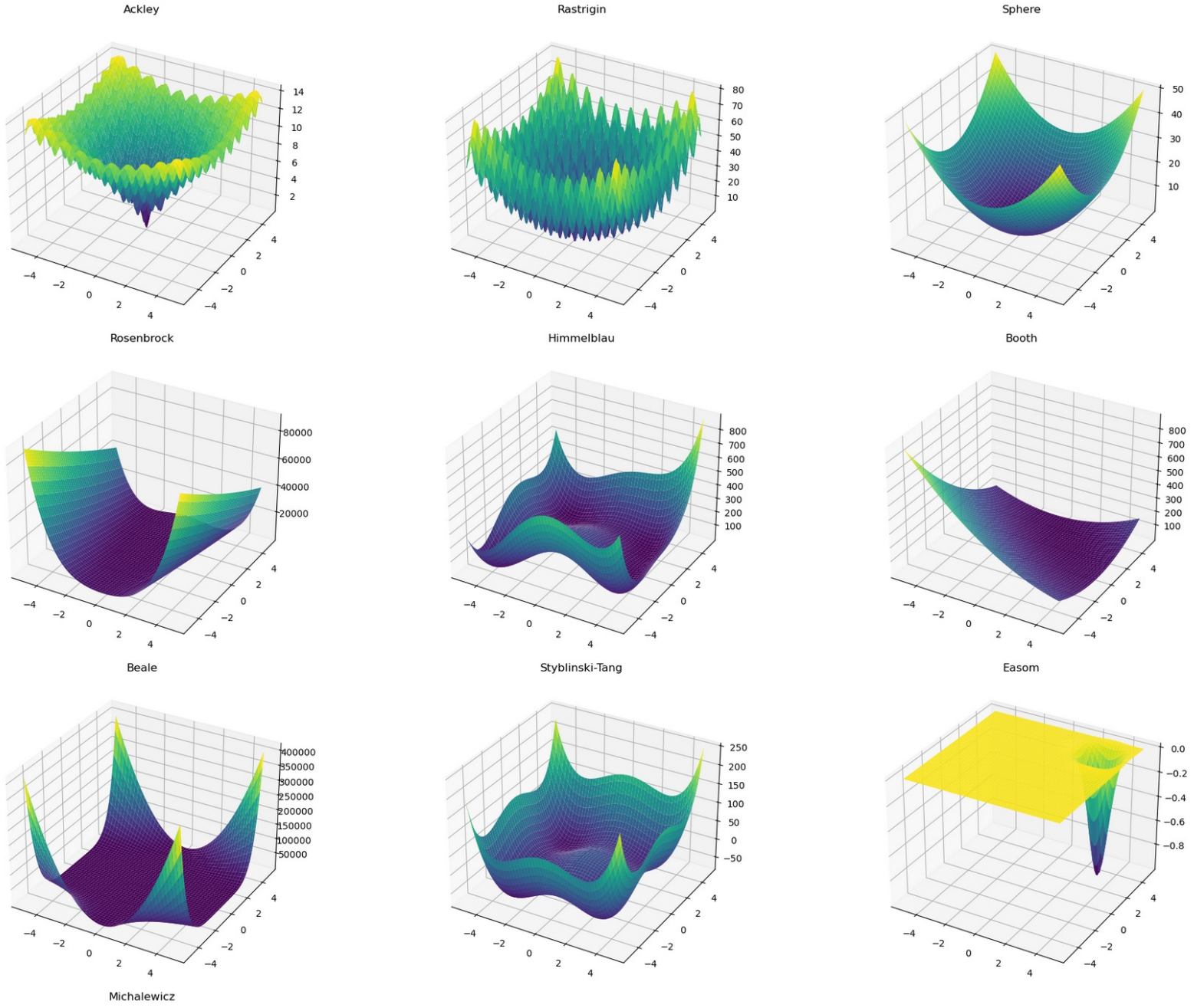
Ashish Pujari

Lecture Outline

1. SGD Variations
2. Deep Neural Networks
3. DNN Design

SGD VARIATIONS

Loss Functions



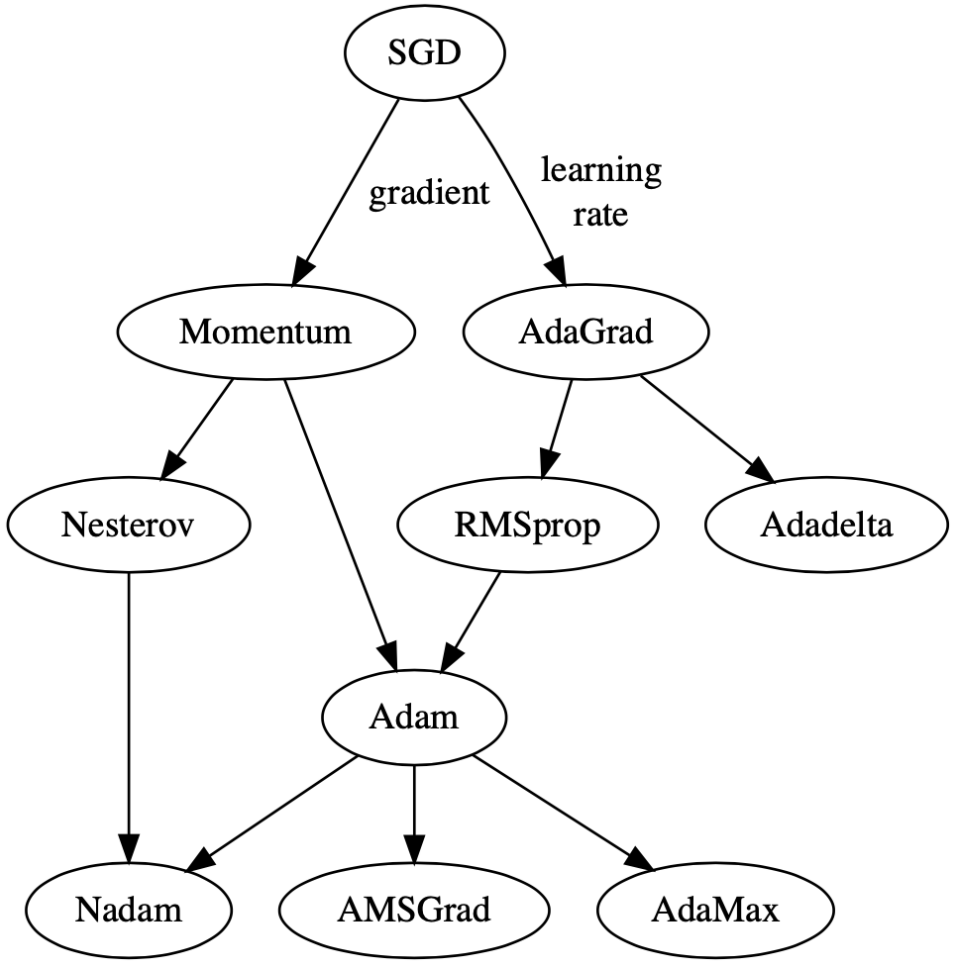
Momentum Based Algorithms

- First-Order Moment (Mean):
  - Average of past gradients m_t (estimate of the mean gradient vector)
  - Helps in determining the direction of the update
  - Used by : Momentum, Nesterov, Adam

- Second-Order Moment (Uncentered Variance):
  - Uncentered variance of past gradients v_t (estimate of the variance of the gradient vector)
  - Helps in scaling the step size of the update
  - Used by : AdaGrad, AdaDelta, RMSProp, Adam

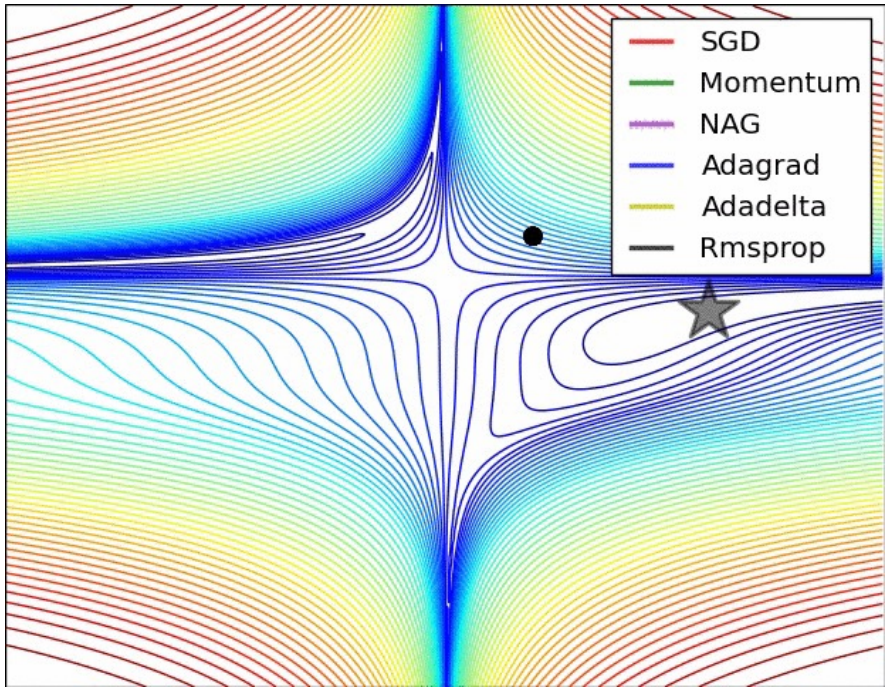
SGD: Variants

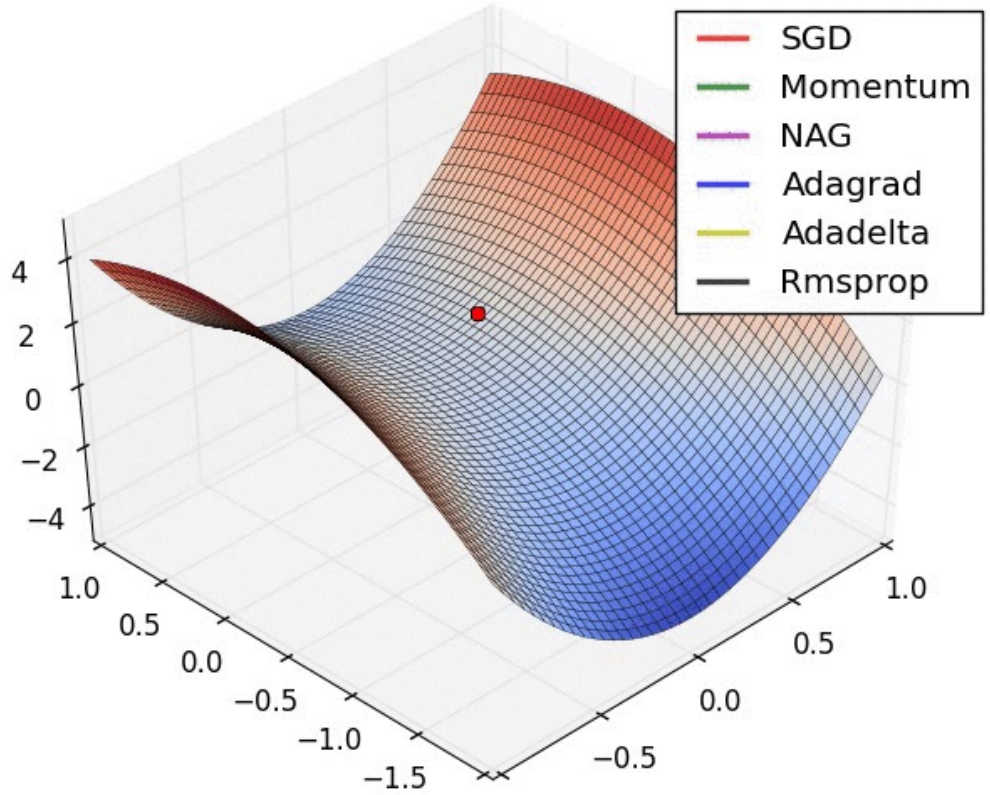
Optimiser	Year	Learning Rate	Gradient
Momentum	1964		✓
AdaGrad	2011	✓	
RMSprop	2012	✓	
Adadelta	2012	✓	
Nesterov	2013		✓
Adam	2014	✓	✓
AdaMax	2015	✓	✓
Nadam	2015	✓	✓
AMSGrad	2018	✓	✓



https://towardsdatascience.com/10-gradient-descent-optimisation-algorithms-86989510b5e9

SGD: Variants





Alec Radford

Optimizing Gradient Descent - http://ruder.io/optimizing-gradient-descent/

RMS-Prop (Root Mean Square Propagation) (2012)

- Compute square gradients for each parameter

$$g_t{}^2 = (\nabla J(w)^2)$$

- Squared Gradients Exponential Moving Average

$$E[g^2] = \beta \cdot E[g^2] + (1 - \beta) \cdot g_t{}^2$$

- Parameter Update

$$w_t = w_{t-1} - \frac{\eta}{\sqrt{E[g^2] + \epsilon}} \cdot \nabla J(w)$$

  - where:
    - w: Parameter being updated
    - η: Learning rate
    - ϵ: Small constant added to the denominator to avoid division by zero.
    - $E[g^2]$: EWMA of the squared gradients. Represents historical information about the gradients.
    - β : Decay rate parameter (usually close to 1).
    - $\nabla J(w)$: Gradient of the loss function JJ with respect to the parameter w.

https://optimization.cbe.cornell.edu/index.php?title=RMSProp

Adam (Adaptive Moment Estimation) (2014)

- Exponentially Weighted Averages of Gradients and Squared Gradients

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) {g_t}^2$$

- Bias Correction

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \quad , \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

- Parameter Update

$$w_t = w_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

- where
  - m: First moment estimation (mean) of the gradients.
  - v: Second moment estimation (uncentered variance) of the gradients.
  - ϵ is a small number,
  - β_1, β_2 : Decay rate parameters (usually close to 1).

https://arxiv.org/pdf/1412.6980v9.pdf

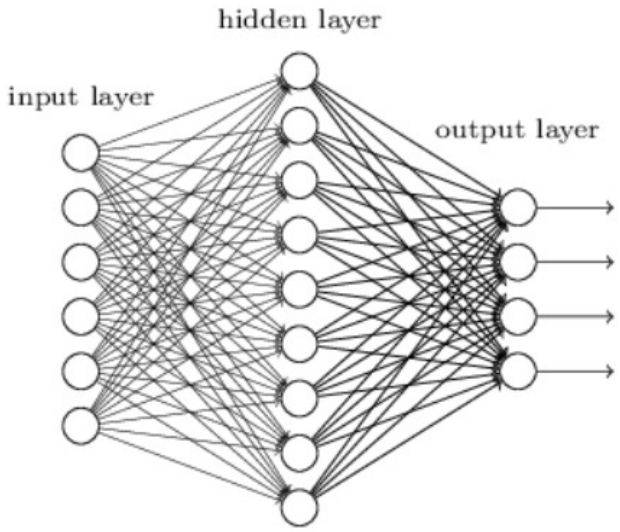
SGD: Best Practices

- Normalize input features.
- Select appropriate mini-batch size.
- Use proper initialization and regularization techniques.
- Implement learning rate scheduling.
- Rapid prototyping: Use adaptive techniques like Adam
- Best results: Use vanilla gradient descent with momentum; slower to converge

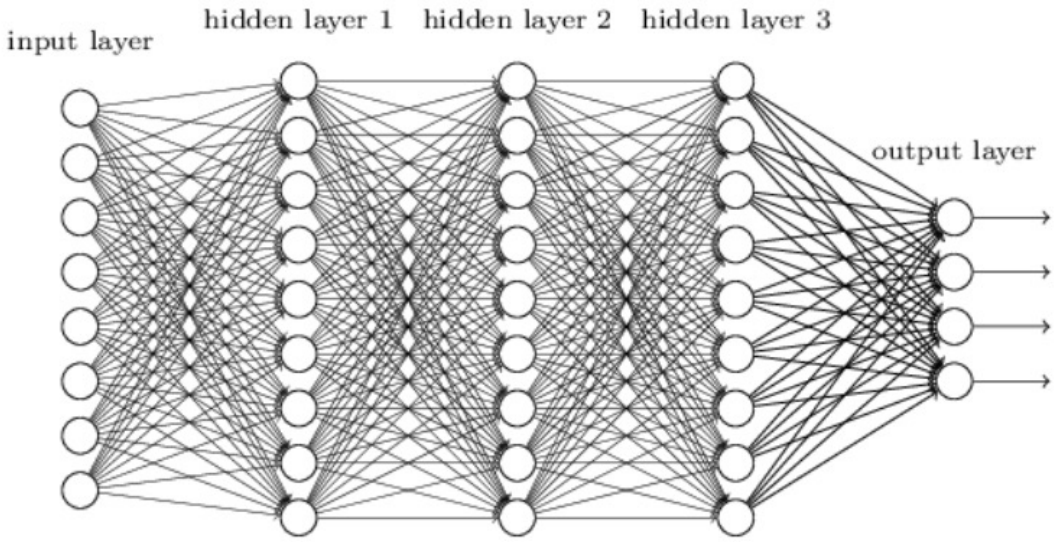
DNN

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



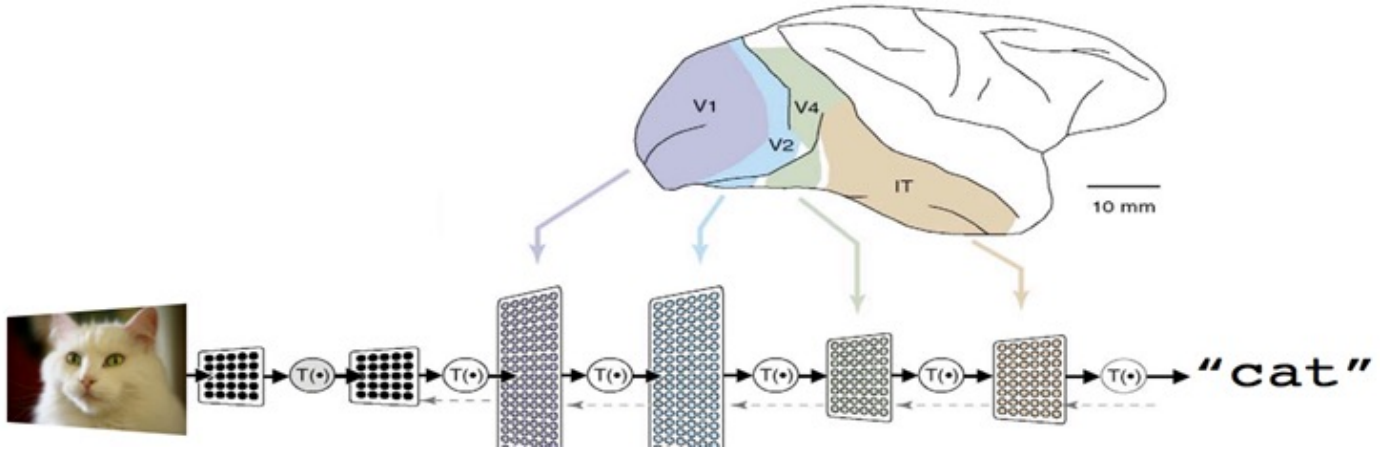
Artificial Neural Network



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

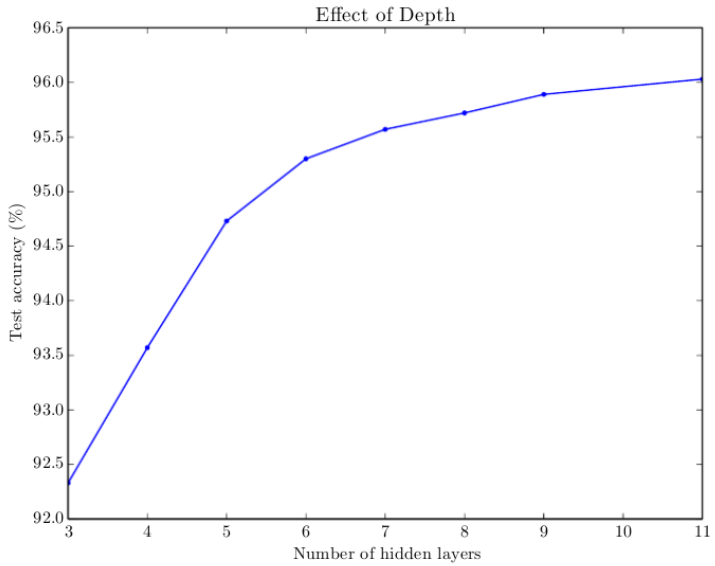
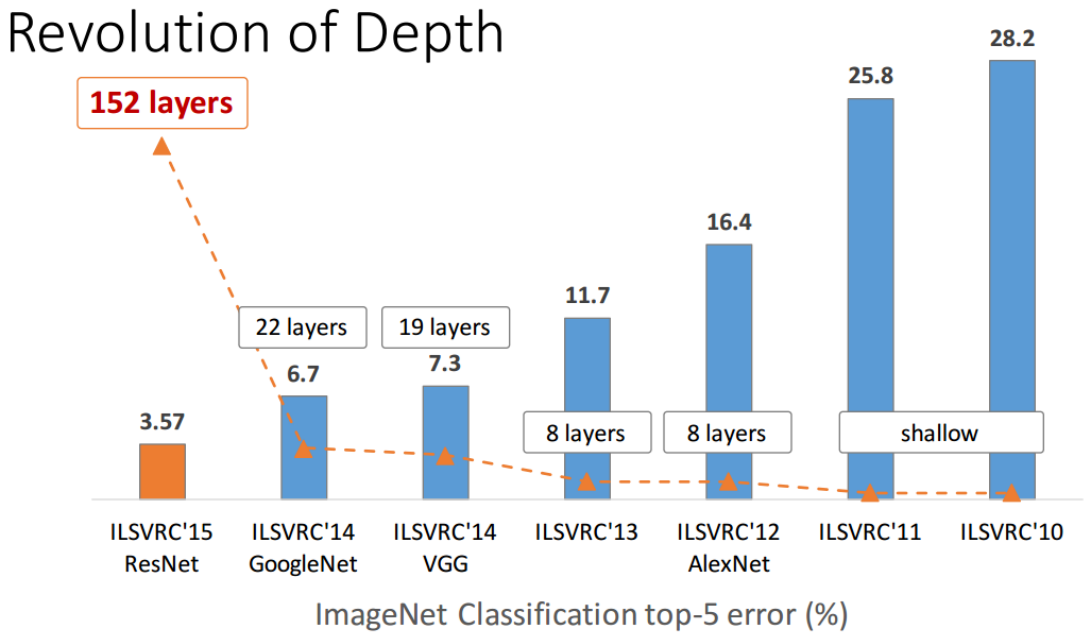


Figure 6.6: Empirical results showing that deeper networks generalize better when used to transcribe multi-digit numbers from photographs of addresses. Data from Goodfellow *et al.* (2014d). The test set accuracy consistently increases with increasing depth. See Fig. 6.7 for a control experiment demonstrating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

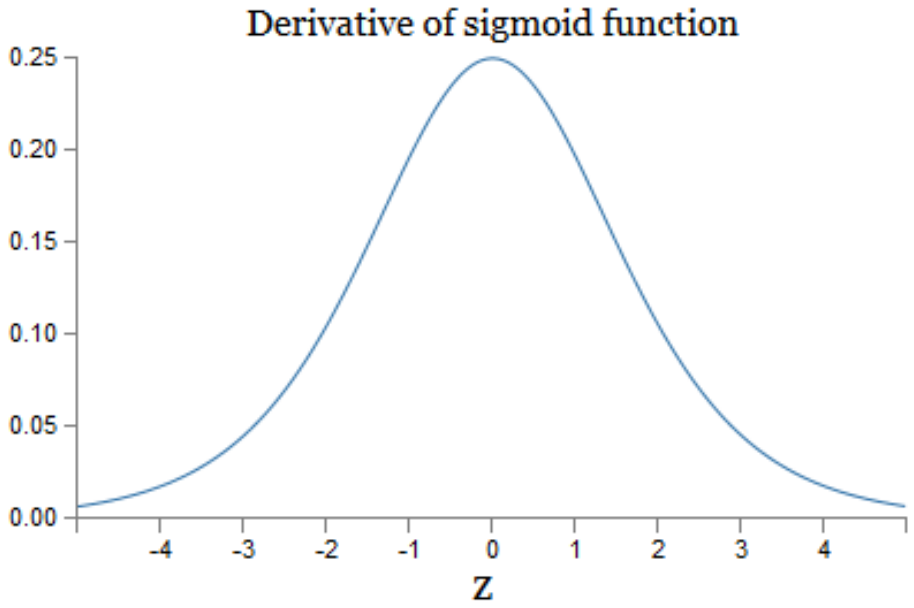


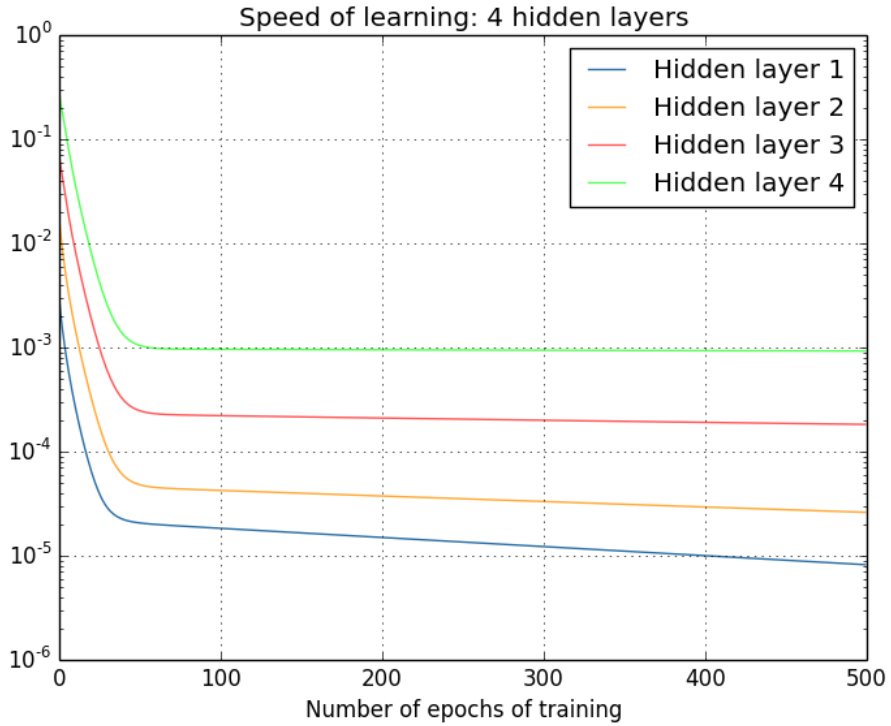
DNN Challenges

- 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

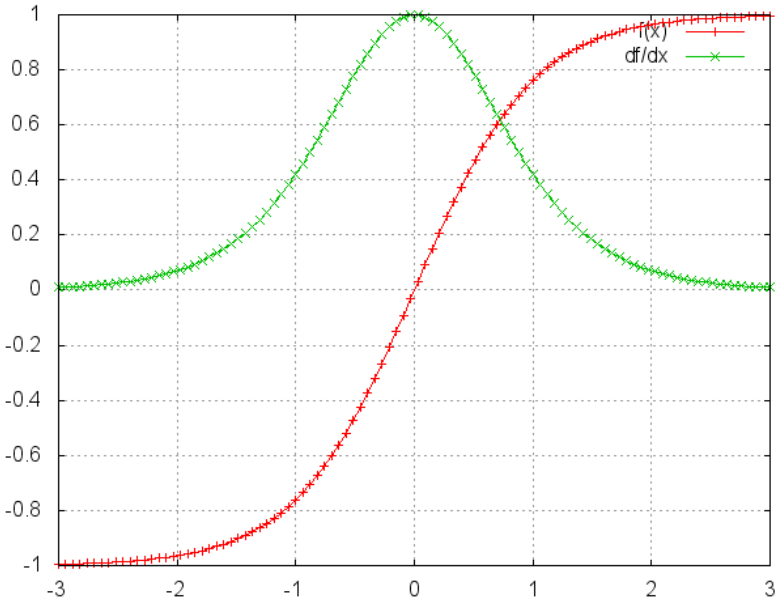
- Saturating nonlinearities (like tanh or sigmoid) can not be used for deep networks as they tend to get stuck in the saturation region as the network grows deeper

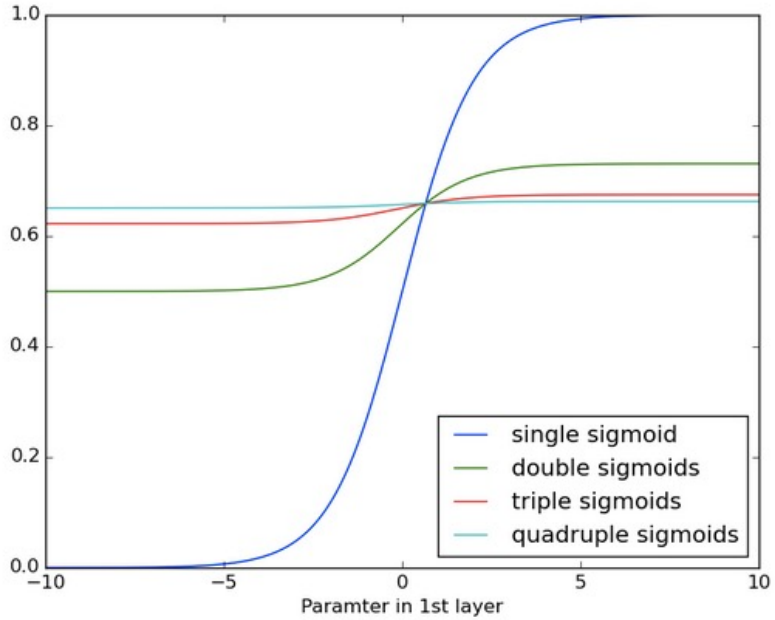




Vanishing Gradients

- 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

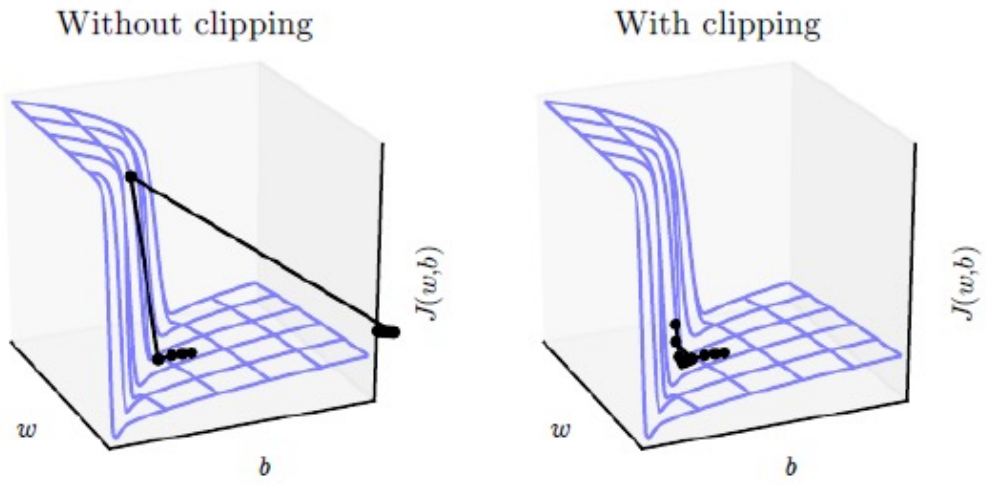




https://deeplearning4j.org/lstm.html

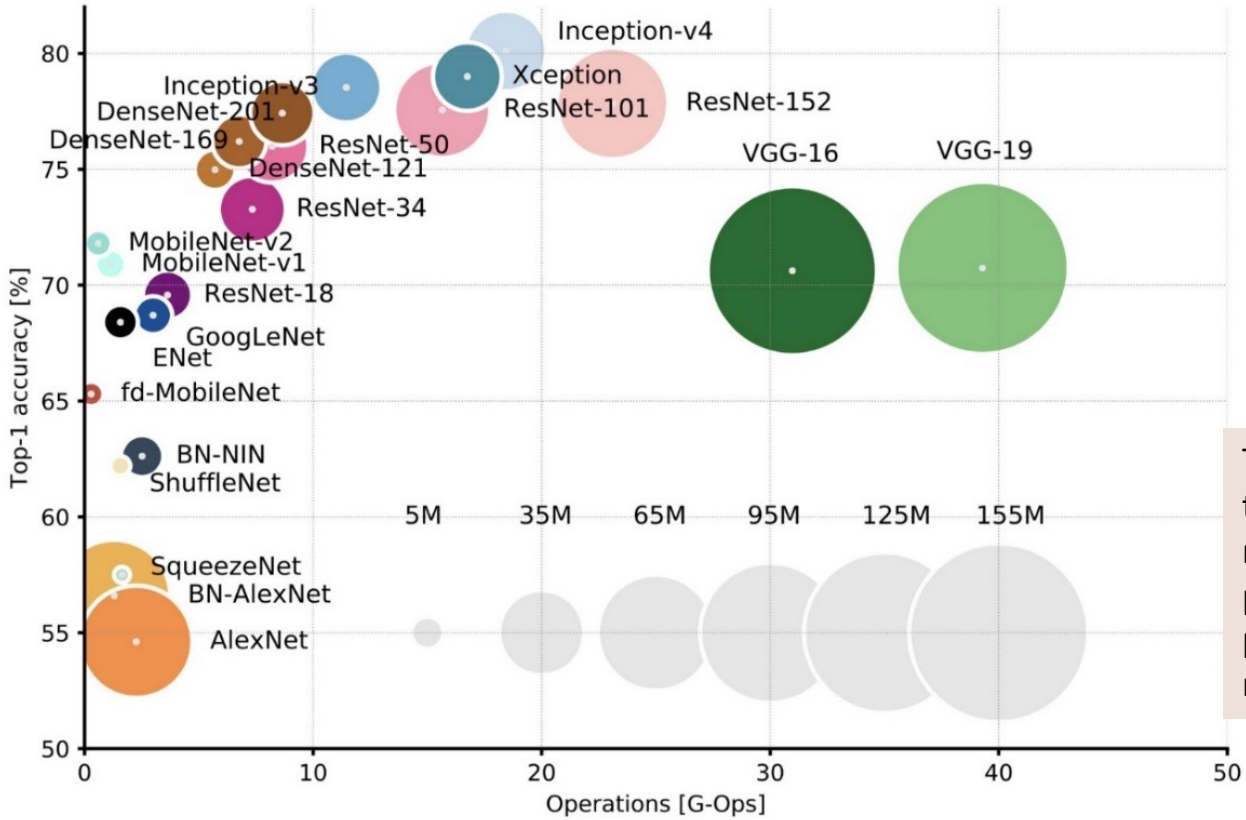
Exploding Gradient

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



Overparameterization

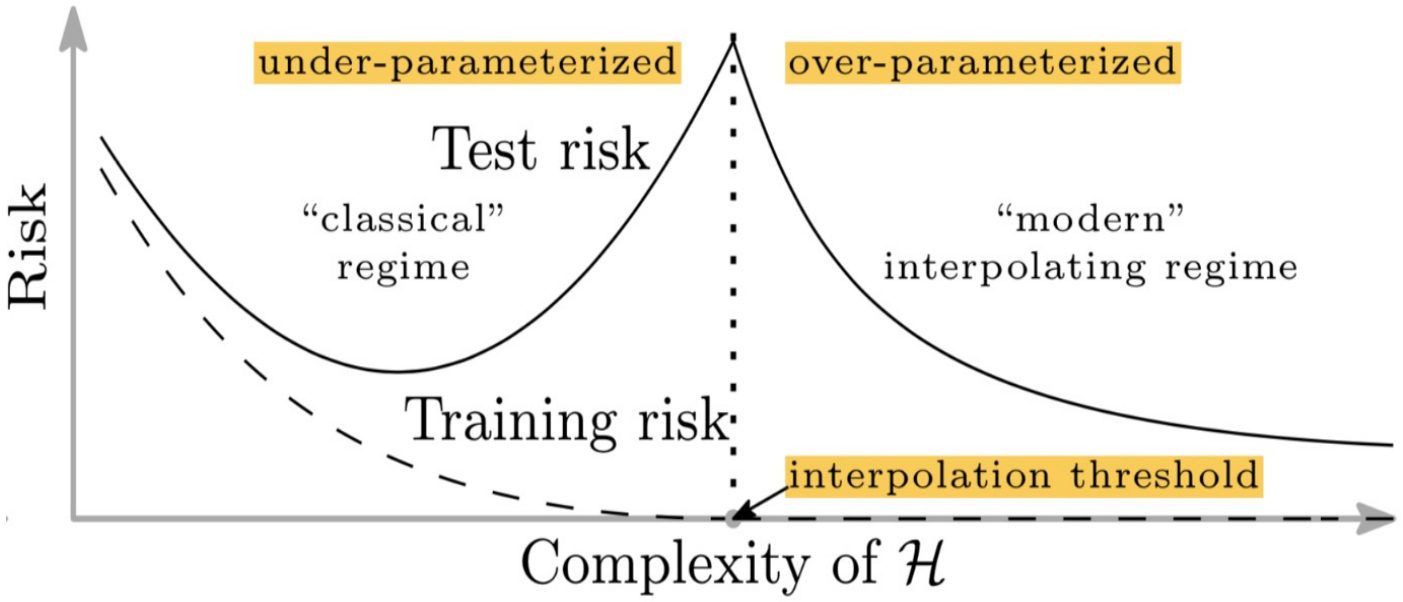
- Models where the number of parameters is significantly larger than the sample size
- Despite massive sizes, DNNs have exhibited remarkable generalization performance



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

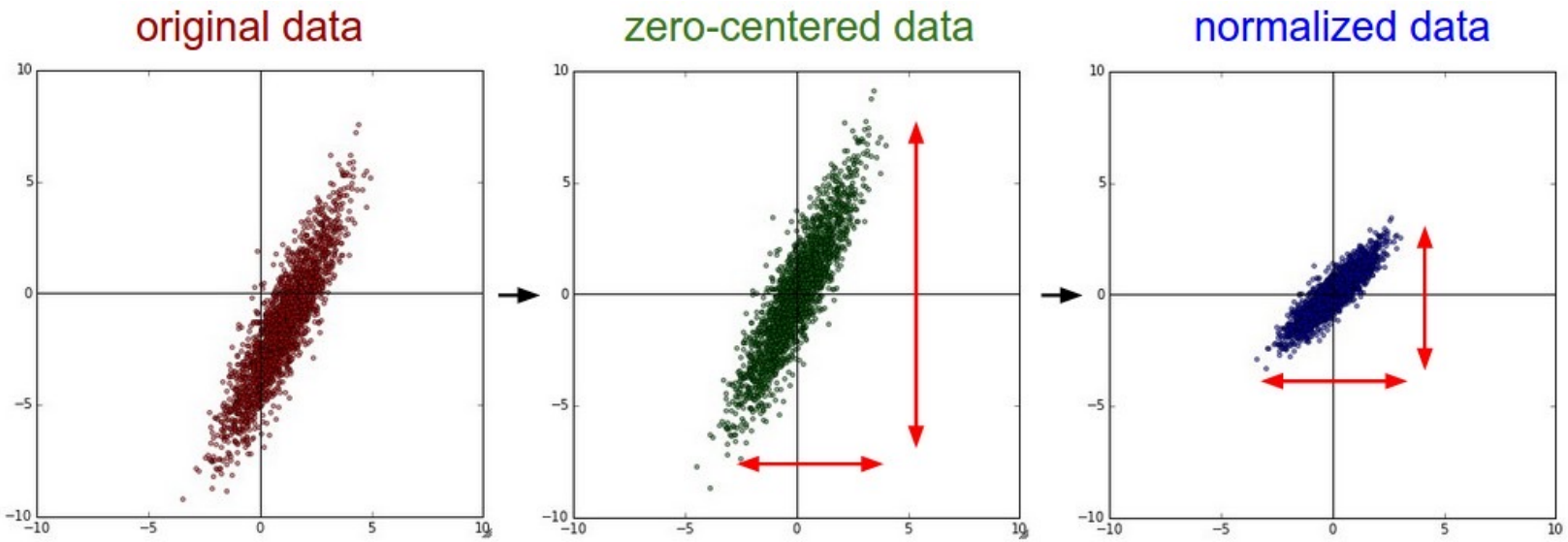
Data Preprocessing: Normalization

- Mean subtraction - centering the data around origin

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

- Normalization - dimensions approximately at same scale

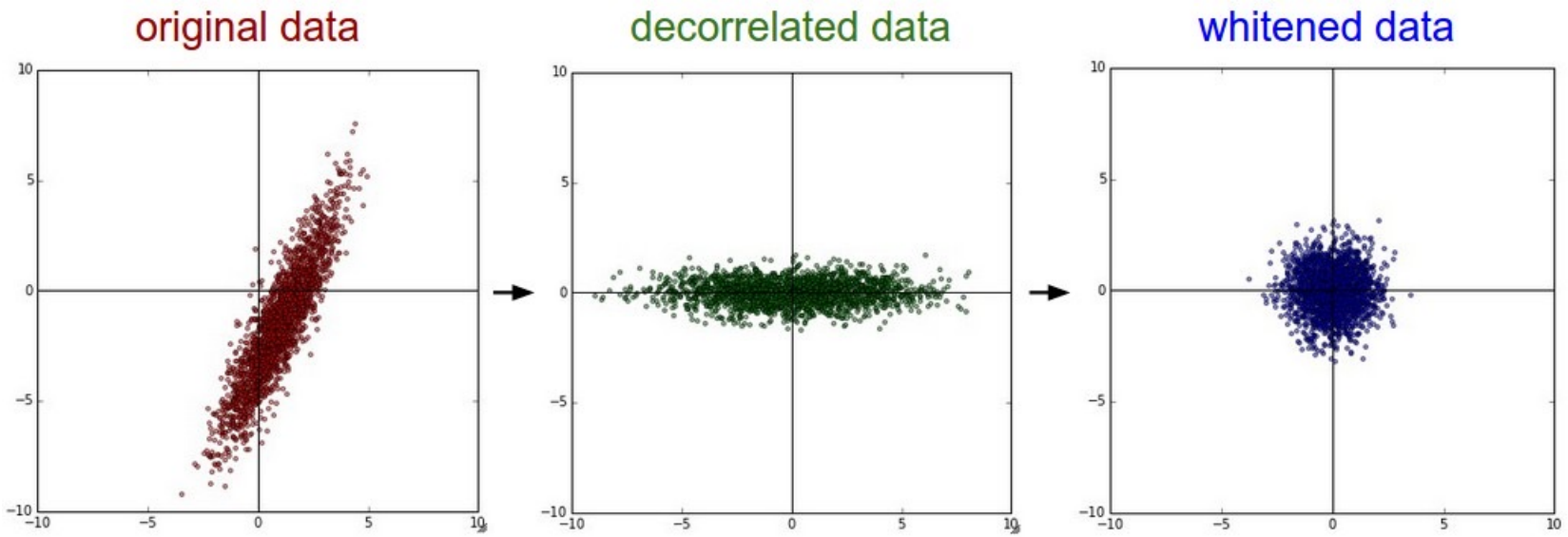
```
X /= np.std(X, axis = 0)
```



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

Data Preprocessing: PCA, Whitening

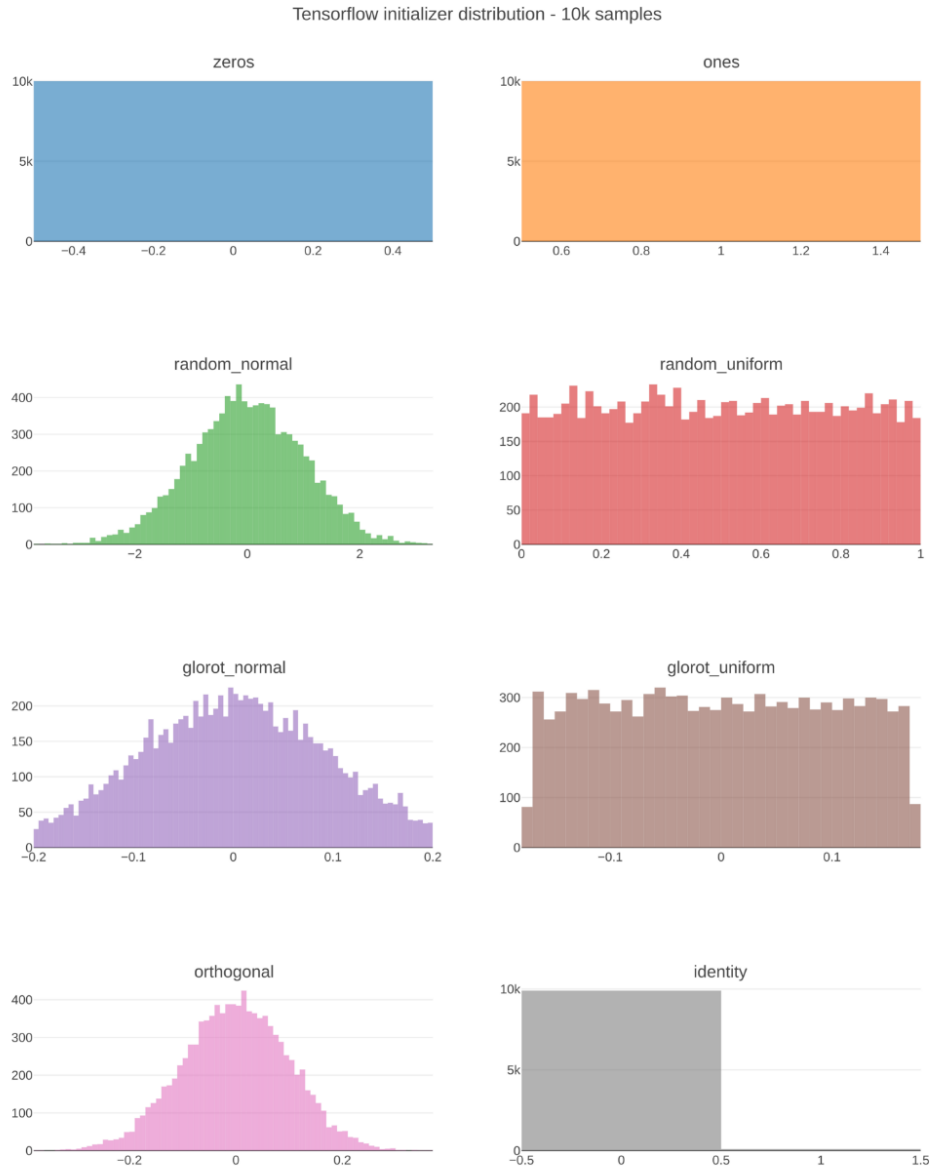
- Center data , calculate co-variance matrix
- SVD factorization, decorrelation
- PCA - dimensionality reduction
- Whitening - stretching data into an isotropic gaussian blob to normalize the scale



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

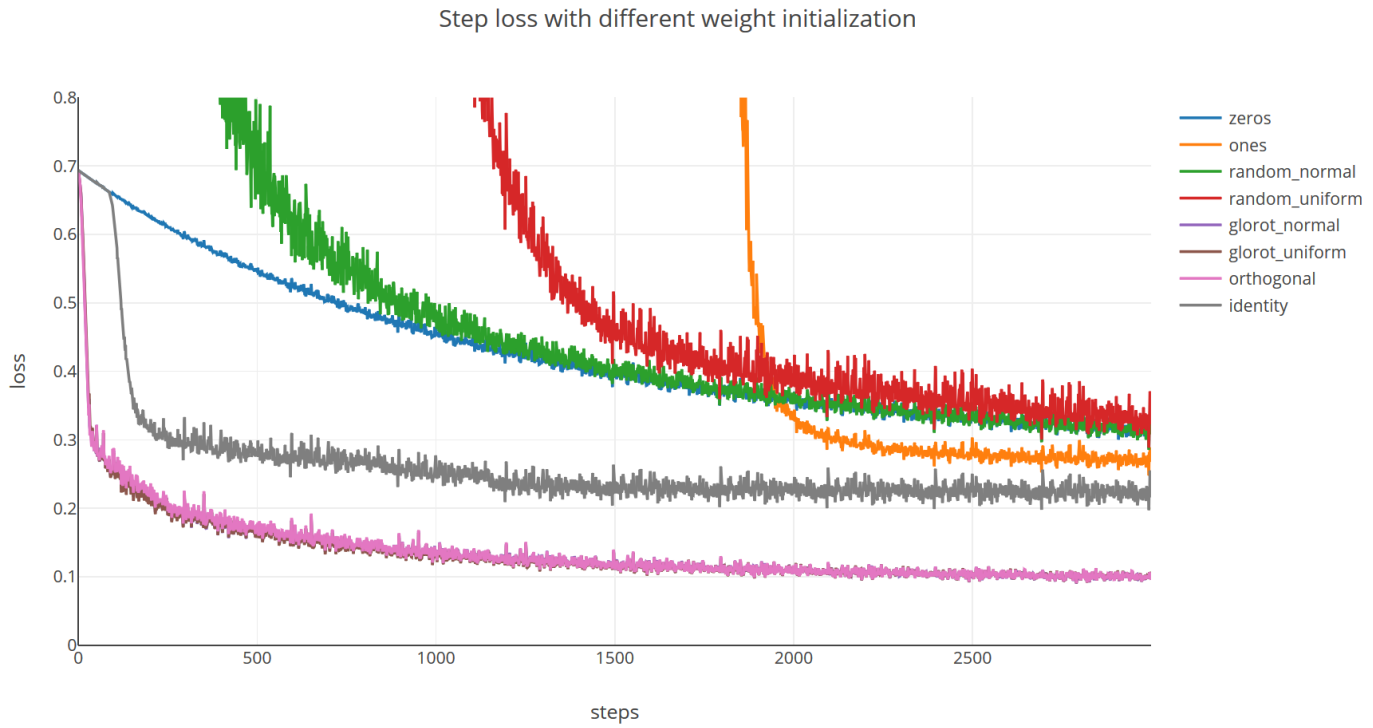
Weight Initialization

- 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 adding terms to loss function 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

Regularization: L1 vs L2

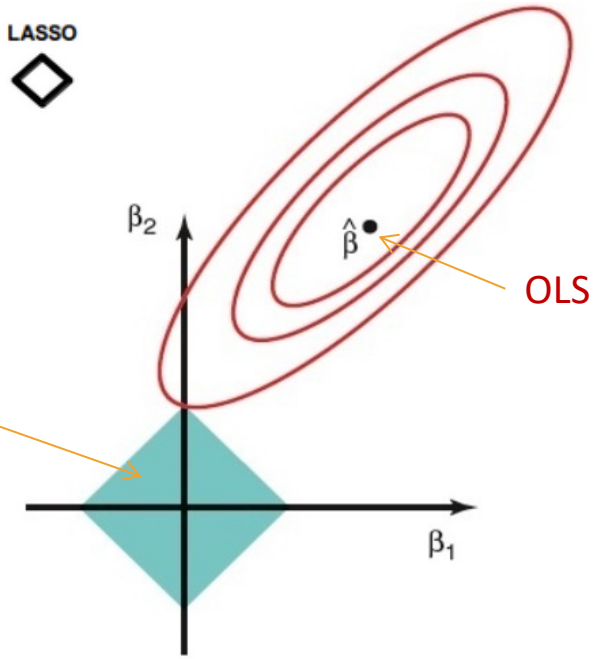
L1 Penalty

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}|\beta_j| = \mathrm{RSS}+\lambda\sum_{j=1}^{p}|\beta_j|.$$

L2 Penalty

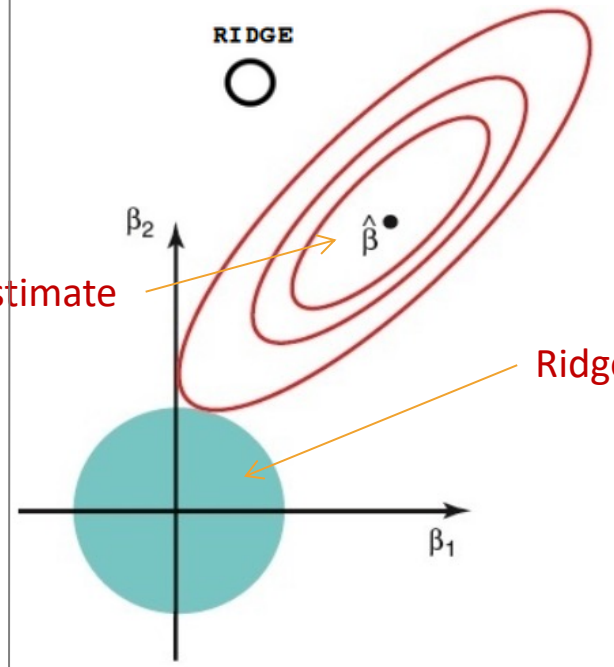
$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}\beta_j^2 = \mathrm{RSS}+\lambda\sum_{j=1}^{p}\beta_j^2$$

Contour plot of Lasso regression objective function



OLS Estimate

Lasso Estimate



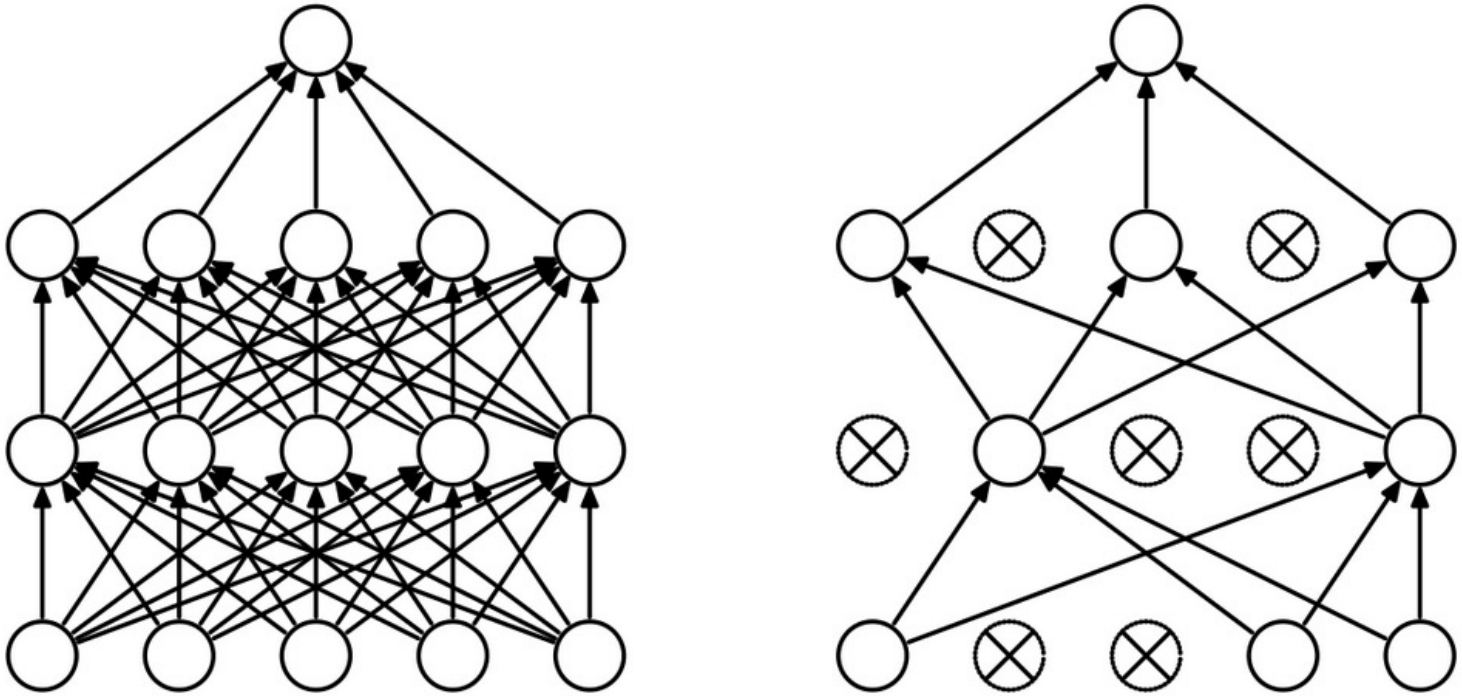
Ridge Estimate

Contour plot of Ridge regression objective function

L1 introduces sparsity since parameters can go to zero. This makes models easier to store and compute.

Dropout

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



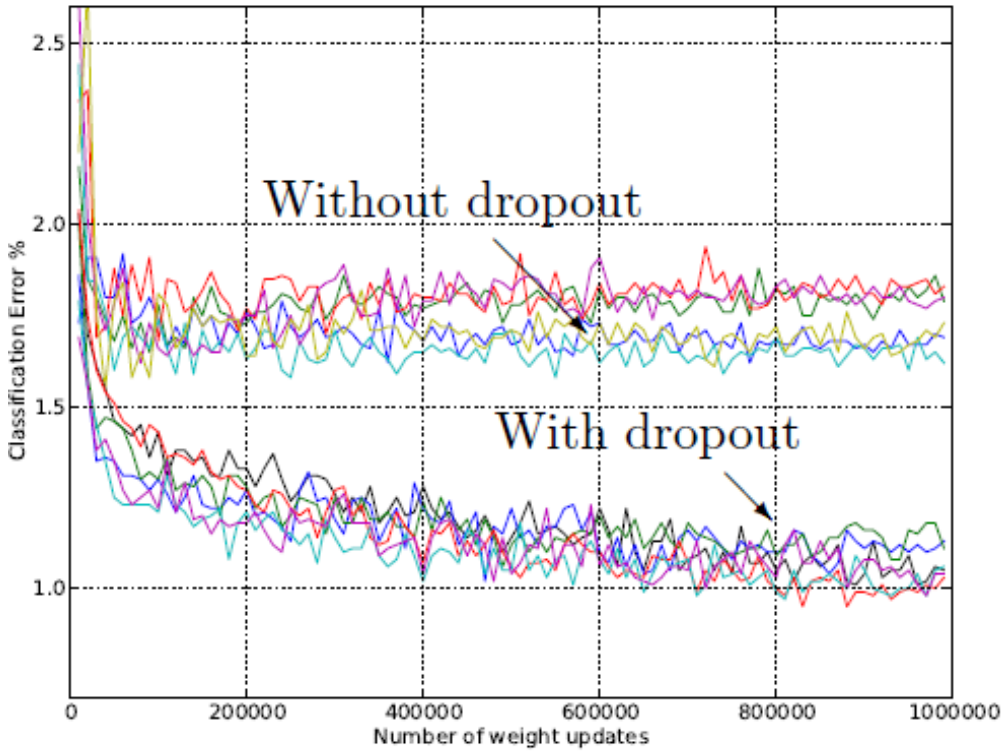
(a) Standard Neural Net

(b) After applying dropout.

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

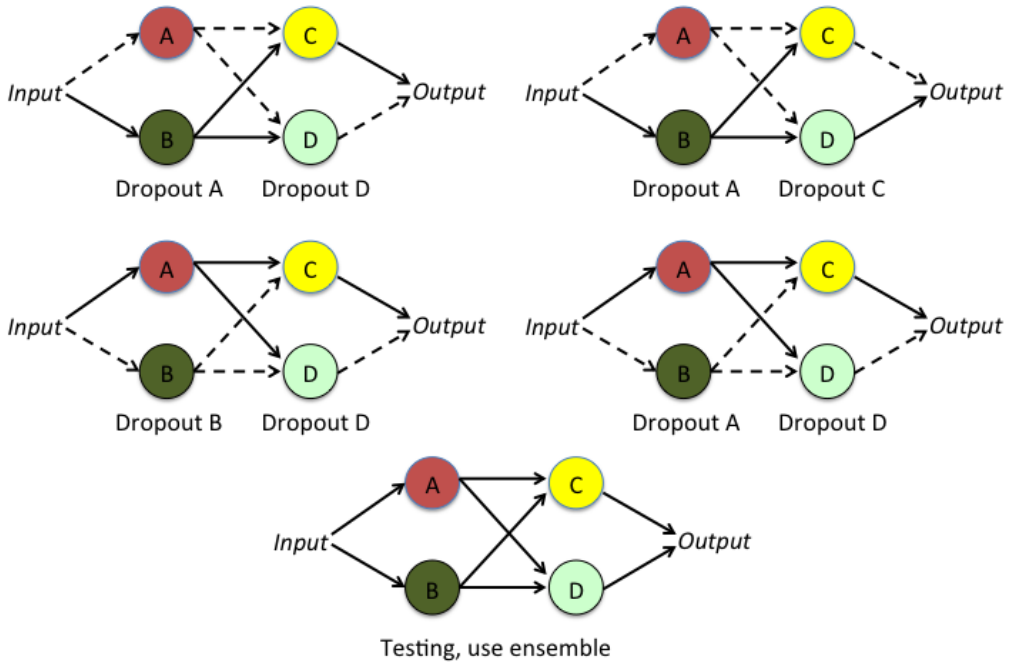
Dropout

- Prevents overfitting by combining exponentially many different ANN architectures efficiently



Dropout: Ensemble Effect

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



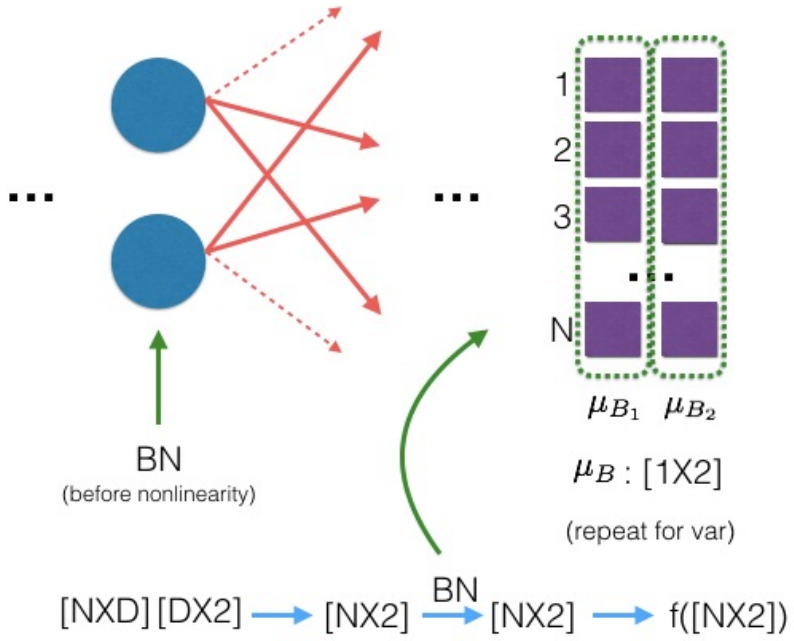
2 layers and 4 neurons in each layer, ${}^{4}C_{2}$ X ${}^{4}C_{2}$ = 36 ; takes average over 36 models.

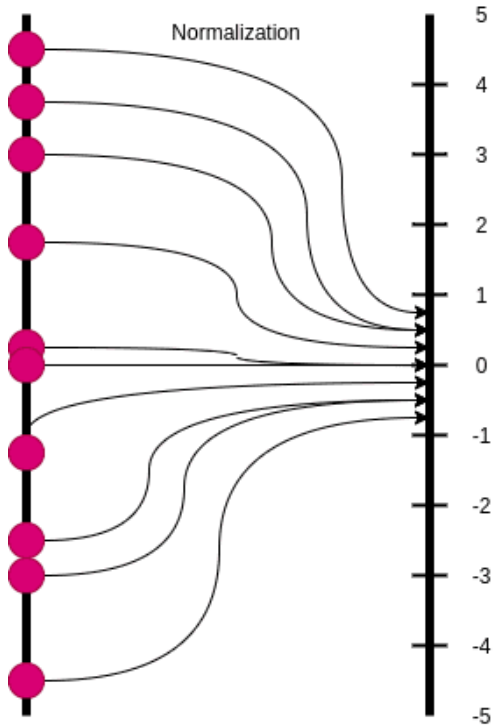
2 layers with 100 neurons in each layer, takes average over 24502500 possible models

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.
- Can decrease training time and result in better performance





Batch Normalization

- Stabilizes Training
- Faster Convergence
- Regularization Effect
- Reduces vanishing gradients
- Reduces sensitivity to
  - Weights initialization
  - Batch size
  - Learning Rates
- Enables Deeper Networks

Input

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

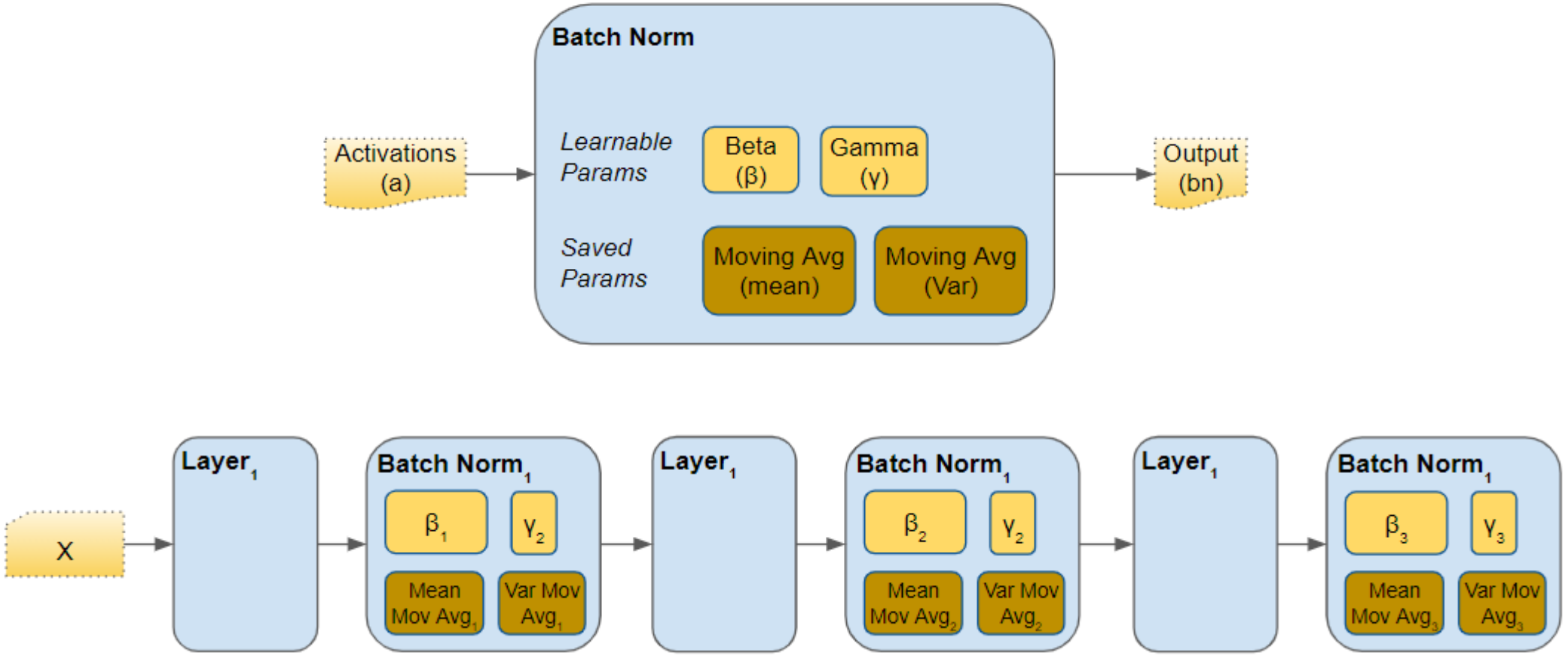
$$\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2$$ Mini-batch variance

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

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

Output

Batch Normalization: Parameters



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

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

DNN Training Best Practices

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

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

Model Interoperability: ONNX

- Open format built to represent ML models
- Makes it easier to access hardware optimizations.

```
pip install -U tf2onnx

# Using TF functions
onnx_model, _ = tf2onnx.convert.from_keras(model,
input_signature, opset=13)
onnx.save(onnx_model, "dst/path/model.onnx")

# Convert from saved TF model
python -m tf2onnx.convert --saved-model tensorflow-
model-path --output model.onnx
```