Introduction to Neural Networks

ACP in AI&MLOps Cohort 4

Deepak Subramani
Assistant Professor
Dept. of Computational and Data Science
Indian Institute of Science Bengaluru

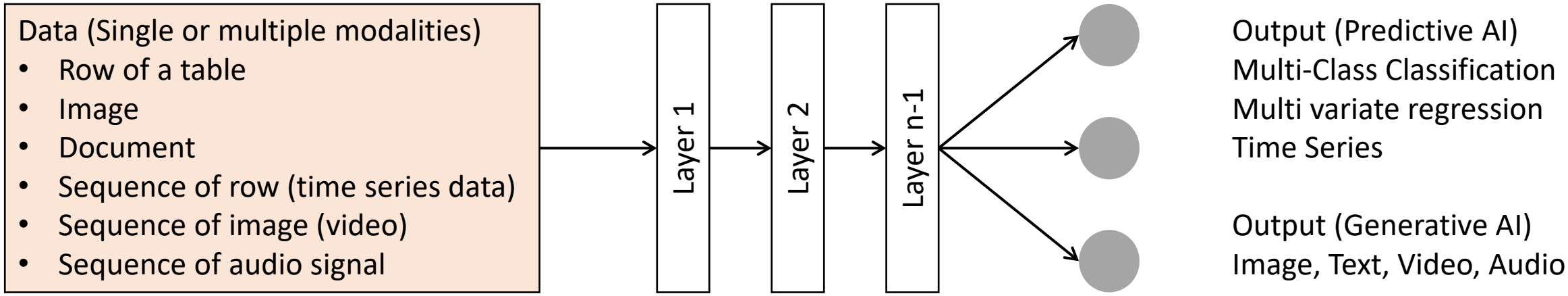
Learning Objectives

- What is a neural network?
- What are neurons, layers, models?
- What are parameters and hyperparameters?
- What is a loss function?
- What is an iteration, batch, epoch?

Quiz

- What will you use for questions?
  - Chat
  - QA

Neural Network

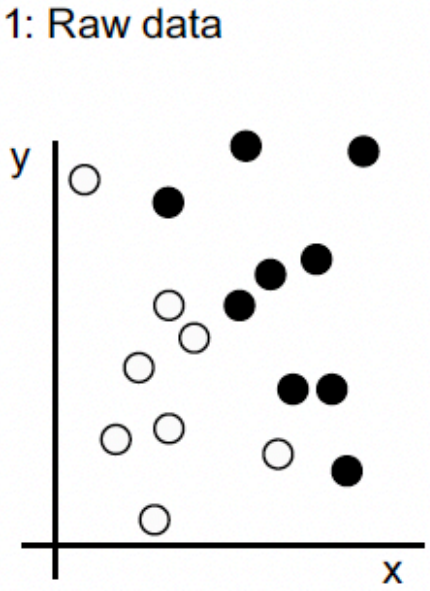


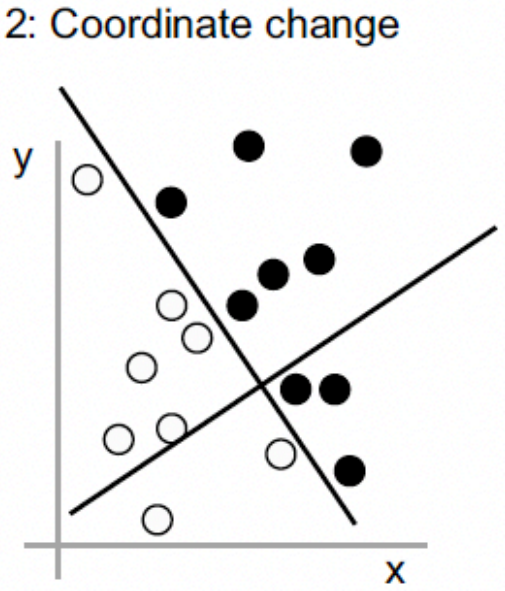
- Each Layer Consists of Neurons
- Neurons have a linear combination of input, followed by application of nonlinear activation function. Have parameters (weights and bias)
- Several Neurons form a layer
- Several Layers form a network
- Input has data
- Output layer depends on task
- Important Layers: Dense, Convolution, Multi-Head Attention, Masked MHA

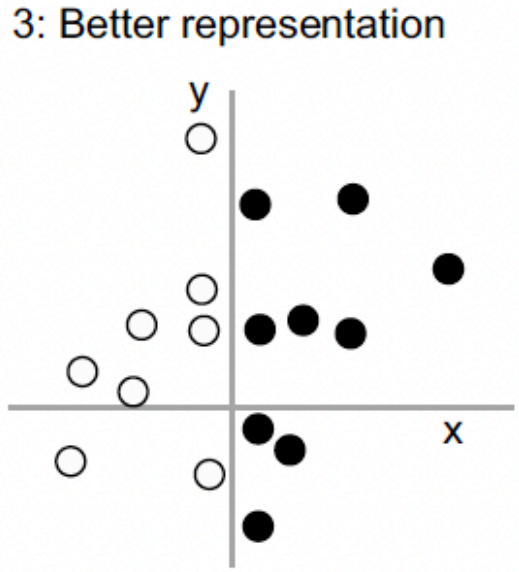
Department of Computational and Data Sciences

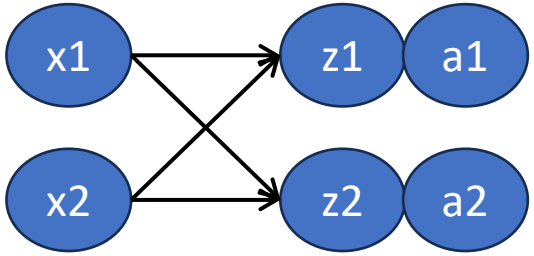
Indian Institute of Science
भारतीय विज्ञान संस्थान

Efficient Representation of Data



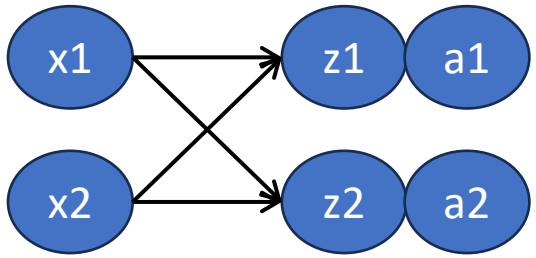




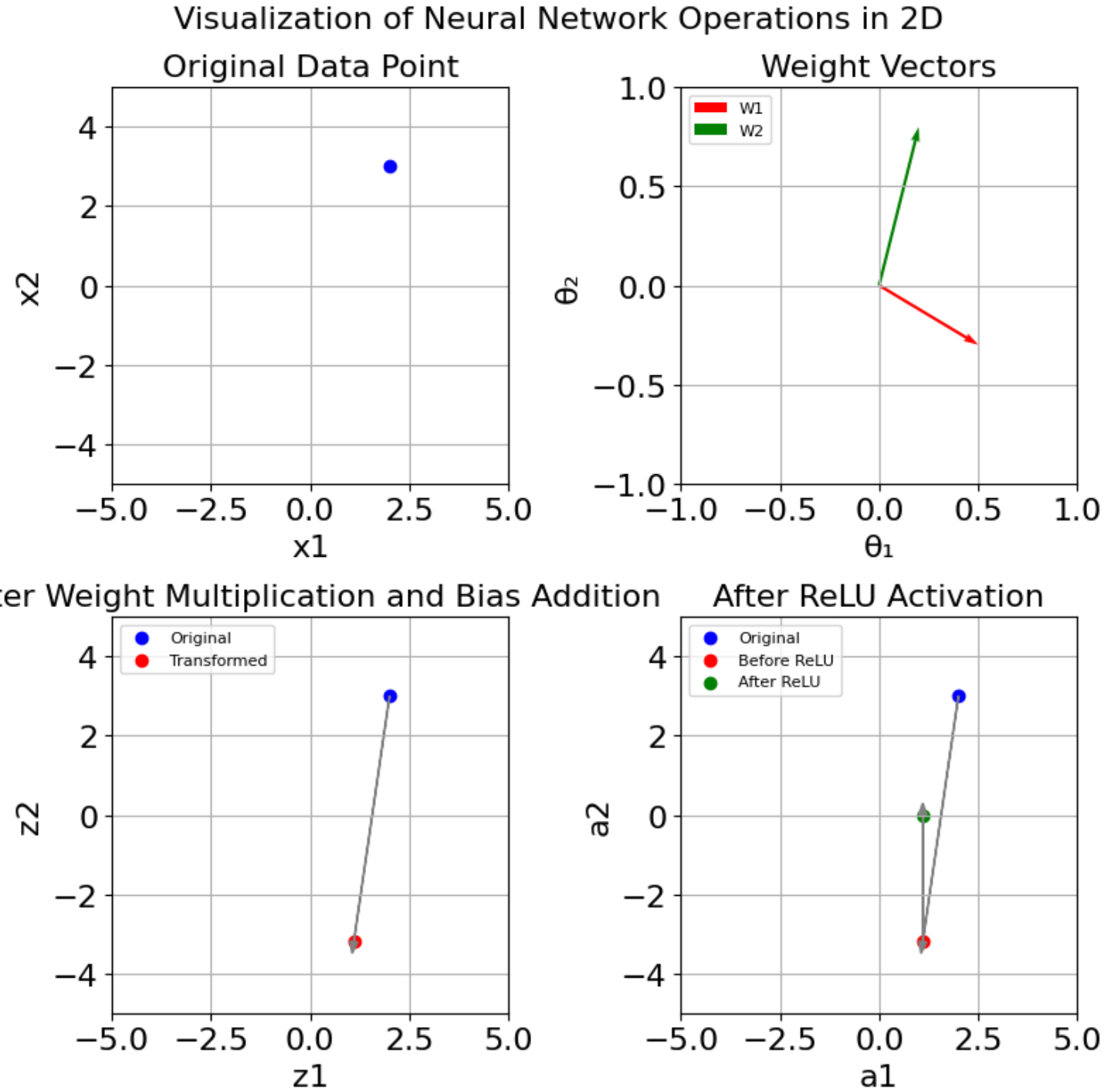


Deepak Subramani, deepakns@iisc.ac.in

Operations in a neural network



- x1,x2 are input neurons
- $z1 = \theta_{1,w1}x1 + \theta_{2,w1}x2 + b1$
- $z2 = \theta_{1,w2}x1 + \theta_{2,w2}x2 + b2$
- $a1 = \max(0, z1)$
- $a2 = \max(0, z2)$



What is a Neuron?

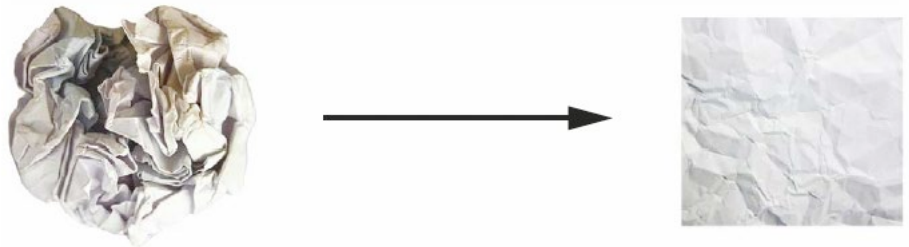
- Computational Unit, Data Processing Unit
- It does two activities
  - Take a weighted linear combination of input, add a bias
  - Apply an activation function on the result

Activation Functions

- Linear $a = z$
- Rectified Linear Unit (ReLU) $a = \max(0, z)$
- Sigmoid $a = \frac{1}{1+e^{-z}}$
- Softmax $a_k = \frac{e^{z_k}}{\sum_{k=1}^{K} e^{z_k}}$

Visualizing the action of a neural network

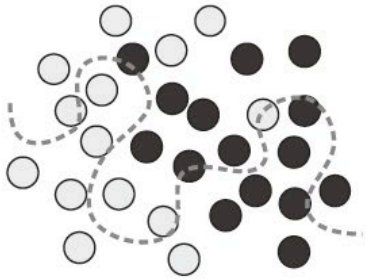
- Neural layers transform the data to different representations
- The transformations are effected through weights and biases
- These weights and biases are learnt from data during training
- The learnt representation make the downstream task easier
- Prior to deep neural networks, these representations were hand crafted using a mechanism called as feature engineering (from domain knowledge)



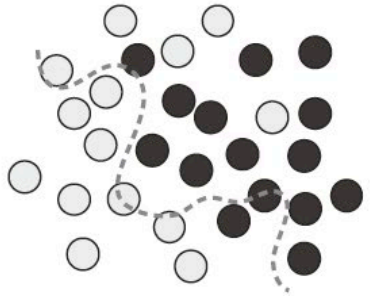
Layers of a neural model unflods the crumbled paper ball (data in complex manifolds) to an unfolded paper (simple representation)

Decision Boundaries with Training

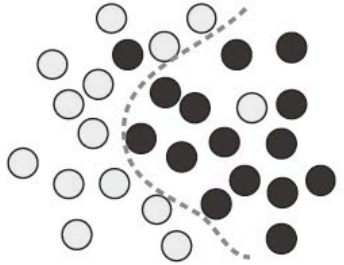
Before training:
the model starts
with a random initial state.



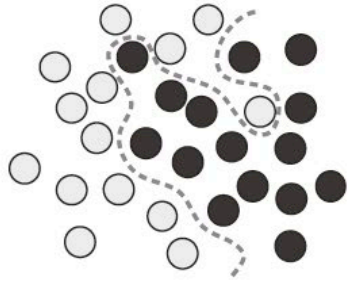
Beginning of training:
the model gradually
moves toward a better fit.



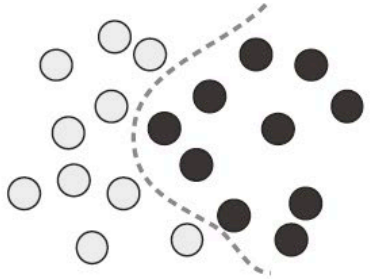
Further training: a robust
fit is achieved, transitively,
in the process of morphing
the model from its initial
state to its final state.



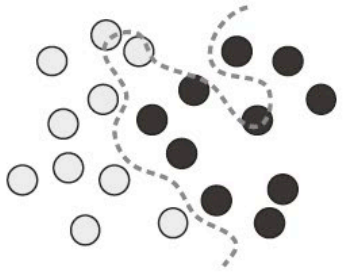
Final state: the model
overfits the training data,
reaching perfect training loss.



Test time: performance
of robustly fit model
on new data points



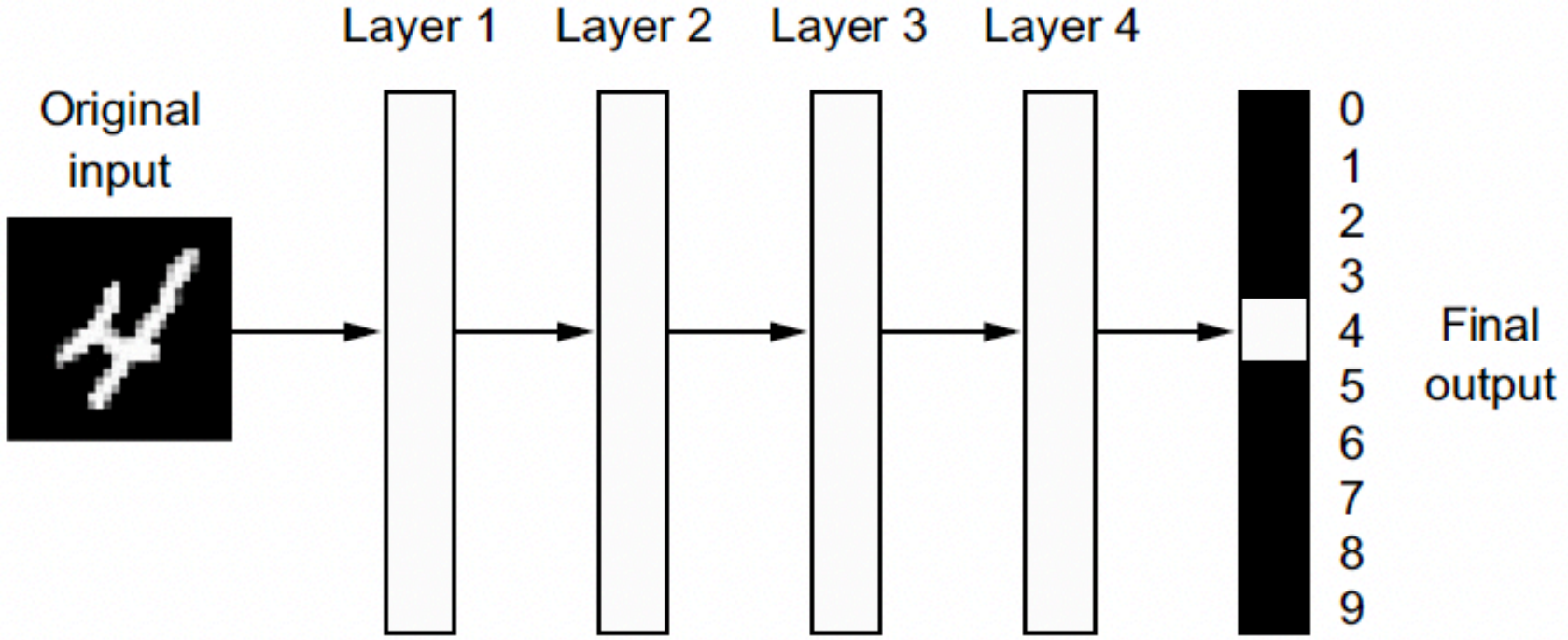
Test time: performance
of overfit model
on new data points



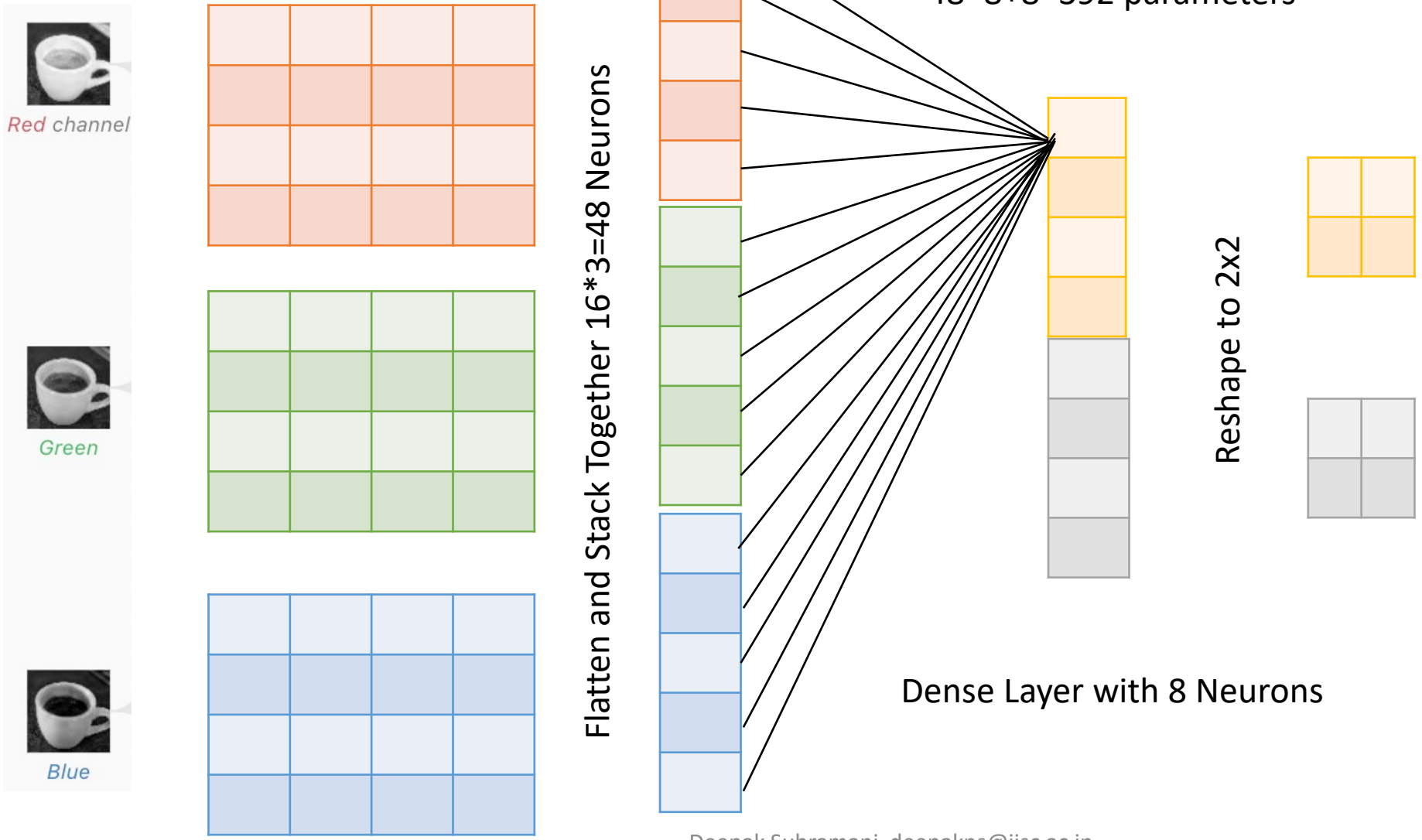
Neural Network for Classification

- 2D image is flattened to a vector
- Vector is passed through layers
- Final output layer has 10 neurons corresponding to 10 class classification

MNIST Hand Written Digit Recognition



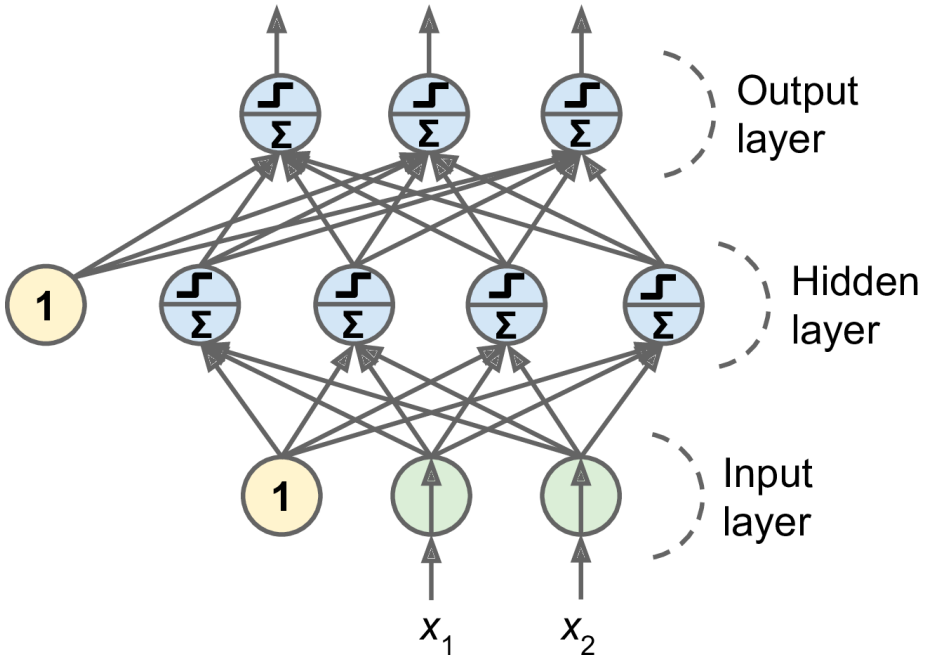
Neuron Arrangement in Dense Layer



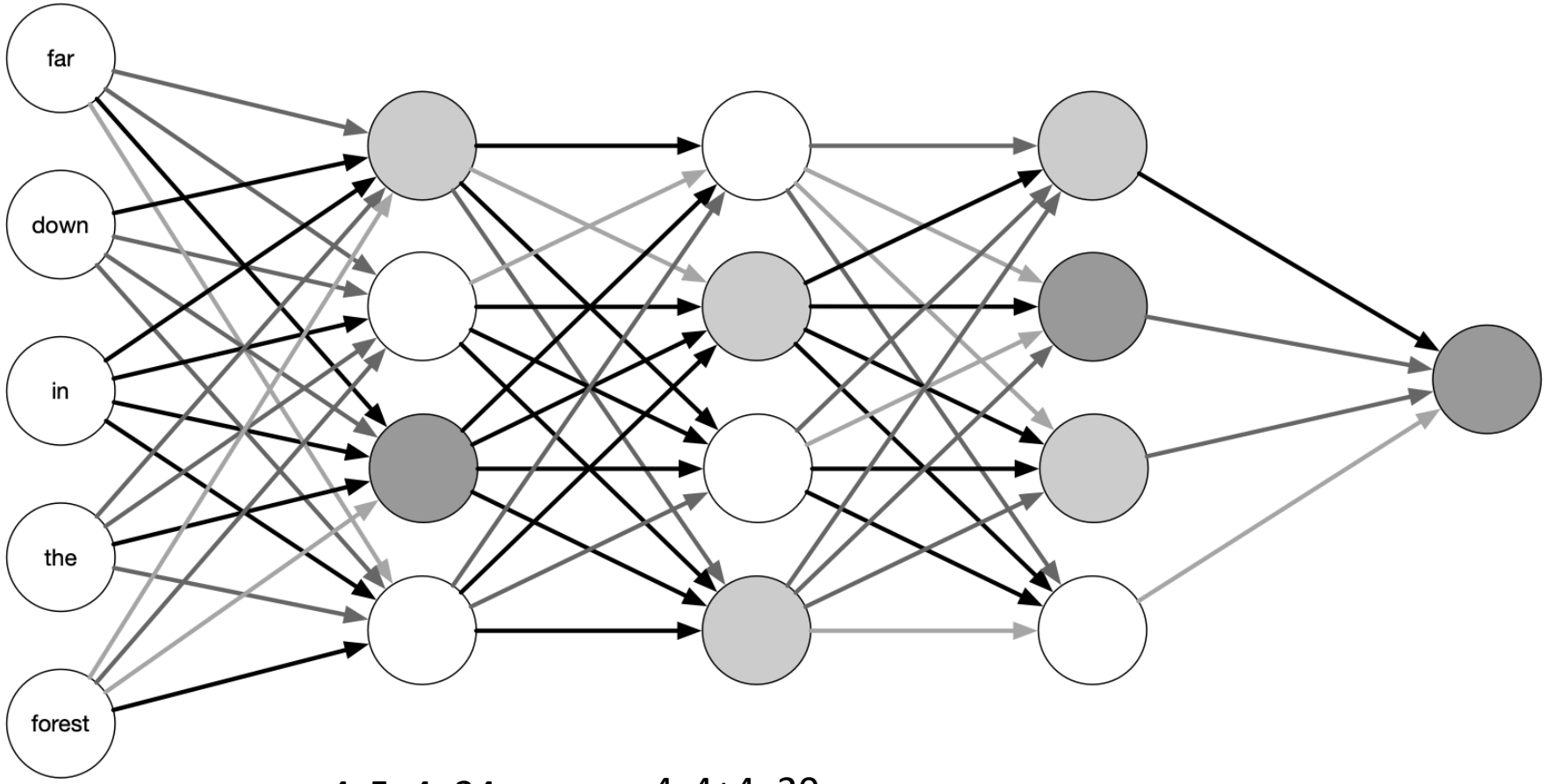
Deepak Subramani, deepakns@iisc.ac.in

Dense Neural Network

- Input layer is a passthrough layer
- Hidden layers are composed of neurons
- Output layer returns
  - a number (as many neurons as quantities of interest) for regression tasks and
  - probability of a class for classification tasks (as many neurons as number of classes)
- Layers closer to the input are called as lower layers
- Layers closer to the output are called as upper layers
- Here the signal only flows in one direction, so this architecture is called as a feedforward neural network (FNN)



Calculating Size of a Dense Neural Network



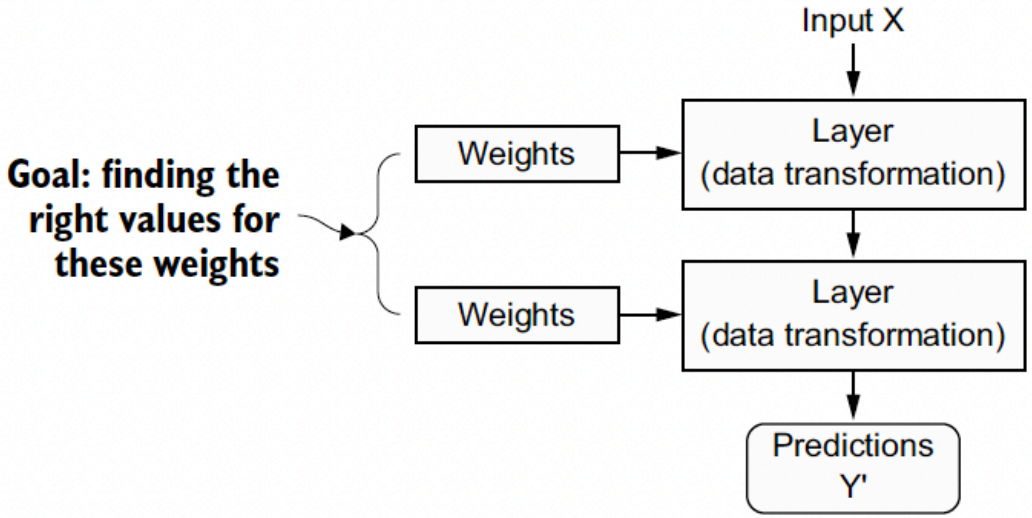
4x5+4=24 4x4+4=20 4x4+4=20 1x4+1=5

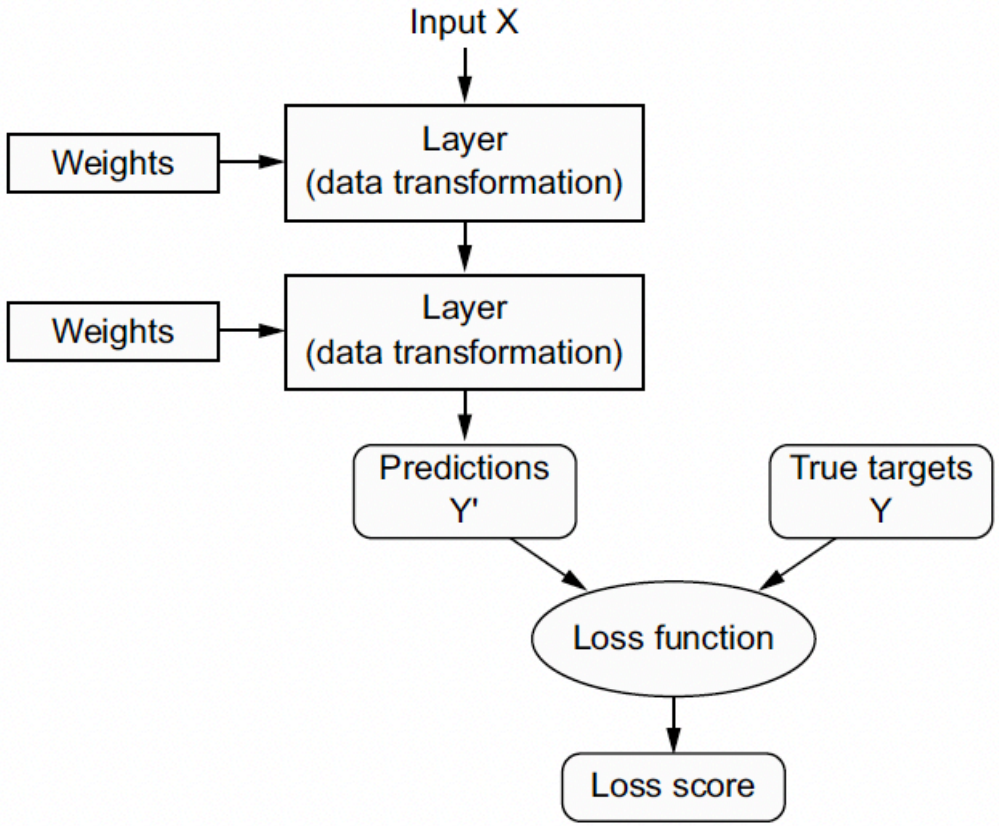
Deepak Subramani, deepakns@iisc.ac.in

Training the Neural Network

Goal of training is to find parameters (Weights and bias of every neuron)

Ground truth supervision data gives us the loss





Loss Function

- Regression – Mean Square Error

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2$$

m – num data points
K – num of classes

- Classification – Cross Entropy

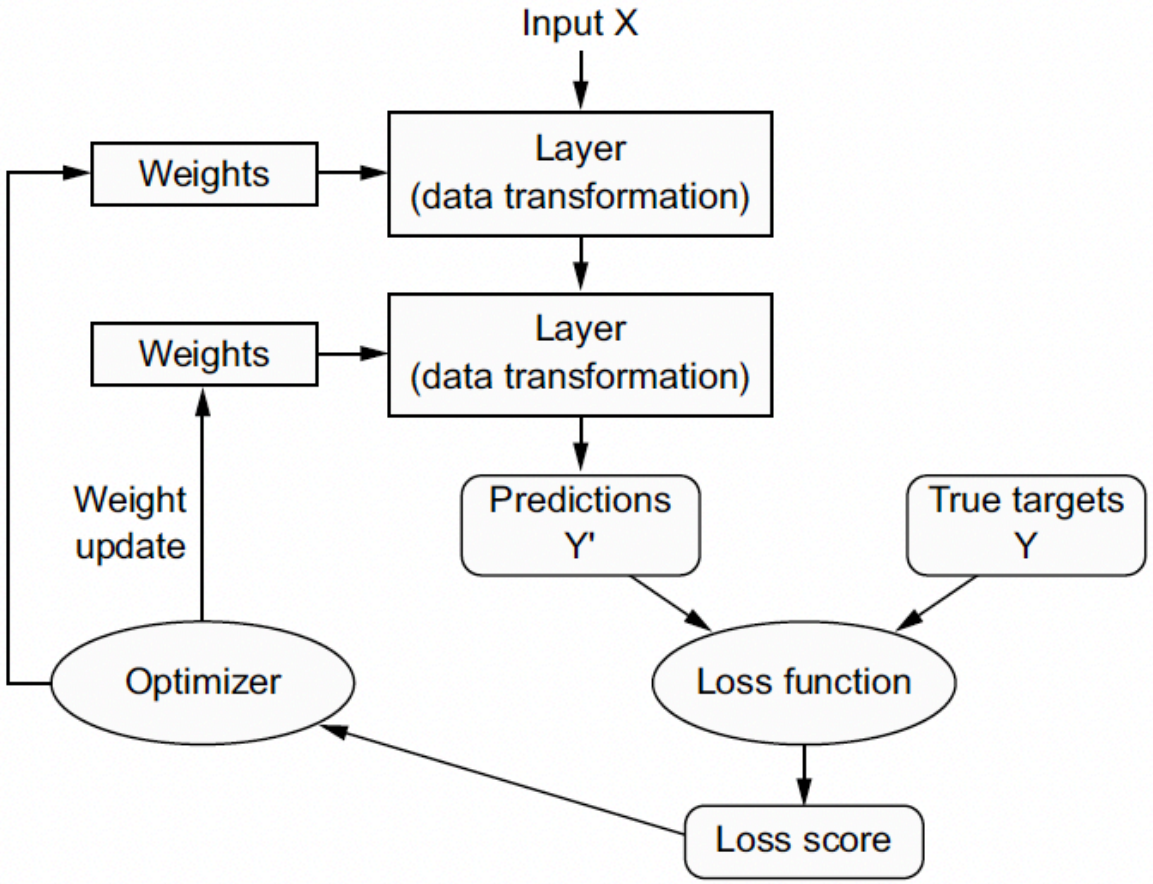
$$CE = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_{i,k}\log \hat{p}_{i,k}$$

Training the Neural Network

An optimizer adjusts the weights (and bias) in such a way that the loss function reduces

The process of finding these weights (and bias) is called training

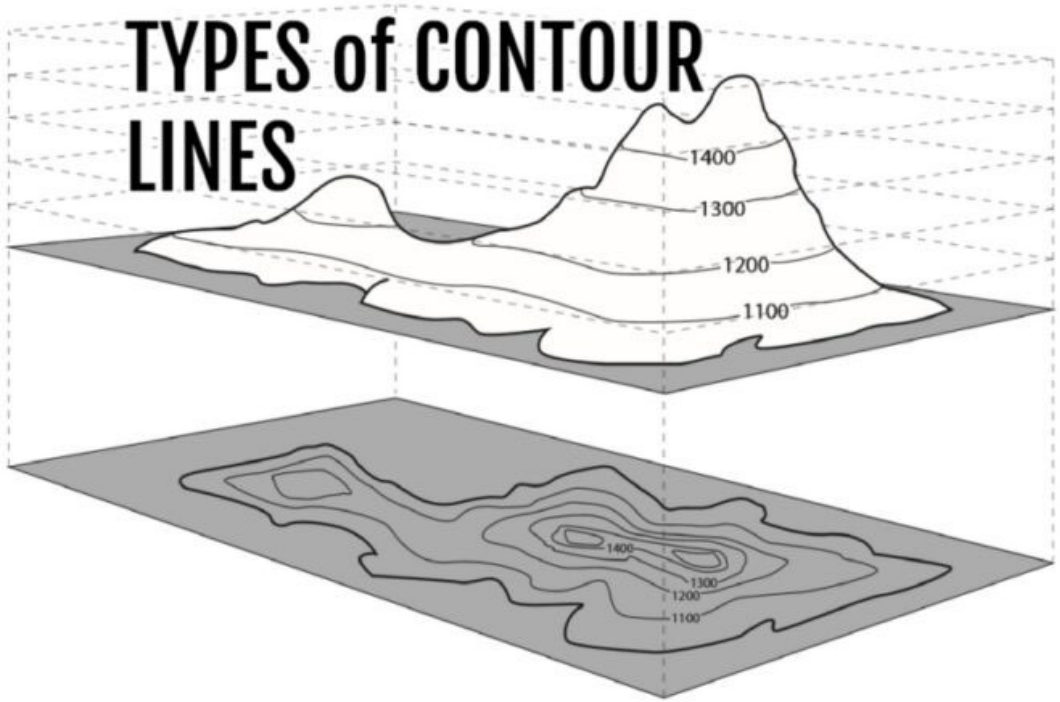
A training algorithm called Stochastic Gradient Descent (variant called ADAM) is used for the training



Department of Computational and Data Sciences

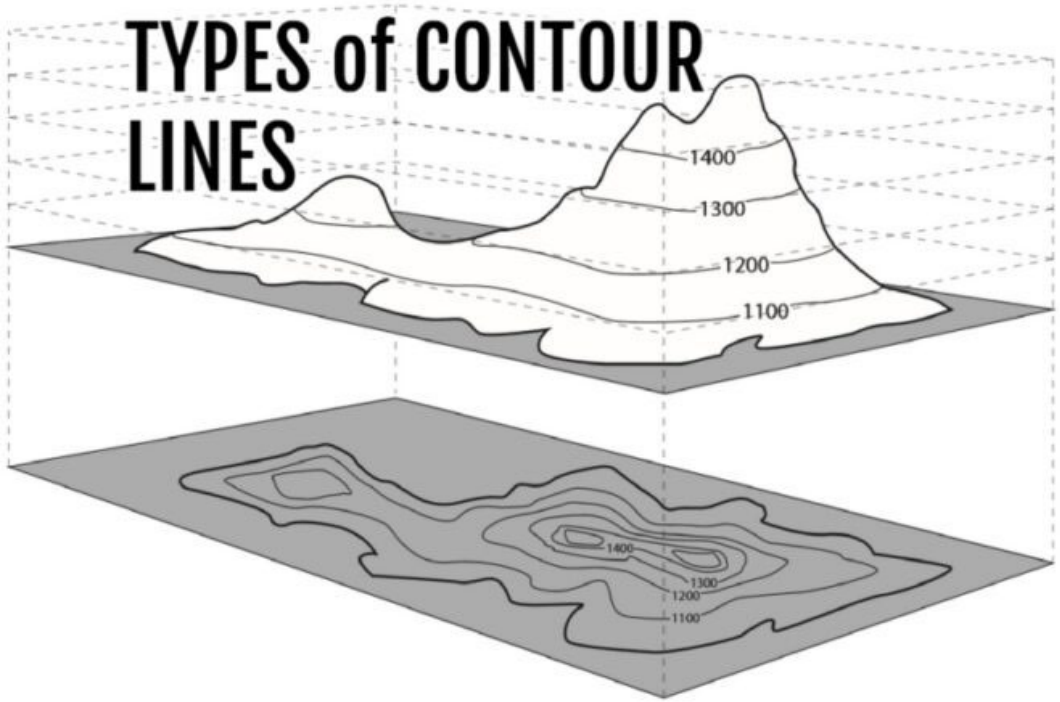
Indian Institute of Science
भारतीय विज्ञान संस्थान

Gradient Descent: Figure Explanation



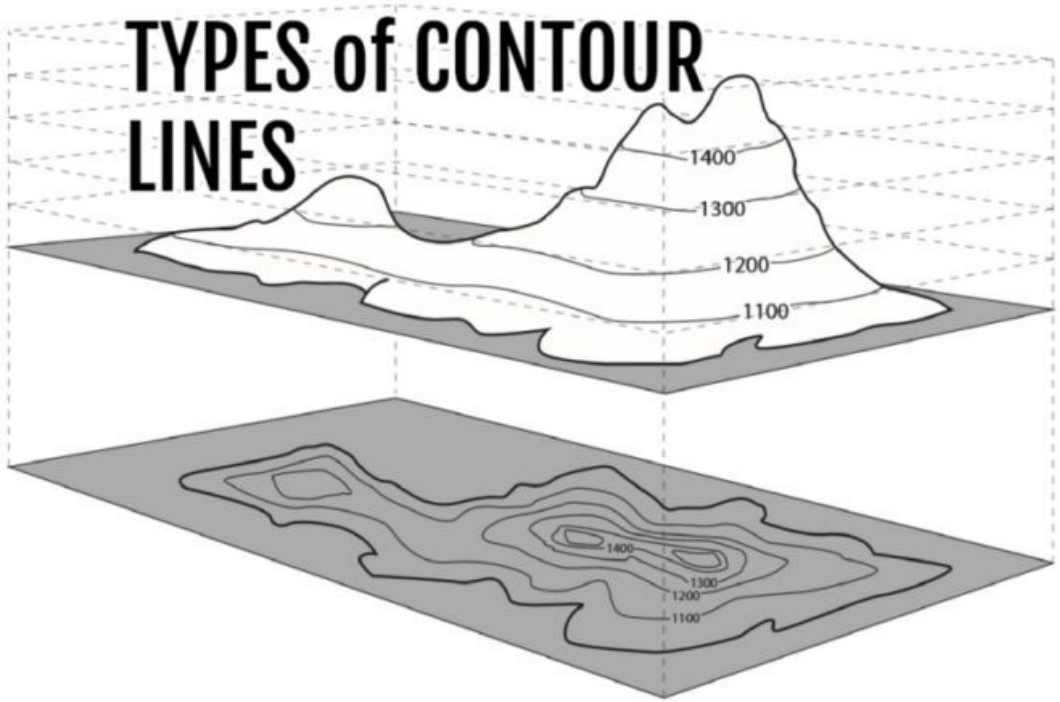
Deepak Subramani, deepakns@iisc.ac.in

Gradient Descent: Figure Explanation



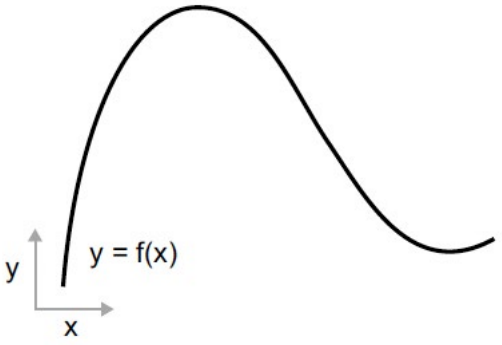
Deepak Subramani, deepakns@iisc.ac.in

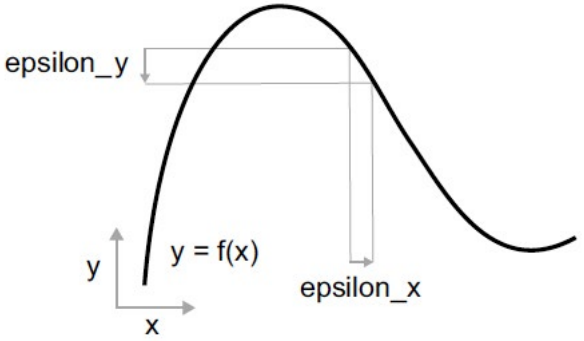
Gradient Descent: Figure Explanation

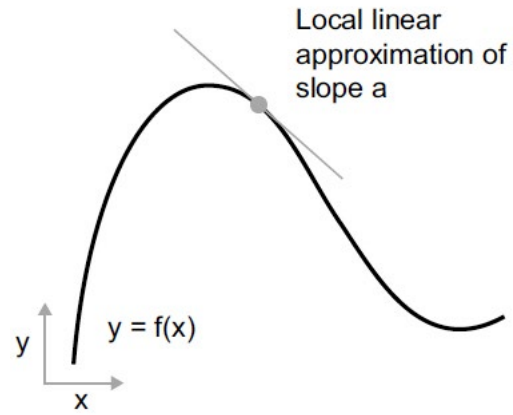


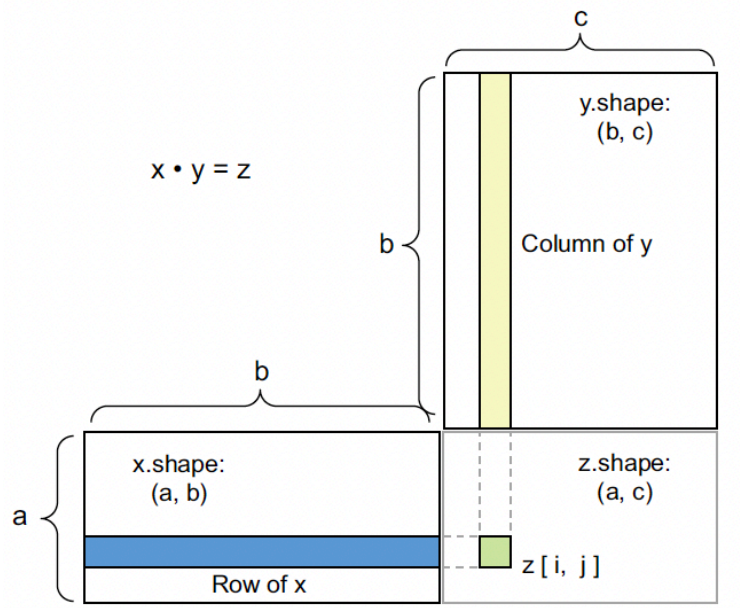
Deepak Subramani, deepakns@iisc.ac.in

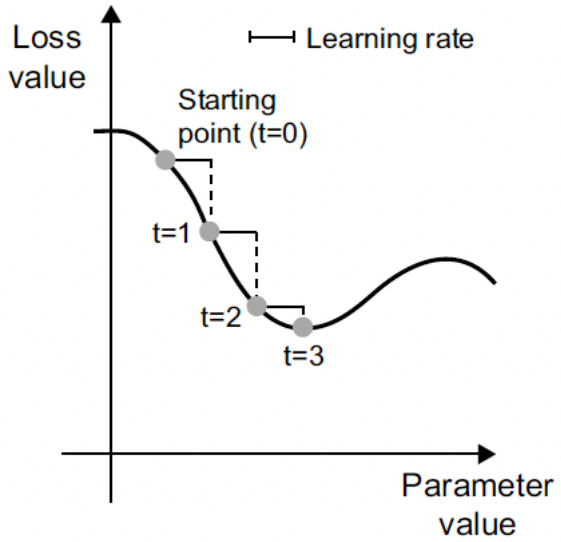
Math through figures

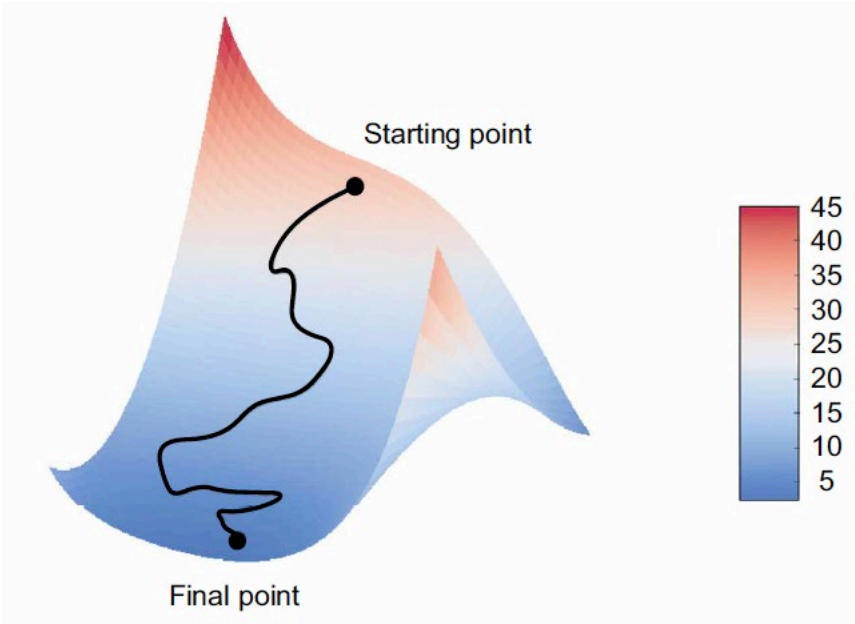








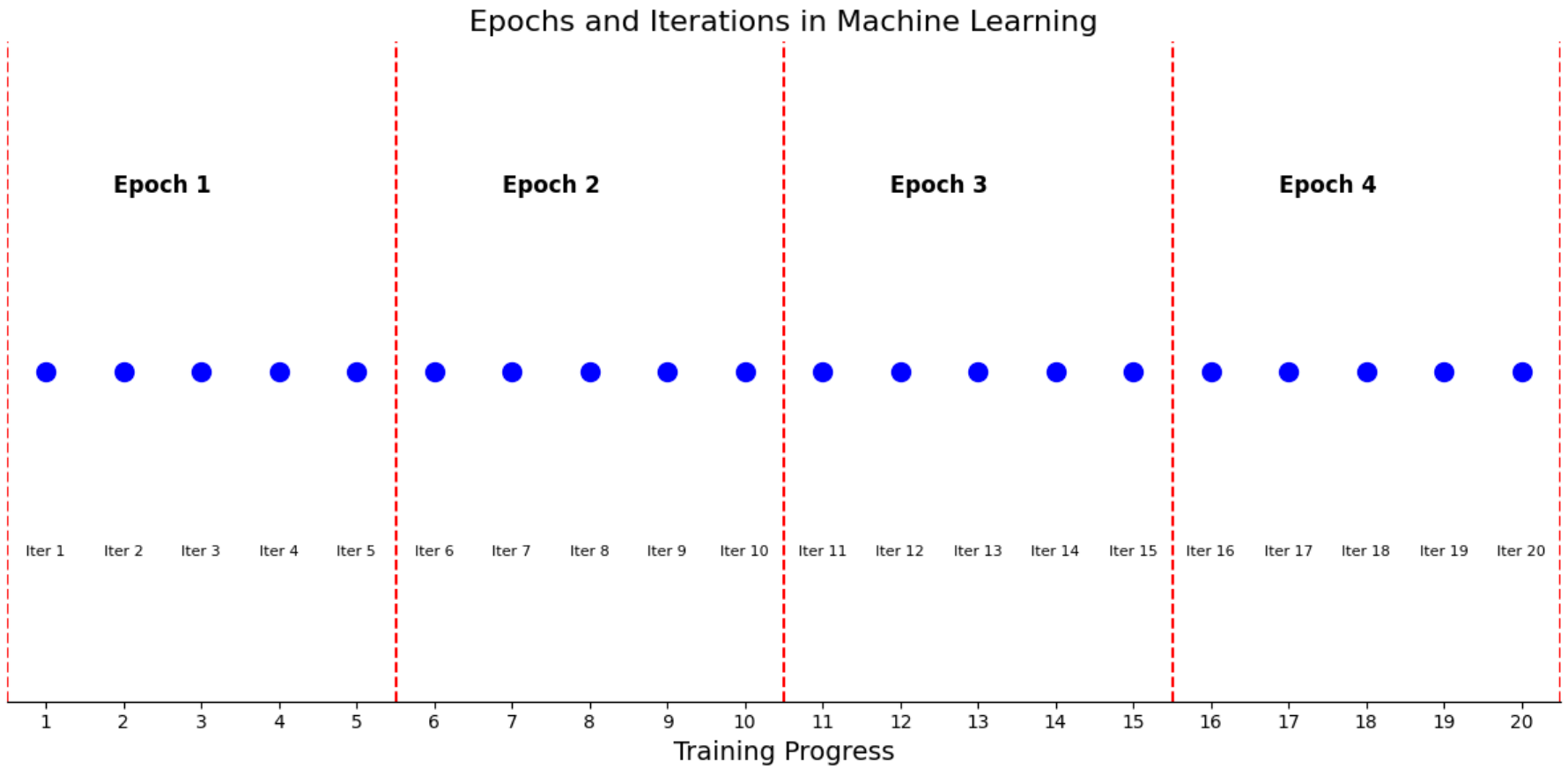




Deepak Subramani, deepakns@iisc.ac.in

Iteration, Batch, Epoch

- Data Set = 1000 samples
- Batch Size = 200
- One iteration uses 200 data points to calculate loss and derivative of the loss with respect to all params
- 5 iterations comprise one epoch
- In this example 4 epochs are shown
- In each epoch all the parameters are updated 5 times (in this example)
- Epoch 2 starts with Iter 6
- It updates the parameters at the end of Iter 5 (Epoch 1)

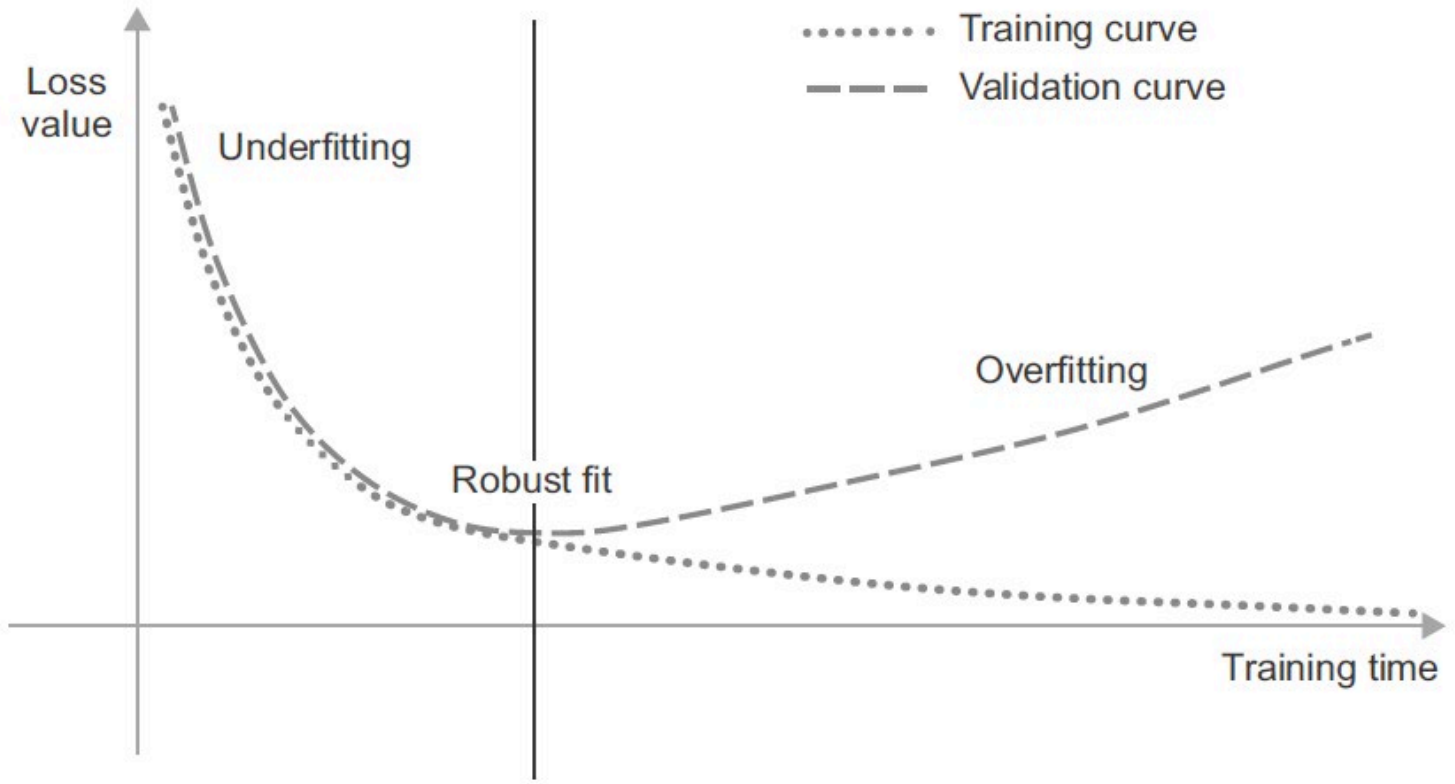


Deepak Subramani, deepakns@iisc.ac.in

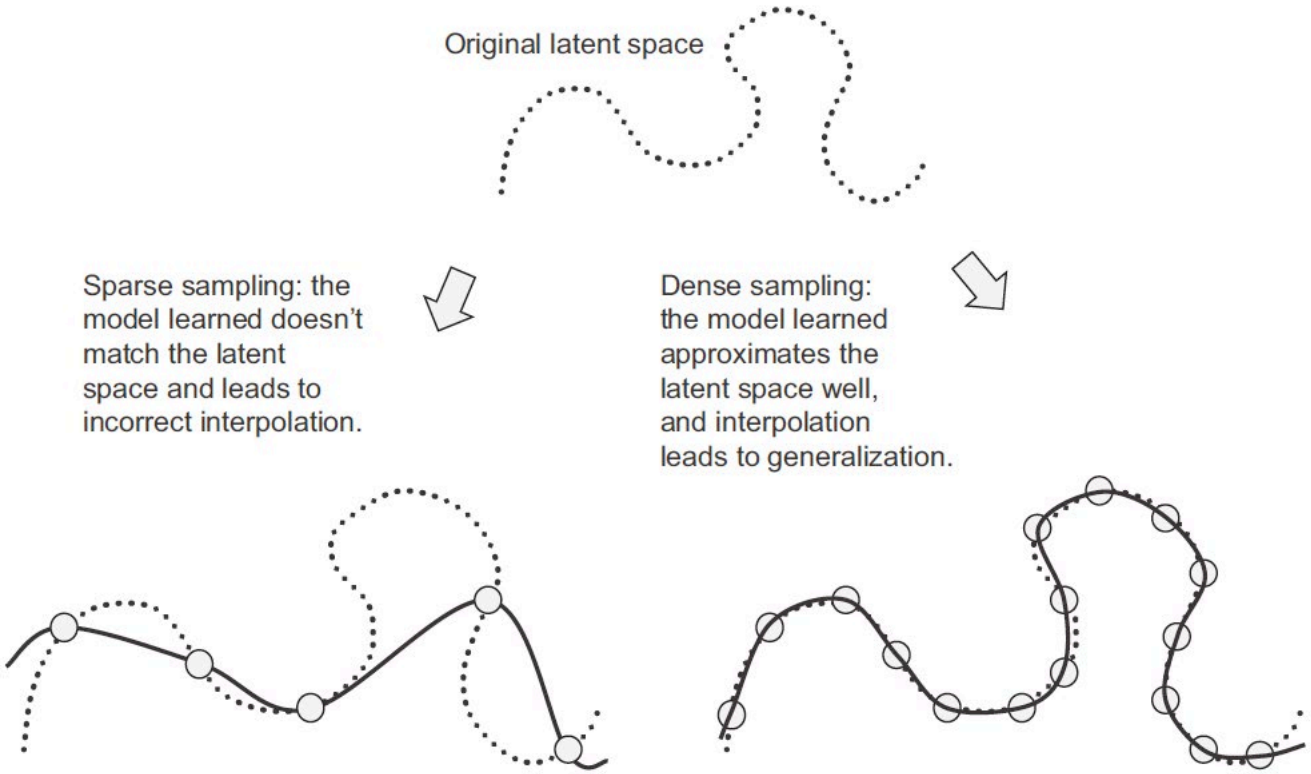
Parameters and Hyperparameters

Aspect	Parameters	Hyperparameters
Definition	Variables learned from training data	Configurations set before training
Examples	Weights in neural networks, coefficients in linear regression	Learning rate, number of trees in a random forest, batch size in neural networks
Computed by	Learning algorithm (SGD, or Tree growth)	Set manually or through (brute force or optimized) search techniques (Cross validation, Grid search)
Influence	Directly affect model predictions	Affect the training process and model performance

When to stop? - Training and Validation Error



How much data is needed?



Regularization to avoid overfitting

- Capacity of a neural network is its tendency to overfit
- More the number of parameters, the more its capacity (a.k.a degrees of freedom)
- Regularization in a neural network is a mathematical trick to penalize the neural network such that it reduces the capacity to overfit
- Typically norm-based regularization in which a penalty term (consisting of the parameters) is added to the loss
- Sometimes drop-out regularization in performed
- Batch normalization, layer normalization and early stopping all act as regularization

Tensorflow Playground

- https://playground.tensorflow.org/
- Exercises:
  1. Understand patterns learned by the neural network
  2. Understand different activation functions
     - What is the time to solution, how are the decision boundaries
  3. Local minima for small networks. Have only one hidden layer with three neurons
  4. Underfitting. One hidden layer with two neurons
  5. One layer with 8 neurons. Train multiple times. Network hardly gets stuck. Lesson: Large networks seldom get stuck in local minima, and even if they do, it is close to the global minima
  6. Vary other parameters and learn.

Department of Computational and Data Sciences

Week 03

भारतीय विज्ञान संस्थान

- Additional Material: Advanced gradient descent algorithms, Initialization, Stopping

Faster Optimizers

Deepak Subramani
Assistant Professor
Dept. of Computational and Data Science
Indian Institute of Science Bengaluru

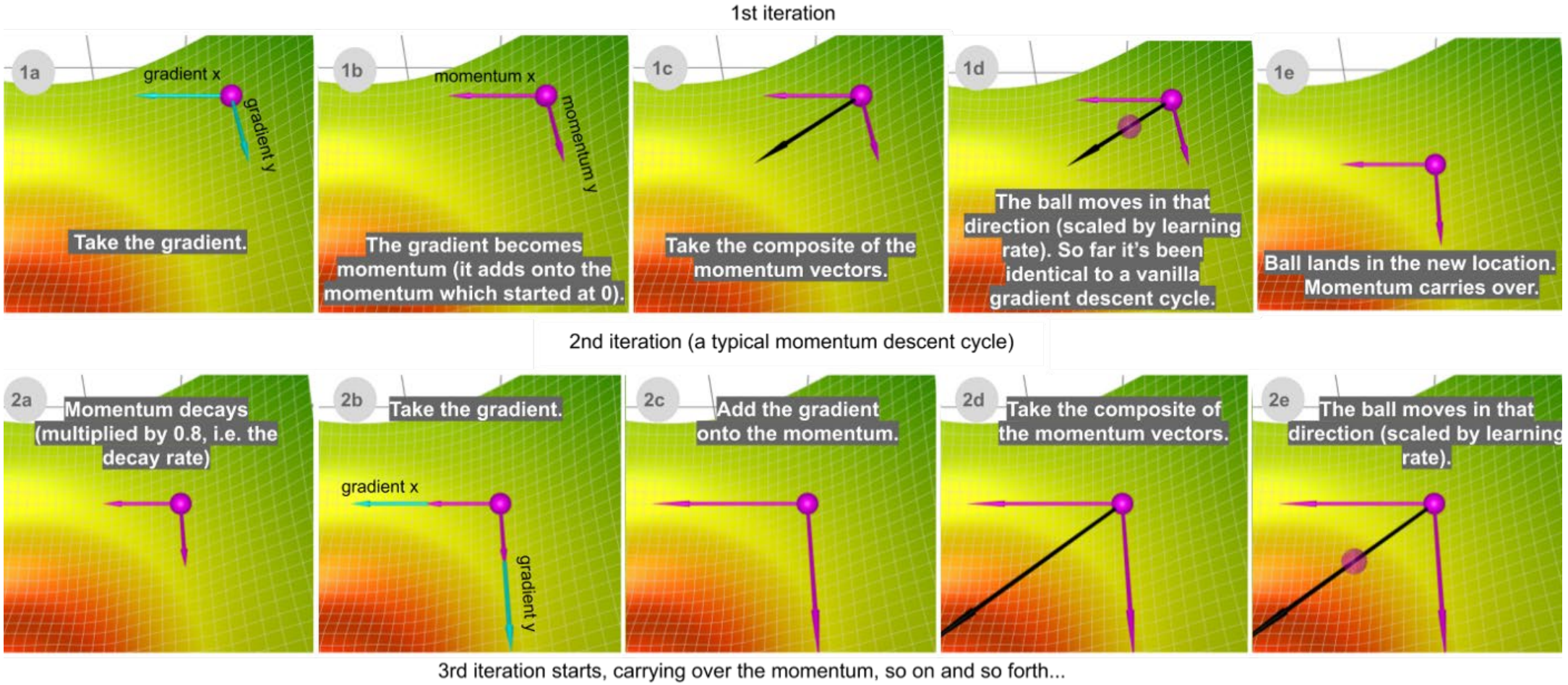
Momentum optimization

- Recall Gradient Descent iteration
  - $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
  - The obj. fn $J(\theta)$ is the loss evaluated on a mini batch
- GD cares only about the local gradient, it does not care about where the gradients are evaluated
- But imagine a ball rolling over a smooth surface, as the ball starts moving it's velocity increases – it accelerates!
- Force is the rate of change of momentum which is related to acceleration
- Momentum algorithm maintains a momentum vector that gets updated in every iteration
- Parameters are updated by the momentum vector
- Momentum Algorithm
  - $\boldsymbol{m} \leftarrow \beta \boldsymbol{m} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
  - $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{m}$

Blue Ball – GD
Purple Ball – Momentum Optimization
Blue vectors – gradients
Purple Vectors – Momentum

Simulator: https://github.com/lilipads/gradient_descent_viz

Momentum Step-By-Step



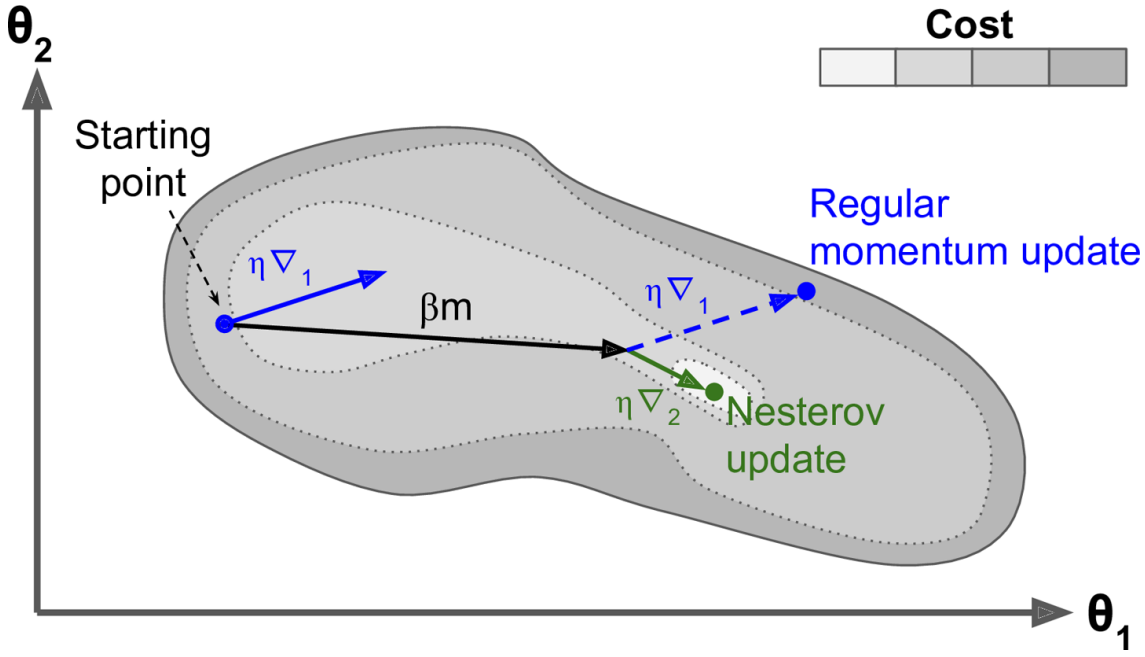
Source: https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c

Momentum: Points to Note

- β is a hyperparameter called the momentum which is set between 0 (high friction) and 1 (no friction)
- Typical Momentum value is 0.9
- If gradient remains constant, the maximum size of weight update (called as the "terminal velocity") is given by
  - $term.vel. = \frac{1}{1-\beta}\eta\nabla J(\theta)$
  - If $\beta = 0.9$, then Momentum is roughly 10 times faster than gradient descent
- GD comes down steep sloes fast, but struggle near valleys
- Momentum comes down faster and faster and reaches the valley
- In deep networks without BN, upper layers have inputs with different scales, so using momentum helps
- Momentum makes the optimizer overshoot and oscillate, hence adding a bit of friction helps
- Keras: optimizer = keras.optimizer.SGD(learning_rate=0.001, momentum=0.9)
- Yet another hyperparameter!!! – But usually 0.9 works, no need for GridSearchCV

Nesterov Accelerated Gradient

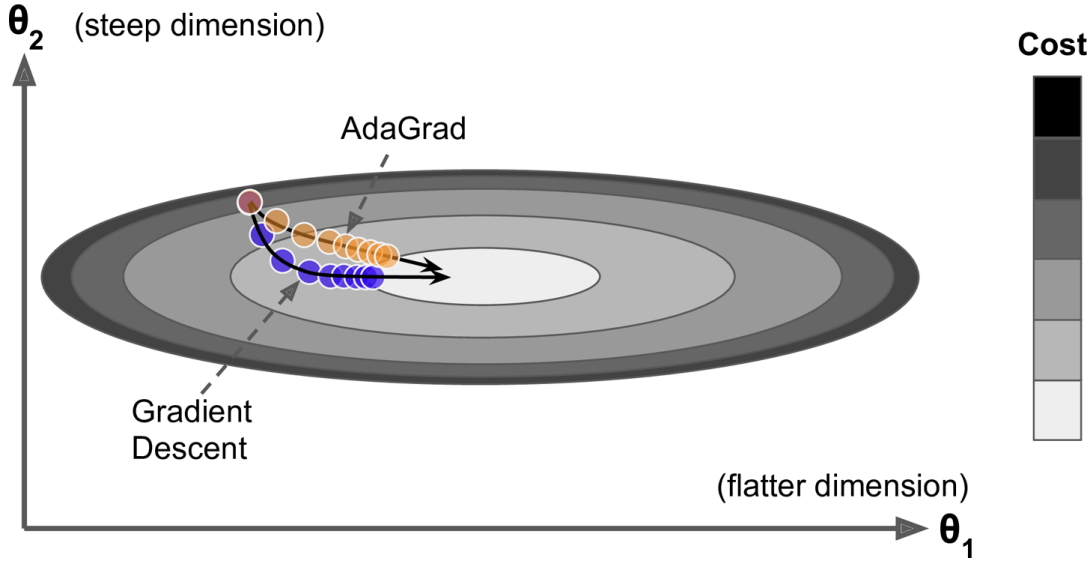
- A modification to momentum proposed by Yurii Nesterov in 1983
- Instead of using the gradient at the current parameter value, use the gradient of the loss at a point slightly ahead ($\boldsymbol{\theta} + \beta \boldsymbol{m}$)
- Nesterov Accelerated Gradient (NAG) or Nesterov Momentum Optimization Algorithm
  - $\boldsymbol{m} \leftarrow \beta \boldsymbol{m} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta} + \beta \boldsymbol{m})$
  - $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{m}$
- **Why is this small tweak important?**
  - **In general momentum will be pointing in the right direction**
  - **So it is slightly more accurate to use the gradient farther ahead (see fig)**
  - https://distill.pub/2017/momentum/



[Geron Fig 11-6]

AdaGrad

- In objective functions that have an elongated bowl shape
  - One axis is a flatter dimension, and another is a steep dimension
- GD will be more inclined to move in the steeper dimension
- AdaGrad modifies the gradient by scaling down the vector along the steepest dimensions
- AdaGrad Algorithm:
  - $\boldsymbol{s} \leftarrow \boldsymbol{s} + \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \otimes \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ [element-wise mul]
  - $\theta \leftarrow \theta - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \oslash \sqrt{s + \epsilon}$ [element-wise div]
- In effect, the algorithm itself changes the learning rate per dimension
  - Hence it is an adaptive learning rate algorithm
- AdaGrad is not used in practice as it stops too early while training neural networks
- But AdaGrad idea is useful to understand other adaptive learning rate optimizers



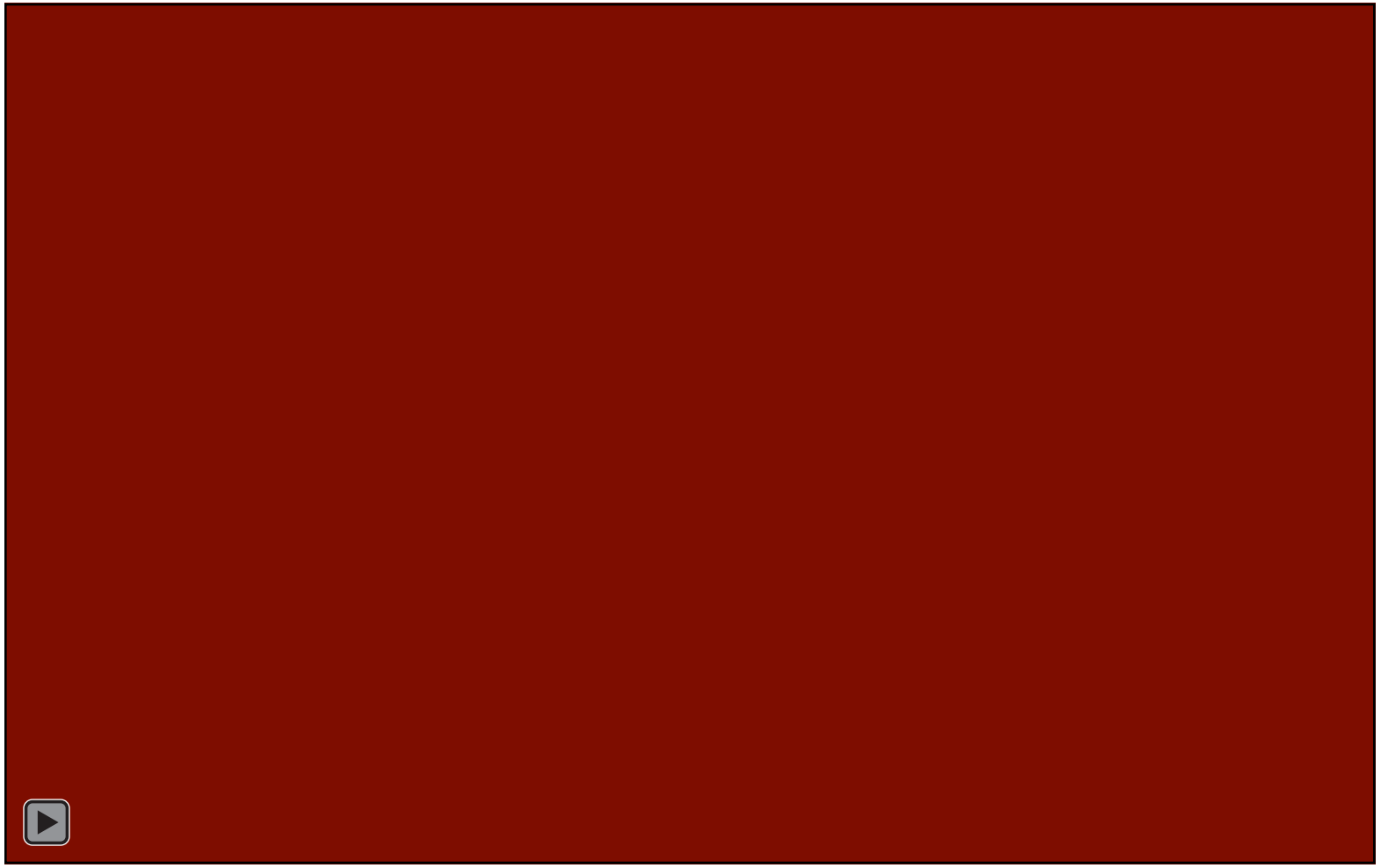
Adagrad corrects its direction earlier than GD for this bowl shaped function

Geron Fig 11-7

RMSProp

- RMSProp fixes the risk of early slowing down showed by AdaGrad
- RMSProp accumulates gradients of only the most recent iterations using an exponential decay mechanism
- RMSProp Algorithm
  - $\boldsymbol{s} \leftarrow \rho \boldsymbol{s} + (1-\rho)\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \otimes \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ [element-wise mul]
  - $\theta \leftarrow \theta - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \oslash \sqrt{s+\epsilon}$ [element-wise div]
- The decay rate ρ is usually set to 0.9, and usually doesn't need tuning
- Keras: optimizer = keras.optimizer.RMSprop(learning_rate=0.01, rho=0.9)
- Trivia: RMSProp was never published. Hinton introduced it in his Coursera class on Neural Networks. So citation is done as "slide 29 in lecture 6"!!!
- RMSProp was the go to algorithm before Adam

RMSProp in Action



Green – RMSProp
Purple – Momentum

Poll

1. Which Algorithms uses adaptive learning rates?
   - RMSProp
   - Momentum
   - Nesterov
   - SGD
2. Which of the following is TRUE?
   - Momentum accelerated algorithm converges slower than SGD
   - Adaptive Learning rate helps to converge faster
   - RMSProp uses momentum
   - AdaGrad uses momentum

Adam

- Adam stands for adaptive moment estimation
- It gets the speed from momentum and ability to adapt learning rate in different dimensions from RMSProp
- It keeps track of exponentially decaying average of past gradients – Momentum
- It keeps track of an exponentially decaying average of past squared gradients (second moment) – RMSProp
- Adam Algorithm:
  1. $m \leftarrow \beta_1 m - (1 - \beta_1)\nabla_\theta \mathrm{J}(\theta)$
  2. $s \leftarrow \beta_2 s + (1 - \beta_2)\nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta)$
  3. $\hat{m} \leftarrow \frac{m}{1-\beta_1^t}$
  4. $\hat{s} \leftarrow \frac{s}{1-\beta_2^t}$
  5. $\theta \leftarrow \theta + \eta \hat{m} \oslash \sqrt{\hat{s} + \epsilon}$
- β_1, β_2 are hyperparameters between 0 and 1. Usually $\beta_1 = 0.9, \beta_2 = 0.99$
- $\epsilon = 10^{-7}$ a smoothing term to avoid division by zero
- t is the iteration count. Steps 3,4 are needed as m, s are initialized to zero and initially a boost is needed

Adam

- Adam is an adaptive learning rate algorithm
- It requires less tuning of η
- The default value can be used directly
- Keras: optimizer = tf.keras.optimizers.Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-07, amsgrad=False, name='Adam')
- Amsgrad: Whether the AMSGRad version of Reddi et al 2018 is used (https://openreview.net/pdf?id=ryQu7f-RZ)

Variants of Adam

- AdaMax
  - Replace step 2 with $s \leftarrow \max(\beta_2 s, \nabla_\theta J(\theta))$
  - Adam scales down the parameter update by the l_2 norm, AdaMax does it by the l_∞ norm
  - In theory, AdaMax is more stable, but in practice it depends on the dataset
  - Usuaully Adam wins, but if Adam has some problem on a particular task you are dealing with, then AdaMax helps
- Nadam
  - Adam with Nesterov trick
  - But in most situations, Adam or RMSProp works well, Nadam is not needed
- AdamW
  - Adam with a weight decay
  - When l2 regularization is used, AdamW is found to be better than Adam with l2 regularization
  - For image classification and character RNN tasks, AdamW generally has a slight edge
  - But even for these, SGD+Momentum gives better generalization empirically
  - See the paper: https://arxiv.org/pdf/1711.05101.pdf

Word of Caution

- Adaptive optimization methods are great, but they can overfit and give poor generalization in some datasets
- If you are in such a situation, keep this caution in mind and try NAG as well

Summary

Table 11-2 compares all the optimizers we've discussed so far (* is bad, ** is average, and *** is good).

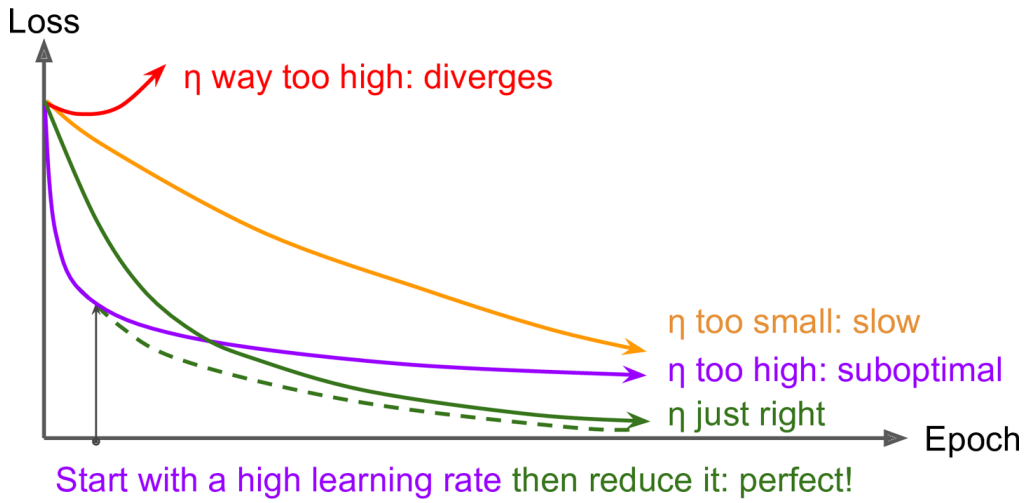
Table 11-2. Optimizer comparison

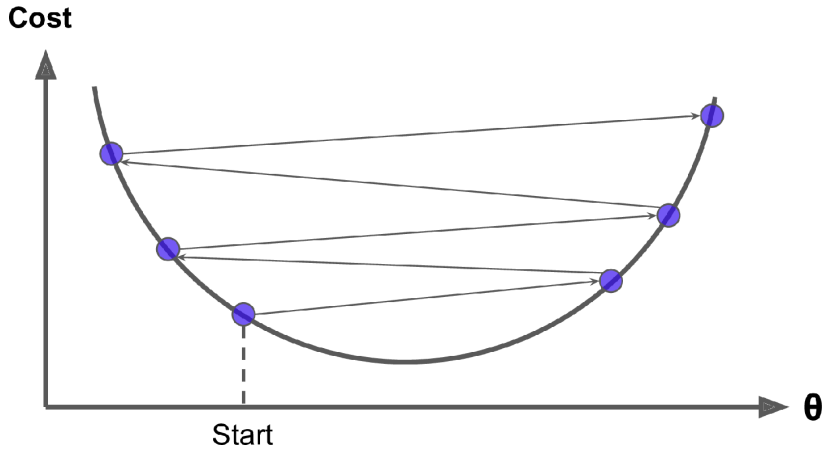
Class	Convergence speed	Convergence quality
`SGD`	*	***
`SGD(momentum=...)`	**	***
`SGD(momentum=..., nesterov=True)`	**	***
`Adagrad`	***	* (stops too early)
`RMSprop`	***	** or ***
`Adam`	***	** or ***
`Nadam`	***	** or ***
`AdaMax`	***	** or ***

[Geron 2nd Edition 2019]

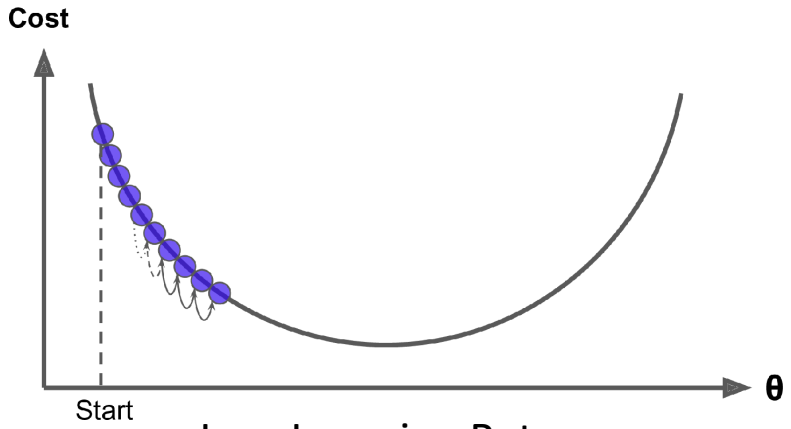
Learning Rate Scheduling

- Too low LR will eventually converge but it will be slow
- Too high LR will diverge
- Using learning rate scheduling will help





High Learning Rate



Low Learning Rate

Scheduling Options

- Power scheduling
  - $\eta(t) = \frac{\eta_0}{\left(1+\frac{t}{s}\right)^c}$ t is the iteration count, s is the number of steps after it η becomes half
- Exponential scheduling
  - $\eta(t) = \eta_0 0.1^{t/s}$
- Piecewise constant scheduling
  - Use a constant learning rate of a number of epochs
- Performance scheduling
  - Measure the validation error every N steps and reduce the learning rate by a factor
- 1cycle scheduling
  - Start by increasing learning rate from η_0 to η_1 linearly and then reduce to η_0 linearly
  - Maximum learning rate is chosen by finding the optimal learning rate and using initial rate as 1/10
  - Momentum can also be subject to 1cycle scheduling
  - Smith 2018 showed that 1cycle scheduling achieved as much accuracy after 100 epochs as with 800 through standard approach [arXiv:1803.09820]

Learning Rate Scheduling

- Exponential scheduling and performance scheduling work very well
- 1cycle works even better and one can use one of the three options well
- Power scheduling is easiest to implement in Keras
  - Keras: optimizer = keras.optimizer.SGD(learning_rate = 0.01, decay = 1e-4)
- Others have to be implemented via callbacks

Regularization

- Neural Networks are prone to overfitting
- They have tens of thousands (even millions in some networks) of parameters
- The flexibility to fit complex datasets comes with the disadvantage that they can overfit
- Three popular regularization techniques are used:
  - ℓ_1 and ℓ_2 regularization
    - Set kernel_regularizer and specify the l1 or l2 factors
  - Dropout
    - At every training step, each neuron has a probability p (called dropout rate) of being temporarily "dropped out"
    - After training, neurons don't get dropped any further
  - Max-norm regularization
    - Scale weights if their norm exceeds a max norm

Dropout Regularization: Points

- Neurons trained with dropout cannot rely on specific other neurons, they have to learn to be useful even when others are not present
- So they tens to become less sensitive to slight changes in the input and so generalize well
- Another view is that is a Neural Network has n paraemters there are virtually 2^n different options for some neurons to be active and others to be inactive
- So when 10,000 training steps are run, it is like there are 10,000 networks
  - The analogy is not perfect, but it is approximately true
- In practice, dropout is applied to the top 1 to 3 layers – empirically this is found to work well
- Important technical detail
  - Suppose p=0.5, then during training only half the neurons are active
  - Consider one neuron, during training only 50% of its inputs are active
  - But during testing, all are active! Roughly the input will now be 2x
  - So we must multiply all weights with (1-p) after training, or divide weights by (1-p) during training
- Dropout makes comparing training and validation loss tricky. Make sure to evaluate without dropout (after training of course)
- Alpha dropout is needed if you use SELU

Summary and Practical Guidelines

Table 11-3. Default DNN configuration

Hyperparameter	Default value
Kernel initializer	He initialization
Activation function	ELU
Normalization	None if shallow; Batch Norm if deep
Regularization	Early stopping (+ℓ_2 reg. if needed)
Optimizer	Momentum optimization (or RMSProp or Nadam)
Learning rate schedule	1cycle

Table 11-4. DNN configuration for a self-normalizing net

Hyperparameter	Default value
Kernel initializer	LeCun initialization
Activation function	SELU
Normalization	None (self-normalization)
Regularization	Alpha dropout if needed
Optimizer	Momentum optimization (or RMSProp or Nadam)
Learning rate schedule	1cycle

Normalize Inputs

Get good data, use unsupervised pretraining, reuse layers

Have Fun!!!