Universidade de São Paulo
Escola Politécnica - Engenharia de Computação e Sistemas Digitais

BackPropagation, Weights Initialization, Learning Rate and Optimizers

Prof. Artur Jordão

Preliminaries

Introduction

Preliminaries

- Gradient Descent (or its Stochastic version)
  - Iteratively reduces the error by updating the parameters (weights) in the direction that incrementally lowers the loss function

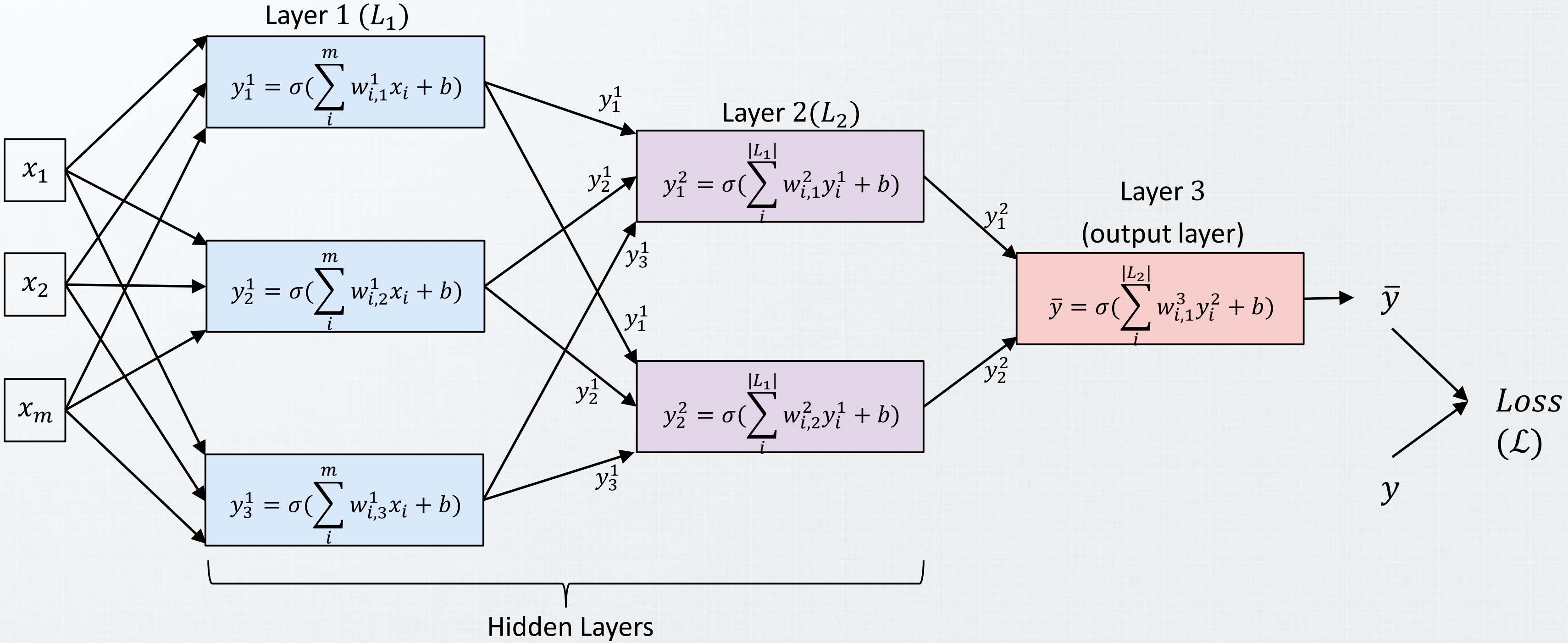
Gradient Descent Algorithm
$W \leftarrow$ Random values
while not converged do
for each $w_i \in W$ do
$w_i \leftarrow w_i - \eta \frac{\partial}{\partial w_i} \mathcal{L}(W)$

Stochastic Gradient Descent Algorithm
$W \leftarrow$ Random values
while not converged do
for each $w_i \in W$ do
$w_i \leftarrow w_i - \eta \frac{1}{\lvert\beta\rvert} \sum_{j=\beta_t}^{n} \frac{\partial}{\partial w_i} \mathcal{L}(W)$

Introduction

Preliminaries

- The MLP architecture poses an important issue
    - How can we update the weights of the Hidden layers? (Solution: Backprograpation)



Backpropagation

Preliminaries

- Backpropagation is an efficient algorithm for computing gradients on neural networks using the chain rule

- The idea is to traverse the network in **reverse order**, from the output to the input layer, according to the **chain rule** from calculus

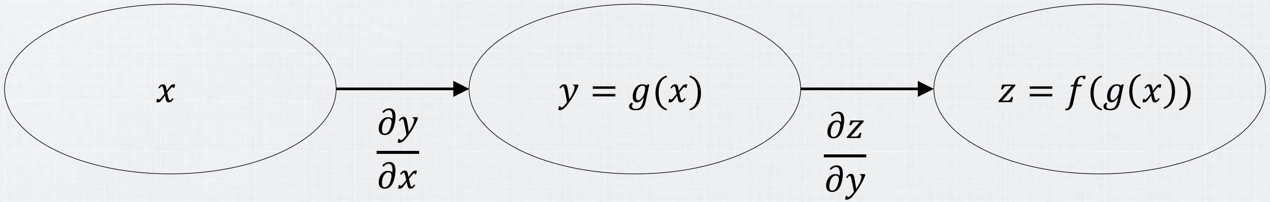
Chain Rule

Preliminaries

- Compute the derivatives of functions formed by composing other functions whose derivatives are known
  - Backpropagation is an algorithm that computes the chain rule

- Let x be a real number. Let f and g both be functions mapping from a real number to a real number. Suppose that $y = g(x)$ and $z = f(g(x)) = f(y)$

- The chain rule states that
  - $\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$

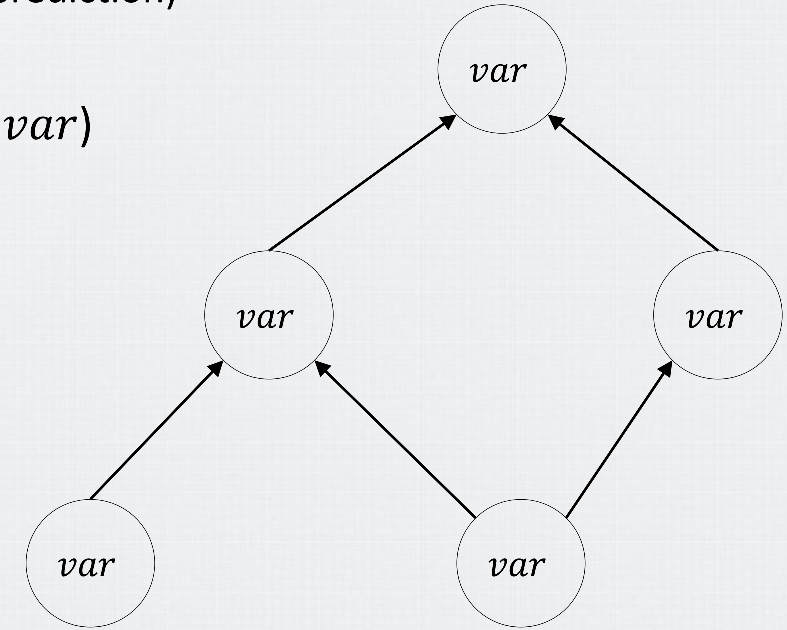


Computational Graph

Preliminaries

- To describe the backpropagation algorithm more precisely, it is helpful to have a more precise computational graph language
  - It allows to understand how a change in one variable brings change on the variable that depends on it (in particular y – the network prediction)

- Each node in the graph indicates a variable (var)
  - Scalar, vector, matrix, tensor, etc.
  - The result of an operation

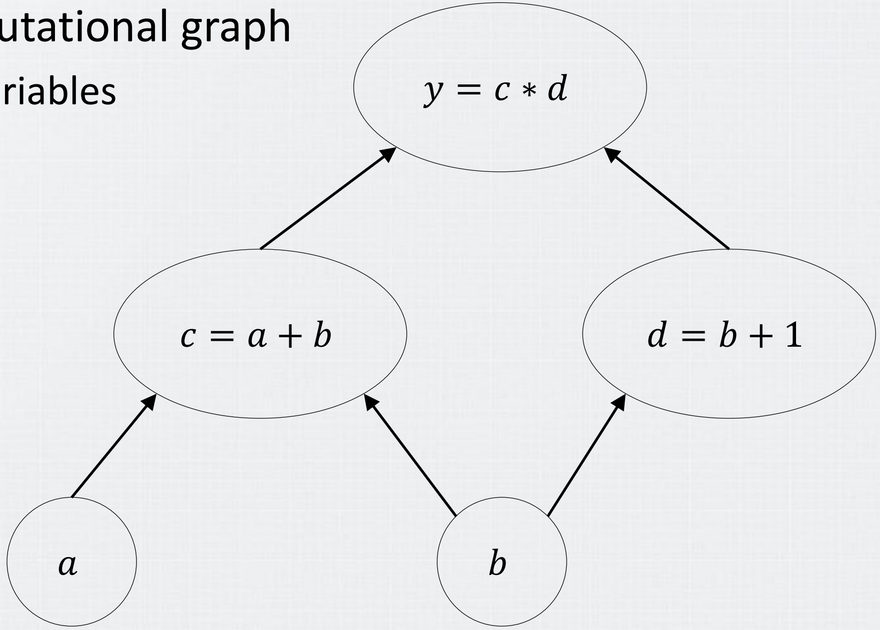


Computational Graph

Preliminaries

- Consider the following expression
  - $y = (a + b) * (b + 1)$

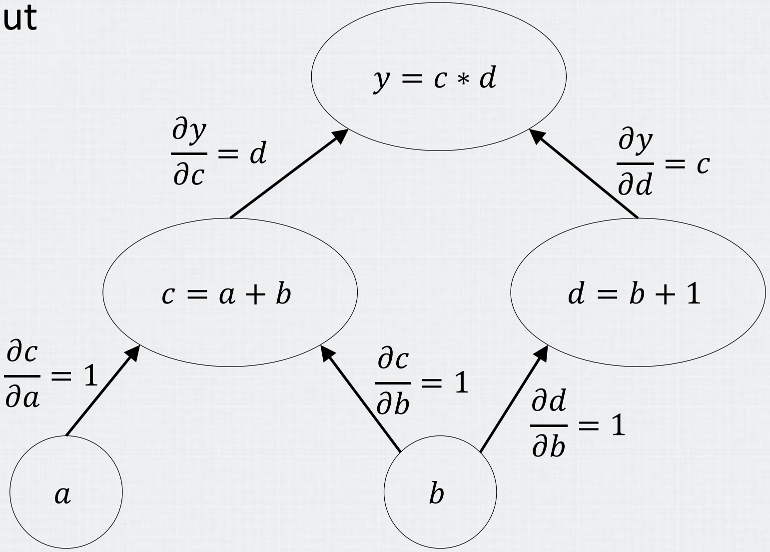
- Such expression has the following computational graph
  - Note that we can create operations as variables



Computational Graph

Preliminaries

- How does a change in one variable bring change in the variable that depends on it (in particular y)?
  - For example, if a affects c how does it affect y: If we make a slight change in the value of a how does y change?
  - Remember that the derivative specifies how to scale a small change in the input in order to obtain the corresponding change in the output

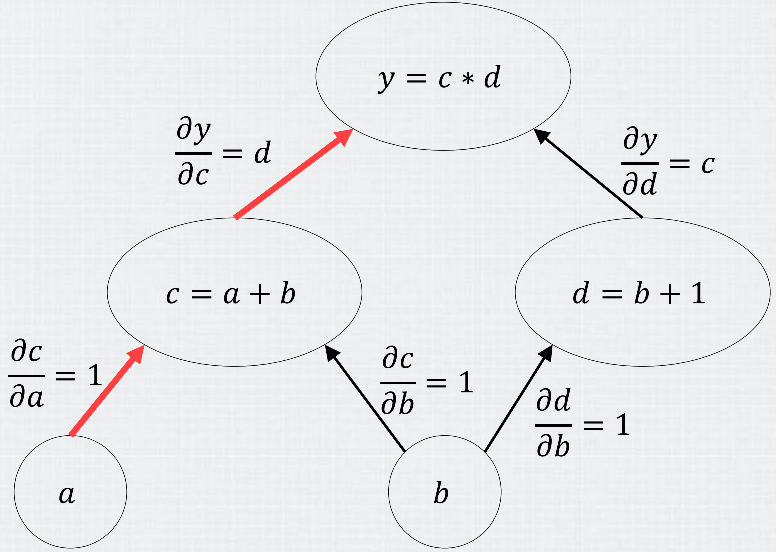


Computational Graph

Preliminaries

- How a affects y:

$$\frac{\partial y}{\partial a} = \frac{\partial y}{\partial c} \times \frac{\partial c}{\partial a} = d \times 1 = d$$



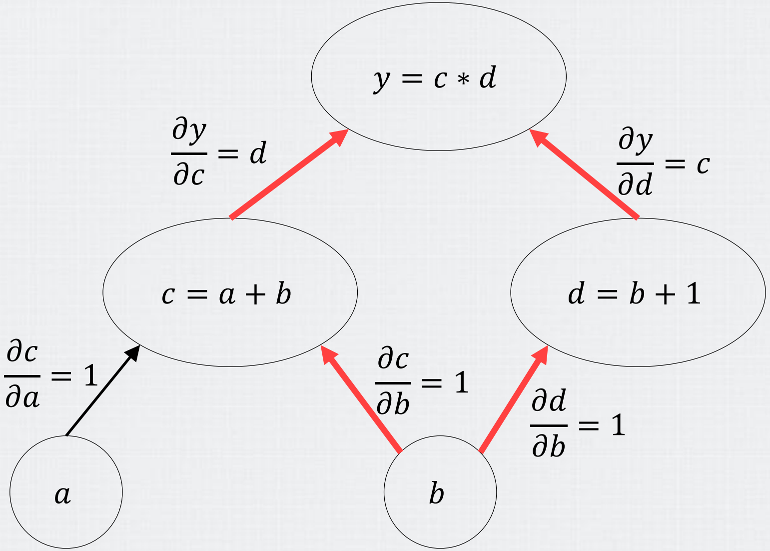
Computational Graph

Preliminaries

- How b affects y:

$$\frac{\partial y}{\partial b} = \frac{\partial y}{\partial d} \times \frac{\partial d}{\partial b} + \frac{\partial y}{\partial c} \times \frac{\partial c}{\partial b} = c \times 1 + d \times 1 = c + d$$

  - When two or more paths in a computational graph join at a node (such as b) we must **sum up** the product of gradients along all of these path



Backpropagation

Definitions

BackPropagation

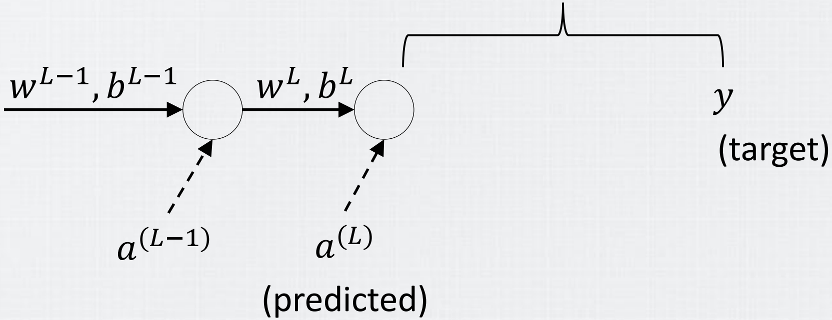
- Consider a simple neural network
  - Two layers with one neuron each

- Consider the loss $(\bar{y} - y)^2$

$a^{(L)}$

Cost for one sample (training example – sample 0)

$$C_0 = \left(a^{(L)} - y\right)^2$$



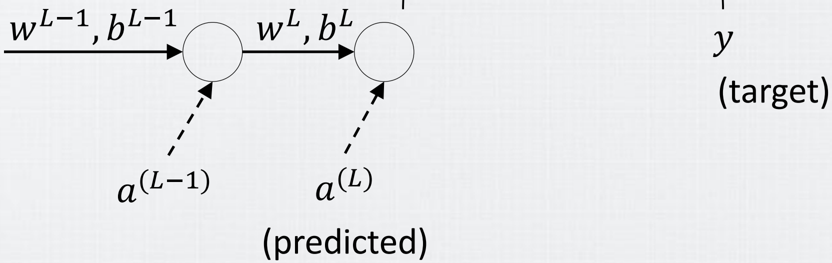
Definitions

BackPropagation

- $a^{(L)} = \sigma\left(\underbrace{w^{(L)}a^{(L-1)} + b^{(L)}}_{z^{(L)}}\right)$
- Rewriting
  - $a^{(L)} = \sigma(z^{(L)})$
- Generalizing
  - $a^{(i)} = \sigma\left(z^{(i)}\right), 1 \leq i \leq L$

Cost for one sample (training example – sample 0)

$$C_0 = \left(a^{(L)} - y\right)^2$$

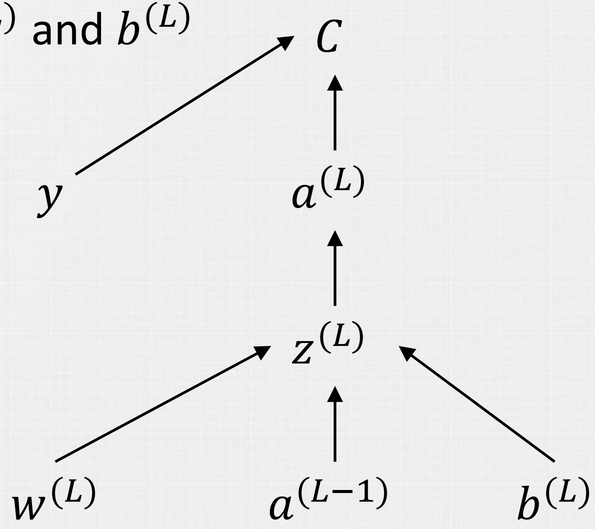


Problem Definition

BackPropagation

- We want to estimate the sensibility of cost (C) to small changes in w and b
  - In other words, the derivative of C w.r.t $w^{(L)}$ and $b^{(L)}$

- Formalizing
  - $\frac{\partial C}{\partial w^{(L)}}$
  - $\frac{\partial C}{\partial b^{(L)}}$

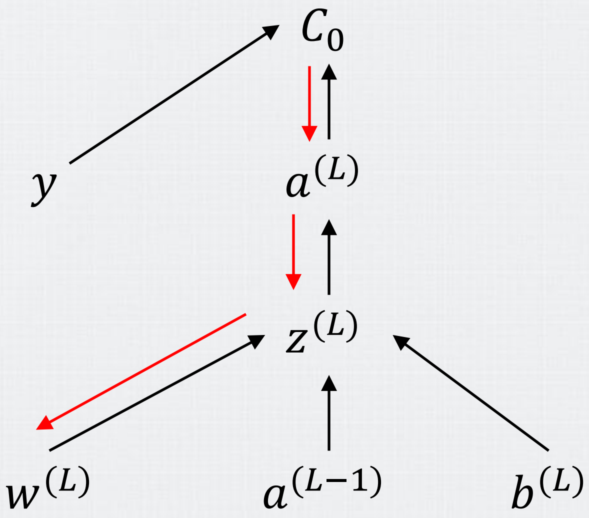


Weights (Last Layer)

BackPropagation

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}}$$

$$a^{(L-1)} \qquad \sigma'(z^{(L)}) \qquad 2(a^{(L)} - y)$$



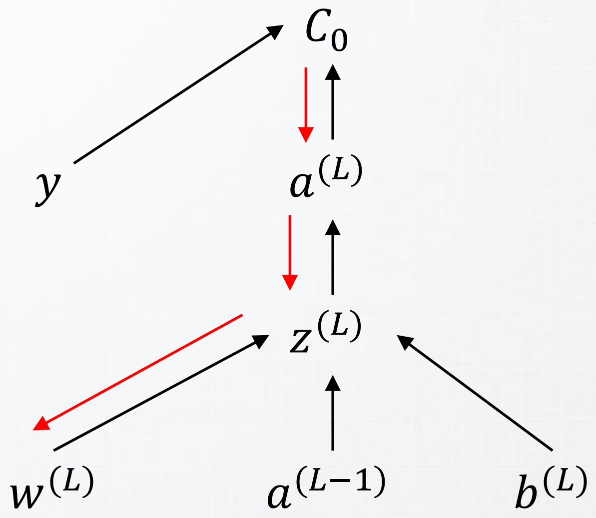
$$\frac{\partial C_0}{\partial w^{(L)}} = a^{(L-1)} \sigma'\left(z^{(L)}\right) 2\left(a^{(L)} - y\right)$$

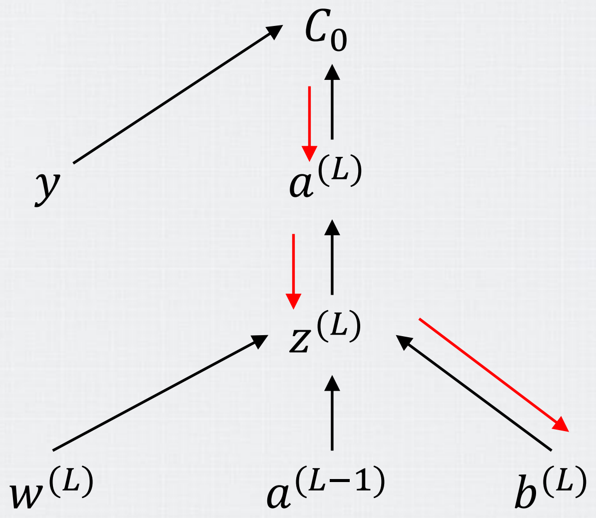
Generalizing across all (n) samples

$$\frac{\partial C}{\partial w^{(L)}} = \frac{1}{n} \sum_{i=0}^{n} \frac{\partial C_i}{\partial w^{(L)}}$$

Bias (Last Layer)

BackPropagation





$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}}$$

$$a^{(L-1)} \sigma'\left(z^{(L)}\right) 2\left(a^{(L)} - y\right)$$

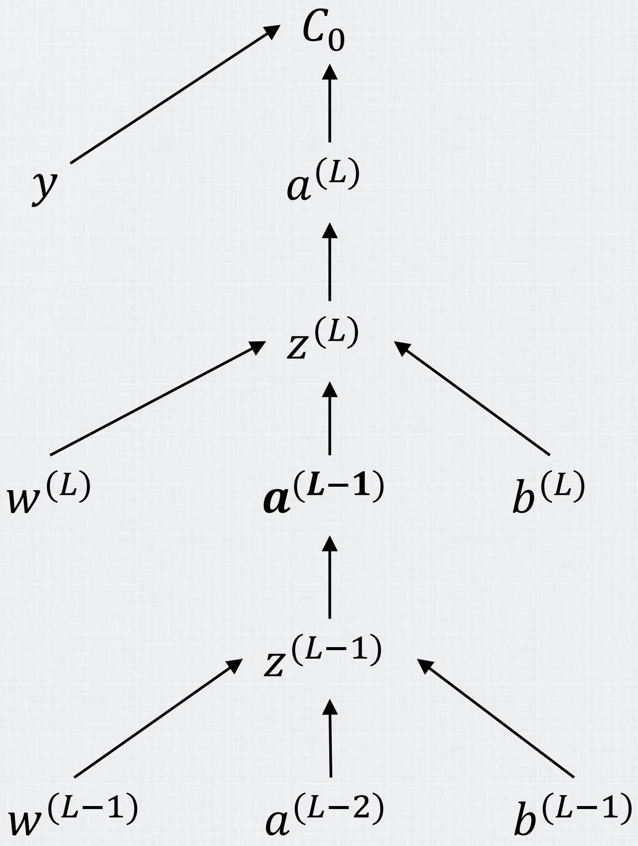
$$\frac{\partial C_0}{\partial b^{(L)}} = \frac{\partial z^{(L)}}{\partial b^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}}$$

$$= \mathbf{1} \sigma'\left(z^{(L)}\right) 2\left(a^{(L)} - y\right)$$

Hidden Layers

BackPropagation

- We need $\frac{\partial C_0}{\partial a^{(L-1)}}$ to compute $\frac{\partial C_0}{\partial w^{(L-1)}}$



Multiple Neurons

BackPropagation

- $C_0 = \sum_{j=0} \left(a_j^{(L)} - y_j\right)^2$
- $z_j = w_{j0}^{(L)} a_0^{(L-1)} + w_{j1}^{(L)} a_1^{(L-1)} + w_{j2}^{(L)} a_2^{(L-1)} + b$
- $z_j = \sum\sum w_{jk}^{(L)} a_k^{(L-1)}$
- $\frac{\partial C_0}{\partial a_k^{(L-1)}} = \sum_{j=0} \frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C_0}{\partial a_j^{(L)}}$
- Generalizing: $\frac{\partial C}{\partial w_{jk}^{(L)}} = a_k^{(L-1)} \sigma'(z_j^{(L)}) \frac{\partial C}{\partial a_j^{(L)}}$

Last Layer

$2(a_j^L - y_j)$

(Squared error)

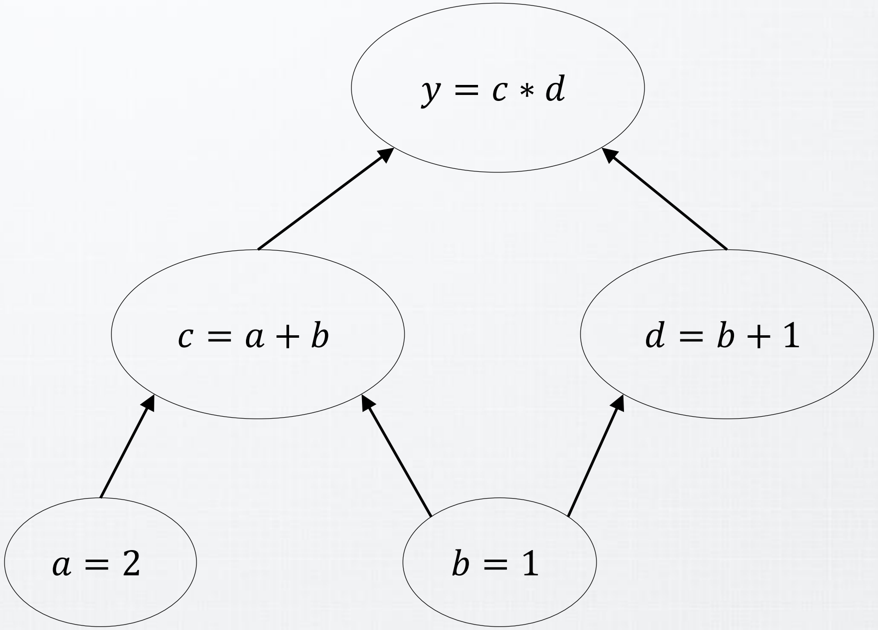
Hidden Layers

$$\sum_j w_{jk}^{(L+1)} \sigma'(z_j^{(L+1)}) \frac{\partial C}{\partial a_j^{(L+1)}}$$

Example Code

BackPropagation

- $y = (a + b) * (b + 1)$




```
import tensorflow as tf

a = tf.Variable([2.], dtype=tf.float32)
b = tf.Variable([1.], dtype=tf.float32)

with tf.GradientTape(persistent=True) as tape:
    tape.watch(a)
    tape.watch(b)
    c = a + b
    d = b + 1
    y = c * d
    loss = y

grad_a = tape.gradient(loss, a)
grad_b = tape.gradient(loss, b)

# Gradients are in grad_a e grad_b
print(grad_a.numpy())
print(grad_b.numpy())
```

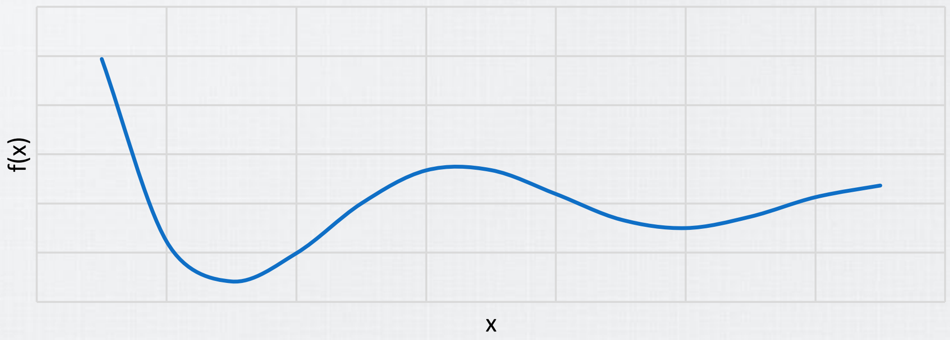
Initialization

Introduction

Initialization

- So far, we have learned how to forward and update the weights iteratively (i.e., SGD)
  - Thus, it requires the user to specify some initial point (parameters) from which to begin the iterations

- Training deep models is a sufficiently difficult task
  - Most algorithms are strongly affected by the choice of initialization
  - The initial point can determine whether the algorithm converges at all



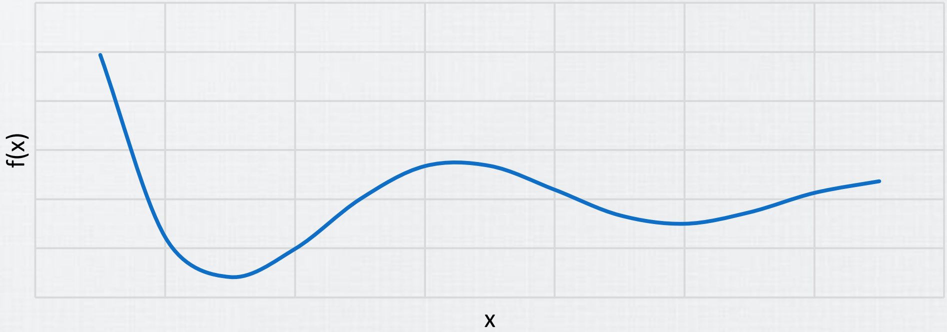
Introduction

Initialization

- With some initial points being so unstable that the algorithm encounters numerical difficulties and fails altogether
  - Vanishing and exploding gradient problem

- The initial point can affect the generalization

- A further difficulty is that some initial points may be beneficial from the **viewpoint of optimization** but detrimental from the **viewpoint of generalization**



Property

Initialization

- Perhaps the only property known with complete certainty is that the initial parameters need to **break symmetry** between different units

- If two hidden units with the same activation function are connected to the same inputs, then these units must have different initial parameters
  - Otherwise, a deterministic learning algorithm applied to a deterministic cost and model will constantly update both of these units in the same way – redundant units

- The goal of having each unit compute a different function motivates **random initialization** of the parameters

Initialization Strategies

Initialization

- Modern initialization strategies are simple and heuristic
  - Popular strategies: Xavier and Kaiming He

- Xavier (Glorot et al., 2010)
  - $W \sim U\left(-\frac{\sqrt{6}}{\sqrt{n_l + n_{l+1}}}, \frac{\sqrt{6}}{\sqrt{n_l + n_{l+1}}}\right)$, where n_l denotes the number of neurons in layer l

- Kaiming He (He et al., 2015)
  - $W \sim N\left(0, \frac{2}{n_l}\right)$, where n_l denotes the number of neurons in layer l

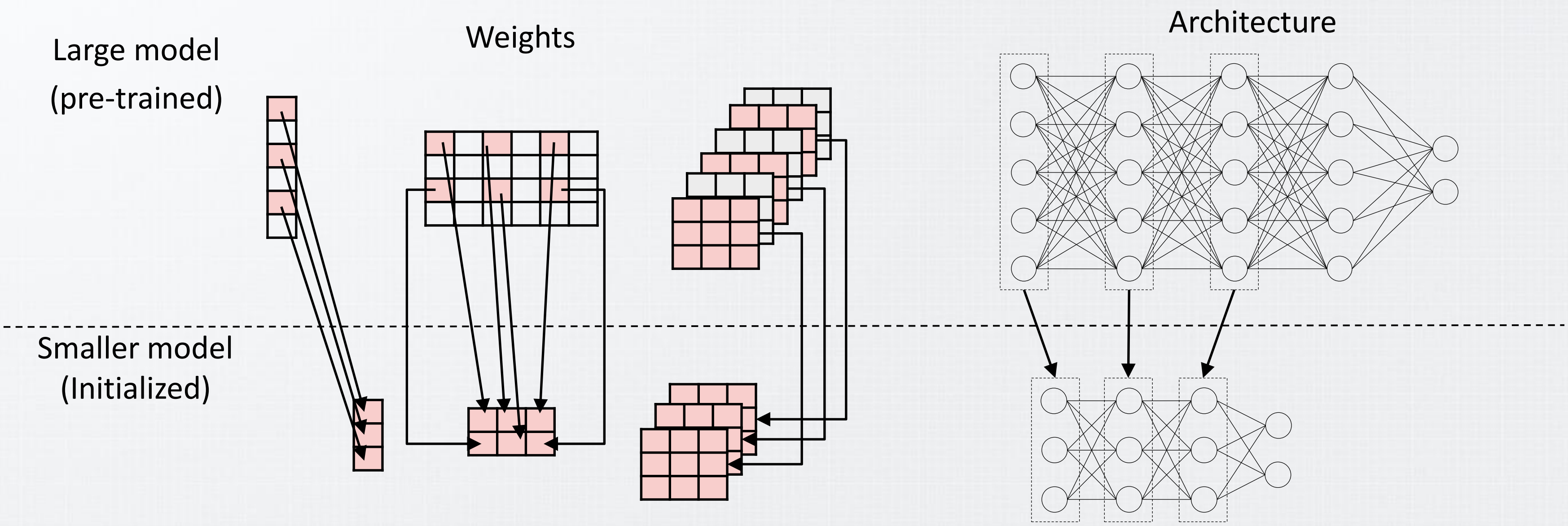
Glorot et al. *Understanding the difficulty of training deep feedforward neural networks*. International Conference on Artificial Intelligence and Statistics (AISTATS), 2010

He et al. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. International Conference on Computer Vision (ICCV), 2015

Initialization Strategies

Initialization

- Weight selection (Xu et al., 2024)
  - It selects weights from a pre-trained large model to initialize a smaller one
  - The method leverages the variety of pre-trained models that are now readily available



Xu et al. *Initializing Models with Larger Ones*. International Conference on Learning Representations (ICLR), 2024

Learning Rate and
Learning Rate Schedulers

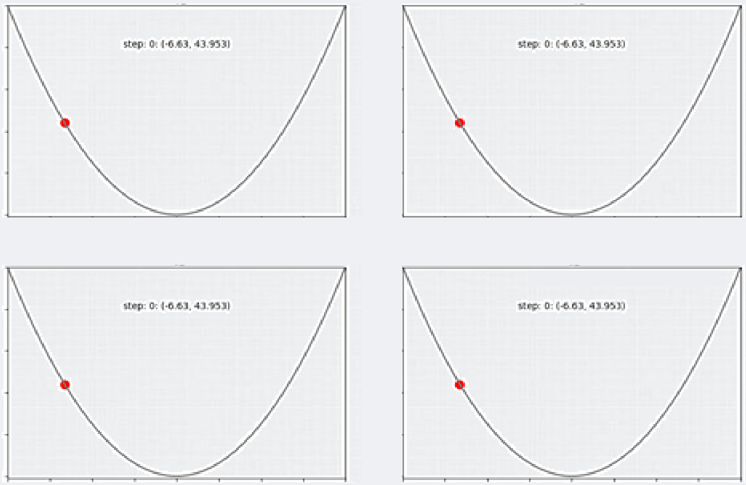
Introduction

Learning Rate

- Remember that the learning rate (η) controls the rate of learning
  - How fast/slow we update the weights

- If η is too large, optimization diverges

- If η is too low, learning proceeds slowly



Gradient Descent Algorithm

$W \leftarrow$ Random values

While not converged do

for each $w_i \in W$ do

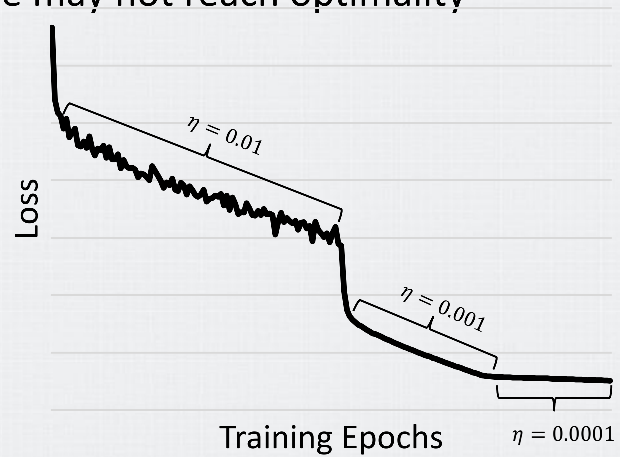
$$w_i \leftarrow w_i - \boldsymbol{\eta} \frac{\partial}{\partial w_i} \mathcal{L}(W)$$

Introduction

Learning Rate

- The learning rate may be chosen by a trial and error scheme
  - It is usually best to choose it by monitoring learning curves that plot the objective function as a function of time (epochs)

- One hypothesis is that large rates help move the optimization over large energy barriers while small rates help converge to a local minimum
  - Therefore, if the learning rate remains unchanged we may not reach optimality



Learning Rate Scheduler

Learning Rate

- It is often useful to lower or adjust the learning rate as the training progresses

- Step decay (He et al., 2016)
  - Drop the learning rate by a multiplicative factor γ (typically 0.1) after every d epochs
  - $\eta = \eta * \gamma$

- Exponential (Li et al. 2020)
  - $\eta_t = \gamma^t$
  - t indicates the t-th epoch

He et al. *Deep Residual Learning for Image Recognition*. Computer Vision and Patttern Recognition (CVPR), 2016

Li et al. *Budgeted Training: Rethinking Deep Neural Network Training under Resource Constraints*. International Conference on Learning Representations (ICLR), 2020

Learning Rate Scheduler

Learning Rate

- Cosine (or cosine annealing) (Loshchilov et al., 2017)
  - $\eta_t = \alpha + \frac{1}{2}(1-\alpha)(1+\cos\left(\pi \frac{t}{T}\right))$
  - α specifies a lower bound (default is zero)

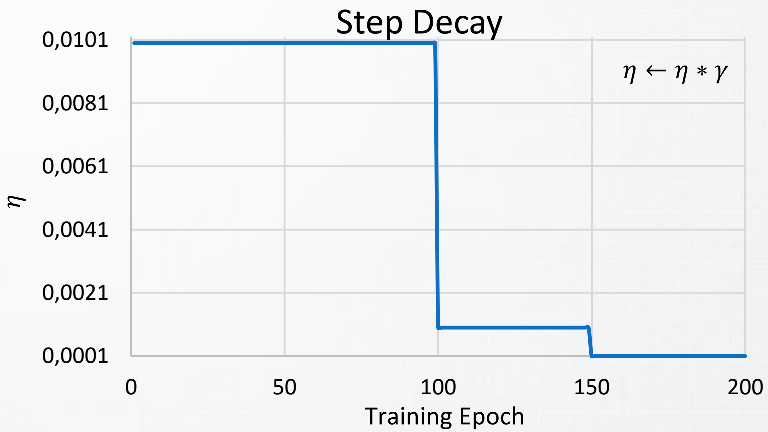
- Linear (Li et al., 2020)
  - $\eta_t = 1 - \frac{t}{T}$

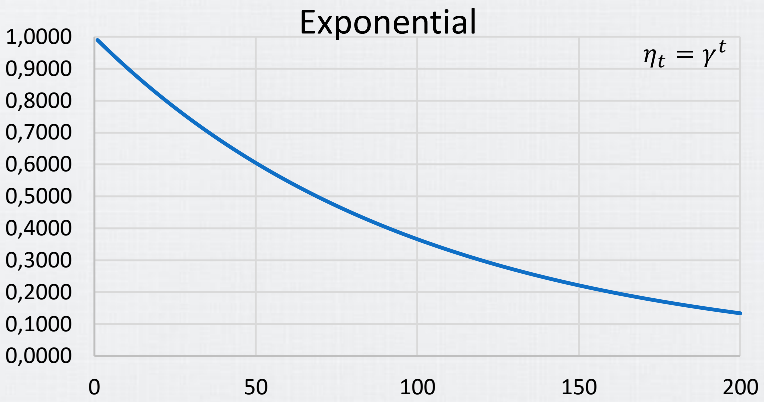
Loshchilov et al. *SGDR: Stochastic Gradient Descent with Warm Restarts*. International Conference on Learning Representations (ICLR), 2017

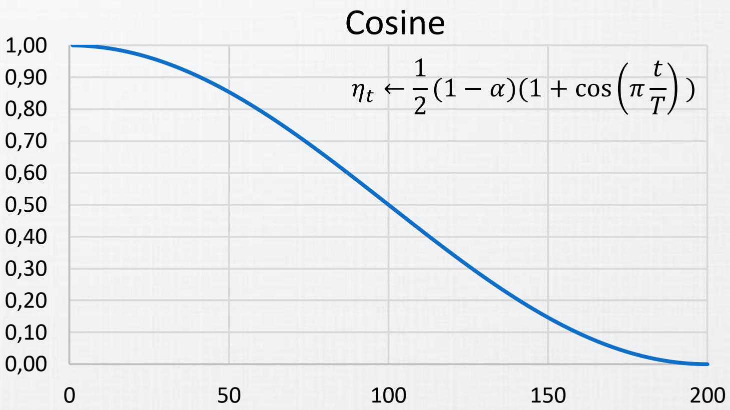
Li et al. *Budgeted Training: Rethinking Deep Neural Network Training under Resource Constraints*. International Conference on Learning Representations (ICLR), 2020

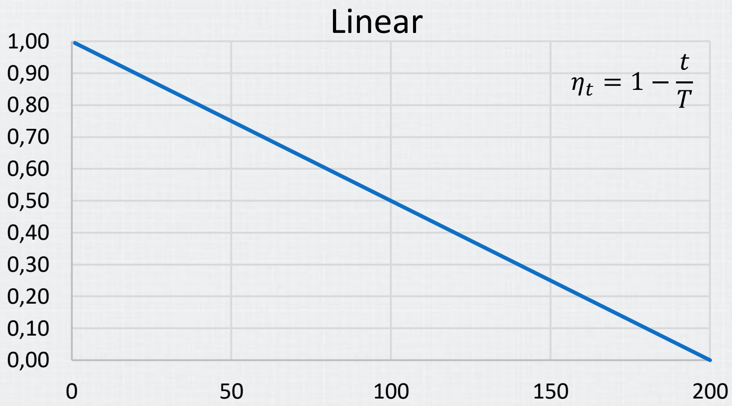
Learning Rate Scheduler

Learning Rate









Relationship with Batch Size

Learning Rate

- In practice, most works **fix the batch size**, β, during training and **decay the learning rate**

- Smithet et al. (2018) showed that increasing batch sizes at a linear rate during training is as effective as decaying learning rates
  - Therefore, it is equally effective (in terms of training/test error reached) to gradually increase batch size during training while fixing the learning rate

Smithet al. *Don't Decay The Learning Rate, Increase The Batch Size*. International Conference on Learning Representations (ICLR), 2018

Optimizers

Introduction

Optimizers

- The last ingredient involving the development of a neural network is how to find the parameter values that minimize this loss

- The process is to choose initial parameter values (initialization) and then iterate the following two steps:
  I. Compute the derivatives (gradients) of the loss with respect to the parameters
  II. Adjust the parameters based on the gradients to decrease the loss

- After repeating this process many iterations (epochs), we hope to reach the overall minimum of the loss function

Introduction

Optimizers

- The goal of an optimization algorithm is to find parameters θ that minimize the loss
    - $\theta^* = argmin_\theta(\mathcal{L}(\theta))$

- There are many families of optimization algorithms

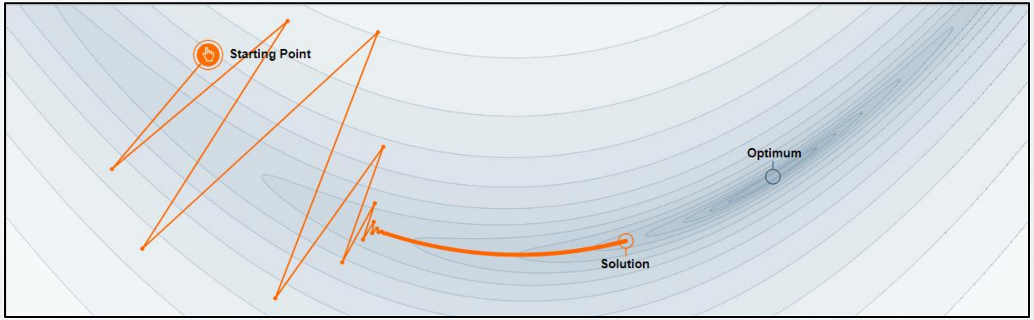
- Standard methods for training neural networks are iterative
    - Iterative means that they adjust the parameters repeatedly in such a way that the loss decreases

Iterative Optimization
$\theta \leftarrow$ Xavier or Kaiming He Initialization
While not converged do
(i) Compute the derivatives (gradients) of the loss w.r.t the parameters
(ii) Adjust the parameters based on the gradients to decrease the loss

Momentum

Optimizers

- Drawbacks in SGD optimization
  - Gradients estimated from **small batches** will often have **high variance** and may point in entirely the wrong direction
  - Easily fooled by small adversarial perturbations and fail to provide adequate uncertainty estimates (Pagliardini et al., 2023)



Pagliardini et al. *Agree To Disagree: Diversity Through Disagreement for Better Transferability*. International Conference on Learning Representations (ICLR), 2023

Momentum

Optimizers

- The momentum algorithm accumulates a running average of **past gradients** and continues to move in their direction

- The algorithm introduces a variable v (initialized with zero) that plays the role of velocity
  - It is the direction and speed at which the parameters move through parameter space
  - v accumulates the gradient elements ∇_θ

- $\alpha \in [0, 1)$ determines how quickly the contributions of previous gradients decay
  - Common values of α used in practice include 0.5, 0.9, and 0.99
  - $\alpha = 0$, we recover gradient descent

Update Rule

$$v \leftarrow \alpha v - \eta \nabla_\theta$$
$$\theta \leftarrow \theta + v$$

Momentum

Optimizers

- The larger α is relative to η, the more previous gradients affect the current direction

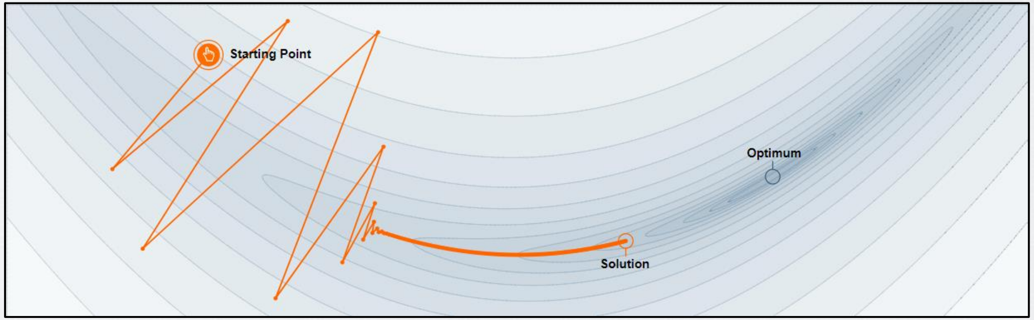
Stochastic Gradient Descent Algorithm with Momentum
$\theta \leftarrow$ Xavier or Kaiming He Initialization
While not converged do
$\quad g \leftarrow \frac{1}{\lvert\beta\rvert} \sum_{i=\beta_t}^{n} \nabla \mathcal{L}^i(\theta)$
$\quad v \leftarrow \alpha v - \eta g$
$\quad \theta \leftarrow \theta + v$

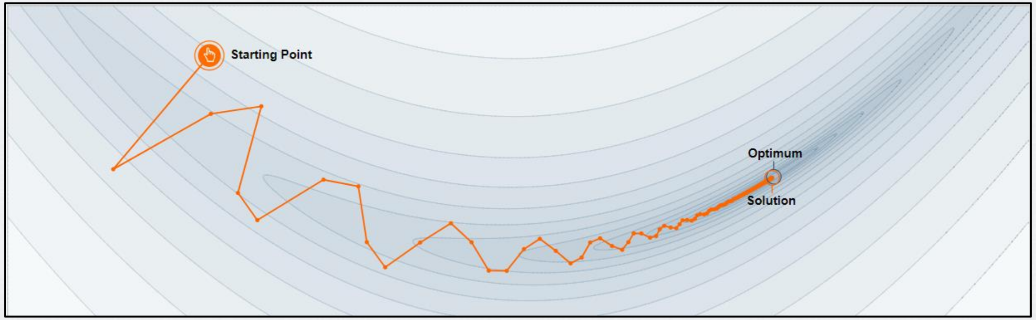
Momentum

Optimizers

- Momentum playground
  - https://distill.pub/2017/momentum/

- Parameters
  - $\eta = 0.0034$
  - $\alpha = 0.81$





Nesterov Momentum

Optimizers

- The Nesterov Momentum modifies the momentum algorithm to use the gradient at the **projected future position**
  - The difference between Nesterov momentum and standard momentum is where the gradient is evaluated

- With Nesterov momentum the gradient is evaluated **after applying the current velocity: $\boldsymbol{\theta} + \boldsymbol{\alpha v}$**

Momentum Update Rule

$$v \leftarrow \alpha v - \eta \nabla_\theta \left(\frac{1}{|\beta|} \sum_{i=\beta_t}^{n} \mathcal{L}(f(x_i, \boldsymbol{\theta}), y_i) \right)$$
$$\theta \leftarrow \theta + v$$

Nesterov Momentum Update Rule

$$v \leftarrow \alpha v - \eta \nabla_\theta \left(\frac{1}{|\beta|} \sum_{i=\beta_t}^{n} \mathcal{L}(f(x_i, \boldsymbol{\theta} + \boldsymbol{\alpha v}), y_i) \right)$$
$$\theta \leftarrow \theta + v$$

Nesterov Momentum

Optimizers

- SGD with Nesterov Momentum

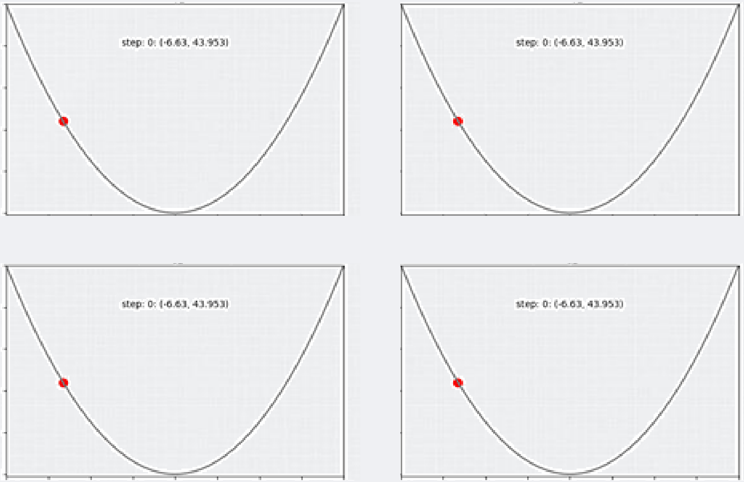
Stochastic Gradient Descent Algorithm with Nesterov Momentum
$\theta \leftarrow$ Xavier or Kaiming He Initialization
While not converged do
$\quad \hat{\theta} \leftarrow \theta + \alpha v$ ⊳ Apply interim update
$\quad g \leftarrow \frac{1}{\lvert\beta\rvert} \sum_{i=\beta_t}^{n} \nabla \mathcal{L}^i(\hat{\theta})$
$\quad v \leftarrow \alpha v - \eta g$
$\quad \theta \leftarrow \theta + v$

Algorithms with Adaptive Learning Rates

Optimizers

- SGD updates all parameters with the same learning rate (η)
  - Even its versions with Momentum and Nesterov momentum

- **Learning rate** is one of the hyperparameters that is the **most difficult to set**
  - It has a significant impact on model performance



AdaGrad

Optimizers

- Many problems produce sparse gradients
  - Some features occur far less frequently than others
  - Parameters associated with infrequent features only receive meaningful updates whenever these features occur

- Adaptive subgradient (AdaGrad)
  - It adapts a learning rate for each component of θ

AdaGrad

Optimizers

- Update rule
  - $\theta_i \leftarrow \theta_i - \frac{\eta}{\varepsilon+\sqrt{s_i}} \nabla_{\theta_i}$
  - $s_i \leftarrow s_i + (\nabla\theta_i)^2$
  - η global learning rate – typically set to a default value of 0.01
  - ε small constant to prevent division by zero

- The parameters with the **largest** partial derivative of the loss have a correspondingly **rapid decrease** in their learning rate

- The parameters with **small** partial derivatives have a relatively **small decrease** in their learning rate

Adam

Optimizers

- The name “Adam” derives from the phrase adaptive moments

- Adam is generally regarded as being fairly robust to the choice of hyperparameters

Adam Algorithm
$\theta \leftarrow$ Xavier or Kaiming He Initialization, $t \leftarrow 0, s \leftarrow 0, r \leftarrow 0, p_1 \leftarrow 0, p_2 \leftarrow 0$
While not converged do
$\quad g \leftarrow$ Computed gradient using θ on loss $\mathcal{L}$
$\quad t \leftarrow t + 1$
$\quad s \leftarrow p_1 s + (1 - p_1) g$ ⊳ Update the first moment
$\quad r \leftarrow p_2 r + (1 - p_2) g \odot g$ ⊳ Update the second moment
$\quad \hat{s} \leftarrow s/(1 - p_1^t), \hat{r} \leftarrow r/(1 - p_2^t)$
$\quad \Delta\theta \leftarrow -\eta \frac{\hat{s}}{\varepsilon + \sqrt{\hat{r}}}$
$\quad \theta \leftarrow \theta + \Delta\theta$

The Zoo of Optimizers

Optimizers

- Some deep learning models are sensitive to choice of the optimizer (Liu et al., 2020; Davis et al., 2021)

- Previous works have argued that Adam often provides competitive performance (Shmidt et al. (2021); Schneirder et al. (2019))

- The choice of which algorithm to use depends on the cost of hyperparameter tuning

Liu et al. *Understanding the Difficulty of Training Transformers*. Empirical Methods in Natural Language Processing (EMNLP), 2020

Davis et al. *Catformer: Designing Stable Transformers via Sensitivity Analysis*. International Conference on Machine Learning (ICML), 2021

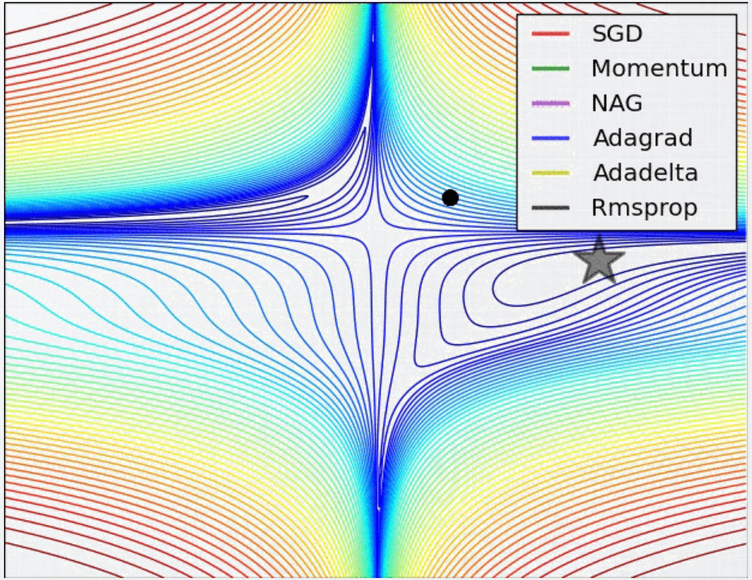
Schmidt et al. *Descending through a Crowded Valley - Benchmarking Deep Learning Optimizers*. International Conference on Machine Learning (ICML), 2021

Schneider et al. *DeepOBS: A Deep Learning Optimizer Benchmark Suite*. International Conference on Learning Representations (ICLR), 2019

The Zoo of Optimizers

Optimizers

- In practice, some architectures (i.e., residual networks) prefer SGD over optimizers (Dosovitskiy et al. 2021)
  - Therefore, unfortunately, the best optimizer depends on the architecture × task



Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale* . International Conference on Learning Representations (ICLR), 2021

Hyperparameters

Introduction

Hyperparameters

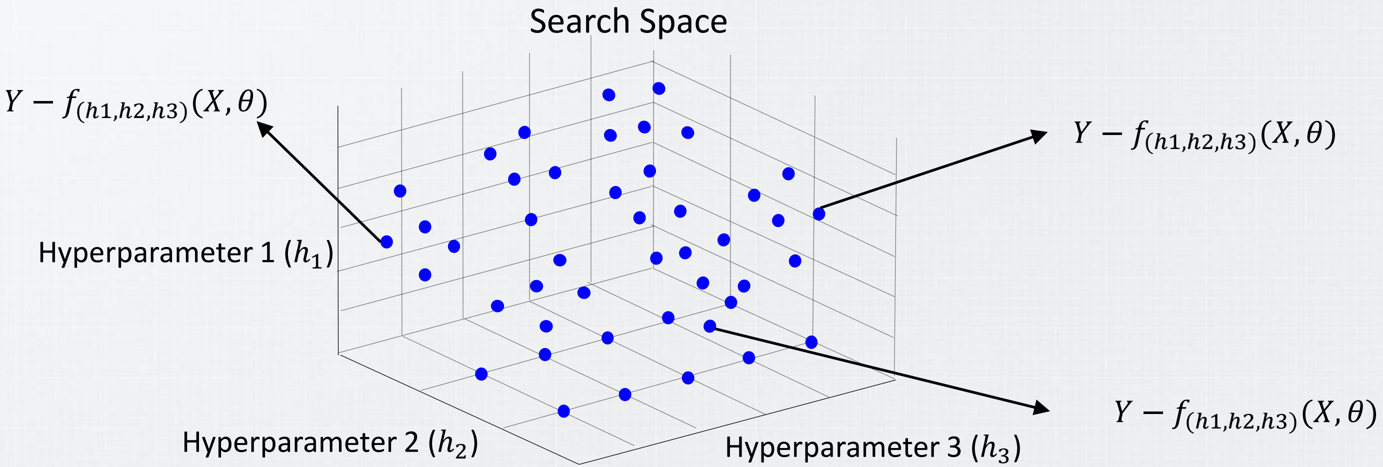
- Optimizer (and its parameters), batch size and learning rate schedule
  - All these choices are named **Hyperparameters**

- Hyperparameters directly affect the final model performance
  - Importantly, they are distinct from the model parameters

Introduction

Hyperparameters

- To find the best hyperparameters, a common practice is to train many models with different hyperparameters and choose the best one using a validation set
  - Such a strategy is referred to as **hyperparameter search**
  - Unfortunately, a single configuration of hyperparameters may be too expensive (many GPU hours/days)

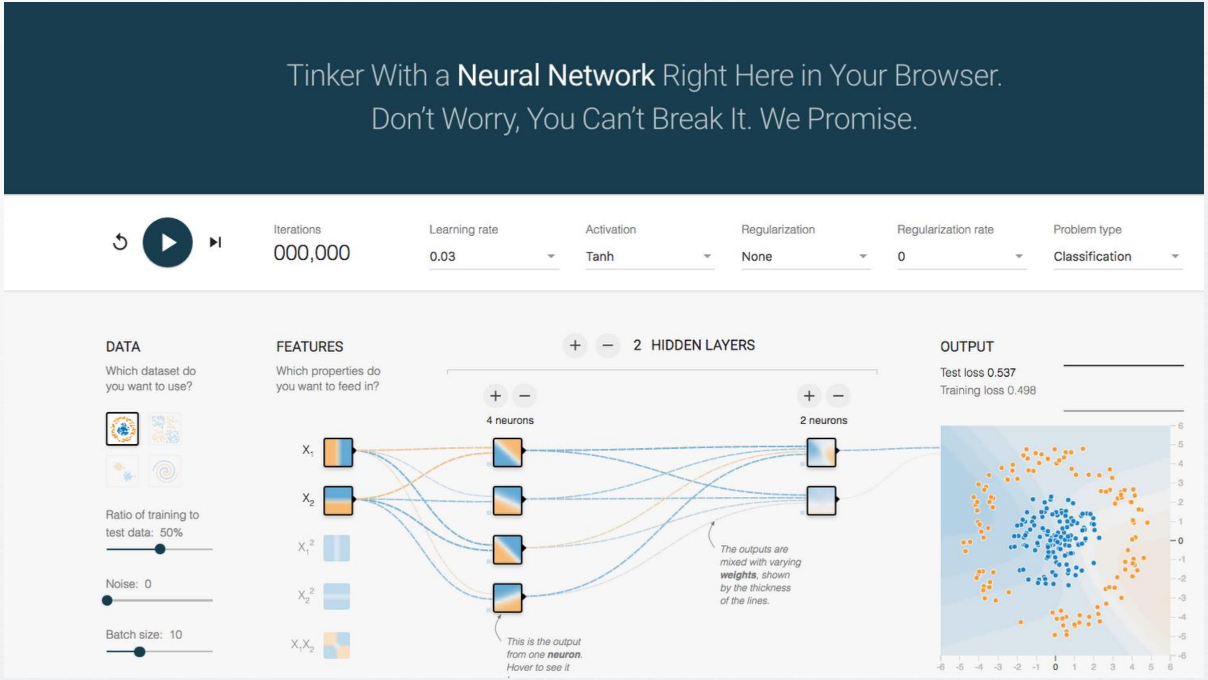


Neural Network Playground

Tensorflow Playground

Neural Network Playground

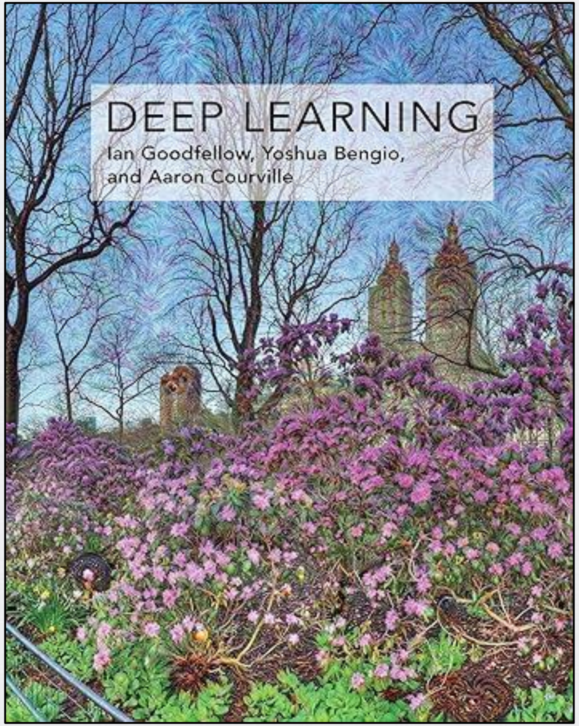
- Experiment with the basics of neural networks using TensorFlow Playground
  - https://playground.tensorflow.org/



Bibliography

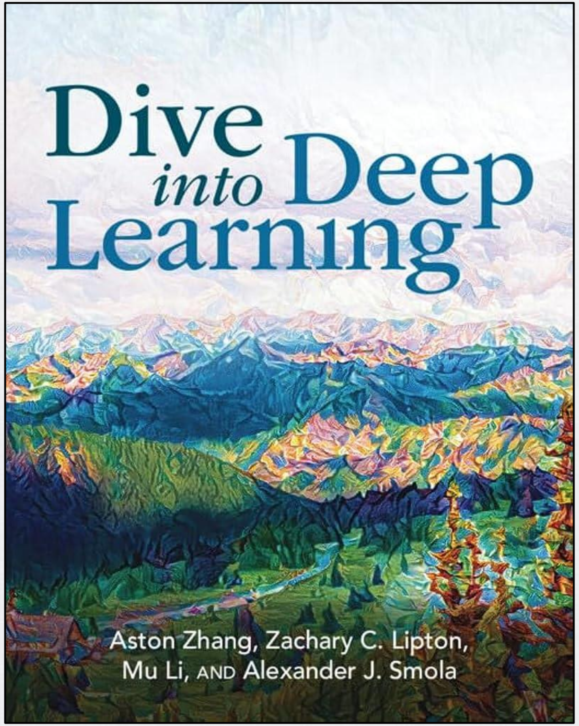
Bibliography

- Deep Learning
  - Chapter 8
    - 8.3.1 Stochastic Gradient Descent
    - 8.3.2 Momentum
    - 8.3.3 Nesterov Momentum
    - 8.4 Parameter Initialization Strategies
    - 8.5.1 AdaGrad
    - 8.5.2 RMSProp
    - 8.5.3 Adam



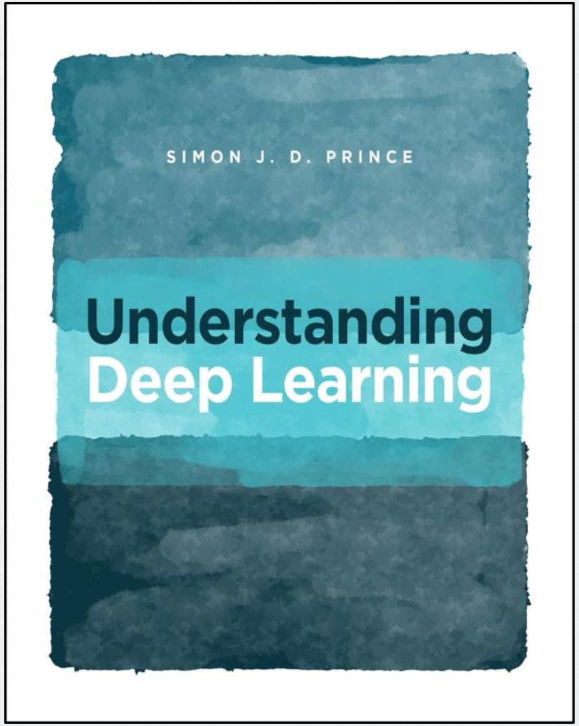
Bibliography

- Dive into Deep Learning
    - Chapter 5
        - 5.4.1 Vanishing and Exploding Gradients
    - Chapter 12
        - 12.4.2 Dynamic Learning Rate



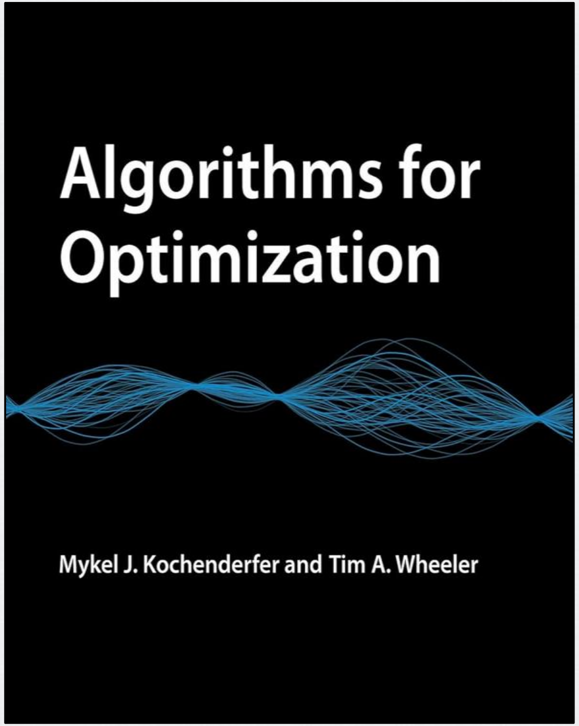
Bibliography

- Understanding Deep Learning
  - Chapter 6
    - 6.1 Gradient descent
    - 6.3 Momentum
    - 6.4 Adam



Bibliography

- Algorithms for Optimization
  - Chapter 5
    - 5.3 Momentum
    - 5.4 Nesterov Momentum



Bibliography

- Schmidt et al. *Descending through a Crowded Valley - Benchmarking Deep Learning Optimizers.* International Conference on Machine Learning (ICML), 2021

- Schneider et al. *DeepOBS: A Deep Learning Optimizer Benchmark Suite*. International Conference on Learning Representations (ICLR), 2019

ICLR
International Conference On Learning Representations

ICML
International Conference On Machine Learning