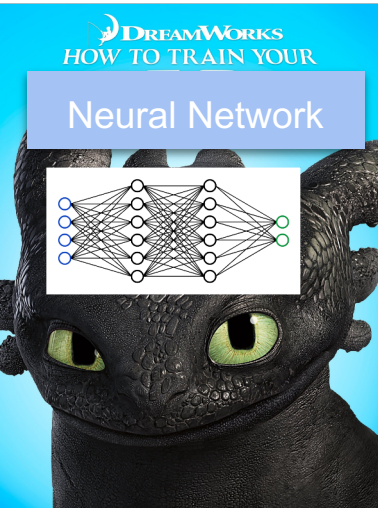


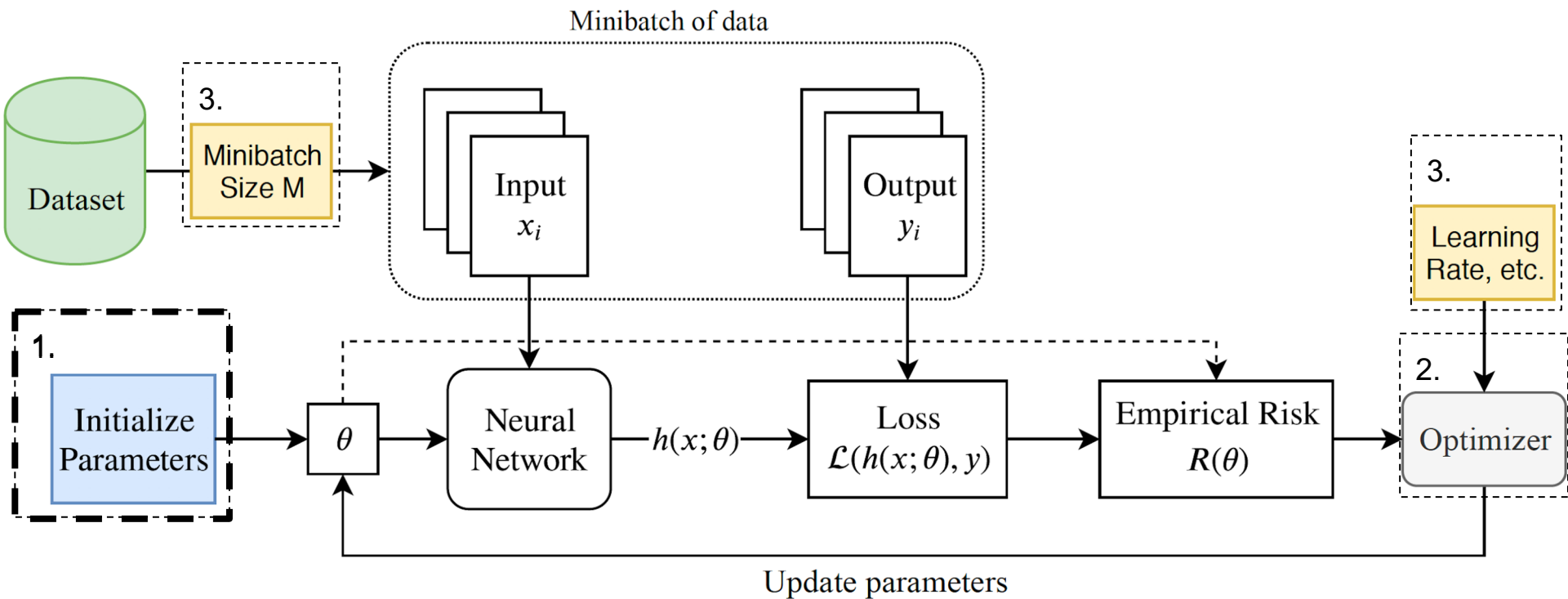
# COMS4995 NNDL Tutorial: Neural Network Training



# Overview

- Review: Overall Training Loop
- Initialization
- Optimization
  - Gradient Descent
  - Momentum
  - Learning Rate Schedulers: Adagrad, RMSProp, Adam
- Hyperparameter tuning: learning rate, batch size, regularization

# Neural Network Training Loop



# Initialization of Parameters

Initial parameters of the neural network can affect the **gradients** and learning

## Idea 1: **Constant** initialization

- *Result:* For fully connected layers: identical gradients, identical neurons. Bad!

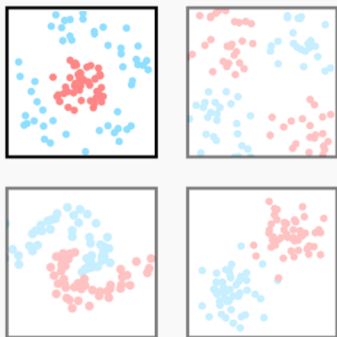
## Idea 2: **Random** weights, to break symmetry

- Too **large** of initialization: **exploding** gradients
- Too **small** of initialization: **vanishing** gradients

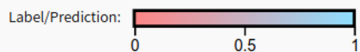
# Interactive Demo: Initialization

## 1. Choose input dataset

Select a training dataset.



This legend details the color scheme for labels, and the values of the weights/gradients.



Node Type:  Input  Relu  Sigmoid

## 2. Choose initialization method

Select an initialization method for the values of your neural network parameters<sup>1</sup>.

Zero  Too small  Appropriate  Too large

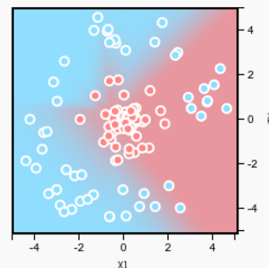
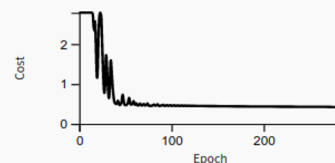


Select whether to visualize the weights or gradients of the network above.

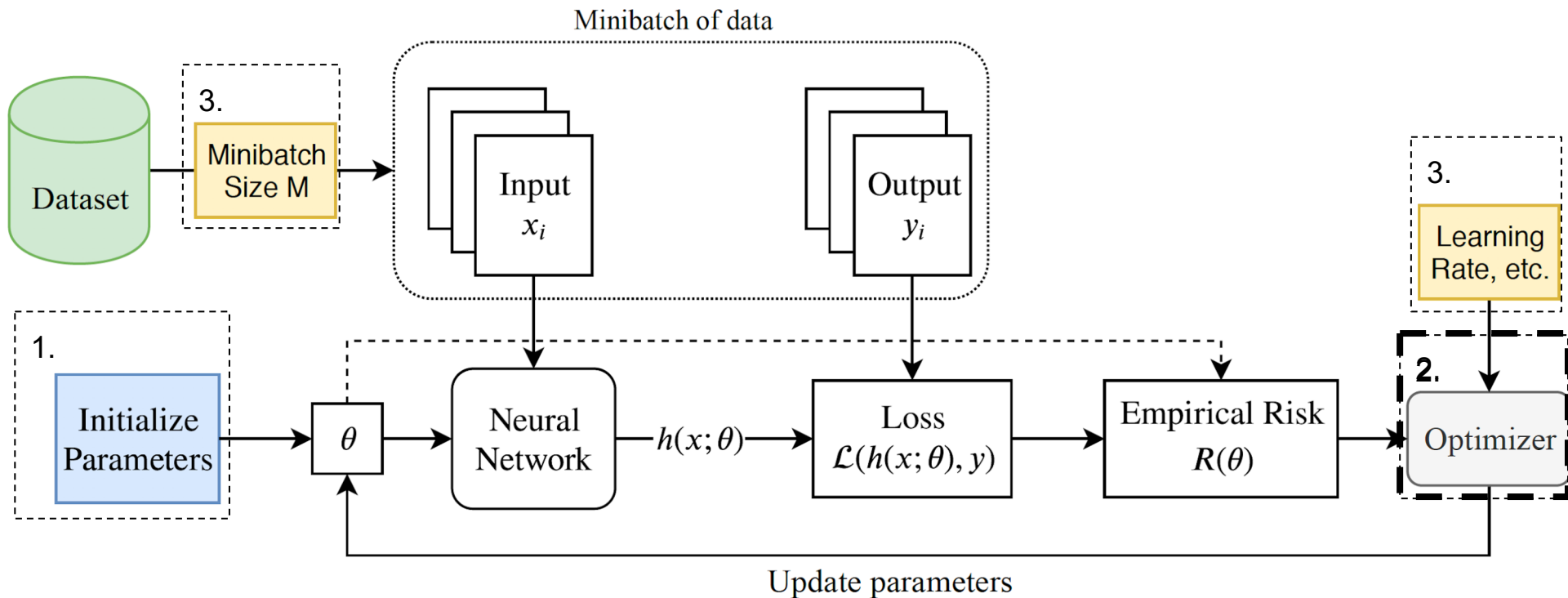
Weight  Gradient

## 3. Train the network.

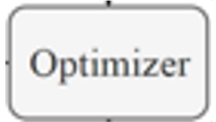
Observe the cost function and the decision boundary.



# Neural Network Training Loop



# Optimization



- **Optimization:** (informal) Minimize (or maximize) some quantity.
- Applications:
  - Engineering: Minimize fuel consumption of an automobile
  - Economics: Maximize returns on an investment
  - Supply Chain Logistics: Minimize time taken to fulfill an order
  - Life: Maximize happiness

# Optimization: Batch Gradient Descent

Optimizer

## Batch Gradient Descent:

- Initialize the parameters randomly
- For each iteration, do until convergence:

$$\theta^{(k+1)} = \theta^{(k)} - \eta \nabla R(\theta^{(k)})$$

$\eta \in \mathbb{R}^+$

Learning rate (a small step)



# Gradient Descent

Optimizer

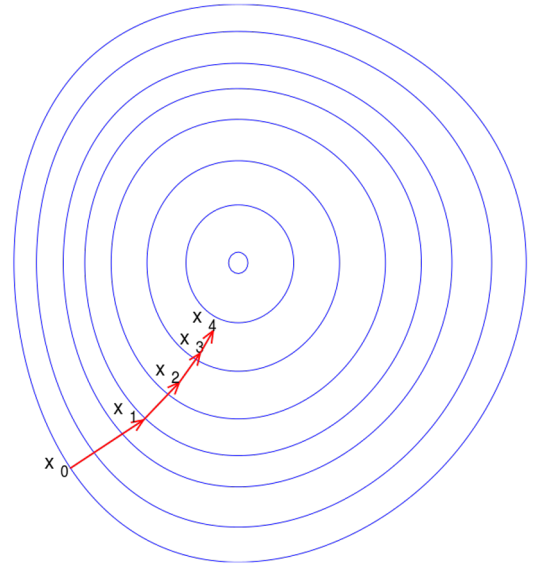
Geometric interpretation:

- Gradient is perpendicular to the tangent of the level set curve
- Given the current point, negative gradient direction decreases the function fastest

Alternative interpretation:

- Minimizing the first-order Taylor approx of  $f$  keep the new point close to the current point

$$f(x^t) + \nabla f(x^t)^T (x - x^t) + \frac{1}{2\eta} \|x - x^t\|_2^2$$



Source: Wikipedia

# Stochastic Gradient Descent

- Initialize the parameters randomly
- For each iteration, do until convergence:
  - Randomly select a training sample (or a small subset of the training samples)
  - Conduct gradient descent:

$$\theta^{(k+1)} = \theta^{(k)} - \eta \nabla f_i(\theta^{(k)})$$

- **Intuition:** A noisy approximation of the gradient of the whole dataset
- **Pro:** each update requires a small amount of training data, good for training algorithms for a large-scale dataset

- **Tips**

- Subsample *without* replacement so that you visit each point on each pass through the dataset ("epoch")
- Divide the log-likelihood estimate by the size of mini-batches, making learning rate invariant to the mini-batch size.

# Gradient Descent with Momentum

- Initialize the parameters randomly
- For each iteration, do until convergence:

- Update the momentum

$$\delta^{(k+1)} = -\eta \nabla R(\theta^{(k)}) + \alpha \delta^{(k)}$$

- Conduct gradient descent:

$$\theta^{(k+1)} = \theta^{(k)} + \delta^{(k+1)}$$

- **Pro:** “accelerate” learning by accumulating some “velocity/momentum” using the past gradients

# Learning Rate Schedulers

Optimizer

What if we want to be able to have a **per-parameter learning rate**?

- Certain parameter may be more sensitive (i.e. have higher curvature)

# Learning Rate Schedulers: Adagrad

- Initialize the parameters randomly
- For each iteration, do until convergence:
  - Conduct gradient descent on i-th parameter:

$$\theta_{k+1,i} = \theta_{k,i} - \frac{\eta}{\sqrt{G_{k,i} + \epsilon}} \cdot \nabla R(\theta_{k,i})$$

$$G_{k,i} = G_{k-1,i} + (\nabla R(\theta_{k,i}))^2$$

**Intuition:** It increases the learning rate for more sparse features and decreases the learning rate for less sparse ones, according to the history of the gradient

# Learning Rate Schedulers: RMSprop/Adadelta

Optimizer

- Initialize the parameters randomly
- For each iteration, do until convergence:
  - Conduct gradient descent on i-th parameter:

$$\theta_{k+1,i} = \theta_{k,i} - \frac{\eta}{\sqrt{G_{k,i} + \epsilon}} \cdot \nabla R(\theta_{k,i})$$

$$G_{k,i} = \gamma G_{k-1,i} + (1 - \gamma) (\nabla R(\theta_{k,i}))^2$$

**Intuition:** Unlike Adagrad, the denominator places a significant weight on the most recent gradient. This also helps avoid decreasing learning rate too much.

# Learning Rate Schedulers: Adam



Optimizer

- Initialize the parameters randomly
- For each iteration, do until convergence:
  - Conduct gradient descent on i-th parameter:

$$\theta^{(k+1),i} = \theta^{(k,i)} + \frac{\eta \hat{m}^{(k,i)}}{\sqrt{\hat{G}^{(k,i)} + \epsilon}} \cdot \nabla R(\theta^{(k,i)})$$

$$G^{(k,i)} = \gamma G^{(k-1,i)} + (1 - \gamma) (\nabla R(\theta^{(k,i)}))^2$$

$$m^{(k,i)} = \alpha m^{(k-1,i)} + (1 - \alpha) \nabla R(\theta^{(k,i)})$$

$$\hat{m}^{(k,i)} = \frac{m^{(k,i)}}{1 - \alpha^t}$$

$$\hat{G}^{(k,i)} = \frac{G^{(k,i)}}{1 - \gamma^t}$$

Bias-corrected forms of  $m^{(k,i)}$ ,  $G^{(k,i)}$

TITLE

Adam: A method for stochastic optimization

D Kingma, J Ba

International Conference on Learning Representation

[Paper Link](#)

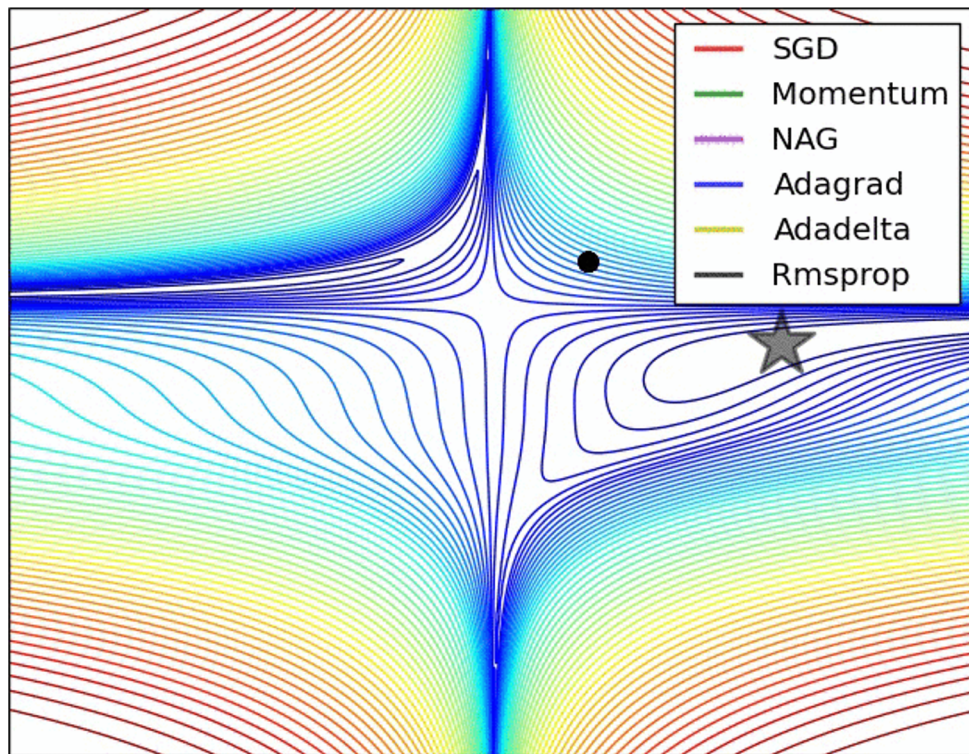
CITED BY

36831

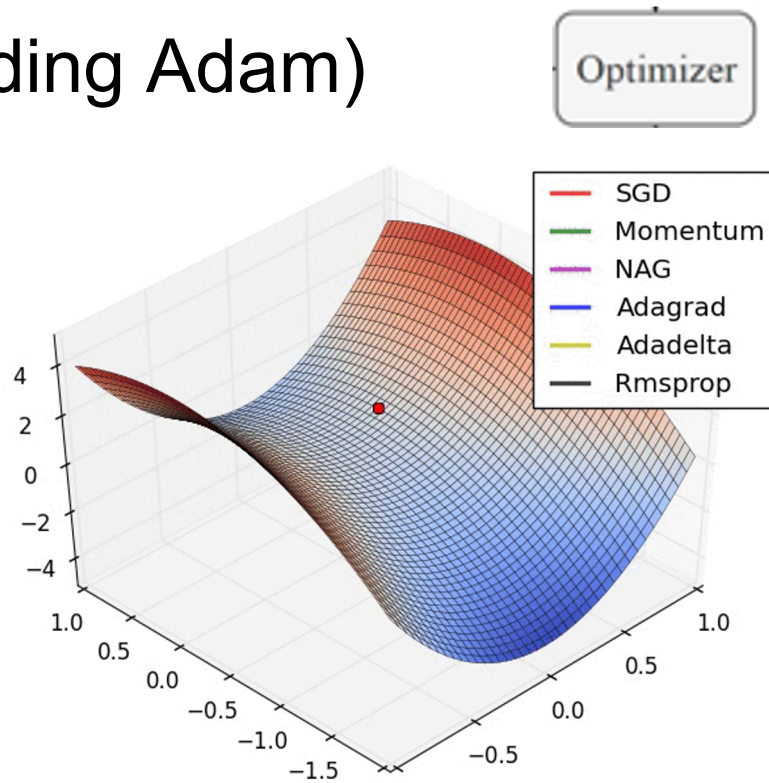
YEAR

2015

# Optimizers Comparison (excluding Adam)



SGD optimization on loss surface contours



SGD optimization on loss surface contours



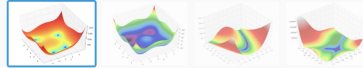
# Interactive Demo: Optimizers



In this visualization, you can compare optimizers applied to different cost functions and initialization. For a given cost landscape (1) and initialization (2), you can choose optimizers, their learning rate and decay (3). Then, press the play button to see the optimization process (4). There's no explicit model, but you can assume that finding the cost function's minimum is equivalent to finding the best model for your task.

## 1. Choose a cost landscape

Select an **artificial landscape**  $\mathcal{J}(w_1, w_2)$ .



## 2. Choose initial parameters

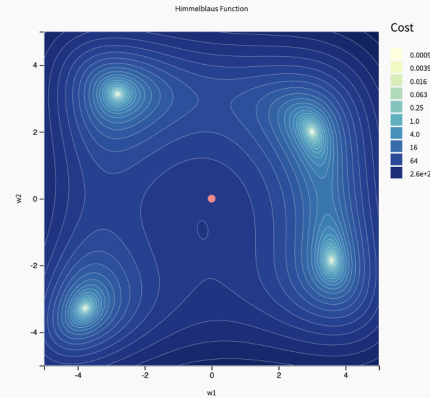
On the **cost landscape graph**, drag the **red dot** to choose initial parameter values and thus the initial value of the cost.

## 3. Choose an optimizer

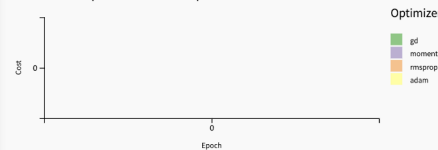
Select the optimizer(s) and hyperparameters.

Optimizer	Learning Rate	Learning Rate Decay
<input checked="" type="checkbox"/> Gradient Descent	<input type="text" value="0.001"/>	<input type="text" value="0"/>
<input checked="" type="checkbox"/> Momentum	<input type="text" value="0.001"/>	<input type="text" value="0"/>
<input checked="" type="checkbox"/> RMSprop	<input type="text" value="0.001"/>	<input type="text" value="0"/>
<input checked="" type="checkbox"/> Adlam	<input type="text" value="0.001"/>	<input type="text" value="0"/>

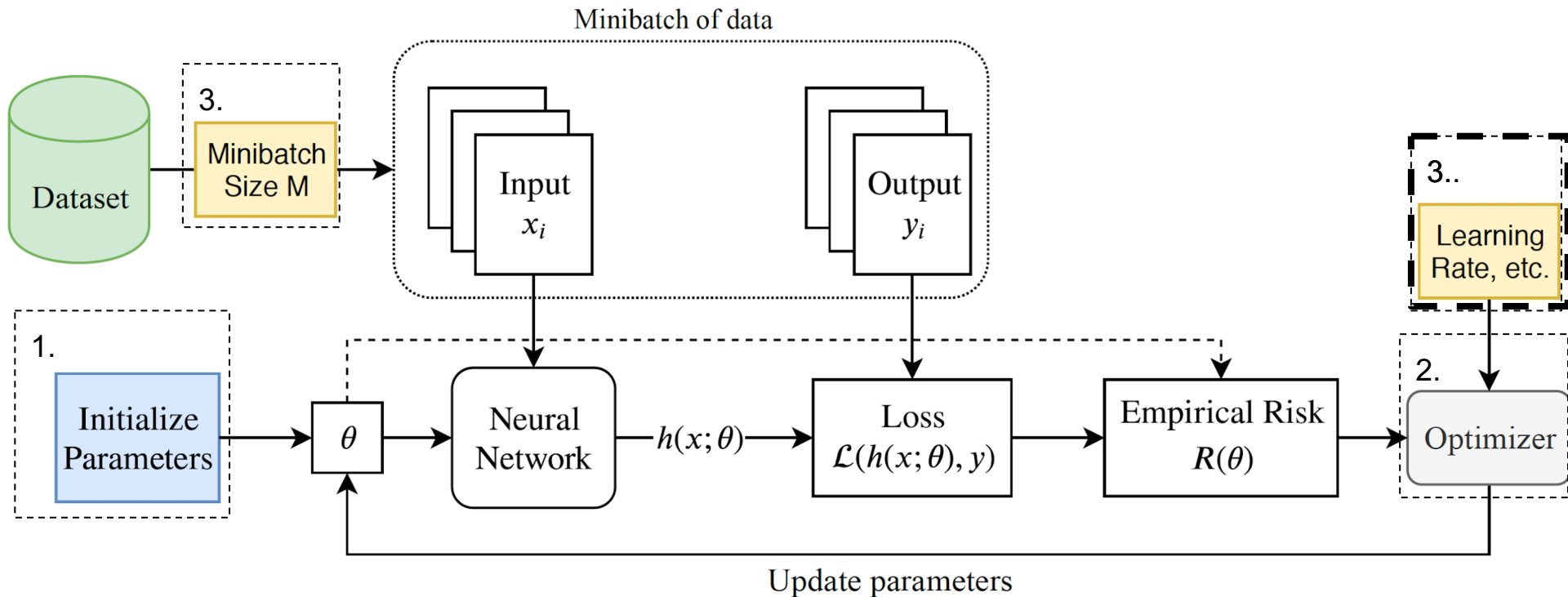
This 2D plot describes the cost function's value for different values of the two parameters ( $w_1, w_2$ ). The lighter the color, the smaller the cost value.



The graph below shows how the value of the cost changes through successive epochs for each optimizer.



# Neural Network Training Loop



# Learning Rate

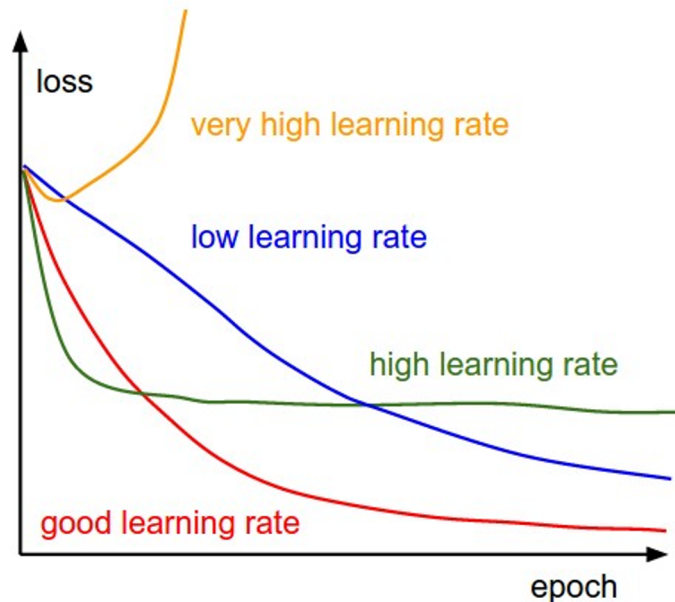
Learning  
Rate, etc.

Ideal Learning Rate should be:

- Should not be too big (objective will blow up )
- Should not be too small (takes longer to converge)

Convergence criteria:

- Change in objective function is close to zero
- Gradient norm is close to zero
- Validation error starts to increase (early-stopping)



Idealized cartoon depiction of different learning rates.

Image Credit: Andrej Karpathy

# Learning Rate: Decay Schedule

Anneal (decay) learning rate over time so the parameters can settle into a local minimum. Typical decay strategies:

- 1. **Step Decay:** reduce by factor every few epochs (e.g. a half every 5 epochs, or by 0.1 every 20 epochs), or when validation error stops improving
- 2. **Exponential Decay:** Set learning rate according to the equation

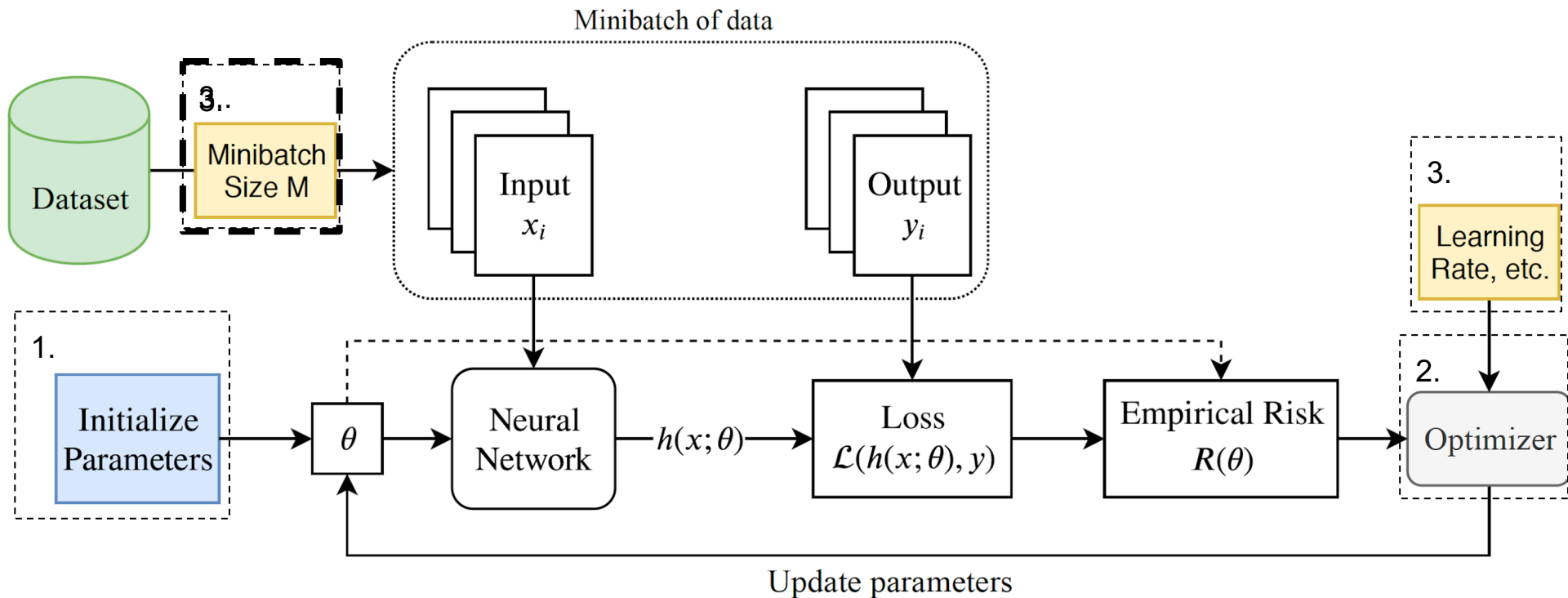
$$\eta(t) = \eta_0 e^{-kt}$$

Iteration number (points to  $t$ )  
Hyperparam (points to  $\eta_0$  and  $k$ )

- 1. **1/t decay:**

$$\eta(t) = \frac{\eta_0}{1+kt}$$

# Neural Network Training Loop



# Batch Size

**Batch Size:** the number of training data points for computing the empirical risk at each iteration.

- Typical small batches are powers of 2: 32, 64, 128, 256, 512,
- Large batches are in the thousands

**Large Batch Size** has:

- **Fewer frequency of updates**
- More **accurate** gradient
- More **parallelization** efficiency / accelerates wallclock training
- **May hurt generalization**, perhaps by causing the algorithm to find poorer local optima/plateau.

# Batch Size

Minibatch  
Size M

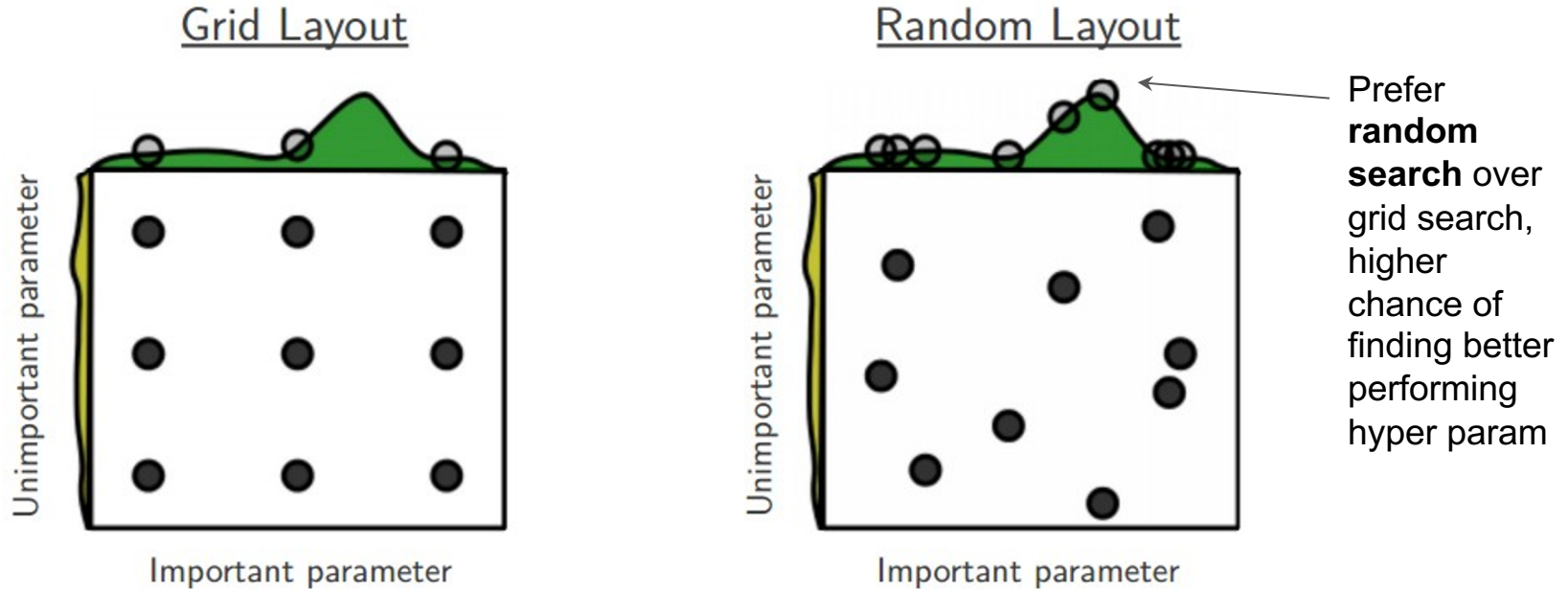
Related papers on batch size:

- [Goyal et al., Accurate, large minibatch SGD](#)
  - Proposes to increase the learning rate by of the minibatch size
- [Hoffer et al., Train longer generalize better](#)
  - Proposes to increase the learning rate by **square root** of the minibatch size
- [Smith et al., Don't decay the learning rate, increase the batch size](#)
  - Increasing batch size reduce noise, while maintaining same step size

# Hyperparameter Tuning

Minibatch  
Size M

Several approaches for tuning multiple hyperparameters together:





# Hyperparameter Tuning

Search hyperparameter on **log scale**:

- **learning\_rate = 10 \*\* uniform(-6, 1)**
  - Learning rate and regularization strength have multiplicative effects on the training dynamics
- Start from coarse ranges then narrow down, or expand range if near the boundary of range

One validation fold vs cross-validation:

- Simplifies code base to just use one (sizeable) validation set vs doing cross validation

# References

- Notes and tutorials from other courses:
  - [Toronto ECE521 \(Winter 2017\) tutorial on Training neural network](#)
  - [Stanford's CS231n notes](#) on [Stochastic Gradient Descent](#), [Setting up data and loss](#), and [Training neural networks](#)
  - [Deeplearning.ai's](#) interactive notes on [Initialization](#) and [Parameter optimization in neural networks](#)
  - Jimmy Ba's Talk for [Optimization in Deep Learning](#) at [Deep Learning Summer School 2019](#)
- Academic/white papers:
  - [SGD](#) tips and tricks from Leon Bottou
  - [Efficient BackProp](#) from Yann LeCun
  - [Practical Recommendations for Gradient-Based Training of Deep Architectures](#) from Yoshua Bengio