Machine Learning Foundations
Optimization

State-of-the-Art Approaches for
Accurate and Efficient Model Fitting

Jon Krohn, Ph.D.

jonkrohn.com/talks

github.com/jonkrohn/ML-foundations

Machine Learning Foundations
Optimization

Slides: jonkrohn.com/talks

Code: github.com/jonkrohn/ML-foundations

Stay in Touch:

jonkrohn.com to sign up for email newsletter

linkedin.com/in/jonkrohn

jonkrohn.com/youtube

twitter.com/JonKrohnLearns

The Pomodoro Technique

Rounds of:

- 25 minutes of work
- with 5 minute breaks

Questions best handled at breaks, so save questions until then.

When people ask questions that have already been answered, do me a favor and let them know, politely providing response if appropriate.

Except during breaks, I recommend attending to this lecture only as topics are not discrete: Later material builds on earlier material.

POLL: Multiple Choice!

Which of the ML Foundations classes did you attend?

- Intro to Linear Algebra
- Linear Algebra II
- Calculus I
- Calculus II
- Probability & Information Theory
- Intro to Statistics
- Algorithms & Data Structures
- Optimization
- NONE

POLL

What is your level of familiarity with Machine Learning?

- Little to no exposure, or exposure to theory only
- Experience applying machine learning with code
- Experience applying machine learning with code and some understanding of the underlying theory
- Experience applying machine learning with code and strong understanding of the underlying theory

ML Foundations Series

Optimization **builds upon** and is **foundational for**:

1. **Intro to Linear Algebra**
2. Linear Algebra II: Matrix Operations
3. **Calculus I: Limits & Derivatives**
4. **Calculus II: Partial Derivatives & Integrals**
5. Probability & Information Theory
6. Intro to Statistics
7. Algorithms & Data Structures
8. **Optimization**

Deeply understanding machine learning models.

Optimization

1. Optimization Approaches
2. Gradient Descent
3. “Fancy” Deep Learning Optimizers

Optimization

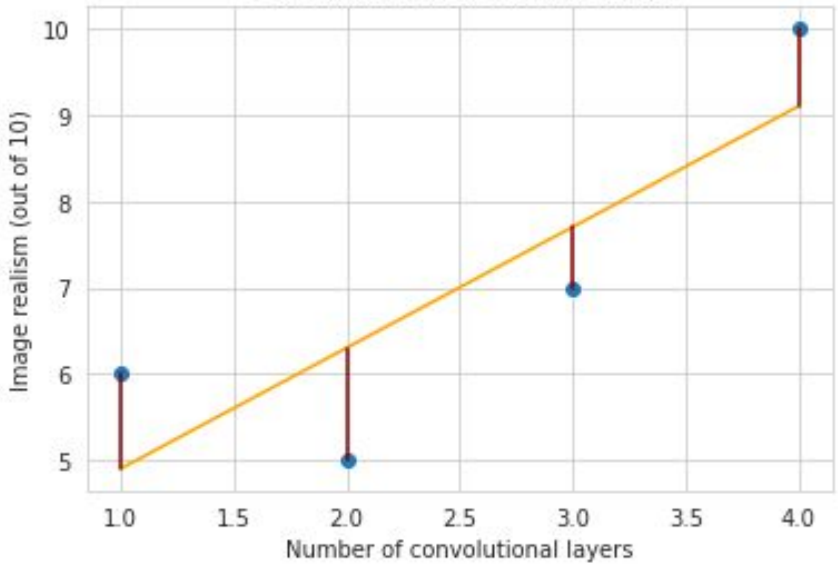
1. **Optimization Approaches**
2. Gradient Descent
3. “Fancy” Deep Learning Optimizers

Segment 1: Optimization Approaches

- The Statistical Approach to Regression: Ordinary Least Squares
- When Statistical Approaches to Optimization Break Down
- The Machine Learning Solution

Statistical Approach to Regression

Ordinary Least Squares:



Quick *hands-on code review*:

`6-statistics.ipynb`

(Deep) ML vs Frequentist Statistics

Primarily, I use (deep) ML to train my production algorithms.

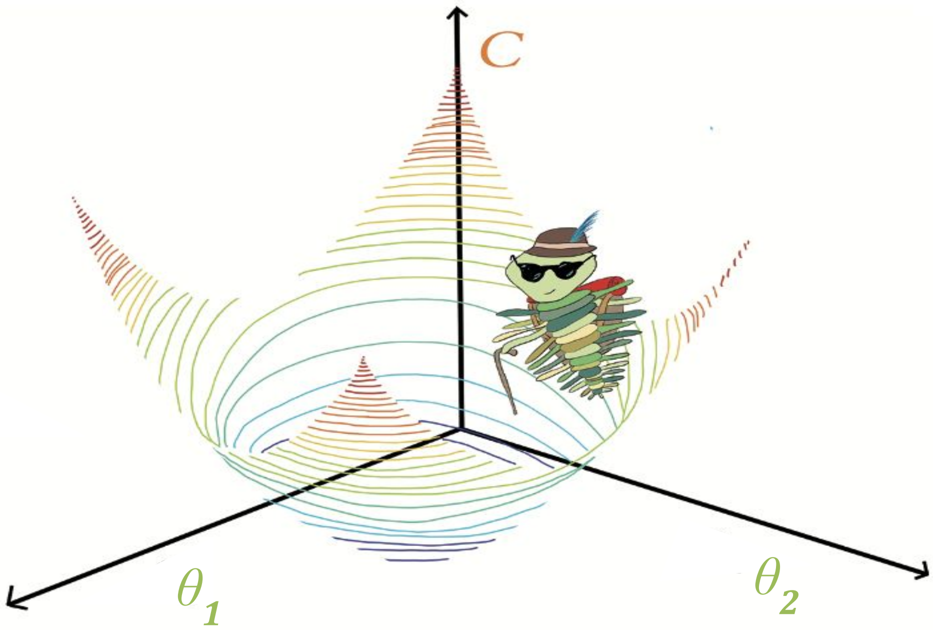
However, **I regularly use frequentist statistics** to:

- Better understand training data
- Clean training data
  - Investigate/remove outliers
  - Transform toward standard normal with Box-Cox
- Make decisions with a quantitative degree of confidence w.r.t.:
  - Model hyperparameters
  - Model outputs
    - Where misclassifications occur
    - Whether there are unwanted biases
- Occasionally, train models with relatively few data and features

(Deep) ML vs Frequentist Statistics

In general, the gradient-based optimization of **ML** becomes necessary:

- When we have thousands of data points or more
  - SGD overcomes RAM / numerical computation (big O) constraints



(Deep) ML vs Frequentist Statistics

In particular, **deep learning** enables us to:

- Handles many features (esp. large files: images, video, audio)
- Handle many outputs; exotic architectures / training strategies
- Automatically identify hierarchical, highly abstract patterns
- Automatically fit interaction terms
- Automatically fit non-linear relationships

However: As we move from frequentist stats to ML, and particularly to deep learning, it can come at the cost of explainability / understanding.

Optimization

1. Optimization Approaches
2. **Gradient Descent**
3. “Fancy” Deep Learning Optimizers

Segment 2: Gradient Descent

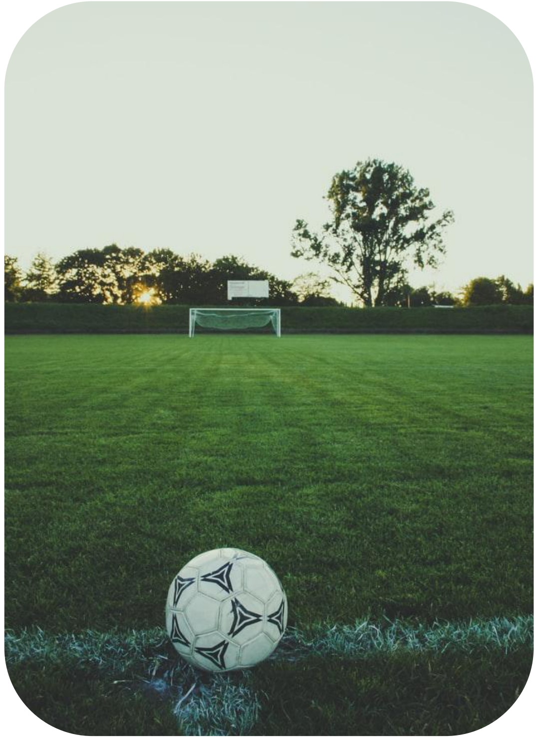
- Objective Functions
- Cost / Loss / Error Functions
- Minimizing Cost with Gradient Descent
- Learning Rate
- Critical Points, incl. Saddle Points
- Gradient Descent from Scratch with PyTorch
- The Global Minimum and Local Minima
- Mini-Batches and Stochastic Gradient Descent (SGD)
- Learning Rate Scheduling
- Maximizing Reward with Gradient Ascent

Objective Function

- A.k.a. the **criterion**
- Some function $f(\boldsymbol{x})$ that we minimize or maximize by adjusting parameters
- Minimizing $f(\boldsymbol{x})$:
  - Generally more common than maximizing in ML
  - $f(\boldsymbol{x})$ may be called **cost**, **loss**, or **error** function
  - $\boldsymbol{x}^*$ denotes $\boldsymbol{x}$ at which $f(\boldsymbol{x})$ is minimized
  - Our immediate next focus
- Maximizing $f(\boldsymbol{x})$:
  - Common in particular ML subfields
    - E.g., reinforcement learning **reward** function
  - $\boldsymbol{x}^* = \arg\max f(\boldsymbol{x})$
  - Covered briefly later

Hands-on code demo

`8-optimization.ipynb`

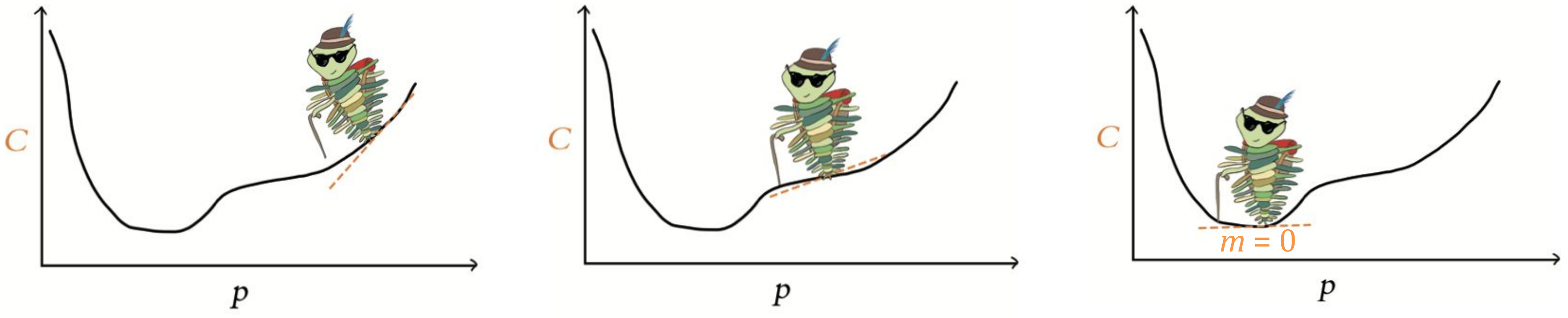


unsplash.com/photos/KWQ2kQtxiKE

Minimizing Cost with Gradient Descent

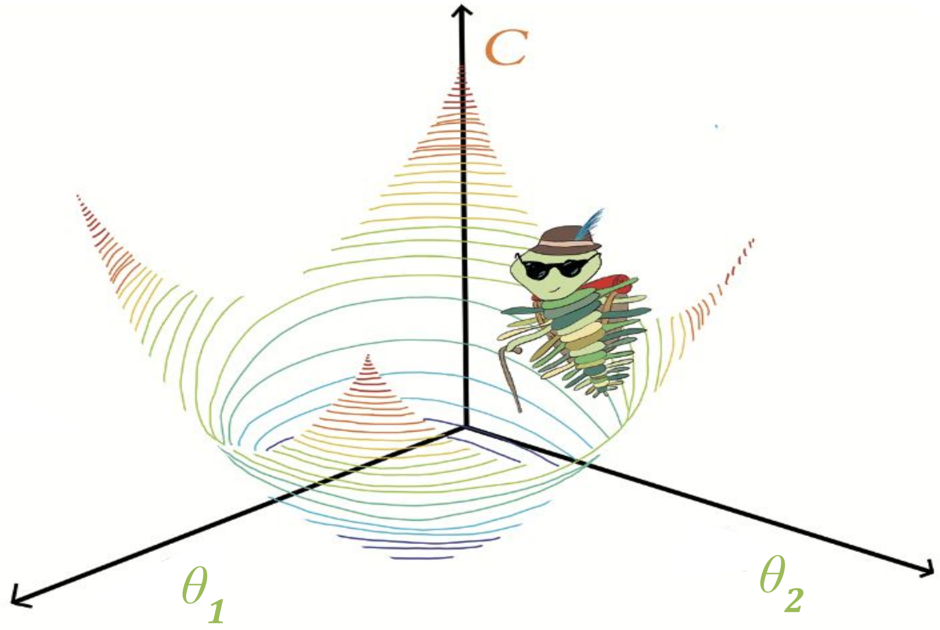
First suggested by French mathematician Augustin-Louis Cauchy in 1847

In a model with a single parameter p:



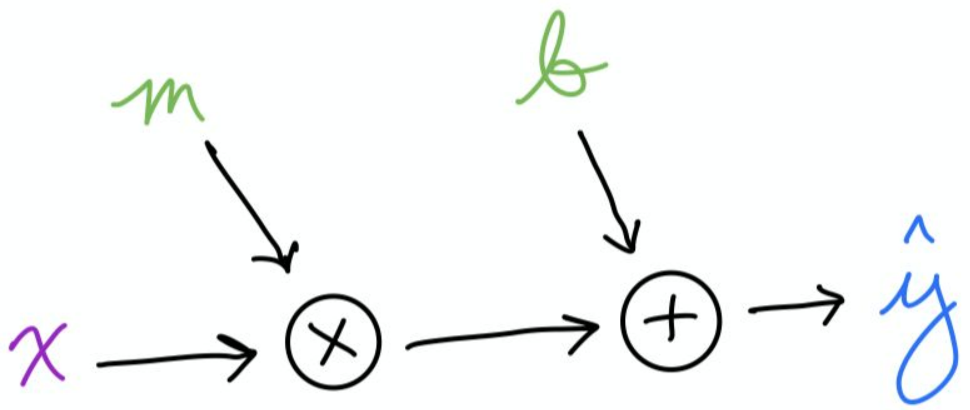
Minimizing Cost with Gradient Descent

In a model with two parameters, θ_1 and θ_2:

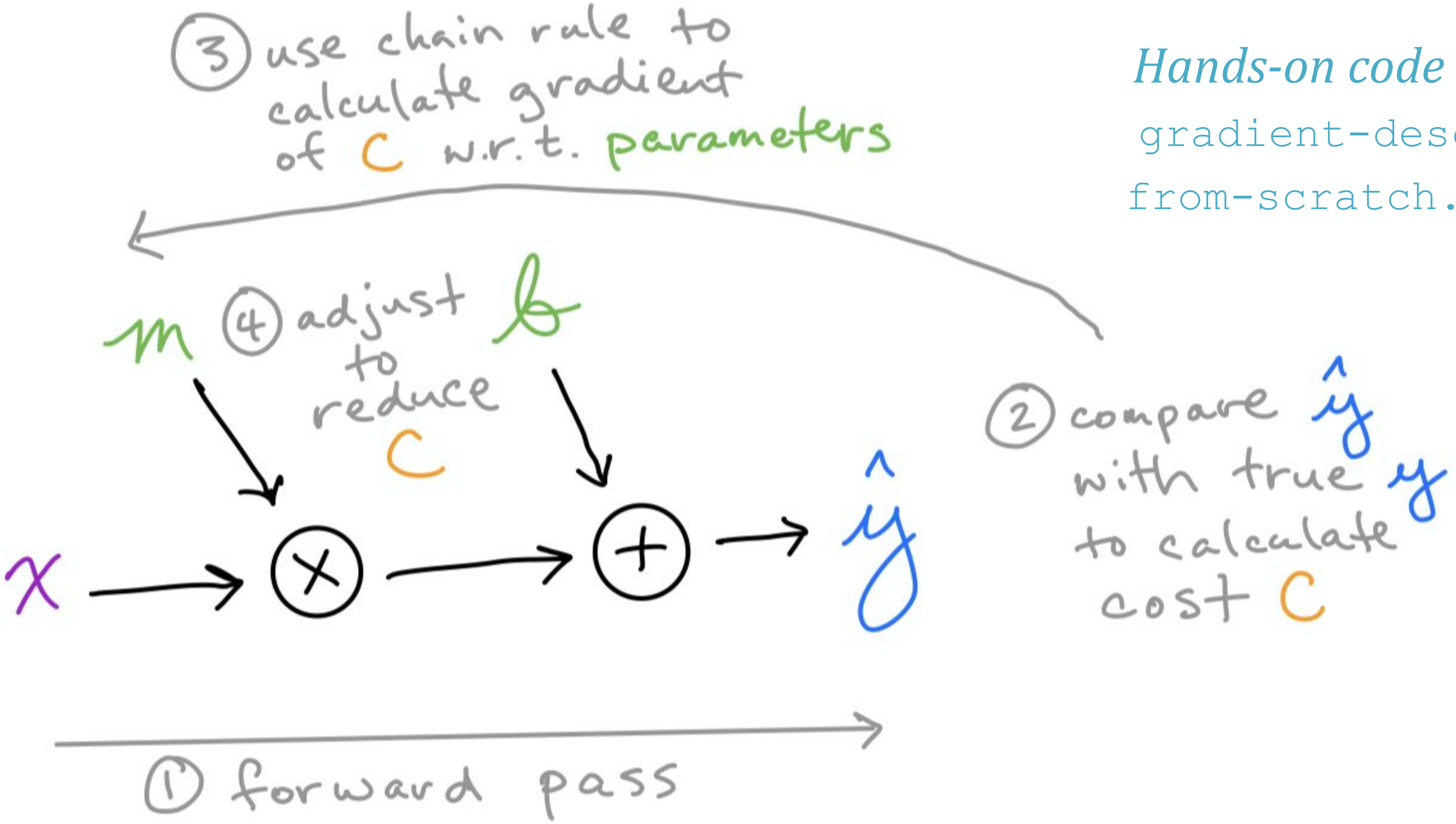


Linear Regression DAG

- Line equation $y = mx + b$ as DAG (from *Calc I* and *Calc II*)
  - Nodes are input, output, parameters, or operations
  - Edges are tensors (but non-operation nodes can be too)



Fitting a Linear Regression with GD

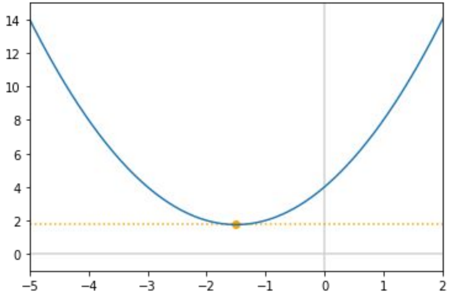


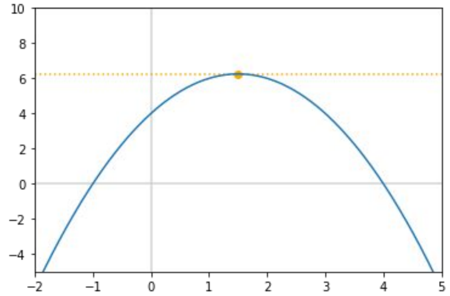
Hands-on code demo:

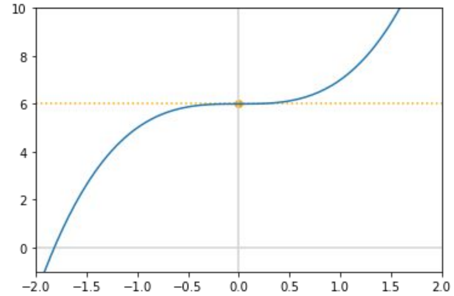
`gradient-descent-from-scratch.ipynb`

Critical Points

- With function $f(x)$, location in curve where $f'(x) = 0$
- Three types:
  a. **Minimum**:
     - Most frequently discussed in ML
     - $f(x)$ higher on both sides
  b. **Maximum**:
     - $f(x)$ lower on both sides
     - Example coming up
  c. **Saddle point**
     - $f(x)$ is lower on one side and higher on the other





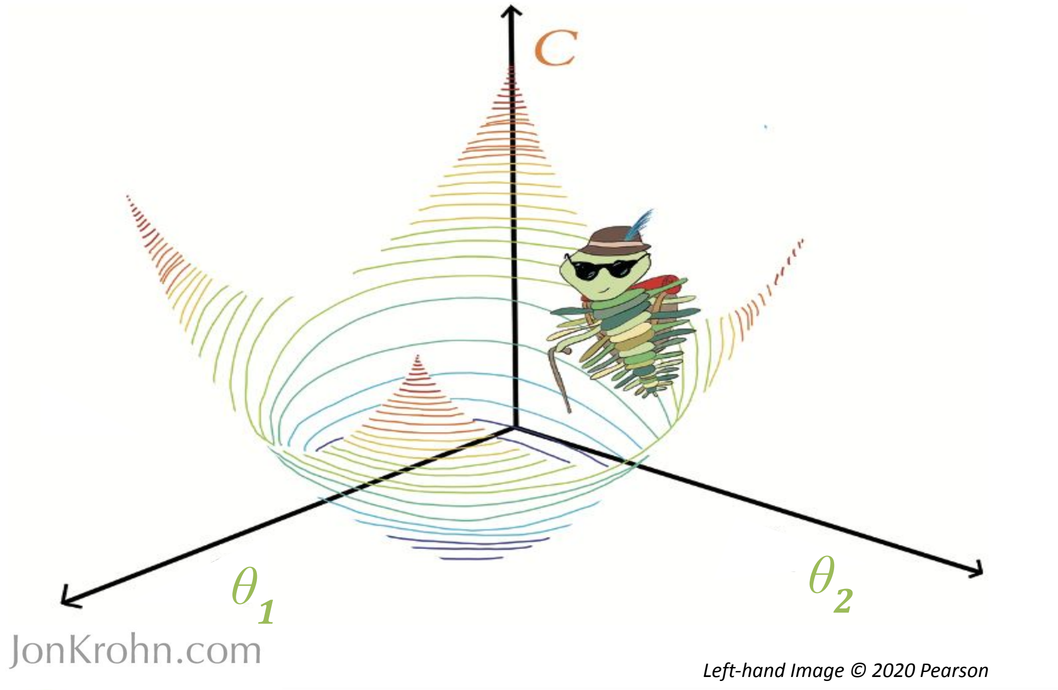


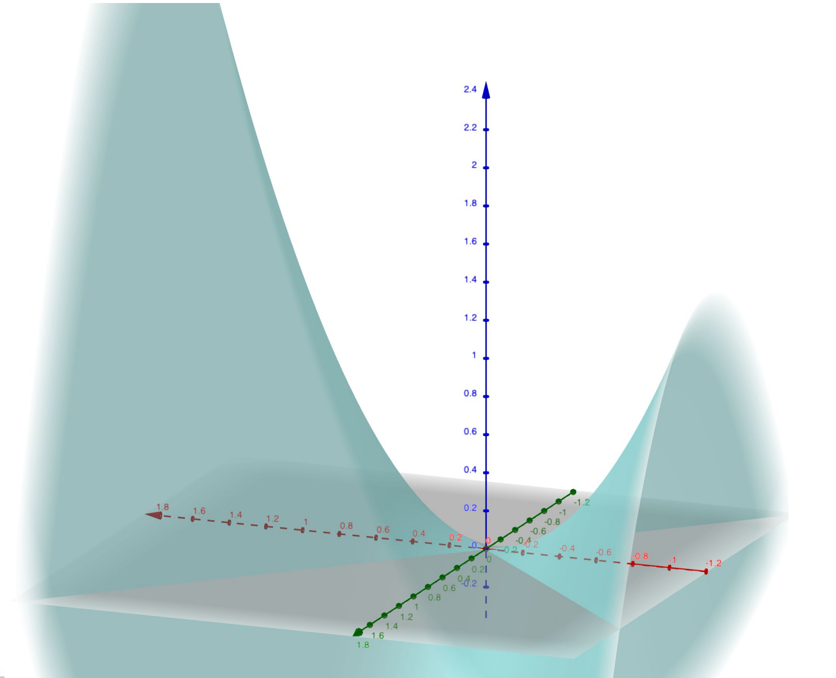
Hands-on code demo

`8-optimization.ipynb`

Critical Points in Higher-Dim. Spaces

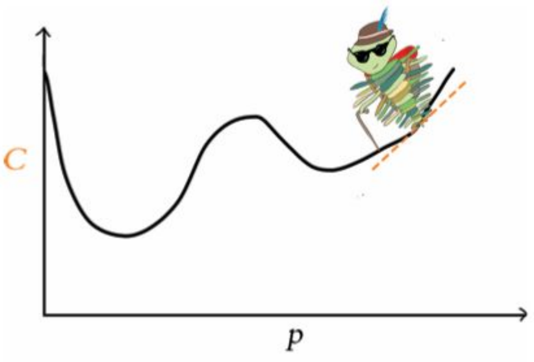
- Point where all elements of gradient (e.g., of ∇C) are zero
- **Minimum**: C is larger in all directions
- **Maximum**: C is smaller in all directions
- **Saddle Point**: C is smaller and larger in at least one direction

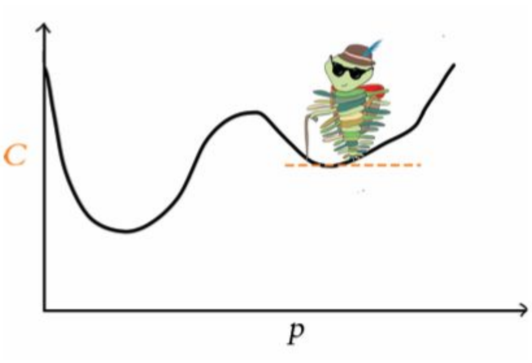




Global vs Local Minima

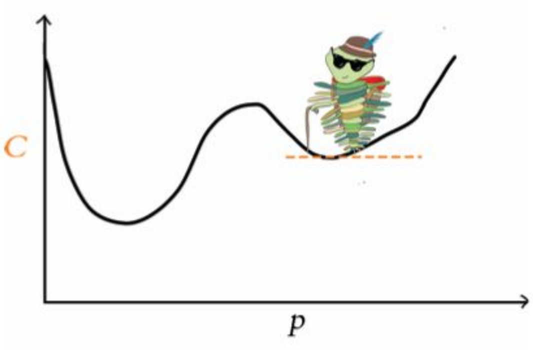
- A given minimum may not be the **global minimum**
  - In which case, it is a **local minimum**
- (Analogous concept for *global maximum* vs *local maxima*)

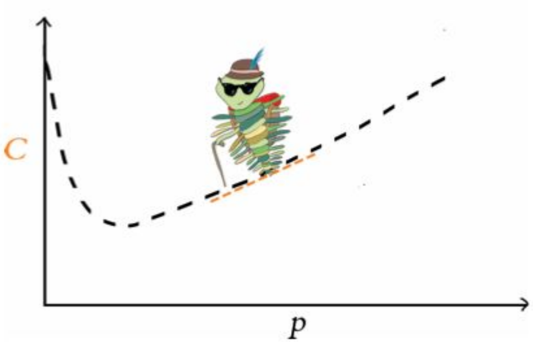




Stochastic Gradient Descent (SGD)

- Randomly (*stochastically*) sample training-data **mini-batches**
- With each mini-batch, descend gradient (in serial)
- Training round time complexity is effectively O(1) while GD is O(n)
- Helps escape local minima to find global minimum:

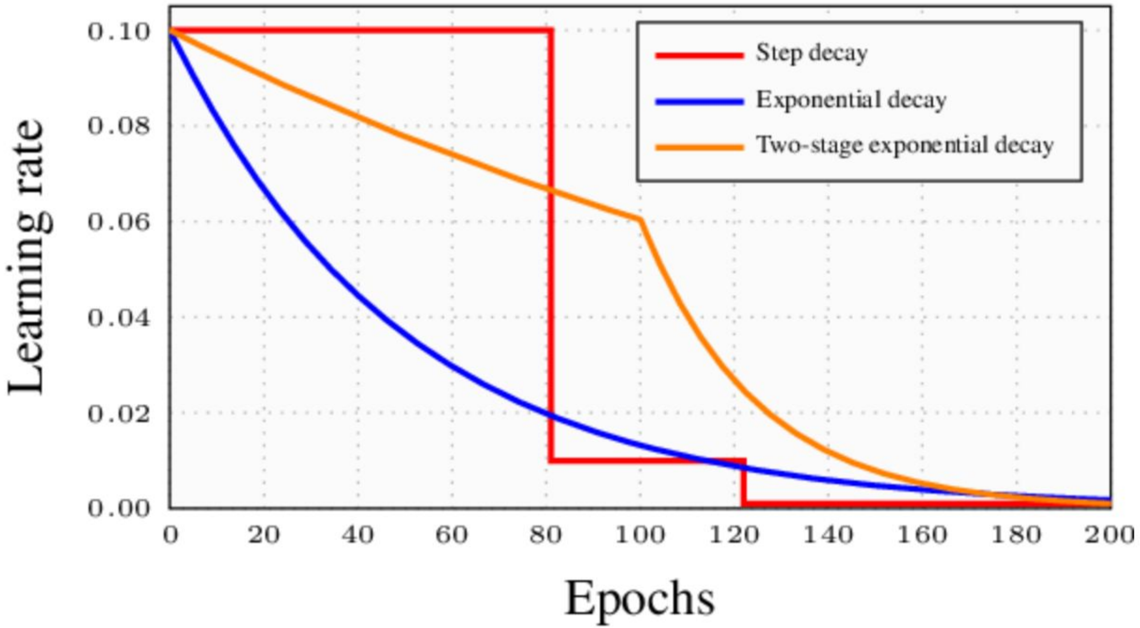




Hands-on code demo

`SGD-from-scratch.ipynb`

Learning Rate Scheduling



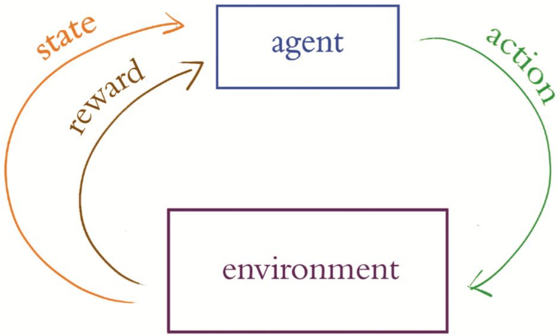
Hands-on code demo

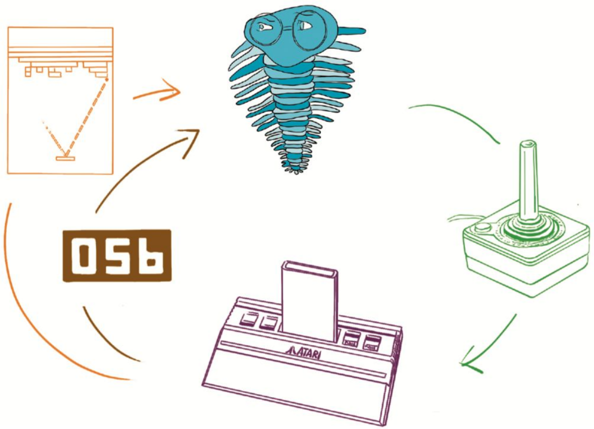
`learning-rate-scheduling.ipynb`

Image from [Hsueh et al. (2019). WACV]

Gradient Ascent

- To maximize *f*(*x*):
  - We can use GD to minimize -*f*(*x*)
- Common in **reinforcement learning**
  - RL's a category of ML problems
- Resources for further study:
  - Chapters 4 & 13 of Krohn (2020) *Deep Learning Illustrated*
  - Graesser & Keng (2020) *Foundations of Deep RL*
  - Sutton & Barto (2018) *RL: An Intro* (2nd ed.)
- **Hill climbing** with discrete parameters in Russell & Norvig (2020) *A.I.* (4th ed.)





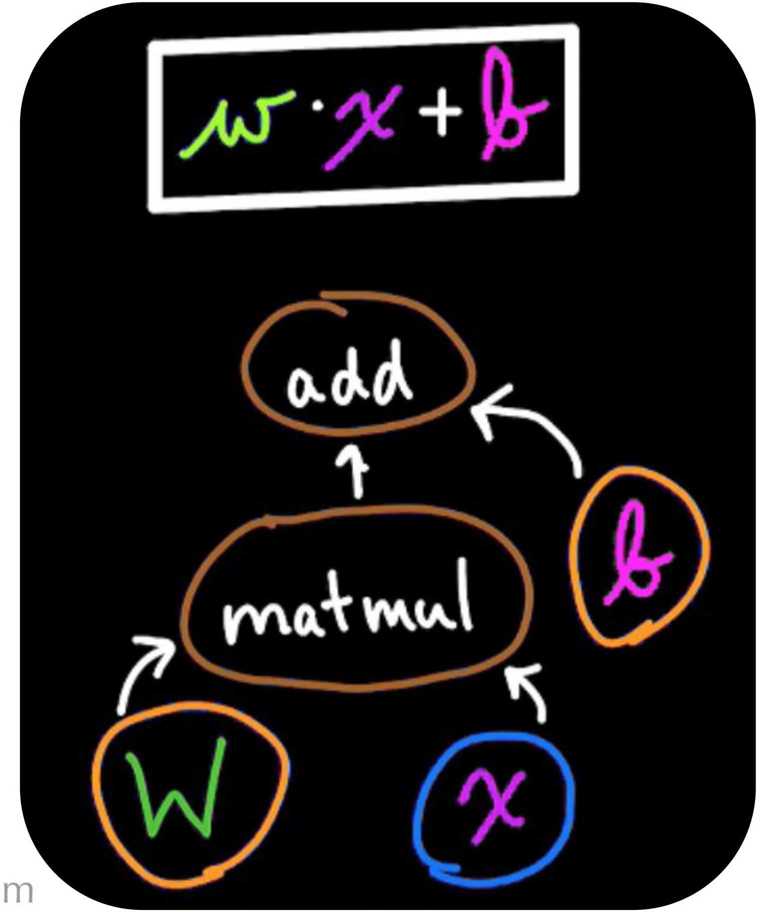
Optimization

1. Optimization Approaches
2. Gradient Descent
3. **"Fancy" Deep Learning Optimizers**

Segment 3: "Fancy" DL Optimizers

- A Layer of Artificial Neurons in PyTorch
- Jacobian Matrices
- Hessian Matrices and Second-Order Optimization
- Momentum
- Nesterov Momentum
- AdaGrad
- AdaDelta
- RMSProp
- Adam
- Nadam
- Training a Deep Neural Net

Dense Neuron Layer DAG



- Operation
- Placeholder tensor input
- Variable tensor input
- **Vector tensor output**

Hands-on code demo:

`artificial-neurons.ipynb`

Jacobian Matrices

We've only calculated partial derivatives where function has scalar output.

- E.g., cost C is a scalar
- All partial derivatives can be stored in a gradient (e.g., ∇C), a vector

An artificial neuron layer is a function typically with vector output.

- E.g., 128 activations of vector $\boldsymbol{a}$ output in *Artificial Neurons* notebook
- **Jacobian matrix** represents all possible partial derivatives

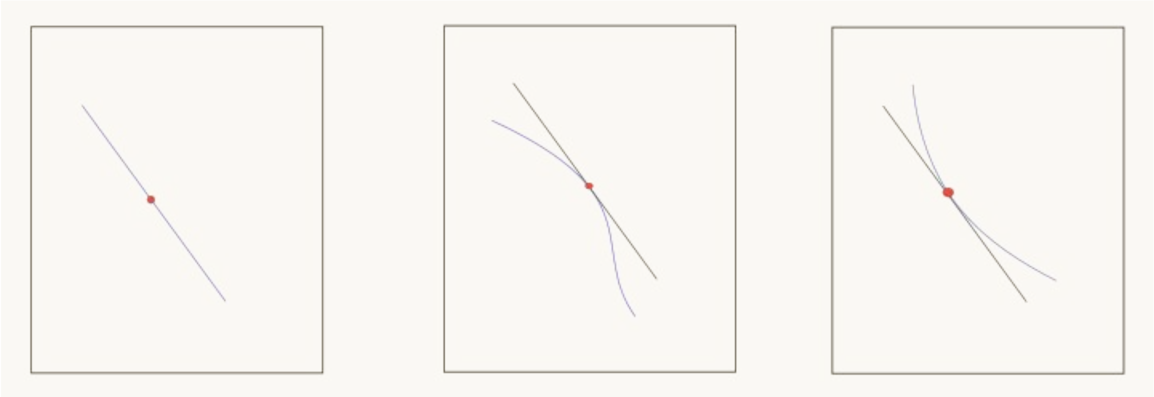
See:

`8-optimization.ipynb`

Second-Order Optimization

First-order optimization

- Uses first partial derivatives only, e.g., gradient descent
- Considers the following situations identical:



Second-order optimization:

- Accounts for curvature of function
- Uses gradient and **Hessian matrix** of second partial derivatives

Image from blog.paperspace.com/intro-to-optimization-momentum-rmsprop-adam

Hessian Matrix

- Jacobian matrix of the gradient
  - I.e., partial derivatives of vector of first partial derivatives
- Denoted as $\boldsymbol{H}(f)(\boldsymbol{x})$, where:
  - $\boldsymbol{x}$ is a vector tensor
  - Elements hold all possible second partial derivatives of $f(\boldsymbol{x})$

$$\boldsymbol{H}(f)(\boldsymbol{x})_{i,j} = \frac{\partial^2}{\partial x_i \partial x_j} f(\boldsymbol{x})$$

Second-Order Optimization

- *Left*: Gradient descent alone
- *Right*: Gradient descent with function curvature accounted for by applying **Newton's method** to Hessian matrix

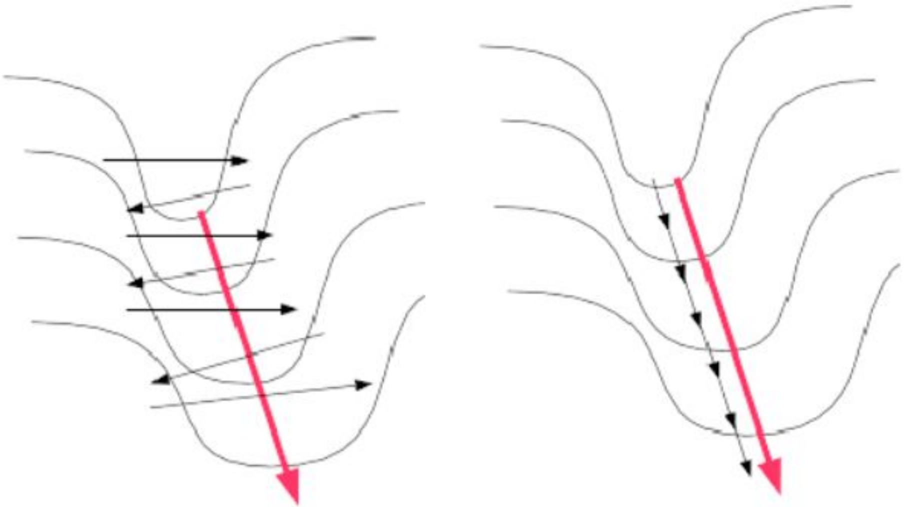


Image from Seidl (2012)

Momentum

Hessian is computationally inefficient for many DL models

- E.g., with 784-pixel input, 615k partial derivatives
- Instead, we can use compute-efficient tricks

Momentum

- Track gradients from previous steps
- Allow trajectory to influence current step

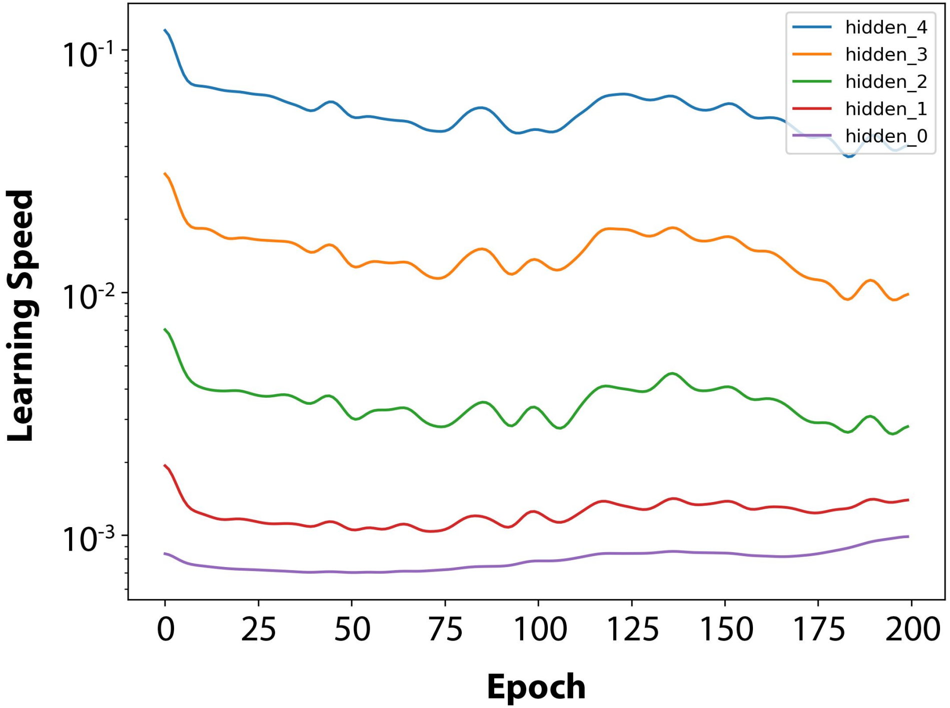
Nesterov momentum

- Incremental performance improvement
- Slight look-ahead to where momentum may lead

unsplash.com/photos/3m6vbzY69s4

Adaptive Optimizers

Each parameter's learning rate is adjusted individually:



Adaptive Optimizers

AdaGrad ("adaptive gradient")

- Shortcoming: as gradients accumulate, division by large value (step count) eventually results in extremely small learning rate

AdaDelta and **RMSProp** ("root mean square propagation")

- Resolves shortcoming by using moving average of most recent gradients instead of all of them (with decay in case of RMSProp)

Adam ("adaptive moment estimation")

- RMSProp but with improvement to avoid skew toward zero learning rate that can tend to happen at training start

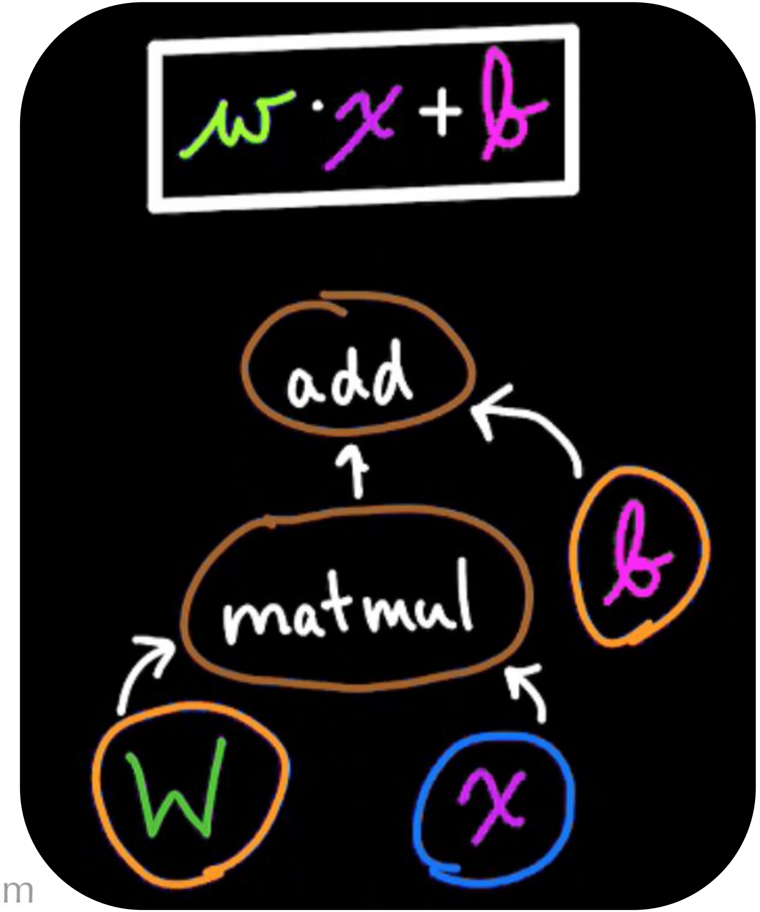
Nadam ("Nesterov Adam")

- Analogous incremental improvement using slight look-ahead

Which Optimizer?

- For deep learning, I default to Nadam
- Could experiment with RMSProp
- For simpler ML models, and sometimes even for deep learning, SGD with carefully tuned learning rate schedule can result in lowest validation cost (though training time may be longer)

Dense Neuron Layer DAG



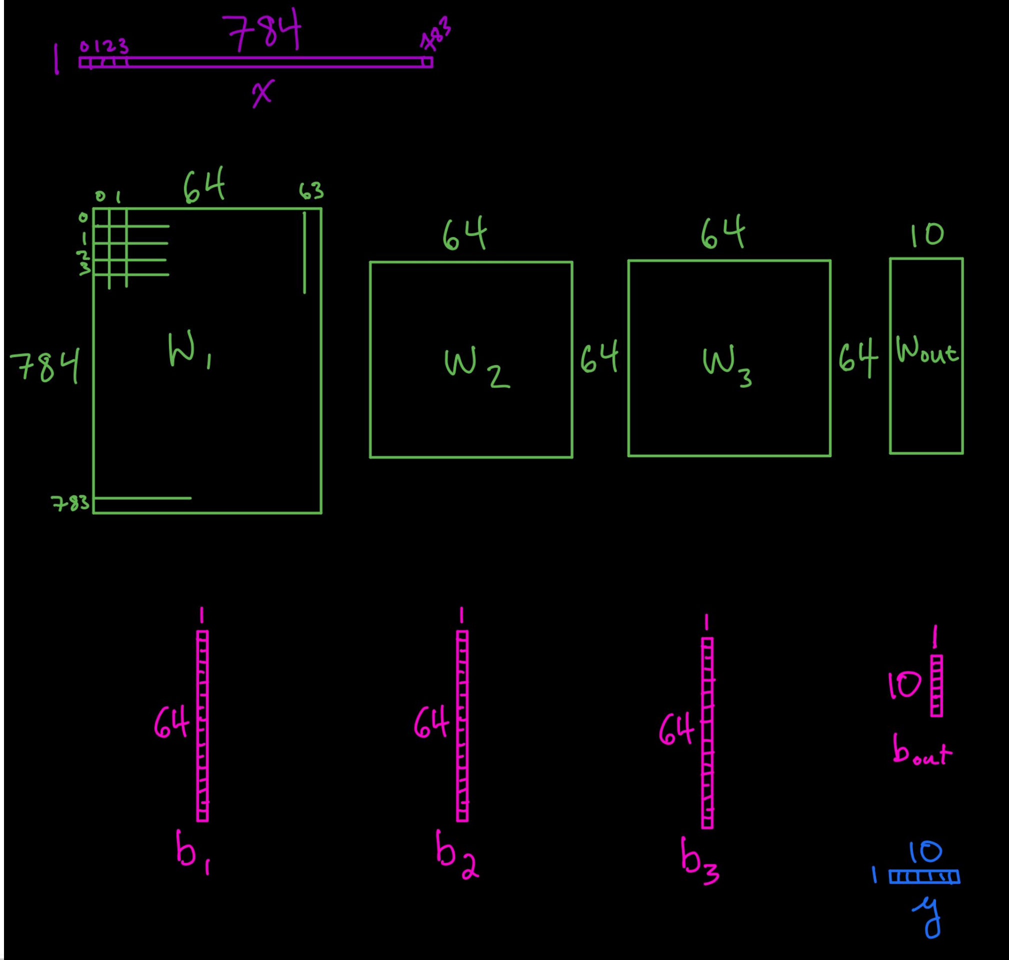
- Operation
- Placeholder tensor input
- Variable tensor input

Hands-on code demo:

`artificial-neurons.ipynb`

Training a Deep Neural Net

Recall this figure from *Lin Alg I* →

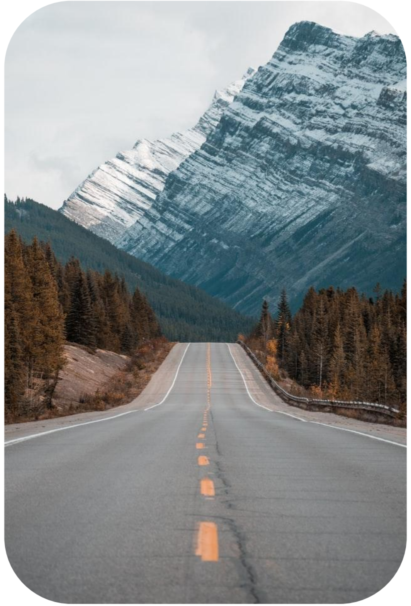


See:

- jonkrohn.com/deepTF1
- jonkrohn.com/convTF1
- jonkrohn.com/convTF2
- jonkrohn.com/deepPT

What’s next for you?

If you’ve followed this series, you’ve been exposed to all of the foundational subjects underlying ML.



You can now dig into ML books and papers to cover:

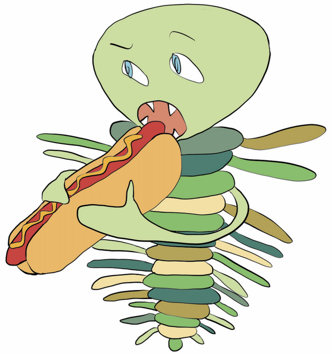
- Supervised learning, e.g.:
  - Linear regression
  - Classification with logistic regression
  - Support vector machines
  - Decision trees, incl. random forests, boosted trees
- Unsupervised learning, e.g., clustering, generation
- Deep learning-specific un/supervised learning approaches

unsplash.com/photos/z8ct_Q3oCqM

Resources for Further Study

Machine learning in general:

- Géron (2019) *Hands-on ML*
- Hastie et al. (2009) *Elements of Statistical Learning* (2nd ed.)
- Bishop (2006) *Pattern Recognition and Machine Learning*
- Murphy (2012) *ML: a Probabilistic Perspective*



Deep learning in particular:

- Krohn (2020) *Deep Learning Illustrated*
  - Chapter 9 for detail on optimizers
- Sebastian Ruder's (2016) optimization blog post
- O'Reilly *Deep Learning: Complete Guide* playlist
- Nielsen (2015) *Neural Networks and Deep Learning*
- Goodfellow et al. (2016) *Deep Learning*

What's next for *ML Foundations*?

O'Reilly Live Trainings: full, timing-reworked series in ~2021

YouTube recordings

- First segment of first subject public today
- Working as quickly as I can to publish more every week

Studio recordings

- For O'Reilly platform, *with fully-worked solutions*
- Oct: first two subjects (linear algebra I & II)
- Nov: next two subjects (calculus I & II)
- Early 2021: remaining subjects

Book

- Writing as quickly as I can
- "Rough cut" of early chapters in O'Reilly ~early 2021

Stay in Touch

jonkrohn.com to sign up for email newsletter

linkedin.com/in/jonkrohn

youtube.com/c/JonKrohnLearns

twitter.com/JonKrohnLearns

NEBULA

PLACEHOLDER
FOR:

5-Minute Timer

PLACEHOLDER
FOR:

10-Minute Timer

PLACEHOLDER
FOR:

15-Minute Timer