Diving Into Deep Learning: Part 1 – A Deep Learning Crash Course

Elizabeth Newman
Emory University

What Is Deep Learning?

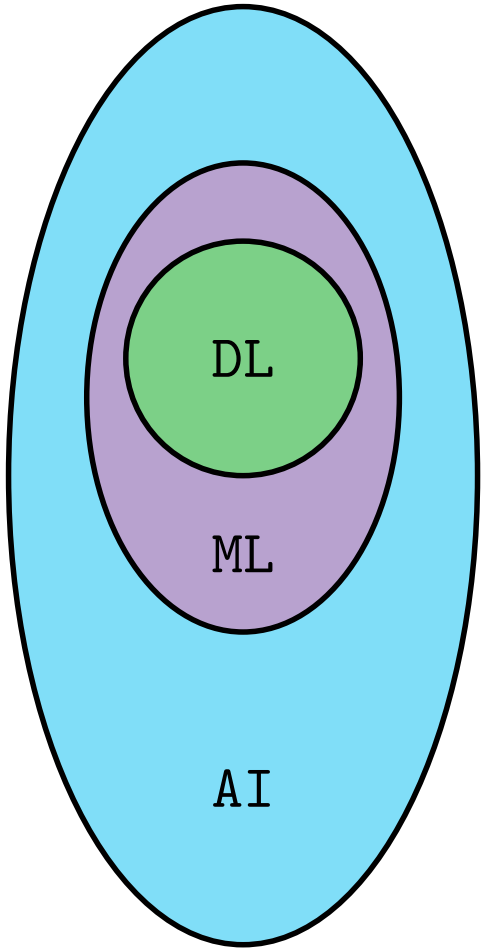
Let's ask Chat GPT!

Deep learning is a subfield of machine learning and artificial intelligence (AI) that focuses on the development of **artificial neural networks** *that are capable of learning and making predictions or decisions on their own. The term "deep" in deep learning refers to the fact that these neural networks are typically designed with multiple layers, allowing them to learn representations of data at different levels of abstraction.*

Deep learning algorithms learn from large amounts of labeled data, and their architecture allows them to automatically extract features and patterns from raw data, such as images, speech, text, and other types of data. These learned representations can then be used for tasks such as **image and speech recognition, natural language processing, autonomous driving, and many other applications**.

Deep learning has revolutionized many fields, including computer vision, speech recognition, natural language processing, and robotics, among others. It has achieved state-of-the-art performance in various tasks, surpassing traditional machine learning approaches in many cases. Deep learning has also been **enabled by advances in computing power**, *particularly in graphics processing units (GPUs) and distributed computing, which have made it possible to train large neural networks on vast amounts of data.*

Overall, deep learning has become a powerful and widely used approach in modern AI, allowing machines to automatically **learn complex patterns and make predictions or decisions** *with high accuracy, opening up new possibilities for applications across numerous domains.*

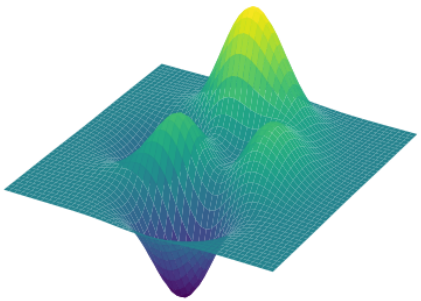


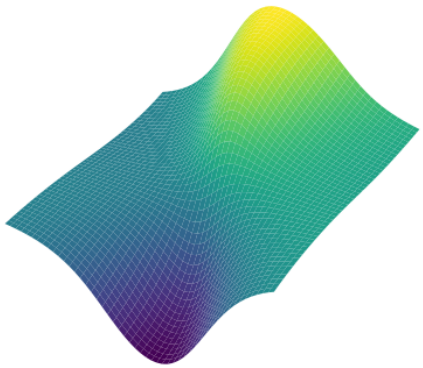
What Can Deep Neural Networks Do?

What Can Deep Neural Networks Do?

coordinates
(x_1, x_2)

function values
$f(x_1, x_2)$

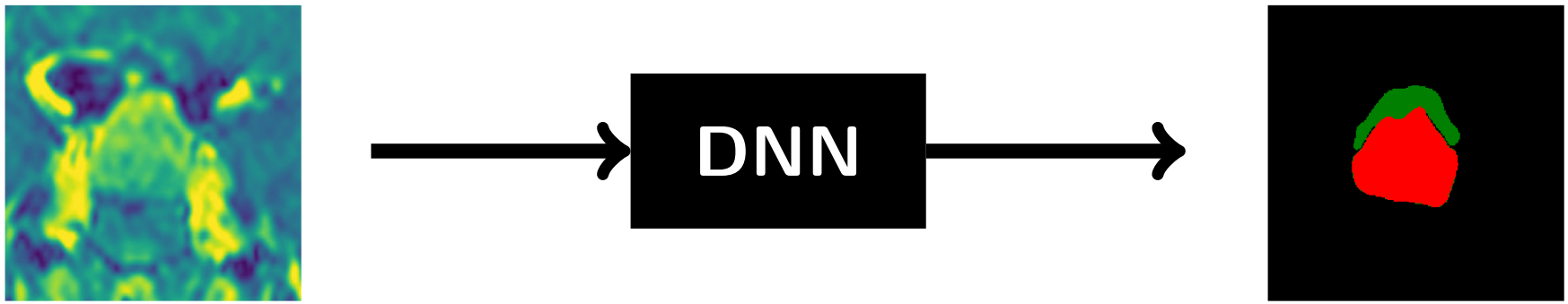




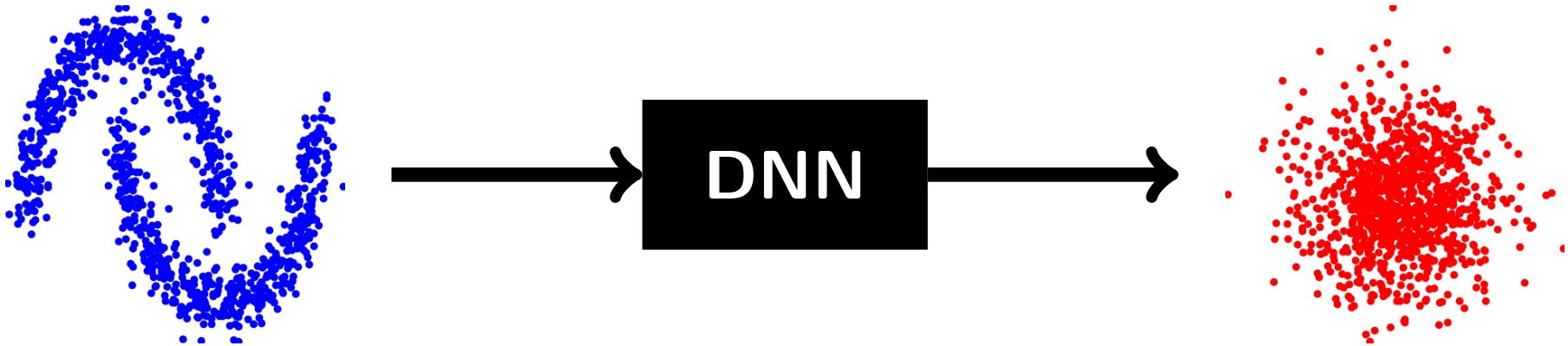
What Can Deep Neural Networks Do?

"Dog"

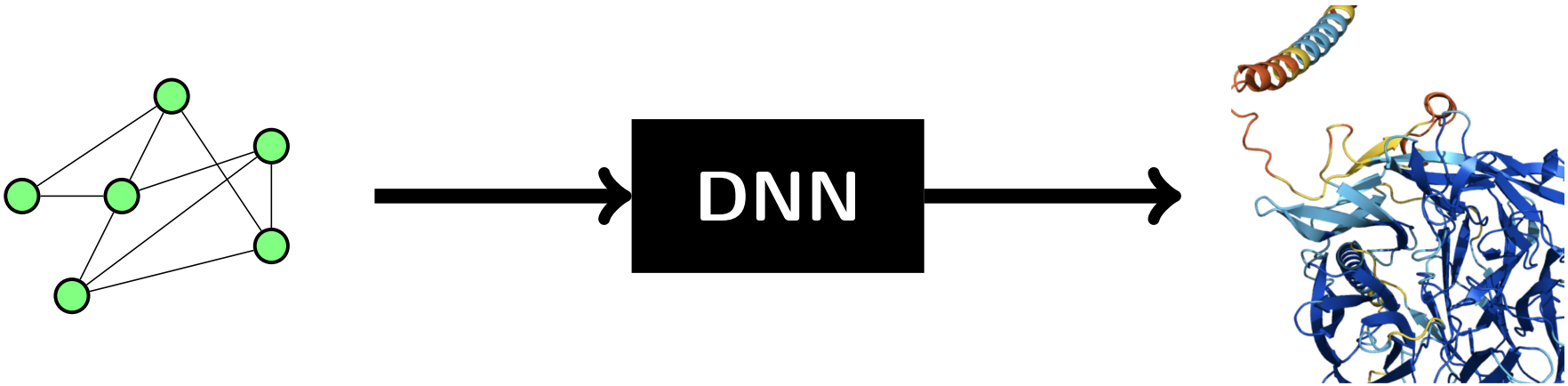
What Can Deep Neural Networks Do?



What Can Deep Neural Networks Do?



What Can Deep Neural Networks Do?



The Goal of Deep Learning

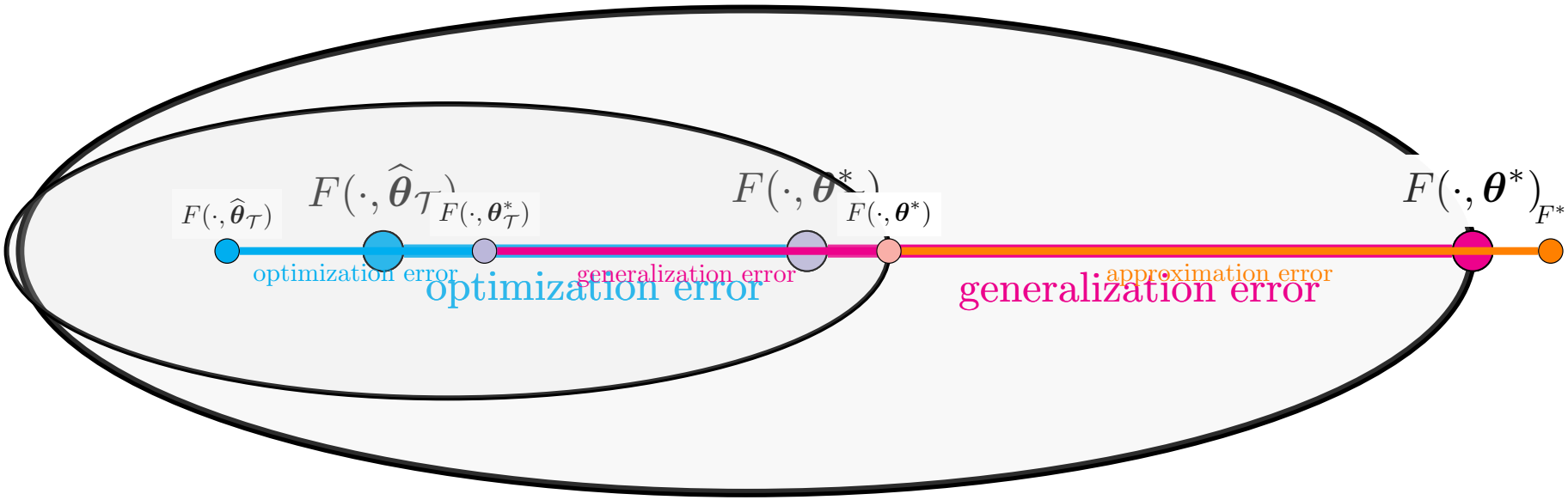
Find a parameterized mapping $F(\mathbf{y}, \boldsymbol{\theta}) \approx \mathbf{c}$ for all input-target pairs $(\mathbf{y}, \mathbf{c})$.

The Training Problem

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{\mathcal{T}} L(F(\mathbf{y}, \boldsymbol{\theta}), \mathbf{c}) + R(\boldsymbol{\theta})$$

The Learning Problem

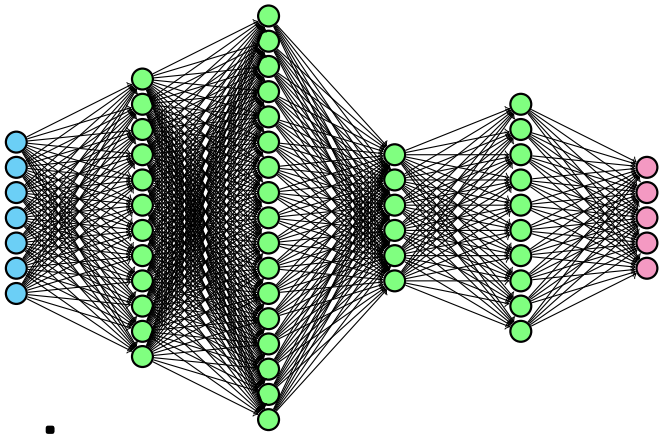
$$\min_{\boldsymbol{\theta}} \mathbb{E} L(F(\mathbf{y}, \boldsymbol{\theta}), \mathbf{c}) + R(\boldsymbol{\theta})$$



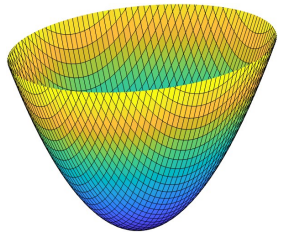
Bottou, Curtis, and Nocedal, "Optimization Methods for Large-Scale Machine Learning"; Lu et al., "DeepXDE: A deep learning library for solving differential equations"

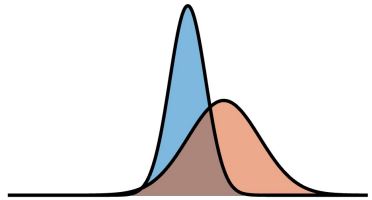
A Recipe for Deep Learning

Architecture

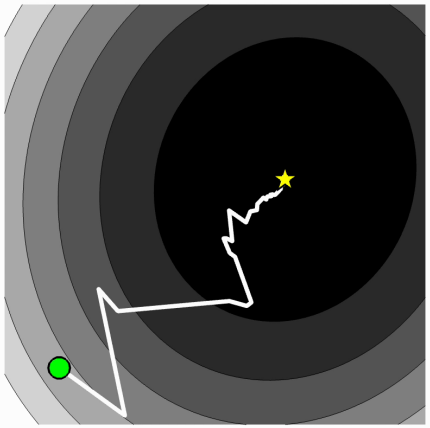


Objective Function

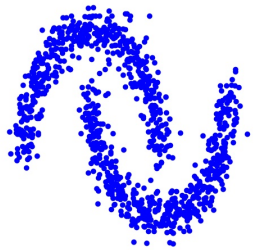


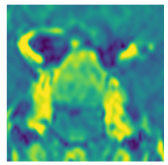


Optimizer

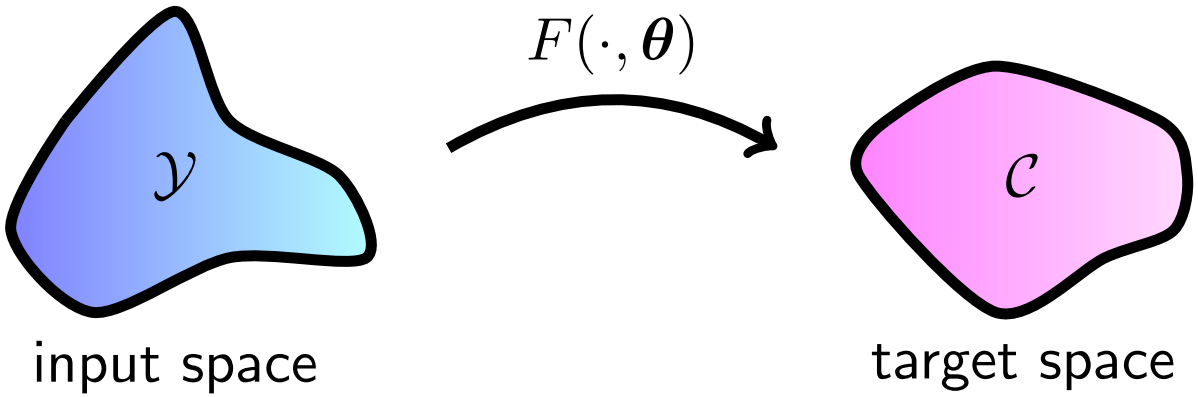


Data





Architectures 101



A **neural network** $F : \mathcal{Y} \times \Theta \to \mathcal{C}$ is a **composite function**

$$F(\mathbf{y}, \boldsymbol{\theta}) = f_d(f_{d-1}(\cdots(f_2(f_1(\mathbf{y}, \boldsymbol{\theta}_1), \boldsymbol{\theta}_2), \cdots), \boldsymbol{\theta}_{d-1}), \boldsymbol{\theta}_d) F(\mathbf{y}, \boldsymbol{\theta}) = f_d(f_{d-1}(\cdot$$

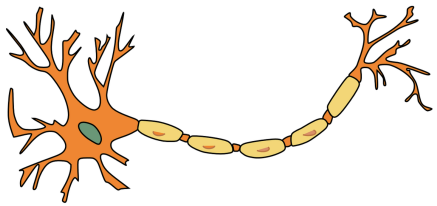
- **weights:** $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_d)$ lying in parameter space Θ
- **layers:** $f_i : \mathbb{R}^{n_i} \times \Theta_i \to \mathbb{R}^{n_{i+1}}$ **deep** = many layers
- **input features:** $\mathbf{y} \in \mathcal{Y} \subseteq \mathbb{R}^n$
- **hidden features:** $\mathbf{u}_i = f_i(\mathbf{u}_{i-1}, \boldsymbol{\theta}_i)$ for $i = 1, \ldots, d$ with $\mathbf{u}_0 = \mathbf{y}$.
- **target features:** $\mathbf{c} \in \mathcal{C} \subseteq \mathbb{R}^m$

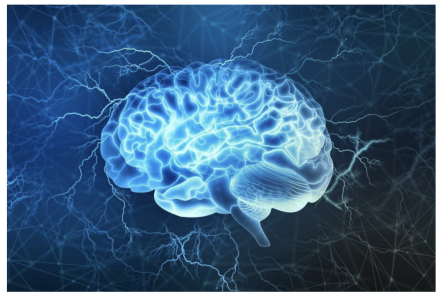
A Biological Perspective on Layers

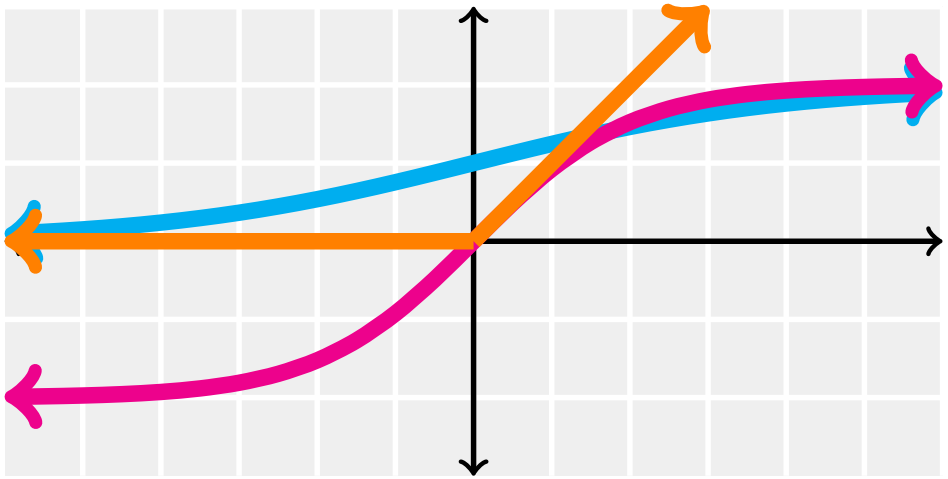
$$f_j(\mathbf{u}_{j-1}, \underbrace{(\mathbf{K}_j, \mathbf{b}_j)}_{\boldsymbol{\theta}_j}) = \sigma(\mathbf{K}_j \mathbf{u}_{j-1} + \mathbf{b}_j)$$

affine transformation + activation

- **weight** matrix $\mathbf{K}_j$
- **bias** vector $\mathbf{b}_j$
- **activation function** $\sigma : \mathbb{R} \to \mathbb{R}$ (pointwise)







$$\sigma(z) = \frac{1}{1+e^{-z}}$$

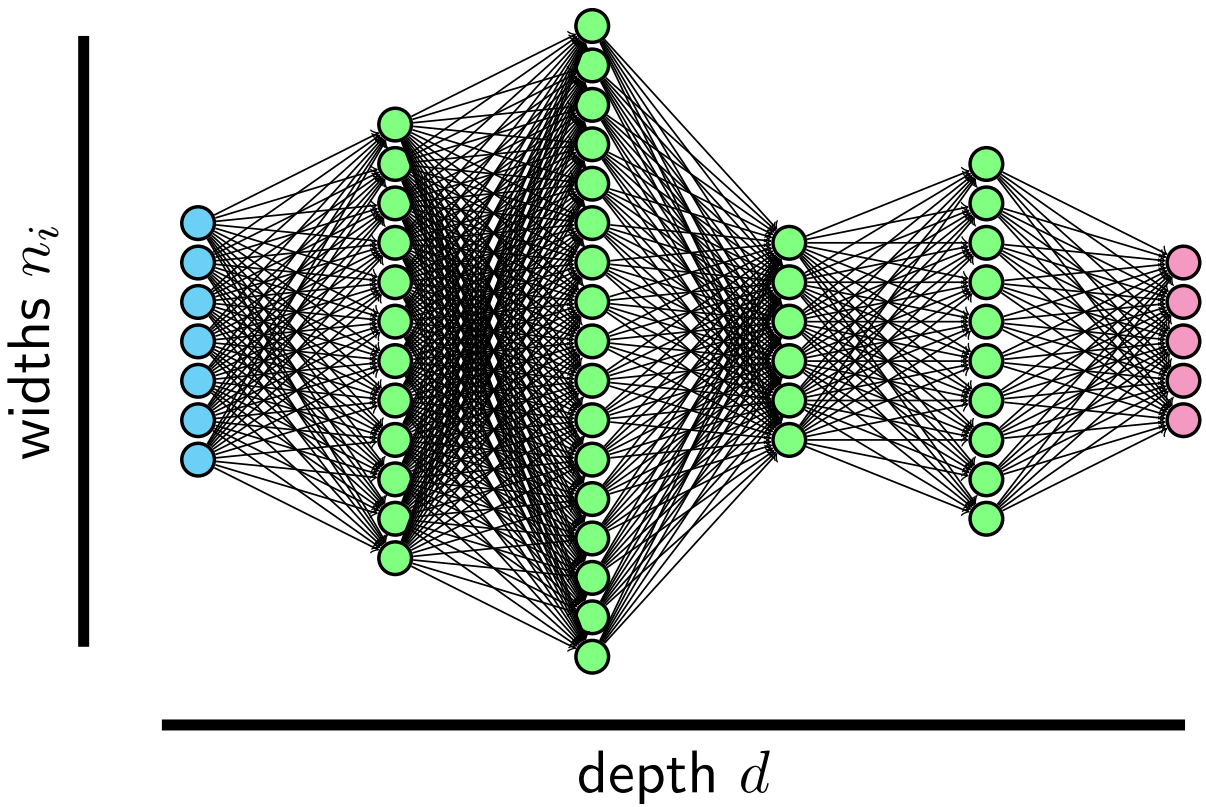
$$\sigma(z) = \tanh(z)$$

$$\sigma(z) = \max(z, 0)$$

https://sciencetrends.com/labeled-neuron-diagram/, https://mindsetsd.com/how-neurons-work-in-the-brain/

Putting It Together

$$F(\mathbf{y}, \boldsymbol{\theta}) = f_d(f_{d-1}(\cdots(f_2(f_1(\mathbf{y}, \boldsymbol{\theta}_1), \boldsymbol{\theta}_2), \cdots), \boldsymbol{\theta}_{d-1}), \boldsymbol{\theta}_d)$$

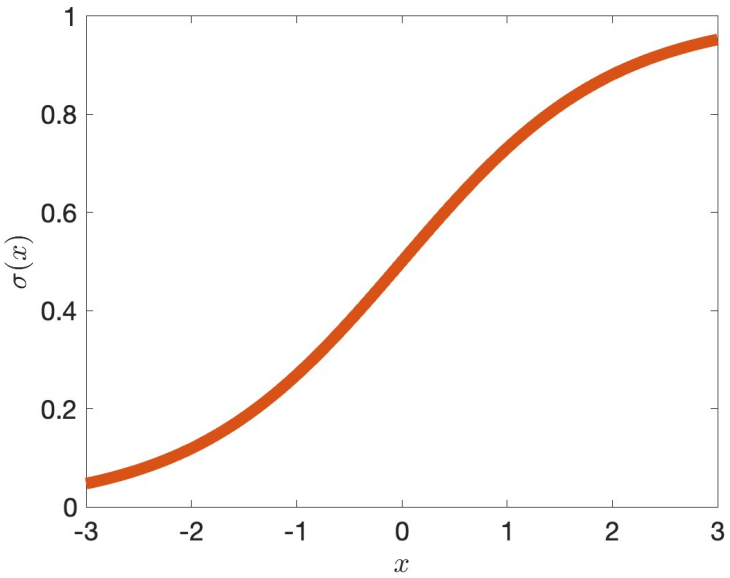


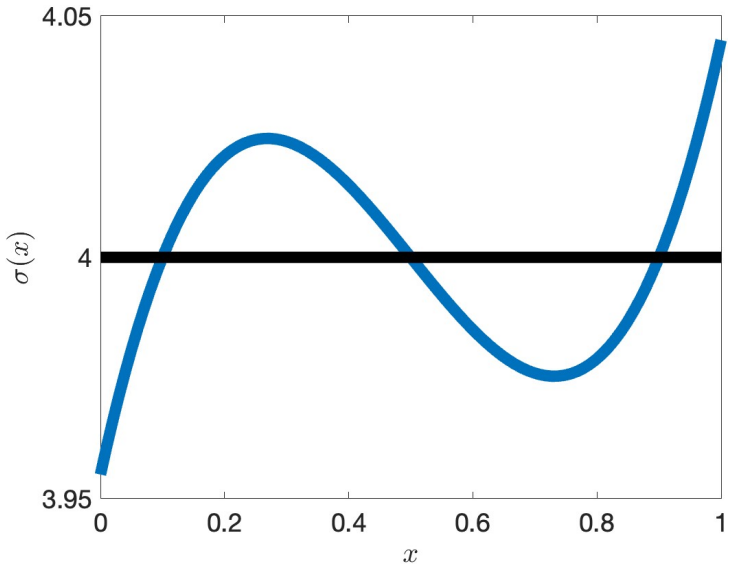
Universal Approximation

Theorem (slightly simplified)

Let $f : \mathcal{C}([0,1]^n) \to \mathbb{R}$ and let σ be a sigmoidal function. Given $\varepsilon > 0$, there exists a dimension $m \in \mathbb{N}$ such that for $\mathbf{K} \in \mathbb{R}^{m \times n}$ and $\mathbf{w}, \mathbf{b} \in \mathbb{R}^m$, we have

$$\left| \mathbf{w}^\top \sigma(\mathbf{K}\mathbf{x} + \mathbf{b}) - f(\mathbf{x}) \right| < \varepsilon \quad \textit{for all} \quad \mathbf{x} \in [0,1]^n.$$





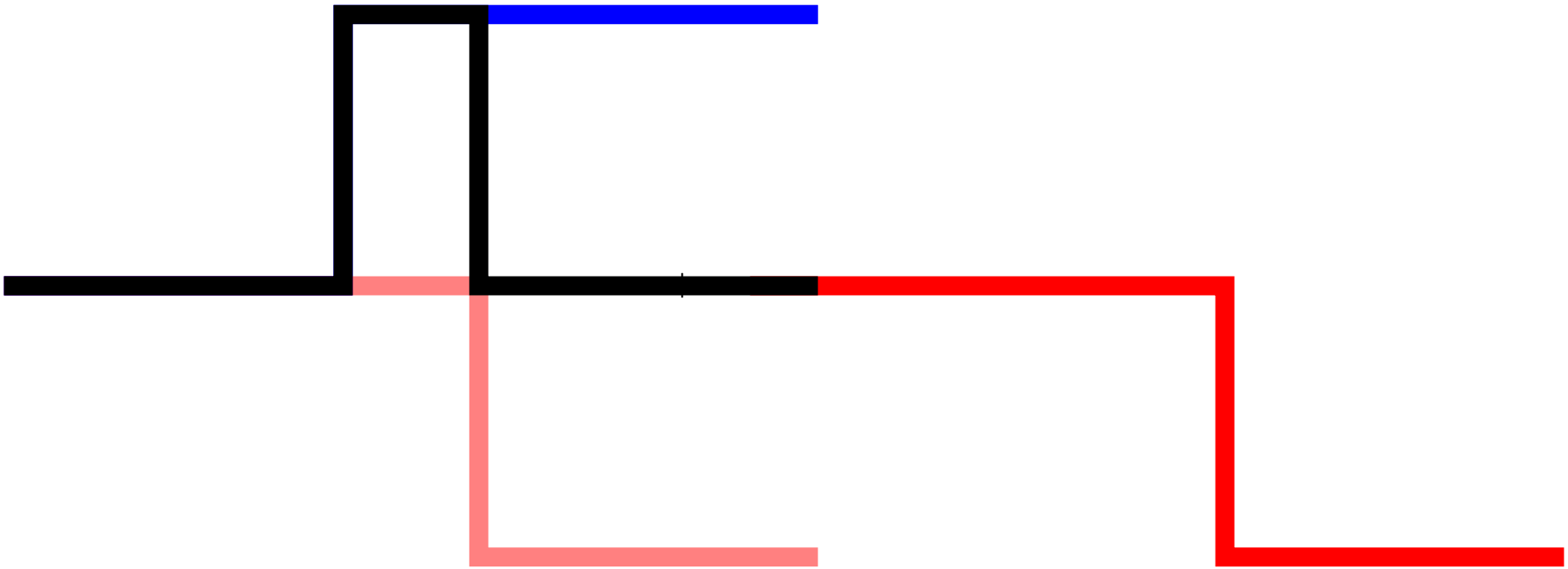
Cybenko, "Approximation by superpositions of a sigmoidal function"

Universal Approximation

Theorem (slightly simplified)

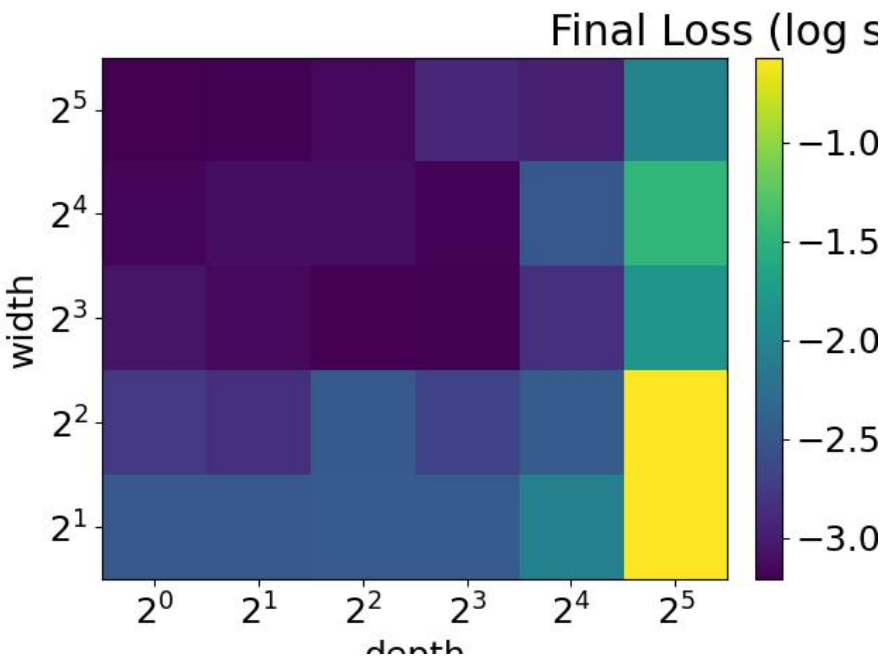
Let $f : \mathcal{C}([0,1]^n) \to \mathbb{R}$ *and let* σ *be a sigmoidal function. Given* $\varepsilon > 0$*, there exists a dimension* $m \in \mathbb{N}$ *such that for* $\mathbf{K} \in \mathbb{R}^{m \times n}$ *and* $\mathbf{w}, \mathbf{b} \in \mathbb{R}^m$*, we have*

$$\left| \mathbf{w}^\top \sigma(\mathbf{K}\mathbf{x} + \mathbf{b}) - f(\mathbf{x}) \right| < \varepsilon \quad \textit{for all} \quad \mathbf{x} \in [0,1]^n.$$

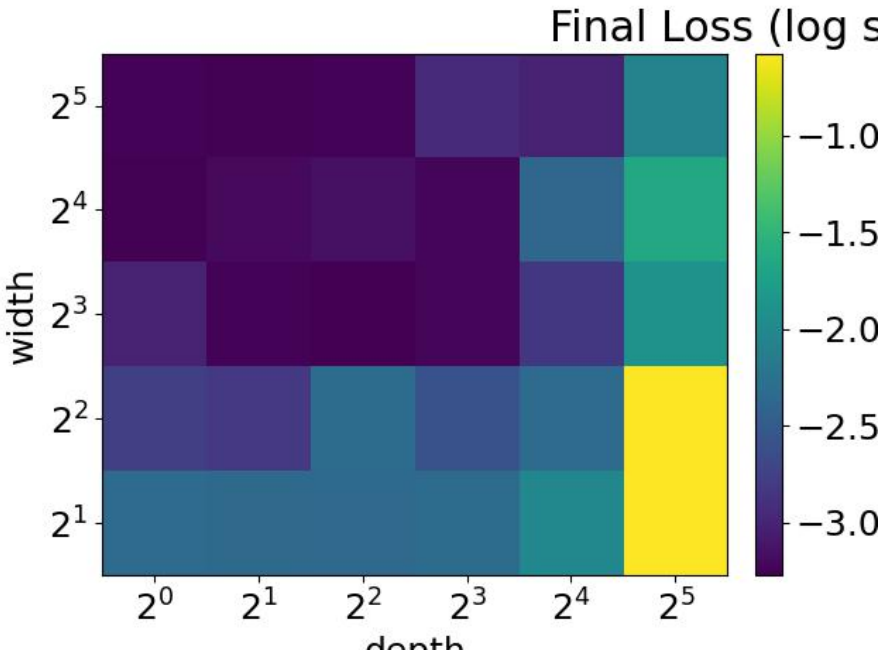


Width vs. Depth

Final Training Loss



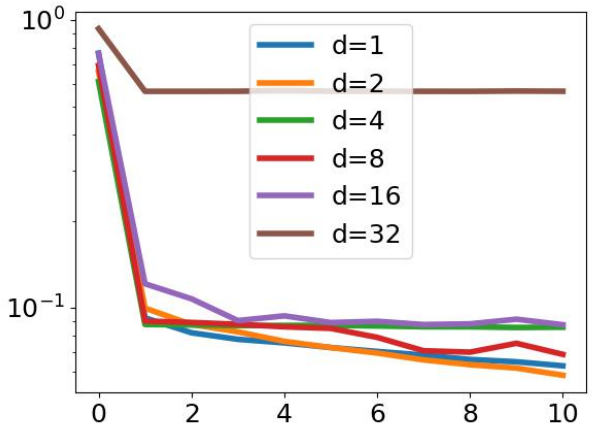
Final Validation Loss



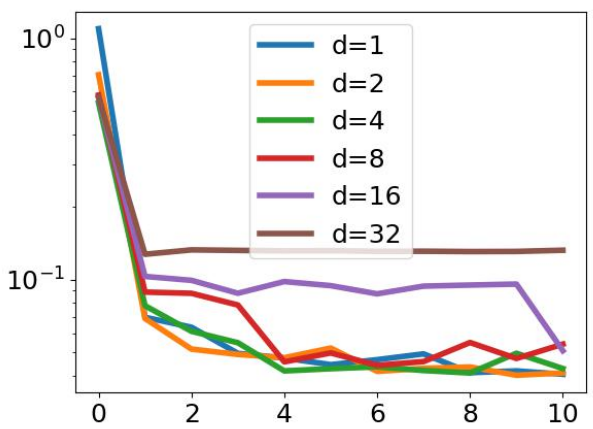
Setup: $F : \mathbb{R} \times \Theta \to \mathbb{R}$, fully-connected, trained with Adam for fixed number of epochs.

Width vs. Depth

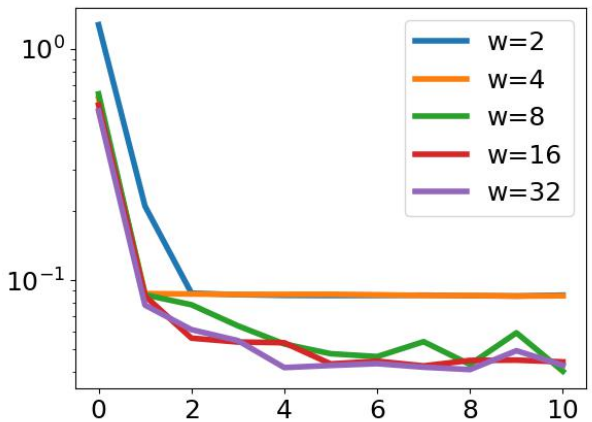
width $= 4$



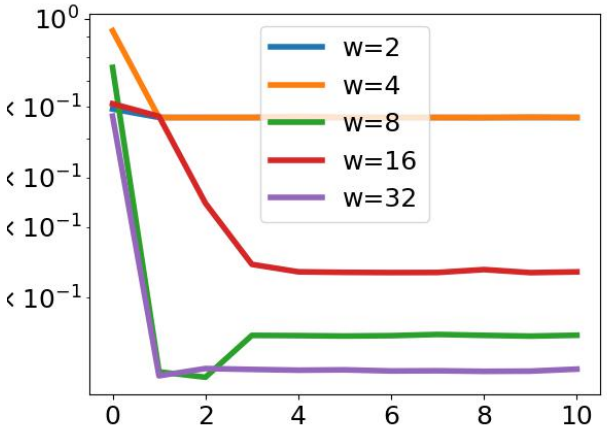
width $= 32$



depth $= 4$



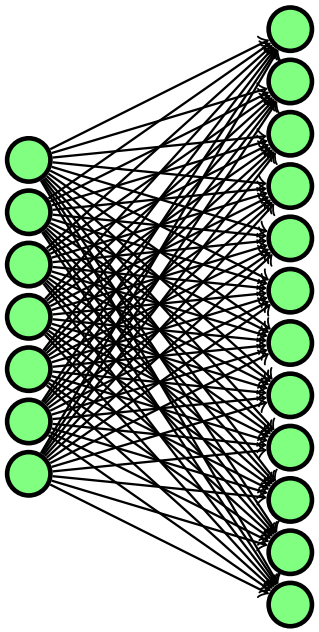
depth $= 32$



Many Types of Layers

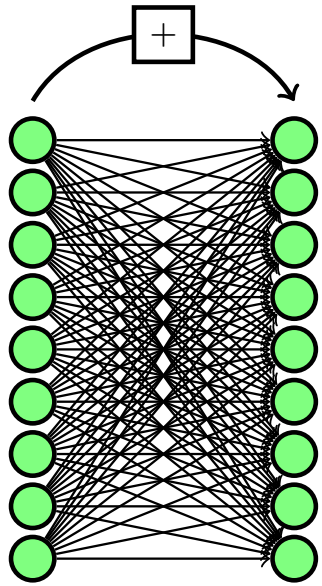
Fully-Connected

$$\mathbf{u}_j = \sigma(\mathbf{K}_j \mathbf{u}_{j-1} + \mathbf{b}_j)$$



Residual Layer

$$\mathbf{u}_j = \mathbf{u}_{j-1} + \sigma(\mathbf{K}_j \mathbf{u}_{j-1} + \mathbf{b}_j)$$

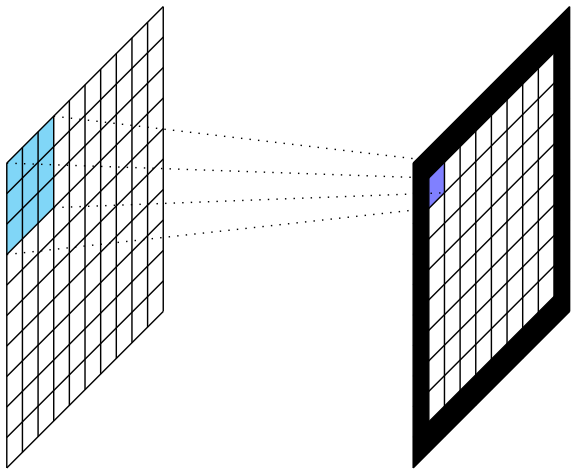


Krizhevsky, Sutskever, and Hinton, "ImageNet Classification with Deep Convolutional Neural Networks"; He et al., "Deep Residual Learning for Image Recognition"

Many Types of Layers

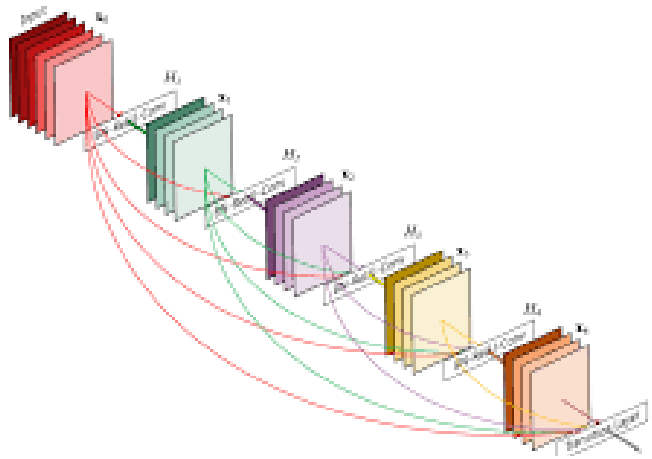
Convolutional Layers

$$\mathbf{u}_j = \sigma(\mathbf{K}(\boldsymbol{\theta}_j)\mathbf{u}_{j-1} + \mathbf{b}_j)$$



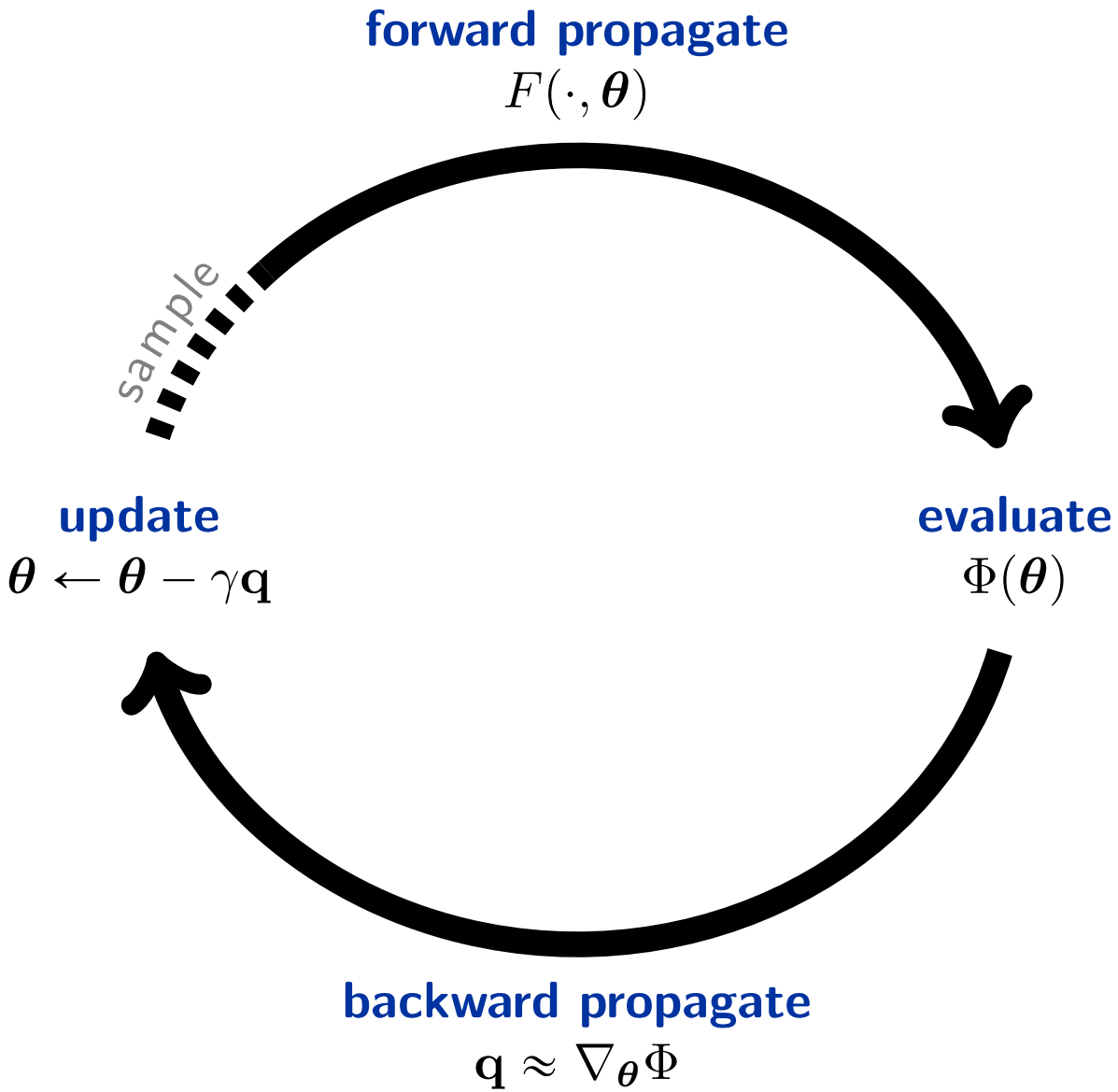
Dense Layers

$$\mathbf{u}_j = f(\mathbf{u}_{j-1}, \mathbf{u}_{j-2}, \ldots, \mathbf{u}_0, \boldsymbol{\theta})$$

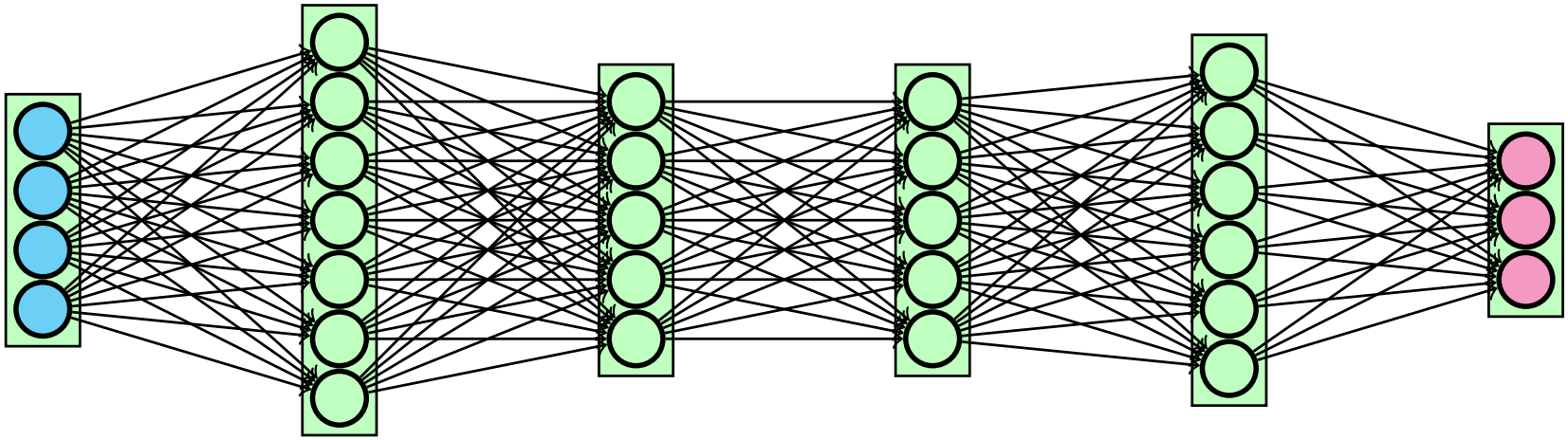


Krizhevsky, Sutskever, and Hinton, "ImageNet Classification with Deep Convolutional Neural Networks"; Huang et al., "Densely Connected Convolutional Networks"

The Training Cycle



Forward Propagation: Composition

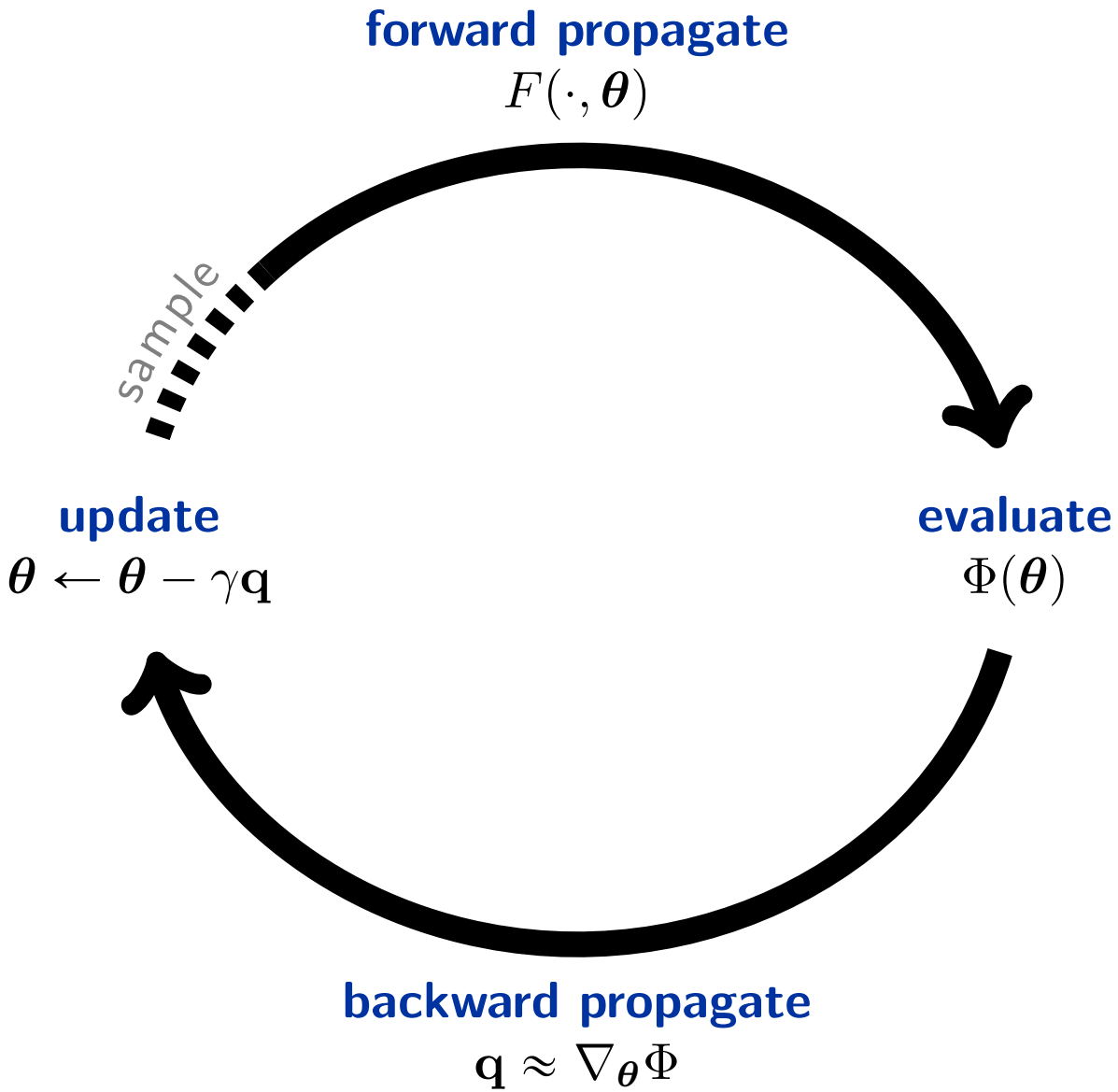


$\mathbf{u}_0$ $\mathbf{u}_1$ $\mathbf{u}_2$ $\mathbf{u}_3$ $\mathbf{u}_4$ $\mathbf{u}_5$

$\mathbf{y}$ $f_1(\mathbf{u}_0, \boldsymbol{\theta}_1)$ $f_2(\mathbf{u}_1, \boldsymbol{\theta}_2)$ $f_3(\mathbf{u}_2, \boldsymbol{\theta}_3)$ $f_4(\mathbf{u}_3, \boldsymbol{\theta}_4)$ $f_5(\mathbf{u}_4, \boldsymbol{\theta}_5)$

$$F(\mathbf{y}, \boldsymbol{\theta}) = f_d\left(f_{d-1}\left(\cdots\left(f_2\left(f_1\left(\mathbf{y}, \boldsymbol{\theta}_1\right), \boldsymbol{\theta}_2\right), \cdots\right), \boldsymbol{\theta}_{d-1}\right), \boldsymbol{\theta}_d\right)$$

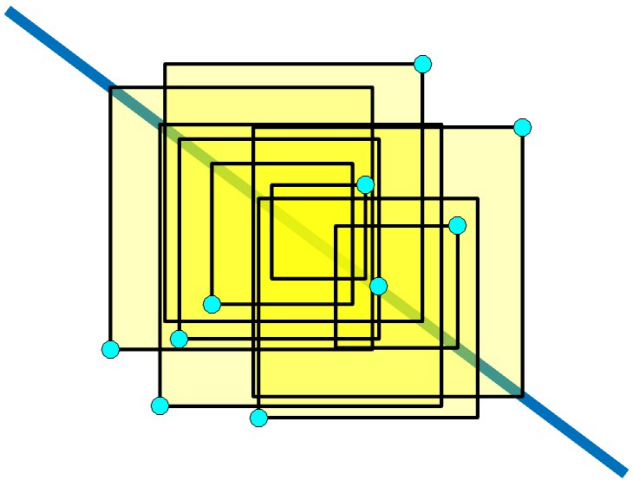
The Training Cycle



Loss Functions

Least Squares

$$L_{\text{LS}}(F(\mathbf{y}, \boldsymbol{\theta}), \mathbf{c}) = \tfrac{1}{2}\|F(\mathbf{y}, \boldsymbol{\theta}) - \mathbf{c}\|_2^2$$



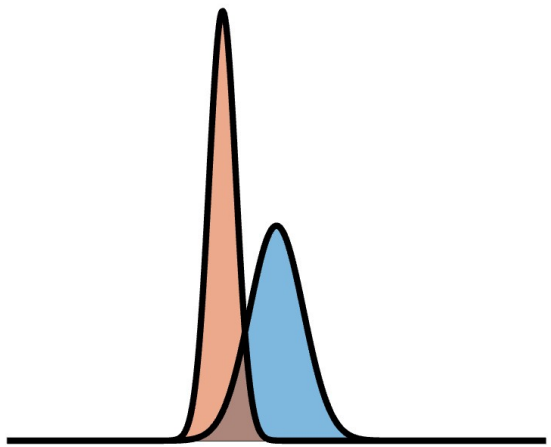
best approximation to data

Cross Entropy

$$L_{\text{CE}}(F(\mathbf{y}, \boldsymbol{\theta}), \mathbf{c}) = -\mathbf{c}^\top \log \sigma(F(\mathbf{y}, \boldsymbol{\theta}))$$

where σ is the **softmax function**

$$\sigma(\mathbf{z}) = \frac{\exp(\mathbf{z})}{\mathbf{1}^\top \exp(\mathbf{z})}$$

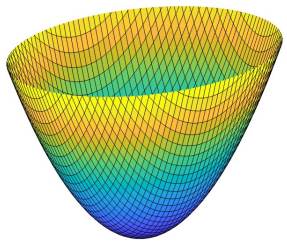


distance between distributions

Loss Functions

Least Squares

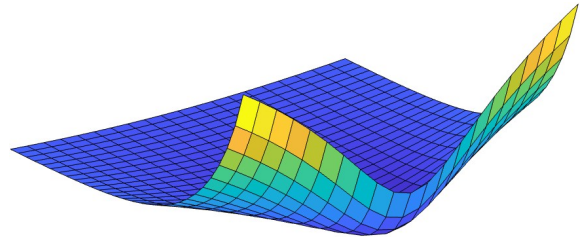
$$L_{\text{LS}}(F(\mathbf{y}, \boldsymbol{\theta}), \mathbf{c}) = \tfrac{1}{2}\|F(\mathbf{y}, \boldsymbol{\theta}) - \mathbf{c}\|_2^2$$



- $L_{\text{LS}} : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}_+$
- function approximation
- convex in first argument

Cross Entropy

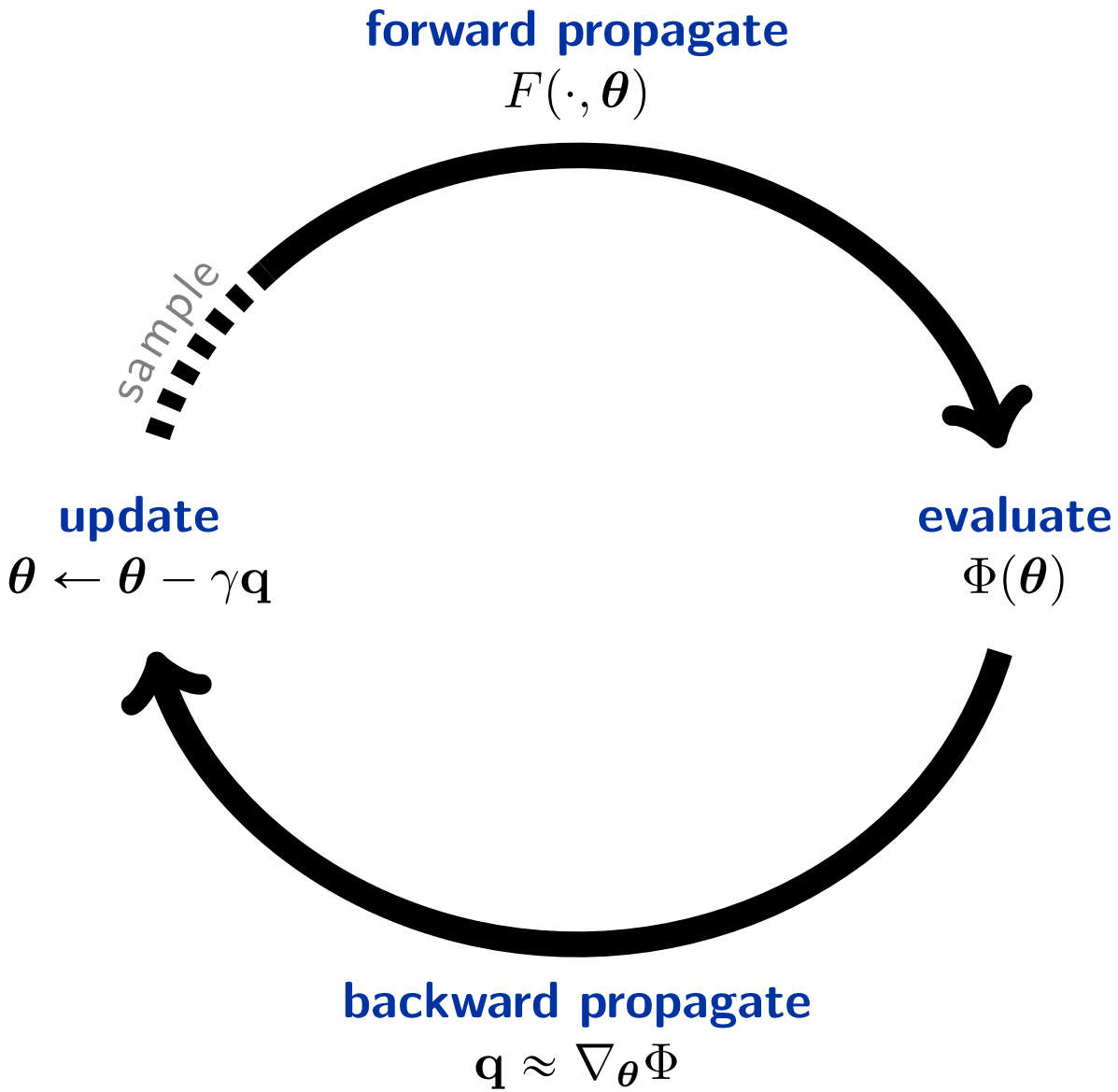
$$L_{\text{CE}}(F(\mathbf{y}, \boldsymbol{\theta}), \mathbf{c}) = -\mathbf{c}^\top \log \sigma(F(\mathbf{y}, \boldsymbol{\theta}))$$



- $L_{\text{CE}} : \mathbb{R}^m \times \Delta^m \to \mathbb{R}_+$
- classification
- convex in first argument

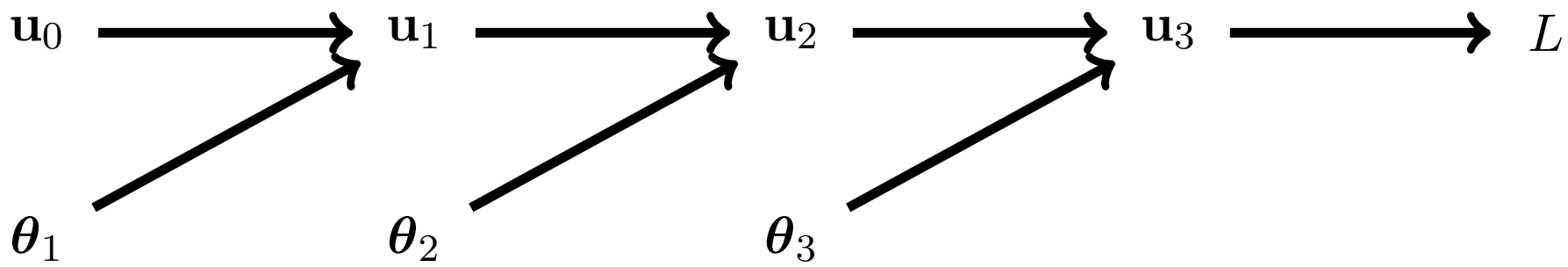
Here, $\Delta^m = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{1}^\top \mathbf{x} = 0, x_i \geqslant 0 \text{ for } i = 1, \ldots, m\}$ is the unit simplex.

The Training Cycle

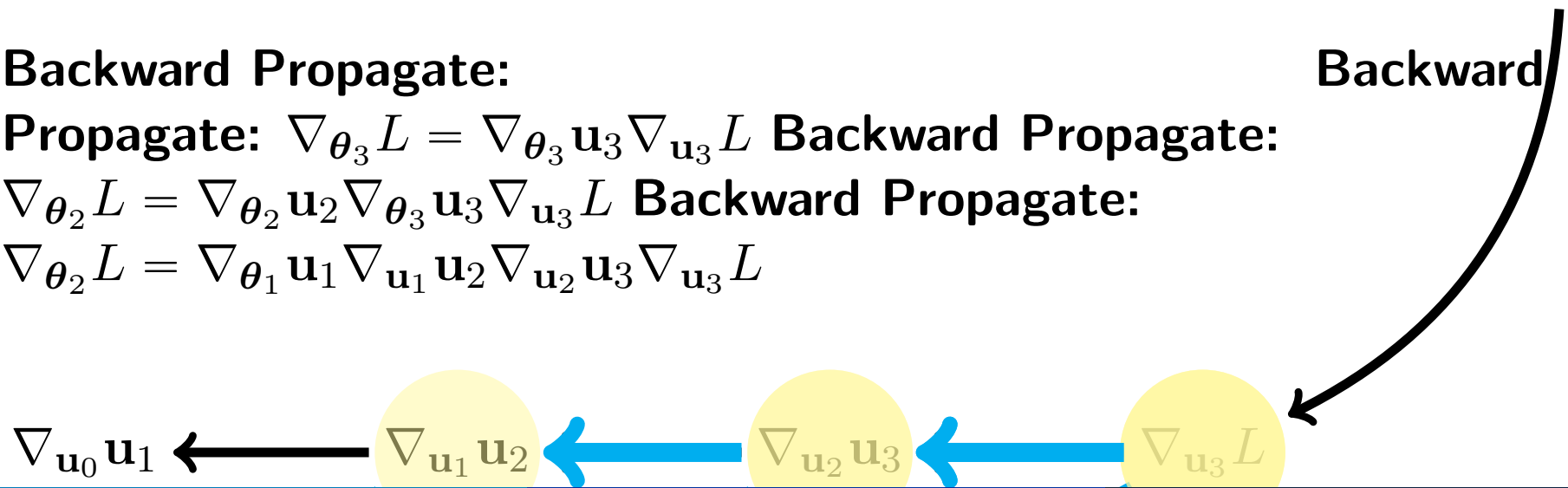


Backward Propagation: Chain Rule

Forward Propagate: $f_3(\mathbf{u}_2, \boldsymbol{\theta}_3)$ **Forward Propagate:** $f_3(f_2(\mathbf{u}_1, \boldsymbol{\theta}_2), \boldsymbol{\theta}_3)$ **Forward Propagate:** $f_3(f_2(f_1(\mathbf{u}_0, \boldsymbol{\theta}_1), \boldsymbol{\theta}_2), \boldsymbol{\theta}_3)$



Backward Propagate: $\nabla_{\boldsymbol{\theta}_3} L = \nabla_{\boldsymbol{\theta}_3} \mathbf{u}_3 \nabla_{\mathbf{u}_3} L$ **Backward Propagate:** $\nabla_{\boldsymbol{\theta}_2} L = \nabla_{\boldsymbol{\theta}_2} \mathbf{u}_2 \nabla_{\boldsymbol{\theta}_3} \mathbf{u}_3 \nabla_{\mathbf{u}_3} L$ **Backward Propagate:** $\nabla_{\boldsymbol{\theta}_2} L = \nabla_{\boldsymbol{\theta}_1} \mathbf{u}_1 \nabla_{\mathbf{u}_1} \mathbf{u}_2 \nabla_{\mathbf{u}_2} \mathbf{u}_3 \nabla_{\mathbf{u}_3} L$ **Backward**



Example: Backward Propagation

Architecture: $F : \mathbb{R}^2 \times \Theta \to \mathbb{R}^2$

$$F(\mathbf{y}, \boldsymbol{\theta}) = \mathbf{W}a(\mathbf{K}\mathbf{y} + \mathbf{b})$$

Forward:

$$\begin{aligned}\mathbf{u}_1 &= a(\mathbf{K}\mathbf{y} + \mathbf{b}) \\ \mathbf{u}_2 &= \mathbf{W}\mathbf{u}_1\end{aligned}$$

Loss: $L : \mathbb{R}^2 \times \Delta^2 \to \mathbb{R}_+$

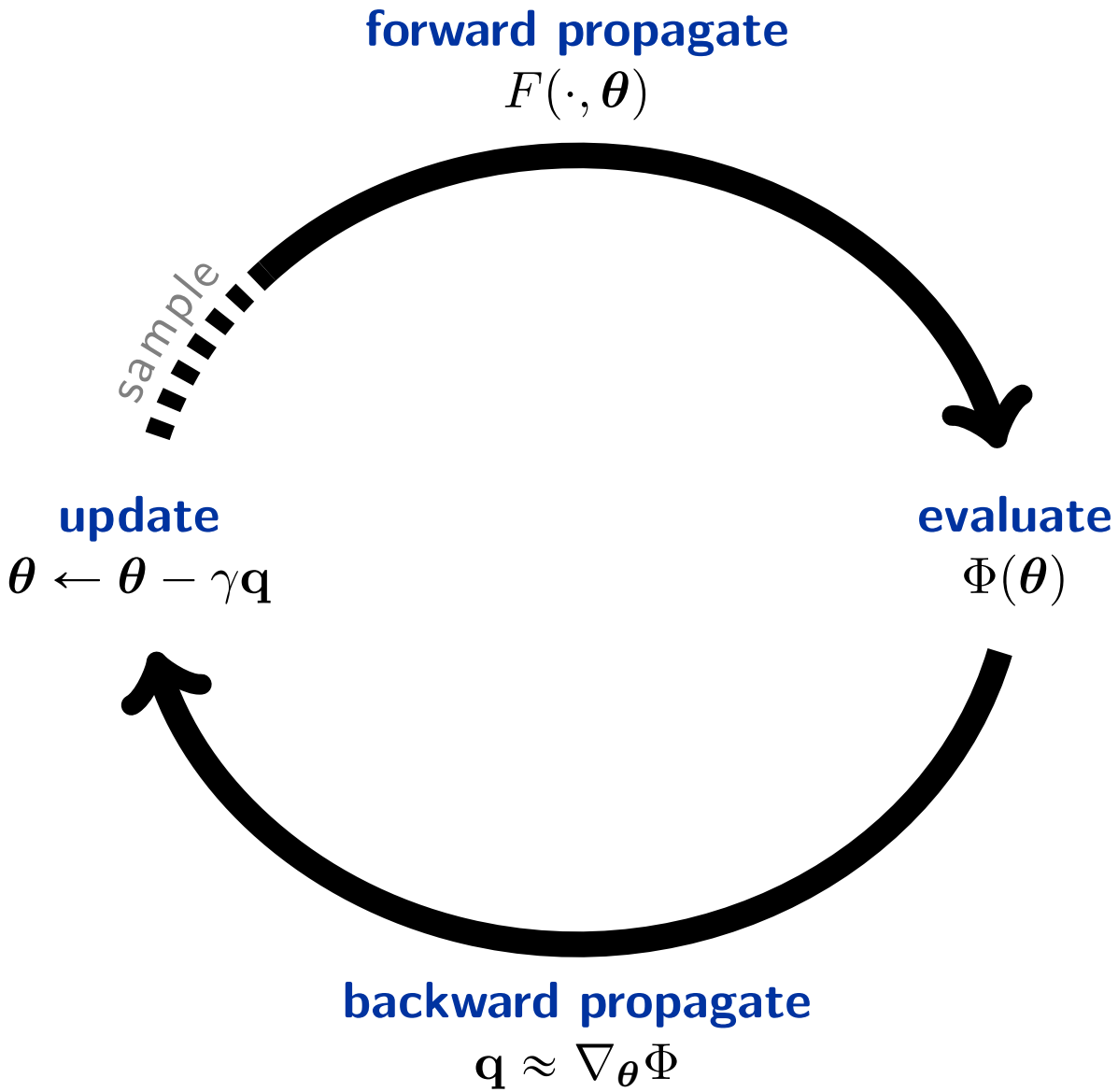
$$\begin{aligned}L(\mathbf{u}_2, \mathbf{c}) &= -\mathbf{c}^\top \log \sigma(\mathbf{u}_2) \\ \nabla_{\mathbf{u}_2} L(\mathbf{u}_2, \mathbf{c}) &= \mathbf{c} - \sigma(\mathbf{u}_2)\end{aligned}$$

Backward:

$$\begin{aligned}\nabla_{\mathbf{W}} L(\mathbf{u}_2, \mathbf{c}) &= \nabla_{\mathbf{W}} \mathbf{u}_2 \nabla_{\mathbf{u}_2} L = (\mathbf{c} - \sigma(\mathbf{u}_2))\mathbf{u}_1^\top \\ \nabla_{\mathbf{u}_1} L(\mathbf{u}_2, \mathbf{c}) &= \nabla_{\mathbf{u}_1} \mathbf{u}_2 \nabla_{\mathbf{u}_2} L = \mathbf{W}^\top(\mathbf{c} - \sigma(\mathbf{z})) \\ \nabla_{\mathbf{K}} L(\mathbf{u}_2, \mathbf{c}) &= \nabla_{\mathbf{K}} \mathbf{u}_1 \nabla_{\mathbf{u}_1} L = \operatorname{diag}(a'(\mathbf{K}\mathbf{y} + \mathbf{b}))(\mathbf{c} - \sigma(\mathbf{u}_2))\mathbf{y}^\top \\ \nabla_{\mathbf{b}} L(\mathbf{u}_2, \mathbf{c}) &= \nabla_{\mathbf{b}} \mathbf{u}_1 \nabla_{\mathbf{u}_1} L = \operatorname{diag}(a'(\mathbf{K}\mathbf{y} + \mathbf{b}))(\mathbf{c} - \sigma(\mathbf{u}_2)) \\ \nabla_{\mathbf{y}} L(\mathbf{u}_2, \mathbf{c}) &= \nabla_{\mathbf{y}} \mathbf{u}_1 \nabla_{\mathbf{u}_1} L = \mathbf{K}^\top \operatorname{diag}(a'(\mathbf{K}\mathbf{y} + \mathbf{b}))(\mathbf{c} - \sigma(\mathbf{u}_2))\end{aligned}$$

The notation $\mathbf{c} \in \Delta^2$ means $\mathbf{c}$ is in the unit simplex; that is, $c_1, c_2 \geqslant 0$ and $c_1 + c_2 = 1$.

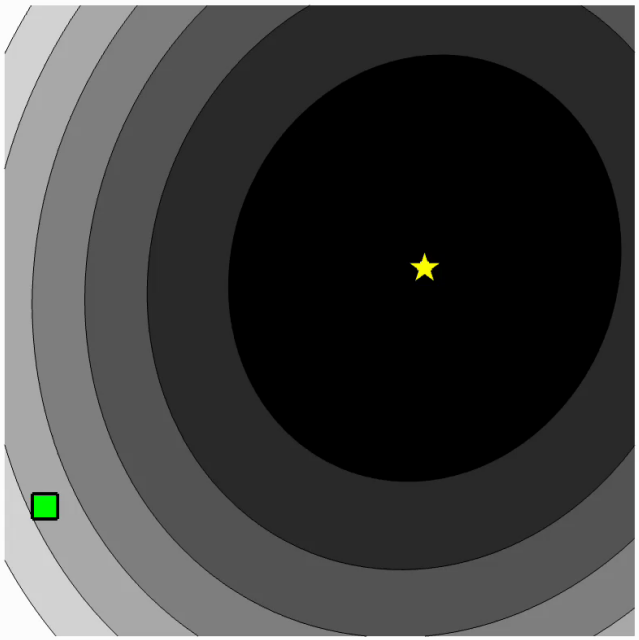
The Training Cycle



Gradient Descent vs. Stochastic Gradient

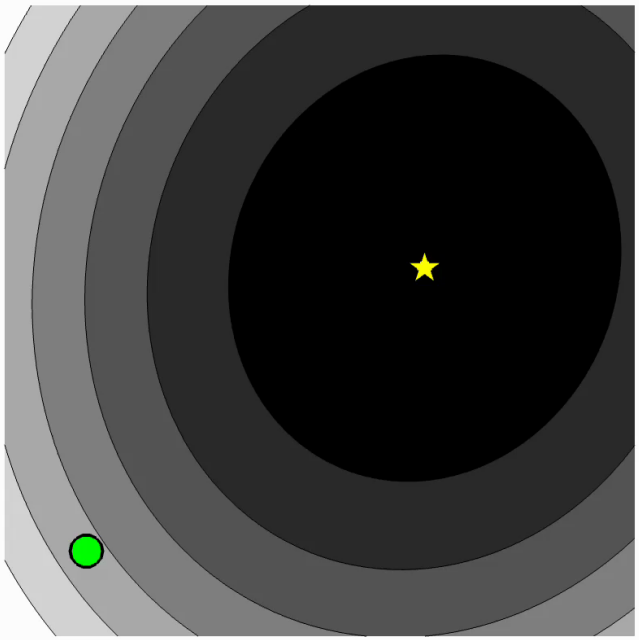
Gradient Descent

$$\min_{\boldsymbol{\theta}} \Phi_{\mathcal{T}}(\boldsymbol{\theta}) \equiv \frac{1}{|\mathcal{T}|} \sum_{(\mathbf{y},\mathbf{c})\in\mathcal{T}} L(F(\mathbf{y},\boldsymbol{\theta}),\mathbf{c})$$



Stochastic Gradient

$$\min_{\boldsymbol{\theta}} \Phi(\boldsymbol{\theta}) \equiv \mathbb{E}\ L(F(\mathbf{y},\boldsymbol{\theta}),\mathbf{c})$$



GD vs. SG

Gradient Descent

$$\min_{\boldsymbol{\theta}} \Phi_{\mathcal{T}}(\boldsymbol{\theta}) \equiv \frac{1}{|\mathcal{T}|} \sum_{(\mathbf{y},\mathbf{c})\in\mathcal{T}} L(F(\mathbf{y},\boldsymbol{\theta}),\mathbf{c})$$

Goal: Find stationary point

$$\nabla\Phi_{\mathcal{T}}(\boldsymbol{\theta}^*) = \mathbf{0}$$

Iteration:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha_k \nabla\Phi_{\mathcal{T}}(\boldsymbol{\theta}_k)$$

- deterministic (empirical risk/sample average approximation)
- descent (with well-chosen step sizes)
- parallelizable, yet computationally demanding iterates
- potential to overfit

Stochastic Gradient

$$\min_{\boldsymbol{\theta}} \Phi(\boldsymbol{\theta}) \equiv \mathbb{E}\ L(F(\mathbf{y},\boldsymbol{\theta}),\mathbf{c})$$

Goal: Let $\Phi_i(\boldsymbol{\theta}) = L(F(\mathbf{y}_i,\boldsymbol{\theta}),\mathbf{c}_i)$.

$$\nabla\Phi(\widehat{\boldsymbol{\theta}}) = \mathbb{E}[\nabla\Phi_i(\widehat{\boldsymbol{\theta}})] = \mathbf{0}$$

Iteration:

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha_i \nabla\Phi_i(\boldsymbol{\theta}_i)$$

- stochastic (expected risk/stochastic approximation)
- no guaranteed decrease
- scalable, cheap iterates, yet hard to parallelize
- implicit regularization due to randomness

Bottou, Curtis, and Nocedal, "Optimization Methods for Large-Scale Machine Learning"; Robbins and Monro, "A Stochastic Approximation Method"; Nocedal and Wright, *Numerical Optimization*

Stochastic Gradient Convergence

Assumption 1

The objective function Φ is **differentiable** and **bounded below**.

Assumption 2

The objective function Φ is **continuously differentiable** and $\nabla\Phi$ is **Lipschitz continuous** with Lipschitz constant $L > 0$; that is,

$$\|\nabla\Phi(\boldsymbol{\theta}) - \nabla\Phi(\widetilde{\boldsymbol{\theta}})\|_2 \leqslant L\|\boldsymbol{\theta} - \widetilde{\boldsymbol{\theta}}\|_2 \quad \text{for all} \quad \boldsymbol{\theta}, \widetilde{\boldsymbol{\theta}} \in \Theta$$

Assumption 3

The stochastic gradient is an unbiased estimate of the expected gradient; that is,

$$\mathbb{E}_i[\nabla\Phi_i(\boldsymbol{\theta})] = \nabla\Phi(\boldsymbol{\theta})$$

where $\mathbb{E}_i$ is the expectation conditioned on current sample $(\mathbf{y}_i, \mathbf{c}_i)$ and weights $\boldsymbol{\theta}_i$.

Bottou, Curtis, and Nocedal, "Optimization Methods for Large-Scale Machine Learning"

Stochastic Gradient Convergence

Given the stochastic iterate

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha_i \nabla \Phi_i(\boldsymbol{\theta}_i)$$

Bound the objective function value

$$\begin{aligned}\Phi(\boldsymbol{\theta}_{i+1}) &\leqslant \Phi(\boldsymbol{\theta}_i) + \nabla\Phi(\boldsymbol{\theta}_i)^\top(\boldsymbol{\theta}_{i+1} - \boldsymbol{\theta}_i) + \tfrac{1}{2}L\|\boldsymbol{\theta}_{i+1} - \boldsymbol{\theta}_i\|_2^2 \qquad \textit{by Lipschitz continuity} \\ &= \Phi(\boldsymbol{\theta}_i) - \alpha_i \nabla\Phi(\boldsymbol{\theta}_i)^\top \nabla\Phi_i(\boldsymbol{\theta}_i) + \tfrac{1}{2}\alpha_i^2 L\|\nabla\Phi_i(\boldsymbol{\theta}_i)\|_2^2\end{aligned}$$

Take the conditional expectation

$$\begin{aligned}\mathbb{E}_i[\Phi(\boldsymbol{\theta}_{i+1})] &\leqslant \mathbb{E}_i[\Phi(\boldsymbol{\theta}_i)] - \alpha_i \mathbb{E}_i[\nabla\Phi(\boldsymbol{\theta}_i)^\top \nabla\Phi_i(\boldsymbol{\theta}_i)] + \tfrac{1}{2}\alpha_i^2 L\mathbb{E}_i[\|\nabla\Phi_i(\boldsymbol{\theta}_i)\|_2^2] \\ &= \mathbb{E}_i[\Phi(\boldsymbol{\theta}_i)] - \alpha_i\|\nabla\Phi(\boldsymbol{\theta}_i)\|_2^2 + \tfrac{1}{2}\alpha_i^2 L\mathbb{E}_i[\|\nabla\Phi_i(\boldsymbol{\theta}_i)\|_2^2] \qquad \textit{unbiased gradient}\end{aligned}$$

Choose step sizes to ensure decrease in the long run

$$\sum_i \alpha_i = \infty \qquad \text{and} \qquad \sum_i \alpha_i^2 < \infty$$

Bottou, Curtis, and Nocedal, "Optimization Methods for Large-Scale Machine Learning"

Mini-Batch Stochastic Gradient

Pseudocode

while not converged
  random partition $\mathcal{T} = \bigsqcup_{i=1}^{n_b} \mathcal{T}_i$
  for $i = 1, \ldots, n_b$
    for $(\mathbf{y}_j, \mathbf{c}_j) \in \mathcal{T}_i$
      $\hat{\mathbf{c}}_j = f(\mathbf{y}_j, \boldsymbol{\theta}_i)$
      $\phi_j = L(\hat{\mathbf{c}}_j, \mathbf{c}_j)$
    end for
    $\Phi_i = \frac{1}{|\mathcal{T}_i|} \sum_{j=1}^{|\mathcal{T}_i|} \phi_j$
    $\mathbf{g}_i = \nabla \Phi_i(\boldsymbol{\theta}_i)$
    $\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha_i \mathbf{g}_i$
  end for
end while

PyTorch Code

```
for epoch in range(max_epochs):
    # shuffle data
    shuffle_idx = randperm(n_samples)

    for epoch in range(nb):
        # select batch
        idx = shuffle_idx[i*nb:(i+1)*nb]
        yb, cb = y_train[idx], c_train[idx]

        # zero out gradients
        optimizer.zero_grad()

        # forward propagate
        cb_hat = net(yb)

        # evaluate
        phi = loss(cb_hat, cb)

        # backward propagate
        phi.backward()

        # update
        optimizer.step()
```

Popular Stochastic Gradient Variants

1: **while** not converged

2: random partition $\mathcal{T} = \bigsqcup_{i=1}^{n_b} \mathcal{T}_i$

3: **for** $i = 1, \ldots, n_b$

4: **for** $(\mathbf{y}_j, \mathbf{c}_j) \in \mathcal{T}_i$

5: $\widehat{\mathbf{c}}_j = F(\mathbf{y}_j, \boldsymbol{\theta}_i)$

6: $\phi_j = L(\widehat{\mathbf{c}}_j, \mathbf{c}_j)$

end for

7: $\Phi_i = \frac{1}{|\mathcal{T}_j|} \sum_{j=1}^{|\mathcal{T}_j|} \phi_j$

8: $\mathbf{g}_i = \nabla \Phi_i$

9: $\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha_i \mathbf{g}_i$

end for

end while

Popular Stochastic Gradient Variants

Stochastic Gradient (SG)

$$1: \mathbf{g}_i = \nabla \Phi_i$$

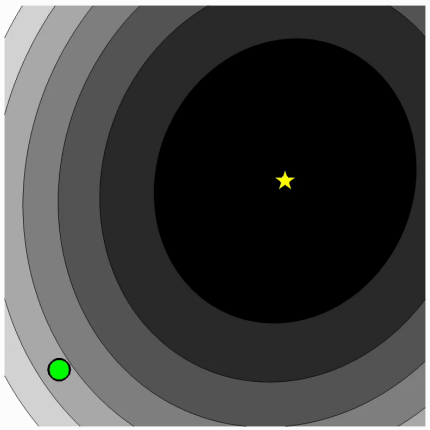
$$2: \boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha_i \mathbf{g}_i$$

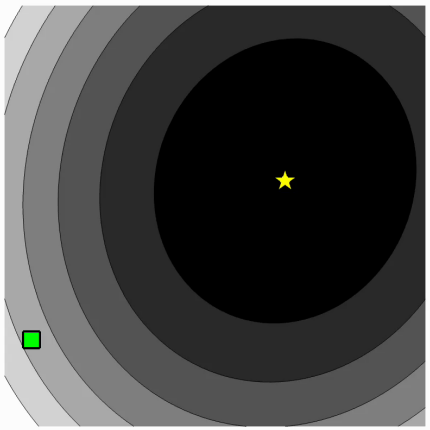
SG + Momentum (SGM)

$$1: \mathbf{g}_i = \nabla \Phi_i$$

$$2: \mathbf{m}_i = \beta \mathbf{m}_{i-1} + (1-\beta)\mathbf{g}_i$$

$$3: \boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha_i \mathbf{m}_i$$





Sutskever et al., "On the importance of initialization and momentum in deep learning"

Popular Stochastic Gradient Variants

Stochastic Gradient (SG)

$$1: \mathbf{g}_i = \nabla \Phi_i$$

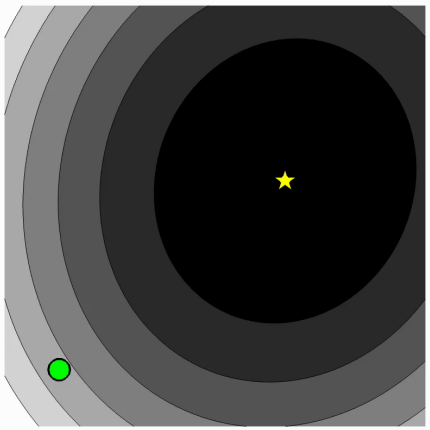
$$2: \boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha_i \mathbf{g}_i$$

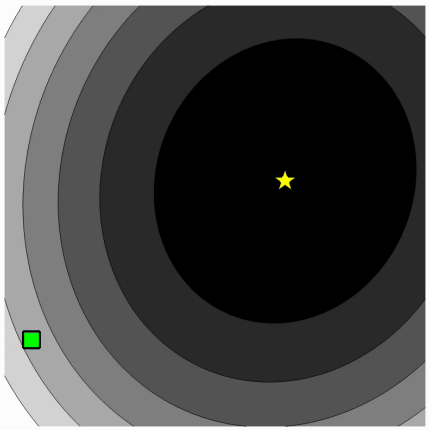
Adaptive Subgrad. (AdaGrad)

$$1: \mathbf{g}_i = \nabla \Phi_i$$

$$2: \mathbf{v}_i = \mathbf{v}_{i-1} + \mathbf{g}_i^2$$

$$3: \boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha_i \left(\frac{\mathbf{g}_i}{\sqrt{\mathbf{v}_i} + \varepsilon} \right)$$





Duchi, Hazan, and Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization"

Popular Stochastic Gradient Variants

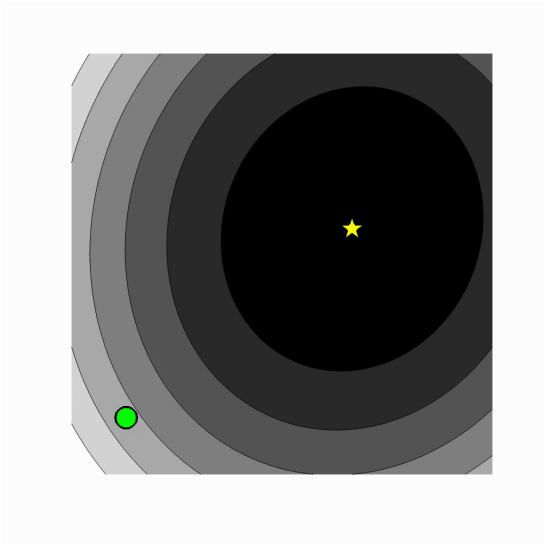
Stochastic Gradient (SG)

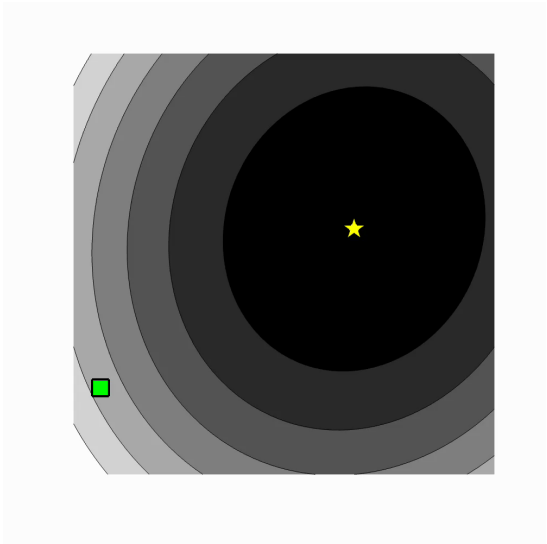
$$1: \mathbf{g}_i = \nabla \Phi_i$$

$$2: \boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha_i \mathbf{g}_i$$

Adaptive Momentum (Adam)

$$\begin{aligned}
1: &\quad \mathbf{g}_i = \nabla \phi_i \\
2: &\quad \mathbf{m}_i = \beta_1 \mathbf{m}_{i-1} + (1-\beta_1)\mathbf{g}_i \\
3: &\quad \mathbf{v}_i = \beta_2 \mathbf{v}_{i-1} + (1-\beta_2)\mathbf{g}_i^2 \\
6: &\quad \widehat{\mathbf{m}}_i = \mathbf{m}_i / (1+\widehat{\beta}_1^i) \\
7: &\quad \widehat{\mathbf{v}}_i = \mathbf{v}_i / (1+\widehat{\beta}_2^i) \\
8: &\quad \boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha_i \left(\frac{\widehat{\mathbf{m}}}{\sqrt{\widehat{\mathbf{v}}}+\varepsilon} \right)
\end{aligned}$$

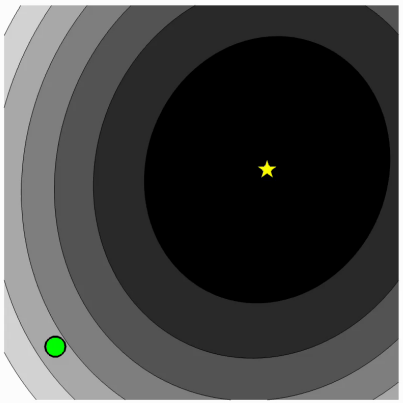




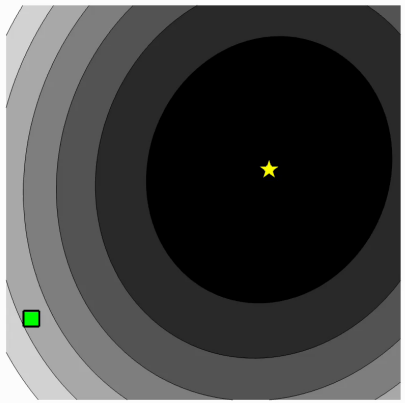
Kingma and Ba, *Adam: A Method for Stochastic Optimization*

Popular Stochastic Gradient Variants

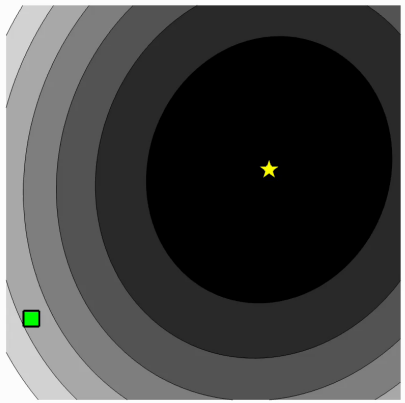
SG



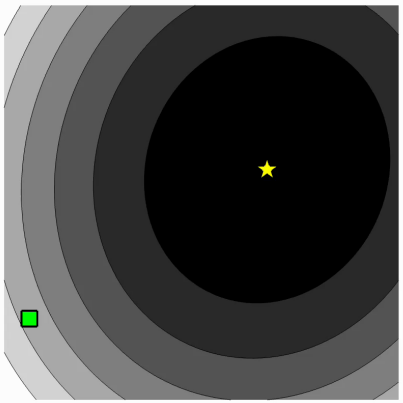
SGM



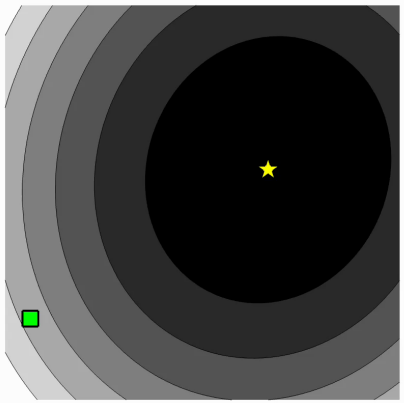
AdaGrad



GD



Adam



Your Turn!

Play with the notebooks in the repository!

https://github.com/elizabethnewman/dnn101

References I

- Bottou, Léon, Frank E. Curtis, and Jorge Nocedal. "Optimization Methods for Large-Scale Machine Learning". In: *SIAM Review* 60.2 (2018), pp. 223–311.
- Cybenko, G. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals and Systems* 2.4 (1989), pp. 303–314.
- Duchi, John, Elad Hazan, and Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *Journal of Machine Learning Research* 12.61 (2011), pp. 2121–2159.
- He, Kaiming et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778.
- Huang, G. et al. "Densely Connected Convolutional Networks". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, July 2017, pp. 2261–2269.
- Kingma, Diederik P. and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017.
- Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012.
- Lu, Lu et al. "DeepXDE: A deep learning library for solving differential equations". In: *SIAM Review* 63 (Feb. 2021), pp. 208–228.

References II


Nocedal, Jorge and Stephen J. Wright. *Numerical Optimization*. 2e. New York, NY, USA: Springer, 2006.


Robbins, Herbert and Sutton Monro. "A Stochastic Approximation Method". In: *The Annals of Mathematical Statistics* 22.3 (1951), pp. 400–407.


Sutskever, Ilya et al. "On the importance of initialization and momentum in deep learning". In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1139–1147.

Cross Entropy + KL Divergence

Entropy (Expected Surprise!)

$$H(p) = \sum_{x \in \mathcal{X}} p(x) \log \left(\frac{1}{p(x)} \right)$$

Cross Entropy

$$H(p, q) = \sum_{x \in \mathcal{X}} p(x) \log \left(\frac{1}{q(x)} \right)$$

Kullback-Leibler Divergence (Expected Relative Entropy)

$$\begin{aligned} D_{KL}(p||q) &= \sum_{x \in \mathcal{X}} p(x) \log \left(\frac{p(x)}{q(x)} \right) \\ &= H(p, q) - H(p) \end{aligned}$$

Cross Entropy + Maximum Likelihood

Setup

- parameterized estimated probability distribution $q_{\boldsymbol{\theta}}$ (e.g., DNN)
- empirical distribution p (e.g., labels)
- N outcomes (e.g., training samples)

Likelihood

$$\mathcal{L}(\boldsymbol{\theta}) = \prod_{x \in \mathcal{X}} q_{\boldsymbol{\theta}}(x)^{Np(x)}$$

Cross Entropy

$$-\frac{1}{N} \log \mathcal{L}(\boldsymbol{\theta}) = -\sum_{x \in \mathcal{X}} p(x) \log q_{\boldsymbol{\theta}}(x) = H(p, q).$$

Cross Entropy Minimization

Claim: $H(p, q)$ is minimized when p and q are the same distributions; that is, show

$$H(p) \leqslant H(p, q).$$

This is also known as Gibb's inequality.

Sketch of Proof: significant simplifying assumption: $p(x), q(x) > 0$ for all $x \in \mathcal{X}$.

(Step 1): Show $D_{KL}(p, q) = H(q, p) - H(p) \geqslant 0$

$$\begin{aligned} D_{KL}(p, q) &= -\sum_{x \in \mathcal{X}} p(x) \log\left(\frac{q(x)}{p(x)}\right) \\ &\geqslant -\sum_{x \in \mathcal{X}} p(x) \left(\frac{q(x)}{p(x)} - 1\right) \\ &= -\sum_{x \in \mathcal{X}} q(x) + \sum_{x \in \mathcal{X}} p(x) \\ &= 0. \end{aligned}$$

(Step 2): Show $D_{KL}(p, q) = 0$ if and only if $p = q$. This is fairly straightforward with the simplifying assumption.