Neural Network Training

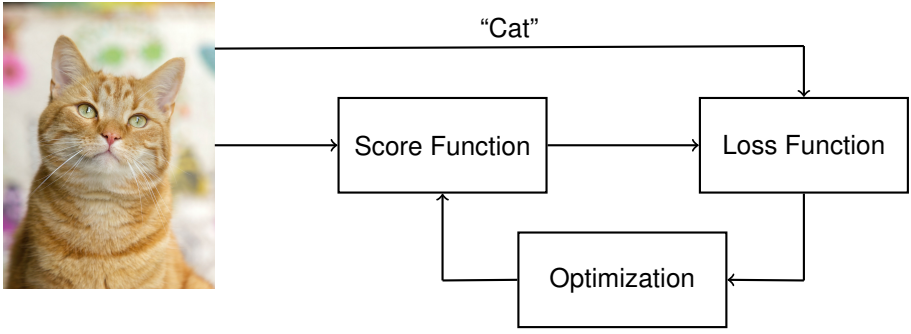
Intelligent Architectures (5LIL0)

dr Alexios Balatsoukas-Stimming

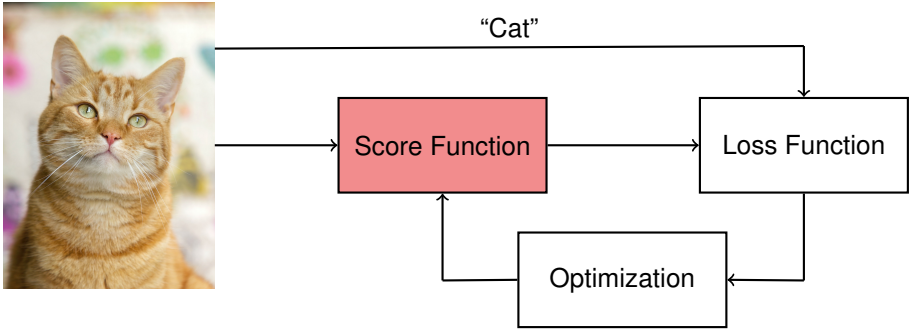
Department of Electrical Engineering, Electronic Systems Group

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

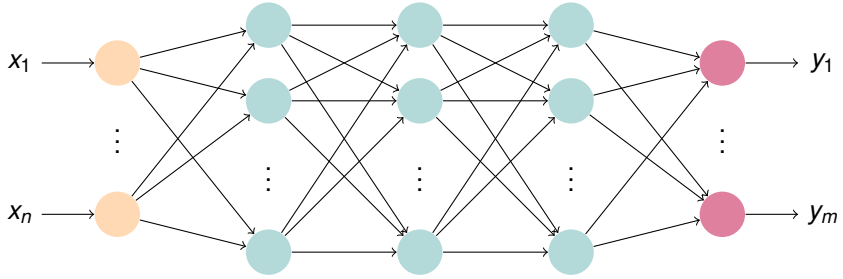
Last Time



Last Time



Last Time

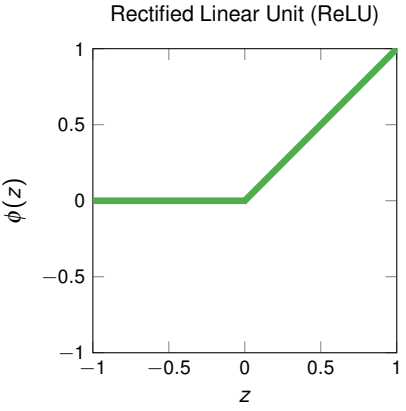


Neural network score function ingredients:

- Input layer
- Hidden layers
- Output layer
- Weights, biases, and activation functions

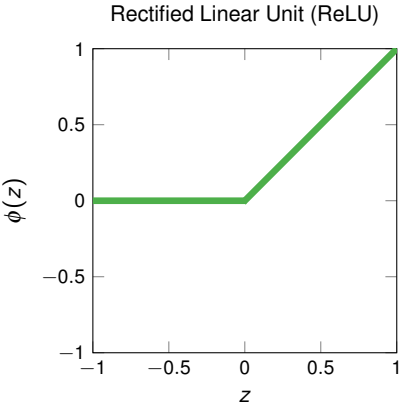
Last Time

- **Activation functions:**

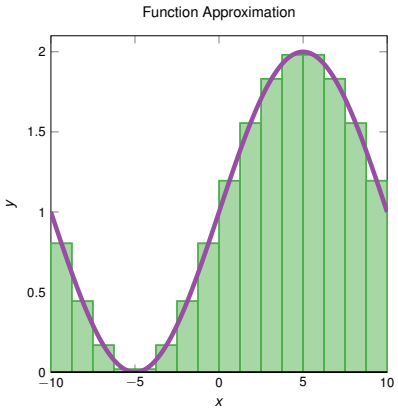


Last Time

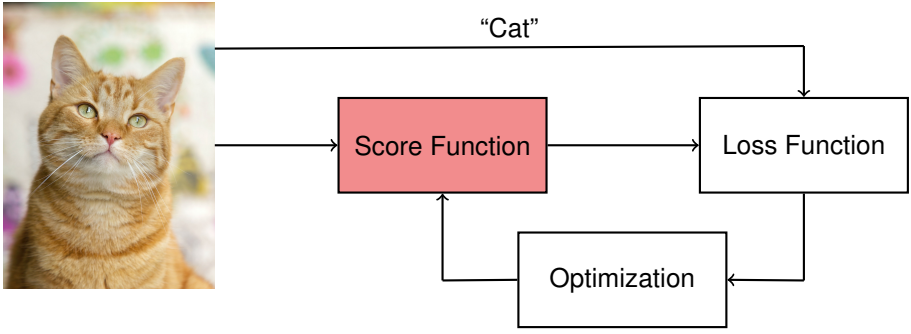
- **Activation functions:**



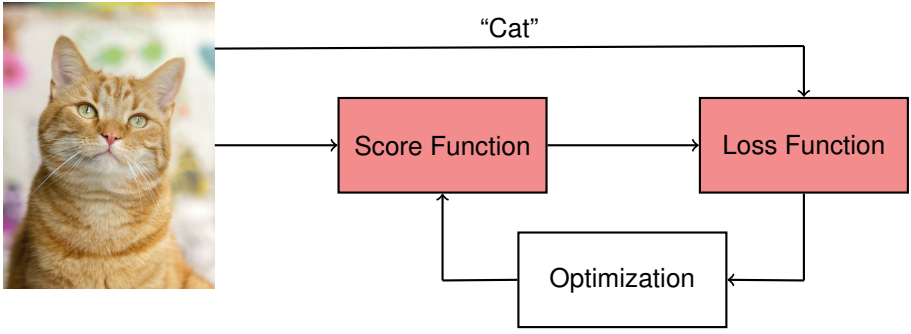
- **Universal approximation theorem:**



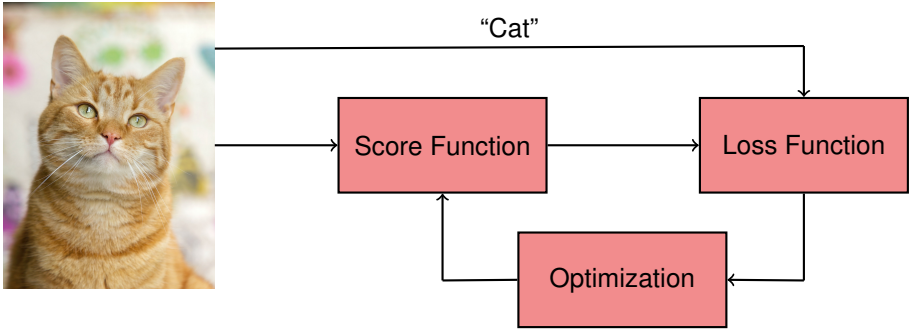
Today



Today



Today



Neural Network Training

Goal

Tune the weights and biases so that the neural network implements a desired function.

Neural Network Training

Goal

Tune the weights and biases so that the neural network implements a desired function.

Two common classes of functions:

1. **Regression** (e.g., predicting tomorrow's temperature)
2. **Classification** (e.g., image recognition)

Neural Network Training

Goal

Tune the weights and biases so that the neural network implements a desired function.

Two common classes of functions:

1. **Regression** (e.g., predicting tomorrow's temperature)
2. **Classification** (e.g., image recognition)

Training via **supervised learning**. Given a labeled dataset with N pairs of inputs and desired outputs $\left\{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right\}_{i=1}^{N}$:

Neural Network Training

Goal

Tune the weights and biases so that the neural network implements a desired function.

Two common classes of functions:

1. **Regression** (e.g., predicting tomorrow's temperature)
2. **Classification** (e.g., image recognition)

Training via **supervised learning**. Given a labeled dataset with N pairs of inputs and desired outputs $\left\{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right\}_{i=1}^{N}$:

1. Compute the score function (**forward pass**).

Neural Network Training

Goal

Tune the weights and biases so that the neural network implements a desired function.

Two common classes of functions:

1. **Regression** (e.g., predicting tomorrow's temperature)
2. **Classification** (e.g., image recognition)

Training via **supervised learning**. Given a labeled dataset with N pairs of inputs and desired outputs $\left\{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right\}_{i=1}^{N}$:

1. Compute the score function (**forward pass**).
2. Evaluate the loss function using the score function output and labels.

Neural Network Training

Goal

Tune the weights and biases so that the neural network implements a desired function.

Two common classes of functions:

1. **Regression** (e.g., predicting tomorrow's temperature)
2. **Classification** (e.g., image recognition)

Training via **supervised learning**. Given a labeled dataset with N pairs of inputs and desired outputs $\left\{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right\}_{i=1}^{N}$:

1. Compute the score function (**forward pass**).
2. Evaluate the loss function using the score function output and labels.
3. Update neural network weights/biases to reduce the loss function (**backward pass**).

Neural Network Training

Goal

Tune the weights and biases so that the neural network implements a desired function.

Two common classes of functions:

1. **Regression** (e.g., predicting tomorrow's temperature)
2. **Classification** (e.g., image recognition)

Training via **supervised learning**. Given a labeled dataset with N pairs of inputs and desired outputs $\left\{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right\}_{i=1}^{N}$:

1. Compute the score function (**forward pass**).
2. Evaluate the loss function using the score function output and labels.
3. Update neural network weights/biases to reduce the loss function (**backward pass**).
4. Repeat.

Loss Functions

Intuitive definition

A loss function is a measure of “wrongness”: it represents the “cost” of getting the neural network output $\hat{\mathbf{y}}$ when the desired output was $\mathbf{y}$. We denote it by $L(\mathbf{y}, \hat{\mathbf{y}})$.

Loss Functions

Intuitive definition

A loss function is a measure of "wrongness": it represents the "cost" of getting the neural network output $\hat{\mathbf{y}}$ when the desired output was $\mathbf{y}$. We denote it by $L(\mathbf{y}, \hat{\mathbf{y}})$.

- The choice of loss function **depends on the application**.

Loss Functions

Intuitive definition

A loss function is a measure of "wrongness": it represents the "cost" of getting the neural network output $\hat{\mathbf{y}}$ when the desired output was $\mathbf{y}$. We denote it by $L(\mathbf{y}, \hat{\mathbf{y}})$.

- The choice of loss function **depends on the application**. Typical choices:
  - **Regression:** squared error

$$L(\mathbf{y}, \hat{\mathbf{y}}) = |\mathbf{y} - \hat{\mathbf{y}}|_2^2$$

Loss Functions

Intuitive definition

A loss function is a measure of "wrongness": it represents the "cost" of getting the neural network output $\hat{\mathbf{y}}$ when the desired output was $\mathbf{y}$. We denote it by $L(\mathbf{y}, \hat{\mathbf{y}})$.

- The choice of loss function **depends on the application**. Typical choices:
  - **Regression:** squared error

$$L(\mathbf{y}, \hat{\mathbf{y}}) = |\mathbf{y} - \hat{\mathbf{y}}|_2^2 = \sum_{i=1}^{M} (y_i - \hat{y}_i)^2$$

Loss Functions

Intuitive definition

A loss function is a measure of "wrongness": it represents the "cost" of getting the neural network output $\hat{\mathbf{y}}$ when the desired output was $\mathbf{y}$. We denote it by $L(\mathbf{y}, \hat{\mathbf{y}})$.

- The choice of loss function **depends on the application**. Typical choices:
  - **Regression:** squared error

$$L(\mathbf{y}, \hat{\mathbf{y}}) = |\mathbf{y} - \hat{\mathbf{y}}|_2^2 = \sum_{i=1}^{M} (y_i - \hat{y}_i)^2$$

  - **Classification:** binary cross-entropy

$$L(\mathbf{y}, \hat{\mathbf{y}}) = -\mathbf{y}^\top \log(\hat{\mathbf{y}}) - (1 - \mathbf{y})^\top \log(1 - \hat{\mathbf{y}})$$

Loss Functions

Intuitive definition

A loss function is a measure of "wrongness": it represents the "cost" of getting the neural network output $\hat{\mathbf{y}}$ when the desired output was $\mathbf{y}$. We denote it by $L(\mathbf{y}, \hat{\mathbf{y}})$.

- The choice of loss function **depends on the application**. Typical choices:
  - **Regression:** squared error

$$L(\mathbf{y}, \hat{\mathbf{y}}) = |\mathbf{y} - \hat{\mathbf{y}}|_2^2 = \sum_{i=1}^{M} (y_i - \hat{y}_i)^2$$

  - **Classification:** binary cross-entropy

$$L(\mathbf{y}, \hat{\mathbf{y}}) = -\mathbf{y}^\top \log(\hat{\mathbf{y}}) - (1 - \mathbf{y})^\top \log(1 - \hat{\mathbf{y}}) = -\sum_{i=1}^{M} \left(y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \right)$$

Neural Network Training

- Training solves the following optimization problem:

$$\min_{\mathbf{w},\mathbf{b}} \sum_{i=1}^{N} L(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)})$$

Neural Network Training

- Training solves the following optimization problem:

$$\min_{\mathbf{w},\mathbf{b}} \sum_{i=1}^{N} L(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)}) = \min_{\mathbf{w},\mathbf{b}} \sum_{i=1}^{N} L\left(\mathbf{y}^{(i)}, f\left(\mathbf{x}^{(i)}, \mathbf{w}, \mathbf{b}\right)\right)$$

Neural Network Training

- Training solves the following optimization problem:

$$\min_{\mathbf{w},\mathbf{b}} \sum_{i=1}^{N} L(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)}) = \min_{\mathbf{w},\mathbf{b}} \sum_{i=1}^{N} L\left(\mathbf{y}^{(i)}, f\left(\mathbf{x}^{(i)}, \mathbf{w}, \mathbf{b}\right)\right)$$

How?

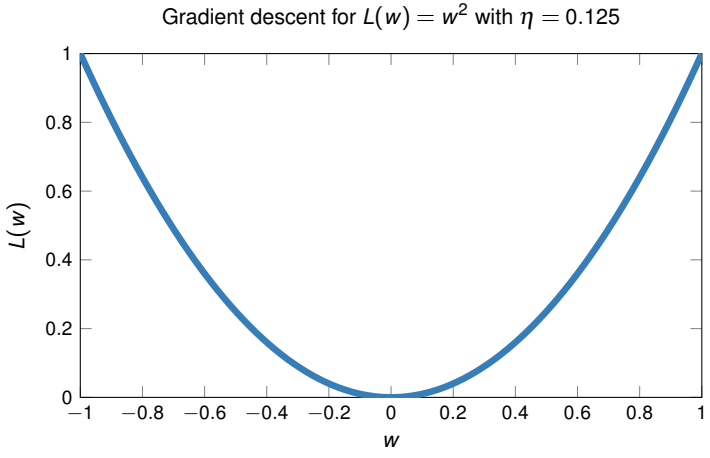
We use an iterative minimum finding technique called **gradient descent**:

$$\mathbf{w}_k = \mathbf{w}_{k-1} - \eta \cdot \frac{\partial L}{\partial \mathbf{w}_{k-1}},$$

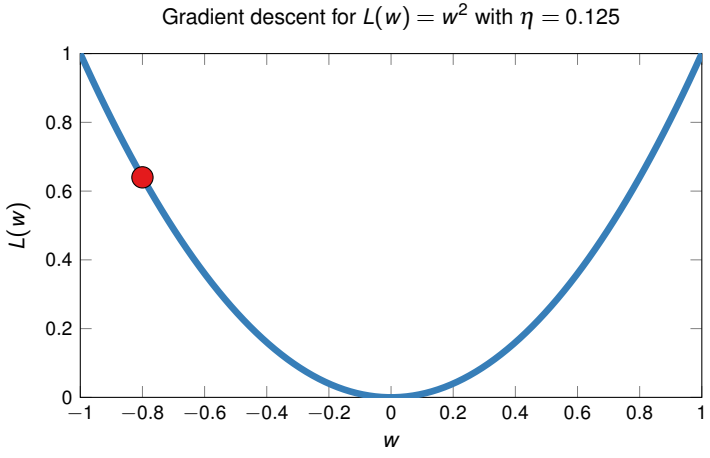
$$\mathbf{b}_k = \mathbf{b}_{k-1} - \eta \cdot \frac{\partial L}{\partial \mathbf{b}_{k-1}},$$

where $k = 1, \ldots$ is the iteration and η is called the **learning rate**.

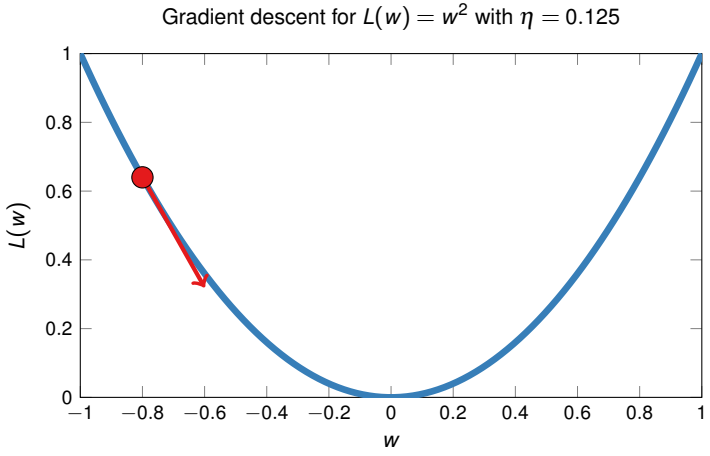
Gradient Descent



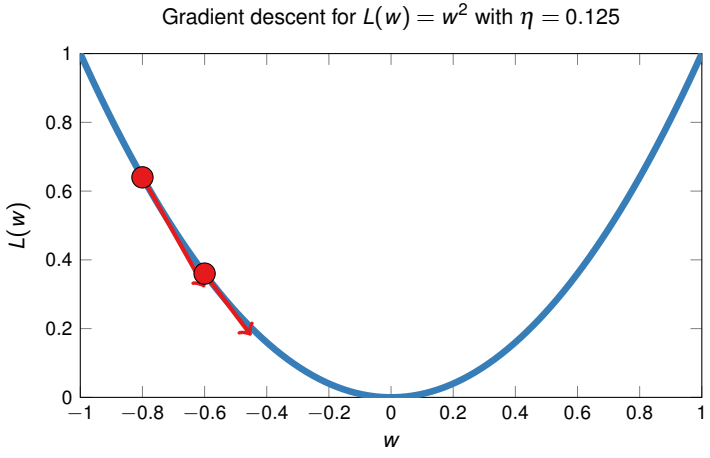
Gradient Descent



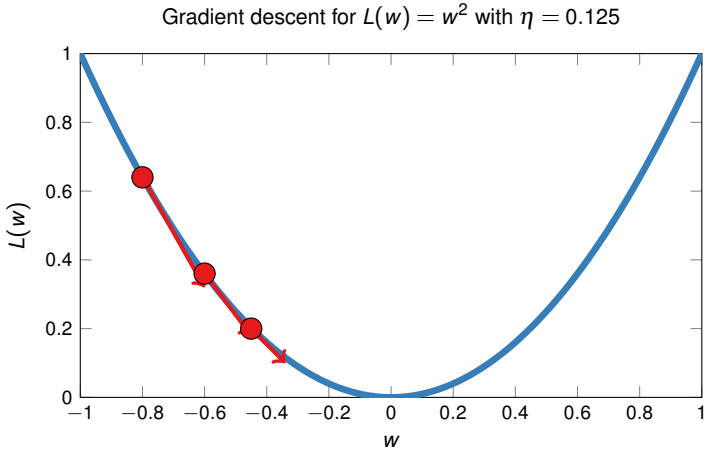
Gradient Descent



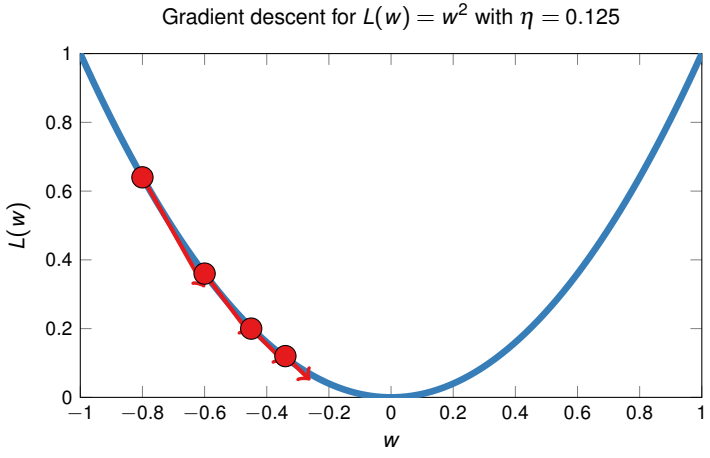
Gradient Descent



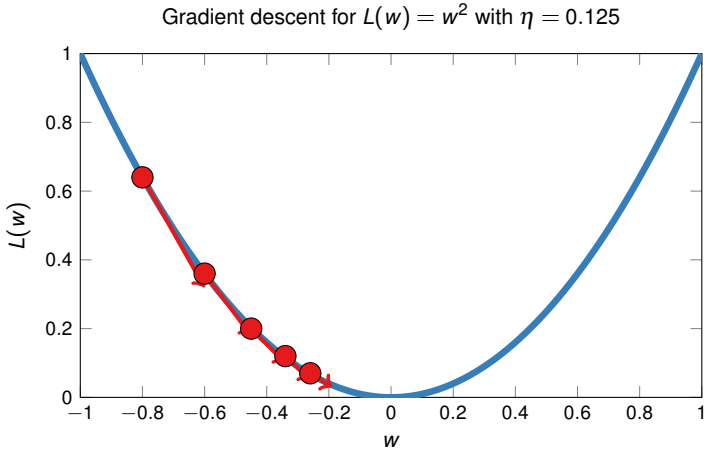
Gradient Descent



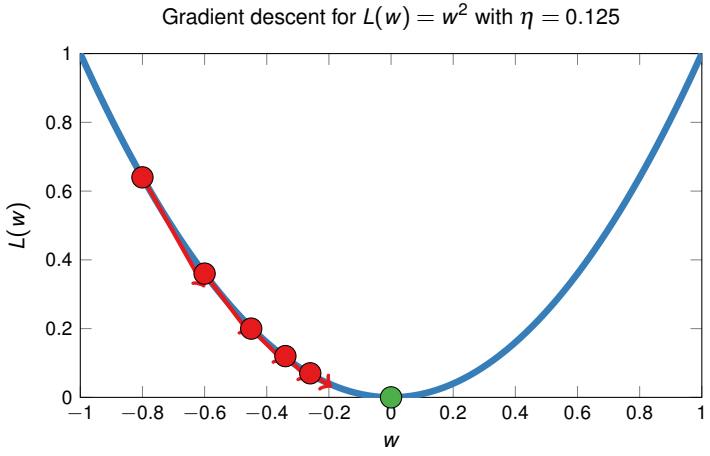
Gradient Descent



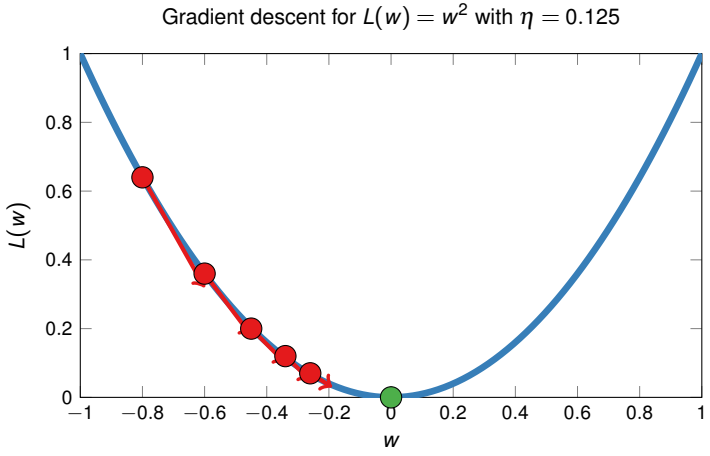
Gradient Descent



Gradient Descent



Gradient Descent



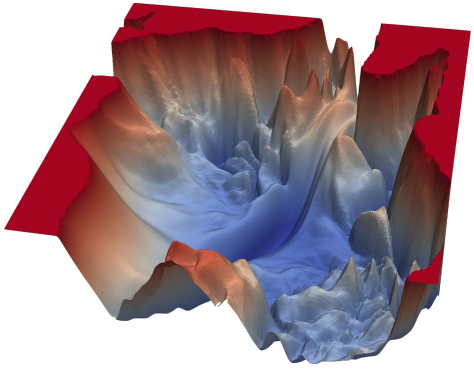
Challenge: a good choice of the learning rate η is crucial!

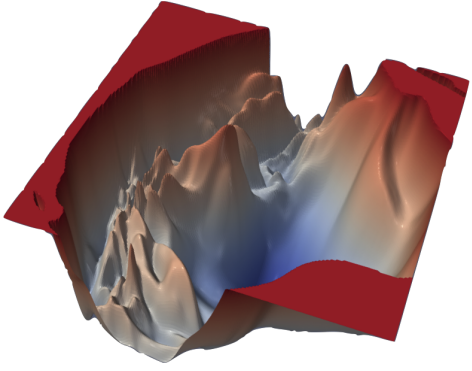
Why Gradient Descent?

- For $L(w) = w^2$, we could have found the minimum analytically: $\frac{\partial L(w)}{\partial w} = 0 \Leftrightarrow w = 0$

Why Gradient Descent?

- For $L(w) = w^2$, we could have found the minimum analytically: $\frac{\partial L(w)}{\partial w} = 0 \Leftrightarrow w = 0$
- In reality, the loss functions look more like this [1]:





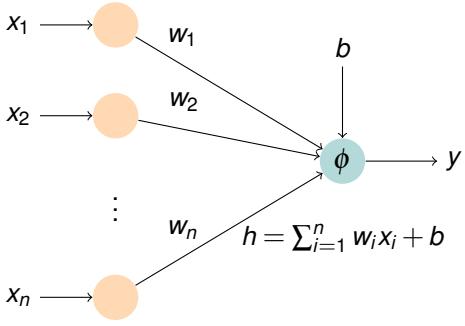
[1] H. Li, Z. Xu, G. Taylor, C. Studer, T. Goldstein, "Visualizing the loss landscape of neural nets," 2018.

Backpropagation

Backpropagation is an efficient way to calculate $\frac{\partial L}{\partial \mathbf{w}}$ and $\frac{\partial L}{\partial \mathbf{b}}$ required for gradient descent.

Backpropagation

Backpropagation is an efficient way to calculate $\frac{\partial L}{\partial \mathbf{w}}$ and $\frac{\partial L}{\partial \mathbf{b}}$ required for gradient descent.

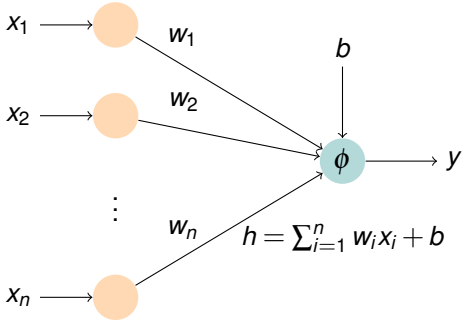


Let's simplify

- Single-neuron neural network.
- Squared error loss function.
- Training set with $N = 1$ samples.

Backpropagation

Backpropagation is an efficient way to calculate $\frac{\partial L}{\partial \mathbf{w}}$ and $\frac{\partial L}{\partial \mathbf{b}}$ required for gradient descent.



Let's simplify

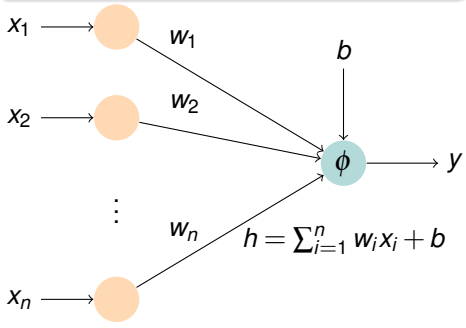
- Single-neuron neural network.
- Squared error loss function.
- Training set with $N = 1$ samples.

Loss function: $L(y, \hat{y}) = (y - \hat{y})^2$

Backpropagation

Chain Rule

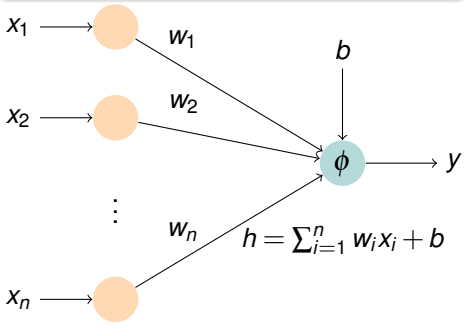
Use helper variable z: $\frac{\partial x}{\partial y} = \frac{\partial x}{\partial z} \cdot \frac{\partial z}{\partial y}$



Backpropagation

Chain Rule

Use helper variable z: $\frac{\partial x}{\partial y} = \frac{\partial x}{\partial z} \cdot \frac{\partial z}{\partial y}$



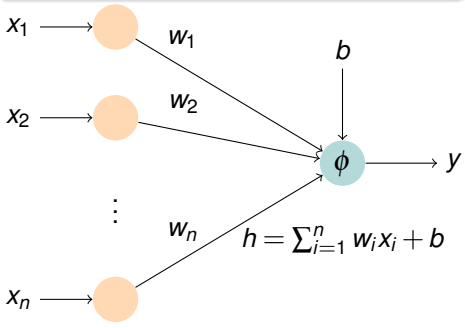
Let's calculate $\frac{\partial L}{\partial w_j}$:

$$\frac{\partial L}{\partial w_j} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial w_j}$$

Backpropagation

Chain Rule

Use helper variable z: $\frac{\partial x}{\partial y} = \frac{\partial x}{\partial z} \cdot \frac{\partial z}{\partial y}$



Let's calculate $\frac{\partial L}{\partial w_j}$:

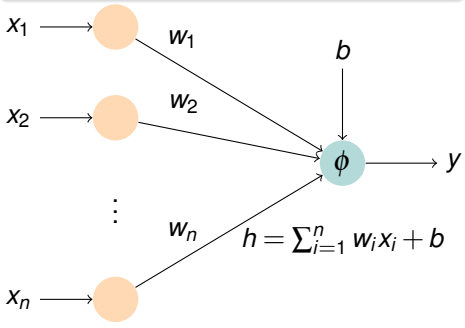
$$\frac{\partial L}{\partial w_j} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial w_j}$$

$$= \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial h} \cdot \frac{\partial h}{\partial w_j}$$

Backpropagation

Chain Rule

Use helper variable z: $\frac{\partial x}{\partial y} = \frac{\partial x}{\partial z} \cdot \frac{\partial z}{\partial y}$



Let's calculate $\frac{\partial L}{\partial w_j}$:

$$\frac{\partial L}{\partial w_j} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial w_j}$$
$$= \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial h} \cdot \frac{\partial h}{\partial w_j}$$

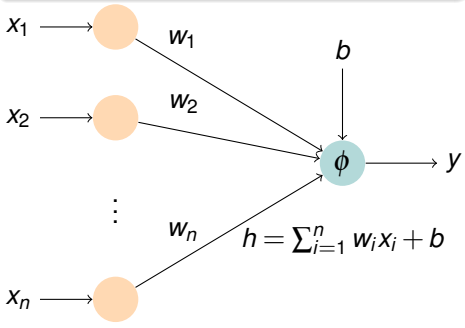
Now, we have:

$$\frac{\partial L}{\partial y} = \frac{\partial (y - \hat{y})^2}{\partial y} = 2(y - \hat{y})$$

Backpropagation

Chain Rule

Use helper variable z: $\frac{\partial x}{\partial y} = \frac{\partial x}{\partial z} \cdot \frac{\partial z}{\partial y}$



Let's calculate $\frac{\partial L}{\partial w_j}$:

$$\frac{\partial L}{\partial w_j} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial w_j}$$
$$= \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial h} \cdot \frac{\partial h}{\partial w_j}$$

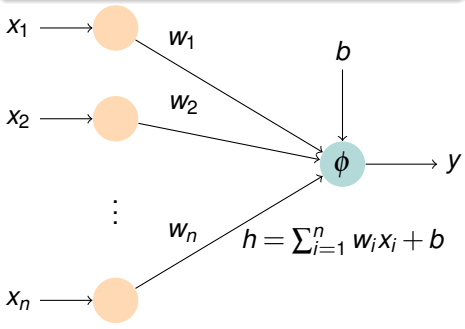
Now, we have:

$$\frac{\partial L}{\partial y} = \frac{\partial (y - \hat{y})^2}{\partial y} = 2(y - \hat{y})$$
$$\frac{\partial y}{\partial h} = \frac{\partial \phi(h)}{\partial h} = \phi'(h)$$

Backpropagation

Chain Rule

Use helper variable z: $\frac{\partial x}{\partial y} = \frac{\partial x}{\partial z} \cdot \frac{\partial z}{\partial y}$



Let's calculate $\frac{\partial L}{\partial w_j}$:

$$\frac{\partial L}{\partial w_j} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial w_j}$$
$$= \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial h} \cdot \frac{\partial h}{\partial w_j}$$

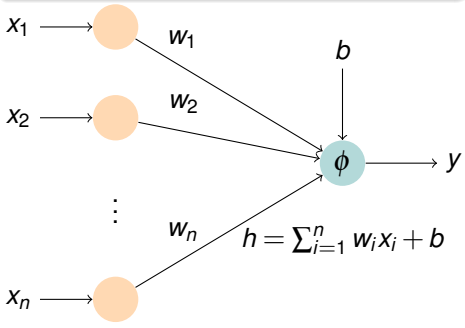
Now, we have:

$$\frac{\partial L}{\partial y} = \frac{\partial (y - \hat{y})^2}{\partial y} = 2(y - \hat{y})$$
$$\frac{\partial y}{\partial h} = \frac{\partial \phi(h)}{\partial h} = \phi'(h)$$
$$\frac{\partial h}{\partial w_j} = \frac{\partial \sum_{i=1}^{n} w_i x_i + b}{\partial w_j} = x_j$$

Backpropagation

Chain Rule

Use helper variable z: $\frac{\partial x}{\partial y} = \frac{\partial x}{\partial z} \cdot \frac{\partial z}{\partial y}$



Let's calculate $\frac{\partial L}{\partial w_j}$:

$$\frac{\partial L}{\partial w_j} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial w_j}$$
$$= \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial h} \cdot \frac{\partial h}{\partial w_j}$$

Now, we have:

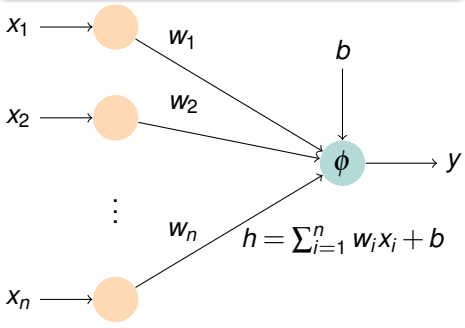
$$\frac{\partial L}{\partial y} = \frac{\partial (y - \hat{y})^2}{\partial y} = 2(y - \hat{y})$$
$$\frac{\partial y}{\partial h} = \frac{\partial \phi(h)}{\partial h} = \phi'(h)$$
$$\frac{\partial h}{\partial w_j} = \frac{\partial \sum_{i=1}^{n} w_i x_i + b}{\partial w_j} = x_j$$

Finally: $\frac{\partial L}{\partial w_j} = 2(y - \hat{y}) \cdot \phi'(h) \cdot x_j$

Backpropagation

Chain Rule

Use helper variable z: $\frac{\partial x}{\partial y} = \frac{\partial x}{\partial z} \cdot \frac{\partial z}{\partial y}$



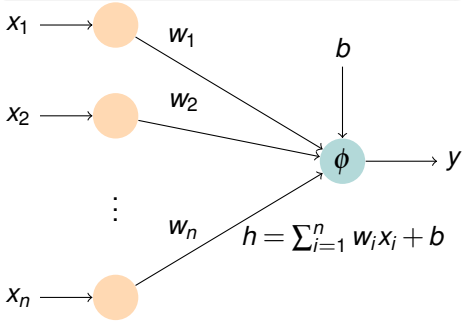
For $\frac{\partial L}{\partial b}$, one change:

$$\frac{\partial h}{\partial b} = \frac{\partial \sum_{i=1}^{n} w_i x_i + b}{\partial b} = 1$$

Backpropagation

Chain Rule

Use helper variable z: $\frac{\partial x}{\partial y} = \frac{\partial x}{\partial z} \cdot \frac{\partial z}{\partial y}$



For $\frac{\partial L}{\partial b}$, one change:

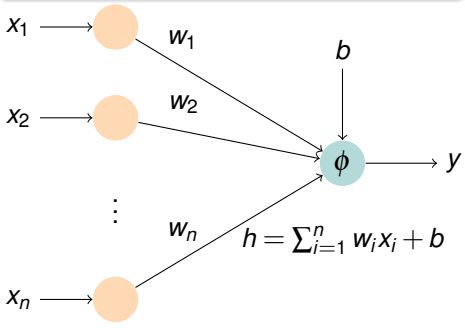
$$\frac{\partial h}{\partial b} = \frac{\partial \sum_{i=1}^{n} w_i x_i + b}{\partial b} = 1$$

Finally: $\frac{\partial L}{\partial b} = 2(y - \hat{y}) \cdot \phi'(h)$

Backpropagation

Chain Rule

Use helper variable z: $\frac{\partial x}{\partial y} = \frac{\partial x}{\partial z} \cdot \frac{\partial z}{\partial y}$



For $\frac{\partial L}{\partial b}$, one change:

$$\frac{\partial h}{\partial b} = \frac{\partial \sum_{i=1}^{n} w_i x_i + b}{\partial b} = 1$$

Finally: $\frac{\partial L}{\partial b} = 2(y - \hat{y}) \cdot \phi'(h)$

Activation function derivatives

- **tanh**: $\phi'(h) = 1 - \tanh^2(h)$
- **sigmoid**: $\phi'(h) = \sigma(h)(1 - \sigma(h))$
- **ReLU**: $\phi'(h) = \begin{cases} 0, & h < 0, \\ 1, & h > 0. \end{cases}$

Putting It All Together

At each training step k:

1. Do a forward pass using $\mathbf{w}_{k-1}$ and $\mathbf{b}_{k-1}$ and calculate loss function.

Putting It All Together

At each training step k:

1. Do a forward pass using $\mathbf{w}_{k-1}$ and $\mathbf{b}_{k-1}$ and calculate loss function.
2. Calculate $\frac{\partial L}{\partial \mathbf{w}_{k-1}}$ and $\frac{\partial L}{\partial \mathbf{b}_{k-1}}$ using backpropagation.

Putting It All Together

At each training step k:

1. Do a forward pass using $\mathbf{w}_{k-1}$ and $\mathbf{b}_{k-1}$ and calculate loss function.
2. Calculate $\frac{\partial L}{\partial \mathbf{w}_{k-1}}$ and $\frac{\partial L}{\partial \mathbf{b}_{k-1}}$ using backpropagation.
3. Take gradient descent step: $\mathbf{w}_k = \mathbf{w}_{k-1} - \eta \cdot \frac{\partial L}{\partial \mathbf{w}_{k-1}}$ and $\mathbf{b}_k = \mathbf{b}_{k-1} - \eta \cdot \frac{\partial L}{\partial \mathbf{b}_{k-1}}$.

Putting It All Together

At each training step k:

1. Do a forward pass using $\mathbf{w}_{k-1}$ and $\mathbf{b}_{k-1}$ and calculate loss function.
2. Calculate $\frac{\partial L}{\partial \mathbf{w}_{k-1}}$ and $\frac{\partial L}{\partial \mathbf{b}_{k-1}}$ using backpropagation.
3. Take gradient descent step: $\mathbf{w}_k = \mathbf{w}_{k-1} - \eta \cdot \frac{\partial L}{\partial \mathbf{w}_{k-1}}$ and $\mathbf{b}_k = \mathbf{b}_{k-1} - \eta \cdot \frac{\partial L}{\partial \mathbf{b}_{k-1}}$.

A Note on Gradient Descent

There are three flavors of gradient descent (GD):

1. **Stochastic GD**: updates weights and biases after every sample (our example).

Putting It All Together

At each training step k:

1. Do a forward pass using $\mathbf{w}_{k-1}$ and $\mathbf{b}_{k-1}$ and calculate loss function.
2. Calculate $\frac{\partial L}{\partial \mathbf{w}_{k-1}}$ and $\frac{\partial L}{\partial \mathbf{b}_{k-1}}$ using backpropagation.
3. Take gradient descent step: $\mathbf{w}_k = \mathbf{w}_{k-1} - \eta \cdot \frac{\partial L}{\partial \mathbf{w}_{k-1}}$ and $\mathbf{b}_k = \mathbf{b}_{k-1} - \eta \cdot \frac{\partial L}{\partial \mathbf{b}_{k-1}}$.

A Note on Gradient Descent

There are three flavors of gradient descent (GD):

1. **Stochastic GD**: updates weights and biases after every sample (our example).
2. **Mini-batch GD**: updates weights and biases after every mini-batch of $B \ll N$ samples.

Putting It All Together

At each training step k:

1. Do a forward pass using $\mathbf{w}_{k-1}$ and $\mathbf{b}_{k-1}$ and calculate loss function.
2. Calculate $\frac{\partial L}{\partial \mathbf{w}_{k-1}}$ and $\frac{\partial L}{\partial \mathbf{b}_{k-1}}$ using backpropagation.
3. Take gradient descent step: $\mathbf{w}_k = \mathbf{w}_{k-1} - \eta \cdot \frac{\partial L}{\partial \mathbf{w}_{k-1}}$ and $\mathbf{b}_k = \mathbf{b}_{k-1} - \eta \cdot \frac{\partial L}{\partial \mathbf{b}_{k-1}}$.

A Note on Gradient Descent

There are three flavors of gradient descent (GD):

1. **Stochastic GD**: updates weights and biases after every sample (our example).
2. **Mini-batch GD**: updates weights and biases after every mini-batch of $B \ll N$ samples.
3. **Batch GD**: updates weights and biases after every batch of N samples.

Putting It All Together

At each training step k:

1. Do a forward pass using $\mathbf{w}_{k-1}$ and $\mathbf{b}_{k-1}$ and calculate loss function.
2. Calculate $\frac{\partial L}{\partial \mathbf{w}_{k-1}}$ and $\frac{\partial L}{\partial \mathbf{b}_{k-1}}$ using backpropagation.
3. Take gradient descent step: $\mathbf{w}_k = \mathbf{w}_{k-1} - \eta \cdot \frac{\partial L}{\partial \mathbf{w}_{k-1}}$ and $\mathbf{b}_k = \mathbf{b}_{k-1} - \eta \cdot \frac{\partial L}{\partial \mathbf{b}_{k-1}}$.

A Note on Gradient Descent

There are three flavors of gradient descent (GD):

1. **Stochastic GD**: updates weights and biases after every sample (our example).
2. **Mini-batch GD**: updates weights and biases after every mini-batch of $B \ll N$ samples.
3. **Batch GD**: updates weights and biases after every batch of N samples.

One pass through the entire N-sample dataset is called an **epoch**.

What's Missing?

What do we need to generalize backpropagation to **any neural network**?

What's Missing?

What do we need to generalize backpropagation to **any neural network**?

1. More neurons per layer

What's Missing?

What do we need to generalize backpropagation to **any neural network**?

1. More neurons per layer
2. More hidden layers

What's Missing?

What do we need to generalize backpropagation to **any neural network**?

1. More neurons per layer
2. More hidden layers
3. More training samples

What's Missing?

What do we need to generalize backpropagation to **any neural network**?

1. More neurons per layer
2. More hidden layers
3. More training samples
4. Weight and bias initialization ($\mathbf{w}_0$ and $\mathbf{b}_0$)

What's Missing?

What do we need to generalize backpropagation to **any neural network**?

1. More neurons per layer $\rightarrow$ **Assignment 1**
2. More hidden layers $\rightarrow$ **Assignment 1**
3. More training samples $\rightarrow$ **Assignment 1**
4. Weight and bias initialization ($\mathbf{w}_0$ and $\mathbf{b}_0$)

What's Missing?

What do we need to generalize backpropagation to **any neural network**?

1. More neurons per layer $\rightarrow$ **Assignment 1**
2. More hidden layers $\rightarrow$ **Assignment 1**
3. More training samples $\rightarrow$ **Assignment 1**
4. Weight and bias initialization ($\mathbf{w}_0$ and $\mathbf{b}_0$) $\rightarrow$ **After the break!**

Time For A Break

> "*An ounce of practice is generally worth more than a ton of theory.*"
>
> – Ernst F. Schumacher, "Small Is Beautiful: A Study of Economics As If People Mattered," 1973.

Just Initialize to a Constant?

Question

Can we simply initialize all weights to some constant (e.g., zero)?

Just Initialize to a Constant?

Question

Can we simply initialize all weights to some constant (e.g., zero)? **Not a good idea!**

Just Initialize to a Constant?

Question

Can we simply initialize all weights to some constant (e.g., zero)? **Not a good idea!**

- Let's assume that $w_i = c \in \mathbb{R}$:

Just Initialize to a Constant?

Question

Can we simply initialize all weights to some constant (e.g., zero)? **Not a good idea!**

- Let's assume that $w_i = c \in \mathbb{R}$:
  1. Then, all neuron outputs in a layer are the same.

Just Initialize to a Constant?

Question

Can we simply initialize all weights to some constant (e.g., zero)? **Not a good idea!**

- Let's assume that $w_i = c \in \mathbb{R}$:
  1. Then, all neuron outputs in a layer are the same.
  2. Each row of the weight matrix **W** for each layer will be updated in the same way.

Just Initialize to a Constant?

Question

Can we simply initialize all weights to some constant (e.g., zero)? **Not a good idea!**

- Let's assume that $w_i = c \in \mathbb{R}$:
  1. Then, all neuron outputs in a layer are the same.
  2. Each row of the weight matrix **W** for each layer will be updated in the same way.
  3. **Only one effective neuron per layer!**

Just Initialize to a Constant?

Question

Can we simply initialize all weights to some constant (e.g., zero)? **Not a good idea!**

- Let's assume that $w_i = c \in \mathbb{R}$:
  1. Then, all neuron outputs in a layer are the same.
  2. Each row of the weight matrix **W** for each layer will be updated in the same way.
  3. **Only one effective neuron per layer!**

- **Problem:** Neuron symmetry.

Just Initialize to a Constant?

Question

Can we simply initialize all weights to some constant (e.g., zero)? **Not a good idea!**

- Let's assume that $w_i = c \in \mathbb{R}$:
  1. Then, all neuron outputs in a layer are the same.
  2. Each row of the weight matrix **W** for each layer will be updated in the same way.
  3. **Only one effective neuron per layer!**

- **Problem:** Neuron symmetry.
- **Solution:** Random initialization.

Random Weight Initialization

- Two common random initializations:

Random Weight Initialization

- Two common random initializations:

Xavier Initialization [1]

Random weights according to:

$$w \sim \mathcal{N}\left(0, \frac{6}{n_{\text{in}} + n_{\text{out}}}\right),$$

where n_{in} and n_{out} are the number of incoming and outgoing weights.

- Works best for **sigmoid**/**tanh** activation functions.

[1] X. Glorot and Y. Bengio, "Understanding the Difficulty of Training Deep Feedforward Neural Networks," 2010.

Random Weight Initialization

- Two common random initializations:

Xavier Initialization [1]

Random weights according to:

$$w \sim \mathcal{N}\left(0, \frac{6}{n_{\text{in}} + n_{\text{out}}}\right),$$

where n_{in} and n_{out} are the number of incoming and outgoing weights.

- Works best for **sigmoid/tanh** activation functions.

He Initialization [2]

Random weights according to:

$$w \sim \mathcal{N}\left(0, \frac{2}{n_{\text{in}}}\right),$$

where n_{in} is the number of incoming weights.

- Works best for **ReLU-type** activations.

[1] X. Glorot and Y. Bengio, "Understanding the Difficulty of Training Deep Feedforward Neural Networks," 2010.
[2] K. He, X. Zhang, S. Ren, J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," 2015.

Improved Stochastic Gradient Descent Methods

Stochastic Gradient Descent (SGD)

$$w_k = w_{k-1} - \eta \cdot \frac{\partial L}{\partial w_{k-1}}$$

Improved Stochastic Gradient Descent Methods

Stochastic Gradient Descent (SGD)

$$w_k = w_{k-1} - \eta \cdot \frac{\partial L}{\partial w_{k-1}}$$

Disadvantages:

- Gets stuck in local minima

Improved Stochastic Gradient Descent Methods

Stochastic Gradient Descent (SGD)

$$w_k = w_{k-1} - \eta \cdot \frac{\partial L}{\partial w_{k-1}}$$

Disadvantages:

- Gets stuck in local minima
- Converges slowly

Improved Stochastic Gradient Descent Methods

Stochastic Gradient Descent (SGD)

$$w_k = w_{k-1} - \eta \cdot \frac{\partial L}{\partial w_{k-1}}$$

Disadvantages:

- Gets stuck in local minima
- Converges slowly

SGD With Momentum

$$v_k = \alpha v_{k-1} - \frac{\partial L}{\partial w_{k-1}}$$

$$w_k = w_{k-1} + \eta \cdot v_k$$

Intuition: gradient update has velocity.

Improved Stochastic Gradient Descent Methods

Stochastic Gradient Descent (SGD)

$$w_k = w_{k-1} - \eta \cdot \frac{\partial L}{\partial w_{k-1}}$$

Disadvantages:

- Gets stuck in local minima
- Converges slowly

SGD With Momentum

$$v_k = \alpha v_{k-1} - \frac{\partial L}{\partial w_{k-1}}$$

$$w_k = w_{k-1} + \eta \cdot v_k$$

Intuition: gradient update has velocity.

Advantage:

- Can escape some local minima

Improved Stochastic Gradient Descent Methods

Stochastic Gradient Descent (SGD)

$$w_k = w_{k-1} - \eta \cdot \frac{\partial L}{\partial w_{k-1}}$$

Disadvantages:

- Gets stuck in local minima
- Converges slowly

SGD With Momentum

$$v_k = \alpha v_{k-1} - \frac{\partial L}{\partial w_{k-1}}$$

$$w_k = w_{k-1} + \eta \cdot v_k$$

Intuition: gradient update has velocity.

Advantage:

- Can escape some local minima

Disadvantage:

- Spirals to solution

Improved Stochastic Gradient Descent Methods (cont'd)

RMSProp

$$g_k = \gamma g_{k-1} + (1-\gamma)\left(\frac{\partial L}{\partial w_{k-1}}\right)^2$$

$$w_k = w_{k-1} - \frac{\eta}{\sqrt{g_k}} \cdot \frac{\partial L}{\partial w_{k-1}}$$

Intuition: each weight has a different and adaptive learning rate.

Improved Stochastic Gradient Descent Methods (cont'd)

RMSProp

$$g_k = \gamma g_{k-1} + (1-\gamma)\left(\frac{\partial L}{\partial w_{k-1}}\right)^2$$

$$w_k = w_{k-1} - \frac{\eta}{\sqrt{g_k}} \cdot \frac{\partial L}{\partial w_{k-1}}$$

Intuition: each weight has a different and adaptive learning rate.

Advantage:

- Reduces spiralling

Improved Stochastic Gradient Descent Methods (cont'd)

RMSProp

$$g_k = \gamma g_{k-1} + (1-\gamma)\left(\frac{\partial L}{\partial w_{k-1}}\right)^2$$

$$w_k = w_{k-1} - \frac{\eta}{\sqrt{g_k}} \cdot \frac{\partial L}{\partial w_{k-1}}$$

Intuition: each weight has a different and adaptive learning rate.

Advantage:

- Reduces spiralling

Disadvantage:

- Can still get stuck

Improved Stochastic Gradient Descent Methods (cont'd)

RMSProp

$$g_k = \gamma g_{k-1} + (1-\gamma)\left(\frac{\partial L}{\partial w_{k-1}}\right)^2$$

$$w_k = w_{k-1} - \frac{\eta}{\sqrt{g_k}} \cdot \frac{\partial L}{\partial w_{k-1}}$$

Intuition: each weight has a different and adaptive learning rate.

Advantage:

- Reduces spiralling

Disadvantage:

- Can still get stuck

Adam

$$f_k = \beta_1 f_{k-1} + (1-\beta_1)\frac{\partial L}{\partial w_{k-1}}$$

$$s_k = \beta_2 s_{k-1} + (1-\beta_2)\left(\frac{\partial L}{\partial w_{k-1}}\right)^2$$

$$\hat{f}_k = \frac{f_k}{1-(b_1)^{t-1}}$$

$$\hat{s}_k = \frac{s_k}{1-(b_2)^{t-1}}$$

$$w_k = w_{k-1} - \eta \cdot \frac{\hat{f}_k}{\sqrt{\hat{s}_k} + \varepsilon}$$

Intuition: gradient and second moment averages improve convergence.

Improved Stochastic Gradient Descent Methods (cont'd)

RMSProp

$$g_k = \gamma g_{k-1} + (1-\gamma)\left(\frac{\partial L}{\partial w_{k-1}}\right)^2$$

$$w_k = w_{k-1} - \frac{\eta}{\sqrt{g_k}} \cdot \frac{\partial L}{\partial w_{k-1}}$$

Intuition: each weight has a different and adaptive learning rate.

Advantage:

- Reduces spiralling

Disadvantage:

- Can still get stuck

Adam

$$f_k = \beta_1 f_{k-1} + (1-\beta_1)\frac{\partial L}{\partial w_{k-1}}$$

$$s_k = \beta_2 s_{k-1} + (1-\beta_2)\left(\frac{\partial L}{\partial w_{k-1}}\right)^2$$

$$\hat{f}_k = \frac{f_k}{1-(b_1)^{t-1}}$$

$$\hat{s}_k = \frac{s_k}{1-(b_2)^{t-1}}$$

$$w_k = w_{k-1} - \eta \cdot \frac{\hat{f}_k}{\sqrt{\hat{s}_k} + \varepsilon}$$

Intuition: gradient and second moment averages improve convergence.

Advantage: Improves convergence

Improved Stochastic Gradient Descent Methods (cont'd)

RMSProp

$$g_k = \gamma g_{k-1} + (1-\gamma)\left(\frac{\partial L}{\partial w_{k-1}}\right)^2$$

$$w_k = w_{k-1} - \frac{\eta}{\sqrt{g_k}} \cdot \frac{\partial L}{\partial w_{k-1}}$$

Intuition: each weight has a different and adaptive learning rate.

Advantage:

- Reduces spiralling

Disadvantage:

- Can still get stuck

Adam

$$f_k = \beta_1 f_{k-1} + (1-\beta_1)\frac{\partial L}{\partial w_{k-1}}$$

$$s_k = \beta_2 s_{k-1} + (1-\beta_2)\left(\frac{\partial L}{\partial w_{k-1}}\right)^2$$

$$\hat{f}_k = \frac{f_k}{1-(b_1)^{t-1}}$$

$$\hat{s}_k = \frac{s_k}{1-(b_2)^{t-1}}$$

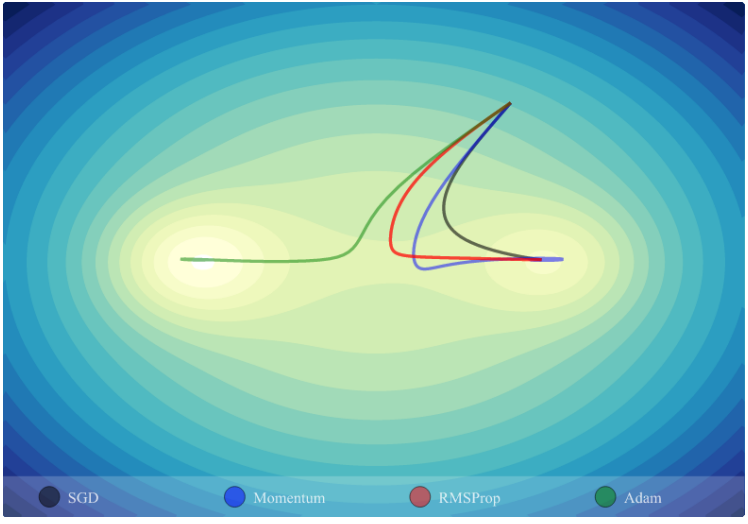
$$w_k = w_{k-1} - \eta \cdot \frac{\hat{f}_k}{\sqrt{\hat{s}_k} + \varepsilon}$$

Intuition: gradient and second moment averages improve convergence.

Advantage: Improves convergence

Disadvantage: Can still get stuck

Improved SGD Methods Visualized



Data Normalization

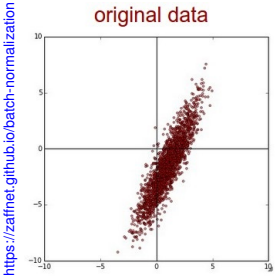
- Training data normalization improves training convergence!

Data Normalization

- Training data normalization improves training convergence!
  1. Calculate mean μ and standard deviation σ of each dimension of the training data.
  2. Normalize each dimension of each training sample as $x_{\text{norm}} = \frac{x-\mu}{\sigma}$.

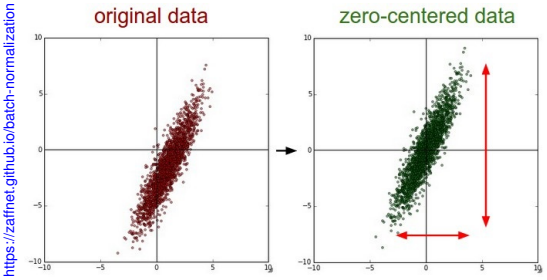
Data Normalization

- Training data normalization improves training convergence!
  1. Calculate mean μ and standard deviation σ of each dimension of the training data.
  2. Normalize each dimension of each training sample as $x_{\text{norm}} = \frac{x-\mu}{\sigma}$.



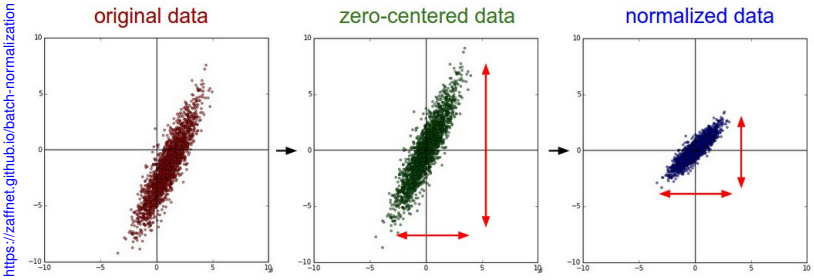
Data Normalization

- Training data normalization improves training convergence!
  1. Calculate mean μ and standard deviation σ of each dimension of the training data.
  2. Normalize each dimension of each training sample as $x_{\text{norm}} = \frac{x-\mu}{\sigma}$.



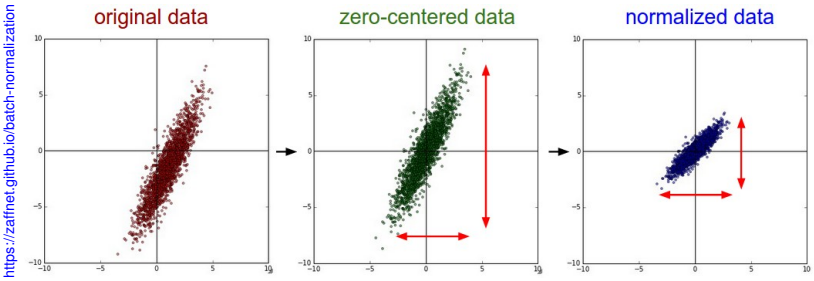
Data Normalization

- Training data normalization improves training convergence!
  1. Calculate mean μ and standard deviation σ of each dimension of the training data.
  2. Normalize each dimension of each training sample as $x_{\text{norm}} = \frac{x-\mu}{\sigma}$.



Data Normalization

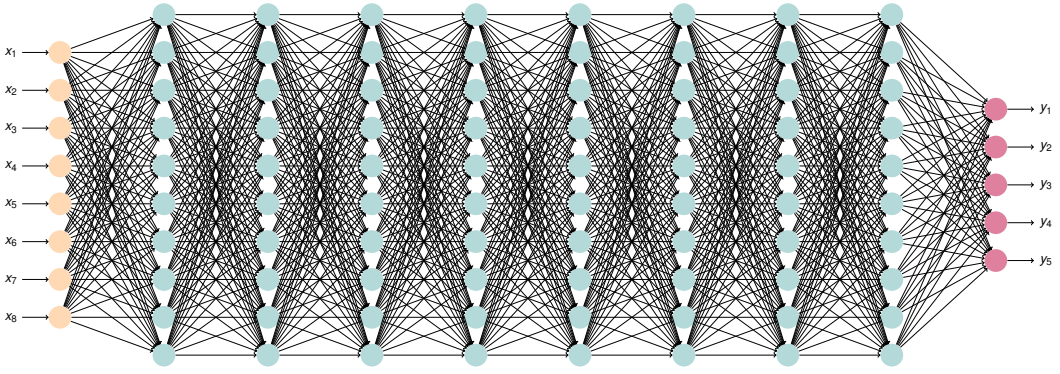
- Training data normalization improves training convergence!
  1. Calculate mean μ and standard deviation σ of each dimension of the training data.
  2. Normalize each dimension of each training sample as $x_{\text{norm}} = \frac{x-\mu}{\sigma}$.



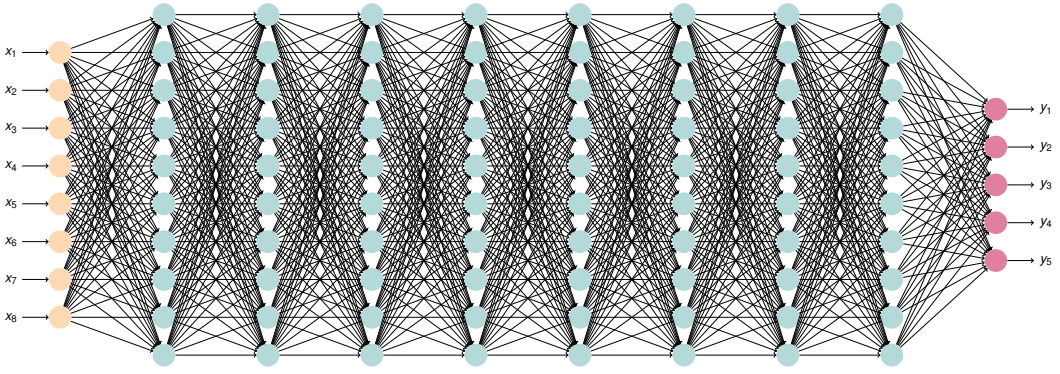
- **Batch normalization** extends the idea to internal layers [1].

[1] S. Ioffe, C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," 2015.

Deep Learning Frameworks



Deep Learning Frameworks



- Various deep learning frameworks automate the implementation and the training!

Deep Learning Framework Overview

1. **TensorFlow:**
   - Open source, developed by Google Brain mainly for deep learning
   - Both high-level and low-level APIs
   - Multi-language support

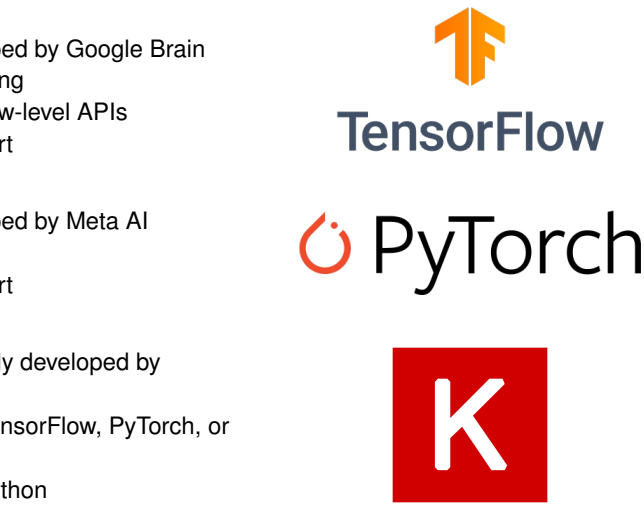
Deep Learning Framework Overview

1. **TensorFlow:**
   - Open source, developed by Google Brain mainly for deep learning
   - Both high-level and low-level APIs
   - Multi-language support
2. **PyTorch:**
   - Open source, developed by Meta AI
   - Low-level API
   - Multi-language support

TensorFlow

Deep Learning Framework Overview

1. **TensorFlow:**
   - Open source, developed by Google Brain mainly for deep learning
   - Both high-level and low-level APIs
   - Multi-language support
2. **PyTorch:**
   - Open source, developed by Meta AI
   - Low-level API
   - Multi-language support
3. **Keras:**
   - Open source, originally developed by François Chollet
   - High-level API with TensorFlow, PyTorch, or JAX backend
   - Mainly for use with Python



Conclusion

Summary:

1. Training a neural network (NN) is the process of choosing “good” weights and biases.
2. Training is done using **gradient descent** combined with **backpropagation**.
3. Weight and bias initialization is crucial, but good heuristics exist.
4. Remember to **normalize** your data!
5. Frameworks (e.g., Keras) do the heavy lifting.

Conclusion

Summary:

1. Training a neural network (NN) is the process of choosing “good” weights and biases.
2. Training is done using **gradient descent** combined with **backpropagation**.
3. Weight and bias initialization is crucial, but good heuristics exist.
4. Remember to **normalize** your data!
5. Frameworks (e.g., Keras) do the heavy lifting.

Next time:

1. Overfitting and how to overcome it.
2. Hyperparameter tuning.
3. Convolutional neural networks.
4. Introduction to optimizations.