Gradient Descent

Towards Neural Networks

Justin Stevens
Undergraduate AI Society
April 2nd, 2019

Outline

1. Decision Making
   - Perceptrons
   - Activation Functions
2. Classifying Digits through MNIST

Should I Stay or Should I Go?

Let's say I'm deciding on a given day whether or not to go to an Edmonton Oilers game. Here are the factors that will influence my decision:

Should I Stay or Should I Go?

Let's say I'm deciding on a given day whether or not to go to an Edmonton Oilers game. Here are the factors that will influence my decision:

- Are the tickets cheap or expensive?
- Do I have the time to go?
- Do I care about the team they're playing?

Should I Stay or Should I Go?

Let's say I'm deciding on a given day whether or not to go to an Edmonton Oilers game. Here are the factors that will influence my decision:

- Are the tickets cheap or expensive?
- Do I have the time to go?
- Do I care about the team they're playing?

We'll make my decision by encoding each possible input as a vector $\bar{\mathbf{x}}$:

Should I Stay or Should I Go?

Let's say I'm deciding on a given day whether or not to go to an Edmonton Oilers game. Here are the factors that will influence my decision:

- Are the tickets cheap or expensive?
- Do I have the time to go?
- Do I care about the team they're playing?

We'll make my decision by encoding each possible input as a vector $\bar{\mathbf{x}}$:

Ticket Prices	Availability	Interest	$\bar{\mathbf{x}}$
Cheap	Yes	Yes	$(1,1,1)$
Cheap	No	No	$(1,0,0)$
Cheap	Yes	No	$(1,1,0)$
Cheap	No	Yes	$(1,0,1)$
Expensive	Yes	Yes	$(0,1,1)$
Expensive	No	No	$(0,0,0)$
Expensive	No	Yes	$(0,0,1)$
Expensive	Yes	No	$(0,1,0)$

How Will I Make my Decision?

Let's say I don't care much about price, but I do care about my availability and interest. In this case, the corresponding weights might be $\bar{\mathbf{w}} = (1, 6, 3)$.

How Will I Make my Decision?

Let's say I don't care much about price, but I do care about my availability and interest. In this case, the corresponding weights might be $\bar{\mathbf{w}} = (1, 6, 3)$. We can then compute the dot product $\bar{\mathbf{w}} \cdot \bar{\mathbf{x}}$ for each possible input:

Ticket Prices	Availability	Interest	$\bar{\mathbf{x}}$	$\bar{\mathbf{w}} \cdot \bar{\mathbf{x}}$
Cheap	Yes	Yes	$(1, 1, 1)$	10
Cheap	No	No	$(1, 0, 0)$	1
Cheap	Yes	No	$(1, 1, 0)$	7
Cheap	No	Yes	$(1, 0, 1)$	4
Expensive	Yes	Yes	$(0, 1, 1)$	9
Expensive	No	No	$(0, 0, 0)$	0
Expensive	No	Yes	$(0, 0, 1)$	3
Expensive	Yes	No	$(0, 1, 0)$	6

How Will I Make my Decision?

Let's say I don't care much about price, but I do care about my availability and interest. In this case, the corresponding weights might be $\bar{\mathbf{w}} = (1, 6, 3)$. We can then compute the dot product $\bar{\mathbf{w}} \cdot \bar{\mathbf{x}}$ for each possible input:

Ticket Prices	Availability	Interest	$\bar{\mathbf{x}}$	$\bar{\mathbf{w}} \cdot \bar{\mathbf{x}}$
Cheap	Yes	Yes	$(1, 1, 1)$	10
Cheap	No	No	$(1, 0, 0)$	1
Cheap	Yes	No	$(1, 1, 0)$	7
Cheap	No	Yes	$(1, 0, 1)$	4
Expensive	Yes	Yes	$(0, 1, 1)$	9
Expensive	No	No	$(0, 0, 0)$	0
Expensive	No	Yes	$(0, 0, 1)$	3
Expensive	Yes	No	$(0, 1, 0)$	6

We can now define my **activation threshold**, t, which will determine whether or not I go to the game, represented in binary.

Formula for Decision Making

The general formula for my decision to go to the Oilers game is

$$\text{output} = \begin{cases} 0 & \text{if } \bar{\mathbf{w}} \cdot \bar{\mathbf{x}} < t \\ 1 & \text{if } \bar{\mathbf{w}} \cdot \bar{\mathbf{x}} \geq t. \end{cases}$$

Formula for Decision Making

The general formula for my decision to go to the Oilers game is

$$\text{output} = \begin{cases} 0 & \text{if } \bar{\mathbf{w}} \cdot \bar{\mathbf{x}} < t \\ 1 & \text{if } \bar{\mathbf{w}} \cdot \bar{\mathbf{x}} \geq t. \end{cases}$$

For instance, if $t = 9$, we see I'll only go if I'm both available and interested.

Formula for Decision Making

The general formula for my decision to go to the Oilers game is

$$\text{output} = \begin{cases} 0 & \text{if } \bar{\mathbf{w}} \cdot \bar{\mathbf{x}} < t \\ 1 & \text{if } \bar{\mathbf{w}} \cdot \bar{\mathbf{x}} \geq t. \end{cases}$$

For instance, if $t = 9$, we see I'll only go if I'm both available and interested. If $t = 7$, I'll also go if the tickets are cheap and I'm available:

Ticket Prices	Availability	Interest	$\bar{\mathbf{x}}$	$\bar{\mathbf{x}} \cdot \bar{\mathbf{w}}$
Cheap	**Yes**	**Yes**	$(1,1,1)$	**10**
Cheap	No	No	$(1,0,0)$	1
Cheap	**Yes**	**No**	$(1,1,0)$	**7**
Cheap	No	Yes	$(1,0,1)$	4
Expensive	**Yes**	**Yes**	$(0,1,1)$	**9**
Expensive	No	No	$(0,0,0)$	0
Expensive	No	Yes	$(0,0,1)$	3
Expensive	Yes	No	$(0,1,0)$	6

Perceptrons

This is a simplified model of a **perceptron**. The idea was developed by Frank Rosenblatt at Cornell in 1957, and is often used in psychology.

Perceptrons

This is a simplified model of a **perceptron**. The idea was developed by Frank Rosenblatt at Cornell in 1957, and is often used in psychology.

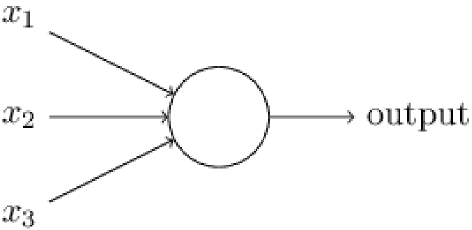

Figure 1: *Source: Nielsen*

Perceptrons

This is a simplified model of a **perceptron**. The idea was developed by Frank Rosenblatt at Cornell in 1957, and is often used in psychology.

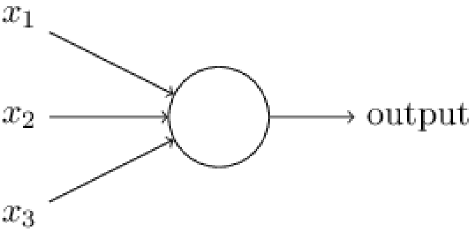

Figure 1: *Source: Nielsen*

Each of these lines collect evidence and are weighted to produce an output.

Perceptrons

This is a simplified model of a **perceptron**. The idea was developed by Frank Rosenblatt at Cornell in 1957, and is often used in psychology.

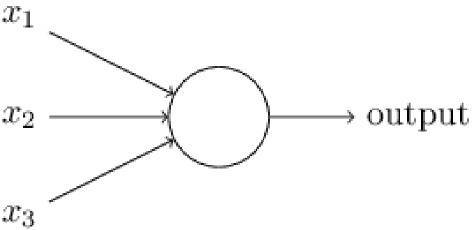

Figure 1: *Source: Nielsen*

Each of these lines collect evidence and are weighted to produce an output. In practice, our inputs and outputs don't necessarily have to be binary; they can be real-valued. We therefore have to define a new activation function.

Introducing the Bias

Instead of comparing our weighted sum to a threshold, we instead *add* a bias, b, to our weighted sum. We write this as $\bar{\mathbf{w}} \cdot \bar{\mathbf{x}} + b$ instead.

Introducing the Bias

Instead of comparing our weighted sum to a threshold, we instead *add* a bias, b, to our weighted sum. We write this as $\bar{\mathbf{w}} \cdot \bar{\mathbf{x}} + b$ instead. Then

$$\text{output} = \begin{cases} 0 & \text{if } \bar{\mathbf{w}} \cdot \bar{\mathbf{x}} + b < 0 \\ 1 & \text{if } \bar{\mathbf{w}} \cdot \bar{\mathbf{x}} + b \geq 0. \end{cases}$$

This is known as the *heaviside step function*. We'll extend our model to multiple outputs soon, but first we'll examine other activation functions.

Introducing the Bias

Instead of comparing our weighted sum to a threshold, we instead *add* a bias, b, to our weighted sum. We write this as $\bar{\mathbf{w}} \cdot \bar{\mathbf{x}} + b$ instead. Then

$$\text{output} = \begin{cases} 0 & \text{if } \bar{\mathbf{w}} \cdot \bar{\mathbf{x}} + b < 0 \\ 1 & \text{if } \bar{\mathbf{w}} \cdot \bar{\mathbf{x}} + b \geq 0. \end{cases}$$

This is known as the *heaviside step function*. We'll extend our model to multiple outputs soon, but first we'll examine other activation functions.

Rectified Linear Unit

If we want our outputs to be non-negative, we use the **rectified linear unit**,

$$f(x) = \max\{0, x\}.$$

Rectified Linear Unit

If we want our outputs to be non-negative, we use the **rectified linear unit**,

$$f(x) = \max\{0, x\}.$$

Graphically, we can see:

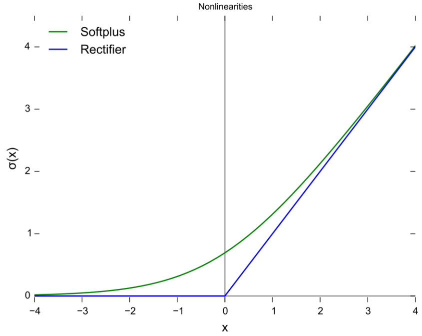

Figure 2: Rectifier, and a smooth approximation $\log(1 + e^x)$. (*Source: Wikipedia*).

Sigmoid Function

As we saw above, our output doesn't necessarily have to be a 0 or 1; using a rectified linear unit, it can be any non-negative number. However, for computational purposes, it's easiest if our outputs live in the range $(0, 1)$.

Sigmoid Function

As we saw above, our output doesn't necessarily have to be a 0 or 1; using a rectified linear unit, it can be any non-negative number. However, for computational purposes, it's easiest if our outputs live in the range $(0, 1)$. We now define the **sigmoid** or logistic function, $\sigma(z) = \frac{1}{1+e^{-z}}$.

Sigmoid Function

As we saw above, our output doesn't necessarily have to be a 0 or 1; using a rectified linear unit, it can be any non-negative number. However, for computational purposes, it's easiest if our outputs live in the range $(0, 1)$. We now define the **sigmoid** or logistic function, $\sigma(z) = \frac{1}{1+e^{-z}}$. Graphically,

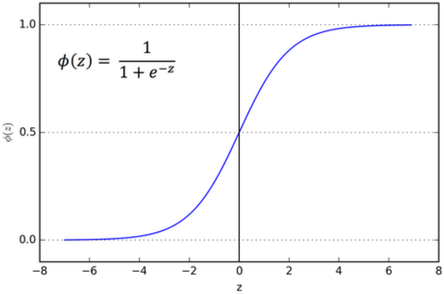

Figure 3: As $z \to \infty$, we see $\sigma(z) \to 1$. Alternatively, as $z \to -\infty$, $\sigma(z) \to 0$. (*Source: Towards Data Science*).

Outline

Example Images

In **supervised learning** problems, we're given a set of training data with labels, which we try to learn. We'll use a generalization of the perceptron with different neurons, for which we try to learn the best possible weights.

Example Images

In **supervised learning** problems, we're given a set of training data with labels, which we try to learn. We'll use a generalization of the perceptron with different neurons, for which we try to learn the best possible weights.

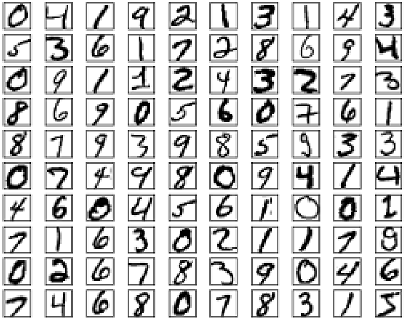

Figure 4: How would you devise a system for a **computer** to classify the digits? How can we best utilize the data set, known as MNIST? (*Source: Nielsen*)

MNIST Dataset

- The MNIST database contains seventy thousand handwritten digits.

MNIST Dataset

- The MNIST database contains seventy thousand handwritten digits.
  - Each data-point contains both an image, and the desired digit.
  - $60,000$ images are designated for training, and $10,000$ for testing:

```
import tensorflow as tf
from tensorflow import keras
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
```

MNIST Dataset

- The MNIST database contains seventy thousand handwritten digits.
  - Each data-point contains both an image, and the desired digit.
  - $60,000$ images are designated for training, and $10,000$ for testing:

```
import tensorflow as tf
from tensorflow import keras
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
```

- Each image contains pixels ranging 0 to 255, in decreasing darkness.

MNIST Dataset

- The MNIST database contains seventy thousand handwritten digits.
  - Each data-point contains both an image, and the desired digit.
  - $60,000$ images are designated for training, and $10,000$ for testing:

```
import tensorflow as tf
from tensorflow import keras
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
```

- Each image contains pixels ranging 0 to 255, in decreasing darkness.
- An individual image is a 28×28 array of pixels.

MNIST Dataset

- The MNIST database contains seventy thousand handwritten digits.
  - Each data-point contains both an image, and the desired digit.
  - $60,000$ images are designated for training, and $10,000$ for testing:

```
import tensorflow as tf
from tensorflow import keras
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
```

- Each image contains pixels ranging 0 to 255, in decreasing darkness.
- An individual image is a 28×28 array of pixels.
- The desired digit is represented as a number from 0 to 9.

MNIST Dataset

- The MNIST database contains seventy thousand handwritten digits.
  - Each data-point contains both an image, and the desired digit.
  - $60,000$ images are designated for training, and $10,000$ for testing:

```python
import tensorflow as tf
from tensorflow import keras
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
```

- Each image contains pixels ranging 0 to 255, in decreasing darkness.
- An individual image is a 28×28 array of pixels.
- The desired digit is represented as a number from 0 to 9.

We'll build a model from the training images that will learn to classify digits!

What we're building towards

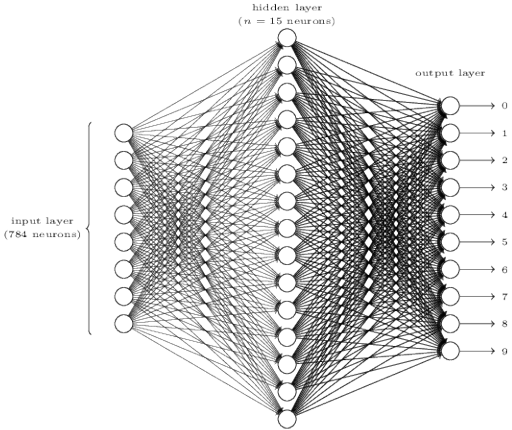

Figure 5: A simple neural network structure. The input vectors on the left hand side have $28 \times 28 = 784$ inputs for each pixel, and the output layer has 10 digits. (*Source: Nielsen*)

Extending our Model

All of our weights and bias will be initialized from a normal distribution with mean 0 and standard deviation 1.

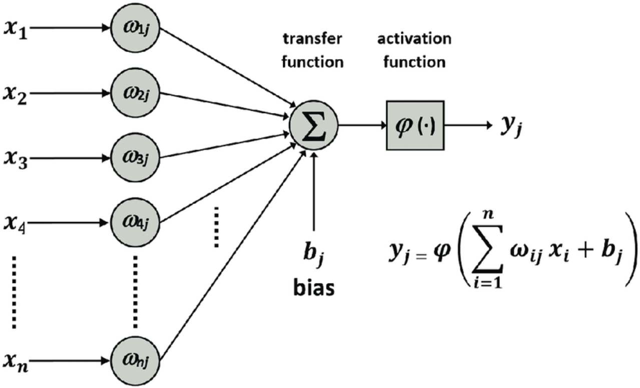

Figure 6: *Source: Daniel Alvarez, InTech*

Hidden Layer

The role of the **hidden layer** is to hold intermediate calculations. These will in turn be used to compute the output layer. To produce the hidden layer, we must have an 784×15 weight matrix, as seen below:

$$W = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1,15} \\ w_{21} & w_{22} & \cdots & w_{2,15} \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ w_{784,1} & w_{784,2} & \cdots & w_{784,15} \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ \vdots \\ x_{784} \end{pmatrix}.$$

We take the dot product of each **column** with our input vector **x**. We then add our bias vector, **b**, which is 15×1. We finally apply our activation:

$$\mathbf{h} = \sigma(W^T \mathbf{x} + \mathbf{b}).$$

Output Layer

We must now define a transformation from $\mathbb{R}^{15}$ to $\mathbb{R}^{10}$, which we can do using a 10×15 weight matrix $\hat{W}$. We can then add a 10×1 bias vector, $\hat{\mathbf{b}}$.

Output Layer

We must now define a transformation from $\mathbb{R}^{15}$ to $\mathbb{R}^{10}$, which we can do using a 10×15 weight matrix $\hat{W}$. We can then add a 10×1 bias vector, $\hat{\mathbf{b}}$. We aren't done yet! We want the output to be the probability an image is a specific digit. To do so, we use a **softmax** activation.

Output Layer

We must now define a transformation from $\mathbb{R}^{15}$ to $\mathbb{R}^{10}$, which we can do using a 10×15 weight matrix $\hat{W}$. We can then add a 10×1 bias vector, $\hat{\mathbf{b}}$. We aren't done yet! We want the output to be the probability an image is a specific digit. To do so, we use a **softmax** activation. The formula is

$$\text{softmax}(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{10} e^{z_k}}, \quad 1 \leq j \leq 10.$$

Output Layer

We must now define a transformation from $\mathbb{R}^{15}$ to $\mathbb{R}^{10}$, which we can do using a 10×15 weight matrix $\hat{W}$. We can then add a 10×1 bias vector, $\hat{\mathbf{b}}$. We aren't done yet! We want the output to be the probability an image is a specific digit. To do so, we use a **softmax** activation. The formula is

$$\text{softmax}(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{10} e^{z_k}}, \quad 1 \leq j \leq 10.$$

Notice the sum of these values will always be 1.

Output Layer

We must now define a transformation from $\mathbb{R}^{15}$ to $\mathbb{R}^{10}$, which we can do using a 10×15 weight matrix $\hat{W}$. We can then add a 10×1 bias vector, $\hat{\mathbf{b}}$. We aren't done yet! We want the output to be the probability an image is a specific digit. To do so, we use a **softmax** activation. The formula is

$$\text{softmax}(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{10} e^{z_k}}, \quad 1 \le j \le 10.$$

Notice the sum of these values will always be 1. The full computation is

$$\mathbf{o} = \text{softmax}\left(\hat{W}\mathbf{h} + \hat{\mathbf{b}}\right).$$

Output Layer

We must now define a transformation from $\mathbb{R}^{15}$ to $\mathbb{R}^{10}$, which we can do using a 10×15 weight matrix $\hat{W}$. We can then add a 10×1 bias vector, $\hat{\mathbf{b}}$. We aren't done yet! We want the output to be the probability an image is a specific digit. To do so, we use a **softmax** activation. The formula is

$$\text{softmax}\left(\mathbf{z}\right)_j = \frac{e^{z_j}}{\sum_{k=1}^{10} e^{z_k}}, \quad 1 \leq j \leq 10.$$

Notice the sum of these values will always be 1. The full computation is

$$\mathbf{o} = \text{softmax}\left(\hat{W}\mathbf{h} + \hat{\mathbf{b}}\right).$$

For instance, $\text{softmax}\left(\begin{pmatrix} 4 \\ 2 \\ 1 \end{pmatrix}\right) = \begin{pmatrix} 0.8438 \\ 0.1142 \\ 0.0420 \end{pmatrix}$ has max probability 84.38%.

One Hot Encoding

Once we've computed the output, we need a way to compare it to our desired result. However, $\mathbf{o}$ is a 10×1 vector, whereas our desired digit $y_{\text{train}}(\mathbf{x})$ is a scalar. We therefore encode the digit as a 10×1 vector:

One Hot Encoding

Once we've computed the output, we need a way to compare it to our desired result. However, $\mathbf{o}$ is a 10×1 vector, whereas our desired digit $y_{\text{train}}(\mathbf{x})$ is a scalar. We therefore encode the digit as a 10×1 vector:

$$\begin{array}{ccccc}
\begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} & \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix} & \dots & \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix} \\
\hline
0 & 1 & 2 & \cdots & 9
\end{array}$$

One Hot Encoding

Once we've computed the output, we need a way to compare it to our desired result. However, $\mathbf{o}$ is a 10×1 vector, whereas our desired digit $y_{\text{train}}(\mathbf{x})$ is a scalar. We therefore encode the digit as a 10×1 vector:

$$\begin{array}{ccccc} \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} & \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix} & \cdots & \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix} \\ \hline 0 & 1 & 2 & \cdots & 9 \end{array}$$

The code for this is relatively simple:

```
y_test=keras.utils.to_categorical(y_test, num_classes=10)
y_train=keras.utils.to_categorical(y_train, num_classes=10)
```

Negative Log Likelihood

To compute how accurate our model was at predicting a given value, we need a **loss** function. In this case, it's easiest to use *negative log likelihood*.

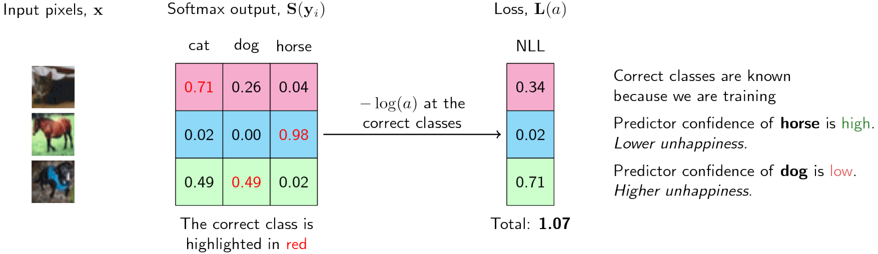

Figure 7: *Source: LJ Mirand*

To compute the loss for an individual training example, $\mathbf{x}$, with one-hot encoded label $y_{\text{train}}(\mathbf{x})$, and output $\mathbf{o}$, we compute

$$L(\mathbf{x}) = -y_{\text{train}}(\mathbf{x}) \cdot \log \mathbf{o} = -\log(o_j),$$

where j is the true label.

Training Parameters

Notice, in total we have $784 \times 15 + 15 \times 1 + 10 \times 15 + 10 \times 1 = 11,935$ parameters to train on.

Training Parameters

Notice, in total we have $784 \times 15 + 15 \times 1 + 10 \times 15 + 10 \times 1 = 11,935$ parameters to train on. Now, how much does our output depend on each of these parameters?

Training Parameters

Notice, in total we have $784 \times 15 + 15 \times 1 + 10 \times 15 + 10 \times 1 = 11,935$ parameters to train on. Now, how much does our output depend on each of these parameters? To answer this, we need the chain rule from calculus.

References

- Michael Nielsen: Using neural nets to recognize handwritten digits
- Towards Data Science: A Beginner's Guide to Neural Networks
- 3d Visualizing a Neural Network
- Understanding softmax and the negative log-likelihood