The Mathematical Intuition of Deep Learning

Wade Copeland

23 October, 2024

Why Learn the Mathematics of Deep Learning?

Learning the math behind a statistical method is like learning the secret to a magic trick. Once you know the math, you can never be fooled by the same trick again.

2024 Nobel Prize in Physics[1]

The Nobel Prize in Physics 2024

Ill. Niklas Elmehed © Nobel Prize Outreach

John J. Hopfield

Prize share: 1/2

Ill. Niklas Elmehed © Nobel Prize Outreach

Geoffrey E. Hinton

Prize share: 1/2

The Nobel Prize in Physics 2024 was awarded jointly to John J. Hopfield and Geoffrey E. Hinton "for foundational discoveries and inventions that enable machine learning with artificial neural networks"

Applications to the Oversight Mission

- Each IG is responsible for the oversight of agencies that will implement deep learning to make more informed and better decisions, but come with risks such as bias, privacy, lack of transparency, and ethical complexities.
- Each OIG will use deep leaning to provide better oversight.
  - Auditors will use deep learning to write better recommendations based on data about how clients responded to previous recommendations.
  - Evaluators will use deep learning to find evidence of systemic problems in large volumes of data.
  - Investigators will use deep learning to find leads based on patterns observed their case data.

The Deep in Deep Learning

- Deep learning always refers to deep neural networks.
- A deep neural network is a neural network with more than two layers.
- ChatGPT Version 4 is reported to have 1.8 trillion parameters across 120 layers[2]!

Preliminaries

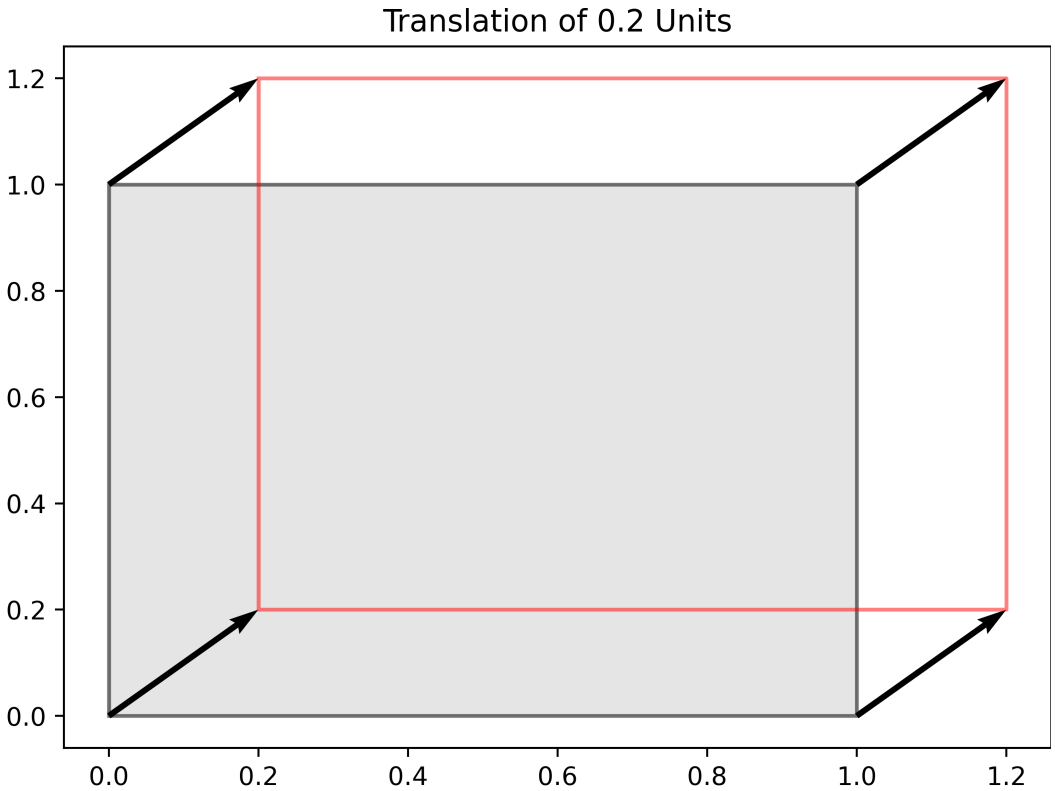
- This presentation is aimed at the someone who wants to learn about deep learning without having a background in sophisticated mathematics.
- We briefly introduce some concepts from *linear algebra* and *differential calculus*, but will focus on an intuitive understanding.

Matricies

- A *matrix* is a grid of numbers with N rows and P columns denoted $X_{N\times P}$.
  - $X_{2\times 2} = \begin{bmatrix} x_1 & x_2 \\ x_3 & x_4 \end{bmatrix}$ is a 2×2 matrix.
  - $X_{1\times 1} = \begin{bmatrix} x_1 \end{bmatrix}$ is a 1×1 matrix, also called a *scalar*.
  - $X_{1\times 2} = \begin{bmatrix} x_1 & x_2 \end{bmatrix}$ is a 1×2 matrix, also called a *vector*.

Matrix Translation

- *Matrix translation* is the addition/subtraction of a constant to every element of a matrix.
  - The columns of $X_{2\times 6} = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$ describe a square in 2-dimensional space. We can translate the square by 0.2 units by adding 0.2 to each element of $X_{2\times 6}$.

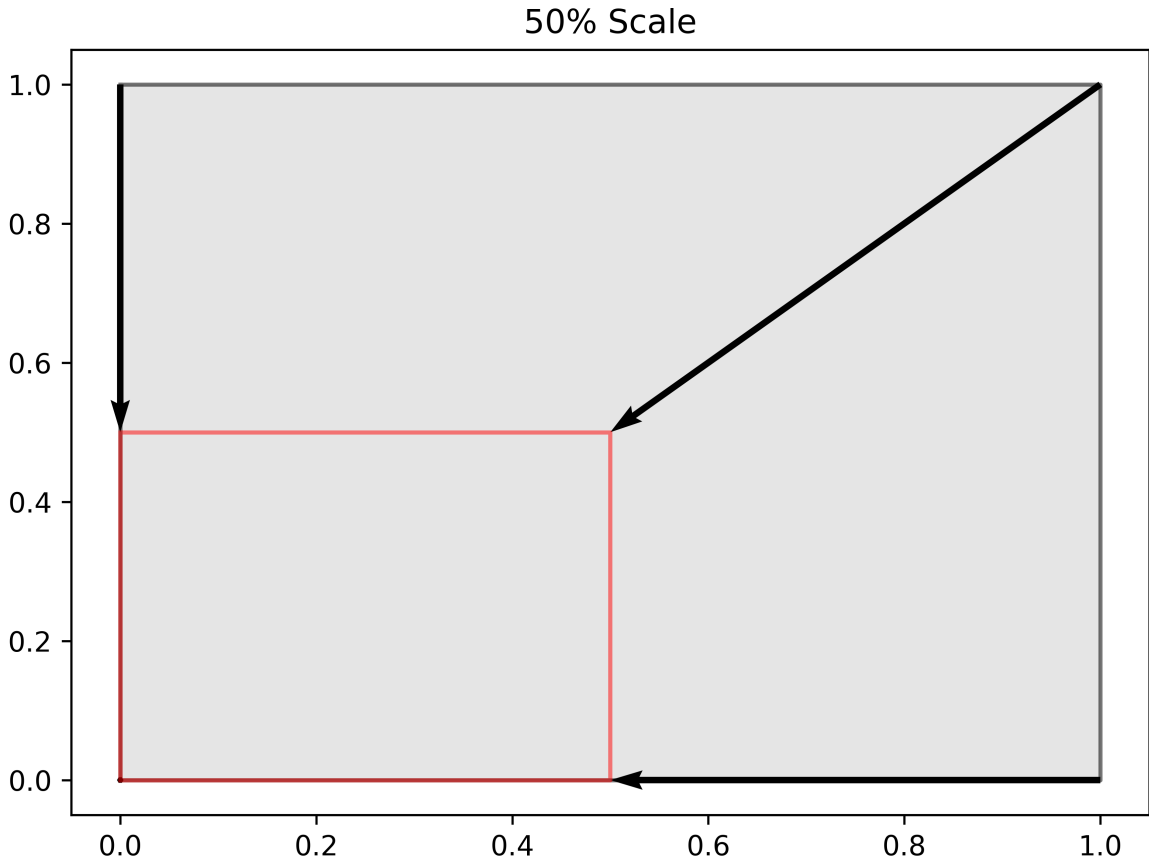


Matrix Transformation

- *Matrix transformation* is achieved through the matrix-specific operation called the *dot product* where the rows of X and the columns of Y are multiplied and summed.
  - The dot product of
    $$X_{1\times 2} \cdot Y_{2\times 1} = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 y_1 + x_2 y_2 \end{bmatrix}$$
  - If the number of columns in X does not equal the number of rows in Y then the dot product operation is not defined.
  - The size of the resulting matrix is equal to the number of rows in X and the number of columns in Y.

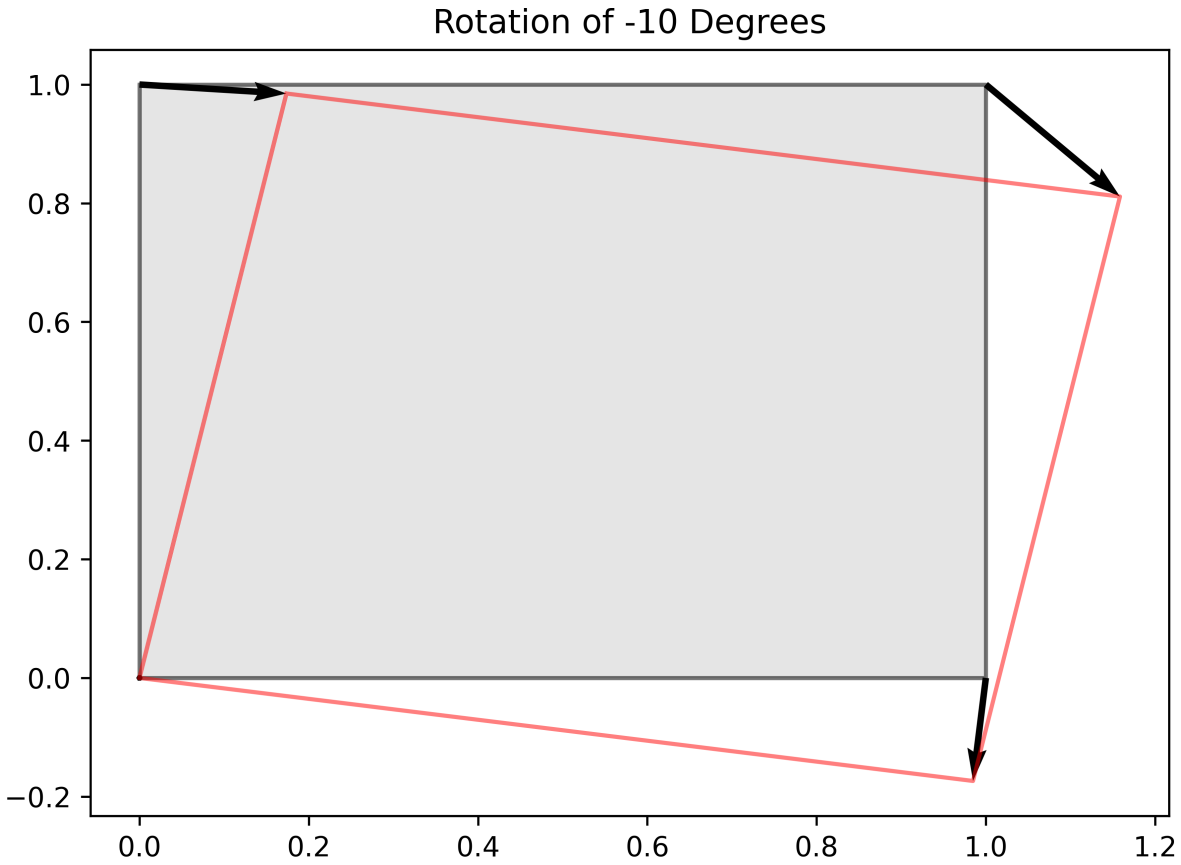
Matrix Transformation: Scale

- $X_{2\times 6}$ can be scaled by 50% taking the dot product of $\begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$ and $X_{2\times 6}$.



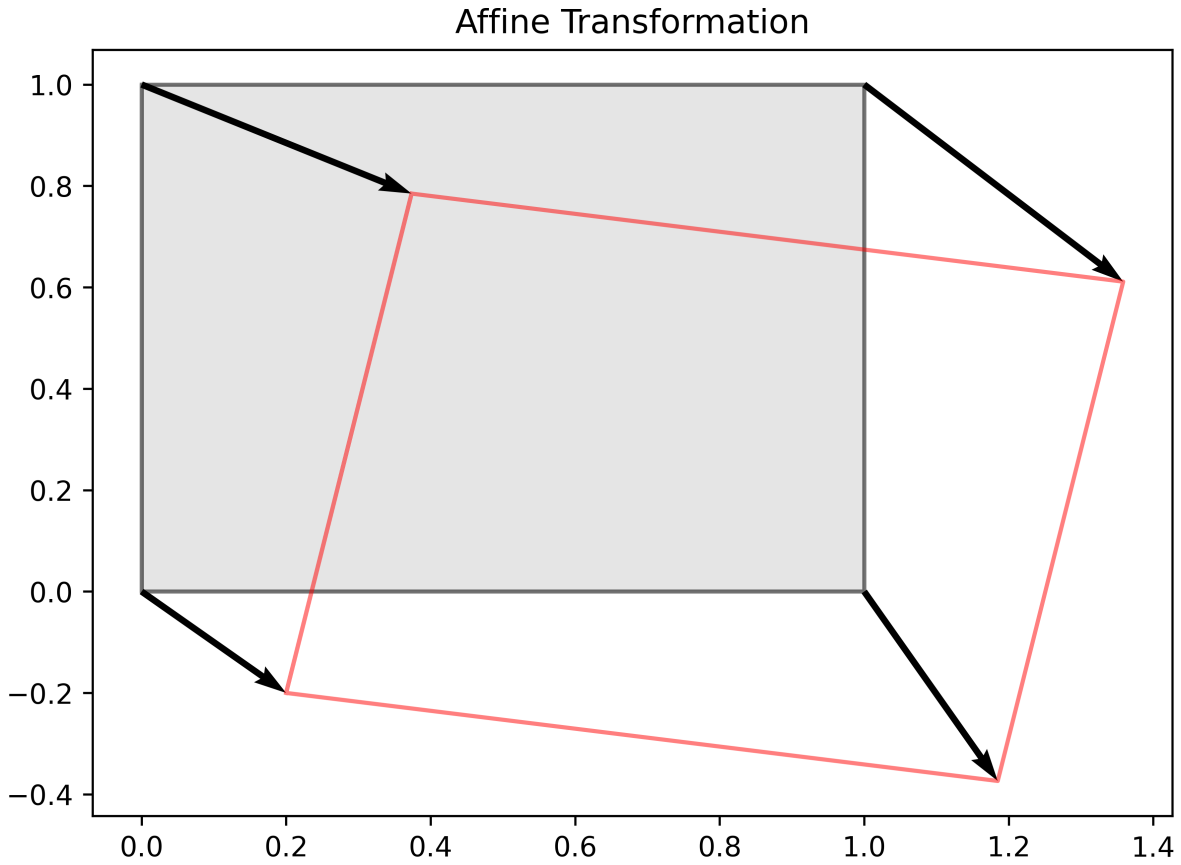
Matrix Transformation: Rotation

- $X_{2\times 6}$ can be rotated by negative 10 degrees by taking the dot product of $\begin{bmatrix} cos(0.175) & -sin(0.175) \\ sin(0.175) & cos(0.175) \end{bmatrix}$ and $X_{2\times 6}$.



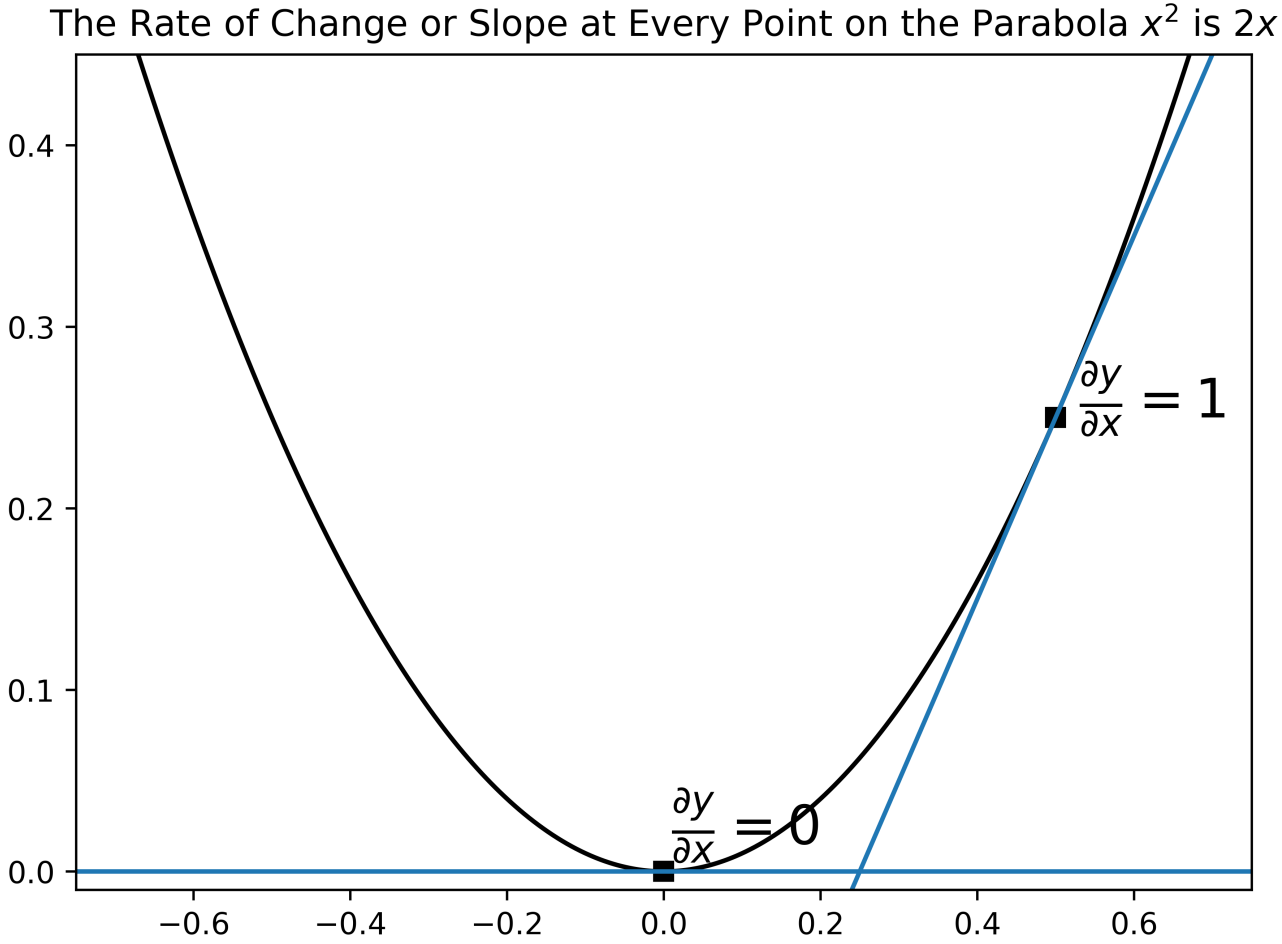
Matris Transformation: Affine

- An affine transformation of $X_{2\times 6}$ combines transformation and translation. For example, combining the rotation transformation of -10 degrees and translation of 0.2 units.



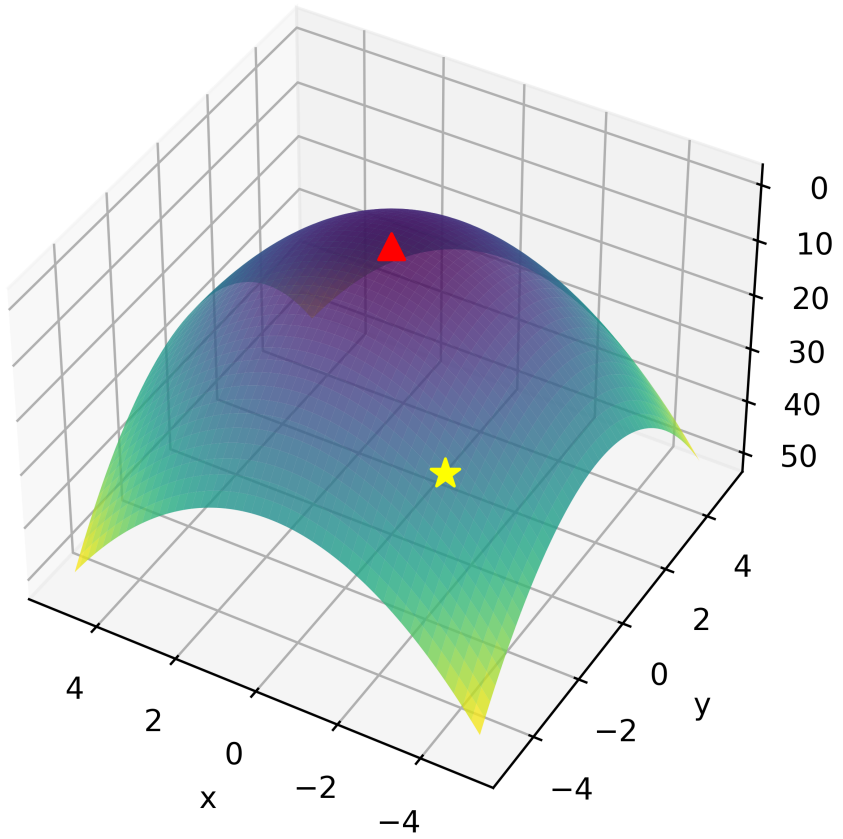
Rate of Change: The Derivative

- A *derivative* measures the rate of change or slope at any point denoted by $\frac{\partial y}{\partial x}$, which is read "the change in y with respect to the change in x."



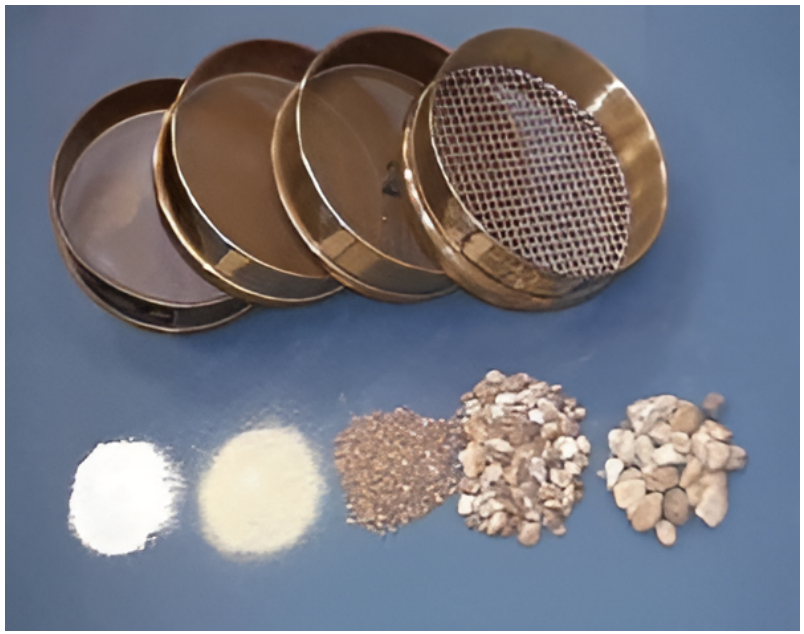
Rate of Change: The Gradient

- The *gradient* is a generalization of the derivative that measures the rate of change or slope in every direction. For example, if we have the surface $z = x^2 + y^2$ the slope in the x direction is $\frac{\partial z}{\partial x} = 2x$ and the slope in the y direction is $\frac{\partial z}{\partial y} = 2y$.



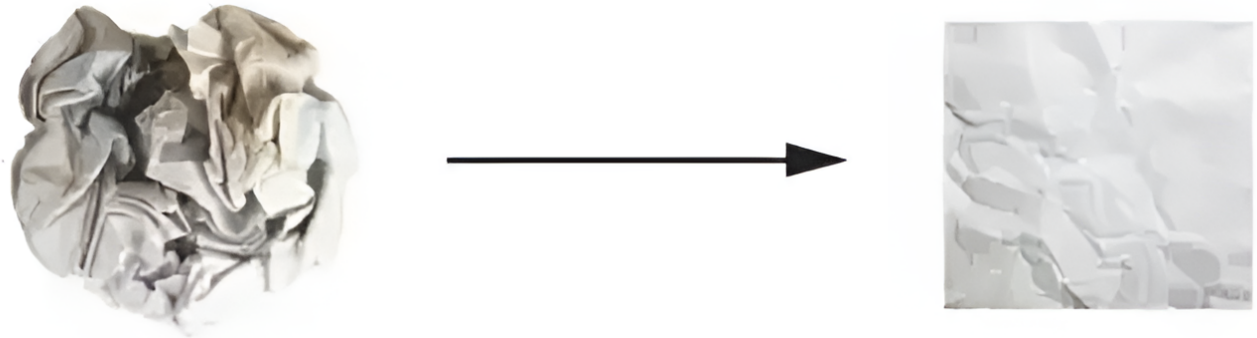
Deep Learning Intuition I

- A neural network is an information sieve that takes as input the outcomes, and the features we want to use to predict the outcomes[3].
- At the top of the sieve is the most granular representation of the data, and for each successive layer down, the data becomes more and more refined, that is, better at predicting the outcome[3].



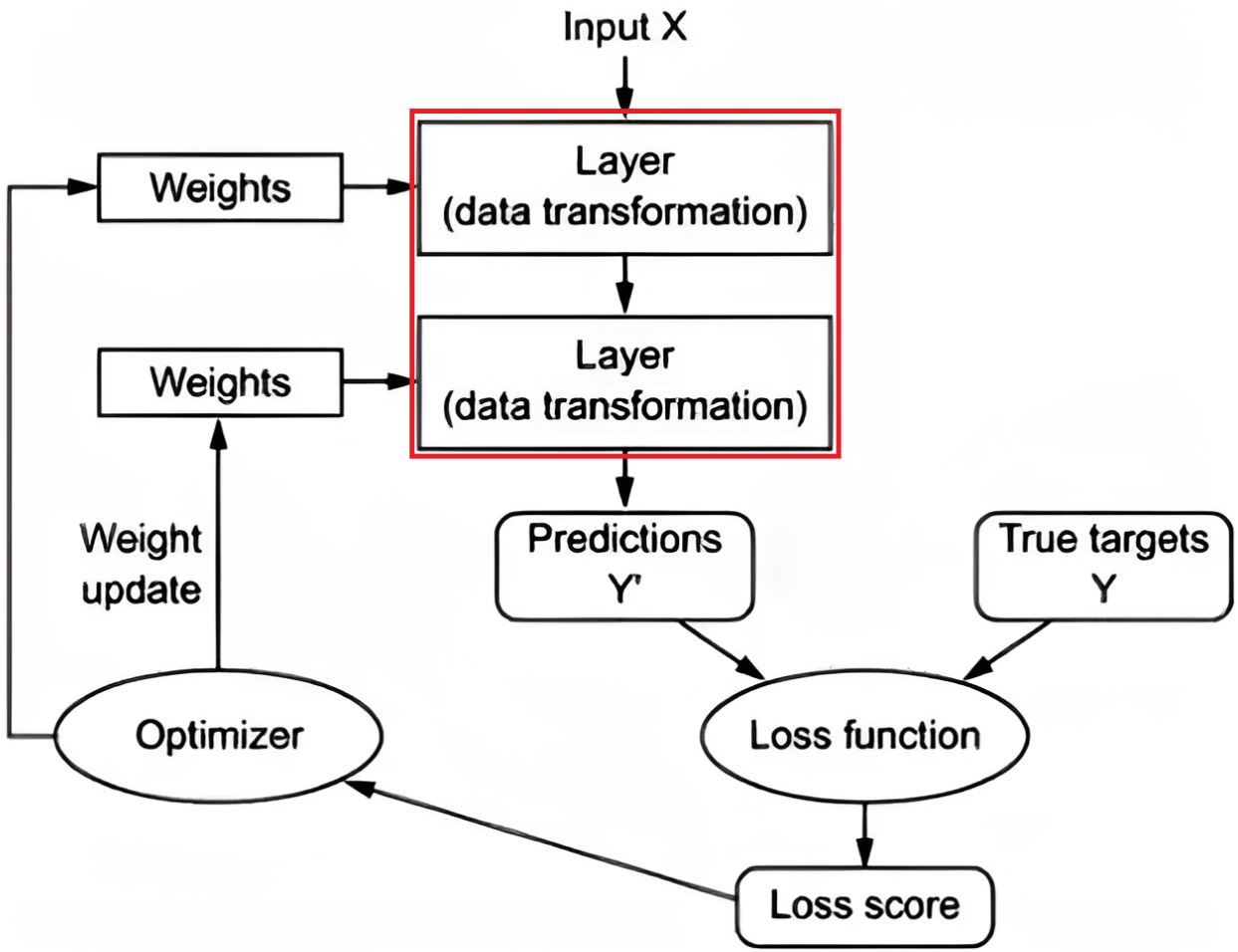
Deep Learning Intuition II

- Consider a blue and red piece of paper crumpled together and your goal is to uncrumple them. Each movement of your fingers to uncrumple the papers is like a layer of a neural network, until finally at the last layer the colors are fully separated[3].



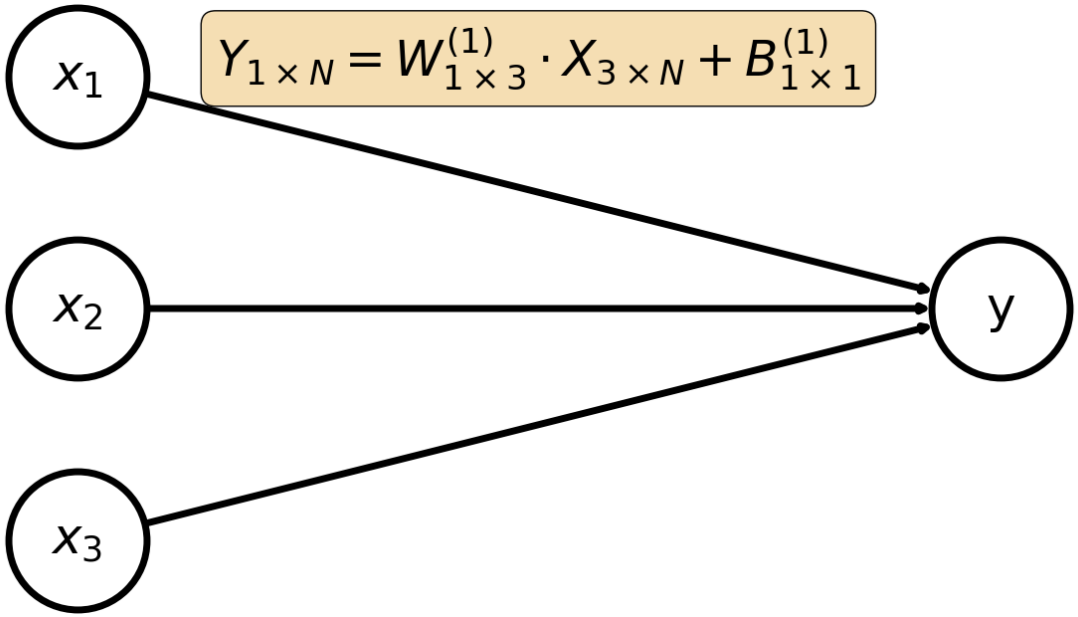
Deep Learning Roadmap: Layers

- The full process of fitting a neural network is shown below. As a first step, we will explain the mathematical representation of the layers[3].



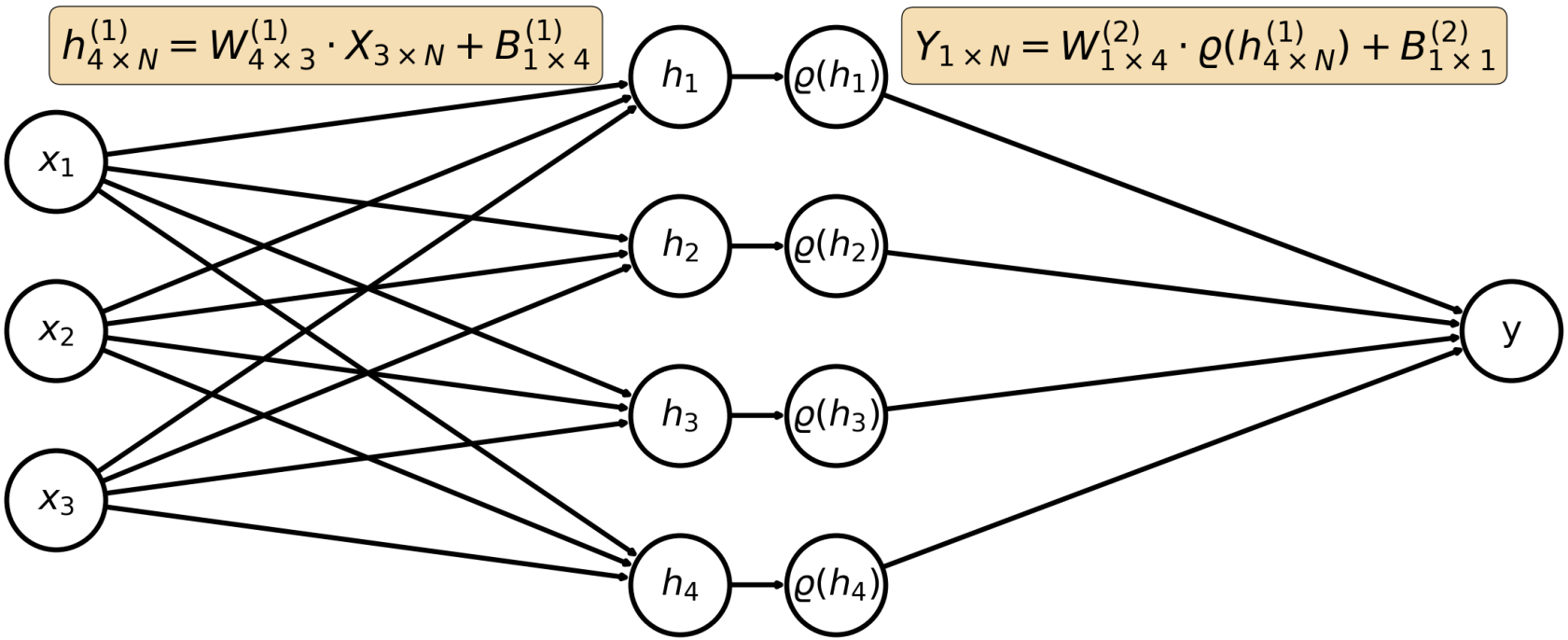
Representation of a Single-Layer Neural Network

- A single layer neural network is represented by the affine transformation $Y = W \cdot X + B$[4].



Representation of a Two-Layer Neural Network

- A two layer neural network includes a layer of *neurons* between the features and the outcome. These are called *hidden neurons* because they are not observed in the input data[4].



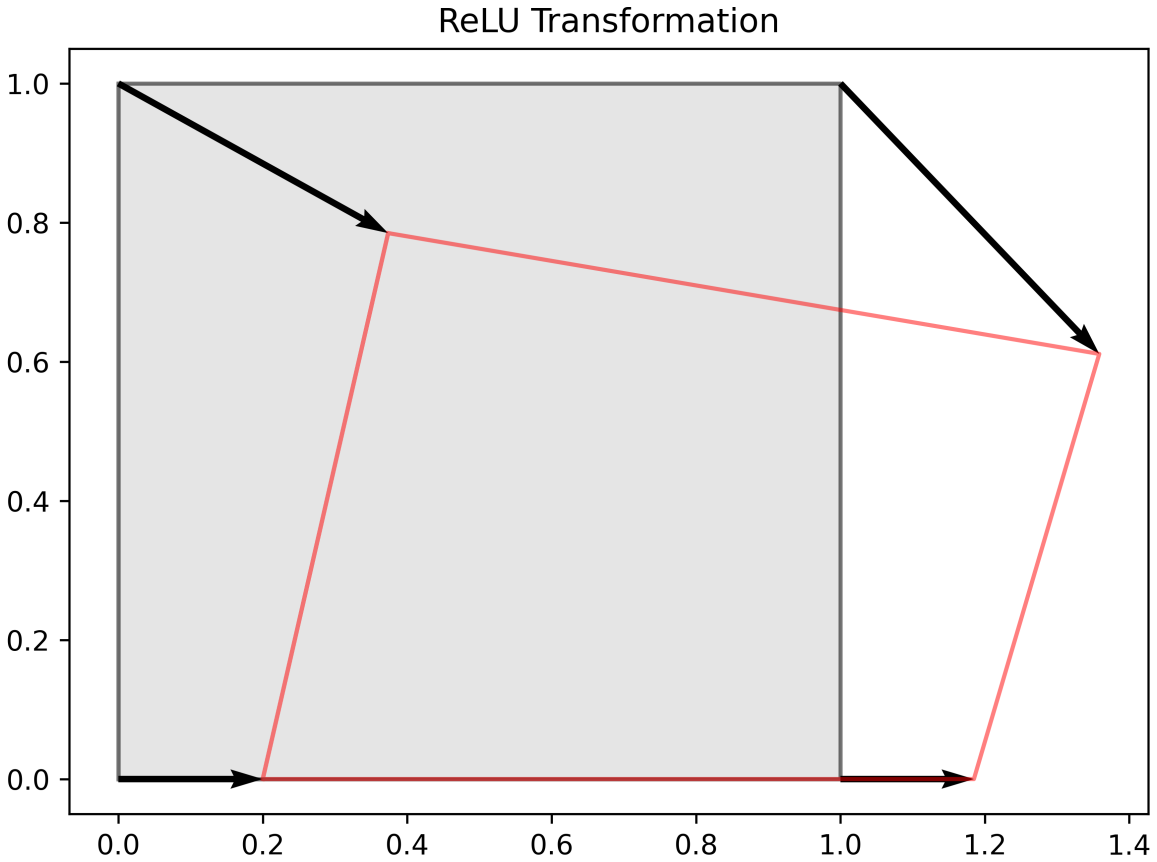
What Is $\varrho(h)$ and Why Do We Need It?

- The function $\varrho(h)$ is a non-linear transformation applied to the otherwise linear affine transformation, also called an *activation function*[4].
- A two-layer neural network without a non-linear transformation between layers is a single-layer neural network in disguise[3]!

$$
\begin{aligned}
Y_{1\times N} = \\
W^{(2)}_{1\times 4} \cdot h^{(1)}_{4\times N} + B^{(2)}_{1\times 1} = \\
W^{(2)}_{1\times 4} \cdot (W^{(1)}_{4\times 3} \cdot X_{3\times N} + B^{(1)}_{1\times 4}) + B^{(2)}_{1\times 1} = \\
(W^{(2)}_{1\times 4} \cdot W^{(1)}_{4\times 3}) \cdot X_{3\times N} + (W^{(2)}_{1\times 4} B^{(1)}_{1\times 4} + B^{(2)}_{1\times 1})
\end{aligned}
$$

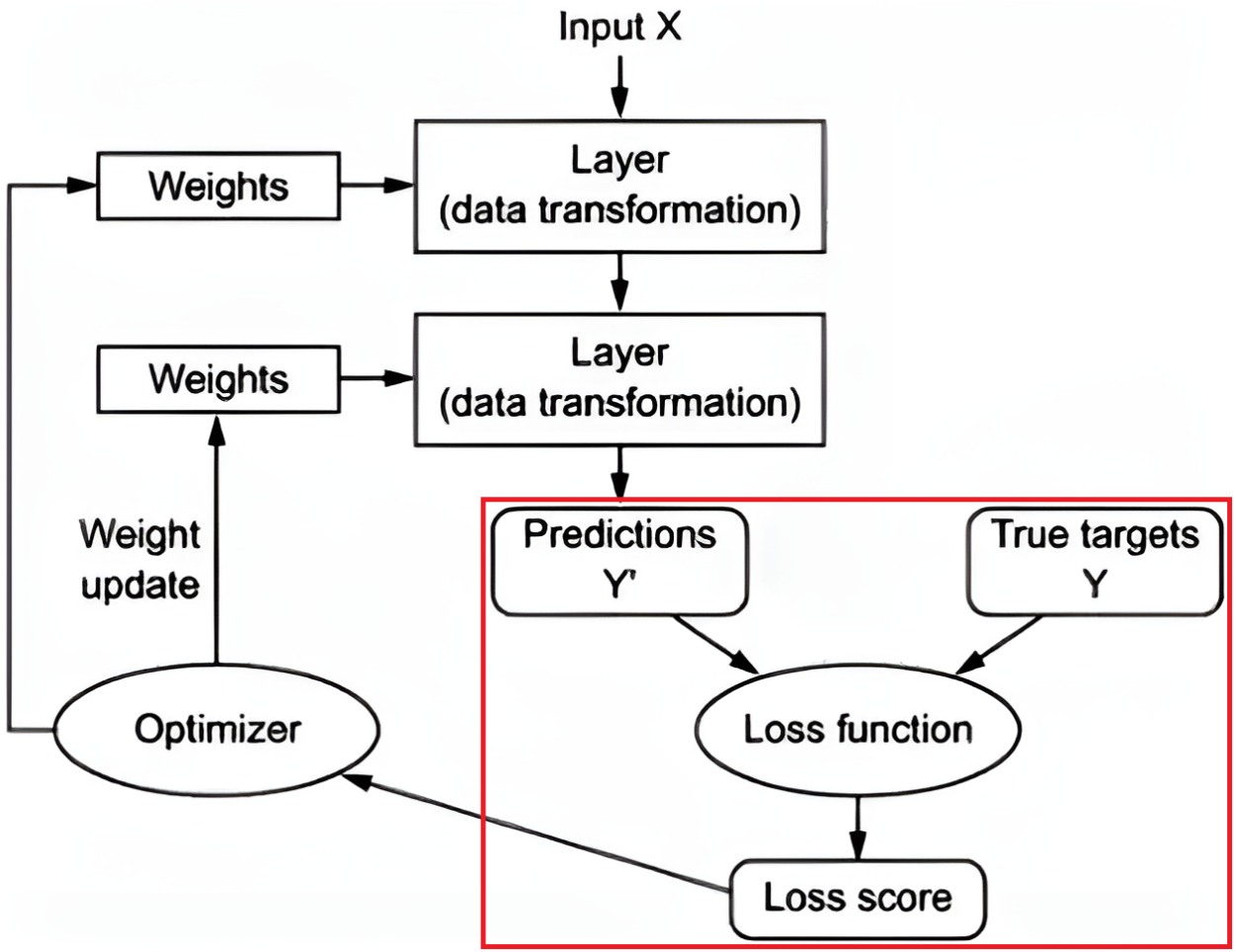
Rectified Linear Unit (ReLU)

- A common $\varrho(h)$ is the *ReLU* transformation which replaces all negative values with 0[4].



Deep Learning Roadmap: Loss

- After a neural network is fit, we need a way to measure how good it is at predicting the outcome of interest. This is called the *loss function*[3].



Measures of Loss

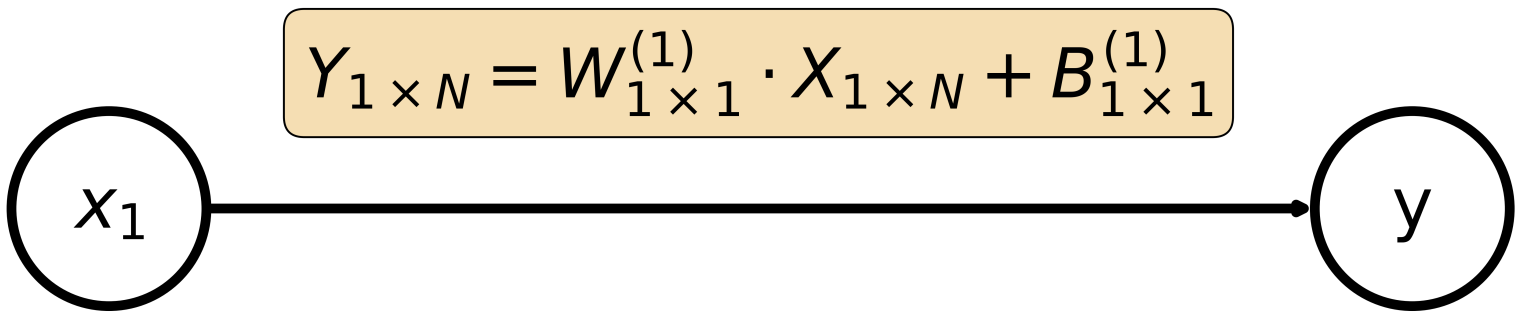
- For a numeric outcome a common measure of loss is the *mean squared error*, which measures the average of the squared deviations from the observed outcomes and our predictions of them[5].

$$e = \frac{1}{n}\sum_{i=1}^{n}(y - y')^2$$

- There are many other measure of loss, such as *categorical cross entropy*, which can be used to measure loss when the outcome is categorical[3].

Forward Pass I

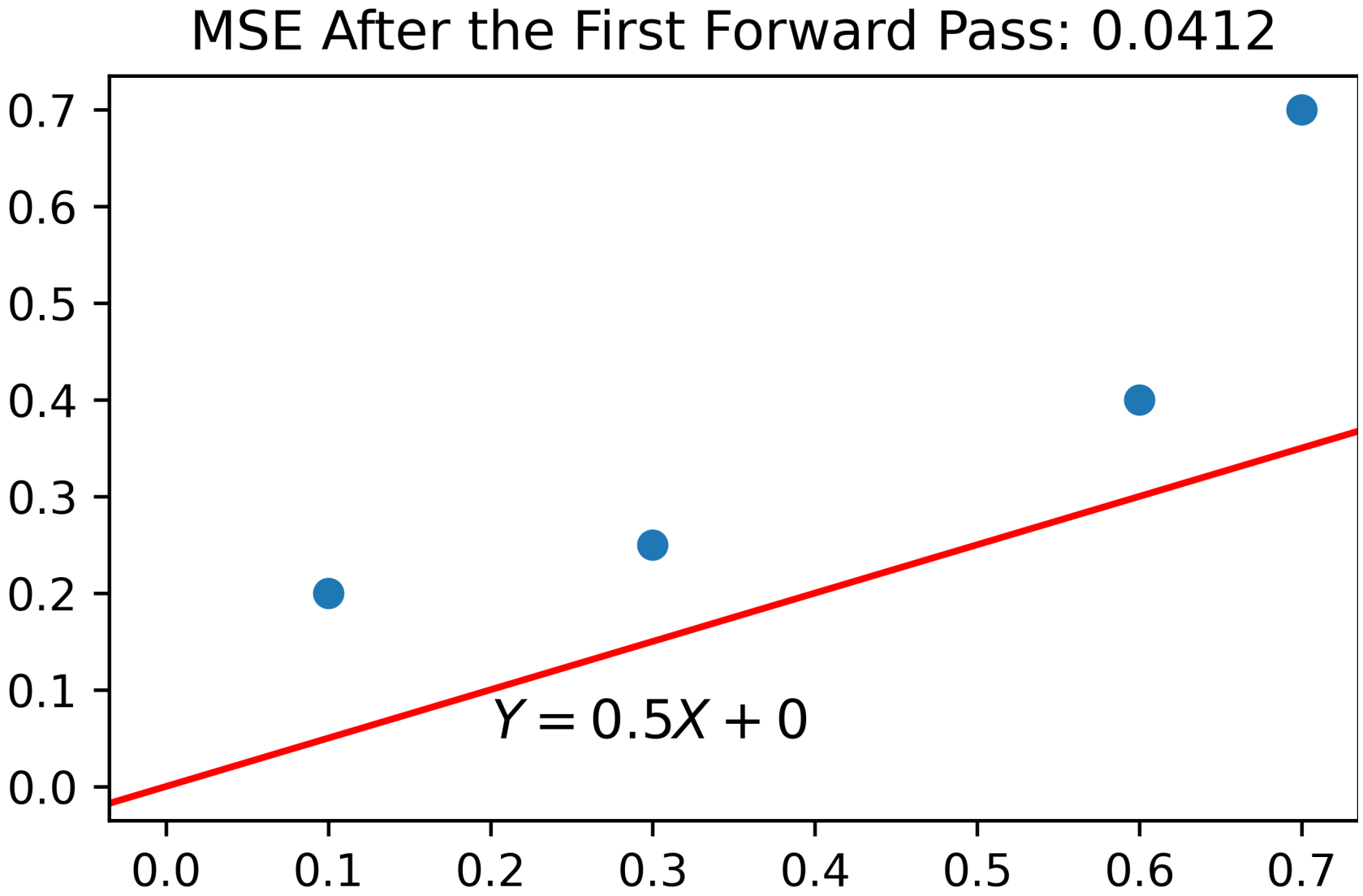
- Consider the most basic of neural networks, a single feature used to predict a single outcome.
- The *forward pass* calculates the predictions of the outcome from the input data using the values of W and B^3.



Forward Pass II

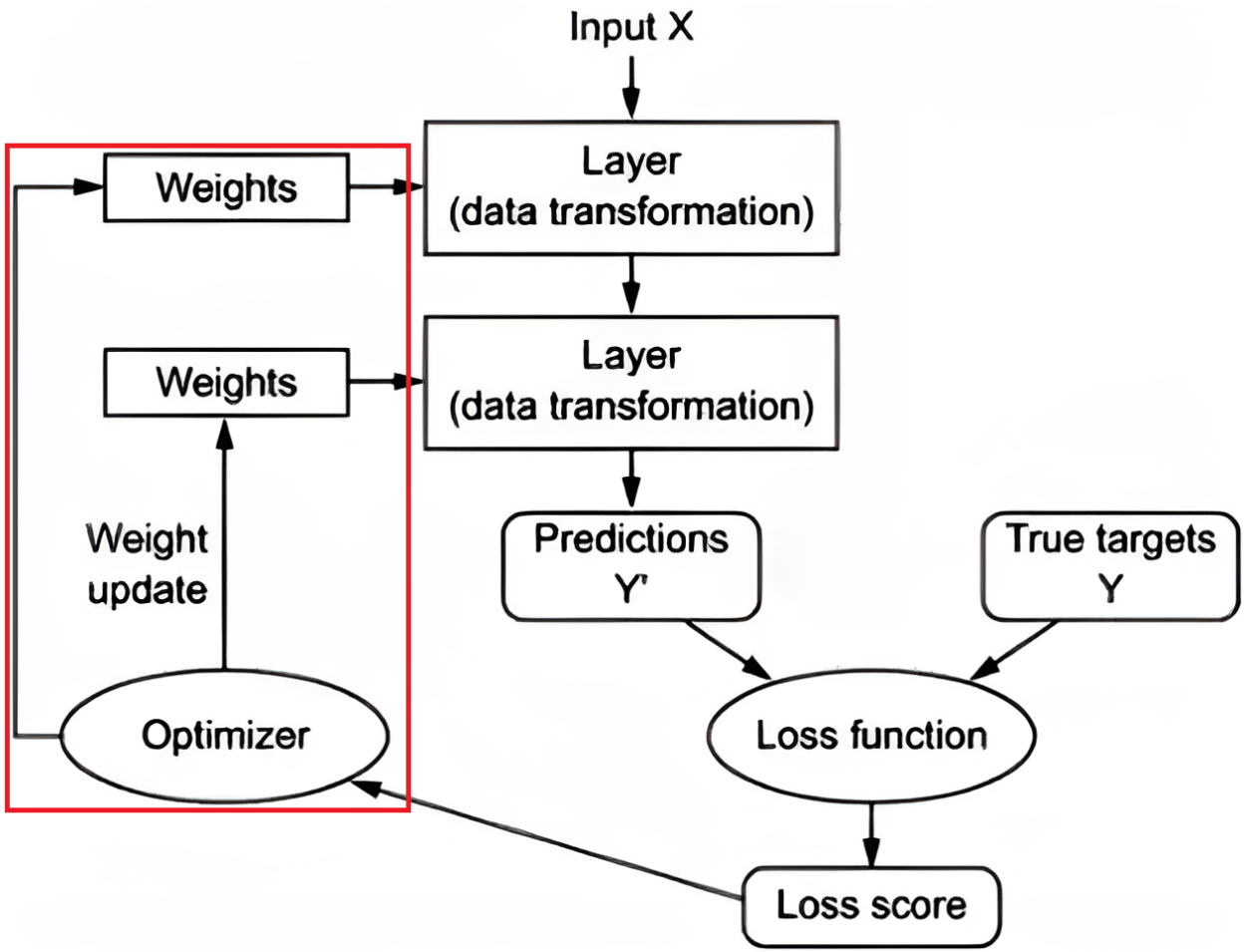
- Let $Y = \begin{bmatrix} 0.2 & 0.25 & 0.4 & 0.7 \end{bmatrix}$, $X = \begin{bmatrix} 0.1 & 0.3 & 0.6 & 0.7 \end{bmatrix}$, $W = \begin{bmatrix} 0.5 \end{bmatrix}$, $B = \begin{bmatrix} 0 \end{bmatrix}$.
- The initial predictions from the first forward pass are $Y' = W \cdot X + B = \begin{bmatrix} 0.5 \end{bmatrix} \cdot \begin{bmatrix} 0.2 & 0.25 & 0.4 & 0.7 \end{bmatrix} + \begin{bmatrix} 0 \end{bmatrix} = \begin{bmatrix} 0.05 & 0.15 & 0.3 & 0.35 \end{bmatrix}$.
- The calculated loss using the MSE is $e = \frac{1}{4}((0.2 - 0.05)^2 + (0.25 - 0.15)^2 + (0.4 - 0.3)^2 + (0.7 - 0.35)^2) = 0.0412$.

Forward Pass III



Deep Learning Roadmap: Optimization

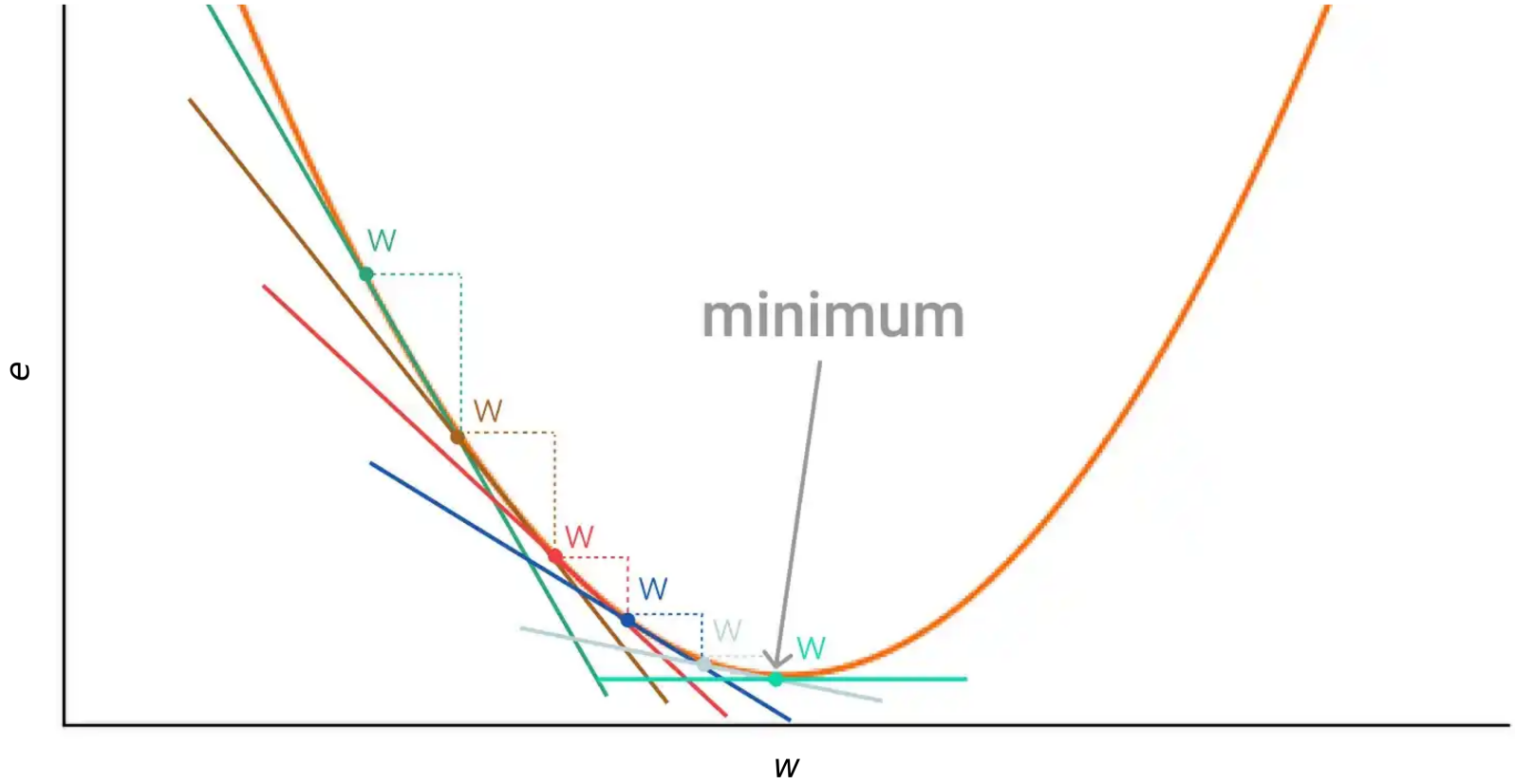
- The optimization step has the goal of *learning* the values of W and B that minimize the MSE[3].



Gradient Descent I

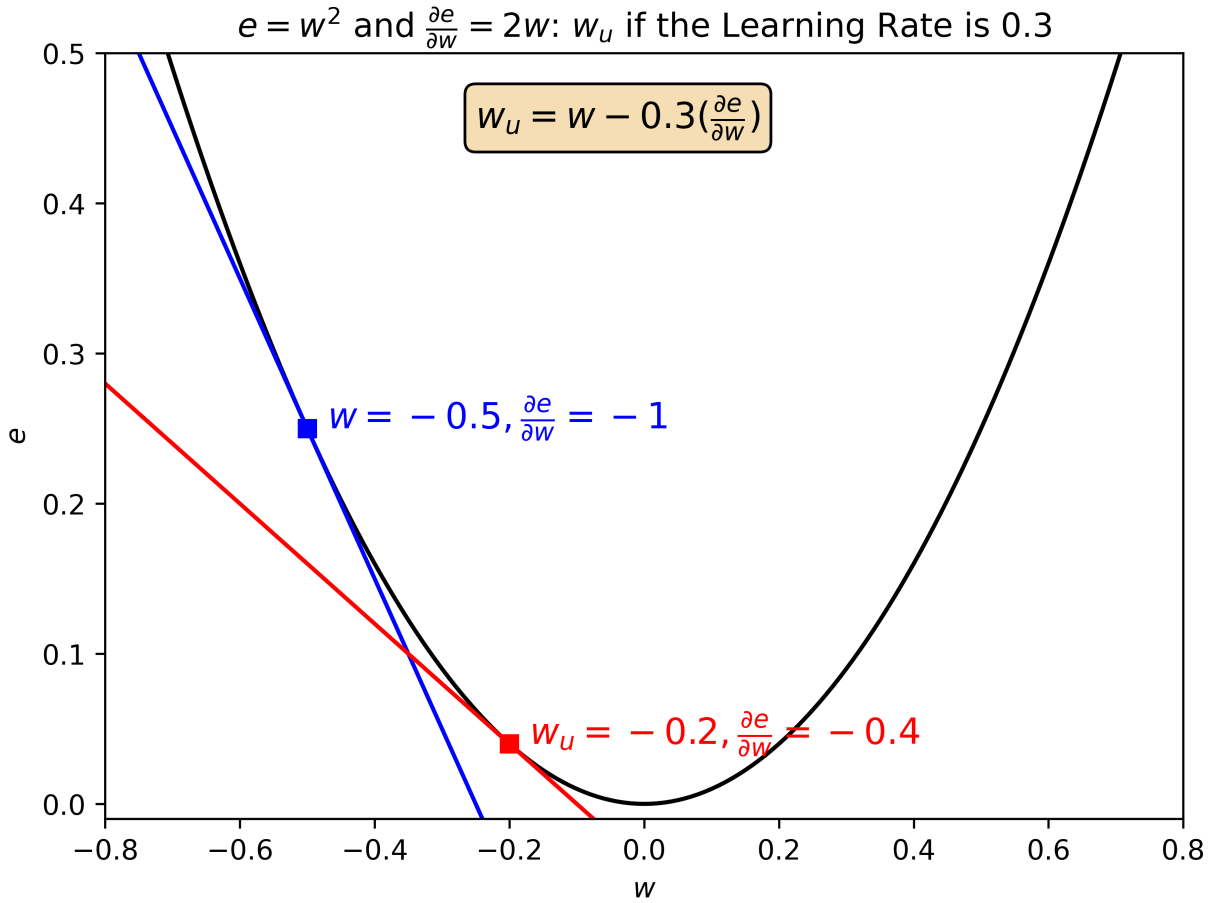
- Optimization is done numerically through an algorithm called *gradient descent*.
- Gradient descent works by stepping down a function in the opposite direction of the gradient[6].

The Gradient Descent Algorithm to Find the Minimum of a Parabola



Gradient Descent II

- In general, if we move w and b in the opposite direction of the gradient $\frac{\partial e}{\partial w}$ and $\frac{\partial e}{\partial b}$, then the MSE will decrease[3].
  - The updated value of w is $w_u = w - \alpha(\frac{\partial e}{\partial w})$.
  - The updated value of b is $b_u = b - \alpha(\frac{\partial e}{\partial b})$.

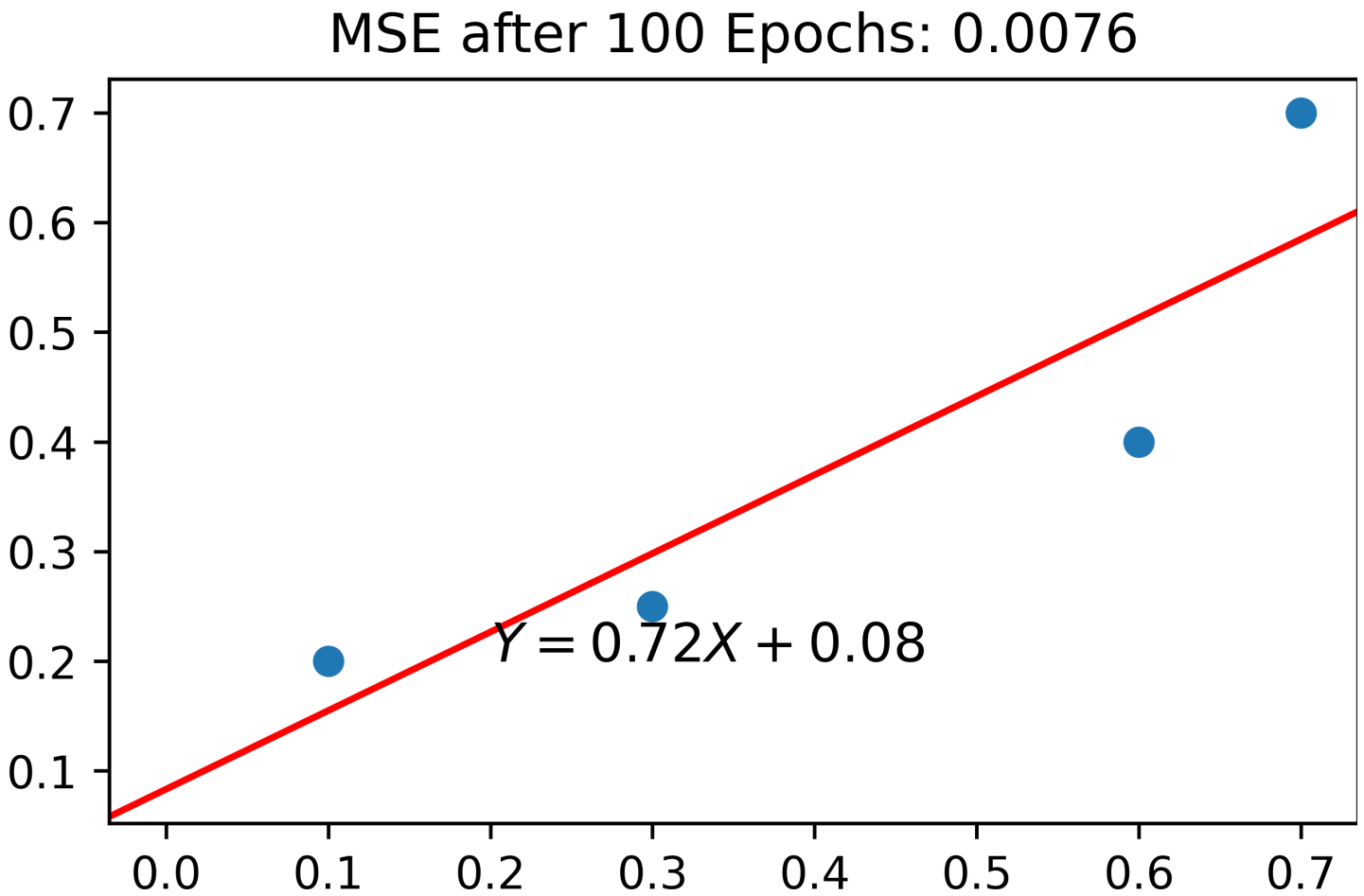


A Note About Calculating Gradients

- In the previous slides we made the assumption that calculating the gradient of a loss function is easy, but for neural networks with many layers and thousands of parameters, this is not true.
- The problem of efficiently calculating the gradient is solved using *backpropagation*.
  - Backpropagation works by chaining together the gradients of very simple expressions that are part of the computation graph of a neural network to calculate the otherwise very complicated gradient for the entire neural network.
  - All of the individual gradients taken together have the appearance of being taped together across the computation graph of a neural network, which is why backpropagation is sometimes called *gradient tape*.

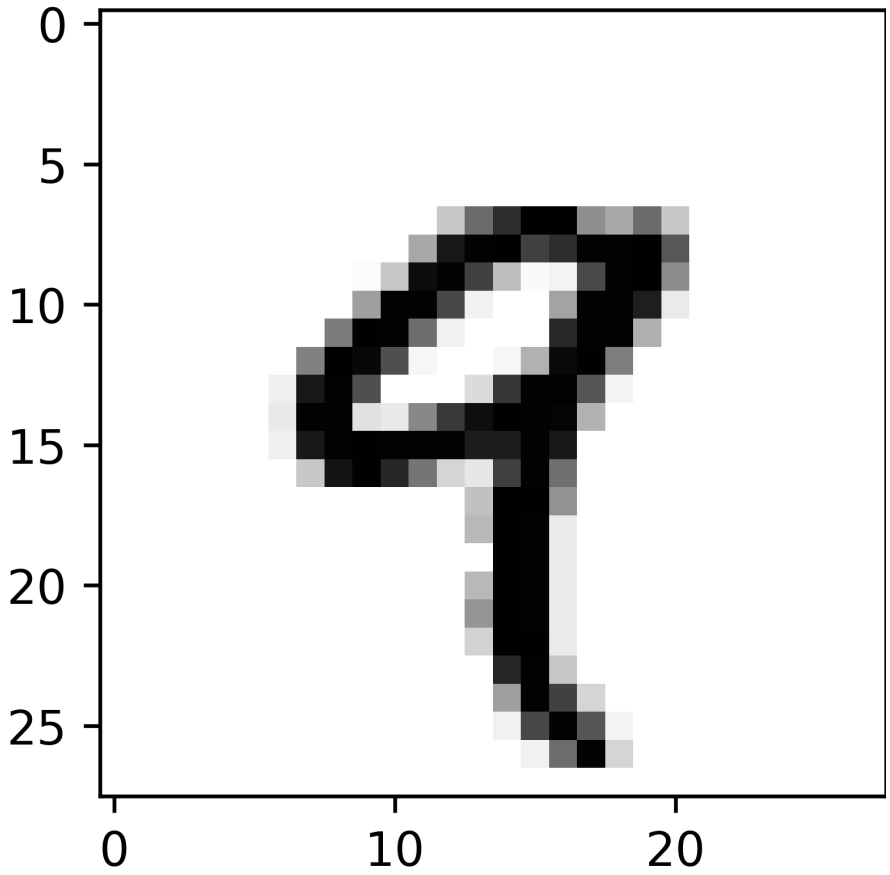
Backward Pass

- Adjusting the values of W and B using gradient descent is called is called a *backward pass*[3].
- Each cycle of a forward pass followed by a backward pass is called an *epoch*[3].



A Real Example: Digit Image Classification

- One of the best use cases for neural networks is image classification.
- We will use the MNIST data set that is 70,000 images of hand-written digits 0-9 that are 28x28 pixels[7].



A Real Example: Data Setup

- For analysis each image map of 28x28 pixels needs to be converted into a vector of size 28*28 = 784.
- The images are separated into a set of 60,000 images for training the model and a set of 10,000 images for testing the prediction accuracy in unseen data.

```
from tensorflow.keras.datasets import mnist

(train_images, train_labels), (test_images, test_labels) = (
    mnist.load_data()
)

train_images = train_images.reshape((60000, 28 * 28))
test_images = test_images.reshape((10000, 28 * 28))
```

A Real Example: Model Setup

- The first layer of the neural network contains 512 hidden neurons and uses the ReLU activation function.
- The output layer of the neural network is a 10-way Softmax classification layer that returns the probability that each digit belongs to a specific class.

```
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
    layers.Dense(512, activation="relu"),
    layers.Dense(10, activation="softmax")
])
```

A Real Example: Model Compilation

- The **loss** argument specifies how the model loss is calculated at the end of each forward pass.
- The **optimizer** argument specifics the optimization algorithm used to update the parameter values for the backward pass.
- The **metric** argument specifics how we will assess the model performance.

```
model.compile(
    loss="sparse_categorical_crossentropy",
    optimizer="rmsprop",
    metrics=["accuracy"])
```

A Real Example: Model Fit

- In the test data, the fitted neural network correctly classified each digit greater than 95% of the time!
- The difference in the accuracy of the training data and test data is an example of *overfitting*[3].

```
_ = model.fit(
    train_images, train_labels, epochs=5, batch_size=128, verbose = 0)

train_acc = model.evaluate(train_images, train_labels, verbose = 0)[1]
test_acc = model.evaluate(test_images, test_labels, verbose = 0)[1]

train_acc
```

```
## 0.9768166542053223
```

```
test_acc
```

```
## 0.9617999792098999
```

Summary

- This presentation peeled back the layers of what appears to be the *magic* of deep learning.
  - We discussed how *layers* in a neural network are like sieves for creating increasingly purified information.
  - We showed that the mathematical representation of each layer is just a combination of *matrix translation* and *matrix transformation*.
  - We explained that the purpose of the *activation function* applied to each layer is to detect non-linearity in the output function.
  - We showed how after each *forward pass* from the input to the output of the model, how the parameters in each layer are updated in the *backward pass* using *gradient descent* and *backpropagation*.
  - We concluded by presenting an example of fitting a neural network model using real data.

References I

1. NobelPrize.org. (2024). *The nobel prize in physics 2024*. https://www.nobelprize.org/prizes/physics/2024/summary/
2. Arya, N. (2023, July 19). *GPT-4 details have been leaked!* https://www.kdnuggets.com/2023/07/gpt4-details-leaked.html
3. Chollet, F. (2021). *Deep learning with python*. Simon; Schuster.
4. Berner, J., Grohs, P., Kutyniok, G., & Petersen, P. (2021). The modern mathematics of deep learning. *arXiv Preprint arXiv:2105.04026*, *78*.
5. Fox, J. (2008). *Applied regression analysis and generalized linear models*. Sage publications.

References II

6. Bento, C. (2021, June 2). *Stochastic gradient descent explained in real life*. https://towardsdatascience.com/stochastic-gradient-descent-explained-in-real-life-predicting-your-pizzas-cooking-time-b7639d5e6a32
7. Chollet, F. et al. (2015). *Keras*. https://keras.io.