Practice of Deep Learning
Day 1, Part 4/4

Matthieu De Mari

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

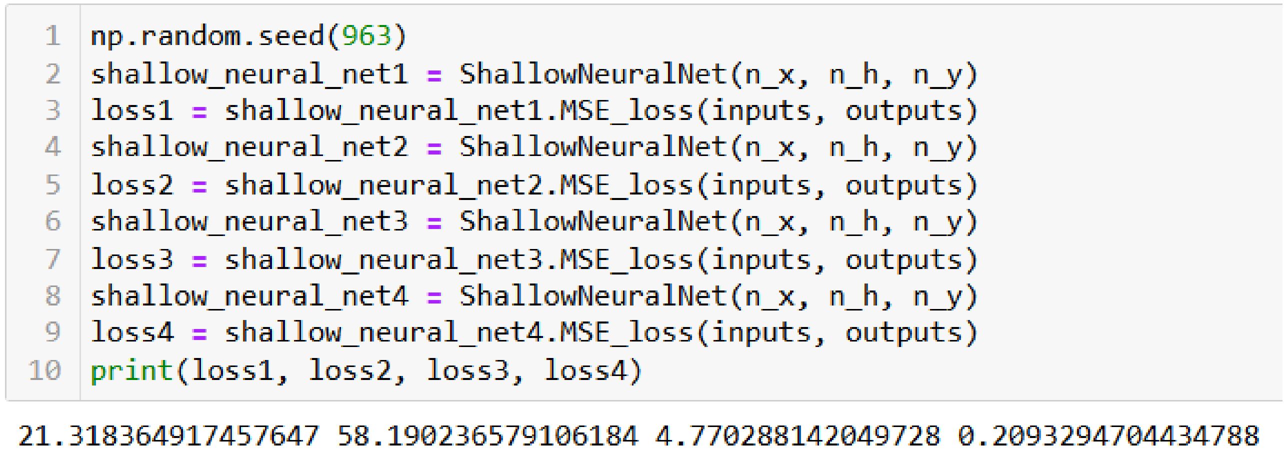
About this lecture

1. How to **implement a shallow Neural Network** manually and define a **forward propagation** method for it?
2. Next million dollar question: How to **train a shallow Neural Network** using **backpropagation**?
3. How to define **backward propagation** and **trainer** functions?

The need for a training procedure

At the moment, we have coded a model that can initialize trainable parameters randomly, formulate predictions and evaluate its own performance.

- It is great, but we have no way to adjust the weights manually.
- We can only try a few different initialization and pray the RNG gods to give us trainable parameters with a good loss.
- **We need a training procedure!**
- **What could it be…?**



```
np.random.seed(963)
shallow_neural_net1 = ShallowNeuralNet(n_x, n_h, n_y)
loss1 = shallow_neural_net1.MSE_loss(inputs, outputs)
shallow_neural_net2 = ShallowNeuralNet(n_x, n_h, n_y)
loss2 = shallow_neural_net2.MSE_loss(inputs, outputs)
shallow_neural_net3 = ShallowNeuralNet(n_x, n_h, n_y)
loss3 = shallow_neural_net3.MSE_loss(inputs, outputs)
shallow_neural_net4 = ShallowNeuralNet(n_x, n_h, n_y)
loss4 = shallow_neural_net4.MSE_loss(inputs, outputs)
print(loss1, loss2, loss3, loss4)
```

```
21.318364917457647 58.190236579106184 4.770288142049728 0.2093294704434788
```

The backpropagation mechanism

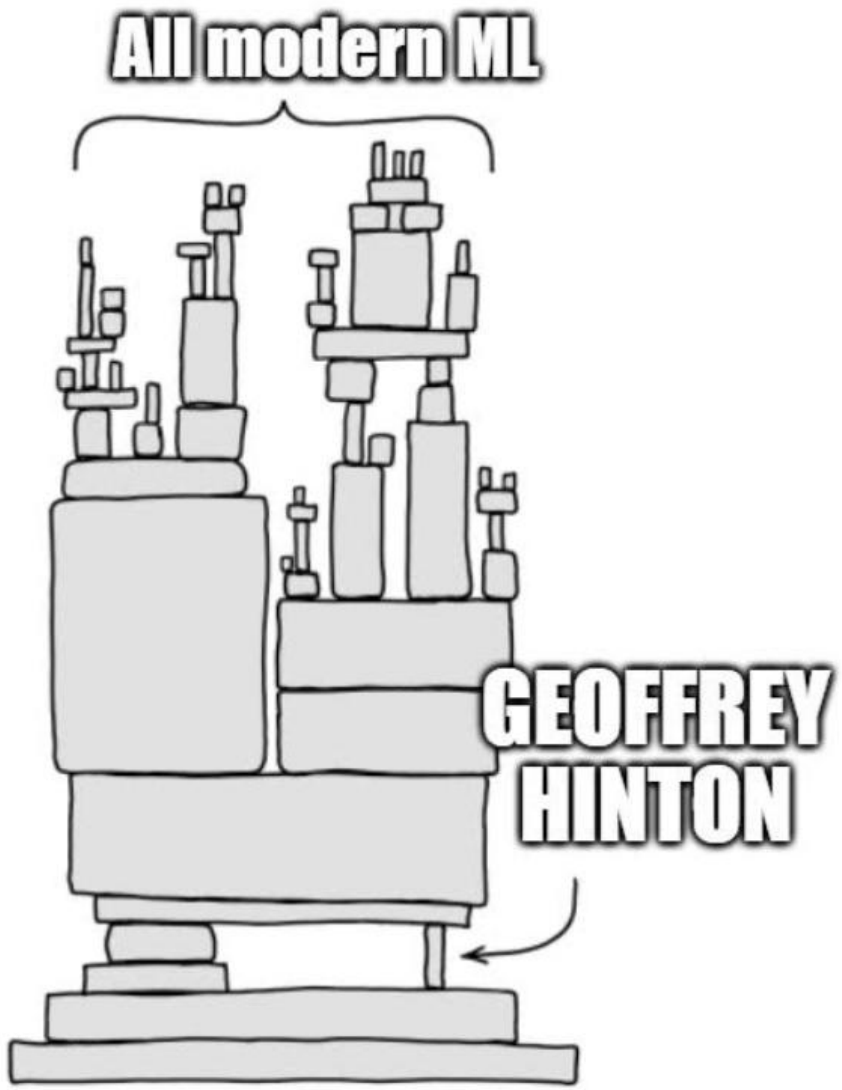
Definition (Backpropagation):

Backpropagation is an **algorithm** used to **train** neural networks.

It is a type of **gradient descent** (what a surprise!) algorithm that allows the network to learn by **adjusting the weights** of the connections between the neurons in the network.

Introduced by **Hinton** and **Rumelhart** [Rumelhart1986].

(Or was it? [Medium2020])



The backpropagation mechanism

Definition (Backpropagation):

Backpropagation is an **algorithm** used to **train** neural networks.

It is a type of **gradient descent** (what a surprise!) algorithm that allows the network to learn by **adjusting the weights** of the connections between the neurons in the network.

Introduced by **Hinton** and **Rumelhart** [Rumelhart1986].

(Or was it? [Medium2020])

Backpropagation will tell our model **how to adjust** the W and b matrices in each layer, to improve the neural network model loss and prediction capabilities.

It will be implemented in the **backward()** method of our model.

Backpropagation is heavy to manually calculate and implement…

Strongly encouraging to try and redo the math/implementation shown hereafter on your own!

That's the best way to learn and you can always refer to these slides for answers!

After math is correct, have a look at how to implement it in a backward method (also shown in slides, but probably will be shown rapidly in class!)

Backpropagation in our model

While our Neural Network seems to be complicated, at the end of the day, training it consists of solving a yet another **optimization problem**.

$$W_1^*, b_1^*, W_2^*, b_2^* = \arg \min_{W_1, b_1, W_2, b_2} \left[\frac{1}{M} \left(Y_{pred}(X, W_1, b_1, W_2, b_2) - Y \right)^2 \right]$$

$$W_1^*, b_1^*, W_2^*, b_2^* = \arg \min_{W_1, b_1, W_2, b_2} \left[\frac{1}{M} (W_2(W_1 X + b_1) + b_2 - Y)^2 \right]$$

With M being the number of samples in the dataset. We will use **gradient descent**, like we did for the Linear Regression, except that this time we have 4 parameters being matrices.

Before we continue

Before we continue, a quick note about square matrices.

$$W_1^*, b_1^*, W_2^*, b_2^* = \arg \min_{W_1,b_1,W_2,b_2} \left[\frac{1}{M} \left(Y_{pred}(X, W_1, b_1, W_2, b_2) - Y \right)^2 \right]$$

$$W_1^*, b_1^*, W_2^*, b_2^* = \arg \min_{W_1,b_1,W_2,b_2} \left[\frac{1}{M} (W_2(W_1X + b_1) + b_2 - Y)^2 \right]$$

Note: The notation $(A)^2$ is somewhat abusive when A is a matrix, the correct way would be to write $A.A^T$.

But in the formula above, that would lead to a very large formula (too large for slides!)

Chain rule, to the rescue!

First, let us denote the error term ϵ.

$$\epsilon = Y_{pred}(X, W_1, b_1, W_2, b_2) - Y = W_2(W_1 X + b_1) + b_2 - Y.$$

Using the chain rule, we can simply compute $\frac{\partial L}{\partial W_2}$ as

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial Y_{pred}} \frac{\partial Y_{pred}}{\partial W_2}$$

Chain rule, to the rescue!

Since we have

$$\frac{\partial L}{\partial Y_{pred}} = \frac{2}{M} Y_{pred}(X, W_1, b_1, W_2, b_2) - Y = \frac{2\epsilon}{M}$$

And

$$\frac{\partial Y_{pred}}{\partial W_2} = W_1 X + b_1$$

The gradient descent update rule for W_2 is then defined as:

$$W_2 \leftarrow W_2 - \frac{2\epsilon\alpha}{M}(W_1 X + b_1)$$

Chain rule, to the rescue!

We have just computed the formula for the gradient descent update rule for W_2, which was defined as:

$$W_2 \leftarrow W_2 - \frac{2\epsilon\alpha}{M}(W_1X + b_1)$$

Similarly, we can define the gradient descent update rule for b_2 as:

$$b_2 \leftarrow b_2 - \frac{2\epsilon\alpha}{M}$$

Chain rule, to the rescue!

Similarly, we have:

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial Y_{pred}} \frac{\partial Y_{pred}}{\partial W_1} = \left(\frac{2\epsilon}{M}\right)(W_2 X)$$

$$And\ \frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial Y_{pred}} \frac{\partial Y_{pred}}{\partial b_1} = \left(\frac{2\epsilon}{M}\right)(W_2)$$

And therefore, we have:

$$W_1 \leftarrow W_1 - \frac{2\epsilon\alpha}{M} W_2 X$$

$$b_1 \leftarrow b_1 - \frac{2\epsilon\alpha}{M} W_2$$

Now that we have all update rules…

We have figured the four update rules to use for backpropagation…

$$W_2 \leftarrow W_2 - \frac{2\epsilon\alpha}{M}(W_1 X + b_1)$$

$$b_2 \leftarrow b_2 - \frac{2\epsilon\alpha}{M}$$

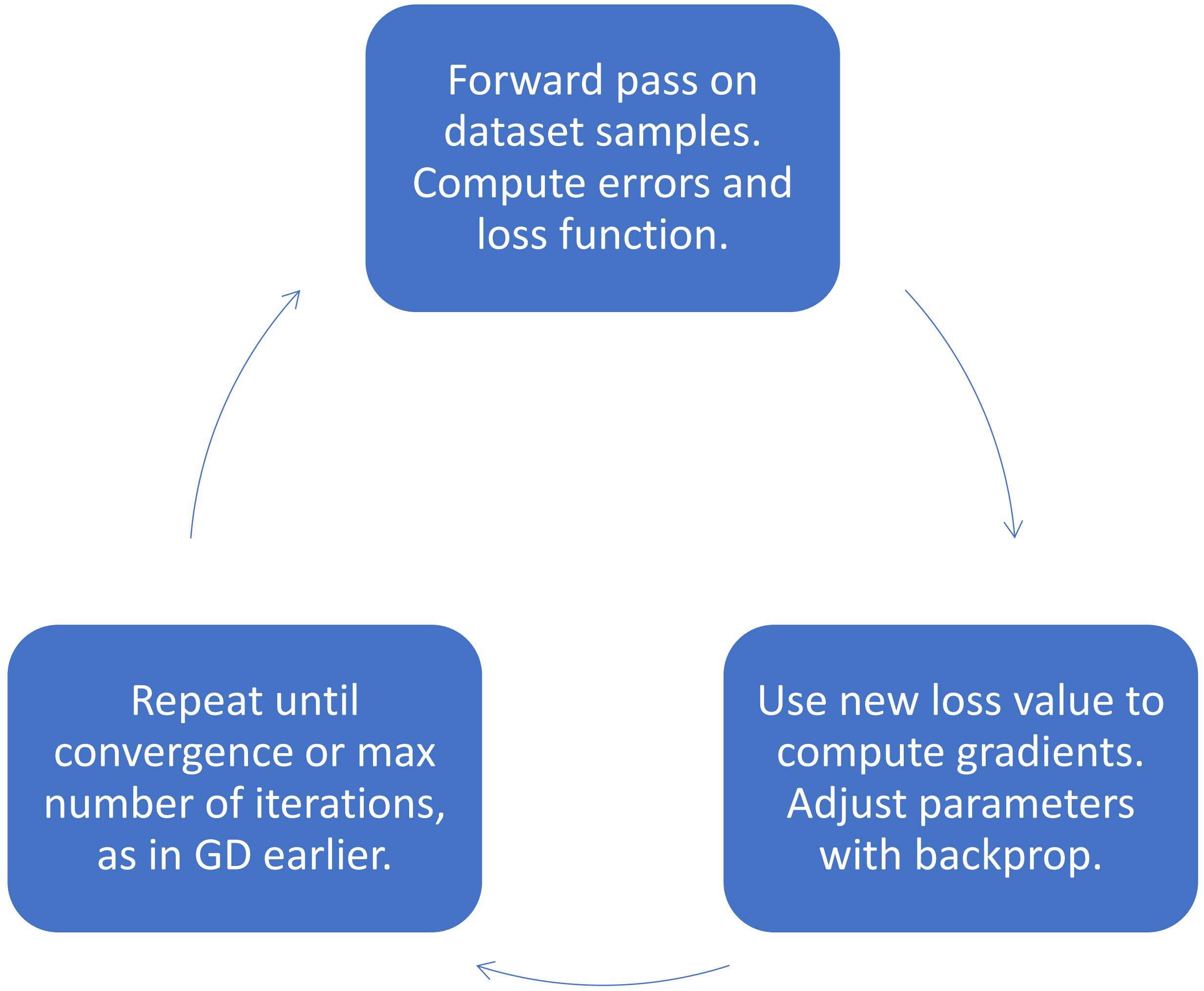
$$W_1 \leftarrow W_1 - \frac{2\epsilon\alpha}{M} W_2 X$$

$$b_1 \leftarrow b_1 - \frac{2\epsilon\alpha}{M} W_2$$

Now it is about implementing two additional methods for our Neural Network class:

1. **Backward method:** It performs the calculation of gradients, in matrix form, and parameters adjustments using the gradient descent update rules shown on the left.
2. **Trainer method:** It repeats the backward method, reusing the gradient descent for-loop function from earlier in Notebook 2, until a maximal number of iterations is reached or convergence is seen.

Training procedure, in short.


Forward pass on dataset samples. Compute errors and loss function.
Use new loss value to compute gradients. Adjust parameters with backprop.
Repeat until convergence or max number of iterations, as in GD earlier.

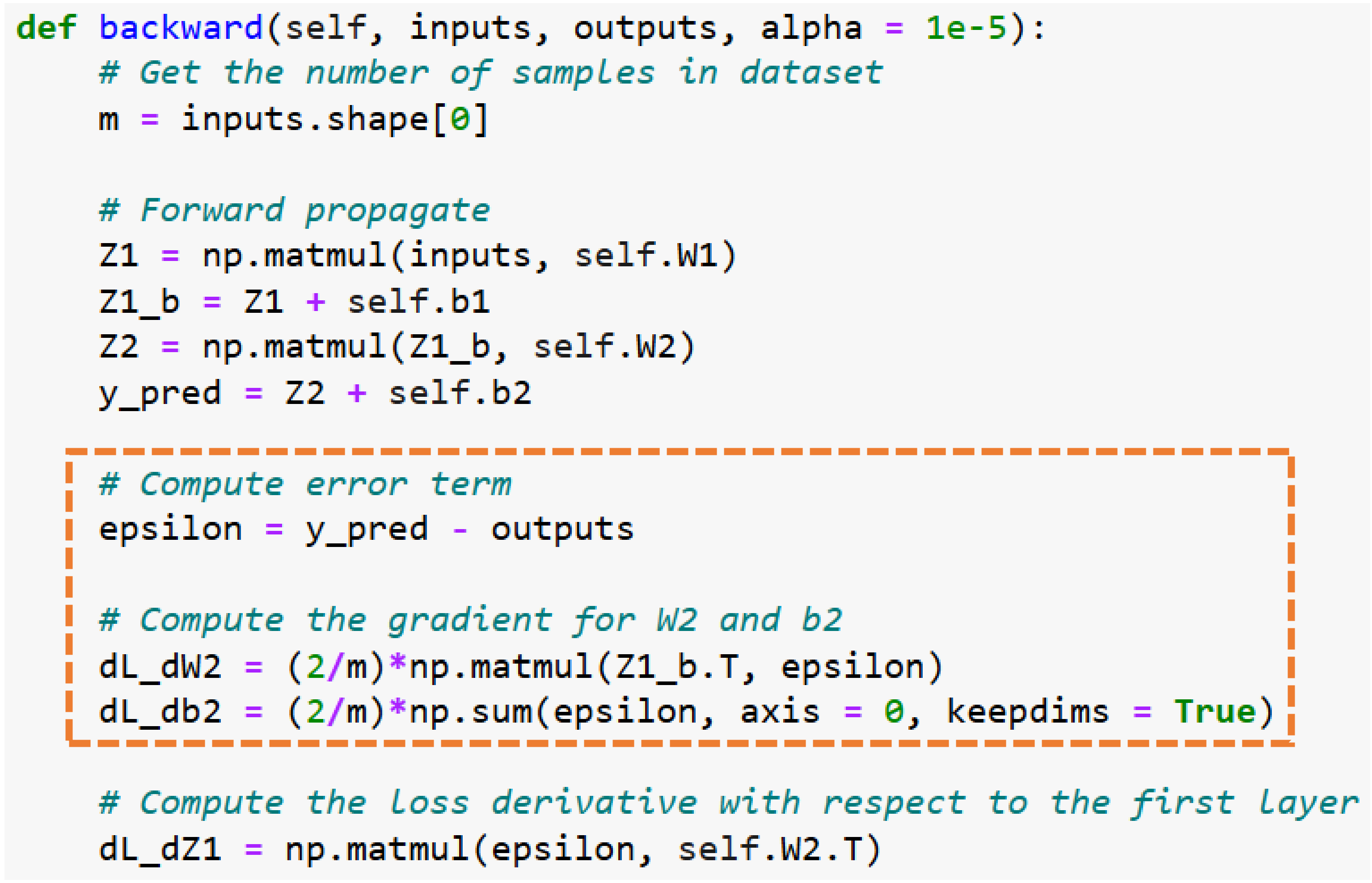
Backpropagation in our model

Update rules for W_2 and b_2

$$\epsilon = W_2(W_1X + b_1) + b_2 - Y$$
$$= Y_{pred} - Y$$

$$W_2 \leftarrow W_2 - \frac{2\epsilon\alpha}{M}(W_1X + b_1)$$

$$b_2 \leftarrow b_2 - \frac{2\epsilon\alpha}{M}$$



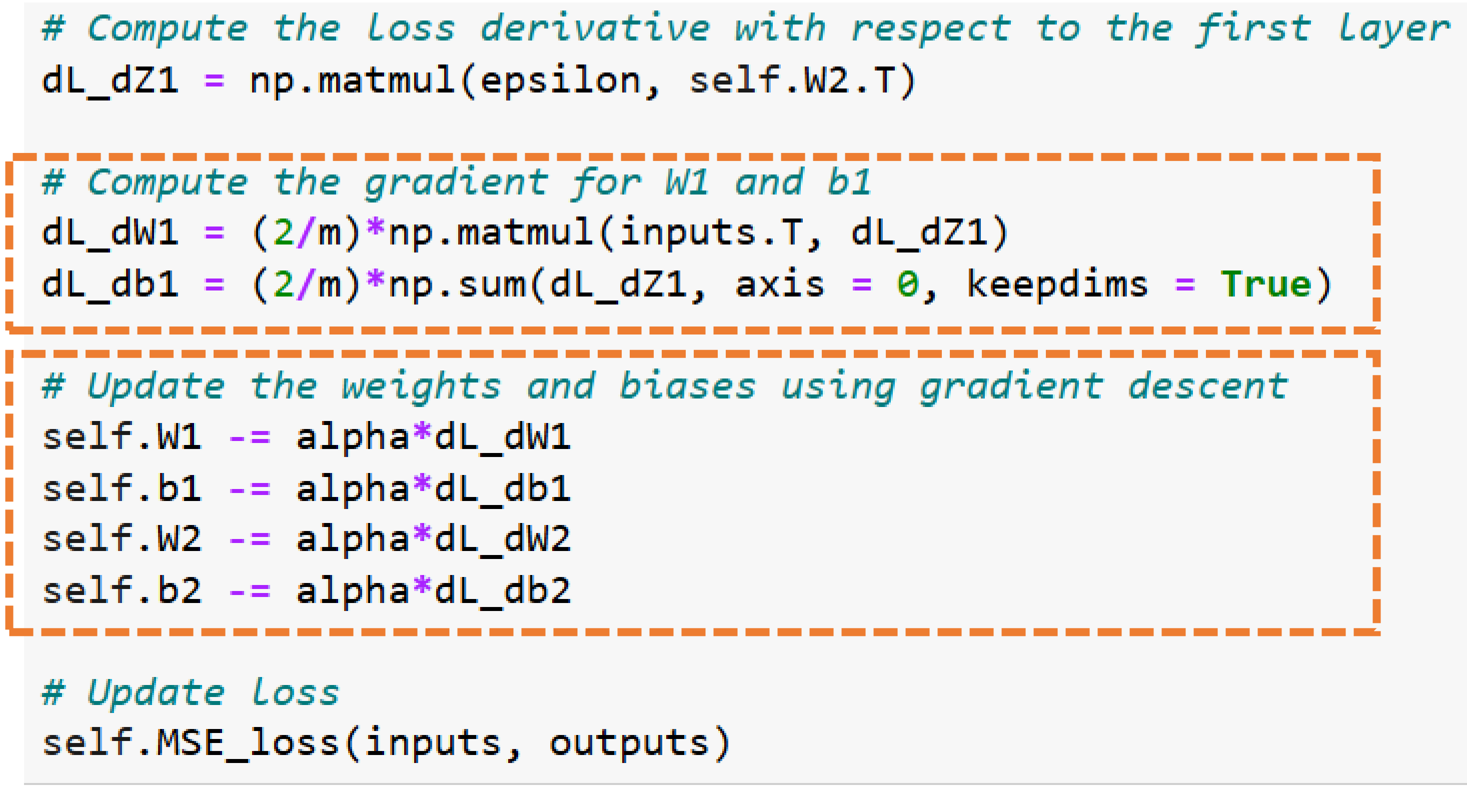
```
def backward(self, inputs, outputs, alpha = 1e-5):
    # Get the number of samples in dataset
    m = inputs.shape[0]

    # Forward propagate
    Z1 = np.matmul(inputs, self.W1)
    Z1_b = Z1 + self.b1
    Z2 = np.matmul(Z1_b, self.W2)
    y_pred = Z2 + self.b2

    # Compute error term
    epsilon = y_pred - outputs

    # Compute the gradient for W2 and b2
    dL_dW2 = (2/m)*np.matmul(Z1_b.T, epsilon)
    dL_db2 = (2/m)*np.sum(epsilon, axis = 0, keepdims = True)

    # Compute the loss derivative with respect to the first layer
    dL_dZ1 = np.matmul(epsilon, self.W2.T)
```

Backpropagation in our model

Update rules for W_1 and b_1

$$W_1 \leftarrow W_1 - \frac{2\epsilon\alpha}{M} W_2 X$$

$$b_1 \leftarrow b_1 - \frac{2\epsilon\alpha}{M} W_2$$

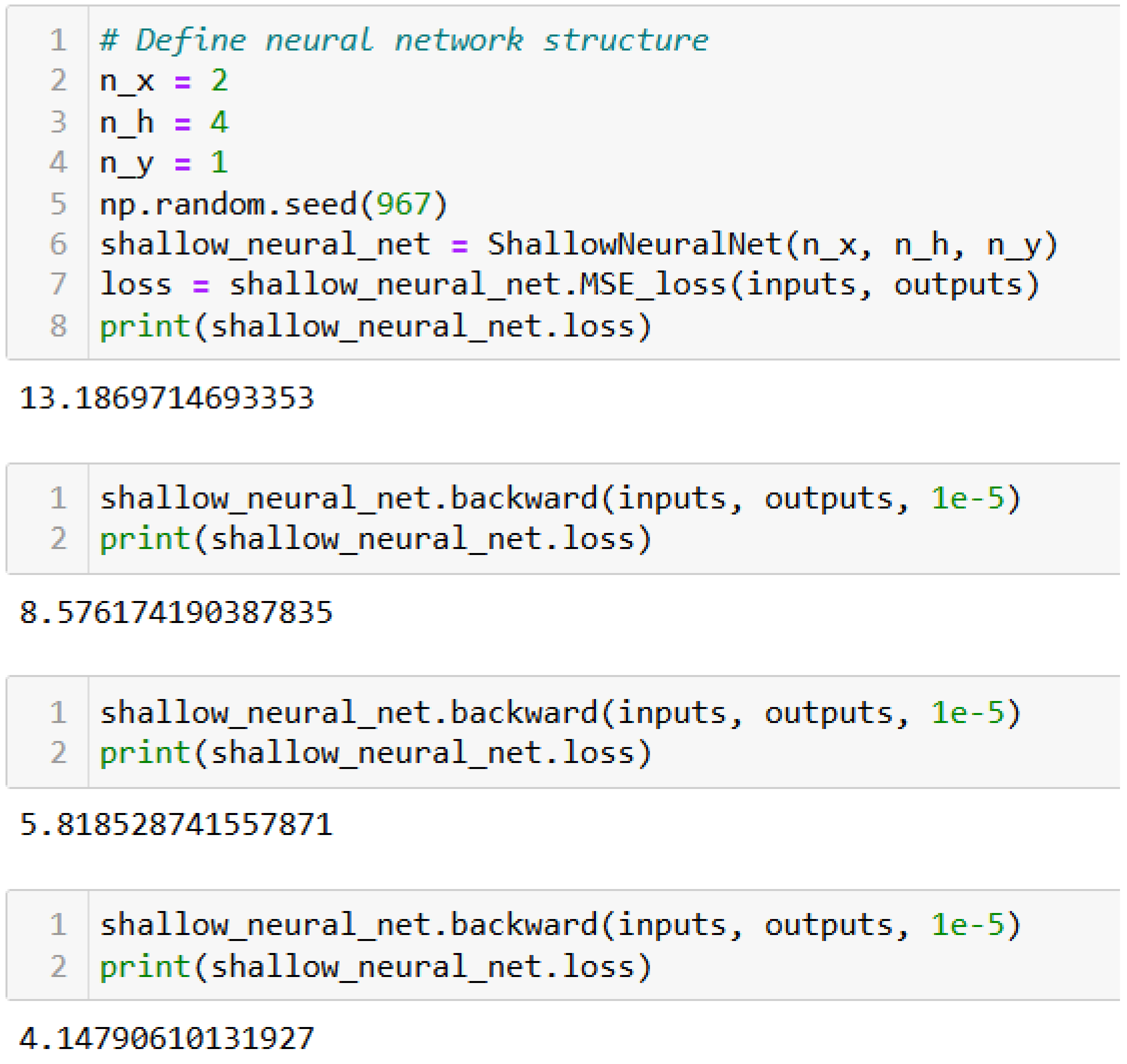


```
# Compute the loss derivative with respect to the first layer
dL_dZ1 = np.matmul(epsilon, self.W2.T)

# Compute the gradient for W1 and b1
dL_dW1 = (2/m)*np.matmul(inputs.T, dL_dZ1)
dL_db1 = (2/m)*np.sum(dL_dZ1, axis = 0, keepdims = True)

# Update the weights and biases using gradient descent
self.W1 -= alpha*dL_dW1
self.b1 -= alpha*dL_db1
self.W2 -= alpha*dL_dW2
self.b2 -= alpha*dL_db2

# Update loss
self.MSE_loss(inputs, outputs)
```

Backpropagation in our model

Effect of the backpropagation

- Every time we call the backward() method, our model will update and improve its parameters W_1, W_2, b_1 and b_2.
- After every call of backward(), our model seems to become better at the task in question, shown by improvements in loss values, which are decreasing after each iteration of the backward() method.



```
# Define neural network structure
n_x = 2
n_h = 4
n_y = 1
np.random.seed(967)
shallow_neural_net = ShallowNeuralNet(n_x, n_h, n_y)
loss = shallow_neural_net.MSE_loss(inputs, outputs)
print(shallow_neural_net.loss)
```

```
13.1869714693353
```

```
shallow_neural_net.backward(inputs, outputs, 1e-5)
print(shallow_neural_net.loss)
```

```
8.576174190387835
```

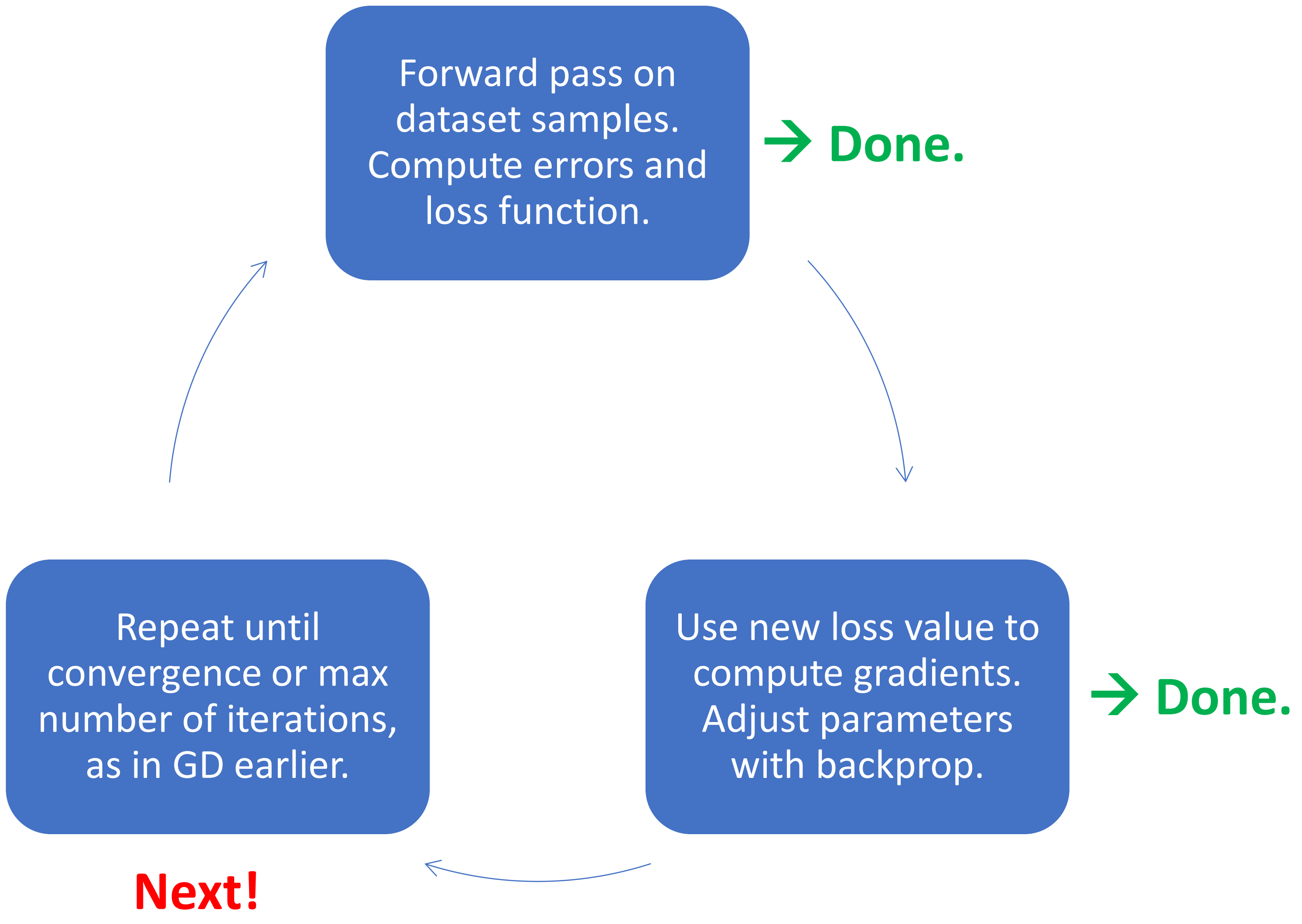
```
shallow_neural_net.backward(inputs, outputs, 1e-5)
print(shallow_neural_net.loss)
```

```
5.818528741557871
```

```
shallow_neural_net.backward(inputs, outputs, 1e-5)
print(shallow_neural_net.loss)
```

```
4.14790610131927
```

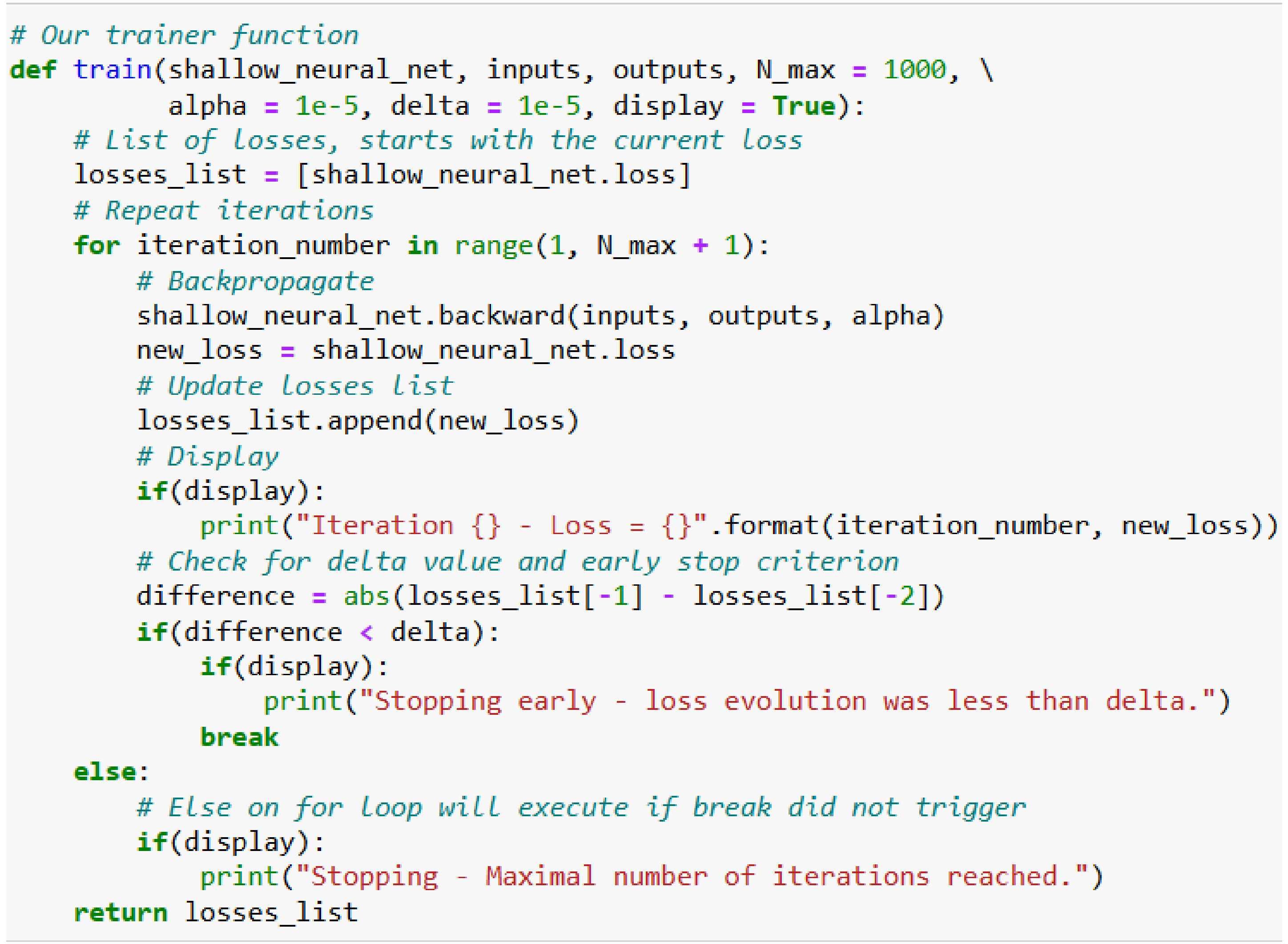
Training procedure, in short (reminder).


Forward pass on dataset samples. Compute errors and loss function.
→ Done.
Use new loss value to compute gradients. Adjust parameters with backprop.
→ Done.
Repeat until convergence or max number of iterations, as in GD earlier.
Next!

Trainer function for our model

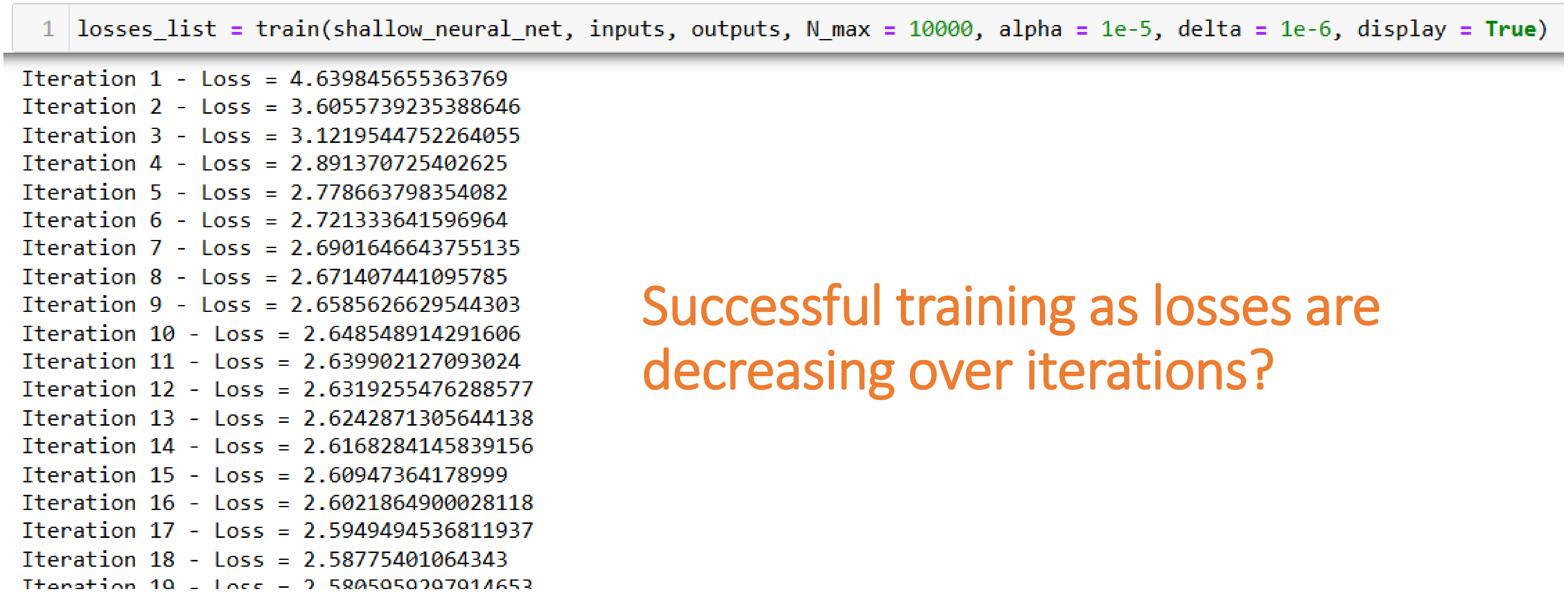
Train function implementation

- Iterate the backward() method for a given number of iterations $N_{\max}$.
- Display new loss values and append them in a list for display later.
- Also implemented an early stopping, which will break for loop if no change in parameters during an iteration of backward().
- Similar to GD in Notebook 2!



```
# Our trainer function
def train(shallow_neural_net, inputs, outputs, N_max = 1000, \
          alpha = 1e-5, delta = 1e-5, display = True):
    # List of losses, starts with the current loss
    losses_list = [shallow_neural_net.loss]
    # Repeat iterations
    for iteration_number in range(1, N_max + 1):
        # Backpropagate
        shallow_neural_net.backward(inputs, outputs, alpha)
        new_loss = shallow_neural_net.loss
        # Update losses list
        losses_list.append(new_loss)
        # Display
        if(display):
            print("Iteration {} - Loss = {}".format(iteration_number, new_loss))
        # Check for delta value and early stop criterion
        difference = abs(losses_list[-1] - losses_list[-2])
        if(difference < delta):
            if(display):
                print("Stopping early - loss evolution was less than delta.")
            break
    else:
        # Else on for loop will execute if break did not trigger
        if(display):
            print("Stopping - Maximal number of iterations reached.")
    return losses_list
```

Trainer function for our model



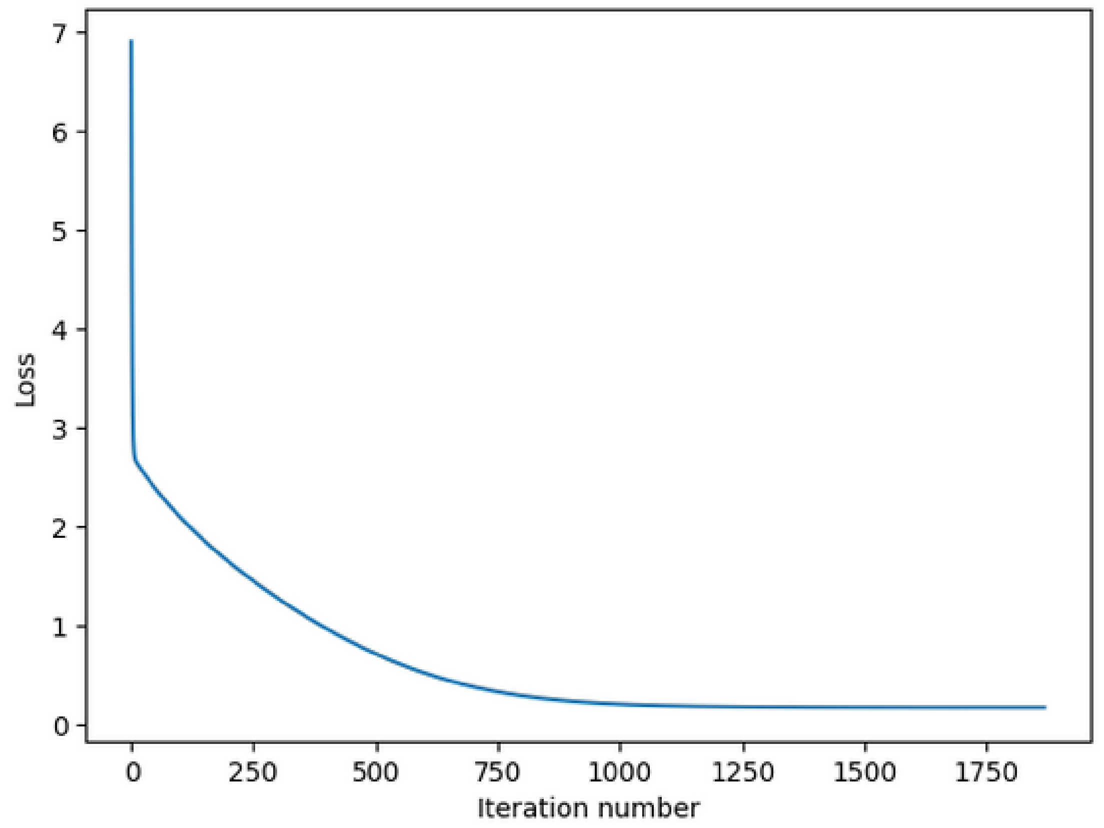
```
losses_list = train(shallow_neural_net, inputs, outputs, N_max = 10000, alpha = 1e-5, delta = 1e-6, display = True)
```

```
Iteration 1 - Loss = 4.639845655363769
Iteration 2 - Loss = 3.6055739235388646
Iteration 3 - Loss = 3.1219544752264055
Iteration 4 - Loss = 2.891370725402625
Iteration 5 - Loss = 2.778663798354082
Iteration 6 - Loss = 2.721333641596964
Iteration 7 - Loss = 2.6901646643755135
Iteration 8 - Loss = 2.671407441095785
Iteration 9 - Loss = 2.6585626629544303
Iteration 10 - Loss = 2.648548914291606
Iteration 11 - Loss = 2.639902127093024
Iteration 12 - Loss = 2.6319255476288577
Iteration 13 - Loss = 2.6242871305644138
Iteration 14 - Loss = 2.6168284145839156
Iteration 15 - Loss = 2.60947364178999
Iteration 16 - Loss = 2.6021864900028118
Iteration 17 - Loss = 2.5949494536811937
Iteration 18 - Loss = 2.58775401064343
Iteration 19 - Loss = 2.5805959297914653
```

Successful training as losses are decreasing over iterations?

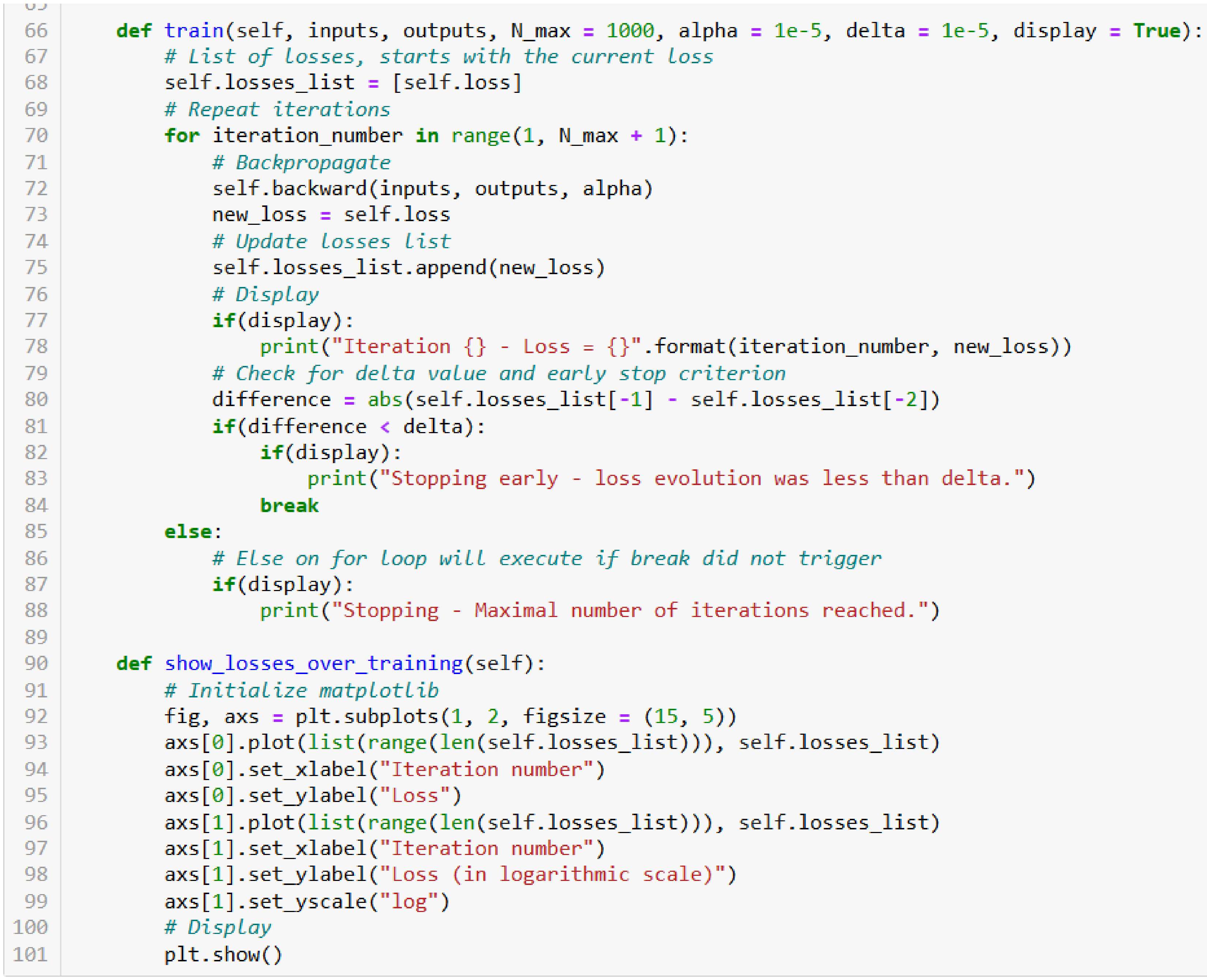
Trainer function for our model

The curves below are called **training curves** and are useful to show how the training went (make an extra viz method to be used after training?).



Loss has decreased and converged.
Confirmed training?

Finally, we will make the trainer function and the performance curves function into methods for our class.



```python
    def train(self, inputs, outputs, N_max = 1000, alpha = 1e-5, delta = 1e-5, display = True):
        # List of losses, starts with the current loss
        self.losses_list = [self.loss]
        # Repeat iterations
        for iteration_number in range(1, N_max + 1):
            # Backpropagate
            self.backward(inputs, outputs, alpha)
            new_loss = self.loss
            # Update losses list
            self.losses_list.append(new_loss)
            # Display
            if(display):
                print("Iteration {} - Loss = {}".format(iteration_number, new_loss))
            # Check for delta value and early stop criterion
            difference = abs(self.losses_list[-1] - self.losses_list[-2])
            if(difference < delta):
                if(display):
                    print("Stopping early - loss evolution was less than delta.")
                break
        else:
            # Else on for loop will execute if break did not trigger
            if(display):
                print("Stopping - Maximal number of iterations reached.")

    def show_losses_over_training(self):
        # Initialize matplotlib
        fig, axs = plt.subplots(1, 2, figsize = (15, 5))
        axs[0].plot(list(range(len(self.losses_list))), self.losses_list)
        axs[0].set_xlabel("Iteration number")
        axs[0].set_ylabel("Loss")
        axs[1].plot(list(range(len(self.losses_list))), self.losses_list)
        axs[1].set_xlabel("Iteration number")
        axs[1].set_ylabel("Loss (in logarithmic scale)")
        axs[1].set_yscale("log")
        # Display
        plt.show()
```

Backpropagation is heavy to manually calculate and implement…
Strongly encouraging to try and redo the math/implementation shown on your own!

That’s the best way to learn and you can always refer to these slides for answers!

After math is correct, have a look at how to implement it in a backward method (also shown in slides, but probably will be shown rapidly in class!)

Remember, you have no homework this week, so let us call that your homework!

Conclusion (Day 1)

- Reminders of Machine Learning
- Linear
- Using a normal equation to train a linear regression
- Gradient descent as a training procedure for linear regression
- Polynomial Regression
- Regularization in Ridge/Lasso Regression
- Train-test split
- Overfitting and underfitting
- Generalization
- Sigmoid and Logistic functions
- From linear regression to logistic regression
- Using logistic regression for binary classification

Conclusion (Day 1)

- Implementing a shallow neural network in Numpy
- Forward propagation method, to formulate predictions
- The backpropagation mechanism, as the gradient descent on Neural Network
- Backward method for training and trainer functions to iterate backward iterations
- Performance/training curves

Next week?

- Initializations to break symmetries
- Exploding and vanishing gradients
- Activation functions and non-linearities in Neural Networks
- Advanced optimizers
- Validation sets, early stopping, saver and loader functions

Learn more about these topics

Out of class, supporting papers, for those of you who are curious.

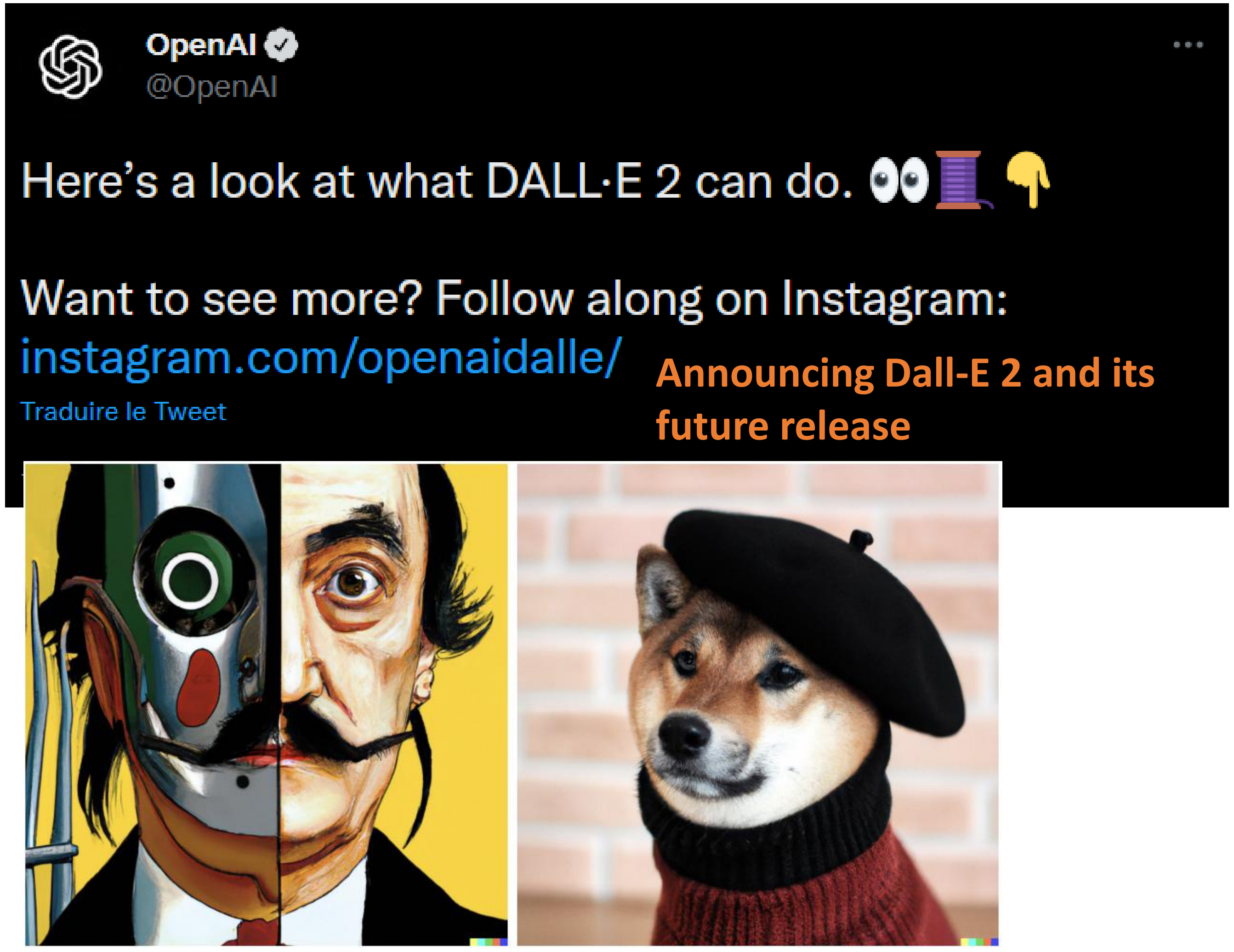
- [Géron2019] A. Géron , “How Neural Networks Work”, 2019.
- [Yamins2016] Yamins et al., “Deep neural networks are robust computational models of the human visual system”, 2016.
- [Kriegeskorte2013] Kriegeskorte and Kievit, “Neural Network Models of the Human Brain”, 2013.
- [Rumelhart1986] D.E. **Rumelhart**, G. **Hinton**, Williams, “Learning representations by back-propagating errors“, 1986. https://www.nature.com/articles/323533a0

Learn more about these topics

Keep track of important names and follow their social media (Follow them on Scholar, Twitter, or whatever works for you!)

- **Geoffrey Hinton: Emeritus Professor** at **University of Toronto**, **one of the three Godfathers of Deep Learning** and **2018 Turing Award** winner (highest distinction in Computer Science), Nobel Prize Physics 2025. Formerly, Google. https://www.cs.toronto.edu/~hinton/ https://scholar.google.co.uk/citations?user=JicYPdAAAAAJ&hl=en
- **David Rumelhart: Former professor** at **University of California**, credited for inventing **backpropagation along** with **Hinton**. Passed away in 2011. https://www.nytimes.com/2011/03/19/health/19rumelhart.html

Twitter, the theater for AI/DL drama and announcements



OpenAI
@OpenAI

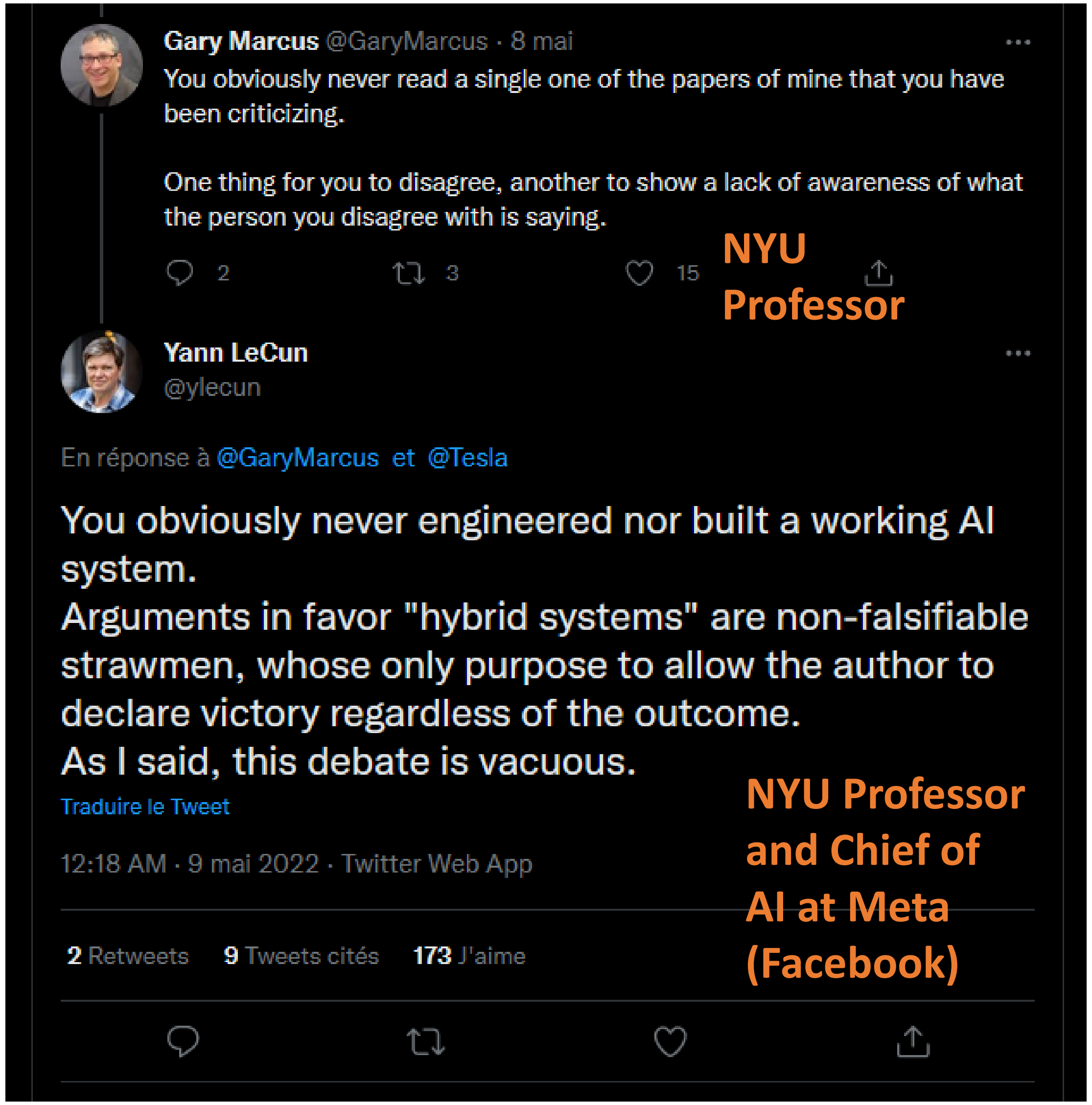
Here's a look at what DALL·E 2 can do. 👀🧵👇

Want to see more? Follow along on Instagram: instagram.com/openaidalle/
Traduire le Tweet

vibrant portrait painting of Salvador Dalí with a robotic half face

a shiba inu wearing a beret and black turtleneck

Announcing Dall-E 2 and its future release



Gary Marcus @GaryMarcus · 8 mai
You obviously never read a single one of the papers of mine that you have been criticizing.

One thing for you to disagree, another to show a lack of awareness of what the person you disagree with is saying.

2 3 15

NYU Professor

Yann LeCun
@ylecun

En réponse à @GaryMarcus et @Tesla

You obviously never engineered nor built a working AI system.
Arguments in favor "hybrid systems" are non-falsifiable strawmen, whose only purpose to allow the author to declare victory regardless of the outcome.
As I said, this debate is vacuous.
Traduire le Tweet

12:18 AM · 9 mai 2022 · Twitter Web App

2 Retweets 9 Tweets cités 173 J'aime

NYU Professor and Chief of AI at Meta (Facebook)

Learn more about these topics

Some extra (easy) reading and videos for those of you who are curious.

- [Quanta2021] “Artificial Neural Nets Finally Yield Clues to How Brains Learn“, 2021. https://www.quantamagazine.org/artificial-neural-nets-finally-yield-clues-to-how-brains-learn-20210218/
- [MITNews2022] “Study urges caution when comparing neural networks to the brain”, 2022. https://news.mit.edu/2022/neural-networks-brain-function-1102
- [Medium2020] “Who Invented Backpropagation? Hinton Says He Didn’t, but His Work Made It Popular“, 2020. https://medium.com/syncedreview/who-invented-backpropagation-hinton-says-he-didnt-but-his-work-made-it-popular-e0854504d6d1/

Let us call it finish with a quiz

Day 1 Knowledge check!

Day 1 Knowledge Check Quiz

- As part of your assessment, please take the time to answer the following knowledge check/quiz below.

- **Deadline: XXXX**