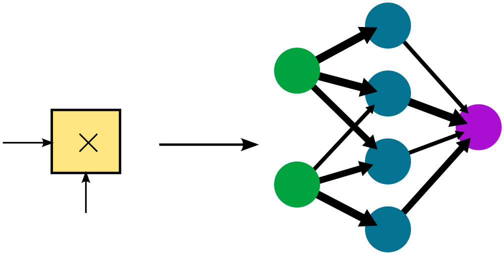
From operations to neural network

Principles of object-oriented programming & neural networks

Gökçe Aydos



Learning goals

Learning goals

- apply object-oriented programming principles to implement a neural network
- understand the components of a neural network

Fundamentals

income ($)	house-age (years)	…	house-value ($)
83252	41	…	452600
83014	21	…	358500
…	…	…	…

income ($)	house-age (years)	...	house-value ($)
83252	41	...	452600
83014	21	...	358500
...	...	...	...

Goal learn weights $w_1, w_2, \ldots$ such that:

$$w_1 \cdot 83252 + w_2 \cdot 41 + ... \approx 452600$$
$$w_1 \cdot 83014 + w_2 \cdot 21 + ... \approx 358500$$

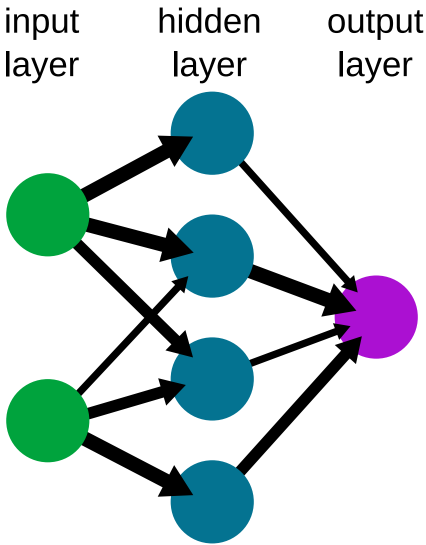


Figure 1: a neural network with a single hidden layer

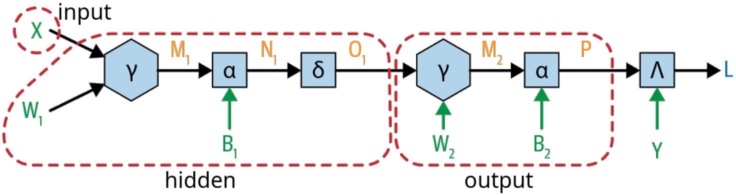


Figure 2: Components of layers and loss

Implementation

Goal: Implementation using object-oriented programming (OOP)

Class ideas?

Goal: Implementation using object-oriented programming (OOP)

Class ideas?

- `Operation`
- `ParameterOperation`
- `Layer`
- `NeuralNetwork`
- `Optimizer`
- `Trainer`

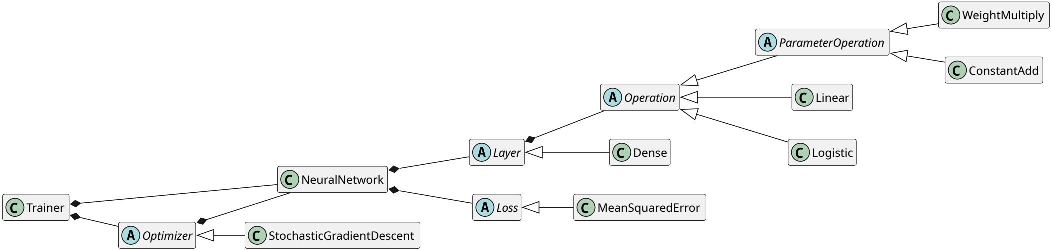


Figure 3: Implementation example

Code

…

Questions to ponder 🤔

1. Does it make sense to inherit `Loss` from `Operation`?
2. What happens if we use no activation function (a linear activation)? Try with single and many layers.
3. What happens if we take the the sum of the errors as a loss function?
4. What happens if we don't standardize the features before we use them for training?
5. What happens if we choose a learning rate of 1?

Takeaways

- `breakpoint()` is useful for debugging while interacting with the program in `ipython`
- many bugs through Numpy broadcasting, etc scalar multiplying a `(3, 1)` array with a `(3,)` array. Assertions help.

References

- Code
- 2019, Weidman, Deep Learning from Scratch
  - there are occasional mistakes in the book, refer to the errata in doubt
  - German version

Appendix

x	y
1	2
2	4
3	6
4	?
-1	?

x	y
1	1.99
2	4.02
3	5.98
4	?
-1	?

x_1	x_2	y
1	1	1.5
2	1	2.5
2	2	3
3	1	?
-1	2	?

❓ Assume the machine learned some w_is. Are the predictions correct? How do we test?

❓ Assume the machine learned some w_is. Are the predictions correct? How do we test?

❗ We can compare each prediction p_i (using w_is) with the actual house value (y_i).

❓ How do we test the prediction quality using math?

❓ How do we test the prediction quality using math?

❗ For example by using *mean squared error* (MSE).

$$\frac{(y_1 - p_1)^2 + (y_2 - p_2)^2 + ... + (y_n - p_n)^2}{n}$$

MSE is - a *loss function* - measures how *erroneous* the prediction is

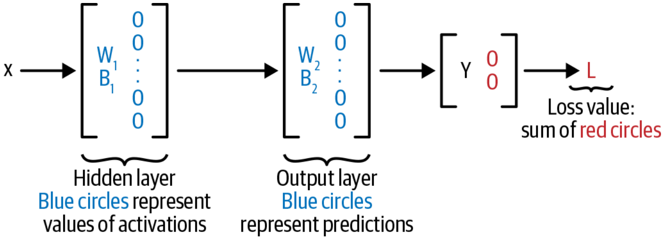
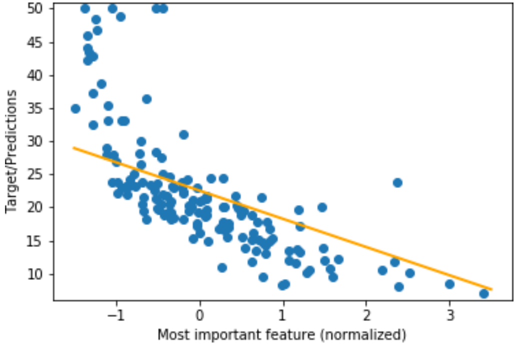
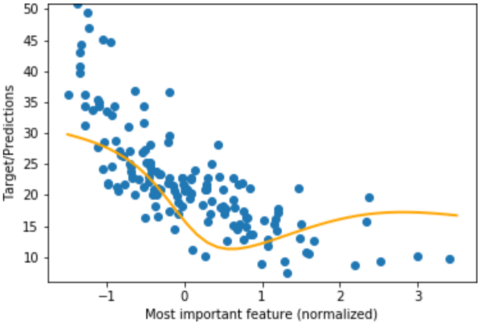


Figure 4: Alternative view to components

Linear vs non-linear activation





Goal: minimum loss by training:

- ▶ Pick random parameters (weights)
- ▶ Make predictions for a batch of inputs
- ▶ Compute loss
- ▶ Find the parameters (weights) that minimize the loss

❓ How can we find these parameters?

Goal: minimum loss by training:

- Pick random parameters (weights)
- Make predictions for a batch of inputs
- Compute loss
- Find the parameters (weights) that minimize the loss

❓ How can we find these parameters?

❗ Taking the partial derivative with respect to each parameter (*gradient*). However we typically cannot find the exact minimum, because the loss function can get very complex.

❓ What do we do now?

Alternative perspective follows:

❓ You are stuck on a mountain. How would you get down?

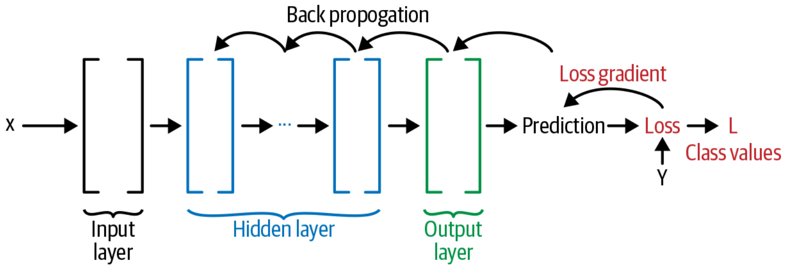
Approach: Find out how much we should change each parameter so that the loss decreases.

❓ How can we find out how much the loss L changes if we e.g., increase the parameter w_1 by 1?

Approach: Find out how much we should change each parameter so that the loss decreases.

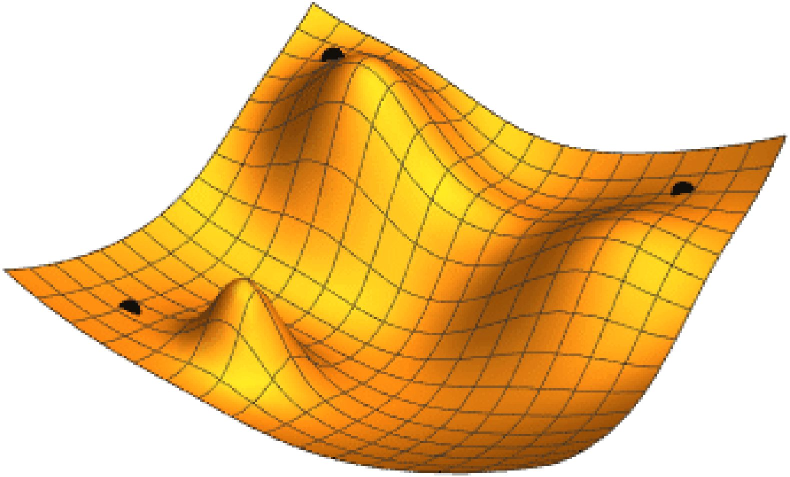
❓ How can we find out how much the loss L changes if we e.g., increase the parameter w_1 by 1?

❗ By computing $\frac{\partial L}{\partial w_1}(w_1 = 1)$. If the result is positive, then we decrease w_1 and vice-versa. This is called *back-propagation*.



Gradient descent II

Procedure: Pick a random point, move in direction of the descending path:



Revised algorithm

Goal: minimum loss by training:

- Pick random parameters (weights)
- Repeat:
  - Make predictions for a batch of inputs
  - Compute loss
  - Back-propagate
  - Modify the parameters so that the loss decreases a bit
  - Stop if the loss does not decrease significantly or after a timeout ##

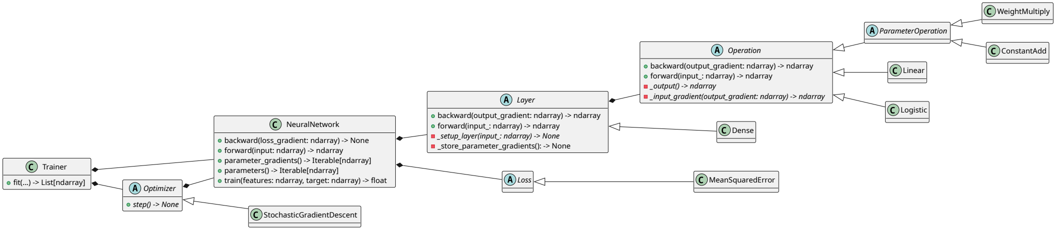


Figure 5: Implementation example with methods

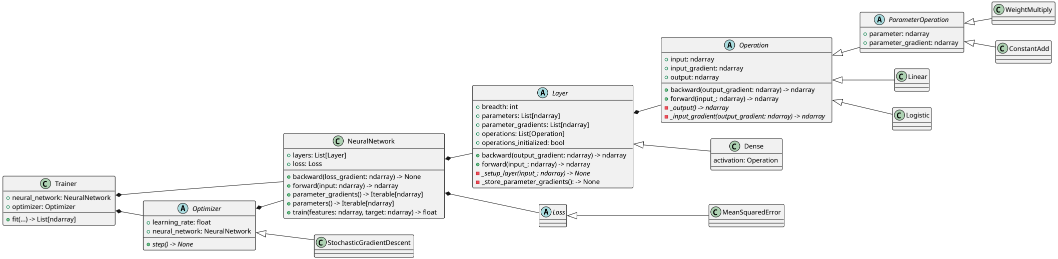


Figure 6: Implementation example with every component

Why OOP?

- encapsulation of features in a single component — more convenient for humans to classify components of a program
- reusability of components, extensibility