experiments

November 8, 2023

[186]:
```
import numpy as np

import ann
from activation_functions import sigmoid
from cost_functions import squared_error, mean_squared_error
from number_generation import zeros, normal
import matplotlib.pyplot as plt
```

[187]:
```
training_examples = np.identity(8)
number_generation_kwargs = {'mu': 0, 'sigma': 0.1}
costs = []
```

Model 1 (1000 epochs)

[188]:
```
ann_model_1 = ann.ANN(8, [3], 8, sigmoid, squared_error, normal, zeros,␣
 ↪add_bias=True, **number_generation_kwargs)
ann_model_1.train(training_examples, training_examples, 1000, 0.1)
```

[189]:
```
predictions = ann_model_1.feed_forward(training_examples)
for i, prediction in enumerate(predictions):
    print(f"input: {training_examples[i]}, output: {np.around(prediction, 2)}")

cost = round(mean_squared_error(predictions, training_examples), 5)
costs.append(cost)
print(f"cost: {cost}")
```

```
input: [1. 0. 0. 0. 0. 0. 0. 0.], output: [0.15 0.12 0.12 0.1  0.14 0.1  0.2
0.11]
input: [0. 1. 0. 0. 0. 0. 0. 0.], output: [0.14 0.13 0.11 0.12 0.15 0.13 0.15
0.11]
input: [0. 0. 1. 0. 0. 0. 0. 0.], output: [0.14 0.13 0.2  0.17 0.13 0.17 0.19
0.15]
input: [0. 0. 0. 1. 0. 0. 0. 0.], output: [0.11 0.13 0.15 0.21 0.13 0.26 0.1
0.14]
input: [0. 0. 0. 0. 1. 0. 0. 0.], output: [0.15 0.14 0.12 0.12 0.18 0.13 0.19
0.11]
input: [0. 0. 0. 0. 0. 1. 0. 0.], output: [0.11 0.13 0.16 0.25 0.14 0.31 0.09
0.14]
input: [0. 0. 0. 0. 0. 0. 1. 0.], output: [0.18 0.14 0.15 0.09 0.17 0.08 0.31
```

```
0.13]
input: [0. 0. 0. 0. 0. 0. 0. 1.], output: [0.13 0.12 0.14 0.14 0.12 0.15 0.14
0.13]
cost: 1.56273
```

[190]:
```
for i, layer in enumerate(ann_model_1.network):
    print(f"layer {i + 1} weights: \n{np.around(layer.weights, 2)}")
    print(f"layer {i + 1} biases: \n{np.around(layer.bias, 2)}")
    print()
```

```
layer 1 weights:
[[ 1.    0.61 -0.69 -1.07  0.84 -1.41  1.21 -0.13]
 [ 0.43 -0.04  0.45 -0.1  -0.63 -0.58  0.2   0.68]
 [-0.08  0.44 -0.64  0.8   0.1   1.08 -1.42  0.18]]
layer 1 biases:
0

layer 2 weights:
[[ 0.43 -0.22 -0.78]
 [ 0.03 -0.74 -0.25]
 [-1.15 -0.08 -1.19]
 [-1.6  -0.65  0.28]
 [ 0.36 -1.26 -0.52]
 [-1.89 -1.05  0.66]
 [ 0.85 -0.29 -2.03]
 [-0.73  0.02 -0.5 ]]
layer 2 biases:
[[-1.55]
 [-1.41]
 [-0.55]
 [-0.78]
 [-1.06]
 [-0.53]
 [-0.88]
 [-1.3 ]]
```

Model 2 (10 000 epochs)

[191]:
```
ann_model_2 = ann.ANN(8, [3], 8, sigmoid, squared_error, normal, zeros,␣
 ↪add_bias=True, **number_generation_kwargs)
ann_model_2.train(training_examples, training_examples, 10000, 0.1)
```

[192]:
```
predictions = ann_model_2.feed_forward(training_examples)
for i, prediction in enumerate(predictions):
    print(f"input: {training_examples[i]}, output: {np.around(prediction, 2)}")

cost = round(mean_squared_error(predictions, training_examples), 5)
costs.append(cost)
print(f"cost: {cost}")
```

```
input: [1. 0. 0. 0. 0. 0. 0. 0.], output: [0.93 0.   0.   0.05 0.05 0.   0.06 0.
]
input: [0. 1. 0. 0. 0. 0. 0. 0.], output: [0.   0.93 0.05 0.05 0.   0.   0.06 0.
]
input: [0. 0. 1. 0. 0. 0. 0. 0.], output: [0.   0.04 0.91 0.   0.   0.11 0.
0.03]
input: [0. 0. 0. 1. 0. 0. 0. 0.], output: [0.05 0.05 0.   0.91 0.   0.   0.
0.06]
input: [0. 0. 0. 0. 1. 0. 0. 0.], output: [0.04 0.   0.   0.   0.91 0.11 0.
0.04]
input: [0. 0. 0. 0. 0. 1. 0. 0.], output: [0.   0.   0.05 0.   0.06 0.83 0.05 0.
]
input: [0. 0. 0. 0. 0. 0. 1. 0.], output: [0.04 0.03 0.   0.   0.   0.11 0.91 0.
]
input: [0. 0. 0. 0. 0. 0. 0. 1.], output: [0.   0.   0.06 0.05 0.06 0.   0.
0.93]
cost: 0.04051
```

[193]:
```python
for i, layer in enumerate(ann_model_2.network):
    print(f"layer {i + 1} weights: \n{np.around(layer.weights, 2)}")
    print(f"layer {i + 1} biases: \n{np.around(layer.bias, 2)}")
    print()
```

```
layer 1 weights:
[[ 3.06 -3.85 -4.   -3.29  3.82  3.63  3.39 -3.55]
 [-3.64 -3.39  3.87 -3.87  3.63  3.64 -3.98  2.48]
 [-3.53  3.01  2.95 -3.36 -3.73  4.06  3.62 -4.11]]
layer 1 biases:
0

layer 2 weights:
[[ 5.95 -6.33 -6.31]
 [-6.55 -6.28  5.94]
 [-5.64  5.52  5.06]
 [-5.77 -6.08 -5.8 ]
 [ 5.18  5.32 -5.31]
 [ 3.73  3.77  3.87]
 [ 5.22 -5.57  5.28]
 [-6.36  5.76 -6.88]]
layer 2 biases:
[[-2.74]
 [-2.68]
 [-7.76]
 [ 2.84]
 [-7.8 ]
 [-9.53]
 [-7.76]
 [-2.44]]
```

Weight interpretation: It looks like that in the first layer almost always at least one weight of a column is positive while the others are negative, the same goes for the weights in layer 2 but then row wise. This could mean that the model. makes outputs that should be 0 a bigger negative number and 1 a bigger positive number.

Model 3 (100 000 epochs)

[194]:
```
ann_model_3 = ann.ANN(8, [3], 8, sigmoid, squared_error, normal, zeros,␣
 ↪add_bias=True, **number_generation_kwargs)
ann_model_3.train(training_examples, training_examples, 100000, 0.1)
```

[195]:
```
predictions = ann_model_3.feed_forward(training_examples)
for i, prediction in enumerate(predictions):
    print(f"input: {training_examples[i]}, output: {np.around(prediction, 2)}")

cost = round(mean_squared_error(predictions, training_examples), 5)
costs.append(cost)
print(f"cost: {cost}")
```

```
input: [1. 0. 0. 0. 0. 0. 0. 0.], output: [0.98 0.02 0.   0.   0.01 0.   0.
0.02]
input: [0. 1. 0. 0. 0. 0. 0. 0.], output: [0.01 0.98 0.   0.01 0.   0.   0.02 0.
]
input: [0. 0. 1. 0. 0. 0. 0. 0.], output: [0.   0.   0.98 0.01 0.   0.01 0.02 0.
]
input: [0. 0. 0. 1. 0. 0. 0. 0.], output: [0.   0.01 0.01 0.98 0.02 0.   0.   0.
]
input: [0. 0. 0. 0. 1. 0. 0. 0.], output: [0.02 0.   0.   0.01 0.97 0.01 0.   0.
]
input: [0. 0. 0. 0. 0. 1. 0. 0.], output: [0.   0.   0.01 0.   0.01 0.98 0.
0.01]
input: [0. 0. 0. 0. 0. 0. 1. 0.], output: [0.   0.01 0.01 0.   0.   0.   0.97
0.01]
input: [0. 0. 0. 0. 0. 0. 0. 1.], output: [0.01 0.   0.   0.   0.   0.01 0.02
0.98]
cost: 0.00228
```

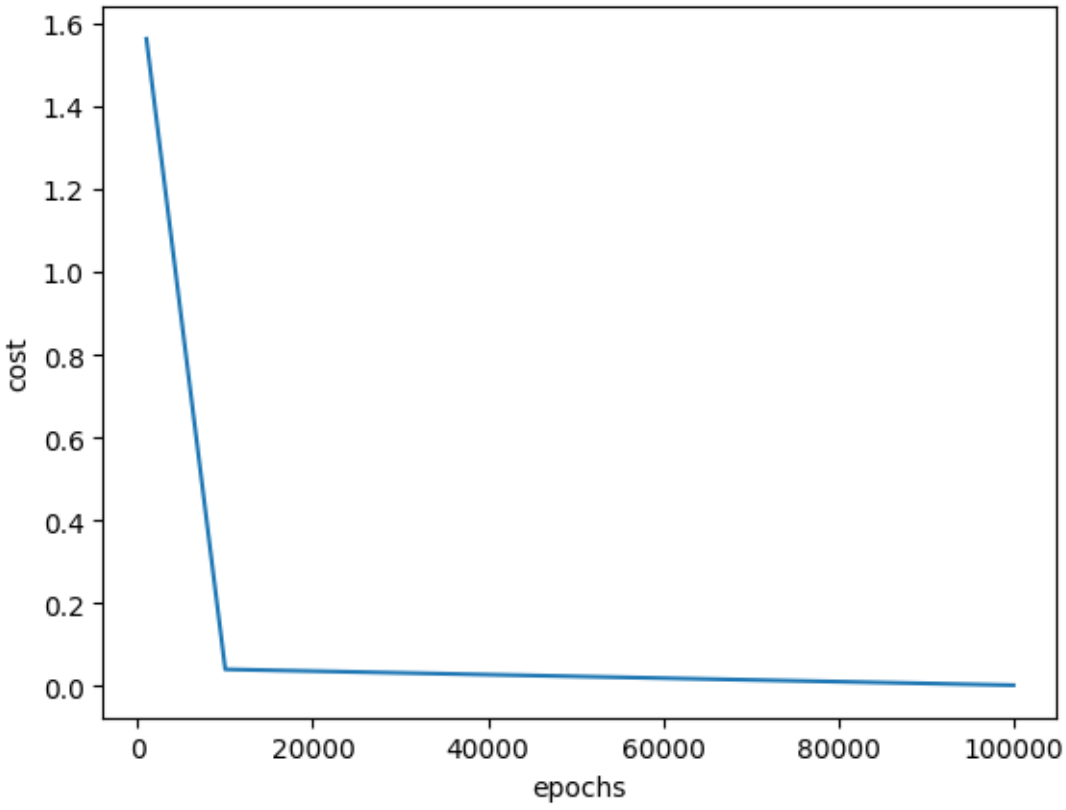
[196]:
```
for i, layer in enumerate(ann_model_3.network):
    print(f"layer {i + 1} weights: \n{np.around(layer.weights, 2)}")
    print(f"layer {i + 1} biases: \n{np.around(layer.bias, 2)}")
    print()
```

```
layer 1 weights:
[[-5.33  1.09  2.86  4.89 -1.48 -4.27  5.21 -4.83]
 [-4.77 -5.02  4.21 -4.02 -1.77  4.72  4.64  2.11]
 [ 0.44  4.95 -4.91 -1.24 -6.07 -1.57  4.88  4.93]]
layer 1 biases:
0
```

```
layer 2 weights:
[[-13.86 -11.82   5.73]
 [  4.31  -9.47  11.55]
 [  7.14   6.73  -9.48]
 [ 10.43  -9.92  -7.37]
 [ -6.19  -7.14 -16.3 ]
 [-10.64   9.44  -8.94]
 [  7.28   5.96   7.32]
 [ -8.92   4.54  10.26]]
layer 2 biases:
[[  0.36]
 [-10.9 ]
 [ -9.34]
 [ -4.67]
 [  5.78]
 [ -3.76]
 [-16.78]
 [-10.34]]
```

[221]:
```
x = np.array(costs)
y = np.array([1000, 10000, 100000])

plt.plot(y, x)
plt.xlabel("epochs")
plt.ylabel("cost")
plt.show()
```



Since the cost only decreased by 0.02792. Somewhere around 10 000 epochs lays the optimal amount. So for the next model 10 000 epochs will be used.

Model 4 (No bias)

[197]:
```
ann_model_4 = ann.ANN(8, [3], 8, sigmoid, squared_error, normal, zeros,␣
 ↪add_bias=False, **number_generation_kwargs)
ann_model_4.train(training_examples, training_examples, 10000, 0.1)
```

[198]:
```
predictions = ann_model_4.feed_forward(training_examples)
for i, prediction in enumerate(predictions):
    print(f"input: {training_examples[i]}, output: {np.around(prediction, 2)}")

cost = round(mean_squared_error(predictions, training_examples), 5)
print(f"cost: {cost}")
```

```
input: [1. 0. 0. 0. 0. 0. 0. 0.], output: [0.88 0.   0.   0.06 0.06 0.   0.09
0.07]
input: [0. 1. 0. 0. 0. 0. 0. 0.], output: [0.   0.88 0.06 0.06 0.06 0.   0.
0.07]
input: [0. 0. 1. 0. 0. 0. 0. 0.], output: [0.   0.09 0.88 0.09 0.09 0.09 0.   0.
]
input: [0. 0. 0. 1. 0. 0. 0. 0.], output: [0.07 0.07 0.08 0.28 0.28 0.06 0.09
```

```
0.08]
input: [0. 0. 0. 0. 1. 0. 0. 0.], output: [0.06 0.07 0.08 0.28 0.28 0.06 0.09
0.08]
input: [0. 0. 0. 0. 0. 1. 0. 0.], output: [0.   0.   0.08 0.06 0.06 0.88 0.07 0.
]
input: [0. 0. 0. 0. 0. 0. 1. 0.], output: [0.1  0.   0.   0.09 0.09 0.1  0.86 0.
]
input: [0. 0. 0. 0. 0. 0. 0. 1.], output: [0.08 0.09 0.   0.09 0.09 0.   0.
0.88]
cost: 0.37656
```

[199]:
```
for i, layer in enumerate(ann_model_4.network):
    print(f"layer {i + 1} weights: \n{np.around(layer.weights, 2)}")
    print(f"layer {i + 1} biases: \n{np.around(layer.bias, 2)}")
    print()
```

```
layer 1 weights:
[[-4.47  4.55  4.82 -1.36 -1.35  1.48 -1.83 -0.53]
 [ 1.69 -4.79 -0.55 -1.37 -1.38  4.83  4.47 -1.71]
 [ 4.66  1.22 -2.04 -1.4  -1.4  -4.82 -0.58  4.88]]
layer 1 biases:
0

layer 2 weights:
[[-14.91  -1.36   3.34]
 [  3.15 -14.53  -1.29]
 [  5.91  -7.21 -11.07]
 [ -1.56  -1.53  -1.48]
 [ -1.54  -1.54  -1.49]
 [ -1.82   3.58 -15.34]
 [-10.23   5.92  -7.3 ]
 [ -7.51 -10.52   6.42]]
layer 2 biases:
0
```

The cost is 0.52 for the model without bias with the same amount of epochs.

Model 5 (learning rate 100)

[200]:
```
ann_model_5 = ann.ANN(8, [3], 8, sigmoid, squared_error, normal, zeros,␣
 ↪add_bias=True, **number_generation_kwargs)
ann_model_5.train(training_examples, training_examples, 10000, 100)
```

[201]:
```
predictions = ann_model_5.feed_forward(training_examples)
for i, prediction in enumerate(predictions):
    print(f"input: {training_examples[i]}, output: {np.around(prediction, 2)}")

cost = round(mean_squared_error(predictions, training_examples), 5)
print(f"cost: {cost}")
```

```
input: [1. 0. 0. 0. 0. 0. 0. 0.], output: [0. 0. 0. 0. 0. 1. 0. 0.]
input: [0. 1. 0. 0. 0. 0. 0. 0.], output: [0. 0. 0. 0. 0. 1. 0. 0.]
input: [0. 0. 1. 0. 0. 0. 0. 0.], output: [0. 0. 0. 0. 0. 1. 0. 0.]
input: [0. 0. 0. 1. 0. 0. 0. 0.], output: [0. 0. 0. 0. 0. 1. 0. 0.]
input: [0. 0. 0. 0. 1. 0. 0. 0.], output: [0. 0. 0. 0. 0. 1. 0. 0.]
input: [0. 0. 0. 0. 0. 1. 0. 0.], output: [0. 0. 0. 0. 0. 1. 0. 0.]
input: [0. 0. 0. 0. 0. 0. 1. 0.], output: [0. 0. 0. 0. 0. 1. 0. 0.]
input: [0. 0. 0. 0. 0. 0. 0. 1.], output: [0. 0. 0. 0. 0. 1. 0. 0.]
cost: 3.5
```

[202]:
```
for i, layer in enumerate(ann_model_5.network):
    print(f"layer {i + 1} weights: \n{np.around(layer.weights, 2)}")
    print(f"layer {i + 1} biases: \n{np.around(layer.bias, 2)}")
    print()
```

```
layer 1 weights:
[[ 0.07  0.01  0.21  0.14 -0.15 -1.24 -0.03  0.04]
 [-0.1  -0.01 -0.12 -0.09  0.02  0.11 -0.02 -0.03]
 [ 0.19  0.07  0.05 -0.08 -0.09 -0.26  0.04 -0.01]]
layer 1 biases:
0

layer 2 weights:
[[-6.12 -6.97 -6.75]
 [-6.11 -6.63 -6.36]
 [-6.06 -6.7  -6.38]
 [-6.03 -6.58 -6.41]
 [-5.94 -6.42 -6.29]
 [ 5.87  6.48  6.07]
 [-6.63 -6.96 -6.71]
 [-5.82 -6.28 -6.08]]
layer 2 biases:
[[-13.27]
 [-12.62]
 [-12.66]
 [-12.55]
 [-12.24]
 [ 12.06]
 [-13.66]
 [-11.86]]
```

Model 6 (learning rate 10)

[203]:
```
ann_model_6 = ann.ANN(8, [3], 8, sigmoid, squared_error, normal, zeros,␣
 ↪add_bias=True, **number_generation_kwargs)
ann_model_6.train(training_examples, training_examples, 10000, 10)
```

[204]:
```
predictions = ann_model_6.feed_forward(training_examples)
for i, prediction in enumerate(predictions):
    print(f"input: {training_examples[i]}, output: {np.around(prediction, 2)}")

cost = round(mean_squared_error(predictions, training_examples), 5)
print(f"cost: {cost}")
```

```
input: [1. 0. 0. 0. 0. 0. 0. 0.], output: [0.99 0.   0.   0.   0.   0.   0.   0.
]
input: [0. 1. 0. 0. 0. 0. 0. 0.], output: [0.   0.99 0.01 0.01 0.   0.   0.   0.
]
input: [0. 0. 1. 0. 0. 0. 0. 0.], output: [0.01 0.   0.99 0.   0.   0.   0.   0.
]
input: [0. 0. 0. 1. 0. 0. 0. 0.], output: [0.   0.01 0.   0.99 0.   0.   0.   0.
]
input: [0. 0. 0. 0. 1. 0. 0. 0.], output: [0.   0.01 0.   0.   0.99 0.01 0.   0.
]
input: [0. 0. 0. 0. 0. 1. 0. 0.], output: [0.   0.   0.   0.   0.   0.99 0.   0.
]
input: [0. 0. 0. 0. 0. 0. 1. 0.], output: [0.   0.   0.   0.   0.   0.   0.99 0.
]
input: [0. 0. 0. 0. 0. 0. 0. 1.], output: [0.   0.   0.   0.   0.   0.   0.01
0.99]
cost: 0.00023
```

[205]:
```
for i, layer in enumerate(ann_model_6.network):
    print(f"layer {i + 1} weights: \n{np.around(layer.weights, 2)}")
    print()
```

```
layer 1 weights:
[[ 6.34 -7.33 -0.21 -0.84 -0.94  5.22  5.24 -1.71]
 [ 0.39 -0.72 -5.71  5.63 -5.54 -4.84  5.14  2.55]
 [ 5.61  0.31  3.38  4.63 -6.   -1.56 -4.53 -5.46]]

layer 2 weights:
[[ 15.28   1.74  10.99]
 [-29.8   -4.47   5.01]
 [ -1.8  -17.27  13.52]
 [ -7.91  11.95  10.94]
 [-10.65 -13.41 -15.84]
 [ 15.66 -11.94  -5.05]
 [ 11.64   9.59  -7.32]
 [-13.6   10.76 -11.49]]
```

Model 7 (learning rate 0.01)

[206]:
```
ann_model_7 = ann.ANN(8, [3], 8, sigmoid, squared_error, normal, zeros,␣
 ↪add_bias=True, **number_generation_kwargs)
ann_model_7.train(training_examples, training_examples, 10000, 0.01)
```

[207]:
```
predictions = ann_model_7.feed_forward(training_examples)
for i, prediction in enumerate(predictions):
    print(f"input: {training_examples[i]}, output: {np.around(prediction, 2)}")

cost = round(mean_squared_error(predictions, training_examples), 5)
print(f"cost: {cost}")
```

```
input: [1. 0. 0. 0. 0. 0. 0. 0.], output: [0.18 0.09 0.14 0.17 0.09 0.19 0.14
0.16]
input: [0. 1. 0. 0. 0. 0. 0. 0.], output: [0.1  0.43 0.11 0.16 0.36 0.06 0.16
0.12]
input: [0. 0. 1. 0. 0. 0. 0. 0.], output: [0.14 0.09 0.17 0.12 0.14 0.27 0.11
0.13]
input: [0. 0. 0. 1. 0. 0. 0. 0.], output: [0.18 0.15 0.14 0.21 0.12 0.14 0.17
0.17]
input: [0. 0. 0. 0. 1. 0. 0. 0.], output: [0.07 0.32 0.11 0.09 0.42 0.08 0.11
0.09]
input: [0. 0. 0. 0. 0. 1. 0. 0.], output: [0.17 0.05 0.22 0.11 0.1  0.47 0.1
0.13]
input: [0. 0. 0. 0. 0. 0. 1. 0.], output: [0.14 0.14 0.11 0.16 0.14 0.11 0.15
0.14]
input: [0. 0. 0. 0. 0. 0. 0. 1.], output: [0.16 0.1  0.13 0.16 0.11 0.14 0.14
0.15]
cost: 1.40664
```

[208]:
```
for i, layer in enumerate(ann_model_7.network):
    print(f"layer {i + 1} weights: \n{np.around(layer.weights, 2)}")
    print(f"layer {i + 1} biases: \n{np.around(layer.bias, 2)}")
    print()
```

```
layer 1 weights:
[[-0.54 -0.43  0.67 -1.28  1.26  1.21 -0.57 -0.5 ]
 [-0.27  1.28 -0.61 -0.12  1.51 -2.08  0.59  0.2 ]
 [ 1.1  -1.96  0.49  0.26 -1.3   1.06  0.56  0.92]]
layer 1 biases:
0

layer 2 weights:
[[-0.96 -0.95  0.45]
 [-0.69  0.97 -2.83]
 [ 0.29 -1.36 -0.22]
 [-1.67 -0.64 -0.36]
 [ 0.96  1.06 -2.1 ]
 [ 0.96 -2.89  0.46]
 [-1.05 -0.03 -0.24]
 [-0.97 -0.47  0.19]]
layer 2 biases:
[[-1.11]
```

```
 [-0.42]
 [-1.14]
 [-0.43]
 [-1.51]
 [-0.89]
 [-1.21]
 [-1.25]]
```

With a learning rate of 100 the gradient overshoots and does not learn correctly, with a rate of 10 it learns almost perfectly within 10 000 epochs, and with a learning rate of 0.01 it didn't converge within 10 000 epochs.

Model 8 (weights initialized as 0, 1000 epochs)

[209]:
```
ann_model_8 = ann.ANN(8, [3], 8, sigmoid, squared_error, zeros, zeros,␣
 ↪add_bias=True, **number_generation_kwargs)
ann_model_8.train(training_examples, training_examples, 1000, 0.1)
```

[210]:
```
predictions = ann_model_8.feed_forward(training_examples)
for i, prediction in enumerate(predictions):
    print(f"input: {training_examples[i]}, output: {np.around(prediction, 2)}")

cost = round(mean_squared_error(predictions, training_examples), 5)
print(f"cost: {cost}")
```

```
input: [1. 0. 0. 0. 0. 0. 0. 0.], output: [0.12 0.12 0.12 0.12 0.12 0.12 0.12
0.12]
input: [0. 1. 0. 0. 0. 0. 0. 0.], output: [0.12 0.12 0.12 0.12 0.12 0.12 0.12
0.12]
input: [0. 0. 1. 0. 0. 0. 0. 0.], output: [0.12 0.12 0.12 0.12 0.12 0.12 0.12
0.12]
input: [0. 0. 0. 1. 0. 0. 0. 0.], output: [0.12 0.12 0.12 0.12 0.12 0.12 0.12
0.12]
input: [0. 0. 0. 0. 1. 0. 0. 0.], output: [0.12 0.12 0.12 0.12 0.12 0.12 0.12
0.12]
input: [0. 0. 0. 0. 0. 1. 0. 0.], output: [0.12 0.12 0.12 0.12 0.12 0.12 0.12
0.12]
input: [0. 0. 0. 0. 0. 0. 1. 0.], output: [0.12 0.12 0.12 0.12 0.12 0.12 0.12
0.12]
input: [0. 0. 0. 0. 0. 0. 0. 1.], output: [0.12 0.12 0.12 0.12 0.12 0.12 0.12
0.12]
cost: 1.75001
```

[211]:
```
for i, layer in enumerate(ann_model_8.network):
    print(f"layer {i + 1} weights: \n{np.around(layer.weights, 2)}")
    print(f"layer {i + 1} biases: \n{np.around(layer.bias, 2)}")
    print()
```

```
layer 1 weights:
[[0.07 0.07 0.07 0.07 0.07 0.07 0.07 0.07]
```

```
 [0.07 0.07 0.07 0.07 0.07 0.07 0.07 0.07]
 [0.07 0.07 0.07 0.07 0.07 0.07 0.07 0.07]]
layer 1 biases:
0

layer 2 weights:
[[-0.55 -0.55 -0.55]
 [-0.55 -0.55 -0.55]
 [-0.55 -0.55 -0.55]
 [-0.55 -0.55 -0.55]
 [-0.55 -0.55 -0.55]
 [-0.55 -0.55 -0.55]
 [-0.55 -0.55 -0.55]
 [-0.55 -0.55 -0.55]]
layer 2 biases:
[[-1.09]
 [-1.09]
 [-1.09]
 [-1.1 ]
 [-1.09]
 [-1.1 ]
 [-1.09]
 [-1.09]]
```

Model 9 (weights initialized as 0, 10000 epochs)

[212]:
```
ann_model_9 = ann.ANN(8, [3], 8, sigmoid, squared_error, zeros, zeros,␣
 ↪add_bias=True, **number_generation_kwargs)
ann_model_9.train(training_examples, training_examples, 10000, 0.1)
```

[213]:
```
predictions = ann_model_9.feed_forward(training_examples)
for i, prediction in enumerate(predictions):
    print(f"input: {training_examples[i]}, output: {np.around(prediction, 2)}")

cost = round(mean_squared_error(predictions, training_examples), 5)
print(f"cost: {cost}")
```

```
input: [1. 0. 0. 0. 0. 0. 0. 0.], output: [0.18 0.1  0.18 0.18 0.09 0.   0.09
0.07]
input: [0. 1. 0. 0. 0. 0. 0. 0.], output: [0.09 0.15 0.09 0.09 0.   0.09 0.15
0.15]
input: [0. 0. 1. 0. 0. 0. 0. 0.], output: [0.18 0.1  0.18 0.18 0.09 0.   0.09
0.07]
input: [0. 0. 0. 1. 0. 0. 0. 0.], output: [0.18 0.1  0.18 0.18 0.09 0.   0.09
0.07]
input: [0. 0. 0. 0. 1. 0. 0. 0.], output: [0.25 0.08 0.25 0.25 0.83 0.   0.07
0.05]
input: [0. 0. 0. 0. 0. 1. 0. 0.], output: [0.05 0.2  0.05 0.05 0.   0.79 0.23
0.27]
```

```
input: [0. 0. 0. 0. 0. 0. 1. 0.], output: [0.08 0.15 0.08 0.08 0.   0.12 0.16
0.16]
input: [0. 0. 0. 0. 0. 0. 0. 1.], output: [0.08 0.16 0.08 0.08 0.   0.17 0.16
0.17]
cost: 1.28603
```

[214]:
```
for i, layer in enumerate(ann_model_9.network):
    print(f"layer {i + 1} weights: \n{np.around(layer.weights, 2)}")
    print(f"layer {i + 1} biases: \n{np.around(layer.bias, 2)}")
    print()
```

```
layer 1 weights:
[[-1.22  0.47 -1.22 -1.22 -3.65  3.37  0.62  0.78]
 [-1.22  0.47 -1.22 -1.22 -3.65  3.37  0.62  0.78]
 [-1.22  0.47 -1.22 -1.22 -3.65  3.37  0.62  0.78]]
layer 1 biases:
0

layer 2 weights:
[[-0.68 -0.68 -0.68]
 [ 0.36  0.36  0.36]
 [-0.67 -0.67 -0.67]
 [-0.68 -0.68 -0.68]
 [-6.39 -6.39 -6.39]
 [ 3.46  3.46  3.46]
 [ 0.51  0.51  0.51]
 [ 0.73  0.73  0.73]]
layer 2 biases:
[[-1.07]
 [-2.43]
 [-1.07]
 [-1.06]
 [ 2.09]
 [-8.72]
 [-2.67]
 [-3.09]]
```

Model 10 (weights initialized as 0, 10000 epochs, no bias)

[215]:
```
ann_model_10 = ann.ANN(8, [3], 8, sigmoid, squared_error, zeros, zeros,␣
 ↪add_bias=False, **number_generation_kwargs)
ann_model_10.train(training_examples, training_examples, 10000, 0.1)
```

[216]:
```
predictions = ann_model_10.feed_forward(training_examples)
for i, prediction in enumerate(predictions):
    print(f"input: {training_examples[i]}, output: {np.around(prediction, 2)}")

cost = round(mean_squared_error(predictions, training_examples), 5)
print(f"cost: {cost}")
```

```
input: [1. 0. 0. 0. 0. 0. 0. 0.], output: [0.12 0.12 0.12 0.12 0.12 0.12 0.12
0.12]
input: [0. 1. 0. 0. 0. 0. 0. 0.], output: [0.12 0.12 0.12 0.12 0.12 0.12 0.12
0.12]
input: [0. 0. 1. 0. 0. 0. 0. 0.], output: [0.12 0.12 0.12 0.12 0.12 0.12 0.12
0.12]
input: [0. 0. 0. 1. 0. 0. 0. 0.], output: [0.12 0.12 0.12 0.12 0.12 0.12 0.12
0.12]
input: [0. 0. 0. 0. 1. 0. 0. 0.], output: [0.12 0.12 0.12 0.12 0.12 0.12 0.12
0.12]
input: [0. 0. 0. 0. 0. 1. 0. 0.], output: [0.12 0.12 0.12 0.12 0.12 0.12 0.12
0.12]
input: [0. 0. 0. 0. 0. 0. 1. 0.], output: [0.12 0.12 0.12 0.12 0.12 0.12 0.12
0.12]
input: [0. 0. 0. 0. 0. 0. 0. 1.], output: [0.12 0.12 0.12 0.12 0.12 0.12 0.12
0.12]
cost: 1.75
```

[217]:
```python
for i, layer in enumerate(ann_model_10.network):
    print(f"layer {i + 1} weights: \n{np.around(layer.weights, 2)}")
    print(f"layer {i + 1} biases: \n{np.around(layer.bias, 2)}")
    print()
```

```
layer 1 weights:
[[0.24 0.24 0.24 0.24 0.24 0.24 0.24 0.24]
 [0.24 0.24 0.24 0.24 0.24 0.24 0.24 0.24]
 [0.24 0.24 0.24 0.24 0.24 0.24 0.24 0.24]]
layer 1 biases:
0

layer 2 weights:
[[-1.16 -1.16 -1.16]
 [-1.16 -1.16 -1.16]
 [-1.16 -1.16 -1.16]
 [-1.16 -1.16 -1.16]
 [-1.16 -1.16 -1.16]
 [-1.16 -1.16 -1.16]
 [-1.16 -1.16 -1.16]
 [-1.16 -1.16 -1.16]]
layer 2 biases:
0
```

With the weights initialized as 0 the model learns minimally, this has probably to do with the fact that almost all backpropagation formula's use multiplication or the dot product only the bias can have non zero values.

Model 11 (regularization, train with first 6 examples)

[218]:
```
ann_model_11 = ann.ANN(8, [3], 8, sigmoid, squared_error, normal, zeros,␣
 ↪add_bias=True, **number_generation_kwargs)
ann_model_11.train(training_examples[0:6], training_examples[0:6], 10000, 0.1)
```

[219]:
```
predictions = ann_model_11.feed_forward(training_examples[6:8])
for i, prediction in enumerate(predictions):
    print(f"input: {training_examples[6:8][i]}, output: {np.around(prediction,␣
 ↪2)}")

cost = round(mean_squared_error(predictions, training_examples[6:8]), 5)
print(f"cost: {cost}")
```

```
input: [0. 0. 0. 0. 0. 0. 1. 0.], output: [0.03 0.03 0.02 0.04 0.03 0.02 0.   0.
]
input: [0. 0. 0. 0. 0. 0. 0. 1.], output: [0.01 0.02 0.03 0.04 0.03 0.03 0.   0.
]
cost: 1.99211
```

[220]:
```
for i, layer in enumerate(ann_model_11.network):
    print(f"layer {i + 1} weights: \n{np.around(layer.weights, 2)}")
    print(f"layer {i + 1} biases: \n{np.around(layer.bias, 2)}")
    print()
```

```
layer 1 weights:
[[-3.53 -4.15  3.97  2.33 -1.46  3.27 -0.2   0.14]
 [-3.03  3.2  -3.85  3.72 -3.34  2.78 -0.07 -0.08]
 [ 3.22 -0.88  0.64  4.17 -4.   -3.99  0.01 -0.09]]
layer 1 biases:
0

layer 2 weights:
[[-5.71 -4.02  5.62]
 [-7.36  5.85 -1.95]
 [ 6.79 -6.59  0.87]
 [ 1.99  4.26  5.88]
 [-2.36 -5.14 -6.88]
 [ 4.79  2.98 -6.98]
 [-1.26 -1.38 -1.44]
 [-1.37 -1.42 -1.35]]
layer 2 biases:
[[-1.94]
 [-1.94]
 [-4.22]
 [-9.12]
 [ 3.44]
 [-4.25]
 [-3.44]
 [-3.43]]
```

The model is not able to generalize to the last 2 examples, this is probably because the model is overfitted to the first 6 examples.