Wprowadzenie do sieci neuronowych

Piotr Krzywicki

UNIVERSITAS VARSOVIENSIS
UNIWERSYTET OTWARTY
UNIWERSYTETU WARSZAWSKIEGO

Przypomnienie

Zadanie klasyfikacji czy regresji polegają na dopasowaniu do otykietowanego zbioru danych X, y funkcji, dla której z dostatecznie dużą dokładnością zachodzi $f_\theta(x) \approx y$ dla losowych x, y z tego zbioru.

Przypomnienie

Żeby tego dokonać, uznajemy, że naszą funkcję f_θ daje się sparametryzować wagami θ.

Przypomnienie

Przykładowy schemat wygląda następująco:

0. Do problemu dobierz architekturę modelu, parametryzowaną wagami θ, funkcję błędu $\mathcal{L}(x, y, \theta)$, której niska oczekiwana wartość dla danych wag gwarantuje dobrą jakość modelu
1. Niech X to dane wejściowe, y to etykiety
2. Zainicjalizuj parametry modelu θ
3. Powtarzaj dopóki $\mathcal{L}(\theta)$ nie będzie wystarczająco niska
    - Minimalizuj $\mathcal{L}(\theta)$ zmieniając wagi θ
        - zazwyczaj powyższa minimalizacja oparta jest na gradiencie
    - Innymi słowy zazwyczaj $\theta = \theta - \alpha * \nabla_\theta \mathcal{L}(\theta)$

Przypomnienie

Algorytm regresji liniowej zakłada:

$$f_\theta(x) = \theta^T x$$
$$\mathcal{L}(x, y, \theta) = (f_\theta(x) - y)^2$$

Algorytm regresji logistcznej zakłada:

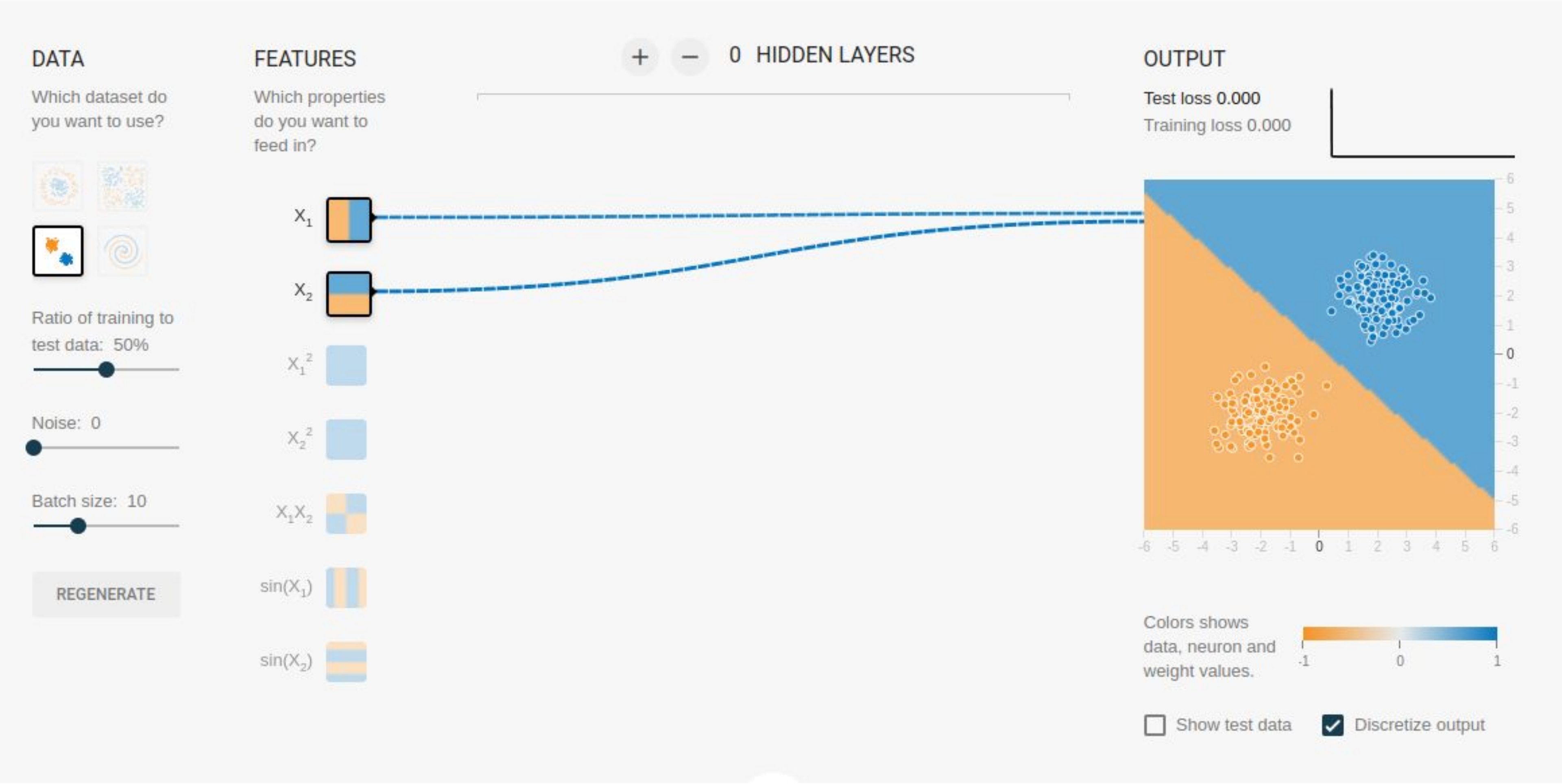
$$f_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$
$$\mathcal{L}(x, y, \theta) = H(f_\theta(x), y)$$

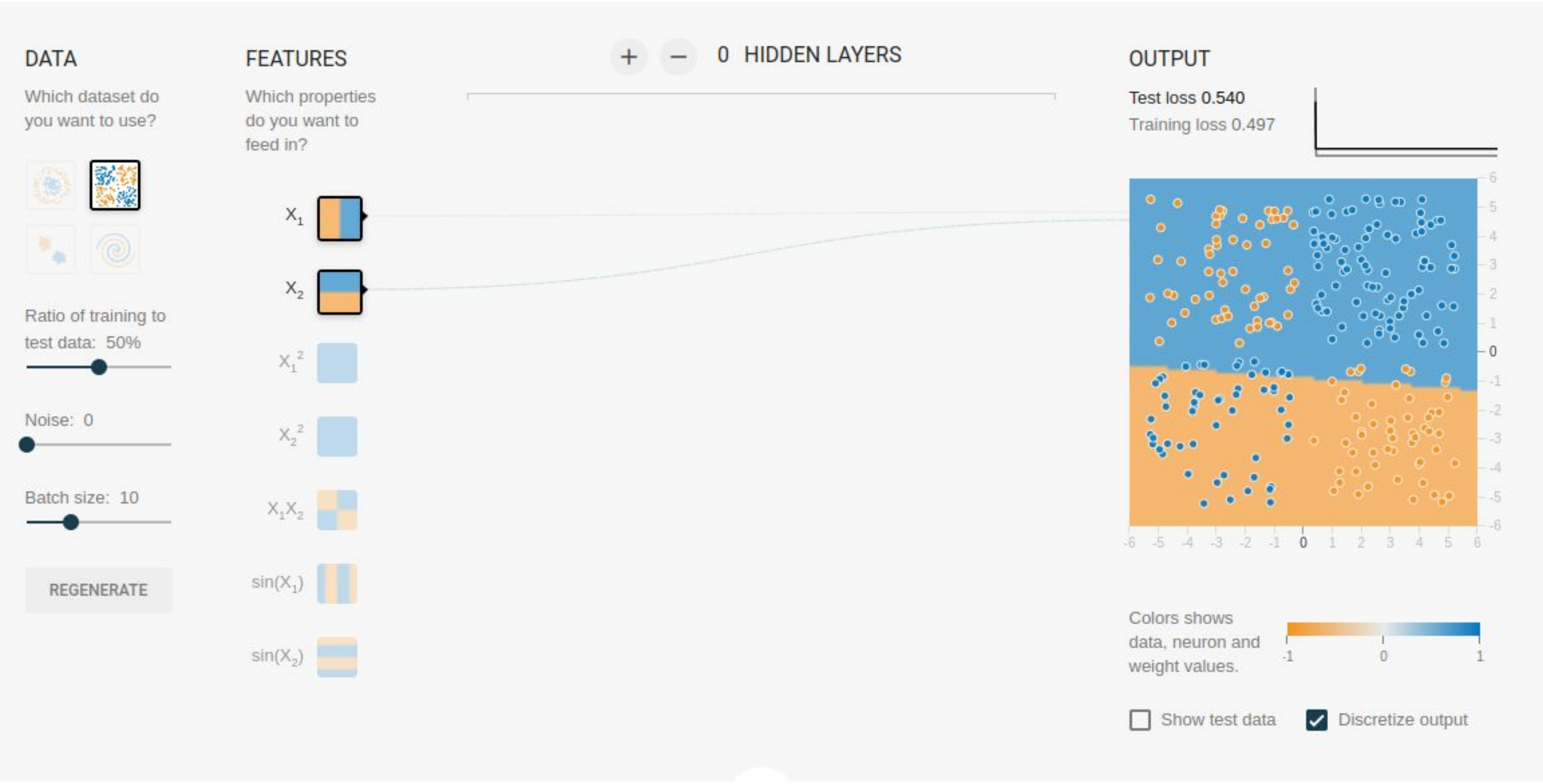
Algorytm regresji logistcznej z wieloma klasami (softmax regression/multinomal regression) zakłada:

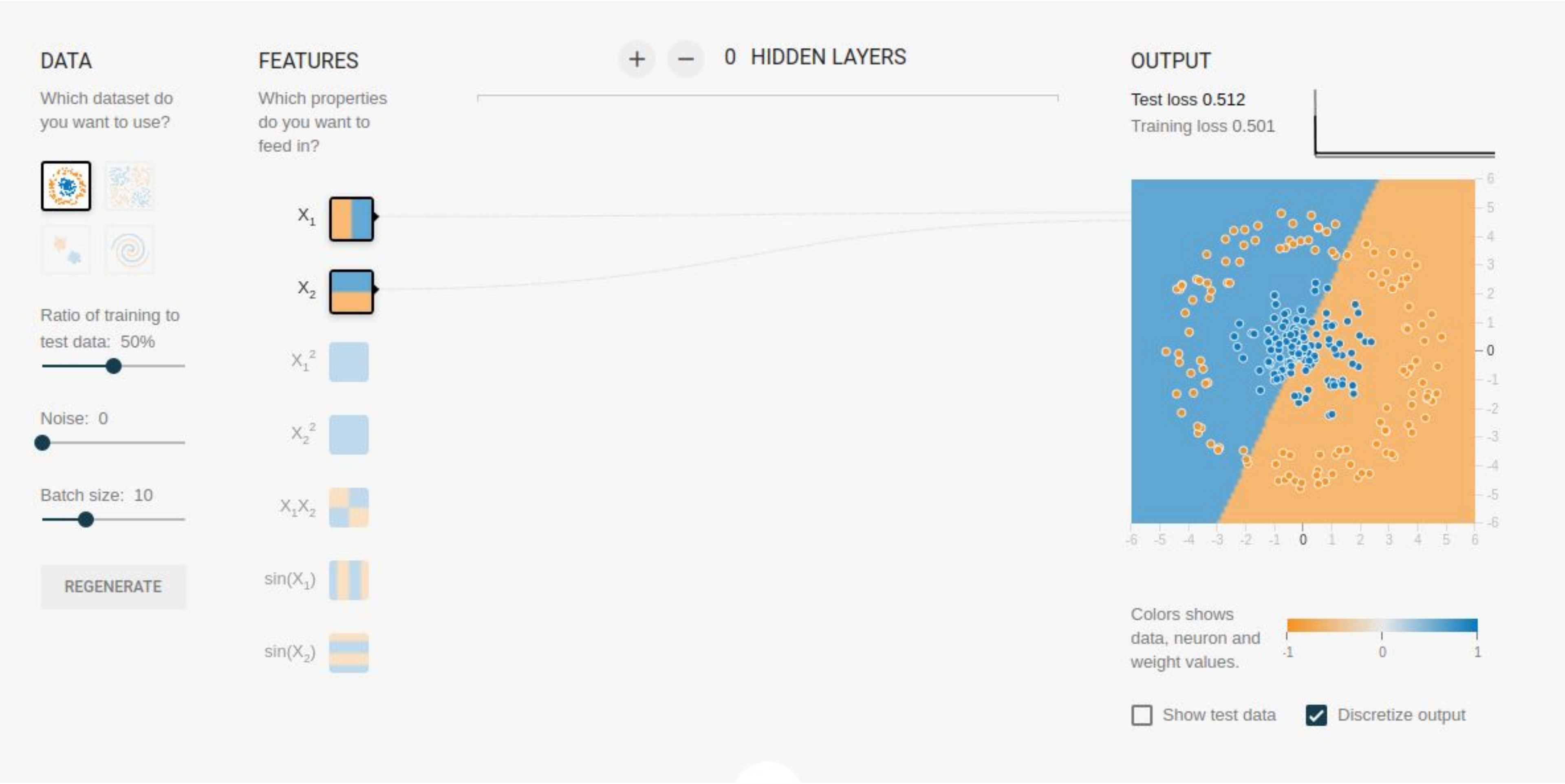
$$f_\theta(x) = softmax(\theta^T x)$$

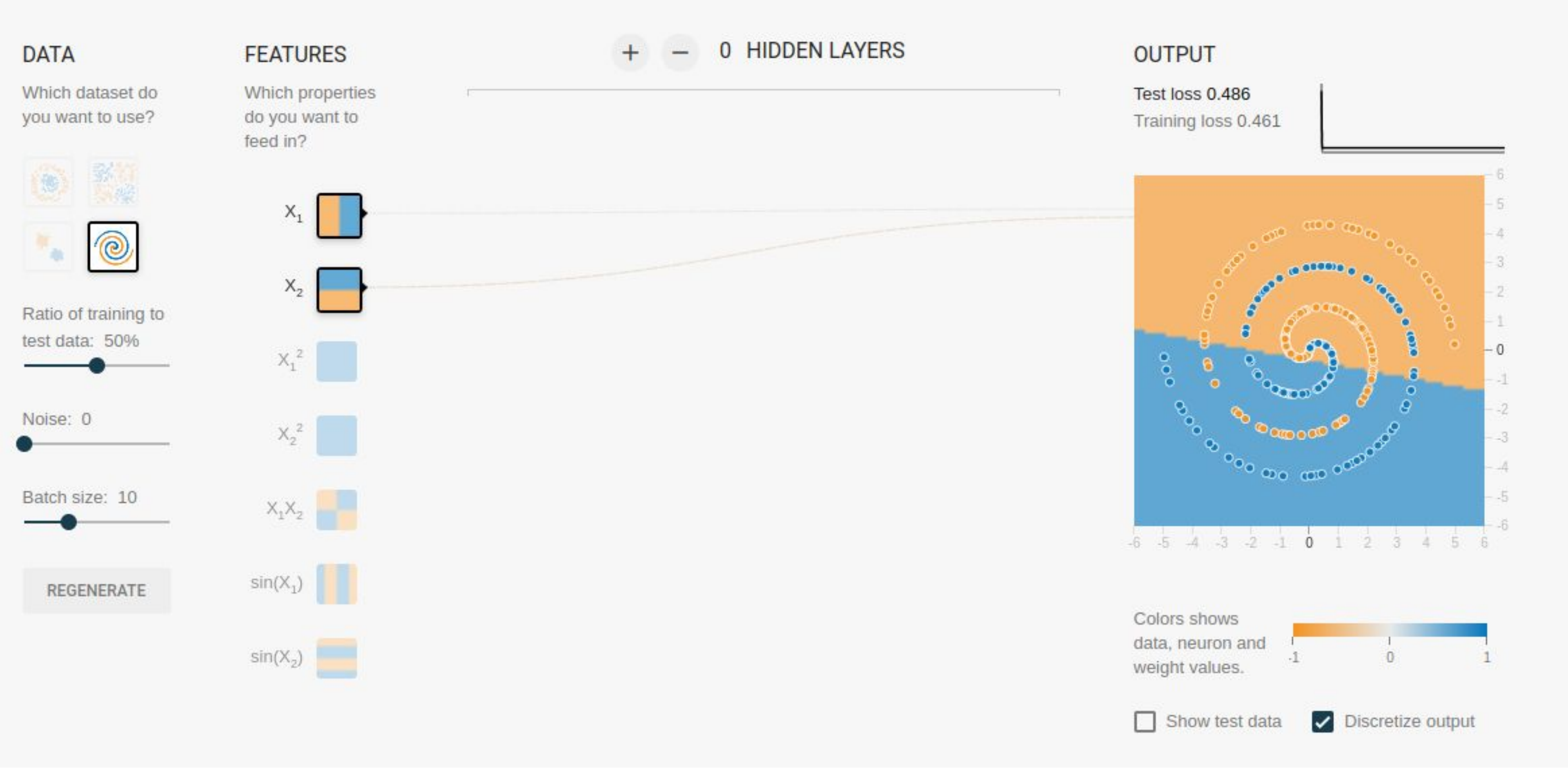
gdzie

$$softmax(x_1, \ldots, x_n) = \left(\frac{exp(x_i)}{exp(x_1) + \ldots + exp(x_n)}\right)_i$$
$$\mathcal{L}(x, y, \theta) = H(f_\theta(x), y)$$


DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
X1
X2
X1^2
X2^2
X1X2
sin(X1)
sin(X2)
+ − 0 HIDDEN LAYERS
OUTPUT
Test loss 0.000
Training loss 0.000
Colors shows data, neuron and weight values.
-1 0 1
Show test data
Discretize output


DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
X1
X2
X1^2
X2^2
X1X2
sin(X1)
sin(X2)
0 HIDDEN LAYERS
OUTPUT
Test loss 0.540
Training loss 0.497
Colors shows data, neuron and weight values.
-1
0
1
Show test data
Discretize output

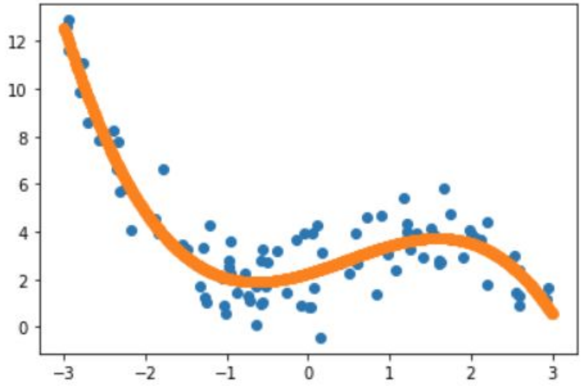

DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
X_1
X_2
X_1^2
X_2^2
X_1X_2
$\sin(X_1)$
$\sin(X_2)$
+ – 0 HIDDEN LAYERS
OUTPUT
Test loss 0.512
Training loss 0.501
Colors shows data, neuron and weight values.
-1 0 1
Show test data
Discretize output


DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
X1
X2
X1^2
X2^2
X1X2
sin(X1)
sin(X2)
+ – 0 HIDDEN LAYERS
OUTPUT
Test loss 0.486
Training loss 0.461
6 5 4 3 2 1 0 -1 -2 -3 -4 -5 -6
-6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6
Colors shows data, neuron and weight values.
-1 0 1
Show test data
Discretize output

Przypomnienie – feature engineering

Zwizualizujmy dopasowanie naszego modelu:

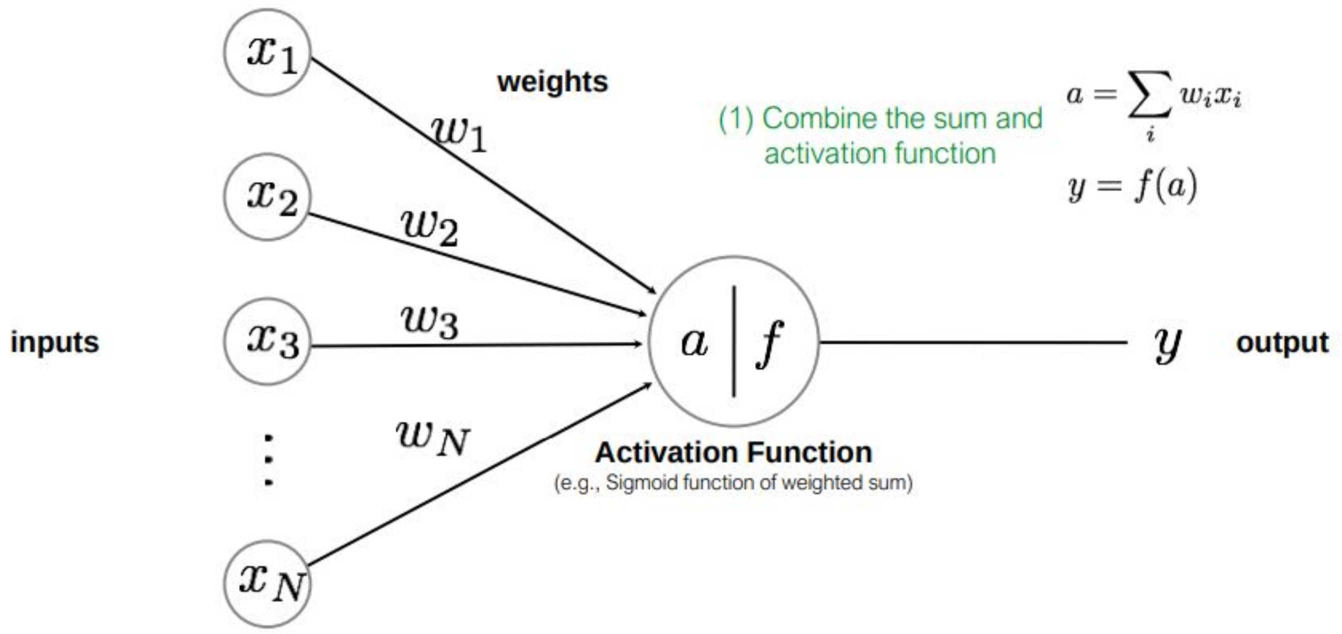
```
[ ] xs = np.linspace(-3, 3, 1000)
    plt.scatter(X[:, 0], y)
    plt.scatter(xs,
    """
    TODO:
    your code goes here
    """
    )
    plt.show()
```



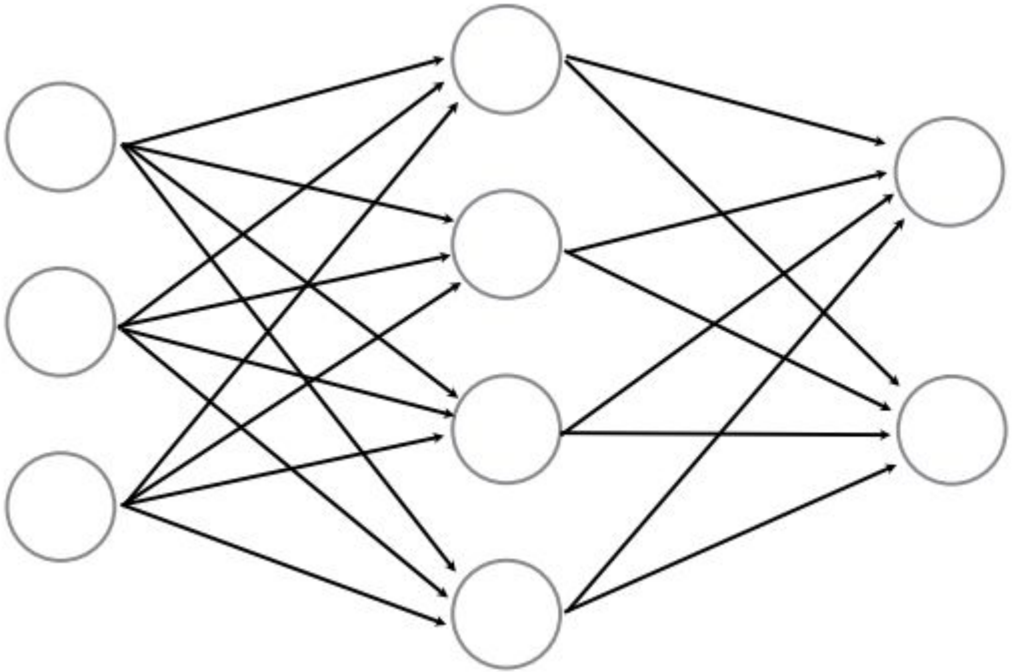
Jak rozwiązać te problemy?

- Feature engineering (inżynieria cech)
- Silniejszy, nieliniowy model

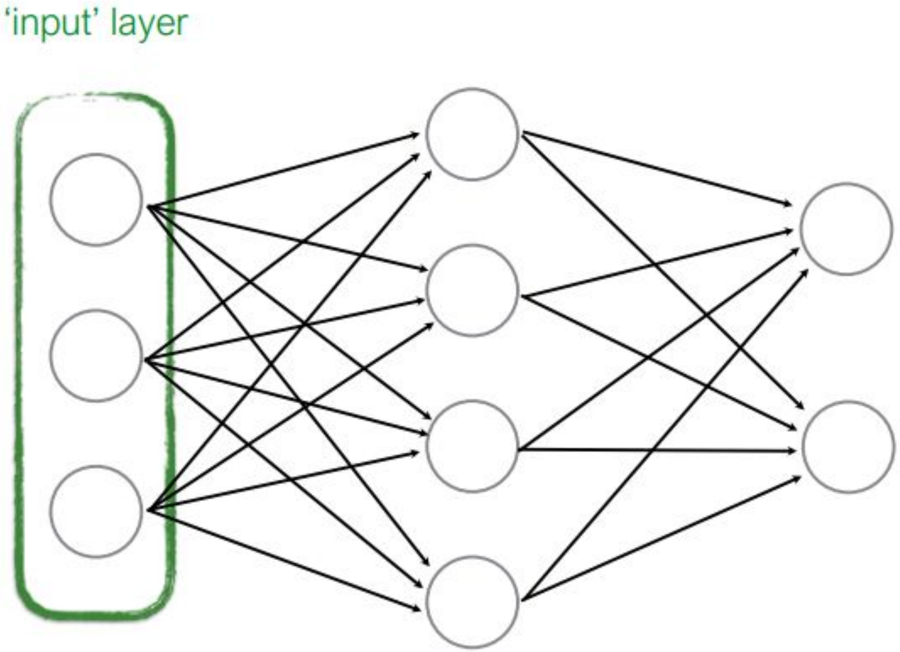
Model Regresji Liniowej, ale inaczej



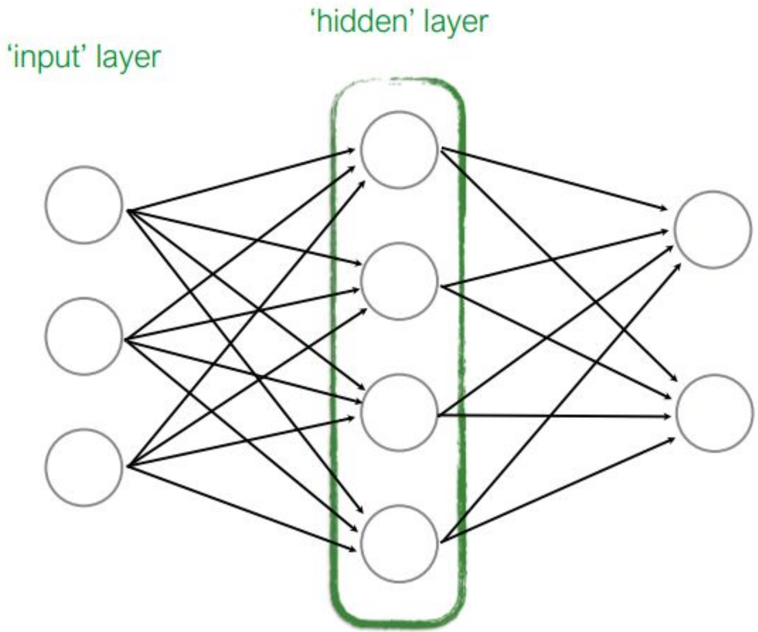
Dlaczego by nie dołożyć dodatkowych węzłów?



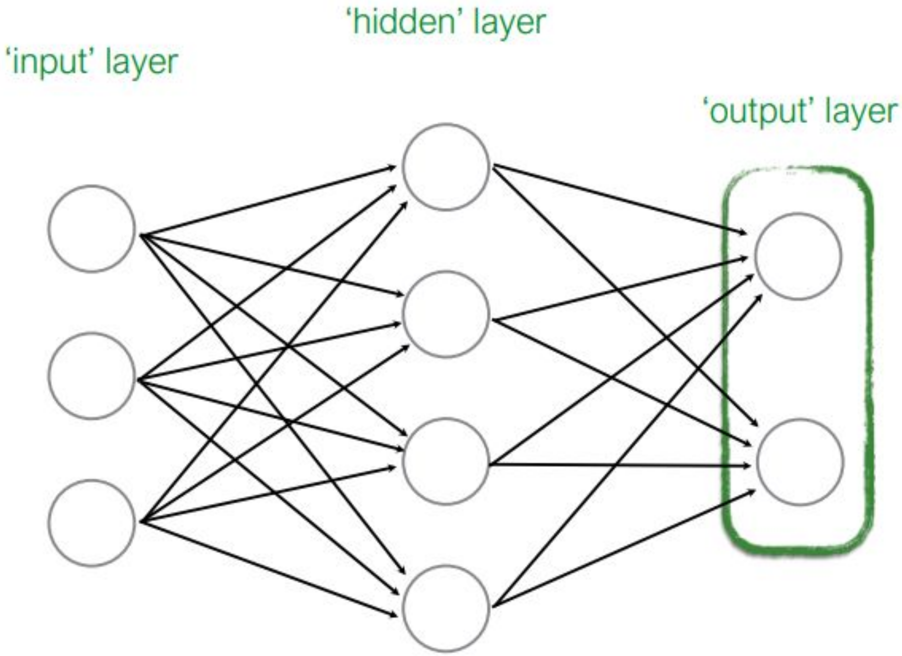
Nazewnictwo



Nazewnictwo



Nazewnictwo



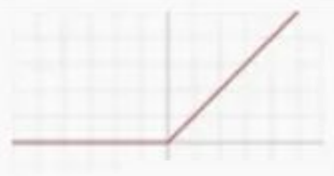
Problem? To nie jest silniejsza klasa modeli

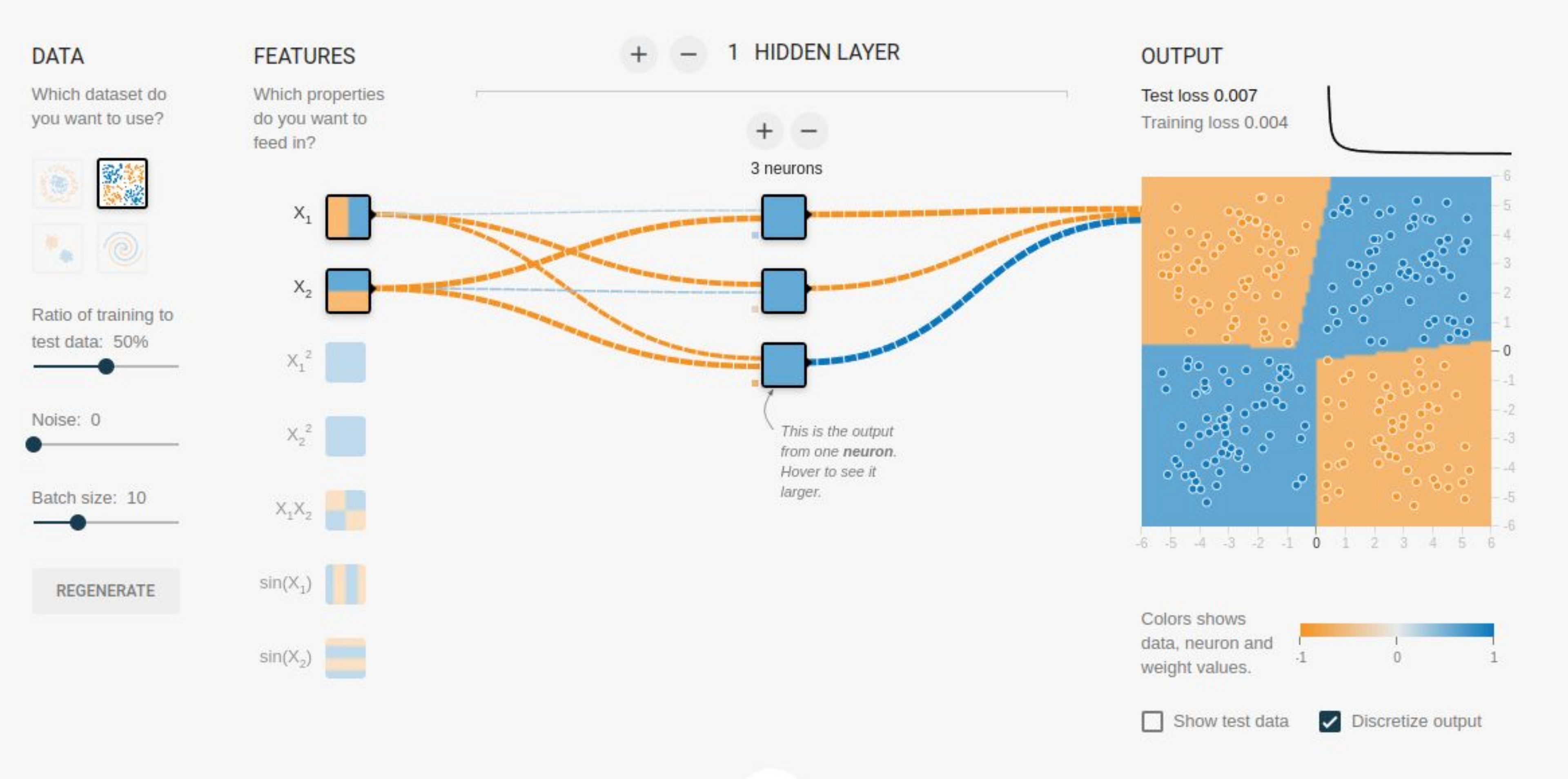
$$y = W_2 W_1 x = W' x,\ \text{gdzie}\, W = W_2 W_1$$

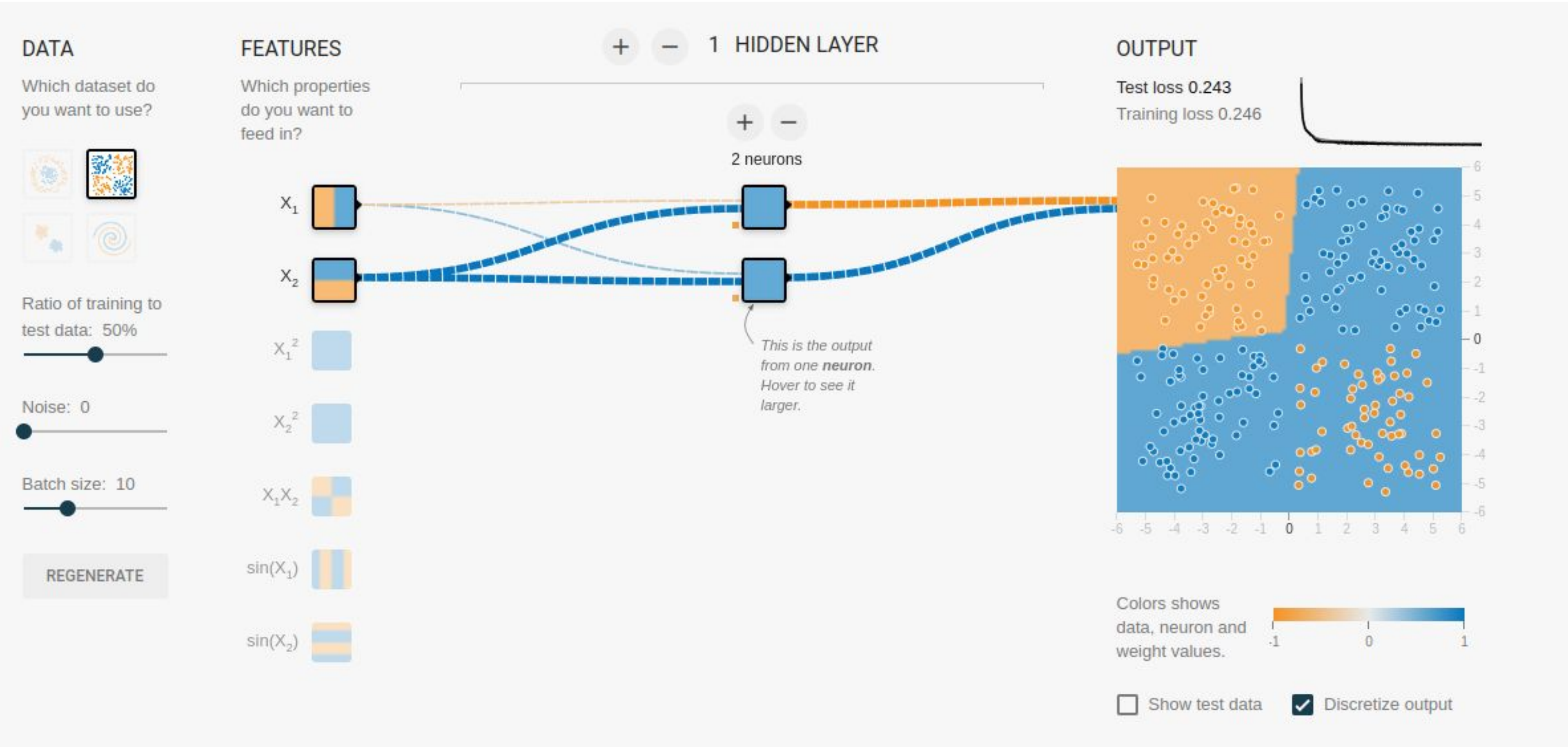
Złamanie liniowości

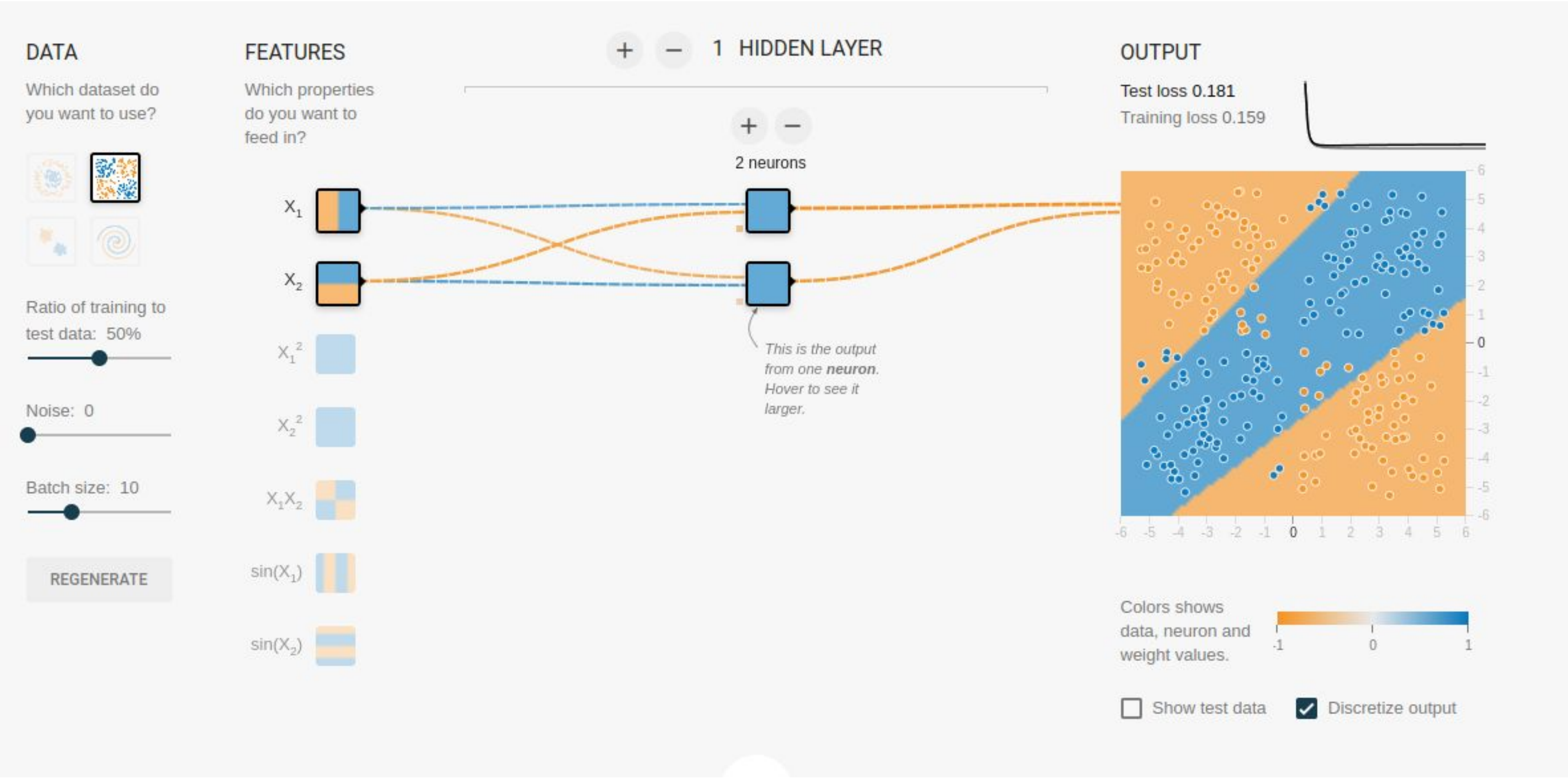
$$y = f(W_2 f(W_1 x)),\ \text{gdzie } f \text{ nieliniowa}$$

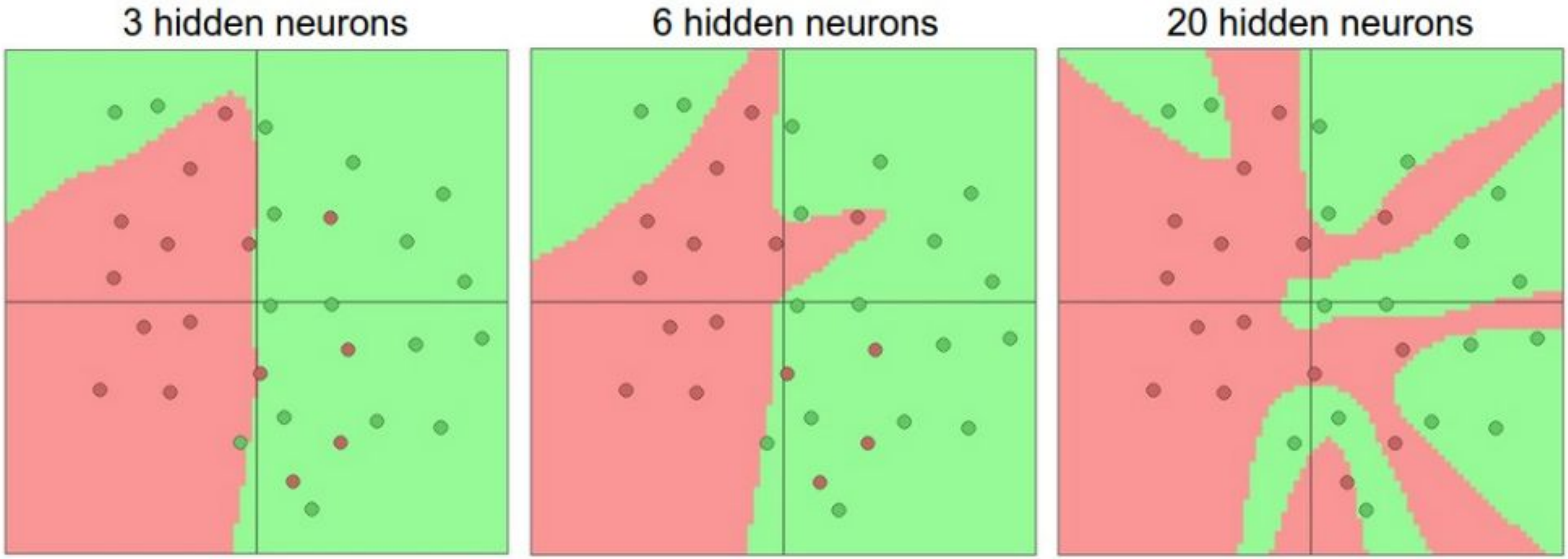
Sprawdzone wybory funkcji aktywacji f

Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$
		$f(x) = \begin{cases} \alpha x & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$
ArcTan		$f(x) = \tan^{-1}(x)$

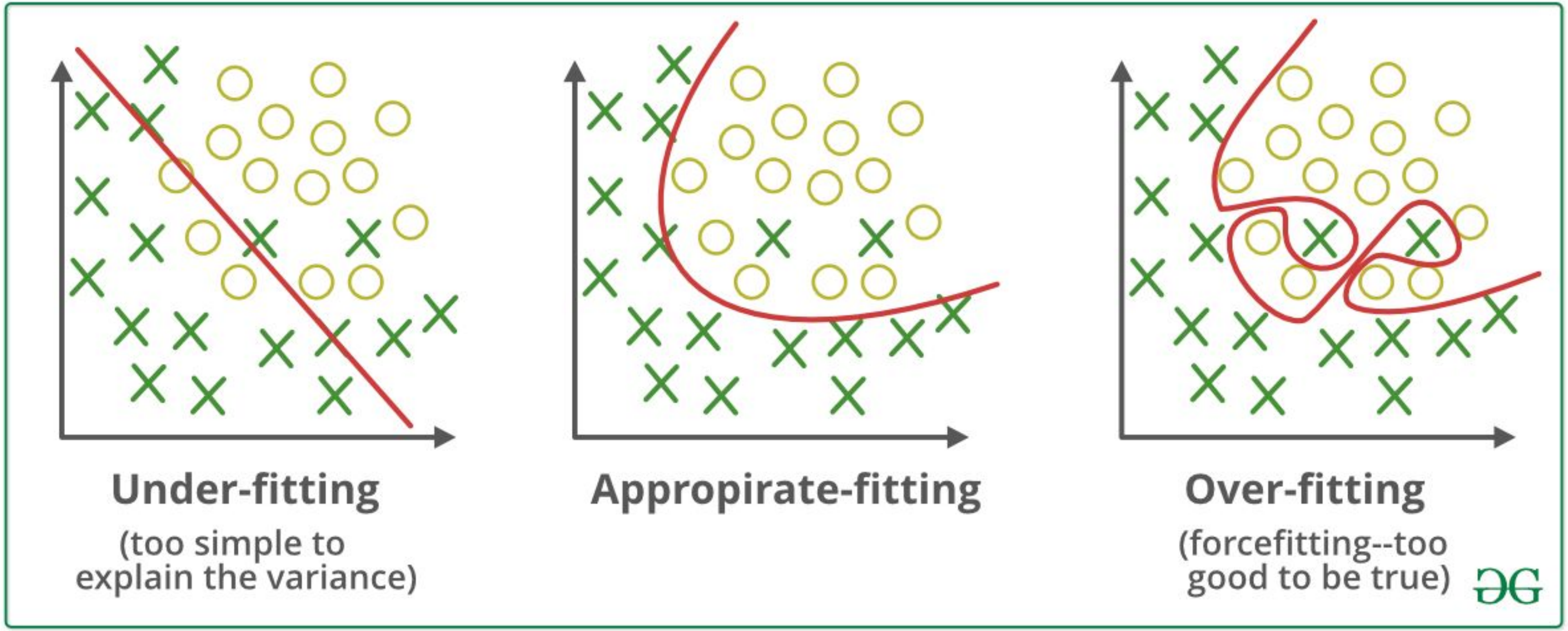

DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
X1
X2
X1 2
X2 2
X1X2
sin(X1)
sin(X2)
1 HIDDEN LAYER
3 neurons
This is the output from one neuron. Hover to see it larger.
OUTPUT
Test loss 0.007
Training loss 0.004
Colors shows data, neuron and weight values.
-1
0
1
Show test data
Discretize output

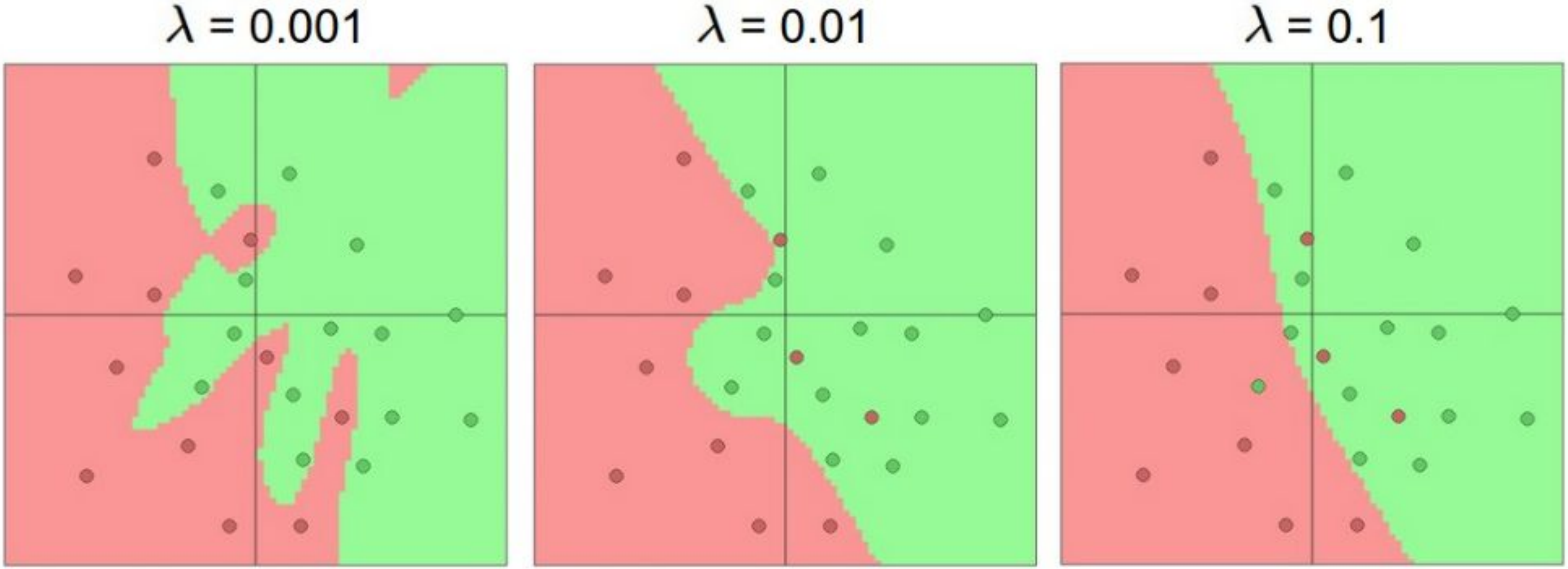

DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
X1
X2
X1^2
X2^2
X1X2
sin(X1)
sin(X2)
1 HIDDEN LAYER
2 neurons
This is the output from one neuron. Hover to see it larger.
OUTPUT
Test loss 0.243
Training loss 0.246
Colors shows data, neuron and weight values.
-1
0
1
Show test data
Discretize output


DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
X1
X2
X1^2
X2^2
X1X2
sin(X1)
sin(X2)
+ − 1 HIDDEN LAYER
+ −
2 neurons
This is the output from one neuron. Hover to see it larger.
OUTPUT
Test loss 0.181
Training loss 0.159
Colors shows data, neuron and weight values.
-1 0 1
Show test data
Discretize output

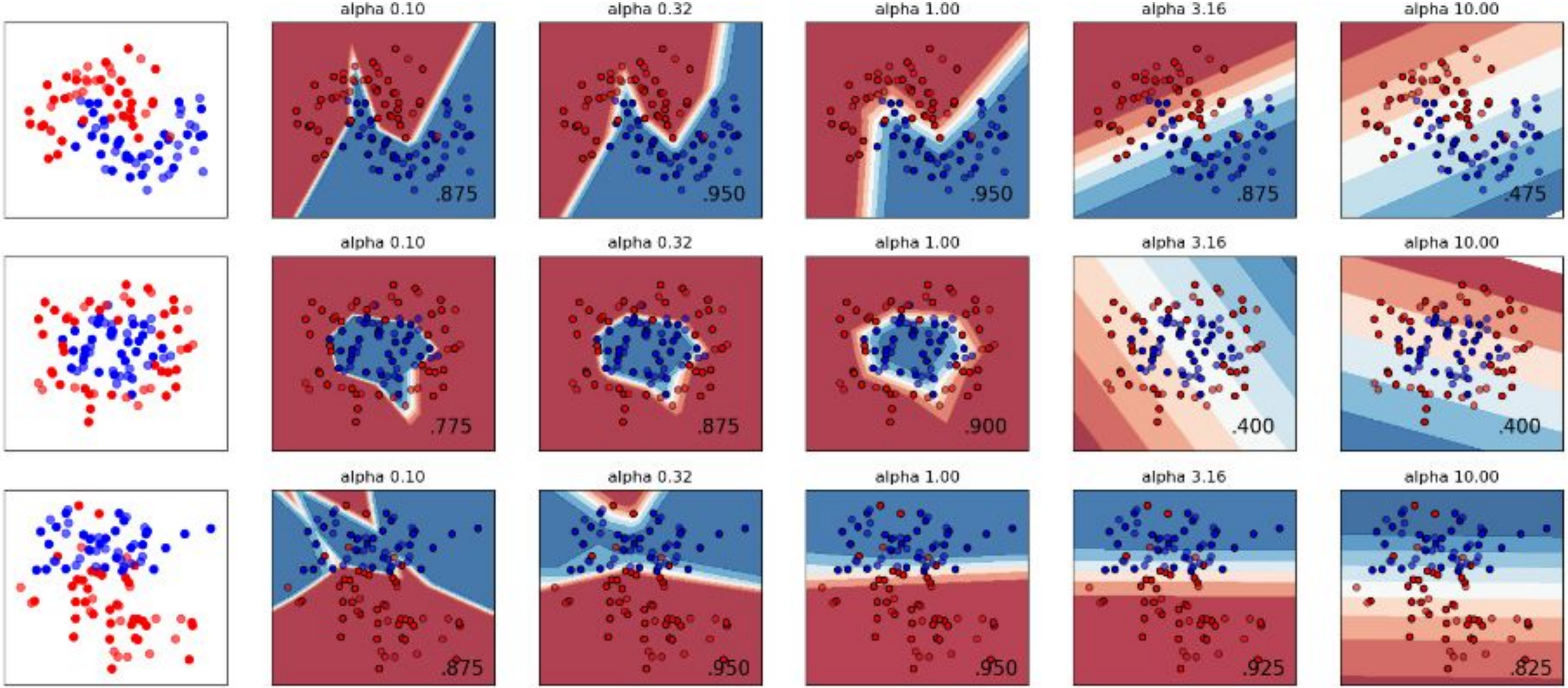

3 hidden neurons
6 hidden neurons
20 hidden neurons

Przypomnienie – problem przeuczenia (overfitting)

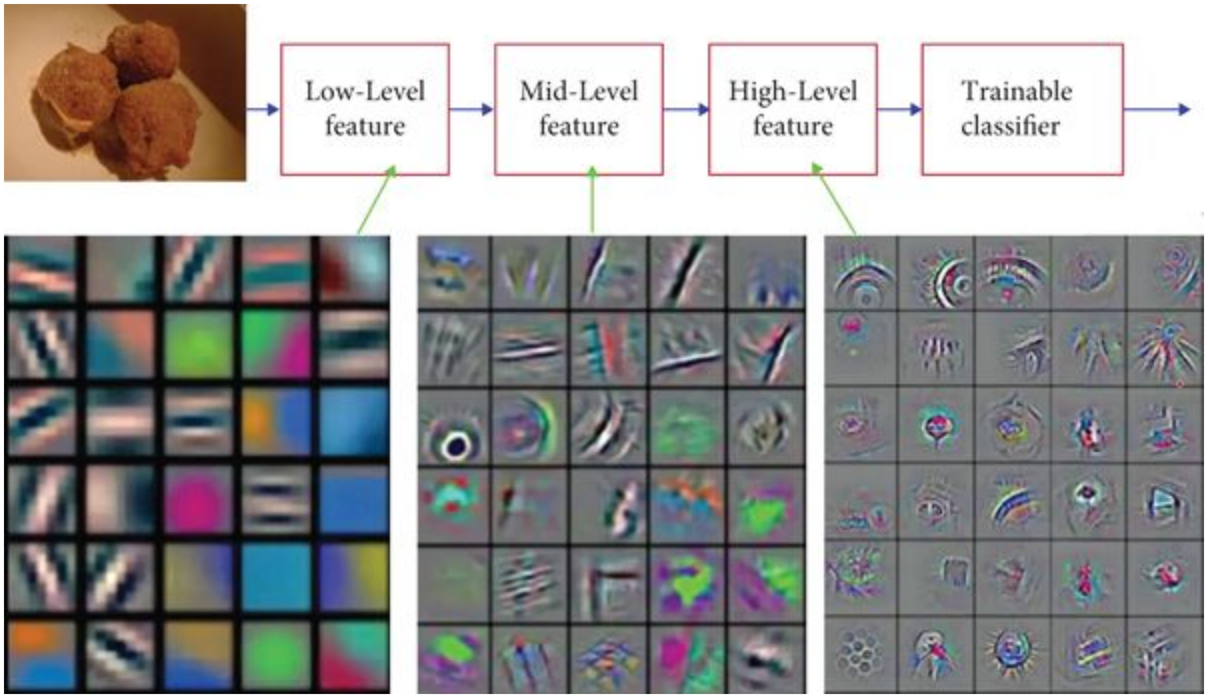




λ = 0.001
λ = 0.01
λ = 0.1


alpha 0.10
alpha 0.32
alpha 1.00
alpha 3.16
alpha 10.00
.875
.950
.950
.875
.475
alpha 0.10
alpha 0.32
alpha 1.00
alpha 3.16
alpha 10.00
.775
.875
.900
.400
.400
alpha 0.10
alpha 0.32
alpha 1.00
alpha 3.16
alpha 10.00
.875
.950
.950
.925
.825

Hierarchiczność/kompozycyjność reprezentacji



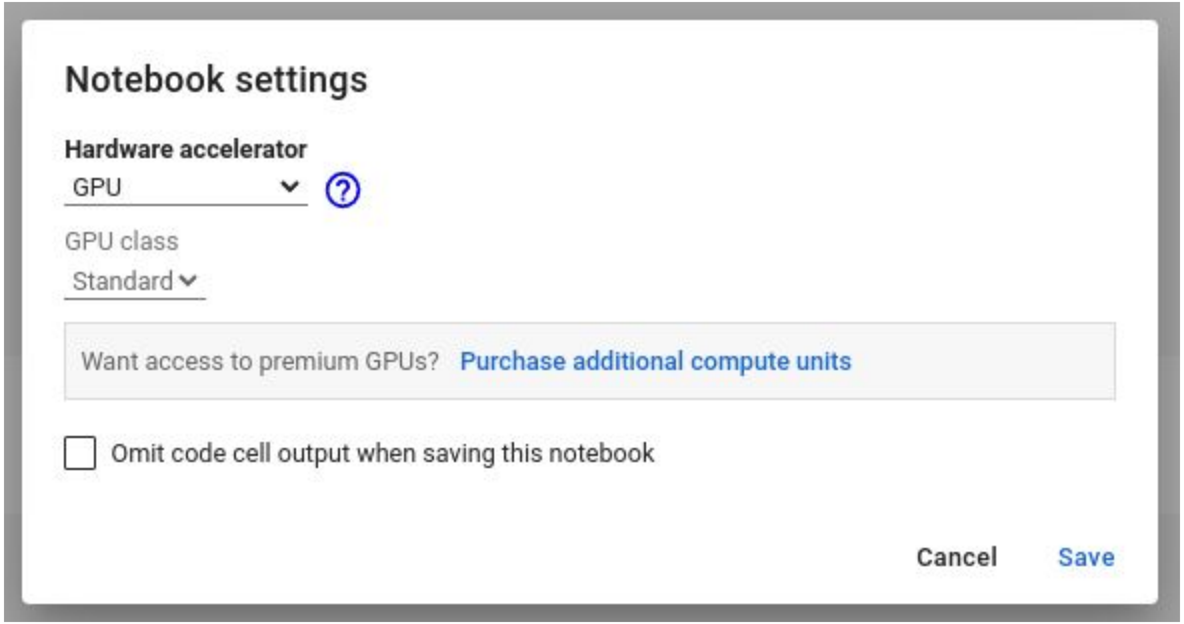
Zadanie labowe 0

- Przykłady na slajdach zostały stworzone z użyciem https://playground.tensorflow.org/
- Proszę poeksperymentować ze:
  - Zbiorami danych
  - Architekturą sieci neuronowej
  - Współczynnikiem uczenia (learning rate)
  - Funkcją aktywacji

Zadanie labowe 1

- Będziemy rozwiązywać zadanie klasyfikacji z użyciem sieci neuronowych
- Będziemy pracować na zbiorze danych CIFAR10
- Porównamy jakość modelu sieci neuronowej z modelem liniowym

Zadanie labowe 1



Zadanie labowe 1

```
!nvidia-smi
```

```
Thu Apr 13 07:00:44 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.85.12    Driver Version: 525.85.12    CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   71C    P8    11W /  70W |      0MiB / 15360MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Zadanie labowe 1

Krok 1: Import podstawowy bibliotek

```
[2] import torch
    import torch.nn as nn
    import torch.optim as optim
    import torchvision
    import torchvision.transforms as transforms
```

Zadanie labowe 1

Krok 2: Wczytanie zbioru danych CIFAR10

```
# Define the transforms to be applied to the CIFAR-10 dataset
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
```

Zadanie labowe 1

```
# Load the CIFAR-10 training and test datasets
trainset = torchvision.datasets.CIFAR10(
    root='./data',
    train=True,
    download=True,
    transform=transform
)
```

Zadanie labowe 1

```
trainloader = torch.utils.data.DataLoader(
    trainset,
    batch_size=4,
    shuffle=True,
    num_workers=2
)
```

Zadanie labowe 1

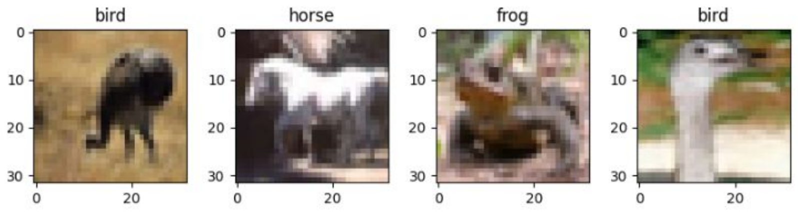
```python
testset = torchvision.datasets.CIFAR10(
    root='./data',
    train=False,
    download=True,
    transform=transform
)

testloader = torch.utils.data.DataLoader(
    testset,
    batch_size=4,
    shuffle=False,
    num_workers=2
)
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
100%|██████████| 170498071/170498071 [00:03<00:00, 43232169.68it/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
```

Zadanie labowe 1

Krok 3: Wizualizacja zbioru danych CIFAR10

```
import matplotlib.pyplot as plt
import numpy as np

# Define the classes for the CIFAR-10 dataset
classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

# Get some random training images
dataiter = iter(trainloader)
images, labels = dataiter.__next__()

# Show the images
fig, axes = plt.subplots(1, len(images), figsize=(10,2))
for i, image in enumerate(images):
    # Unnormalize the image
    image = image / 2 + 0.5
    np_image = image.numpy()

    # Transpose the channels to display the image
    transposed = np.transpose(np_image, (1, 2, 0))

    # Show the image
    axes[i].imshow(transposed)
    axes[i].set_title(classes[labels[i]])

# Display the images
plt.show()
```



Zadanie labowe 1

Krok 4. Definicja modelu sieci neuronowej oraz modelu liniowego

```
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(32*32*3, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, 10)

    def forward(self, x):
        """
          TODO: your code goes here.
          Hints:
            * flatten the image before passing it through layers
            * use self.fc1, self.fc2, self.fc3
            * apply nonlinearity nn.functional.relu after every layer
        """
        return x
```

Zadanie labowe 1

```python
class LinearModel(nn.Module):
  def __init__(self):
      super(LinearModel, self).__init__()
      self.linear = nn.Linear(32*32*3, 10)

  def forward(self, x):
        """
          TODO: your code goes here.
          Hints:
            * flatten the image before passing it through layers
            * use self.linear
        """
        return x
```

Zadanie labowe 1

Krok 5. Implementacja pętli uczącej

```
[ ] def train(net, criterion, optimizer, trainloader, device):
        running_loss = 0.0
        for i, data in enumerate(trainloader, 0):
            inputs, labels = data
            """
              TODO: move the inputs and labels to device
            """

            optimizer.zero_grad()

            """
              TODO:
                1. pass the inputs through the model to obtain the outputs
                2. calculate the criterion loss, based on the outputs and labels
                3. calculate the gradients using loss object
                4. perform the optimization step using optimizer
            """

            running_loss += loss.item()

        return running_loss / len(trainloader)
```

Zadanie labowe 1

Krok 6. Wyuczenie modeli

```python
# Define the device to be used
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

# Define the MLP model and optimizer
mlp_net = MLP().to(device)
mlp_criterion = nn.CrossEntropyLoss()
mlp_optimizer = optim.SGD(mlp_net.parameters(), lr=0.001, momentum=0.9)
"""
  TODO:
    experiment with the learning rate
"""
```

Zadanie labowe 1

```
# Define the linear model and optimizer
linear_net = LinearModel().to(device)
linear_criterion = nn.CrossEntropyLoss()
linear_optimizer = optim.SGD(linear_net.parameters(), lr=0.001, momentum=0.9)
"""
  TODO:
    experiment with the learning rate
"""
```

Zadanie labowe 1

```python
# Define the number of epochs
epochs = 10
"""
  TODO:
    experiment with length of the training, does adding more epochs helps?
"""

# Train the MLP model
for epoch in range(epochs):
    train_mlp_loss = train(mlp_net, mlp_criterion, mlp_optimizer, trainloader, device)
    print('Epoch %d MLP loss: %.3f' % (epoch + 1, train_mlp_loss))

# Train the linear model
for epoch in range(epochs):
    train_linear_loss = train(linear_net, linear_criterion, linear_optimizer, trainloader, device)
    print('Epoch %d Linear loss: %.3f' % (epoch + 1, train_linear_loss))
```

Zadanie labowe 1

```
Epoch 1 MLP loss: 1.649
Epoch 2 MLP loss: 1.419
Epoch 3 MLP loss: 1.304
Epoch 4 MLP loss: 1.215
Epoch 5 MLP loss: 1.136
Epoch 6 MLP loss: 1.066
Epoch 7 MLP loss: 0.993
Epoch 8 MLP loss: 0.930
Epoch 9 MLP loss: 0.870
Epoch 10 MLP loss: 0.810
Epoch 1 Linear loss: 2.153
Epoch 2 Linear loss: 2.106
Epoch 3 Linear loss: 2.080
Epoch 4 Linear loss: 2.067
Epoch 5 Linear loss: 2.057
Epoch 6 Linear loss: 2.057
Epoch 7 Linear loss: 2.046
Epoch 8 Linear loss: 2.046
Epoch 9 Linear loss: 2.039
Epoch 10 Linear loss: 2.030
```

Zadanie labowe 1

Krok 7. Ewaluacja modeli na zbiorze testowym

```
# Evaluate the models on the test set
mlp_correct = 0
linear_correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        images, labels = images.to(device), labels.to(device)
        mlp_outputs = mlp_net(images)
        _, mlp_predicted = torch.max(mlp_outputs.data, 1)
        linear_outputs = linear_net(images)
        _, linear_predicted = torch.max(linear_outputs.data, 1)
        total += labels.size(0)
        mlp_correct += (mlp_predicted == labels).sum().item()
        linear_correct += (linear_predicted == labels).sum().item()

mlp_accuracy = """ TODO: your code goes here """
linear_accuracy = """ TODO: your code goes here """

print('MLP Accuracy on the test images: %d %%' % mlp_accuracy)
print('Linear Accuracy on the test images: %d %%' % linear_accuracy)
```

```
MLP Accuracy on the test images: 53 %
Linear Accuracy on the test images: 33 %
```

Zadanie labowe 1

Krok 8. Wizualizacja predykcji obu modeli

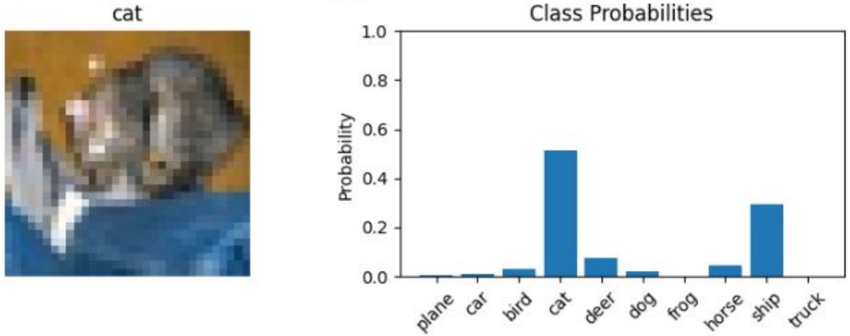
```
# Get a batch of test images
testloader = torch.utils.data.DataLoader(
    testset,
    batch_size=32,
    shuffle=False,
    num_workers=2
)

dataiter = iter(testloader)
images, labels = dataiter.__next__()

# Move the images to the device
images = images.to(device)

# Get the MLP predictions and output probabilities
mlp_outputs = mlp_net(images)
mlp_predicted = torch.max(mlp_outputs, 1)[1]
mlp_probs = torch.nn.functional.softmax(mlp_outputs, dim=1)

# Get the linear predictions and output probabilities
linear_outputs = linear_net(images.view(images.size(0), -1))
linear_predicted = torch.max(linear_outputs, 1)[1]
linear_probs = torch.nn.functional.softmax(linear_outputs, dim=1)
```

Zadanie labowe 1

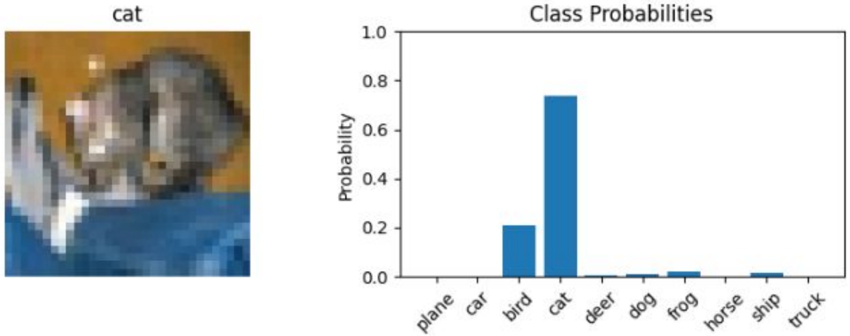
```
# Define the labels for the classes
classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

# Define a function to show an image along with its predicted class and probabilities
def imshow_probs(img, title, probs):
    img = img / 2 + 0.5     # unnormalize the image
    npimg = img.numpy()
    fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(8,3))
    ax1.imshow(np.transpose(npimg, (1, 2, 0)))
    ax1.set_title(title)
    ax1.axis('off')
    ax2.bar(classes, probs)
    ax2.set_xticklabels(classes, rotation=45)
    ax2.set_ylim([0, 1])
    ax2.set_ylabel('Probability')
    ax2.set_title('Class Probabilities')
    fig.tight_layout()

# Show the images with their predicted classes and probabilities for the MLP model
for i in range(10):
    print("MLP:")
    mlp_probs_i = mlp_probs[i].cpu().detach().numpy()
    imshow_probs(images[i].cpu(), classes[mlp_predicted[i]], mlp_probs_i)
    plt.show()
    print("Linear:")
    linear_probs_i = linear_probs[i].cpu().detach().numpy()
    imshow_probs(images[i].cpu(), classes[linear_predicted[i]], linear_probs_i)
    plt.show()
```

Zadanie labowe 1

```
MLP:
<ipython-input-11-f1c0206d2b07>:37: UserWarning: FixedFormatter should only be used together with FixedLocator
  ax2.set_xticklabels(classes, rotation=45)
```

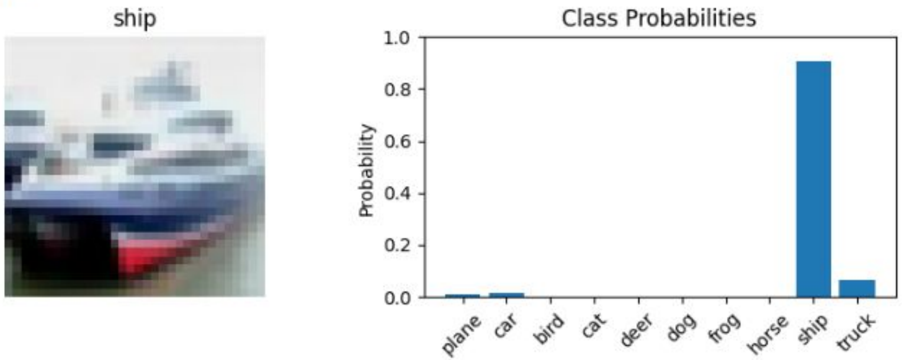



```
Linear:
```

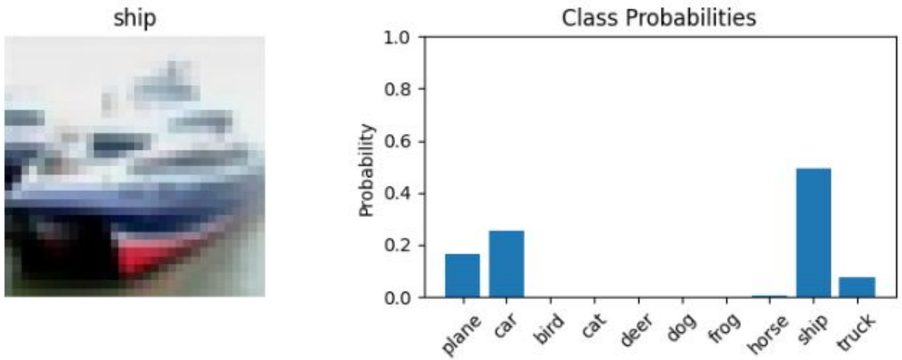


Zadanie labowe 1

MLP:

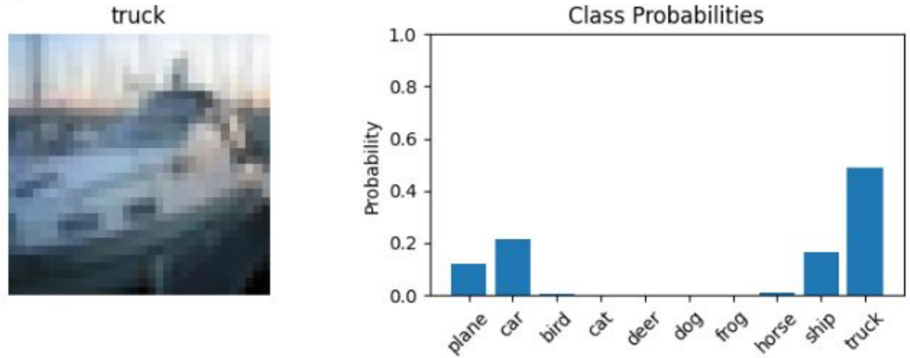


Linear:

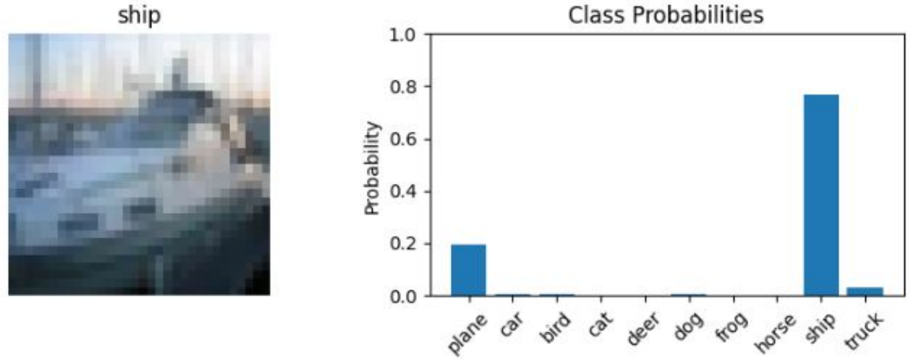


Zadanie labowe 1

MLP:

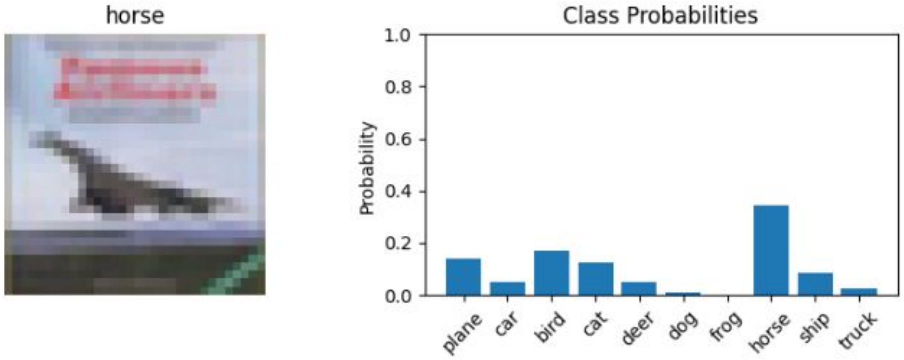


Linear:

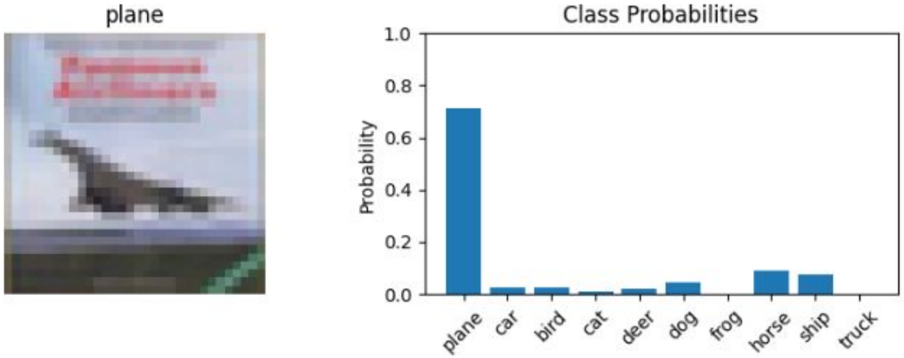


Zadanie labowe 1

MLP:

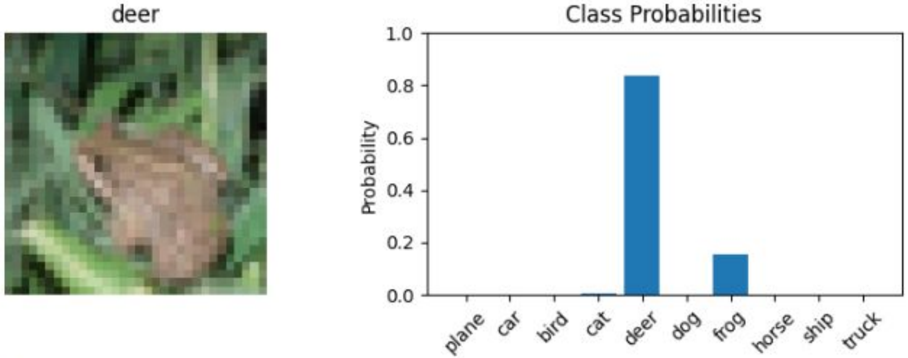


Linear:

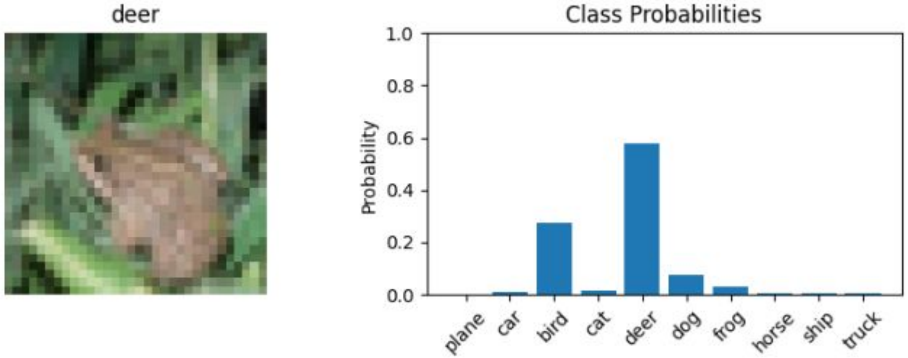


Zadanie labowe 1

MLP:

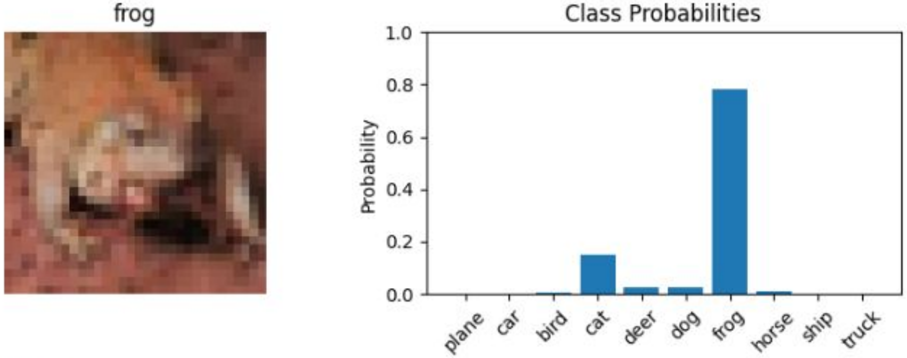


Linear:

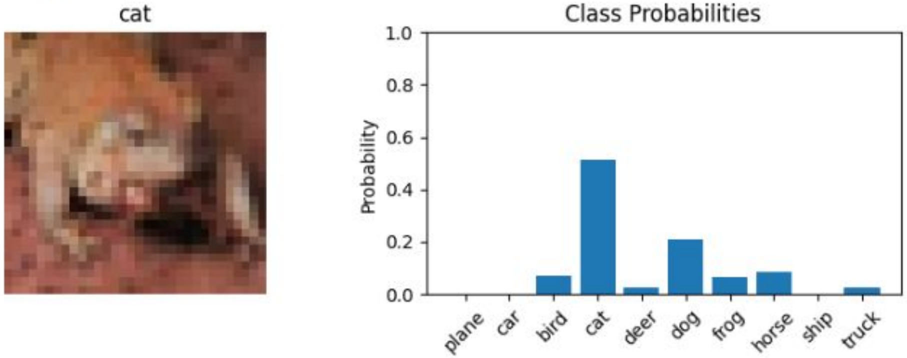


Zadanie labowe 1

MLP:

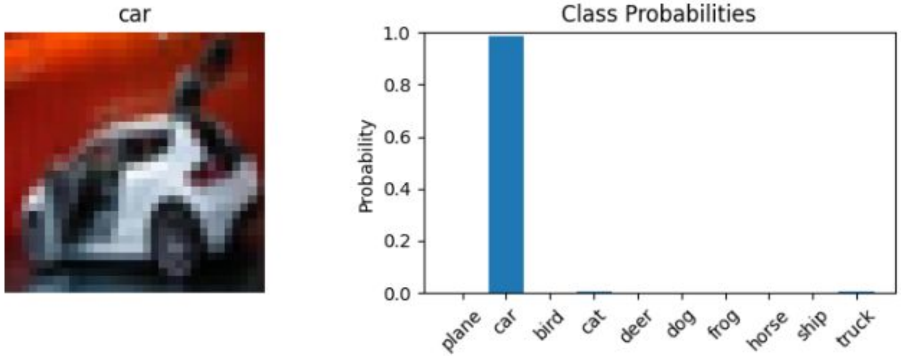


Linear:

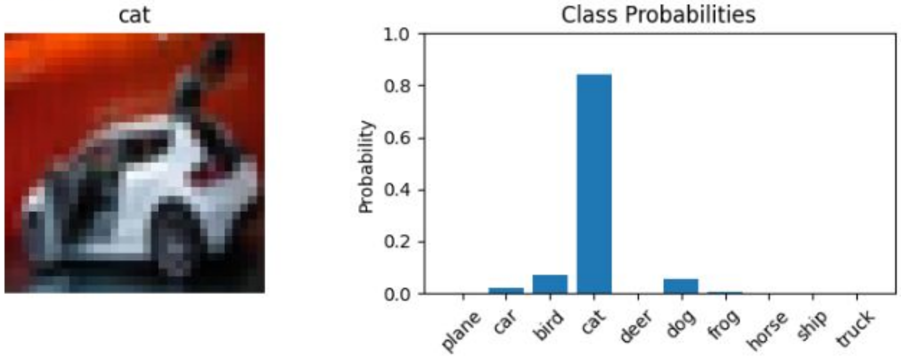


Zadanie labowe 1

MLP:

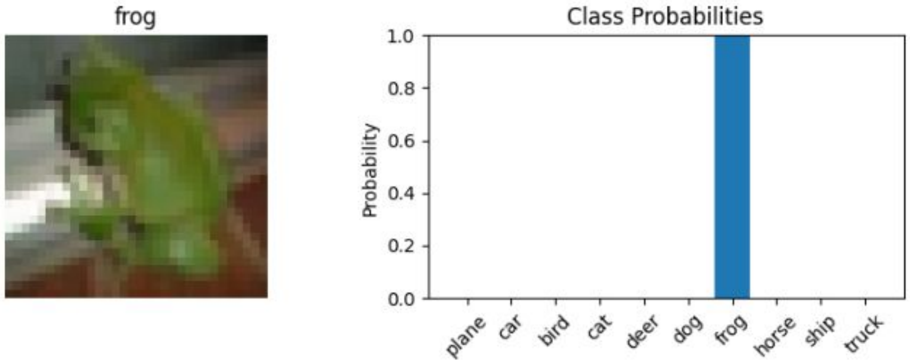


Linear:

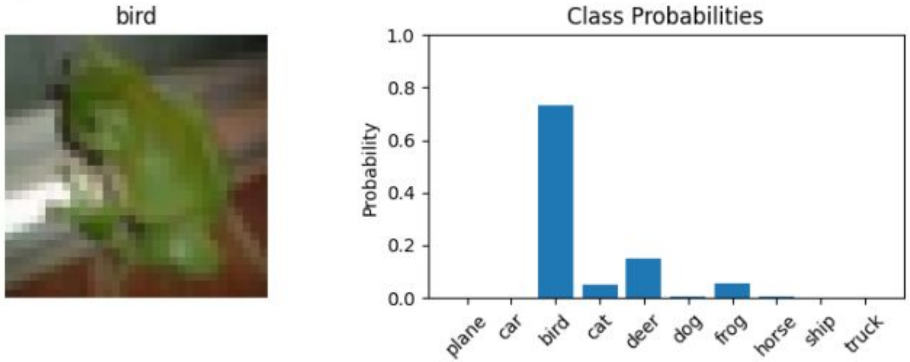


Zadanie labowe 1

MLP:

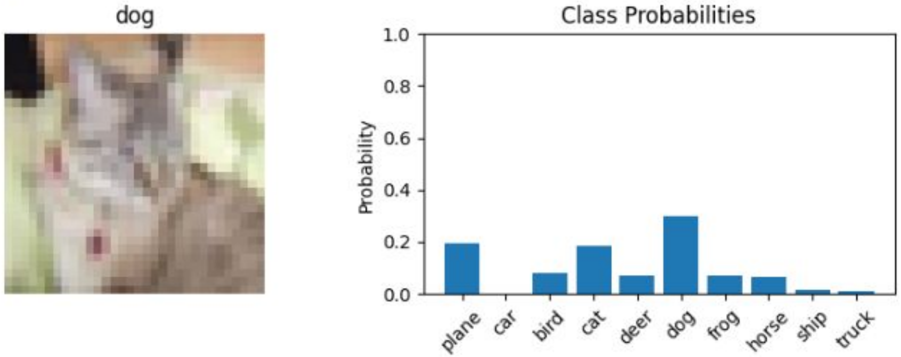


Linear:

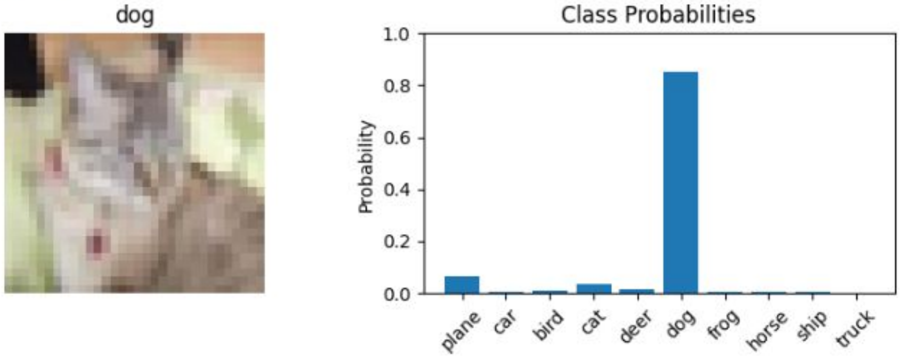


Zadanie labowe 1

MLP:

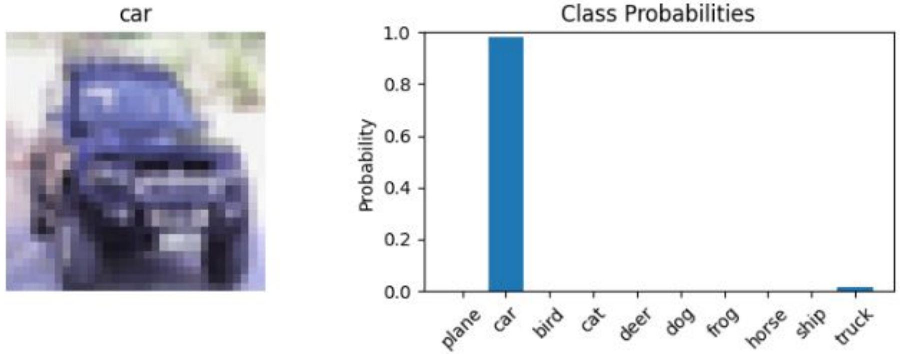


Linear:



Zadanie labowe 1

MLP:



Linear: