PyTorch

# Machine Learning & Deep Learning (Barcha uchun)

## <06> Mantiqiy Regressiya (Logistic Regression)

Mansurbek Abdullaev

https://uzbek.gitbook.io/ai/

mansurbek.comchemai@gmail.com

@MansurbekUST

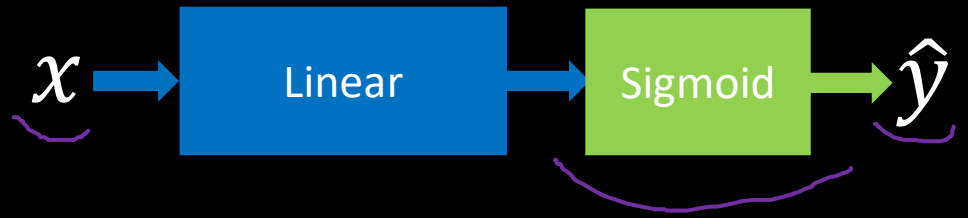
# Chiziqli model


| Soat (x) | Baho(y) |
|---|---|
| 1 | 2 |
| 2 | 4 |
| 3 | 6 |
| 4 | ? |

# Ikkilik bashorat (0 yoki 1) (binary prediction)

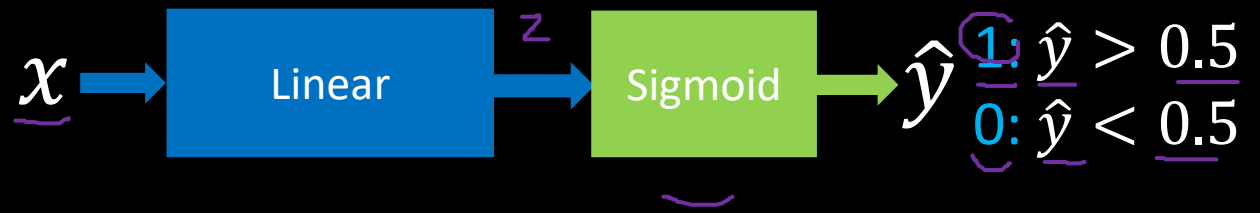
- N soat o’qiganda, imtihondan o’tish yoki yiqilish?
- GPA va GRE natijalarga ko’ra Oxford universitetiga, qabul qilinish yoki qilinmaslik?
- O’zbekiston Janubiy Koreyaga qarshi futbol o’ynaganda, yutish yoki yutqazishi?
- ……

# Chiziqlidan → Ikkilikga (Yo'q/Ha---0/1)

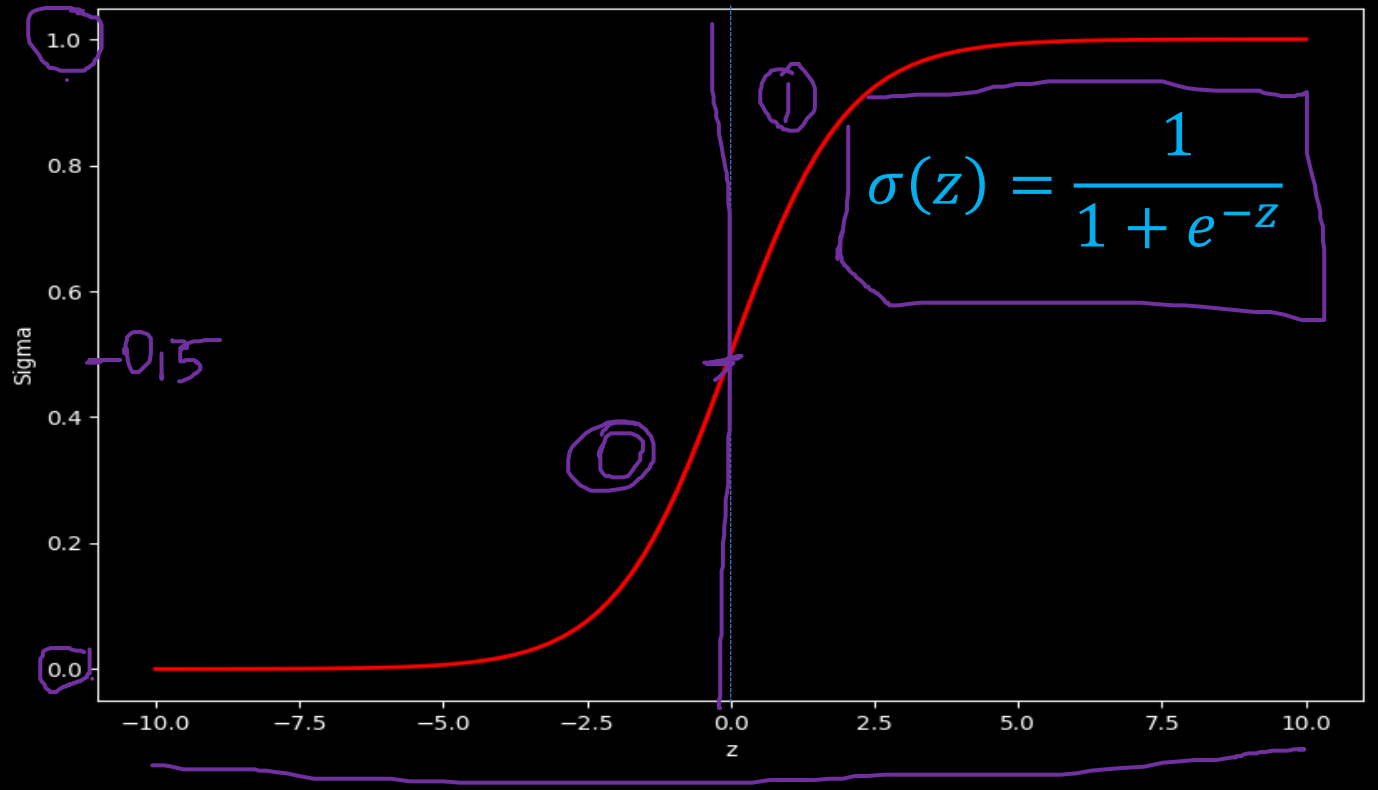




| Soat (x) | Baho(y) | Ha/Yo'q |
|---|---|---|
| 1 | 2 | 0 |
| 2 | 4 | 0 |
| 3 | 6 | 1 |
| 4 | ? | ? |

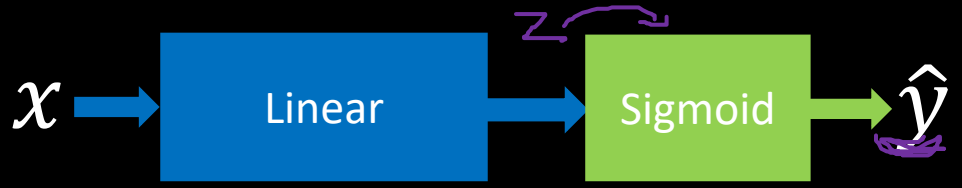
# Sigmoid funksiyasi


| Soat (x) | Baho(y) | Ha/Yo'q |
|---|---|---|
| 1 | 2 | 0 |
| 2 | 4 | 0 |
| 3 | 6 | 1 |
| 4 | ? | ? |



# Sigmoid funksiyasi

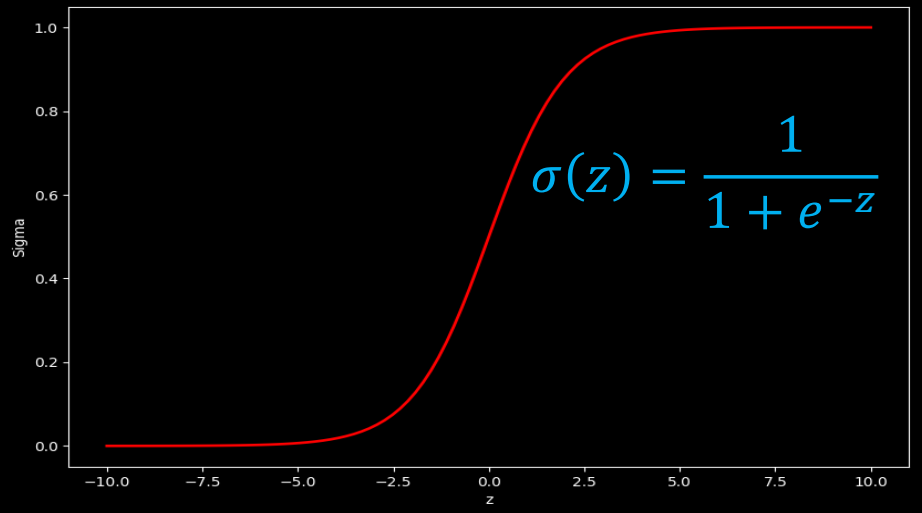




$$\hat{y} = x * w + b$$

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

$$\hat{y} = \sigma(x * w + b)$$

| Soat (x) | Baho(y) | Ha/Yo'q |
|---|---|---|
| 1 | 2 | 0 |
| 2 | 4 | 0 |
| 3 | 6 | 1 |
| 4 | ? | ? |



# Cross entropy loss



$$\hat{y} = x * w + b$$

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

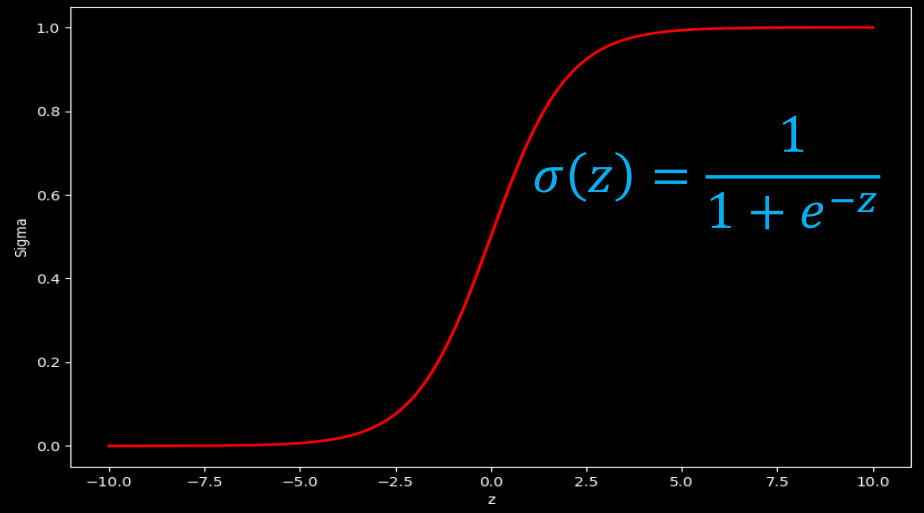
$$\hat{y} = \sigma(x * w + b)$$

MSE

$$loss = \frac{1}{N}\sum_{n=1}^{N}(\hat{y}_n - y_n)^2$$

$$loss = -\frac{1}{N}\sum_{n=1}^{N} y_n log\hat{y}_n + (1 - y_n)\log(1 - \hat{y}_n)$$

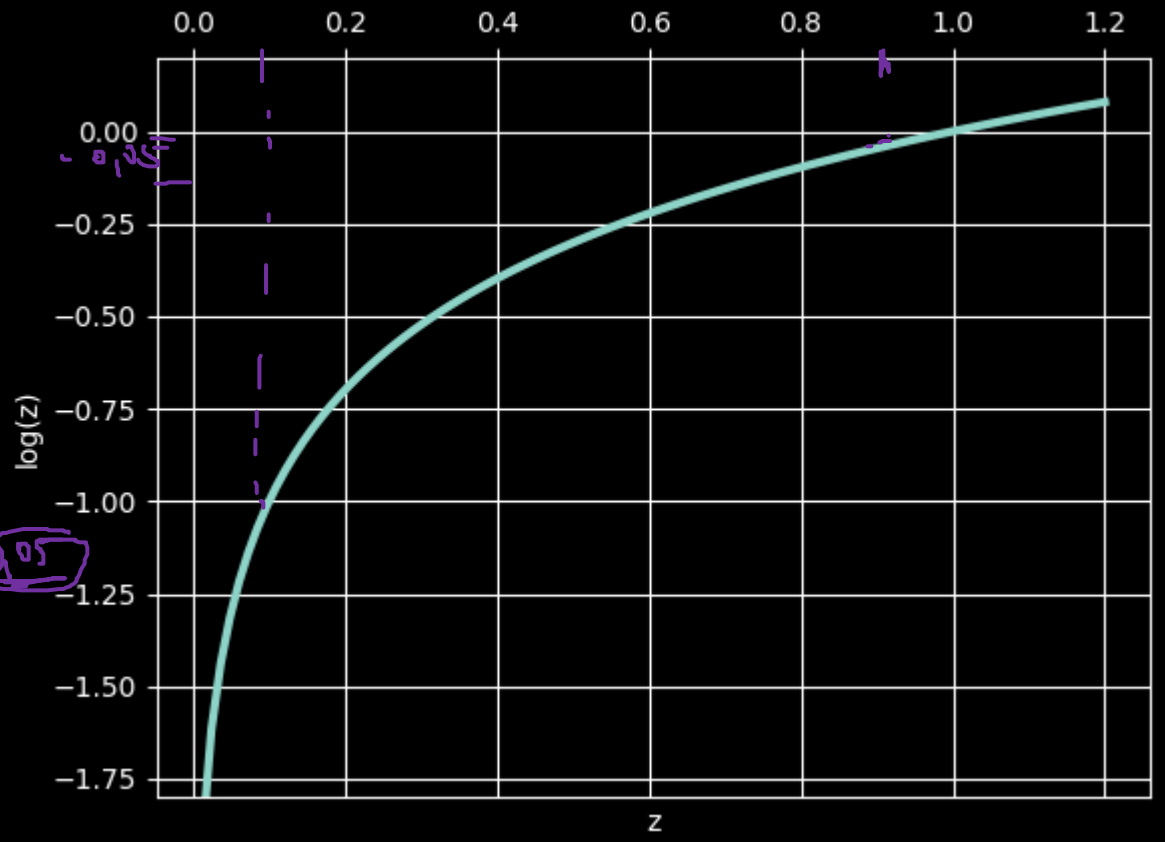
| Soat (x) | Baho(y) | Ha/Yo'q |
|---|---|---|
| 1 | 2 | 0 |
| 2 | 4 | 0 |
| 3 | 6 | 1 |
| 4 | ? | ? |



# (Binary) Cross entropy loss

$$loss = -\frac{1}{N}\sum_{n=1}^{N} y_n log\hat{y}_n + (1 - y_n)\log(1 - \hat{y}_n)$$

| y | y_pred | loss |
|---|---|---|
| 1 | 0.2 | |
| 1 | 0.8 | |
| 0 | 0.1 | $-1 \cdot \log(1-0.1) \simeq -1 \cdot (-0.05) = 0.05$ |
| 0 | 0.9 | $-1 \cdot \log(1-0.9) \simeq -1 \cdot (-1) = 1$ |



# Mantiqiy regressiya (Logistic regression)


$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

$$\hat{y} = \sigma(x * w + b)$$

```python
#(1) Class yordamida model qurib olish --> "Model"
class Model(torch.nn.Module):
    def __init__(self):
        #Bu yerda nn.Module bu yerda super class(Pytorch)
        super().__init__()
        #torch.nn.Linear(#kirish, #chiqish) chiziqli model
        self.linear = torch.nn.Linear(1,1) #1ta kirish & 1ta chiqish
    #Metod yordamida to'g'ri hisoblash arxitikturasini kiritamiz(forward pass)
    def forward(self, x):
        y_pred = torch.sigmoid(self.linear(x))
        return y_pred
```

$$= -\frac{1}{N}\sum_{n=1}^{N} y_n log\hat{y}_n + (1 - y_n)\log(1 - \hat{y}_n)$$

```python
criterion = torch.nn.BCELoss(reduction='mean')
```

# Documentation

## SIGMOID

CLASS `torch.nn.Sigmoid` [SOURCE]

Applies the element-wise function:

$$\text{Sigmoid}(x) = \sigma(x) = \frac{1}{1 + \exp(-x)}$$

Shape:

- Input: $(N, *)$ where * means, any number of additional dimensions
- Output: $(N, *)$, same shape as the input

## BCELOSS

CLASS `torch.nn.BCELoss(weight: Optional[torch.Tensor] = None, size_average=None, reduce=None, reduction: str = 'mean')` [SOURCE]

Creates a criterion that measures the Binary Cross Entropy between the target and the output:

The unreduced (i.e. with `reduction` set to `'none'`) loss can be described as:

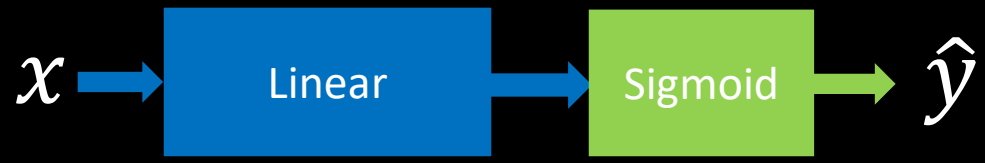
$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = -w_n \left[ y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n) \right],$$

# To’liq kod

```
 kutubxonalarni chaqirib olish
orch
tlarni tensor ko'rinishida yuklab olish
 torch.Tensor([[1.],
               [2.],
               [3.],
               [4.]])
k = torch.Tensor([[0.],
               [0.],
               [1.],
               [1.]])

ss yordamida model qurib olish --> "Model"
del(torch.nn.Module):
__init__(self):
#Bu yerda nn.Module bu yerda super class(Pytorch)
super().__init__()
#torch.nn.Linear(#kirish, #chiqish) chiziqli model
self.linear = torch.nn.Linear(1,1) #1ta kirish & 1ta chiqish
od yordamida to'g'ri hisoblash arxitiktu rasini kiritamiz(forward pass)
forward(self, x):
y_pred = torch.sigmoid(self.linear(x))
return y_pred

 model    Z
del()
model)
s va optimizer larni tanlab olish
n = torch.nn.BCELoss(reduction='mean')
r = torch.optim.SGD(model.parameters(), lr=0.01)
ining(3.1),   Backward(3.2), Step(3.3)
>Training
h in range(1000):    #Epochlar soni 1000
ed = model(x_soat)
s|||xatolikni hisoblash va chop qilish
 = criterion(y_pred, y_ikkilik)
t(f'Epoch: {epoch} | Loss: {loss.item()} ')

mizer.zero_grad() #Har bir epoch uchun grad ni 0 ga tenglashtirib olish
2)-->Backpropagation|||Teskari hisoblash
.backward()
3)--> Step||| w ning qiymatini  yangilash
mizer.step()
t uchun qiymat||| Ushbu qiymatimiz ham tensor bo'lishi kerak
\n Trainingdan so'ng bashorat qilib ko'ramiz \n{'=' * 50}")
 uchun bashorat
 = model(torch.tensor([[1.0]]))
1 soat o'qilganda imtihondan o'ta olish: {hour_var.item():.4f} |  50% dan yuqori: {hour_var.item() > 0.5}")
 uchun bashorat
```

## Class (OOP) yordamida modelni qurib olish

1




2 Loss va optimizer larni tanlash (PyTorch API dan)

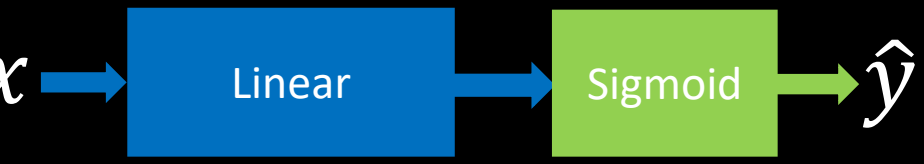
3 O’rgatish (Training) sikli → forward, backward, step

# To’liq kod

```
 kutubxonalarni chaqirib olish
orch
tlarni tensor ko'rinishida yuklab olish
 torch.Tensor([[1.],
               [2.],
               [3.],
               [4.]])
k = torch.Tensor([[0.],
               [0.],
               [1.],
               [1.]])

ss yordamida model qurib olish --> "Model"
del(torch.nn.Module):
__init__(self):
#Bu yerda nn.Module bu yerda super class(Pytorch)
super().__init__()
#torch.nn.Linear(#kirish, #chiqish) chiziqli model
self.linear = torch.nn.Linear(1,1) #1ta kirish & 1ta chiqish
od yordamida to'g'ri hisoblash arxitikturasini kiritamiz(forward pass)
forward(self, x):
y_pred = torch.sigmoid(self.linear(x))
return y_pred

 model    Z
del()
model)
s va optimizer larni tanlab olish
n = torch.nn.BCELoss(reduction='mean')
r = torch.optim.SGD(model.parameters(), lr=0.01)
ining(3.1),   Backward(3.2), Step(3.3)
>Training
h in range(1000):    #Epochlar soni 1000
ed = model(x_soat)
s|||xatolikni hisoblash va chop qilish
 = criterion(y_pred, y_ikkilik)
t(f'Epoch: {epoch} | Loss: {loss.item()} ')

mizer.zero_grad() #Har bir epoch uchun grad ni 0 ga tenglashtirib olish
2)-->Backpropagation|||Teskari hisoblash
.backward()
3)--> Step||| w ning qiymatini  yangilash
mizer.step()
t uchun qiymat||| Ushbu qiymatimiz ham tensor bo'lishi kerak
\n Trainingdan so'ng bashorat qilib ko'ramiz \n{'=' * 50}")
 uchun bashorat
 = model(torch.tensor([[1.0]]))
1 soat o'qilganda imtihondan o'ta olish: {hour_var.item():.4f} |  50% dan yuqori: {hour_var.item() > 0.5}")
 uchun bashorat
```

```
Epoch: 949 | Loss: 0.4715662598
Epoch: 950 | Loss: 0.4714177250
Epoch: 951 | Loss: 0.4712693393
Epoch: 952 | Loss: 0.4711209833
Epoch: 953 | Loss: 0.4709728062
Epoch: 954 | Loss: 0.4708246886
Epoch: 955 | Loss: 0.4706766605
Epoch: 956 | Loss: 0.4705287516
Epoch: 957 | Loss: 0.4703809022
Epoch: 958 | Loss: 0.4702331721
Epoch: 959 | Loss: 0.4700855314
Epoch: 960 | Loss: 0.4699379801
Epoch: 961 | Loss: 0.4697905182
Epoch: 962 | Loss: 0.4696431756
Epoch: 963 | Loss: 0.4694959521
Epoch: 964 | Loss: 0.4693488180
Epoch: 965 | Loss: 0.4692017436
Epoch: 985 | Loss: 0.46628084778785706
Epoch: 986 | Loss: 0.46613579988479614
Epoch: 987 | Loss: 0.46599081158638
Epoch: 988 | Loss: 0.4658459722995758
Epoch: 989 | Loss: 0.46570122241973877
Epoch: 990 | Loss: 0.4655565619468689
Epoch: 991 | Loss: 0.4654119908809662
Epoch: 992 | Loss: 0.46526750922203064
Epoch: 993 | Loss: 0.46512308716773987
Epoch: 994 | Loss: 0.4649788439273834
Epoch: 995 | Loss: 0.46483466029167175
Epoch: 996 | Loss: 0.46469053626060486
Epoch: 997 | Loss: 0.4645466208457947
Epoch: 998 | Loss: 0.4644026756286621
Epoch: 999 | Loss: 0.4642588198184967

 Trainingdan so'ng bashorat qilib ko'ramiz
==================================================
1 soat o'qilganda imtihondan o'ta olish: 0.3904 |  50% dan yuqori: False
7 soat o'qilganda imtihondan o'ta olish: 0.9676 |  50% dan yuqori : True
```

# Vazifa 6-1:




| | |
|---|---|
| `nn.ReLU` | Applies the rectified linear unit function element-wise: |
| `nn.ReLU6` | Applies the element-wise function: |
| `nn.RReLU` | Applies the randomized leaky rectified liner unit function, element-wise, as described in the paper: |
| `nn.SELU` | Applied element-wise, as: |
| `nn.CELU` | Applies the element-wise function: |
| `nn.GELU` | Applies the Gaussian Error Linear Units function: |
| `nn.Sigmoid` | Applies the element-wise function: |
| `nn.SiLU` | Applies the silu function, element-wise. |
| `nn.Softplus` | Applies the element-wise function: |
| `nn.Softshrink` | Applies the soft shrinkage function elementwise: |
| `nn.Softsign` | Applies the element-wise function: |