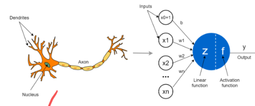
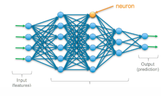
Day 3 / 100
Learning how to Assemble Neurons

Day 3 - Assemble Neuron
Day 4 - Use Actvation Functin
Day 5 - Full Model Class

What is Neuron

A neuron in deep learning is a **math version of a brain neuron** that receives information, decides if it matters, and passes it forward.





1. It **receives signals** from other neurons
2. It **decides** whether the signal is strong enough
3. If yes, it **sends a signal forward**

1. **Input comes in** Tensor
2. Each input has a **weight**
3. Everything is **added together**
4. A rule (activation function) decides:
   - ❌ Ignore it
   - ✅ Pass it forward

Brain Neuron 🧠	AI Neuron 💻
Receives signals	Receives numbers
Signal strength	Weight
Decision to fire	Activation function
Sends signal	Output value

Types of Neuron

Neurons

Basic
- Linear
- Flatten
- Dropout
- Batch Normalization
- Sequential

Output
- Softmax
- Sigmoid
- LogSoftmax

CNN
- Conv2D
- Conv1D
- Conv3D
- Deconvolutional

Pooling
- Max Pooling
- Average Pooling
- Adaptive Pooling

RNN
- RNN
- LSTM
- GRU

Attention Layers
- Embedding
- MultiHeadAttention
- Transformer Encoded Layer
- Transformer Decoded Layer

Etc
- Layer Normalize
- Alpha Dropout
- Noise Layer
- Graph Attention / Conv
- etc.....

Neuron Layer Stacking looks

1.Import torch and torch.neural net
2.Create class (M)
3.Create __init__
4.Call Super init
5.Init all Layer going to be used
6.Create Forward
7.Make the flow
8.Return the pred

```python
import torch
import torch.nn as nn

class SimpleANN(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(2, 2)
        self.fc2 = nn.Linear(2, 1)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x

model =SimpleANN()
x = torch.tensor([1.0, 2.0])
print(model(x))
```

A toolbox for building neural networks

nn.Module
To Track weights & gradient
To add GPU compactibility
A NN is about to be made

in Init: We define all Layers

super - Help parent class get ready
pass our layers

Actual Math
How we build layer by layer

Rule# Every PyTorch model must inherit from nn.Module.

#forward is called as soon as model init
#init is called as soon as code run

About Neurons

$$O = \sum w_i I + b$$

#1. Linear Layer

input = [3, 5]
weights = [0.2, 0.4]
bias = 0.1

output = (3 × 0.2) + (5 × 0.4) + 0.1
= 0.6 + 2.0 + 0.1
= 2.7

#2. Sequential Layer

Input (4 values)
→ Linear (3 neurons)
→ ReLU
→ Dropout
→ Linear (1 neuron)
→ Output

```python
model = nn.Sequential(
    nn.Linear(4, 3),
    nn.ReLU(),
    nn.Dropout(0.5),
    nn.Linear(3, 1)
)
```

#Rule 1. Dimension Matching Rule

(out_features of previous layer) = (in_features of next layer)

Input shape

```python
nn.Linear(10, 20),
nn.Linear(20, 5)
```

Output shape

```python
nn.Linear(8, 100)
nn.Linear(100,10000)
nn.Linear(10000,10000000)
nn.Linear(10000000, 10)
```

Stack Neurons

define in Init

```python
nn.Sequential(
    nn.Linear(10, 20),
    nn.Linear(20, 5)
)
```

in forward pass x one by one :

```python
x = self.fc1(x)
x = self.fc2(x)
```

Just call this in forward()

```python
x = self.fc2(self.fc1(x))
```

#3. Dropout Layer

Dropout **randomly turns off neurons** during training

```python
dropout = nn.Dropout(p=0.5)
```

```
[10, 20, 30, 40]
[10, 0, 30, 0]   #50% are killed
```

#to prevent overfitting

#4. Flatten Layer

Flatten **reshapes data into a single line**.

```
[1, 2]
[3, 4]
[1, 2, 3, 4]
```

1. **Neural networks expect 1D input**
2. **Images are 2D / 3D**

#5. Batch Normalization

It rescales values so learning doesn't go crazy.

```python
bn = nn.BatchNorm1d(3)
```

```
[2, 200, 4000]

[-0.5, 0.1, 0.4]   (same meaning, safer range)
```

```python
model = nn.Sequential(
    nn.Flatten(),          # reshape
    nn.Linear(4, 3),       # learn
    nn.BatchNorm1d(3),     # stabilize
    nn.Dropout(0.5),       # regularize
    nn.Linear(3, 1)        # final output
)
```

About In and Out

```python
nn.Linear(x, y)
```

- Incoming Tensor (flatten if needed)
- Output Value
  - **Regression 1**
  - **Classification n**

operates on the *last dimension* of a tensor

```python
torch.tensor(5.0)   # shape: ()
```

```python
torch.tensor([1.0, 2.0, 3.0])
```
→ `nn.Linear(3, y)`

```python
x = torch.tensor([
    [1.0, 2.0, 3.0],
    [4.0, 5.0, 6.0]
])
```
2×3 → `layer = nn.Linear(3, y)`

```python
x = torch.randn(4, 10, 8)
```
→ `nn.Linear(8, y)`

See you in Practise Session Today

Here, Practise is Important

Neural Stacking is Foundation for whole DL