Τεχνητή Νοημοσύνη και Μηχανική Μάθηση

Κωνσταντίνος Καραμανής

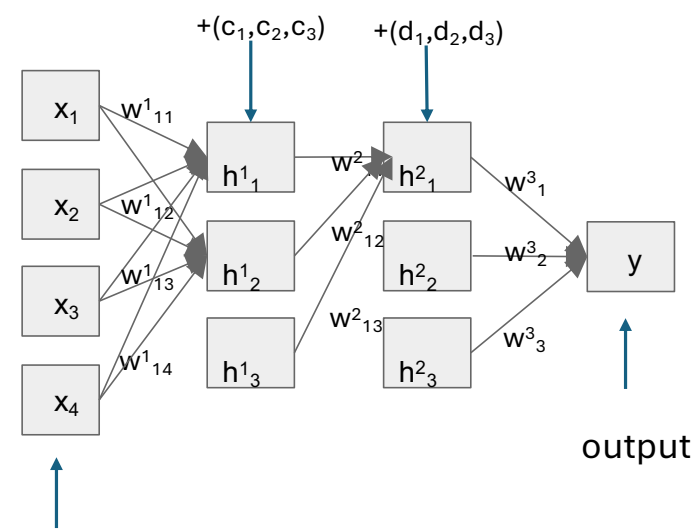
The University of Texas at Austin & Archimedes/Athena RC

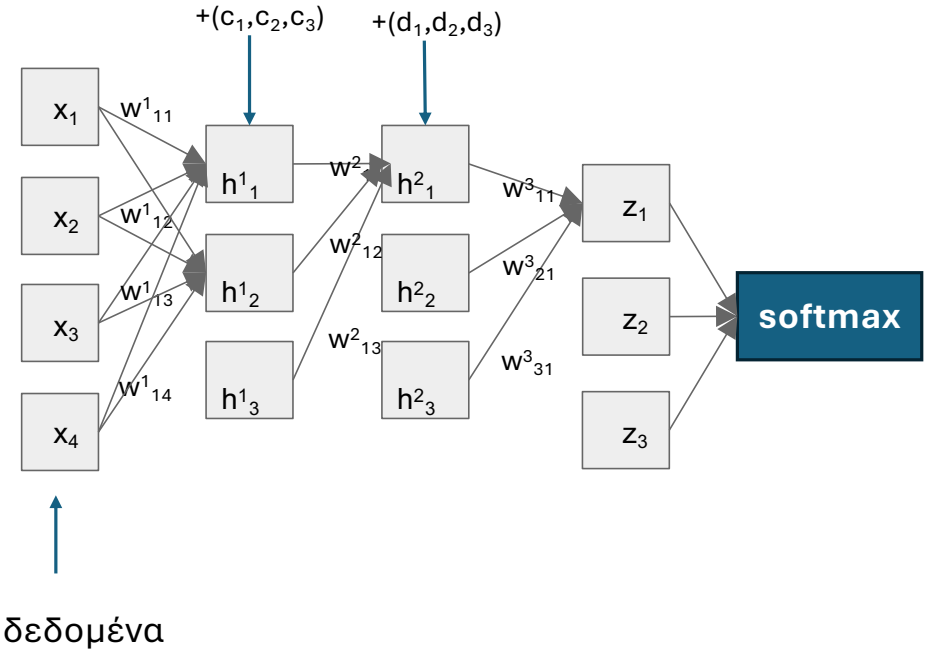
constantine@utexas.edu

https://caramanis.github.io/

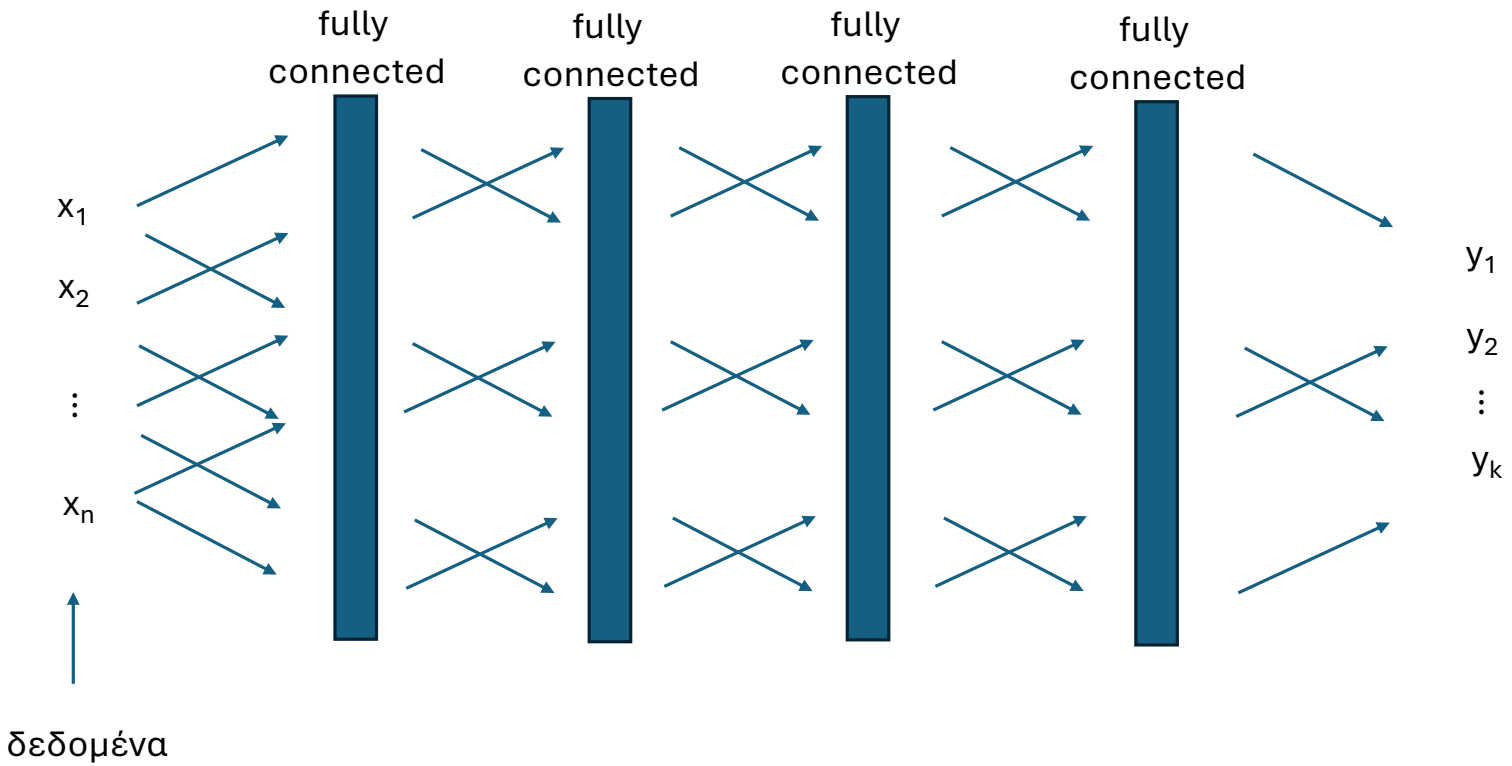
Ας θυμηθούμε τα προηγούμενα...

Fully Connected + Softmax

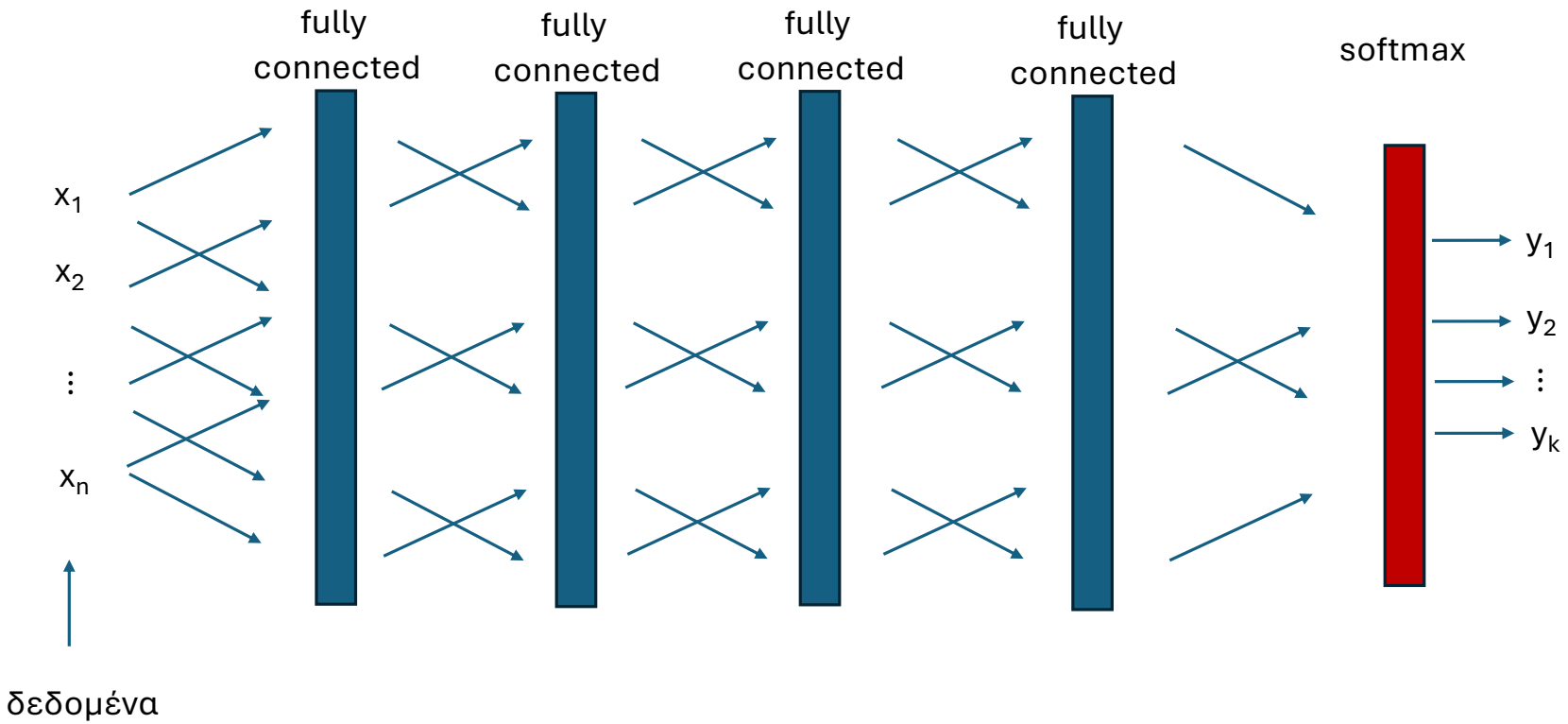




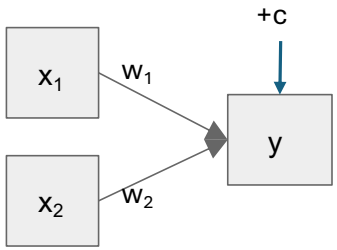
Fully Connected + ReLU

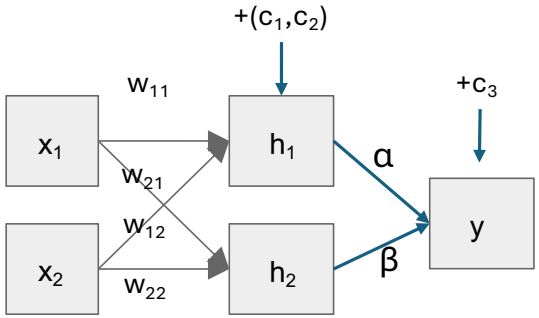


Fully Connected + ReLU

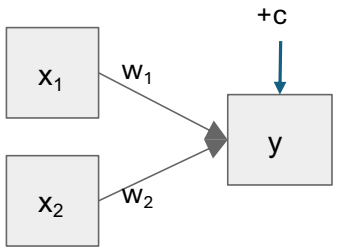


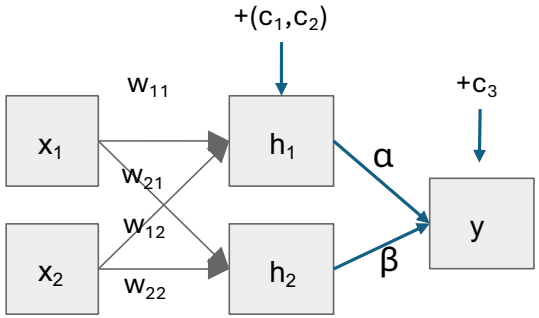
Αρκούν τα Γραμμικά Επίπεδα;





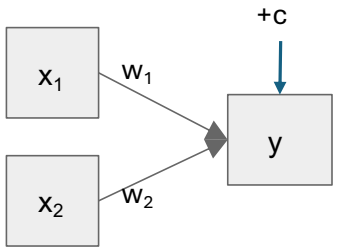
Αρκούν τα Γραμμικά Επίπεδα;

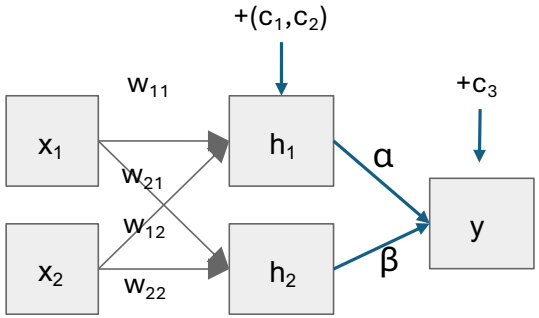




$$y = w_1 x_1 + w_2 x_2 + c$$

Αρκούν τα Γραμμικά Επίπεδα;

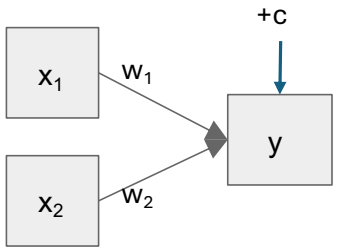


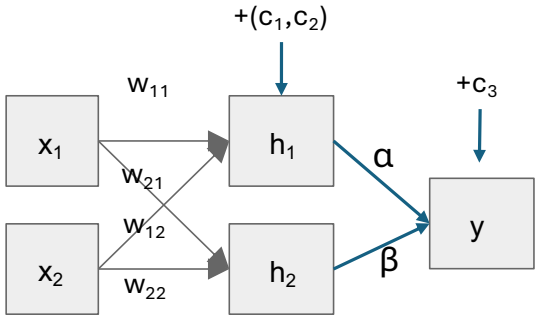


$y = w_1 x_1 + w_2 x_2 + c$

$y = \alpha h_1 + \beta h_2 + c_3$

Αρκούν τα Γραμμικά Επίπεδα;

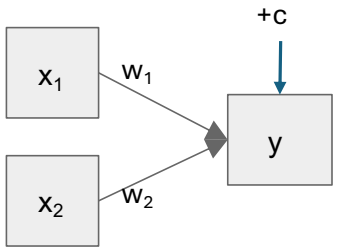


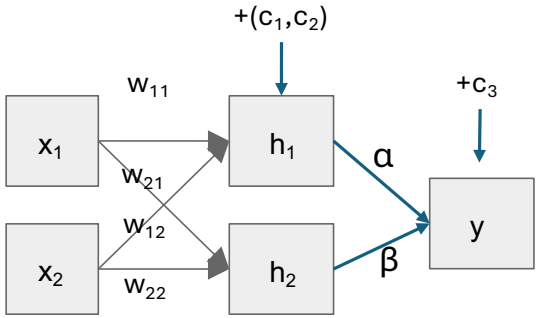


$$y = w_1 x_1 + w_2 x_2 + c$$

$$y = \alpha h_1 + \beta h_2 + c_3 = \alpha(w_{11} x_1 + w_{12} x_2 + c_1) + \beta(w_{21} x_1 + w_{22} x_2 + c_2) + c_3$$

Αρκούν τα Γραμμικά Επίπεδα;





$$y = w_1 x_1 + w_2 x_2 + c$$

$$y = \alpha h_1 + \beta h_2 + c_3 = \alpha(w_{11}x_1 + w_{12}x_2 + c_1) + \beta(w_{21}x_1 + w_{22}x_2 + c_2) + c_3$$

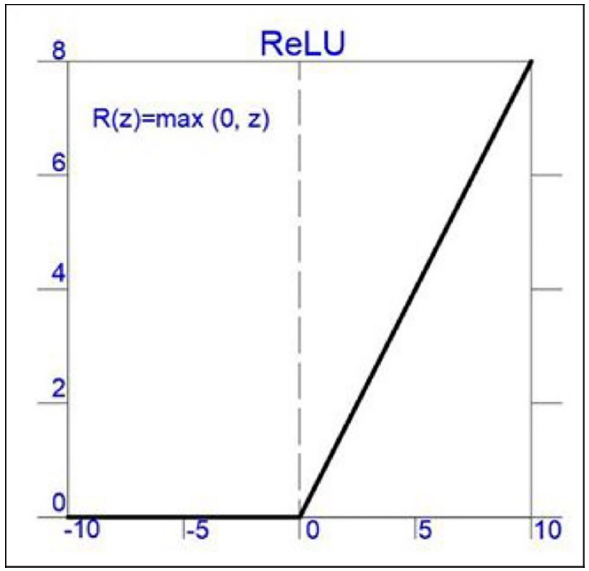
$$= (\alpha w_{11} + \beta w_{21})x_1 + (\alpha w_{12} + \beta w_{22})x_2 + (\alpha c_1 + \beta c_2) + c_3$$

$$= \widetilde{w_1}x_1 + \widetilde{w_2}x_2 + \tilde{c}$$

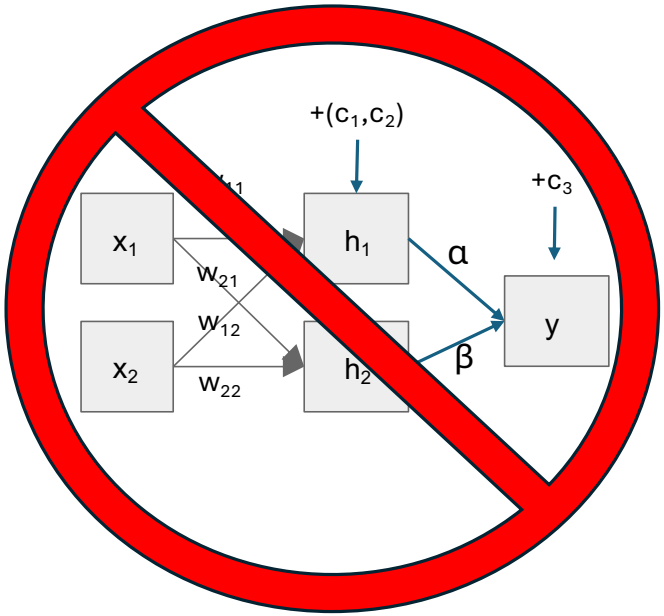
Μη-Γραμμικά Επίπεδα: ReLU

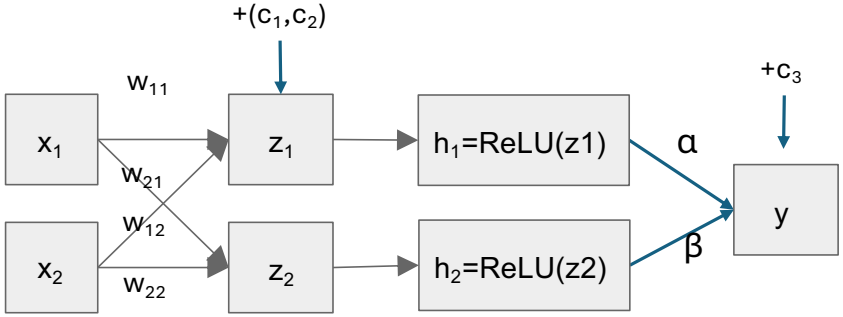
ReLU(α) = max(0,α)

α	ReLU(α)
2	2
-1	0
4	4

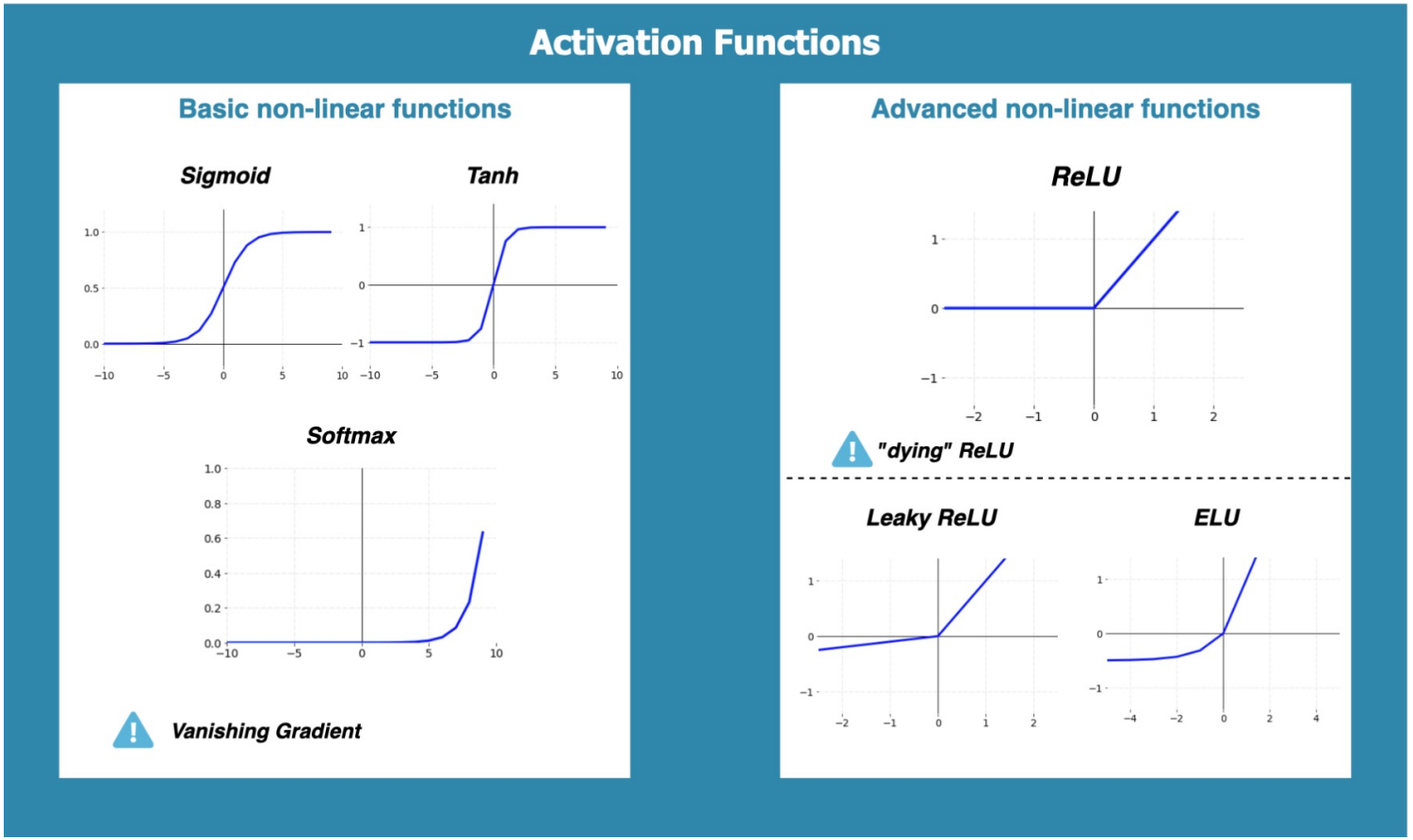


Μη-Γραμμικά Επίπεδα: ReLU

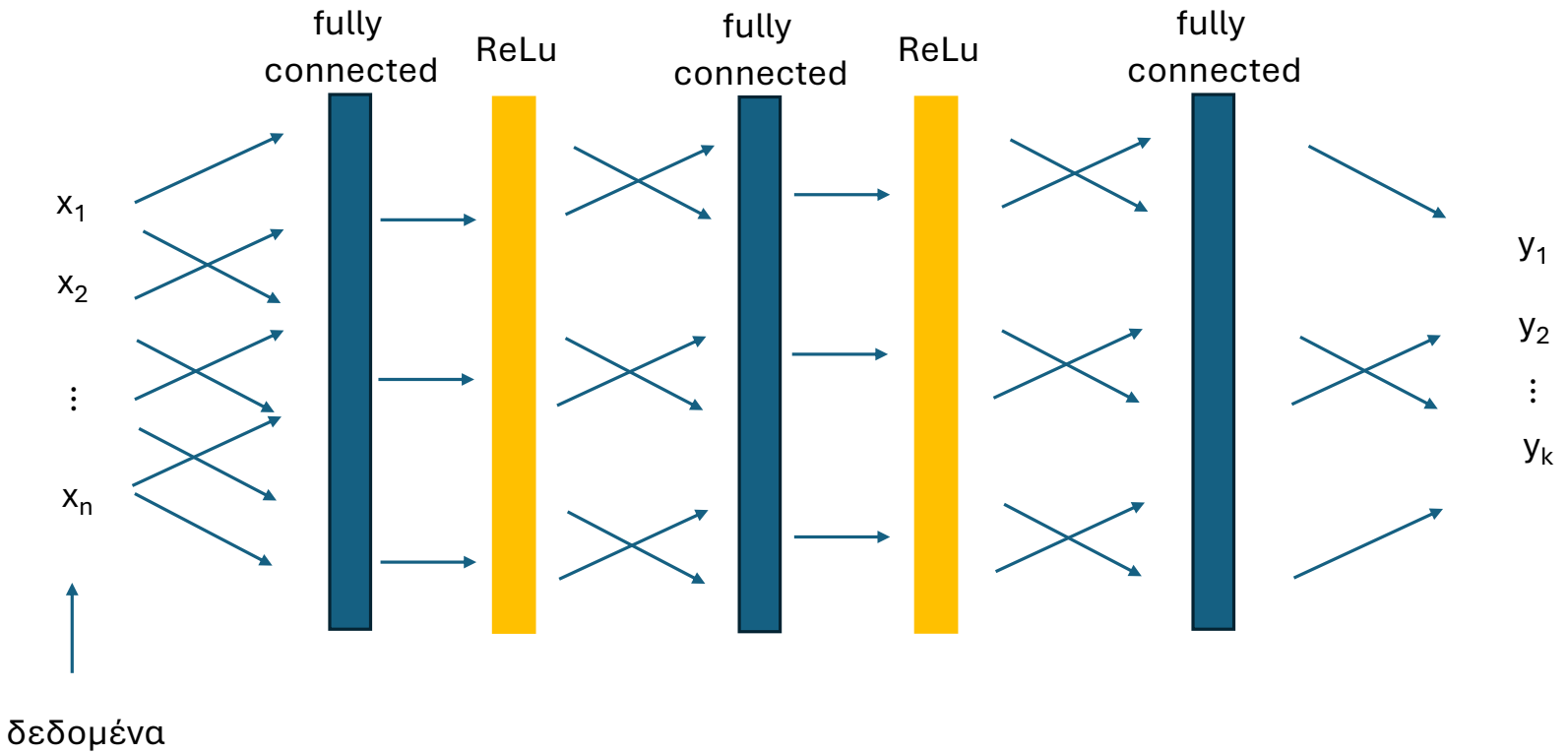




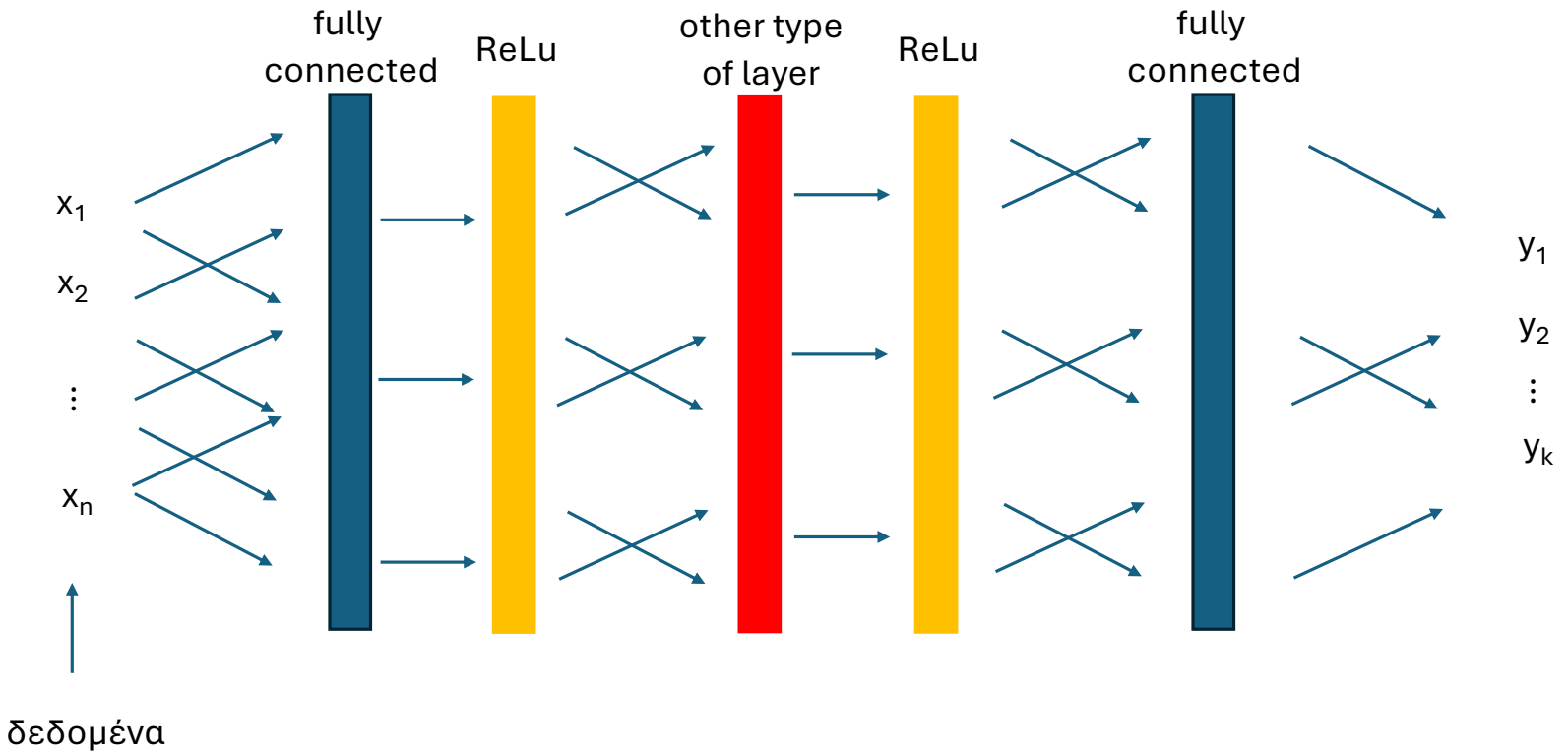
Γιατί ReLU και όχι κάτι άλλο;



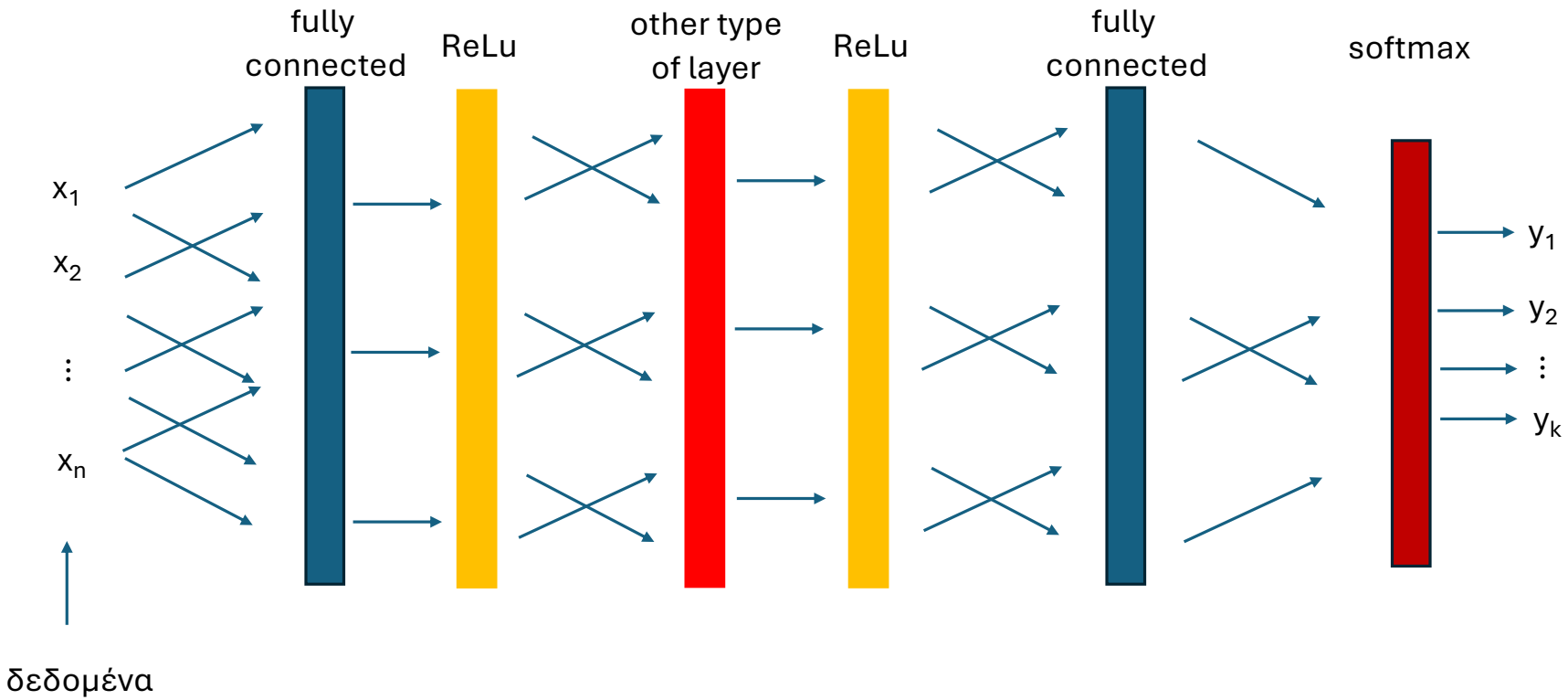
Fully Connected + ReLU



Fully Connected + ReLU + other



Fully Connected + ReLU + SoftMax + other



Πως τα χρησιμοποιούμε

Δέντρα απόφασης

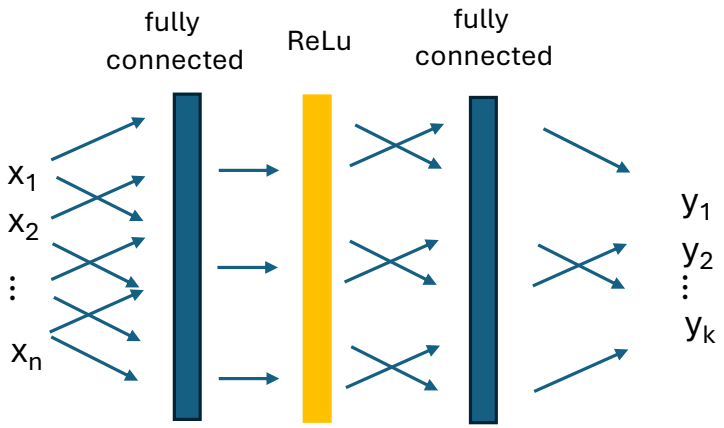
1. mymodel = DecisionTrees(max_depth=3) – ορίζουμε την οικογένεια
2. mymodel.fit(X,y) – βρίσκουμε παραμέτρους που συμφωνούν με (X,y)
3. mymodel.predict(x) – υπολογίζουμε προβλέψεις για τα x

Για νευρωνικά δίκτυα, αυτές οι τρείς ιδέες παραμένουν, αλλά οι εντολές αλλάζουν

model = MyNNet()

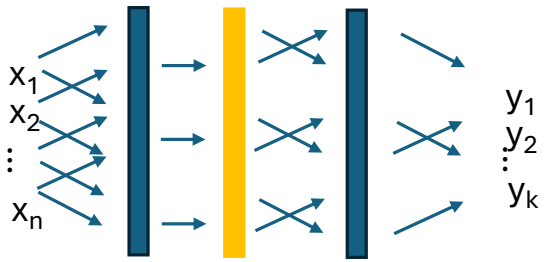
Ορίζουμε κάθε επίπεδο:

MyNNet = x→ fc1→ReLU→fc2→y



model = MyNNet()

model = MyNNet()



Ορίζουμε κάθε επίπεδο:

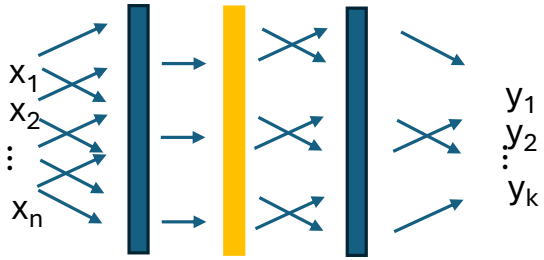
MyNNet = x→ fc1→ReLU→fc2→y

```python
# Define the neural network
class MyNNet(nn.Module):
    def __init__(self):
        super(MyNNet, self).__init__()
        self.fc1 = nn.Linear(3, 2)  # 3 inputs to 2 outputs
        self.fc2 = nn.Linear(2, 1)  # 2 inputs to 1 output
        self.relu = nn.ReLU()

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

model = MyNNet()

model = MyNNet()



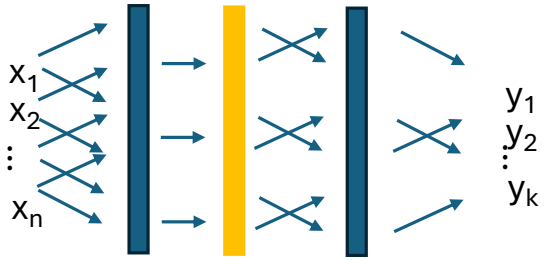

```
def __init__(self):
    super(MyNNet, self).__init__()
    self.fc1 = nn.Linear(3, 2)
    self.fc2 = nn.Linear(2, 1)
    self.relu = nn.ReLU()
```

```
# Define the neural network
class MyNNet(nn.Module):
    def __init__(self):
        super(SimpleNet, self).__init__()
        self.fc1 = nn.Linear(3, 2)  # 3 inputs to 2 outputs
        self.fc2 = nn.Linear(2, 1)  # 2 inputs to 1 output
        self.relu = nn.ReLU()

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

model = MyNNet()

model = MyNNet()




```
def __init__(self):
    super(MyNNet, self).__init__()
    self.fc1 = nn.Linear(3, 2)
    self.fc2 = nn.Linear(2, 1)
    self.relu = nn.ReLU()
```

self.fc1

self.relu

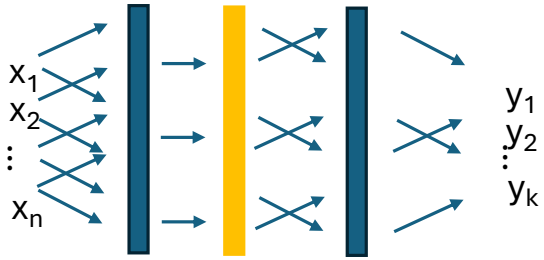
self.fc2

```
# Define the neural network
class MyNNet(nn.Module):
    def __init__(self):
        super(MyNNet, self).__init__()
        self.fc1 = nn.Linear(3, 2)  # 3 inputs to 2 outputs
        self.fc2 = nn.Linear(2, 1)  # 2 inputs to 1 output
        self.relu = nn.ReLU()

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

model = MyNNet()

model = MyNNet()




```
def __init__(self):
    super(MyNNet, self).__init__()
    self.fc1 = nn.Linear(3, 2)
    self.fc2 = nn.Linear(2, 1)
    self.relu = nn.ReLU()
```

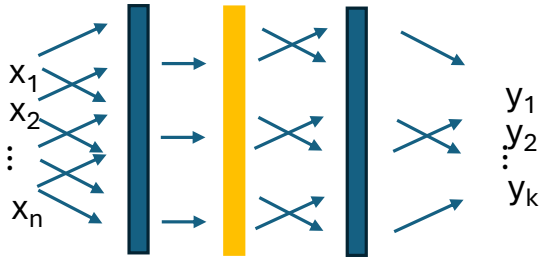
self.fc1

self.relu

self.fc2

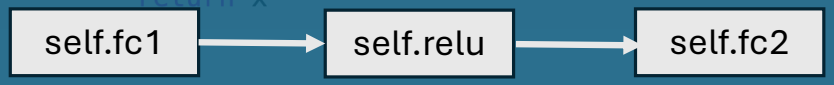
```
def forward(self, x):
    x = self.fc1(x)
    x = self.relu(x)
    x = self.fc2(x)
    return x
```

model = MyNNet()

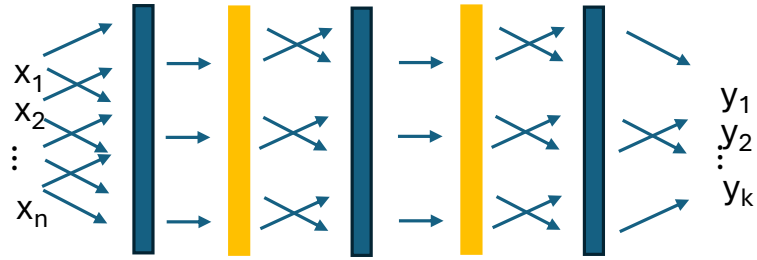



```
def __init__(self):
    super(MyNNet, self).__init__()
    self.fc1 = nn.Linear(3, 2)
    self.fc2 = nn.Linear(2, 1)
    self.relu = nn.ReLU()
```

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



model = BigNNet()



Ορίζουμε κάθε επίπεδο:

BigNNet = x→ fc1→ReLU→fc2

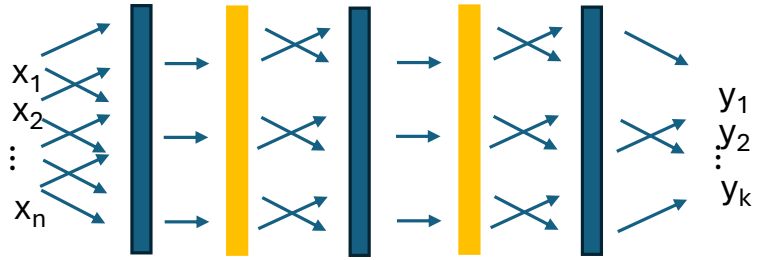
ReLU → fc3 →y

model = BigNNet()

```
# Define the neural network
class BigNNet(nn.Module):
    def __init__(self):
        super(BigNNet, self).__init__()
        self.fc1 = nn.Linear(3, 2)  # 3 inputs to 2 outputs
        self.fc2 = nn.Linear(2, 2) # 2 inputs to 2 outputs
        self.fc3 = nn.Linear(2, 1)  # 2 inputs to 1 output
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.relu(x)
        x = self.fc3(x)
        return x
```

model = BigNNet()




```
def __init__(self):
    super(BigNNet, self).__init__()
    self.fc1 = nn.Linear(3, 2)
    self.fc2 = nn.Linear(2, 2)
    self.fc3 = nn.Linear(2, 1)
    self.relu = nn.ReLU()
```

```
# Define the neural network
class BigNNet(nn.Module):
    def __init__(self):
        super(BigNNet, self).__init__()
        self.fc1 = nn.Linear(3, 2)  # 3 inputs to 2 outputs
        self.fc2 = nn.Linear(2, 2) # 2 inputs to 2 outputs
        self.fc3 = nn.Linear(2, 1)  # 2 inputs to 1 output
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.relu(x)
        x = self.fc3(x)
        return x
```

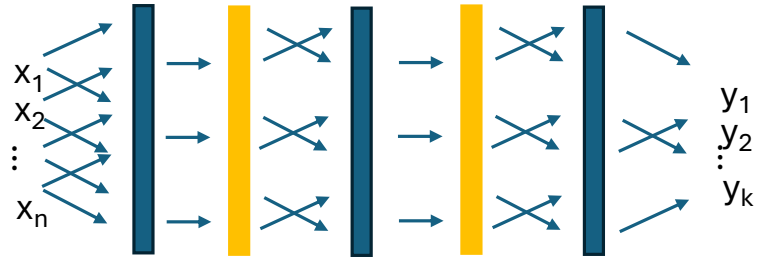
Op

Bi

ReLU → fc3 →y

model = BigNNet()

model = BigNNet()




```
def __init__(self):
    super(BigNNet, self).__init__()
    self.fc1 = nn.Linear(3, 2)
    self.fc2 = nn.Linear(2, 2)
    self.fc3 = nn.Linear(2, 1)
    self.relu = nn.ReLU()
```

self.fc1 | self.relu | self.fc2 | self.fc3

```
# Define the neural network
class BigNNet(nn.Module):
    def __init__(self):
        super(BigNNet, self).__init__()
        self.fc1 = nn.Linear(3, 2)  # 3 inputs to 2 outputs
        self.fc2 = nn.Linear(2, 2)  # 2 inputs to 2 outputs
        self.fc3 = nn.Linear(2, 1)  # 2 inputs to 1 output
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.relu(x)
        x = self.fc3(x)
        return x
```

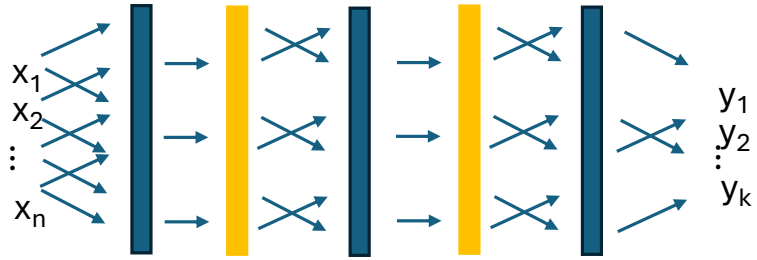
Op

Bi

ReLU → fc3 → y

model = BigNNet()

model = BigNNet()



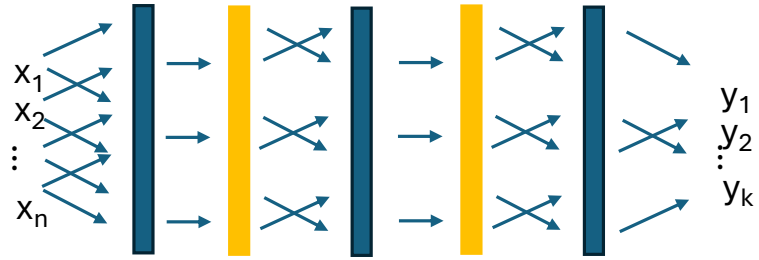

```
def __init__(self):
    super(BigNNet, self).__init__()
    self.fc1 = nn.Linear(3, 2)
    self.fc2 = nn.Linear(2, 2)
    self.fc3 = nn.Linear(2, 1)
    self.relu = nn.ReLU()
```

self.fc1 | self.relu | self.fc2 | self.fc3

ReLU → fc3 → y

```
def forward(self, x):
    x = self.fc1(x)
    x = self.relu(x)
    x = self.fc2(x)
    x = self.relu(x)
    x = self.fc3(x)
    return x
```

model = BigNNet()

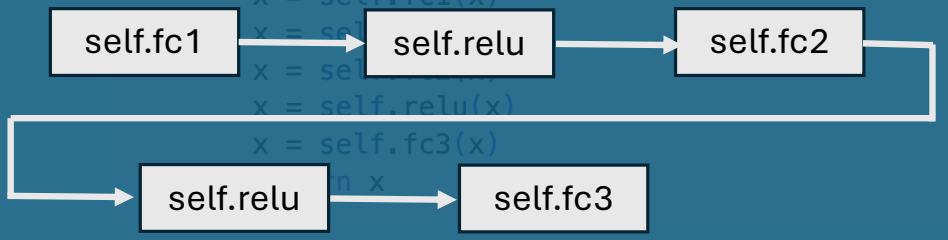



```
def __init__(self):
    super(BigNNet, self).__init__()
    self.fc1 = nn.Linear(3, 2)
    self.fc2 = nn.Linear(2, 2)
    self.fc3 = nn.Linear(2, 1)
    self.relu = nn.ReLU()
```

self.fc1 self.relu self.fc2 self.fc3

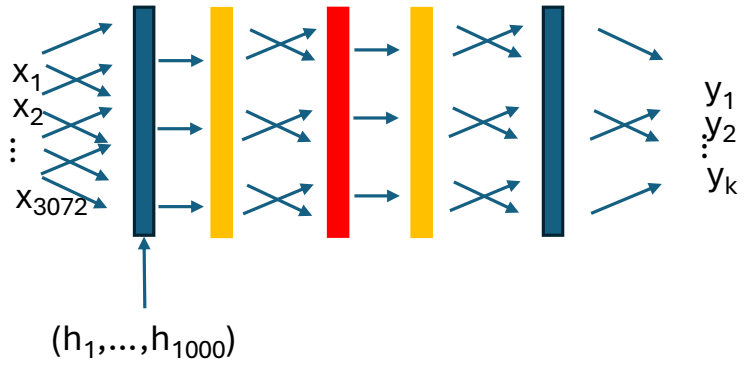
ReLU → fc3 →y

```
def forward(self, x):
    x = self.fc1(x)
    x = self.relu(x)
    x = self.fc2(x)
    x = self.relu(x)
    x = self.fc3(x)
    return x
```



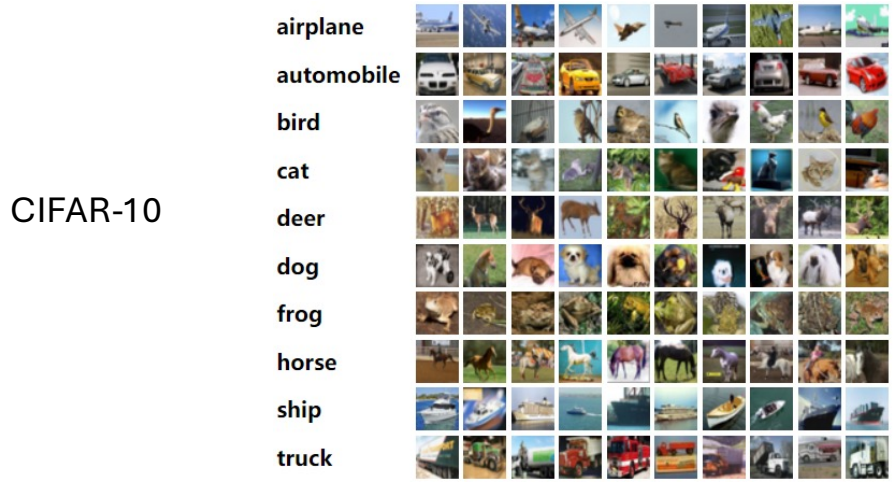
model.fit(X,y)

Η εντολή model.fit(X,y) ψάχνει για παραμέτρους που ελαχιστοποιούν την απώλεια / μεγιστοποιούν την ακρίβεια, στα δεδομένα εκπαίδευσης.

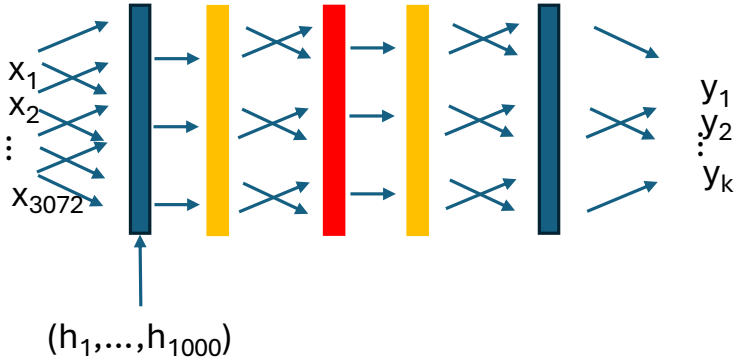


model.fit(X,y)

Η εντολή model.fit(X,y) ψάχνει για παραμέτρους που ελαχιστοποιούν την απώλεια / μεγιστοποιούν την ακρίβεια, στα δεδομένα εκπαίδευσης.



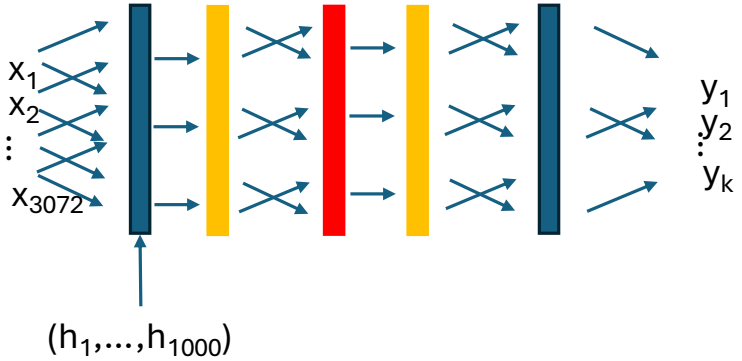
CIFAR-10



model.fit(X,y)

Η εντολή model.fit(X,y) ψάχνει για παραμέτρους που ελαχιστοποιούν την απώλεια / μεγιστοποιούν την ακρίβεια, στα δεδομένα εκπαίδευσης.

Άσκηση: η CIFAR-10 έχει 3.072 features. Εάν το 1ο επίπεδο είναι fully-connected και έχει 1.000 νευρώνες, πόσες παραμέτρους έχει το νευρωνικό δίκτυο μόνο σε αυτό το πρώτο επίπεδο;

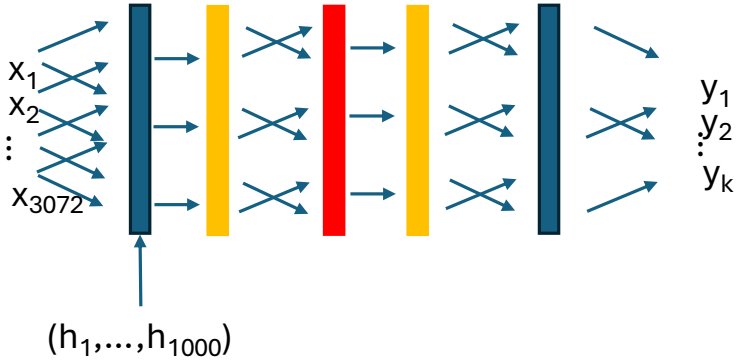


model.fit(X,y)

Η εντολή model.fit(X,y) ψάχνει για παραμέτρους που ελαχιστοποιούν την απώλεια / μεγιστοποιούν την ακρίβεια, στα δεδομένα εκπαίδευσης.

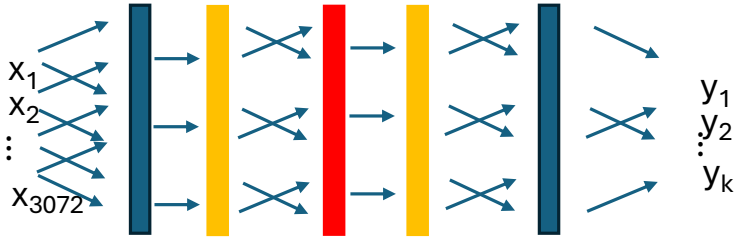
Άσκηση: η CIFAR-10 έχει 3.072 features. Εάν το 1ο επίπεδο είναι fully-connected και έχει 1.000 νευρώνες, πόσες παραμέτρους έχει το νευρωνικό δίκτυο μόνο σε αυτό το πρώτο επίπεδο;

Απάντηση: 3.072 * 1.000 + 1.000 = 3.073.000



model.fit(X,y)

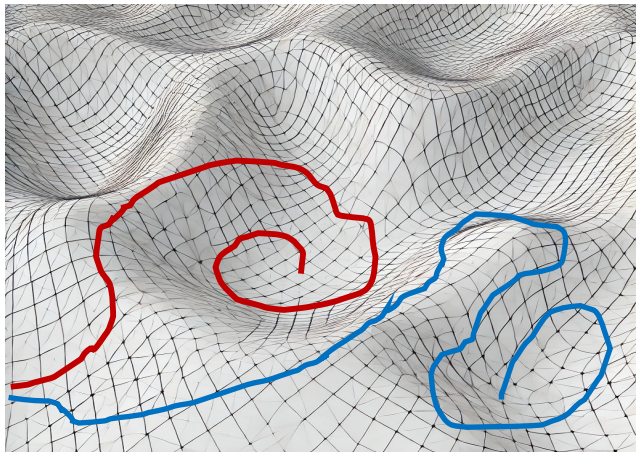
Η εντολή model.fit(X,y) ψάχνει για παραμέτρους που ελαχιστοποιούν την απώλεια / μεγιστοποιούν την ακρίβεια, στα δεδομένα εκπαίδευσης.



Τα νευρωνικά δίκτυα έχουν **πολλές παραμέτρους**, και η εύρεση των καλύτερων παραμέτρων είναι δύσκολο/αδύνατον.

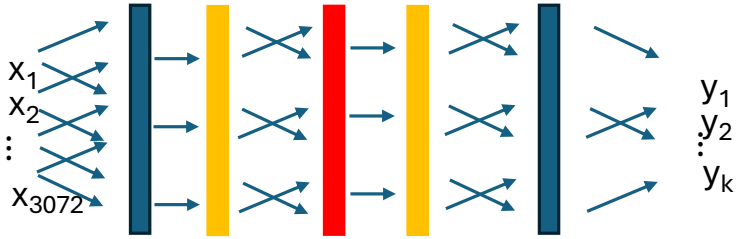
Αντί αυτού, ψάχνουμε ~~καλύτερες~~ **αρκετά καλές** παραμέτρους.

model.fit(X,y) → train(model, data, **optimizer**, **epochs**)



model.fit(X,y)

Η εντολή model.fit(X,y) ψάχνει για παραμέτρους που ελαχιστοποιούν την απώλεια / μεγιστοποιούν την ακρίβεια, στα δεδομένα εκπαίδευσης.



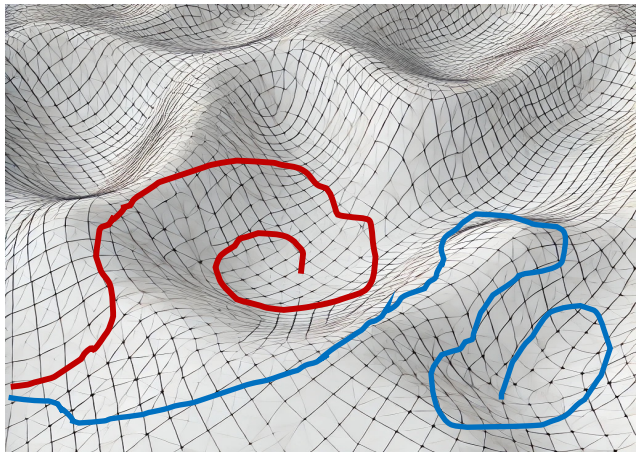
Τα νευρωνικά δίκτυα έχουν **πολλές παραμέτρους**, και η εύρεση των καλύτερων παραμέτρων είναι δύσκολο/αδύνατον.

Αντί αυτού, ψάχνουμε ~~καλύτερες~~ **αρκετά καλές** παραμέτρους.

model.fit(X,y) → train(model, data, **optimizer**, **epochs**)

Πώς ψάχνουμε:optim.Adam

πόσο ψάχνουμε



model.predict(x)

model(x)

Πως τα χρησιμοποιούμε

Νευρωνικά Δίκτυα

1. Ορίζουμε τα επίπεδα: MyNeuralNetwork
2. mymodel = MyNeuralNetwork() – ορίζουμε την οικογένεια
3. train(mymodel,data,optimizer,epochs) – βρίσκουμε παραμέτρους που συμφωνούν με (X,y)
4. mymodel(x) – υπολογίζουμε προβλέψεις για τα x