Python

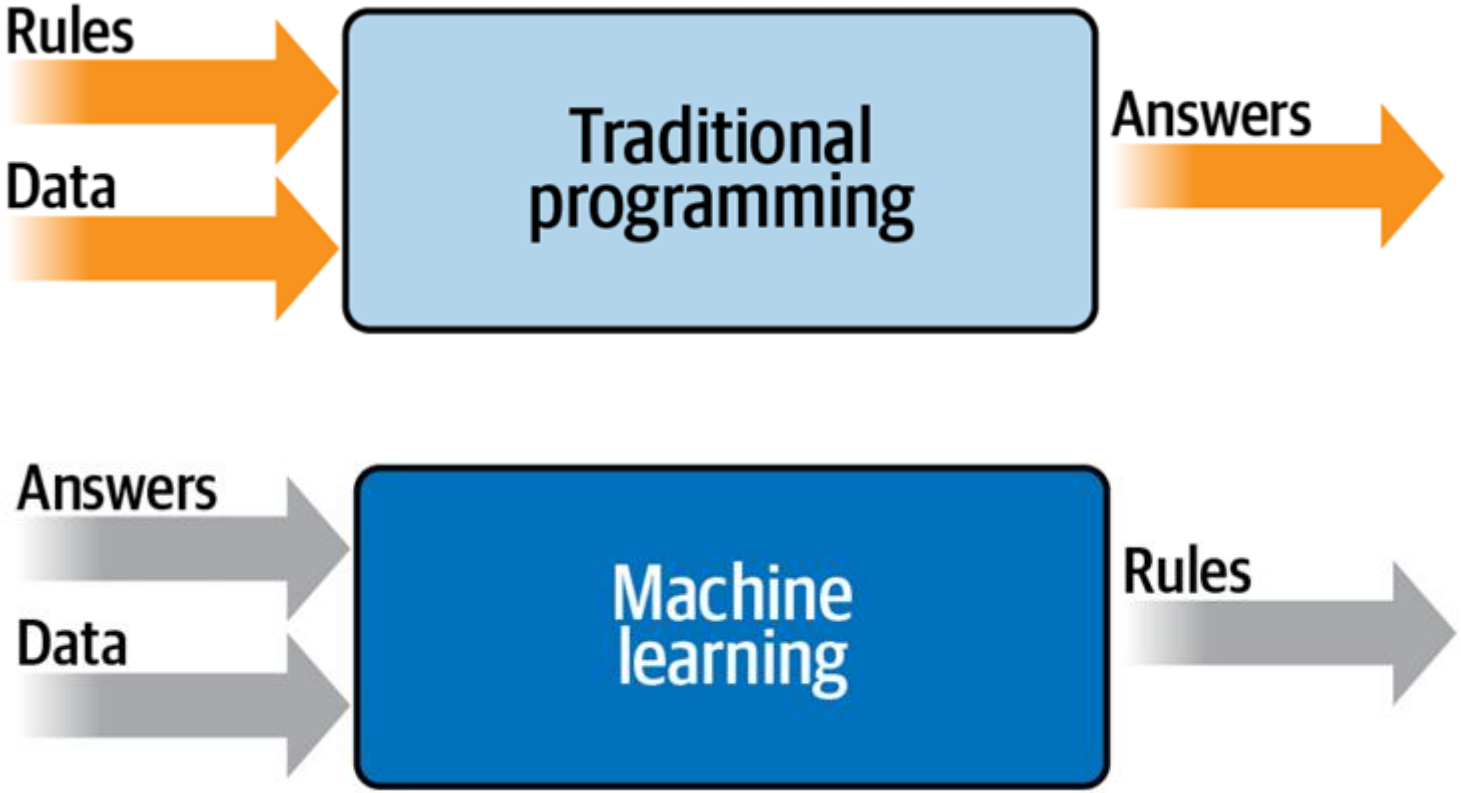
11. gyakorlat

Strohmayer Ádám

Agenda

- Neurális hálók
- Tensorflow/Pytorch
- Fine-tuning

Gépi tanulás



A matek része

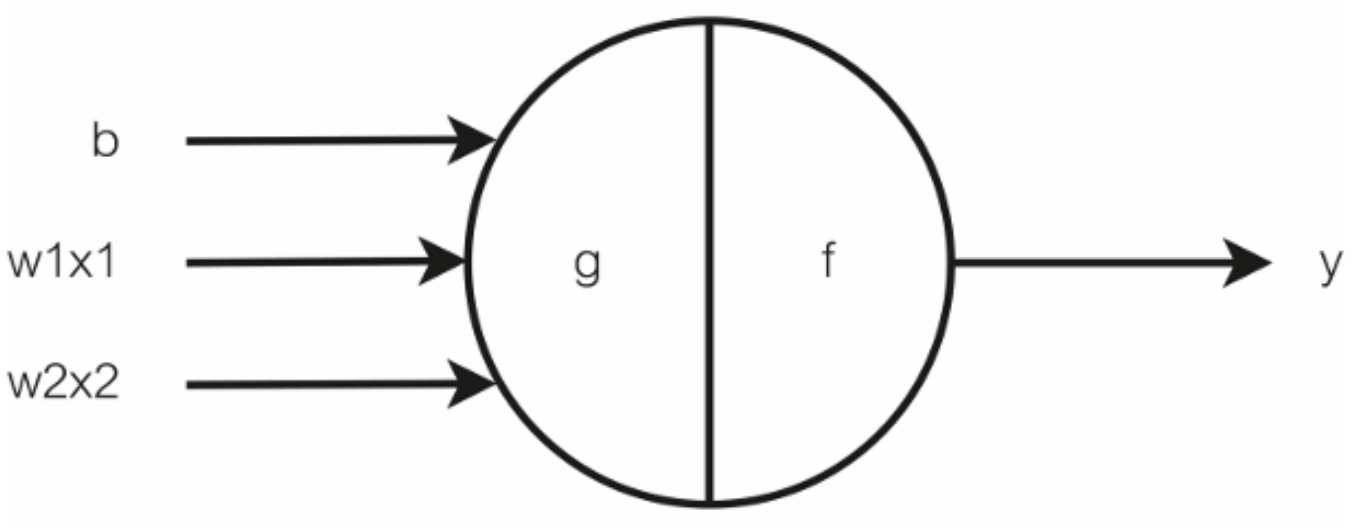
Mi a kapcsolat a számok között?

x1	x2	y
4	2	8
1	2	5
0	5	10
2	1	4

Milyen „tanulási módszerrel” állapítottuk ezt meg?

A neuron

w – súlyok
x – értékek
b - bias



$$g(x) = w1x1 + w2x2 + \cdots + wnxn + b = \sum_{i=1}^{n} wixi + b$$

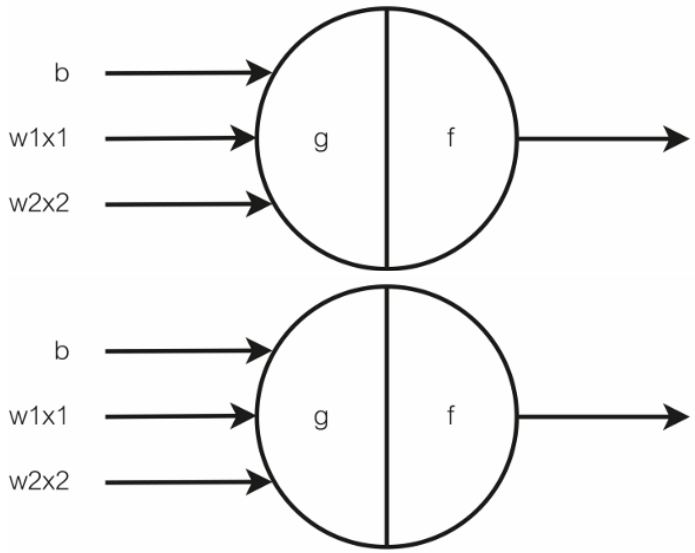
b – **torzítás mértéke** (ált. konstans)
g függvény **összegzi** az eddig kapott értékeket
f **aktivációs függvény** veszi g eredményét,
majd továbbadja transzformálva (miért?)

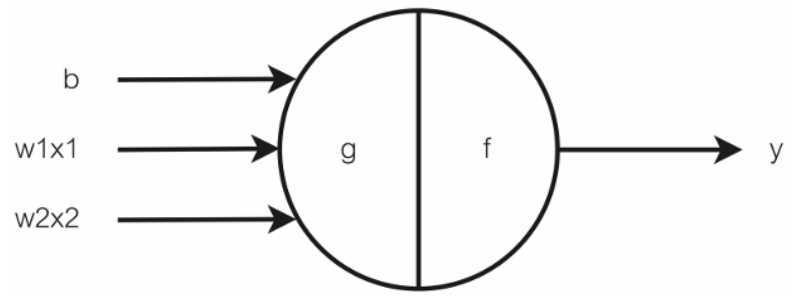
$$f(z) = \begin{cases} -1, z \leq 0 \\ 1, z > 0 \end{cases}$$

$$y = f(g(x)) = \begin{cases} -1, g(x) \leq 0 \\ 1, g(x) > 0 \end{cases}$$

Neuronos műveletek

Súlyok lehetnének vektorok is...
Minden súly egy-egy része a vektornak

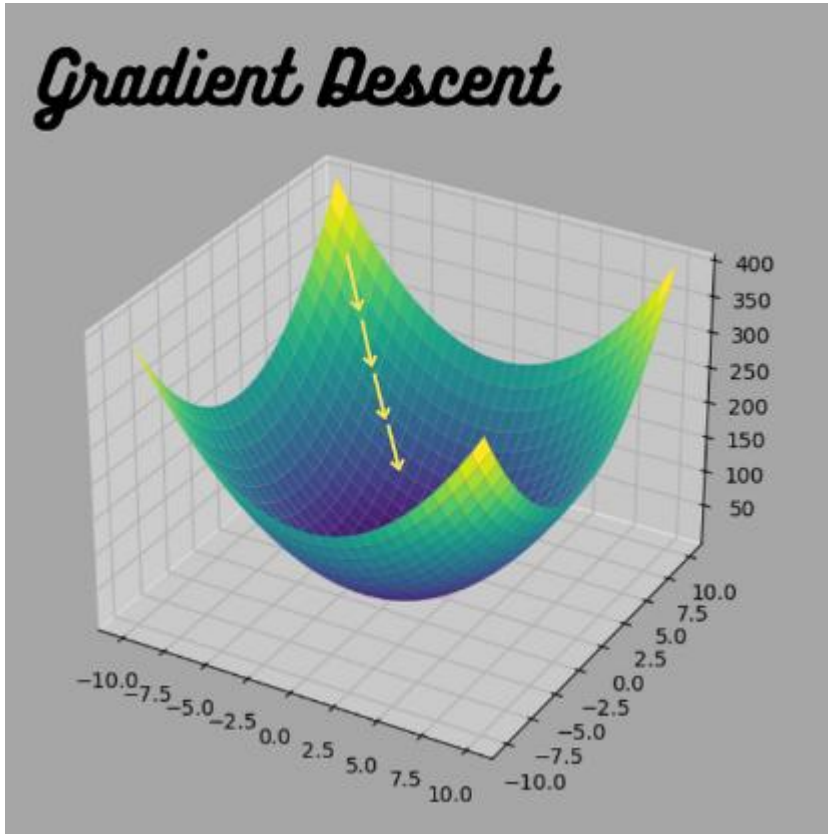




Gépi tanulásos algoritmusokkal, optimalizálókkal, veszteségfüggvényekkel a lehető leggyorsabban szeretnénk megtanítani a modellt, hogy elérje az y értékünket (címkénket).

Neuronos műveletek

Optimalizációs algoritmus példa:
Gradient descent



Keressük az optimális megoldást!

Vektorokkal próbálunk közelíteni
a legoptimálisabb értékhez
Legkisebb különbséghez közelítünk ezzel (mean_squared_error)

Forrás: https://medium.com/@jaleeladejumo/
gradient-descent-from-scratch-batch-gradient-descent-stochastic-gradient-descent-and-mini-batch-def681187473

Példa: Keressük meg az egyenlet megoldását!

```
layer0 = Dense(units=1, input_shape=[1]) #egy paraméter (neuron), egy inputtal

model = Sequential([layer0]) #szekvenciális modell a réteggel/rétegekkel
model.compile(optimizer = "sgd", loss = "mean_squared_error")
#optimalizációs algoritmus, veszteségfüggvény, összerakjuk a modell alapját (compile)

#több értéket adunk meg a tanításhoz, megfelelő címkeszámmal
input = np.array([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], dtype=float) #legyen ez x
target =np.array([-3.0, -1.0, 1.0, 3.0, 5.0, 7.0], dtype=float) #látjuk: 2x-1

model.fit(input, target, epochs=500) #epoch - egy tanítási ciklus – itt tanítjuk a modellt

#adott réteg súlyaival megnézzük, hogy mit tanult a modell:
print(f"Tanult súlyok: {layer0.get_weights()}") #kb. 2 (szorzó), kb. -1 (bias)

print(model.predict(np.array([10.0]))) #19-nek kéne kijönnie
```

Neurális hálók

Kérdés: hogyan ismerjük fel a kólát?

- Palack alakja
- Palack anyaga
- Palack címkéje
- Kupak színe
- Kupak anyaga
- Folyadék színe
- Folyadék íze

A gép ezeket nem érti (kezdetben)

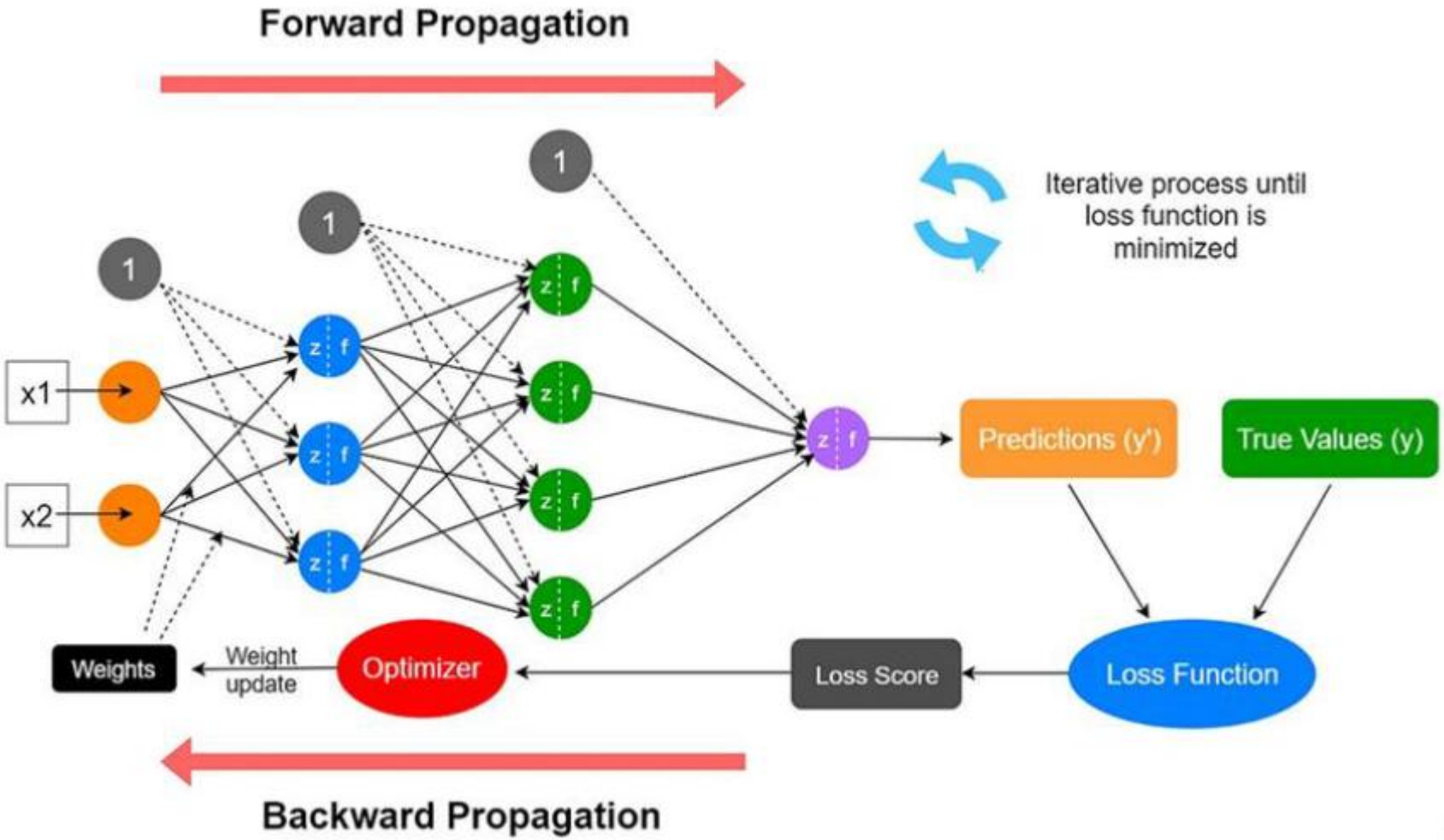
Neurális hálók

Gyakorlatban: gépek adott értékek alapján dolgoznak
Több rétegen keresztül próbálnak összefüggéseket találni

Ötlet: képek átadhatóak **mátrixokként** (tenzorokként)
– minden pixel egy-egy érték (koordináta/szín)

Mátrixokat vektorokká kell lapítani, hogy beilleszkedjenek a hálónkba!

A tanulási folyamat



Tensorflow/Pytorch

Mélytanulásos keretrendszerek implementációs különbségekkel
Számítási gráfok alapján végzik a tanítást

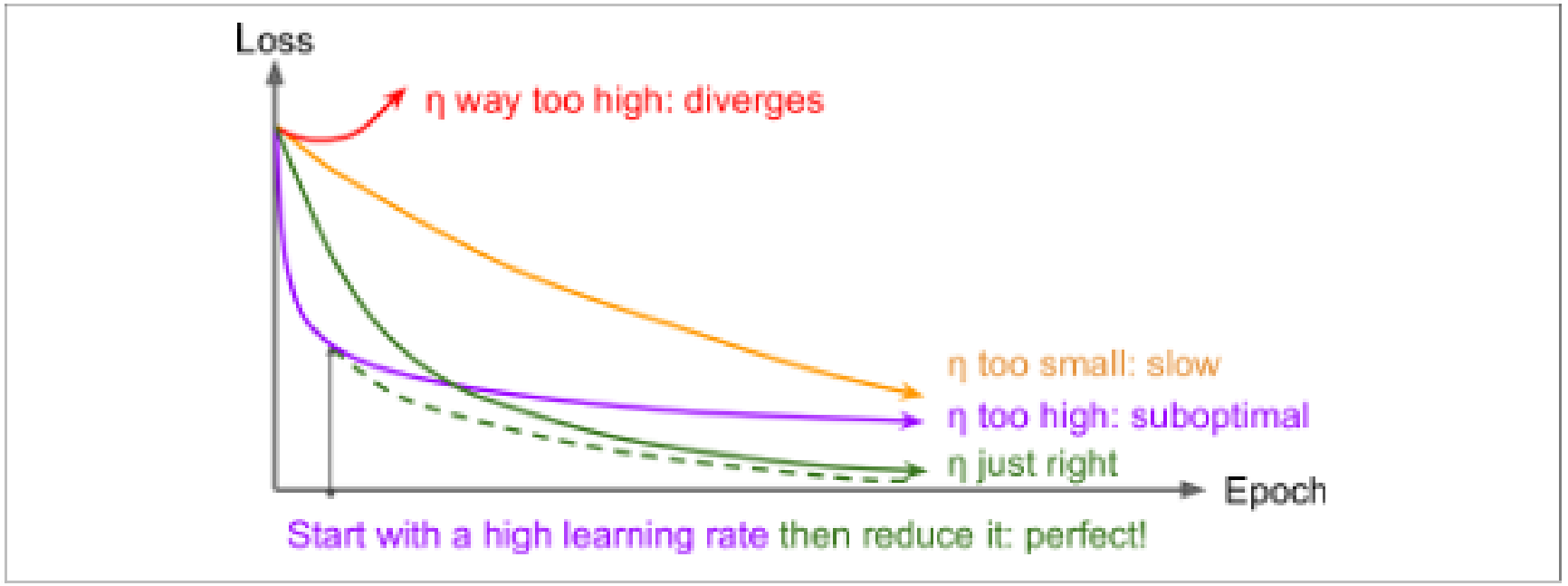
Pytorch objektumorientáltabb elven működik,
Tensorflow deklaratívabb

Pytorch teljes kontrollt ad a modelltanítás felett,
Tensorflow fit metódus mögé rejti a működést általában

Neuronok száma, modelltanítás

Neuronok száma tetszőleges lehet egy hálóban, de

- több neuron – lassabb tanulás, **túltanulás veszélye**
- kevesebb neuron – gyorsabb tanulás, **alultanulás veszélye**



Túltanulás megakadályozása

Túltanulás megakadályozható **callbackekkel**

Tanítófüggvénynek (fit) **megadhatóak ellenőrzőfüggvények**

```
class myCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs=None):
        if(logs.get('accuracy') > 0.95):
            print("\n", r"95% pontosság elérve, a modelltanításnak vége!") #regexmentes string
            self.model.stop_training = True
```

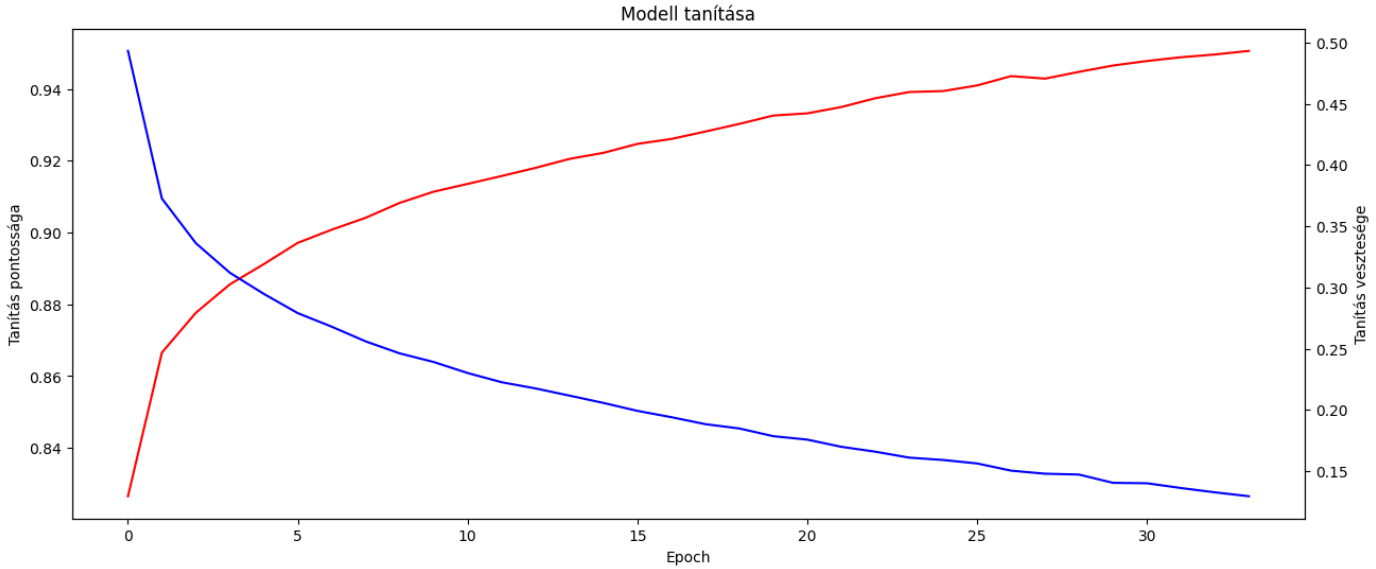
Callback példa

```
class myCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs=None):
        if(logs.get('accuracy') > 0.95):
            print("\n", r"95% pontosság elérve, a modelltanításnak vége!") #regexmentes string
            self.model.stop_training = True

callbacks = myCallback()
mnist = tf.keras.datasets.fashion_mnist

(training_images, training_labels), (test_images, test_labels) = mnist.load_data()

#fekete-fehér képek:
training_images = training_images/255.0 #0-255 skála normalizálása 0-1 közé
test_images = test_images/255.0


model = tf.keras.models.Sequential([tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation=tf.nn.relu),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax) #legnagyobbat kerekíti fel
    ])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
history = model.fit(training_images, training_labels, epochs=50, callbacks=[callbacks])
```

Kiértékelés

Tanítás után teljesen új adathalmazra (a validációsra) is futtatjuk a modellt (itt már nem tanul):

```
testloss, testaccuracy = model.evaluate(test_images, test_labels)
#modell kiértékelése


print(testaccuracy, max(history.history["accuracy"]))
#0.8848000168800354 0.9505666494369507
```



Fine-tuning

Fine-tuning

Modellek lementhetőek, illetve módosíthatóak is!

Van : cicafelismerő modell
Kell : kutya-cica-egyéb felismerő modell

Már meglévő modellekre plusz kutyafelismerő rétegeket rakhatunk fel!

modell mentése : catmodel = model.save(„mappa/catmodel”)
modell betöltése : loaded_catmodel = tf.keras.models.load_model(„mappa/catmodel”)

Rétegfagyasztás

Már meglévő modellrétegeket **be tudjuk fagyasztani**, így nem módosulnak a backpropagation által!

Ekkor nem kell egész modelleket újra tanítani, bővíthetjük őket újabb és újabb rétegekkel.

Ezzel több funkciót, tudást építhetünk a már meglévő modellünkre!

pl. már szarvast is fel tudna ismerni

Fine-tuning példa

```
loaded_catmodel = tf.keras.models.load_model("catmodel")

catmodel_dogmodel_other = tf.keras.models.Sequential(loaded_catmodel.layers[:-1])
#utolsó klasszifikációs réteget elhagyjuk az indexeléssel
catmodel_dogmodel_other.add(tf.keras.layers.Dense(3, activation="softmax"))
#pl. eddig szigmoid volt - eldöntő függvény (most pl. klasszifikál 3 lehetőségre)

for layer in catmodel_dogmodel_other.layers[:-1]:
    layer.trainable = False #befagyasztás, ezután compile kell

catmodel_dogmodel_other.compile(loss = "categorical_crossentropy", optimizer = "sgd",
                                metrics = ["accuracy"])
#ezután majd újra tanítás, kiértékelés kell
```

Források: Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (Aurélien Géron)
AI and Machine Learning for Coders A Programmers Guide to Artificial Intelligence (Laurence Moroney)

Köszönöm a figyelmet!