1. ŠTA SU STABLA ODLUKE?

STABLA ODLUČIVANJA SU NADZIRANI ALGORITMI ML I KORISTE SE ZA KLASIFIKACIJU I REGRESIJU. ONI DIJELE PODATKE NA PODGRUPE NA OSNOVU ODLUKA KOJE SE DONOSE NA ČVORIŠTIMA STABLA. SVAKA ODLUKA SE ZASNIVA NA VRIJEDNOSTI ATRIBUTA ULAZNIH PODATAKA, A KONAČNI ČVOROVI PREDSTAVLJAJU PREDIKCIJE.

2. KAKO FUNKCIONIŠU?

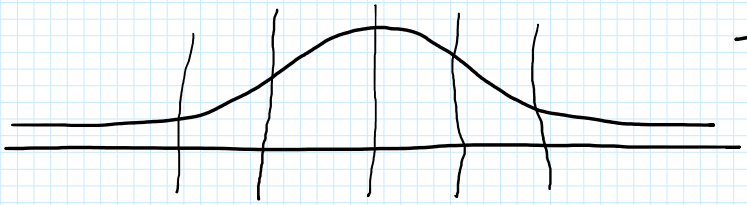
- KORIJENI ČVOR: POČETNA TAČKA KOJA SADRŽI CIJELI DATASET
- UNUTRAŠNJI ČVOROVI: DONOSE ODLUKU NA OSNOVU ATRIBUTA KOJI DIJELE DATASET U 2 I VIŠE GRUPA
- LISTOVI: PREDSTAVLJAJU KONAČNE ODLUKE ILI PREDIKCIJE NA OSNOVU ULAZNIH ATRIBUTA

3. KRITERIJUMI ZA ODVAJANJE:

- KORISTE SE RAZLIČITI ALGORITMI)

GINI NEČISTOĆA: MJERA NEČISTOĆE ILI NEUJEDNAČENOSTI GRUPE.

$$GINI = 1 - \sum_{i=1}^{n} (p_i)^2$$



- MJERA ODSTUPANJA NASUMIČNO IZABRANOG ELEMENTA OD NASUMIČNO IZABRANOG PODIOKA RASPODJELE TOG SKUPA

NIŽA VRIJEDNOST OZNAČAVA DA JE GRUPA HOMOGENIJA

- INFORMACIONI DOBITAK (ENTHROPY, ENTROPIJA) MJERA NEIZVJESNOSTI ILI NESIGURNOSTI. CILJE DA ENTROPIJU SMANJIMO SA SVAKOM PODJELOM.

22 NEURALNA MREŽA

- NEURALNE MREŽE: NADGLEDANI ALGORITMI ML-A INSPIRISANI STRUKTUROM I FUNKCIJOM MOZGA. KORISTE SE ZA RJEŠAVANJE RAZLIČITIH PROBLEMA KAO ŠTO SU: KLASIFIKACIJA, REGRESIJA, SEGMENTACIJA SLIKE I MNOGI

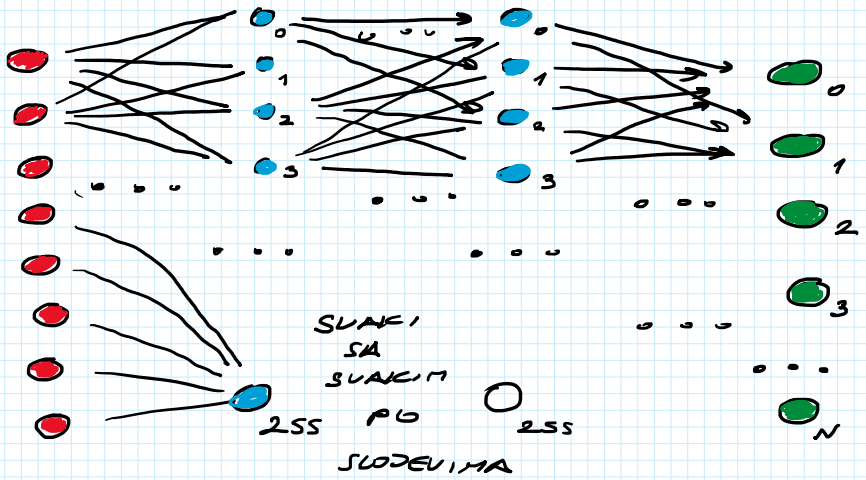
DRUGI

- DRUGO IME: MLP → MULTI-LAYERED-PERCEPTRON
- OSNOVNI KONCEPT JOŠTE ČUVAR ODNOSNO NEURON

ULAZNI NEURONI PREDSTAVLJAJU ULAZNE PODATKE

SKRIVENI NEURONI: SLOJEVI IZMEĐU ULAZA I IZLAZA KOJI TRANSFORMIŠU ULAZNE PODATKE.

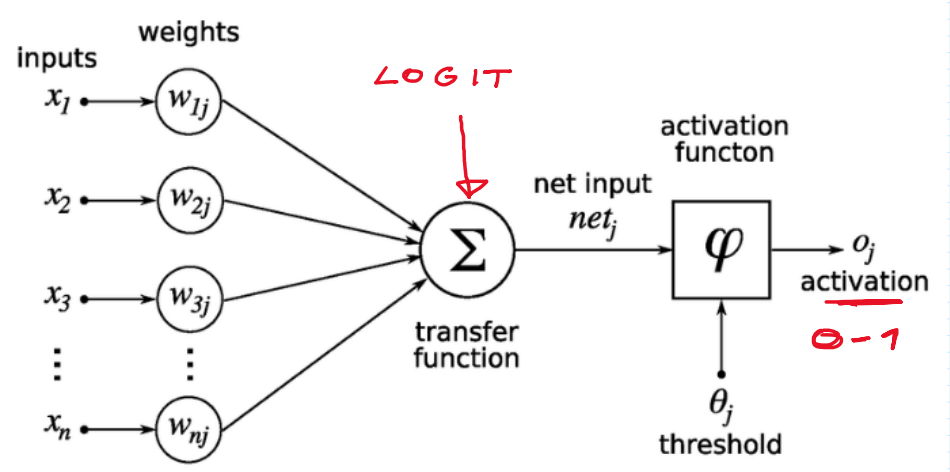
IZLAZNI NEURONI: PREDSTAVLJAJU IZLAZ MREŽE, ODNOSNO PREDIKCIJE



8 256 256 N: BROJ KLASA

(TREBALO BI DA BUDE 7, GREŠKA U BG)

JEDAN NEURON IZ SKRIVENIH SLOJEVA



WEIGHTS (TEŽINE): PARAMETRI KOJI POVEZUJU NEURONE IZMEĐU SLOJEVA I UTIČU NA ZNAČAJ SVAKOG ULAZA.

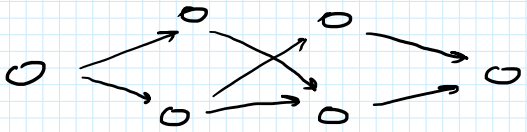
PRISTRANOST (BIAS): DODATNI PARAMETAR KOJI SE DODAJE LINEARNOJ KOMBINACIJI TEŽINA I ULAZA KAKO BI SE POMJERILE AKTIVACIONE FUNKCIJE

AKTIVACIONE FUNKCIJE :
- TRANSFORMIŠU LINEARNE KOMBINACIJE ULAZA U IZLAZE SVAKOG NEURONA.

SIGMOID FUNKCIJA : $$S(x) = \frac{1}{1+e^{-x}}$$

RELU FUNKCIJA :

$$R(x) = \begin{cases} x > 0 , & x \\ x = 0 , & 0 \end{cases}$$



FORWARD PROPAGATION

PROLAZIMO OD INPUTA DO OUTPUTA

BACKWARD PASS (BACKPROPAGATION)

- IDEMO OD OUTPUTA SA GREŠKOM I MIJENJAMO TEŽINE PUT NAZAD DO INPUTA SA CILJEM SMANJIVANJA GREŠKE

$$C = \frac{1}{2N} \sum \| \text{AKTIVACIONA FUNKCIJA}(x) - y(x) \|^2$$

ŠTA JE LOGIT

LOGIT JE LINEARNA KOMBINACIJA TEŽINA I ULAZA KOJA SE KORISTI KAO ULAZ U AKTIVACIONU FUNKCIJU. KONCEPT SE ČESTO

$$\text{LOGIT } z = \sum_{i=1}^{N} w_i \cdot x_i + b$$

w_i - TEŽINA KOJA SE PRIMJENJUJE NA i-TI ULAZ

x_i - i-ti ULAZ

b - bias (ofset)

ONO ŠTO DOBIJEMO ONDA UBACUJEMO U AKTIVACIONU FUNKCIJU

$$\sigma(z) = \frac{1}{1+e^{-z}}$$ I ONO ŠTO DOBIJEMO JE

URIDJENOST AKTIVACIONE FUNKCIJE IZMEĐU 0 i 1

ANOTACIJA KODA

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

data = pd.read_csv('crop.csv')

print(data.head())

print(data.isnull().sum())

data = data.dropna()

encoder = LabelEncoder()
data['Crop'] = encoder.fit_transform(data['Crop'])

X = data[['Nitrogen', 'Phosphorus', 'Potassium', 'Temperature', 'Humidity', 'pH_Value', 'Rainfall']]
y = data['Crop']

X = (X - X.mean()) / X.std()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

BIBLIOTEKE

ČITANJE CSV FAJLA

ŠTAMPAM UZORAK

IZBACUJEM NA URIDJENOSTI

NORMALIZOVANJE PODATAKA → $\frac{X - X_{prosjek}}{\text{STANDARDNA DEVIJACIJA OD } X}$

Z-SCORE

80:20 SPLIT

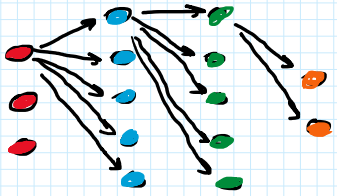
I REZULTAT SU NORMALIZOVANI PODACI GDJE JE PROSJEK =0, A STANDARDNA DEVIJACIJA 1

HIPER PARAMETRI

```
learning_rate = 0.01
nb_epochs = 100
batch_size = 32

nb_input = X_train.shape[1]
nb_hidden1 = 256
nb_hidden2 = 256
nb_classes = len(np.unique(y))
```

BATCH DA MOŽEMO DA UBACUJEMO MALO PO MALO



BROJ RAZLIČITIH OUTPUTOVA, UMJESTO SLUČAJA BROJ RAZLIČITIH BILJAKA

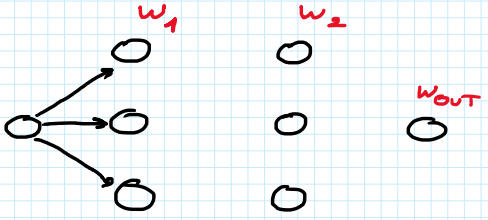
IGRALIŠTE

```
w = {
    '1': tf.Variable(tf.random.normal([nb_input, nb_hidden1], dtype=tf.float64)),
    '2': tf.Variable(tf.random.normal([nb_hidden1, nb_hidden2], dtype=tf.float64)),
    'out': tf.Variable(tf.random.normal([nb_hidden2, nb_classes], dtype=tf.float64))
}

b = {
    '1': tf.Variable(tf.random.normal([nb_hidden1], dtype=tf.float64)),
    '2': tf.Variable(tf.random.normal([nb_hidden2], dtype=tf.float64)),
    'out': tf.Variable(tf.random.normal([nb_classes], dtype=tf.float64))
}

f = {
    '1': tf.nn.relu,
    '2': tf.nn.relu,
    'out': tf.nn.softmax
}
```

MI TE MATRICE ISPUNJAVAMO RANDOM BROJEVIMA !!!

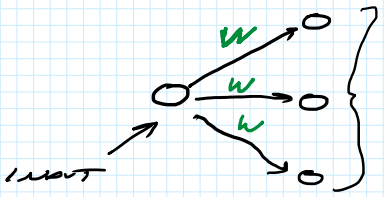


W_1 = INPUT i 256 NEURONA (3 ZA PRIMJER)

PRAVI MATRICU SA INPUT BROJ REDOVA I 256 KOLONA

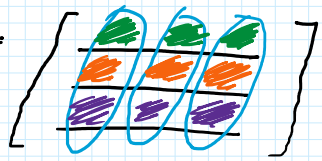
W_1 = [ ] JEDAN RED ZBOG 1 INPUT NEURONA

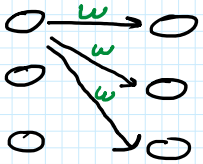
BROJ KOLONA ZA NEURONE



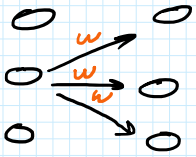
W_2 = BROJ NEURONA 1. SLOJA / BROJ NEURONA 2. SLOJA

W_2 =

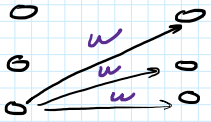




ILI

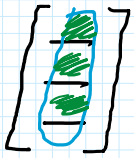


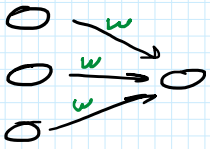
ILI



W_{OUT} = BROJ NEURONA 2. SLOJA / OUTPUT

W_{OUT} =





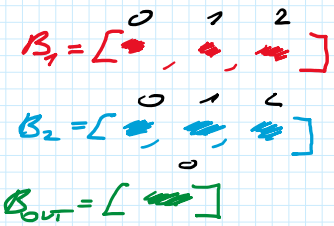
```
b = {
    '1': tf.Variable(tf.random.normal([nb_hidden1], dtype=tf.float64)),
    '2': tf.Variable(tf.random.normal([nb_hidden2], dtype=tf.float64)),
    'out': tf.Variable(tf.random.normal([nb_classes], dtype=tf.float64))
}
```

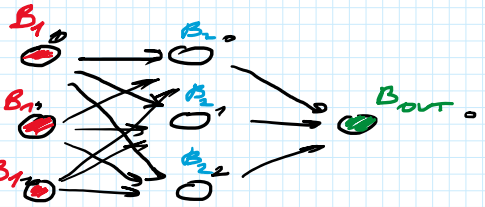
B = BIAS ILI OFSET

B_1 = BIAS PRVOG SLOJA PREDSTAVLJA NIZ VRIJEDNOSTI

B_2 = -//- DRUGOG SLOJA -//-

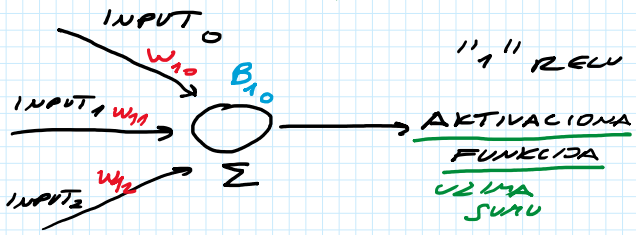
B_{OUT} = -//- OUTPUTA -//-



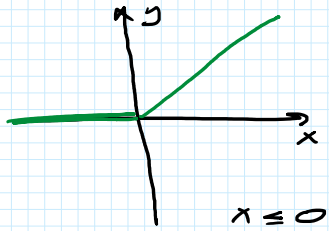


```
f = {
    '1': tf.nn.relu,
    '2': tf.nn.relu,
    'out': tf.nn.softmax
}
```

PAKUJEMO AKTIVACIONE FUNKCIJE U MAPU



RELU



$x \leq 0 \quad y = 0$

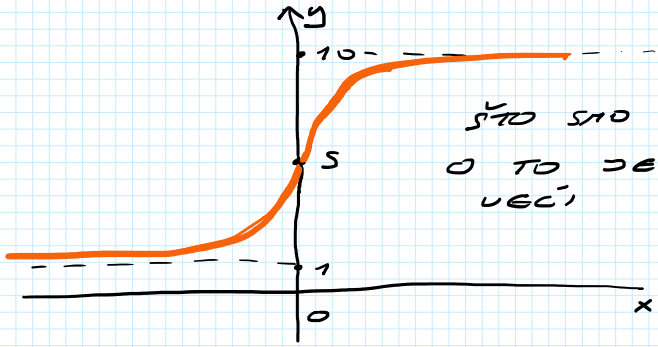
$x > 0 \quad y = x$

ZA 1. i 2. SLOJ JE RELU, A ZA OUTPUT JE SOFTMAX

SOFTMAX

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

- σ = softmax
- $\vec{z}$ = input vector
- e^{z_i} = standard exponential function for input vector
- K = number of classes in the multi-class classifier
- e^{z_j} = standard exponential function for output vector
- e^{z_j} = standard exponential function for output vector



ŠTO SMO BLIŽI 1 TO JE NAGIB VEĆI

POKRETANJE MREŽE

```
def runNN(x):                    OVO RADI ZA SVE NEURONE ODJEDNOM
    z1 = tf.add(tf.matmul(x, w['1']), b['1'])
    a1 = f['1'](z1)
    z2 = tf.add(tf.matmul(a1, w['2']), b['2'])
    a2 = f['2'](z2)
    z_out = tf.add(tf.matmul(a2, w['out']), b['out'])
    out = f['out'](z_out)

    pred = tf.argmax(out, 1)
    return pred, z_out
```

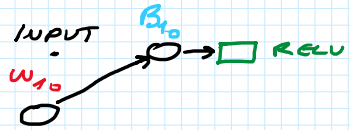
ZA $Z1_0$ DEĐAN NEURON

$Z1 = (\text{INPUT} \cdot \text{WEIGHT}) + \text{BIAS}$

$W_1 \quad B_1$

$A_1 = \text{AKTIVACIONA FUNKCIJA}(Z1)$

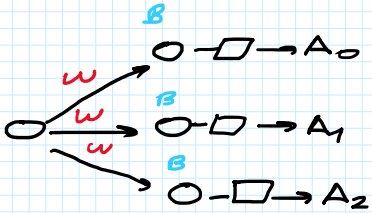
RELU



$$\Sigma\left(\begin{bmatrix}\text{INPUT}\\ \text{BROJ}\end{bmatrix} \cdot \begin{bmatrix}W_0 & W_1 & W_2\end{bmatrix}\right) + \begin{bmatrix}B_0 & B_1 & B_2\end{bmatrix} = Z_{1_0}$$

$A_{1_0} = \text{RELU}(Z_{1_0})$

ČITAV Z_1



$Z_1 = I \cdot W + B$

$$\underset{1\times 1}{[I]} \cdot [W_0 \quad W_1 \quad W_2] + [B_0 \quad B_1 \quad B_2] = [Z_{1_0} \quad Z_{1_1} \quad Z_{1_1}]$$

$[I]_{1\times 1} \cdot [W_0 \; W_1 \; W_2] + [B_0 \; B_1 \; B_2] = [Z_{1_0} \; Z_{1_1} \; Z_{1_1}]$

POSLIJE A_1 KORAKA

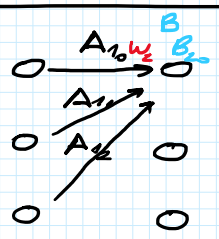
$A_1 = [A_{1_0} \; A_{1_1} \; A_{1_2}]$

POJEDINAČNI LOGITI

Z2 KORAK

$Z2 = (A_1 \cdot W_2) + B_2$

$\left([A_{1_0}, A_{1_1}, A_{1_2}]_{1\times 3} \cdot \underset{3\times 3}{W_2} \right) + B_2$



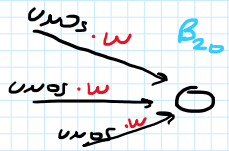
SVAKI SA SVAKIM ZVAČI I VEZA SA TEŽINAMA

$B_2 = [B_{2_0}, B_{2_1}, B_{2_2}]$

$Z_2 = [Z_{2_0}, Z_{2_1}, Z_{2_3}]$

$A_2 = RELU(Z_2)$

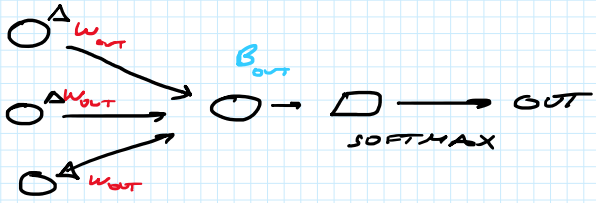
$A_{2_0} = RELU(Z_{2_0})$



$A_{2_0} = RELU\left(\sum_i^{BROJ\ UNOSA} (UNOS_i \cdot W_{2_i}) + B_{2_0} \right)$

PRIČEMU UNOS JE ZAPRAVO A_1

$Z\_OUT = (A_2 \cdot W_{OUT}) + B_{OUT}$



ISTO KAO I PRIJE

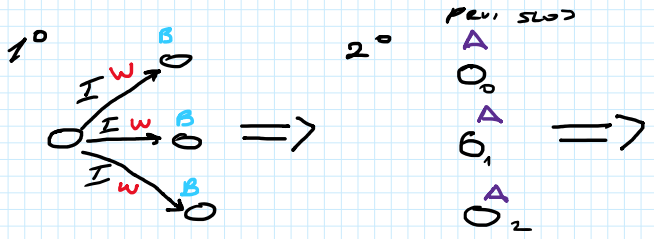
```
pred = tf.argmax(out, 1)
return pred, z_out
```

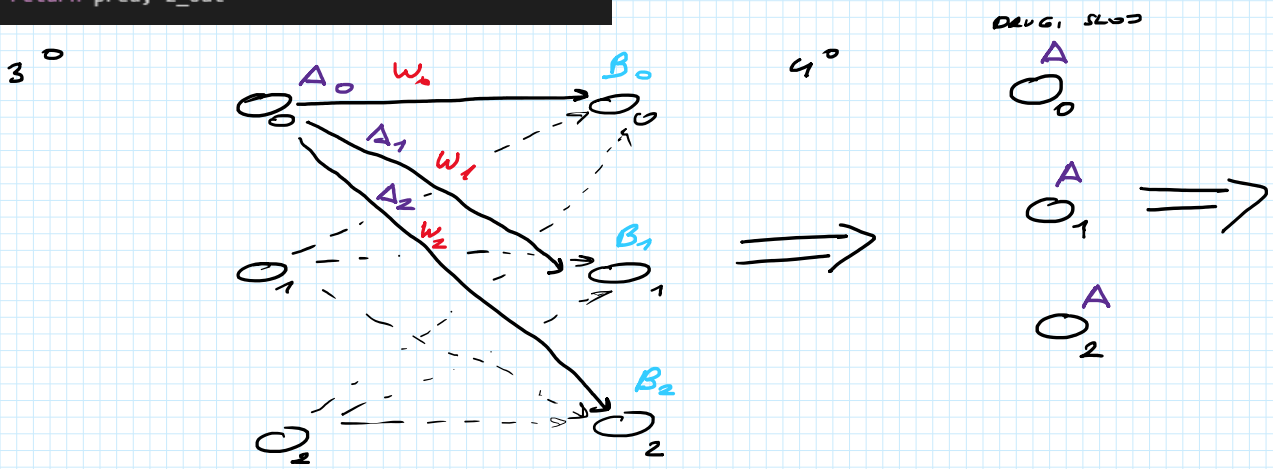
IZ SLOJA OUTPUTA, UZIMA ONU LABELU ILI INDEKS (NA OSNOVU OUTPUT NEURONA KOJI JE IMAO MAX)

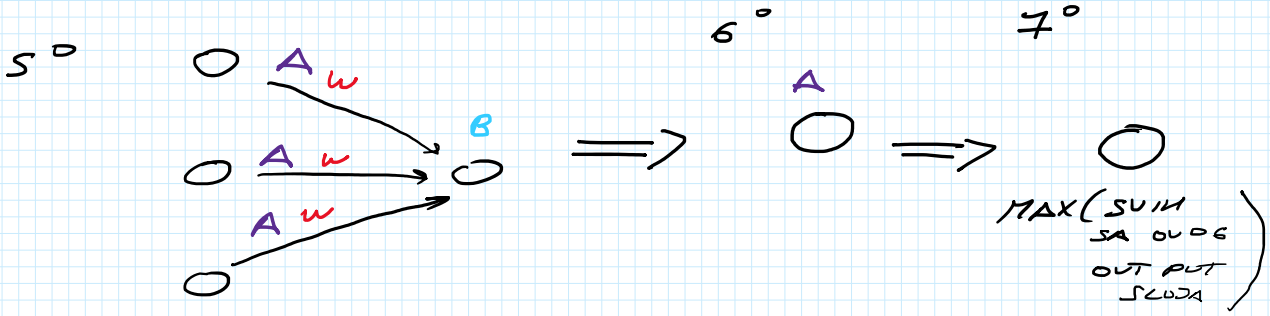
FORWARD PROPAGATION U 7 KORAKA

```python
def runNN(x):
    z1 = tf.add(tf.matmul(x, w['1']), b['1'])
    a1 = f['1'](z1)
    z2 = tf.add(tf.matmul(a1, w['2']), b['2'])
    a2 = f['2'](z2)
    z_out = tf.add(tf.matmul(a2, w['out']), b['out'])
    out = f['out'](z_out)

    pred = tf.argmax(out, 1)
    return pred, z_out
```







- PRI PRVOM PROLASKU VRIJEDNOSTI ZA MATRICE SU RANDOM POSTAVLJENE ZA WEIGHT I BIAS.

- NAKON PROLASKA IDE RAČUNANJE GREŠKE SA MSE I ONDA BACK PROPAGATION NA OSNOVU TOGA DA DIRAMO I MIJENJAMO WEIGHTS I BIAS.

- NAKON DOVOLJNO EPOHA TE RANDOM VRIJEDNOSTI BUDU SMANJENE ILI POVEĆANE NA ODGOVARAJUĆE KOJE NAM DAJU DOBRE REZULTATE

UČENJE = BACKPROPAGATION

```python
for epoch in range(nb_epochs):
    epoch_loss = 0
    nb_batches = int(len(X_train) / batch_size)
    for i in range(nb_batches):
        x_batch = X_train[i*batch_size : (i+1)*batch_size, :]
        y_batch = y_train[i*batch_size : (i+1)*batch_size]

        with tf.GradientTape() as tape:
            _, z_out = runNN(x_batch)
            loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=z_out, labels=y_batch))
        gradients = tape.gradient(loss, [w['1'], w['2'], w['out'], b['1'], b['2'], b['out']])
        opt.apply_gradients(zip(gradients, [w['1'], w['2'], w['out'], b['1'], b['2'], b['out']]))

        epoch_loss += loss

    epoch_loss /= nb_batches
    print(f'Epoch: {epoch+1}/{nb_epochs}, Avg loss: {epoch_loss:.5f}')
```

100 EPOHA

FORWARD

LOSS FUNKCIJA

BACK PROPAGATION

$$L = \text{mean}\left(-\sum_i y_{\text{batch},i} \cdot \log\left(\frac{\exp(z_{\text{out},i})}{\sum_j \exp(z_{\text{out},j})}\right)\right)$$

LOSS FUNKCIJA

POKRETANJE NAKON UČENJA

```python
pred, _ = runNN(X_test)
pred_correct = tf.equal(pred, tf.argmax(y_test, 1))
accuracy = tf.reduce_mean(tf.cast(pred_correct, tf.float32))

print(f'Test accuracy: {accuracy:.3f}')
```

ČIČA MIČA, GOTOVA JE PRIČA