

Stranice predmeta Duboko učenje (FER)

- Generativni modeli
- Ograničeni Boltzmanov stroj
 - 1. zadatak
 - 2. zadatak
 - 3. zadatak
- Varijacijski autoenkoder
 - 4. zadatak

4. vježba: Generativni modeli (GM)

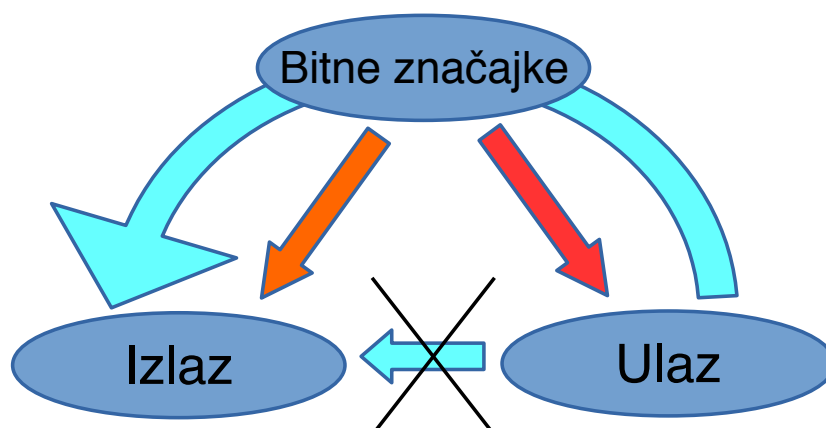
v1.2-2018

U ovoj vježbi pozabavit ćete se s generativnim modelima. Njihova glavna razlika u odnosu na diskriminativne modele je u tome što su predviđeni za generiranje uzoraka karakterističnih za distribuciju uzoraka korištenih pri treniranju. Da bi to mogli raditi na odgovarajući način, nužno je da mogu naučiti bitne karakteristike uzoraka iz skupa za treniranje. Jedna moguća reprezentacija tih bitnih karakteristika je distribucija ulaznih vektora, a model bi uz pomoć takve informacije mogao generirati više uzoraka koji su vjerojatniji (više zastupljeni u skupu za treniranje), a manje uzoraka koji su manje vjerojatni.

Distribucija uzoraka iz skupa za treniranje može se opisati distribucijom vjerojatnosti više varijabli $p(\mathbf{x})$. Vjerojatnost uzoraka za treniranje $\mathbf{x}^{(i)}$ trebala bi biti visoka dok bi vjerojatnost ostalih uzoraka trebala biti niža. Nasuprot tome, diskriminativni modeli se, na više ili manje izravne načine, fokusiraju na aposteriornu vjerojatnost razreda d

$$p(d|\mathbf{x}) = \frac{p(d)p(\mathbf{x}|d)}{p(\mathbf{x})}$$

Gornji izraz sugerira da bi poznavanje $p(\mathbf{x})$ mogla biti korisna informacija i za diskriminativne modele, iako je oni u pravilu ne koriste direktno. Ipak, logična je pretpostavka da bi preciznije poznavanje $p(\mathbf{x})$ moglo pomoći u boljoj procjeni $p(d|\mathbf{x})$. Tu ideju dodatno podupire i razumna pretpostavka, da su i ulazni uzorci i odgovarajući razred d (izlaz), posljedica istih bitnih značajki. Ulazni uzorci sadrže u sebi znatnu količinu bitnih informacija, ali često sadrže i šum koji dodatno otežava modeliranje direktne veze sa izlazom. Model veze izlaza i bitnih značajki, očekivano je jednostavniji nego direktna veza ulaza i izlaza.



Ovakva razmišljanja upućuju na upotrebu generativnih modela za ekstrakciju bitnih značajki. Njihova primarna namjena - generiranje uzoraka - tada je u drugom planu. Nakon treniranja, na njih se može nadograditi dodatni diskriminativni model (npr. MLP) koji na temelju bitnih značajki "lako" određuje željeni izlaz. Ova vježba fokusira se na treniranje generativnih modela.

Ograničeni Boltzmanov stroj (RBM)

Boltzmanov stroj (BM) je **stohastička rekurzivna generativna** mreža koja treniranjem nastoji maksimizirati $p(\mathbf{x}^{(i)})$, a temelji se na Boltzmanovoj distribuciji prema kojoj je vjerojatnost stanja \mathbf{x} to manja, što je veća njegova energija $E(\mathbf{x})$ prema slijedećem izrazu

$$p(\mathbf{x}) \propto e^{-\frac{E(\mathbf{x})}{kT}}$$

Umnožak Boltzmanove konstanta k i termodinamičke temperature T se ignorira, odnosno postavlja na 1.

Pojedini elementi stanja BM-a x_j su binarni i mogu poprimiti vrijednosti 0 i 1. Energetska funkcija $E(\mathbf{x})$ kod BM-a određena je elementima stanja x_j i težinama veza w_{ji} između njih te pripadajućim pomacima b_j .

$$E(\mathbf{x}) = - \left(\frac{1}{2} \sum_{i=1}^N \sum_{\substack{j=1 \\ j \neq i}}^N w_{ji} x_j x_i + \sum_{j=1}^N b_j x_j \right) = - \left(\frac{1}{2} \mathbf{x}^T \mathbf{W} \mathbf{x} + \mathbf{b}^T \mathbf{x} \right)$$

Matrica \mathbf{W} je simetrična te ima nule na glavnoj dijagonali. Vjerojatnost pojedinog uzorka definiramo kao

$$p(\mathbf{x}; \mathbf{W}, \mathbf{b}) = \frac{e^{-E(\mathbf{x})/T}}{\sum_{\mathbf{x}} e^{-E(\mathbf{x})/T}} = \frac{e^{\frac{1}{2} \mathbf{x}^T \mathbf{W} \mathbf{x} + \mathbf{b}^T \mathbf{x}}}{Z(\mathbf{W}, \mathbf{b})}$$

$Z(\mathbf{W})$ se naziva particijska funkcija, a uloga joj je normaliziranje vjerojatnosti kako bi

$$\sum_{\mathbf{x}} p(\mathbf{x}; \mathbf{W}, \mathbf{b}) = 1$$

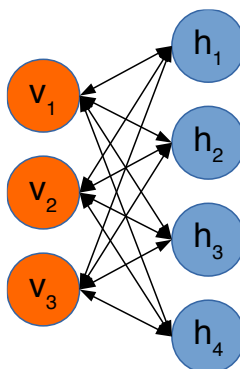
Prema odabranoj energetske funkciji i Boltzmanovoj distribuciji određena je vjerojatnost da pojedini element mreže ima vrijednost 1

$$p(x_j = 1) = \frac{1}{1 + e^{-\sum_{i=1}^N w_{ji} x_i - b_j}} = \sigma \left(\sum_{i=1}^N w_{ji} x_i + b_j \right)$$

Kako bismo energetske funkcijom BM-a mogli opisati korelacije višeg reda, odnosno kompleksnije međusobne veze pojedinih elemenata vektora podataka, uvodimo tzv. skrivene varijable \mathbf{h} . Stvarne podatke nazivamo vidljivim slojem i označavamo s \mathbf{v} , dok skrivene varijable čine skriveni sloj \mathbf{h} .

$$\mathbf{x} = (\mathbf{v}, \mathbf{h})$$

RBM je mreža u kojoj nisu dozvoljene međusobne povezanosti unutar istog sloja. To ograničenje (od tuda ime Restricted Boltzman Machine) omogućuje jednostavno osvježavanje stanja mreže. Iako ima poznatu svrhu, skriveni sloj \mathbf{h} i njegova distribucija $p(\mathbf{h})$ nisu poznati.



Energija mreže sada postaje

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{v}^T \mathbf{W} \mathbf{h} - \mathbf{b}^T \mathbf{h} - \mathbf{a}^T \mathbf{v}$$

Matrica \mathbf{W} sadrži težine veza između vidljivog i skrivenog sloja i više nije simetrična, a vektori \mathbf{a} i \mathbf{b} sadrže pomake vidljivog i skrivenog sloja. Prema novoj strukturi i prethodnoj jednadžbi za vjerojatnost pojedinog elementa dobivamo dvije jednadžbe za osvježavanje stanja RBM-a.

$$p(v_i = 1) = \sigma \left(\sum_{j=1}^N w_{ji} h_j + a_i \right) \text{ za vidljivi sloj}$$

$$p(h_j = 1) = \sigma \left(\sum_{i=1}^N w_{ji} v_i + b_j \right) \text{ za skriveni sloj}$$

Uzorkovanje vrijednosti pojedine varijable provodi se prema gornje dvije jednadžbe i pomoću generatora slučajnih brojeva.

```
def sample_prob(probs):
    """Uzorkovanje vektora x prema vektoru vjerojatnosti p(x=1) = probs"""
    return tf.to_float(tf.random_uniform(tf.shape(probs)) <= probs)
```

Učenje RBM-a

Prisjetimo se da želimo maksimizirati vjerojatnost svih uzoraka za učenje (stvaranih podataka) koji su u RBM-u predstavljeni vidljivim slojem. Stoga maksimiziramo umnožak svih $p(\mathbf{v}^{(j)})$ gdje je

$$p(\mathbf{v}; \mathbf{W}, \mathbf{a}, \mathbf{b}) = \sum_{\mathbf{h}} p(\mathbf{v}, \mathbf{h}; \mathbf{W}, \mathbf{a}, \mathbf{b}) = \sum_{\mathbf{h}} \frac{e^{\mathbf{v}^T \mathbf{W} \mathbf{h} + \mathbf{b}^T \mathbf{h} + \mathbf{a}^T \mathbf{v}}}{Z(\mathbf{W}, \mathbf{a}, \mathbf{b})}$$

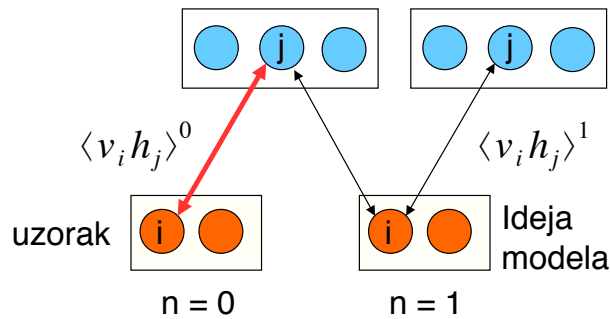
Maksimiziramo logaritam umnoška vjerojatnosti svih vidljivih vektora.

$$\ln \left[\prod_{n=1}^N p(\mathbf{v}^{(n)}; \mathbf{W}, \mathbf{a}, \mathbf{b}) \right]$$

Da bismo to postigli trebamo odrediti parcijalne derivacije s obzirom na parametre mreže

$$\begin{aligned} \frac{\partial}{\partial w_{ij}} \ln \left[\prod_{n=1}^N p(\mathbf{v}^{(n)}; \mathbf{W}, \mathbf{a}, \mathbf{b}) \right] &= \sum_{n=1}^N \left[v_i^{(n)} h_j^{(n)} - \sum_{\mathbf{v}, \mathbf{h}} v_i h_j p(\mathbf{v}, \mathbf{h}; \mathbf{W}, \mathbf{a}, \mathbf{b}) \right] = N \left[\langle v_i h_j \rangle_{P(\mathbf{h}|\mathbf{v}^{(n)}; \mathbf{W}, \mathbf{b})} - \langle v_i h_j \rangle_{P(\mathbf{v}, \mathbf{h}; \mathbf{W}, \mathbf{a}, \mathbf{b})} \right] \\ \frac{\partial}{\partial b_j} \ln \left[\prod_{n=1}^N p(\mathbf{v}^{(n)}; \mathbf{W}, \mathbf{a}, \mathbf{b}) \right] &= \sum_{n=1}^N \left[h_j^{(n)} - \sum_{\mathbf{v}, \mathbf{h}} h_j p(\mathbf{v}, \mathbf{h}; \mathbf{W}, \mathbf{a}, \mathbf{b}) \right] = N \left[\langle h_j \rangle_{P(\mathbf{h}|\mathbf{v}^{(n)}; \mathbf{W}, \mathbf{b})} - \langle h_j \rangle_{P(\mathbf{v}, \mathbf{h}; \mathbf{W}, \mathbf{a}, \mathbf{b})} \right] \\ \frac{\partial}{\partial a_j} \ln \left[\prod_{n=1}^N p(\mathbf{v}^{(n)}; \mathbf{W}, \mathbf{a}, \mathbf{b}) \right] &= \sum_{n=1}^N \left[v_j^{(n)} - \sum_{\mathbf{v}, \mathbf{h}} v_j p(\mathbf{v}, \mathbf{h}; \mathbf{W}, \mathbf{a}, \mathbf{b}) \right] = N \left[\langle v_j \rangle - \langle v_j \rangle_{P(\mathbf{v}, \mathbf{h}; \mathbf{W}, \mathbf{a}, \mathbf{b})} \right] \end{aligned}$$

Konačni izrazi sve tri jednadžbe sadrže po dvije komponente u kojima $\langle \rangle$ zagrade označavaju usrednjene vrijednosti za N ulaznih uzoraka (obično je to veličina mini grupe). Prvi pribrojnici u konačnim izrazima odnose se na stanja mreže kada su ulazni uzorci fiksirani u vidljivom sloju. Za određivanje odgovarajućih stanja skrivenog sloja \mathbf{h} dovoljno je svaki element h_j odrediti prema izrazu za $p(h_j = 1)$. Drugi pribrojnici odnose se na stanja mreže bez fiksnog vidljivog sloja pa se ta stanja mogu interpretirati kao nešto što mreža zamišlja na temelju trenutne konfiguracije parametara (\mathbf{W} , \mathbf{a} i \mathbf{b}). Da bi došli do tih stanja trebamo iterativno naizmjenice računati nova stanja slojeva (**Gibsovo uzorkovanje**) prema izrazima za $p(h_j = 1)$ i $p(v_i = 1)$. Zbog izostanka međusobnih veza elemenata istog sloja, u jednoj iteraciji se prvo paralelno uzorkuju svi skriveni elementi, a nakon toga svi elementi vidljivog sloja. Teoretski, broj iteracija treba biti velik kao bi dobili ono što mreža "stvarno" misli, odnosno kako bi došli do stacionarne distribucije. Tada je svejedno koje početno stanje vidljivog sloja uzmemo. Praktično rješenje ovog problema je Contrastive Divergence (CD) algoritam gdje je dovoljno napraviti svega k iteracija (gdje je k mali broj, često i samo 1), a za početna stanja vidljivog sloja uzimamo ulazne uzorke. Iako je ovo odmak od teorije, u praksi se pokazalo da dobro funkcionira. Vizualizacija CD-1 algoritma dana je na slici.



Korekcija težina i pomaka za ulazni uzorak v_i , tada se može realizirati na slijedeći način:

$$\Delta w_{ij} = [\langle v_i h_j \rangle^0 - \langle v_i h_j \rangle^1] \quad \Delta b_j = \eta [\langle h_j \rangle^0 - \langle h_j \rangle^1] \quad \Delta a_i = \eta [\langle v_i \rangle^0 - \langle v_i \rangle^1]$$

Faktor učenja η obično se postavlja na vrijednost manju od 1. Prvi pribrojnik u izrazu za Δw_{ij} često se naziva pozitivna faza, a drugi pribrojnik, negativna faza.

U zadacima će se koristiti MNIST baza. Iako pikseli u MNIST slikama mogu poprimiti realne vrijednosti iz raspona $[0, 1]$, svaki piksel možemo promatrati kao vjerojatnost binarne varijable $p(v_i = 1)$. Ulazne varijable tada možemo tretirati kao stohastičke binarne varijable s [Bernoulijevom razdiobom](#) i zadanom vjerojatnosti distribucijom $p(v_i = 1)$.

1. zadatak

Implementirajte RBM koji koristi CD-1 za učenje. Ulazni podaci neka su MNIST brojevi. Vidljivi sloj tada mora imati 784 elementa, a skriveni sloj neka ima 100 elemenata. Kako su vrijednosti ulaznih uzoraka (slika) realne u rasponu $[0, 1]$, oni mogu poslužiti kao $p(v_i = 1)$ pa za inicijalne vrijednosti vidljivog sloja trebate provesti uzorkovanje. Koristite mini grupe veličine 100 uzoraka, a treniranje neka ima 100 epoha.

Podzadaci:

1. Vizualizirajte težine \mathbf{W} ostvarene treniranjem te pokušajte interpretirati ostvarene težine pojedinih skrivenih neurona.
2. Vizualizirajte rezultate rekonstrukcije prvih 20 testnih uzoraka MNIST baze. Kao rezultat rekonstrukcije koji ćete prikazati, koristite $p(v_i = 1) = \sigma \left(\sum_{j=1}^N w_{ji} h_j + a_i \right)$, umjesto binarnih vrijednosti dobivenih uzorkovanjem.
3. Ispitajte učestalost uključenosti elemenata skrivenog sloja te vizualizirajte naučene težine \mathbf{W} soritrane prema učestalosti
4. Preskočite inicijalno uzorkovanje/binarizaciju na temelju ulaznih uzoraka, već ulazne uzorke (realne u rasponu $[0, 1]$) koristite kao ulazne vektore \mathbf{v} . Koliko se tako dobijeni RBM razlikuje od prethodnog?
5. Povećajte broj Gibsovih uzorkovanja k u CD- k . Koje su razlike?
6. Provedite eksperimente za manji i veći broj skrivenih neurona. Što opažate kod težina i rekonstrukcija?
7. Ispitajte efekt variranja koeficijenta učenja.
8. Slučajno inicijalizirajte skriveni sloj, provedite nekoliko Gibbsovih uzorkovanja te vizualizirajte generirane uzorke vidljivog sloja

Koristite slijedeći predložak s pomoćnom datotekom [utils.py](#).

NAPOMENA: Osim nadopunjavanja koda koji nedostaje, predložak se treba prilagođavati prema potrebi, a može i prema vlastitim preferencijama. Stoga **budite oprezni s tvrdnjama da neki dio koda ne radi!**

```
import tensorflow as tf
import numpy as np
from tensorflow.examples.tutorials.mnist import input_data
from utils import tile_raster_images
import math
import matplotlib.pyplot as plt
```

```

%matplotlib inline
plt.rcParams['image.cmap'] = 'jet'

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
trX, trY, teX, teY = mnist.train.images, mnist.train.labels, mnist.test.images, \
    mnist.test.labels

def weights(shape):
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)

def bias(shape):
    initial = tf.zeros(shape, dtype=tf.float32)
    return tf.Variable(initial)

def sample_prob(probs):
    """Uzorkovanje vektora x prema vektoru vjerojatnosti p(x=1) = probs"""
    return tf.to_float(tf.random_uniform(tf.shape(probs)) <= probs)

def draw_weights(W, shape, N, stat_shape, interpolation="bilinear"):
    """Vizualizacija težina
    W -- vektori težina
    shape -- tuple dimenzije za 2D prikaz težina - obično dimenzije ulazne slike, npr.
    N -- broj vektora težina
    shape_state -- dimezije za 2D prikaz stanja (npr. za 100 stanja (10,10)
    """
    image = (tile_raster_images(
        X=W.T,
        img_shape=shape,
        tile_shape=(int(math.ceil(N/stat_shape[0])), stat_shape[0]),
        tile_spacing=(1, 1))
    plt.figure(figsize=(10, 14))
    plt.imshow(image, interpolation=interpolation)
    plt.axis('off')

def draw_reconstructions(ins, outs, states, shape_in, shape_state, N):
    """Vizualizacija ulaza i pripadajućih rekonstrukcija i stanja skrivenog sloja
    ins -- ualzni vektori
    outs -- rekonstruirani vektori
    states -- vektori stanja skrivenog sloja
    shape_in -- dimezije ulaznih slika npr. (28,28)
    shape_state -- dimezije za 2D prikaz stanja (npr. za 100 stanja (10,10)
    N -- broj uzoraka
    """
    plt.figure(figsize=(8, int(2 * N)))
    for i in range(N):
        plt.subplot(N, 4, 4*i + 1)
        plt.imshow(ins[i].reshape(shape_in), vmin=0, vmax=1, interpolation="nearest")
        plt.title("Test input")
        plt.axis('off')
        plt.subplot(N, 4, 4*i + 2)
        plt.imshow(outs[i][0:784].reshape(shape_in), vmin=0, vmax=1, interpolation="nearest")
        plt.title("Reconstruction")
        plt.axis('off')
        plt.subplot(N, 4, 4*i + 3)
        plt.imshow(states[i].reshape(shape_state), vmin=0, vmax=1, interpolation="nearest")
        plt.title("States")
        plt.axis('off')
    plt.tight_layout()

def draw_generated(stin, stout, gen, shape_gen, shape_state, N):
    """Vizualizacija zadanih skrivenih stanja, konačnih skrivenih stanja i pripadajućih
    stin -- početni skriveni sloj

```

```

stout -- rekonstruirani vektori
gen -- vektori stanja skrivenog sloja
shape_gen -- dimezije ulaznih slika npr. (28,28)
shape_state -- dimezije za 2D prikaz stanja (npr. za 100 stanja (10,10)
N -- broj uzoraka
"""

plt.figure(figsize=(8, int(2 * N)))
for i in range(N):

    plt.subplot(N, 4, 4*i + 1)
    plt.imshow(stin[i].reshape(shape_state), vmin=0, vmax=1, interpolation="nearest")
    plt.title("set state")
    plt.axis('off')
    plt.subplot(N, 4, 4*i + 2)
    plt.imshow(stout[i][0:784].reshape(shape_state), vmin=0, vmax=1, interpolation="nearest")
    plt.title("final state")
    plt.axis('off')
    plt.subplot(N, 4, 4*i + 3)
    plt.imshow(gen[i].reshape(shape_gen), vmin=0, vmax=1, interpolation="nearest")
    plt.title("generated visible")
    plt.axis('off')
plt.tight_layout()

Nh = 100 # Broj elemenata prvog skrivenog sloja
h1_shape = (10,10)
Nv = 784 # Broj elemenata vidljivog sloja
v_shape = (28,28)
Nu = 5000 # Broj uzoraka za vizualizaciju rekonstrukcije

gibbs_sampling_steps = 1
alpha = 0.1

g1 = tf.Graph()
with g1.as_default():

    X1 = tf.placeholder("float", [None, 784])
    w1 = weights([Nv, Nh])
    vb1 = bias([Nv])
    hb1 = bias([Nh])

    h0_prob =
    h0 = sample_prob(h0_prob)
    h1 = h0

    for step in range(gibbs_sampling_steps):
        v1_prob =
        v1 =
        h1_prob =
        h1 =

    w1_positive_grad =
    w1_negative_grad =

    dw1 = (w1_positive_grad - w1_negative_grad) / tf.to_float(tf.shape(X1)[0])

    update_w1 = tf.assign_add(w1, alpha * dw1)
    update_vb1 = tf.assign_add(vb1, alpha * tf.reduce_mean(X1 - v1, 0))
    update_hb1 = tf.assign_add(hb1, alpha * tf.reduce_mean(h0 - h1, 0))

    out1 = (update_w1, update_vb1, update_hb1)

    v1_prob =

```

```

v1 =

err1 = X1 - v1_prob
err_sum1 = tf.reduce_mean(err1 * err1)

initialize1 = tf.global_variables_initializer()

batch_size = 100
epochs = 100
n_samples = mnist.train.num_examples
total_batch = int(n_samples / batch_size) * epochs

sess1 = tf.Session(graph=g1)
sess1.run(initialize1)

for i in range(total_batch):
    batch, label = mnist.train.next_batch(batch_size)
    err, _ = sess1.run([err_sum1, out1], feed_dict={X1: batch})

    if i%(int(total_batch/10)) == 0:
        print(i, err)

wls = w1.eval(session=sess1)
vb1s = vb1.eval(session=sess1)
hb1s = hb1.eval(session=sess1)
vr, h1s = sess1.run([v1_prob, h1], feed_dict={X1: teX[0:Nu,:]})

# vizualizacija težina
draw_weights(wls, v_shape, Nh, h1_shape)

# vizualizacija rekonstrukcije i stanja
draw_reconstructions(teX, vr, h1s, v_shape, h1_shape, 200)

# vizualizacija jedne rekonstrukcije s postepenim dodavanjem doprinosa aktivnih skriven.
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def draw_rec(inp, title, size, Nrows, in_a_row, j):
    """ Iscrtavanje jedne iteracije u kreiranju vidljivog sloja
    inp - vidljivi sloj
    title - naslov sličice
    size - 2D dimenzije vidljivog sloja
    Nrows - maks. broj redaka sličica
    in-a-row . broj sličica u jednom redu
    j - pozicija sličice u gridu
    """
    plt.subplot(Nrows, in_a_row, j)
    plt.imshow(inp.reshape(size), vmin=0, vmax=1, interpolation="nearest")
    plt.title(title)
    plt.axis('off')

def reconstruct(ind, states, orig, weights, biases):
    """ Slijedno iscrtavanje rekonstrukcije vidljivog sloja
    ind - indeks znamenke u orig (matrici sa znamenkama kao recima)
    states - vektori stanja ulaznih vektora
    orig - originalni ulazni vektori
    weights - matrica težina
    biases - vektori pomaka vidljivog sloja
    """
    j = 1
    in_a_row = 6
    Nimg = states.shape[1] + 3

```

[illegible]


```

out_1 = sess1.run(v1), feed_dict={h0: r_input})

# Emulacija dodatnih Gibbsovih uzorkovanja pomoću feed_dict
for i in range(1000):
    out_1_prob, out_1, hout1 = sess1.run(v1_prob, v1, h1), feed_dict={x1: out_1})

draw_generated(r_input, hout1, out_1_prob, v_shape, h1_shape, 50)

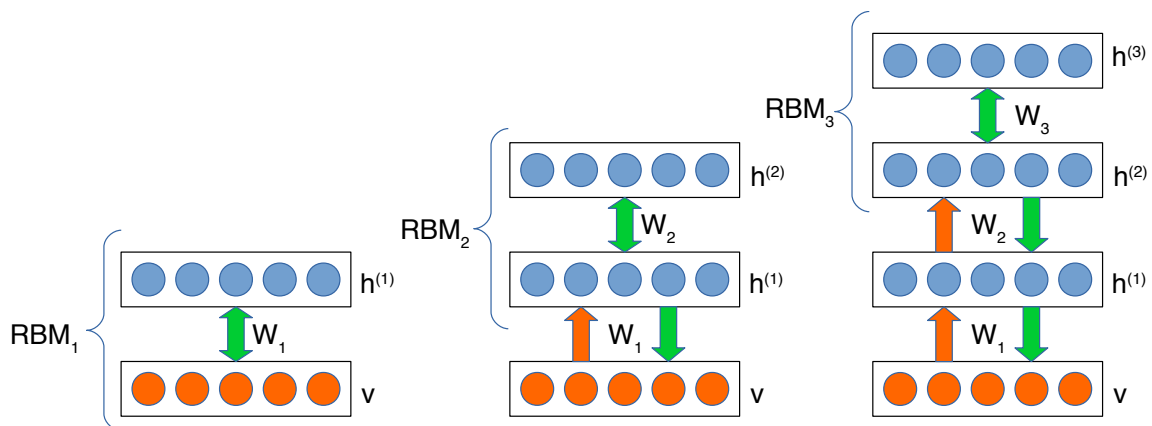
```

2. zadatak

Deep belief Network (DBN) je duboka mreža koja se dobije slaganjem više RBM-ova jednog na drugi, pri čemu se svaki slijedeći RBM pohlepno trenira pomoću skrivenog ("izlaznog") sloja prethodnog RBM-a (osim prvog RBM-a koji se trenira direktno s ulaznim uzorcima). Teoretski, tako izgrađen DBN trebao bi povećati $p(\mathbf{v})$ što nam je i cilj. Korištenje DBN, odnosno rekonstrukcija ulaznog uzorka provodi se prema donjoj shemi. U prolazu prema gore određuju se skriveni slojevi iz vidljivog sloja dok se ne dođe do najgornjeg RBM-a, zatim se na njemu provede CD-k algoritam, nakon čega se, u prolasku prema dolje, određuju niži skriveni slojevi dok se ne dođe do rekonstruiranog vidljivog sloja. Težine između pojedinih slojeva su iste u prolazu gore kao i u prolazu prema dolje. Implementirajte troslojni DBN koji se sastoji od dva pohlepno pretrenirana RBM-a. Prvi RBM neka je isit kao i u 1. zadatku, a drugi RBM neka ima skriveni sloj od 100 elemenata.

Podzadaci:

1. Vizualizirajte težine \mathbf{W}_1 i \mathbf{W}_2 ostvarene treniranjem.
2. Vizualizirajte rezultate rekonstrukcije prvih 20 testnih uzoraka MNIST baze.
3. Postavite broj skrivenih varijabli gornjeg RBM-a jednak broju elemenata vidljivog sloja donjeg RBM-a, a inicijalne težine \mathbf{W}_2 postavite na \mathbf{W}_1^T . Koji su efekti promjene? Vizualizirajte uzorke krovnog skrivenog sloja kao matrice 28x28.
4. Slučajno inicijalizirajte krovni skriveni sloj, provedite nekoliko Gibbsovih uzorkovanja te vizualizirajte generirane uzorke vidljivog sloja - usporedite s prethodnim zadatkom.



Koristite slijedeći predložak uz prtedložak 1. zadatka:

```

Nh2 = Nh # Broj elemenata drugog skrivenog sloja
h2_shape = h1_shape

gibbs_sampling_steps = 2
alpha = 0.1

g2 = tf.Graph()
with g2.as_default():

    x2 = tf.placeholder("float", [None, Nh])
    wla = tf.Variable(wls)

```

```

vb1a = tf.Variable(vb1s)
hb1a = tf.Variable(hb1s)
w2 = weights([Nh, Nh2])
hb2 = bias([Nh2])

hlup_prob =
hlup =
h2up_prob =
h2up =
h2down = h2up

for step in range(gibbs_sampling_steps):
    h1down_prob =
    h1down =
    h2down_prob =
    h2down =

w2_positive_grad =
w2_negative_grad =

dw2 = (w2_positive_grad - w2_negative_grad) / tf.to_float(tf.shape(hlup)[0])

update_w2 = tf.assign_add(w2, alpha * dw2)
update_hb1a = tf.assign_add(hb1a, alpha * tf.reduce_mean(hlup - h1down, 0))
update_hb2 = tf.assign_add(hb2, alpha * tf.reduce_mean(h2up - h2down, 0))

out2 = (update_w2, update_hb1a, update_hb2)

# rekonstrukcija ulaza na temelju krovnog skrivenog stanja h3
# ...
# ...
v_out_prob =
v_out =

err2 = X2 - v_out_prob
err_sum2 = tf.reduce_mean(err2 * err2)

initialize2 = tf.global_variables_initializer()

batch_size = 100
epochs = 100
n_samples = mnist.train.num_examples

total_batch = int(n_samples / batch_size) * epochs

sess2 = tf.Session(graph=g2)
sess2.run(initialize2)
for i in range(total_batch):
    # iteracije treniranja
    #...
    #...
    if i%(int(total_batch/10)) == 0:
        print(i, err)

w2s, hb1s, hb2s = sess2.run([w2, hb1a, hb2], feed_dict={X2: batch})
vr2, h2downs = sess2.run([v_out_prob, h2down], feed_dict={X2: tex[0:Nu,:]})

# vizualizacija težina
draw_weights(w2s, h1_shape, Nh2, h2_shape, interpolation="nearest")

# vizualizacija rekonstrukcije i stanja
draw_reconstructions(tex, vr2, h2downs, v_shape, h2_shape, 200)

```

```
# Generiranje uzoraka iz slučajnih vektora krovnog skrivenog sloja
#...
#...
# Emulacija dodatnih Gibbsovih uzorkovanja pomoću feed_dict
#...
#...
```

Kako bi se dodatno poboljšala generativna svojstva DBN-a, može se provesti generativni fine-tuning parametara mreže. U 2. zadatku, prilikom rekonstruiranja korištene su iste težine i pomaci u prolascima prema dolje i prema gore. Kod fine-tuninga, parametri koji vežu sve slojeve osim dva najgornja, razdvajaju se u dva skupa. Matrice težina između nižih slojeva dijele se na: \mathbf{R}_n za prolaz prema gore i \mathbf{W}'_n za prolaz prema dolje. Inicijalno su obje matrice jednake originalnoj matrici \mathbf{W}_n . Kod prolaza prema gore (faza budnosti - wake phase) određuju se nova stanja viših skrivenih slojeva $\mathbf{s}^{(n)}$ iz nižih stanja $\mathbf{s}^{(n-1)}$ pomoću matrica \mathbf{R} postupkom uzorkovanja ($\text{sample}(\sigma(\mathbf{R}_n \mathbf{s}^{(n-1)} + \mathbf{b}_n^{up})) \rightarrow \mathbf{s}^{(n)}$). Pri prolasku prema dolje (faza spavanja - sleep phase) određuju se "rekonstrukcije" nižih stanja $\mathbf{s}^{(n-1)}$ iz $\mathbf{s}^{(n)}$ i matrica \mathbf{W}' ($\text{sample}(\sigma(\mathbf{W}'_n \mathbf{s}^{(n)} + \mathbf{b}_{n-1}^{down})) \rightarrow \mathbf{s}^{(n-1)}$). Najgornja dva sloja su klasični RBM i dijele istu matricu težina za prolazke u oba smjera, a modificiranje tih težina provodi se na isti način kao u 1. zadatku.

Treniranje težina između nižih slojeva je drugačije. Matrice \mathbf{W}'_n se korigiraju kada se određuju nova stanja pomoću matrica \mathbf{R}_n u prolasku prema gore. U prolasku prema dolje korigiraju se matrice \mathbf{R}_n . Vektori pomaka pojedinih slojeva \mathbf{b}_n se isto dijele na varijante za prolaz prema gore \mathbf{b}_n^{up} i za prolaz prema dolje \mathbf{b}_n^{down} . Inicijalni pomaci jednaki su originalnim pomacima \mathbf{b} .

Za korekciju matrica \mathbf{W}'_n prilikom prolaska prema gore ($\text{sample}(\sigma(\mathbf{R}_n \mathbf{s}^{(n-1)} + \mathbf{b}_n^{up})) \rightarrow \mathbf{s}^{(n)}$) provodi se i $\text{sample}(\sigma(\mathbf{W}'_n \mathbf{s}^{(n)} + \mathbf{b}_{n-1}^{down})) \rightarrow \mathbf{s}^{(n-1)_{novo}}$. Korekcija elemenata radi se na slijedeći način $\Delta w'_{ij} = \eta s_j^{(n)} (s_i^{(n-1)} - s_i^{(n-1)_{novo}})$. Korekcija pomaka za prolaz prema dolje provodi se na slijedeći način $\Delta b_i^{down} = \eta (s_i^{(n-1)} - s_i^{(n-1)_{novo}})$.

Za korekciju matrica \mathbf{R}_n prilikom prolaska prema dolje ($\text{sample}(\sigma(\mathbf{W}'_n \mathbf{s}^{(n)} + \mathbf{b}_{n-1}^{down})) \rightarrow \mathbf{s}^{(n-1)}$) provodi se i $\text{sample}(\sigma(\mathbf{R}_n \mathbf{s}^{(n-1)} + \mathbf{b}_n^{up})) \rightarrow \mathbf{s}^{(n)_{novo}}$. Korekcija elemenata radi se na slijedeći način $\Delta r_{ij} = \eta s_i^{(n-1)} (s_j^{(n)} - s_j^{(n)_{novo}})$. Korekcija pomaka za prolaz prema dolje provodi se na slijedeći način $\Delta b_i^{up} = \eta (s_i^{(n)} - s_i^{(n)_{novo}})$.

Navedeni postupak provodi se za svaki uzorak za treniranje te se naziva up-down algoritam (ponegdje i wake-sleep algoritam).

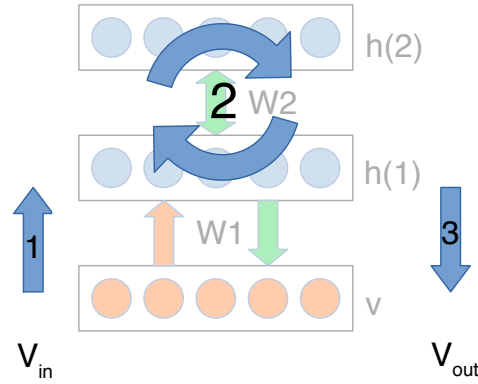
HINT: pseudokod za treniranje četveroslojnog DBN-a nalazi se u dodacima ovog [članka](#)

3. Zadatak

Implementirajte postupak generativnog fine-tuninga na DBN iz 2. zadatka. Za treniranje gornjeg RBM-a koristite CD-2.

Podzadaci:

1. Vizualizirajte konačne varijante matrica \mathbf{W}' i \mathbf{R} .
2. Vizualizirajte rezultate rekonstrukcije prvih 20 testnih uzoraka MNIST baze.
3. Slučajno inicijalizirajte krovni skriveni sloj, provedite nekoliko Gibbsovih uzorkovanja te vizualizirajte generirane uzorke vidljivog sloja - usporedite s prethodnim zadacima



Koristite slijedeći predložak, kao i predloške 1. i 2. zadatka.

```
#
beta = 0.01

g3 = tf.Graph()
with g3.as_default():

    X3 = tf.placeholder("float", [None, Nv])
    r1_up = tf.Variable(w1s)
    w1_down = tf.Variable(tf.transpose(w1s))
    w2a = tf.Variable(w2s)
    hb1_up = tf.Variable(hb1s)
    hb1_down = tf.Variable(hb1as)
    vb1_down = tf.Variable(vb1s)
    hb2a = tf.Variable(hb2s)

    # wake pass
    h1_up_prob =
    h1_up = #  $s^{(n)}$  u pripremi
    v1_up_down_prob =
    v1_up_down = #  $s^{(n-1)}$  u tekstu pripreme

    # top RBM Gibbs passes
    h2_up_prob =
    h2_up =
    h2_down = h2_up
    for step in range(gibbs_sampling_steps):
        h1_down_prob =
        h1_down =
        h2_down_prob =
        h2_down =

    # sleep pass
    v1_down_prob =
    v1_down = #  $s^{(n-1)}$  u pripremi
    h1_down_up_prob =
    h1_down_up = #  $s^{(n)}$  u pripremi

    # generative weights update during wake pass
    update_w1_down = tf.assign_add(w1_down, beta * tf.matmul(tf.transpose(h1_up), X3 - v1_down))
    update_vb1_down = tf.assign_add(vb1_down, beta * tf.reduce_mean(X3 - v1_down))

    # top RBM update
    w2_positive_grad =
```

```

w2_negative_grad =
dw3 =
update_w2 = tf.assign_add(w2a, beta * dw3)
update_hb1_down = tf.assign_add(hb1_down, beta * tf.reduce_mean(h1_up - h1_down, 0))
update_hb2 = tf.assign_add(hb2a, beta * tf.reduce_mean(h2_up - h2_down, 0))

# recognition weights update during sleep pass
update_r1_up = tf.assign_add(r1_up, beta * tf.matmul(tf.transpose(v1_down_prob), h1_
update_hb1_up = tf.assign_add(hb1_up, beta * tf.reduce_mean(h1_down - h1_down_up, 0)

out3 = (update_w1_down, update_vb1_down, update_w2, update_hb1_down, update_hb2, up

err3 = X3 - v1_down_prob
err_sum3 = tf.reduce_mean(err3 * err3)

initialize3 = tf.global_variables_initializer()

batch_size = 100
epochs = 100
n_samples = mnist.train.num_examples

total_batch = int(n_samples / batch_size) * epochs

sess3 = tf.Session(graph=g3)
sess3.run(initialize3)
for i in range(total_batch):
    #...
    err, _ = sess3.run([err_sum3, out3], feed_dict={X3: batch})

    if i%(int(total_batch/10)) == 0:
        print(i, err)

w2ss, r1_ups, w1_downs, hb2ss, hb1_ups, hb1_downs, vb1_downs = sess3.run(
    [w2a, r1_up, w1_down, hb2a, hb1_up, hb1_down, vb1_down], feed_dict={X3: batch})
vr3, h2_downs, h2_down_probs = sess3.run([v1_down_prob, h2_down, h2_down_prob], feed

# vizualizacija težina
draw_weights(r1_ups, v_shape, Nh, h1_shape)
draw_weights(w1_downs.T, v_shape, Nh, h1_shape)
draw_weights(w2ss, h1_shape, Nh2, h2_shape, interpolation="nearest")

# vizualizacija rekonstrukcije i stanja
Npics = 5
plt.figure(figsize=(8, 12*4))
for i in range(20):

    plt.subplot(20, Npics, Npics*i + 1)
    plt.imshow(testX[i].reshape(v_shape), vmin=0, vmax=1)
    plt.title("Test input")
    plt.subplot(20, Npics, Npics*i + 2)
    plt.imshow(vr[i][0:784].reshape(v_shape), vmin=0, vmax=1)
    plt.title("Reconstruction 1")
    plt.subplot(20, Npics, Npics*i + 3)
    plt.imshow(vr2[i][0:784].reshape(v_shape), vmin=0, vmax=1)
    plt.title("Reconstruction 2")
    plt.subplot(20, Npics, Npics*i + 4)
    plt.imshow(vr3[i][0:784].reshape(v_shape), vmin=0, vmax=1)
    plt.title("Reconstruction 3")
    plt.subplot(20, Npics, Npics*i + 5)
    plt.imshow(h2_downs[i][0:Nh2].reshape(h2_shape), vmin=0, vmax=1, interpolation="nearest")
    plt.title("Top states 3")
plt.tight_layout()

```

```
# Generiranje uzoraka iz slučajnih vektora krovnog skrivenog sloja
#...
#...
# Emulacija dodatnih Gibbsovih uzorkovanja pomoću feed_dict
#...
#...
```

Varijacijski autoenkoder (VAE)

Autoenkoder je mreža s prolazom u naprijed koja za treniranje koristi backpropagation te može imati duboku strukturu. Autoenkoderi su gnereativne mreže sa karakterističnom dvodjelonom strukturom. Prvi dio naziva se enkoder i preslikava (enkodira) ulazni sloj u skriveni sloj. Drugi dio je dekoder i preslikava skriveni sloj u izlazni sloj. Primarni cilj takve mreže je postići što veću sličnost ulaza i izlaza za svaki uzorak za treniranje, maksimizirajući neku mjeru sličnosti. Primarni cilj je jednostavan, što autoenkodere svrstava u mreže koje se treniraju bez nadzora. Maksimalna uspješnost može se jednostavno postići direktnim kopiranjem ulaz izlaz, no to nije u skladu sa skrivenim ciljem. Uvjetno rečeno skriveni cilj, koji je u stvari jedini interesantan, je naučiti bitne značajke uzoraka iz skupa za treniranje. Da bi se to postiglo, i izbjeglo direktno kopiranje, koriste se razne tehnike regularizacije. Alternativno, može se koristiti druga mjera uspješnosti, poput maksimiziranja vjerojatnosti. U svakom slučaju, varijable skrivenog sloja \mathbf{z} zadužene su za otkrivanje bitnih značajki ulaznih uzoraka.

Varijacijski autoenkoderi (VAE) su Autoenkoderi koji maksimiziraju vjerojatnost $p(\mathbf{x})$ svih uzoraka za treniranje $\mathbf{x}^{(i)}$. Kod VAE se ne koriste dodatne regularizacijske tehnike, ali neke od njih se mogu uključiti u VAE (npr. kombinacija VAE i denoising autoenkodera daje bolje rezultate). Bitne značajke u skrivenom sloju \mathbf{z} tada imaju ulogu u modeliranju $p(\mathbf{x})$.

$$p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z}; \theta)p(\mathbf{z})d\mathbf{z}$$

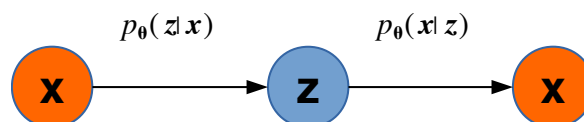
Vjerojatnosti sa desne strane jednadžbe su jednako nepoznate kao i $p(\mathbf{x})$, no njih ćemo aproksimirati Gausovim distribucijama. Θ su parametri modela i njih određujemo kroz postupak treniranja. Dodatni cilj nam je minimizirati količinu uzorkovanja, koje je obično nužno porvoditi kod estimacija nepoznatih distribucija. U ovom slučaju, pogodno je maksimizirati logaritam vjerojatnosti.

$$\log_{\theta} p(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}) = \sum_{i=1}^N \log_{\theta} p(\mathbf{x}^{(i)})$$

Za $p(\mathbf{z})$ odabiremo normalnu distribuciju

$$p(\mathbf{z}) = N(0, 1)$$

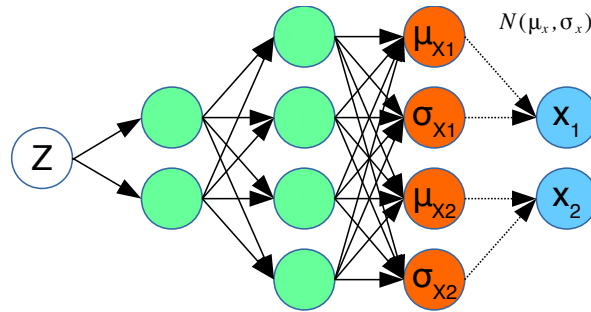
Time ih ograničavamo i naizgled onemogućavamo u tome da predstavljaju bitne značajke.



Dekoderskim dijelom modeliramo uvjetnu distribuciju $p(\mathbf{x}|\mathbf{z})$ kao normalnu distribuciju, a parametre te distribucije određuju parametri mreže Θ .

$$p_{\theta}(x|z) = N(\mu_x(z), \sigma_x(z))$$

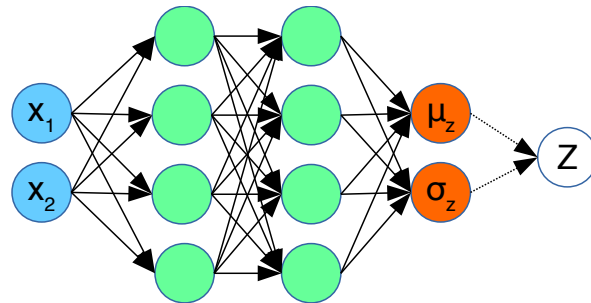
Parametri Θ uključuju težine i pomake svih neurona skrivenih slojeva dekodera te se određuju kroz treniranje. Složenost $p(\mathbf{x}|\mathbf{z})$ ovisi o broju skrivenih slojeva i broju neurona u njima. Crtkane linije na dijagramu označavaju operacije uzorkovanja. U dijagramu je radi preglednosti prikazana samo jedna varijabla z , umjesto vektora skrivenih varijabli \mathbf{z} .



Dekoderski dio

Sada bi trebalo odrediti $p(\mathbf{z}|\mathbf{x})$, takav da gornje pretpostavke dobro funkcioniraju. Kako nemamo načina odrediti odgovarajući $p(\mathbf{z}|\mathbf{x})$, aproksimirati ćemo ga normalnom distribucijom $q(\mathbf{z}|\mathbf{x})$, ali ćemo pažljivo odrediti parametre te zamjenske distribucije.

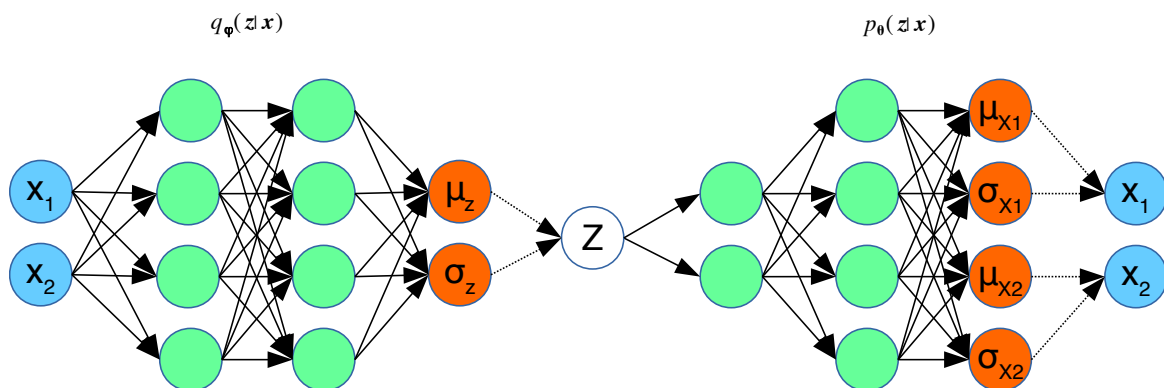
$$q_{\phi}(\mathbf{z}|\mathbf{x}) = N(\mu_{\mathbf{z}}(\mathbf{x}), \sigma_{\mathbf{z}}(\mathbf{x}))$$



Enkoderski dio

Slično kao i kod dekodera, parametri Φ uključuju težine i pomake skrivnih slojeva enkodera te se oni određuju kroz postupak treniranja. Kompleksnost $q(\mathbf{z}|\mathbf{x})$ ovisi o broju skrivenih slojeva i broju neurona u njima.

Model je sada potpun, samo nam još treba funkcija cilja koju možemo optimizirati ispravnim odabirom parametara Θ i Φ .



Neuroni koji predstavljaju srednje vrijednosti i standardne devijacije obično nemaju nelinearne aktivacijske funkcije. Kako smo već naveli, želja nam je maksimizirati

$$\log_{\theta} p(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}) = \sum_{i=1}^N \log_{\theta} p(\mathbf{x}^{(i)})$$

Prikladnom transformacijom $\log(p(\mathbf{x}))$, što je jedan pribrojnik u grnjoj jednažbi, možemo prikazati kao

$$\log(p(\mathbf{x})) = D_{KL}(q(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z}|\mathbf{x})) - D_{KL}(q(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})) + E_{q(\mathbf{z}|\mathbf{x})}(\log(p(\mathbf{x}|\mathbf{z})))$$

D_{KL} je [Kullback–Leibler divergencija](#) i predstavlja mjeru sličnosti dviju distribucija. Kako $p(\mathbf{z}|\mathbf{x})$ mijenjamo sa $q(\mathbf{z}|\mathbf{x})$, logično je težiti tome da te dvije distribucije budu što sličnije. Tada bi KL divergencija u prvom pribrojniku težila maksimumu, no kako nam je $p(\mathbf{z}|\mathbf{x})$ nepoznat, maksimiziramo preostala dva pribrojnika. Te dvije komponente zajedno čine donju varijacijsku granicu L od $\log(p(\mathbf{x}))$ te maksimizacijom donje granice podižemo ukupnu vjerojatnost ulaznog uzorka \mathbf{x} .

$$L(\theta, \phi, \mathbf{x}^{(i)}) = -D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)}) \parallel p(\mathbf{z})) + E_{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}[\log(p_\theta(\mathbf{x}^{(i)}|\mathbf{z}))]$$

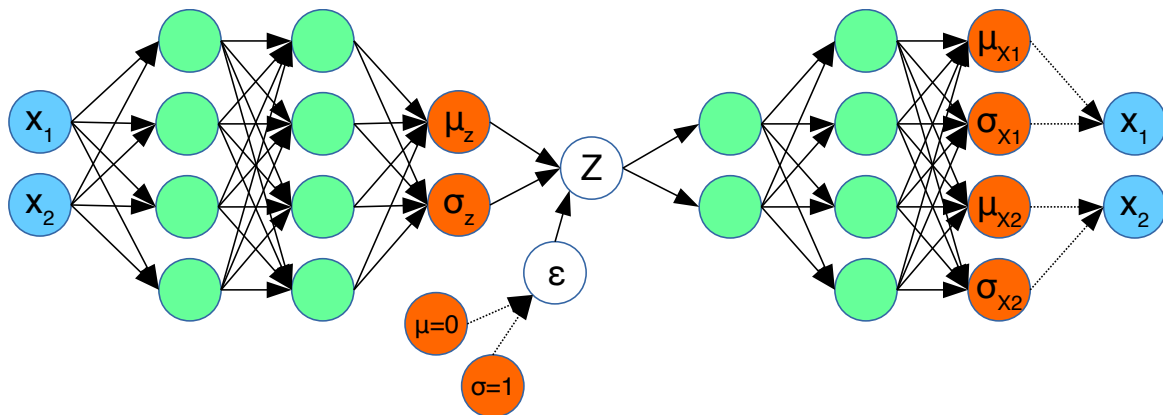
Drugi pribrojnik u gornjoj jednadžbi možemo promatrati kao mjeru uspješnosti rekonstrukcije (mogući maksimum je $\log(1)$ kada skriveni sloj \mathbf{z} omogućuje savršenu rekonstrukciju). Prvi član se smatra regularizacijskom komponentom te on potiče izjedačava je distribucija $q(\mathbf{z}|\mathbf{x})$ i $p(\mathbf{z})$.

Uz odabrane aproksimacije $q_\phi(z|x) = N(\mu_z(x), \sigma_z(x))$ $p(z) = N(0, 1)$ $p_\theta(x|z) = N(\mu_x(z), \sigma_x(z))$ dvije komponente donje varijacijske granice postaju

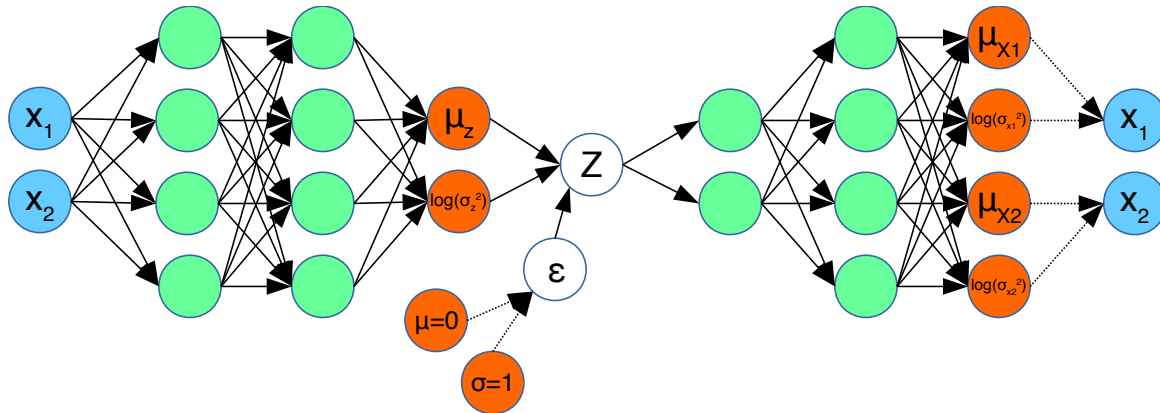
$$-D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)}) \parallel p(\mathbf{z})) = \frac{1}{2} \sum_j (1 + \log(\sigma_{z_j}^{(i)2}) - \mu_{z_j}^{(i)2} - \sigma_{z_j}^{(i)2})$$

$$E_{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}[\log(p_\theta(\mathbf{x}^{(i)}|\mathbf{z}))] \approx \frac{1}{K} \sum_{k=1}^K \log(p_\theta(\mathbf{x}^{(i)}|\mathbf{z}^{(i,k)})) \approx - \sum_j \frac{1}{2} \log(\sigma_{x_j}^{(i,k)2}) + \frac{(x_j^{(i)} - \mu_{x_j}^{(i,k)})^2}{2\sigma_{x_j}^{(i,k)2}}$$

Obično se K postavlja na 1 kako bi se smanjila količina uzorkovanja, uz uvjet da je veličina minibatcha barem 100. Konačni izrazi za dvije komponente sada nam daju konačnu funkciju cilja za jedan ulazni uzorak. Optimizira se prosječna vrijednost za sve uzorke $\mathbf{x}^{(i)}$! Prethodno je potrebno još malo modificirati strukturu mreže kako bi omogućili backpropagation i u enkoderski sloj. Nužno je stohastičke neurone \mathbf{z} pretvoriti u determinističke neurone s stohastičkim dodatkom (generatorom šuma ϵ po normalnoj razdiobi $N(0, 1)$).



Primijetite da konačna struktura mreže uključuje stohastičko uzorkovanje, no dijelovi mreže gdje se to događa, ne utječu na propagaciju gradijenta pogreške. To uključuje i same izlaze mreže koji, možda neočekivano, ne sudjeluju u funkciji cilja. U konačnom izrazu funkcije cilja, javljaju se srednje vrijednosti i standardne devijacije izlaza i skrivenih varijabli, a to su zapravo izlazi enkodera i dekodera. Za standardne devijacije je karakteristično da su one uvijek pozitivne, no izlaz iz mreže ne mora nužno biti. Kako bi se iskoristio puni raspon i smanjio broj potrebnih izračuna, izlazi mreže postavljaju se na $\log(\sigma^2)$ umjesto σ .



Konačni algoritam treniranja VAE sada je:

1. Inicijaliziraj parametre Θ i Φ
2. Ponavljaj
3. Odaberi slučajni minibatch \mathbf{X}^M
4. Uzorkuj ϵ
5. Odredi gradijent od L s obzirom na Θ i Φ
6. Izračunaj nove vrijednosti za Θ i Φ prema gradijentu
7. Dok Θ i Φ ne konvergiraju

Dakle, ovim postupkom maksimiziramo donju granicu log vjerojatnosti ulaznih uzoraka. To nam ulijeva sigurnost da će i sama log vjerojatnost rasti, ali nije garancija. Teoretski, može se desiti da donja granica raste, a sama vjerojatnost pada, ali u praksi to najčešće nije slučaj. Moguće objašnjenje ovog efekta leži u činjenici da uz dovoljno složen enkoder, $q(\mathbf{z}|\mathbf{x})$ postaje dovoljno kompleksna i omogućuje približavanje distribuciji $p(\mathbf{z}|\mathbf{x})$, čime se maksimizira i prvi (zanemareni) član izraza za $\log(p(\mathbf{x}))$.

Generiranje novih uzoraka provodi se samo dekoderskim dijelom uz slučajnu inicijalizaciju skrivenog sloja \mathbf{z} prema zadanoj distribuciji $p(\mathbf{z}) = N(\mathbf{0}, \mathbf{I})$ ili nekom ciljanim vektorom \mathbf{z} .

I u ovom zadatku se koristi MNIST baza, čije slike i ovaj puta tretiramo kao niz zamišljenih binarnih piksela x_i sa Bernoulijevom distribucijom i vjerojatnosti zadanom ulaznom vrijednosti piksela $p(x_i = 1) = x_i^{\text{in}}$. Tada je bolje da dekode implementira Bernouliju razdiobu umjesto Gaussove. Izlaz dekodera tada može predstavljati vjerojatnost $p(x_i = 1)$, što je ujedno i očekivana vrijednost izlaza x_i . Sama vjerojatnost može biti definirana kao

$$p(x_i^{\text{out}} = 1) = \sigma \left(\sum_{j=1}^N w_{ji} h_j + b_i \right)$$

gdje su \mathbf{W} i \mathbf{b} težine i pomaci koji povezuju zadnji sloj dekodera (\mathbf{h}) sa vjerojatnosti izlazne varijable x_i . U skladu s ovom izmjenom, nužno je izmijeniti funkciju cilja, preciznije, njezin dio koji se odnosi na točnost rekonstrukcije

$$\mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x}^{(i)})} [\log(p_{\theta}(\mathbf{x}^{(i)}|\mathbf{z}))]$$

Kod binarnih varijabli s Bernoulijevom razdiobom, izraz postaje

$$\mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x}^{(i)})} [\log(p_{\theta}(\mathbf{x}^{(i)}|\mathbf{z}))] = - \sum_j [x_j^{\text{in}} \log p(x_j^{\text{out}} = 1) + (1 - x_j^{\text{in}}) \log(1 - p(x_j^{\text{out}} = 1))] = \sum_j H(p(x_j = 1), p(x_j^{\text{out}} = 1))$$

H je unakrsna entropija. Srećom, Tensorflow nudi gotovu funkciju za $H(\mathbf{x}, \sigma(\mathbf{y}))$: `tf.nn.sigmoid_cross_entropy_with_logits(y, x)`. Obratite pažnju da prvi argument funkcije nije $p(x_j = 1)$!

4. Zadatak

Implementirajte VAE sa 20 skrivenih varijabli z . Ulazni podaci neka su MNIST brojevi. Enkoder i dekodeer neka imaju po dva skrivena sloja, svaki sa 200 neurona sa "softplus" aktivacijskim funkcijama.

Podzadaci:

1. Vizualizirajte rezultate rekonstrukcije za prvih 20 testnih uzoraka MNIST baze.
2. Vizualizirajte distribucije srednjih vrijednosti i standardnih devijacija skrivenih varijabli z za primjereni broj ulaznih uzoraka
3. Ponovite treniranje iz prethodna 2 podzadatka za samo 2 elementa u skrivenenom sloju z .
4. Vizualizirajte raspored testnih uzoraka u 2D prostoru skrivenih varijabli.

Koristite slijedeći predložak:

```
import numpy as np
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
import os
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['image.cmap'] = 'jet'

mnist = input_data.read_data_sets('../MNIST_data/', one_hot=True)
n_samples = mnist.train.num_examples

learning_rate = 0.001
batch_size = 100

n_hidden_recog_1=200 # 1 sloj enkodera
n_hidden_recog_2=200 # 2 sloj enkodera
n_hidden_gener_1=200 # 1 sloj dekodera
n_hidden_gener_2=200 # 2 sloj dekodera
n_z=2 # broj skrivenih varijabli
n_input=784 # MNIST data input (img shape: 28*28)
in_shape = (28,28)

def get_canvas(Z, ind, nx, ny, in_shape, batch_size, sess):
    """Crtanje rekonstrukcija na odgovarajućim pozicijama u 2D prostoru skrivenih varijabli
    Z -- skriveni vektori raspoređeni u gridu oko ishodišta
    ind -- indeksi za rezanje Z-a na batch_size blokove za slanje u graf -zbog problema
    nx -- raspon grida po x osi - skrivena varijabla z0
    ny -- raspon grida po y osi - skrivena varijabla z1
    in_shape -- dimenzije jedne rekonstrukcije i.e. ulazne sličice
    batch_size -- veličina minibatcha na koji je graf naviknut
    sess -- session grafa mreže
    """
    # get reconstructions for visualiations
    X = np.empty((0,in_shape[0]*in_shape[1])) # empty array for concatenation
    # split hidden vectors into minibatches of batch_size due to TF random generator limitation
    for batch in np.array_split(Z,ind):
        # fill up last batch to full batch_size if neccessary
        # this addition will not be visualized, but is here to avoid TF error
        if batch.shape[0] < batch_size:
            batch = np.concatenate((batch, np.zeros((batch_size-batch.shape[0], batch_size))))
        # get batch_size reconstructions and add them to array of previous reconstructions
        X = np.vstack((X, sess.run(x_reconstr_mean_out, feed_dict={z: batch})))
    # make canvas with reconstruction tiles arranged by the hidden state coordinates of
    # this is achieved for all reconstructions by clever use of reshape, swapaxes and array slicing
    return (X[0:nx*ny,:].reshape((nx*ny,in_shape[0],in_shape[1])).swapaxes(0,1)
            .reshape((in_shape[0],ny,nx*in_shape[1])).swapaxes(0,1)[::-1,:,:)
            .reshape((ny*in_shape[0],nx*in_shape[1])))
```

```

def draw_reconstructions(ins, outs, states, shape_in, shape_state):
    """Vizualizacija ulaza i pripadajućih rekonstrukcija i stanja skrivenog sloja
    ins -- ulazni vektori
    outs -- rekonstruirani vektori
    states -- vektori stanja skrivenog sloja
    shape_in -- dimezije ulaznih slika npr. (28,28)
    shape_state -- dimezije za 2D prikaz stanja (npr. za 100 stanja (10,10))
    """
    plt.figure(figsize=(8, 12*4))
    for i in range(20):

        plt.subplot(20, 4, 4*i + 1)
        plt.imshow(ins[i].reshape(shape_in), vmin=0, vmax=1, interpolation="nearest")
        plt.title("Test input")
        plt.subplot(20, 4, 4*i + 2)
        plt.imshow(outs[i][0:784].reshape(shape_in), vmin=0, vmax=1, interpolation="nearest")
        plt.title("Reconstruction")
        plt.subplot(20, 4, 4*i + 3)
        plt.imshow(states[i][0:(shape_state[0] * shape_state[1])].reshape(shape_state),
                    vmin=-4, vmax=4, interpolation="nearest")
        plt.colorbar()
        plt.title("States")
    plt.tight_layout()

def plot_latent(inmat, labels):
    """Crtanje pozicija uzoraka u 2D latentnom prostoru
    inmat -- matrica latentnih stanja
    labels -- labela klas
    """
    plt.figure(figsize=(8, 6))
    plt.axis([-4, 4, -4, 4])
    plt.gca().set_autoscale_on(False)

    plt.scatter(inmat[:, 0], inmat[:, 1], c=np.argmax(labels, 1))
    plt.colorbar()
    plt.xlabel('z0')
    plt.ylabel('z1')

def save_latent_plot(name):
    """Spremanje trenutnog figure-a
    name -- ime datoteke
    """
    plt.savefig(name)

def weight_variable(shape, name):
    """Kreiranje težina"""
    # http://andyljones.tumblr.com/post/110998971763/an-explanation-of-xavier-initializ
    return tf.get_variable(name, shape=shape,
                           initializer=tf.contrib.layers.xavier_initializer())

def bias_variable(shape):
    """Kreiranje pomaka"""
    initial = tf.zeros(shape, dtype=tf.float32)
    return tf.Variable(initial)

def variable_summaries(var, name):
    """Prikupljanje podataka za Tensorboard"""
    with tf.name_scope(name):
        mean = tf.reduce_mean(var)
        tf.summary.scalar('mean', mean)
        stddev = tf.sqrt(tf.reduce_mean(tf.square(var - mean)))
        tf.summary.scalar('stddev', stddev)

```

```

tf.summary.scalar('max', tf.reduce_max(var))
tf.summary.scalar('min', tf.reduce_min(var))
tf.summary.histogram(name, var)

def vae_layer(input_tensor, input_dim, output_dim, layer_name, act=tf.nn.softplus):
    """Kreiranje jednog skrivenog sloja"""
    # Adding a name scope ensures logical grouping of the layers in the graph.
    with tf.name_scope(layer_name):
        # This Variable will hold the state of the weights for the layer
        weights = weight_variable([input_dim, output_dim], layer_name + '/weights')
        variable_summaries(weights, 'weights')
        tf.summary.tensor_summary('weightsT', weights)
        biases = bias_variable([output_dim])
        variable_summaries(biases, 'biases')
        preactivate = tf.matmul(input_tensor, weights) + biases
        tf.summary.histogram('pre_activations', preactivate)
        activations = act(preactivate, name='activation')
        tf.summary.histogram('activations', activations)
    return activations

tf.reset_default_graph()

sess = tf.InteractiveSession()

# definicije ulaznog tenzora
x =

# definirajte enkoderski dio
layer_e1 = vae_layer(x, n_input, n_hidden_recog_1, 'layer_e1')
layer_e2 =

with tf.name_scope('z'):
    # definirajte skrivene varijable i pripadajući generator šuma
    z_mean = vae_layer(layer_e2, n_hidden_recog_2, n_z, 'z_mean', act=tf.identity)
    z_log_sigma_sq =
    eps = tf.random_normal((batch_size, n_z), 0, 1, dtype=tf.float32)

    z = tf.add(z_mean, tf.multiply(tf.sqrt(tf.exp(z_log_sigma_sq)), eps))
    tf.summary.histogram('activations', z)

# definirajte dekoderski dio
layer_d1 = vae_layer(z, n_z, n_hidden_gener_1, 'layer_d1')
layer_d2 =

# definirajte srednju vrijednost rekonstrukcije
x_reconstr_mean =

x_reconstr_mean_out = tf.nn.sigmoid(x_reconstr_mean)

# definirajte dvije komponente funkcije cijene
with tf.name_scope('cost'):
    cost1 =
    tf.summary.histogram('cross_entropy', cost1)
    cost2 =
    tf.summary.histogram('D_KL', cost2)
    cost = tf.reduce_mean(tf.reduce_sum(cost1,1) + tf.reduce_sum(cost2,1)) # average cost
    tf.summary.histogram('cost', cost)

# ADAM optimizer
with tf.name_scope('train'):
    optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)

# Prikupljanje podataka za Tensorboard

```

```

merged = tf.summary.merge_all()

init = tf.global_variables_initializer()

saver = tf.train.Saver()

n_epochs = 100
train_writer = tf.summary.FileWriter('train', sess.graph)

sess.run(init)

total_batch = int(n_samples / batch_size)
step = 0
for epoch in range(n_epochs):
    avg_cost = 0.

    for i in range(total_batch):
        batch_xs, _ = mnist.train.next_batch(batch_size)
        # Fit training using batch data
        opt, cos = sess.run([optimizer, cost], feed_dict={x: batch_xs})
        # Compute average loss
        avg_cost += cos / n_samples * batch_size

    # Display logs per epoch step
    if epoch%(int(n_epochs/10)) == 0:
        print("Epoch:", '%04d' % (epoch+1),
              "cost=", "{:.9f}".format(avg_cost))
        run_options = tf.RunOptions(trace_level=tf.RunOptions.FULL_TRACE)
        run_metadata = tf.RunMetadata()
        summary, _ = sess.run([merged, optimizer], feed_dict={x: batch_xs},
                              options=run_options, run_metadata=run_metadata)
        train_writer.add_run_metadata(run_metadata, 'epoch%03d' % epoch)
        train_writer.add_summary(summary, i)

        saver.save(sess, os.path.join('train', "model.ckpt"), epoch)

train_writer.close()

# vizualizacija rekonstrukcije i stanja
x_sample = mnist.test.next_batch(100)[0]
x_reconstruct, z_out = sess.run([x_reconstr_mean_out, z], feed_dict={x: x_sample})

draw_reconstructions(x_sample, x_reconstruct, z_out, (28, 28), (4,5)) # prilagodite dim

# Vizualizacija raspored testnih uzoraka u 2D prostoru skrivenih varijabli - 1. način
x_sample, y_sample = mnist.test.next_batch(5000)
z_mu, z_sigma = sess.run([z_mean, z_log_sigma_sq], feed_dict={x: x_sample})

plot_latent(z_mu, y_sample)
#save_latent_plot('trt.png')

# Vizualizacija raspored testnih uzoraka u 2D prostoru skrivenih varijabli - 2. način

nx = ny = 21
x_values = np.linspace(-3, 3, nx)
y_values = np.linspace(-3, 3, ny)

canvas = np.empty((28*ny, 28*nx))

# Trikovito popunjavanje rezultata za grid zbog fiksirane veličine z batcha u grafu
# Valjda će se to riješiti u nekoj budućoj verziji TF
Xi, Yi = np.meshgrid(x_values, y_values)
Z = np.column_stack((Xi.flatten(), Yi.flatten()))

```

```

X = np.empty((0,28*28))
ind = list(range(batch_size, nx*ny, batch_size))
for i in np.array_split(Z,ind):
    if i.shape[0] < batch_size:
        i = np.concatenate((i, np.zeros((batch_size-i.shape[0], i.shape[1]))), 0)
    X = np.vstack((X, sess.run(x_reconstr_mean_out, feed_dict={z: i})))

for i, yi in enumerate(y_values):
    for j, xi in enumerate(x_values):
        canvas[(nx-i-1)*28:(nx-i)*28, j*28:(j+1)*28] = X[i*nx+j].reshape(28, 28)

plt.figure(figsize=(8, 10))
plt.imshow(canvas, origin="upper")
plt.xticks( np.linspace(14,588-14,11), np.round(np.linspace(-3,3,11), 2) )
plt.yticks( np.linspace(14,588-14,11), np.round(np.linspace(3,-3,11), 2) )
plt.xlabel('z0')
plt.ylabel('z1')
plt.tight_layout()

# Vizualizacija ugašenih elemenata skrivenog sloja - 1. način

# Pomoćna funkcija za crtanje boxplot grafova
def boxplot_vis(pos, input_data, label_x, label_y):
    ax = fig.add_subplot(130+pos)
    plt.boxplot(input_data, 0, '', 0, 0.75)
    ax.set_xlabel(label_x)
    ax.set_ylabel(label_y)
    return ax

fig = plt.figure(figsize=(15,4))

# Vizualizacija statistike za z_mean
boxplot_vis(1,z_mu, 'Z mean values', 'Z elemets')

# Vizualizacija statistike za z_sigma
ax = boxplot_vis(2, np.square(np.exp(z_sigma)), 'Z sigma values', 'Z elemets')
ax.set_xlim([-0.05,1.1])

# Vizualizacija statistike za težine ulaza u dekođer
test = tf.get_default_graph().get_tensor_by_name("layer_d1/weights:0")
weights_d1 = test.eval(session=sess)
boxplot_vis(3, weights_d1.T, 'Weights to dekođer', 'Z elemets')

# Vizualizacija ugašenih elemenata skrivenog sloja - 2. način

from mpl_toolkits.mplot3d import Axes3D

# Funkcija za crtanje 3D bar grafa
def bargraph_vis(pos, input_data, dims, color, labels):
    ax = fig.add_subplot(120+pos, projection='3d')
    xpos, ypos = np.meshgrid(range(dims[0]), range(dims[1]))
    xpos = xpos.flatten('F')
    ypos = ypos.flatten('F')
    zpos = np.zeros_like(xpos)

    dx = np.ones_like(zpos)
    dy = np.ones_like(zpos) * 0.5
    dz = input_data.flatten()

    ax.bar3d(xpos, ypos, zpos, dx, dy, dz, color=color)
    ax.view_init(elev=30., azimuth=5)
    ax.set_xlabel(labels[0])

```

```
ax.set_ylabel(labels[1])
ax.set_zlabel(labels[2])


fig = plt.figure(figsize=(15,7))

# 3D bar graf za z_mean
labels = ('Samples', 'Hidden elements', 'Z mean')
bargraph_vis(1, z_mu, [200, z_mu.shape[1]], 'g', labels)

# 3D bar graf za težine iz z_mena u dekođer
labels = ('Decoder elements', 'Hidden elements Z', 'Weights')
bargraph_vis(2, weights_d1.T, weights_d1.T.shape, 'y', labels)
```

Bonus zadatak - Tensorboard

Predložak za 4. zadatak sadrži kod za prikupljanje podataka koji se mogu prikazati pomoću [Tensorboard](#). Pokrenite Tensorboard i provjerite koje su informacije dostupne o treniranom VAE.

 dlunizg
ivan.kreso@fer.hr