Tim *Swiftly*:

- **Teodora Kocić [17190]**
- **Stefan Aleksić [16995]**

Varijacioni autokoderi

Veštačka Inteligencija

Sadržaj

1. UVOD

U skorijem vremenskom periodu mašinsko učenje je naučna oblast koja svojim izuzetno brzim napretkom i razvojem privlači pažnju sve većeg broja ljudi. Oslanjajući se na ogromne količine podataka, dobro projektovane mreže i arhitekturu, kao i različite tehnike za ubrzanje rešavanja problema, ova oblast uspeva da proizvede podatke izuzetnog kvaliteta, kao što su slike, audio, tekst, itd. Svakako jedan od modela koji je dosta korišćen u mašinskom učenju i ističe se po rezultatima koje generiše jeste **varijacioni autokoder** – VAE.

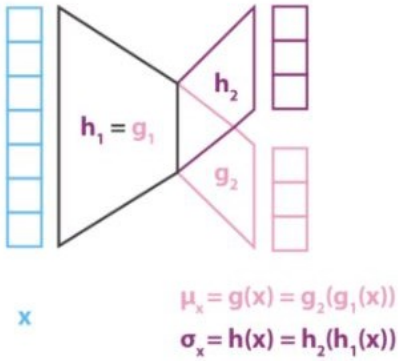
U osnovi, **varijacioni autokoder** koji za vreme *treniranja* podataka vrši kodiranje ulaznih podataka tako da latencija podataka, odnosno vreme koje protekne od momenta delovanja stimulusa do momenta kada nastane reakcija na stimulus, bude dovoljno dobra za generisanje potpuno novih podataka. Sam pridev u nazivu ovog autokodera je posledica bliske veze između regularizacije i metoda varijacione interference, koji je poznat i često korišćen u oblasti statistike. Danas se uglavnom kada govorimo o implementaciji VAE-a koriste neuronske mreže. Na samom kraju dokumenta biće opisano jedno od *open source* rešenja koje se može primeniti za kompletnu implementaciju **varijacionog autokodera** uz testiranje iste.

2. Pojam neuronskih mreža

Postavimo problem modela koji zavisi od tri funkcije f, g, h, a korišćenjem *varijacione interference*[1] generiše algoritam optimizacije koji bi trebalo da izgeneriše funkcije f^*, g^*, h^* koje bi bile optimalno rešenje koje se može primeniti na odabir parova kodiranje/dekodiranje (bitno za problem koji je glavna tema ovog dokumenta). Funkcije f, g, h predstavićemo kao *neuronske mreže*[2]. U praksi, funkcije g i h nisu dve potpuno nezavisne mreže, jer poseduju zajednički deo arhitekture i težine:

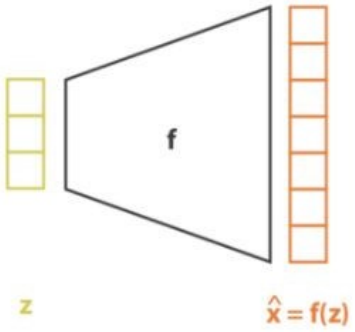
$$g(x) = g_2(g_1(x)) \qquad h(x) = h_2(h_1(x)) \quad g_1(x) = h_1(x)$$

Polazimo od pretpostavke da su *matrice kovarijanse*[3] multi-dimenziona Gausova (normalna) raspodela sa dijagonalnim matricama kovarijanse. Zbog ove aproksimacije sledi da vektor $h(x)$ sadrži elemente glavne dijagonale matrice kovarijanse i ujedno je iste dimenzije kao i vektor $g(x)$.



Slika 1 *Enkoder VAE-a*

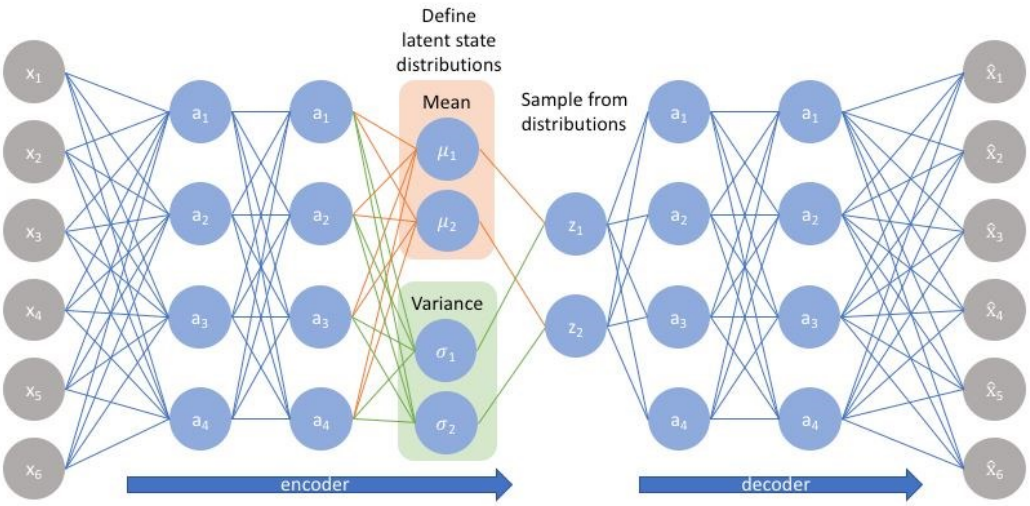
Za dekodiranje koristi se fiksna normalna raspodela. Takođe, za definisanje u kontekstu našeg problema, korišćena je neuronska mreža i dekoder, što se može predstaviti na sledeći način:



Slika 2 *Dekoder VAE-a*

[1] Tehnika kojom se vrši aproksimacija kompleksne distribucije.
[2] Sistem koji se sastoji od određenog broja međusobno povezanih procesora/čvorova, ili procesnih elemenata koje nazivamo veštačkim neuronima.
[3] Matrice koje prikazuju vezu između kovarijansi datih vektora. Kovarijansa predstavlja očekivanu vrednost koja zavisi od devijacije bilo koje dve komponente ispitivanih podataka, tj. vektora.

Korišćenjem određenih matematičkih alata moguće je minimizovati gubljenje podataka kroz mrežu, što će kasnije biti pokazano na konkretnom primeru za varijacioni autokoder. [1]



Slika 3 *Primena neuronskih mreža na problem VAE-a*

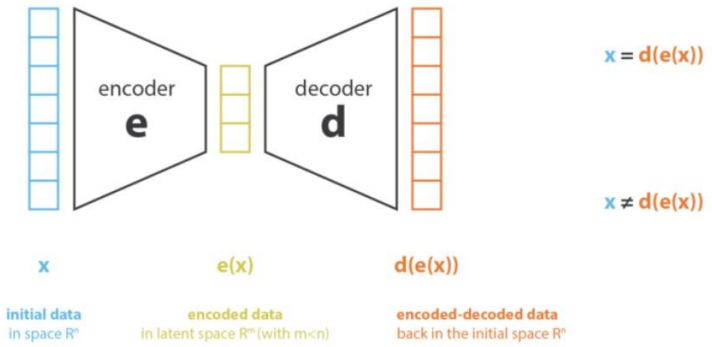
Na početku teksta ukratko je rečena suština varijacionog autokodera. U daljem tesktu biće reči o svim pojmovima koji su korišćeni prilikom definisanja varijacionog autokodera, takođe biće opisan način rada varijacionog autokodera gde će se videti upotreba opisanih neuronskih mreža. Obradićemo redukciju dimenzija i pojam autokodera uopšteno, kako bismo približili principe na kojima je zasnovan VAE. Dalje će biti jasno definisana razlika između (klasičnih) autokodera i varijacionih autokodera i nakon ovog dela trebalo bi da bude potpuno jasno zbog čega je toliko važan VAE.

3. Pojam redukcije dimenzija i *klasičnih* autokodera

U mašinskom učenju, pod pojmom **redukcija dimenzija** podrazumeva se proces kojim se smanjuje broj (količina) informacija koja se koristi za predstavljanje nekog podatka. Ovo se može postići na dva načina:

- Iz postojećeg skupa informacija biraju se određene,
- Na osnovu postojećeg skupa informacija se formira neki novi, obima manjeg od trenutnog.

Ovo nalazi veliku primenu kada je potrebno da se izvrši vizuelno predstavljanje nekih podataka (slike), kod zaštite informacija, kod izračunavanja ogromne količine podataka, itd.

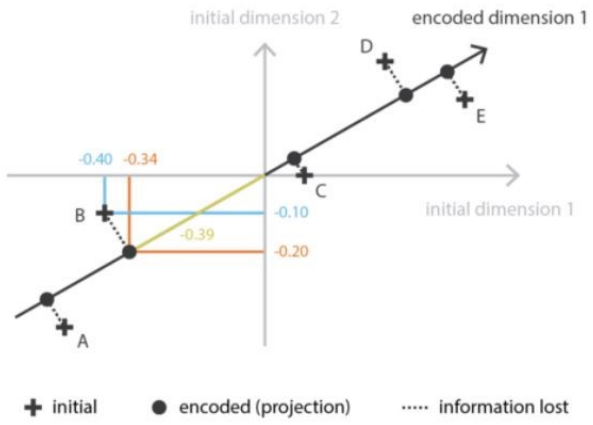


Slika 4 *Primer redukcije dimenzije primenom kodera i dekodera*

Proces kojim se dobijaju nove informacije na osnovu starih, na jedan oda dva prethodno navedena načina, naziva se **kodiranje,** a neuron koji vrši ovaj proces, **koder**. Proces suprotan ovom jeste **dekodiranje**, a neuron koji vrši ovaj proces **dekoder**. U skladu sa time uvodimo izraz **kompresije podataka**, koji zapravo predstavlja redukciju dimenzije. Te tako koder vrši kompresiju podataka (iz inicijalnog prostora u **kodirani prostor**, koji se još naziva i **prostor latencije**), dok dekoder vrši tzv. **dekompresiju podataka**. Prilikom ove kompresije podataka može, a vrlo često i dolazi do gubitka nekih informacija, koje primenom dekompresije ne mogu biti povraćene, što je prikazano na slici 4.

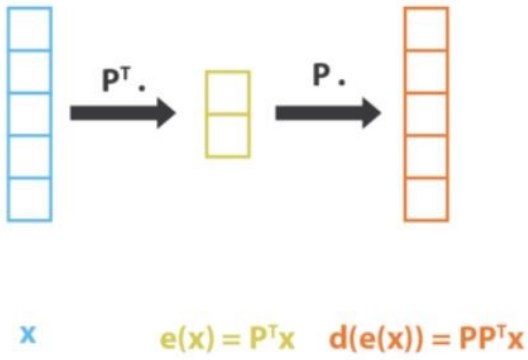
Vidimo da je prostor pre kodiranja, kao i nakon dekodiranja kodiranih podataka dimenzije n, međutim nakon kodiranja ovaj prostor je dimenzije m, za koju važi da je m manje od n, međutim može se desiti potencijalni gubitak informacija koji je nepovratan nakon faze dekodiranja (slučaj predstavljen drugom jednačinom za vektor x).

Glavni cilj koji treba biti zadovoljen procesom redukcije dimenzija jeste da se pronađe par koder/dekoder koji dovodi do minimalnog gubitka informacija u podatku. Dakle taj par treba da zadrži maksimalan broj informacija u podatku nakon kompresije podataka, a da prilikom dekompresije vrši minimalan broj grešaka rekonstrukcije. Danas se kao matematički alat za dokazivanje kompresije i dekompresije podataka koristi **princip komponentne analize**.



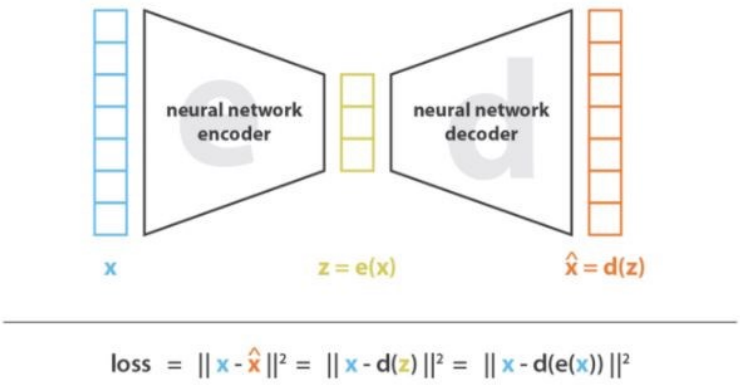
Slika 5 *Princip komponentne analize na primeru kodiranja i dekodiranja*

Ideja je generisanje nezavisnog prostora (prostor nakon kodiranja), koji se može predstaviti kao linearna kombinacija ulaznih informacija (prostor pre kodiranja), pri čemu je projekcija generisanih informacija što približnija početnim informacijama (gledano u Euklidskom prostoru). Vizuelna reprezentacija ove ideje data je na slikama 5 i 6.



Slika 6 *Princip komponentne analize na primeru kodiranja i dekodiranja*

U daljem tekstu govoriće se uopšteno o autokoderima i biće opisan način na koji se koriste **neuronske mreže** u svrhu opisivanja redukcije dimenzije i implementacije autokodera. Autokoder implementira koder i dekoder kao neuronske mreže i uči kako izabrati najbolji par za kodiranje/dekodiranje prolaskom kroz brojne iteracije, gde se koriste i odgovarajući algoritmi za optimizaciju. U svakoj iteraciji, arhitektura autokodera (koder + dekoder) dobija određene informacije koje dalje poredi sa inicijalnim informacijama i vrši propagaciju greške (naučeno iz pristiglih i postojećih informacija) kroz čitavu arhitekturu kako bi se izvršilo ažuriranje težine mreža. [2]

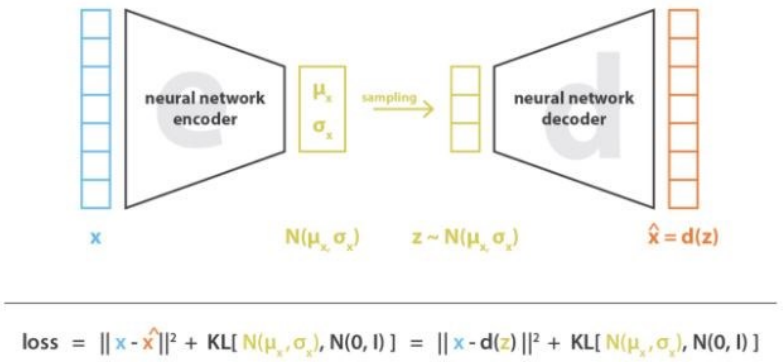


Slika 7 *Ilustracija autokodera i njegova funkcija gubitka*

Kako su i koder i dekoder funkcije nelinearne, sama arhitektura autokodera je komplikovana i potrebno je primeniti visok stepen redukcije dimenzija sa minimalnim gubicima u pogledu informacija. Treba imati na umu da ovo dovodi do smanjenja prostora latencije, ali i povećanje „dubine“ autokodera. Suštinski autokoder vrši smanjivanje broja dimenzija, tj. kao izlaz dobijamo kompresovane podatke.

4. Definicija varijacionog autokodera

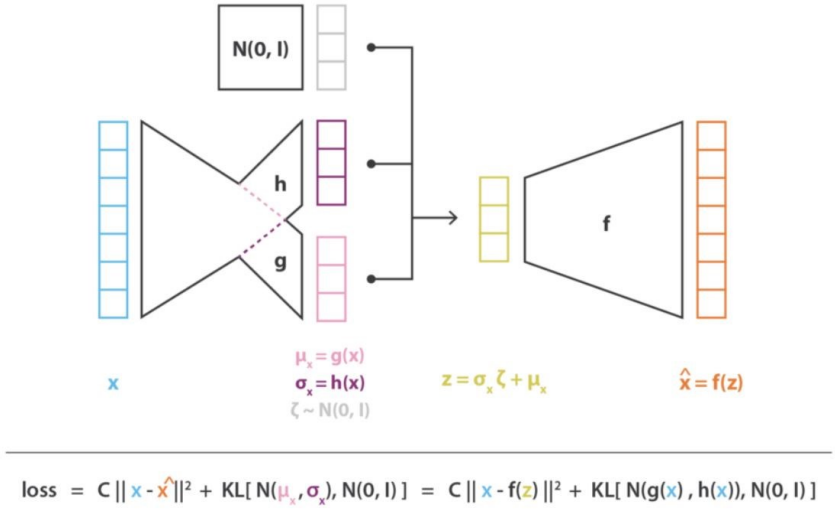
Varijacioni autokoder, za razliku od autokodera na osnovu naučenih informacija generiše nove podatke. VAE predstavlja autokoder koji korišćenjem regularizacije obezbeđuje zadovoljavajući prostor latencije, čija su svojstva takva da dozvoljavaju proces generisanja novih informacija. Veoma sličan autokoderu, VAE poseduje koder i dekoder koji smanjuju pojavu grešaka između kodiranih/dekodiranih podataka i inicijalnih podataka. Razlika između VAE-a i klasičnog autokodera je u tome što VAE umesto kodiranja jedinstvenih podataka vrši distribuciju kodiranih podataka na prostor latencije.



Slika 8 *VAE i njegova funkcija gubitka (povećava stepen efikasnosti kodiranja i prostor latencije)*

VAE dakle ne proizvodi podatke koji su kompresovani, već nakon kodiranja podataka kao rezultat dobija se raspodela iz koje bi bili generisani potpuno novi podaci. VAE potom kodirane podatke koji su dobijeni u okviru neke proizvoljne raspodele treba da skalira na normalnu (Gausovu) raspodelu. Pored distribucije kodiranih podataka pridodaje se još jedna komponenta (vektor) koji pripada ovoj normalnoj raspodeli. Izvlačenjem nasumičnog uzorka iz ovog vektora i kombinovanjem tog uzorka sa već postojećim, kodiranim podacima generiše se uzorak koji bi trebalo da simulira raspodelu koja nije normalizovana [4]. U terminologiji VAE-a prethodno opisani postupak predstavlja tzv. **trik reparametrizacije** i to je jedan od najbitnijih koraka u radu samog varijacionog autokodera. Matematički dokaz ovoga leži u primeni Monte-Karlovog metoda, međutim ovo neće biti detaljnije obrađivano jer nije predmet ovog projekta. Na slici 8 data je vizuelizacija načina kodiranja VAE-a. [2]

[4] Raspodela u kojoj su se prvobitno nalazili kodirani podaci



Slika 9 *Reprezentacija VAE-a*

5. Jedan od načina implementacije VAE-a

Razmotrićemo jedno od potencijalnih rešenja za implementaciju varijacionog autokodera i na praktičnom primeru objasniti implementaciju tehnika i procesa o kojima je bilo diskutovano u prethodnom tekstu. U nastavku dat je kod implementacije. [3]

```
import tensorflow as tf
import numpy as np

## networks - convolutional variational autoencoder
class CVAE(tf.keras.Model):

  def __init__(self, latent_dim):
    super(CVAE, self).__init__()
    self.latent_dim = latent_dim
    self.encoder = tf.keras.Sequential(
        [
            tf.keras.layers.InputLayer(input_shape=(28, 28, 1)),
            tf.keras.layers.Conv2D(
                filters=32, kernel_size=3, strides=(2, 2), activation='relu'),
            tf.keras.layers.Conv2D(
                filters=64, kernel_size=3, strides=(2, 2), activation='relu'),
            tf.keras.layers.Flatten(),
            # No activation
            tf.keras.layers.Dense(latent_dim + latent_dim),
        ]
    )


    self.decoder = tf.keras.Sequential(
        [
            tf.keras.layers.InputLayer(input_shape=(latent_dim,)),
            tf.keras.layers.Dense(units=7*7*32, activation=tf.nn.relu),
            tf.keras.layers.Reshape(target_shape=(7, 7, 32)),
            tf.keras.layers.Conv2DTranspose(
                filters=64, kernel_size=3, strides=2, padding='same',
                activation='relu'),
            tf.keras.layers.Conv2DTranspose(
                filters=32, kernel_size=3, strides=2, padding='same',
                activation='relu'),
            # No activation
            tf.keras.layers.Conv2DTranspose(
                filters=1, kernel_size=3, strides=1, padding='same'),
        ]
    )


  @tf.function
  def sample(self, eps=None):
    if eps is None:
      eps = tf.random.normal(shape=(100, self.latent_dim))
    return self.decode(eps, apply_sigmoid=True)


  def encode(self, x):
    mean, logvar = tf.split(self.encoder(x), num_or_size_splits=2, axis=1)
    return mean, logvar


  def reparameterize(self, mean, logvar):
    eps = tf.random.normal(shape=mean.shape)
    return eps * tf.exp(logvar * .5) + mean
```

```
  def decode(self, z, apply_sigmoid=False):
    logits = self.decoder(z)
    if apply_sigmoid:
      probs = tf.sigmoid(logits)
      return probs
    return logits


latent_dim = 8 # set the dimensionality of the latent space to a plane (2) for visualization
model = CVAE(latent_dim)


## loss and optimiser
optimizer = tf.keras.optimizers.Adam(1e-4)

def log_normal_pdf(sample, mean, logvar, raxis=1):
  log2pi = tf.math.log(2. * np.pi)
  return tf.reduce_sum(
      -.5 * ((sample - mean) ** 2. * tf.exp(-logvar) + logvar + log2pi),
      axis=raxis)

def compute_loss(model, x): #evidence lower bound
  mean, logvar = model.encode(x)
  z = model.reparameterize(mean, logvar)
  x_logit = model.decode(z)
  cross_ent = tf.nn.sigmoid_cross_entropy_with_logits(logits=x_logit, labels=x)
  logpx_z = -tf.reduce_sum(cross_ent, axis=[1, 2, 3])
  logpz = log_normal_pdf(z, 0., 0.)
  logqz_x = log_normal_pdf(z, mean, logvar)
  return -tf.reduce_mean(logpx_z + logpz - logqz_x)




## training
# @tf.function
def train_step(model, x, optimizer):
  with tf.GradientTape() as tape:
    loss = compute_loss(model, x)
  gradients = tape.gradient(loss, model.trainable_variables)
  optimizer.apply_gradients(zip(gradients, model.trainable_variables))




epochs = 10
for epoch in range(1, epochs + 1):
  for train_x in train_dataset:
    train_step(model, train_x, optimizer)


  # test loss
  elbo_test = [-compute_loss(model, test_x).numpy() for test_x in test_dataset]
  print('Epoch: {}, Test ELBO: {:0.5f}'.format(epoch, sum(elbo_test)/len(elbo_test)))


  # test images
  mean, logvar = model.encode(test_sample)
  z = model.reparameterize(mean, logvar)
  predictions = model.sample(z)
  image_grid = tf.concat([tf.concat(tf.unstack(predictions,8*8)[i*8:(i+1)*8],0) for i in range(8)],1)
  Image.fromarray((tf.squeeze(image_grid)*255).numpy().astype('uint8')).save('test_samples_e{:02d}.jpg'.fo
rmat(epoch))


print('Training done.')
```

Algoritmom se vrši generisanje novih slika na osnovu nekih ulaznih podataka koji se pribavljaju iz specifične baze podataka.
Klasa koja se koristi za kreiranje VAE-a jeste klasa CVAE u datom algoritmu, koja najpre postavlja dimenzije prostora latencije i uzima se uglavnom neka mala vrednost (kasnije kod test primera koristi se vrednost 8).

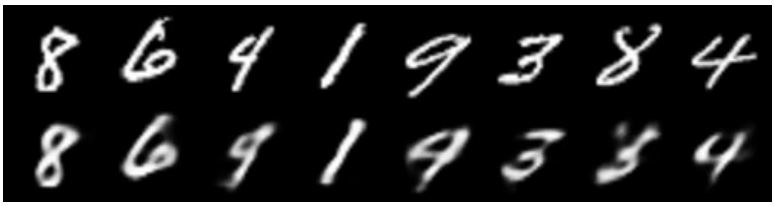
- Implementacija kodera:
  Metodom *InputLayers(…)* učitava se slika dimenzije 28x28 sa samo jednim kanalom (tj. Bitovi mogu biti 0 ili 1, tako da je reč o crno-beloj slici). Metoda *Conv2D(…)* postavlja filtere, tj. povećava broj kanala, menja i u ovom slučaju smanjuje dimenziju slike za pola, zbog postavljenog atributa *strides* na vrednost 2. Kako je reč o varijacionom autokoderu, objašnjeno je ranije kako zapravo on radi i zbog distribucije kodiranih podataka imamo veću dubinu prostora latencije, pa za posledicu ovoga kao argument metode *Dense(…)* prosleđuje se vrednost (2 * dimenzija latentnog prostora), dok bi u slučaju autokodera bila prosleđena vrednost (1 * dimenzija latentnog prostora).

- Implementacija dekodera:
  Proces obrade podataka ide u smeru suprotnom od obrade u okviru kodera. Pa su i sve metode korišćene ovde upravo inverzne odgovarajućim metodama korišćenim u implementaciji kodera.

  Ono što je veoma bitno napomenuti je činjenica da sve do sada obrađene metode menjaju uglavnom na neki način dimenzije ulaznih podataka, međutim u toku izvršavanja svake od metoda naša mreža uči, ništa nije fiksno, odnosno težine kod neuronskih mreža su promenljive, tako da mreže neprestano konvergiraju.

- Implementacija načina rada VAE-a:
  Najpre metodom *sample(…)* izvlači nasumičan uzorak iz normalne raspodele (usko vezano sa kodiranim podacima). Metodama *encode(…)* i *decode(…)* je omogućeno pokretanje kodera i dekodera. Metoda *reparameterize(…)* predstavlja implementaciju trika reparametrizacije, koji smo opisali. Metoda *log_normal_pdf(…)* definiše normalnu (Gausovu) raspodelu. U okviru metode *compute_loss(…)* vrši se preračunavanje funkcije gubitka informacija, u promenljivoj z smešten je vektor koji zapravo predstavlja uzorak dobijen kao posledica reparametrizacije, dok se za smeštanje vektora konačne slike koristi promenljiva *x_logit*.

- Treniranje mreže i testiranje
  Metoda *train_step(…)* u svakoj iteraciji računa gradijente na osnovu novih ulaznih informacija i nad takvim podacima odradi optimizaciju. Treniranje se vrši u ovom slučaju za 10 epoha. U okviru svake epohe pokrene se treniranje mreže.
  elbo_test(…) nakon svake završene epohe računa gubitak koji se u njoj desio i ove informacije su od izuzetne važnosti za proces generalizacije. Na kraju se vrši testiranje mreže pozivom pojedinih metoda klase CVAE. Slike se smeštaju u grid, čije se kreiranje vrši pre kreiranja same klase CVAE, međutim to nije priloženo u kodu jer akcenat je na implementaciji varijacionog autokodera.



Slika 10 *Ulaz i izlaz Varijacionog autokodera*

Objašnjeni algoritam je napisan u Python-u. Biblioteke koje omogućavaju realizaciju i pokretanje algoritma su *tensorflow*, koja sa sobom povlači i biblioteku *numpy*.

Što se tiče instalacije *tensorflow* biblioteke alatom *pip* (Package Installer For Python), preporučujemo da pratite uputstvo na zvaničnom sajtu. Ukoliko verzija Python-a koju imate instaliranu nije u opsegu verzija [3.7 – 3.9], moraćete skinuti neku iz opsega, a onda uneti u konzoli

$$py\ -X\ -m\ pip\ install\ --upgrade\ tensorflow$$

Gde je X novoinstalirana verzija Python-a.

6. Reference

[1] J. Jordan, "Variational autoencoders," March 2018. [Online]. Available: https://www.jeremyjordan.me/variational-autoencoders/.

[2] J. Rocca, "Understanding Variational Autoencoders [VAEs]," September 2019. [Online]. Available: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73?gi=200b24599fd9.

[3] U. T. College, "Convolutional Variational Autoencoder," January 2022. [Online]. Available: https://www.tensorflow.org/tutorials/generative/cvae.