|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Калужский филиал федерального государственного автономного образовательного учреждения высшего образования**  **«Московский государственный технический университет  имени Н.Э. Баумана  (национальный исследовательский университет)»**  **(КФ МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ ИУК «Информатика и управление»\_

КАФЕДРА ИУК5 «Системы обработки информации»\_

**ЛАБОРАТОРНАЯ РАБОТА №2.1**

**«НЕЙРОННАЯ ПЕРЕДАЧА СТИЛЯ. ГЕНЕРАТИВНОСОСТЯЗАТЕЛЬНЫЕ СЕТИ»**

**по дисциплине: «Методы глубокого обучения»**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Выполнил: студент группы ИУК5-21М | |  |  | А. Э. Дармограй | |
|  | | (Подпись) |  | (И.О. Фамилия) | |
| Проверил: | |  |  | Ю. С. Белов | |
|  | | (Подпись) |  | (И.О. Фамилия) | |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | | | | |
|  | - Балльная оценка:  - Оценка: | | | |

Калуга, 2025

**Цель:** получение практических навыков построения вариационных автокодировщиков.

**Задачи:**

* Изучить модель глубокого обучения VAE.
* Разработать модель вариационного автокодировщика.

**Результатами работы являются:**

* Разработанная модель вариационного автокодировщика
* Подготовленный отчет

**Задание**: разработать модель вариационного автокодировщика.

**Выполнение работы**

Код доступен в репозитории GitHub:

<https://github.com/Dariarty/Deep_Learning_Methods>

Данную лабораторную работы выполнял на Python версии 3.7.9 и Tensorflow версии 1.15

Код лабораторной работы №2.1:

<https://github.com/Dariarty/Deep_Learning_Methods/blob/main/src/LAB_2_1/vae.ipynb>

В данной работе разработана модель вариационного автокодировщика

|  |
| --- |
| *#В данной работе использую Python 3.7.9 и Tensorflow 1.15*  **import** sys  **import** tensorflow **as** tf  **from** tensorflow **import** keras  **import** os  os**.**environ['TF\_CPP\_MIN\_LOG\_LEVEL'] **=** '3'  *# Вывод версий Python и Tensorflow*  print("Python", sys**.**version)  print("Tensorflow", tf**.**\_\_version\_\_) |

Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 18:58:18) [MSC v.1900 64 bit (AMD64)]

Tensorflow 1.15.0

Импорт библиотек и загрузка датасета MNIST

|  |
| --- |
| **import** numpy **as** np  **from** tensorflow.examples.tutorials.mnist **import** input\_data  **import** matplotlib.pyplot **as** plt  np**.**random**.**seed(42)  tf**.**set\_random\_seed(42)  **%matplotlib** inline  mnist **=** input\_data**.**read\_data\_sets('MNIST\_data')  input\_size **=** 28 **\*** 28  num\_digits **=** 10 |

Extracting MNIST\_data\train-images-idx3-ubyte.gz

Extracting MNIST\_data\train-labels-idx1-ubyte.gz

Extracting MNIST\_data\t10k-images-idx3-ubyte.gz

Extracting MNIST\_data\t10k-labels-idx1-ubyte.gz

Определение гиперпараметров модели

Настраиваются параметры обучения: количество эпох, размер скрытого пространства, структура слоёв энкодера, декодера и классификатора, вес функции потерь классификации, скорость обучения и т.д.

|  |
| --- |
| params **=** {  *# кодировщик на простой сети прямого распространения*  'encoder\_layers': [128],  *# как и декодер (CNN лучше, но не хочу усложнять код)*  'decoder\_layers': [128],  *# нужно для условий, разобранных далее*  'digit\_classification\_layers': [128],  *# функция активации используется всеми подсетями*  'activation': tf**.**nn**.**sigmoid,  *# стандартное отклонение P(x|z) обсуждалось выше*  'decoder\_std': 0.5,  *# размерность скрытого пространства*  'z\_dim': 10,  *# нужно для условий, разобранных далее*  'digit\_classification\_weight': 10.0,  'epochs': 10,  'batch\_size': 100,  'learning\_rate': 0.001  } |

Определение архитектуры VAE

Реализуются три функции:

* encoder — кодирует изображение в скрытое пространство;
* decoder — восстанавливает изображение из скрытого вектора;
* digit\_classifier — классифицирует изображение по цифре

|  |
| --- |
| **def** encoder(x, layers):  **for** layer **in** layers:  x **=** tf**.**layers**.**dense(x, layer, activation**=**params['activation'])  mu **=** tf**.**layers**.**dense(x, params['z\_dim'])  var **=** 1e-5 **+** tf**.**exp(tf**.**layers**.**dense(x, params['z\_dim']))  **return** mu, var  **def** decoder(z, layers):  **for** layer **in** layers:  z **=** tf**.**layers**.**dense(z, layer, activation**=**params['activation'])  mu **=** tf**.**layers**.**dense(z, input\_size)  **return** tf**.**nn**.**sigmoid(mu)  **def** digit\_classifier(x, layers):  **for** layer **in** layers:  x **=** tf**.**layers**.**dense(x, layer, activation**=**params['activation'])  logits **=** tf**.**layers**.**dense(x, num\_digits)  **return** logits  images **=** tf**.**placeholder(tf**.**float32, [**None**, input\_size])  digits **=** tf**.**placeholder(tf**.**int32, [**None**])  *# кодируем изображение в распределение по скрытому пространству*  encoder\_mu, encoder\_var **=** encoder(images,  params['encoder\_layers'])  *# отбираем вектор из скрытого пространства, используя трюк с повторной параметризацией*  eps **=** tf**.**random\_normal(shape**=**[tf**.**shape(images)[0], params['z\_dim']],  mean**=**0.0, stddev**=**1.0)  z **=** encoder\_mu **+** tf**.**sqrt(encoder\_var) **\*** eps  *# Классифицируем цифры*  digit\_logits **=** digit\_classifier(images,  params['digit\_classification\_layers'])  digit\_prob **=** tf**.**nn**.**softmax(digit\_logits)  *# Декодируем в изображение скрытый вектор, связанный с классификацией цифр*  decoded\_images **=** decoder(tf**.**concat([z, digit\_prob], axis**=**1),  params['decoder\_layers'])  *# потеря состоит в том, насколько хорошо можем восстановить изображение*  loss\_reconstruction **=** **-**tf**.**reduce\_sum(  tf**.**contrib**.**distributions**.**Normal(decoded\_images,  params['decoder\_std'])**.**log\_prob(images),axis**=**1)  *# и как далеко распределение по скрытому пространству*  *# от предыдущего. Если предыдущее является*  *# стандартным гауссовским распределением, а*  *# в результате получается нормальное распределение*  *# с диагональной конвариантной матрицей, то*  *# KL-расхождение становится аналитически разрешимым,и получаем*  loss\_prior **=** **-**0.5 **\*** tf**.**reduce\_sum(1 **+** tf**.**log(encoder\_var) **-**  encoder\_mu **\*\*** 2 **-** encoder\_var, axis**=**1)  loss\_auto\_encode **=** tf**.**reduce\_mean(loss\_reconstruction **+**  loss\_prior,axis**=**0)  loss\_digit\_classifier **=** params['digit\_classification\_weight'] **\*** tf**.**reduce\_mean(tf**.**nn**.**sparse\_softmax\_cross\_entropy\_with\_logits(labels**=**digits, logits**=**digit\_logits), axis**=**0)  loss **=** loss\_auto\_encode **+** loss\_digit\_classifier  train\_op **=** tf**.**train**.**AdamOptimizer(params['learning\_rate'])**.**minimize(loss) |

Обучение модели

Обучим модель оптимизации двух функций потерь — VAE и классификации — с помощью SGD. На каждом шаге обучается модель, затем считается ошибка на всём наборе и сохраняются изображения, сгенерированные моделью на основе скрытых векторов и меток

|  |
| --- |
| print("Количество образов в MNIST:", mnist**.**train**.**num\_examples)  print("Размер батча:", params['batch\_size'])  print("Обучение в течение эпох:", params['epochs'])  samples **=** []  losses\_auto\_encode **=** []  losses\_digit\_classifier **=** []  sess **=** tf**.**Session(config**=**tf**.**ConfigProto(log\_device\_placement**=True**))  sess**.**run(tf**.**global\_variables\_initializer())  **for** epoch **in** range(params['epochs']):  **for** \_ **in** range(int(mnist**.**train**.**num\_examples **/** params['batch\_size'])):  batch\_images, batch\_digits **=** mnist**.**train**.**next\_batch(params['batch\_size'])  sess**.**run(train\_op, feed\_dict**=**{images: batch\_images, digits: batch\_digits})  train\_loss\_auto\_encode, train\_loss\_digit\_classifier **=** sess**.**run([loss\_auto\_encode, loss\_digit\_classifier],  {images: mnist**.**train**.**images, digits: mnist**.**train**.**labels})  losses\_auto\_encode**.**append(train\_loss\_auto\_encode)  losses\_digit\_classifier**.**append(train\_loss\_digit\_classifier)  sample\_z **=** np**.**tile(np**.**random**.**randn(1, params['z\_dim']), reps**=**[num\_digits, 1])  gen\_samples **=** sess**.**run(decoded\_images, feed\_dict**=**{z: sample\_z, digit\_prob: np**.**eye(num\_digits)})  samples**.**append(gen\_samples)  print("Обучение завершено") |

Количество образов в MNIST: 55000

Размер батча: 100

Обучение в течение эпох: 10

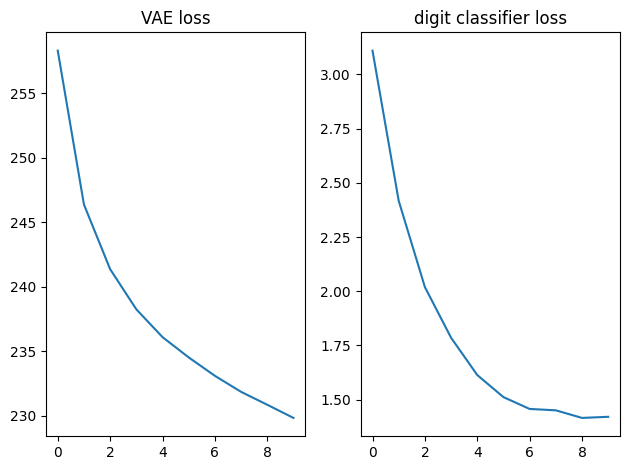
Device mapping:

Обучение завершено

Визуализация графиков потерь

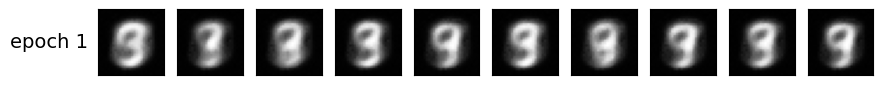
|  |
| --- |
| plt**.**subplot(121)  plt**.**plot(losses\_auto\_encode)  plt**.**title('VAE loss')  plt**.**subplot(122)  plt**.**plot(losses\_digit\_classifier)  plt**.**title('digit classifier loss')  plt**.**tight\_layout() |

Как видим, обе функции потерь уменьшаются.



Визуализация сгенерированных изображений

|  |
| --- |
| **def** plot\_samples(samples, epoch):  IMAGE\_WIDTH **=** 1  plt**.**figure(figsize**=**(IMAGE\_WIDTH **\*** num\_digits **\*** 1, IMAGE\_WIDTH **\*** num\_digits))  **for** digit, image **in** enumerate(samples[epoch]):  plt**.**subplot(1, num\_digits, digit **+** 1)  plt**.**imshow(image**.**reshape((28, 28)), cmap**=**'Greys\_r')  plt**.**gca()**.**xaxis**.**set\_visible(**False**)  **if** digit **==** 0:  plt**.**gca()**.**yaxis**.**set\_ticks([])  plt**.**ylabel('epoch {}'**.**format(epoch **+** 1),  verticalalignment**=**'center',  horizontalalignment**=**'right',  rotation**=**0,  fontsize**=**14)  **else**:  plt**.**gca()**.**yaxis**.**set\_visible(**False**)    plot\_samples(samples, 0)  plot\_samples(samples, len(samples)**-**1) |





Как видно по сгенерированным изображениям, к 10 эпохе есть четкие очертания цифр.

**Вывод:** в ходе выполнения лабораторной работы были сформированы практические навыки по построению вариационных автокодировщиков. Были изучена модель глубокого обучения VAE, а также разработана модель вариационного автокодировщика