ЛЕКЦИЯ 7

Изучение выявления Man in the Middle атак нейронными сетями

КЛАССИФИКАЦИЯ

Нейронные сети:

- Глубокие нейронные сети (DNN)
- Сверточные нейронные сети (CNN)
- Рекуррентные нейронные сети (RNN).
- Нейронная сеть Bidirectional Encoder Representations from Transformers (BERT)

ЭТАПЫ ВЫЯВЛЕНИЯ MITM-АТАК С ПОМОЩЬЮ НЕЙРОННЫХ СЕТЕЙ

Сбор данных

Для эффективного выявления атак MitM необходимо собрать сетевые данные, содержащие как **обычный трафик**, так и **трафик, характерный для MitM-атак**.

Источники данных:

- **Лог-файлы сетевых устройств** (роутеры, коммутаторы, серверы)
- **Пакеты трафика** (Wireshark, Tcpdump)
- **NetFlow и sFlow статистика**

Открытые датасеты (CICIDS2017, NSL-KDD, UNSW-NB15)

Ключевые признаки аномального поведения:

- **Несоответствие MAC-адресов и IP-адресов** (отравление ARP)
- **Высокая задержка в ответах** (proxy-перехват)
- **Изменения в SSL-сертификатах** (атакующий использует свой сертификат)
- **Необычные DNS-запросы** (перенаправление трафика)
- **Резкое увеличение трафика** на нестандартных портах

ЭТАПЫ ВЫЯВЛЕНИЯ MITM-АТАК С ПОМОЩЬЮ НЕЙРОННЫХ СЕТЕЙ

```
MitM
```

	0	1	2	3	4	5	6	7	8	9	...	10
0	1.000000	1294.000000	0.000000e+00	1.000000	1294.000000	0.000000e+00	1.000000	1294.000000	0.000000e+00	1.000000	...	0.000000e+0
1	1.000000	1514.000000	0.000000e+00	1.000000	1514.000000	0.000000e+00	1.000000	1514.000000	0.000000e+00	1.000000	...	0.000000e+0
2	1.999505	1294.000000	6.984919e-10	1.999703	1294.000000	2.328306e-10	1.999901	1294.000000	6.984919e-10	1.999990	...	0.000000e+0
3	2.998985	1294.000000	9.313226e-10	2.999391	1294.000000	4.656613e-10	2.999797	1294.000000	6.984919e-10	2.999980	...	6.984919e-1
4	3.998061	1294.000000	9.313226e-10	3.998836	1294.000000	2.328306e-10	3.999612	1294.000000	6.984919e-10	3.999961	...	2.328306e-1
...	...	...	...	...	...	...	...	...	...	...	...	
2504262	380.551133	1331.684771	1.895906e+05	650.148549	1336.357995	1.865743e+05	1973.033776	1339.428666	1.851917e+05	19664.853626	...	1.813631e+0
2504263	379.629067	1332.165017	1.891785e+05	649.176314	1336.631637	1.863354e+05	1972.036680	1339.517190	1.851133e+05	19663.862251	...	1.813090e+0
2504264	380.276647	1332.643182	1.887678e+05	649.814659	1336.904590	1.860970e+05	1972.670406	1339.605640	1.850348e+05	19664.496996	...	1.812549e+0
2504265	379.354717	1333.121248	1.883566e+05	648.842152	1337.177530	1.858584e+05	1971.672375	1339.694090	1.849564e+05	19663.504357	...	1.812009e+0
2504266	379.951197	1333.597306	1.879468e+05	649.427959	1337.449804	1.856203e+05	1972.252742	1339.782469	1.848780e+05	19664.085817	...	1.811468e+0

2504267 rows × 115 columns

ЭТАПЫ ВЫЯВЛЕНИЯ MITM-АТАК С ПОМОЩЬЮ НЕЙРОННЫХ СЕТЕЙ

Предобработка данных

- Сетевые данные, полученные на предыдущем этапе, должны быть очищены и приведены в удобный формат для машинного обучения.

Действия:

- **Удаление дубликатов** и неинформативных записей
- **Очистка данных** (устранение отсутствующих значений, обработка выбросов)
- **Приведение категориальных данных** к числовому виду (например, с помощью One-Hot Encoding)
- **Нормализация и стандартизация** (Min-Max Scaling, Standard Scaling)

ЭТАПЫ ВЫЯВЛЕНИЯ MITM-АТАК С ПОМОЩЬЮ НЕЙРОННЫХ СЕТЕЙ

```
scaler.fit(MitM)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:1688: FutureWarning: Feature names only support names th
at are all strings. Got feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
  warnings.warn(
```

```
MinMaxScaler()
```

```
MitM_scaled = pd.DataFrame(scaler.transform(MitM))
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:1688: FutureWarning: Feature names only support names th
at are all strings. Got feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
  warnings.warn(
```

```
with open('MitM_scaler.pkl', 'wb') as file:
    pickle.dump(scaler, file)
```

```
MitM_scaled
```

	0	1	2	3	4	5	6	7	8	9	...	106	107	108	109	1
0	0.680594	0.811159	0.884536	0.368931	0.813469	0.880903	0.372045	0.586947	0.878381	0.371202	...	0.426234	0.954144	0.846794	0.904155	0.8812
1	0.618364	0.593748	0.874648	0.369892	0.710273	0.877792	0.364657	0.848117	0.878255	0.368929	...	0.429657	0.954144	0.846794	0.902173	0.8811
2	0.149055	0.761982	0.878075	0.376165	0.840070	0.879933	0.367722	0.938136	0.880932	0.362809	...	0.426239	0.954144	0.846794	0.647965	0.8807
3	0.045775	0.648012	0.876872	0.372446	0.761011	0.876963	0.374175	0.911456	0.877625	0.375386	...	0.425237	0.954144	0.846794	0.290110	0.8815
4	0.706189	0.466479	0.860038	0.261018	0.448934	0.855961	0.270552	0.442087	0.851410	0.281942	...	0.424160	0.954144	0.846794	0.909432	0.8773
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2504262	0.023012	0.580222	0.857112	0.428156	0.694649	0.864136	0.409059	0.872832	0.873982	0.383464	...	0.426494	0.954144	0.846794	0.159904	0.8814
2504263	0.259131	0.302684	0.843414	0.321658	0.349847	0.846513	0.305853	0.409280	0.848726	0.291985	...	0.430383	0.954144	0.846794	0.808057	0.8768
2504264	0.889435	0.705402	0.889849	0.331843	0.780692	0.888373	0.335660	0.910191	0.885616	0.346850	...	0.420949	0.954144	0.846794	0.906788	0.8811
2504265	0.658944	0.727583	0.881733	0.356368	0.796838	0.877444	0.372796	0.915024	0.876662	0.378450	...	0.428650	0.954144	0.846794	0.903414	0.8813
2504266	0.877150	0.651064	0.866764	0.396554	0.747148	0.870116	0.390237	0.896983	0.875249	0.379784	...	0.423274	0.954144	0.846794	0.906455	0.8811

2504267 rows × 116 columns

ЭТАПЫ ВЫЯВЛЕНИЯ MITM-АТАК С ПОМОЩЬЮ НЕЙРОННЫХ СЕТЕЙ

Chi_square feature selection ¶

```
bestfeatures = SelectKBest(score_func=chi2, k=20)
```

```
fit_feat = bestfeatures.fit(MitM_scaled,MitM_labels)
```

```
MitM_scores = pd.DataFrame(fit_feat.scores_)
MitM_columns = pd.DataFrame(MitM_scaled.columns)
```

```
featureScores = pd.concat([MitM_columns,MitM_scores],axis=1)
featureScores.columns = ['Specs','Score']
```

```
print(featureScores.nlargest(20, 'Score'))
```

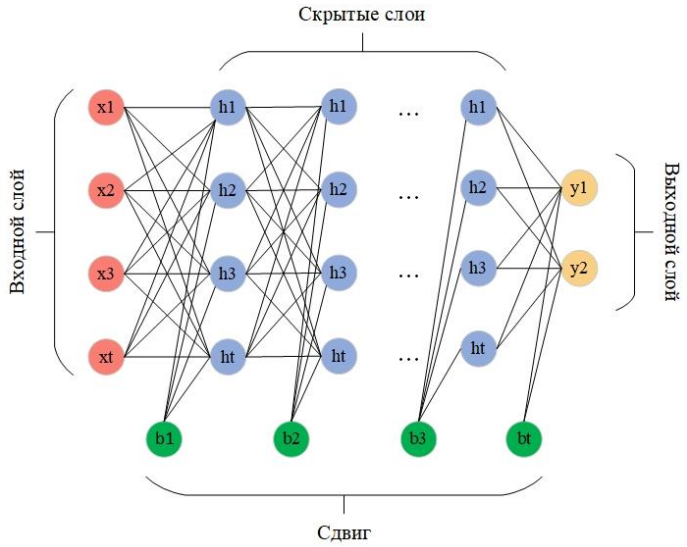
```
     Specs          Score
0        0  284570.859509
59      59   50057.824037
78      78   50057.824037
28      28   49969.049551
109    109   43181.048574
13      13   37499.951751
64      64   29401.928838
52      52    5751.688007
75      75    5751.688007
25      25    5723.137072
57      57    5114.790811
54      54    4722.101536
47      47    4672.799643
61      61    4633.207703
110    110    4563.681203
60      60    4563.208046
29      29    4553.725155
14      14    4553.498041
103    103    4543.808886
46      46    4543.692973
```

ЭТАПЫ ВЫЯВЛЕНИЯ MITM-АТАК С ПОМОЩЬЮ НЕЙРОННЫХ СЕТЕЙ

- **Разделение данных на обучающую и тестовую выборки**
- Данные разделяются следующим образом:
- **Обучающая выборка (Train Set)**: 70-80% данных
- **Тестовая выборка (Test Set)**: 20-30% данных
- Дополнительно можно использовать **кросс-валидацию (k-fold cross-validation)** для более точной оценки моделей.

ГЛУБОКИЕ НЕЙРОННЫЕ СЕТИ

- **Глубокие нейронные сети (DNN)** представляют собой модель нейронных сетей с двумя и более скрытыми слоями. Нейронная сеть состоит из входного слоя, содержащего входные данные, скрытых слоев, включающих узлы, называемые нейронами, и выходного слоя, содержащего один или несколько нейронов



ГЛУБОКИЕ НЕЙРОННЫЕ СЕТИ

- При этом $x = x_1, x_2, ..., x_f$ является входным вектором, $w_1, w_2, ..., w_i$ веса соединения каждого уровня,
- $b_1, b_2, ..., b_i$ - вектор смещения. Уровни от l_2 до l_{n-1} образуют скрытые слои, $y_1, y_2, ..., y_m$ является выходным вектором.
- Элементы скрытых и выходных слоев называются нейронами. Они представлены функциями активации, отвечающими за нелинейное функциональное отображение между входными данными и переменной отклика. Самыми популярными функциями активации являются сигмоидная функция, функция гиперболического тангенса (*tanh*), выпрямленная линейная единица (*ReLu*) и *softmax*. Сигмоидная функция в основном используется на выходном слое в бинарной классификации, так как определяет выходное значение как 0 или 1. Функция *tanh* – это улучшенная версия сигмоидной функции с разницей лишь в том, что в *tanh* выходные значения находятся в диапазоне от -1 до 1. В скрытых слоях чаще всего применяется функция активации *ReLu*. Это приводит к выходному значению 0, если он получает отрицательный вход *x*, иначе для положительных входов он возвращает *x* без изменений, подобно линейной функции.
- Функция *softmax* применяется в многоклассовой классификации, вычисляя вероятность того, что каждое вхождение принадлежит к заранее определенному классу, и корректирует выходные значения для каждого класса так, чтобы они находились в диапазоне от 0 до 1. Функция *softmax* обычно используется только для выходного слоя.

ГЛУБОКИЕ НЕЙРОННЫЕ СЕТИ

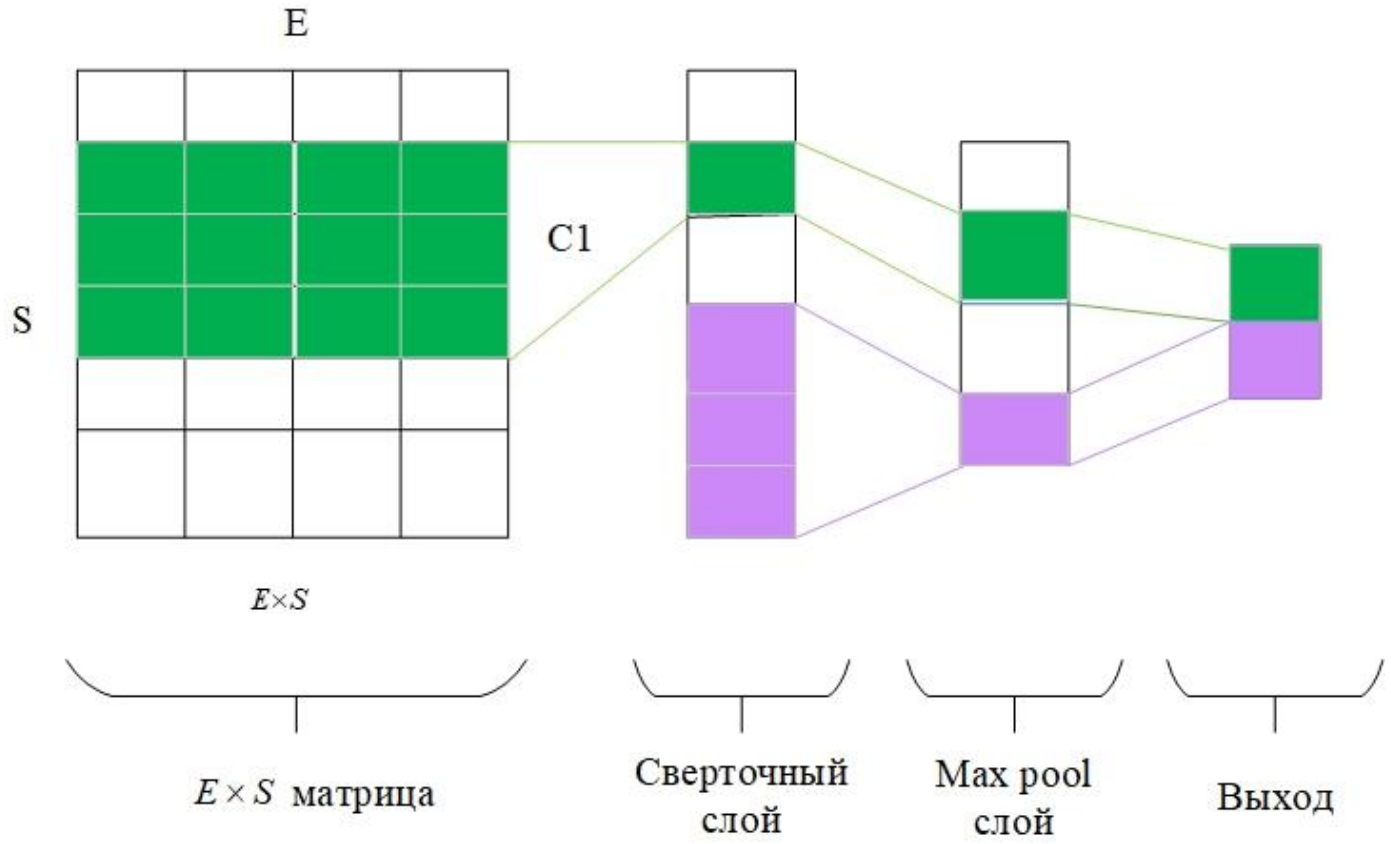
```
with strategy.scope():
    model_dnn = Sequential()
    model_dnn.add(Dense(32, input_dim=20))
    model_dnn.add(Activation('tanh'))
    model_dnn.add(Dropout(0.4))
    model_dnn.add(Dense(16))
    model_dnn.add(Activation('tanh'))
    model_dnn.add(Dense(1))
    model_dnn.add(Activation('sigmoid'))
    optimizer = Adam(learning_rate=0.000008)
    model_dnn.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy', Precision(), Recall(),
                                                                                 tfa.metrics.FBetaScore(num_classes=2,average="micro"
    model_dnn.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 32)                672

 activation (Activation)     (None, 32)                0

 dropout (Dropout)           (None, 32)                0

 dense_1 (Dense)             (None, 16)                528

 activation_1 (Activation)   (None, 16)                0

 dense_2 (Dense)             (None, 1)                 17

 activation_2 (Activation)   (None, 1)                 0

=================================================================
Total params: 1,217
Trainable params: 1,217
Non-trainable params: 0
_________________________________________________________________
```

СВЕРТОЧНЫЕ НЕЙРОННЫЕ СЕТИ

Сверточные нейронные сети (CNN) – один из популярных и часто используемых видов нейронных сетей, приобрётший большую популярность благодаря использованию в задачах классификации и распознавания изображений. Они применяются при работе с изображениями, в которых фильтр перемещается по самому изображению. Также сверточные нейронные сети получили распространение и в задачах по распознаванию речи, обработке естественных языков и анализу тональности. При работе с текстовыми данными необходимо учитывать, что слова имеют разную длину, и в векторном представлении их необходимо привести к одинаковой размерности. Для векторного преобразования обычно используются такие вхождения слов, как Word2vec, Glove и FastText.

СВЕРТОЧНЫЕ НЕЙРОННЫЕ СЕТИ



СВЕРТОЧНЫЕ НЕЙРОННЫЕ СЕТИ

```
with strategy.scope():

    model_cnn = Sequential()
    model_cnn.add(Conv1D(filters=nb_filter, kernel_size=filter_length, activation='relu', input_shape=(20,1)))
    model_cnn.add(GlobalMaxPooling1D())
    model_cnn.add(Dense(hidden_dims))
    model_cnn.add(Dropout(0.4))
    model_cnn.add(Activation('relu'))
    model_cnn.add(Flatten())
    model_cnn.add(Dense(32, activation='relu'))
    model_cnn.add(Dropout(0.4))
    model_cnn.add(Dense(16, activation='relu'))
    model_cnn.add(Dropout(0.4))
    model_cnn.add(Dense(1, activation='sigmoid'))
    optimizer = Adam(learning_rate=0.000008)
    model_cnn.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy', Precision(), Recall(),
                                                                                 tfa.metrics.FBetaScore(num_classes=2,average="micro"
    model_cnn.summary()
```

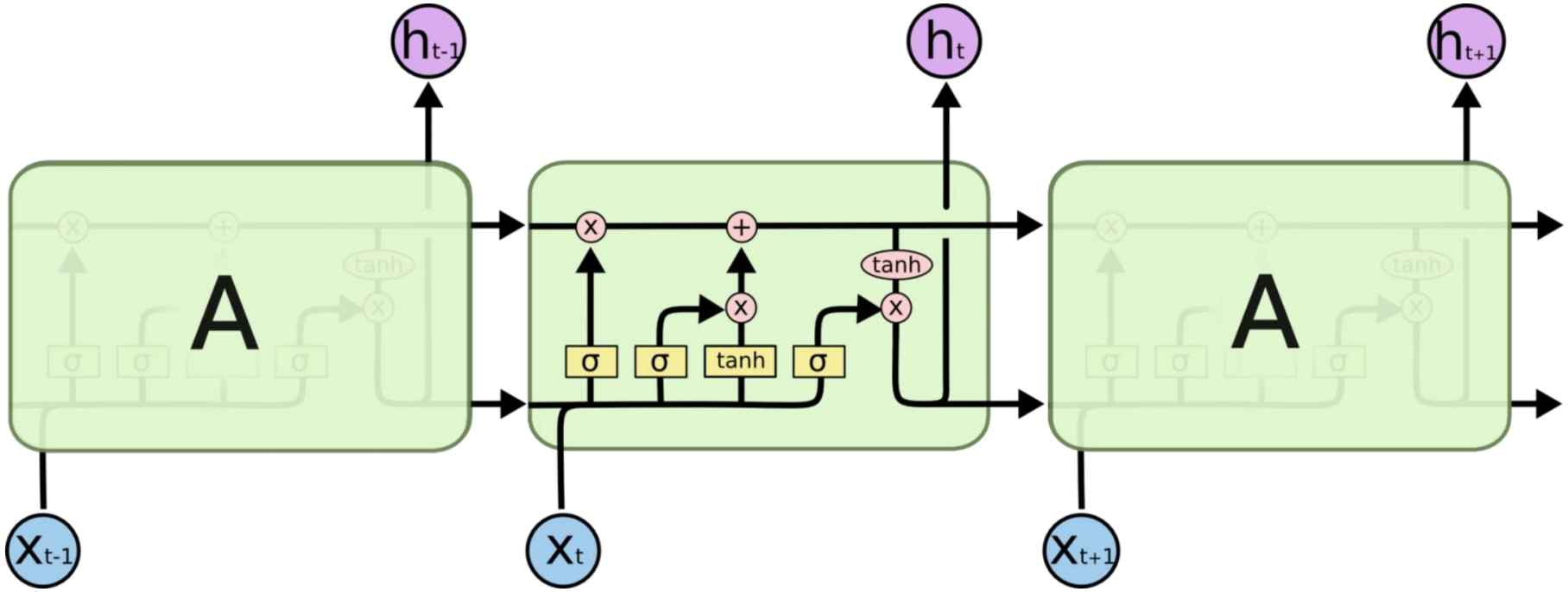
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 18, 250)           1000

 global_max_pooling1d (Globa  (None, 250)              0
 lMaxPooling1D)

 dense (Dense)               (None, 250)               62750

 dropout (Dropout)           (None, 250)               0

 activation (Activation)     (None, 250)               0

 flatten (Flatten)           (None, 250)               0

 dense_1 (Dense)             (None, 32)                8032

 dropout_1 (Dropout)         (None, 32)                0

 dense_2 (Dense)             (None, 16)                528

 dropout_2 (Dropout)         (None, 16)                0

 dense_3 (Dense)             (None, 1)                 17

=================================================================
Total params: 72,327
Trainable params: 72,327
Non-trainable params: 0
```

LONG SHORT-TERM MEMORY - LSTM

- Долгая краткосрочная память (Long short-term memory - LSTM) – особая разновидность архитектуры рекуррентных нейронных сетей, способная к обучению долговременным зависимостям. Они были представлены Зеппом Хохрайтер и Юргеном Шмидхубером в 1997 году, а затем усовершенствованы и популярно изложены в работах многих других исследователей. Они прекрасно решают целый ряд разнообразных задач и в настоящее время широко используются.
- LSTM разработаны специально, чтобы избежать проблемы долговременной зависимости. Запоминание информации на долгие периоды времени – это их обычное поведение, а не что-то, чему они с трудом пытаются обучиться.
- Любая рекуррентная нейронная сеть имеет форму цепочки повторяющихся модулей нейронной сети. В обычной RNN структура одного такого модуля очень проста, например, он может представлять собой один слой с функцией активации tanh (гиперболический тангенс).

LONG SHORT-TERM MEMORY - LSTM

- Структура LSTM также напоминает цепочку, но модули выглядят иначе. Вместо одного слоя нейронной сети они содержат целых четыре, и эти слои взаимодействуют особенным образом



LONG SHORT-TERM MEMORY - LSTM

```
with strategy.scope():

    model_lstm = Sequential()
    model_lstm.add(LSTM(64, input_shape=(20,1), return_sequences = True))
    model_lstm.add(SpatialDropout1D(0.25))
    model_lstm.add(LSTM(32, dropout=0.5, recurrent_dropout=0.5))
    model_lstm.add(Dropout(0.2))
    model_lstm.add(Dense(1, activation='sigmoid'))
    optimizer = Adam(learning_rate=0.000008)
    model_lstm.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy', Precision(), Recall(),
                                                                                  tfa.metrics.FBetaScore(num_classes=2,average="micro'
    model_lstm.summary()
```

```
WARNING:tensorflow:Layer lstm_3 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel
as fallback when running on GPU.
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_2 (LSTM)               (None, 20, 64)            16896

 spatial_dropout1d_1 (Spatia  (None, 20, 64)           0
 lDropout1D)

 lstm_3 (LSTM)               (None, 32)                12416

 dropout_1 (Dropout)         (None, 32)                0

 dense_1 (Dense)             (None, 1)                 33

=================================================================
Total params: 29,345
Trainable params: 29,345
Non-trainable params: 0
_________________________________________________________________
```

СПАСИБО ЗА ВНИМАНИЕ!!!