

ЛЕКЦИЯ 5

Обзор моделей нейронных сетей Dense neural network,
Convolutional neural network, Recurrent neural network, Transformer
для выявления DDoS атак

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 DDOS-АТАК С ПОМОЩЬЮ МАШИННОГО ОБУЧЕНИЯ

Сбор данных

На этом этапе формируется датасет с трафиком сети, который включает как нормальный, так и вредоносный трафик (DDoS-атаки).

Источники данных:

- Лог-файлы серверов
- Захват трафика с помощью **Wireshark, Tcpdump**
- Датасеты, например, **CICDDoS2019, NSL-KDD, CAIDA DDoS**
- Мониторинг в реальном времени через **SNMP, NetFlow, sFlow**

Параметры трафика:

- IP-адреса отправителя и получателя
- Число пакетов за единицу времени
- Размер пакетов
- Количество соединений с одним IP
- Используемые протоколы (TCP, UDP, ICMP)
- Время жизни соединений (TTL)

ЭТАПЫ ВЫЯВЛЕНИЯ DDOS-АТАК С ПОМОЩЬЮ МАШИННОГО ОБУЧЕНИЯ

	index	Unnamed: 0	Flow ID	Src IP	Src Port	Dst IP	Dst Port	Protocol	Timestamp	Flow Duration	...	Fwd Seg Size Min	Act Me
0	1886844	1811706	172.31.0.2-172.31.67.82-53-59931-17	172.31.67.82	59931	172.31.0.2	53	17	20/02/2018 10:44:37	20349	...	8	
1	1667897	4243700	172.31.66.81-13.89.190.129-49673-443-6	172.31.66.81	49673	13.89.190.129	443	6	20/02/2018 12:01:12	77452	...	20	
2	2590662	2634346	172.31.64.87-209.85.203.132-50315-443-6	209.85.203.132	443	172.31.64.87	50315	6	20/02/2018 09:31:35	21010	...	20	
3	2619695	25116	172.217.6.193-192.168.10.12-80-41588-6	192.168.10.12	41588	172.217.6.193	80	6	03/07/2017 06:16:17 PM	5436329	...	0	
4	2387220	3687623	172.31.0.2-172.31.66.28-53-51056-17	172.31.66.28	51056	172.31.0.2	53	17	20/02/2018 09:04:25	70607	...	8	
...	...	...	...	...	...	...	...	...	...	...	...	...	
3186677	1081031	1473462	172.31.69.25-18.219.193.20-80-40358-6	18.219.193.20	40358	172.31.69.25	80	6	16/02/2018 11:25:29 PM	13964	...	0	
3186678	1649871	3439156	172.31.66.116-64.30.228.118-49756-443-6	172.31.66.116	49756	64.30.228.118	443	6	20/02/2018 08:33:35	128223	...	20	
3186679	1621073	4592346	172.31.67.12-216.58.198.66-49851-443-6	172.31.67.12	49851	216.58.198.66	443	6	20/02/2018 08:42:07	52905037	...	20	
3186680	1886020	372111	192.168.10.25-23.194.182.12-59968-443-6	192.168.10.25	59968	23.194.182.12	443	6	03/07/2017 09:33:39 PM	3	...	0	
3186681	992532	1119466	172.31.69.25-18.219.193.20-80-47790-6	18.219.193.20	47790	172.31.69.25	80	6	16/02/2018 11:22:50 PM	15017	...	0	

3186682 rows x 86 columns

ЭТАПЫ ВЫЯВЛЕНИЯ DDOS-АТАК С ПОМОЩЬЮ МАШИННОГО ОБУЧЕНИЯ

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 DDOS-АТАК С ПОМОЩЬЮ МАШИННОГО ОБУЧЕНИЯ

```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
```

```
cat_cols = ['Flow ID', 'Src IP', 'Dst IP', 'Timestamp']
numerical_cols = list(set(DDoS.columns.values.tolist() - set(cat_cols) - set(['Label']) - set(['Label_val'])))
```

```
DDoS_cat = DDoS[['Flow ID', 'Src IP', 'Dst IP', 'Timestamp']]
DDoS_num = DDoS[numerical_cols]
```

```
DDoS['Flow ID']
```

```
0      172.31.0.2-172.31.67.82-53-59931-17
1      172.31.66.81-13.89.190.129-49673-443-6
2      172.31.64.87-209.85.203.132-50315-443-6
3      172.217.6.193-192.168.10.12-80-41588-6
4      172.31.0.2-172.31.66.28-53-51056-17
...
99995   204.46.57.85-192.168.0.2-9440-5000-6
99996   83.190.207.124-192.168.0.2-54034-5000-6
99997   104.125.75.232-192.168.0.2-59426-5000-6
99998   176.33.53.116-192.168.0.2-40697-5000-6
99999   159.101.32.120-192.168.0.2-52542-5000-6
Name: Flow ID, Length: 3286682, dtype: object
```

```
le = LabelEncoder()
```

```
le_list = []
```

```
le.fit(DDoS['Flow ID'])
DDoS_cat['Flow ID'] = le.transform(DDoS_cat['Flow ID'])
le_list.append(le)
le = LabelEncoder()
le.fit(DDoS['Src IP'])
DDoS_cat['Src IP'] = le.transform(DDoS_cat['Src IP'])
le_list.append(le)
le = LabelEncoder()
le.fit(DDoS['Dst IP'])
DDoS_cat['Dst IP'] = le.transform(DDoS_cat['Dst IP'])
le_list.append(le)
le = LabelEncoder()
le.fit(DDoS['Timestamp'])
DDoS_cat['Timestamp'] = le.transform(DDoS_cat['Timestamp'])
le_list.append(le)
```

ЭТАПЫ ВЫЯВЛЕНИЯ DDoS-АТАК С ПОМОЩЬЮ МАШИННОГО ОБУЧЕНИЯ

```
scaler = MinMaxScaler()
```

```
scaler.fit(DDoS_combined)
```

```
MinMaxScaler()
```

```
DDoS_scaled = pd.DataFrame(scaler.transform(DDoS_combined))
```

```
DDoS_scaled
```

	0	1	2	3	4	5	6	7	8	9	...	73	74	75	76	77	
0	0.349130	0.295863	0.221782	0.769459	0.080872	0.000057	0.0	0.000000	0.0	1.661833e-04	...	0.0	1.965699e-05	2.567790e-05	0.000000	0.0	0.000000
1	0.628769	0.294474	0.143003	0.822352	0.061017	0.000323	0.0	0.000000	0.0	6.454333e-04	...	0.0	5.164489e-06	8.336979e-09	0.000000	0.0	0.0067
2	0.468142	0.483225	0.225665	0.719018	0.000000	0.000088	0.0	0.000000	0.0	1.750833e-04	...	0.0	1.903855e-05	8.336979e-09	0.000000	0.0	0.7677
3	0.116046	0.403976	0.218061	0.022607	0.000000	0.015101	0.0	0.000000	0.0	4.499773e-02	...	0.0	7.357906e-08	3.045999e-04	0.000000	0.0	0.0012
4	0.251855	0.293873	0.221782	0.700256	0.044068	0.000196	0.0	0.000000	0.0	5.848583e-04	...	0.0	5.665161e-06	4.333062e-04	0.000000	0.0	0.0000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3286677	0.984616	0.466377	0.365153	0.953876	0.020272	0.000002	1.0	0.172233	0.0	6.298930e-07	...	0.0	8.459306e-04	3.033699e-06	0.000358	1.0	0.0762
3286678	0.998066	0.938667	0.365153	0.953887	0.090211	0.000004	1.0	0.015220	1.0	6.515482e-07	...	1.0	7.757472e-04	5.974297e-06	0.000230	1.0	0.0762
3286679	0.000688	0.023537	0.365153	0.953899	0.033378	0.000003	0.0	0.037385	0.0	8.097211e-06	...	0.0	2.398819e-04	7.416420e-06	0.000329	0.0	0.0762
3286680	0.934107	0.311630	0.365153	0.953910	0.002239	0.000004	1.0	0.049674	1.0	4.030078e-06	...	1.0	5.670139e-05	5.992349e-06	0.000215	1.0	0.0762
3286681	0.040526	0.241005	0.365153	0.953922	0.065551	0.000002	0.0	0.166072	1.0	8.023003e-06	...	0.0	7.473625e-04	5.004525e-06	0.000259	0.0	0.0762

3286682 rows x 83 columns

Chi_square feature selection

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

```
fit_feat = bestfeatures.fit(DDoS_scaled, DDoS['Label_val'])
```

```
DDoS_scores = pd.DataFrame(fit_feat.scores_)  
DDoS_columns = pd.DataFrame(DDoS_scaled.columns)
```

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

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

	Specs	Score
71	71	388598.076703
8	8	370244.162533
66	66	220029.050536
0	0	193953.300683
78	78	145341.568751
67	67	136127.520106
9	9	128360.427367
56	56	113455.768657
12	12	87027.910555
68	68	82895.984496
33	33	82196.174561
51	51	80343.230693
49	49	74135.051586
58	58	70569.787748
47	47	57870.084500
62	62	37909.910790
40	40	35396.278029
70	70	27414.719814
4	4	21201.278943

ЭТАПЫ ВЫЯВЛЕНИЯ DDOS-АТАК С ПОМОЩЬЮ МАШИННОГО ОБУЧЕНИЯ

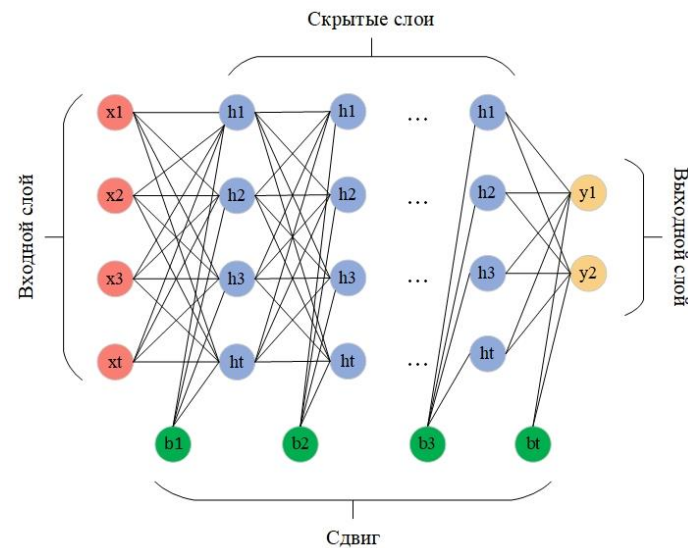
- **Разделение данных на обучающую и тестовую выборки**
- Данные разделяются следующим образом:
 - **Обучающая выборка (Train Set):** 70-80% данных
 - **Тестовая выборка (Test Set):** 20-30% данных
- Дополнительно можно использовать **кросс-валидацию (k-fold cross-validation)** для более точной оценки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

НЕЙРОННЫЕ СЕТИ

- Традиционные алгоритмы машинного обучения позволяют добиться хороших результатов классификации данных. Тем не менее в последнее десятилетие подход на основе нейронных сетей стал доминирующим в области искусственного интеллекта, показывающего высокую точность в таких задачах, как распознавание речи и изображений, классификация текстов и обработка естественных языков. Существуют несколько видов часто используемых нейронных сетей: глубокие нейронные сети (DNN), сверточные нейронные сети (CNN) и рекуррентные нейронные сети (RNN). Одной из самых последних разработок в области обработки естественных языков стала нейронная сеть, основанная на архитектуре Трансформера, названная Bidirectional Encoder Representations from Transformers (BERT).

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 $x = x_1, x_2, \dots, x_n$ является входным вектором, $w_1, w_2, \dots, w_l$ веса соединения каждого уровня,
- $b_1, b_2, \dots, b_l$ - вектор смещения. Уровни от l_2 до l_{n-1} образуют скрытые слои, $y_1, y_2, \dots, y_m$ 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 входное значение 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(),
                                                                                 tf.keras.metrics.FBetaScore(num_classes=2, average='micro')])
    model_dnn.summary()
```

Model: "sequential"

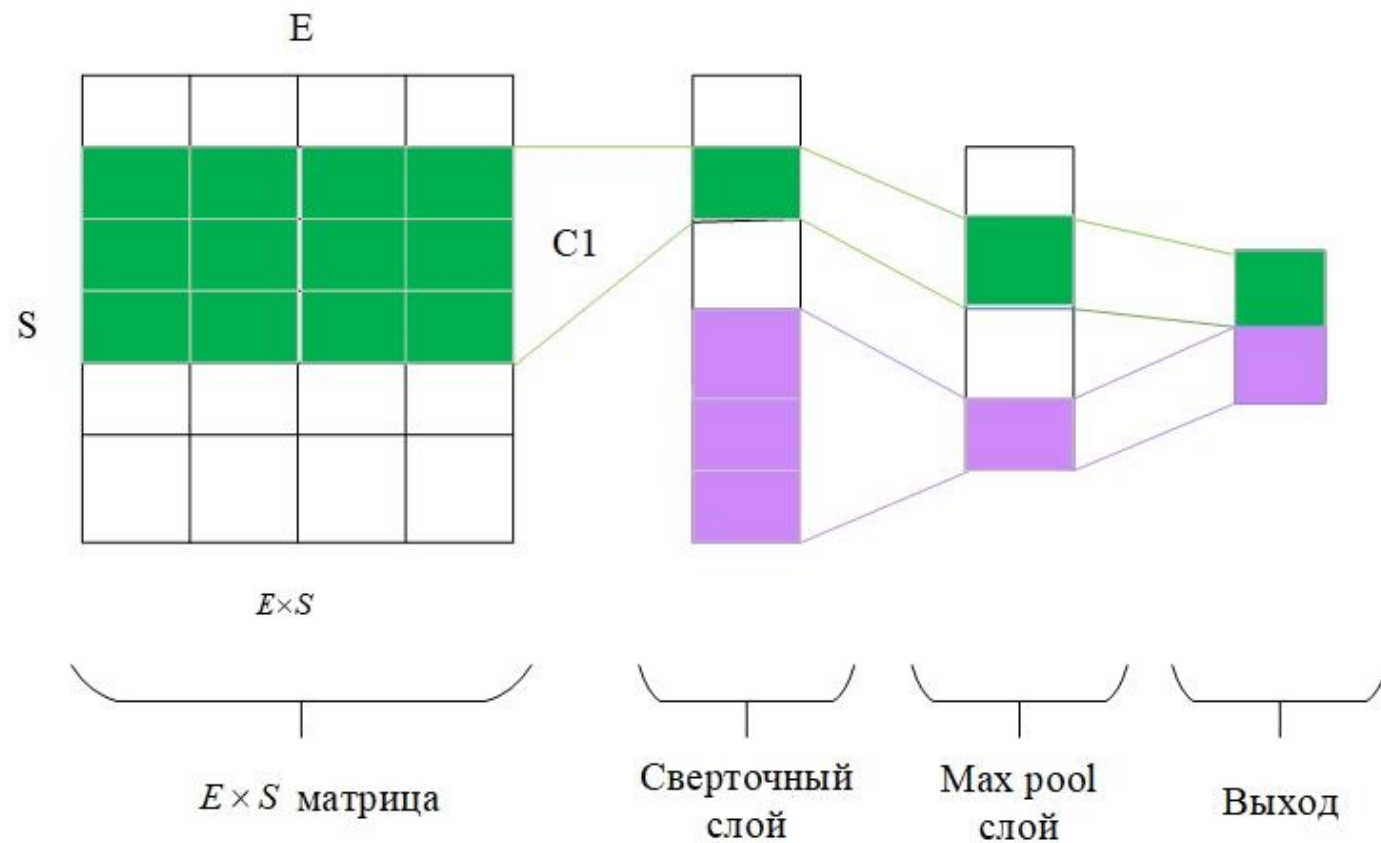
Layer (type)	Output Shape	Param #
dense (Dense)	(None, 32)	672
activation (Activation)	(None, 32)	0
dropout (Dropout)	(None, 32)	0
dense_1 (Dense)	(None, 16)	528
activation_1 (Activation)	(None, 16)	0
dense_2 (Dense)	(None, 1)	17
activation_2 (Activation)	(None, 1)	0

=====
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, приобретший 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=0.00008)  
    model_cnn.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy', Precision(), Recall(),  
                                                                              tf.keras.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 (GlobalMaxPooling1D)	(None, 250)	0
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

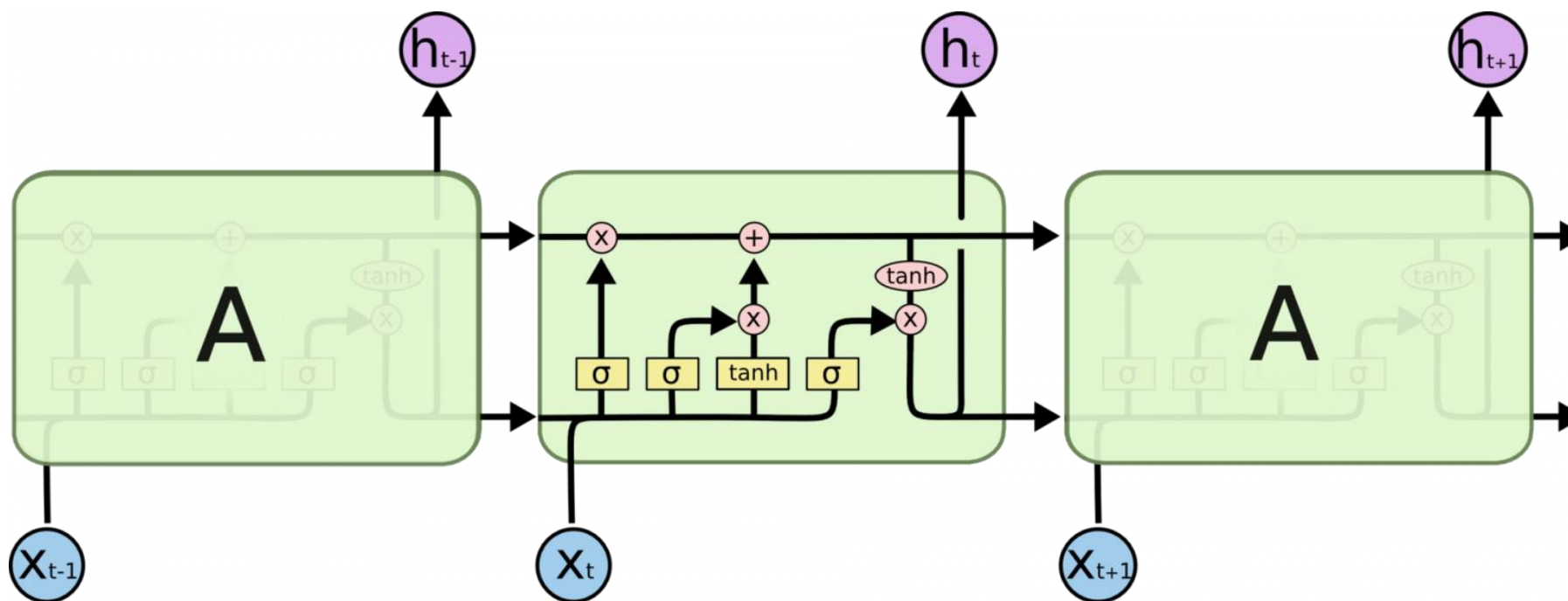
=====
Total params: 72,327
Trainable params: 72,327
Non-trainable params: 0

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 особым 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(),  
                                                                               tf.keras.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 (SpatialDropout1D)	(None, 20, 64)	0
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!!!