

ЛЕКЦИЯ 9

Изучение выявления Вредоносного ПО
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 MALWARE С ПОМОЩЬЮ НЕЙРОННЫХ СЕТЕЙ

Сбор данных

Для обучения модели необходима большая выборка файлов и сетевых данных, содержащих как **вредоносное ПО**, так и **чистые файлы**.

Источники данных:

- **VirusTotal, MalwareBazaar, Malicia Dataset** – базы вредоносного ПО
- **Логи антивирусов и IDS-систем** (Snort, Suricata)
- **Образцы исполняемых файлов** (PE-файлы в Windows, ELF-файлы в Linux)
- **Сетевой трафик** (Wireshark, NetFlow)

Типы данных:

- **Статические признаки** (размер файла, хеши, секции PE, импортированные библиотеки)
- **Динамические признаки** (поведение в виртуальной среде, анализ API-вызовов)
- **Сетевой анализ** (аномалии в HTTP, DNS-запросах, командно-контрольные сервера)

ЭТАПЫ ВЫЯВЛЕНИЯ MALWARE С ПОМОЩЬЮ НЕЙРОННЫХ СЕТЕЙ

malware

	index	SizeOfOptionalHeader	Characteristics	MajorLinkerVersion	MinorLinkerVersion	SizeOfCode	SizeOfInitializedData	SizeOfUninitializedData	Address
0	117796	224	8450	9.0	0	1024	1024	0	
1	23686	224	8450	8.0	0	35328	10752	0	
2	70211	224	271	6.0	0	12288	20480	0	
3	95900	224	8450	8.0	0	167936	8192	0	
4	8183	224	8482	14.0	0	4096	15360	0	
...	...	...	...	...	...	...	...	...	...
216346	210867	224	259	9.0	0	99328	1148928	0	
216347	176116	224	258	10.0	0	119808	375808	0	
216348	99211	224	8462	8.0	0	1536	2048	0	
216349	169002	224	258	10.0	0	28672	445952	16896	
216350	213842	224	783	2.0	56	29184	14848	110592	

216351 rows × 55 columns

ЭТАПЫ ВЫЯВЛЕНИЯ MALWARE С ПОМОЩЬЮ НЕЙРОННЫХ СЕТЕЙ

Предобработка данных

Данные необходимо очистить и подготовить к обучению.

Действия:

- **Удаление дубликатов и выбросов**
- **Приведение категориальных данных к числовому виду**
- **Нормализация признаков** (например, MinMaxScaler)
- **Feature Engineering** – создание новых признаков (например, частота API-вызовов)

ЭТАПЫ ВЫЯВЛЕНИЯ MALWARE С ПОМОЩЬЮ НЕЙРОННЫХ СЕТЕЙ

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
```

```
scaler.fit(malware_data)
```

```
MinMaxScaler()
```

```
malware_scaled = pd.DataFrame(scaler.transform(malware_data))
```

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

malware_data

	index	SizeOfOptionalHeader	Characteristics	MajorLinkerVersion	MinorLinkerVersion	SizeOfCode	SizeOfInitializedData	SizeOfUninitializedData	Address
0	117796	224	8450	9.0	0	1024	1024	0	
1	23686	224	8450	8.0	0	35328	10752	0	
2	70211	224	271	6.0	0	12288	20480	0	
3	95900	224	8450	8.0	0	167936	8192	0	
4	8183	224	8482	14.0	0	4096	15360	0	
...	...	...	...	...	...	...	...	...	...
216346	210867	224	259	9.0	0	99328	1148928	0	
216347	176116	224	258	10.0	0	119808	375808	0	
216348	99211	224	8462	8.0	0	1536	2048	0	
216349	169002	224	258	10.0	0	28672	445952	16896	
216350	213842	224	783	2.0	56	29184	14848	110592	

216351 rows × 54 columns

ЭТАПЫ ВЫЯВЛЕНИЯ MALWARE С ПОМОЩЬЮ НЕЙРОННЫХ СЕТЕЙ

Chi_square feature selection

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

```
fit_feat = bestfeatures.fit(malware_scaled, malware['legitimate'])
```

```
malware_scores = pd.DataFrame(fit_feat.scores_)  
malware_columns = pd.DataFrame(malware_scaled.columns)
```

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

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

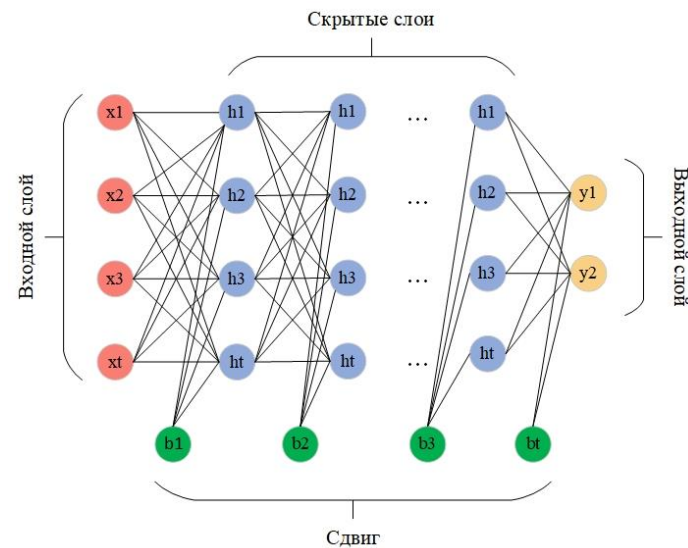
	Specs	Score
0	0	10610.997896
24	24	10285.818917
22	22	3190.340238
2	2	2987.899315
34	34	1301.058610
48	48	1282.168947
47	47	1070.573703
18	18	813.731644
53	53	610.707103
32	32	582.777707
4	4	428.536642
33	33	389.225729
23	23	347.613734
16	16	317.885149
17	17	290.498051
44	44	258.081053
43	43	138.408542
36	36	127.392942
39	39	103.853904
45	45	101.951695

ЭТАПЫ ВЫЯВЛЕНИЯ MALWARE 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 $x = x_1, x_2, \dots, x_n$ 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"

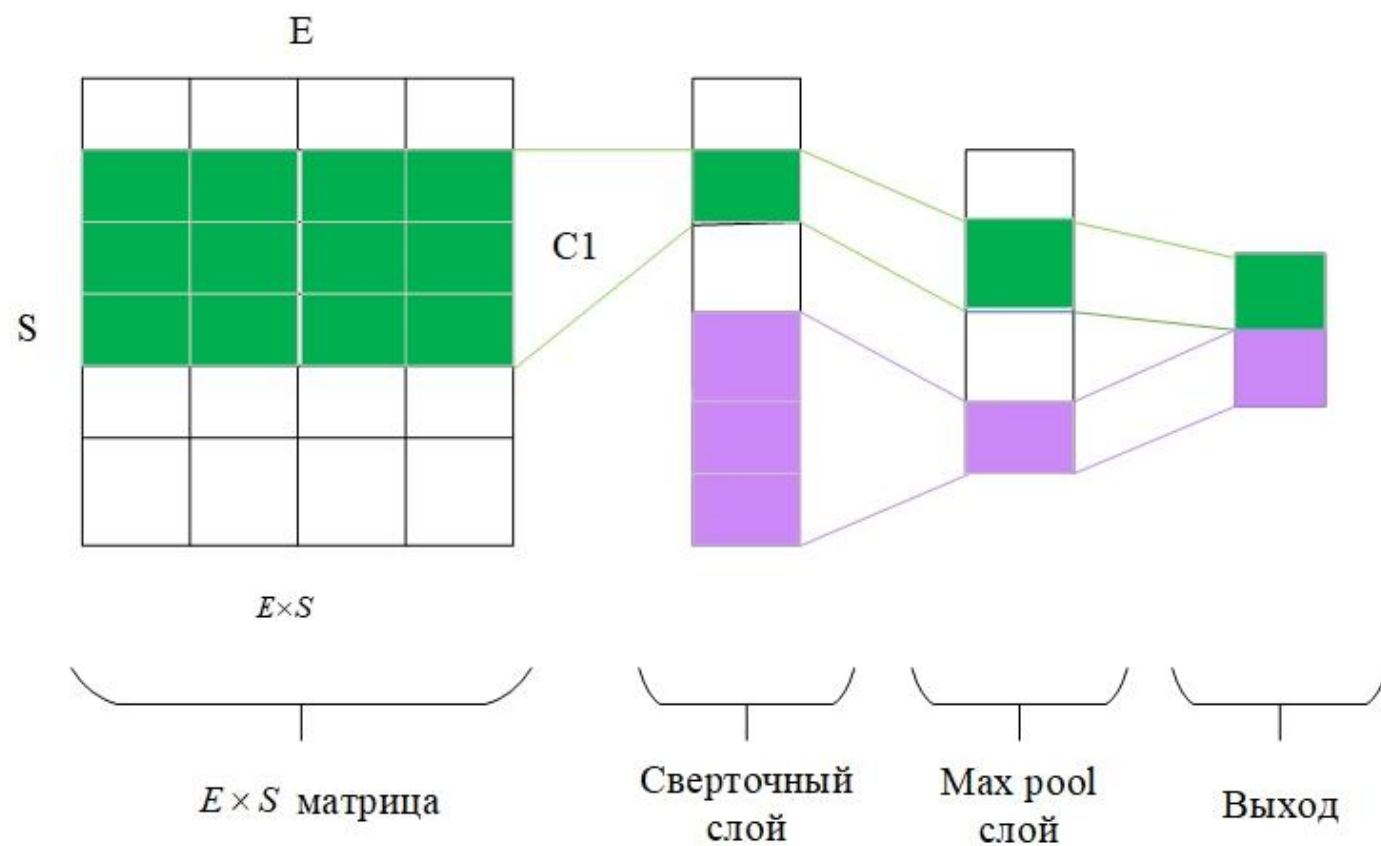
Layer (type)	Output Shape	Param #
dense (Dense)	(None, 32)	672
activation (Activation)	(None, 32)	0
dropout (Dropout)	(None, 32)	0
dense_1 (Dense)	(None, 16)	528
activation_1 (Activation)	(None, 16)	0
dense_2 (Dense)	(None, 1)	17
activation_2 (Activation)	(None, 1)	0

=====
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

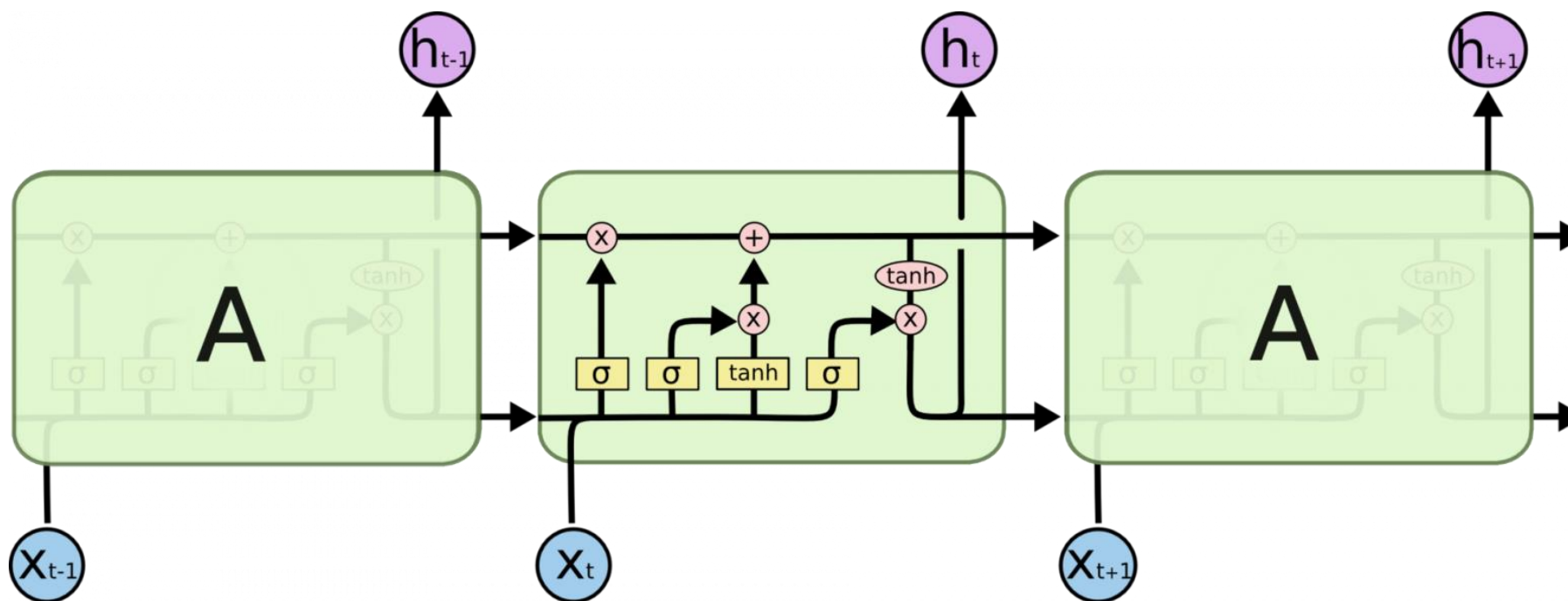
```
=====
Total params: 72,327
Trainable params: 72,327
Non-trainable params: 0
```

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!!!