

301 lines - 51 Removals

```

1 import time
2 import optax
3 import flax.linen as nn
4 from malware_utils import *
5 from dataset import load_dataset
6 from flax.training import train_state
7 from HRR.with_flax import Binding, Unbinding, CosineSimilarity
8
9 from flax import jax_utils
10 from flax.training.common_utils import shard, shard_prng_key
11
12
13 class MHAttention(nn.Module):
14     features: int
15     heads: int = 8
16
17     def setup(self):
18         self.binding = Binding()
19         self.unbinding = Unbinding()
20         self.similarity = CosineSimilarity()
21
22     @nn.compact
23     def __call__(self, query, key, value, mask=None):
24         dense_q = nn.Dense(features=self.features)
25         dense_k = nn.Dense(features=self.features)
26         dense_v = nn.Dense(features=self.features)
27         dense_o = nn.Dense(features=self.features)
28
29         q = dense_q(query) # (B, T, H)
30         k = dense_k(key) # (B, T, H)
31         v = dense_v(value) # (B, T, H)
32
33         q = split(q, self.heads) # (B, h, T, H')
34         k = split(k, self.heads) # (B, h, T, H')
35         v = split(v, self.heads) # (B, h, T, H')
36
37         bind = self.binding(k, v, axis=-1) # (B, h, T, H')
38         bind = np.sum(bind, axis=-2, keepdims=True) # (B,
39         h, 1, H')
40
41         vp = self.unbinding(bind, q, axis=-1) # (B, h, T,
42         H')
43
44         scale = self.similarity(v, vp, axis=-1, keepdims=True) # (B, h, T, 1)
45
46         scale = scale + (1. - mask) * (-1e9)
47         weight = nn.softmax(scale, axis=-2)
48         weighted_value = weight * v
49
50         output = merge(weighted_value)
51         output = dense_o(output)
52         return output
53
54 class FeedForwardLayer(nn.Module):
55     features: int
56     dropout_rate: float = 0.0
57     training: bool = False
58
59     @nn.compact
60     def __call__(self, inputs):
61         x = nn.Dense(features=self.features)(inputs)
62         x = nn.relu(x)
63         x = nn.Dense(features=inputs.shape[-1])(x)

```

572 lines + 327 Additions

```

1 import time
2 import datetime
3
4 import flax.linen as nn
5 import optax
6 from HRR.with_flax import Binding, Unbinding, CosineSimilarity
7 from flax import jax_utils
8 from flax.training import train_state
9 from flax.training.common_utils import shard, shard_prng_key
10
11 from MalwareDetectionHRR.initializations import *
12 from dataset import load_dataset
13 from malware_utils import *
14
15
16 class MHAttention(nn.Module):
17     features: int
18     heads: int = 8
19
20     def setup(self):
21         self.binding = Binding()
22         self.unbinding = Unbinding()
23         self.similarity = CosineSimilarity()
24
25     @nn.compact
26     def __call__(self, query, key, value, mask=None):
27         dense_q = nn.Dense(features=self.features)
28         dense_k = nn.Dense(features=self.features)
29         dense_v = nn.Dense(features=self.features)
30         dense_o = nn.Dense(features=self.features)
31
32         q = dense_q(query) # (B, T, H)
33         k = dense_k(key) # (B, T, H)
34         v = dense_v(value) # (B, T, H)
35
36         q = split(q, self.heads) # (B, h, T, H')
37         k = split(k, self.heads) # (B, h, T, H')
38         v = split(v, self.heads) # (B, h, T, H')
39
40         bind = self.binding(k, v, axis=-1) # (B, h, T, H')
41         bind = np.sum(bind, axis=-2, keepdims=True) # (B,
42         h, 1, H')
43
44         vp = self.unbinding(bind, q, axis=-1) # (B, h, T,
45         H')
46
47         scale = self.similarity(v, vp, axis=-1, keepdims=True) # (B, h, T, 1)
48
49         scale = scale + (1. - mask) * (-1e9)
50         weight = nn.softmax(scale, axis=-2)
51         weighted_value = weight * v
52
53         output = merge(weighted_value)
54         output = dense_o(output)
55         return output
56
57 class FeedForwardLayer(nn.Module):
58     features: int
59     dropout_rate: float = 0.0
60     training: bool = False
61
62     @nn.compact
63     def __call__(self, inputs):
64         x = nn.Dense(features=self.features)(inputs)
65         x = nn.relu(x)
66         x = nn.Dense(features=inputs.shape[-1])(x)

```

```

62         x = nn.Dropout(rate=self.dropout_rate)(x, determini
stic=not self.training)
63         return x
64
65
66 class Encoder(nn.Module):
67     features: int
68     dropout_rate: float
69     training: bool = False
70
71     @nn.compact
72     def __call__(self, inputs, mask=None):
73         lnx = nn.LayerNorm()(inputs)
74         attention = MHAttention(features=inputs.shape[-1])
(query=lnx, key=lnx, value=lnx, mask=mask)
75         attention = nn.Dropout(self.dropout_rate, determini
stic=not self.training)(attention)
76         x = inputs + attention
77
78         lnx = nn.LayerNorm()(x)
79         ffn = FeedForwardLayer(self.features, self.dropout_
rate, training=self.training)(lnx)
80         outputs = x + ffn
81         return outputs
82
83
84 class Embedding(nn.Module):
85     vocab_size: int
86     embed_size: int
87     max_seq_len: int
88
89     def setup(self):
90         self.positions = np.arange(start=0, stop=self.max_s
eq_len, step=1)[np.newaxis, :]
91
92     @nn.compact
93     def __call__(self, inputs):
94         word_embedding = nn.Embed(self.vocab_size, self.emb
ed_size)(inputs)
95         position_embedding = nn.Embed(self.max_seq_len, sel
f.embed_size)(self.positions)
96         return word_embedding + position_embedding
97
98
99 class Network(nn.Module):
100     features: int
101     vocab_size: int
102     embed_size: int
103     max_seq_len: int
104     nlayers: int
105     output_size: int
106     dropout_rate: float
107     training: bool
108
109     @nn.compact
110     def __call__(self, encoder_input):
111         encoder_input = encoder_input.astype('int32') #
(B, T)
112
113         en_mask = np.where(encoder_input > 0, 1., 0.)
114         en_mask = en_mask[:, np.newaxis, :, np.newaxis]
115
116         # embedding
117         x = Embedding(vocab_size=self.vocab_size,
embed_size=self.embed_size,
118 max_seq_len=self.max_seq_len)(encoder
_input)
119
120
121         # class token
122         token = self.param('cls_token', nn.initializers.lec
un_normal(), (1, 1, self.embed_size))
123         token = np.repeat(token, x.shape[0], axis=0)
124         x = np.concatenate([token, x], axis=1)
125
126         # adjust mask
127         en_mask = np.concatenate([np.ones((en_mask.shape
[0], 1, 1, 1)), en_mask], axis=-2)

```

```

65         x = nn.Dropout(rate=self.dropout_rate)(x, determini
stic=not self.training)
66         return x
67
68
69 class Encoder(nn.Module):
70     features: int
71     dropout_rate: float
72     training: bool = False
73
74     @nn.compact
75     def __call__(self, inputs, mask=None):
76         lnx = nn.LayerNorm()(inputs)
77         attention = MHAttention(features=inputs.shape[-1])
(query=lnx, key=lnx, value=lnx, mask=mask)
78         attention = nn.Dropout(self.dropout_rate, determini
stic=not self.training)(attention)
79         x = inputs + attention
80
81         lnx = nn.LayerNorm()(x)
82         ffn = FeedForwardLayer(self.features, self.dropout_
rate, training=self.training)(lnx)
83         outputs = x + ffn
84         return outputs
85
86
87 class Embedding(nn.Module):
88     vocab_size: int
89     embed_size: int
90     max_seq_len: int
91
92     def setup(self):
93         self.positions = np.arange(start=0, stop=self.max_s
eq_len, step=1)[np.newaxis, :]
94
95     @nn.compact
96     def __call__(self, inputs):
97         word_embedding = nn.Embed(self.vocab_size, self.emb
ed_size)(inputs)
98         position_embedding = nn.Embed(self.max_seq_len, sel
f.embed_size)(self.positions)
99         return word_embedding + position_embedding
100
101
102 class Network(nn.Module):
103     features: int
104     vocab_size: int
105     embed_size: int
106     max_seq_len: int
107     nlayers: int
108     output_size: int
109     dropout_rate: float
110     training: bool
111
112     @nn.compact
113     def __call__(self, encoder_input):
114         encoder_input = encoder_input.astype('int32') #
(B, T)
115
116         en_mask = np.where(encoder_input > 0, 1., 0.)
117         en_mask = en_mask[:, np.newaxis, :, np.newaxis]
118
119         # embedding
120         x = Embedding(vocab_size=self.vocab_size,
embed_size=self.embed_size,
121 max_seq_len=self.max_seq_len)(encoder
_input)
122
123
124         # class token
125         token = self.param('cls_token', nn.initializers.lec
un_normal(), (1, 1, self.embed_size))
126         token = np.repeat(token, x.shape[0], axis=0)
127         x = np.concatenate([token, x], axis=1)
128
129         # adjust mask
130         en_mask = np.concatenate([np.ones((en_mask.shape
[0], 1, 1, 1)), en_mask], axis=-2)

```

```

128
129         x = nn.Dropout(self.dropout_rate, deterministic=not
self.training)(x)
130
131         for i in range(self.nlayers):
132             x = Encoder(features=self.features,
133                         dropout_rate=self.dropout_rate,
134                         training=self.training)(x, mask=en_
mask)
135
136             # output
137             x = x[:, 0]
138
139             x = nn.Dense(features=x.shape[-1])(x)
140             x = nn.relu(x)
141
142             output = nn.Dense(self.output_size)(x)
143             output = nn.softmax(output, axis=-1)
144             return output
145
146
147 def initialize_model(model, input_size, init_rngs):
148     init_inputs = np.ones([1, input_size])
149     variables = model.init(init_rngs, init_inputs)['param
s']
150     return variables
151
152
153 def train_step(state, batch, rngs):
154     """ train one step """
155     y_true = batch[1]
156
157     def loss_fn(params):
158         y_pred = state.apply_fn({'params': params}, batch
[0], rngs=rngs)
159         loss = cross_entropy_loss(y_true=y_true, y_pred=y_p
red)
160         return loss, y_pred
161
162     grad_fn = jax.value_and_grad(loss_fn, has_aux=True)
163     (loss, y_pred), grads = grad_fn(state.params)
164     grads = jax.lax.pmean(grads, "batch")
165     grads = grad_check(grads)
166     state = state.apply_gradients(grads=grads)
167     acc = accuracy(y_true=y_true, y_pred=y_pred)
168     metrics = {'loss': loss, 'accuracy': acc}
169     return state, metrics
170
171
172 def predict(state, batch, rngs):
173     y_true = batch[1]
174     y_pred = state.apply_fn({'params': state.params}, batch
[0], rngs=rngs)
175     loss = cross_entropy_loss(y_true=y_true, y_pred=y_pred)
176     acc = accuracy(y_true=y_true, y_pred=y_pred)
177     metrics = {'loss': loss, 'accuracy': acc}
178     return metrics
179
180
181 def train(batch_size, max_seq_len, features=512, embed_size
=256, lr=1e-3, epochs=10):
182     batch_size = batch_size * jax.device_count()
183     vocab_size = 256 + 1
184     n_layer = 1
185     output_size = 2
186     dropout_rate = 0.1
187     name = 'hrrformer'
188

```

```

131
132         x = nn.Dropout(self.dropout_rate, deterministic=not
self.training)(x)
133
134         for i in range(self.nlayers):
135             x = Encoder(features=self.features,
136                         dropout_rate=self.dropout_rate,
137                         training=self.training)(x, mask=en_
mask)
138
139             # output
140             x = x[:, 0]
141
142             x = nn.Dense(features=x.shape[-1])(x)
143             x = nn.relu(x)
144
145             output = nn.Dense(self.output_size)(x)
146             output = nn.softmax(output, axis=-1)
147             return output
148
149
150 def initialize_model(model, input_size, init_rngs):
151     init_inputs = np.ones([1, input_size])
152     variables = model.init(init_rngs, init_inputs)['param
s']
153     return variables
154
155
156 def train_step(state, batch, rngs):
157     """ train one step """
158     y_true = batch[1]
159
160     def loss_fn(params):
161         y_pred = state.apply_fn({'params': params}, batch
[0], rngs=rngs)
162         loss = cross_entropy_loss(y_true=y_true, y_pred=y_p
red)
163         return loss, y_pred
164
165     grad_fn = jax.value_and_grad(loss_fn, has_aux=True)
166     (loss, y_pred), grads = grad_fn(state.params)
167     grads = jax.lax.pmean(grads, "batch")
168     grads = grad_check(grads)
169     state = state.apply_gradients(grads=grads)
170     acc = accuracy(y_true=y_true, y_pred=y_pred)
171     metrics = {'loss': loss, 'accuracy': acc}
172     return state, metrics
173
174
175 def predict_ind(state, batch, rngs):
176     y_true = batch[1]
177     y_pred = state.apply_fn({'params': state.params}, batch
[0], rngs=rngs)
178     loss = cross_entropy_loss(y_true=y_true, y_pred=y_pred)
179     acc = accuracy(y_true=y_true, y_pred=y_pred)
180     metrics = {'loss': loss, 'accuracy': acc}
181     return metrics
182
183
184 def predict(state, batch, rngs):
185     y_true = batch[1]
186     y_pred = state.apply_fn({'params': state.params}, batch
[0], rngs=rngs)
187     loss = cross_entropy_loss(y_true=y_true, y_pred=y_pred)
188     acc = accuracy(y_true=y_true, y_pred=y_pred)
189     metrics = {'loss': loss, 'accuracy': acc}
190     return metrics
191
192
193 def train(batch_size, max_seq_len, features=512, embed_size
=256, lr=1e-3, epochs=10):
194     batch_size = batch_size * jax.device_count()
195     vocab_size = 256 + 1
196     n_layer = 1
197     output_size = 2
198     dropout_rate = 0.1
199     name = 'hrrformer'
200

```

```

189     print('total batch size:', batch_size, 'max seq len:',
190           max_seq_len)
191
192     # load dataset
193     train_loader, test_loader = load_dataset(batch_size=batch_size,
194                                               max_seq_len=max_seq_len,
195                                               shuffle=True,
196                                               num_workers=0)
197
198     # build and initialize network
199     train_model = Network(features=features,
200                           vocab_size=vocab_size,
201                           embed_size=embed_size,
202                           max_seq_len=max_seq_len,
203                           n_layers=n_layer,
204                           output_size=output_size,
205                           dropout_rate=dropout_rate,
206                           training=True)
207
208     test_model = Network(features=features,
209                          vocab_size=vocab_size,
210                          embed_size=embed_size,
211                          max_seq_len=max_seq_len,
212                          n_layers=n_layer,
213                          output_size=output_size,
214                          dropout_rate=dropout_rate,
215                          training=False)
216
217     p_key_next, p_key = jax.random.split(jax.random.PRNGKey
218                                           (0))
219     d_key_next, d_key = jax.random.split(jax.random.PRNGKey
220                                           (0))
221     init_rngs = {'params': p_key, 'dropout': d_key}
222
223     params = initialize_model(model=train_model, input_size
224                               =max_seq_len, init_rngs=init_rngs)
225
226     # optimizer and scheduler
227     steps = 800_000 // batch_size
228     scheduler = optax.exponential_decay(init_value=lr,
229                                         transition_steps=st
230                                         eps,
231                                         decay_rate=.85,
232                                         transition_begin=1,
233                                         end_value=1e-5)
234
235     tx = optax.adam(learning_rate=scheduler)
236     state = train_state.TrainState.create(apply_fn=train_mo
237                                           del.apply, params=params, tx=tx)
238
239     # state = load_model(state, f'weights/{name}_multi_{n_l
240     ayer}_{max_seq_len}.h5')
241     state = jax_utils.replicate(state)
242
243     # train
244
245     history = []
246     train_loss, train_acc = [], []
247     test_loss, test_acc = [], []
248
249     form = 'Epoch {0:>3d}/' + str(epochs) + ', train loss:
250           {1:>8.6f}, train accuracy: {2:>5.2f}%, '
251     form += 'test loss: {3:>8.6f}, test accuracy: {4:>5.2
252           f}%, etc: {5:>.2f}s'
253
254     for epoch in range(1, epochs + 1):
255         train_loss_batch, train_acc_batch = [], []

```

```

201     print()
202     printf('Total batch size: ' + str(batch_size) + ', Max
203           seq len: ' + str(max_seq_len))
204     printf("load dataset")
205
206     # load dataset
207     train_loader, validation_loader, num_train_samples, num
208     _validation_samples = load_dataset(batch_size=batch_size,
209                                         max_seq_len=max_seq_len,
210                                         shuffle=True,
211                                         num_workers=0)
212
213     printf("build and initialize network")
214     # build and initialize network
215     train_model = Network(features=features,
216                           vocab_size=vocab_size,
217                           embed_size=embed_size,
218                           max_seq_len=max_seq_len,
219                           n_layers=n_layer,
220                           output_size=output_size,
221                           dropout_rate=dropout_rate,
222                           training=True)
223
224     validation_model = Network(features=features,
225                                vocab_size=vocab_size,
226                                embed_size=embed_size,
227                                max_seq_len=max_seq_len,
228                                n_layers=n_layer,
229                                output_size=output_size,
230                                dropout_rate=dropout_rate,
231                                training=False)
232
233     p_key_next, p_key = jax.random.split(jax.random.PRNGKey
234                                           (0))
235     d_key_next, d_key = jax.random.split(jax.random.PRNGKey
236                                           (0))
237     init_rngs = {'params': p_key, 'dropout': d_key}
238
239     printf("initialize_model")
240     params = initialize_model(model=train_model, input_size
241                               =max_seq_len, init_rngs=init_rngs)
242
243     printf("optimizer and scheduler")
244
245     # optimizer and scheduler
246     steps = 800_000 // batch_size
247     scheduler = optax.exponential_decay(init_value=lr,
248                                         transition_steps=st
249                                         eps,
250                                         decay_rate=.85,
251                                         transition_begin=1,
252                                         end_value=1e-5)
253
254     tx = optax.adam(learning_rate=scheduler)
255     state = train_state.TrainState.create(apply_fn=train_mo
256                                           del.apply, params=params, tx=tx)
257
258     # state = load_model(state, f'{WEIGHTS_BASE_PATH}{name}
259     _multi_{n_layer}_{max_seq_len}.h5')
260     state = jax_utils.replicate(state)
261
262     printf("training the model...")
263     # training
264
265     history = []
266     train_loss, train_acc = [], []
267     validation_loss, validation_acc = [], []
268
269     form = 'Epoch {0:>3d}/' + str(epochs) + ', train loss:
270           {1:>8.6f}, train accuracy: {2:>5.2f}%, '
271     form += 'validation loss: {3:>8.6f}, validation accurac
272           y: {4:>5.2f}%, etc: {5:>.2f}s'
273
274     for epoch in range(1, epochs + 1):
275         train_loss_batch, train_acc_batch = [], []

```

```

244     state = state.replace(apply_fn=train_model.apply)
245
246     tic1 = time.time()
247     for x_train, y_train in train_loader:
248         p_key_next, p_key = jax.random.split(p_key_next)
249
250         d_key_next, d_key = jax.random.split(d_key_next)
251
252         rngs = {'params': shard_prng_key(p_key), 'dropout': shard_prng_key(d_key)}
253
254         batch = [x_train, y_train]
255         batch = shard(batch)
256
257         state, metrics = jax.pmap(train_step, axis_name="batch", donate_argnums=(0,))(state, batch, rngs)
258
259         train_loss_batch.append(metrics['loss'].mean())
260         train_acc_batch.append(metrics['accuracy'].mean())
261
262     ()
263
264     toc1 = time.time()
265     train_loss.append(sum(train_loss_batch) / len(train_loss_batch))
266     train_acc.append(sum(train_acc_batch) / len(train_acc_batch))
267
268     # test
269     test_loss_batch, test_acc_batch = [], []
270
271     state = state.replace(apply_fn=test_model.apply)
272
273     tic2 = time.time()
274     for x_test, y_test in test_loader:
275         p_key_next, p_key = jax.random.split(p_key_next)
276
277         d_key_next, d_key = jax.random.split(d_key_next)
278
279         rngs = {'params': shard_prng_key(p_key), 'dropout': shard_prng_key(d_key)}
280
281         test_batch = [x_test, y_test]
282         test_batch = shard(test_batch)
283
284         metrics = jax.pmap(predict, axis_name="batch")(state, test_batch, rngs)
285
286         test_loss_batch.append(metrics['loss'].mean())
287         test_acc_batch.append(metrics['accuracy'].mean())
288
289     ()
290
291     toc2 = time.time()
292     test_loss.append(sum(test_loss_batch) / len(test_loss_batch))
293     test_acc.append(sum(test_acc_batch) / len(test_acc_batch))
294
295     etc = (toc1 - tic1) + (toc2 - tic2)
296     history.append(form.format(epoch, train_loss[-1], train_acc[-1], test_loss[-1], test_acc[-1], etc))
297
298     print(history[-1])
299
300     state = jax_utils.unreplicate(state)
301
302     save_model(state, f'weights/{name}_multi_{n_layer}_{max_seq_len}.h5')
303
304     save_history(f'weights/{name}_multi_{n_layer}_{max_seq_len}.csv', history=history)

```

```

263     state = state.replace(apply_fn=train_model.apply)
264
265     tic1 = time.time()
266     for x_train, y_train in train_loader:
267         p_key_next, p_key = jax.random.split(p_key_next)
268
269         d_key_next, d_key = jax.random.split(d_key_next)
270
271         rngs = {'params': shard_prng_key(p_key), 'dropout': shard_prng_key(d_key)}
272
273         batch = [x_train, y_train]
274         batch = shard(batch)
275
276         state, metrics = jax.pmap(train_step, axis_name="batch", donate_argnums=(0,))(state, batch, rngs)
277
278         train_loss_batch.append(metrics['loss'].mean())
279         train_acc_batch.append(metrics['accuracy'].mean())
280
281     ()
282
283     toc1 = time.time()
284     train_loss.append(sum(train_loss_batch) / len(train_loss_batch))
285     train_acc.append(sum(train_acc_batch) / len(train_acc_batch))
286
287     # validating
288     validation_loss_batch, validation_acc_batch = [], []
289
290     state = state.replace(apply_fn=validation_model.apply)
291
292     tic2 = time.time()
293     for x_val, y_val in validation_loader:
294         p_key_next, p_key = jax.random.split(p_key_next)
295
296         d_key_next, d_key = jax.random.split(d_key_next)
297
298         rngs = {'params': shard_prng_key(p_key), 'dropout': shard_prng_key(d_key)}
299
300         validation_batch = [x_val, y_val]
301         validation_batch = shard(validation_batch)
302
303         metrics = jax.pmap(predict, axis_name="batch")(state, validation_batch, rngs)
304
305         validation_loss_batch.append(metrics['loss'].mean())
306         validation_acc_batch.append(metrics['accuracy'].mean())
307
308     ()
309
310     toc2 = time.time()
311     validation_loss.append(sum(validation_loss_batch) / len(validation_loss_batch))
312     validation_acc.append(sum(validation_acc_batch) / len(validation_acc_batch))
313
314     etc = (toc1 - tic1) + (toc2 - tic2)
315     history.append(form.format(epoch, train_loss[-1], train_acc[-1], validation_loss[-1], validation_acc[-1], etc))
316
317     printf(history[-1])
318
319     state = jax_utils.unreplicate(state)
320
321     # Save model weights for the dataset
322
323     save_model(state, f'{WEIGHTS_BASE_PATH}{name}_multi_{n_layer}_{max_seq_len}.h5')
324
325     save_history(f'{WEIGHTS_BASE_PATH}{name}_multi_{n_layer}_{max_seq_len}.csv', history=history)

```

```

317 def test(batch_size, max_seq_len, features=512, embed_size=
256, lr=1e-3):
318     batch_size = batch_size * jax.device_count()
319     vocab_size = 256 + 1
320     n_layer = 1
321     output_size = 2
322     dropout_rate = 0.1
323     name = 'hrrformer'
324
325     print()
326     printf('Total batch size: ' + str(batch_size) + ', Max
seq len: ' + str(max_seq_len))
327     printf("loading dataset for testing...")
328
329     # load dataset
330     test_loader, num_test_samples = load_dataset(batch_size
=32,
331                                                    max_seq_le
n=max_seq_len,
332                                                    shuffle=Tr
ue,
333                                                    num_worker
s=0)
334
335     printf("build and initialize network")
336
337     test_model = Network(features=features,
338                           vocab_size=vocab_size,
339                           embed_size=embed_size,
340                           max_seq_len=max_seq_len,
341                           nlayers=n_layer,
342                           output_size=output_size,
343                           dropout_rate=dropout_rate,
344                           training=False)
345
346     p_key_next, p_key = jax.random.split(jax.random.PRNGKey
(0))
347     d_key_next, d_key = jax.random.split(jax.random.PRNGKey
(0))
348
349     history = []
350     test_loss, test_acc = [], []
351
352     init_rngs = {'params': p_key, 'dropout': d_key}
353     printf("initialize_model")
354     params = initialize_model(model=test_model, input_size=
max_seq_len, init_rngs=init_rngs)
355
356     printf("optimizer and scheduler")
357
358     # optimizer and scheduler
359     steps = 800_000 // batch_size
360     scheduler = optax.exponential_decay(init_value=lr,
361                                         transition_steps=st
eps,
362                                         decay_rate=.85,
363                                         transition_begin=1,
364                                         end_value=1e-5)
365
366     tx = optax.adam(learning_rate=scheduler)
367
368     state = train_state.TrainState.create(apply_fn=test_mod
el.apply, params=params, tx=tx)
369     state = load_model(state,
370                       f'{WEIGHTS_BASE_PATH}{name}_multi_{n
_layer}_{max_seq_len}.h5')
371     state = jax_utils.replicate(state)
372
373     form = 'test loss: {3:>8.6f}, test accuracy: {4:>5.2
f}%, etc: {5:>.2f}s'
374
375     # test
376     test_loss_batch, test_acc_batch = [], []
377     state = state.replace(apply_fn=test_model.apply)
378
379     tic2 = time.time()
380     for x_test, y_test in test_loader:

```

```

381         p_key_next, p_key = jax.random.split(p_key_next)
382         d_key_next, d_key = jax.random.split(d_key_next)
383         rngs = {'params': shard_prng_key(p_key), 'dropout':
shard_prng_key(d_key)}
384
385         test_batch = [x_test, y_test]
386         test_batch = shard(test_batch)
387
388         metrics = jax.pmap(predict, axis_name="batch")(state, test_batch, rngs)
389
390         test_loss_batch.append(metrics['loss'].mean())
391         test_acc_batch.append(metrics['accuracy'].mean())
392
393         toc2 = time.time()
394         test_loss.append(sum(test_loss_batch) / len(test_loss_batch))
395         test_acc.append(sum(test_acc_batch) / len(test_acc_batch))
396
397         etc = (toc2 - tic2)
398         history.append(form.format(0, test_loss[-1], test_acc[-1], test_loss[-1], test_acc[-1], etc))
399         printf(history[-1])
400
401
402     def printf(message):
403         timestamp = datetime.datetime.now()
404         timestamp_str = timestamp.strftime("%Y-%m-%d %H:%M:%S")
405         print(f"[{timestamp_str}] {message}")
406
407
408     def individual_predict(batch_size, max_seq_len, features=512, embed_size=256, lr=1e-3):
409         global prediction_result
410         prediction_result = "None"
411         vocab_size = 256 + 1
412         n_layer = 1
413         output_size = 2
414         dropout_rate = 0.1
415         name = 'hrrformer'
416
417         print()
418         printf('Total batch size: ' + str(batch_size) + ', Max seq len: ' + str(max_seq_len))
419         printf("loading dataset for individual prediction analysis...")
420
421         # load dataset
422         test_loader, num_test_samples = load_dataset(batch_size=1,
max_seq_len=max_seq_len,
shuffle=True,
num_workers=0)
423
424         printf(f"Number of samples to predict: {num_test_samples}")
425
426         printf("build and initialize network")
427
428         test_model = Network(features=features,
vocab_size=vocab_size,
embed_size=embed_size,
max_seq_len=max_seq_len,
n_layers=n_layer,
output_size=output_size,
dropout_rate=dropout_rate,
training=False)
429
430         p_key_next, p_key = jax.random.split(jax.random.PRNGKey(0))
431         d_key_next, d_key = jax.random.split(jax.random.PRNGKey(0))
432
433         history = []

```

```

443     test_loss, test_acc = [], []
444
445     init_rngs = {'params': p_key, 'dropout': d_key}
446     printf("initialize_model")
447     params = initialize_model(model=test_model, input_size=
max_seq_len, init_rngs=init_rngs)
448
449     printf("optimizer and scheduler")
450
451     # optimizer and scheduler
452     steps = 800_000 // batch_size
453     scheduler = optax.exponential_decay(init_value=lr,
transition_steps=st
eps,
decay_rate=.85,
transition_begin=1,
end_value=1e-5)
458
459     tx = optax.adam(learning_rate=scheduler)
460
461     state = train_state.TrainState.create(apply_fn=test_mod
el.apply, params=params, tx=tx)
462     state = load_model(state,
463         f'{WEIGHTS_BASE_PATH}{name}_multi_{n
_layer}_{max_seq_len}.h5')
464     state = jax_utils.replicate(state)
465
466     state = state.replace(apply_fn=test_model.apply)
467     test_loss_batch, test_acc_batch = [], []
468
469     for x_test, y_test in test_loader:
470         p_key_next, p_key = jax.random.split(p_key_next)
471         d_key_next, d_key = jax.random.split(d_key_next)
472         rngs = {'params': shard_prng_key(p_key), 'dropout':
shard_prng_key(d_key)}
473
474         test_batch = [x_test, y_test]
475         test_batch = shard(test_batch)
476         metrics = jax.pmap(predict_ind, axis_name="batch")
(state, test_batch, rngs)
477         test_acc_batch.append(metrics['accuracy'].mean())
478
479         test_acc.append(sum(test_acc_batch) / len(test_acc_batc
h))
480
481     printf(test_acc[0])
482     if str(test_acc[0]) == "100.0":
483         return "Correct"
484     return "Wrong"
485
486
487 def print_pred(prediction_result: str, bool_good: bool, fil
ename: str):
488     file_type = ""
489
490     if MALWARE:
491         file_type = "Malware"
492     elif VULNERABILITY:
493         file_type = "Vulnerable"
494
495     if bool_good:
496         if prediction_result == "Wrong":
497             printf("File '" + filename + "' is incorrectly
predicted as " + file_type + " file")
498         else:
499             printf("File '" + filename + "' is correctly pr
edicted as Benign file")
500     else:
501         if prediction_result == "Wrong":
502             printf("File '" + filename + "' is incorrectly
predicted as Benign file")
503         else:
504             printf("File '" + filename + "' is correctly pr
edicted as " + file_type + " file")
505
506
507 def inidividual_predict_helper():

```



```

293
294
295 if __name__ == '__main__':
296     from math import log2
297
298     seq = [2 ** i for i in range(8, 20)]
299     for seq_len in seq:
300         batch = max(2 ** int(16 - log2(seq_len)), 1)
301         train(batch_size=batch, max_seq_len=seq_len, epochs
=10)

```

```

508     # Algorithm for predict function
509     # 1. load the test data...
510     # 2. for single test file (dict{name->good/bad}):
511     #     predict if it is benign or not (we already know the ans)
512     #     if prediction is failing:
513     #         note down this file details..
514
515     # Batch size and seq_len should be of the best trained model
516     import os
517     import shutil
518
519     # Source directory containing the files you want to process
520     source_directories = [TEST_DATA_PATH_GOOD, TEST_DATA_PATH_BAD]
521
522     if not os.path.exists(TEST_DATA_PATH_GOOD + "temp"):
523         os.mkdir(TEST_DATA_PATH_GOOD + "temp")
524
525     if not os.path.exists(TEST_DATA_PATH_BAD + "temp"):
526         os.mkdir(TEST_DATA_PATH_BAD + "temp")
527
528     # Iterate through files in the source directory
529     for source_directory in source_directories:
530
531         bool_good = True
532         if source_directory.endswith("/good/"):
533             temp_directory = TEST_DATA_PATH_GOOD + "temp"
534         else:
535             bool_good = False
536             temp_directory = TEST_DATA_PATH_BAD + "temp"
537         for filename in os.listdir(source_directory):
538             if not filename == "temp":
539                 source_file = os.path.join(source_directory, filename)
540                 temp_file = os.path.join(temp_directory, filename)
541                 shutil.copy2(source_file, temp_file)
542                 # Perform processing on the file in the temporary directory
543                 prediction_result = individual_predict(batch_size=64, max_seq_len=1024)
544                 print_pred(prediction_result, bool_good, filename)
545                 # After processing, delete the file from the temporary directory
546                 os.remove(temp_file)
547                 os.removedirs(temp_directory)
548
549
550 if __name__ == '__main__':
551
552     from math import log2
553
554     if TRAIN:
555         # seq = [2 ** i for i in range(8, 20)]
556         # for seq_len in seq:
557         #     batch = max(2 ** int(16 - log2(seq_len)), 1)
558
559         #     if batch > 2:
560         #         train(batch_size=128, max_seq_len=512, epochs=10)
561         #         printf("Training completed")
562
563     elif TEST:
564         seq = [2 ** i for i in range(8, 20)]
565         for seq_len in seq:
566             batch = max(2 ** int(16 - log2(seq_len)), 1)
567             if batch > 2 and seq_len <= 1024:
568                 test(batch_size=batch, max_seq_len=seq_len)
569                 printf("Testing completed")
570
571     elif PREDICT:
572         individual_predict_helper()

```

