FiBiNET: Combining Feature Importance and Bilinear feature Interaction for Click-Through Rate Prediction

何荣炜

Abstract

- Problem : Many current works calculate the feature interactions in a simple way such as Hadamard product and inner product and they **care less about the importance of features**.

- FiBiNET
  - the FiBiNET can dynamically learn the importance of features via the Squeeze-Excitation network (**SENET**) mechanism
  - it is able to effectively learn the feature interactions via **bilinear function**.

- As far as we know, different features have various importances for the target task. For example, the feature occupation is more important than the feature hobby when we predict a person’s income. Taking this into consideration, we introduce a Squeeze-and-Excitation network (SENET) to learn the weights of features dynamically.

- Besides, feature interaction is a key challenge in CTR prediction field and many related works calculate the feature interactions in a simple way such as Hadamard product and inner product.
- We propose a new fine-grained way in this paper to calculate the feature interactions with the bilinear function.

RELATED WORK

- Factorization Machine and Its relevant variants (FM && FFM)

- Deep Learning based CTR Models
  - FNN can capture only high-order feature interactions.
  - expertise feature engineering is still needed on the input to the wide part of WDL
  - DeepFM replaces the wide part of WDL with FM and shares the feature embedding between the FM and deep component.
  - xDeepFM) also models the low-order and high-order feature interactions in an explicit way by proposing a novel Compressed Interaction Network (CIN) part.

SENet Module

- The SENET is proved to be successful in image classification tasks and won first place in the ILSVRC 2017 classification task.

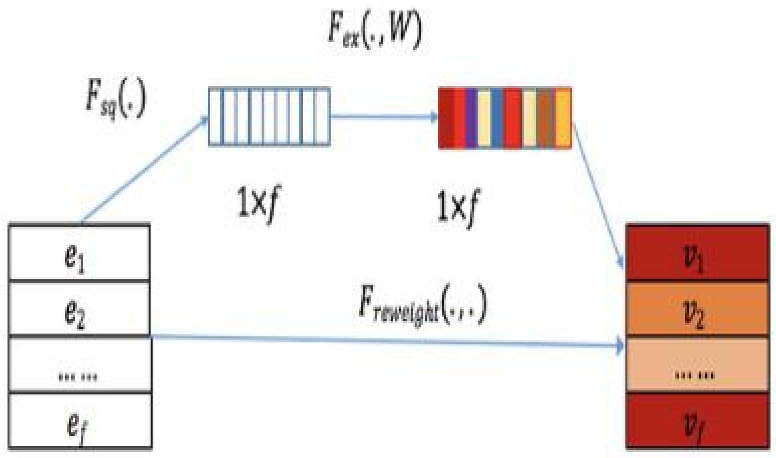
- Sparse Input and Embedding Layer

$E = [e_1, e_2, \cdots, e_i, \cdots, e_f]$, where f denotes the number of fields, $e_i \in R^k$ denotes the embedding of i-th field , and k is the dimension of embedding layer.

- SENET Layer

Using the feature embedding as input, the SENET produces weight vector $A = \{a_1, \cdots, a_i, \cdots, a_f\}$ for field embeddings and then rescales the original embedding E with vector A to get a new embedding (SENET-Like embedding) $V = [v_1, \cdots, v_i, \cdots, v_f]$,

- SENet is comprised of three steps :



- Squeeze
  - Concretely speaking, we use some pooling methods such as max or mean to squeeze the original embedding into a statistic vector Z = [z1,··· ,zi,··· ,zf]

$$z_i = F_{sq}(e_i) = \frac{1}{k}\sum_{t=1}^{k} e_i^{(t)}$$

```
column_num, dimension = _check_fm_columns(params['feature_columns'])
reduction_ratio = params['reduction_ratio']
feature_embeddings = tf.reshape(net, (-1, column_num, dimension))  # (batch_size,column_num, embedding_size)(b,f,k)
original_feature = feature_embeddings
if params['pooling'] == "max":
    feature_embeddings = tf.reduce_max(feature_embeddings, axis=2)   # (b,f) max pooling
else:
    feature_embeddings = tf.reduce_mean(feature_embeddings, axis=2)  #(b,f) mean pooling
```

- Excitation
  - This step can be used to learn the weight of each field embedding based on the statistic vector Z. We use two full connected (FC) layers to learn the weights.
  - The first FC layer is a dimensionality-reduction layer and the second FC layer increases dimensionality

```
reduction_num = max(column_num/reduction_ratio, 1)     # f/r
"""
weight1 = tf.get_variable(name='weight1', shape=[column_num, reduction_num],
                          initializer=tf.glorot_normal_initializer(seed=random.randint(0, 1024)),
                          dtype=tf.float32)
weight2 = tf.get_variable(name='weight2', shape=[reduction_num, column_num],
                           initializer=tf.glorot_normal_initializer(seed=random.randint(0, 1024)),
                           dtype=tf.float32)
"""
att_layer = tf.layers.dense(feature_embeddings, units=reduction_num, activation=tf.nn.relu,
                            kernel_initializer=tf.glorot_uniform_initializer())     # (b, f/r)
att_layer = tf.layers.dense(att_layer, units=column_num, activation=tf.nn.relu,
                              kernel_initializer=tf.glorot_uniform_initializer())  # (b, f)
senet_layer = original_feature * tf.expand_dims(att_layer, axis=-1)    # (b, f, k)
senet_output = tf.layers.flatten(senet_layer)  # (b, f*k)

return senet_output
```

- Re-Weight:
  - It does field-wise multiplication between the original field embedding E and field weight vector A and outputs the new embedding(SENET-Like embedding) V ={v1,···,vi,···,vf}. The SENET-Like embedding V can be calculated as follows:
-

$$V = F_{ReWeight}(A, E) = [a_1 \cdot e_1, \cdots, a_f \cdot e_f] = [v_1, \cdots, v_f] \quad (3)$$

where $a_i \in R$, $e_i \in R^k$, and $v_i \in R^k$.

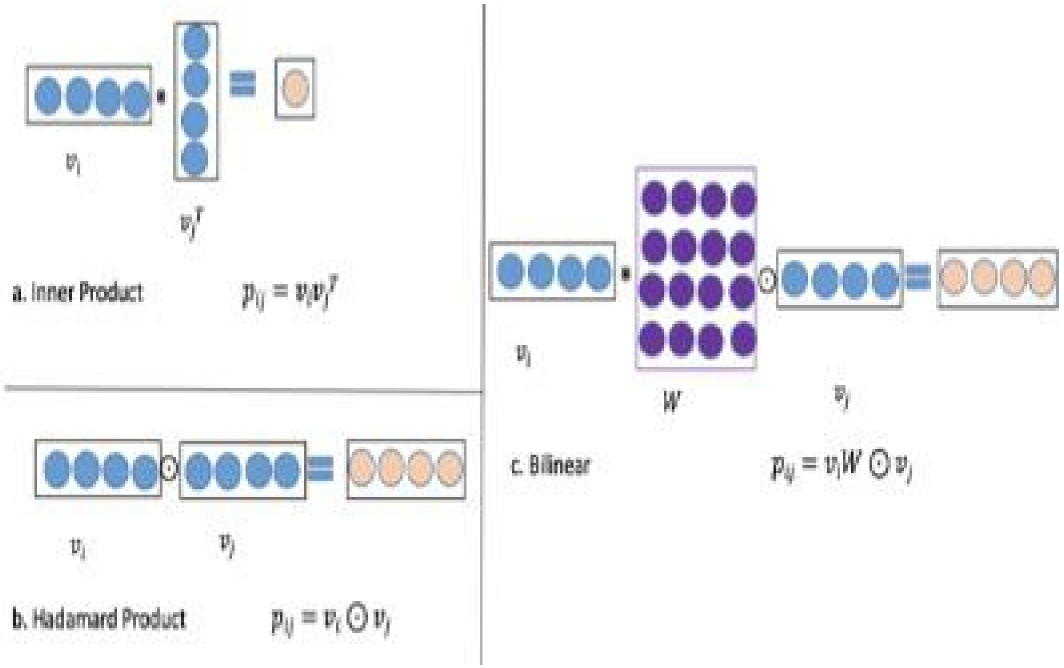
```
senet_layer = original_feature * tf.expand_dims(att_layer, axis=-1)    # (b, f, k)
senet_output = tf.layers.flatten(senet_layer)  # (b, f*k)
```

Bilinear-Interaction Layer

- The forms of inner product and Hadamard product are respectively expressed as {(vi ·vj)xixj}(i,j)∈Rx and {(vi ⊙ vj)xixj}(i,j)∈Rx
  - · denotes the regular inner product,
  - ⊙ denotes the Hadamard product

$$[a_1, a_2, \ldots, a_n] \cdot [b_1, b_2, \ldots, b_n] = \sum_{i=1}^{n} a_i b_i$$

$$[a_1, a_2, \ldots, a_n] \odot [b_1, b_2, \ldots, b_n] = [a_1 b_1, a_2 b_2, \ldots, a_n b_n]$$



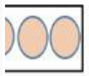
- we propose a more fine-grained method which combines the inner product and Hadamard product to learn the feature interactions

```python
def build_Bilinear_Interaction_layers(net, params):
    # Build Bilinear-Interaction Layer

    column_num, dimension = _check_fm_columns(params['feature_columns'])
    feature_embeddings = tf.reshape(net, (-1, column_num, dimension))  # (batch_size,column_num, embedding_size)(b,f,k)

    element_wise_product_list = []
    count = 0
    for i in range(0, column_num):
        for j in range(i + 1, column_num):
            with tf.variable_scope('weight_', reuse=tf.AUTO_REUSE):
                weight = tf.get_variable(name='weight_' + str(count), shape=[dimension, dimension],
                                         initializer = tf.glorot_normal_initializer(seed = random.randint(0,1024)),
                                         dtype=tf.float32)
            element_wise_product_list.append(
                tf.multiply(tf.matmul(feature_embeddings[:, i, :], weight), feature_embeddings[:, j, :]))
                #tf.multiply(feature_embeddings[:, i, :], feature_embeddings[:, j, :]))
            count += 1
    element_wise_product = tf.stack(element_wise_product_list)  # (f*(f-1)/2,b,k)(把它们组合成一个tensor)
    element_wise_product = tf.transpose(element_wise_product, perm=[1, 0, 2],
                                        name="element_wise_product")  # (b, f*(f-1)/2, k)

    bilinear_output = tf.layers.flatten(element_wise_product)  #(b, f*(f-1)/2*k)
    return bilinear_output
```



W v_j

Combination Layer

- The combination layer concatenates interaction vector p and q and feeds the concatenated vector into the following layer

$$c = F_{concat}(p, q) = [p_1, \cdots, p_n, q_1, \cdots, q_n] = [c_1, \cdots, c_{2n}]$$

```
senet_layer = build_SENET_layers(net, params)
combination_layer = tf.concat([build_Bilinear_Interaction_layers(net, params),
                               build_Bilinear_Interaction_layers(senet_layer, params)], axis=1)
```

Deep Network

- The deep network is comprised of several full-connected layers, which implicitly captures high-order features interactions

$$a^{(l)} = \sigma(W^{(l)} a^{(l-1)} + b^{(l)})$$

```python
def build_deep_layers(net, params):
    # Build the hidden layers, sized according to the 'hidden_units' param.

    for layer_id, num_hidden_units in enumerate(params['hidden_units']):
        net = tf.layers.dense(net, units=num_hidden_units, activation=tf.nn.relu,
                              kernel_initializer=tf.glorot_uniform_initializer())
    return net
```

Output layer

- To summarize, we give the overall formulation of our proposed model' output as

$$\hat{y} = \sigma(w_0 + \sum_{i=0}^{m} w_i x_i + y_d)$$

```
last_layer = build_deep_layers(combination_layer, params)
# head = tf.contrib.estimator.binary_classification_head(loss_reduction=losses.Reduction.SUM)
head = head_lib._binary_logistic_or_multi_class_head(  # pylint: disable=protected-access
    n_classes=2, weight_column=None, label_vocabulary=None, loss_reduction=losses.Reduction.SUM)
logits = tf.layers.dense(last_layer, units=head.logits_dimension,
                         kernel_initializer=tf.glorot_uniform_initializer())
optimizer = tf.train.AdagradOptimizer(learning_rate=params['learning_rate'])
loss = tf.reduce_sum(tf.nn.sigmoid_cross_entropy_with_logits(logits=logits, labels=labels))
train_op = optimizer.minimize(loss, global_step=tf.train.get_global_step())
```

Relationship with FM and FNN.

- Suppose we remove the SENET layer and Bilinear-Interaction layer, it’s not hard to find that our model will be degraded as the FNN.
- When we further remove the DNN part, and at the same time use a constant sum, then the shallow FiBiNET is downgraded to the traditional FM model.

EXPERIMENTS

- (RQ1) How does our model perform as compared to the state-of- the-art methods for CTR prediction?
- (RQ2) Can the different combinations of bilinear and Hadamard functions in Bilinear-Interaction layer impact its performance?
- (RQ3) Can the different field types(Field-All, Field-Each and Field- Interaction) of Bilinear-Interaction layer impact its performance?
- (RQ4) How do the settings of networks influence the performance of our model?
- (RQ5) Which is the most important component in FiBiNET?

(RQ1) How does our model perform as compared to the state-of- the-art methods for CTR prediction?

Table 1: The overall performance of shallow models on Criteo and Avazu datasets. The SE-FM-ALL denotes the shallow model with the Field-All type of Bilinear-Interaction layer.

	Criteo		Avazu	
Model	AUC	Logloss	AUC	Logloss
LR	0.7808	0.4681	0.7633	0.3891
FM	0.7923	0.4584	0.7745	0.3832
FFM	0.8001	0.4525	0.7795	0.3810
AFM	0.7965	0.4541	0.7740	0.3839
SE-FM-All	**0.8021**	**0.4495**	**0.7803**	**0.3800**

Table 2: The overall performance of deep models on Criteo and Avazu datasets. The DeepSE-FM-ALL denotes the deep model with the Field-All type of Bilinear-Interaction layer.

	Criteo		Avazu	
Model	AUC	Logloss	AUC	Logloss
FNN	0.8057	0.4464	0.7802	0.3800
DeepFM	0.8085	0.4445	0.7786	0.3810
DCN	0.7978	0.4617	0.7681	0.3940
XDeepFM	0.8091	0.4461	0.7808	0.3818
DeepSE-FM-All	**0.8103**	**0.4423**	**0.7832**	**0.3786**

(RQ2) Can the different combinations of bilinear and Hadamard functions in Bilinear-Interaction layer impact its performance?

- The '1' denotes that bilinear function is used while 0 means Hadamard product is used.
  - For example, '10' denotes that bilinear function is used as feature interaction method on the original embedding while the Hadamard function is used as feature interaction method on the SENET like embedding.

	Criteo		Avazu	
Combinations	AUC	Logloss	AUC	Logloss
SE-FM_00	0.7989	0.4525	0.7782	0.3818
SE-FM_01	0.8018	0.4500	**0.7797**	0.3808
SE-FM_10	0.8029	0.4488	0.7794	**0.3807**
SE-FM_11	**0.8037**	**0.4479**	0.7770	0.3815
DeepSE-FM-00	**0.8105**	**0.4425**	0.7828	0.3785
DeepSE-FM-01	0.8104	0.4423	**0.7833**	**0.3783**
DeepSE-FM-10	0.8100	0.4427	0.7810	0.3809
DeepSE-FM-11	0.8099	0.4428	0.7805	0.3807

(RQ3) Can the different field types(Field-All, Field-Each and Field-Interaction) of Bilinear-Interaction layer impact its performance?

Table 4: The performance of different field types of Bilinear-Interaction layer.

	Criteo		Avazu	
Field Types	AUC	Logloss	AUC	Logloss
SE-FM-All	0.8021	0.4495	**0.7804**	**0.3800**
SE-FM-Each	0.8037	0.4479	0.7797	0.3812
SE-FM-Interaction	**0.8059**	**0.4460**	0.7785	0.3815
DeepSE-FM-All	0.8103	0.4423	0.7832	0.3786
DeepSE-FM-Each	0.8104	0.4423	**0.7833**	**0.3783**
DeepSE-FM-Interaction	**0.8105**	**0.4421**	0.7828	0.3788

(RQ4) How do the settings of networks influence the performance of our model?

- we change the following hyper-parameters:(1) the dimension of embeddings; (3) the number of neurons per layer in DNN; (4) the depth of DNN.

Table 5: The performance of different embedding sizes on Criteo and Avazu datasets

	Criteo		Avazu	
Embedding-Size	AUC	Logloss	AUC	Logloss
10	**0.8104**	**0.4423**	0.7809	0.3801
20	0.8093	0.4435	0.7810	0.3796
30	0.8071	0.4460	0.7812	0.3799
40	0.8071	0.4464	0.7824	0.3790
50	0.8072	0.4468	**0.7833**	**0.3787**

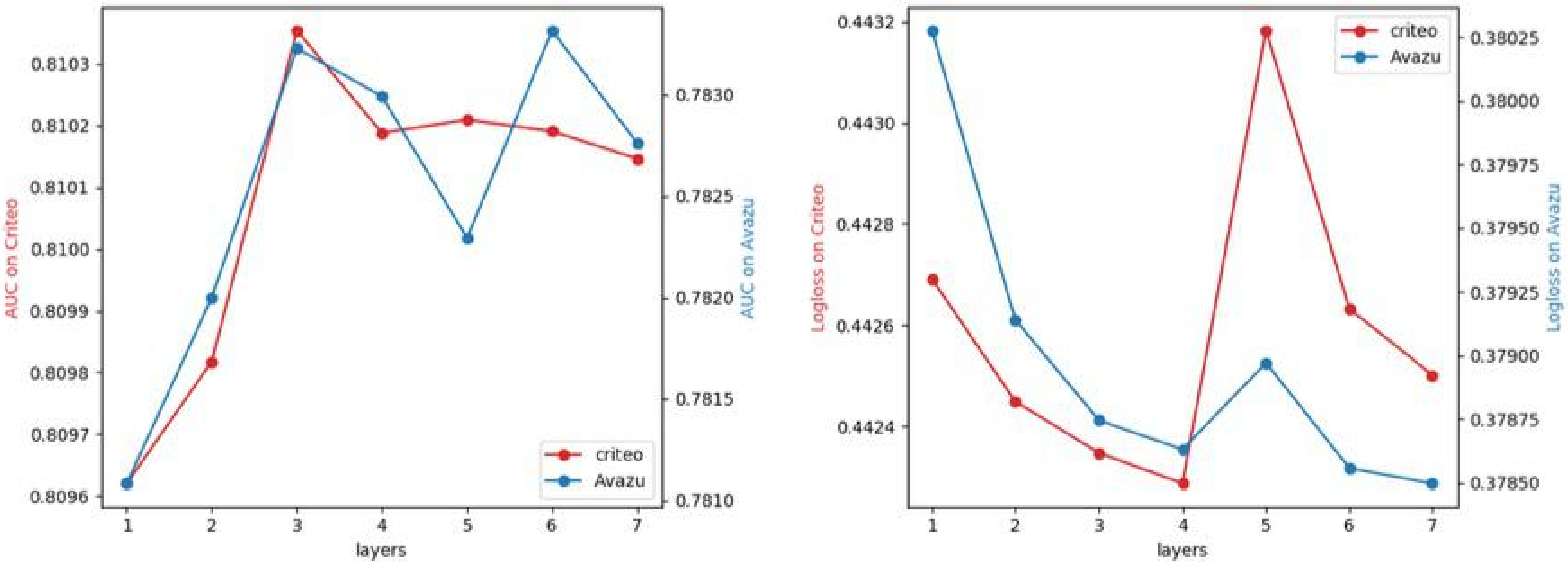


Figure 4: The performance of different number of layers in DNN.

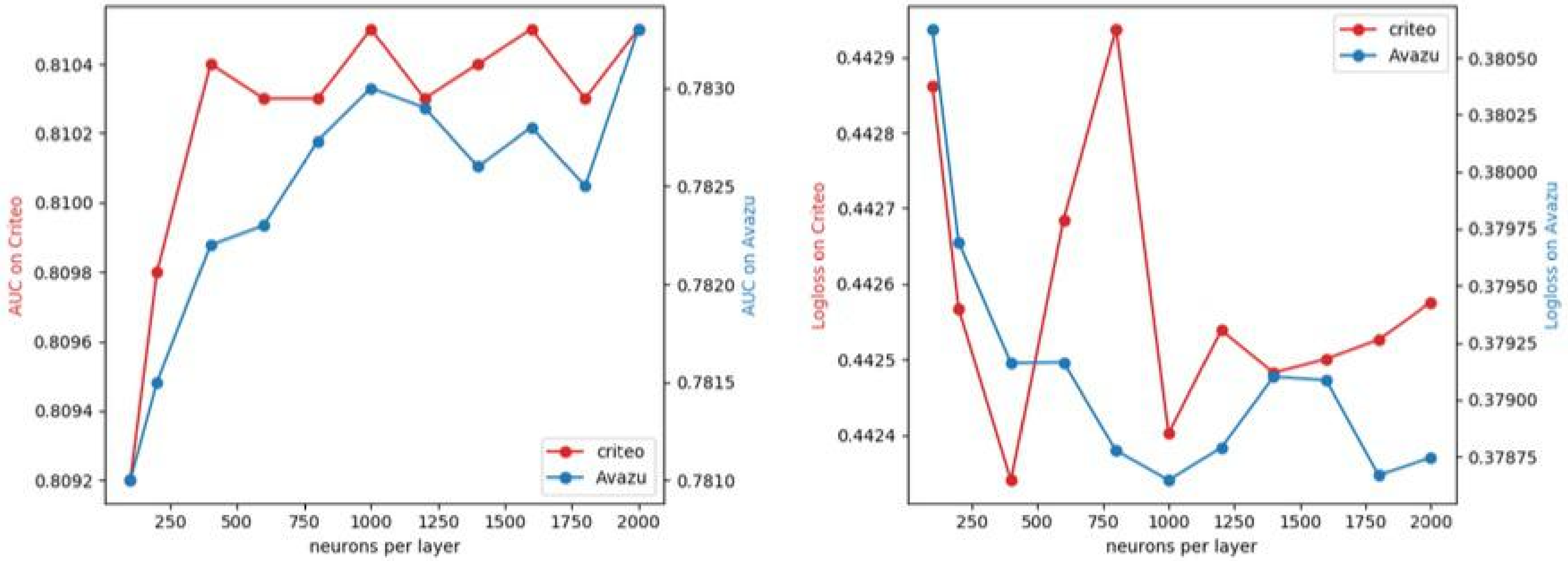


Figure 5: The performance of different number of neurons per layer in DNN

(RQ5) Which is the most important component in FiBiNET?

- Both the Bilinear-Interaction layer and SENET layer are necessary for FiBiNET's performance.
- The Bilinear-Interaction layer is as important as the SENET layer in FiBiNET.

Table 6: The performance of different components in FiBi-NET.

	Criteo		Avazu	
Model	AUC	Logloss	AUC	Logloss
BASE	0.8037	0.4479	0.7797	0.3812
NO-SE	0.7962	0.4552	0.7763	0.3825
NO-BI	0.7986	0.4525	0.7754	0.3829
FM	0.7923	0.4584	0.7745	0.3832
Deep-BASE	0.8104	0.4423	0.7833	0.3783
NO-SE	0.8098	0.4427	0.7822	0.3790
NO-BI	0.8093	0.4435	0.7827	0.3785
FNN	0.8057	0.4464	0.7802	0.3800

Q&A