

Нейронные сети для табличных данных

Agenda

- Factorization Machine
- Deep FM
- TabNet

Factorization Machine

$$\hat{y}(x) := w_0 + \sum_{i=1}^n w_i x_i + \sum_{i=1}^n \sum_{j=i+1}^n \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j$$

$$w_0 \in R, w \in R^n, \mathbf{v} \in R^{n \times k}$$

- w_0 - смещение (bias);
- w_i - определяет влияние каждой характеристики по отдельности;
- $\langle \mathbf{v}_i, \mathbf{v}_j \rangle$ - попарное взаимодействие двух характеристик (второго порядка). То есть вместо того, чтобы сделать один вес для каждой пары, модель делает именно факторизацию обучаемых параметров каждой характеристики и по отдельности строит связи.

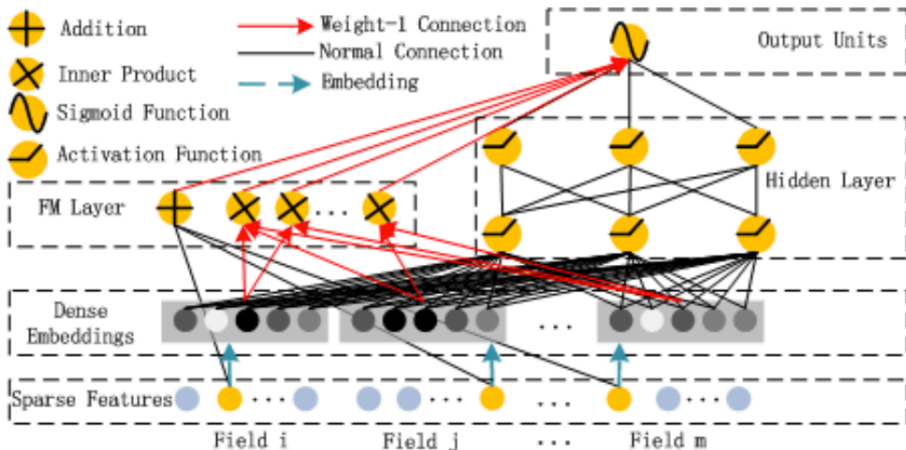
<https://www.csie.ntu.edu.tw/~b97053/paper/Rendle2010FM.pdf>

Factorization Machine

Сложность вычислений для второго порядка $O(kn)$

$$\begin{aligned}& \sum_{i=1}^n \sum_{j=i+1}^n \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j \\&= \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j - \frac{1}{2} \sum_{i=1}^n \langle \mathbf{v}_i, \mathbf{v}_i \rangle x_i x_i \\&= \frac{1}{2} \left(\sum_{i=1}^n \sum_{j=1}^n \sum_{f=1}^k v_{i,f} v_{j,f} x_i x_j - \sum_{i=1}^n \sum_{f=1}^k v_{i,f} v_{i,f} x_i x_i \right) \\&= \frac{1}{2} \sum_{f=1}^k \left(\left(\sum_{i=1}^n v_{i,f} x_i \right) \left(\sum_{j=1}^n v_{j,f} x_j \right) - \sum_{i=1}^n v_{i,f}^2 x_i^2 \right) \\&= \frac{1}{2} \sum_{f=1}^k \left(\left(\sum_{i=1}^n v_{i,f} x_i \right)^2 - \sum_{i=1}^n v_{i,f}^2 x_i^2 \right)\end{aligned}$$

Deep Factorization Machine (DeepFM)



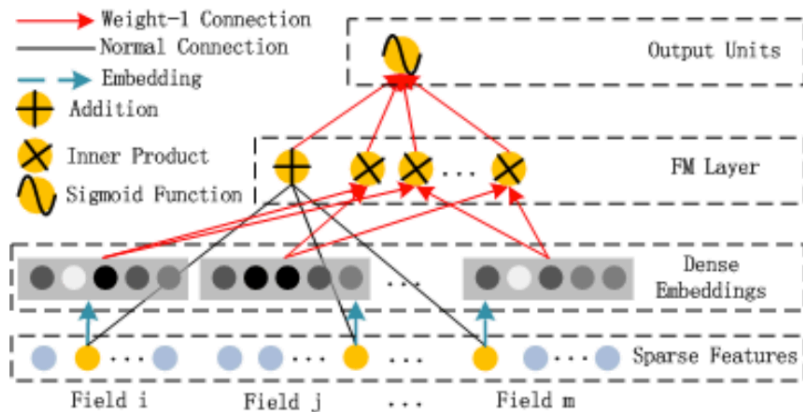
<https://arxiv.org/pdf/1703.04247.pdf>

Deep Factorization Machine (DeepFM)

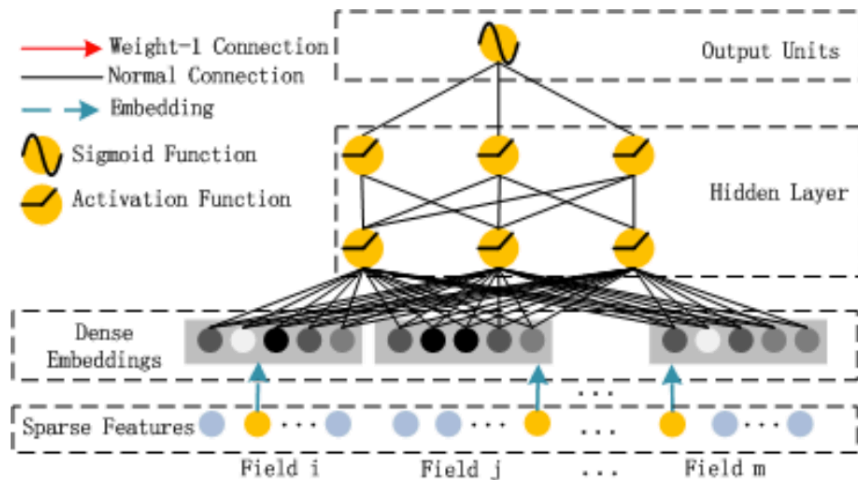
Модель учит объяснять таргет на основе "low- and high-order feature interactions".

$$\hat{y} = \textit{sigmoid}(y_{FM} + y_{DNN})$$

FM component



Deep component



Deep component

Embedding Layer:

$$a^{(0)} = [e_1, e_2, \dots, e_m],$$

Hidden Layer:

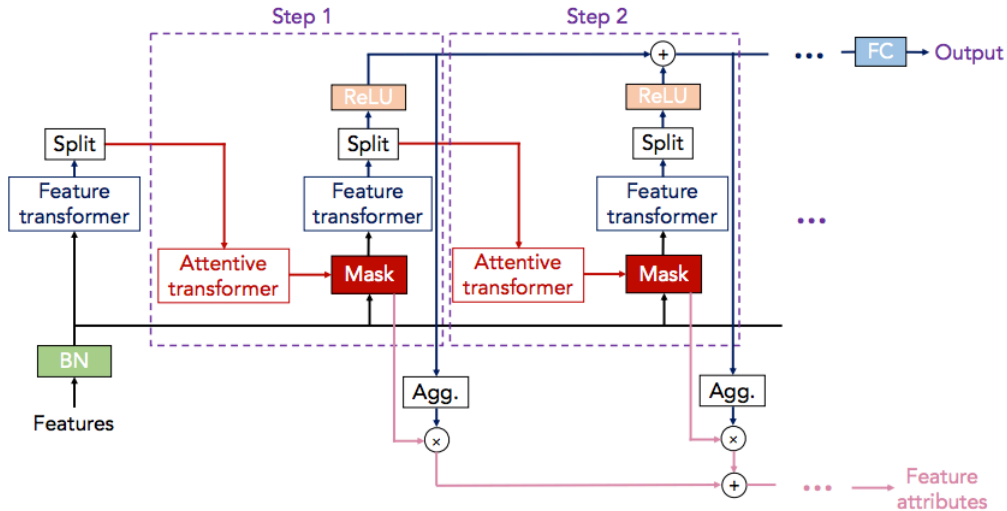
$$a^{(l+1)} = \textit{sigmoid}(W^{(l)}a^{(l)} + b^{(l)})$$

Выход:

$$y_{DNN} = \textit{sigmoid}(W^{|H|+1}a^H + b^{|H|+1}),$$

где $|H|$ - кол-во скрытых слоев.

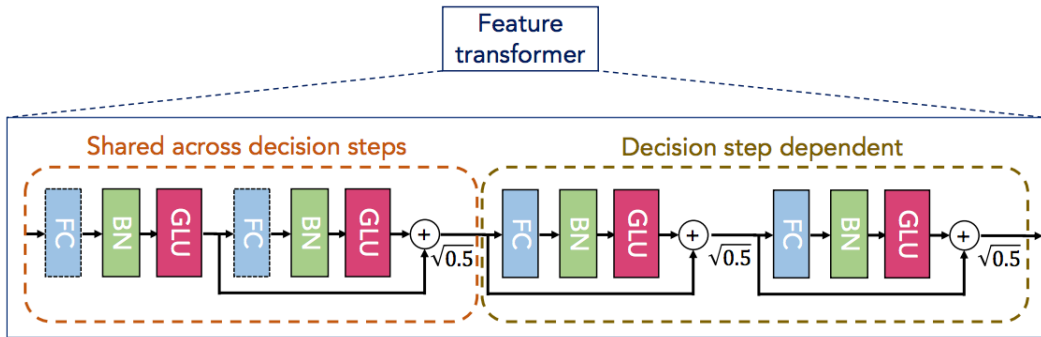
TabNet



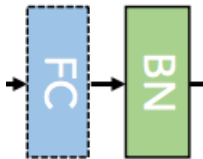
<https://arxiv.org/pdf/1908.07442.pdf>

Предлагает нейронную сеть с последовательным механизмом внимания для интерпретации результата.

Объединяем полносвязные слои в блоки



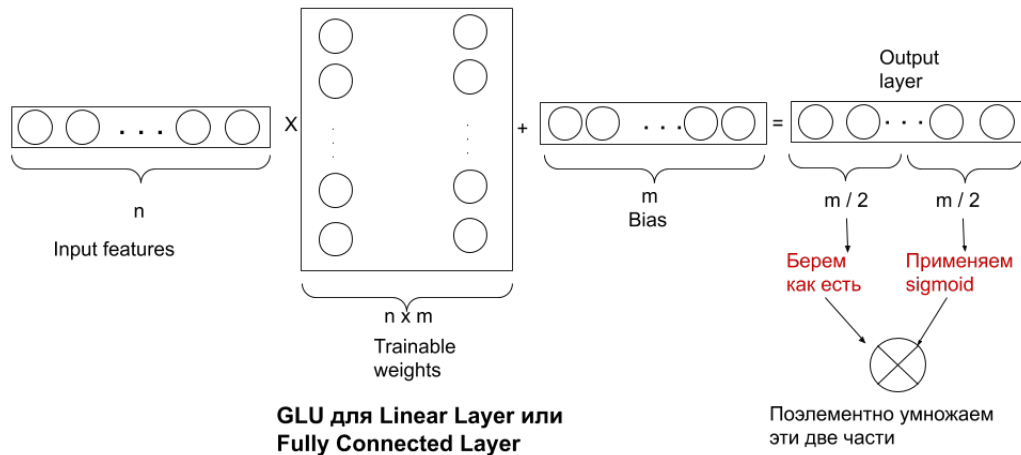
Linear + Batch Norm



`torch.nn.Linear`

`torch.nn.BatchNorm1d`

GLU -- Gated Linear Unit



Выходной тензор из **Feature Transformer** делим на две части:

$$[d[i], a[i]] = f_i(M[i] \cdot f)$$

$$d[i] \in R^{B \times N_d}$$

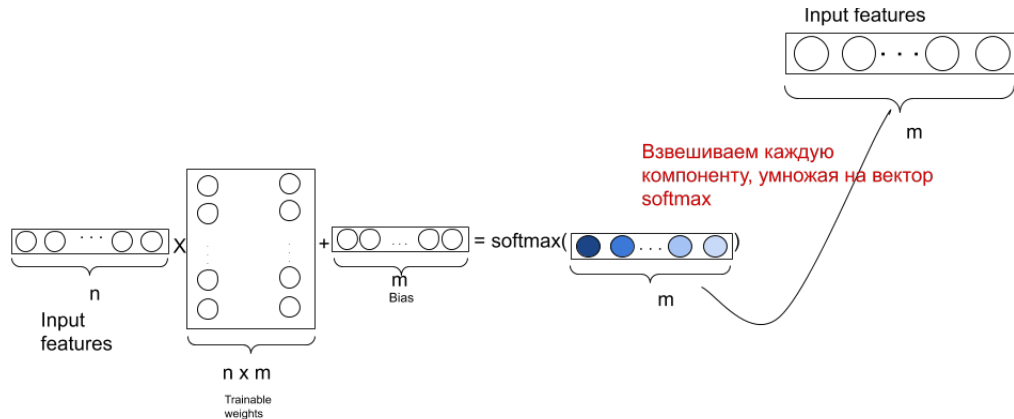
и

$$a[i] \in R^{B \times N_a}$$

Первая часть $d[i]$ идет на агрегацию с другими выходами от каждого шага.

Вторая часть $a[i]$ идет для вычисления маски -- в **Attentive Transformer**.

Механизм внимания



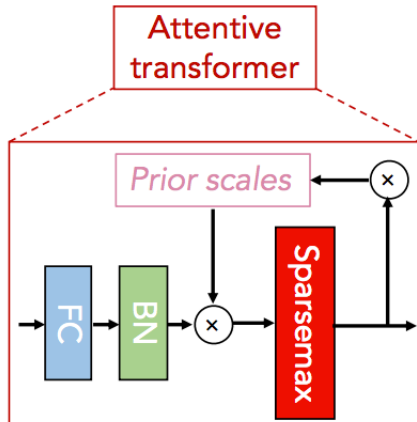
Attention Layer

Механизм внимания

Влияние механизма внимания происходит при backpropagation --
обновлении весов.

Так как происходит произведение на тензор, следовательно, каждый соответствующий вес, который участвовал в вычислении тензора, получает свой дополнительный множитель.

Attentive Transformer - Механизм внимания в TabNet



Sparsemax аналог Softmax

Sparsemax исправляет недостаток *Softmax*: *Softmax* никогда не примет значение ноль.

И для тех случаев, где нужно получить разреженное распределение вероятности, *Softmax* не подходит.

Идея заключается в том, чтобы найти такое пороговое значение, которое бы позволило обнулить некоторые значения и оставить отличные от нуля другие.

Sparsemax

$$\text{sparsemax}_i(\mathbf{z}) = [z_i - \tau(\mathbf{z})]_+$$

$\tau(\mathbf{z})$ - пороговая функция, которая определяет, что обнулить, а что оставить.

Algorithm 1 Sparsemax Evaluation

Input: \mathbf{z}

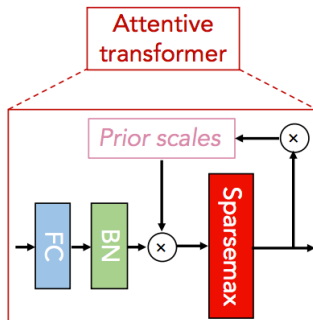
Sort \mathbf{z} as $z_{(1)} \geq \dots \geq z_{(K)}$

Find $k(\mathbf{z}) := \max \left\{ k \in [K] \mid 1 + kz_{(k)} > \sum_{j \leq k} z_{(j)} \right\}$

Define $\tau(\mathbf{z}) = \frac{(\sum_{j \leq k(\mathbf{z})} z_{(j)}) - 1}{k(\mathbf{z})}$

Output: \mathbf{p} s.t. $p_i = [z_i - \tau(\mathbf{z})]_+.$

Механизм внимания в TabNet. Prior Scale



$$M[i] = \text{sparsemax}(P[i-1] \cdot h_i(a[i-1])),$$

где i - номер шага, $h_i(a[i-1])$ выход после BatchNorm, $P[i-1]$ - Prior Scale

Prior Scale - как часто использовалась характеристика до текущего шага

$$P[i] = \prod_{j=1}^i (\gamma - M[j])$$

, где γ - параметр релаксации,
с увеличением γ характеристике будет придаваться бОльший вес.