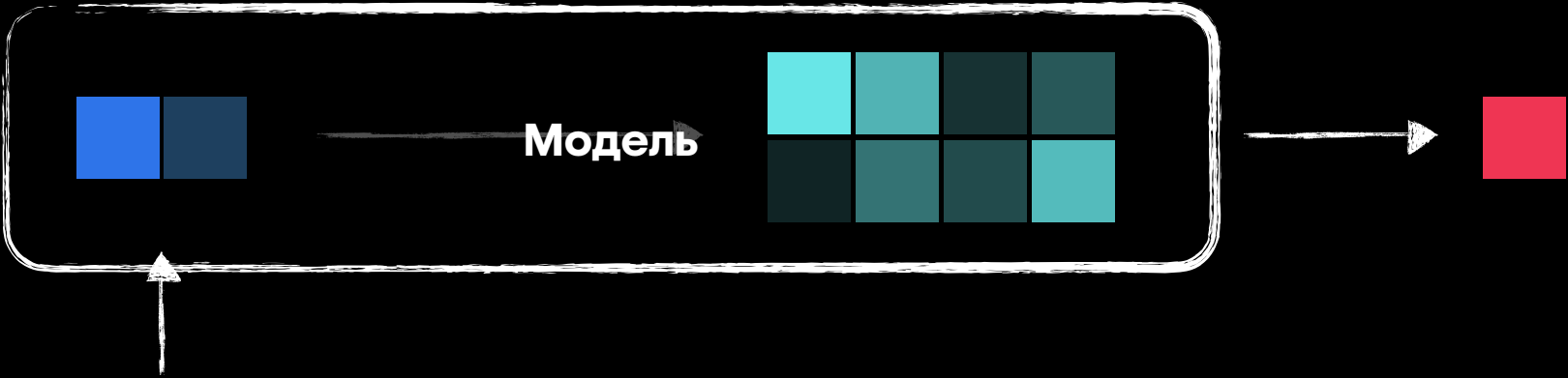


Архитектуры для работы с последовательностями

Обучение представлений данных



QVQLVE...
DIQLTQ...
TVPPMV...

Последовательности



Изображения

Токены и их эмбединги

G A T T A C A

Токены и их эмбединги

G A T T A C A <END>

Токены — смысловые единицы последовательности
(слова, части слов, символы)

Токены и их эмбединги

Эмбединги токенов:
обучаемые векторные представления

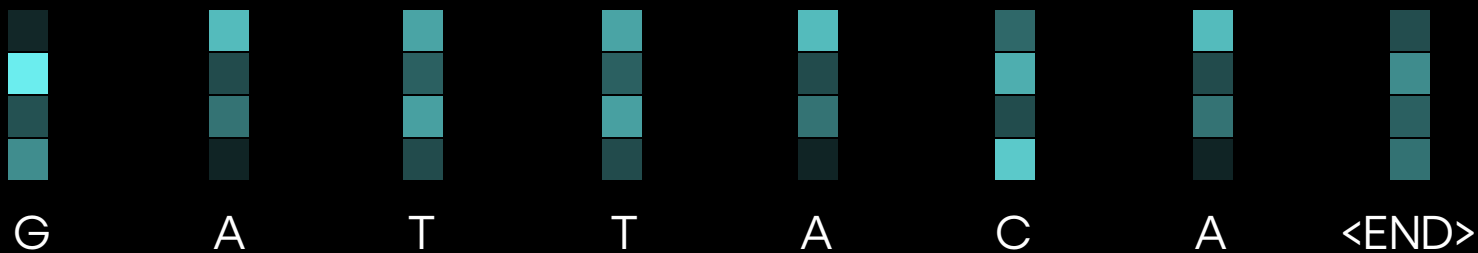
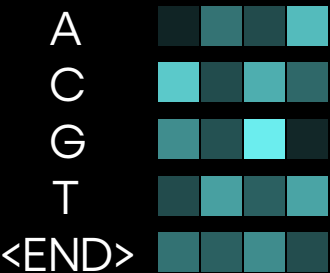


G A T T A C A <END>

Токены — смысловые единицы последовательности
(слова, части слов, символы)













Токены и их эмбединги

Эмбединги токенов:
обучаемые векторные представления



Токены — смысловые единицы последовательности
(слова, части слов, символы)

Рекуррентная сеть (RNN)

A				
C				
G				
T				
<END>				



G



A



T



T



A



C



A



<END>

Рекуррентная сеть (RNN)

Веса рекуррентной
ячейки

$h^{(t+1)}$

=

\tanh

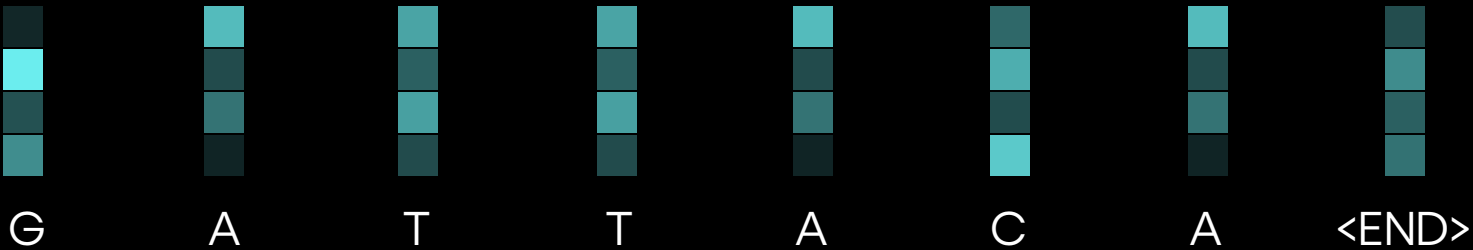
$(W_1 h^{(t)} + W_2 x^{(t)})$

Следующее
состояние

Текущее
состояние

Эмбединг
токена

A	
C	
G	
T	
<END>	



Рекуррентная сеть (RNN)

Веса рекуррентной
ячейки

$h^{(t+1)}$

=

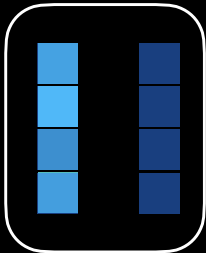
\tanh

$(W_1 h^{(t)} + W_2 x^{(t)})$

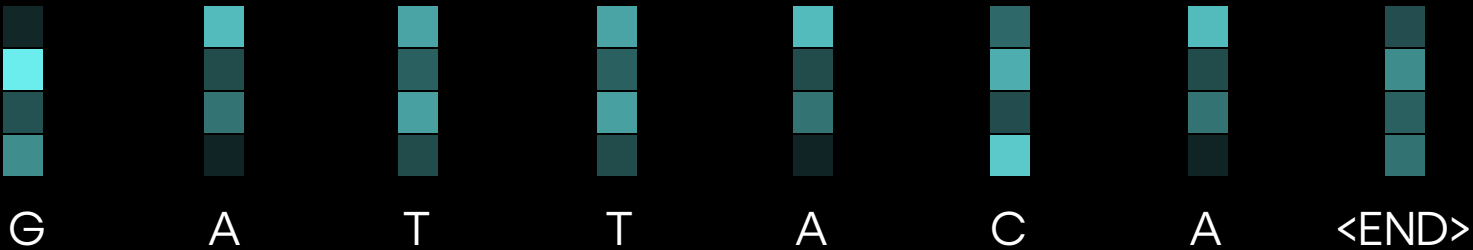
Следующее
состояние

Текущее
состояние

Эмбединг
токена



A	
C	
G	
T	
<END>	



Рекуррентная сеть (RNN)

Веса рекуррентной
ячейки

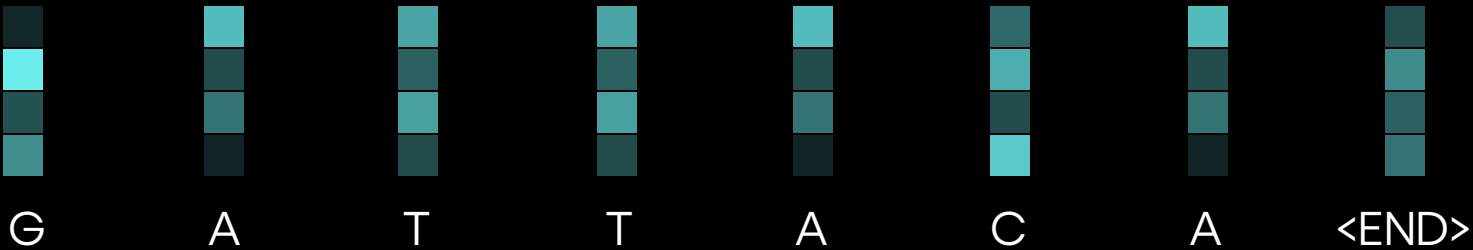
$$h^{(t+1)} = \tanh \left(\overbrace{W_1 h^{(t)} + W_2 x^{(t)}} \right)$$

Следующее
состояние

Текущее
состояние

Эмбединг
токена

A	
C	
G	
T	
<END>	



Рекуррентная сеть (RNN)

Веса рекуррентной
ячейки

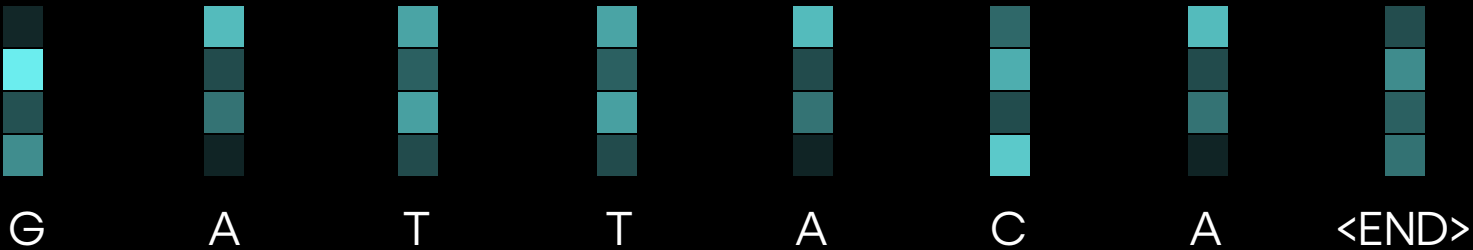
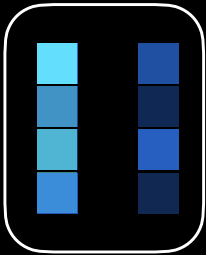
$$h^{(t+1)} = \tanh \left(\overbrace{W_1 h^{(t)} + W_2 x^{(t)}} \right)$$

Следующее
состояние

Текущее
состояние

Эмбединг
токена

A	
C	
G	
T	
<END>	



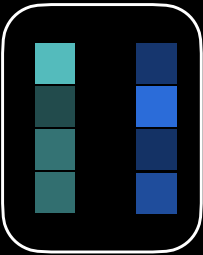
Рекуррентная сеть (RNN)

Веса рекуррентной
ячейки

$$h^{(t+1)} = \tanh \left(\overbrace{W_1 h^{(t)} + W_2 x^{(t)}}^{\text{Веса рекуррентной ячейки}} \right)$$

Следующее состояние Текущее состояние Эмбединг токена

A	
C	
G	
T	
<END>	



G



A



T



T



A



C



A



<END>

Рекуррентная сеть (RNN)

Веса рекуррентной
ячейки

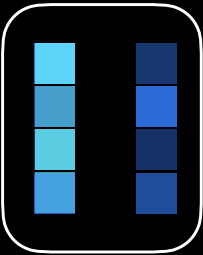
$$h^{(t+1)} = \tanh \left(\overbrace{W_1 h^{(t)} + W_2 x^{(t)}} \right)$$

Следующее
состояние

Текущее
состояние

Эмбединг
токена

A	
C	
G	
T	
<END>	



G



A



T



T



A



C



A



<END>

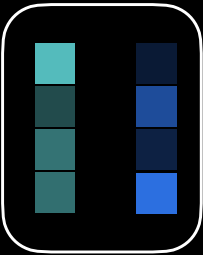
Рекуррентная сеть (RNN)

Веса рекуррентной
ячейки

$$h^{(t+1)} = \tanh \left(\overbrace{W_1 h^{(t)} + W_2 x^{(t)}}^{\text{Веса рекуррентной ячейки}} \right)$$

Следующее состояние Текущее состояние Эмбе́ддинг токена

A	
C	
G	
T	
<END>	



G



A



T



T



A



C



A



<END>

Рекуррентная сеть (RNN)

Веса рекуррентной
ячейки

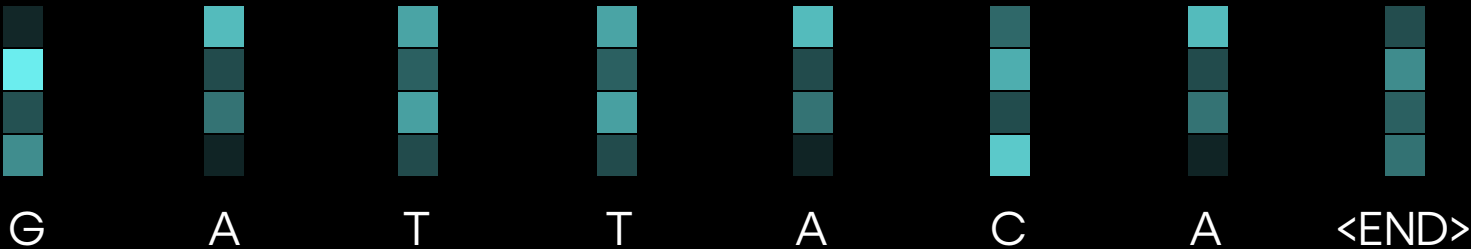
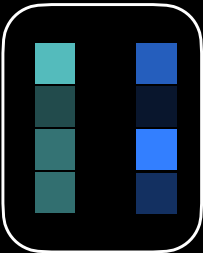
$$h^{(t+1)} = \tanh \left(\overbrace{W_1 h^{(t)} + W_2 x^{(t)}} \right)$$

Следующее
состояние

Текущее
состояние

Эмбединг
токена

A	
C	
G	
T	
<END>	



Рекуррентная сеть (RNN)

Веса рекуррентной
ячейки

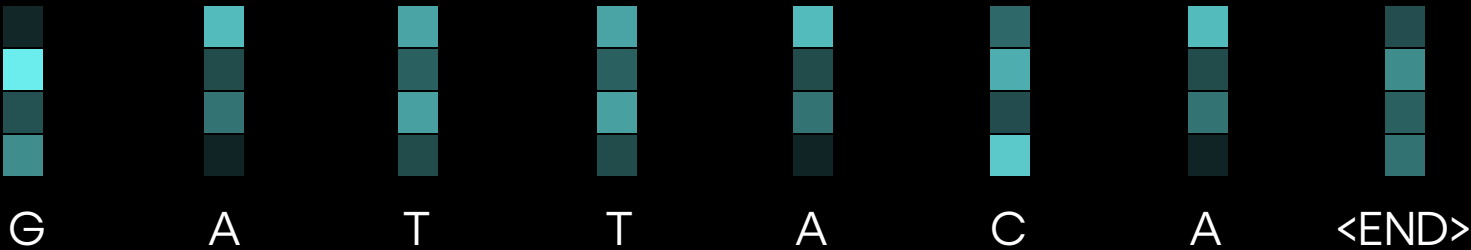
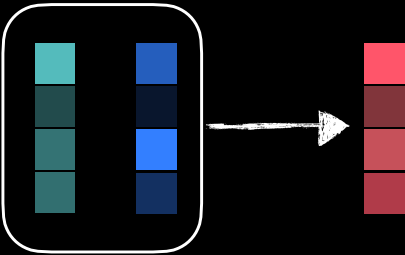
$$h^{(t+1)} = \tanh \left(\overbrace{W_1 h^{(t)} + W_2 x^{(t)}} \right)$$

Следующее
состояние

Текущее
состояние

Эмбединг
токена

A	
C	
G	
T	
<END>	



Генерация последовательности

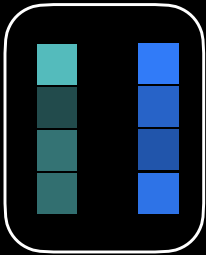
Веса рекуррентной
ячейки

$$h^{(t+1)} = \tanh \left(\overbrace{W_1 h^{(t)} + W_2 x^{(t)}} \right)$$

Следующее
состояние

Текущее
состояние

Эмбединг
токена



A				
C				
G				
T				
<END>				

Генерация последовательности

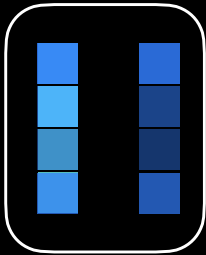
Веса рекуррентной
ячейки

$$h^{(t+1)} = \tanh \left(\overbrace{W_1 h^{(t)} + W_2 x^{(t)}} \right)$$

Следующее
состояние

Текущее
состояние

Эмбединг
токена



A				
C				
G				
T				
<END>				

Генерация последовательности

Веса рекуррентной
ячейки

$h^{(t+1)}$

=

\tanh

$\left(W_1 h^{(t)} + W_2 x^{(t)} \right)$

Следующее
состояние

Текущее
состояние

Эмбединг
токена



G

A				
C				
G				
T				
<END>				

Генерация последовательности

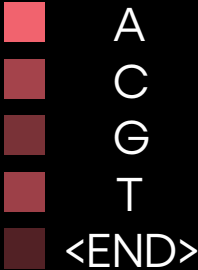
Веса рекуррентной
ячейки

$$h^{(t+1)} = \tanh \left(\overbrace{W_1 h^{(t)} + W_2 x^{(t)}} \right)$$

Следующее
состояние

Текущее
состояние

Эмбединг
токена



Генерация последовательности

Веса рекуррентной
ячейки

$$h^{(t+1)} = \tanh \left(\overbrace{W_1 h^{(t)} + W_2 x^{(t)}}^{\text{Веса рекуррентной ячейки}} \right)$$

Следующее состояние Текущее состояние Эмбединг токена



G



A

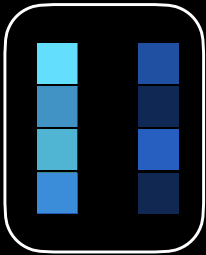
A	
C	
G	
T	
<END>	

Генерация последовательности

Веса рекуррентной
ячейки

$$h^{(t+1)} = \tanh \left(\overbrace{W_1 h^{(t)} + W_2 x^{(t)}} \right)$$

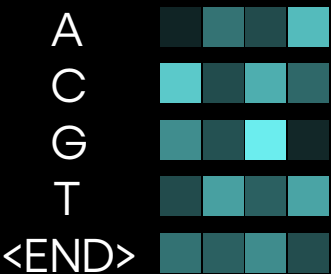
Следующее
состояние Текущее
состояние Эмбединг
токена



G



A



Генерация последовательности















Веса рекуррентной
ячейки

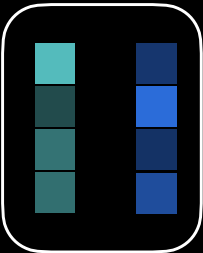
$$h^{(t+1)} = \tanh \left(\overbrace{W_1 h^{(t)} + W_2 x^{(t)}} \right)$$

Следующее
состояние

Текущее
состояние

Эмбединг
токена

A	   
C	   
G	   
T	   
<END>	   



G



A

Генерация последовательности

Веса рекуррентной
ячейки

$h^{(t+1)}$

=

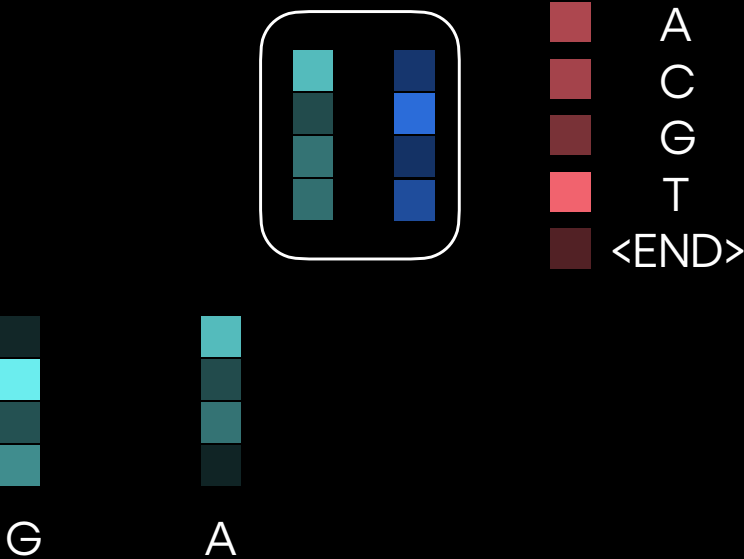
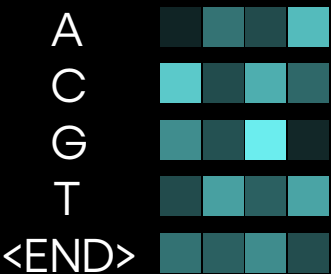
\tanh

$\left(W_1 h^{(t)} + W_2 x^{(t)} \right)$

Следующее
состояние

Текущее
состояние

Эмбединг
токена



A

C

G

T

<END>

G











A

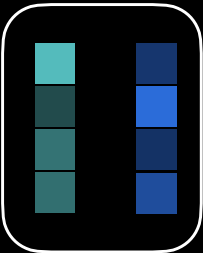
Генерация последовательности

Веса рекуррентной
ячейки

$$h^{(t+1)} = \tanh \left(\overbrace{W_1 h^{(t)} + W_2 x^{(t)}} \right)$$

Следующее
состояние Текущее
состояние Эмбединг
токена

A	   
C	   
G	   
T	   
<END>	   



Генерация последовательности

Веса рекуррентной
ячейки

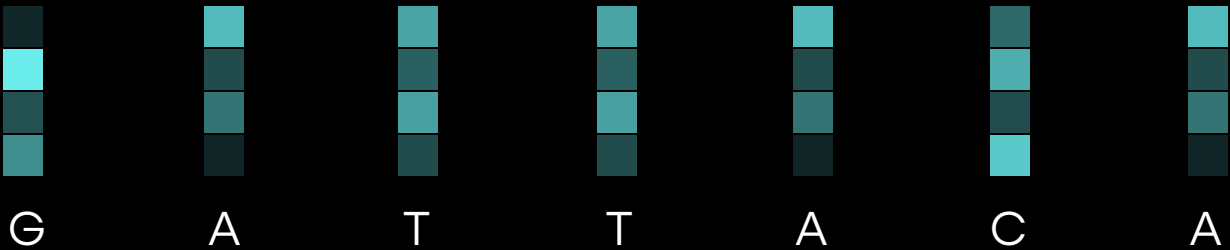
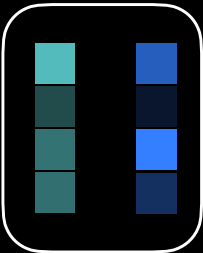
$$h^{(t+1)} = \tanh \left(\overbrace{W_1 h^{(t)} + W_2 x^{(t)}} \right)$$

Следующее
состояние

Текущее
состояние

Эмбединг
токена

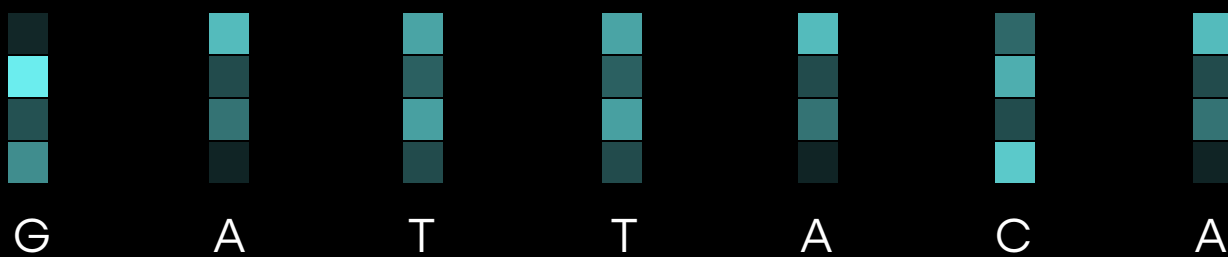
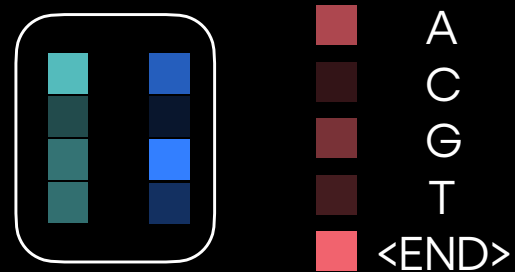
A				
C				
G				
T				
<END>				



Генерация последовательности

$$h^{(t+1)} = \tanh \left(\overbrace{\mathbf{W}_1 h^{(t)} + \mathbf{W}_2 x^{(t)}}^{\text{Веса рекуррентной ячейки}} \right)$$

Следующее состояние Текущее состояние Эмбединг токена



Генерация последовательности

Веса рекуррентной
ячейки

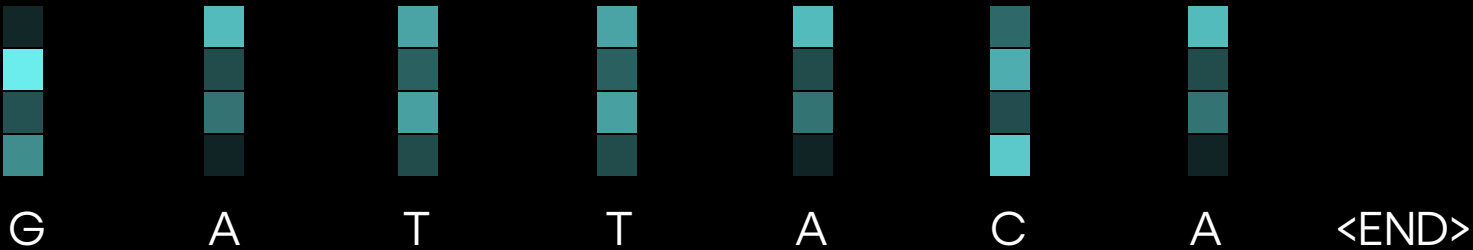
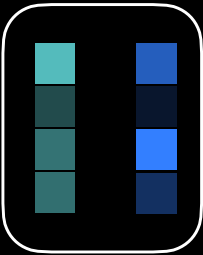
$$h^{(t+1)} = \tanh \left(\overbrace{W_1 h^{(t)} + W_2 x^{(t)}} \right)$$

Следующее
состояние

Текущее
состояние

Эмбединг
токена

A				
C				
G				
T				
<END>				



Рекуррентная сеть на pytorch

Веса рекуррентной
ячейки

$$h^{(t+1)} = \tanh \left(\overbrace{\mathbf{W}_1 h^{(t)} + \mathbf{W}_2 x^{(t)}} \right)$$

Следующее Текущее Эмбеddинг
состояние состояние токена

```
class RNNCell(nn.Module):
    def __init__(self, input_dim: int, hidden_dim: int):
        super().__init__()
        self.linear = nn.Linear(
            input_dim + hidden_dim, hidden_dim
        )

    def forward(self, x: Tensor, h: Tensor) -> Tensor:
        h = torch.cat([x, h], dim=1)
        h = self.linear(h)
        return F.tanh(h)
```

```
class RNN(nn.Module):
    def __init__(self, vocab_size: int, hidden_dim: int) -> None:
        super().__init__()
        self.embed = nn.Embedding(vocab_size, hidden_dim)
        self.init_h = nn.Parameter(data=torch.randn(1, hidden_dim))
        self.rnn = RNNCell(hidden_dim, hidden_dim)
        self.lm_head = nn.Linear(hidden_dim, vocab_size)

    def forward(self, x: Tensor) -> Tensor:
        # x: B x T
        # embed(x): B x T -> B x T x hidden_dim
        B, T = x.shape

        x = self.embed(x) # B x T x hidden_dim
        h = self.init_h.expand((B, -1)) # B x hidden_dim

        logits = [] # T x B x V
        for t in range(T):
            xt = x[:, t, :]
            h = self.rnn.forward(xt, h) # B x hidden
            y = self.lm_head(h).unsqueeze(1) # B x 1 x hidden
            logits.append(y)
            # save prediction for step t + 1

        # lm_head: B x T x hidden -> B x T x V
        return torch.cat(logits, dim=1)
```

Выучивание дальних и разреженных зависимостей

$\mathbf{X} \sim (n, d)$ — последовательность из n токенов размерности d

Выучивание дальних и разреженных зависимостей

$\mathbf{X} \sim (n, d)$ — последовательность из n токенов размерности d

$$\mathbf{h}_i = \sum_{j=1}^{2k+1} \mathbf{w}_j \mathbf{x}_{i+k+1-j} \text{ — 1D-свёртка с размером фильтра } 2k+1$$

Сумма по токенам внутри области видимости

Выучивание дальних и разреженных зависимостей

$\mathbf{X} \sim (n, d)$ — последовательность из n токенов размерности d

$$\mathbf{h}_i = \sum_{j=1}^{2k+1} \mathbf{w}_j \mathbf{x}_{i+k+1-j} \text{ — 1D-свёртка с размером фильтра } 2k+1$$

Сумма по токенам внутри области видимости

$$\mathbf{h}_i = \sum_{j=1}^n g(i-j) \mathbf{x}_j$$

Сумма по всем токенам

Выучивание дальних и разреженных зависимостей

$\mathbf{X} \sim (n, d)$ — последовательность из n токенов размерности d

$$\mathbf{h}_i = \sum_{j=1}^{2k+1} \mathbf{w}_j \mathbf{x}_{i+k+1-j} \text{ — 1D-свёртка с размером фильтра } 2k+1$$

Сумма по токенам внутри области видимости

$$\mathbf{h}_i = \sum_{j=1}^n g(i-j) \mathbf{x}_j \longrightarrow \mathbf{h}_i = \sum_{j=1}^n g(\mathbf{x}_i, \mathbf{x}_j) \mathbf{x}_j$$

Сумма по всем токенам

Scaled dot-product attention

$$\mathbf{h}_i = \sum_{j=1}^n g(\mathbf{x}_i, \mathbf{x}_j) \mathbf{x}_j$$

Scaled dot-product attention

$$\mathbf{h}_i = \sum_{j=1}^n g(\mathbf{x}_i, \mathbf{x}_j) \mathbf{x}_j$$
$$g(\mathbf{x}_i, \mathbf{x}_j) = \text{softmax}_j \left(\frac{1}{\sqrt{d}} \mathbf{x}_i^T \mathbf{x}_j \right)$$

softmax_j — применение softmax к $\left\{ g(\mathbf{x}_i, \mathbf{x}_j) \right\}_{j=1}^n$ независимо по всем i

Scaled dot-product attention

$$\mathbf{h}_i = \sum_{j=1}^n g(\mathbf{x}_i, \mathbf{x}_j) \mathbf{x}_j$$
$$g(\mathbf{x}_i, \mathbf{x}_j) = \text{softmax}_j \left(\frac{1}{\sqrt{d}} \mathbf{x}_i^T \mathbf{x}_j \right)$$

Self-attention

Ключи: $\mathbf{k}_i = \mathbf{W}_k^T \mathbf{x}_i$

Значения: $\mathbf{v}_i = \mathbf{W}_v^T \mathbf{x}_i$

Запросы: $\mathbf{q}_i = \mathbf{W}_q^T \mathbf{x}_i$

$$\mathbf{h}_i = \sum_{j=1}^n \text{softmax}_j \left(\frac{1}{\sqrt{d}} \mathbf{q}_i^T \mathbf{k}_j \right) \mathbf{v}_j$$

softmax_j — применение softmax к $\left\{ g(\mathbf{x}_i, \mathbf{x}_j) \right\}_{j=1}^n$ независимо по всем i

Scaled dot-product attention

$$\mathbf{h}_i = \sum_{j=1}^n g(\mathbf{x}_i, \mathbf{x}_j) \mathbf{x}_j$$
$$g(\mathbf{x}_i, \mathbf{x}_j) = \text{softmax}_j \left(\frac{1}{\sqrt{d}} \mathbf{x}_i^T \mathbf{x}_j \right)$$

Self-attention

Ключи: $\mathbf{k}_i = \mathbf{W}_k^T \mathbf{x}_i$

Значения: $\mathbf{v}_i = \mathbf{W}_v^T \mathbf{x}_i$

Запросы: $\mathbf{q}_i = \mathbf{W}_q^T \mathbf{x}_i$

$$\mathbf{h}_i = \sum_{j=1}^n \text{softmax}_j \left(\frac{1}{\sqrt{d}} \mathbf{q}_i^T \mathbf{k}_j \right) \mathbf{v}_j$$

softmax_j — применение softmax к $\left\{ g(\mathbf{x}_i, \mathbf{x}_j) \right\}_{j=1}^n$ независимо по всем i

В матричной форме

$$\mathbf{K} = \mathbf{X} \mathbf{W}_k$$

$$\mathbf{V} = \mathbf{X} \mathbf{W}_v$$

$$\mathbf{Q} = \mathbf{X} \mathbf{W}_q$$

$$\mathbf{H} = \text{softmax} \left(\frac{\mathbf{Q} \mathbf{K}^T}{\sqrt{k}} \right) \mathbf{V}$$

Scaled dot-product attention

В матричной форме

$$\mathbf{K} = \mathbf{X}\mathbf{W}_k$$

$$\mathbf{V} = \mathbf{X}\mathbf{W}_v$$

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_q$$

$$\mathbf{H} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{k}}\right)\mathbf{V}$$

Scaled dot-product attention

В матричной форме

$$\mathbf{K} = \mathbf{X}\mathbf{W}_k$$

$$\mathbf{V} = \mathbf{X}\mathbf{W}_v$$

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_q$$

$$\mathbf{H} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{k}}\right)\mathbf{V}$$

```
class ScaledDotProductAttention(nn.Module):
    def __init__(self, input_dim: int, hidden_dim: int):
        super().__init__()
        self.W_q = nn.Linear(input_dim, hidden_dim)
        self.W_k = nn.Linear(input_dim, hidden_dim)
        self.W_v = nn.Linear(input_dim, input_dim)

    def forward(self, x: Tensor) -> Tensor:
        q = self.W_q(x)
        k = self.W_k(x)
        v = self.W_v(x)
        _, _, D_k = k.shape

        scores = torch.bmm(q, k.permute(0, 2, 1)) / D_k**0.5
        attention_weights = scores.softmax(-1)
        return torch.bmm(attention_weights, v)
```

Multi-head attention (MHA)

Голова внимания:

$$\mathbf{K}_i = \mathbf{X} \mathbf{W}_{k,i}$$

$$\mathbf{V}_i = \mathbf{X} \mathbf{W}_{v,i}$$

$$\mathbf{Q}_i = \mathbf{X} \mathbf{W}_{q,i}$$

$$\text{SA}_i(\mathbf{X}) = \text{softmax} \left(\frac{\mathbf{Q}_i \mathbf{K}_i^T}{\sqrt{k}} \right) \mathbf{V}_i$$

```
class ScaledDotProductAttention(nn.Module):
    def __init__(self, input_dim: int, hidden_dim: int):
        super().__init__()
        self.W_q = nn.Linear(input_dim, hidden_dim)
        self.W_k = nn.Linear(input_dim, hidden_dim)
        self.W_v = nn.Linear(input_dim, input_dim)

    def forward(self, x: Tensor) -> Tensor:
        q = self.W_q(x)
        k = self.W_k(x)
        v = self.W_v(x)
        _, _, D_k = k.shape

        scores = torch.bmm(q, k.permute(0, 2, 1)) / D_k**0.5
        attention_weights = scores.softmax(-1)
        return torch.bmm(attention_weights, v)
```


Multi-head attention (MHA)

Голова внимания:

$$\mathbf{K}_i = \mathbf{X} \mathbf{W}_{k,i}$$

$$\mathbf{V}_i = \mathbf{X} \mathbf{W}_{v,i}$$

$$\mathbf{Q}_i = \mathbf{X} \mathbf{W}_{q,i}$$

$$\text{SA}_i(\mathbf{X}) = \text{softmax} \left(\frac{\mathbf{Q}_i \mathbf{K}_i^T}{\sqrt{k}} \right) \mathbf{V}_i$$

Конкатенация и репроекция:

$$\text{MHA}(\mathbf{X}) = \left[\text{SA}_1(\mathbf{X}) \parallel \dots \parallel \text{SA}_h(\mathbf{X}) \right] \mathbf{W}_o$$

Отдельная голова Выходная проекция

```
class ScaledDotProductAttention(nn.Module):
    def __init__(self, input_dim: int, hidden_dim: int):
        super().__init__()
        self.W_q = nn.Linear(input_dim, hidden_dim)
        self.W_k = nn.Linear(input_dim, hidden_dim)
        self.W_v = nn.Linear(input_dim, input_dim)

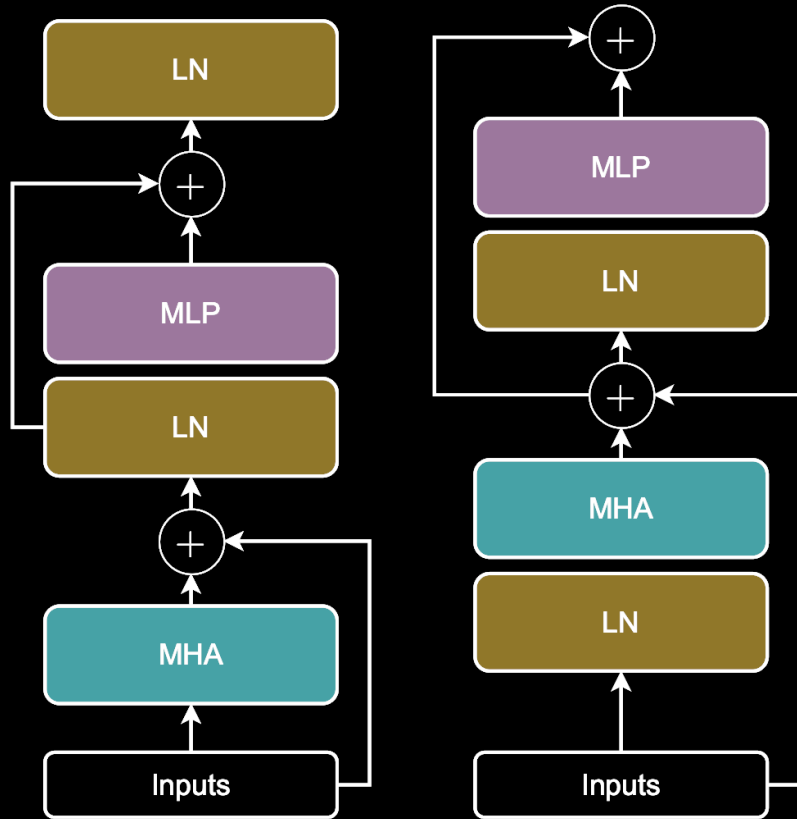
    def forward(self, x: Tensor) -> Tensor:
        q = self.W_q(x)
        k = self.W_k(x)
        v = self.W_v(x)
        _, _, D_k = k.shape

        scores = torch.bmm(q, k.permute(0, 2, 1)) / D_k**0.5
        attention_weights = scores.softmax(-1)
        return torch.bmm(attention_weights, v)

class MultiHeadAttention(nn.Module):
    def __init__(self, input_dim: int, hidden_dim: int, n_heads: int):
        super().__init__()
        self.heads = nn.ModuleList([
            ScaledDotProductAttention(input_dim, hidden_dim)
            for i in range(n_heads)
        ])
        self.W_o = nn.Linear(n_heads * input_dim, input_dim)

    def forward(self, x: Tensor) -> Tensor:
        h = torch.cat([head(x) for head in self.heads], dim=-1)
        return self.W_o(h)
```

Постпроцессинг токенов: нормализация и MLP



(a) Post-normalized block

(b) Pre-normalized block

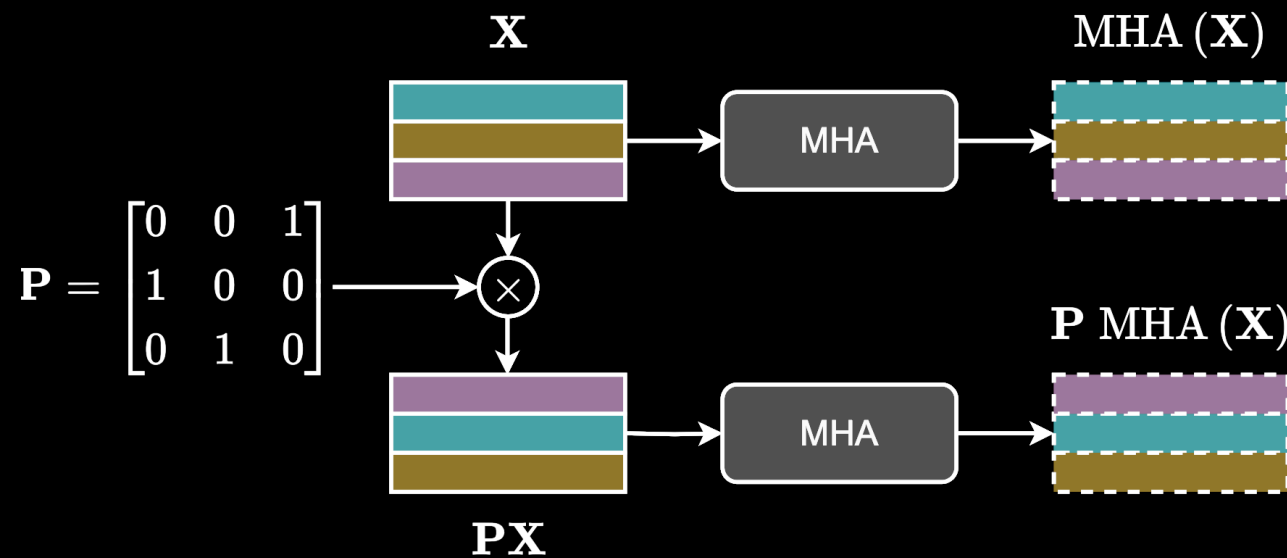
```
class TransformerLayer(nn.Module):
    def __init__(self, input_dim: int, hidden_dim: int, n_heads: int):
        super().__init__()
        self.mha = MultiHeadAttention(input_dim, hidden_dim, n_heads)
        self.norm1 = nn.LayerNorm(input_dim)
        self.mlp = nn.Sequential(
            nn.Linear(input_dim, input_dim * 4),
            nn.ReLU(inplace=True),
            nn.Linear(input_dim * 4, input_dim),
        )
        self.norm2 = nn.LayerNorm(input_dim)

    def forward(self, x: Tensor) -> Tensor:
        z = self.norm1(self.mha(x) + x)
        return self.norm2(self.mlp(z) + z)
```

Позиционные эмбединги

Permutation equivariance

$$\text{MHA}(\mathbf{P}\mathbf{X}) = \mathbf{P} \cdot \text{MHA}(\mathbf{X})$$



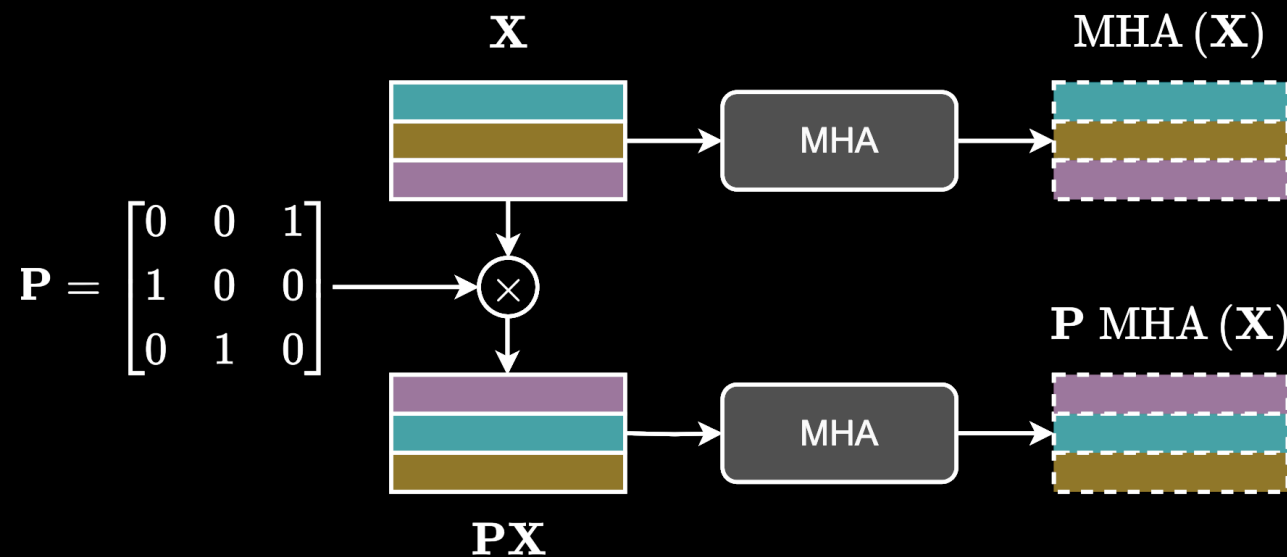
Позиционные эмбединги

Permutation equivariance

$$\text{MHA}(\mathbf{P}\mathbf{X}) = \mathbf{P} \cdot \text{MHA}(\mathbf{X})$$

Абсолютные позиционные эмбединги

$$\text{MHA}(\mathbf{P}\mathbf{X} + \mathbf{S}) \neq \text{MHA}(\mathbf{X} + \mathbf{S})$$



Позиционные эмбединги

Permutation equivariance

$$\text{MHA}(\mathbf{P}\mathbf{X}) = \mathbf{P} \cdot \text{MHA}(\mathbf{X})$$

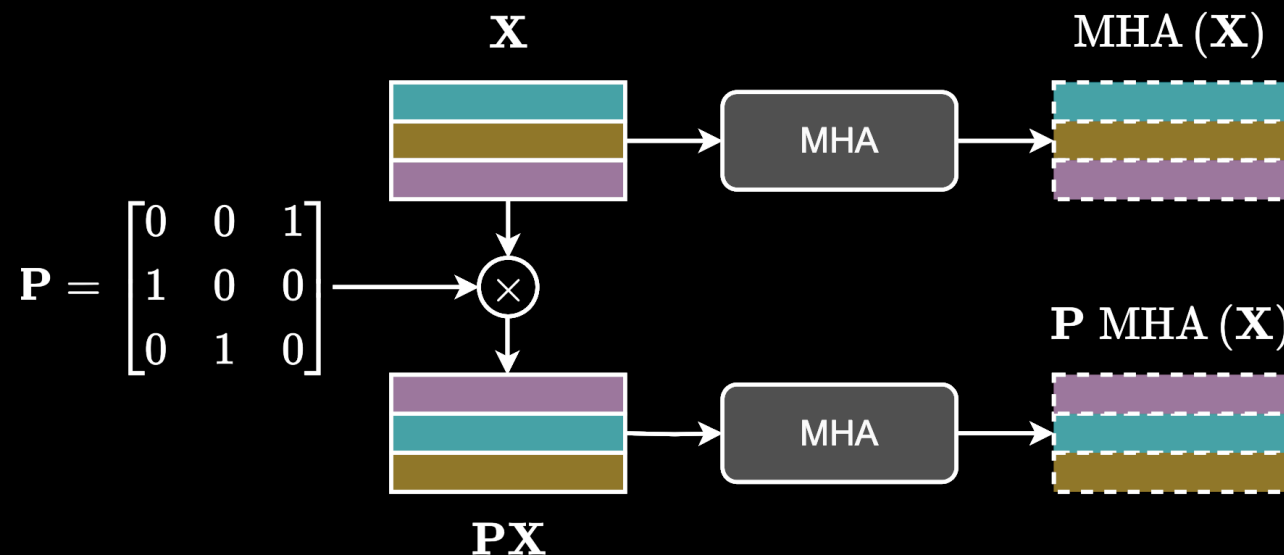
Абсолютные позиционные эмбединги

$$\text{MHA}(\mathbf{P}\mathbf{X} + \mathbf{S}) \neq \text{MHA}(\mathbf{X} + \mathbf{S})$$

Относительные позиционные эмбединги

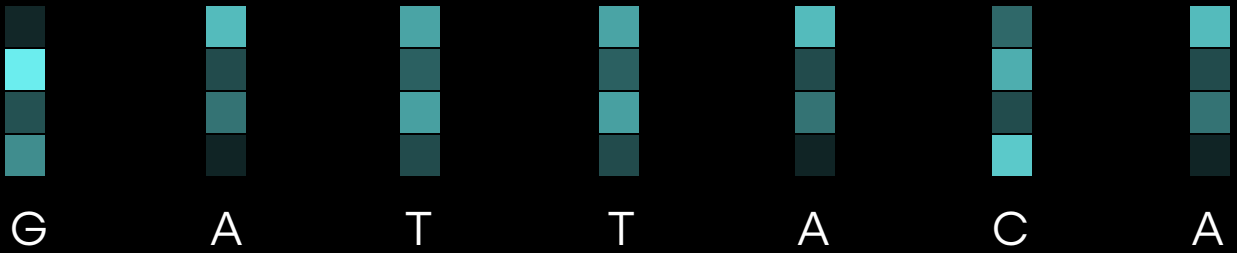
$$\mathbf{h}_i = \sum_{j=1}^n \text{softmax}_j(g(\mathbf{q}_i, \mathbf{k}_j)) \mathbf{v}_j$$

$$g(\mathbf{x}_i, \mathbf{x}_j) \rightarrow g(\mathbf{x}_i, \mathbf{x}_j, i - j)$$

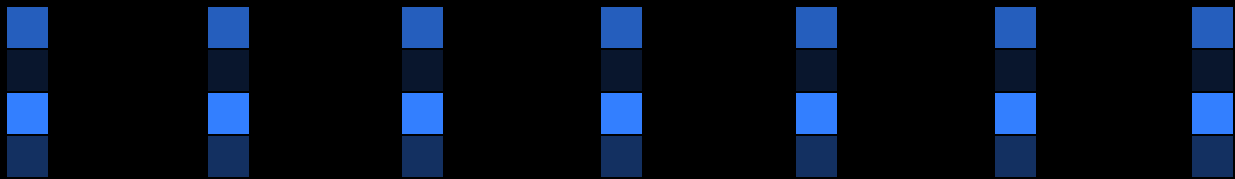


Классификация

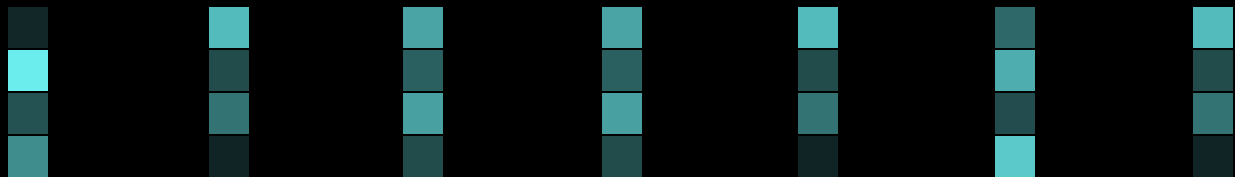
Transformer block



Классификация

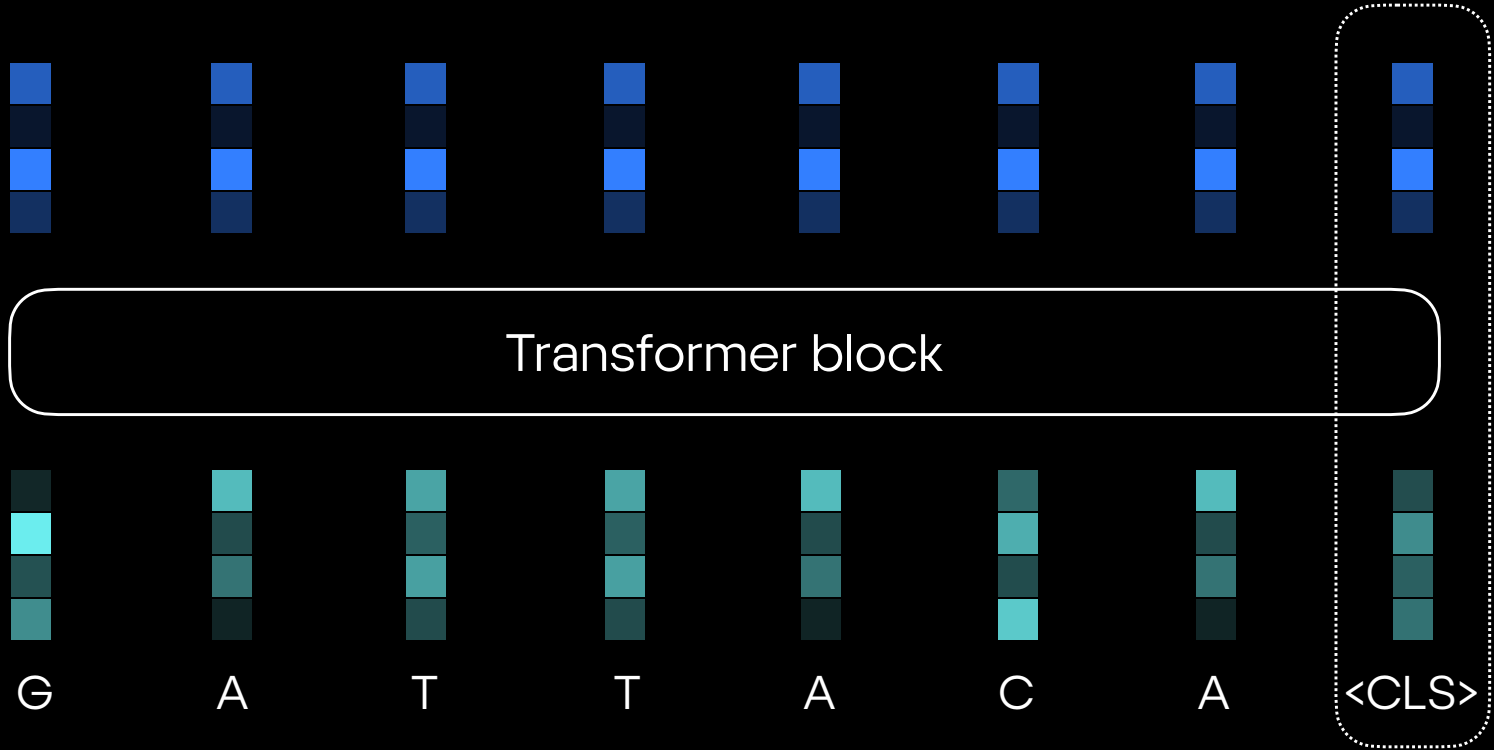


Transformer block



G A T T A C A

Классификация



Классификация

