|  |  |
| --- | --- |
| Gerb-BMSTU_01 | Министерство науки и высшего образования Российской Федерации  Федеральное государственное бюджетное образовательное учреждение  высшего образования  «Московский государственный технический университет  имени Н.Э. Баумана  (национальный исследовательский университет)»  (МГТУ им. Н.Э. Баумана) |

|  |  |
| --- | --- |
| ФАКУЛЬТЕТ | Информатика, искусственный интеллект и системы управления (ИУ) |

|  |  |
| --- | --- |
| КАФЕДРА | Системы обработки информации и управления |

|  |  |
| --- | --- |
| дисциплина | Методы машинного обучения |

|  |
| --- |
| отчет по домашнему заданию № 1 |

|  |
| --- |
| Разбор статьи «OmniNet: всенаправленные представления от трансформаторов» |
| *название домашнего задания* |

|  |  |
| --- | --- |
| Группа | ИУ5-21М |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент | 07.06.2022 |  |  |  | Ся Бэйбэй |
|  | *дата выполнения работы* |  | *подпись* |  | *фамилия, и.о.* |

|  |  |  |  |
| --- | --- | --- | --- |
| Преподаватель |  |  | Гапанюк Ю. E. |
|  | *подпись* |  | *фамилия, и.о.* |

Москва, 2022 г.

# Постановка задачи

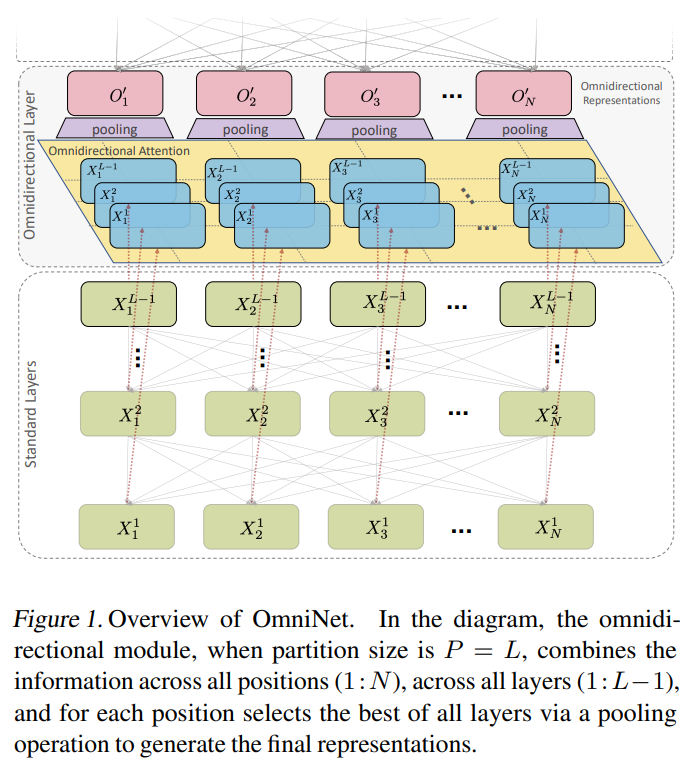
Ссылки на оригинальные статьи: https://arxiv.org/pdf/2103.01075v1.pdf ，<https://arxiv.org/pdf/1706.03762v5.pdf>. В этом документе мы предлагаем OmniNet, который использует omnidirec для того, чтобы сделать выбор между всеми маркерами, подключенными к сети через самосознание. для того чтобы управлять всеми издержками, связанными с подсчетом чувствительности, студенты из оминета параметризуются с помощью быстрых и эффективных моделей самоконтроля. Этот метод имеет отличные характеристики по многочисленным лингвистическим и визуальным заданиям. в частности, OmniNet обладает самыми современными характеристиками WMT EnDe и Enfr, которые превосходят глубинные 60 - ярусные трансформаторы. OmniNet также существенно улучшил задачу идентификации изображений, чем Vit

# Теоретическая часть

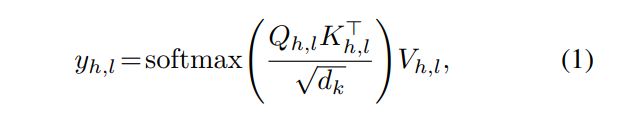
В данной работе основное внимание уделяется методам эффективного обучения всенаправленному вниманию — вниманию по всей ширине и глубине сети. С этой целью мы используем последние достижения в области создания быстрых и эффективных трансформаторов. Многие из этих подходов изучают аппроксимацию с помощью низкоранговой проекции, ядер или разреженности на основе блоков. С этой целью предлагаемый подход использует эти последние достижения, чтобы сделать то, что раньше было невозможно. Используя быстрое и эффективное самовнимание, мы обеспечиваем масштабируемое и мощное всенаправленное внимание.

**SELF-ATTENTION**

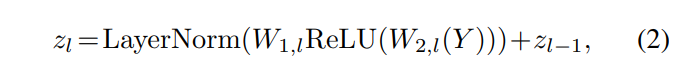
Механизм внутреннего внимания сначала проецирует каждый вход X в представления Q, K, V, используя линейные преобразования, соответствующие запросам, ключам и значениям. Механизм само-внимания, как правило, многоголовый, когда несколько одинаковых линейных проекций выполняются параллельно.



Выход каждой головки внутреннего внимания h на уровне l записывается как:



где , — выход головки на уровне , а — размер каждой головки. Затем выходные данные нескольких головок объединяются, а затем проходят через другое линейное преобразование через ,, которое проецирует конкатенацию всех головок до dm. Это обернуто через нормализацию слоя, за которой следует остаточное соединение, и может бытьзаписывается как: LayerNorm(Wo,l concat([,···,))) +−1 как окончательный результат работы модуля внутреннего внимания. Уровни прямой связи Блок FFN блока Transformer выполняет двухуровневое преобразование, определенное как:

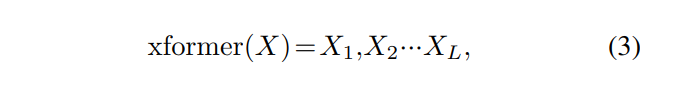


где W1,W2 — обучаемые параметры (весовые преобразования) слоя FFN. Параметры смещения опущены для ясности.

**OmniNet**

Предлагаемый метод OmniNet работает на произвольной многоуровневой архитектуре, которая принимает последовательные входные данные. В нашем описании это обычно относится к многослойной архитектуре X-former в этом разделе. Обратите внимание, что хотя обычно это модель-трансформер, она также может быть произвольным вариантом. Рисунок 1 иллюстрирует краткий обзор предлагаемой архитектуры OmniNet.

1. Всенаправленные представления

В многослойной сети из слоев каждый слой представляет собой последовательность из векторов с размерностью каждый. В частности, OmniNet работает на всех уровнях и объединяет многоуровневую сетевую архитектуру в виде сетки. Мы описываем сеть как xformer, который принимает в качестве входных данных и возвращает тензор размеров × ×.  ****

Где ∈ . Пусть X будет представлением X на уровне i и позиции j последовательности. Механизм OmniNet можно записать так:



Где Attend обозначает произвольный блок внутреннего внимания. Операция IndexSort принимает , , и сортирует, токены в каждой матрице по индексу таким образом, чтобы соседний токен i-го токена в слое был i-м токеном из −1 и + 1 соответственно. Далее, учитывая, что длина входной последовательности равна , для Attend выгодно быть максимально эффективным. В последующих разделах мы опишем три варианта базового линейного механизма внутреннего внимания OmniNet.

Учитывая O ∈ , результат всенаправленного внимания, мы выполняем (.) оператор объединения. Хотя существует множество вариантов объединения операторов, параметризованных или иных, мы используем простую функцию объединения — максимальное объединение шага .



где ∈ . Учитывая , окончательное представление

Расширенная сеть OmniNet определяется как:



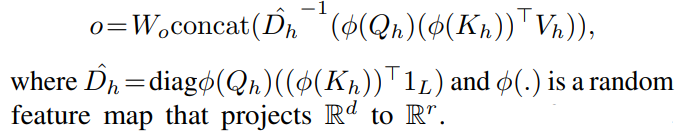
OmniNet и модель основного трансформатора обучаются вместе сквозным способом, т. е. градиенты передаются в обе сети одновременно при каждом обратном проходе.

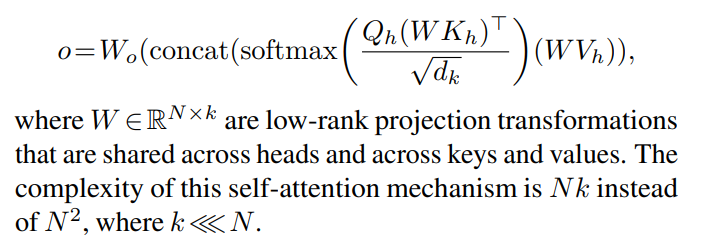
1. Maintaining causality and autoregressive decoding

Ключевым моментом, который следует отметить при использовании IndexSort, является то, что этот порядок позволяет нам применять каузальную маску к функции Attend, а именно, если токены сортируются сначала в соответствии с индексом последовательности, а не сначала по слою, тогда было бы легко применить каузальную маску M , где M[i,j]= 0, когда i≤j, и −inf, когда i> j. Это позволяет использовать OmniNet в настройках авторегрессии.

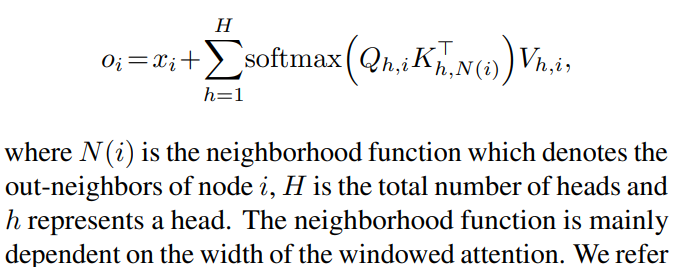
1. Efficient transformers

**Kernel-based**:Этот вариант использует обобщаемое ядро внимание:



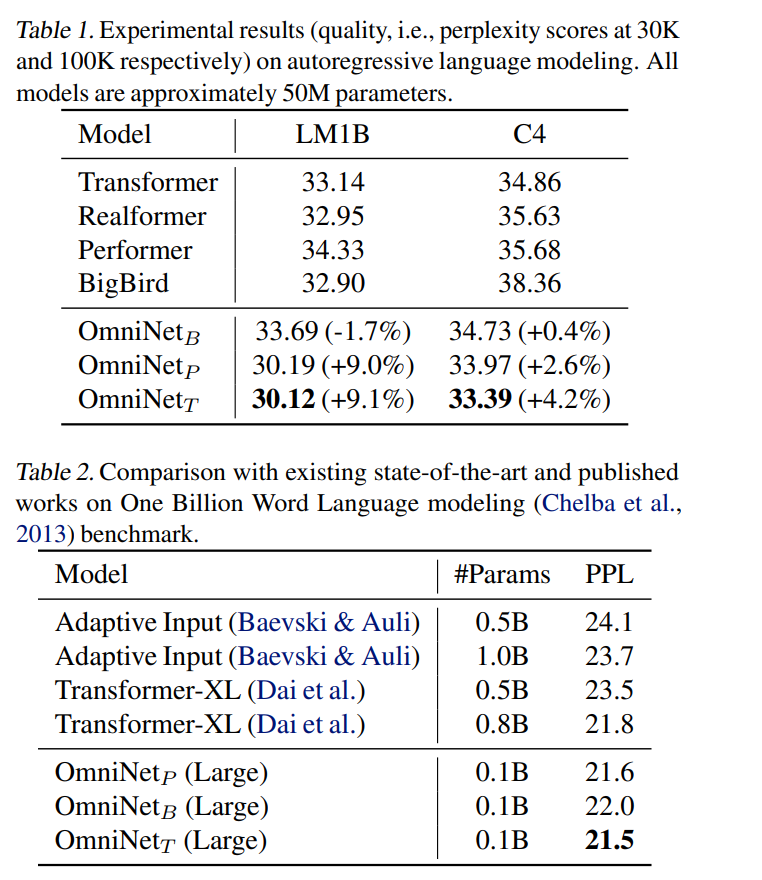
**Low-rank**: 

**Block and Memory based**: Короче говоря, это сочетание ограниченного внимания, глобального внимания и рассеянного внимания. Выход для токена i определяется как:



1. Partitioned Omninet

В этом разделе описываются типы вариантов секционирования, которые мы исследуем в OmniNet. Когда L велико, возможный вход представления в OmniNet может быть чрезвычайно большим.



# Практическая часть

# classes

class PreNorm(nn.Module):

def \_\_init\_\_(self, dim, fn):

super().\_\_init\_\_()

self.norm = nn.LayerNorm(dim)

self.fn = fn

def forward(self, x, \*\*kwargs):

return self.fn(self.norm(x), \*\*kwargs)

class FeedForward(nn.Module):

def \_\_init\_\_(self, dim, mult = 4, dropout = 0.):

super().\_\_init\_\_()

self.net = nn.Sequential(

nn.Linear(dim, dim \* mult),

nn.GELU(),

nn.Dropout(dropout),

nn.Linear(dim \* mult, dim)

)

def forward(self, x):

return self.net(x)

class Attention(nn.Module):

def \_\_init\_\_(

self,

\*,

dim,

heads = 8,

dim\_head = 64,

dropout = 0.,

causal = False

):

super().\_\_init\_\_()

inner\_dim = heads \* dim\_head

self.heads = heads

self.scale = dim\_head \*\* -0.5

self.causal = causal

self.to\_qkv = nn.Linear(dim, inner\_dim \* 3, bias = False)

self.to\_out = nn.Sequential(

nn.Linear(inner\_dim, dim),

nn.Dropout(dropout)

)

def forward(self, x, mask = None):

b, n, d, h, device = \*x.shape, self.heads, x.device

q, k, v = self.to\_qkv(x).chunk(3, dim = -1)

q, k, v = map(lambda t: rearrange(t, 'b n (h d) -> (b h) n d', h = h), (q, k, v))

sim = einsum('b i d, b j d -> b i j', q, k) \* self.scale

max\_neg\_value = -torch.finfo(sim.dtype).max

if exists(mask):

mask = rearrange(mask, 'b i -> b i ()') \* rearrange(mask, 'b j -> b () j')

sim.masked\_fill\_(~mask, max\_neg\_value)

if self.causal:

i, j = sim.shape[-2:]

causal\_mask = torch.ones(i, j, device = device).triu\_(j - i + 1).bool()

causal\_mask = rearrange(causal\_mask, 'i j -> () i j')

sim.masked\_fill\_(causal\_mask, max\_neg\_value)

attn = sim.softmax(dim = -1)

out = einsum('b i j, b j d -> b i d', attn, v)

out = rearrange(out, '(b h) n d -> b n (h d)', h = h)

return self.to\_out(out)

# main class Omninet

class Omninet (nn.Module):

def \_\_init\_\_(

self,

\*,

dim,

depth,

dim\_head = 64,

heads = 8,

pool\_layer\_tokens\_every = 2,

attn\_dropout = 0.,

ff\_dropout = 0.,

feature\_redraw\_interval = 1000

):

super().\_\_init\_\_()

layers = nn.ModuleList([])

for ind in range(depth):

num\_layers = ind + 1

should\_pool = num\_layers % pool\_layer\_tokens\_every

layers.append(nn.ModuleList([

PreNorm(dim, Attention(dim = dim, dim\_head = dim\_head, heads = heads, dropout = attn\_dropout)),

PreNorm(dim, FeedForward(dim = dim, dropout = ff\_dropout)),

PerformerAttention(dim = dim, heads= heads, dim\_head = dim\_head) if should\_pool else None

]))

self.layers = layers

self.pool\_num\_layers = pool\_layer\_tokens\_every

# keep track of redrawing projection matrix for Performer

self.feature\_redraw\_interval = feature\_redraw\_interval

self.register\_buffer('calls\_since\_last\_redraw', torch.tensor(0))

def fix\_projection\_matrices\_(self):

self.feature\_redraw\_interval = None

def check\_redraw\_projections(self):

if not self.training:

return

if exists(self.feature\_redraw\_interval) and self.calls\_since\_last\_redraw >= self.feature\_redraw\_interval:

device = get\_module\_device(self)

fast\_attentions = find\_modules(self, FastAttention)

for fast\_attention in fast\_attentions:

fast\_attention.redraw\_projection\_matrix(device)

self.calls\_since\_last\_redraw.zero\_()

return

self.calls\_since\_last\_redraw += 1

def forward(self, x, mask = None):

self.check\_redraw\_projections()

pool\_num\_layers = self.pool\_num\_layers

hiddens = [x]

for attn, ff, efficient\_attn in self.layers:

x = attn(x, mask = mask) + x

x = ff(x) + x

hiddens.append(x)

if exists(efficient\_attn):

layers\_to\_pool = hiddens[-pool\_num\_layers:]

num\_layers = len(layers\_to\_pool)

all\_tokens = torch.stack(layers\_to\_pool)

all\_tokens = rearrange(all\_tokens, 'l b n d -> b (n l) d')

pool\_attn\_mask = None

if exists(mask):

pool\_attn\_mask = repeat(mask, 'b n -> b (n l)', l = num\_layers)

attended\_tokens = efficient\_attn(all\_tokens, mask = pool\_attn\_mask)

attended\_tokens = rearrange(attended\_tokens, 'b n c -> b c n')

pooled\_tokens = F.max\_pool1d(attended\_tokens, kernel\_size = num\_layers, stride = num\_layers)

x += rearrange(pooled\_tokens, 'b c n -> b n c')

return x

# causal case is sufficiently different to warrant its own class

# use layer axial attention for now, until I rewrite the linear attention cuda kernel

class OmninetCausal(nn.Module):

def \_\_init\_\_(

self,

\*,

dim,

depth,

dim\_head = 64,

heads = 8,

pool\_layer\_tokens\_every = 2,

attn\_dropout = 0.,

ff\_dropout = 0.

):

super().\_\_init\_\_()

self.layer\_pos\_emb = nn.Parameter(torch.randn(depth + 1, dim))

layers = nn.ModuleList([])

for ind in range(depth):

num\_layers = ind + 1

should\_pool = num\_layers % pool\_layer\_tokens\_every

layers.append(nn.ModuleList([

PreNorm(dim, Attention(causal = True, dim = dim, dim\_head = dim\_head, heads = heads, dropout = attn\_dropout)),

PreNorm(dim, FeedForward(dim = dim, dropout = ff\_dropout)),

Attention(dim = dim, heads= heads, dim\_head = dim\_head) if should\_pool else None

]))

self.layers = layers

self.pool\_num\_layers = pool\_layer\_tokens\_every

def forward(self, x, mask = None):

pool\_num\_layers = self.pool\_num\_layers

b = x.shape[0]

pos\_embs = rearrange(self.layer\_pos\_emb, 'n d -> () n d')

x += pos\_embs[:, 0]

hiddens = [x]

for ind, (attn, ff, layer\_axial\_attn) in enumerate(self.layers):

x = attn(x, mask = mask) + x

x = ff(x) + x

x += pos\_embs[:, ind + 1]

hiddens.append(x)

if exists(layer\_axial\_attn):

layers\_to\_pool = hiddens[-pool\_num\_layers:]

num\_layers = len(layers\_to\_pool)

layer\_tokens = rearrange(torch.stack(layers\_to\_pool), 'l b n d -> (b n) l d')

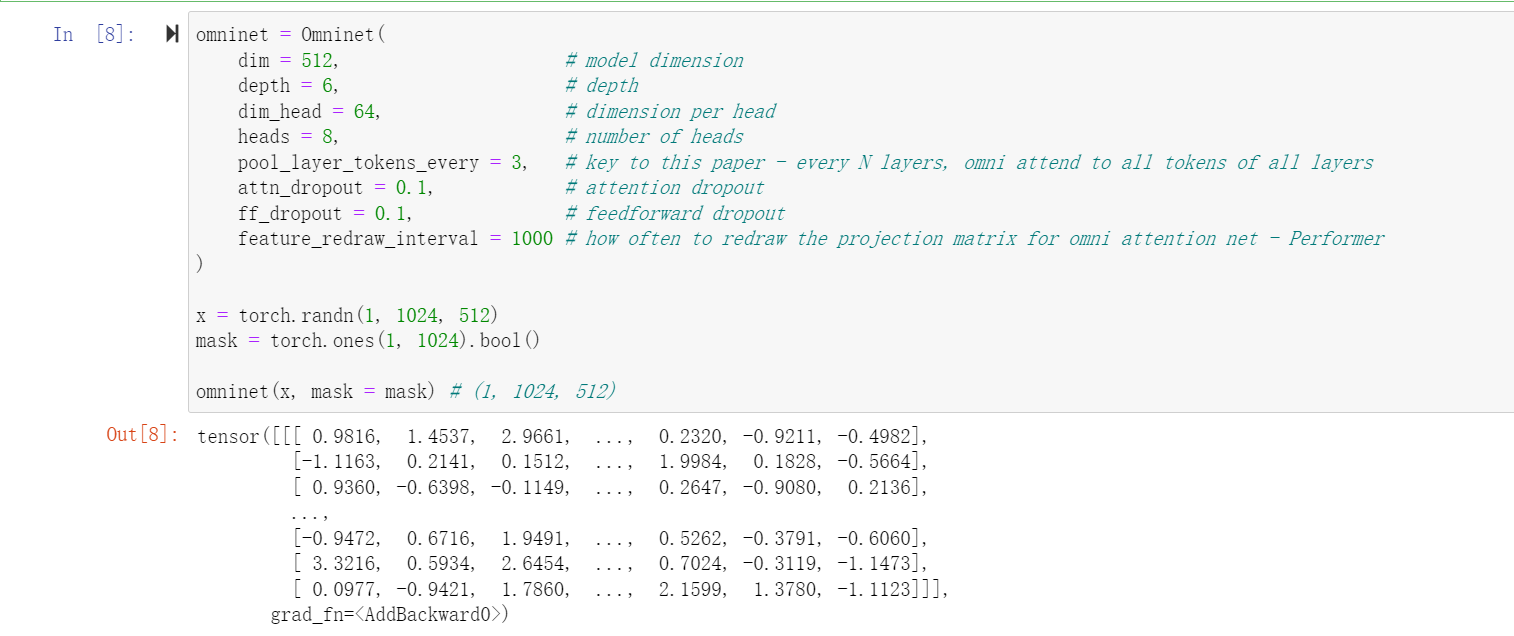
attended\_tokens = layer\_axial\_attn(layer\_tokens)

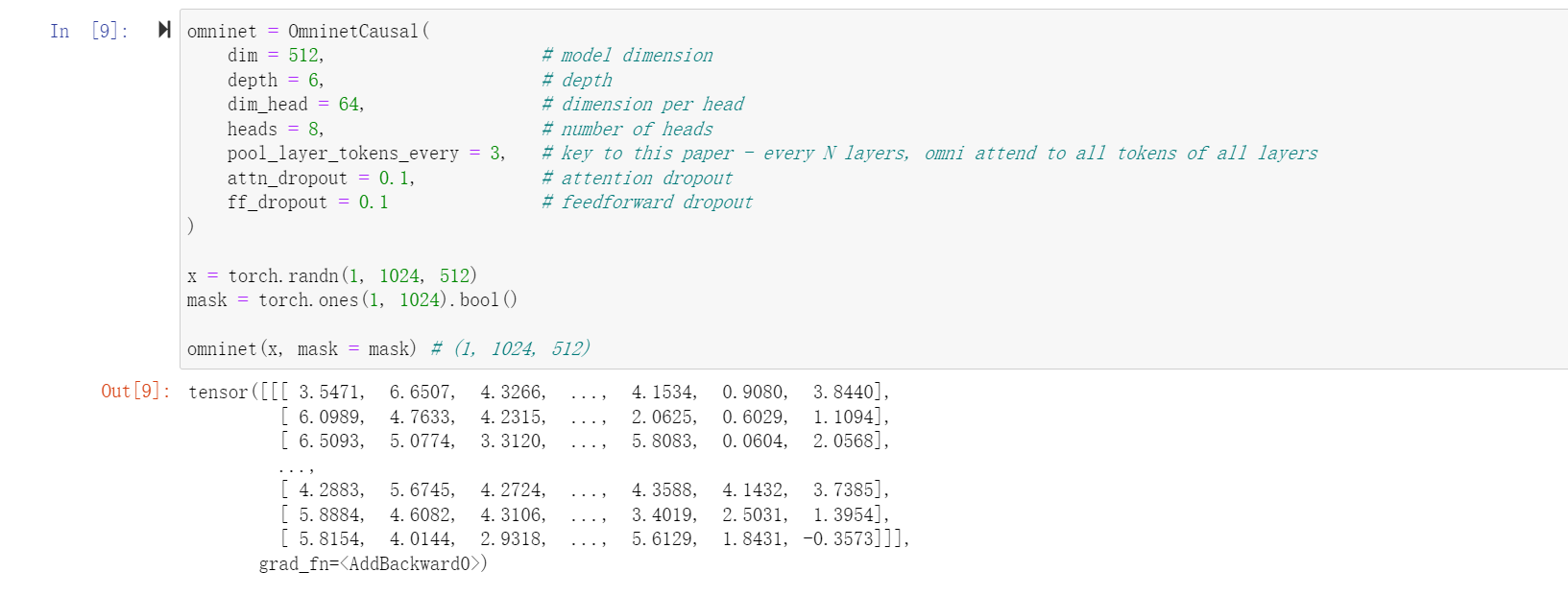
attended\_tokens = rearrange(attended\_tokens, '(b n) l d -> b n l d', b = b)

pooled\_attended\_tokens = attended\_tokens.max(dim = -2).values

x += pooled\_attended\_tokens

return x





# Заключение

В OmniNet вместо того, чтобы поддерживать строго горизонтальное восприимчивое поле, каждому токену разрешается обслуживать все токены во всей сети.Этот процесс также можно интерпретировать как форму экстремального или механизм интенсивного внимания, который имеет рецептивное поле всей ширины и глубины сети. С этой целью всенаправленное внимание изучается с помощью мета-обучателя, который, по сути, является другой моделью, основанной на само-внимании. Чтобы смягчить вычислительные дорогие затраты на полное внимание восприимчивого поля, мы используем эффективные модели внимания к себе.

OmniNet достигает значительных улучшений в этих задачах, включая достижение современной производительности на LM1B, WMT'14 En-De/En-Fr и Long Range Arena. задачи распознавания изображений как при обучении с несколькими выстрелами, так и при точной настройке настроек.

# Список использованных источников

[1] Abnar, S. and Zuidema, W. Quantifying attention flow in transformers. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, 2020.

[2] Baevski, A. and Auli, M. Adaptive input representations for neural language modeling. arXiv preprint arXiv:1809.10853, 2018.

[3] Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473, 2014.

[4] Bapna, A., Chen, M. X., Firat, O., Cao, Y., and Wu, Y. Training deeper neural machine translation models with transparent attention. arXiv preprint arXiv:1808.07561, 2018.

[5] Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, «Attention is all you need» 2017.

[7] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. arXiv preprint arXiv:2005.14165, 2020

[8]Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., and Zagoruyko, S. End-to-end object detection with transformers. arXiv preprint arXiv:2005.12872, 2020.

[9] Chelba, C., Mikolov, T., Schuster, M., Ge, Q., Brants, T., Koehn, P., and Robinson, T. One billion word benchmark for measuring progress in statistical language modeling. arXiv preprint arXiv:1312.3005, 2013.

[10] Chen, M., Radford, A., Child, R., Wu, J., Jun, H., Luan, D., and Sutskever, I. Generative pretraining from pixels. In International Conference on Machine Learning, pp. 1691–1703. PMLR, 2020.

[11] Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J., Mohiuddin, A., Kaiser, L., et al. Rethinking attention with performers. arXiv preprint arXiv:2009.14794, 2020