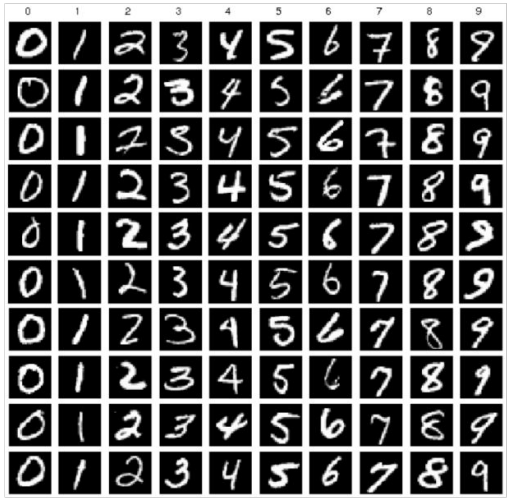
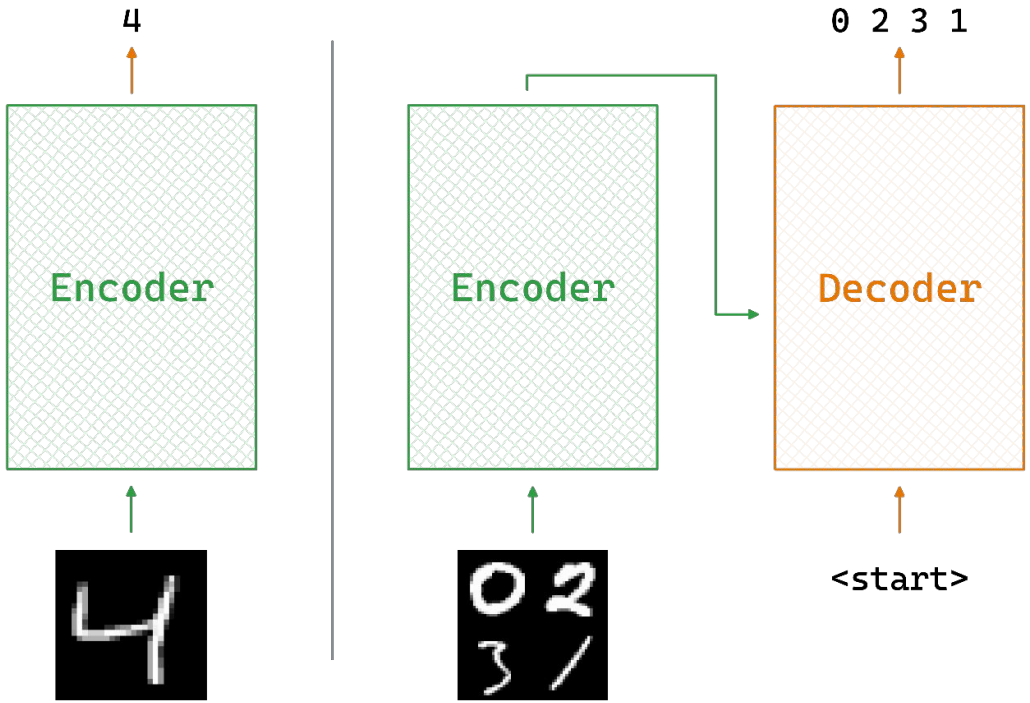
Some Code Is All You Need

A bit of a Q&A

Task





FOUNDERS AND CODERS

Paper

Machine Learning Institute

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* † illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

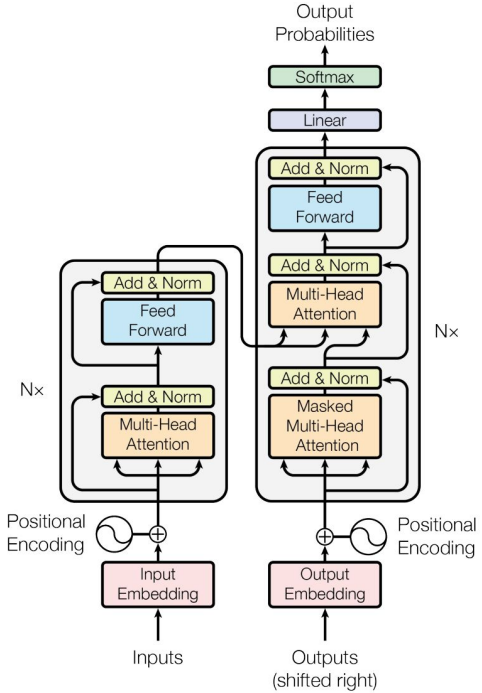
*Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and has been crucially involved in every aspect of this work. Noam proposed scaled dot-product attention, multi-head attention and the parameter-free position representation and became the other person involved in nearly every detail. Niki designed, implemented, tuned and evaluated countless model variants in our original codebase and tensor2tensor. Llion also experimented with novel model variants, was responsible for our initial codebase, and efficient inference and visualizations. Lukasz and Aidan spent countless long days designing various parts of and implementing tensor2tensor, replacing our earlier codebase, greatly improving results and massively accelerating our research.

†Work performed while at Google Brain.

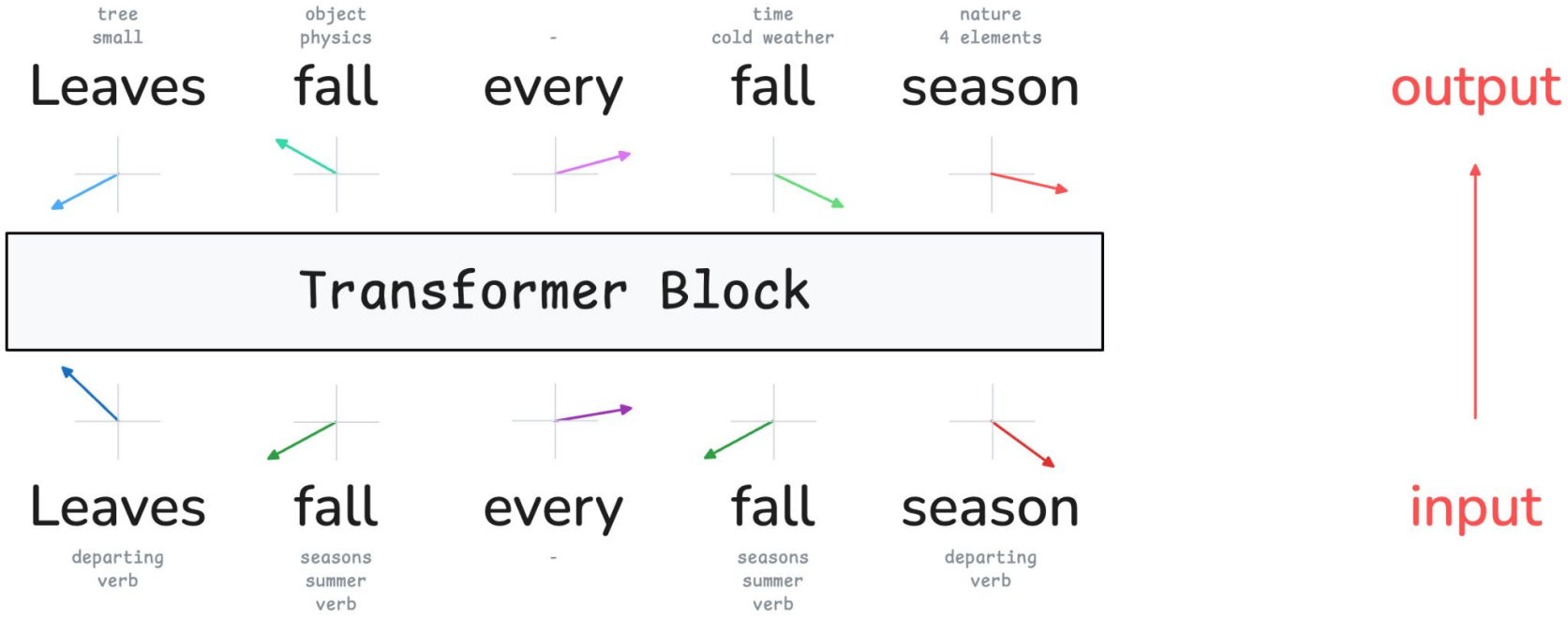
‡Work performed while at Google Research.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

Vaswani et al. 2017



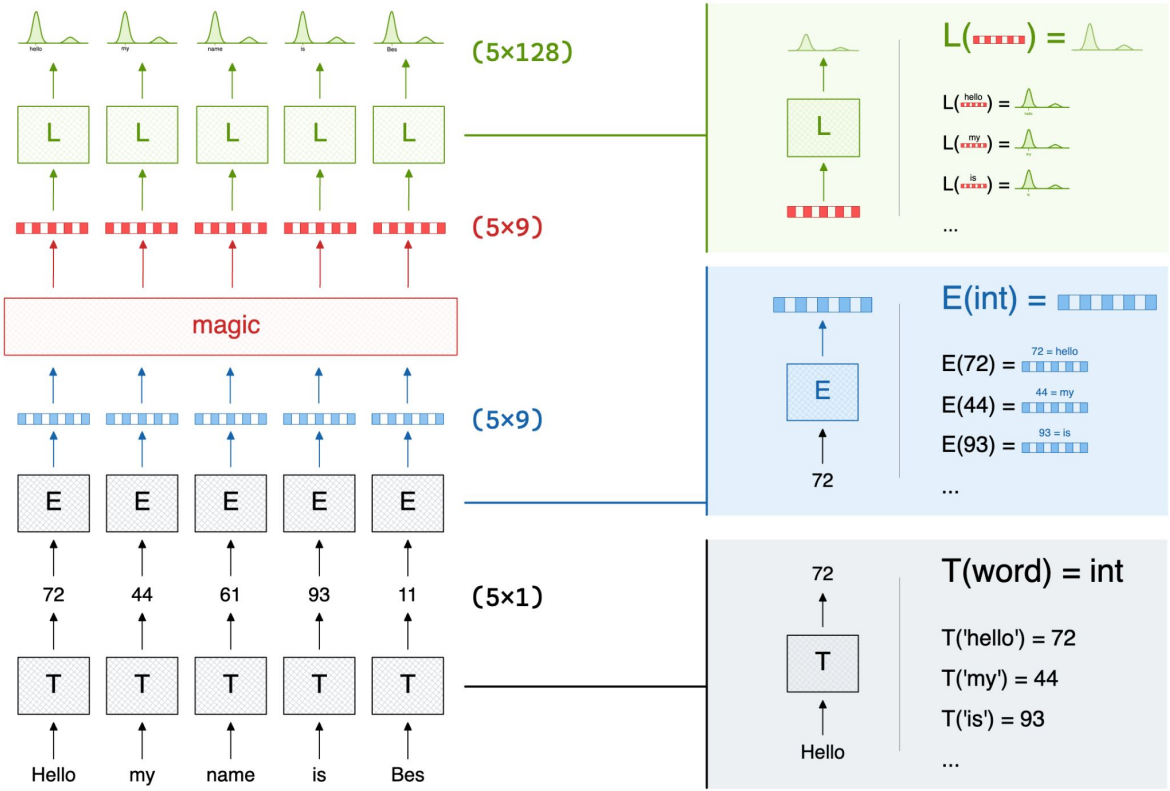
Why “transformer”?



FOUNDERS AND CODERS

word2vec → transformer

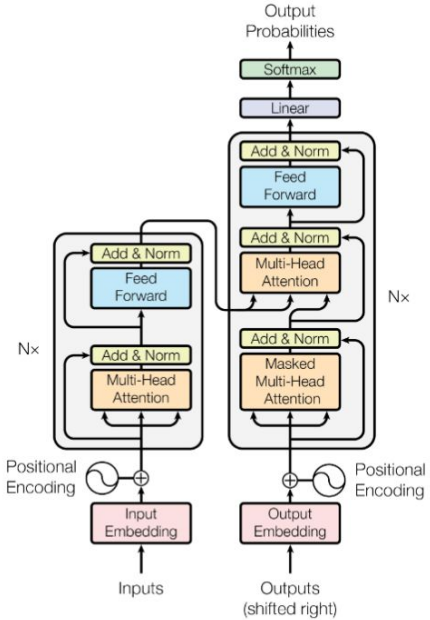
Machine Learning Institute

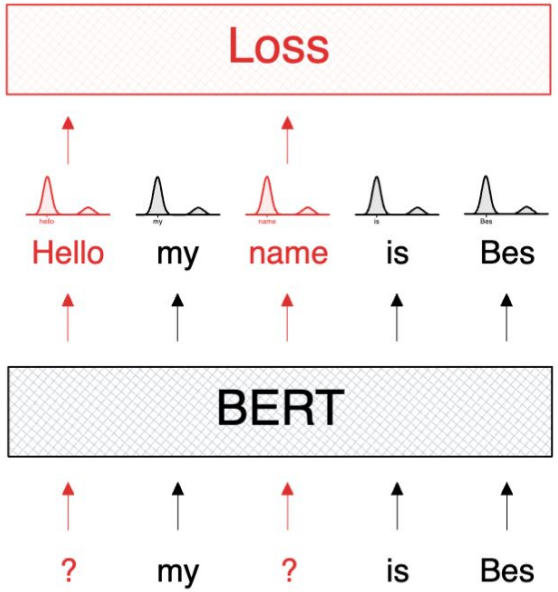


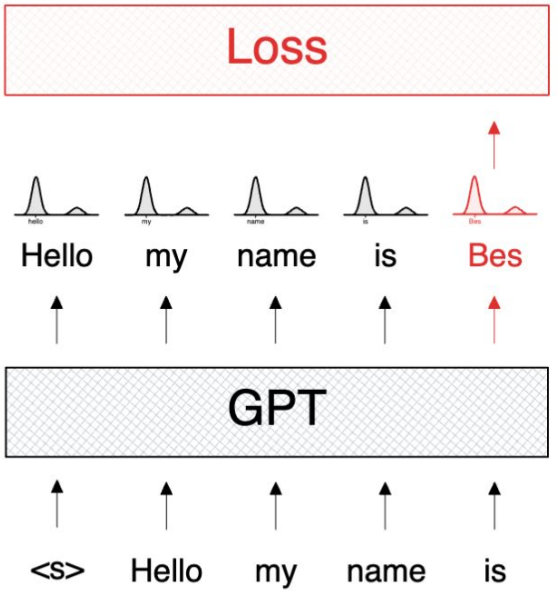
FOUNDERS AND CODERS

Two Flavours

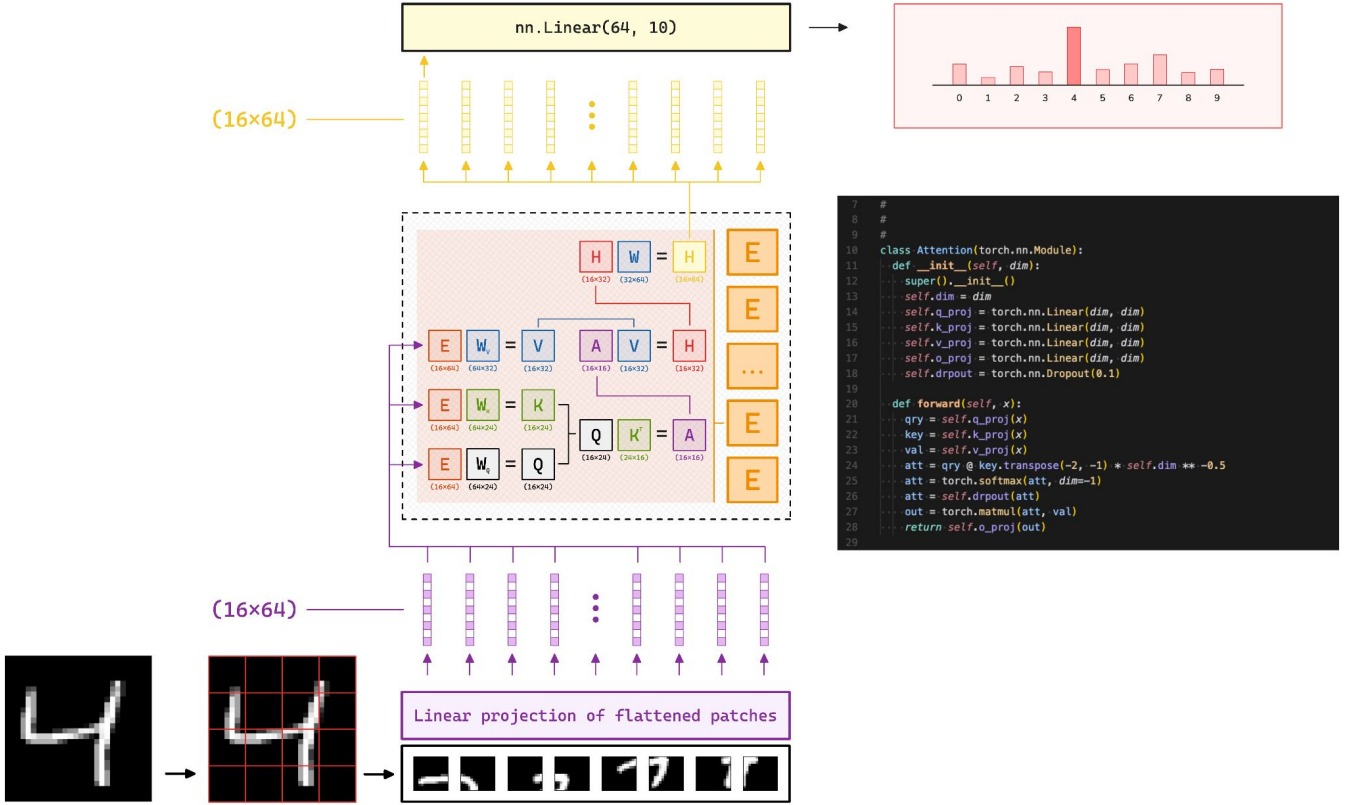
Machine Learning Institute







Encoder




```python
#
#
#
class Attention(torch.nn.Module):
    def __init__(self, dim):
        super().__init__()
        self.dim = dim
        self.q_proj = torch.nn.Linear(dim, dim)
        self.k_proj = torch.nn.Linear(dim, dim)
        self.v_proj = torch.nn.Linear(dim, dim)
        self.o_proj = torch.nn.Linear(dim, dim)
        self.drpout = torch.nn.Dropout(0.1)

    def forward(self, x):
        qry = self.q_proj(x)
        key = self.k_proj(x)
        val = self.v_proj(x)
        att = qry @ key.transpose(-2, -1) * self.dim ** -0.5
        att = torch.softmax(att, dim=-1)
        att = self.drpout(att)
        out = torch.matmul(att, val)
        return self.o_proj(out)
```

FOUNDERS AND CODERS

Code

Machine Learning Institute

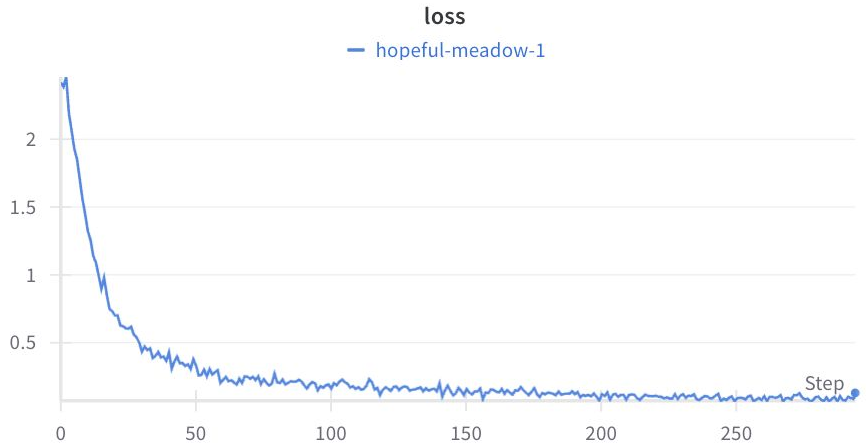
```python
#
#
#
class LookerTrns(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.cls = torch.nn.Parameter(torch.randn(1, 1, 128))
        self.emb = torch.nn.Linear(196, 128)
        self.pos = torch.nn.Embedding(17, 128)
        self.register_buffer('rng', torch.arange(17))
        self.enc = torch.nn.ModuleList([EncoderLayer(128) for _ in range(6)])
        self.fin = torch.nn.Sequential(
            torch.nn.LayerNorm(128),
            torch.nn.Linear(128, 10)
        )

    def forward(self, x):
        B = x.shape[0]                      # [B, 16, 196]
        pch = self.emb(x)                   # [B, 16, 128]
        cls = self.cls.expand(B, -1, -1)    # [B, 1, 128]
        hdn = torch.cat([cls, pch], dim=1)  # [B, 17, 128]
        hdn = hdn + self.pos(self.rng)      # [B, 17, 128]
        for enc in self.enc: hdn = enc(hdn) # [B, 17, 128]
        out = hdn[:, 0, :]                  # [B, 128]
        return self.fin(out)                # [B, 10]
```

```python
#
#
#
#
class EncoderLayer(torch.nn.Module):
    def __init__(self, dim):
        super().__init__()
        self.att = Attention(dim)
        self.ffn = FFN(dim)
        self.ini = torch.nn.LayerNorm(dim)
        self.fin = torch.nn.LayerNorm(dim)

    def forward(self, src):
        out = self.att(src)
        src = src + out
        src = self.ini(src)
        out = self.ffn(src)
        src = src + out
        src = self.fin(src)
        return src

```

```python
#
#
#
class FFN(torch.nn.Module):
    def __init__(self, dim):
        super().__init__()
        self.one = torch.nn.Linear(dim, dim)
        self.drp = torch.nn.Dropout(0.1)
        self.rlu = torch.nn.ReLU(inplace=True)
        self.two = torch.nn.Linear(dim, dim)

    def forward(self, x):
        x = self.one(x)
        x = self.rlu(x)
        x = self.drp(x)
        x = self.two(x)
        return x

```

```python
#
#
#
class Attention(torch.nn.Module):
    def __init__(self, dim):
        super().__init__()
        self.dim = dim
        self.q_proj = torch.nn.Linear(dim, dim)
        self.k_proj = torch.nn.Linear(dim, dim)
        self.v_proj = torch.nn.Linear(dim, dim)
        self.o_proj = torch.nn.Linear(dim, dim)
        self.drpout = torch.nn.Dropout(0.1)

    def forward(self, x):
        qry = self.q_proj(x)
        key = self.k_proj(x)
        val = self.v_proj(x)
        att = qry @ key.transpose(-2, -1) * self.dim ** -0.5
        att = torch.softmax(att, dim=-1)
        att = self.drpout(att)
        out = torch.matmul(att, val)
        return self.o_proj(out)

```

It works!



https://github.com/MLX-FAC/vit

FOUNDERS AND CODERS

Machine Learning Institute

Thank you!