

TTT - Learning to (Learn at Test Time)

	Initial state	Update rule	Output rule	Cost
Naive RNN	$s_0 = \texttt{vector()}$	$s_t = \sigma(\theta_{ss} s_{t-1} + \theta_{sx} x_t)$	$z_t = \theta_{zs} s_t + \theta_{zx} x_t$	$O(1)$
Self-attention	$s_0 = \texttt{list()}$	$s_t = s_{t-1}.\texttt{append}(k_t, v_t)$	$z_t = V_t \texttt{softmax}\left(K_t^T q_t\right)$	$O(t)$
Naive TTT	$W_0 = f.\texttt{params()}$	$W_t = W_{t-1} - \eta \nabla \ell(W_{t-1}; x_t)$	$z_t = f(x_t; W_t)$	$O(1)$

Figure 4. **Top**: A generic sequence modeling layer expressed as a hidden state that transitions according to an update rule. All sequence modeling layers can be viewed as different instantiations of three components in this figure: the initial state, update rule and output rule. **Bottom**: Examples of sequence modeling layers and their instantiations of the three components. The naive TTT layer was shown in Figure 1. Self-attention has a hidden state growing with context, therefore growing cost per token. Both the naive RNN and TTT layer compress the growing context into a hidden state of fixed size, therefore their cost per token stays constant.

公众号 · 新智元

研究人员设计了一类新的序列建模层，其中隐藏状态是模型，更新规则是自监督学习的一个步骤。

由于更新测试序列上隐藏状态的过程，相当于在测试时训练模型，因此此类新层称为测试时训练（ TTT）层。

为了在长上下文中既保持效率，又具有表达能力，需要一个更好的「压缩启发式」（ compression heuristic ）方法。具体来说，就需要将数百万个 token压缩成一个能有效捕捉其底层结构和关系的隐藏状态。

Transformer的 KV cache在长序列的时候非常低效， Manba 的固定长度的中间状态在长序列的时候表达力不够。

????

关键思想是，使用自监督学习来将历史上下文压缩成一个隐藏状态。

方法是将上下文视为一个无标签数据集，而将状态视为一个模型。

具体来说，隐藏状态现在等同于一个模型 f的权重， 输出 **token**就是由更新后权重的模型 **f**对输入所做的预测 。更新规则是在某个自监督损失 ℓ上进行的一步梯度下降:

□□□□□□□□□□ f□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□ -□□□□□□ TTT□□

??

□□□□ TTT□□□□□□□□□□□□□□□□□ Transformer
□□□□□□□□□□□□□□□□□□□□□□□□□□□□ token□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□ outer loop□□□□□□ TTT□□□□ W□□□□□□ inner loop
□□□□□□□□□□□□□□□□□□□□□□ W□□□□ f
□□□□□□□□□□□□□□□□□□□□□□

??

```
class TTT_Layer(nn.Module):
  def __init__(self):
    self.task = Task()

  def forward(self, in_seq):
    state = Learner(self.task)
    out_seq = []
    for tok in in_seq:
      state.train(tok)
      out_seq.append(state.predict(tok))
    return out_seq

class Task(nn.Module):
  def __init__(self):
    self.theta_K = nn.Param((d1, d2))
    self.theta_V = nn.Param((d1, d2))
    self.theta_Q = nn.Param((d1, d2))

  def loss(self, f, x):
    train_view = self.theta_K @ x
    label_view = self.theta_V @ x
    return MSE(f(train_view), label_view)
```

```
class Learner():
  def __init__(self, task):
    self.task = task
    # Linear here, but can be any model
    self.model = Linear()
    # online GD here for simplicity
    self.optim = OGD()

  def train(self, x):
    # grad function wrt first arg
    # of loss, which is self.model
    grad_fn = grad(self.task.loss)
    # calculate inner-loop grad
    grad_in = grad_fn(self.model, x)

    # starting from current params,
    # step in direction of grad_in,
    self.optim.step(self.model, grad_in)

  def predict(self, x):
    test_view = self.task.theta_Q @ x
    return self.model(test_view)
```

Figure 6. Naive implementation of a TTT layer with a linear model and online GD in the style of PyTorch. `TTT_Layer` can be dropped into a larger network like other sequence modeling layers. Training the network will optimize the parameters of `Task` in `TTT_Layer`, because both are subclasses of `nn.Module`. Since `Learner` is not a subclass of `nn.Module`, `state.model` is updated manually in the inner loop for each call of `state.train`. For simplicity, we sometimes overload `model` as `model.parameters`.

公众号 · 新智元

□□□□□□□□□□□□□□□□□□□□ **Task**□□□□□□□□□□□□□□□□ **theta_K theta_V**□□□□□□□ **sequence**□□□□ **theta_K theta_V theta_Q**
□□□□□□□□□□□□□□□

??

研究人员在 Pile上执行了 2k和8k上下文长度的标准实验， Pile是一个用于训练开源 LLM的流行文档数据集。

TTT-MLP（ M）在较大的 FLOP预算下表现稍差。尽管 TTT-MLP在每个模型大小上，都比 TTT-Linear
具有更好的复杂度，但 FLOP的额外成本抵消了这种优势。

在 8k上下文中， TTT-Linear（ M）和 TTT-MLP（ M）的表现均明显优于 Mamba。即使是具有 Transformer 架构的 TTT-MLP（T），性能也比 Mamba略好。

另外，研究人员还观察到了一个非常明显的现象：随着上下文长度变长， TTT层相对于 Mamba
的优势就更大了。

Revision #1
Created 2025-01-11 09:44:05 UTC by Colin
Updated 2025-01-11 09:44:09 UTC by Colin