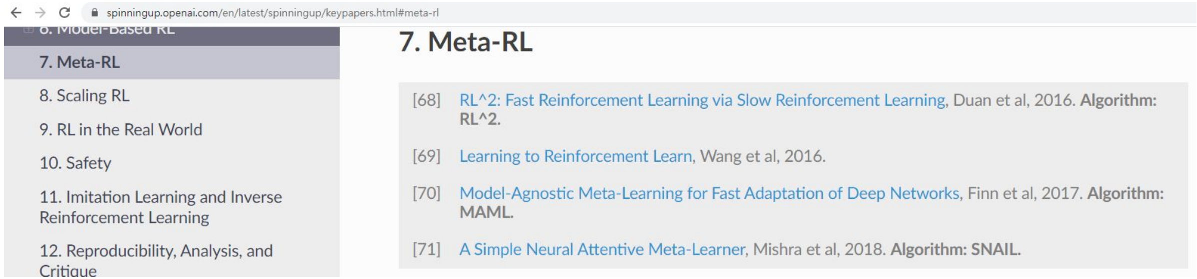
A SIMPLE NEURAL ATTENTIVE META-LEARNER

Nikhil Mishra [*†] Mostafa Rohaninejad[*] Xi Chen[†] Pieter Abbeel[†]
UC Berkeley, Department of Electrical Engineering and Computer Science
Embodied Intelligence
{nmishra, rohaninejadm, c.xi, pabbeel}@berkeley.edu


spinningup.openai.com/en/latest/spinningup/keypapers.html#meta-rl
6. Model-Based RL
7. Meta-RL
8. Scaling RL
9. RL in the Real World
10. Safety
11. Imitation Learning and Inverse Reinforcement Learning
12. Reproducibility, Analysis, and Critique

7. Meta-RL

[68] RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning, Duan et al, 2016. **Algorithm: RL^2.**

[69] Learning to Reinforcement Learn, Wang et al, 2016.

[70] Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks, Finn et al, 2017. **Algorithm: MAML.**

[71] A Simple Neural Attentive Meta-Learner, Mishra et al, 2018. **Algorithm: SNAIL.**

ICLR 2018

reviewer : 강동구

목차

1. 소개
2. 수식 및 목적
3. 관련 논문
4. 전체 구조
5. 메인 블록(dense, TC, attention) 설명
6. Experiments
7. 마치며

소개

인간에 비하면 AI는 적은 데이터로 task가 바뀌는 상황에서 형편없음.

인간은 과거의 지식과 경험을 효율적으로 사용하나 AI는 상대적으로 그렇지 못함.

meta-learning(ML)은 AI의 학습범위를 다양한 task로 넓히려는 분야.

sequence-to-sequence 문제로 볼 수 있고, 핵심은 과거의 경험과 지식을 내면화(internalize), 참조(refer). 이를 위해 두가지 방법론 사용

1. Tempral(causal) convolution : 과거의 정보를 규합
2. Soft(causal) attention : 과거의 특정 정보를 pinpoint

시험 환경(domain)

Supervised few-shot image classification : Omniglot, mini-imagenet

RL : MAB, tabular MDP, visual navigation, Continuous control

the Omniglot dataset Lake et al. Science 2015

1623 characters from 50 different alphabets

Hebrew
Bengali
Greek
Futurama

20 instances of each character

수식 및 목적

$$\mathcal{T} = P(\mathcal{T}_i)$$

distribution of tasks.

Each task $\mathcal{T}_i$ is episodic and defined by inputs x_t, outputs a_t

loss function $\mathcal{L}_i(x_t, a_t)$

a transition distribution $P_i(x_t|x_{t-1}, a_{t-1})$, and an episode length H_i

meta-learner (with parameters θ) models the distribution $\pi(a_t|x_1, \ldots, x_t; \theta)$

$$\min_{\theta} \mathbb{E}_{\mathcal{T}_i \sim \mathcal{T}} \left[\sum_{t=0}^{H_i} \mathcal{L}_i(x_t, a_t) \right],$$

where $x_t \sim P_i(x_t|x_{t-1}, a_{t-1})$, $a_t \sim \pi(a_t|x_1, \ldots, x_t; \theta)$

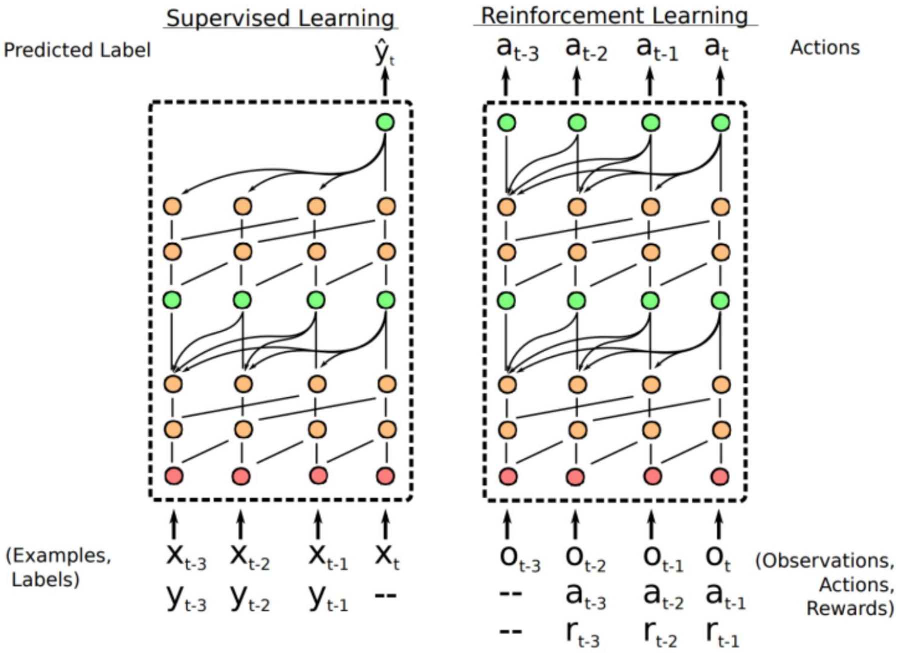
학습은 Task의 mini-batch tasks로 진행. 평가는 다른 task 분포 T에서 sample

관련 논문

1. Metalearning with memory-augmented neural networks(santoro, ICML 2016) : RNN으로 meta-learner를 구현
2. Wavenet: A generative model for raw audio(van den Oord, CoRR 2016) : dilated 1D-convolutions을 사용한 Temporal convolutions(TC)로 sequential data 생성. RNN보다 직접적이고, 더 많은 양의 과거 정보에 접근 가능. 그러나 시퀀스의 길이가 길어지면 문제 발생(dilation, multi layer & residual 만으로는 원천적 해결 불가).
3. Attention is all you need(Vaswani, arXiv, 2017) : soft attention으로 무한히 긴 맥락에서도 특정 정보를 pinpoint 할수있는 모델 소개. 그러나 RL에서의 순서의존성을 해결하기엔 부족.

각각 단점이 있지만 위 언급된 TC와 soft attention은 상호보완 가능!

전체 구조



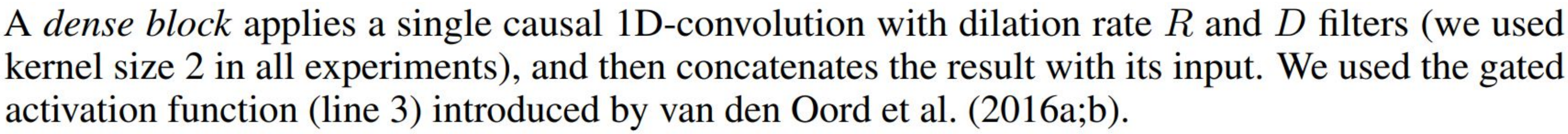
red : input
orange : TC block
green : attention block

SL과 RL로 구분.

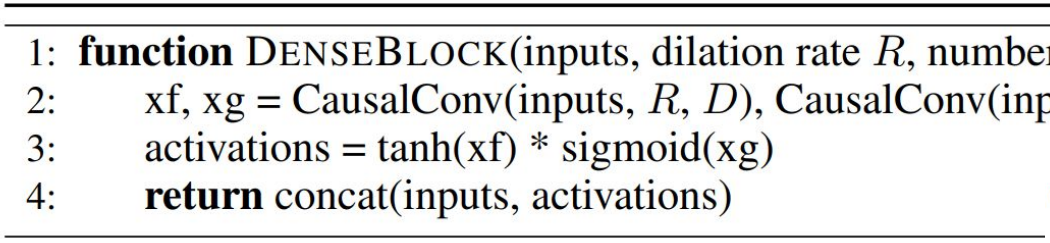
RL의 경우 multiple episode로 확장하기 위해 observation에 episode termination flag 포함.

TRPO + GAE

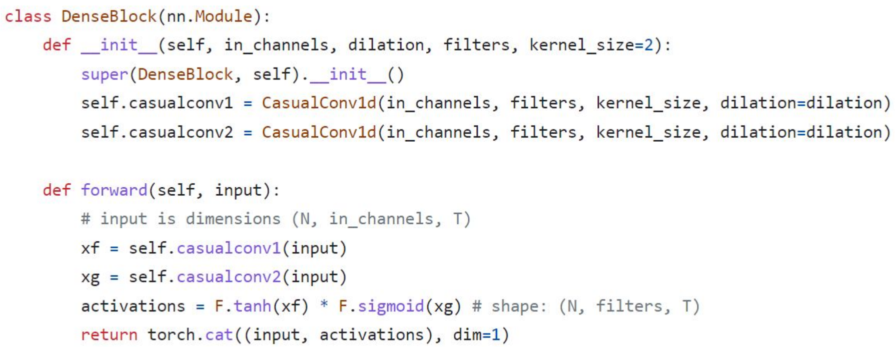
메인 블록 설명(1) : Dense block



A *dense block* applies a single causal 1D-convolution with dilation rate R and D filters (we used kernel size 2 in all experiments), and then concatenates the result with its input. We used the gated activation function (line 3) introduced by van den Oord et al. (2016a;b).



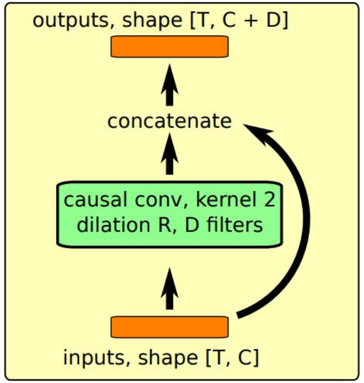
```
1: function DENSEBLOCK(inputs, dilation rate $R$, number of filters $D$):
2:     xf, xg = CausalConv(inputs, $R$, $D$), CausalConv(inputs, $R$, $D$)
3:     activations = tanh(xf) * sigmoid(xg)
4:     return concat(inputs, activations)
```



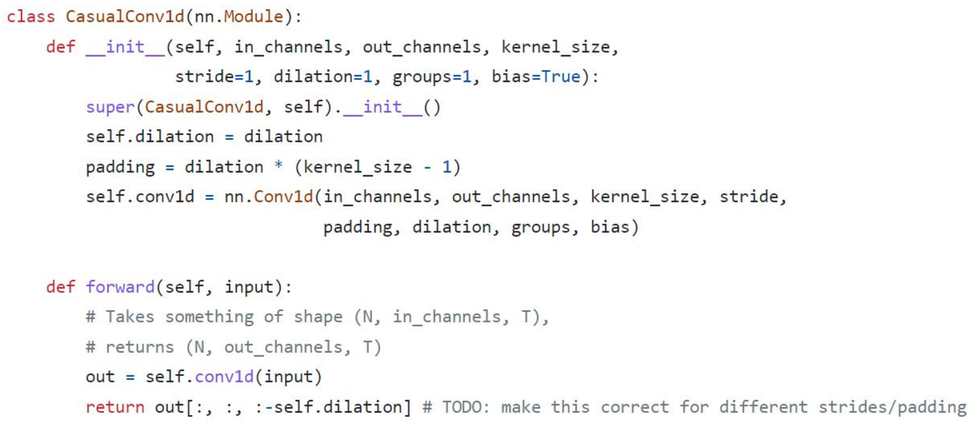
```python
class DenseBlock(nn.Module):
    def __init__(self, in_channels, dilation, filters, kernel_size=2):
        super(DenseBlock, self).__init__()
        self.casualconv1 = CasualConv1d(in_channels, filters, kernel_size, dilation=dilation)
        self.casualconv2 = CasualConv1d(in_channels, filters, kernel_size, dilation=dilation)

    def forward(self, input):
        # input is dimensions (N, in_channels, T)
        xf = self.casualconv1(input)
        xg = self.casualconv2(input)
        activations = F.tanh(xf) * F.sigmoid(xg) # shape: (N, filters, T)
        return torch.cat((input, activations), dim=1)
```

(a) Dense Block (dilation rate R, D filters)



메인 블록 설명(1) : Dense block



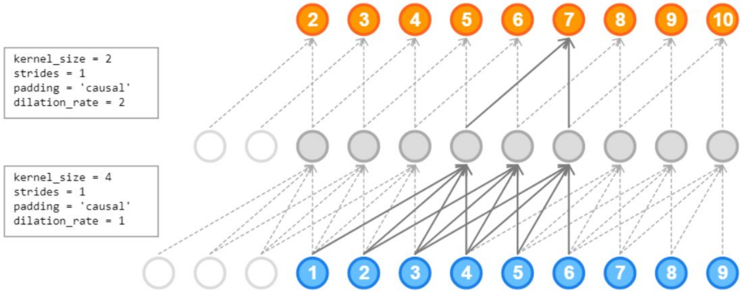
```
class CasualConv1d(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size,
                 stride=1, dilation=1, groups=1, bias=True):
        super(CasualConv1d, self).__init__()
        self.dilation = dilation
        padding = dilation * (kernel_size - 1)
        self.conv1d = nn.Conv1d(in_channels, out_channels, kernel_size, stride,
                                padding, dilation, groups, bias)

    def forward(self, input):
        # Takes something of shape (N, in_channels, T),
        # returns (N, out_channels, T)
        out = self.conv1d(input)
        return out[:, :, :-self.dilation] # TODO: make this correct for different strides/padding
```

t 시점의 출력은 현재와 과거에만 의존하는 causality를 만족하기 위함

input의 양사이드에 padding을 붙이고 계산 후 오른쪽 side를 제거하는 식으로 구현

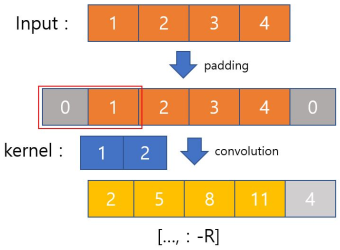
stride = 1
kernel_size = 2로 고정



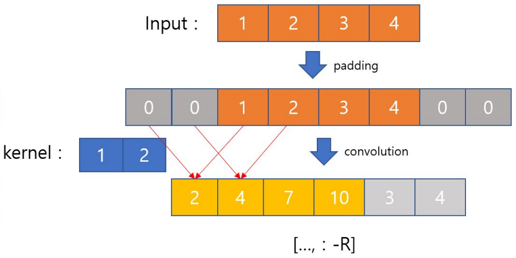
Causal Dilated Convolution

https://stopspoon.tistory.com/48

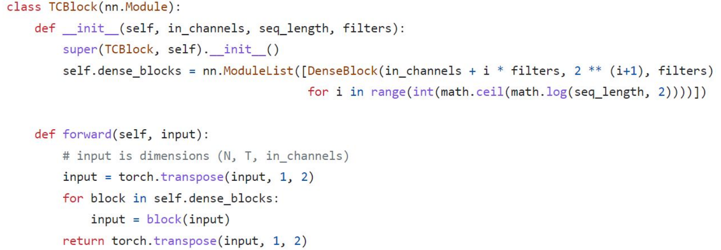
Kernel size(S) = 2
Dilation rate(R) = 1
Stride = 1
Padding = 1 (=R*(S-1))



Kernel size(S) = 2
Dilation rate(R) = 2
Stride = 1
Padding = 2 (=R*(S-1))

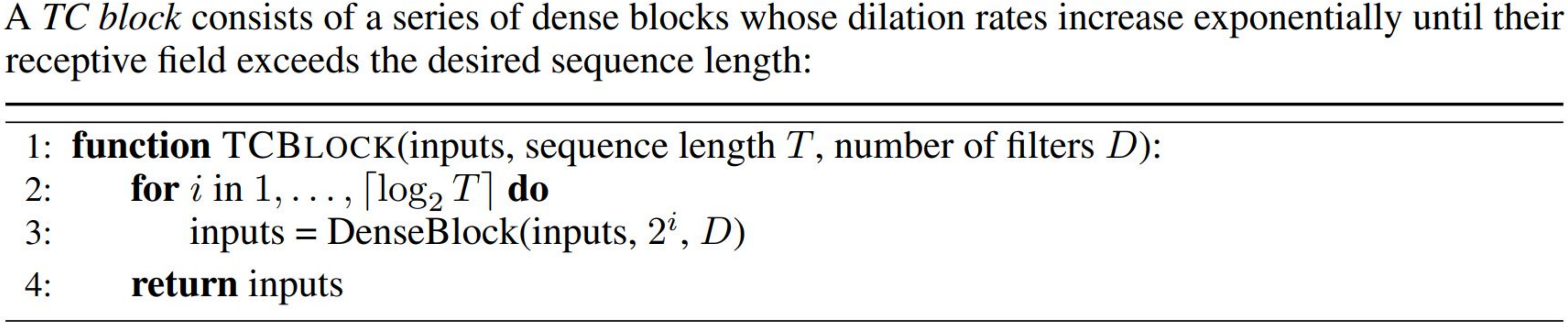


메인 블록 설명(2) : TC block



```
class TCBlock(nn.Module):
    def __init__(self, in_channels, seq_length, filters):
        super(TCBlock, self).__init__()
        self.dense_blocks = nn.ModuleList([DenseBlock(in_channels + i * filters, 2 ** (i+1), filters)
                                           for i in range(int(math.ceil(math.log(seq_length, 2))))])

    def forward(self, input):
        # input is dimensions (N, T, in_channels)
        input = torch.transpose(input, 1, 2)
        for block in self.dense_blocks:
            input = block(input)
        return torch.transpose(input, 1, 2)
```



A *TC block* consists of a series of dense blocks whose dilation rates increase exponentially until their receptive field exceeds the desired sequence length:

```
function TCBLOCK(inputs, sequence length $T$, number of filters $D$):
    for $i$ in $1, \ldots, \lceil \log_2 T \rceil$ do
        inputs = DenseBlock(inputs, $2^i$, $D$)
    return inputs
```

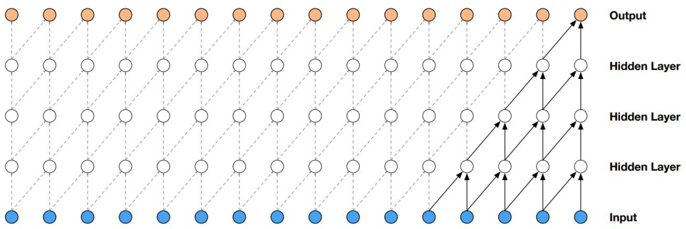
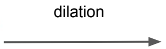


Figure 2: Visualization of a stack of causal convolutional layers.



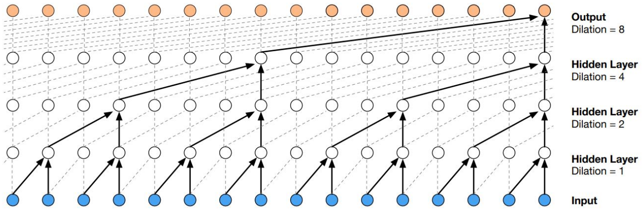
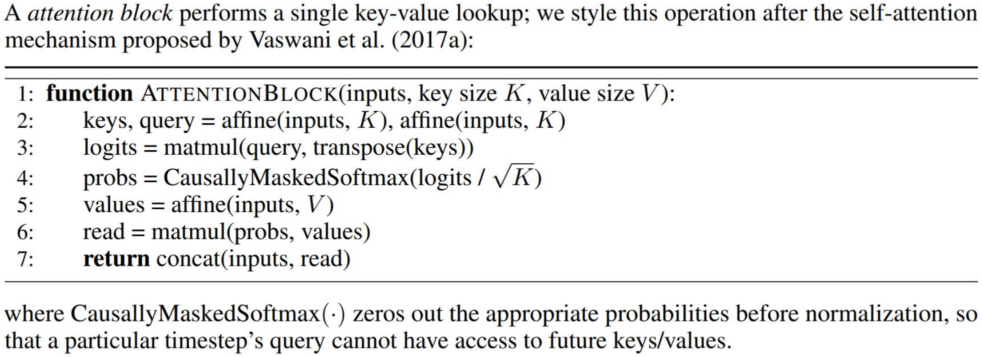


Figure 3: Visualization of a stack of *dilated* causal convolutional layers.

메인 블록 설명(3) : Attention block

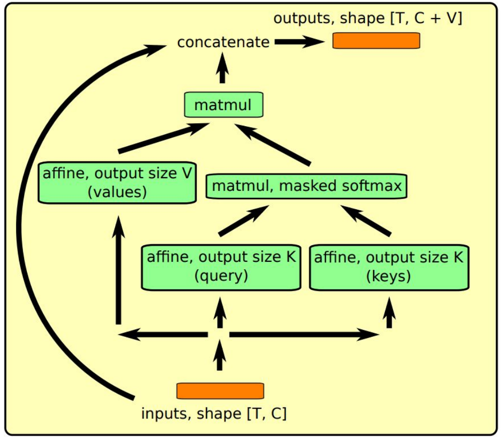


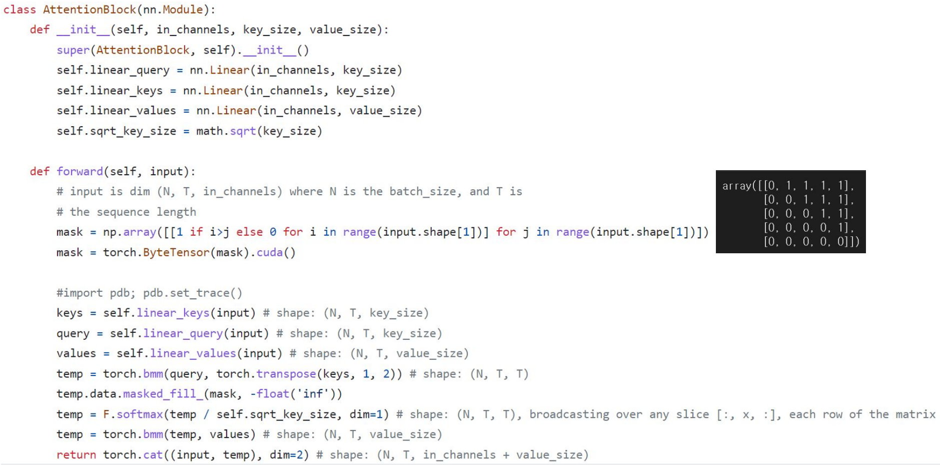
A *attention block* performs a single key-value lookup; we style this operation after the self-attention mechanism proposed by Vaswani et al. (2017a):

```
function AttentionBlock(inputs, key size $K$, value size $V$):
    keys, query = affine(inputs, $K$), affine(inputs, $K$)
    logits = matmul(query, transpose(keys))
    probs = CausallyMaskedSoftmax(logits / $\sqrt{K}$)
    values = affine(inputs, $V$)
    read = matmul(probs, values)
    return concat(inputs, read)
```

where CausallyMaskedSoftmax$(\cdot)$ zeros out the appropriate probabilities before normalization, so that a particular timestep's query cannot have access to future keys/values.

(b) Attention Block (key size K, value size V)


outputs, shape [T, C + V]
concatenate
matmul
affine, output size V (values)
matmul, masked softmax
affine, output size K (query)
affine, output size K (keys)
inputs, shape [T, C]



```python
class AttentionBlock(nn.Module):
    def __init__(self, in_channels, key_size, value_size):
        super(AttentionBlock, self).__init__()
        self.linear_query = nn.Linear(in_channels, key_size)
        self.linear_keys = nn.Linear(in_channels, key_size)
        self.linear_values = nn.Linear(in_channels, value_size)
        self.sqrt_key_size = math.sqrt(key_size)

    def forward(self, input):
        # input is dim (N, T, in_channels) where N is the batch_size, and T is
        # the sequence length
        mask = np.array([[1 if i>j else 0 for i in range(input.shape[1])] for j in range(input.shape[1])])
        mask = torch.ByteTensor(mask).cuda()

        #import pdb; pdb.set_trace()
        keys = self.linear_keys(input) # shape: (N, T, key_size)
        query = self.linear_query(input) # shape: (N, T, key_size)
        values = self.linear_values(input) # shape: (N, T, value_size)
        temp = torch.bmm(query, torch.transpose(keys, 1, 2)) # shape: (N, T, T)
        temp.data.masked_fill_(mask, -float('inf'))
        temp = F.softmax(temp / self.sqrt_key_size, dim=1) # shape: (N, T, T), broadcasting over any slice [:, x, :], each row of the matrix
        temp = torch.bmm(temp, values) # shape: (N, T, value_size)
        return torch.cat((input, temp), dim=2) # shape: (N, T, in_channels + value_size)
```

```
array([[0, 1, 1, 1, 1],
       [0, 0, 1, 1, 1],
       [0, 0, 0, 1, 1],
       [0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0]])
```

Experiments (Base line, 비교대상)

Supervised few-shot image classification : Omniglot, mini-imagenet

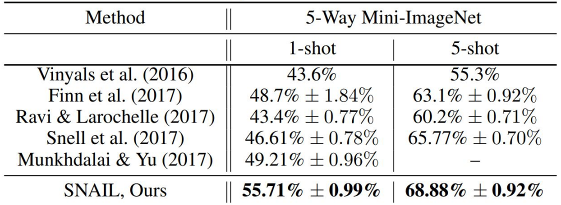
- Matching networks for one shot learning. In Advances in Neural Information Processing Systems. Vinyals(2016)
- Meta networks. Munkhdalai(2017)
- Prototypical networks for few-shot learning. Snell(2017)
- MAML : Model-agnostic meta learning. Finn(2017)

RL : MAB, tabular MDP, visual navigation, Continuous control

- RL^2(LSTM) : Fast reinforcement learning via slow reinforcement learning. Duan(2016)
- Learning to reinforcement learn. Wang(2016)
- MAML : Model-agnostic meta learning. Finn(2017)
- Domain specific algorithm. ex. MAB : gittins. MDP : OPSRL, e-greedy,...

Experiments (Few-shot, MDP, Continuos control, visual naviation)

Table 2: 5-way, 1-shot and 5-shot classification accuracies on mini-ImageNet, with 95% confidence intervals where available. For each task, the best-performing method is highlighted, along with any others whose confidence intervals overlap.



Method	5-Way Mini-ImageNet	
	1-shot	5-shot
Vinyals et al. (2016)	43.6%	55.3%
Finn et al. (2017)	48.7% ± 1.84%	63.1% ± 0.92%
Ravi & Larochelle (2017)	43.4% ± 0.77%	60.2% ± 0.71%
Snell et al. (2017)	46.61% ± 0.78%	65.77% ± 0.70%
Munkhdalai & Yu (2017)	49.21% ± 0.96%	–
SNAIL, Ours	**55.71% ± 0.99%**	**68.88% ± 0.92%**

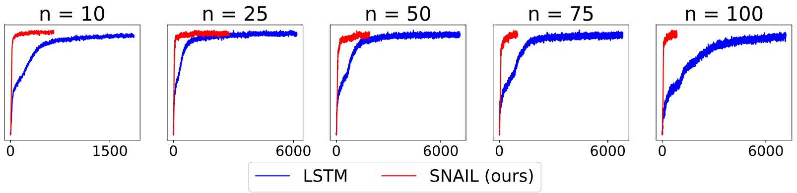
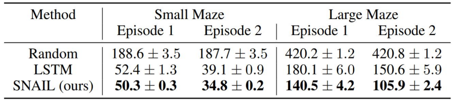


Figure 3: Learning curves of SNAIL (red) and LSTM (blue) on the random MDP task for different values of N. The horizontal axis is the TRPO iteration, and the vertical is average reward.

Table 5: Average time to find the goal on each episode in the small and large mazes. SNAIL solves the mazes the fastest, and improves the most from the first to second episode.



Method	Small Maze		Large Maze	
	Episode 1	Episode 2	Episode 1	Episode 2
Random	188.6 ± 3.5	187.7 ± 3.5	420.2 ± 1.2	420.8 ± 1.2
LSTM	52.4 ± 1.3	39.1 ± 0.9	180.1 ± 6.0	150.6 ± 5.9
SNAIL (ours)	**50.3 ± 0.3**	**34.8 ± 0.2**	**140.5 ± 4.2**	**105.9 ± 2.4**

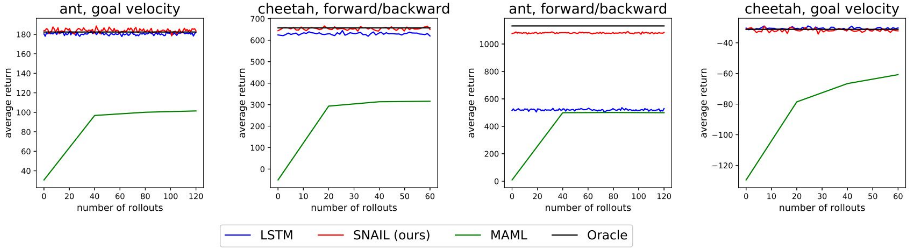


Figure 4: Test-time adaptation curves on simulated locomotion tasks for SNAIL, LSTM, and MAML (which was unrolled for three policy gradient updates). Since SNAIL incorporates experience through its hidden state, it can exploit common task structure to perform optimally within a few timesteps.

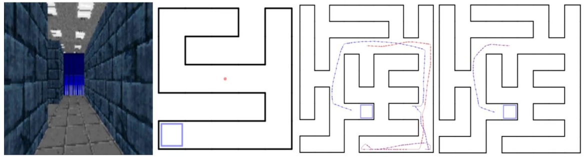


Figure 5: From left to right: (a) A (higher-resolution) example of the observations the agent receives. (b) An example of the mazes used for training (goal shown in blue). (c) The movement of the SNAIL on its first episode in a larger maze, exploring the maze until it finds the goal. (d) The SNAIL's path during its second episode in the same maze as (c). Remembering the goal location, it navigates there directly on the second episode. Maps like in (b), (c), (d) are used for visualization but not available to the agent. In (c), (d), the color progression from red to blue indicates the passage of time (red earlier).

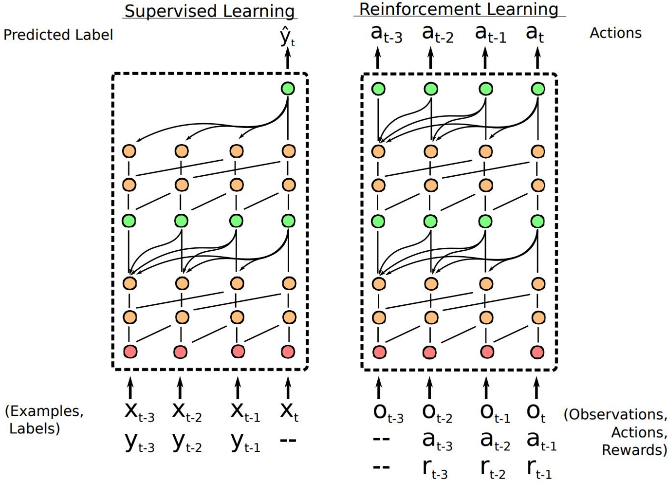
마치며

장점

1. Temporal convolution과 Casal attention을 상호보완한다는 기발한 생각
2. 실험에 다양한 도메인을 사용
3. MAML은 사장된 느낌을 받음.

단점

1. LSTM(RL^2)과 거의 동일한 성능
2. 코드가 강화학습 부분은 없다.


Supervised Learning
Reinforcement Learning
Predicted Label
$\hat{y}_t$
a_{t-3} a_{t-2} a_{t-1} a_t
Actions
(Examples, Labels)
x_{t-3} x_{t-2} x_{t-1} x_t
y_{t-3} y_{t-2} y_{t-1} --
o_{t-3} o_{t-2} o_{t-1} o_t
-- a_{t-3} a_{t-2} a_{t-1}
-- r_{t-3} r_{t-2} r_{t-1}
(Observations, Actions, Rewards)