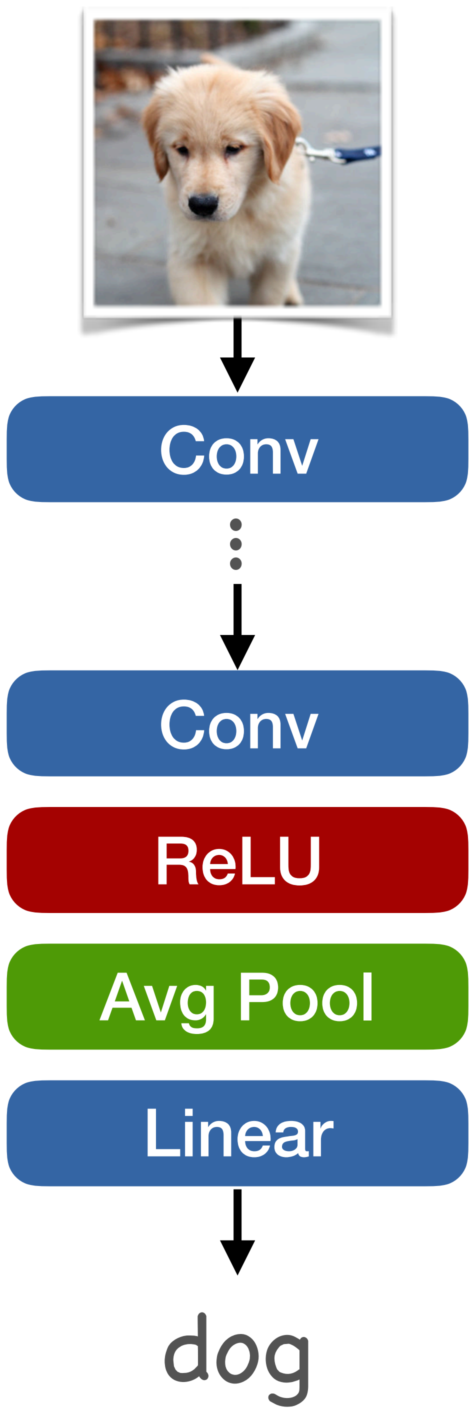
Sequence models

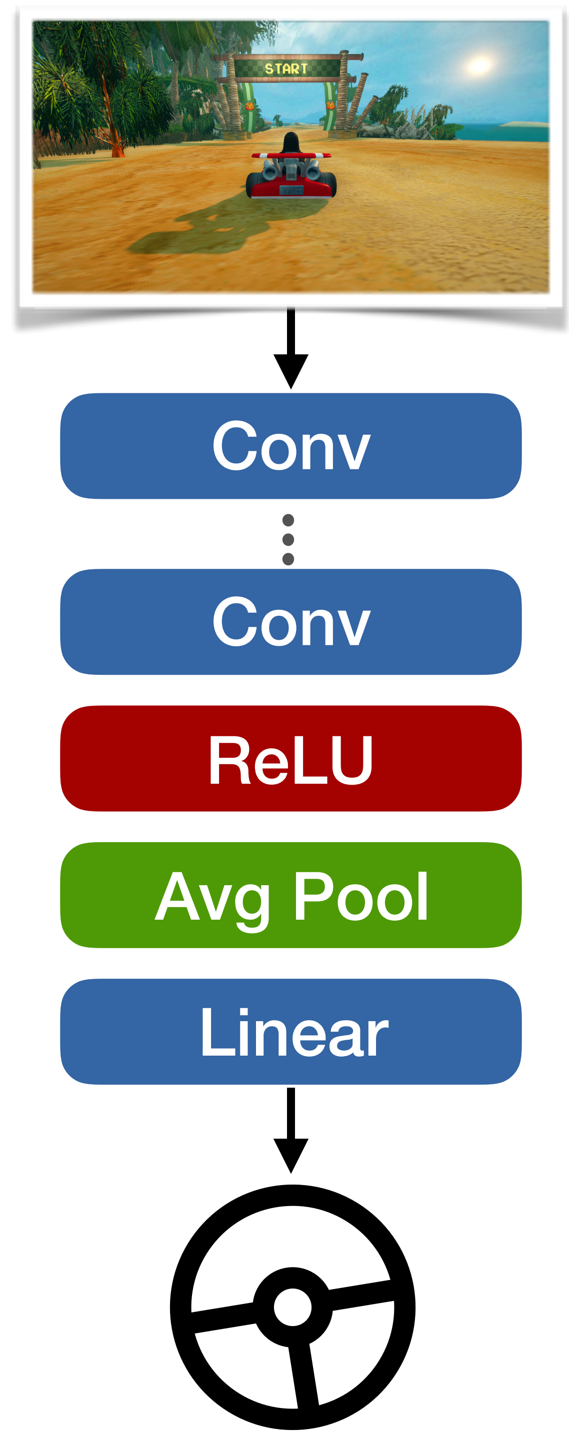
Feedforward models

- (Fixed) order of computation
    - Lower to upper layers
- Once we have the result
    - Discard all activations



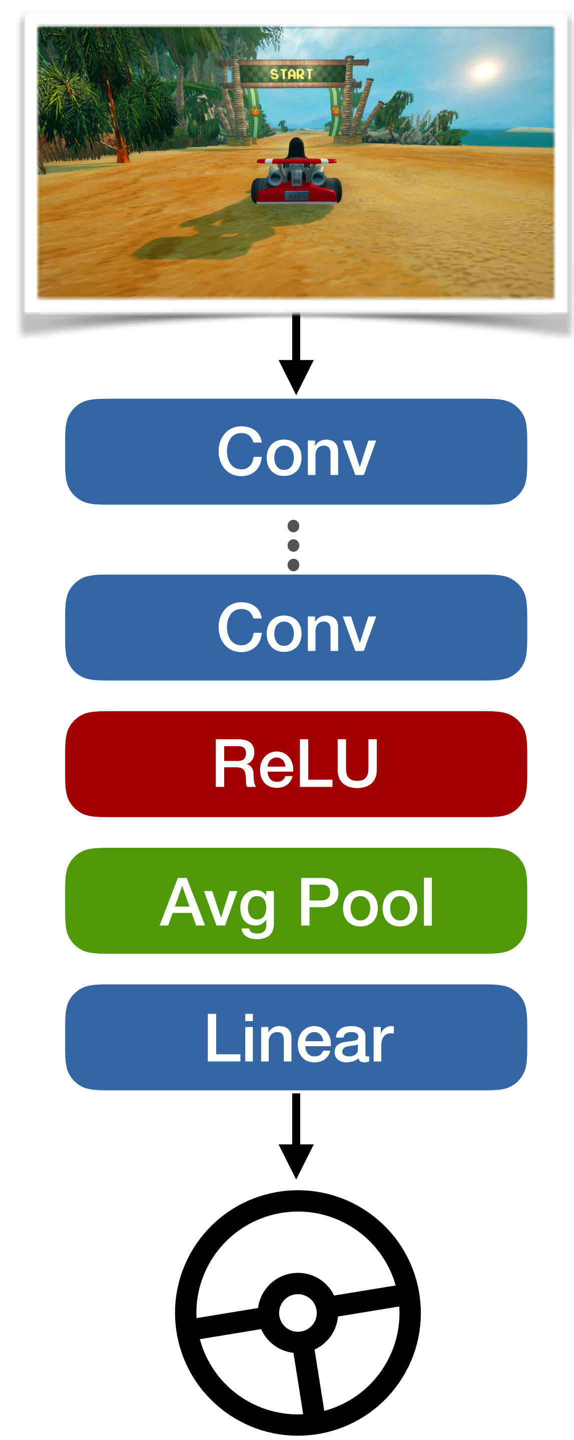
Would you use this to drive a car?

Would you use this to drive a car?

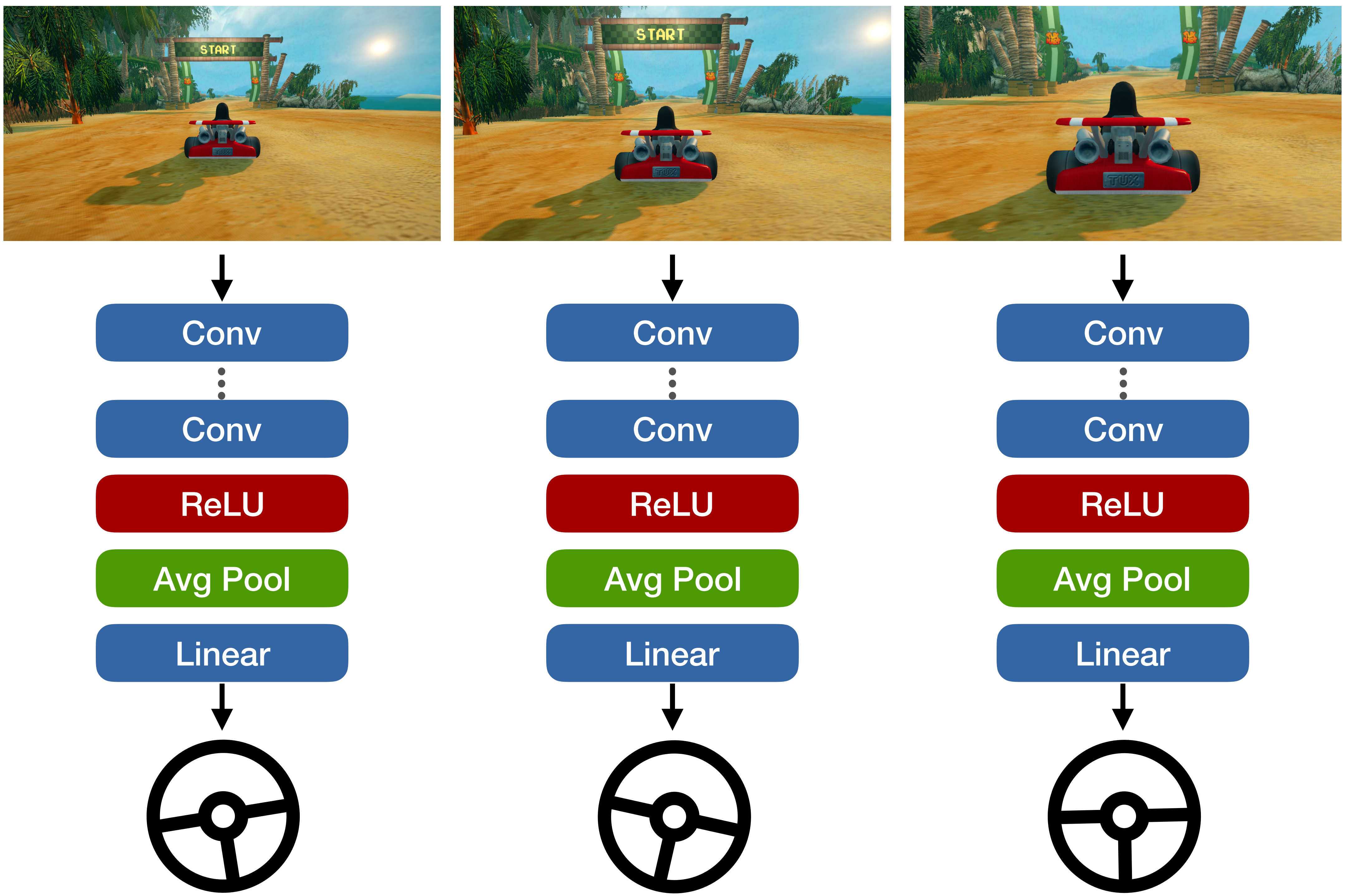


Would you use this to drive a car?

- Hopefully not
  - Independent decision for each frame
  - No state or memory



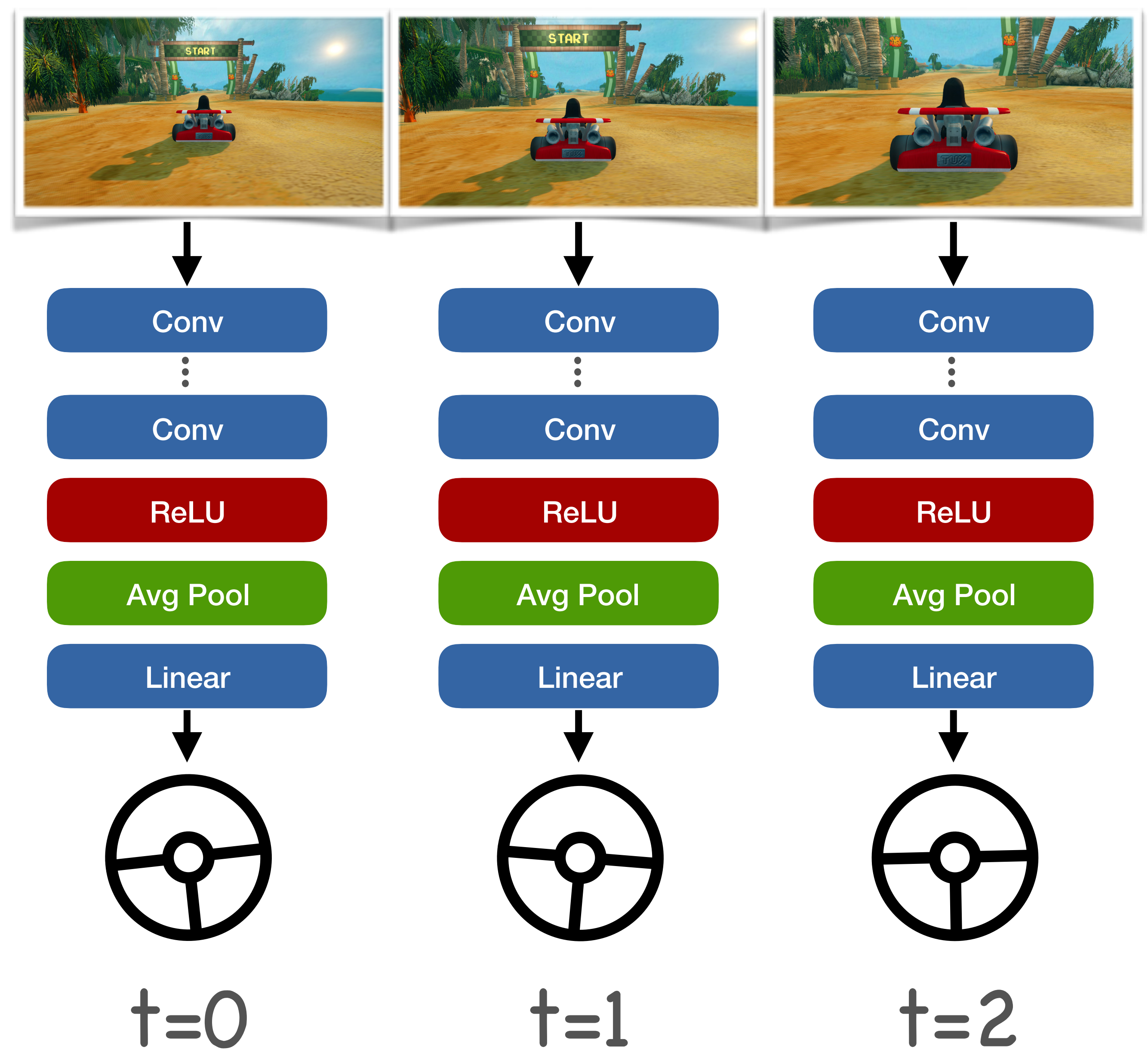
How should we keep state around?



Recurrent neural networks

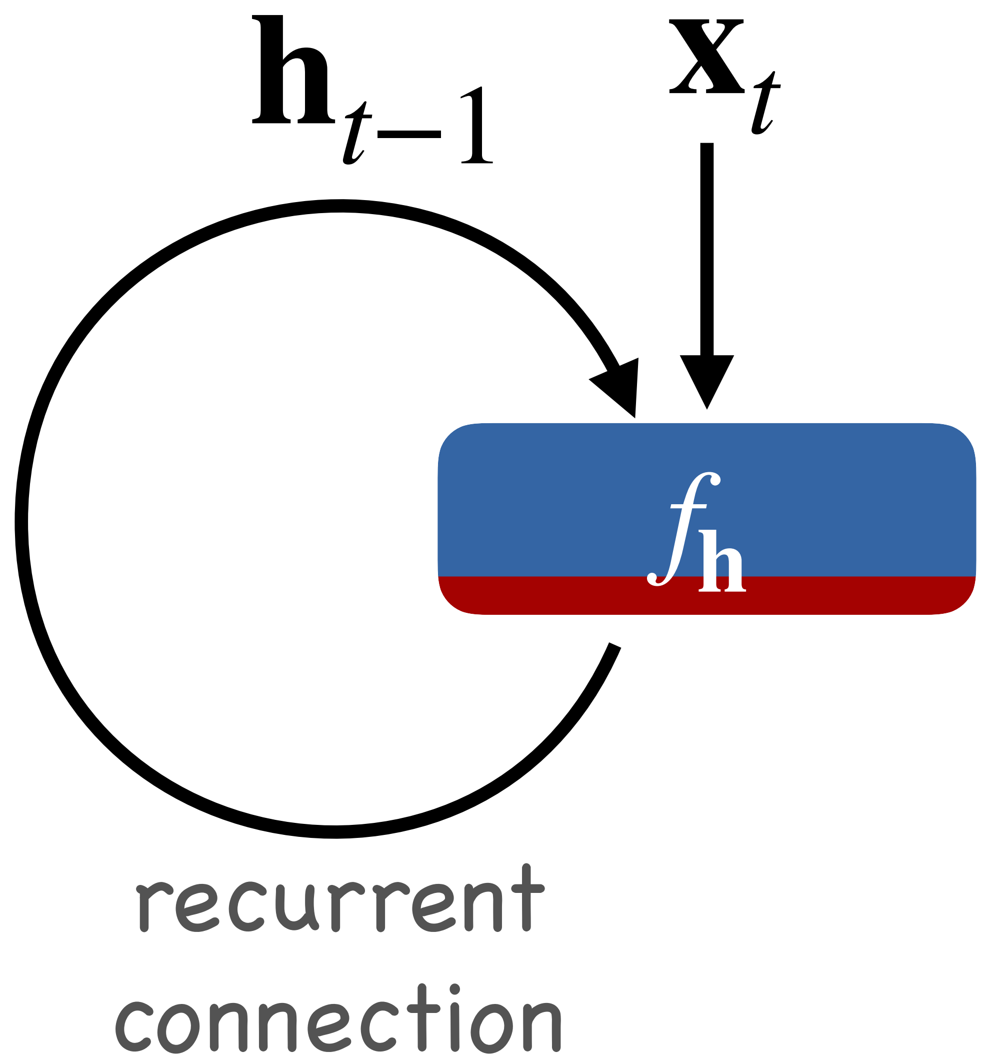
Recurrent neural networks (RNN)

- A network that
  - applies same computation multiple times
  - keeps some state around



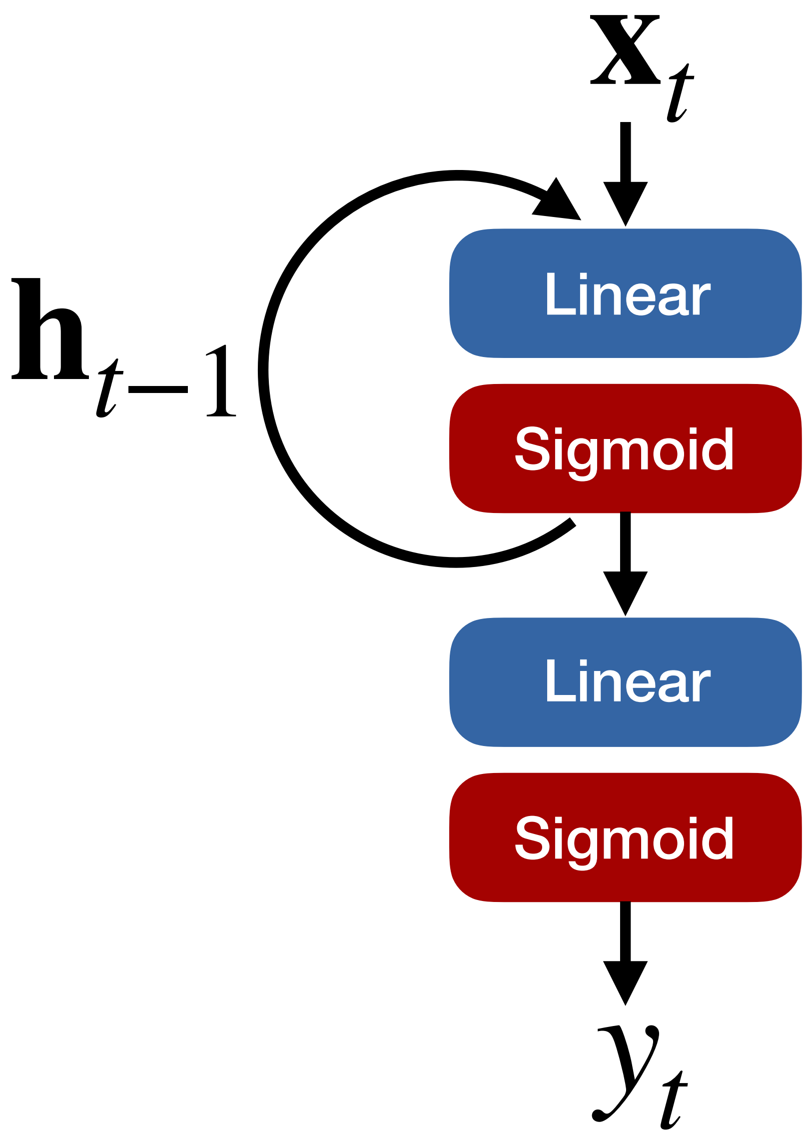
Formal definition

- Basic recurrent unit
  - $\mathbf{h}_t = f_{\mathbf{h}}(\mathbf{x}_t, \mathbf{h}_{t-1}, \theta_{\mathbf{h}})$
- Initial state $\mathbf{h}_0$
  - Learned
  - Zero



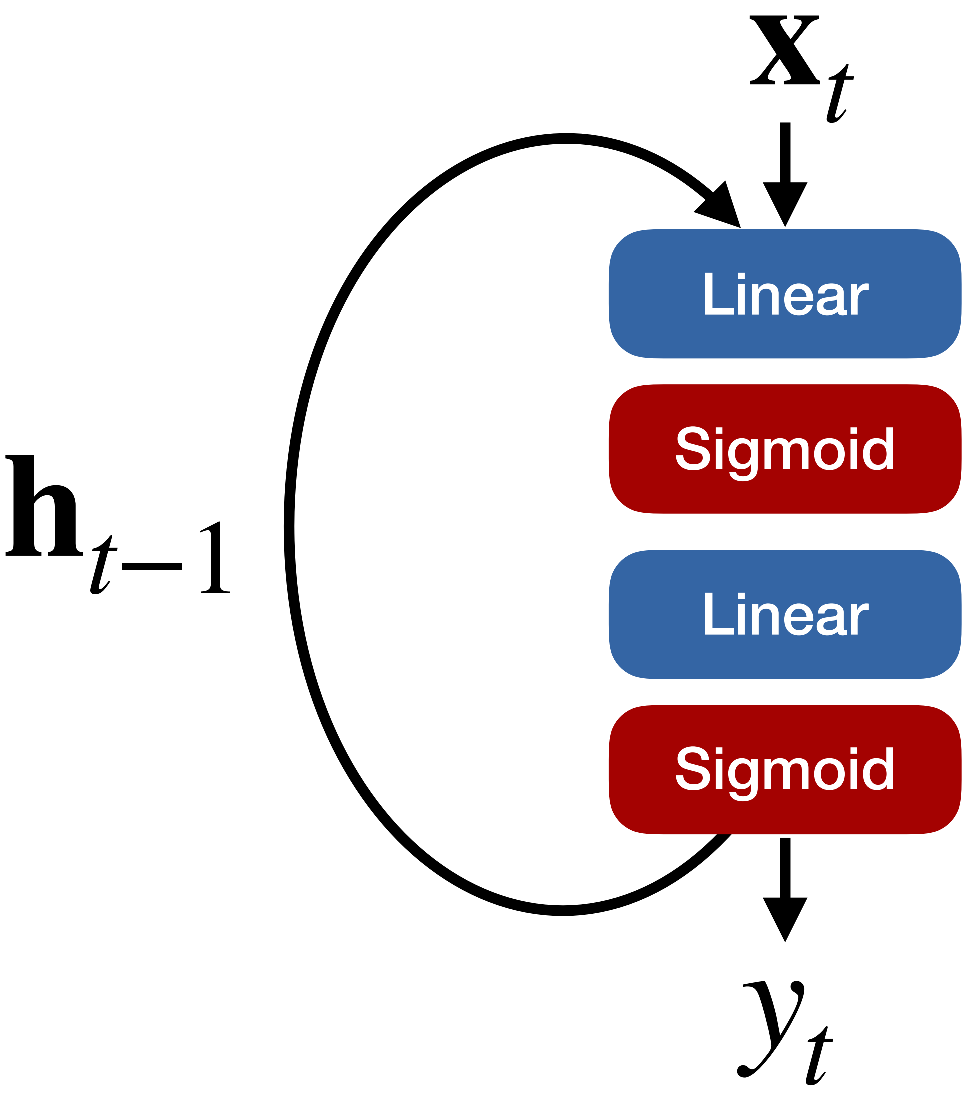
Elman networks

- Recurrent connection within layer



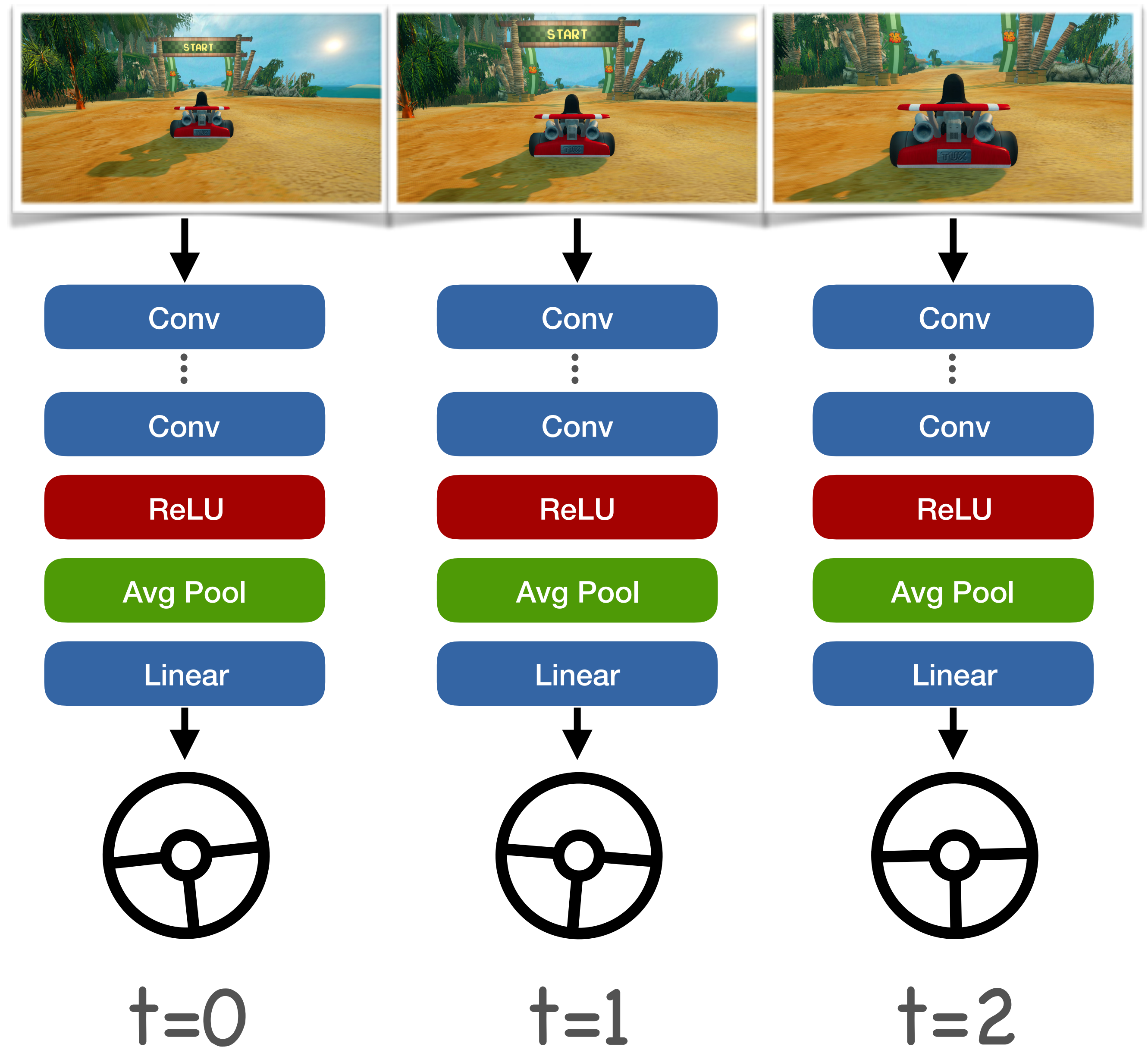
Jordan networks

- Recurrent connection from output to input



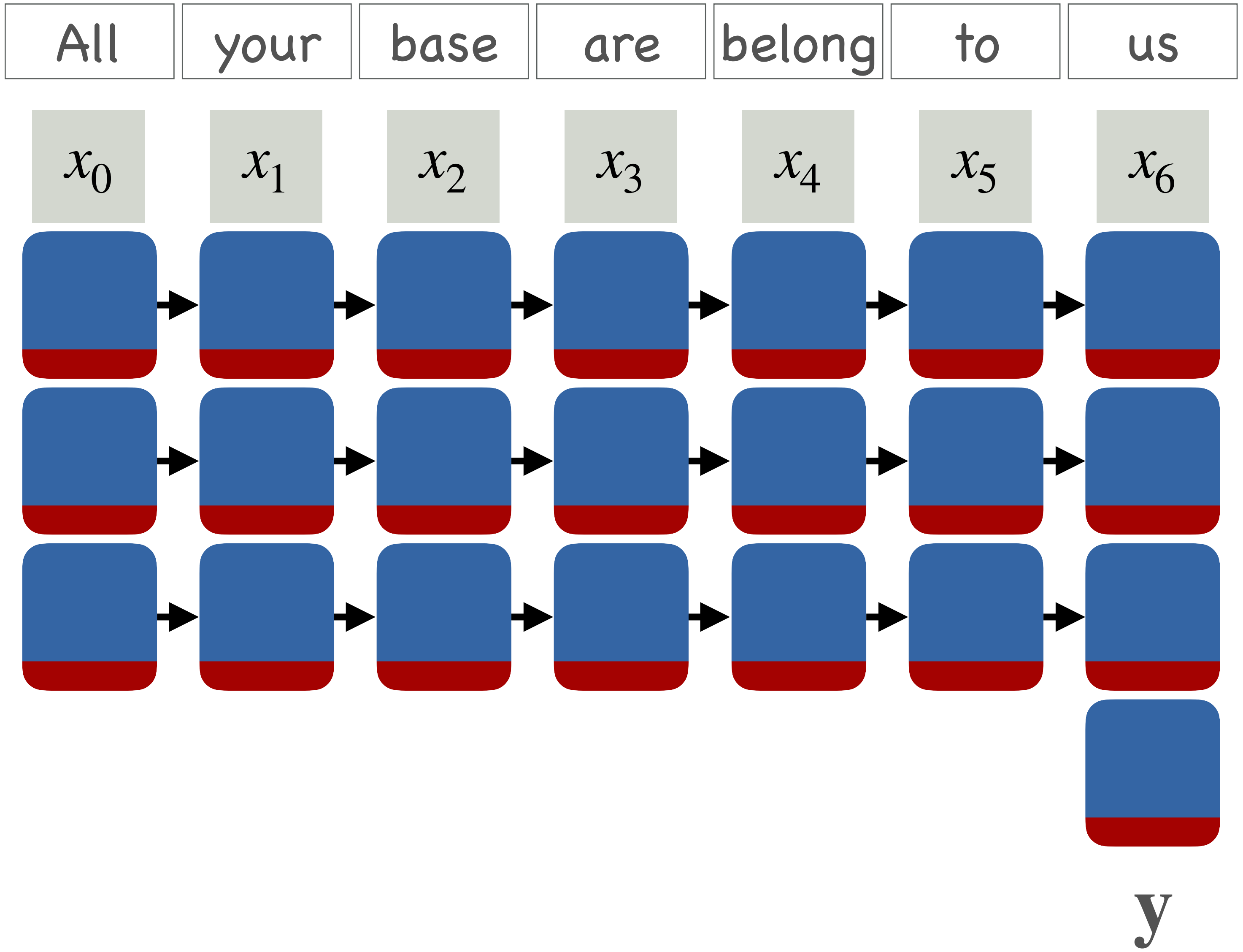
General RNNs

- Feed forward network
  - With feedback connections



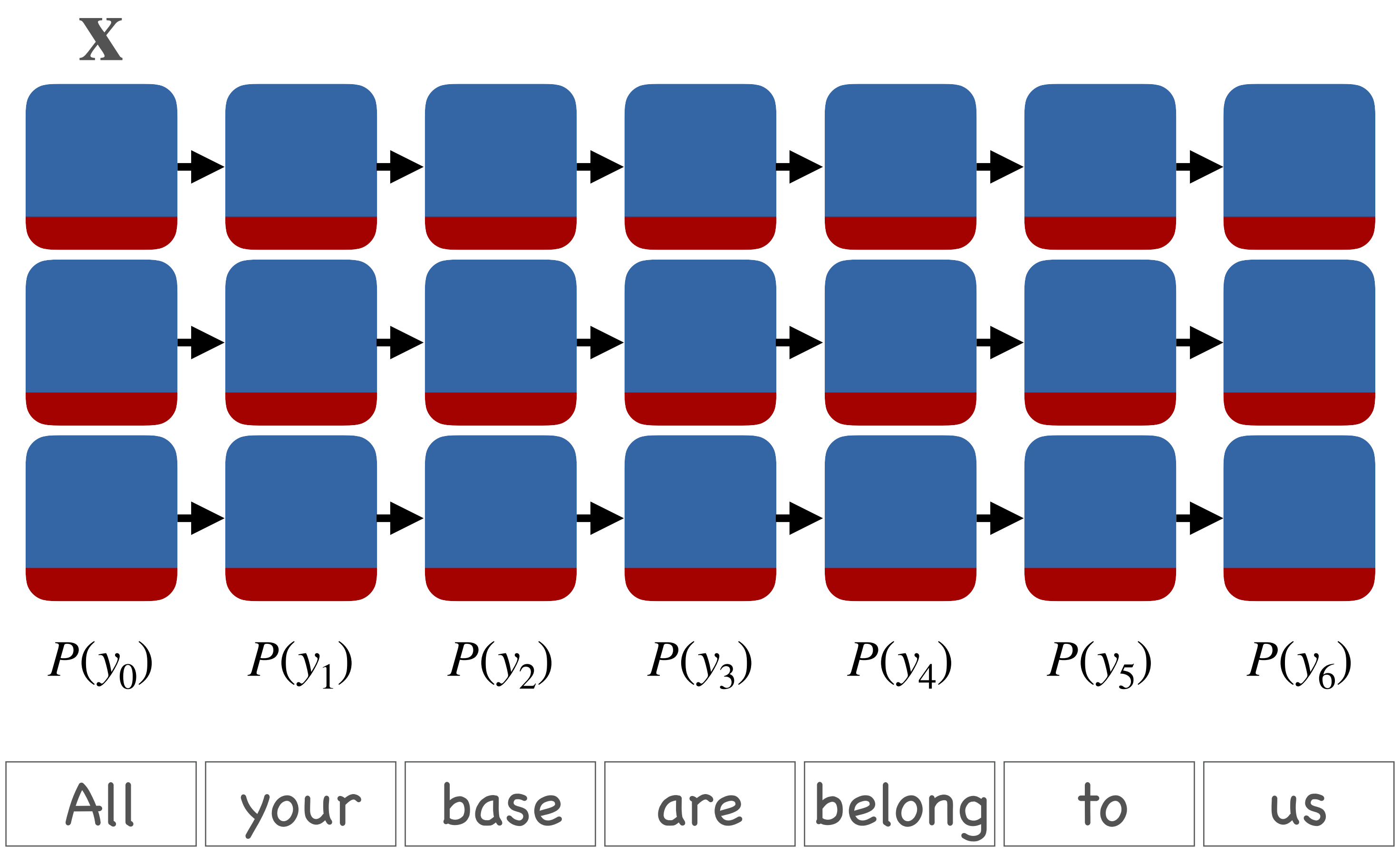
Example: Language understanding

- Reading comprehension
  - Sequence in
  - Vector out



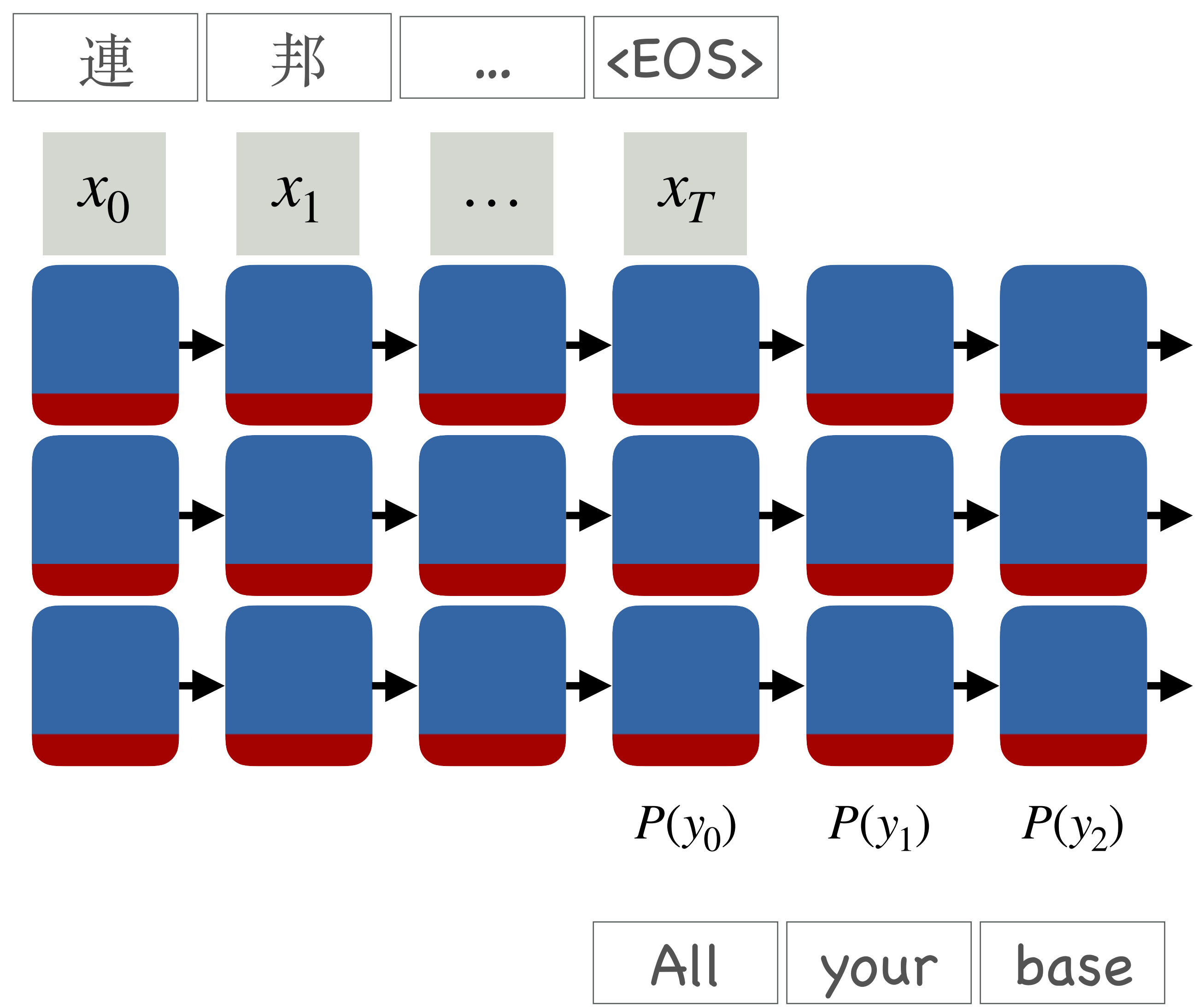
Example: Language generation

- Generate a sentence
  - Vector in
  - Sequence out

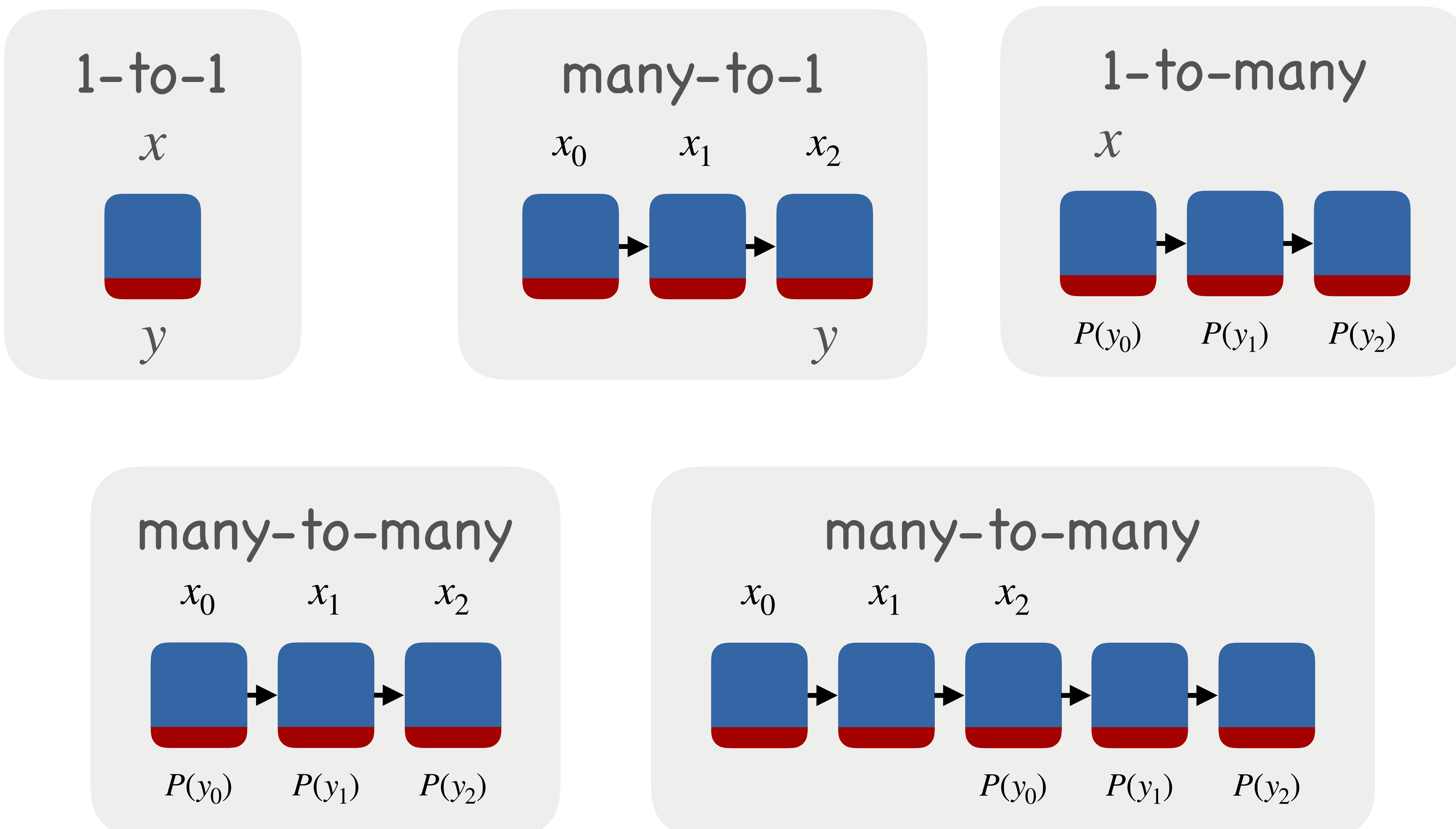


Example: Translation

- Translate sentence from one language to another
  - Sequence in
  - Sequence out



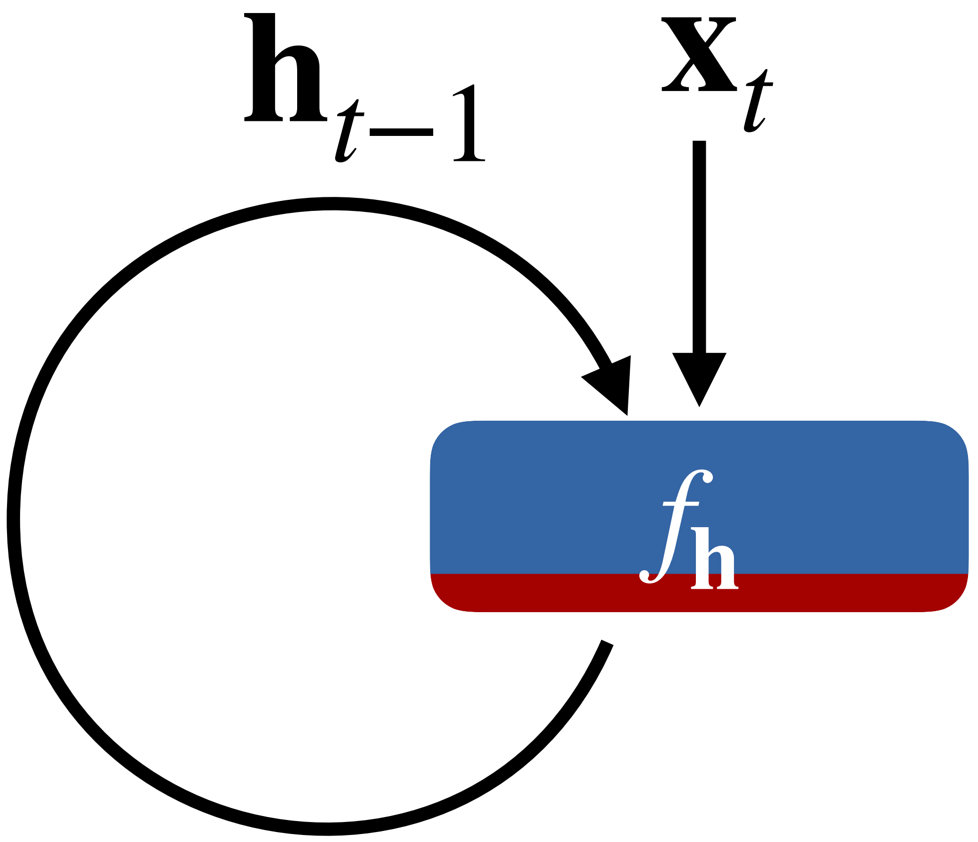
The many RNNs



Training recurrent networks

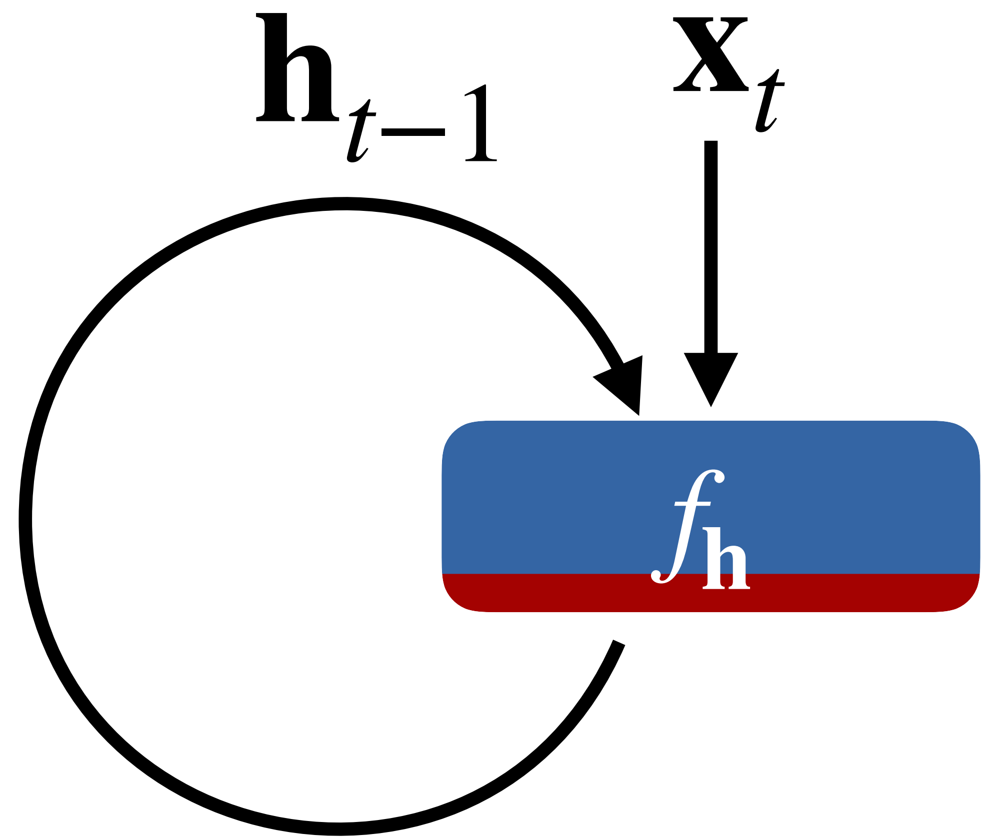
Recurrent Networks

- Processes a sequence
- Feedback connections

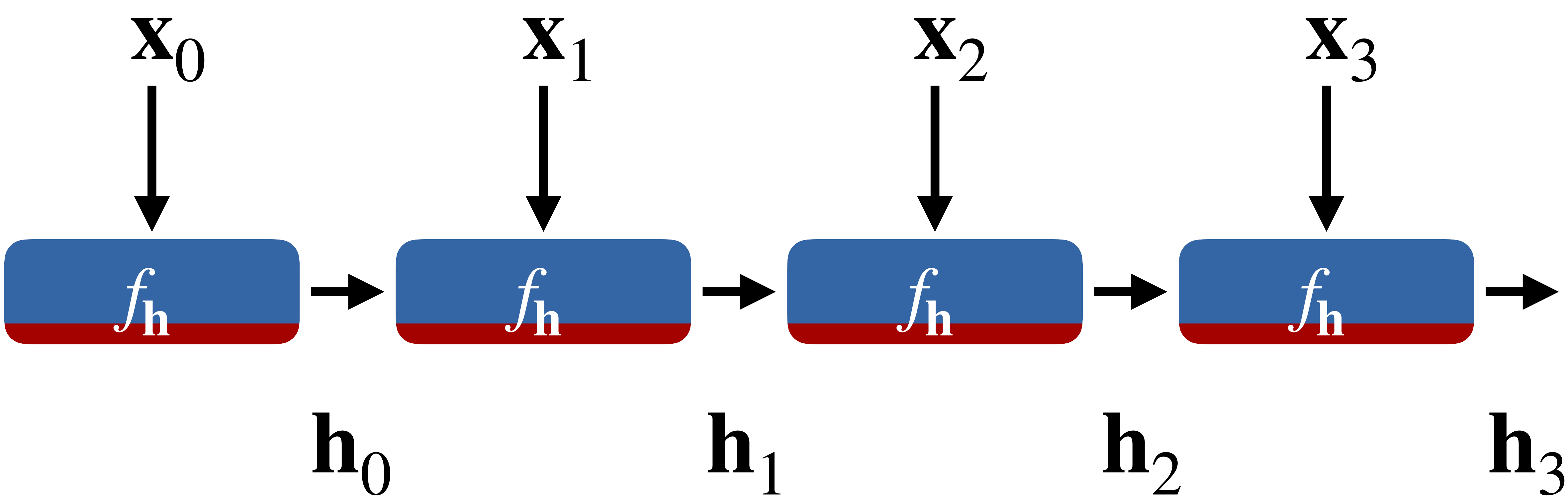


How do we train RNNs?

- No longer a simple forward and backward pass
  - Cycles

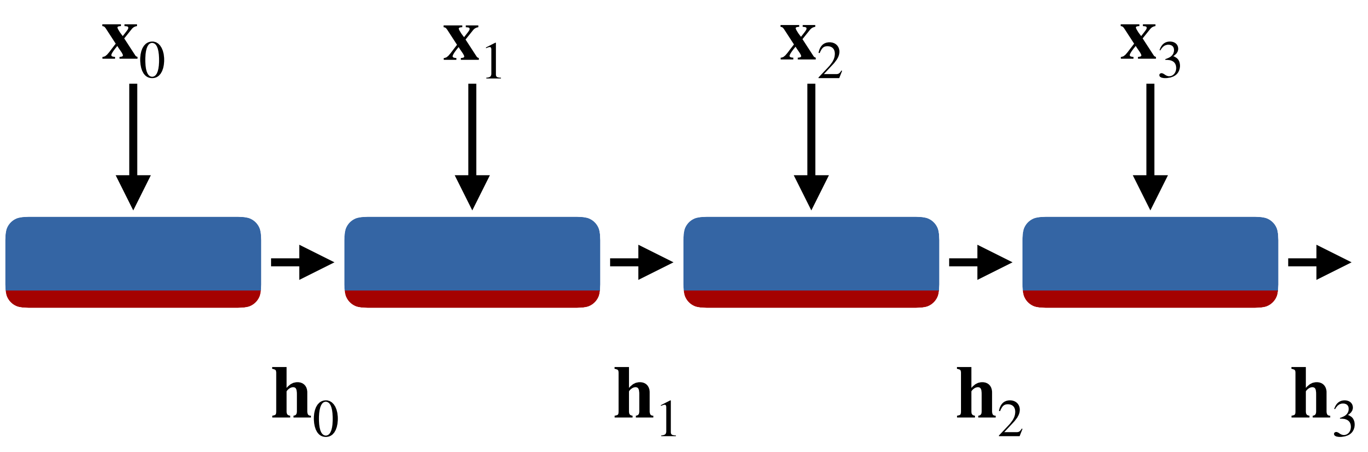


Solution: unrolling through time



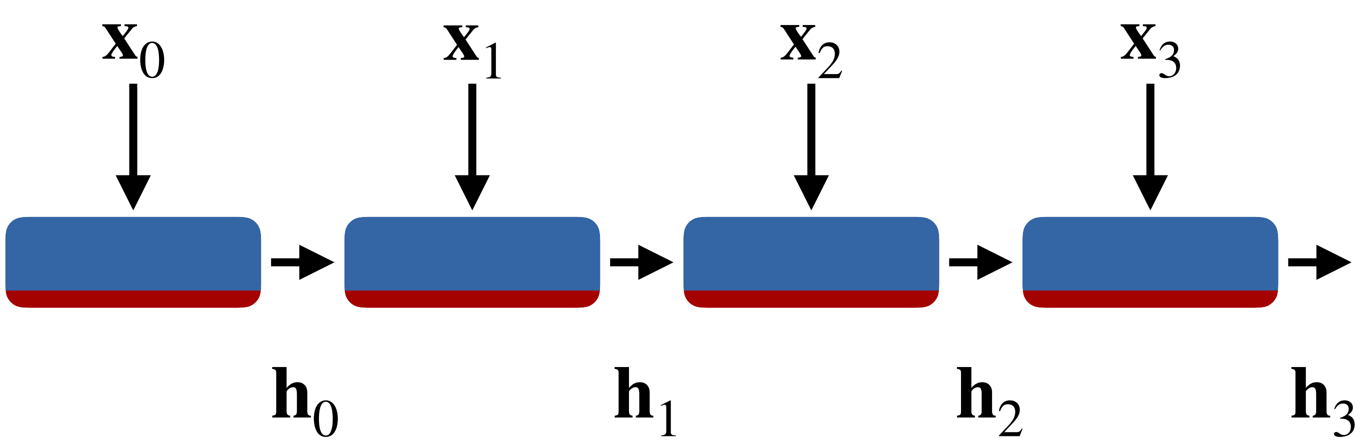
Unrolling through time

- Unrolled RNN (T steps)
  - Feed forward network
  - Shared parameters
- Trained with backprop



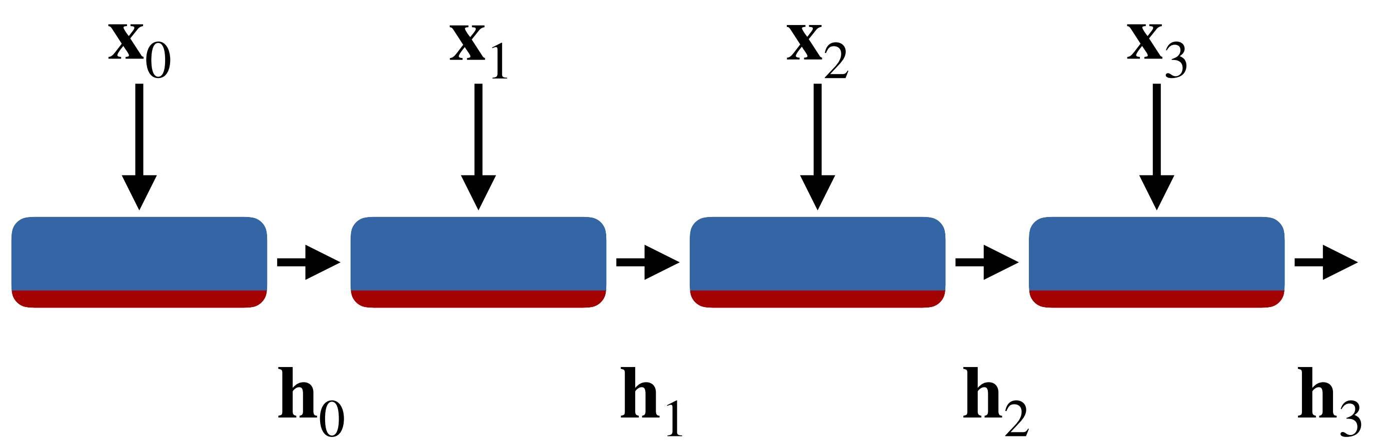
Unrolling through time - Issues

- Long unrolling
  - Computationally expensive
  - Vanishing or exploding gradients



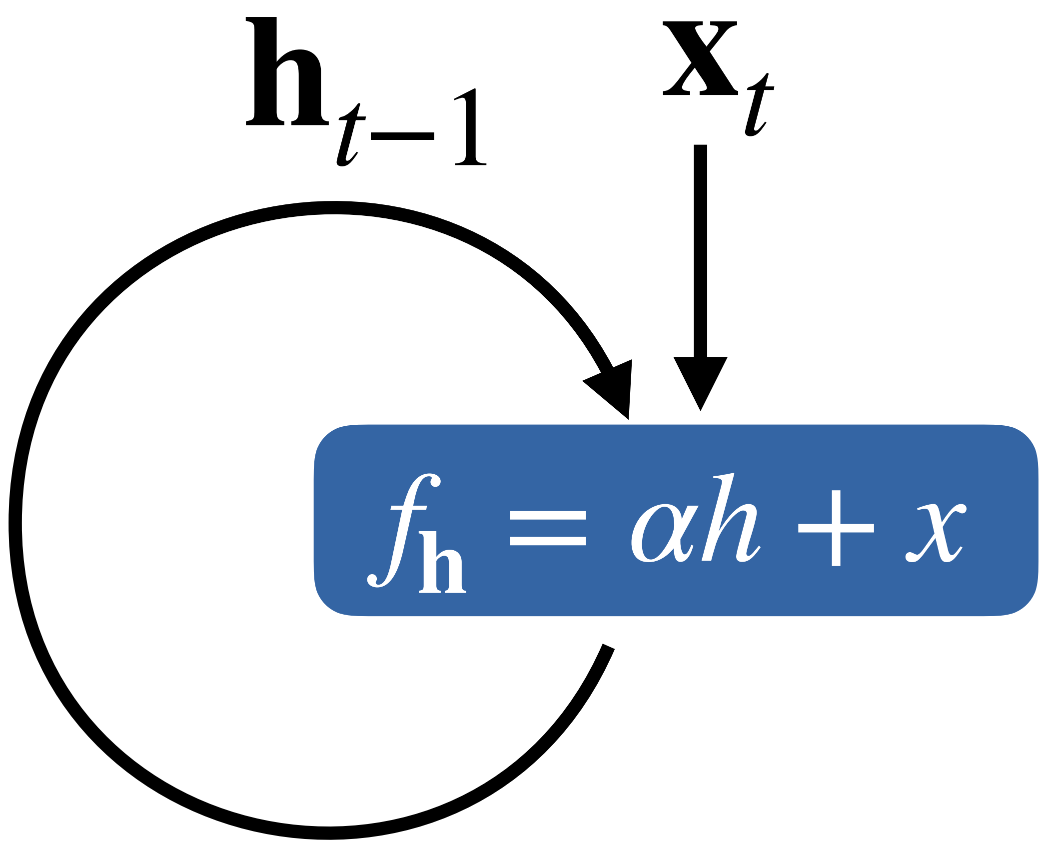
Computation

- Solution (hack)
  - Cut RNN after n steps
    - Set h=0
  - Works well in practice



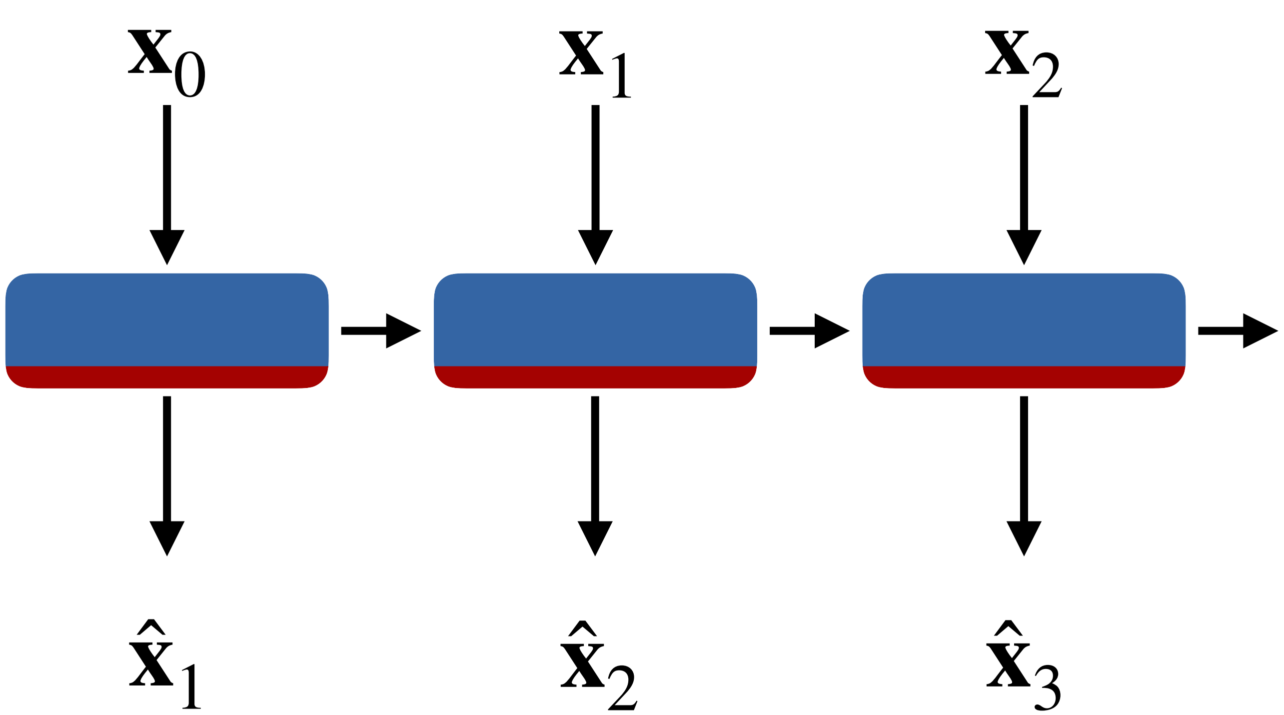
Vanishing and exploding gradients - Simple example

- Linear RNN
  - $\mathbf{h}_t = \alpha \mathbf{h}_{t-1} + \mathbf{x}_t$
- For large t
  - $a > 1 \quad \frac{\partial}{\partial \mathbf{x}_0} \mathbf{h}_t = \alpha^t \to \infty$
  - $a < 1 \quad \frac{\partial}{\partial \mathbf{x}_0} \mathbf{h}_t = \alpha^t \to 0$



Preventing vanishing and exploding gradients

- Generative models
  - Use ground truth inputs



Preventing vanishing and exploding gradients

- Exploding gradients
  - Gradient clipping

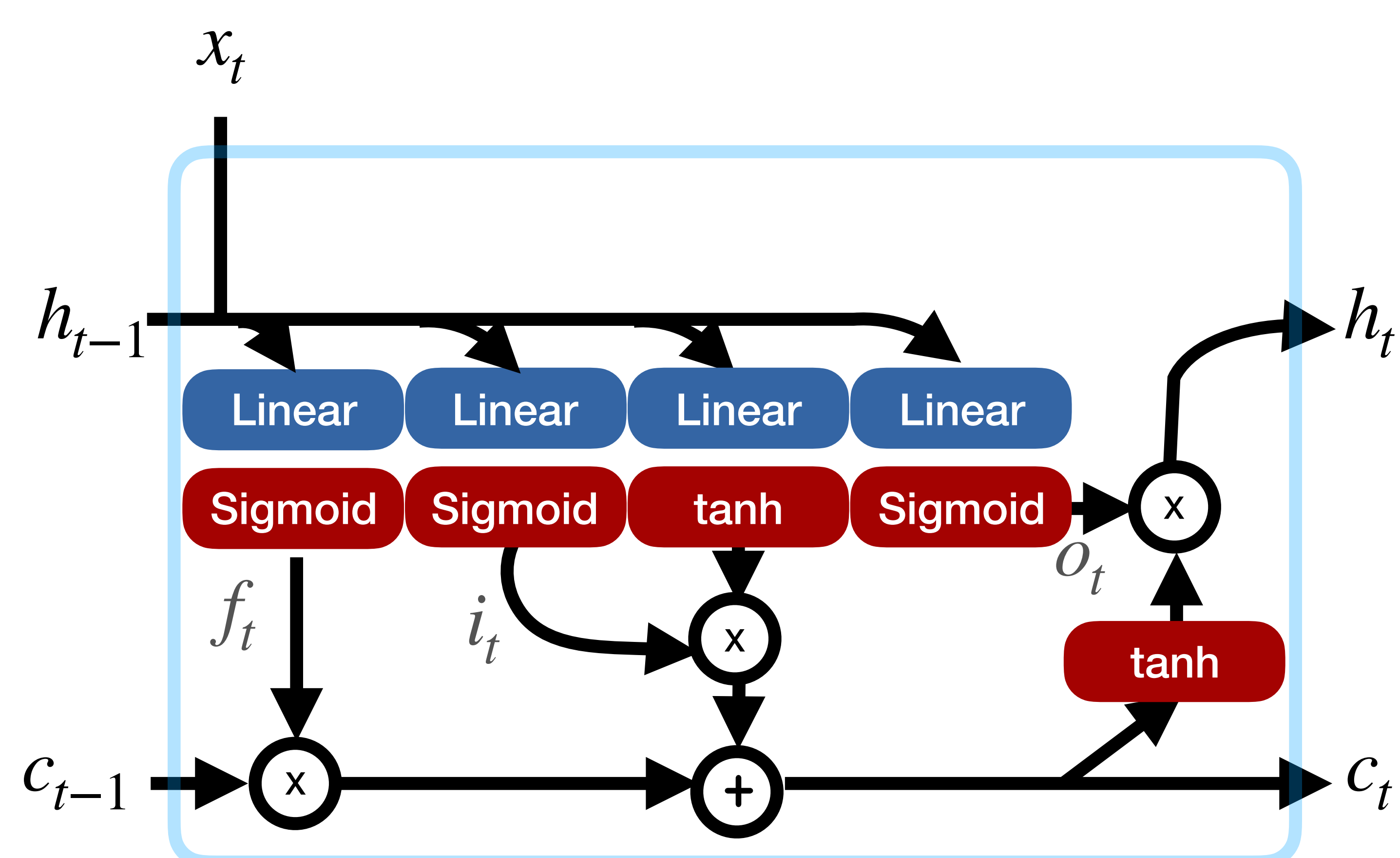
$$\nabla \ell' = \nabla \ell \min\left(1, \frac{\epsilon}{|\nabla \ell|}\right)$$

- Vanishing gradients
  - Different RNN structure

LSTMs and GRUs

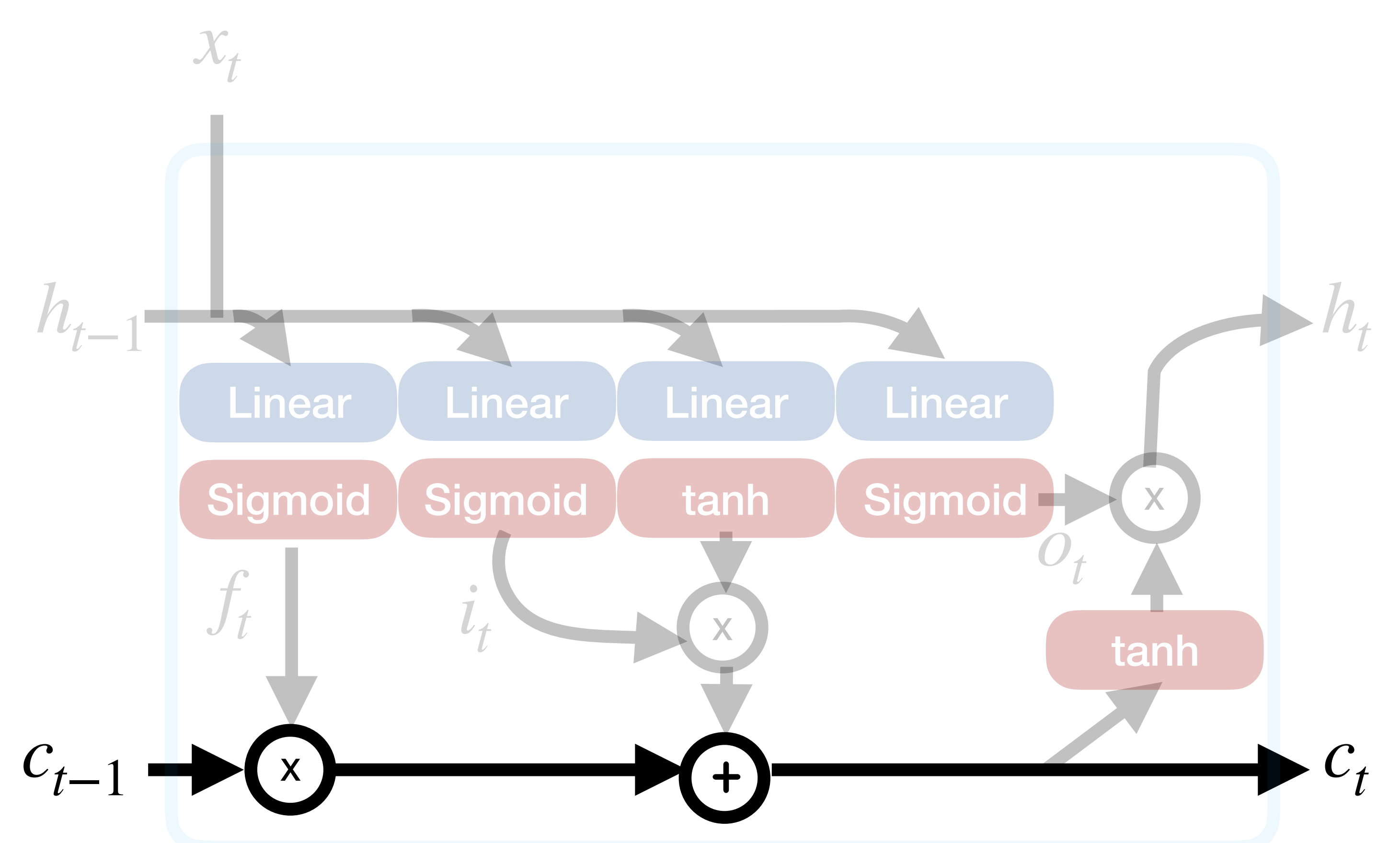
Long short-term memory

- Two recurrent connections
  - Long-term c
  - Short term h
  - Input x
  - Output h



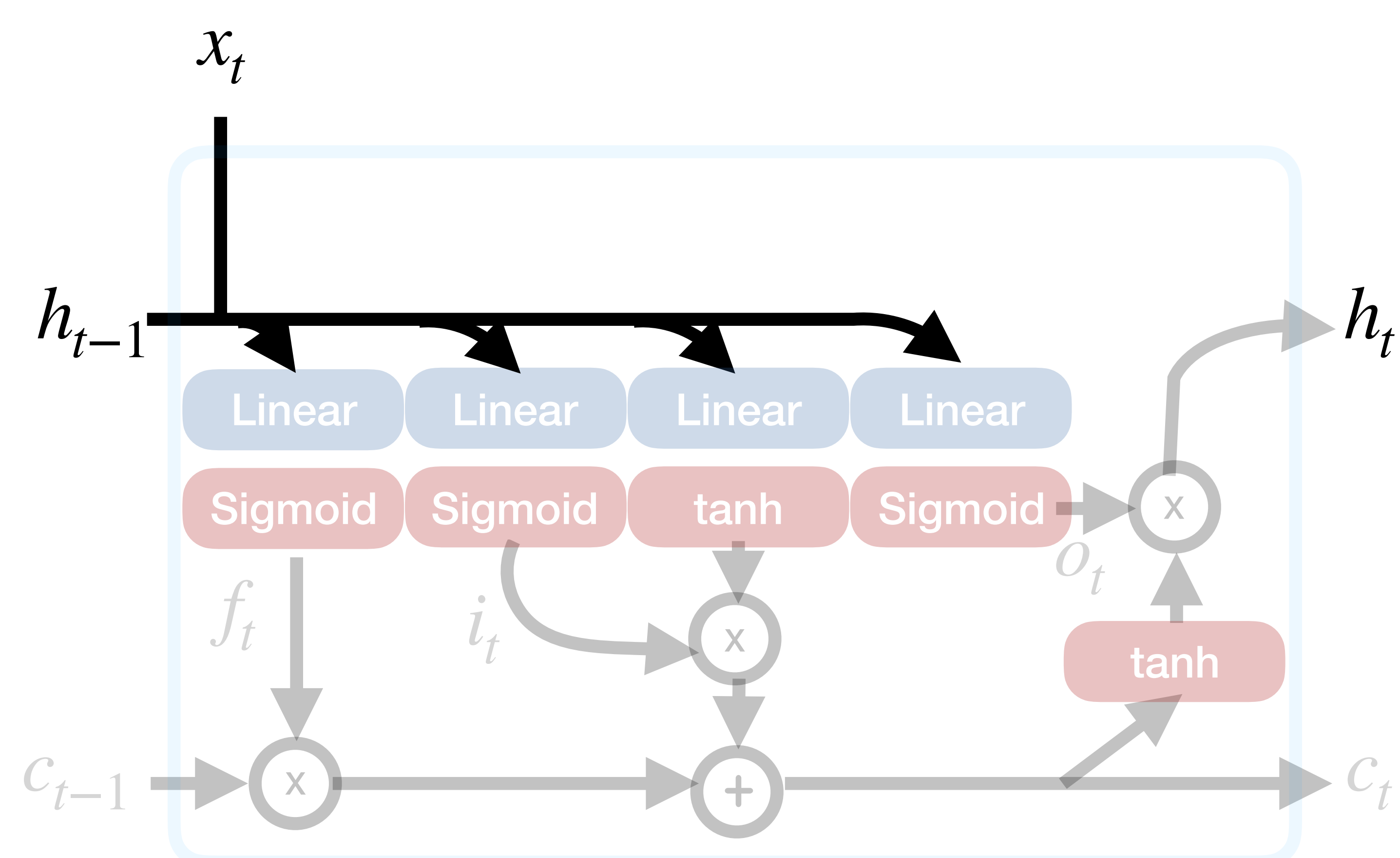
Cell state

- Cell state c
  - Only multiplication and addition
  - Shortcut
    - Similar to ResNets



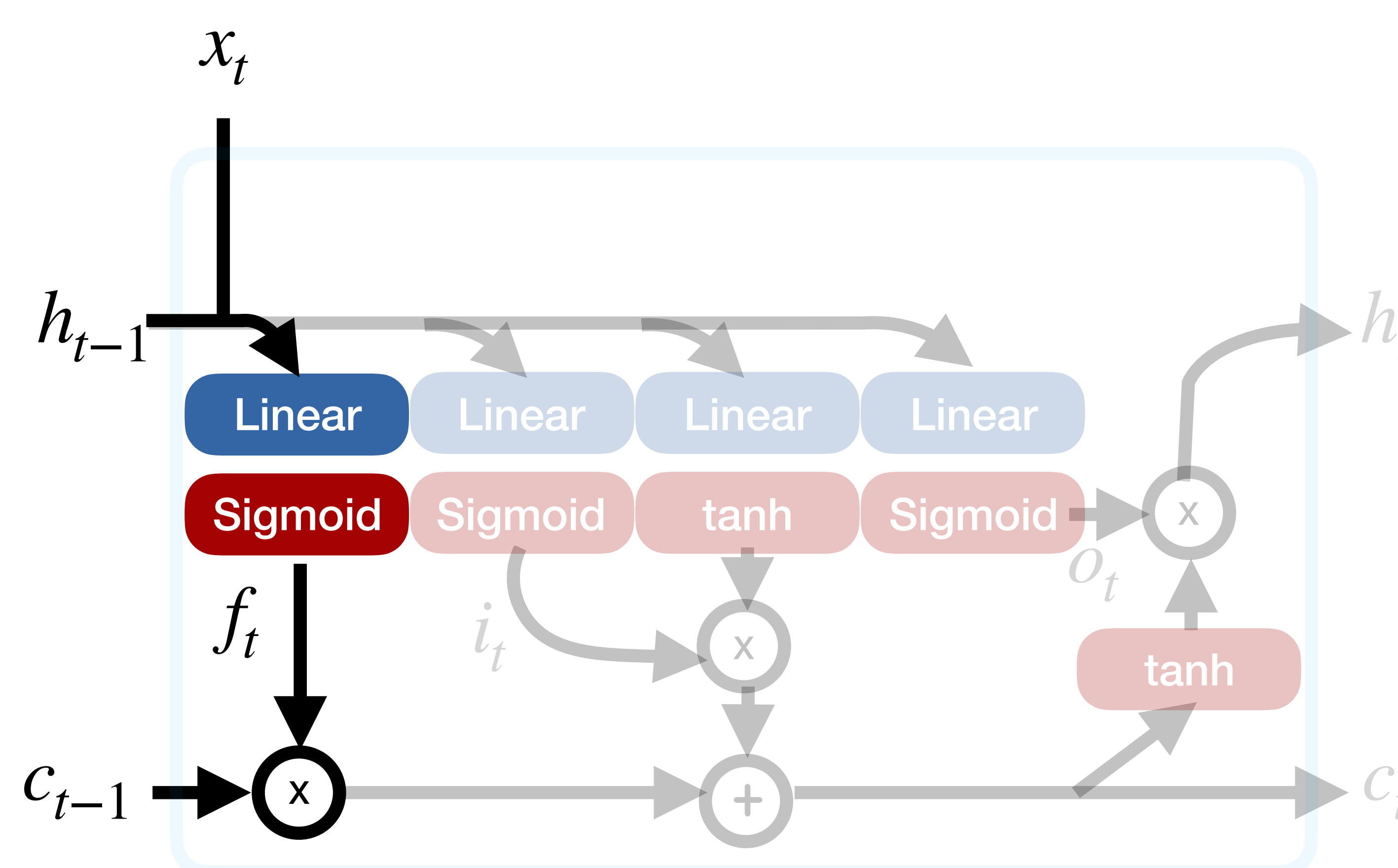
Input

- Input
  - x
  - Previous h



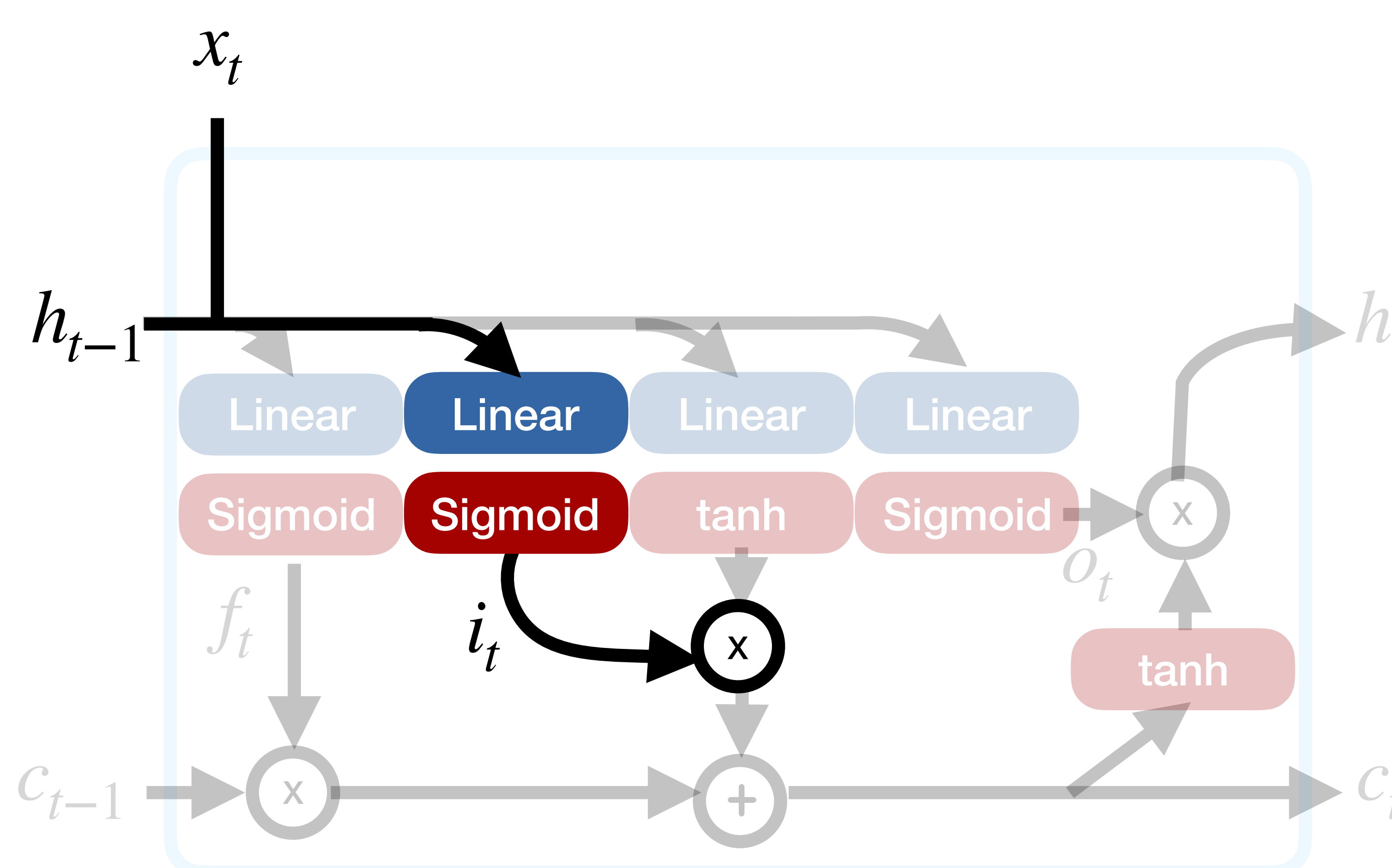
Forget gate

- Forget gate f
  - Clears cell state



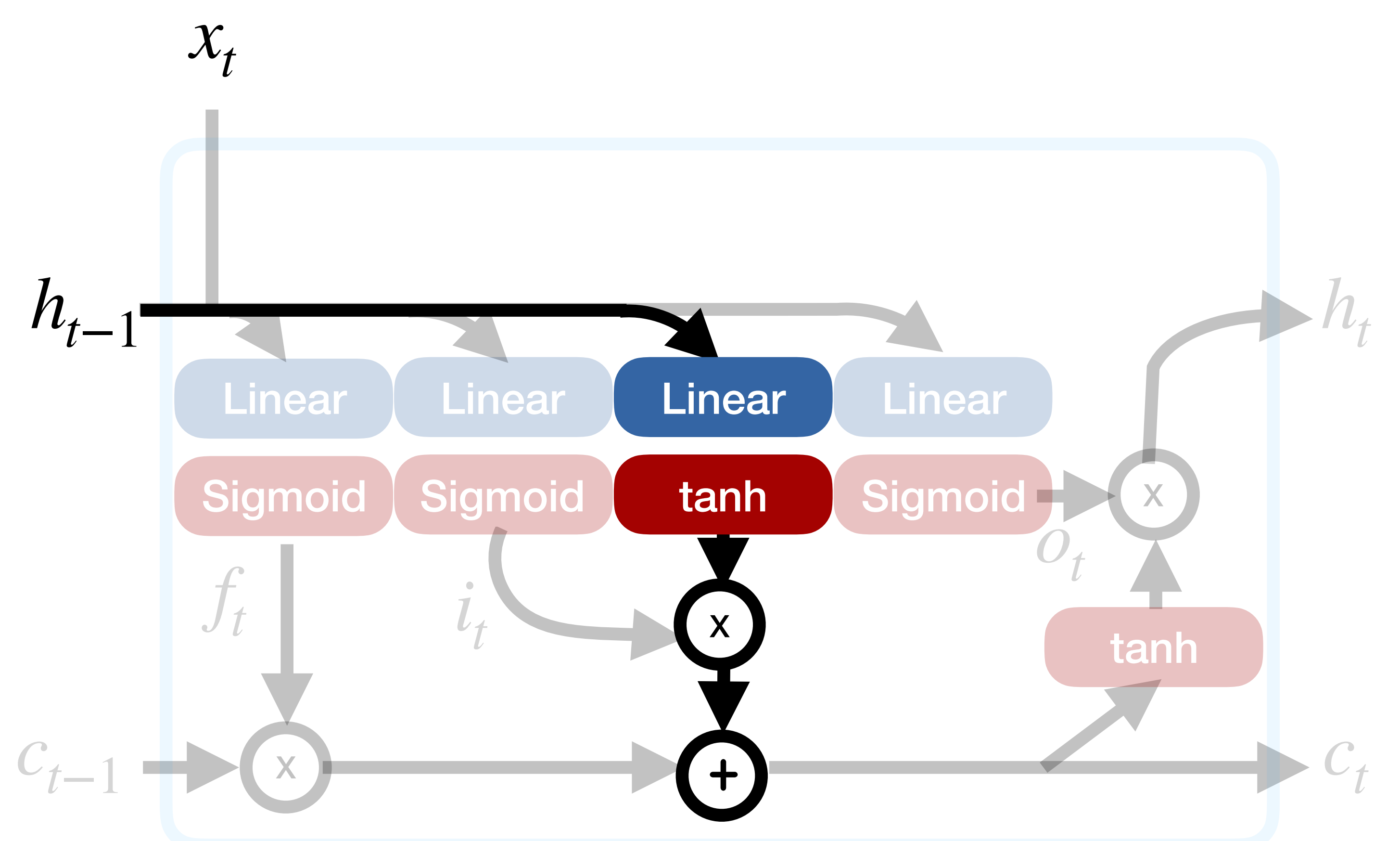
Input gate

- Input gate i
  - Allows state update



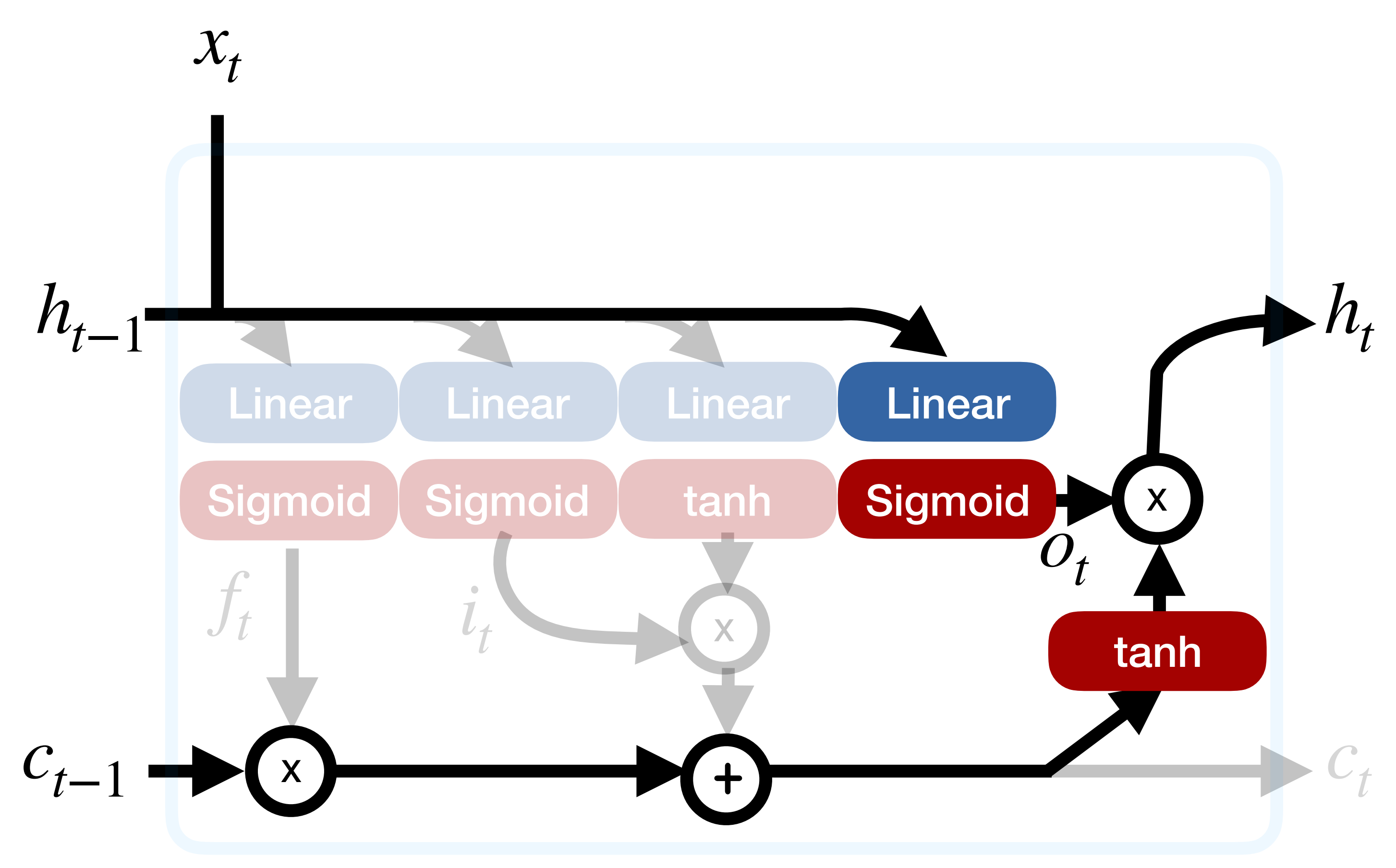
State update

- State update
  - tanh



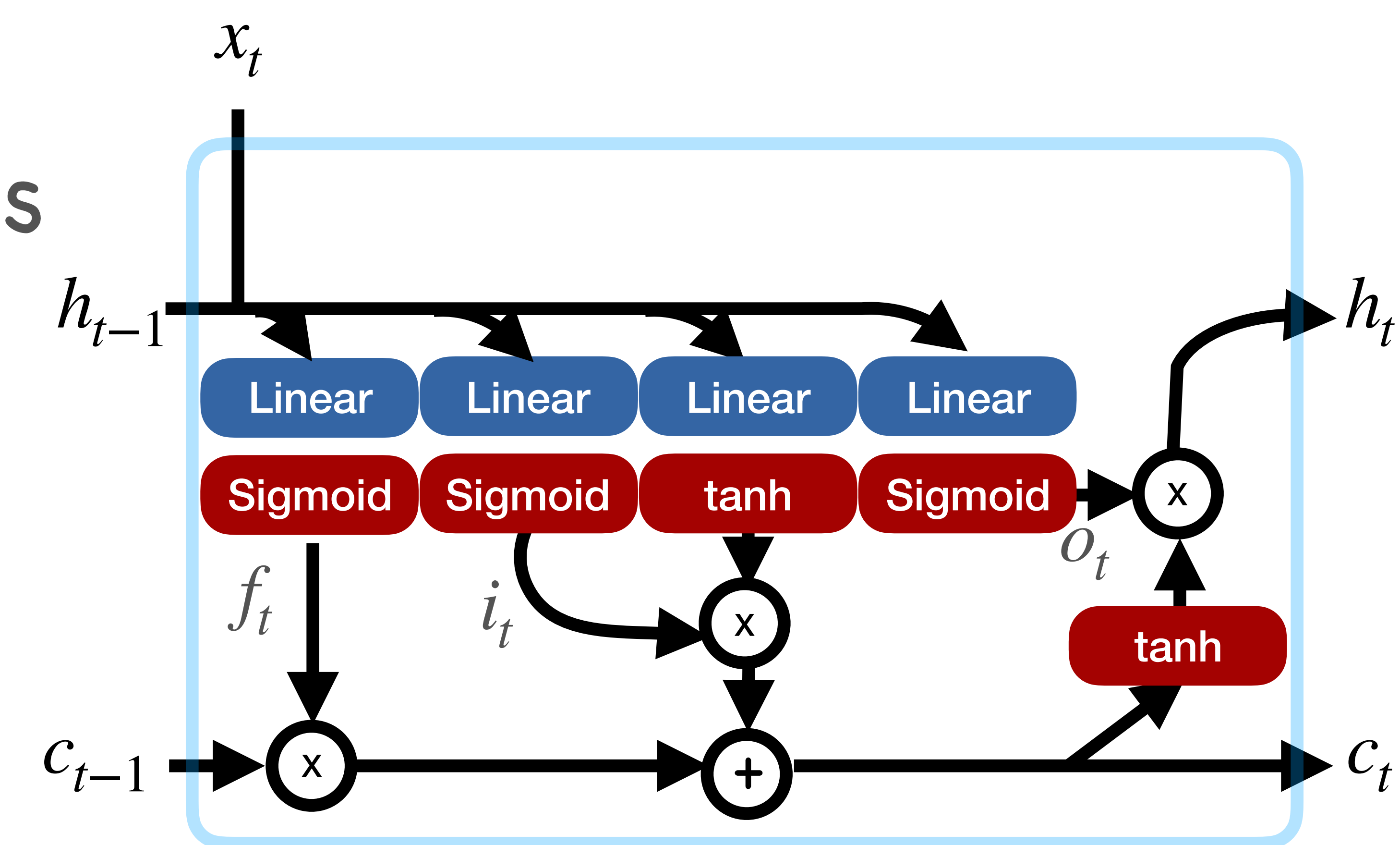
Output

- Output gate o
  - Produce an output?
- Output h
  - tanh of cell state



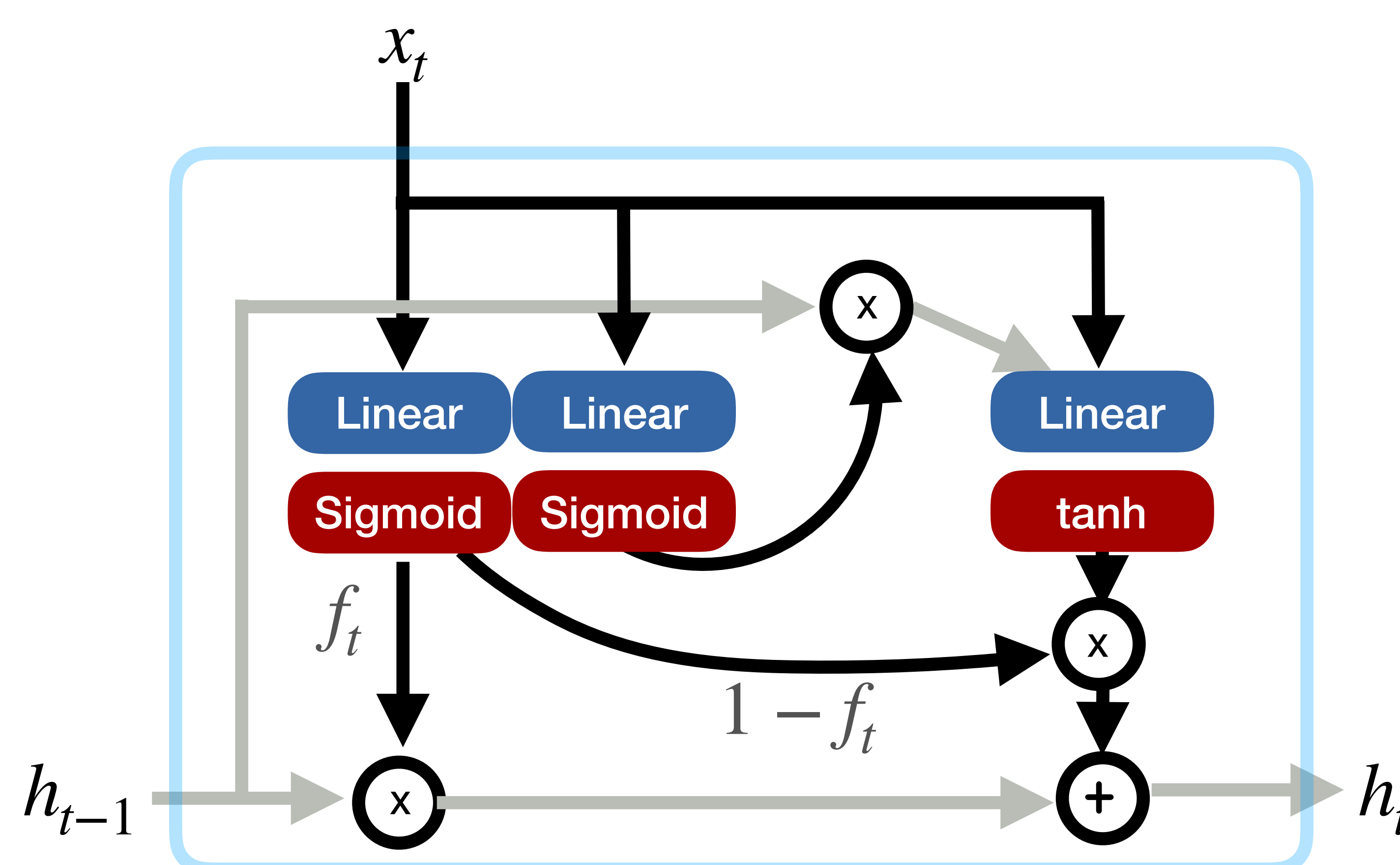
LSTMs

- Can learn to keep state for up to 100 time steps
    - Fewer vanishing gradients
    - Short cut

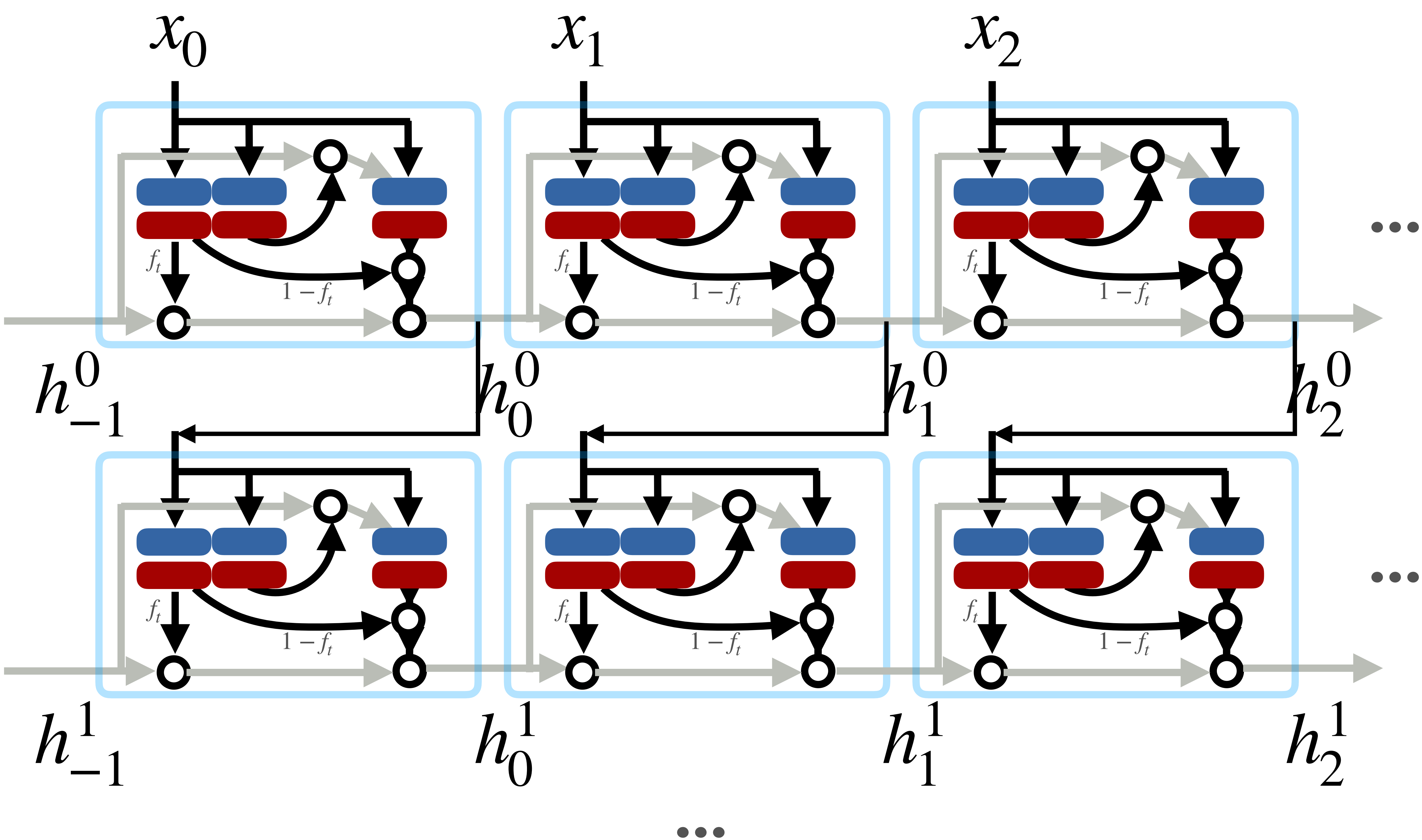


Gated Recurrent Units

- Simpler LSTM
  - Single state
  - Fewer gates
  - Similar performance



LSTM/GRU Networks



LSTM / GRU applications

- Hand writing synthesis
- Natural language processing
- Image generation

Image source: Demo by Alex Graves
http://www.cs.toronto.edu/~graves/

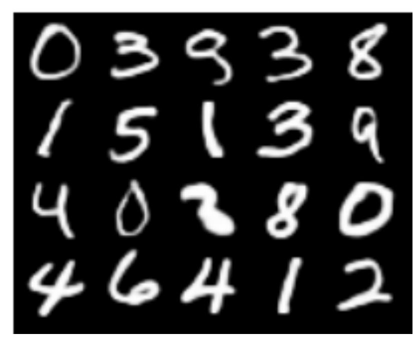


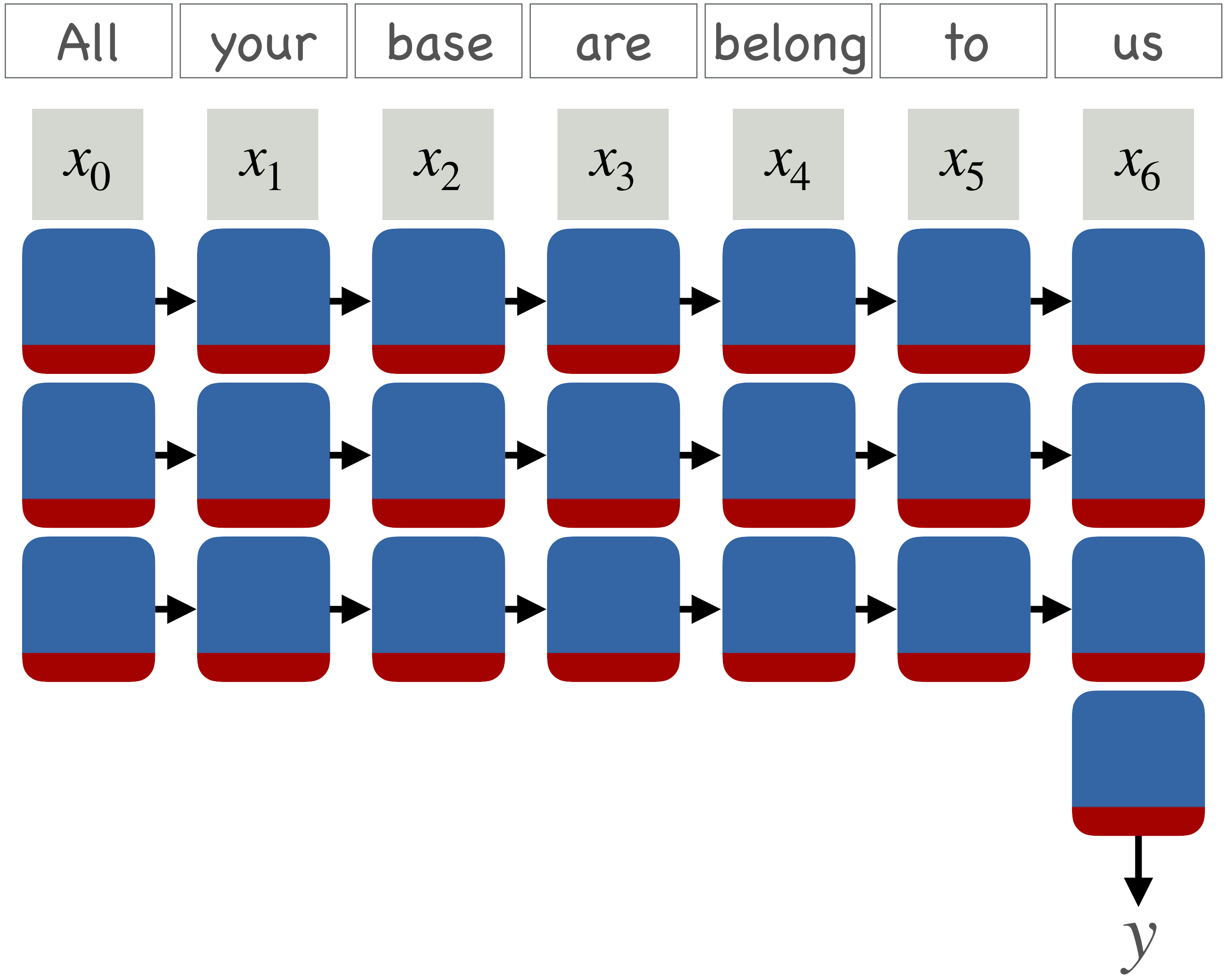
Image source: Gregor et al., https://arxiv.org/pdf/1502.04623.pdf

- Generating Sequences With Recurrent Neural Networks, Graves, arXiv 2013
- Sequence to Sequence Learning with Neural Networks, Sutskever et al., NIPS 2014
- DRAW: A Recurrent Neural Network For Image Generation, Gregor et al., ICML 2015

Temporal convolutions

Recurrent models

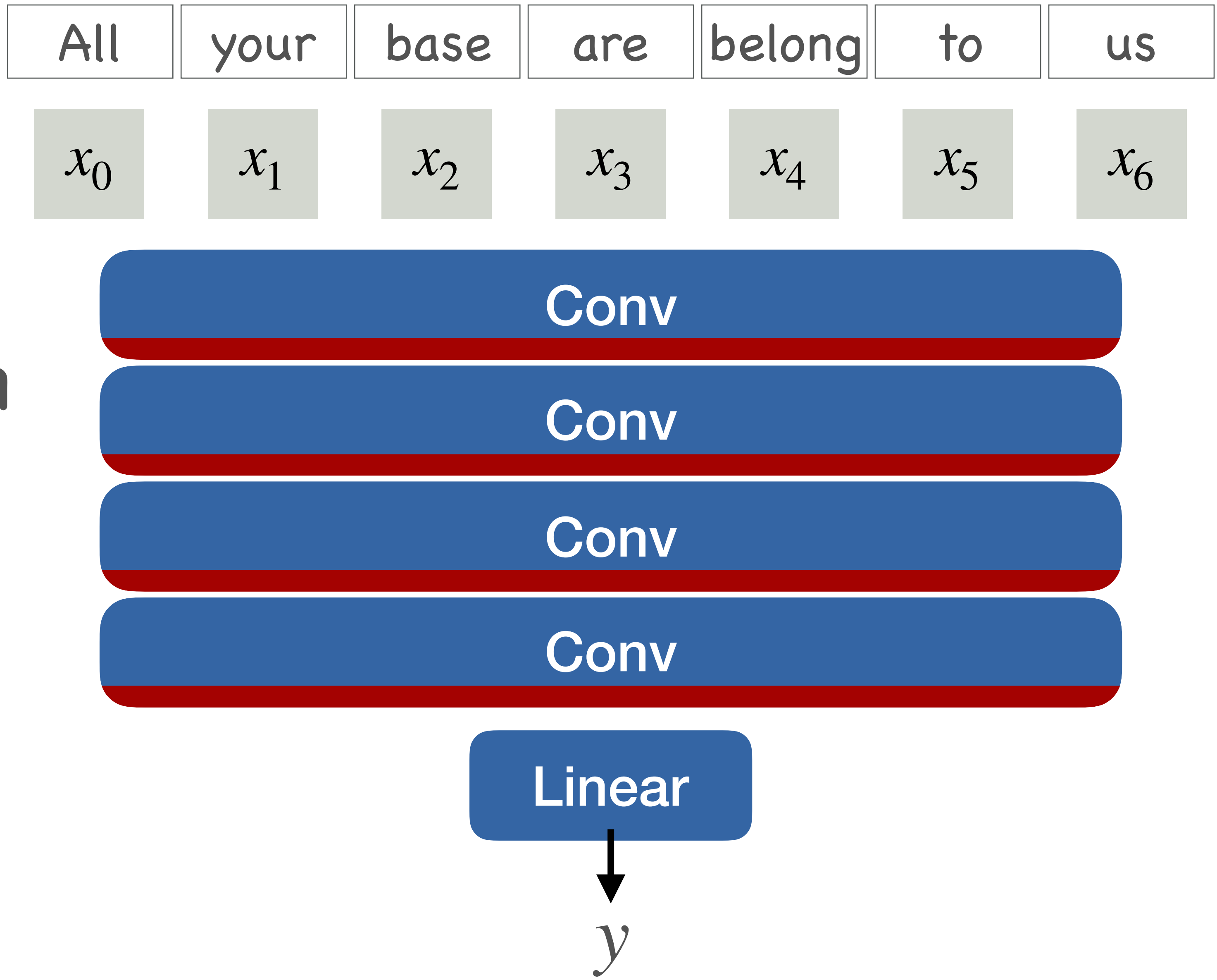
- Advantages
  - Variable input length
  - Variable output length
  - Structured output
- Disadvantage
  - Hard to train
  - Cannot learn dependencies longer than 100 steps



An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling, Bai etal., arXiv 2018

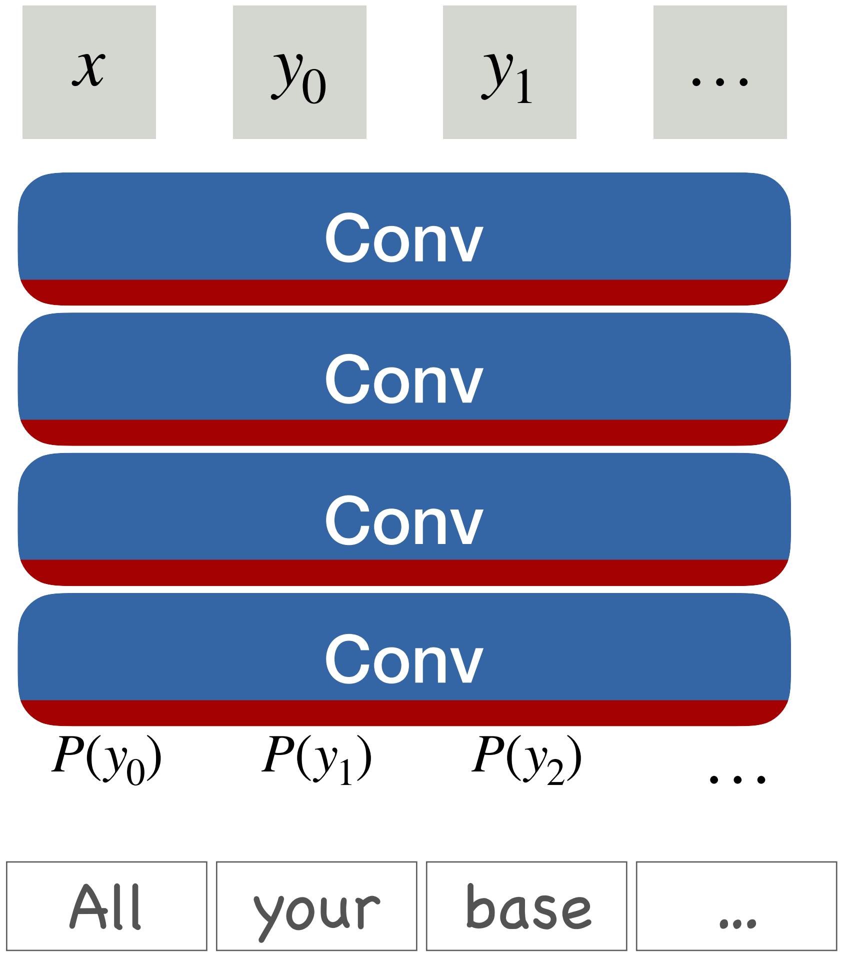
Temporal convolutional networks

- Dilated convolution
  - Exponential growth in receptive field
  - 5-10 layers, receptive field > 100 steps



Sequence generation using convolutions

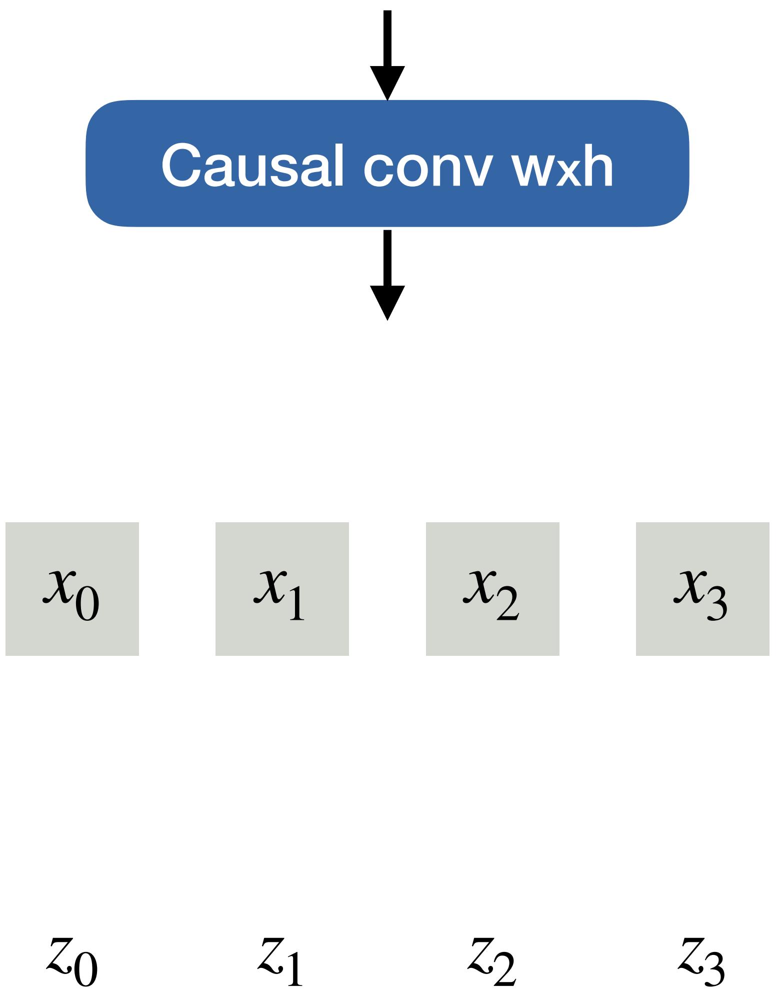
- Causal (masked) convolutions
  - Only look into past
- Auto-regressive model
  - $P(y_0 \mid x) \cdot P(y_1 \mid x, y_0) \cdot P(y_2 \mid x, y_0, y_1) \cdot \ldots$



- Conditional image generation with pixelcnn decoders, Van den Oord et al., *NIPS* 2016
- WaveNet: A generative model for raw audio, Van Den Oord et al., *arXiv* 2016

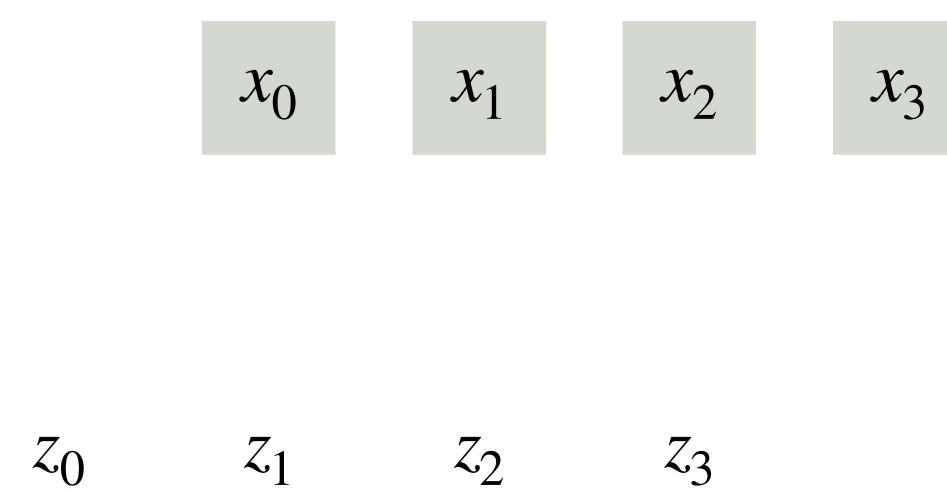
Causal convolution

- Input: $\mathbf{X} \in \mathbb{R}^{T \times C_1}$
- Kernel: $\mathbf{w} \in \mathbb{R}^{w \times C_1 \times C_2}$
- Bias: $\mathbf{b} \in \mathbb{R}^{C_2}$
- Output: $\mathbf{Z}_{t,b} = \mathbf{b}_c + \sum_{i=0}^{w-1} \sum_{j=0}^{C_1-1} \mathbf{X}_{t+i-w,b+j} \mathbf{w}_{i,j}$



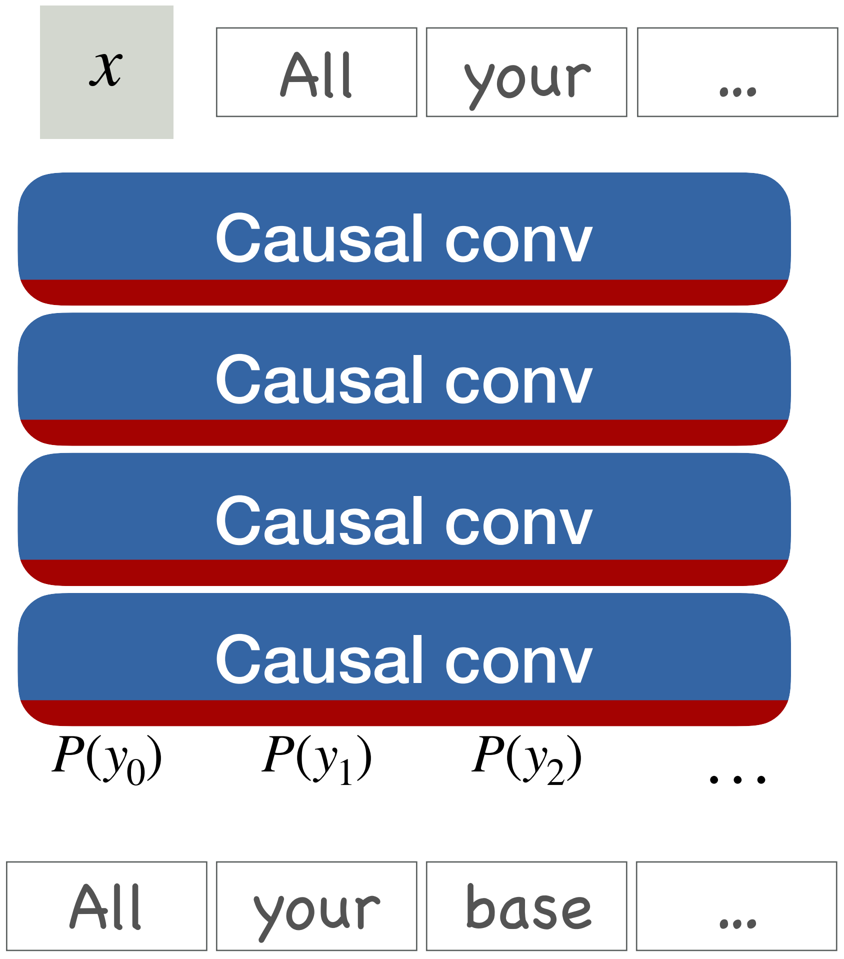
Causal convolution implementation

- Regular convolution
  - Shift output



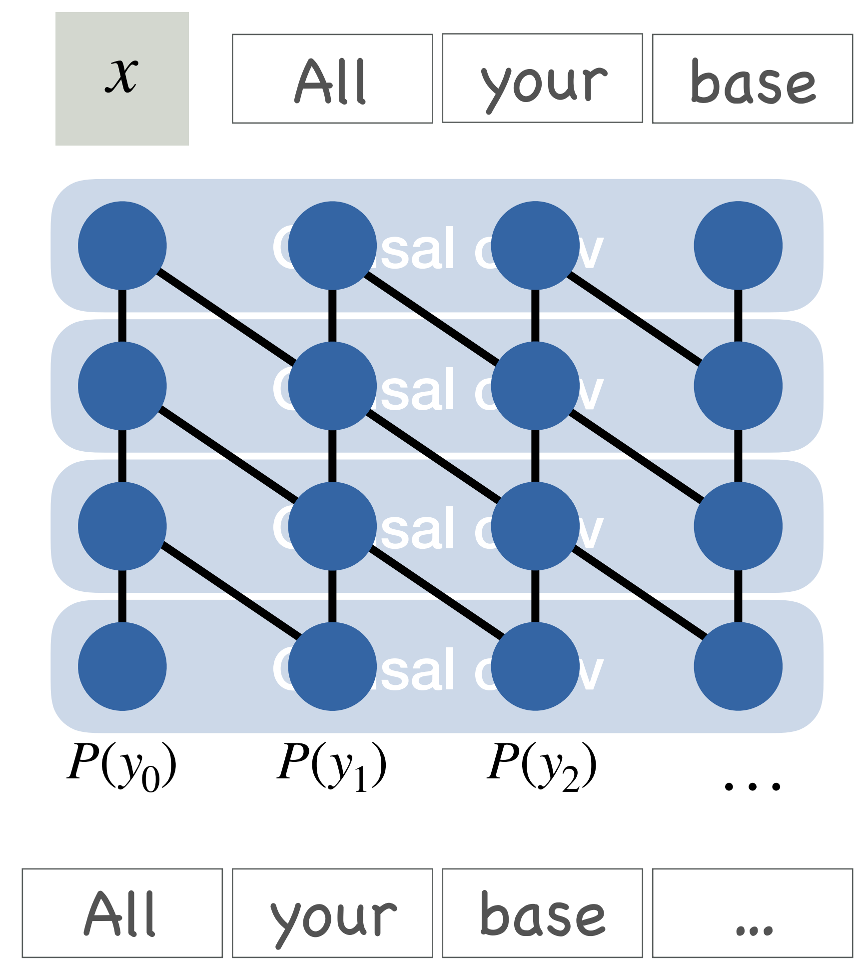
Training with temporal convolutions

- Labels
    - input and output/loss
- Very efficient
    - fully convolutional



Inference with temporal convolutions

- Step by step
  - Harder to implement efficiently



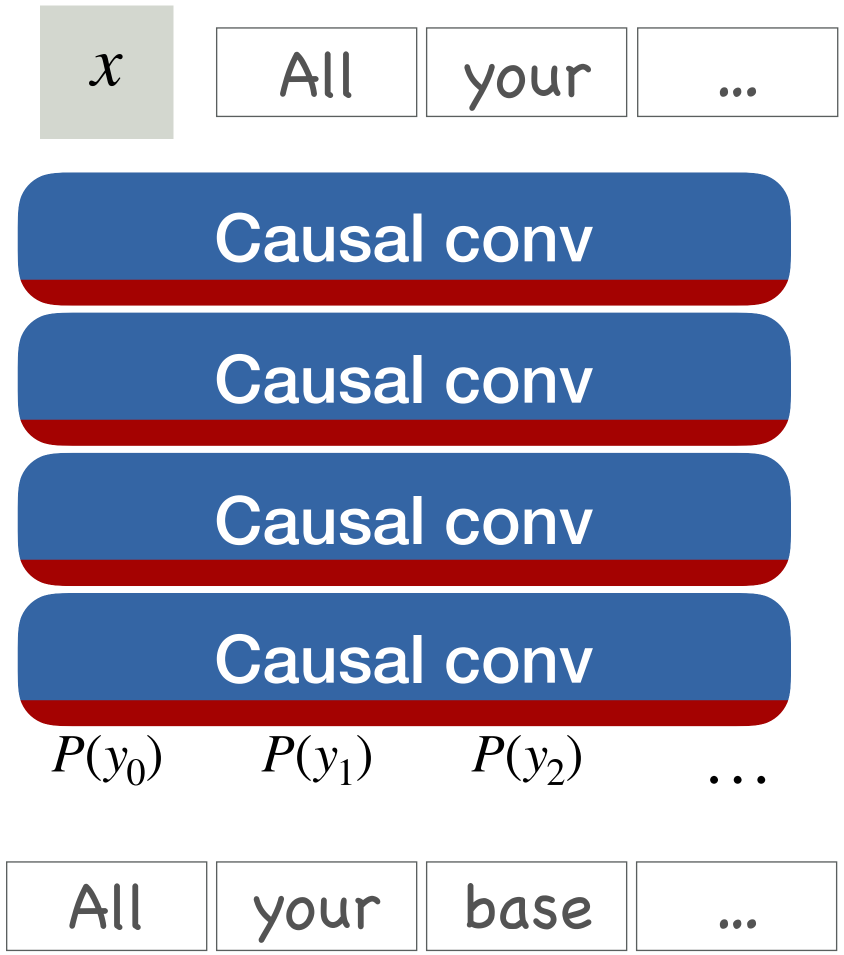
Sampling in sequence models

Sampling

- Example temporal convolutional network
  - Autoregressive

$$P(y_0, y_1, y_2, \ldots) = P(y_0 \mid x) \cdot P(y_1 \mid x, y_0) \cdot P(y_2 \mid x, y_0, y_1) \cdot \cdots$$

- Objective find
  - $\hat{y} = \arg\max_y P(y_0, y_1, y_2, \ldots)$



Greedy sampling

$$P(y_0, y_1, y_2, \ldots) = P(y_0 \mid x) \cdot$$
$$P(y_1 \mid x, y_0) \cdot$$
$$P(y_2 \mid x, y_0, y_1) \cdot$$
$$\ldots$$

- Pick sequentially

  $$\hat{y}_t = \arg\max_{y_t} P(y_t \mid x, \hat{y}_0, \hat{y}_1, \ldots)$$

  - Single sample

  - Not optimal

Sequential sampling

$$P(y_0, y_1, y_2, \ldots) = P(y_0 \mid x) \cdot$$
$$P(y_1 \mid x, y_0) \cdot$$
$$P(y_2 \mid x, y_0, y_1) \cdot$$
$$\ldots$$

- For n iterations
  - Sample sequentially $\hat{y}_t \sim P(y_t \mid x, \hat{y}_0, \hat{y}_1, \ldots)$
- Unbiased sampling
  - Not sample efficient

Beam search

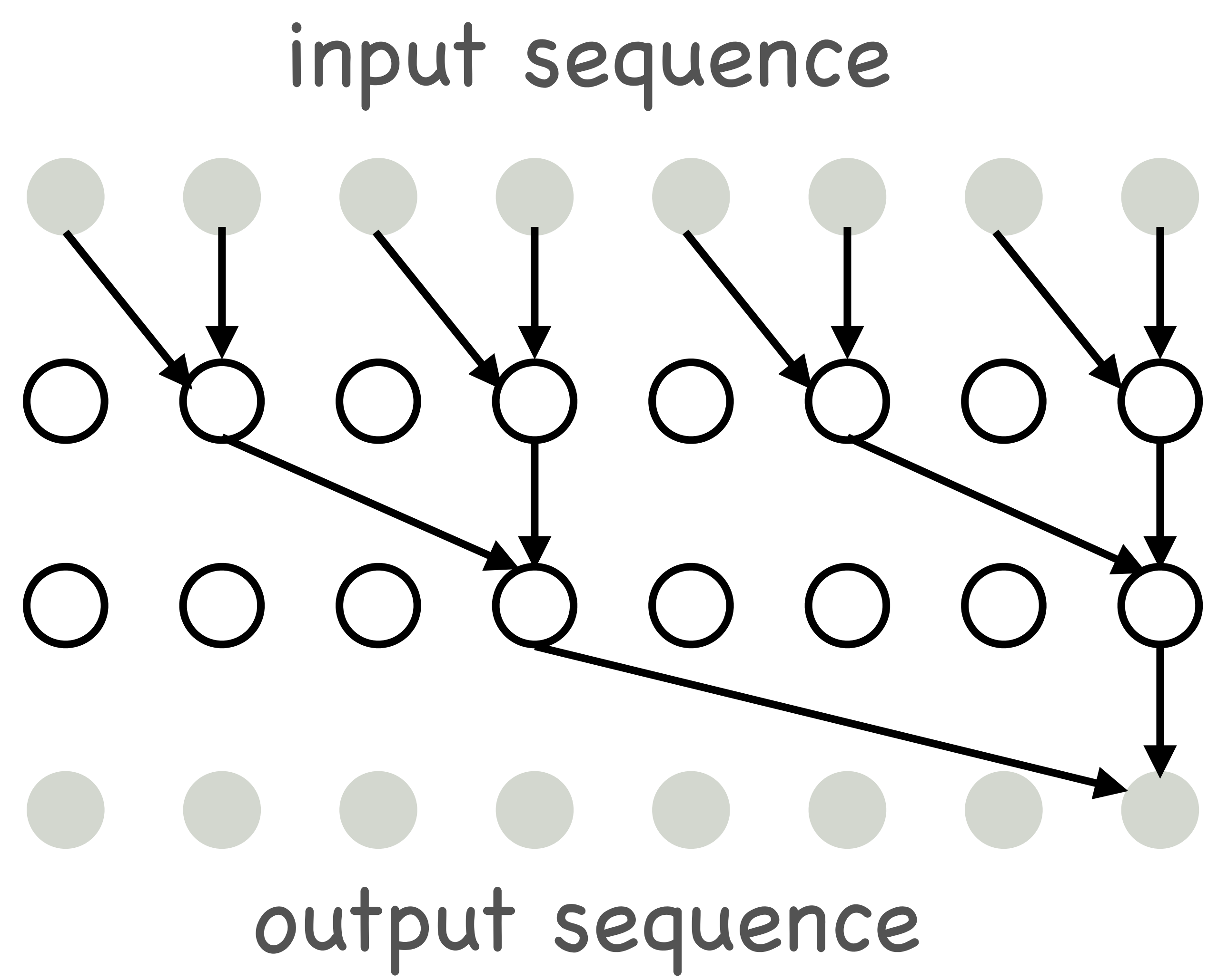
- Biased sampling
    - High likelihood samples
- Generally works best

- Keep top k samples S
    - Largest $P(y_0)$
- For t steps
    - For each $\hat{y}_0, \hat{y}_1, \ldots \in S$
        - Compute $P(x, \hat{y}_0, \hat{y}_1, \ldots, y_t)$
    - Keep top k samples S

Case study: WaveNet

WaveNet

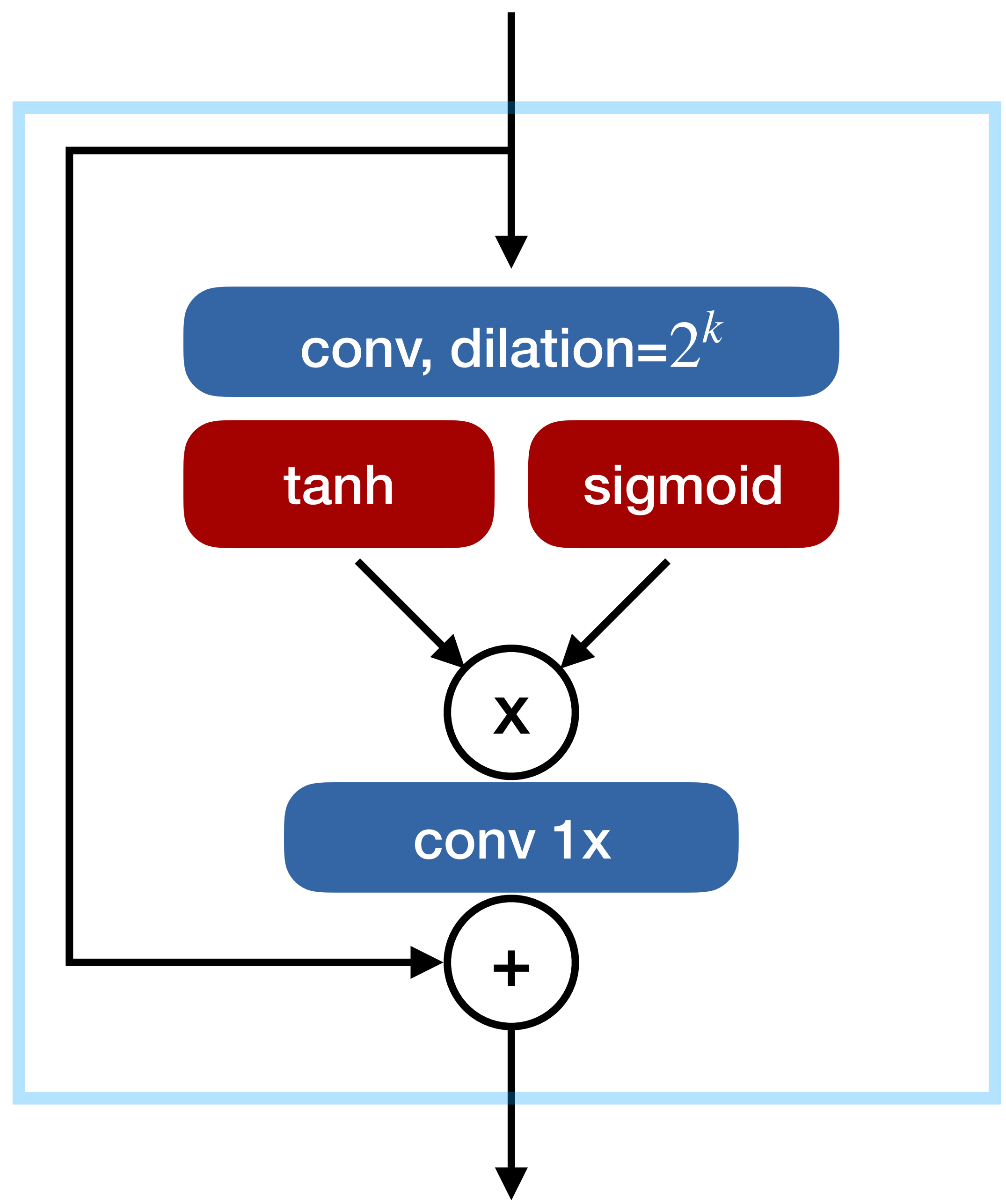
- Autoregressive model for sound synthesis and speech recognition
    - Generates raw waveform
        - Quantized in 8-bit
    - $P(y_t \,|\, x, y_0, \ldots, y_{t-1})$



WaveNet: A generative model for raw audio, Van Den Oord et al., *arXiv* 2016

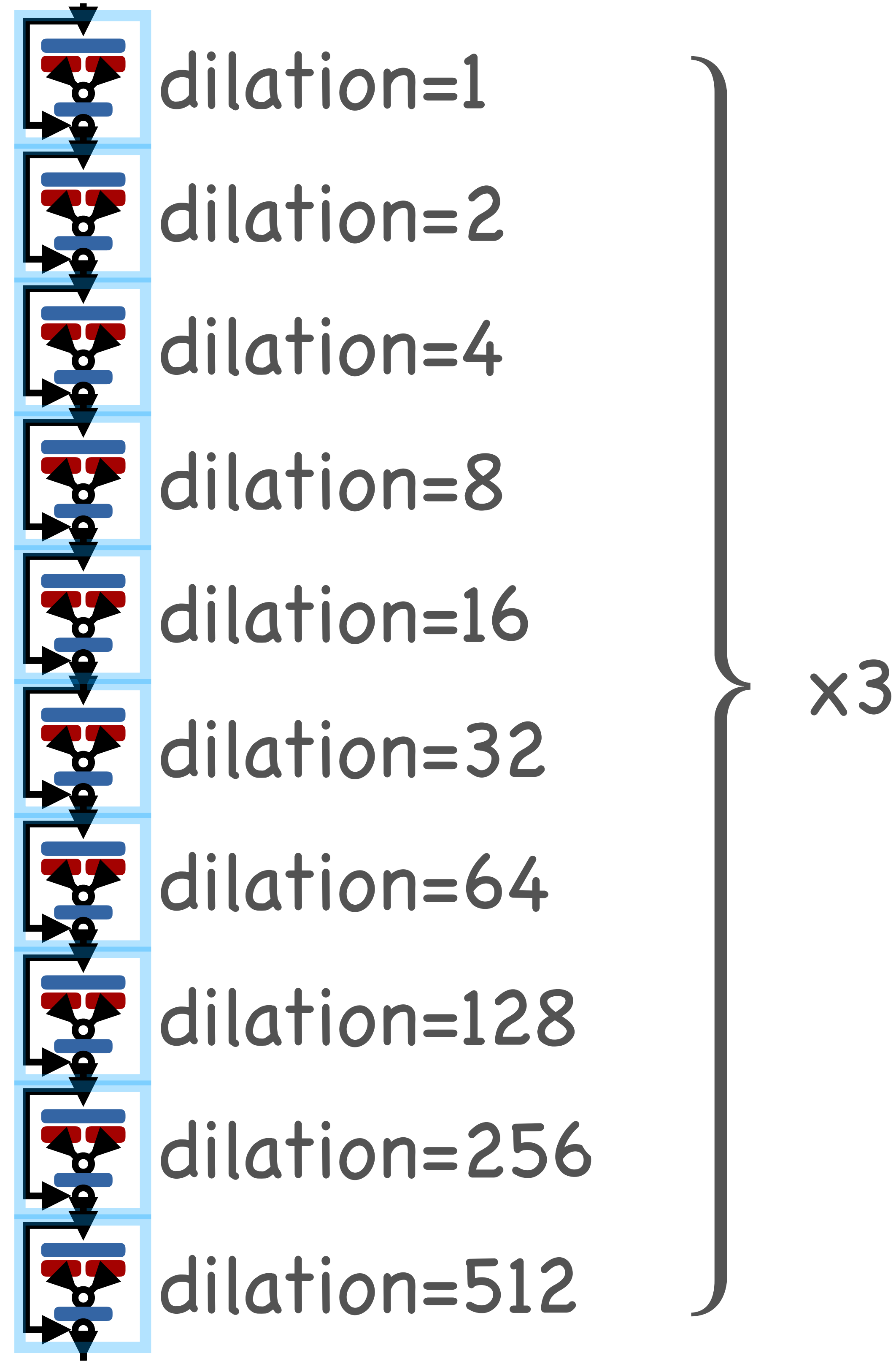
WaveNet – basic building block

- Dilated causal convolution
- Gated activation units



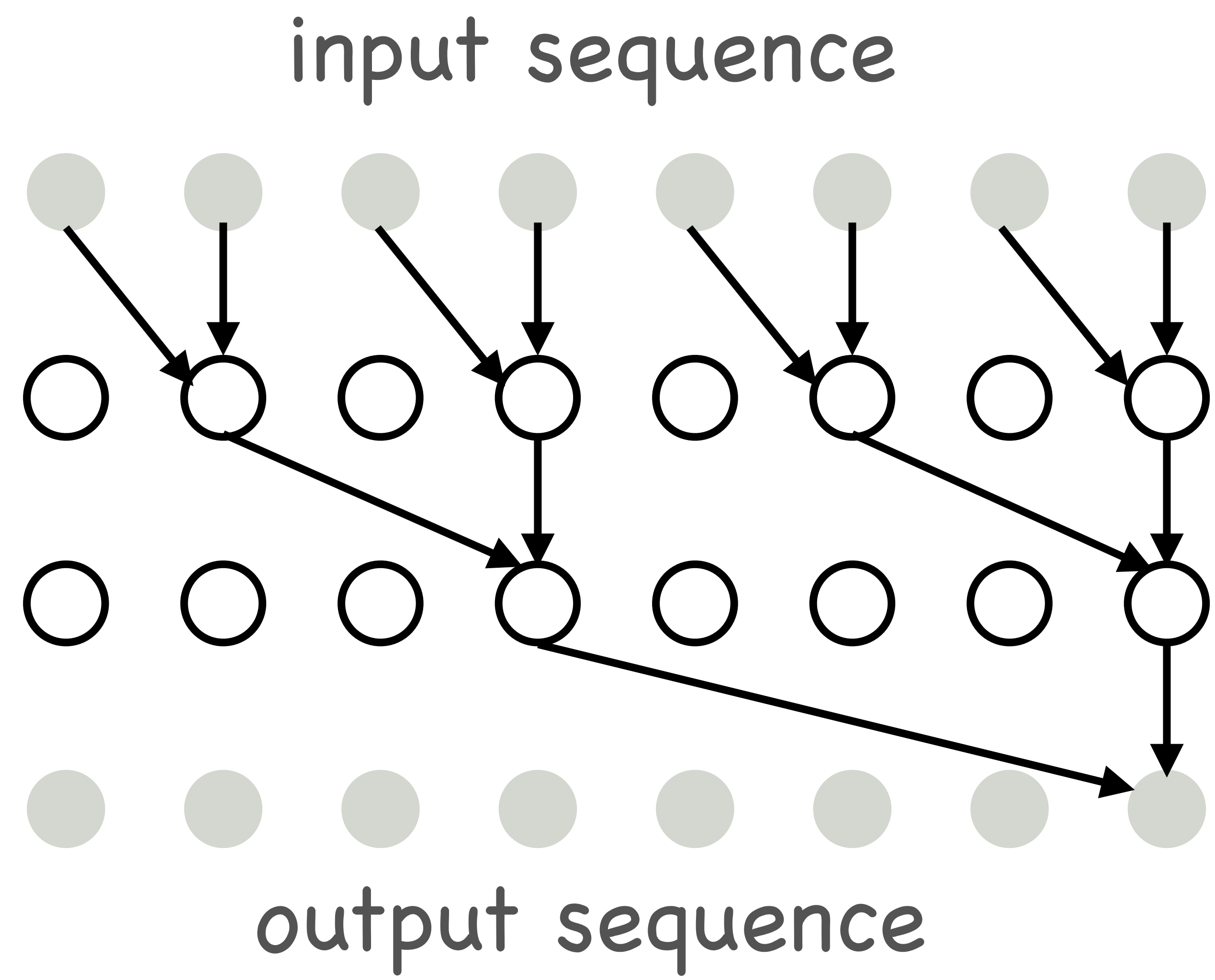
WaveNet

- Input
    - Causal generation y
- Output
    - $P(y_t \mid x, y_0, \ldots, y_{t-1})$



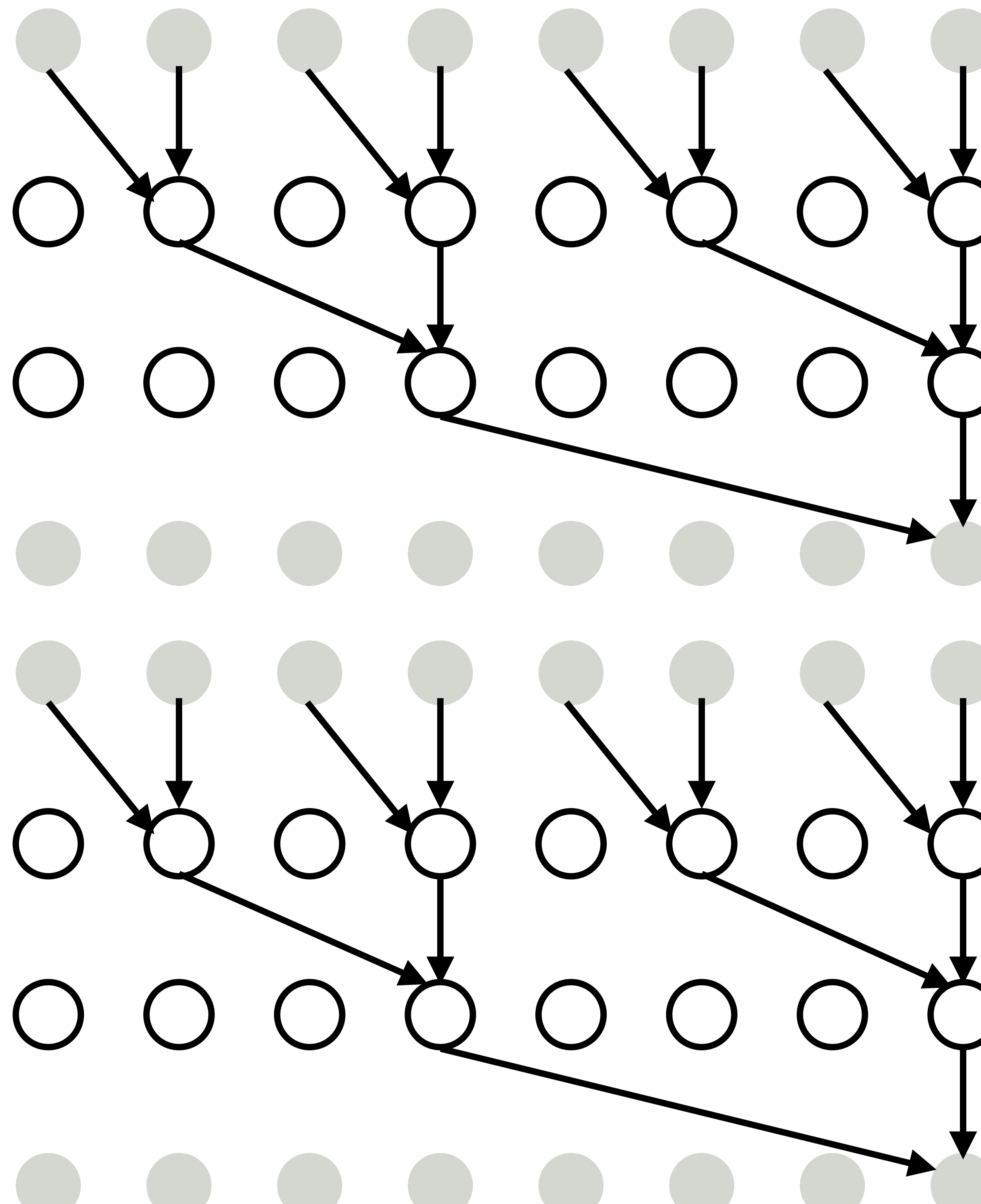
WaveNet

- State-of-the-art music and English speech generation
- Slow



Parallel WaveNet

- Inverse Autoregressive Flow (IAF)
  - Transform noise into sound
  - Single feed forward pass
  - No sampling
- Trained to mimic original WaveNet
- 500k samples / sec, 10x real time
  - Used by Google Assistant



Parallel WaveNet: Fast High-Fidelity Speech Synthesis, van den Oord et al., arXiv 2017

07

January 23, 2024

[2]:
```
%pylab inline
import torch
docs = open('docs.txt').read()
char_set = np.unique(list(docs))
device = torch.device('cuda') if torch.cuda.is_available() else torch.
↪device('cpu')
print('device = ', device)
```

```
%pylab is deprecated, use %matplotlib inline and import the required libraries.
Populating the interactive namespace from numpy and matplotlib
device =  cuda
```

[3]:
```
one_hot = torch.as_tensor(np.array(list(docs))[None,:] == np.array(char_set)[:
↪,None]).float()

def make_random_batch(batch_size, seq_len):
    B = []
    for i in range(batch_size):
        s = np.random.choice(one_hot.size(1)-seq_len)
        B.append(one_hot[:,s:s+seq_len])
    return torch.stack(B, dim=0)
```

[4]:
```
class TCN(torch.nn.Module):
    def __init__(self, layers=[32,64,128,256]):
        super().__init__()
        c = len(char_set)
        L = []
        total_dilation = 1
        for l in layers:
            L.append(torch.nn.ConstantPad1d((2*total_dilation,0), 0))
            L.append(torch.nn.Conv1d(c, l, 3, dilation=total_dilation))
            L.append(torch.nn.ReLU())
            total_dilation *= 2
            c = l
        self.network = torch.nn.Sequential(*L)
        self.classifier = torch.nn.Conv1d(c, len(char_set), 1)
```

```python
    def forward(self, x):
        return self.classifier(self.network(x))

tcn = TCN()
```

[5]:
```python
tcn(one_hot[None,:,:100]).shape
```

[5]: torch.Size([1, 107, 100])

[6]:
```python
%load_ext tensorboard
import tempfile
log_dir = tempfile.mkdtemp()
%tensorboard --logdir {log_dir} --reload_interval 1
```

<IPython.core.display.HTML object>

[7]:
```python
import torch.utils.tensorboard as tb
n_iterations = 10000
batch_size = 128
seq_len = 256

logger = tb.SummaryWriter(log_dir+'/tcn1', flush_secs=1)

# Create the network
tcn = TCN().to(device)

# Create the optimizer
optimizer = torch.optim.Adam(tcn.parameters())

# Create the loss
loss = torch.nn.CrossEntropyLoss()

one_hot = one_hot.to(device)

# Start training
for iterations in range(n_iterations):
    batch = make_random_batch(batch_size, seq_len+1)
    batch_data = batch[:,:,:-1]
    batch_label = batch[:,:,1:].argmax(dim=1)

    o = tcn(batch_data)
    loss_val = loss(o, batch_label)

    logger.add_scalar('train/loss', loss_val, global_step=iterations)

    optimizer.zero_grad()
    loss_val.backward()
```

```
    optimizer.step()
```

[8]:
```
# Inference
def sample(m, length=100):
    S = list("Model")
    for i in range(length):
        data = torch.as_tensor(np.array(S)[None,:] == np.array(char_set)[:
 ↪,None]).float()
        o = m(data[None])[0,:,-1]
        s = torch.distributions.Categorical(logits=o).sample()
        S.append(char_set[s])
    return "".join(S)

print( sample(tcn.cpu()) )
```

```
Modelul porise infiest a nece-words,
From Coear sut encome
To wimmy of dierted, or such him one
I ever a
```

[ ]:
```

```

Attention and transformers

Attention and transformers

- Alternative to convolutions
  - Flexible in time
  - Popular in natural language processing

Attention

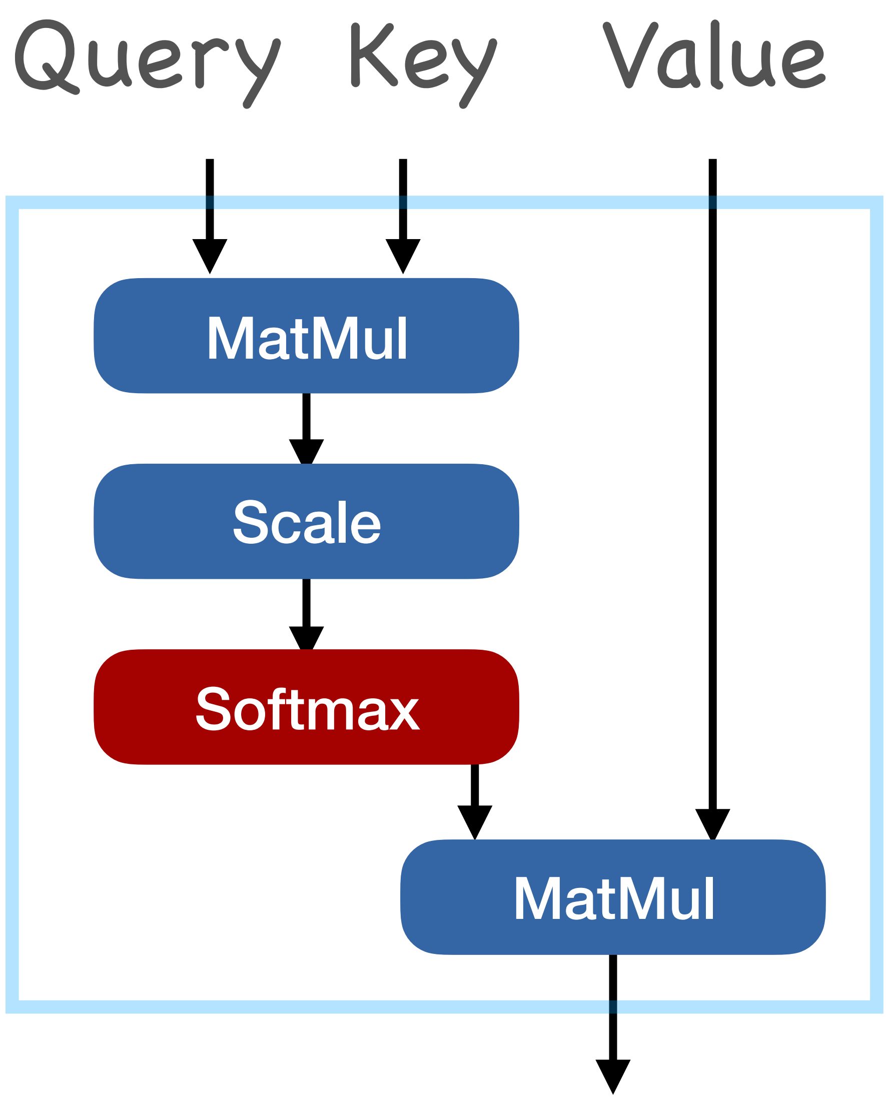
Attention Is All You Need, Vaswani et al., NIPS 2017

Attention

- Weighted average

$$\text{attention}\left(\mathbf{q}, \{\mathbf{k}_0, \mathbf{k}_1, \ldots\}, \{\mathbf{v}_0, \mathbf{v}_1, \ldots\}\right)$$

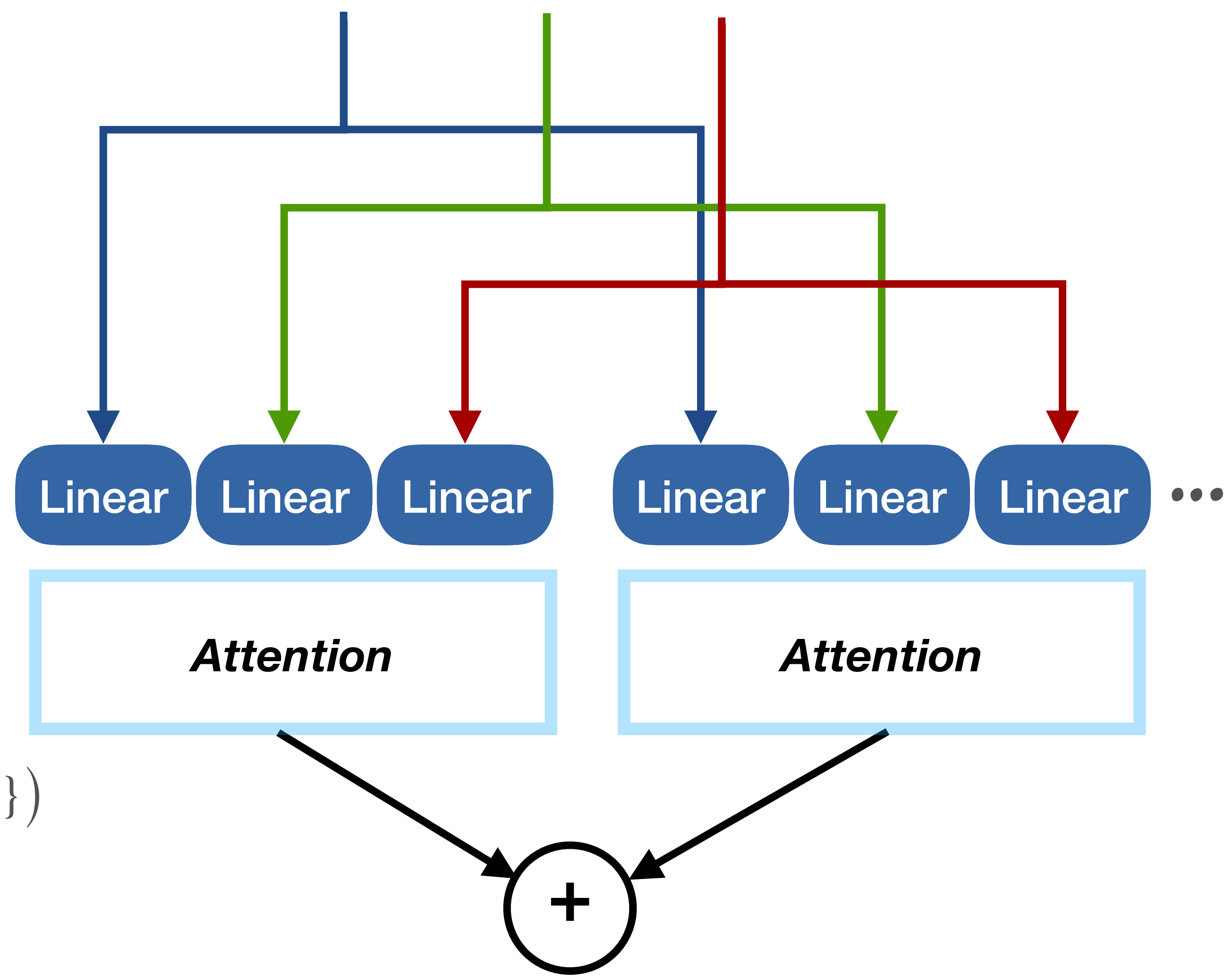
- $$= \frac{\sum_t \mathbf{v}_t \exp\left(\mathbf{k}_t^\top \mathbf{q}/\sqrt{d}\right)}{\sum_t \exp\left(\mathbf{k}_t^\top \mathbf{q}/\sqrt{d}\right)}$$



Multi-head attention

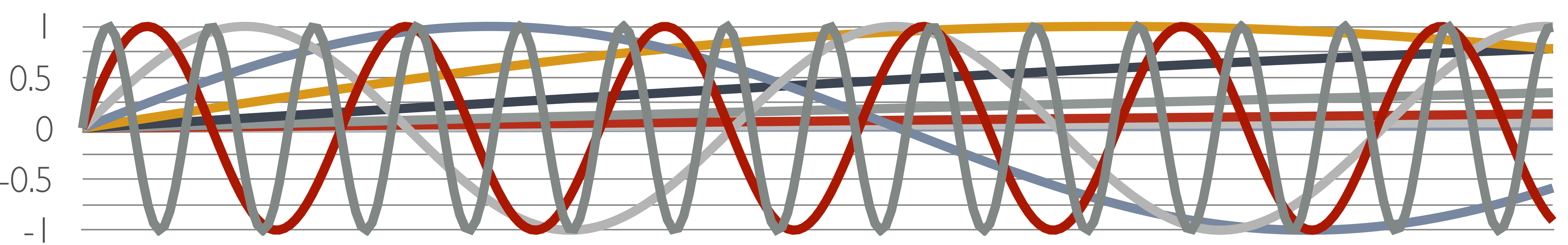
- Multiple attentions concatenated

- $$\text{multihead}\left(\mathbf{q}, \{\mathbf{k}_0, \mathbf{k}_1, \ldots\}, \{\mathbf{v}_0, \mathbf{v}_1, \ldots\}\right)$$ $$= \sum_i \text{attention}\left(\tilde{\mathbf{T}}_i\mathbf{q}, \{\mathbf{T}_i\mathbf{k}_0, \mathbf{T}_i\mathbf{k}_1, \ldots\}, \{\mathbf{W}_i\mathbf{v}_0, \mathbf{W}_i\mathbf{v}_1, \ldots\}\right)$$



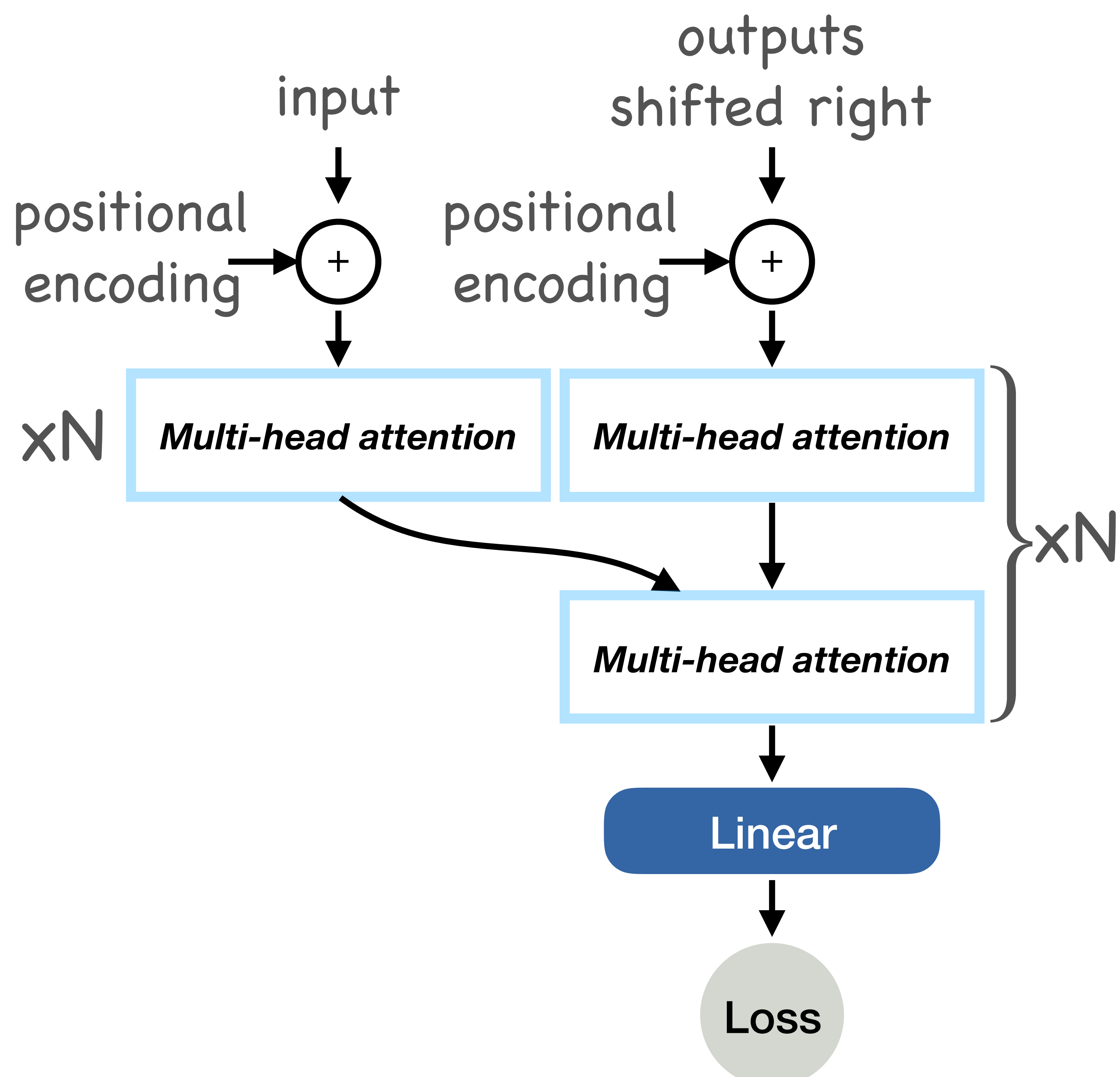
Positional encoding

- Attention is time-invariant
  - Add time back as a feature
    - sine and cosine of position



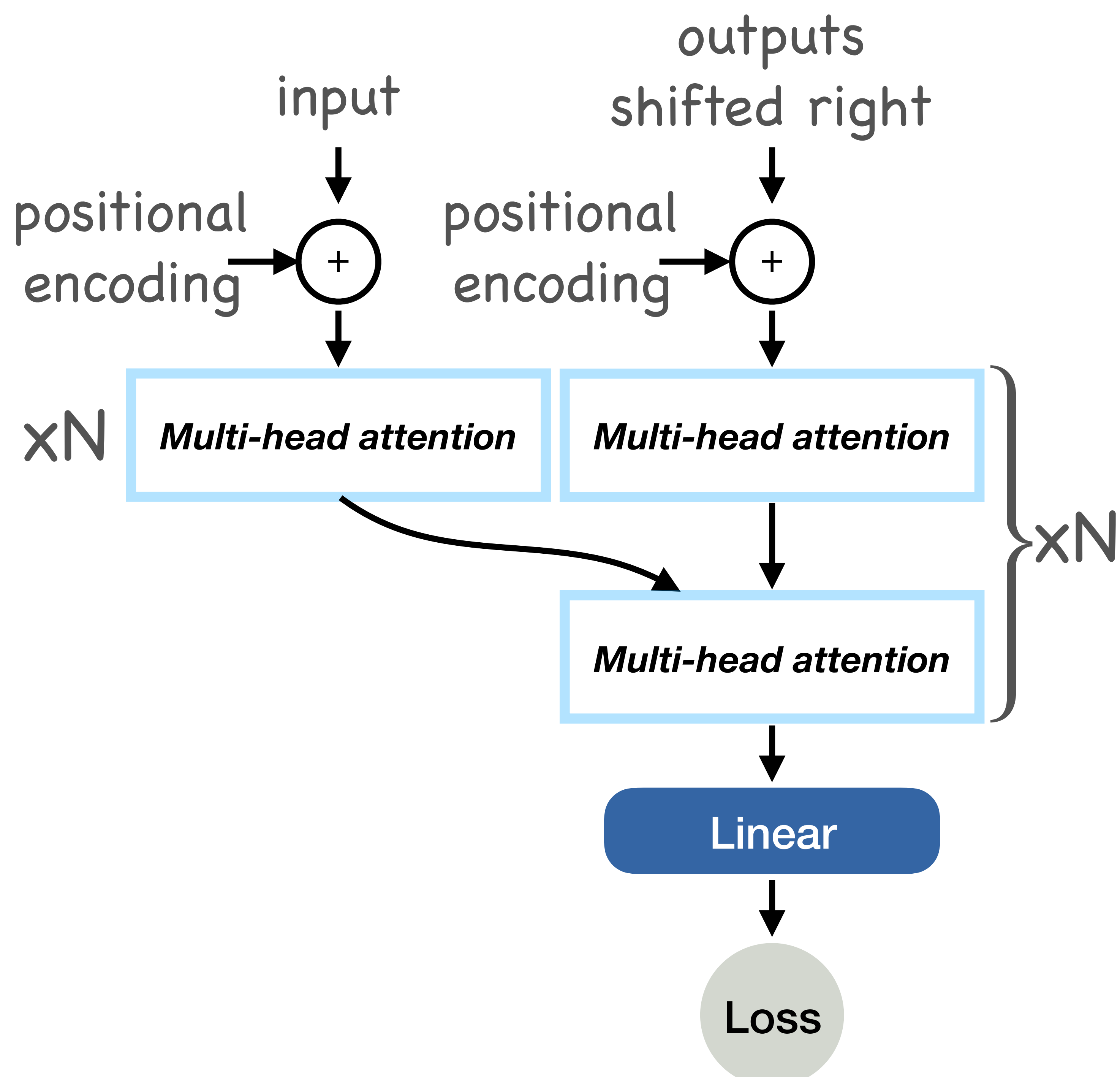
Transformer

- Feed forward
- Easy to train
  - Similar to Temporal CNN
- Causal attention
  - Auto-regressive



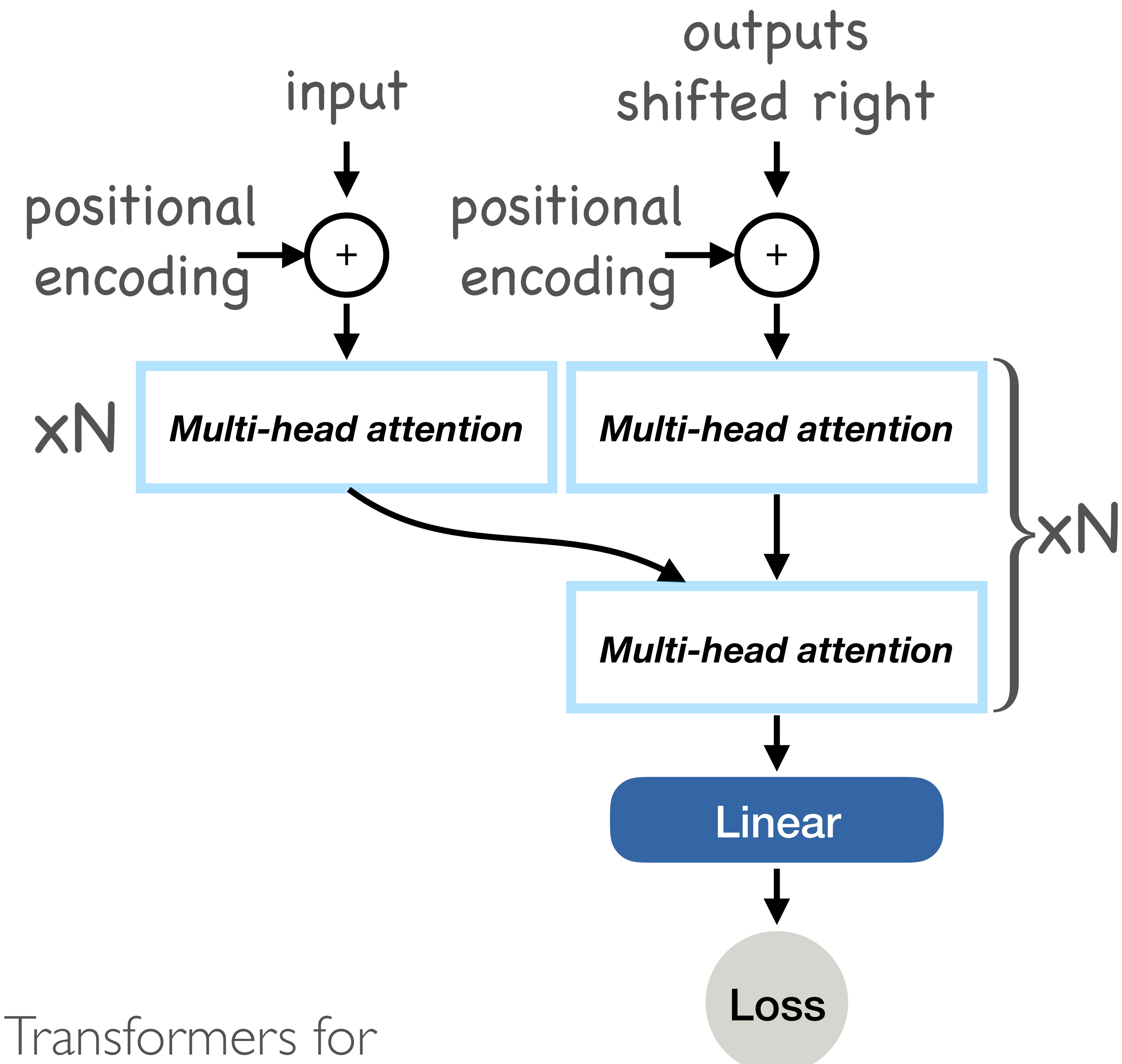
Transformer

- Faster to train
- Better performance
- State of the art performance



Bert

- Large transformer trained unsupervised
  - Predict masked out word
  - Predict next sentence
- Fine-tuned on NLP tasks
  - State-of-the-art for 6 month

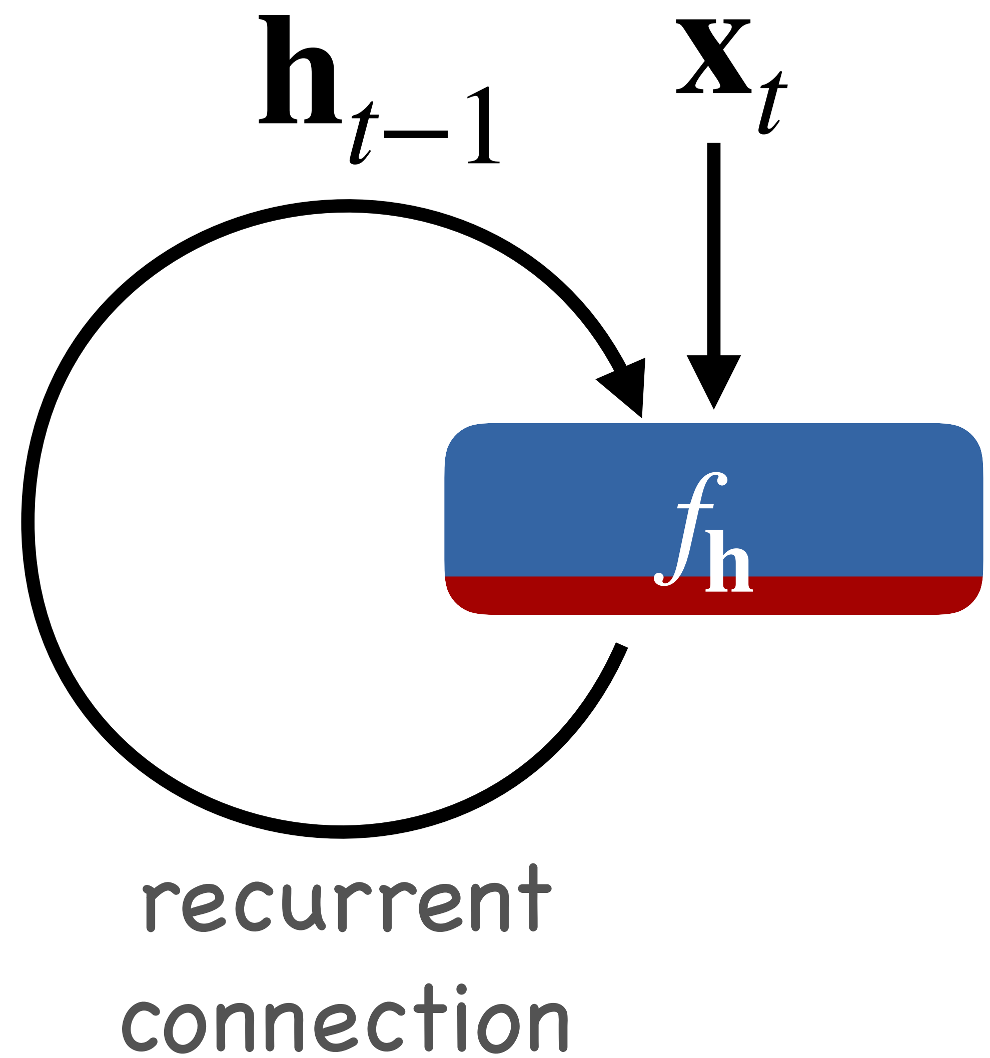


BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Devlin et al., arXiv 2018

Summary

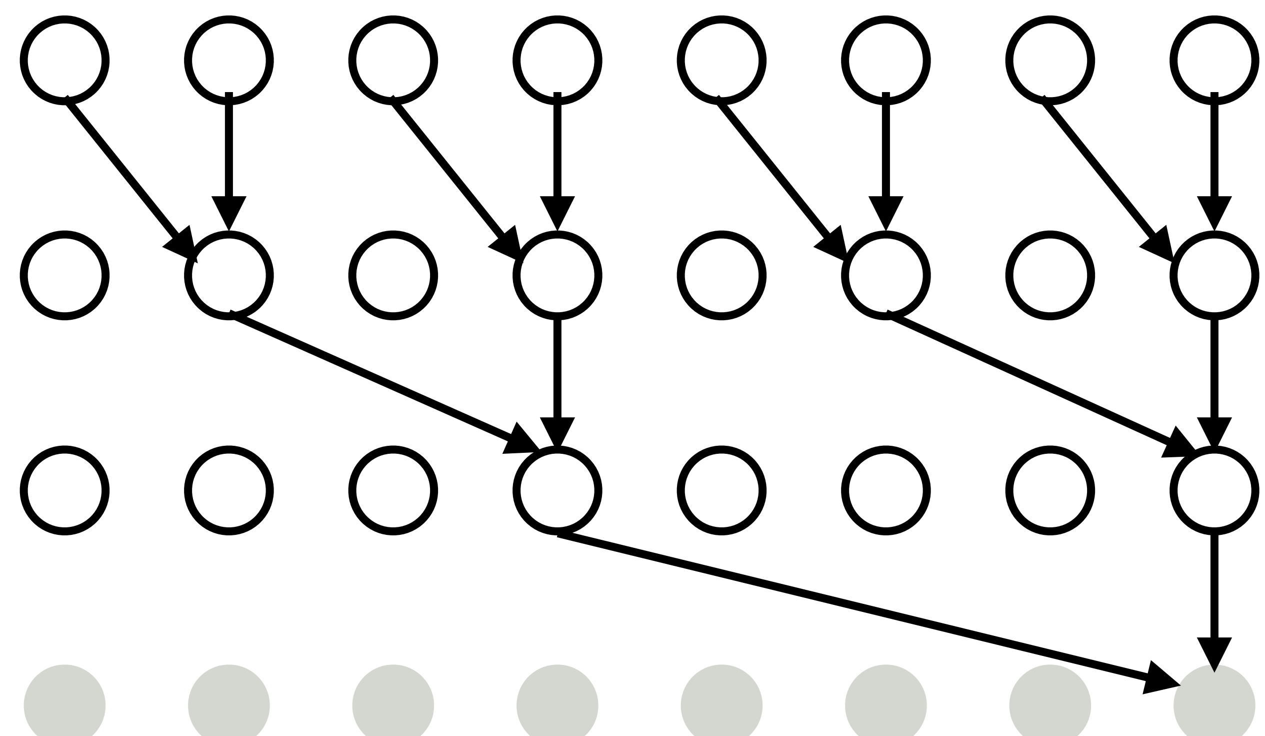
RNNs / LSTMs / GRUs

- Seem like a good idea
  - Too hard to train
  - No longer widely used



Temporal Convolutional Networks

- Fast and efficient training
  - Work well for structured data



Attention / Transformers

- Fast and efficient training
  - Better deals with irregular spacing
  - State-of-the-art in NLP