北京大学
PEKING UNIVERSITY

Recurrent Neural Network

Kun Yuan

Center for Machine Learning Research @ Peking University

Oct. 24, 2023

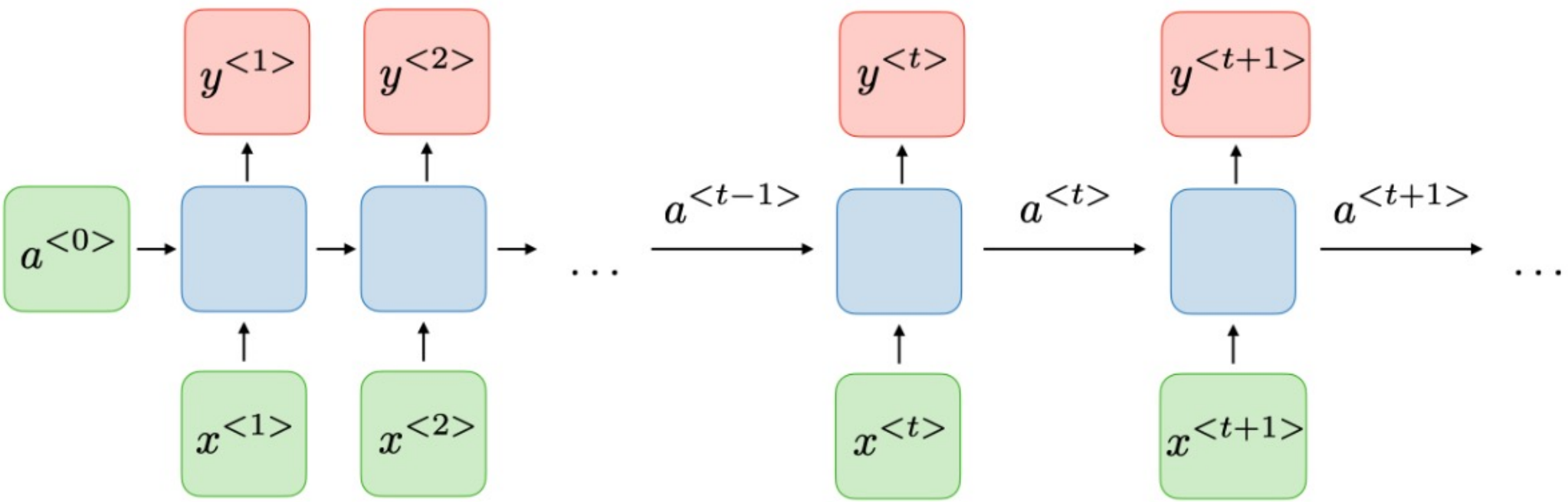
Some of the materials are from a great course [1] and a great book [2]

---

[1] Stanford CS 230 Deep Learning

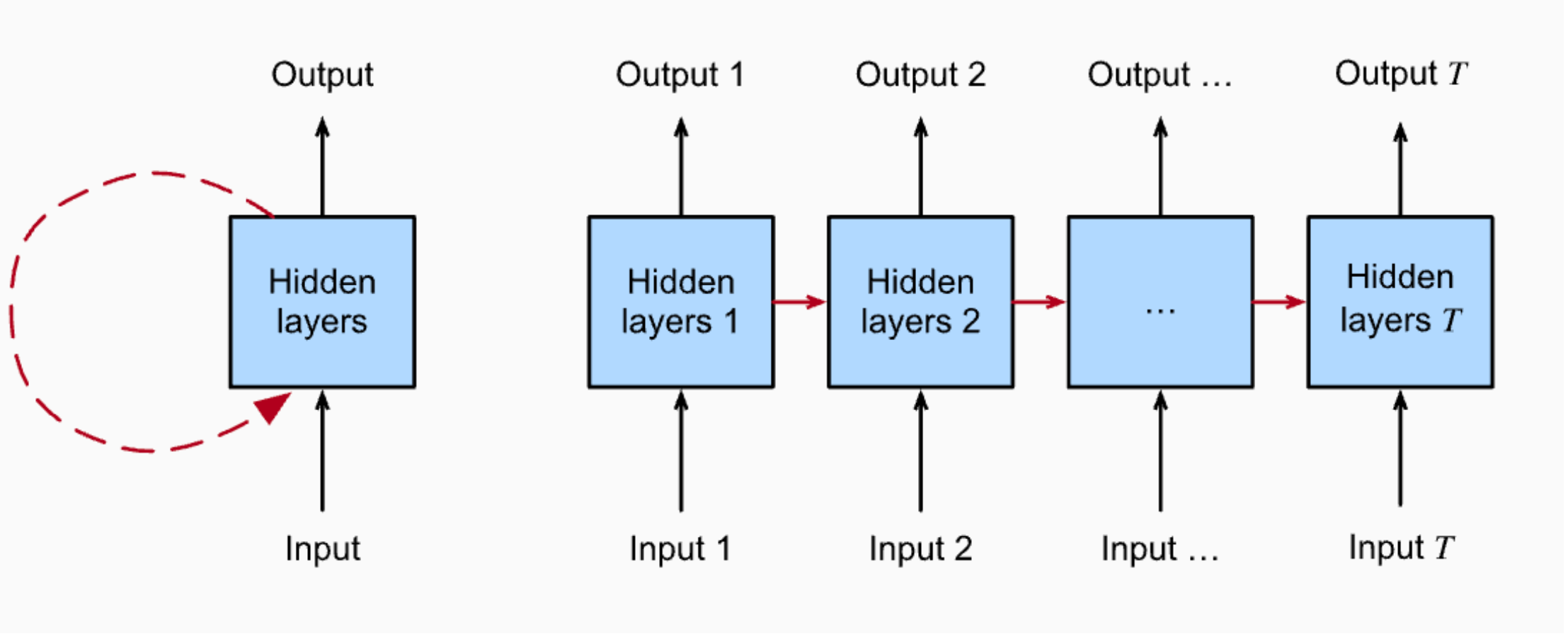
[2] Dive into recurrent neural network

Recurrent Neural Network (RNN)

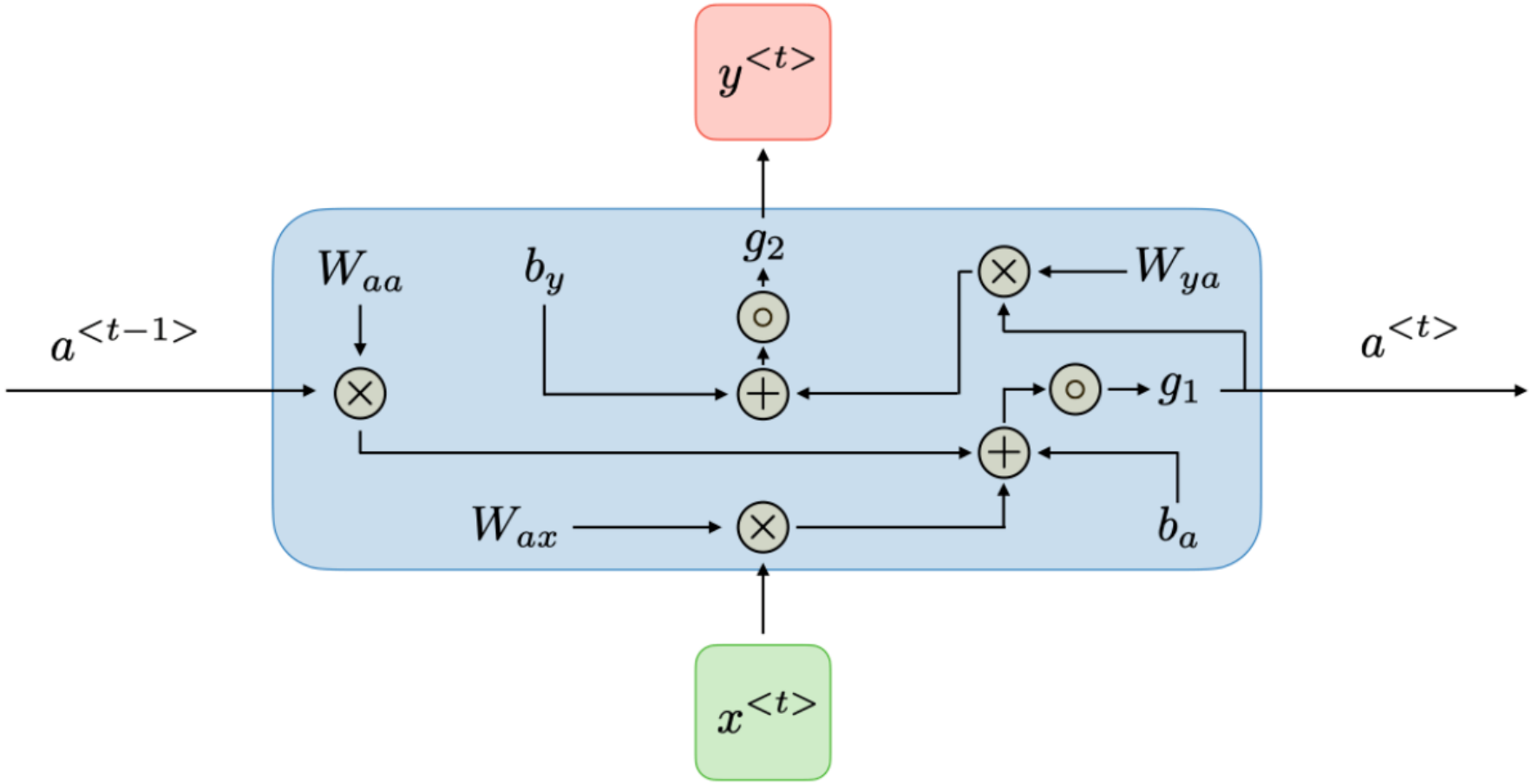


$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a) \quad \text{and} \quad y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$$

RNN


Output
Hidden layers
Input
Output 1
Output 2
Output …
Output T
Hidden layers 1
Hidden layers 2
…
Hidden layers T
Input 1
Input 2
Input …
Input T

RNN



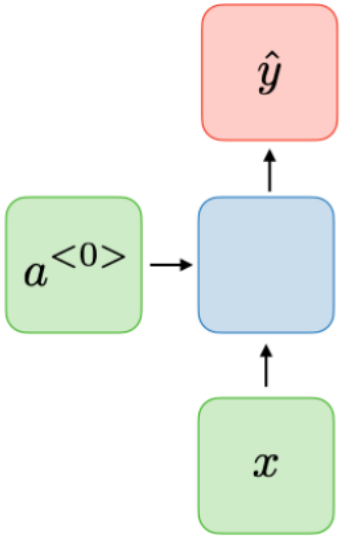
$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a) \quad \text{and} \quad y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$$

RNN

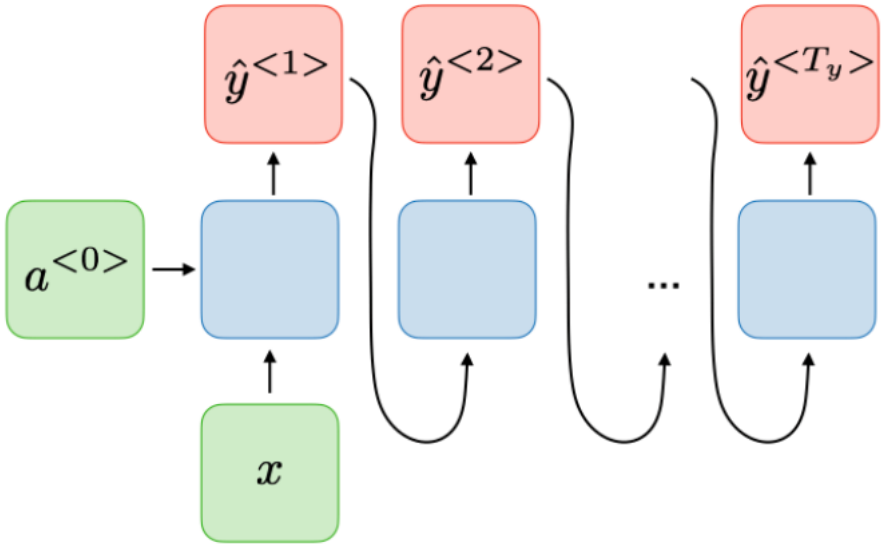
北京大学 PEKING UNIVERSITY

Advantages	Drawbacks
• Possibility of processing input of any length • Model size not increasing with size of input • Computation takes into account historical information • Weights are shared across time	• Computation being slow • Difficulty of accessing information from a long time ago • Cannot consider any future input for the current state

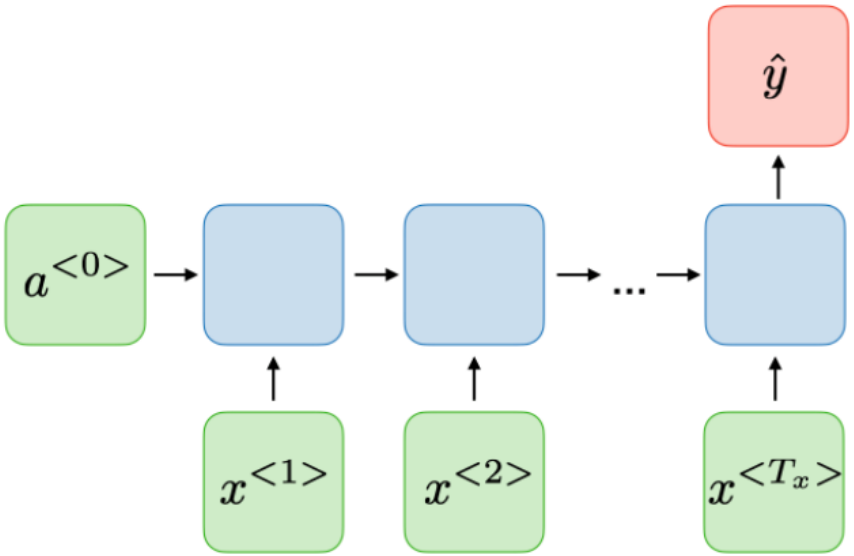
RNN

Type of RNN	Illustration	Example
One-to-one $T_x = T_y = 1$		Traditional neural network

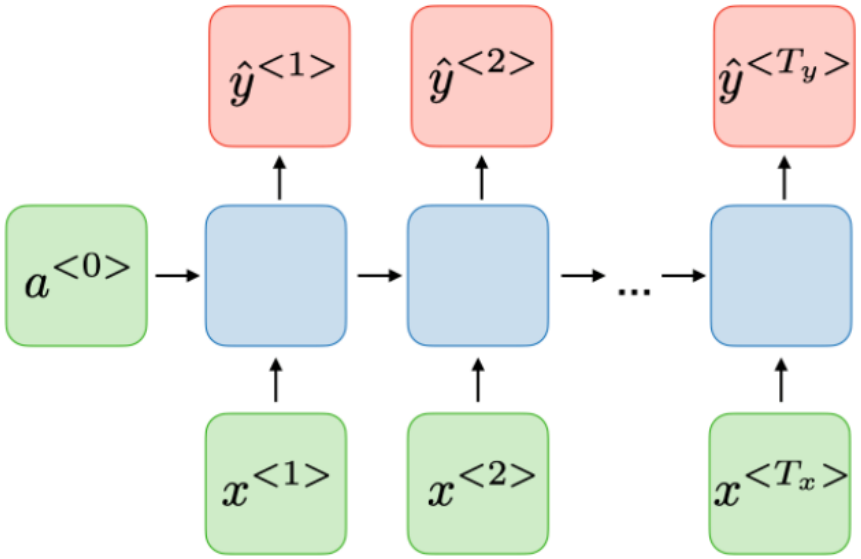
RNN

One-to-many $T_x = 1, T_y > 1$		Music generation

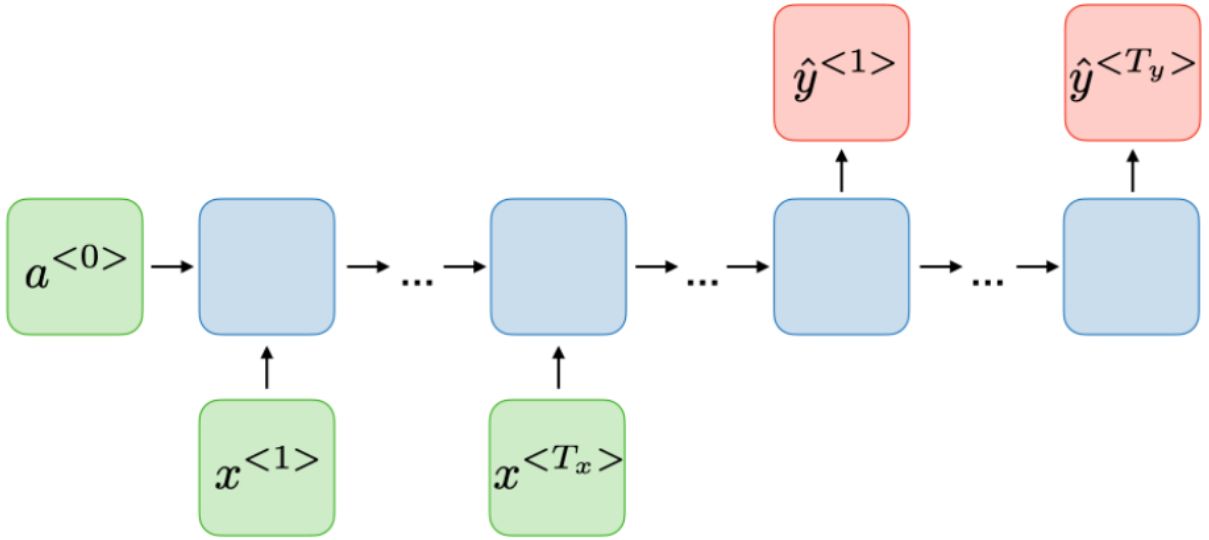
RNN

Many-to-one $T_x > 1, T_y = 1$		Sentiment classification

RNN

Many-to-many $T_x = T_y$		Name entity recognition

RNN

Many-to-many $T_x \neq T_y$		Machine translation

Deep RNN

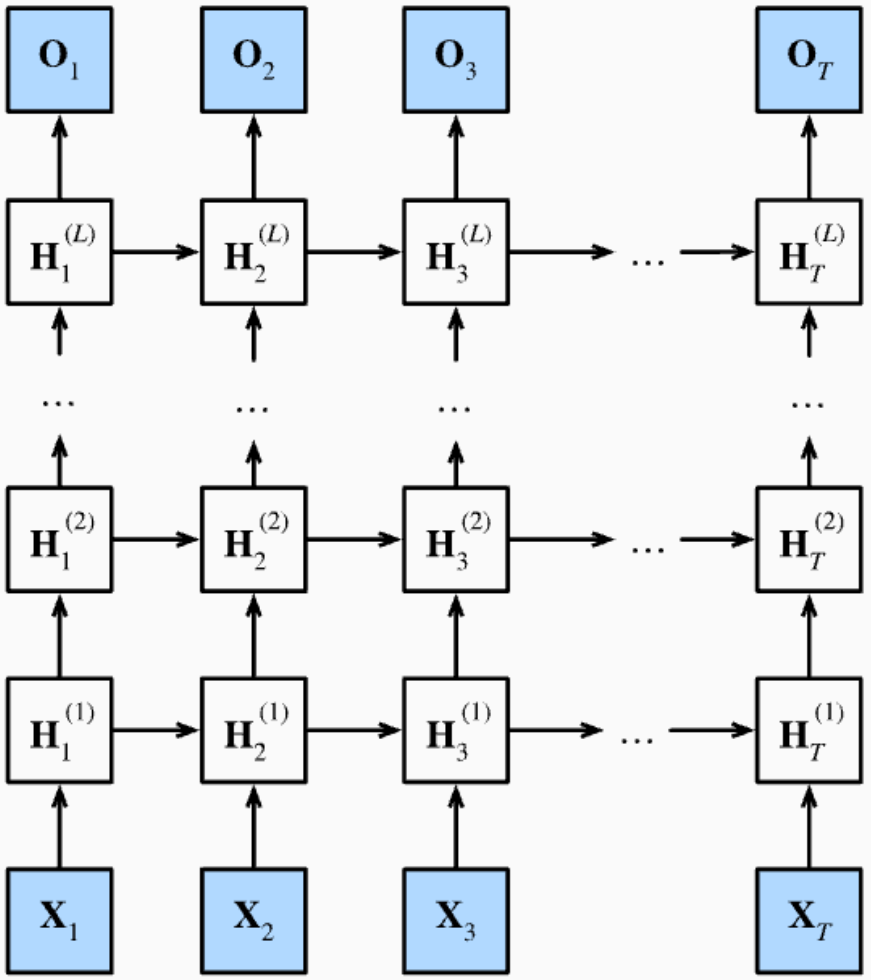


Fig. 10.3.1 Architecture of a deep RNN.

Bidirectional RNN

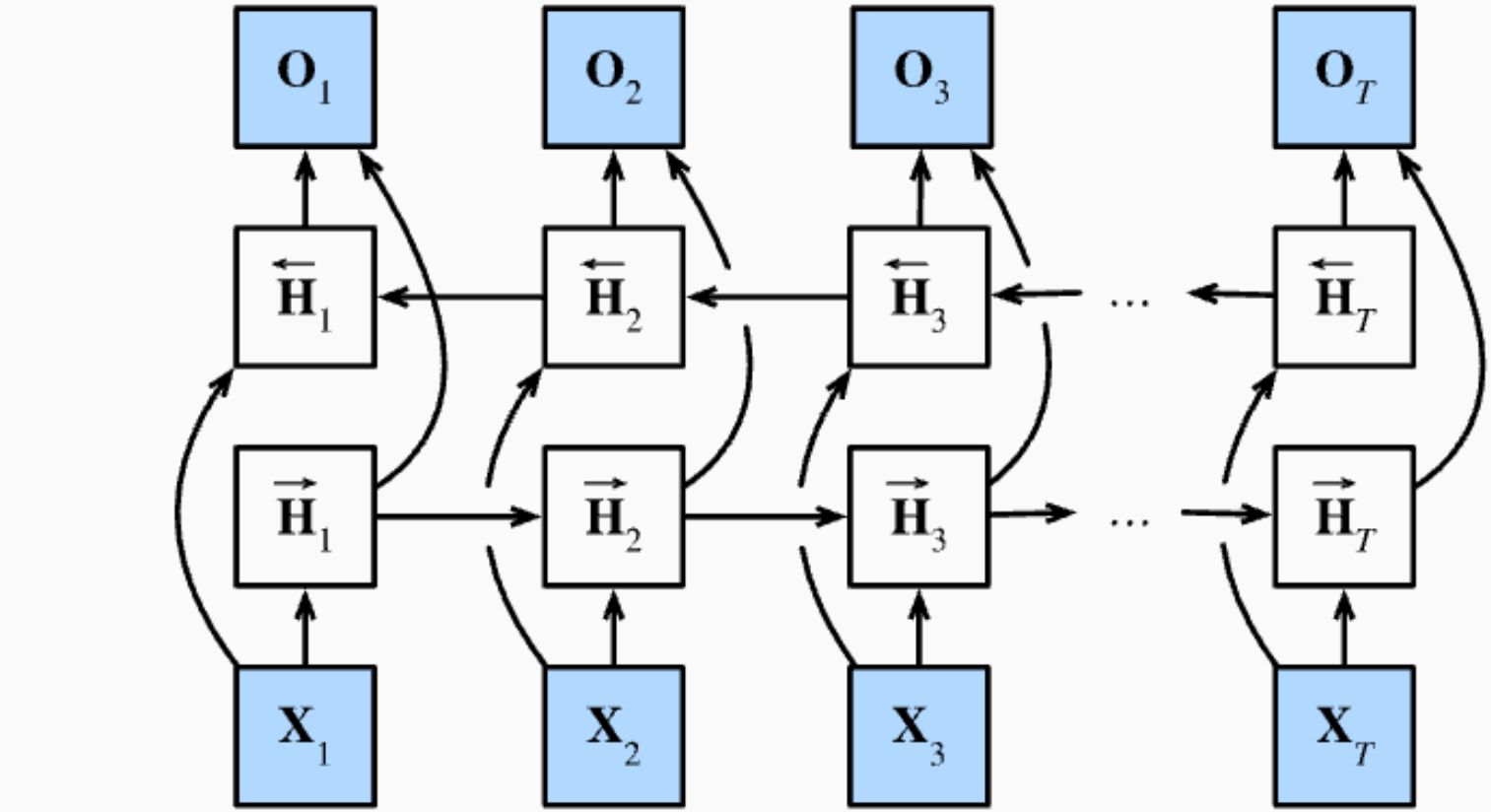


Fig. 10.4.1 Architecture of a bidirectional RNN.

Encoder-decoder

Fig. 10.6.1 The encoder–decoder architecture.

Machine translation

$$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1}).$$

$$\mathbf{c} = q(\mathbf{h}_1, \ldots, \mathbf{h}_T).$$

$$\mathbf{s}_{t'} = g(y_{t'-1}, \mathbf{c}, \mathbf{s}_{t'-1}).$$

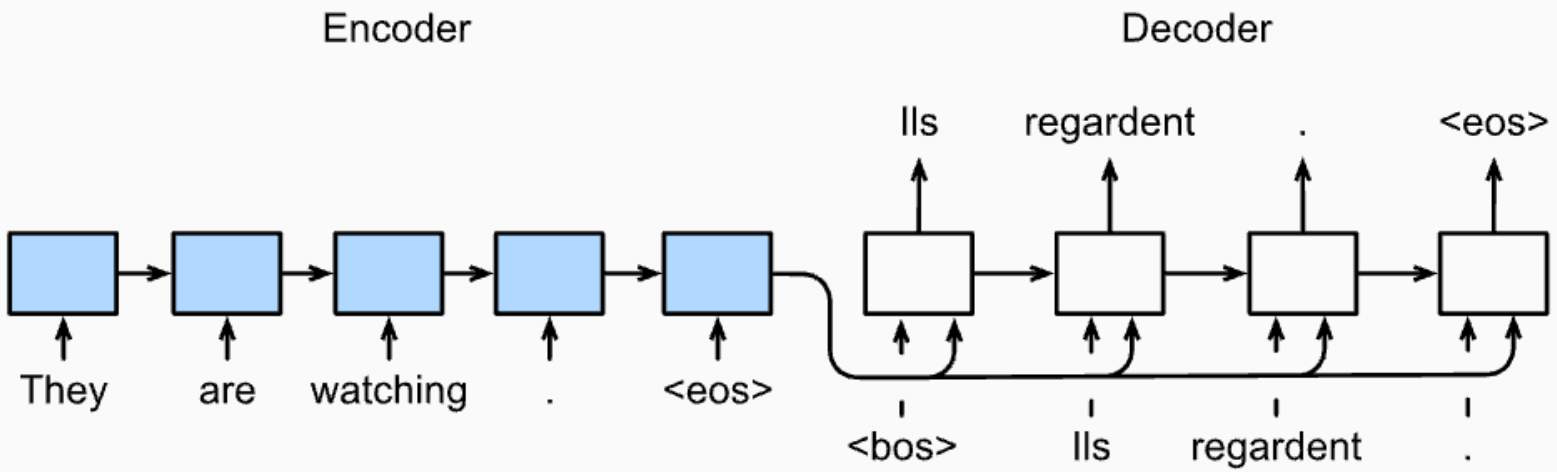


Fig. 10.7.1 Sequence-to-sequence learning with an RNN encoder and an RNN decoder.

Application: Learning to learn

北京大学 PEKING UNIVERSITY

A novel application of RNN is **learning to learn**

Marcin Andrychowicz et.al., “Learning to learn by gradient descent by gradient descent”, NeurIPS 2016

How to optimize neural network efficiently?

北京大学 PEKING UNIVERSITY

- SGD
- Momentum SGD / Nesterov accelerated SGD
- AdaGrad/RMSprop/AdaBound/Adam/AdamW

All these algorithms are designed by hands. They are not designed automatically.

Are these algorithms optimal? Probably not. Are there better algorithms? Probably yes.

How to design better algorithms? We do not know yet.

A brand new idea: Learn-to-Optimize

北京大学 PEKING UNIVERSITY

- Replace hand-designed update rules with a learned update rule

- Classical gradient-based algorithm

$$\theta_{t+1} = \theta_t - \alpha_t \nabla f(\theta_t) .$$

- Learned gradient-based algorithm

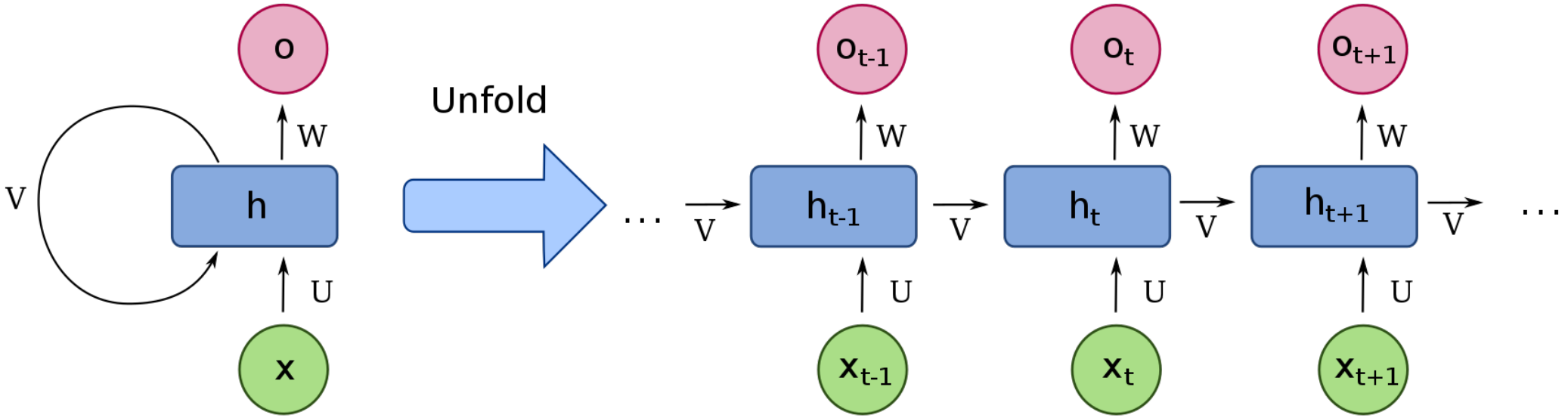
$$\theta_{t+1} = \theta_t + g_t(\nabla f(\theta_t), \phi) .$$

g_t is a mapping to be learned. It is parameterized by ϕ . When taking a gradient, it outputs a better update direction

Review RNN

- RNN is to handle message sequences

$$h_t = p(Ux_t + Vh_{t-1})$$
$$o_t = q(Wh_t)$$



Learned optimizer

- We can use RNN to learn a “good” update direction per iteration
- For each update, RNN takes in a gradient $\nabla f(\theta_t)$, outputs a learned update direction g_t
- Then the optimizee updates θ

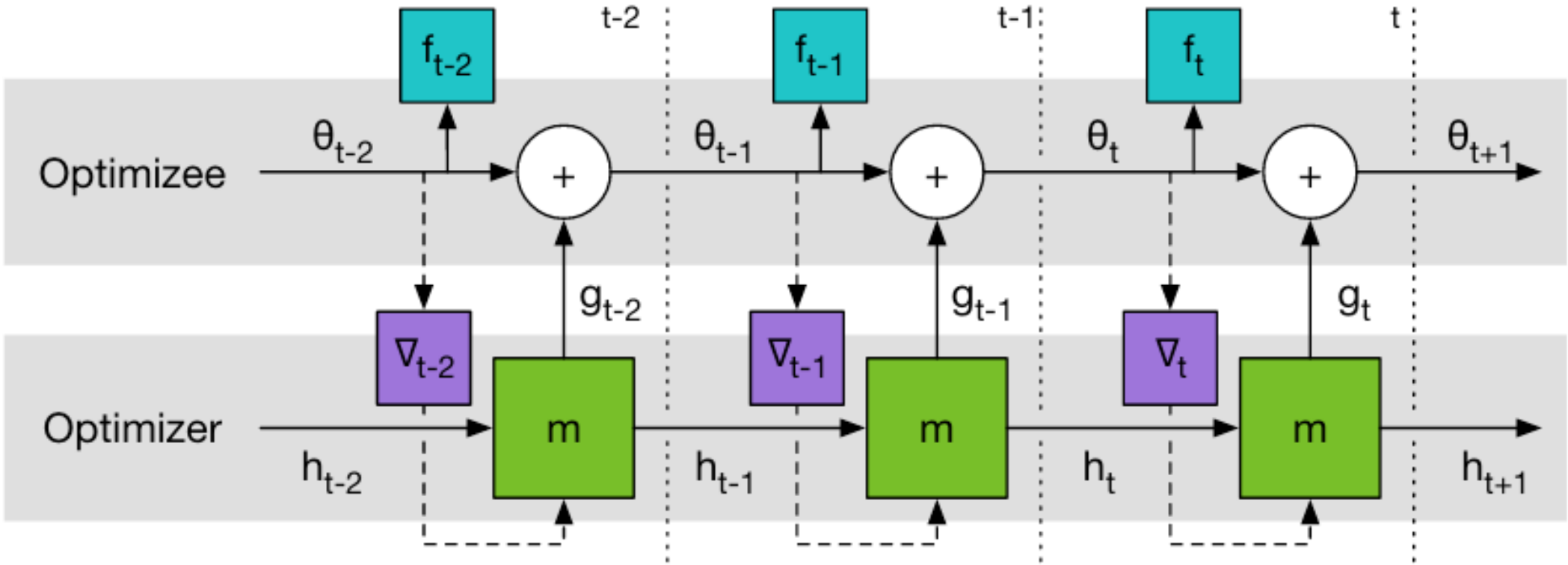
$$h_t = p(U\nabla f(\theta_t) + Vh_{t-1})$$
$$g_t = q(Wh_t)$$

RNN; U, V, W are to be learned

$$\theta_{t+1} = \theta_t + g_t$$

Optimizee

Learned optimizer



How to train optimizer?

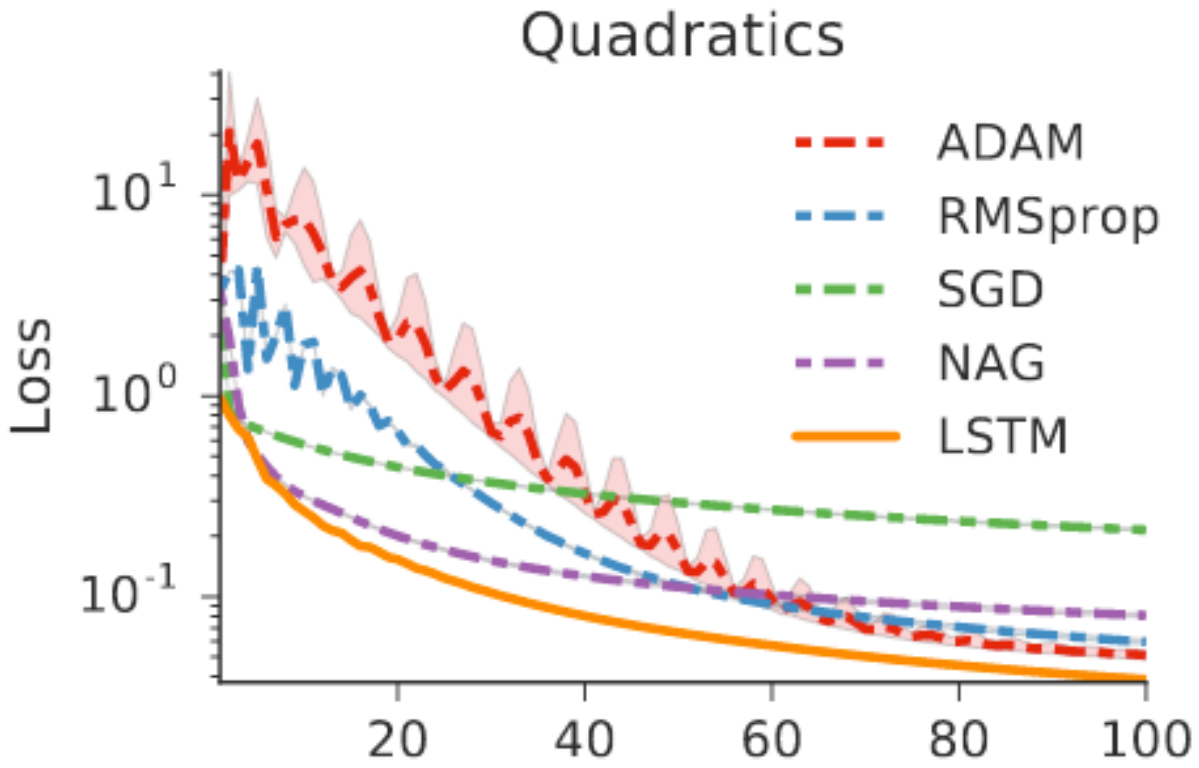
- Loss function: $\mathcal{L}(\phi) = \mathbb{E}_f \left[\sum_{t=1}^{T} w_t f(\theta_t) \right]$ for some predefined w_t

- Learn ϕ by stochastic gradient descent

  - Sample a function f. Compute forward propagation $L(\phi)$
  - Compute stochastic gradient as $\partial L(\phi)/\partial\phi$
  - Update ϕ by stochastic gradient descent
  - Repeat the above procedure until convergence

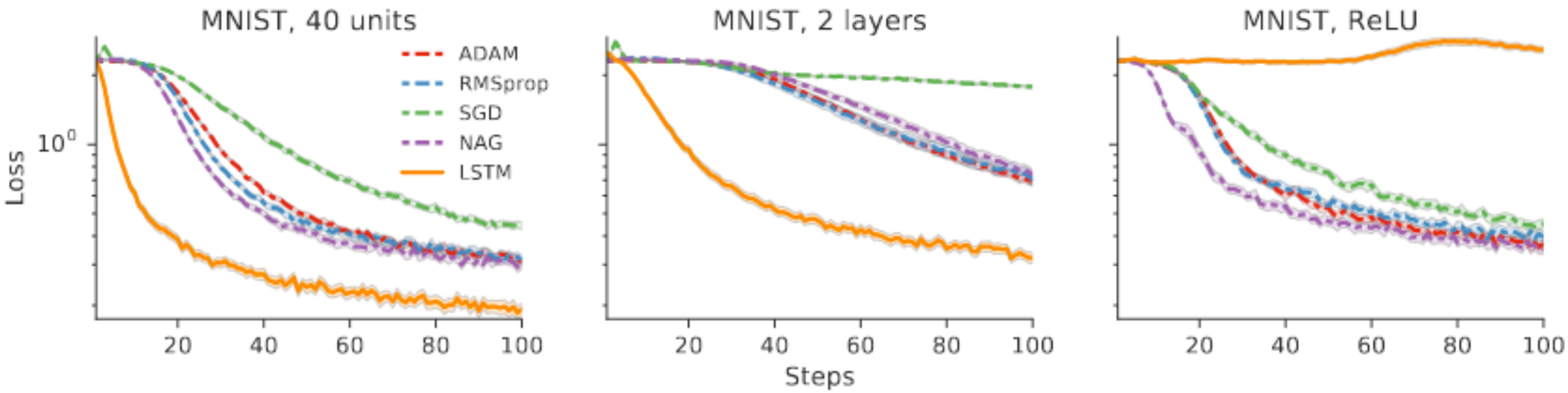
Experiments

- Quadratic functions: $f(\theta) = \|W\theta - y\|_2^2$



Experiments

- MNIST: train on a neural network with 20 hidden units



Experiments

- Neural arts: train with only 1 style with low-resolution; test with different style images and high-resolution

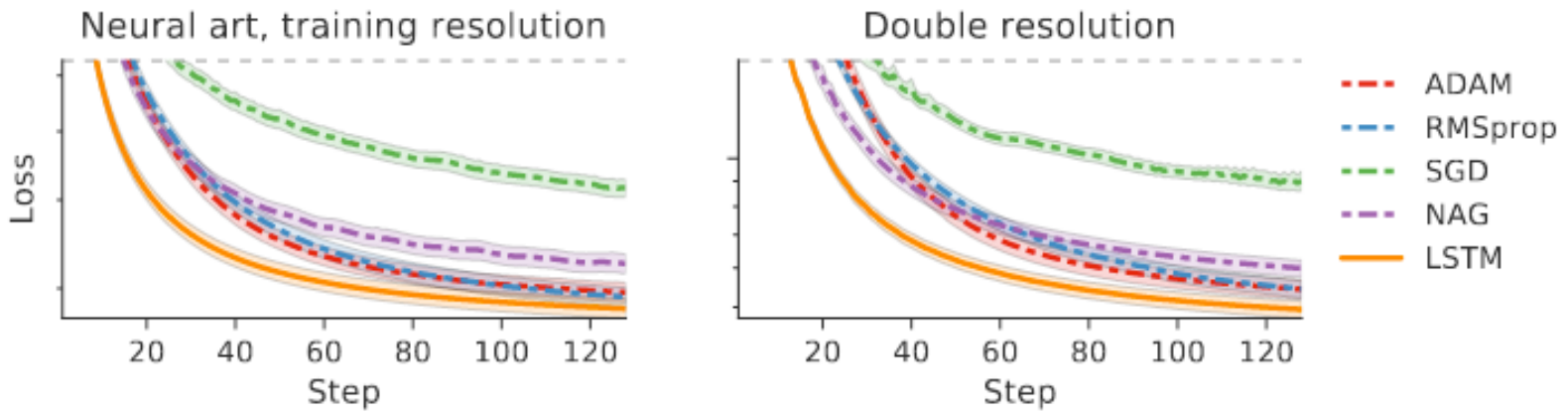


Figure 8: Optimization curves for Neural Art. Content images come from the test set, which was not used during the LSTM optimizer training. Note: the y-axis is in log scale and we zoom in on the interesting portion of this plot. **Left:** Applying the training style at the training resolution. **Right:** Applying the test style at double the training resolution.

Experiments

- Neural arts: train with only 1 style with low-resolution; test with different style images and high-resolution

Figure 9: Examples of images styled using the LSTM optimizer. Each triple consists of the content image (left), style (right) and image generated by the LSTM optimizer (center). **Left:** The result of applying the training style at the training resolution to a test image. **Right:** The result of applying a new style to a test image at double the resolution on which the optimizer was trained.