

# IE 534/CS 598 Deep Learning

University of Illinois at Urbana-Champaign

Fall 2018

Lecture 8

- Recurrent neural networks are widely used to model sequential or temporal dependence.
- Prominent examples include speech and text recognition.
- Sequence of data  $(X_1, Y_1), \dots, (X_t, Y_t), \dots$
- The goal is to estimate a model  $F_t(X_{s \leq t}; \theta)$  to predict  $Y_t$  for all times  $t$ .
- The model is allowed to depend upon the history of the data sequence  $X_t$ .

At a high level, a recurrent neural network is of the form

$$\begin{aligned}\hat{Y}_t &= f_Y(X_t, S_{t-1}; \theta), \\ S_t &= f_S(X_t, S_{t-1}; \theta),\end{aligned}\tag{1}$$

where  $X_t \in \mathbb{R}^d$  is the input data at time  $t$ ,  $\hat{Y}_t \in \mathbb{R}^K$  is the prediction for  $Y_t$ ,  $S_t \in \mathbb{R}^{d_S}$  is the “internal state” of the model,  $f_Y : \mathbb{R}^d \times \mathbb{R}^{d_S} \times \Theta \rightarrow \mathbb{R}^K$ , and  $f_S : \mathbb{R}^d \times \mathbb{R}^{d_S} \times \Theta \rightarrow \mathbb{R}^{d_S}$ .

$S_{t-1}$  is a **nonlinear representation** of the previous data  $X_{t-1}, X_{t-2}, \dots, X_1$ .

The objective function for estimating  $\theta$  is

$$\mathcal{L}(\theta) = \sum_{t=1}^T \rho(Y_t, \hat{Y}_t),\tag{2}$$

## Advantages of recurrent networks:

- A naive implementation  $F_t(X_1, \dots, X_t; \theta)$  is impractical since the input size changes at each time  $t$ .
- For long sequences, it is impractical to include such a long input vector due to computational constraints.
- The size of the model will drastically increase as  $t \rightarrow \infty$ , which could cause the model to overfit during training.

$$\begin{aligned}\hat{Y}_t &= f_Y(X_t, S_{t-1}; \theta), \\ S_t &= f_S(X_t, S_{t-1}; \theta),\end{aligned}\tag{3}$$

- $f_S(\cdot; \theta)$  and  $f_Y(\cdot; \theta)$  are neural networks.
- The parameters  $\theta$  are **shared** across times  $t = 1, 2, \dots, T$ .
- $S_1, \dots, S_T$  depend upon  $\theta$  in a complicated way.
- The backpropagation rule must be carefully derived.

Sharing parameters over time:

- Similar to convolution networks!
- Parsimonious model specification
- Reduces overfitting

$$\mathcal{L}(\theta) = \sum_{t=1}^T \rho(Y_t, \hat{Y}_t). \quad (4)$$

- The computational cost of the backpropagation algorithm is  $\mathcal{O}(T)$ .
- For long sequences, the computational cost is prohibitively large.
- To address this, an *ad hoc* method called “Truncated Backpropagation through Time” is used in practice.

The objective function at the  $k$ -th iteration is:

$$\mathcal{L}^k(\theta^{(k)}) = \sum_{t=\tau k+1}^{(k+1)\tau} \rho(Y_t, \hat{Y}_t), \quad (5)$$

The **truncation length** is  $\tau \ll T$ .

$$\begin{aligned} \hat{Y}_t &= f_Y(X_t, S_{t-1}; \theta^{(k)}), & \tau k + 1 \leq t \leq (k+1)\tau, \\ S_t &= f_S(X_t, S_{t-1}; \theta^{(k)}), & \tau k + 1 \leq t \leq (k+1)\tau. \end{aligned} \quad (6)$$

Notice that

$$S_{\tau k} = g\left(\theta^{(k-1)}\right). \quad (7)$$

The SGD update is

$$\theta^{(k+1)} = \theta^{(k)} - \alpha^{(k)} \nabla_{\theta^{(k)}} \mathcal{L}^k(\theta^{(k)}). \quad (8)$$



$$\begin{aligned}
\mathcal{L}^k(\theta^{(k)}) &= \sum_{t=\tau k+1}^{(k+1)\tau} \rho(Y_t, \hat{Y}_t), \\
\theta^{(k+1)} &= \theta^{(k)} - \alpha^{(k)} \nabla_{\theta^{(k)}} \mathcal{L}^k(\theta^{(k)}).
\end{aligned} \tag{9}$$

Unlike traditional SGD, “Truncated Backpropagation through Time” is **biased**!

Nonetheless, it works well in practice.

The computational cost is  $\mathcal{O}(\tau)$ .

How do we implement mini-batch Truncated Backpropagation through Time?

- Truncated Backpropagation through Time does not fully optimize over the full sequence of the data.
- However, information does flow forward from time  $t$  to **all** future times  $t' > t$  via the internal state  $S_{t'}$ .
- Recurrent neural networks (RNN) can be viewed as very deep networks, where the number of layers is  $T$  (or  $\tau$ ).
- Vanishing gradient problem

## PyTorch

```
model.hidden = ( model.hidden[0].detach(), model.hidden[1].detach())
```

“`Z.detach()`” breaks the computational chain, i.e. it treats  $Z$  as a constant.

PyTorch, with its define-by-run framework, allows for breaking computational graph at any time during training.

## Backpropagation.

$$\begin{aligned}\hat{Y}_t &= f_Y(X_t, S_{t-1}; \theta), \\ S_t &= f_S(X_t, S_{t-1}; \theta), \\ \mathcal{L}(\theta) &= \sum_{t=1}^T \rho(Y_t, \hat{Y}_t).\end{aligned}\tag{11}$$

The parameters  $\theta$  are **shared** across times  $t = 1, 2, \dots, T$ . In particular,  $S_1, \dots, S_T$  depend upon  $\theta$  in a complicated way.

Let us begin by considering a generic function  $G : \Theta^T \rightarrow \mathbb{R}$ . That is,  $G$  is a function  $T$  inputs, each taking values in  $\Theta$ . Then,

$$\begin{aligned} G(\theta, \dots, \theta) &= G(\theta_1, \dots, \theta_T), \\ \theta_t &= g(\theta), \quad t = 1, 2, \dots, T, \end{aligned} \tag{12}$$

where  $g$  is the identity function  $g(z) = z$ . Then, from chain rule, we have that

$$\begin{aligned} \nabla_{\theta} G(\theta, \dots, \theta) &= \sum_{t=1}^T \nabla_{\theta_t} G(\theta_1, \dots, \theta_T) \frac{dg}{d\theta}(\theta) \\ &= \sum_{t=1}^T \nabla_{\theta_t} G(\theta_1, \dots, \theta_T). \end{aligned} \tag{13}$$

Therefore, it is equivalent to derive the backpropagation rule for

$$\begin{aligned}\hat{Y}_t &= f_Y(X_t, S_{t-1}; \theta_t), \\ S_t &= f_S(X_t, S_{t-1}; \theta_t), \\ \tilde{\mathcal{L}}(\theta_1, \dots, \theta_T) &= \sum_{t=1}^T \rho(Y_t, \hat{Y}_t), \\ \theta_t &= g(\theta). \end{aligned} \tag{14}$$

This can be viewed as a multi-layer network where  $\theta_t$  is the parameter vector for the  $t$ -th layer!

We therefore have that

$$\begin{aligned}
 \nabla_{\theta} \mathcal{L}(\theta) &= \nabla_{\theta} \sum_{t=1}^T \rho\left(Y_t, f_Y(X_t, S_{t-1}; \theta_t)\right) \\
 &= \sum_{t=1}^T \nabla_{\theta_t} \tilde{\mathcal{L}}(\theta_1, \dots, \theta_T).
 \end{aligned} \tag{15}$$

Let's begin by considering  $\nabla_{\theta_{\tau}} \tilde{\mathcal{L}}(\theta_1, \dots, \theta_T)$ . If  $\tau = T$ ,

$$\begin{aligned}
 \nabla_{\theta_{\tau}} \tilde{\mathcal{L}}(\theta_1, \dots, \theta_T) &= \nabla_{\theta_{\tau}} \sum_{t=1}^T \rho(Y_t, \hat{Y}_t) \\
 &= \nabla_{\theta_T} \rho(Y_T, \hat{Y}_T) \\
 &= \frac{\partial \rho}{\partial \hat{Y}}(Y_T, \hat{Y}_T) \frac{\partial f_Y}{\partial \theta}(X_T, S_{T-1}; \theta_T).
 \end{aligned} \tag{16}$$



If  $\tau < T$ ,

$$\begin{aligned}
\nabla_{\theta_\tau} \tilde{\mathcal{L}}(\theta_1, \dots, \theta_T) &= \nabla_{\theta_\tau} \sum_{t=1}^T \rho(Y_t, \hat{Y}_t) \\
&= \nabla_{\theta_\tau} \sum_{t=\tau}^T \rho(Y_t, \hat{Y}_t) \\
&= \frac{\partial \rho}{\partial \hat{Y}}(Y_\tau, \hat{Y}_\tau) \frac{\partial f_Y}{\partial \theta}(X_\tau, S_{\tau-1}; \theta_\tau) + \nabla_{\theta_\tau} \sum_{t=\tau+1}^T \rho(Y_t, \hat{Y}_t) \\
&= \frac{\partial \rho}{\partial \hat{Y}}(Y_\tau, \hat{Y}_\tau) \frac{\partial f_Y}{\partial \theta}(X_\tau, S_{\tau-1}; \theta_\tau) + \nabla_{S_\tau} \tilde{\mathcal{L}}(\theta_1, \dots, \theta_T) \frac{\partial S_\tau}{\partial \theta_\tau}.
\end{aligned} \tag{17}$$

Combining results yields

$$\begin{aligned}\nabla_{\theta}\mathcal{L}(\theta) &= \sum_{\tau=1}^T \nabla_{\theta_{\tau}} \tilde{\mathcal{L}}(\theta_1, \dots, \theta_T) \\ &= \sum_{\tau=1}^T \frac{\partial \rho}{\partial \hat{Y}}(Y_{\tau}, \hat{Y}_{\tau}) \frac{\partial f_Y}{\partial \theta}(X_{\tau}, S_{\tau-1}; \theta_{\tau}) \\ &\quad + \sum_{\tau=1}^{T-1} \nabla_{S_{\tau}} \tilde{\mathcal{L}}(\theta_1, \dots, \theta_T) \frac{\partial S_{\tau}}{\partial \theta_{\tau}}.\end{aligned}\tag{18}$$

Define

$$\delta^t = \nabla_{S_t} \tilde{\mathcal{L}}(\theta_1, \dots, \theta_T). \quad (19)$$

Recall that

$$\begin{aligned} \hat{Y}_t &= f_Y(X_t, S_{t-1}; \theta_t), \\ S_t &= f_S(X_t, S_{t-1}; \theta_t), \\ \tilde{\mathcal{L}}(\theta_1, \dots, \theta_T) &= \sum_{t=1}^T \rho(Y_t, \hat{Y}_t), \\ \theta_t &= g(\theta). \end{aligned} \quad (20)$$

By chain rule, for  $t = 1, \dots, T-1$ ,

$$\begin{aligned} \delta^t &= \frac{\partial f_S}{\partial S}(X_{t+1}, S_t; \theta_{t+1}) \delta^{t+1} \\ &+ \frac{\partial f_Y}{\partial S}(X_{t+1}, S_t; \theta_{t+1}) \frac{\partial \rho}{\partial \hat{Y}}(Y_{t+1}, \hat{Y}_{t+1}), \end{aligned} \quad (21)$$

and  $\delta^T = 0$ .

Therefore,

$$\begin{aligned}\nabla_{\theta}\mathcal{L}(\theta) &= \sum_{t=1}^T \frac{\partial \rho}{\partial \hat{Y}}(Y_t, \hat{Y}_t) \frac{\partial f_Y}{\partial \theta}(X_t, S_{t-1}; \theta_t) \\ &\quad + \sum_{t=1}^{T-1} \delta^t \frac{\partial S_t}{\partial \theta_t},\end{aligned}\tag{22}$$

where

$$\begin{aligned}\delta^t &= \frac{\partial f_S}{\partial S}(X_{t+1}, S_t; \theta_{t+1}) \delta^{t+1} \\ &\quad + \frac{\partial f_Y}{\partial S}(X_{t+1}, S_t; \theta_{t+1}) \frac{\partial \rho}{\partial \hat{Y}}(Y_{t+1}, \hat{Y}_{t+1}),\end{aligned}\tag{23}$$

and  $\delta^T = 0$ .

$$\begin{aligned}\nabla_{\theta}\mathcal{L}(\theta) &= \sum_{t=1}^T \frac{\partial \rho}{\partial \hat{Y}}(Y_t, \hat{Y}_t) \frac{\partial f_Y}{\partial \theta}(X_t, S_{t-1}; \theta_t) \\ &\quad + \sum_{t=1}^{T-1} \delta^t \frac{\partial f_S}{\partial \theta}(X_t, S_{t-1}; \theta_t),\end{aligned}\tag{24}$$

since

$$\frac{\partial S_t}{\partial \theta_t} = \frac{\partial f_S}{\partial \theta}(X_t, S_{t-1}; \theta_t).\tag{25}$$

Therefore, the backpropagation algorithm is:

- For  $t = 1, 2, \dots, T$ :
  - Calculate  $(\hat{Y}_t, S_t)$
- $\delta^T = 0$  and set

$$G = \frac{\partial \rho}{\partial \hat{Y}}(Y_T, \hat{Y}_T) \frac{\partial f_Y}{\partial \theta}(X_T, S_{T-1}; \theta_T). \quad (26)$$

- For  $t = T - 1, T - 2, \dots, 1$ :
  - Calculate

$$\begin{aligned} \delta^t &= \frac{\partial f_S}{\partial S}(X_{t+1}, S_t; \theta_{t+1}) \delta^{t+1} \\ &+ \frac{\partial f_Y}{\partial S}(X_{t+1}, S_t; \theta_{t+1}) \frac{\partial \rho}{\partial \hat{Y}}(Y_{t+1}, \hat{Y}_{t+1}). \end{aligned} \quad (27)$$

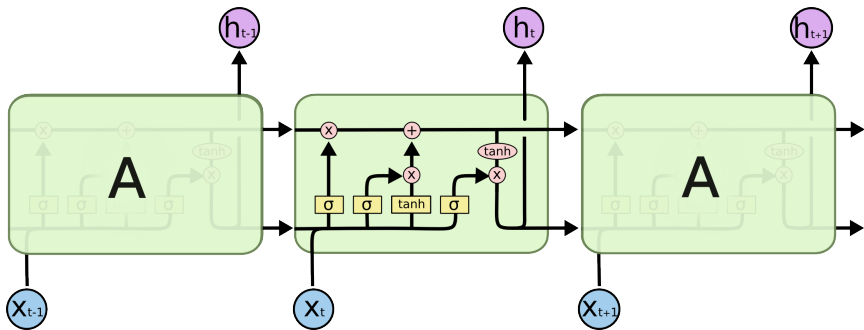
- Calculate

$$G \rightarrow G + \frac{\partial \rho}{\partial \hat{Y}}(Y_t, \hat{Y}_t) \frac{\partial f_Y}{\partial \theta}(X_t, S_{t-1}; \theta_t) + \delta^t \frac{\partial f_S}{\partial \theta}(X_t, S_{t-1}; \theta_t). \quad (28)$$

- Take a gradient descent step in the direction  $-G$ .

The computational cost of the backpropagation through time (BPTT) algorithm is  $\mathcal{O}(T)$ .

## Long short-term memory (LSTM) networks.



Source: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>



- LSTM networks
- Gated Recurrent Unit (GRU) networks

- Mini-batch version of Truncated Backpropagation through Time.
- Skip/shortcut connections
- Gradient clipping

$$\tilde{G} = \max \left( -c, \min(G, c) \right), \quad (29)$$

where  $c > 0$ .

Implementing LSTMs in [PyTorch](#)..

## Residual Networks.

- Deep networks more challenging to train.
- Vanishing gradient problem
- Introduce skip/shortcut connections to facilitate the “flow of information” from lower layers to higher layers.

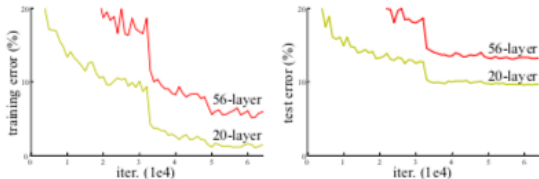


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer “plain” networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

Source: “Deep Residual Learning for Image Recognition” by He et al. (2015)

- If dimensions match, no extra parameters involved in the residual connection.
- If dimensions do not match, can use a linear connection with parameter  $W^{s,\ell}$ .
- Lower computational complexity than VGG network: 3.6 versus 19.6 billion FLOPs.
- Capable of building very deep networks (e.g., 150 layers).

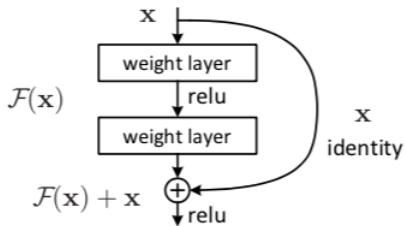


Figure 2. Residual learning: a building block.

Source: "Deep Residual Learning for Image Recognition" by He et al. (2015)





	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	<b>25.03</b>

Source: "Deep Residual Learning for Image Recognition" by He et al. (2015)

method	top-1 err.	top-5 err.
VGG [40] (ILSVRC' 14)	-	8.43 <sup>†</sup>
GoogLeNet [43] (ILSVRC' 14)	-	7.89
VGG [40] (v5)	24.4	7.1
PReLU-net [12]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	<b>19.38</b>	<b>4.49</b>

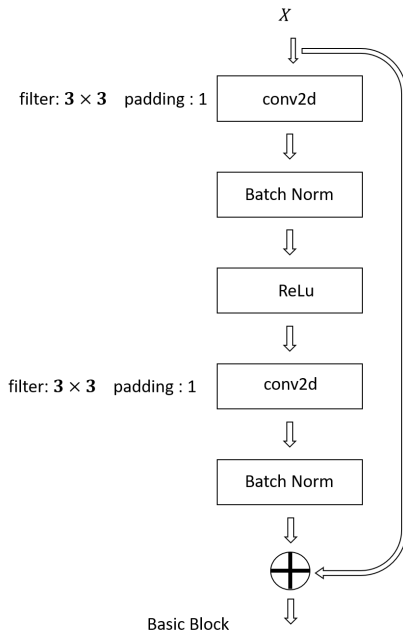
Source: “Deep Residual Learning for Image Recognition” by He et al. (2015)

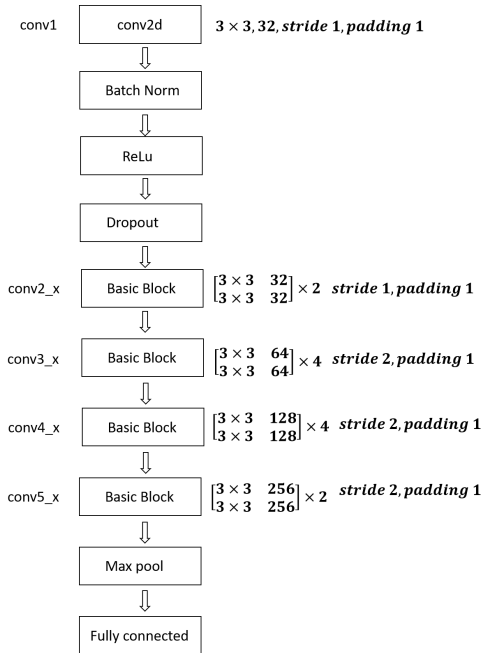


Figure 5. A deeper residual function  $\mathcal{F}$  for ImageNet. Left: a building block (on  $56 \times 56$  feature maps) as in Fig. 3 for ResNet-34. Right: a “bottleneck” building block for ResNet-50/101/152.

Source: “Deep Residual Learning for Image Recognition” by He et al. (2015)

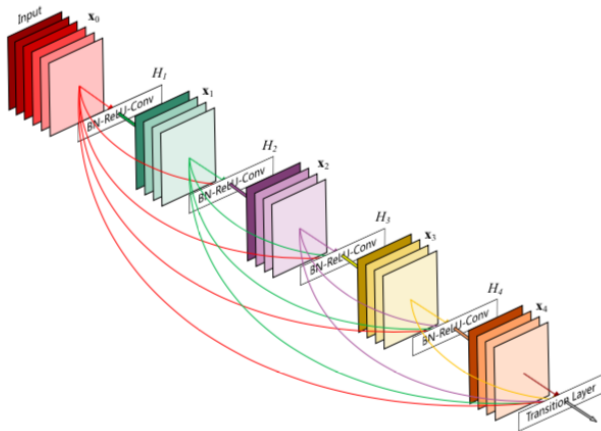
## Homework 4.





- CIFAR100 (60,000 images and 100 classes)
- Train a Residual Network from scratch
- Fine-tune a pre-trained residual network on CIFAR100
- Residual network is pre-trained on ImageNet (14 million images)

- “Densely Connected Networks” by Huang et al. (2016)
- Concatenate hidden layers  $\ell - \tau, \dots, \ell - 1$  as inputs to hidden layer  $\ell$ .
- Memory intensive
- Small number of channels



**Figure 1:** A 5-layer dense block with a growth rate of  $k = 4$ . Each layer takes all preceding feature-maps as input.

Source: “Densely Connected Networks” by Huang et al. (2016)



Method	Depth	Params	C10	C10+	C100	C100+	SVHN
Network in Network [22]	-	-	10.41	8.81	35.68	-	2.35
All-CNN [31]	-	-	9.08	7.25	-	33.71	-
Deeply Supervised Net [20]	-	-	9.69	7.97	-	34.57	1.92
Highway Network [33]	-	-	-	7.72	-	32.39	-
FractalNet [17]	21	38.6M	10.18	5.22	35.34	23.30	2.01
with Dropout/Drop-path	21	38.6M	7.33	4.60	28.20	23.73	1.87
ResNet [11]	110	1.7M	-	6.61	-	-	-
ResNet (reported by [13])	110	1.7M	13.63	6.41	44.74	27.22	2.01
ResNet with Stochastic Depth [13]	110	1.7M	11.66	5.23	37.80	24.58	1.75
	1202	10.2M	-	4.91	-	-	-
Wide ResNet [41]	16	11.0M	-	4.81	-	22.07	-
	28	36.5M	-	4.17	-	20.50	-
with Dropout	16	2.7M	-	-	-	-	1.64
ResNet (pre-activation) [12]	164	1.7M	11.26*	5.46	35.58*	24.33	-
	1001	10.2M	10.56*	4.62	33.47*	22.71	-
DenseNet ( $k = 12$ )	40	1.0M	<b>7.00</b>	5.24	<b>27.55</b>	24.42	1.79
DenseNet ( $k = 12$ )	100	7.0M	<b>5.77</b>	<b>4.10</b>	<b>23.79</b>	<b>20.20</b>	1.67
DenseNet ( $k = 24$ )	100	27.2M	<b>5.83</b>	<b>3.74</b>	<b>23.42</b>	<b>19.25</b>	<b>1.59</b>
DenseNet-BC ( $k = 12$ )	100	0.8M	<b>5.92</b>	4.51	<b>24.15</b>	22.27	1.76
DenseNet-BC ( $k = 24$ )	250	15.3M	<b>5.19</b>	<b>3.62</b>	<b>19.64</b>	<b>17.60</b>	1.74
DenseNet-BC ( $k = 40$ )	190	25.6M	-	<b>3.46</b>	-	<b>17.18</b>	-

Source: “Densely Connected Networks” by Huang et al. (2016)