Natural Language Processing with PyTorch

Week 2 Deep Neural Networks in PyTorch

Fast campus

Review & Warranty

Contents

1. Convolutional Neural Networks
   - The convolutional layer
   - Batch normalization
   - Residual connections
   - Variations: dilated convolution, deconvolution, separable convolution

Contents

2. Recurrent Neural Networks
    - The recurrent layer
    - Gradient vanishing and exploding
    - Gradient clipping
    - Variations: Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU)

Contents

Convolutional Neural Networks

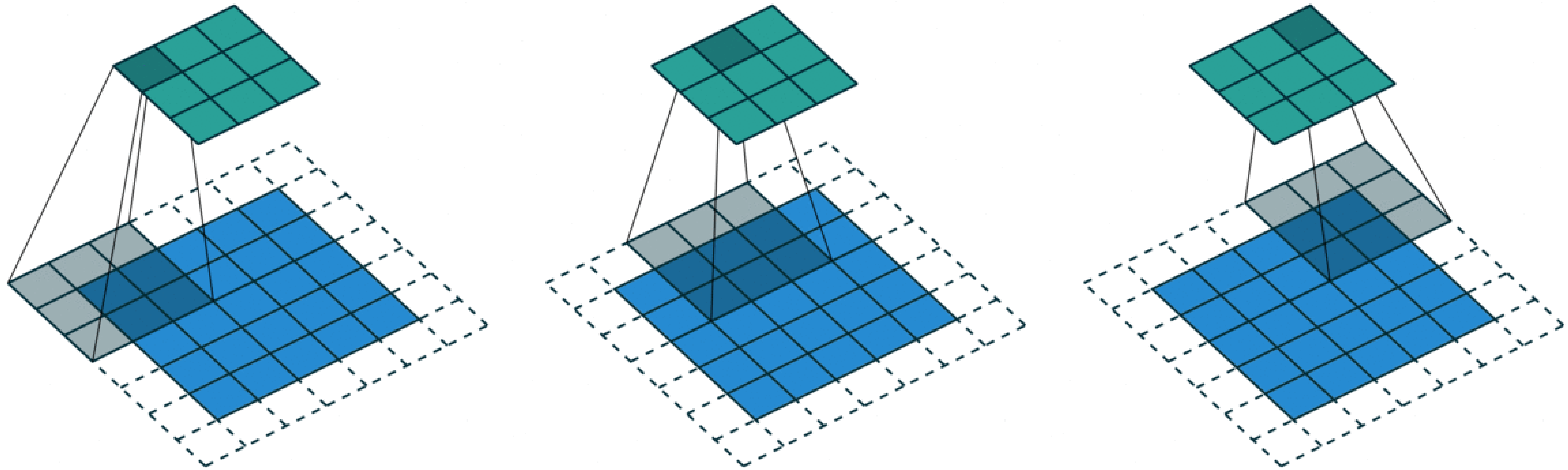
01

Fast campus

Contents

- The convolutional layer
- Batch normalization
- Residual connections
- Variations: dilated convolution, deconvolution, separable convolution

The convolutional layer



```
torch.nn.Conv2d(in_channels=1, out_channels=1, kernel=(3, 3),
stride=2, padding=1, dilation=1, bias=False)
```

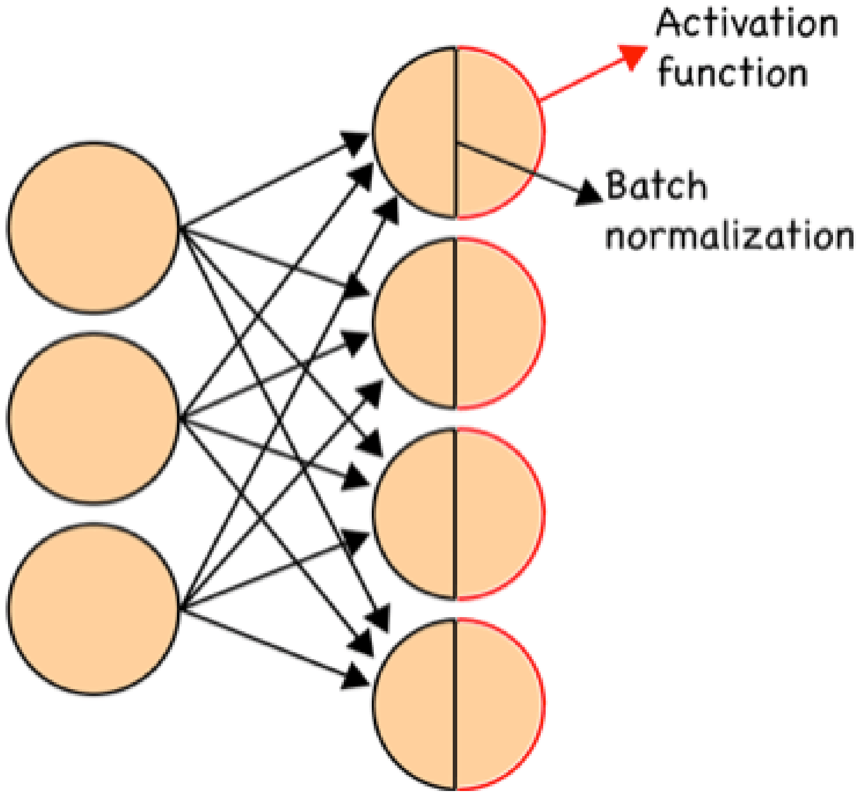

https://github.com/vdumoulin/conv_arithmetic

Batch normalization

> Training Deep Neural Networks is complicated by the fact that **the distribution of each layer's inputs changes during training**, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities. We refer to this phenomenon as **internal covariate shift**, and address the problem by **normalizing layer inputs**.

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift (2015)

Batch normalization



https://shuuki4.wordpress.com/2016/01/13/batch-normalization-%EC%84%A4%EB%AA%85-%EB%B0%8F-%EA%B5%AC%ED%98%84/

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

Residual connections

ResNet

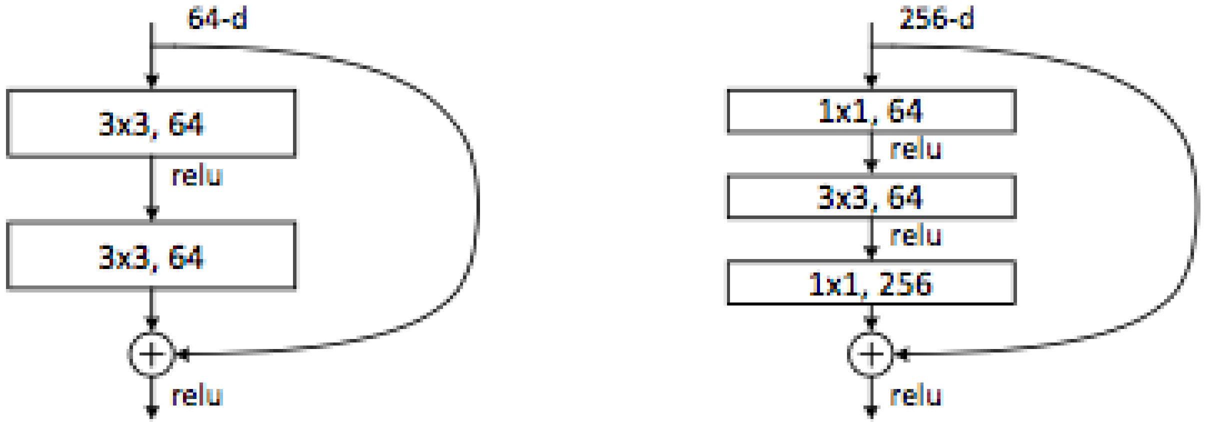


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

Residual connections

DenseNet

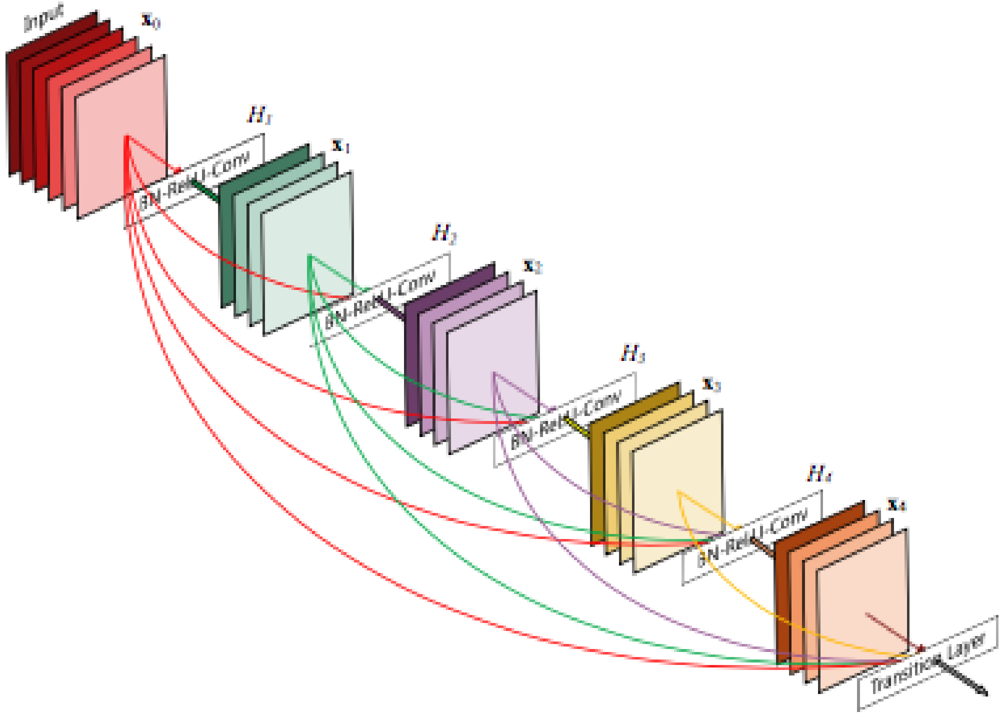


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

Densely Connected Convolutional Networks

Residual connections

CondenseNet

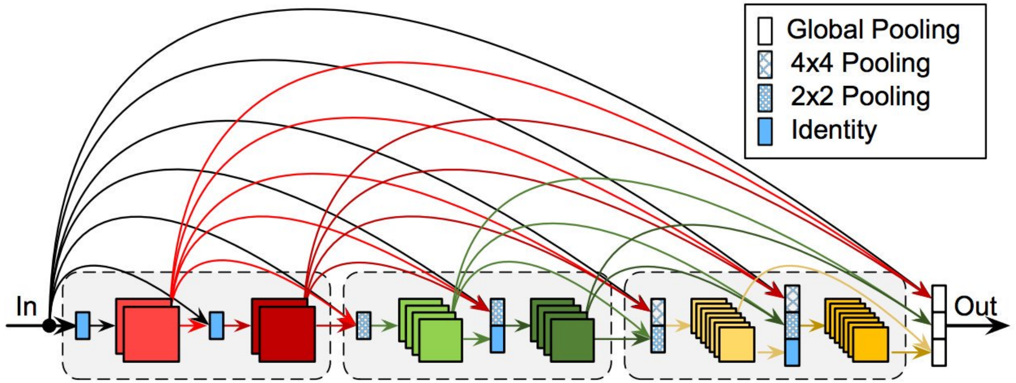
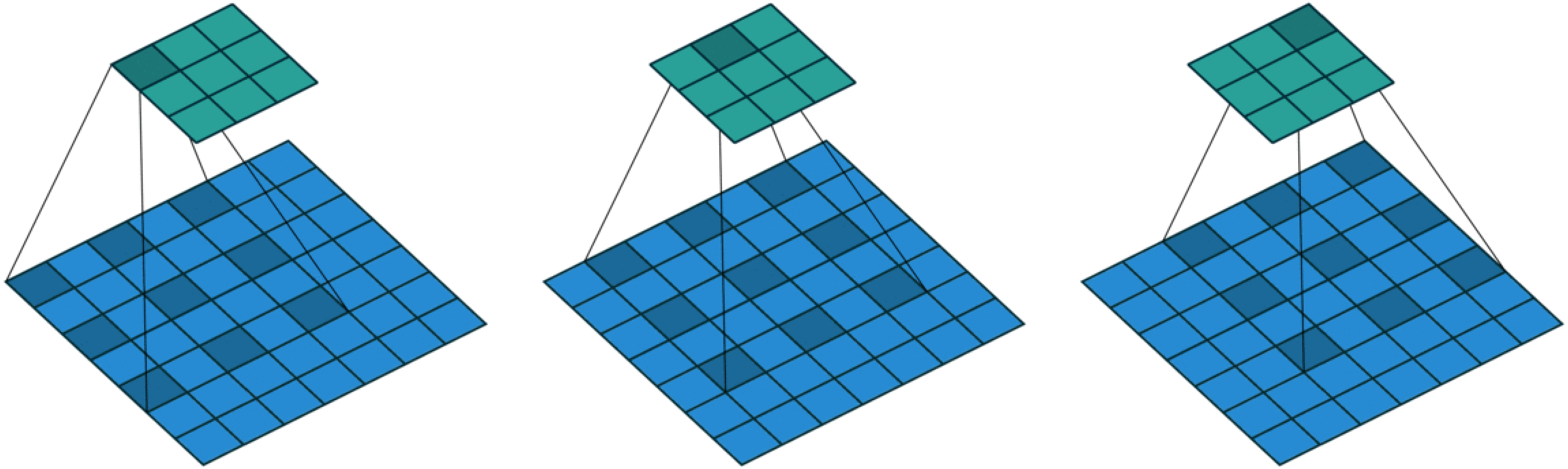


Figure 5. The proposed DenseNet variant. It differs from the original DenseNet in two ways: (1) layers with different resolution feature maps are also directly connected; (2) the growth rate doubles whenever the feature map size shrinks (far more features are generated in the third, yellow, dense block than in the first).

Variations

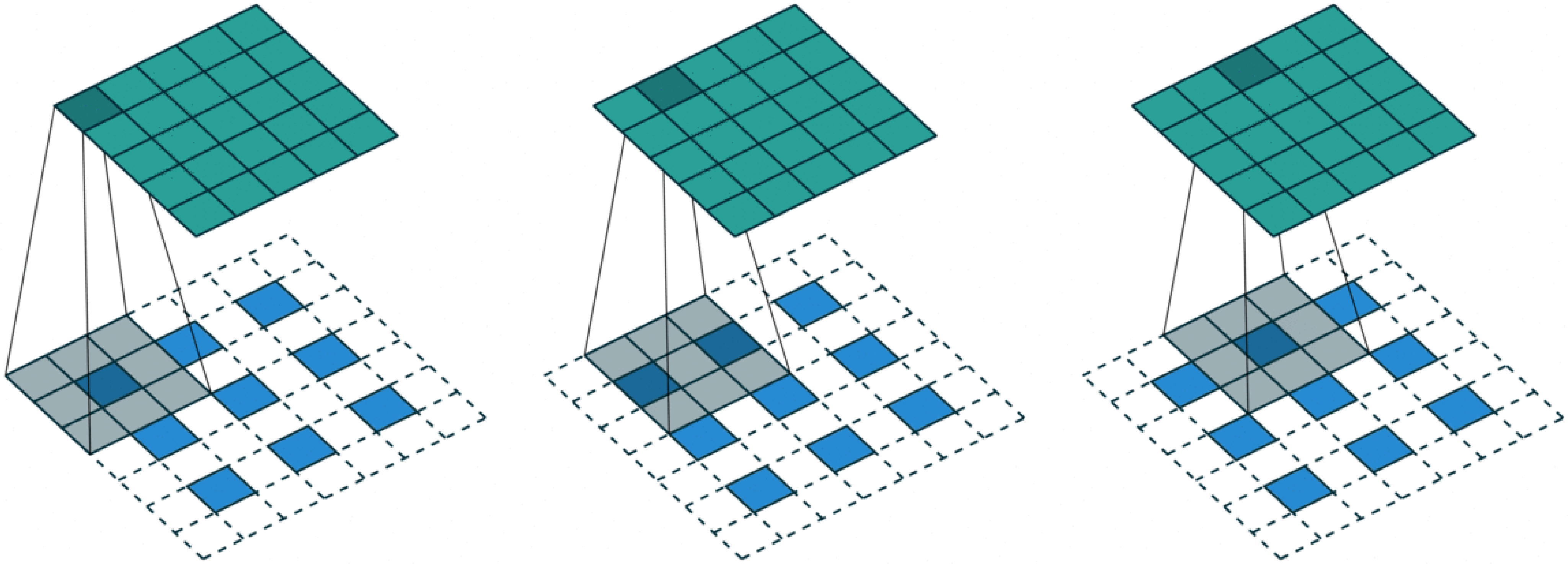
- Dilated convolution (a.k.a. atrous convolution
- Deconvolution (a.k.a. transposed convolution, fractionally strided convolution, upconvolution)
- Separable convolution

Dilated convolution



```
torch.nn.Conv2d(in_channels=1, out_channels=1, kernel=(3, 3),
stride=2, padding=1, dilation=2, bias=False)
```

Deconvolution



```
torch.nn.ConvTranspose2d(in_channels=1, out_channels=1, kernel=(3,
3), stride=2, padding=1, dilation=1, bias=False)
```

Separable convolution

- = Depthwise convolution + Pointwise convolution

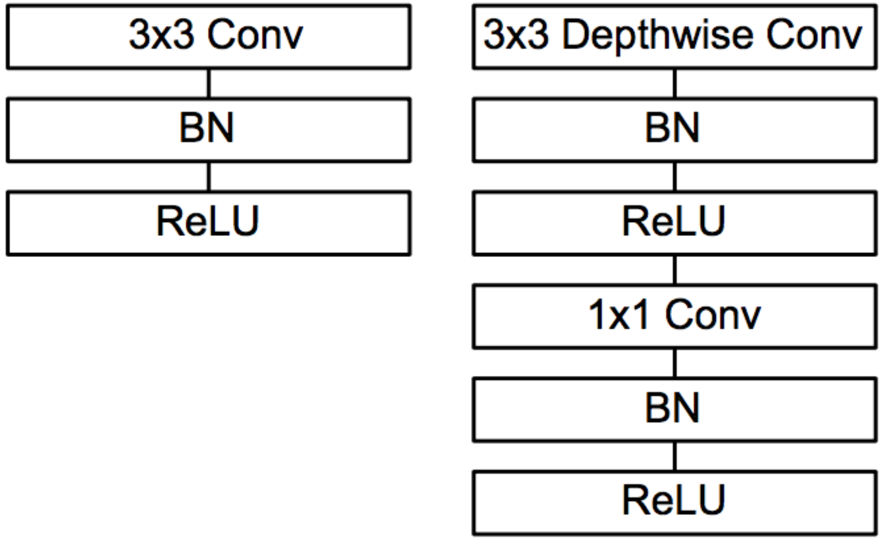
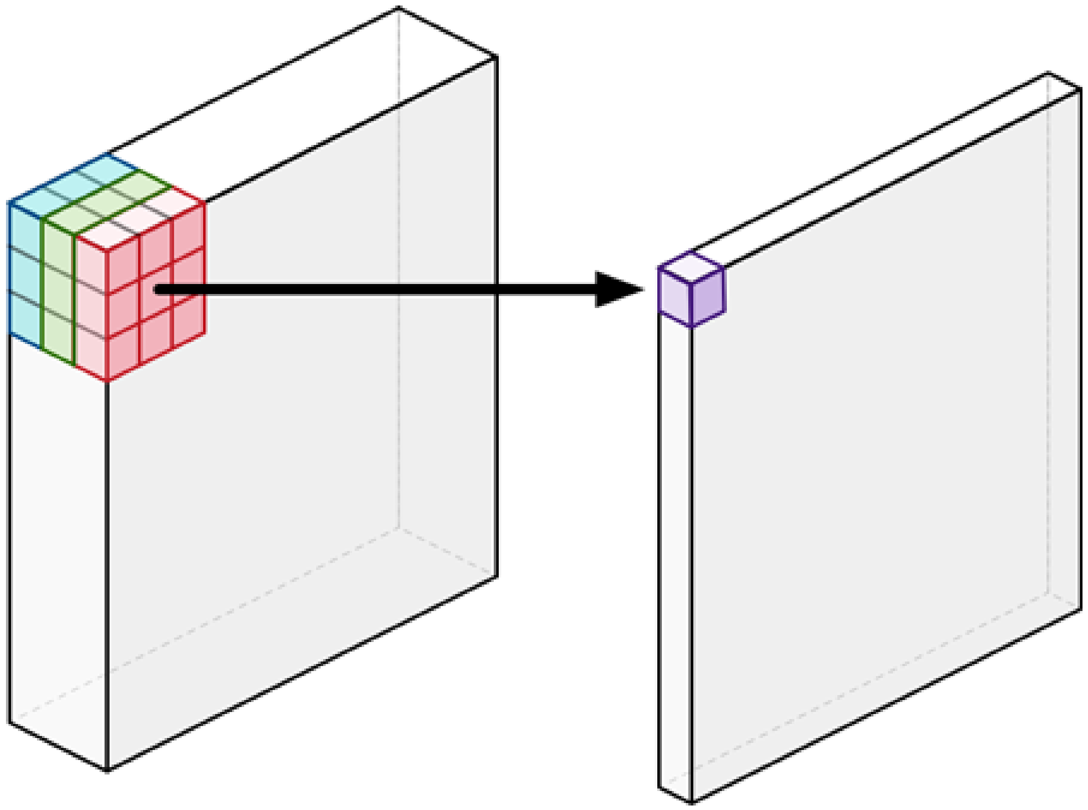


Figure 3. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.

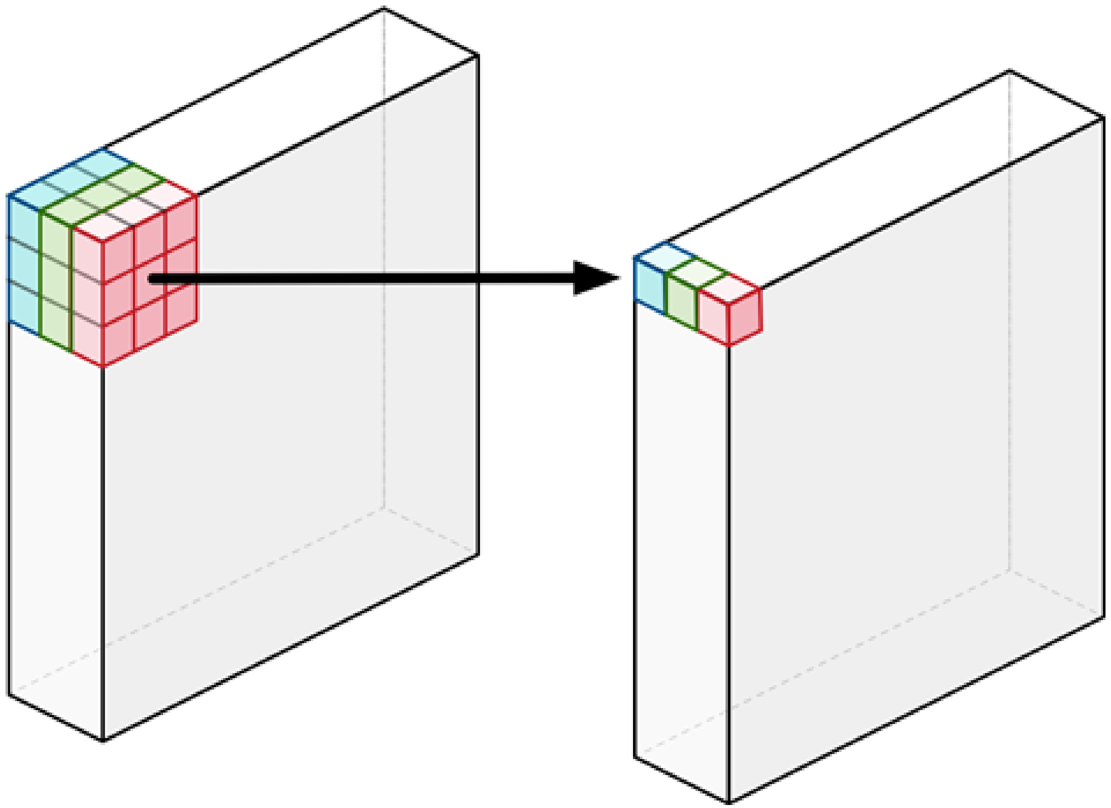
MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications

Separable convolution



```
regular = torch.nn.Conv2d(in_channels=in_channels,
out_channels=out_channels, kernel_size=3, padding=1)
```

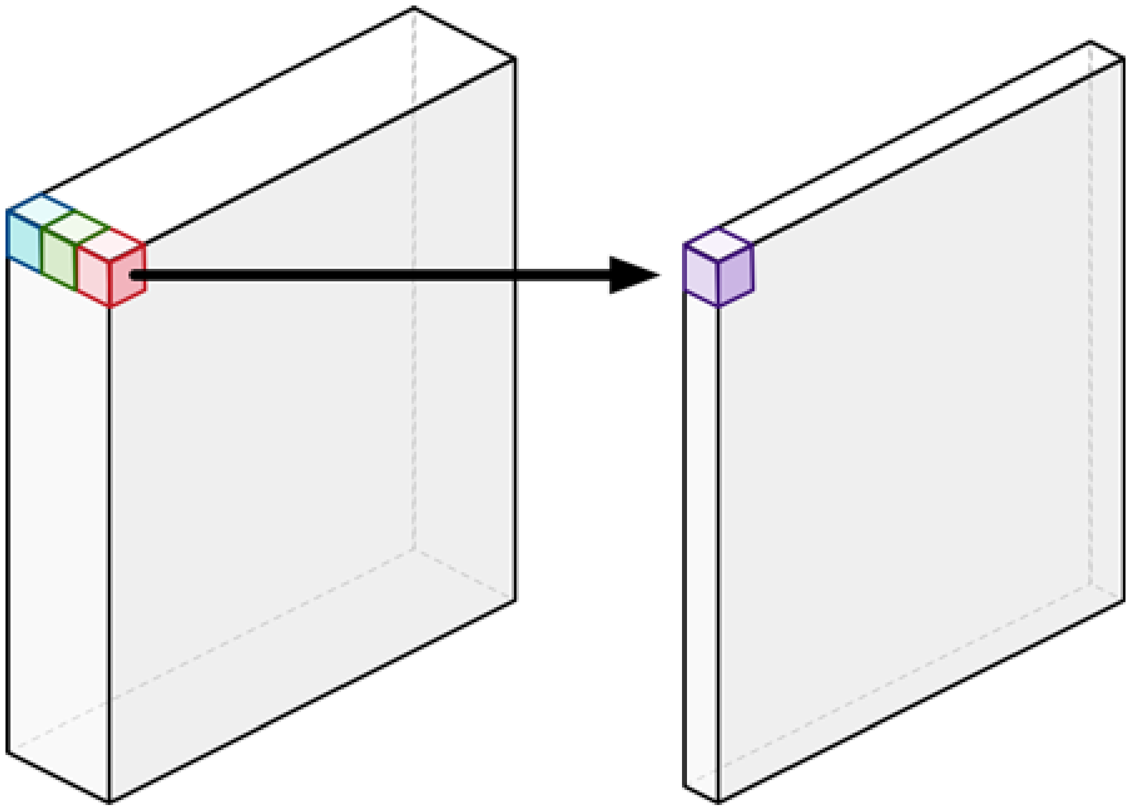
Separable convolution



```
depthwise = torch.nn.Conv2d(in_channels=in_channels,
out_channels=in_channels, kernel_size=3, padding=1,
groups=in_channels)
```


http://machinethink.net/blog/googles-mobile-net-architecture-on-iphone/

Separable convolution



```
pointwise = torch.nn.Conv2d(in_channels=in_channels,
out_channels=out_channels, kernel_size=1)
```

Separable convolution

- = Depthwise convolution + Pointwise convolution

```
depthwise = torch.nn.Conv2d(in_channels=in_channels,
out_channels=in_channels, kernel_size=3, padding=1,
groups=in_channels)
pointwise = torch.nn.Conv2d(in_channels=in_channels,
out_channels=out_channels, kernel_size=1)
```

Recurrent Neural Networks

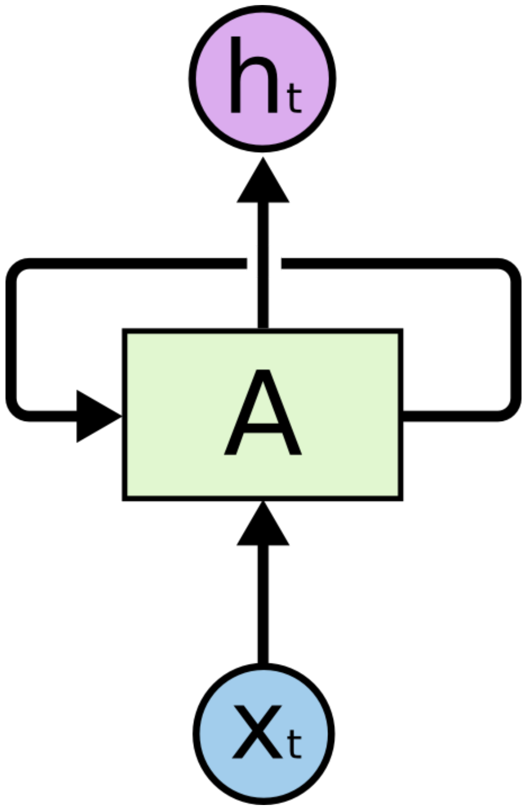
02

Contents

- The recurrent layer
- Gradient vanishing and exploding
- Gradient clipping
- Variations: Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU)

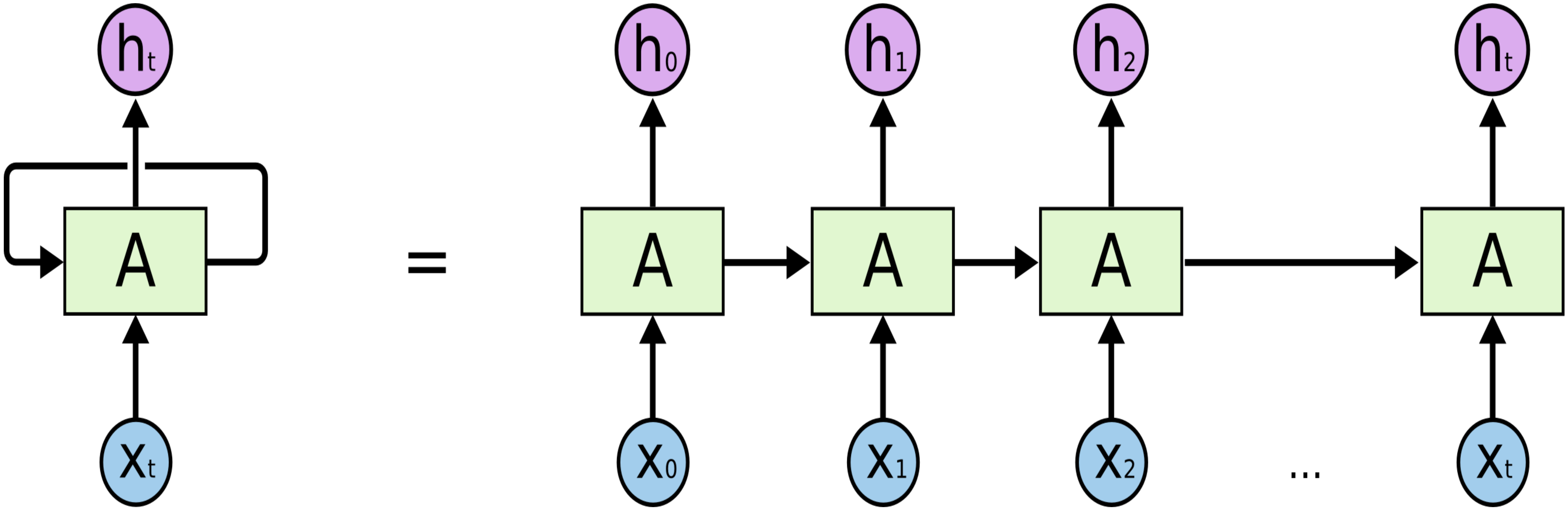
The recurrent layer

A single RNN cell:



The recurrent layer

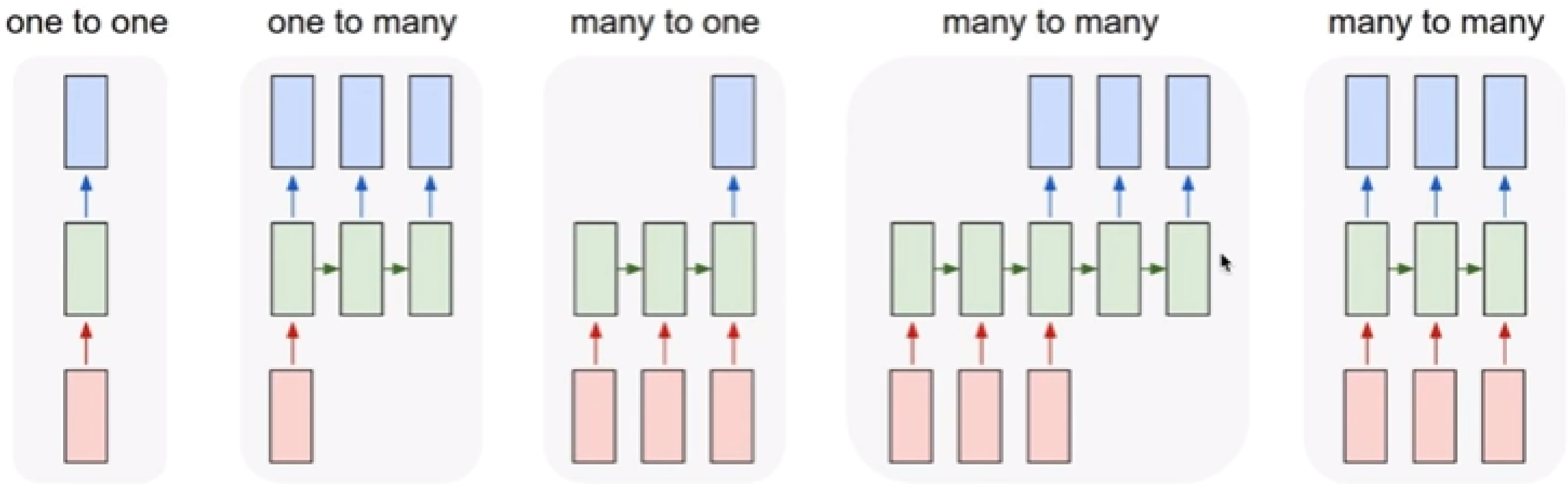
The RNN cell, unrolled:



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

The recurrent layer

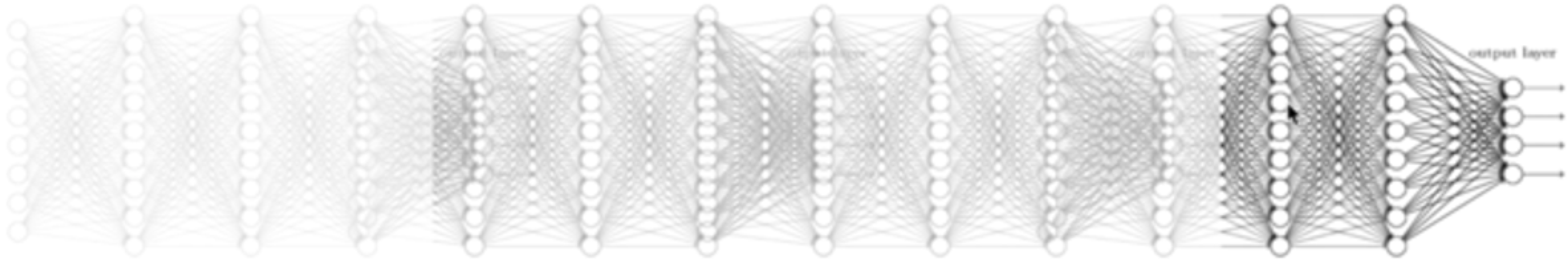
Possibilities:



Gradient vanishing and exploding

Gradient vanishing

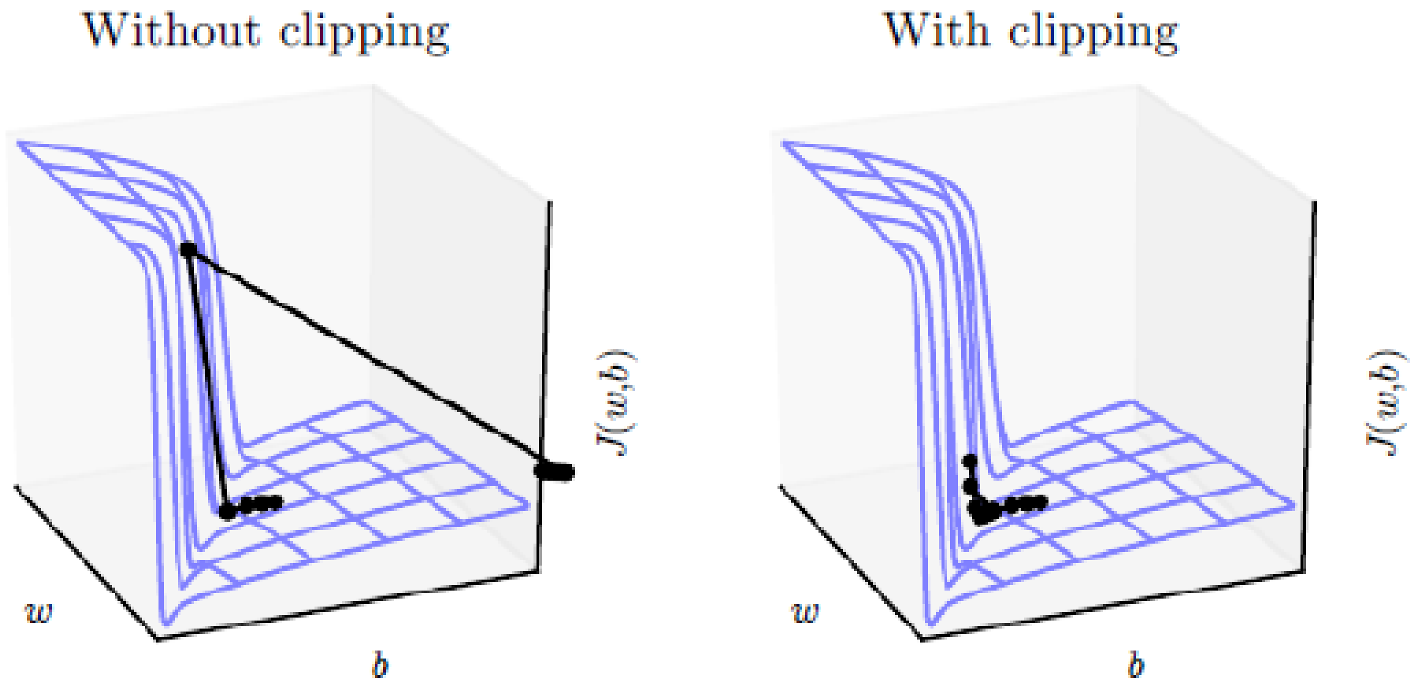
- Symptom: error signals fail to reach the beginning.
- Solution: let error signals skip layers! (e.g. ResNet, LSTM)



Gradient vanishing and exploding

Gradient exploding

- Symptom: error signals explode on "gradient cliffs".



Gradient vanishing and exploding

Gradient exploding

- Solution: set limits on gradients! (e.g. gradient clipping)

Algorithm 1 Pseudo-code for norm clipping

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$

if $\|\hat{\mathbf{g}}\| \geq \textit{threshold}$ **then**

    $\hat{\mathbf{g}} \leftarrow \frac{\textit{threshold}}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$

end if

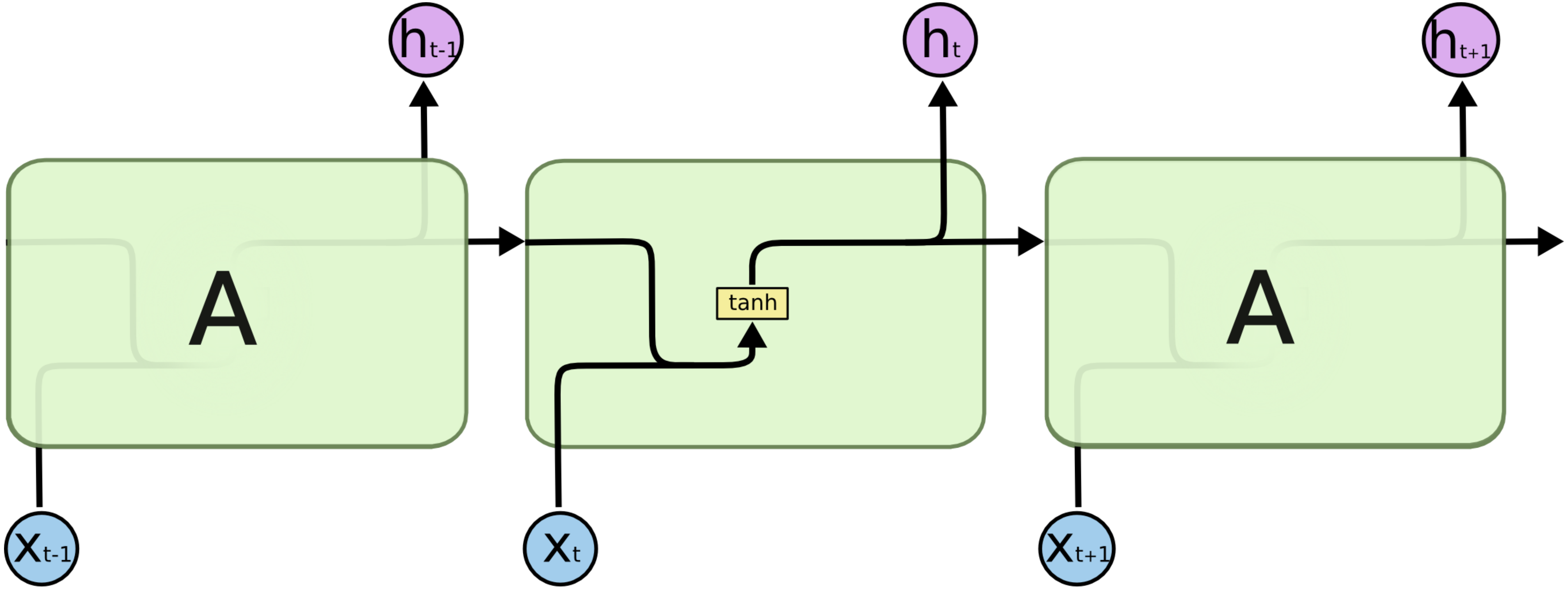
On the difficulty of training recurrent neural networks (2012)

Variations of RNN

- Long Short-Term Memory (LSTM)
- Gated Recurrent Unit (GRU)

Long Short-Term Memory (LSTM)

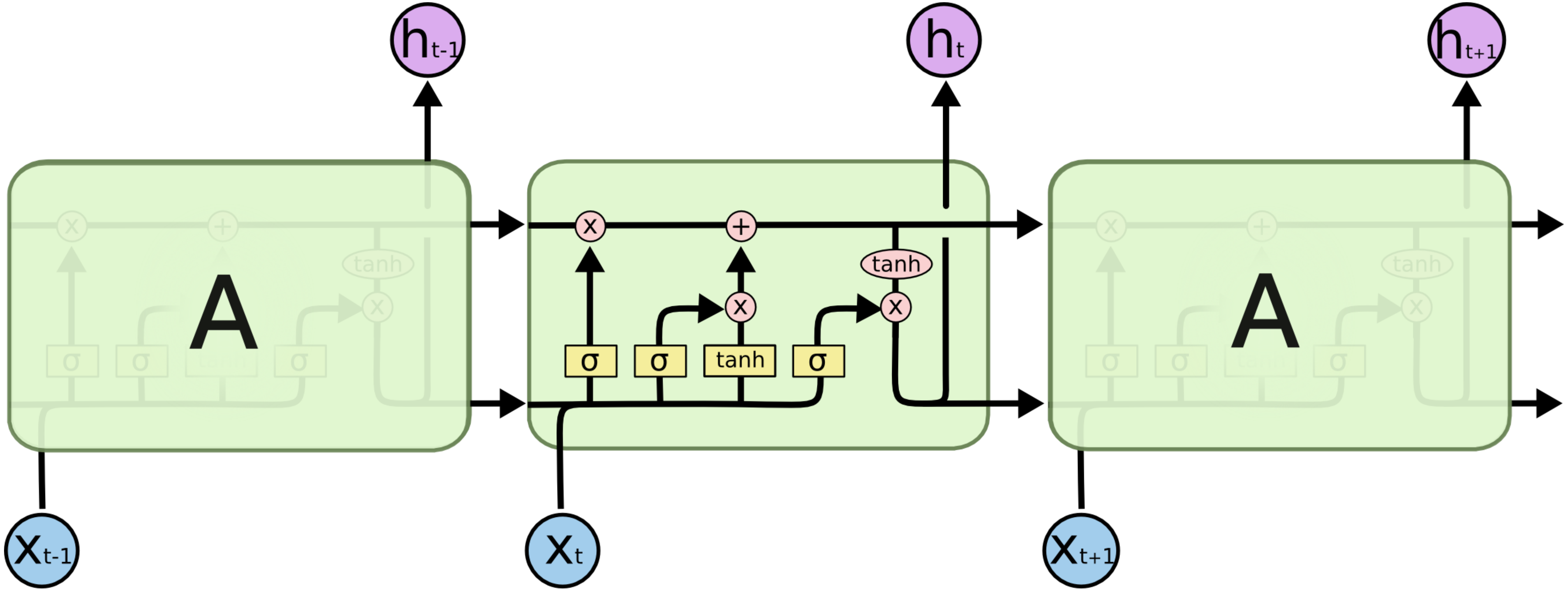
A simple RNN:



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short-Term Memory (LSTM)

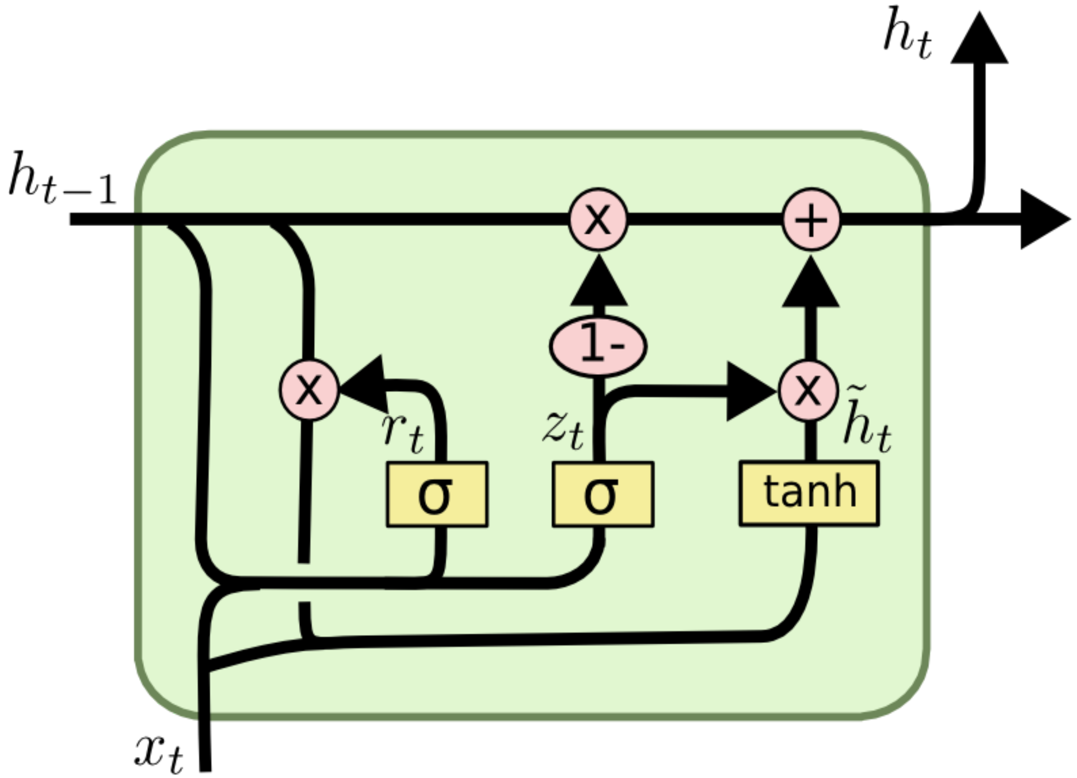
LSTM:



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Gated Recurrent Unit (GRU)

- Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation (2014)



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Cryptocurrency price prediction

https://github.com/juneoh/cryptocurrency_price_prediction

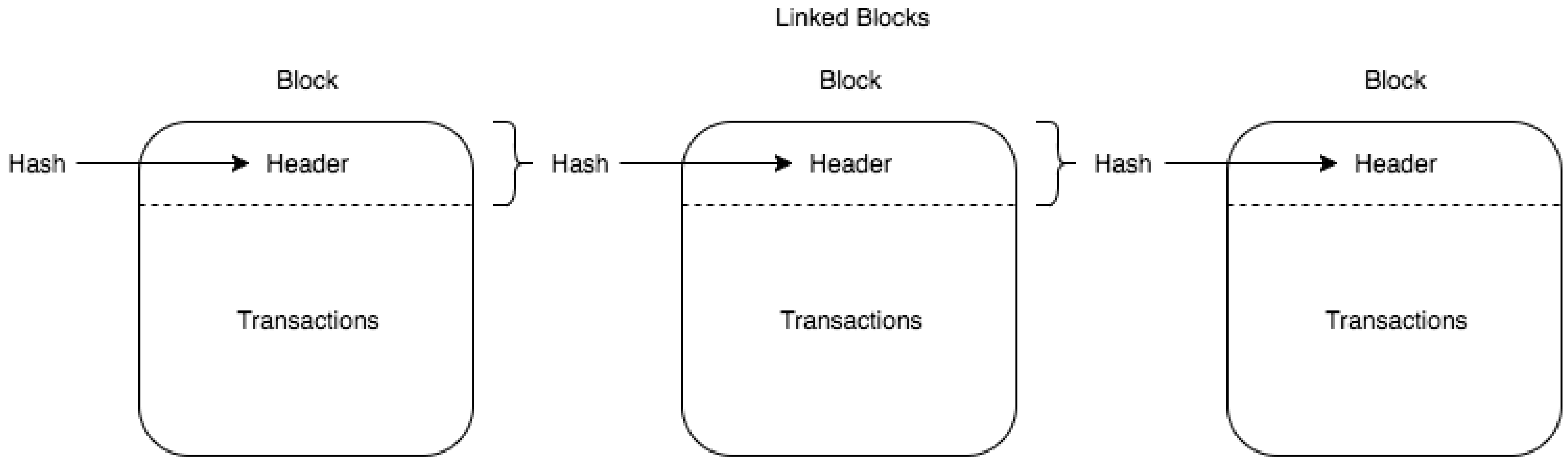
03

Contents

- Cryptocurrency 101
- Obtaining and preprocessing the data
- Building our first CNN model
- Building our first RNN model
- Running live

Cryptocurrency 101

- Cryptocurrency $\neq$ blockchain
- Blockchain-based cryptocurrencies
  - Bitcoin, Ethereum, Bitcoin Cash, ...



Fast

Obtaining and preprocessing the data

Cryptocurrencies: 1911 • Markets: 13630 • Market Cap: $225,598,996,278 • 24h Vol: $13,147,672,457 • BTC Dominance: 53.2%

English ▾ USD ▾

CoinMarketCap Rankings ▾ Trending ▾ Tools ▾ Services ▾

Search

Top 100 Cryptocurrencies By Market Capitalization

Cryptocurrencies ▾ Exchanges ▾ Watchlist

USD ▾ Next 100 → View All

#	Name	Market Cap	Price	Volume (24h)	Circulating Supply	Change (24h)	Price Graph (7d)
1	Bitcoin	$120,125,506,056	$6,967.44	$4,604,577,207	17,240,987 BTC	-0.85%	
2	Ethereum	$28,606,433,954	$281.39	$1,537,527,300	101,659,738 ETH	-2.15%	
3	XRP	$13,232,202,059	$0.333724	$258,134,529	39,650,153,121 XRP *	-2.44%	
4	Bitcoin Cash	$9,351,604,950	$539.87	$344,861,510	17,322,063 BCH	-1.88%	

Building our first CNN model

`cnn.py`

Building our first CNN model

`rnn.py`

Running live

```
python run.py
```

Thank you!