Introduction to Deep Learning (I2DL)

Exercise 11: Transfer Learning & MNIST Classification with RNNs

Today's Outline

- Transfer Learning
- Results Submission Segmentations
- RNNs and LSTMs
- Submission 11: RNN MNIST Classification
  - Start: July 9, 2020 12.00
  - End: July 15, 2020 23.59

Transfer Learning

Transfer Learning

- Problem Statement:
  - Training a Deep Neural Network needs a lot of data
  - Collecting much data is expensive or just not possible
- Idea:
  - Some problems/ tasks are closely related
  - Can we transfer knowledge from one task to another?
  - Can we re-use (at least parts of) a pre-trained network for the new task?

Transfer Learning

Distribution

Large dataset

Distribution

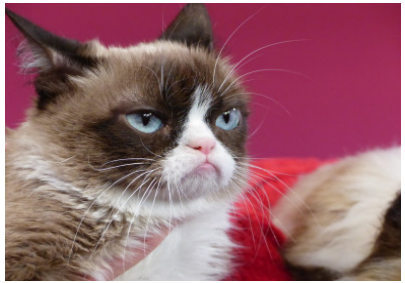
Small dataset

Use what has been learned for another setting

When Transfer Learning makes Sense

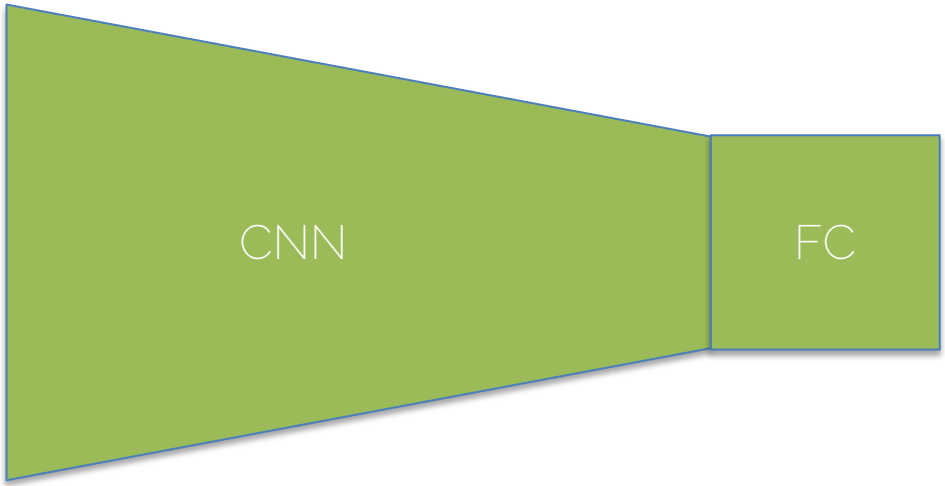
- When task P1 and P2 have the same input (e.g. an RGB image)
- When you have more data for task P1 than for task P2
- When the low-level features for P1 could be useful to learn P2

Transfer Learning: Scenario

- Need to build a Cat classifier
- Only have a few images ~100

Transfer Learning

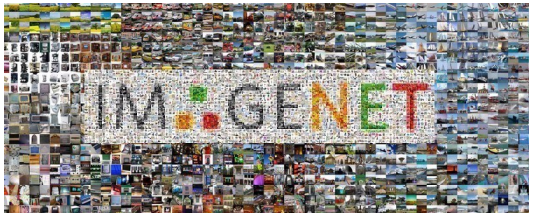


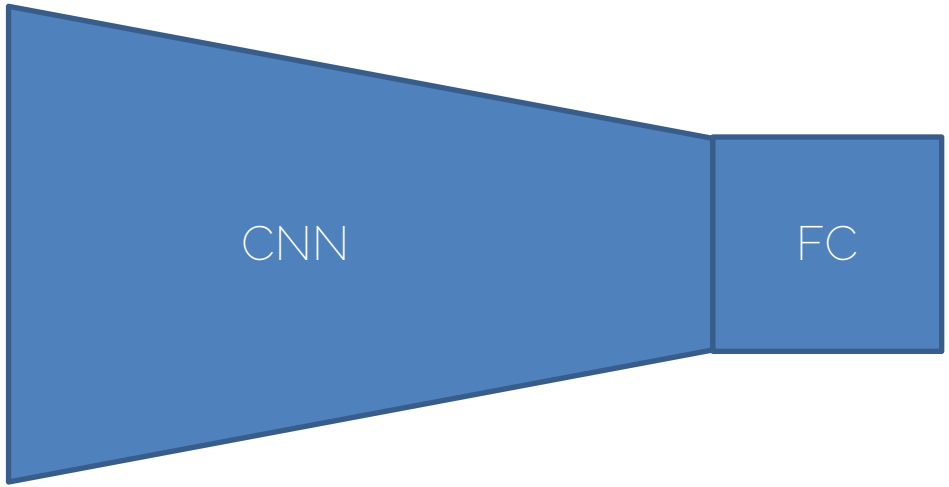
Coloring Legend:

Untrained

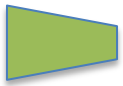
Trained

Transfer Learning





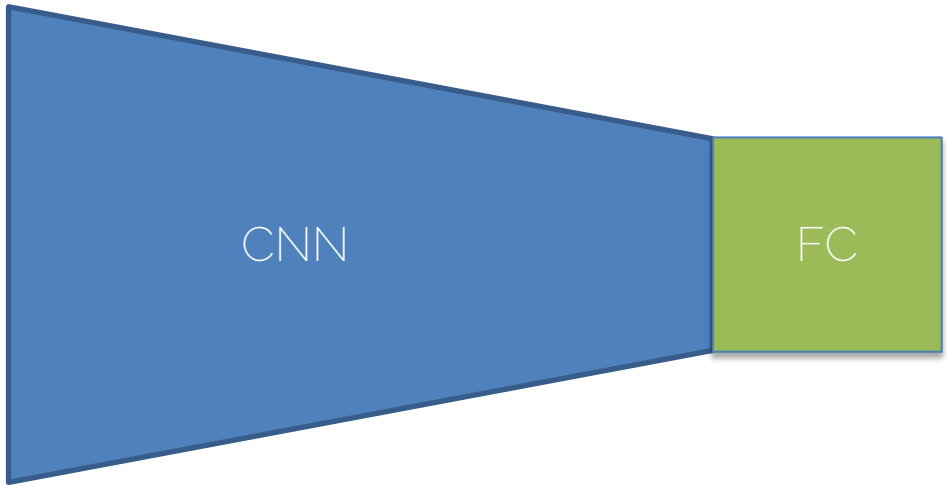
Coloring Legend:



Untrained

Trained

Transfer Learning



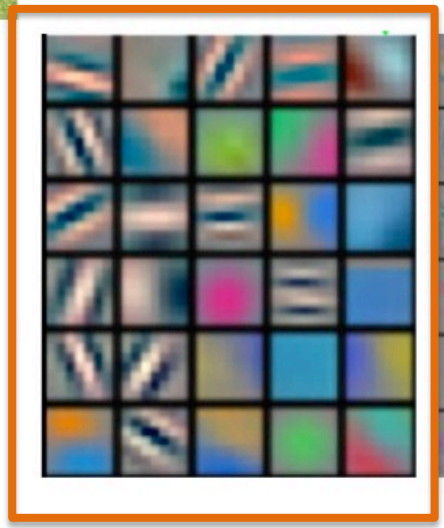
Coloring Legend:

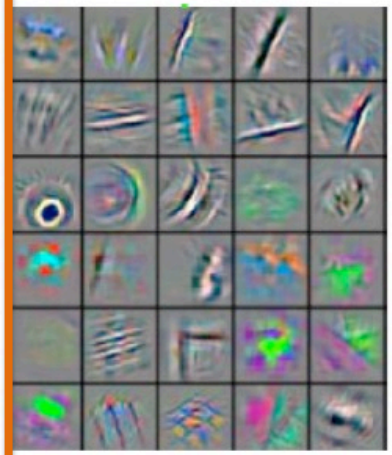
Untrained

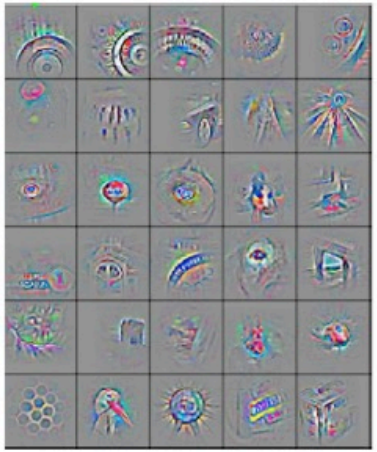
Trained

Why does it work?

Low-level feature → Middle-level feature → Top-level feature







Features such as edges, shapes, corners and intensity, can be shared across tasks

[Zeiler al., ECCV'14] Visualizing and Understanding Convolutional Networks

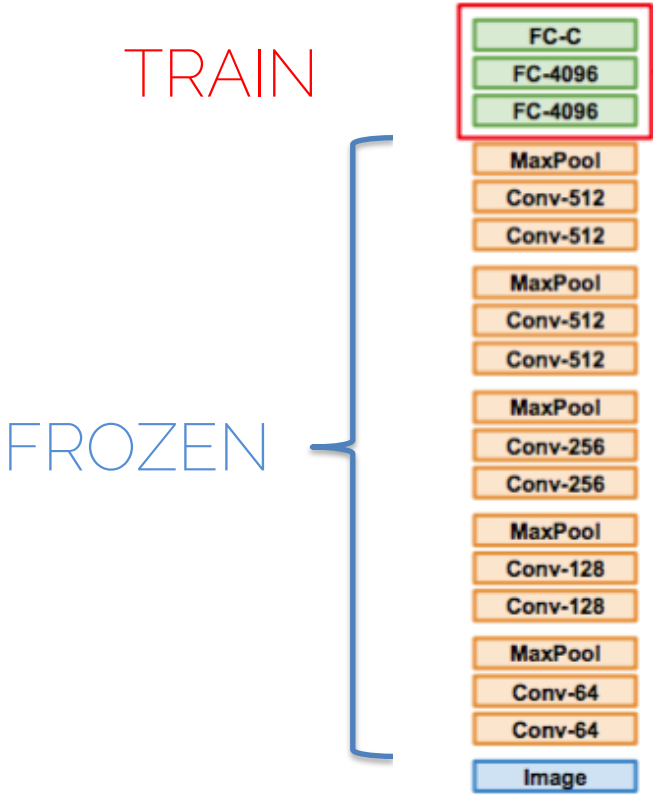
Transfer Learning with VGG

Comparison parameters (VGG):

- Last layer $\approx 10^4$ parameters
- Whole Network: $\approx 10^8$ parameters

Difference in magnitude of order 4

- Classifier can be trained with less data because of transferred weights
- If more data available, more layers can be finetuned with low learning rate



[Donahue et al., ICML'14] DeCAF,
[Razavian et al., CVPRW'14] CNN Features off-the-shelf

Transfer Learning for CIFAR10
(0_transfer_learning_optional.ipynb)

mobilenet_v2

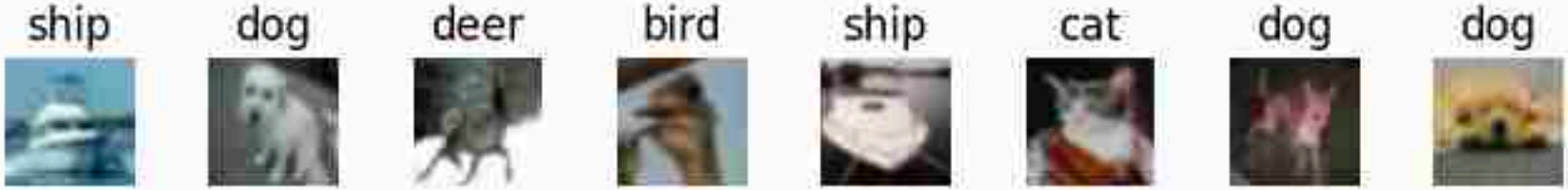
Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

[Sandler et al., CVPR'18] MobileNetV2

What to do?!

- Load pretrained mobilenet_v2 from torchvision
- Extract feature module and discard old classifier
- Add new classification head
- Freeze weights of CNN
- Train model

Best CIFAR10 Model



- https://benchmarks.ai/cifar-10
- https://paperswithcode.com/sota/image-classification-on-cifar-10
- https://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html#43494641522d3130

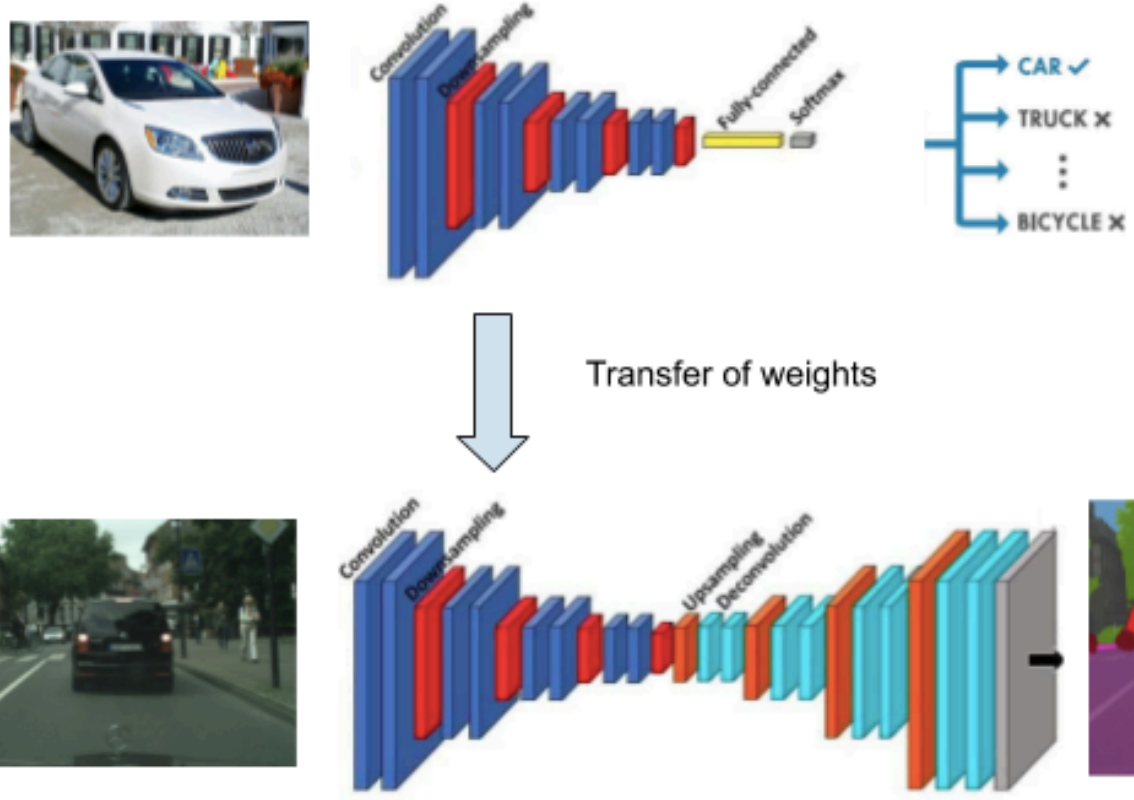
Semantic Segmentation

- Assigning a label to each pixel

Transfer Learning for Segmentation

- Can we use Transfer learning here?
- The CNN trained for image classification contains meaningful information that can be used for segmentation
- We can re-use the convolution layers of the pre-trained models in the encoder layers of the segmentation model
- Using Resnet or VGG pre-trained on ImageNet dataset is a popular choice

Transfer Learning for Segmentation



Leaderboard Submission #10

Leaderboard: Submission 10

Rank	User	Score	Pass
#1	s0597	78.89	✔
#2	s0499	78.70	✔
#3	s0697	77.31	✔
#4	s0568	77.30	✔
#5	s0555	77.24	✔
#6	s0770	77.07	✔
#7	s0699	76.44	✔
#8	s0782	76.40	✔
#9	s0334	75.83	✔
#10	s0288	75.73	✔

Top 3: 77.31

4 Encoder Blocks: Conv – BN – ReLU – MaxPool

```
self.down_03 = nn.Sequential(
    nn.Conv2d(depth*2, depth*4, (3, 3), padding=1),  # shape afterwards: Nxdepth*4*60x60
    nn.BatchNorm2d(depth * 4),
    nn.ReLU(),
    nn.Dropout2d(),
    nn.MaxPool2d((2, 2), 2)  # shape afterwards: Nxdepth*4x30x30
)
```

4 Decoder Blocks : Conv – BN - ReLU – Upsampling

```
self.up_03 = nn.Sequential(
    nn.Conv2d(2*depth*2, depth, (3, 3), padding=1),  # shape afterwards: Nxdepthx60x60
    nn.BatchNorm2d(depth),
    nn.ReLU(),
    # nn.Dropout2d(),
    nn.Upsample(scale_factor=2, mode='bilinear')  # shape afterwards: Nxdepthx120x120
)
```

Rank 2: 78.70

4 Encoder Blocks: Conv – BN – ReLU – Conv – BN – ReLU – MaxPool

```
nn.Conv2d(self.channels*2, self.channels*4, kernel_size=3, stride=1, padding=1),
nn.BatchNorm2d(self.channels*4),
nn.ReLU(),
nn.Conv2d(self.channels*4, self.channels*4, kernel_size=3, stride=1, padding=1),
nn.BatchNorm2d(self.channels*4),
nn.ReLU(),
nn.MaxPool2d(3, 2),
```

4 Decoder Blocks : TransposeConv – Conv – BN – ReLU – Conv – BN – ReLU

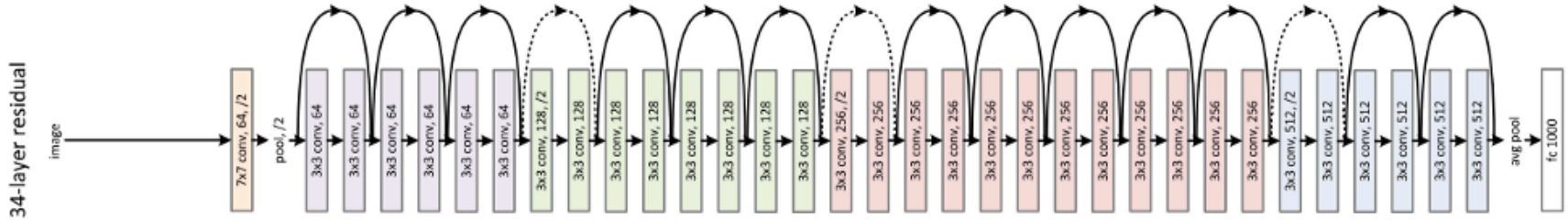
```
nn.ConvTranspose2d(self.channels*8, self.channels*4, 3, 2),
nn.Conv2d(self.channels*4, self.channels*4, kernel_size=3, stride=1, padding=1),
nn.BatchNorm2d(self.channels*4),
nn.ReLU(),
nn.Conv2d(self.channels*4, self.channels*4, kernel_size=3, stride=1, padding=1),
nn.BatchNorm2d(self.channels*4),
nn.ReLU(),
```

Rank 1: 78.89

```
""""""" MODEL """""

self.backbone = ResNetBackBone(input_channels, resnet_channels, n_blocks, block_module)
self.classifier = DeepLabHead(resnet_output_channels, aspp_channels, num_classes, p_dropout)
```

Encoder: ResNet (CNN with skip connections)

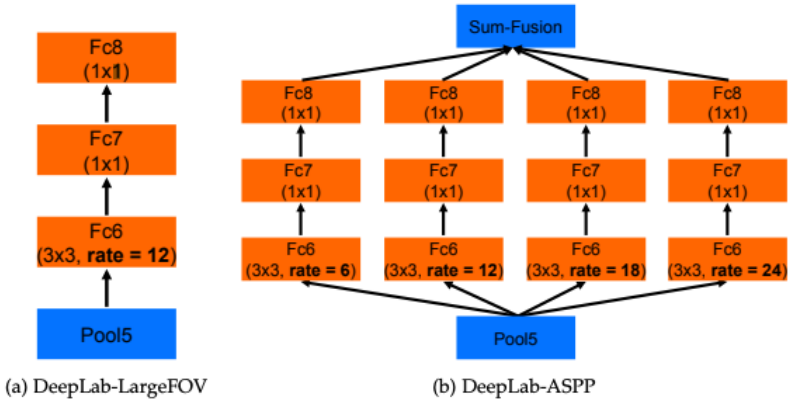


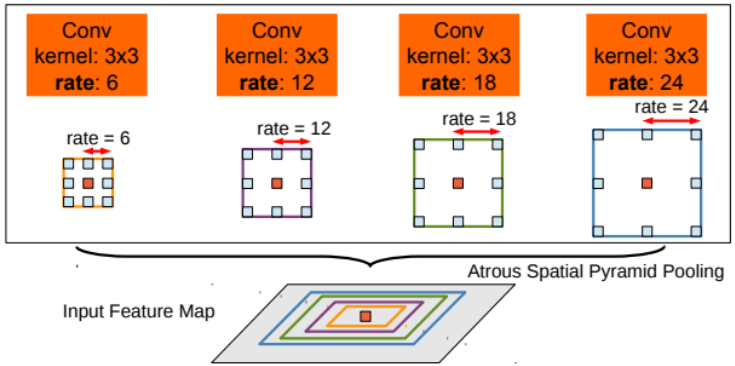
Rank 1: 78.89

```
""""""" MODEL """""

self.backbone = ResNetBackBone(input_channels, resnet_channels, n_blocks, block_module)
self.classifier = DeepLabHead(resnet_output_channels, aspp_channels, num_classes, p_dropout)
```

Decoder: Modified DeepLabV3



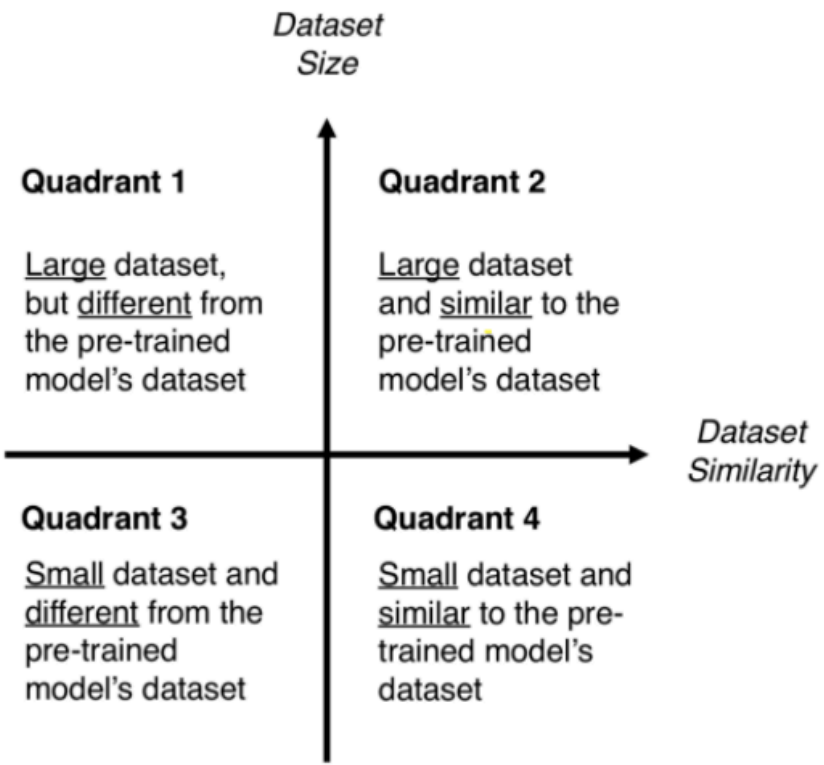


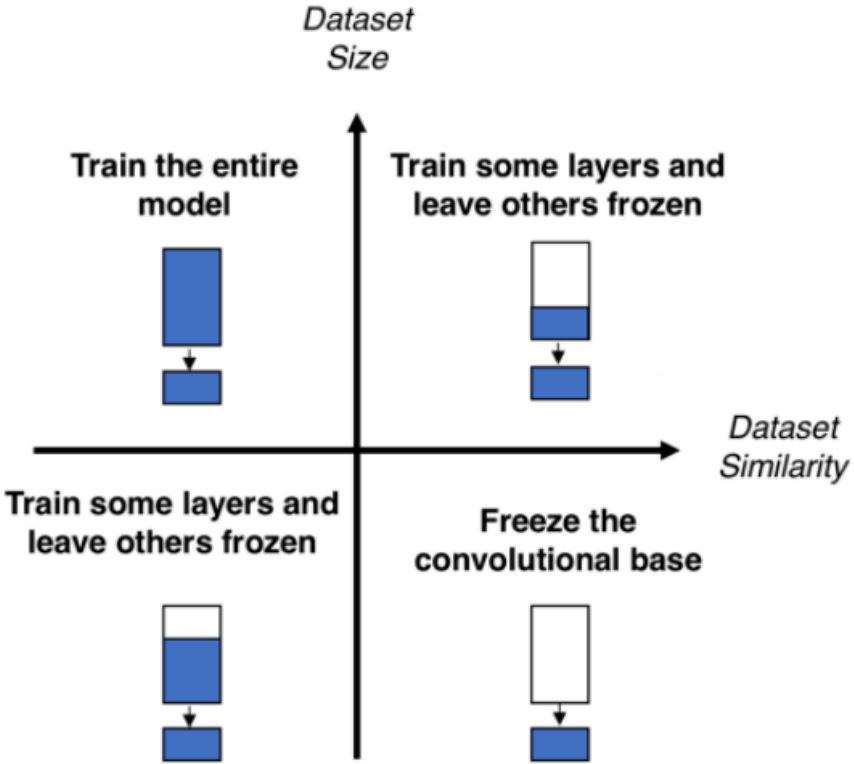
[Liang-Chieh et al., ECCV'18] Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation

Summary Transfer Learning

- Idea:
  - Take network trained on a similar task
  - Add/Remove layers to make it suitable for your task
  - Train new parts while pre-trained parts are frozen
- Advantage:
  - Training of new task requires **less data** and **time**

Summary





https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751

Recurrent Neural Networks

Recurrent Neural Networks

- Vanilla Feed-forward net:

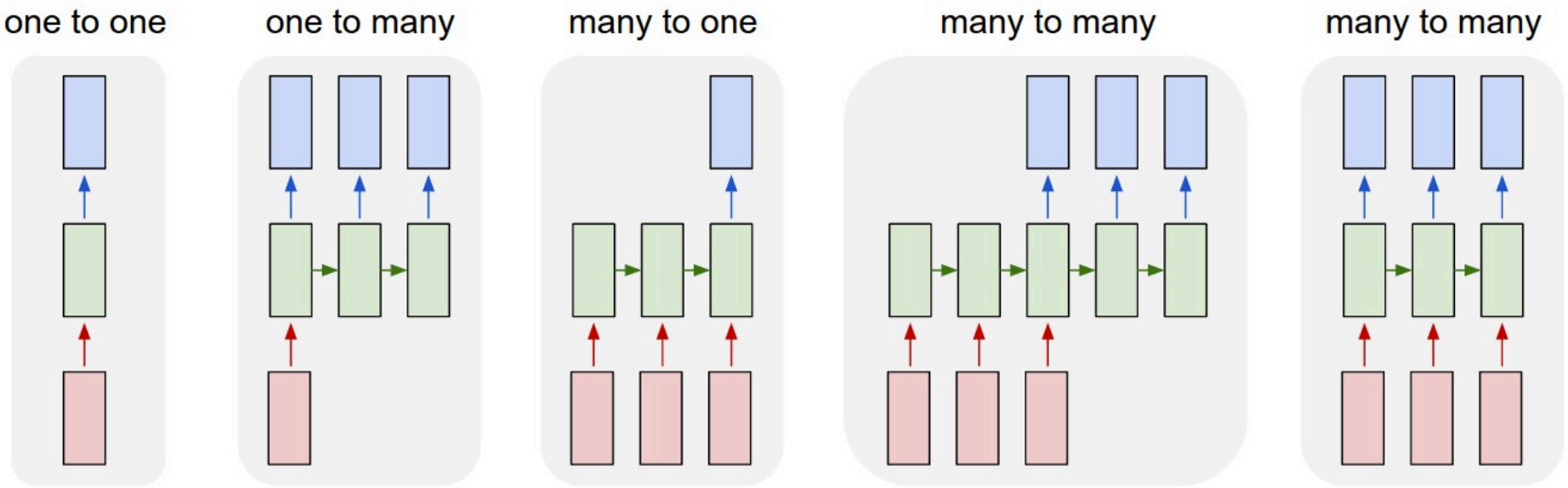
$$y = f(x)$$

- Recurrent neural networks can process sequential data
  - Speech recognition
  - Text to Music generation
  - Music Sentiment classification
  - Ratings DNA sequence analysis
  - Text Translation
  - Video activity recognition

- Recurrent network:

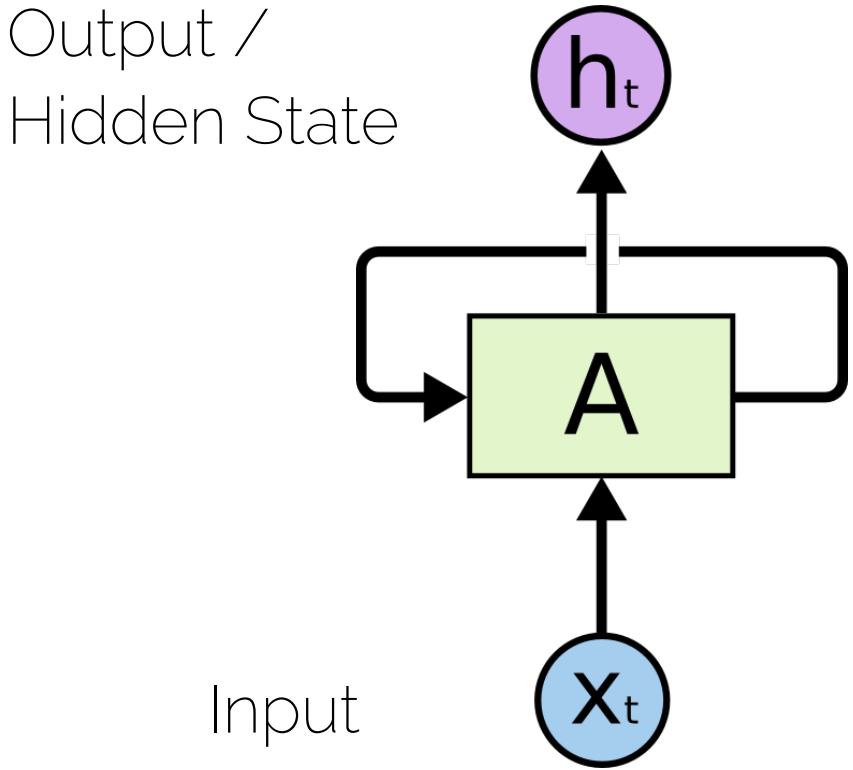
$$y_t = f(x_t, y_{t-1})$$

RNN Concepts



Source: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

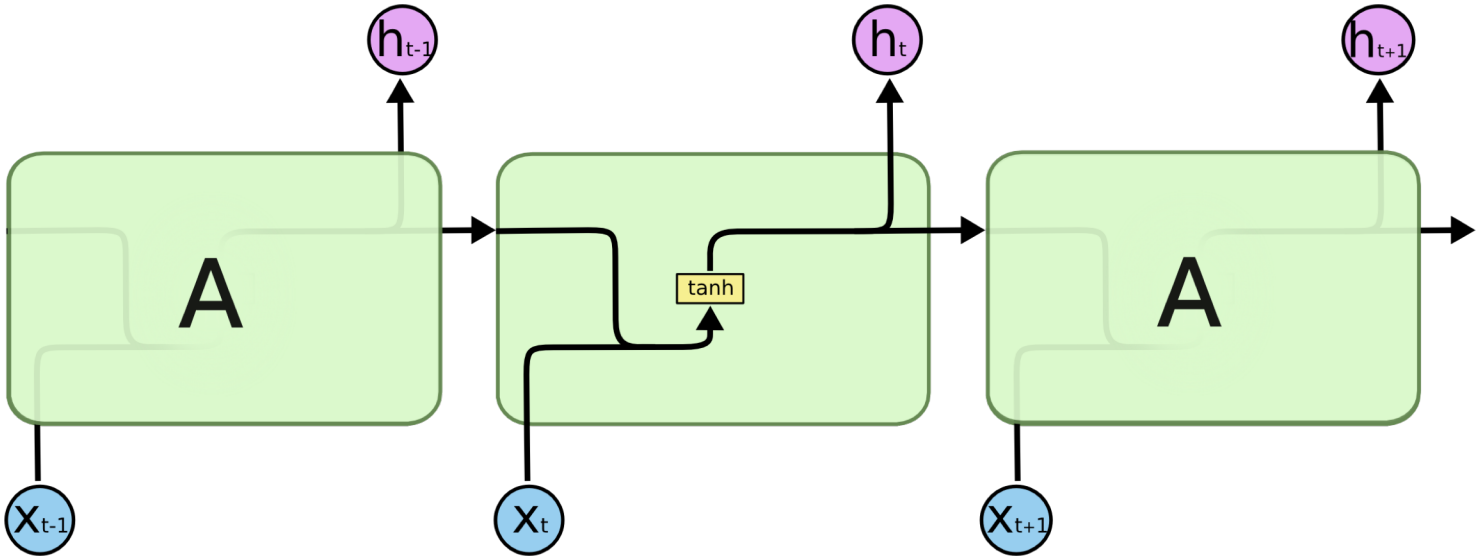
Basic Structure of an RNN



[Olah, https://colah.github.io '15] Understanding LSTMs

Simple RNN

$$h_t = \tanh\left(W \cdot h_{t-1} + V \cdot x_t + b\right)$$



Simple RNN

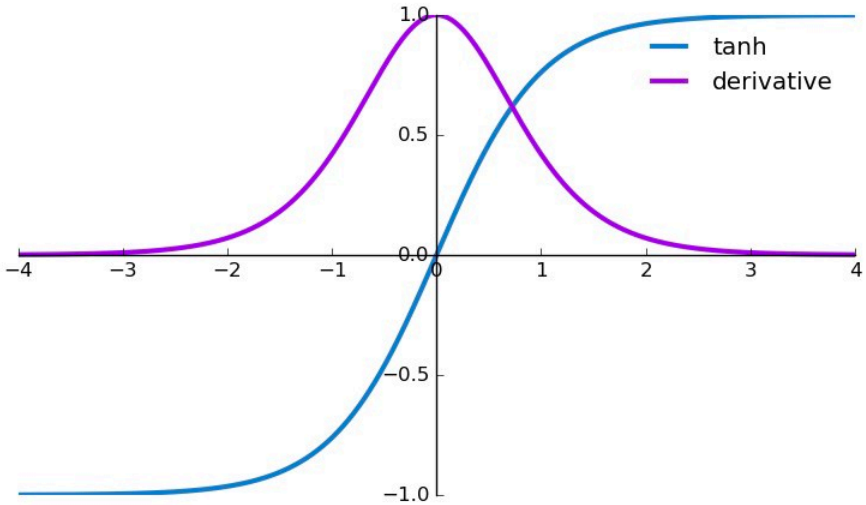
- Computing Gradients

$$h_t = \tanh(W \cdot h_{t-1} + V \cdot x_t + b)$$

$$\sigma(\cdot) = \tanh(\cdot)$$

$$z_t = W \cdot h_{t-1} + V \cdot x_t$$

$$\frac{\partial h_t}{\partial x_\tau} = \frac{\partial \sigma(z_t)}{\partial z_t} \cdot (\mathrm{W} \cdot \frac{\partial h_t}{\partial x_\tau} + \mathrm{V} \cdot \delta_{t\tau})$$



-> Gradient depends on the norm of W and derivative of $\tanh(\cdot)$

Simple RNN

- Computing Gradients

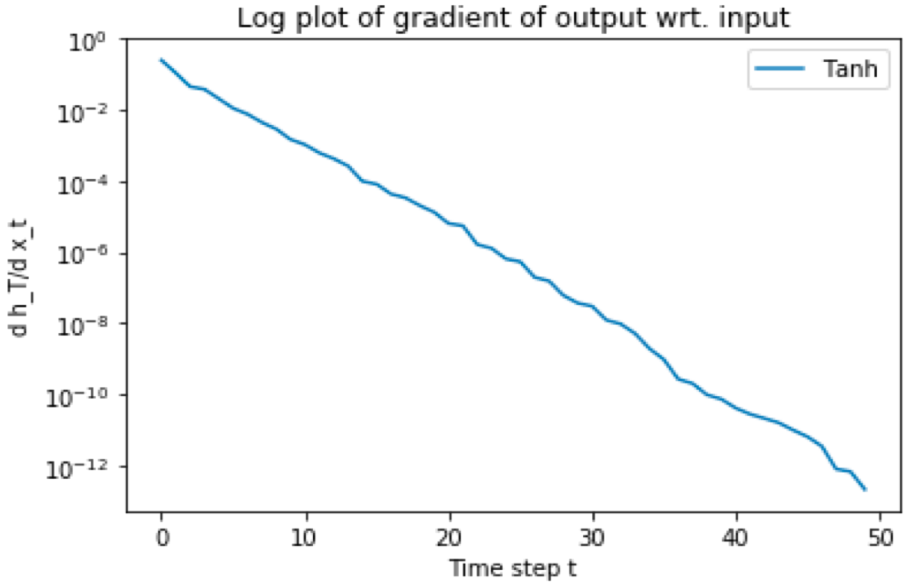
$$h_t = \tanh(W \cdot h_{t-1} + V \cdot x_t + b)$$

$$\sigma(\cdot) = \tanh(\cdot)$$

$$z_t = W \cdot h_{t-1} + V \cdot x_t$$

$$\frac{\partial h_t}{\partial x_\tau} = \frac{\partial \sigma(z_t)}{\partial z_t} \cdot (\mathrm{W} \cdot \frac{\partial h_t}{\partial x_\tau} + \mathrm{V} \cdot \delta_{t\tau})$$

- Vanishing gradient



-> Gradient depends on the norm of W and derivative of $\tanh(\cdot)$

Simple RNN (homework)

$$h_t = \tanh\ (W \cdot h_{t-1} + V \cdot \mathrm{x_t} + b) \qquad z_t = W \cdot h_{t-1} + V \cdot x_t$$

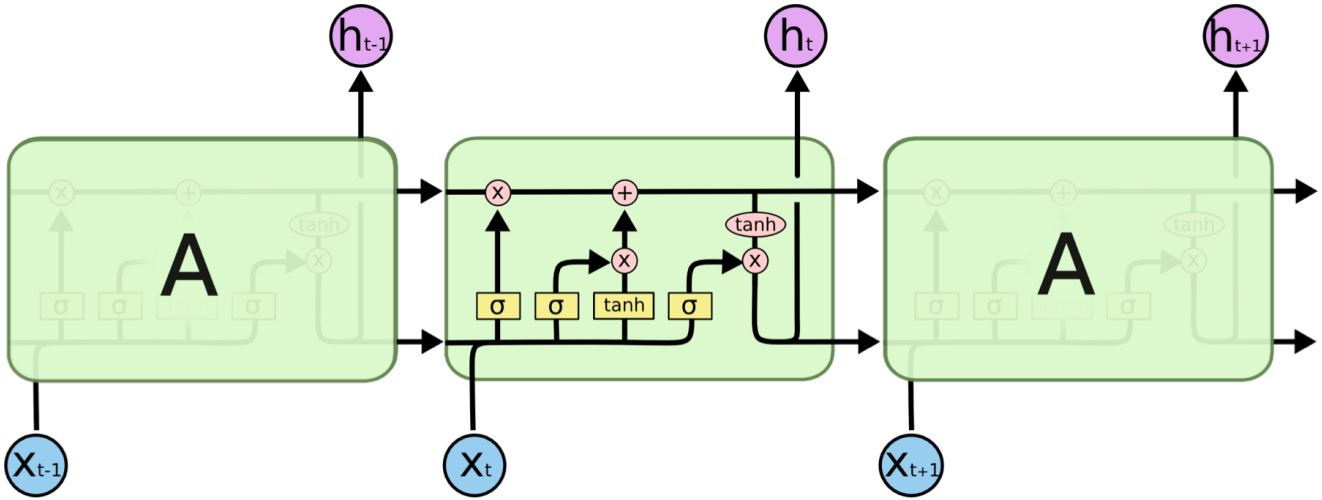
$$\sigma(\cdot) = \tanh(\cdot) \qquad h_0 = 0$$

- Calculate $\frac{\partial h_3}{\partial x_1}$
- Solution

$$\frac{\partial h_3}{\partial x_1} = \frac{\partial \sigma(z_3)}{\partial z_3} (\mathrm{W} \cdot \frac{\partial \sigma(z_2)}{\partial z_2} \left(\mathrm{W}. \frac{\partial \sigma(z_1)}{\partial z_1} \cdot \mathrm{V} \right))$$

LSTM

- In practice, RNNs don't seem to be able to learn long term dependencies
- Long Short Term Memory networks [Hochreiter & Schmidhuber (1997)] solve long-term dependencies with gated architecture



LSTM

- Forget Gate:

$$f_t = \sigma_g(W_f \,.\, h_{t-1} + V_f . x_t + b_f)$$

- Input Gate:

$$i_t = \sigma_g(W_i . h_{t-1} + V_i . x_t + b_i)$$

- Output Gate:

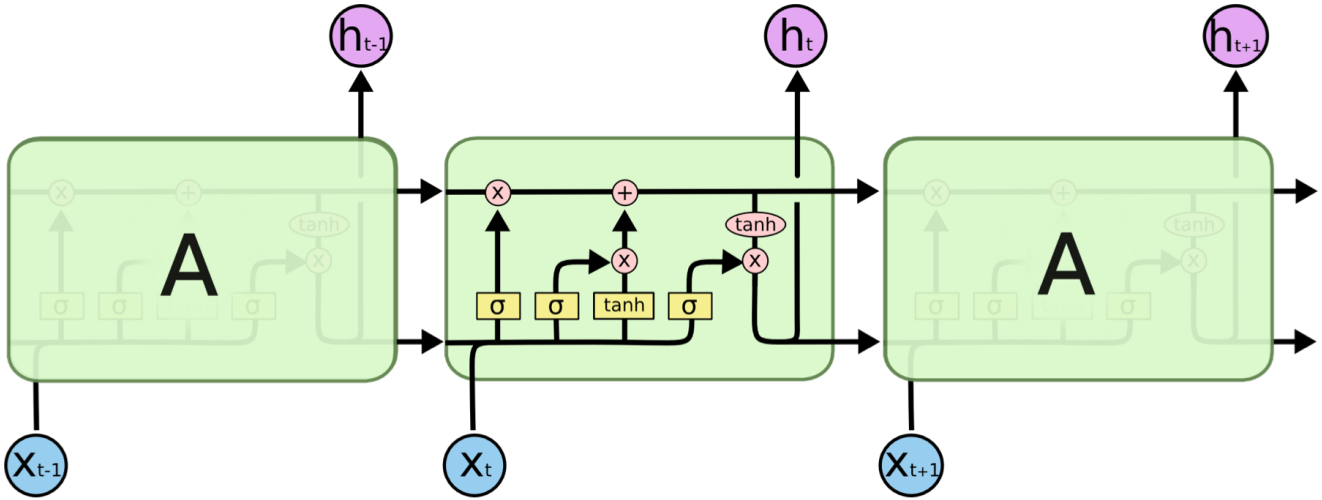
$$o_t = \sigma_g(W_o . h_{t-1} + V_o . x_t + b_f)$$

- Cell State Update:

$$c_t = f_t \cdot c_{t-1} + i_t \; tanh(W_c . h_{t-1} + V_c . x_t + b_c)$$

- Hidden State Update:

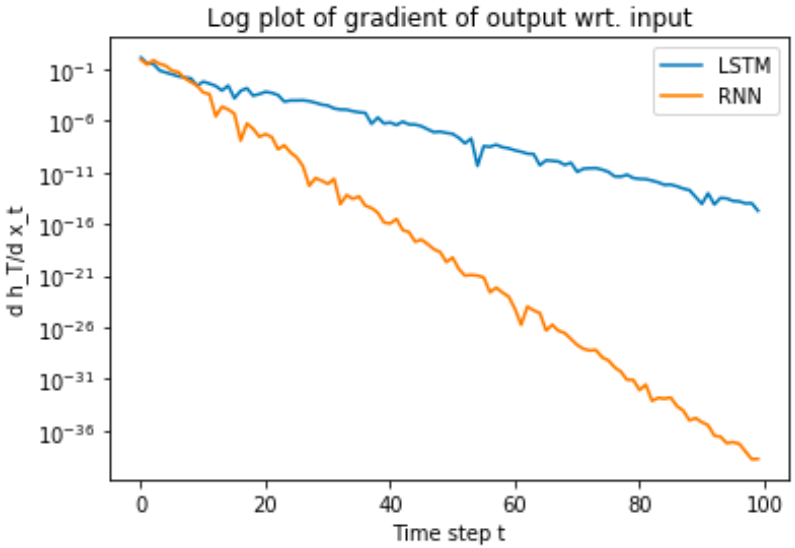
$$h_t = o_t \; tanh(c_t)$$



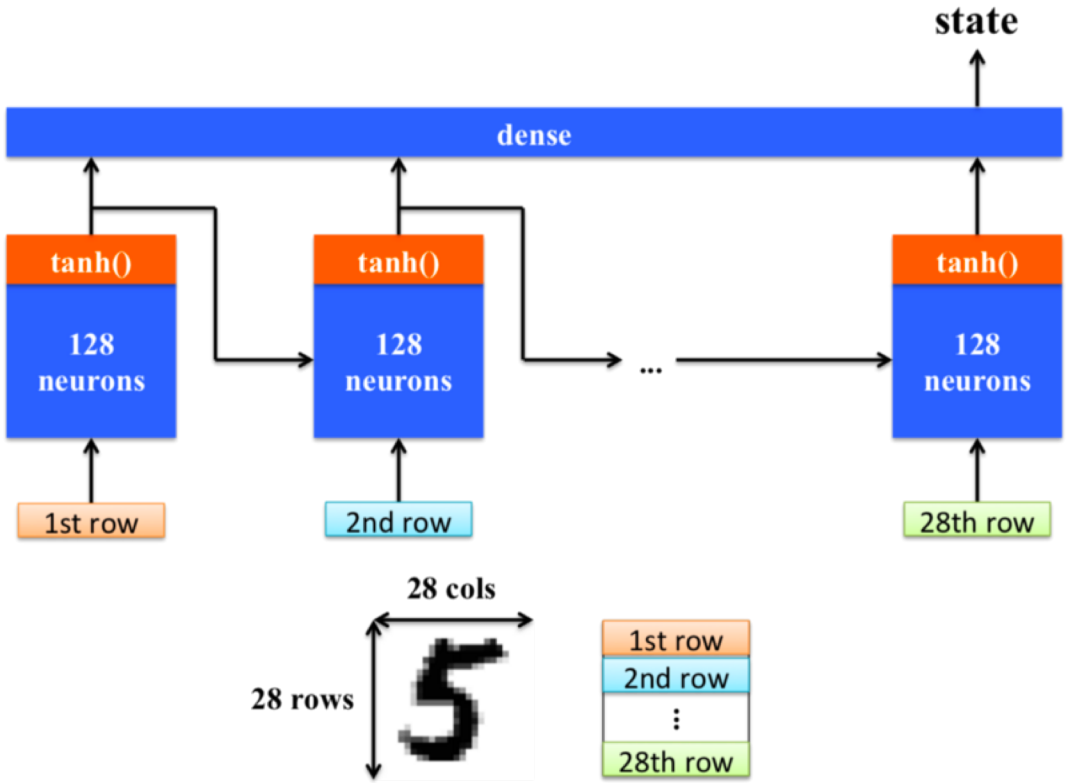
LSTM

- LSTM "resolves" vanishing gradient problem

  - Plot shows log of gradient vs time steps
  - Gradient of RNN decreases much faster than the gradient of the LSTM

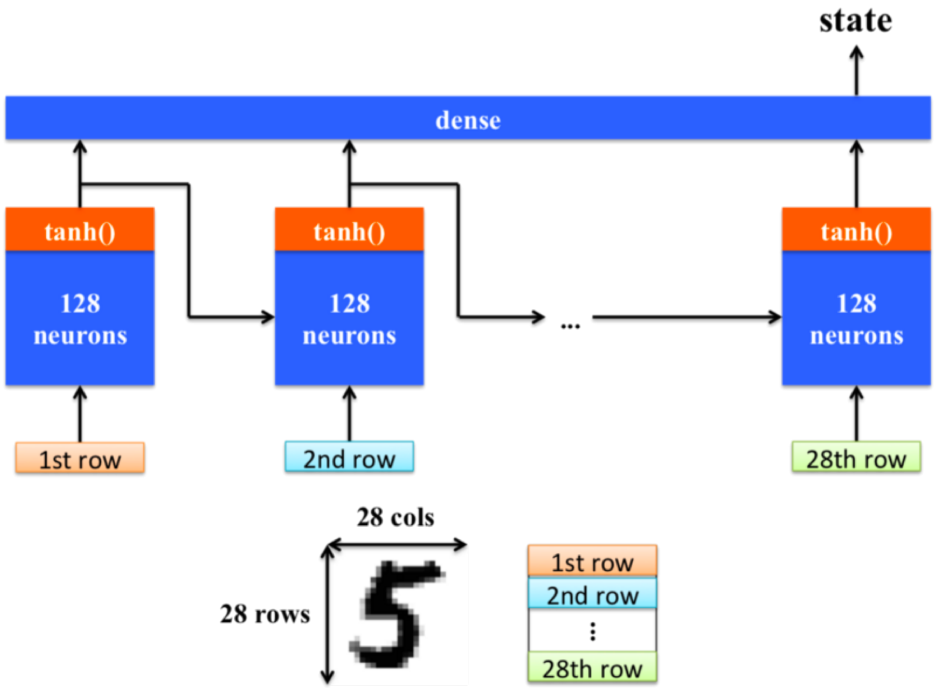


MNIST RNN Classifier



MNIST RNN Classifier

- Each image is 28 pixels (rows) by 28 pixels (cols)
- We treat each image as a sequence of data
- The first row is the first step, second row is the second step and …
- number of steps = number of rows
  number of inputs = number of columns



Submission Goals

- Implement a simple RNN class in Pytorch
- Explore the backpropagation of the gradients in the RNN and discuss the vanishing gradient problem
- Implement a LSTM Network and show that this architecture improves the vanishing gradient problem
- Create a Pytorch Lightning model and train your it for MNIST classification
- Tune hyperparameters of your model and submit your best model to the server to get bonus points!

Submission

- Submission **Start: July 10**, 2020 12.00
- Submission **Deadline : July 16**, 2020 23.59
- Submit your trained Pytorch Model!
- Your model's **accuracy** is all that counts!
  - At least **90%** to pass the submission

See you next week ☺