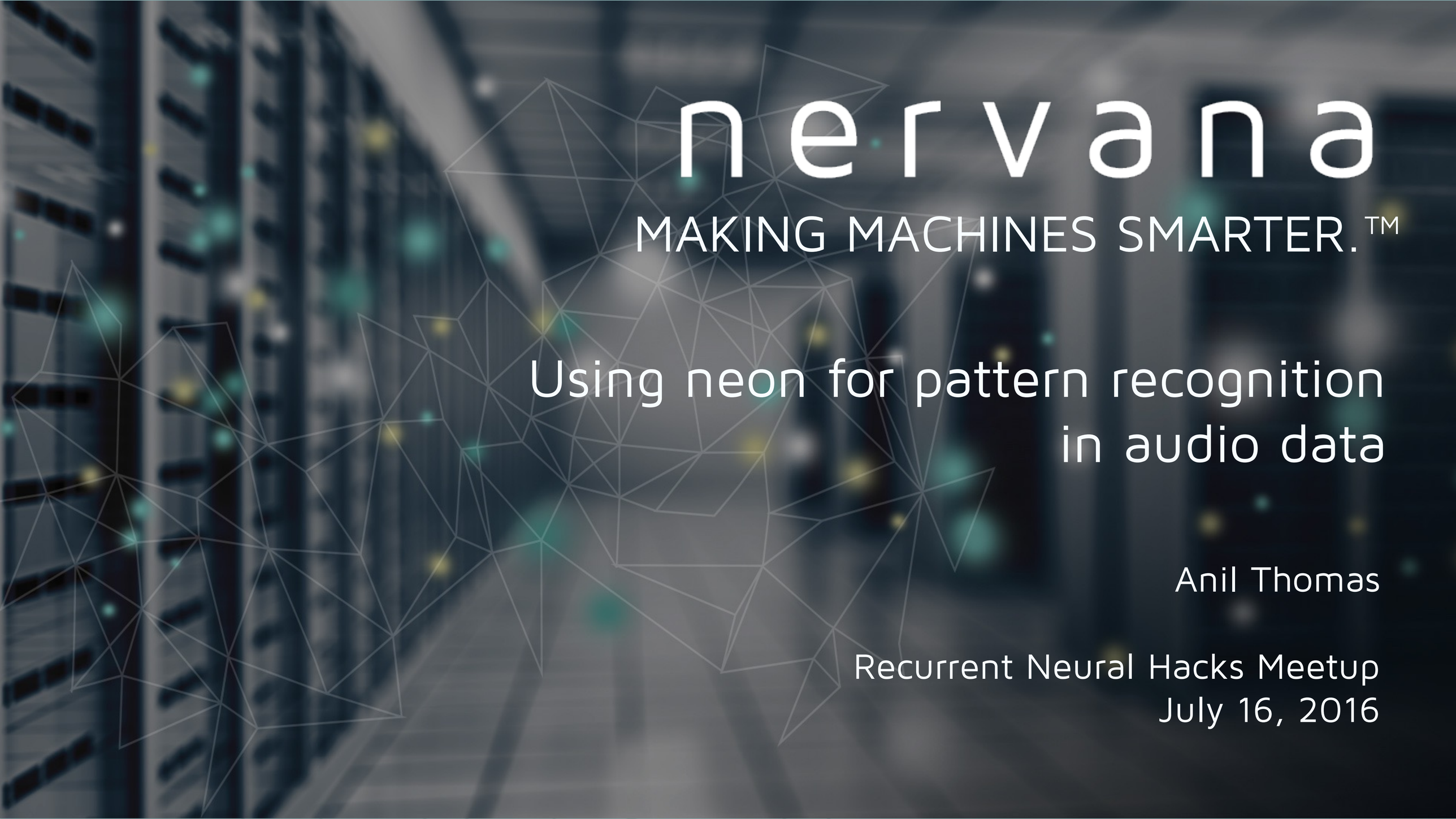


Using neon for pattern recognition in audio data

Anil Thomas

Recurrent Neural Hacks Meetup
July 16, 2016

Outline

- Intro to neon
- Workshop environment setup
- CNN theory
- CNN hands-on
- RNN theory
- RNN hands-on

NEON

Neon

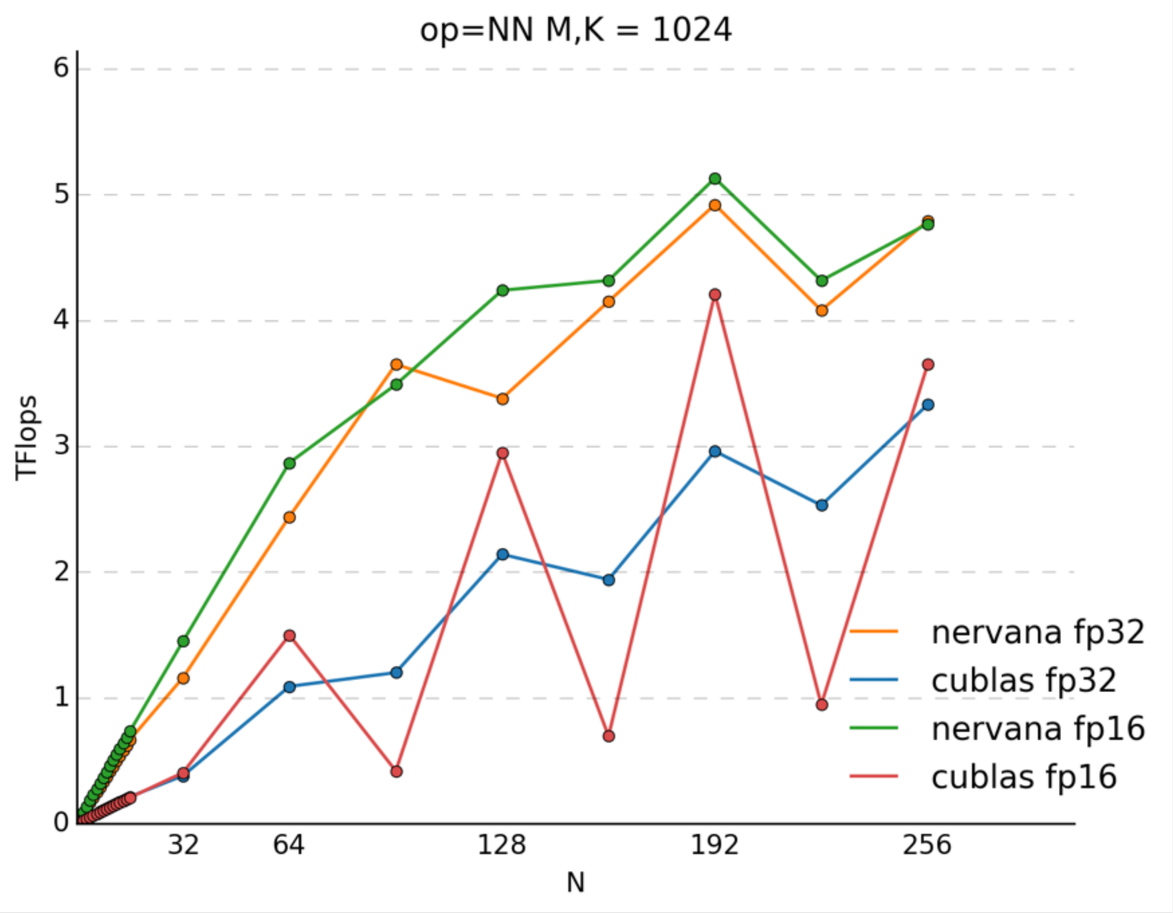
- Modular components
- Extensible, OO design
- Documentation
- neon.nervanasys.com

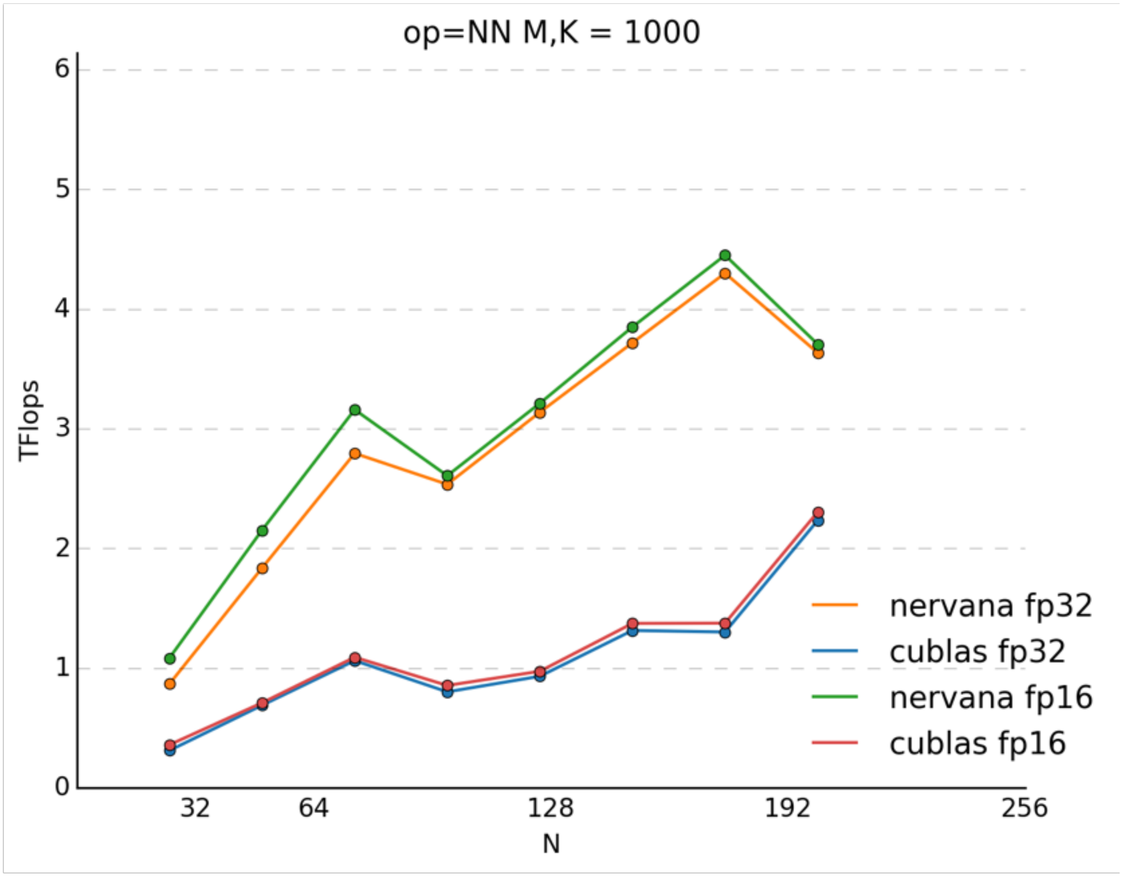
Backends	NervanaCPU, NervanaGPU NervanaMGPU, NervanaEngine (internal)
Datasets	Images: ImageNet, CIFAR-10, MNIST Captions: flickr8k, flickr30k, COCO; Text: Penn Treebank, hutter-prize, IMDB, Amazon
Initializers	Constant, Uniform, Gaussian, Glorot Uniform
Learning rules	Gradient Descent with Momentum RMSProp, AdaDelta, Adam, Adagrad
Activations	Rectified Linear, Softmax, Tanh, Logistic
Layers	Linear, Convolution, Pooling, Deconvolution, Dropout Recurrent, Long Short-Term Memory, Gated Recurrent Unit, Recurrent Sum, LookupTable
Costs	Binary Cross Entropy, Multiclass Cross Entropy, Sum of Squares Error
Metrics	Misclassification, TopKMisclassification, Accuracy

Why neon?

- Fastest according to third-party benchmarks
- Advanced data-loading capabilities
- Open source (including the optimized GPU kernels!)
- Support for distributed computing

Benchmarks for RNNs[1]





GEMM benchmarks compiled by Baidu. Bigger is better.

[1] Erich Elsen, http://svail.github.io/

nervana

Data loading in neon

- Multithreaded
- Non-blocking IO
- Non-blocking decompression and augmentation
- Supports different types of data
  (We will focus on audio in this workshop)
- Automatic ingest
- Can handle huge datasets

WORKSHOP ENV SETUP

Github repo with source code

- https://github.com/anlthms/meetup2
- Music classification examples
- To clone locally:

  git clone https://github.com/anlthms/meetup2.git

Configuring EC2 instance

- Use your own EC2 account at http://aws.amazon.com/ec2/
- Select US West (N. California) zone
- Search for nervana-neon10 within Community AMIs
- Select g2.2xlarge as instance type
- After launching the instance, log in and activate virtual env:

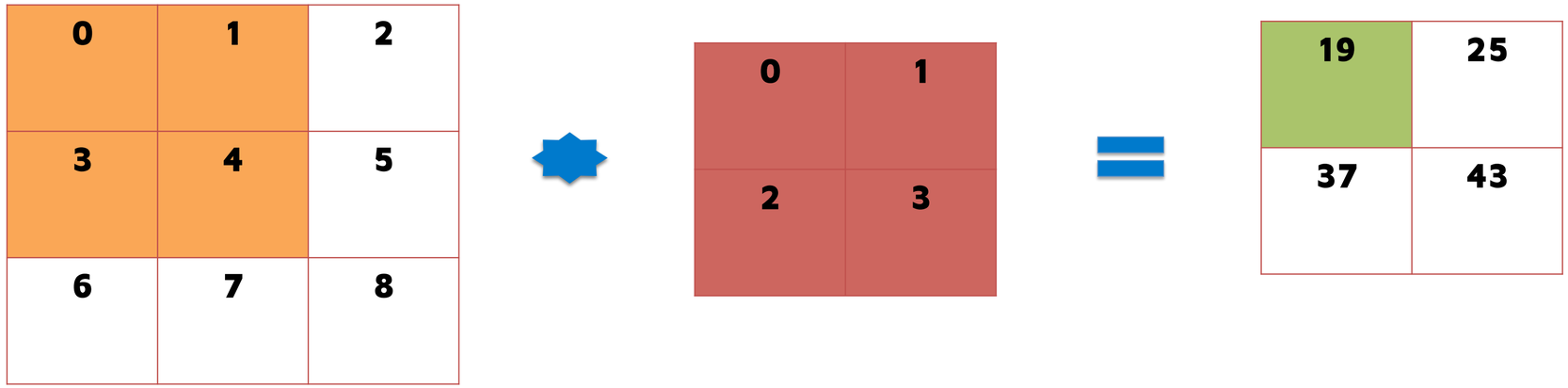
    source ~/neon/.venv/bin/activate

Datasets

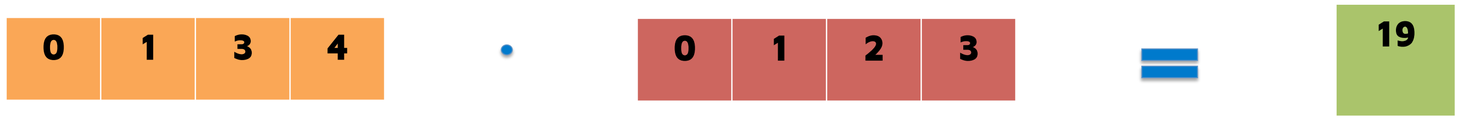
- GTZAN music genre dataset
  - 10 genres
  - 100 clips in each genre
  - Each clip is 30 seconds long
  - Preloaded in the AMI at /home/ubuntu/nervana/music/
- Whale calls dataset from Kaggle
  - https://www.kaggle.com/c/whale-detection-challenge/data
  - 30,000 2-second sound clips
  - Preloaded in the AMI at /home/ubuntu/nervana/wdc/

CNN THEORY

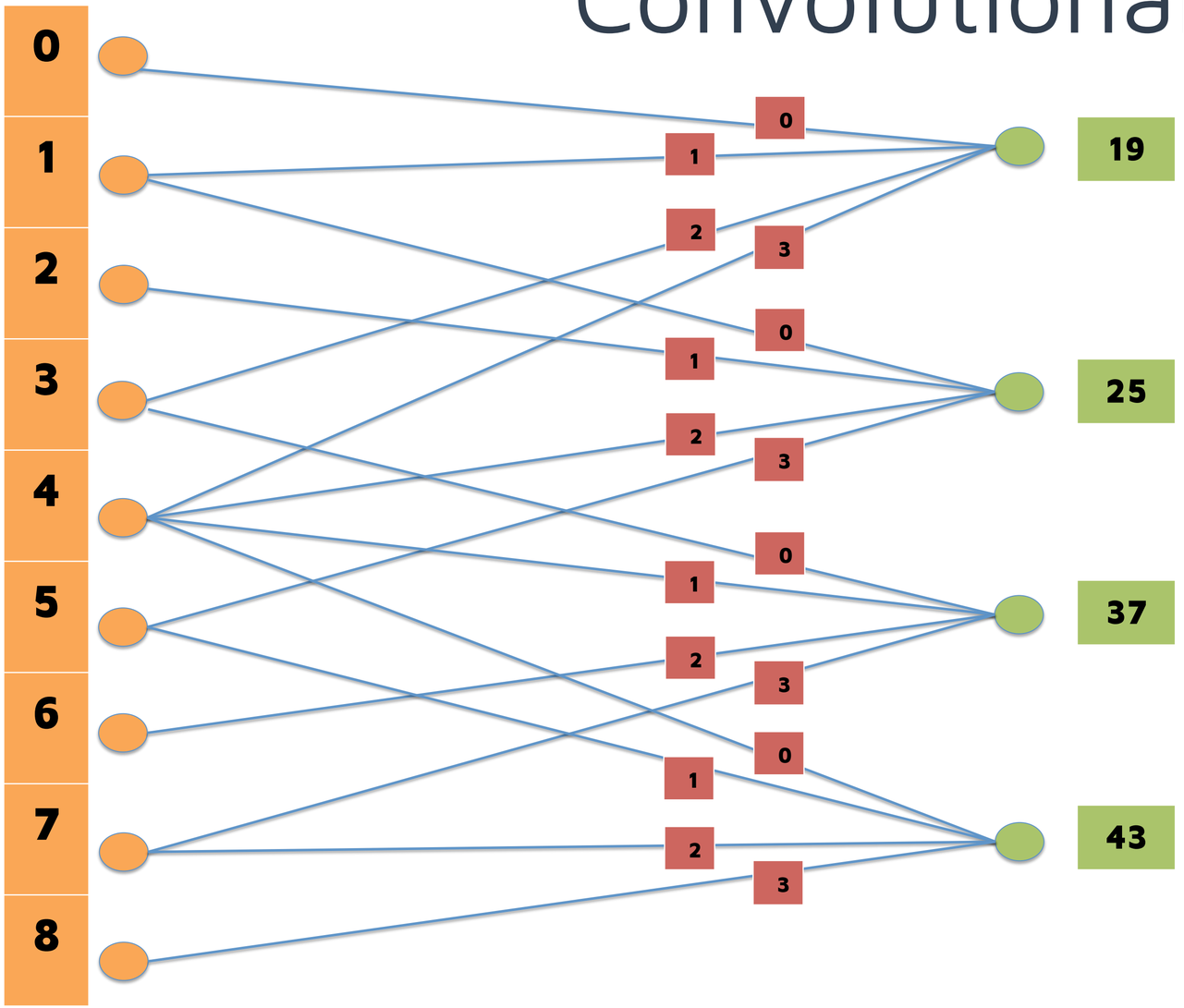
Convolution



- Each element in the output is the result of a dot product between two vectors



Convolutional layer



0	1	2
3	4	5
6	7	8

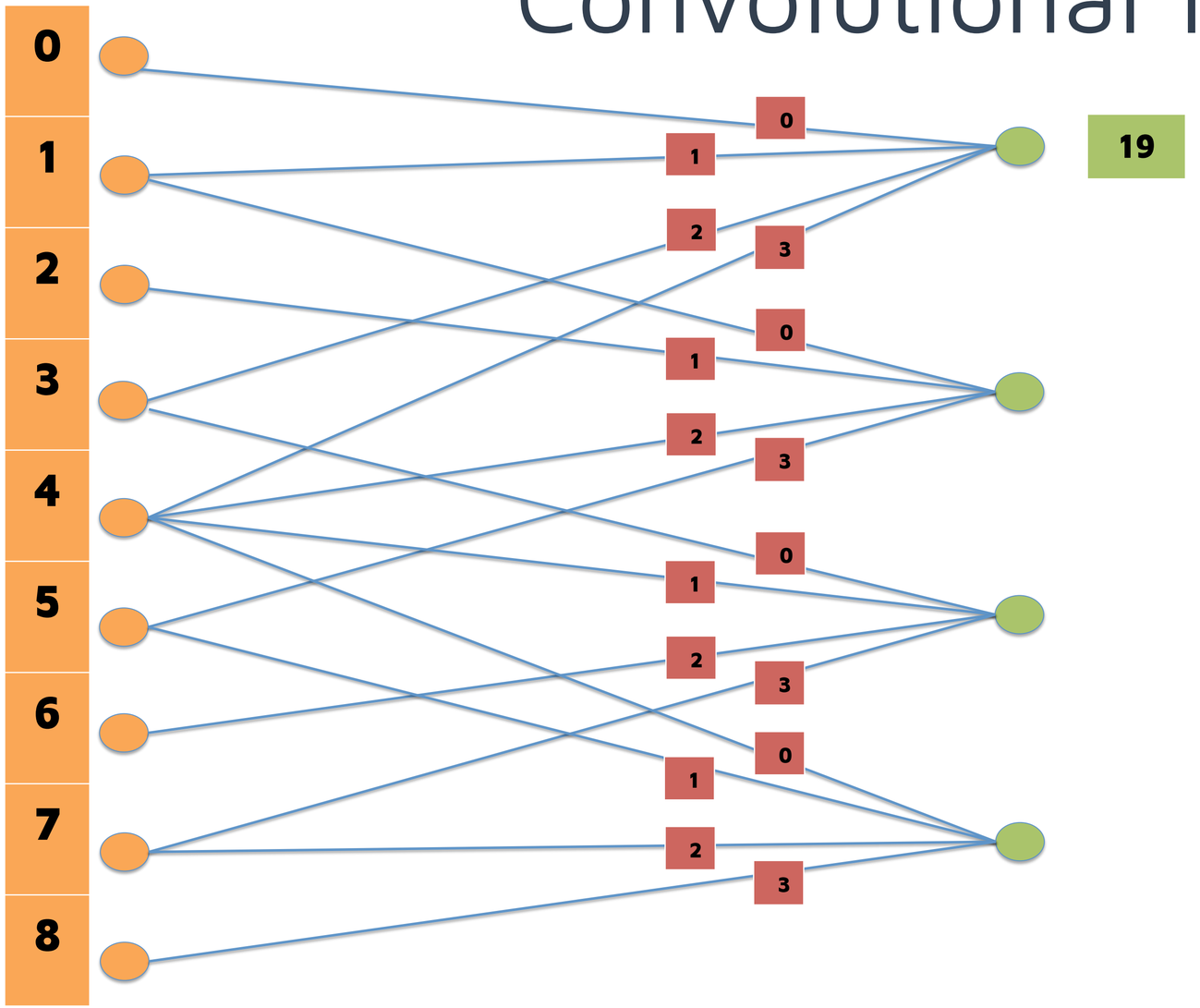
0	1
2	3

19	25
37	43

nervana

14

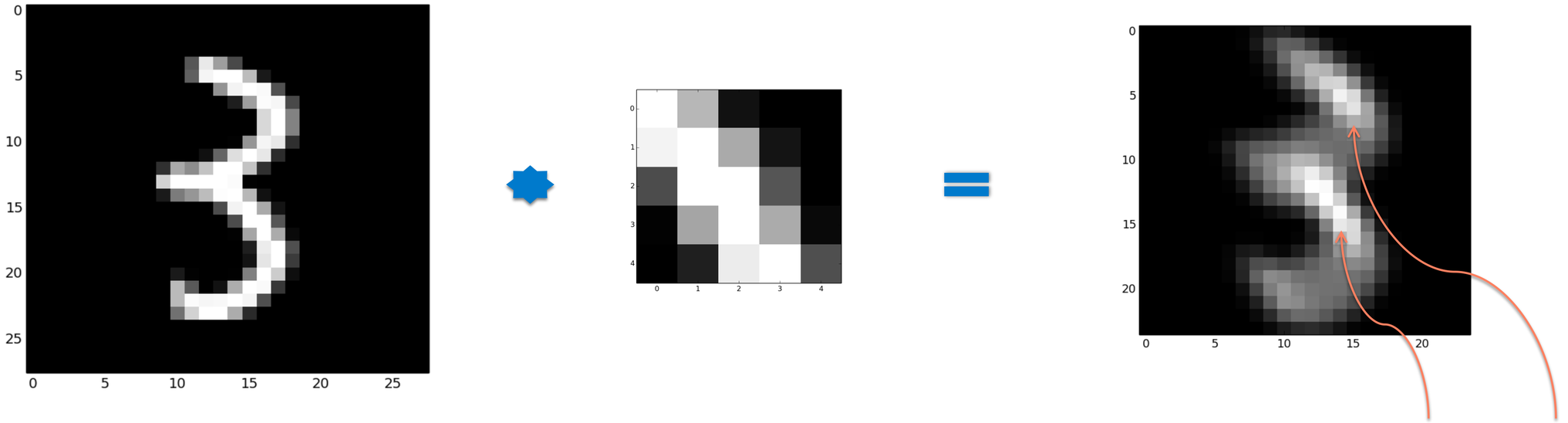
Convolutional layer



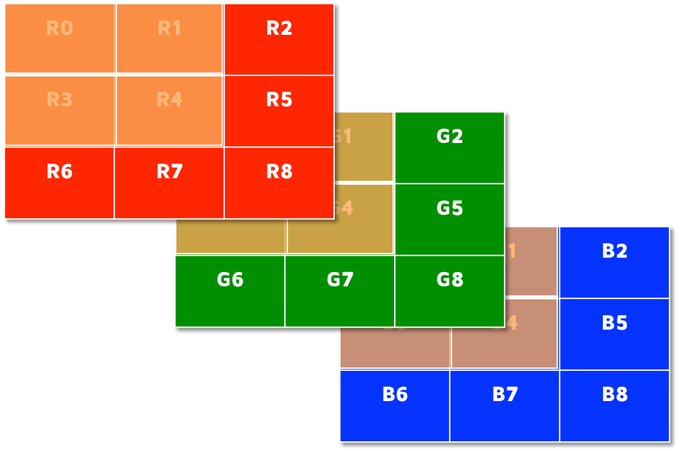
0 x 0 +
1 x 1 +
2 x 3 +
3 x 4 = 19

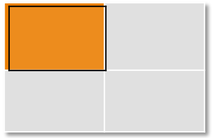
The weights are shared among the units.

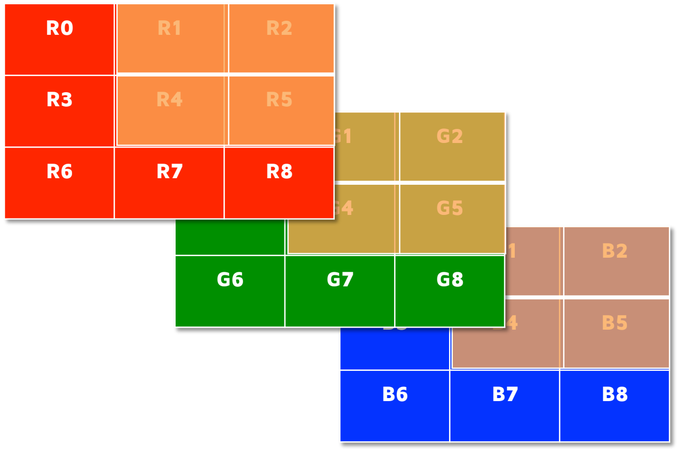
Recognizing patterns

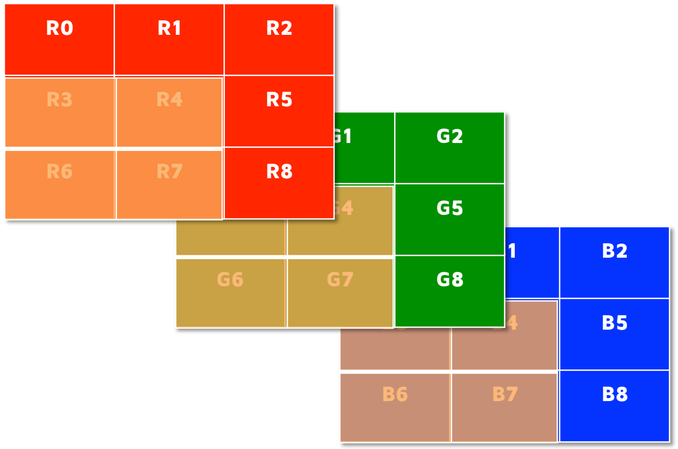


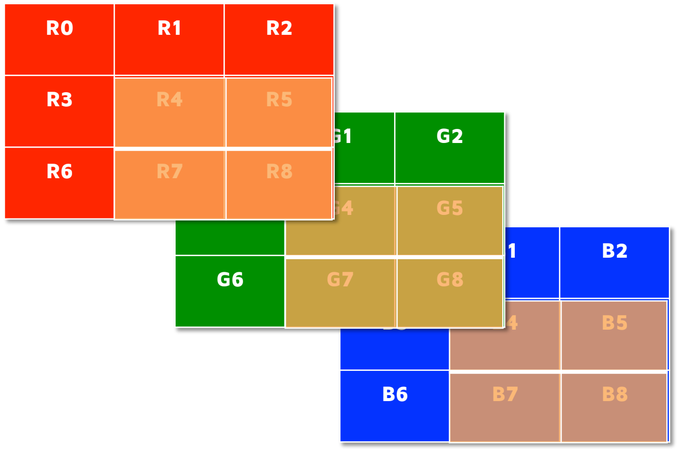
Detected the pattern!

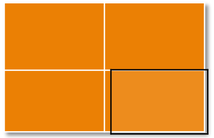

R0
R1
R2
R3
R4
R5
R6
R7
R8
G2
G5
G6
G7
G8
B2
B5
B6
B7
B8



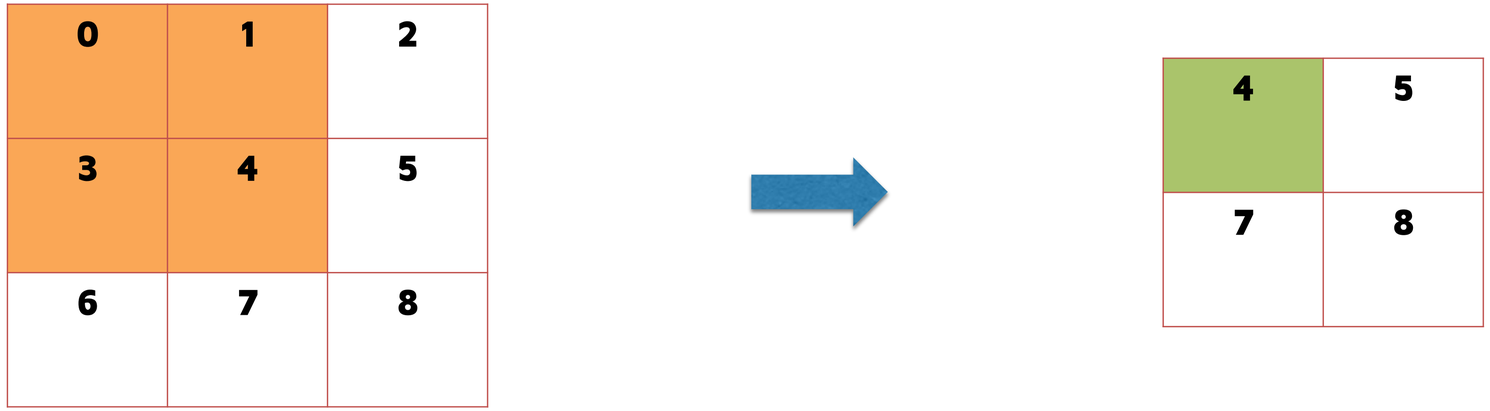

R0
R1
R2
R3
R4
R5
R6
R7
R8
G2
G5
G6
G7
G8
B2
B5
B6
B7
B8


R0
R1
R2
R3
R4
R5
R6
R7
R8
G2
G5
G6
G7
G8
B2
B5
B6
B7
B8


R0
R1
R2
R3
R4
R5
R6
R7
R8
G1
G2
G4
G5
G6
G7
G8
1
B2
4
B5
B6
B7
B8



Max pooling



- Each element in the output is the maximum value within the pooling window

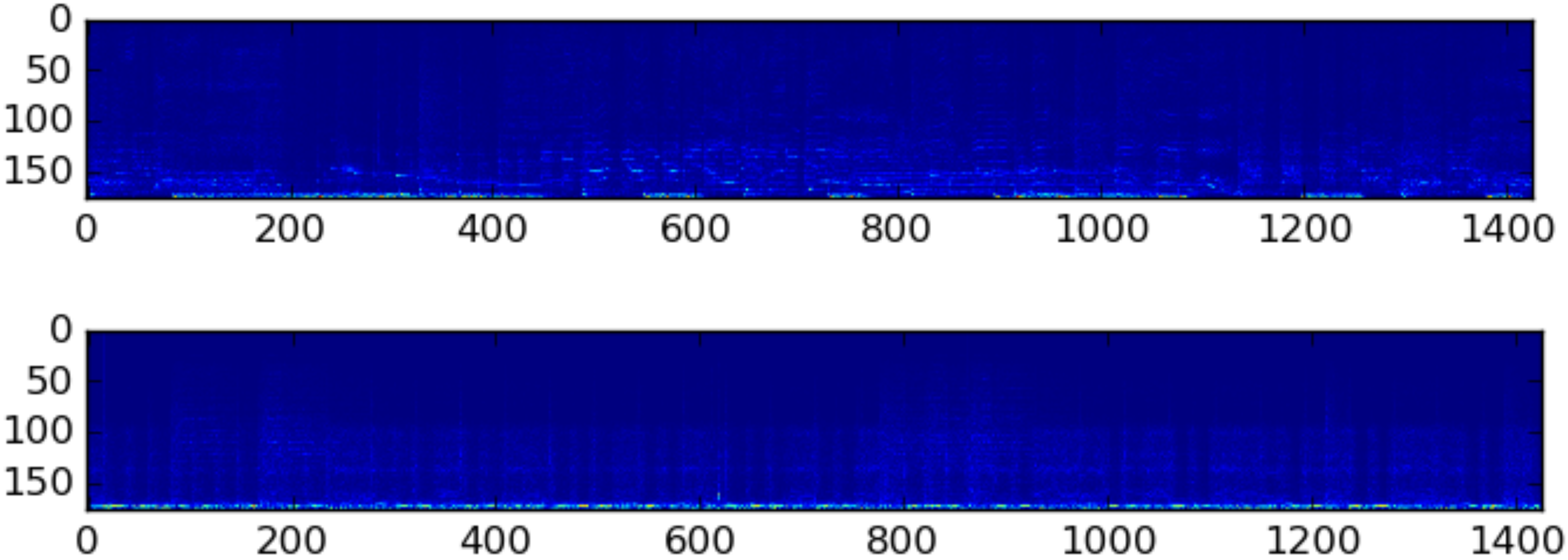
CNN HANDS-ON

CNN example

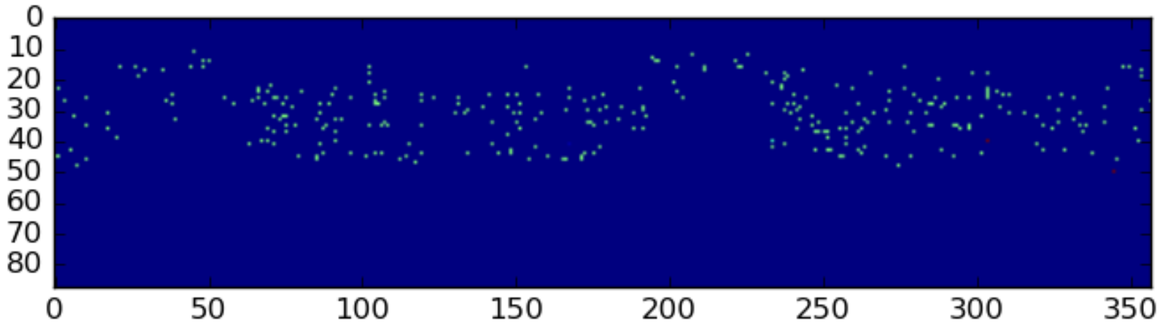
- Source at https://github.com/anlthms/meetup2/blob/master/cnn1.py
- Command line:

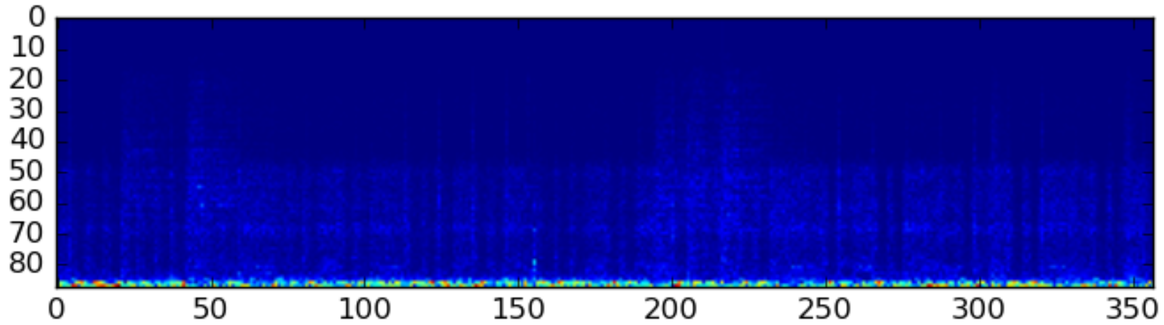
```
./cnn1.py -e 16 -w /home/ubuntu/nervana/music -r 0 -v
```

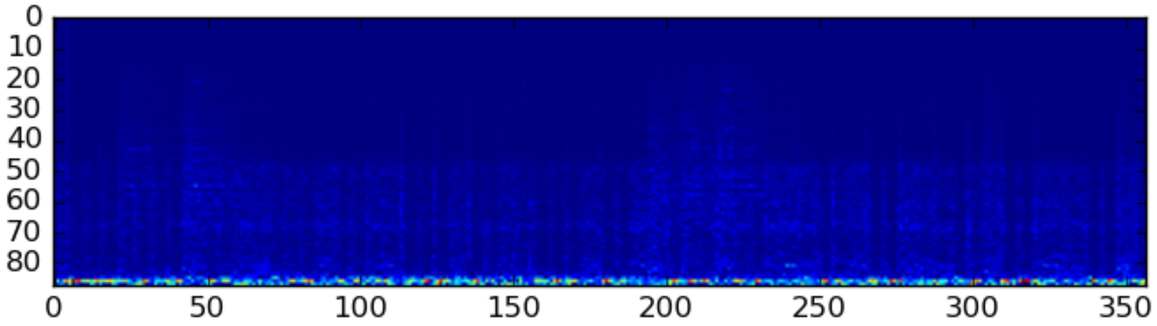
Sample spectrograms

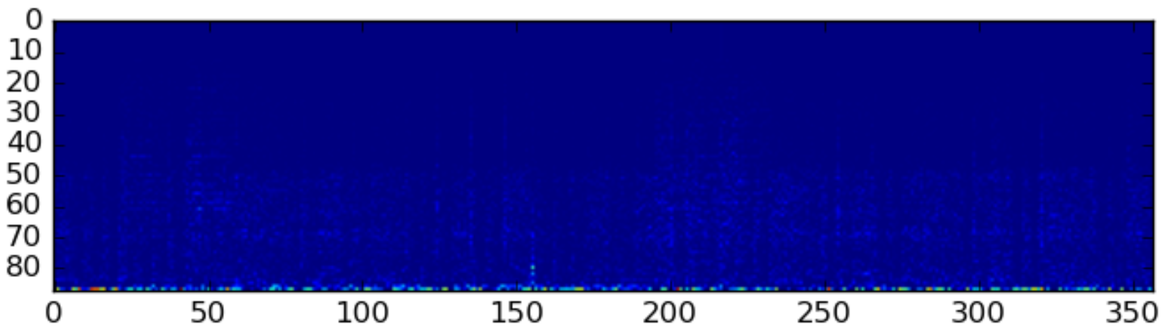


Sample output of conv layer #1

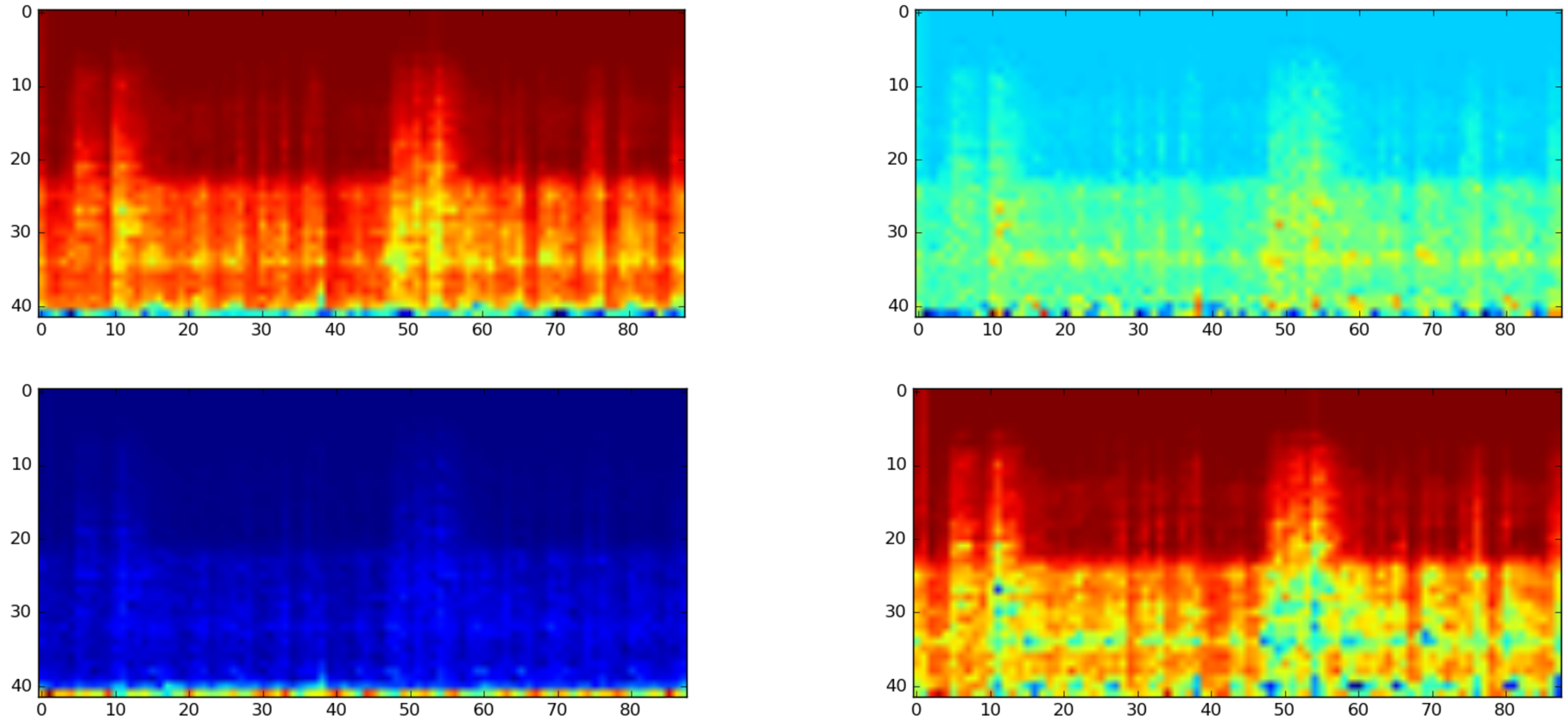






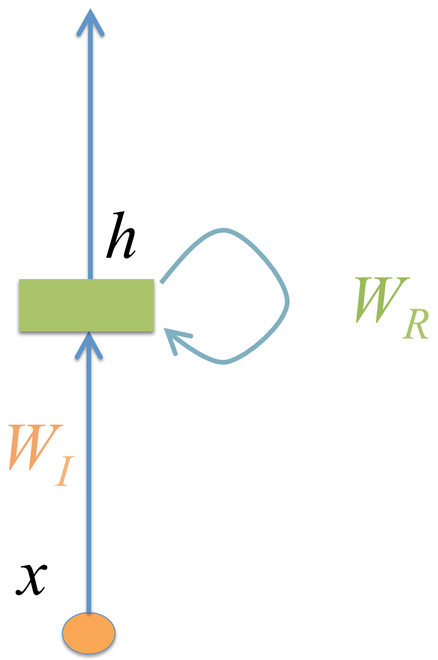


Sample output of conv layer #2



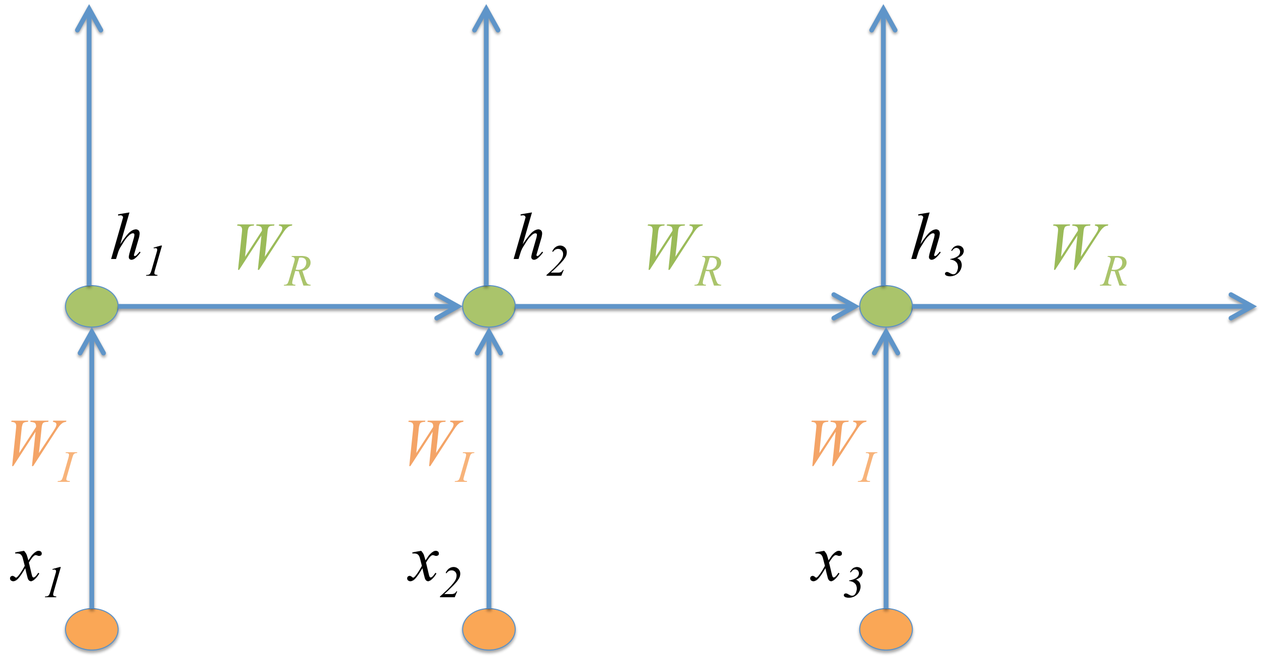
RNN THEORY

Recurrent layer



$$h_t = g\,(\,W_I\,x_t + W_R\,h_{(t-1)}\,)$$

Recurrent layer (unrolled)



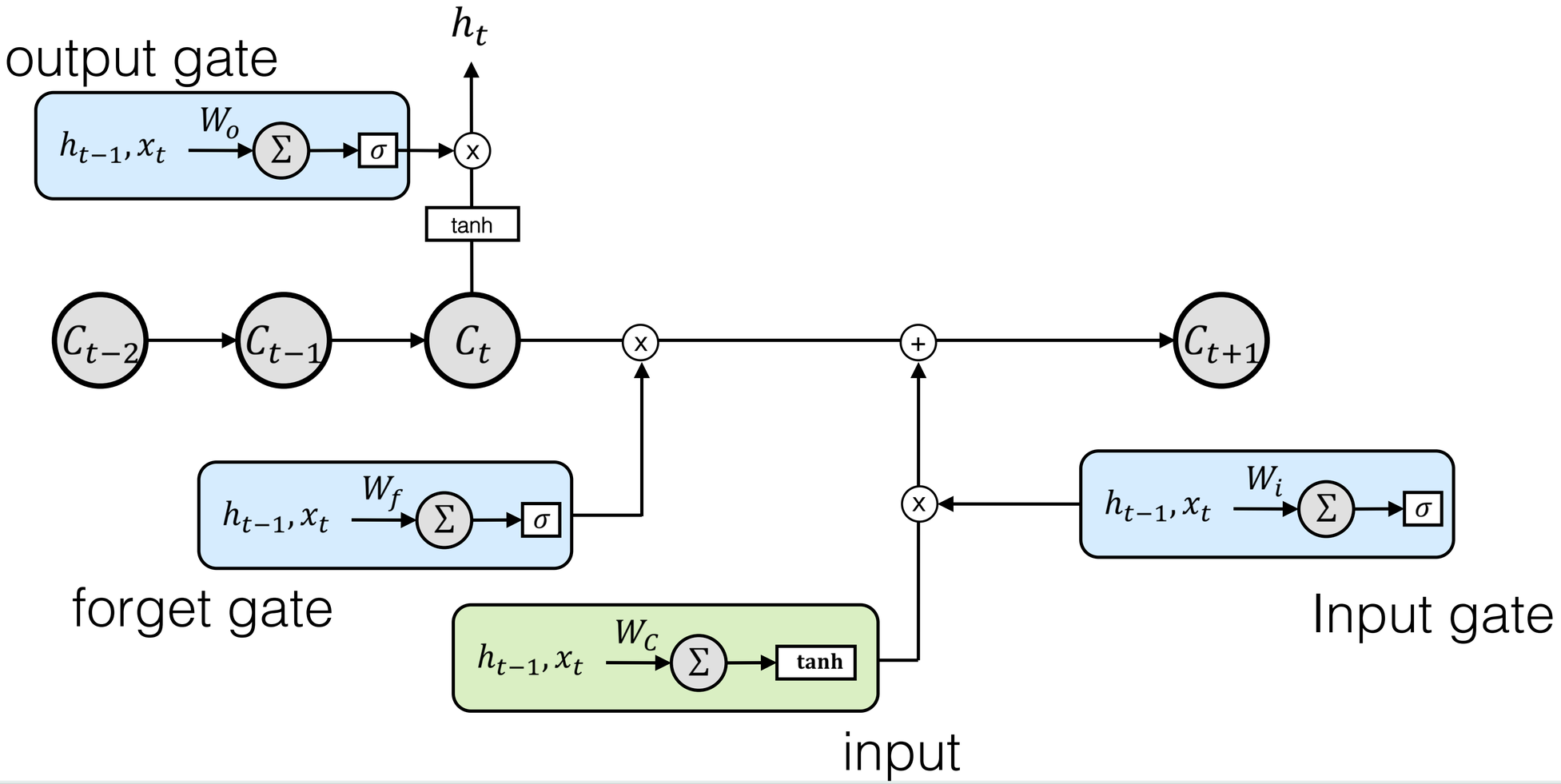
$$h_t = g\,(\,W_I\,x_t + W_R\,h_{(t-1)}\,)$$

- x represents the input sequence
- h is the memory (as well as the output of the recurrent layer)
- h is computed from the past memory and the current input
- h summarizes the sequence up to the current time step

Training RNNs

- Exploding/vanishing gradients
- Initialization
- Gradient clipping
- Optimizers with Adaptive learning rates (e.g. Adagrad)
- LSTM to capture long term dependencies

LSTM


output gate
h_t
h_{t-1}, x_t
W_o
Σ
σ
x
tanh
C_{t-2}
C_{t-1}
C_t
x
+
C_{t+1}
h_{t-1}, x_t
W_f
Σ
σ
forget gate
h_{t-1}, x_t
W_C
Σ
tanh
input
x
h_{t-1}, x_t
W_i
Σ
σ
Input gate

Applications of RNNs

- Image captioning
- Speech recognition
- Machine translation
- Time-series analysis

RNN HANDS-ON

RNN example 1

- Source at https://github.com/anlthms/meetup2/blob/master/rnn1.py
- Command line:

```
./rnn1.py -e 16 -w /home/ubuntu/nervana/music -r 0 -v
```

(Does not work well! This example demonstrates challenges of training RNNs)

RNN example 2

- Source at https://github.com/anlthms/meetup2/blob/master/rnn2.py
- Command line:

```
./rnn2.py -e 16 -w /home/ubuntu/nervana/music -r 0 -v
```

(Uses Glorot init, gradient clipping, Adagrad and LSTM)

RNN example 3

- Source at https://github.com/anlthms/meetup2/blob/master/rnn3.py
- Command line:

```
./rnn3.py -e 16 -w /home/ubuntu/nervana/music -r 0 -v
```

(Uses multiple bi-rnn layers)

Final example

- Source at https://github.com/NervanaSystems/neon/blob/master/examples/whale_calls.py
- Command line:

```
./whale_calls.py -e 16 -w /home/ubuntu/nervana/wdc -r 0 -s whales.pkl -v
```

Network structure of final example

```
Convolution Layer 'Convolution_0': 1 x (81x49) inputs, 128 x (79x24) outputs, 0,0 padding, 1,2 stride
Activation Layer 'Convolution_0_Rectlin': Rectlin
Convolution Layer 'Convolution_1': 128 x (79x24) inputs, 256 x (77x22) outputs, 0,0 padding, 1,1 stride
BatchNorm Layer 'Convolution_1_bnorm': 433664 inputs, 1 steps, 256 feature maps
Activation Layer 'Convolution_1_Rectlin': Rectlin
Pooling Layer 'Pooling_0': 256 x (77x22) inputs, 256 x (38x11) outputs
Convolution Layer 'Convolution_2': 256 x (38x11) inputs, 512 x (37x10) outputs, 0,0 padding, 1,1 stride
BatchNorm Layer 'Convolution_2_bnorm': 189440 inputs, 1 steps, 512 feature maps
Activation Layer 'Convolution_2_Rectlin': Rectlin
BiRNN Layer 'BiRNN_0': 18944 inputs, (256 outputs) * 2, 10 steps
BiRNN Layer 'BiRNN_1': (256 inputs) * 2, (256 outputs) * 2, 10 steps
BiRNN Layer 'BiRNN_2': (256 inputs) * 2, (256 outputs) * 2, 10 steps
RecurrentOutput choice RecurrentLast : (512, 10) inputs, 512 outputs
Linear Layer 'Linear_0': 512 inputs, 32 outputs
BatchNorm Layer 'Linear_0_bnorm': 32 inputs, 1 steps, 32 feature maps
Activation Layer 'Linear_0_Rectlin': Rectlin
Linear Layer 'Linear_1': 32 inputs, 2 outputs
Activation Layer 'Linear_1_Softmax': Softmax
```

More info

Nervana's deep learning tutorials:

https://www.nervanasys.com/deep-learning-tutorials/

Github page:

https://github.com/NervanaSystems/neon

For more information, contact:

info@nervanasys.com

nervana

MAKING MACHINES SMARTER.