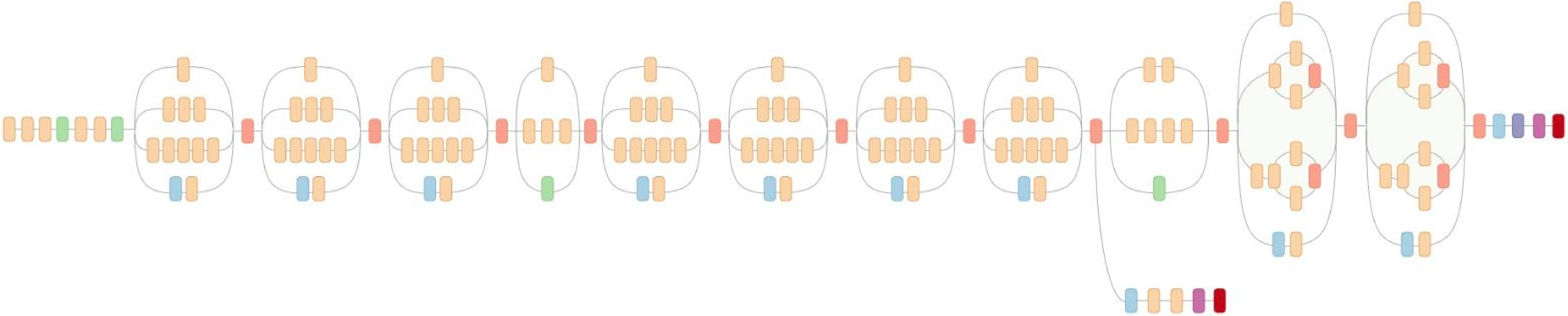


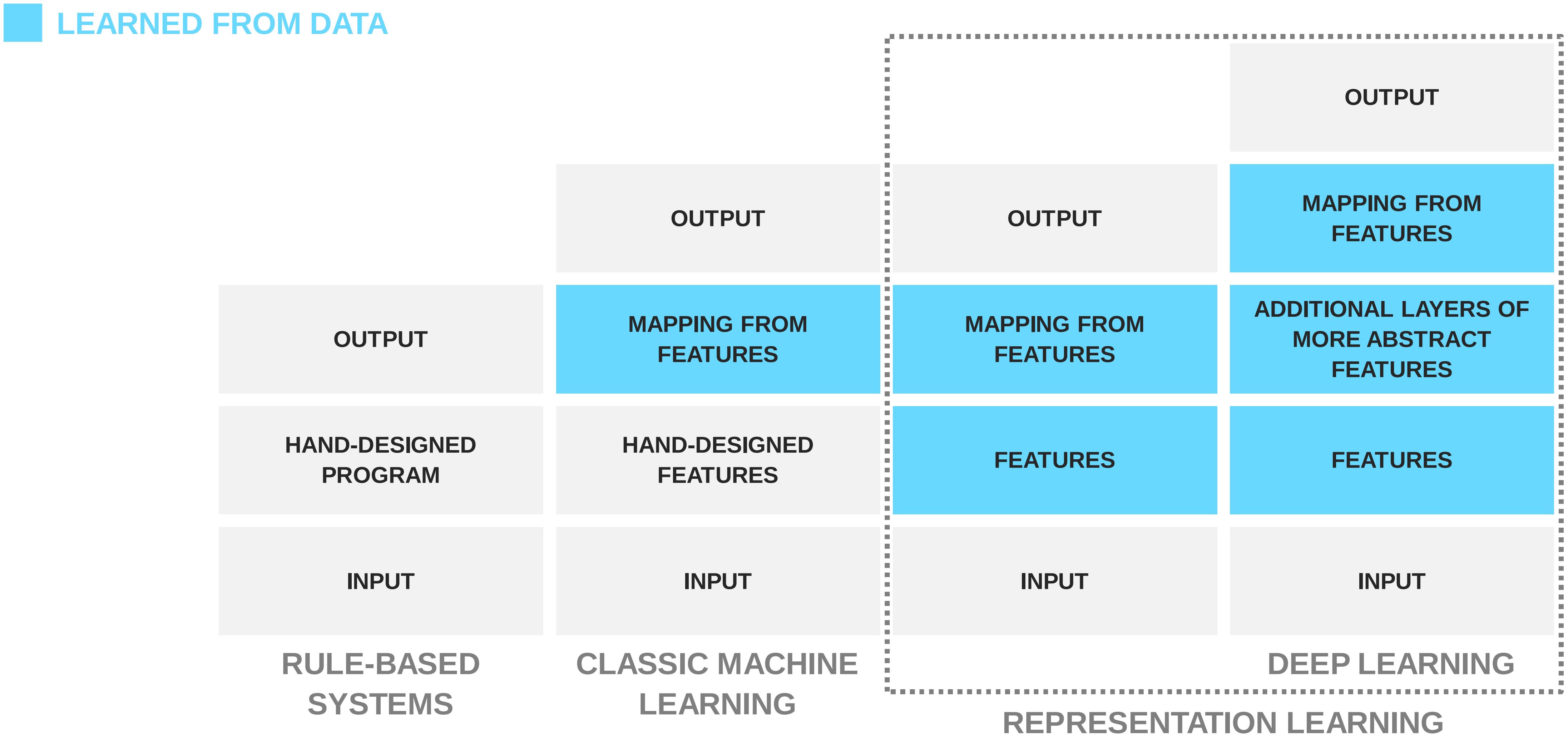
Deep Learning

Presenter: Dr. Denis Krompaß
Siemens Corporate Technology – Machine Intelligence Group
Denis.Krompass@siemens.com

Slides: Dr. Denis Krompaß and Dr. Sigurd Spieckermann

2016/06/01

Deep Learning vs. Classic Data Modeling



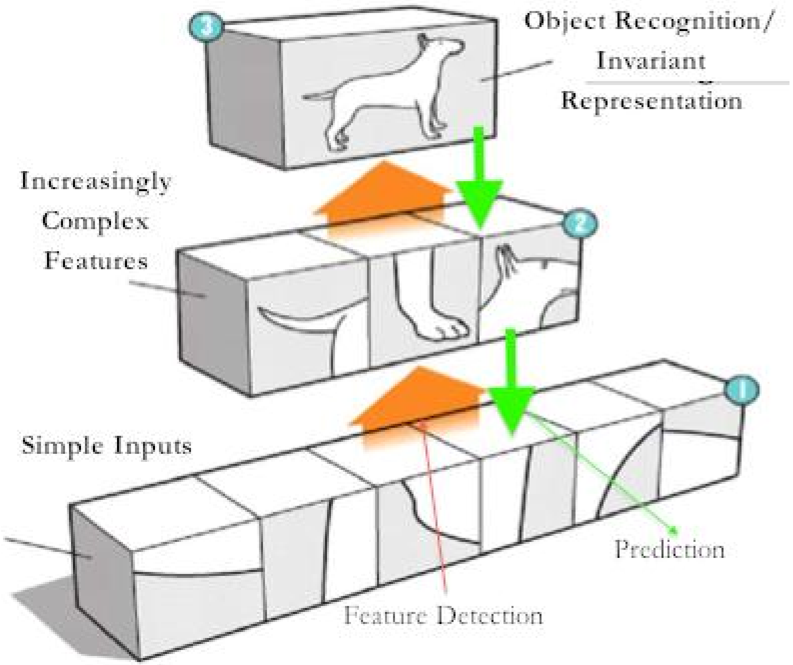
🟦 LEARNED FROM DATA

			OUTPUT
	OUTPUT	OUTPUT	🟦 MAPPING FROM FEATURES
OUTPUT	🟦 MAPPING FROM FEATURES	🟦 MAPPING FROM FEATURES	🟦 ADDITIONAL LAYERS OF MORE ABSTRACT FEATURES
HAND-DESIGNED PROGRAM	HAND-DESIGNED FEATURES	🟦 FEATURES	🟦 FEATURES
INPUT	INPUT	INPUT	INPUT
RULE-BASED SYSTEMS	CLASSIC MACHINE LEARNING		DEEP LEARNING
		REPRESENTATION LEARNING	

2016/06/01

SOURCE: http://www.deeplearningbook.org/contents/intro.html

Deep Learning
Hierarchical Feature Extraction



This illustration only shows the idea!
In reality the learned features are abstract and hard to interpret most of the time.

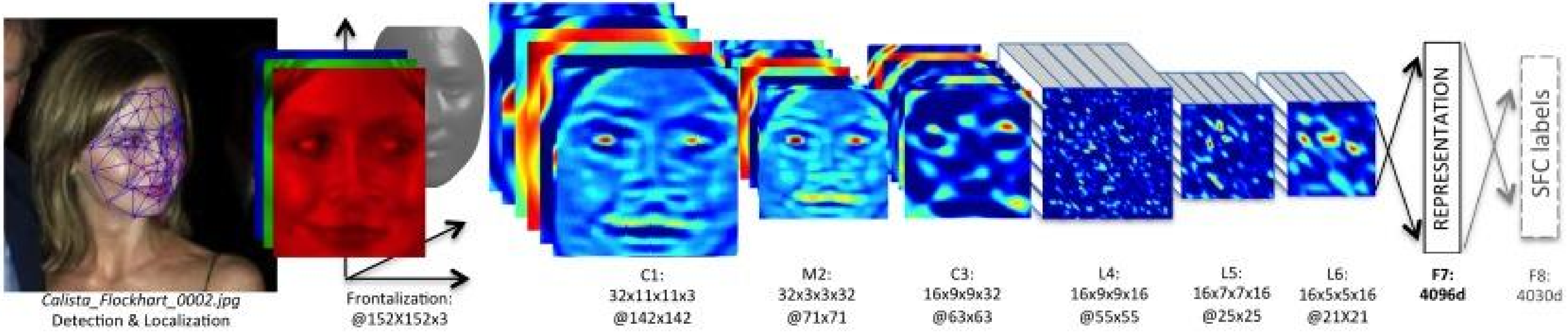
2016/06/01

SOURCE: http://www.eidolonspeak.com/Artificial_Intelligence/SOA_P3_Fig4.png

Deep Learning

Hierarchical Feature Extraction

facebook



SOURCE:
Taigman, Y., Yang, M., Ranzato, M. A., & Wolf, L. (2014). DeepFace: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 1701-1708).

NEURAL NETWORKS HAVE BEEN AROUND FOR DECADES!

(Classic) Neural Networks are an important building block of Deep Learning but there is more to it.

What’s new?

* deprecated

OPTIMIZATION & LEARNING

OPTIMIZATION ALGORITHMS

- Adaptive Learning Rates (e.g. ADAM)
- Evolution Strategies
- Synthetic Gradients
- Asynchronous Training
- …

REPARAMETERIZATION

- Batch Normalization
- Weight Normalization
- …

REGULARIZATION

- Dropout
- DropConnect
- DropPath
- …

MODEL ARCHITECTURES

BUILDING BLOCKS

- Spatial/temporal pooling
- Attention mechanism
- Variational Layers
- Dilated convolution
- Variable-length sequence modeling
- Macro modules (e.g. Residual Units)
- Factorized layers
- …

ARCHITECTURES

- Neural computers and memories
- General purpose image feature extractors (VGG, GoogleLeNet)
- End-to-end models
- Generative Adversarial Networks
- …

SOFTWARE

- Theano
  - Keras
  - Blocks
- TensorFlow
  - Keras
  - Sonnet
  - TensorflowFold
- Torch7
- Caffe
- …

GENERAL

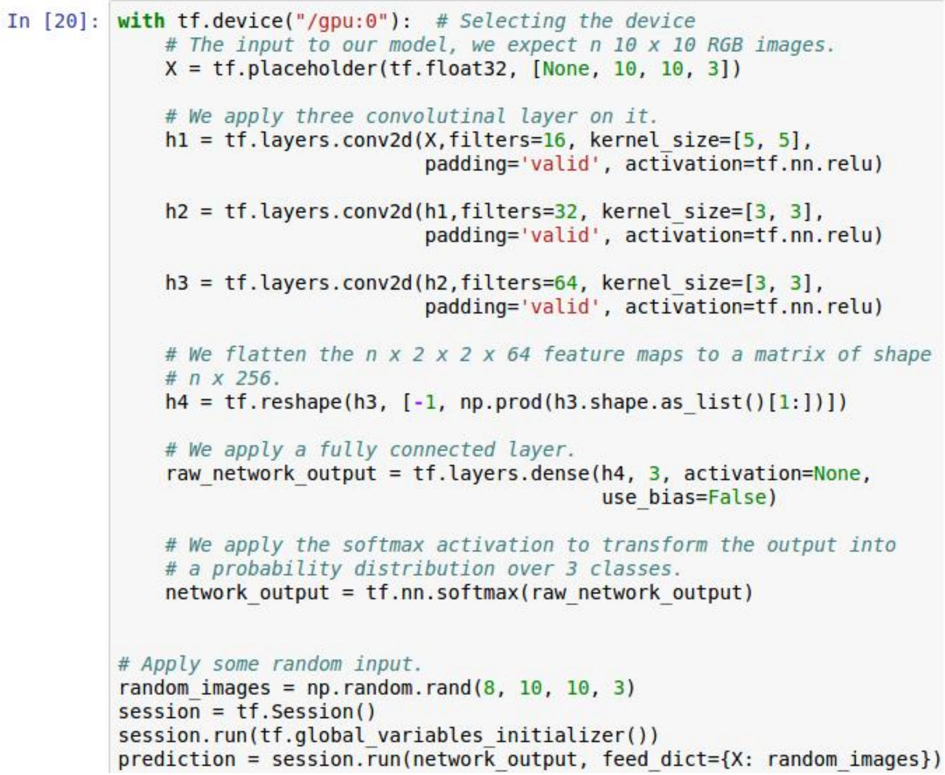
- GPUs
- Hardware accessibility (Cloud)
- Distributed Learning
- Data

2016/06/01

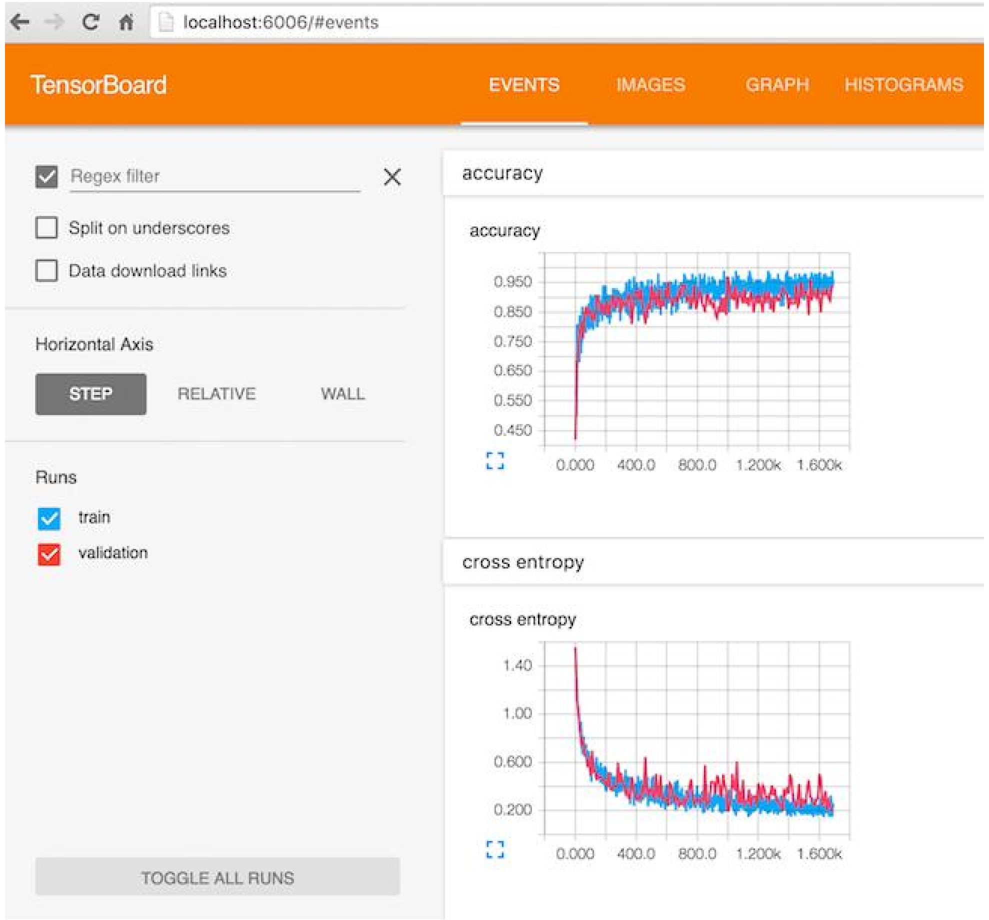
Enabler: Tools

It has never been that easy to build deep learning models!

TensorFlow™



```
In [20]: with tf.device("/gpu:0"):  # Selecting the device
             # The input to our model, we expect n 10 x 10 RGB images.
             X = tf.placeholder(tf.float32, [None, 10, 10, 3])

             # We apply three convolutinal layer on it.
             h1 = tf.layers.conv2d(X,filters=16, kernel_size=[5, 5],
                                   padding='valid', activation=tf.nn.relu)

             h2 = tf.layers.conv2d(h1,filters=32, kernel_size=[3, 3],
                                   padding='valid', activation=tf.nn.relu)

             h3 = tf.layers.conv2d(h2,filters=64, kernel_size=[3, 3],
                                   padding='valid', activation=tf.nn.relu)

             # We flatten the n x 2 x 2 x 64 feature maps to a matrix of shape
             # n x 256.
             h4 = tf.reshape(h3, [-1, np.prod(h3.shape.as_list()[1:])])

             # We apply a fully connected layer.
             raw_network_output = tf.layers.dense(h4, 3, activation=None,
                                                  use_bias=False)

             # We apply the softmax activation to transform the output into
             # a probability distribution over 3 classes.
             network_output = tf.nn.softmax(raw_network_output)


         # Apply some random input.
         random_images = np.random.rand(8, 10, 10, 3)
         session = tf.Session()
         session.run(tf.global_variables_initializer())
         prediction = session.run(network_output, feed_dict={X: random_images})
```



localhost:6006/#events

☑ Regex filter

□ Split on underscores

□ Data download links

Horizontal Axis: [✓ STEP] RELATIVE WALL

Runs

☑ train

☑ validation

Enabler: Data

Deep Learning requires tons of labeled data if the problem is really complex.

# Labeled examples	Example problems solved in the world.
1 – 10	Not worth a try.
10 – 100	Toy datasets.
100 – 1,000	Toy datasets.
1,000 – 10,000	Hand-written digit recognition.
10,000 – 100,000	Text generation.
100,000 – 1,000,000	Question answering, chat bots.
> 1,000,000	Multi language text translation. Object recognition in images/videos.

Enabler: Computing Power for Everyone

Matrix Products are highly parallelizable

$$\boxed{h = X \cdot W}, X \in R^{n \times m}, W \in R^{m \times k}$$

NVIDIA GPUs

Distributed training enables us to train very large deep learning models on tons of data

Deep Learning Research

Companies

Microsoft

OpenAI

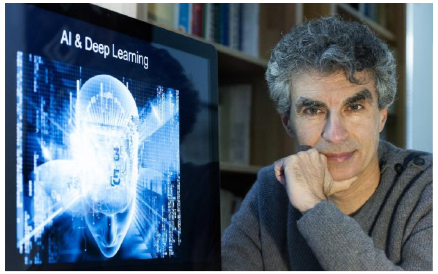
Google

Baidu Research

DeepMind

facebook

People



Yoshua Bengio

Andrew Ng

Geoffrey Hinton

Yann LeCun

Jürgen Schmidhuber

Lecture Overview

- Part I – Deep Learning Model Architecture Design
- Part II – Training Deep Learning Models
- Part III – Deep Learning and Artificial (General) Intelligence

Deep Learning

Part I

Deep Learning Model Architecture Design

Part I – Deep Learning Model Architecture

Basic Building Blocks

- The fully connected layer – Using brute force.
- Convolutional neural network layers – Exploiting neighborhood relations.
- Recurrent neural network layers – Exploiting sequential relations.

Thinking in Macro Structures

- Mixing things up – Generating purpose modules.
- LSTMs and Gating – Simple memory management.
- Attention – Dynamic context driven information selection.
- Inception - Dynamic receptive field expansion.
- Residual Units – Building ultra deep structures.

End-to-End model design

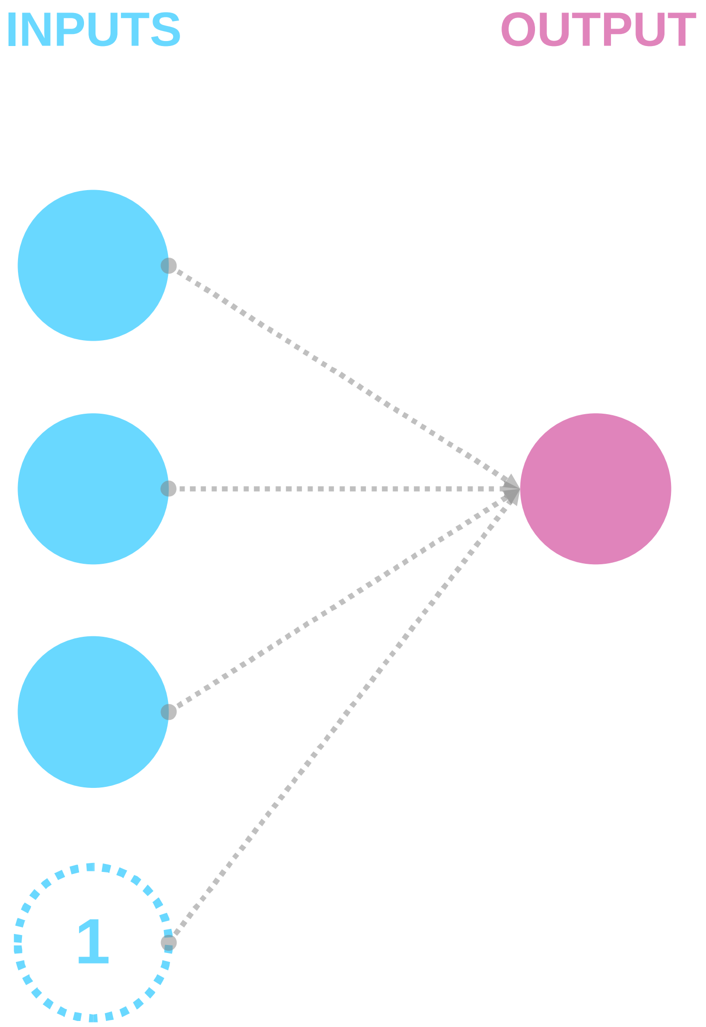
- Example for design choices.
- Real examples.

Deep Learning

Basic Building Blocks

Neural Network Basics

Linear Regression

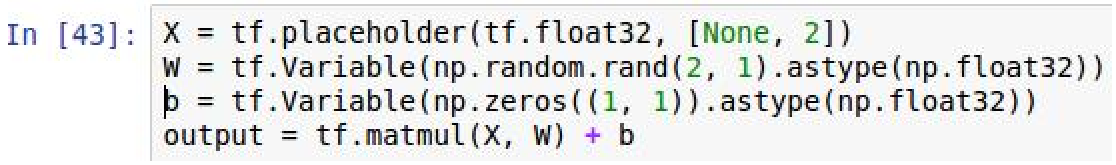


INPUTS

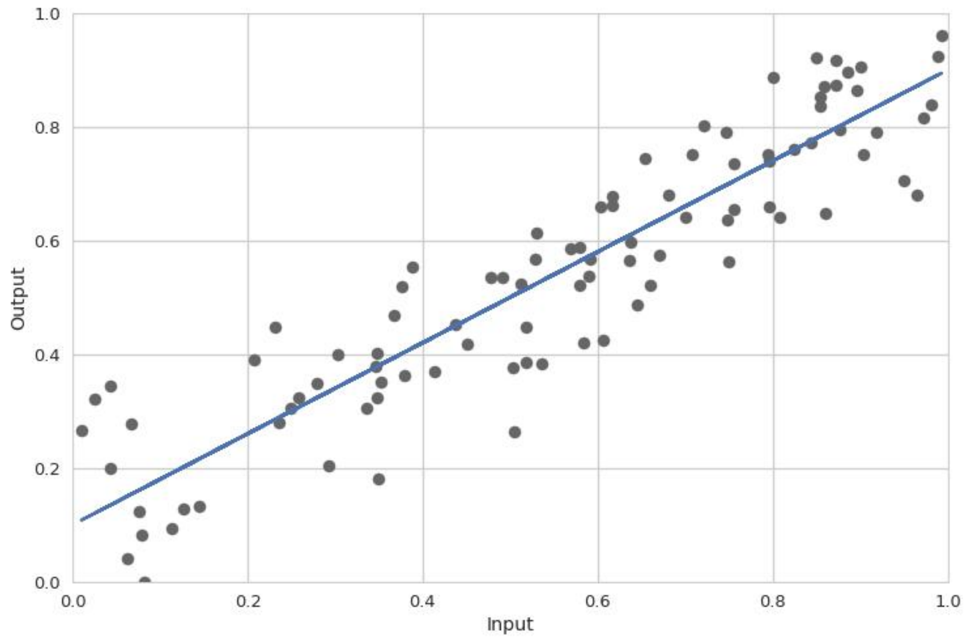
OUTPUT

1

$$\hat{y} = w^T x + b$$

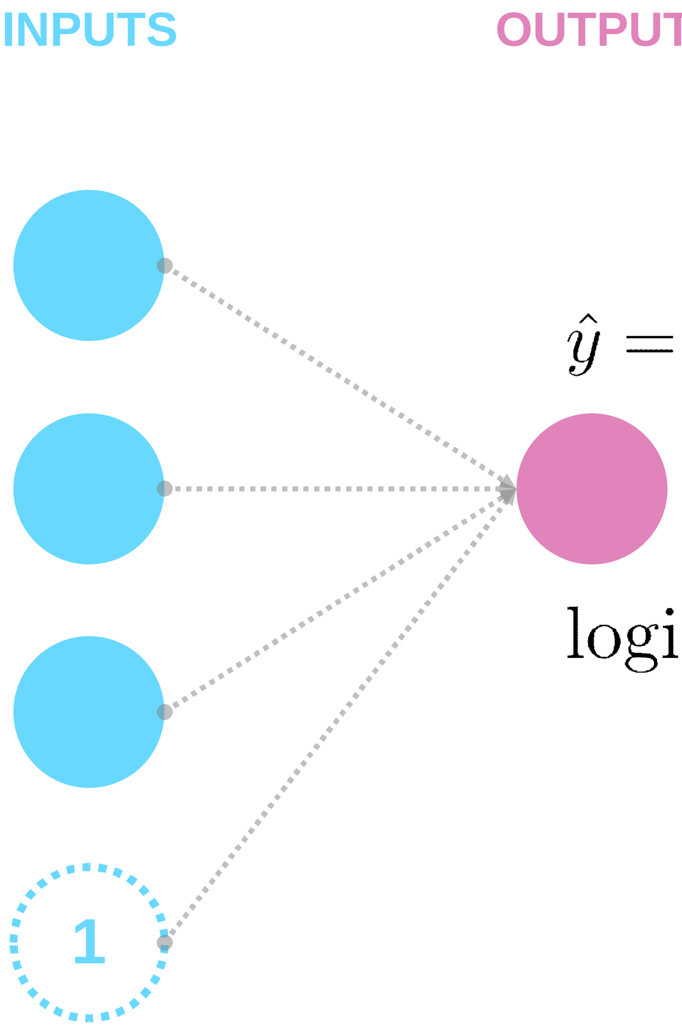


```
In [43]: X = tf.placeholder(tf.float32, [None, 2])
         W = tf.Variable(np.random.rand(2, 1).astype(np.float32))
         b = tf.Variable(np.zeros((1, 1)).astype(np.float32))
         output = tf.matmul(X, W) + b
```



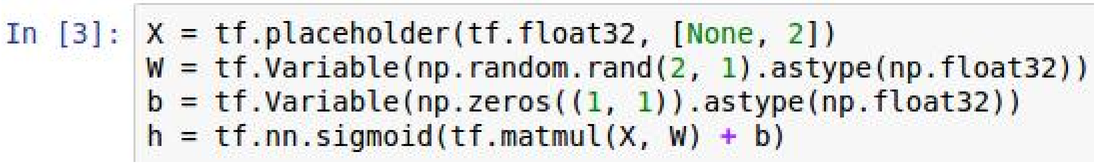
Neural Network Basics

Logistic Regression

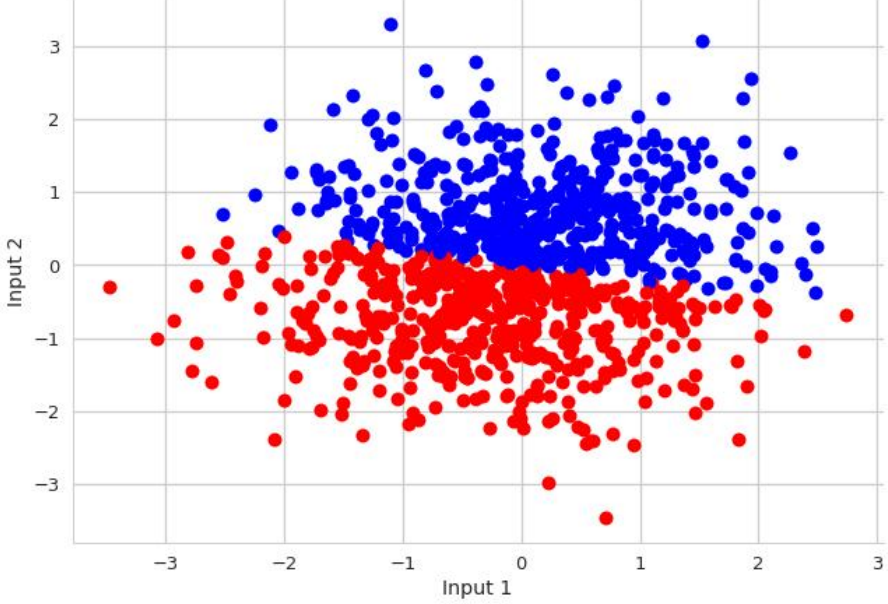


$$\hat{y} = \text{logistic}(w^T x + b)$$

$$\text{logistic}(z) = \frac{1}{1 + e^{-z}}$$



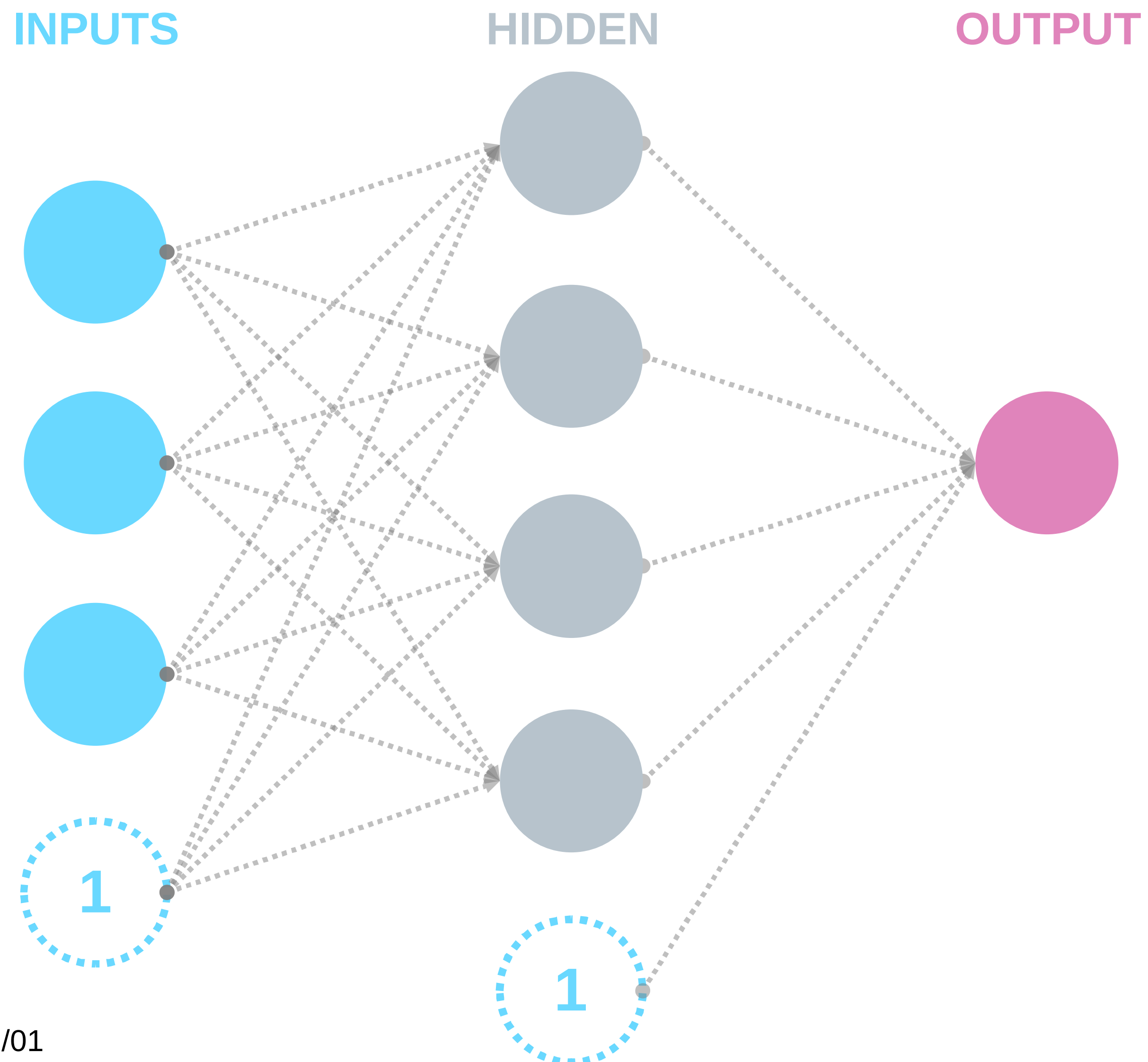
```
In [3]: X = tf.placeholder(tf.float32, [None, 2])
        W = tf.Variable(np.random.rand(2, 1).astype(np.float32))
        b = tf.Variable(np.zeros((1, 1)).astype(np.float32))
        h = tf.nn.sigmoid(tf.matmul(X, W) + b)
```



2016/06/01

Neural Network Basics

Multi-Layer Perceptron

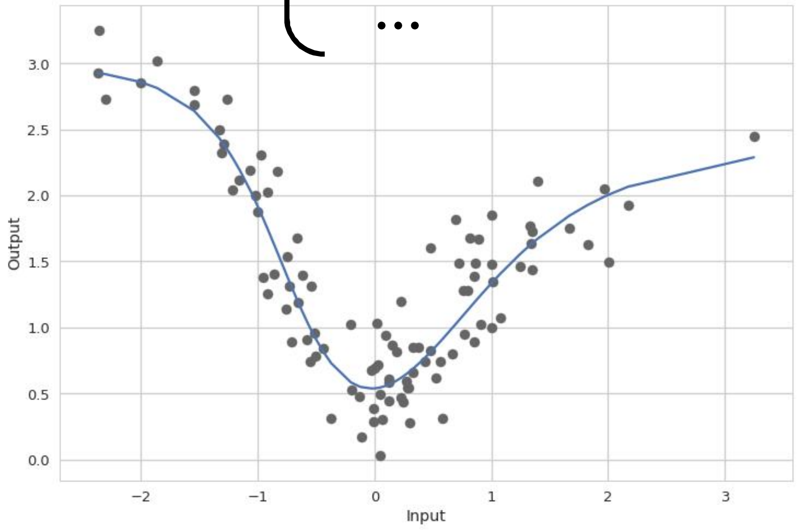


$h^{(1)} = f(W^{(1)}x + b^{(1)})$ ⬅ Hidden Layer

$\hat{y} = W^{(2)}h^{(1)} + b^{(2)}$ ⬅ Output Layer

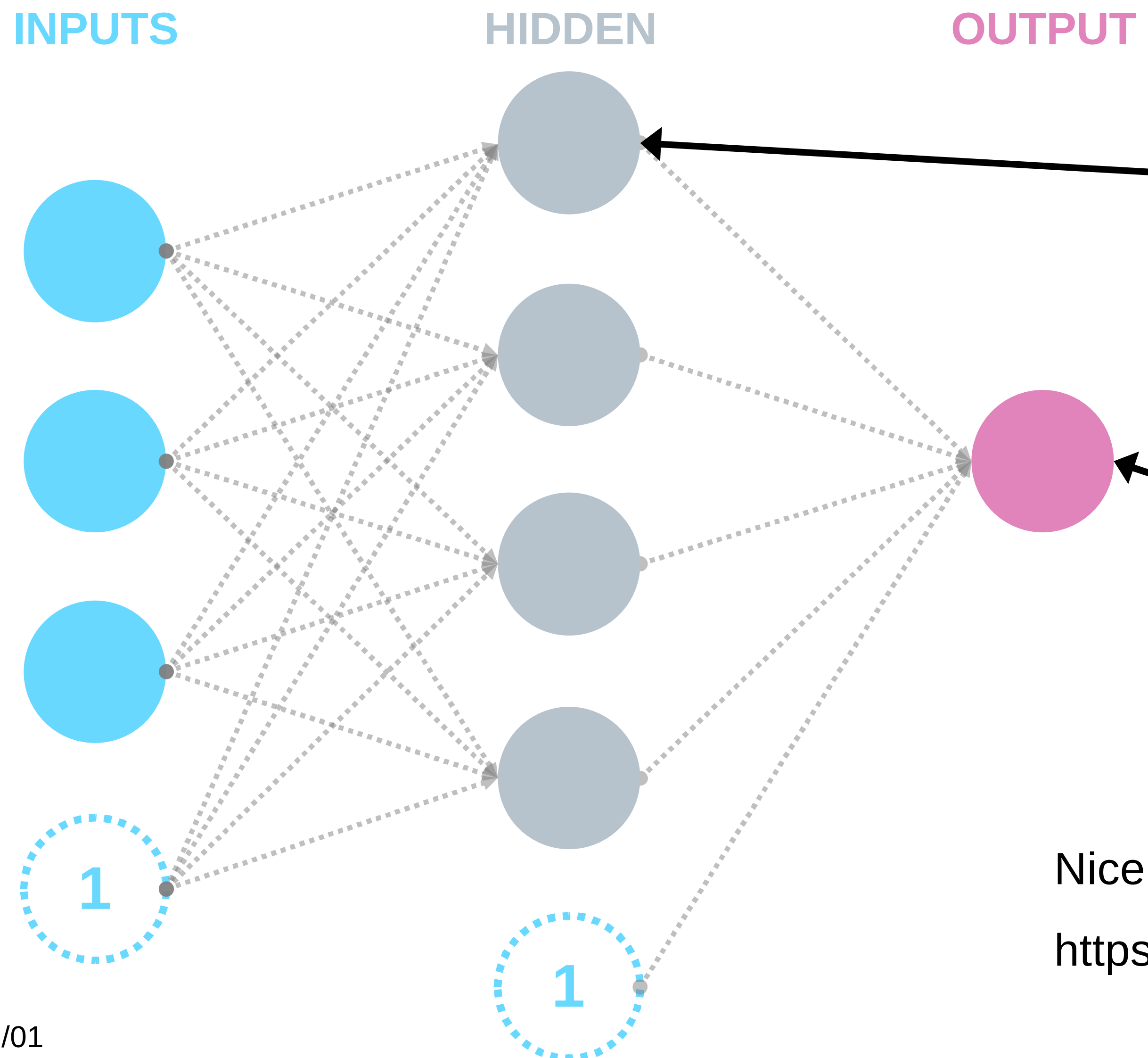
With activation function:

$$f(z) = \begin{cases} \tanh(z) \\ \mathrm{relu}(z) \\ \ldots \end{cases}$$



Neural Network Basics

Activation Functions



Activation Function
identity(h)
tanh(h)
relu(h)
…

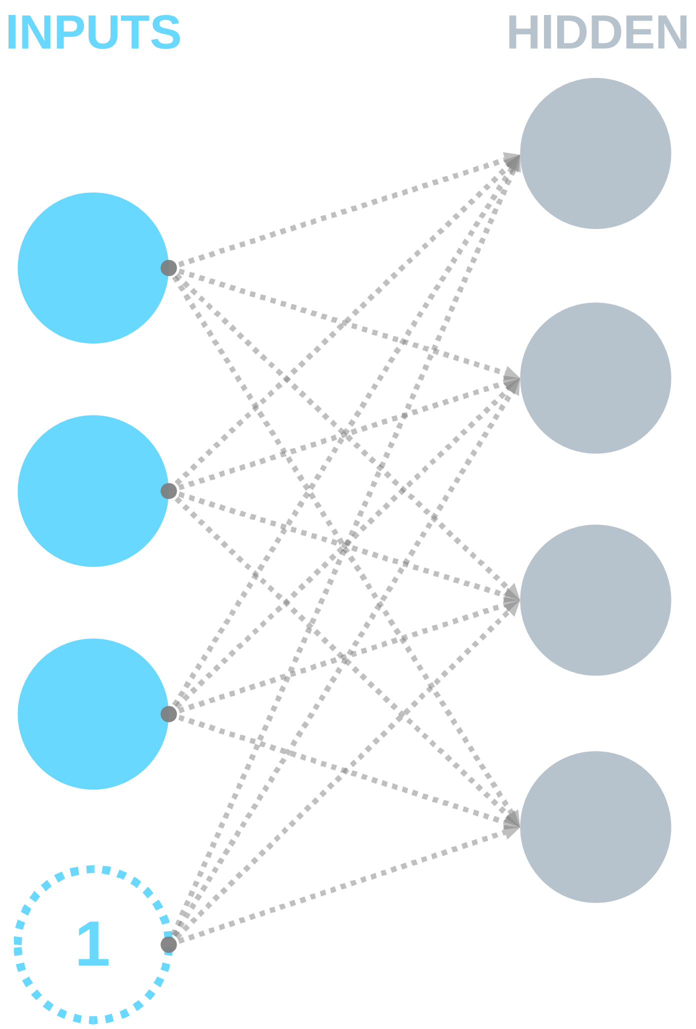
Activation Function	Task
identity(h)	Regression
logistic(h)	Binary Classification
softmax(h)	Multi-Class Classification

Nice overview on activation functions:

https://en.wikipedia.org/wiki/Activation_function

Basic Building Blocks

The Fully Connected Layer



Passing one example:

$$(1 \quad 3 \quad K \quad 2)$$

$$h^{(1)} = f(W^{(1)}x + b^{(1)})$$

$$x Î \ R^{1´ m}, W^{(1)} Î \ R^{m´ k}, b^{(1)} Î \ R^{1´ k}$$

Passing n examples:

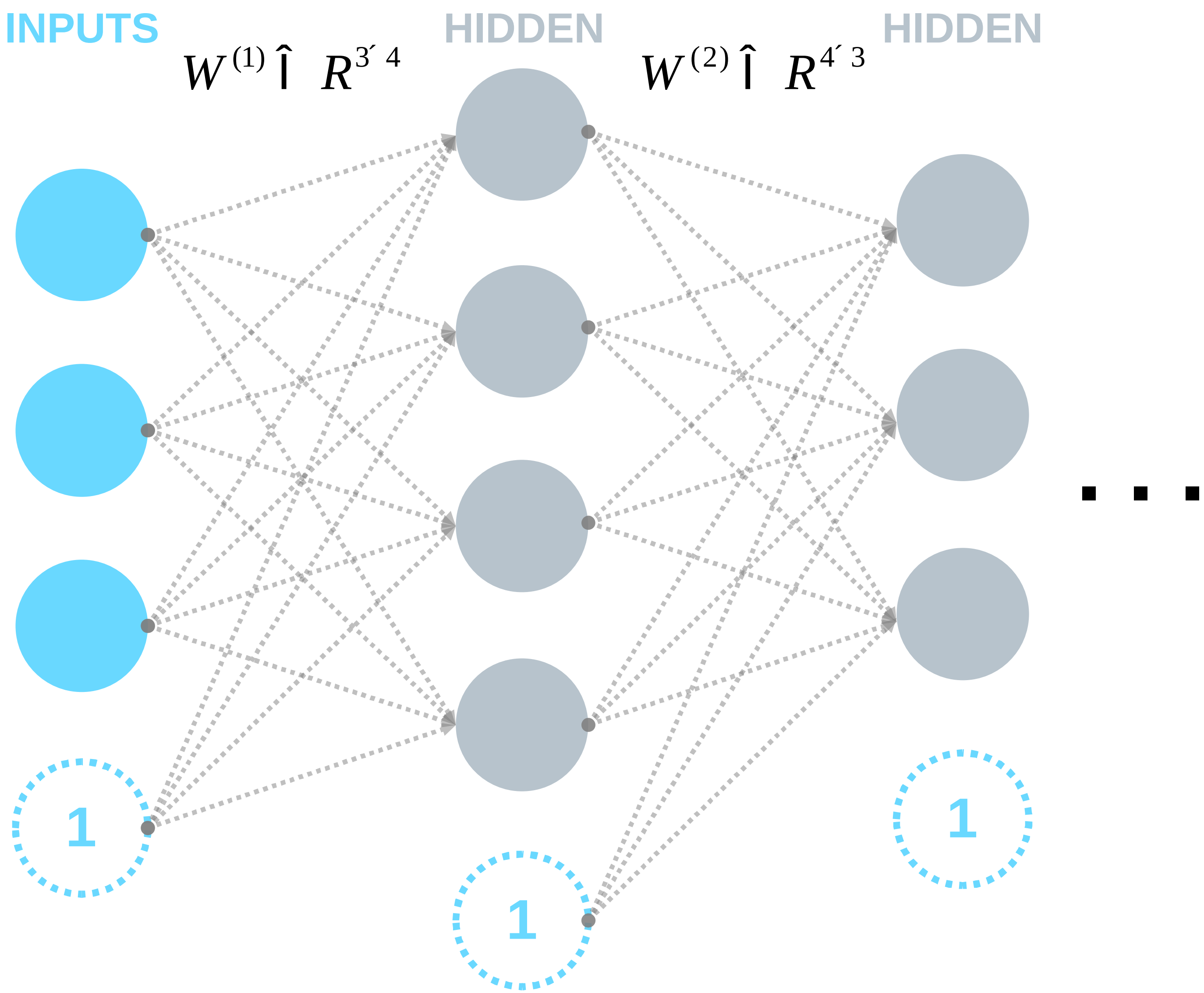
æ1 3 K 2ö
ç5 7 K 1÷
çM M M M÷
ç ÷
ç ÷
è3 4 K 3ø

$$H^{(1)} = f(W^{(1)}X + b^{(1)})$$

$$X Î \ R^{n´ m}, W^{(1)} Î \ R^{m´ k}, b^{(1)} Î \ R^{1´ k}$$

Basic Building Blocks

The Fully Connected Layer – Stacking



$$h^{(1)} = f(W^{(1)}x + b^{(1)})$$

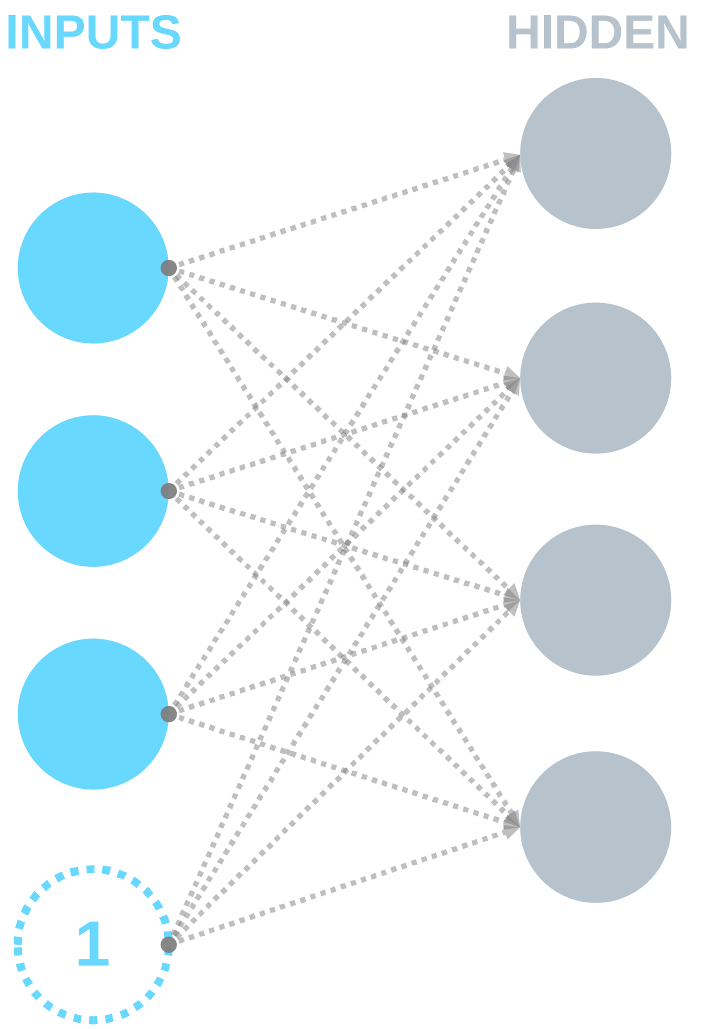
$$h^{(2)} = f(W^{(2)}h^{(1)} + b^{(2)})$$

$$\ldots$$

$$h^{(l)} = f(W^{(l)}h^{(l-1)} + b^{(l)})$$

Basic Building Blocks

The Fully Connected Layer – Using Brute Force



Brute force layer:

• Exploits no assumptions about the inputs.

ØNo weight sharing.

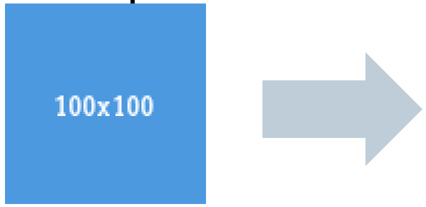
•Simply combines all inputs with each other.

ØExpensive! Often responsible for the largest amount of parameters in a deep learning model.

•Use with care since it can quickly over-parameterize the model

ØCan lead to degenerated solutions.

Examples:



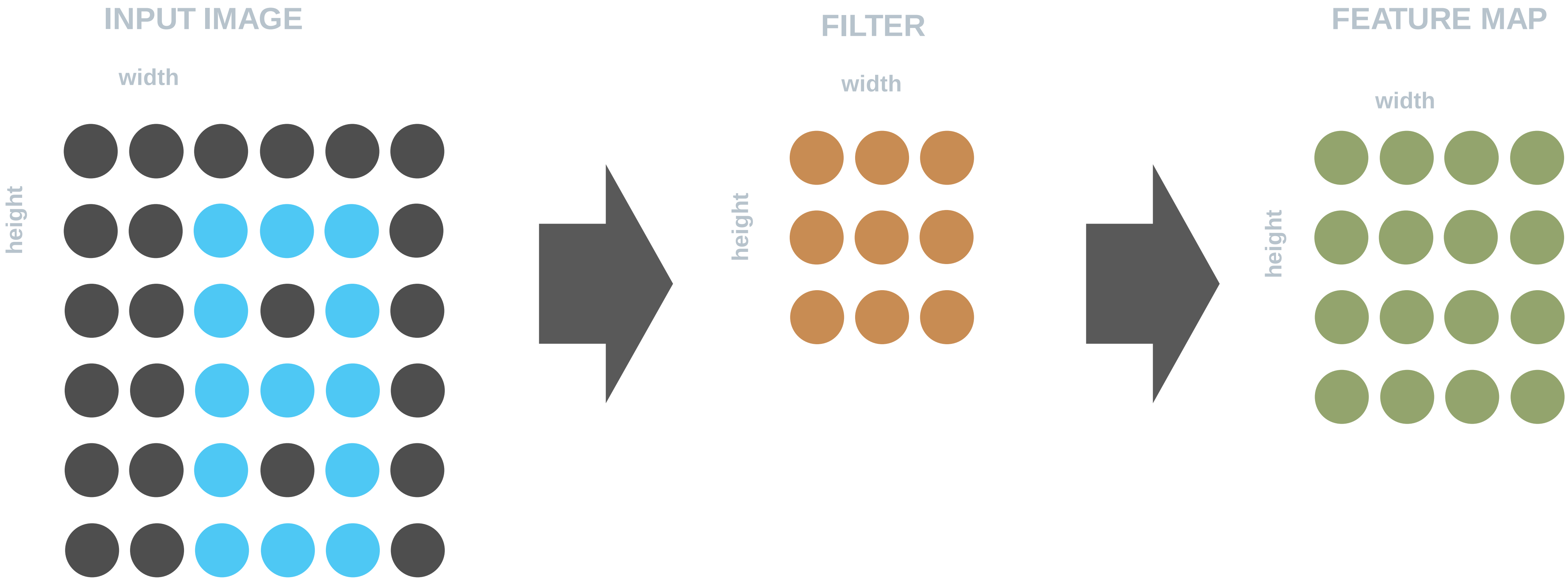
RGB image of shape
100 x 100 x 3

3,000,000 free parameters for a fully connected layer with 100 hidden units!

Two consecutive fully connected layer with 1000 hidden neurons each: 1,000,000 free parameters!

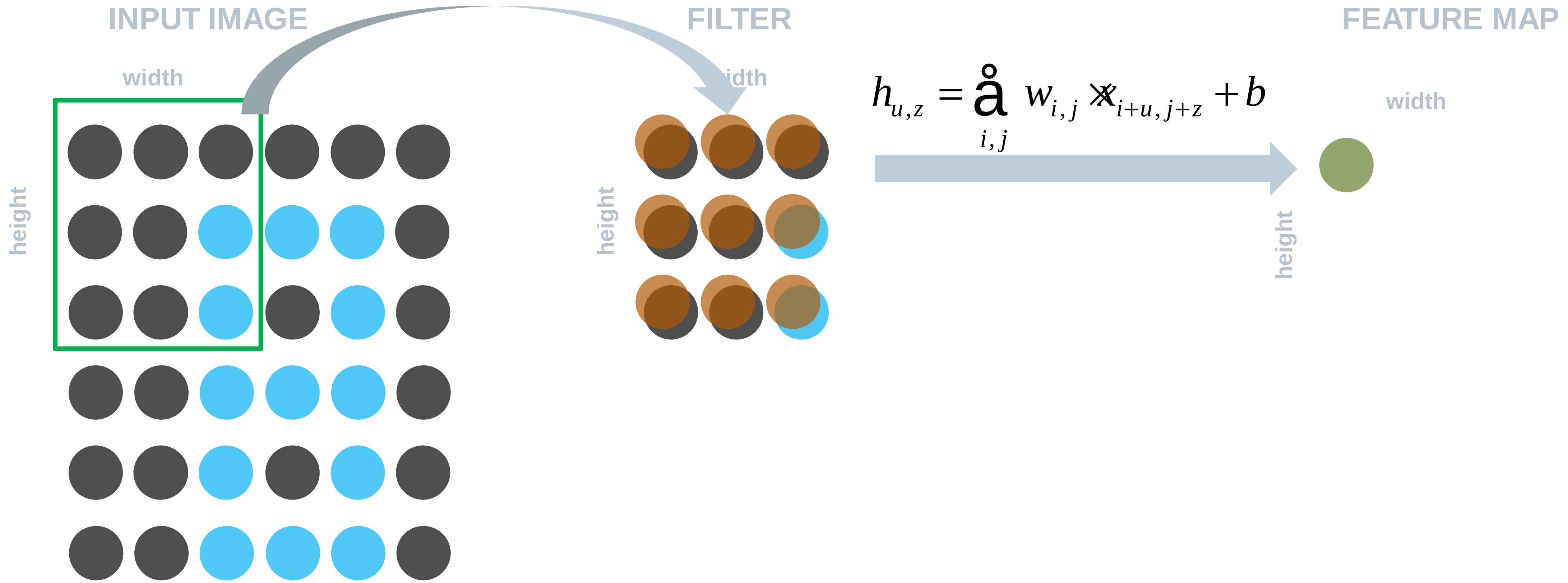
Basic Building Blocks

Convolutional Layer - Convolution of Filters



Basic Building Blocks

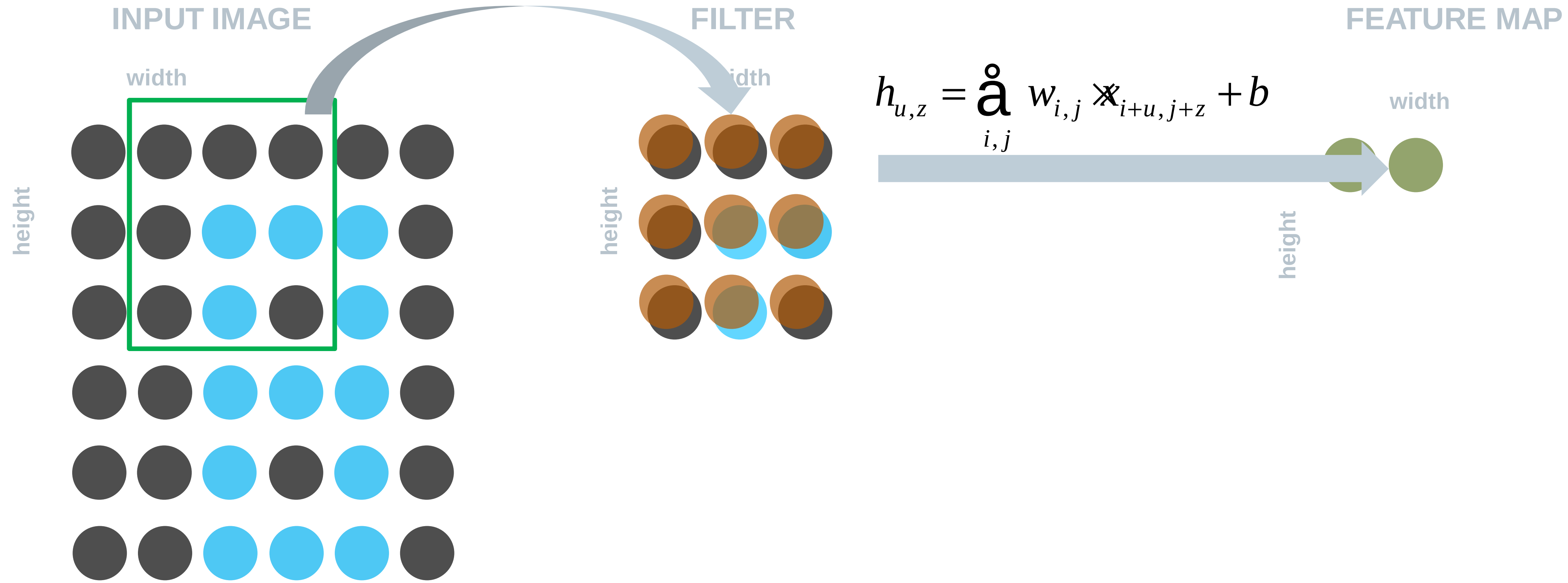
(Valid) Convolution of Filter



$$h_{u,z} = \sum_{i,j} w_{i,j} \cdot x_{i+u,j+z} + b$$

Basic Building Blocks

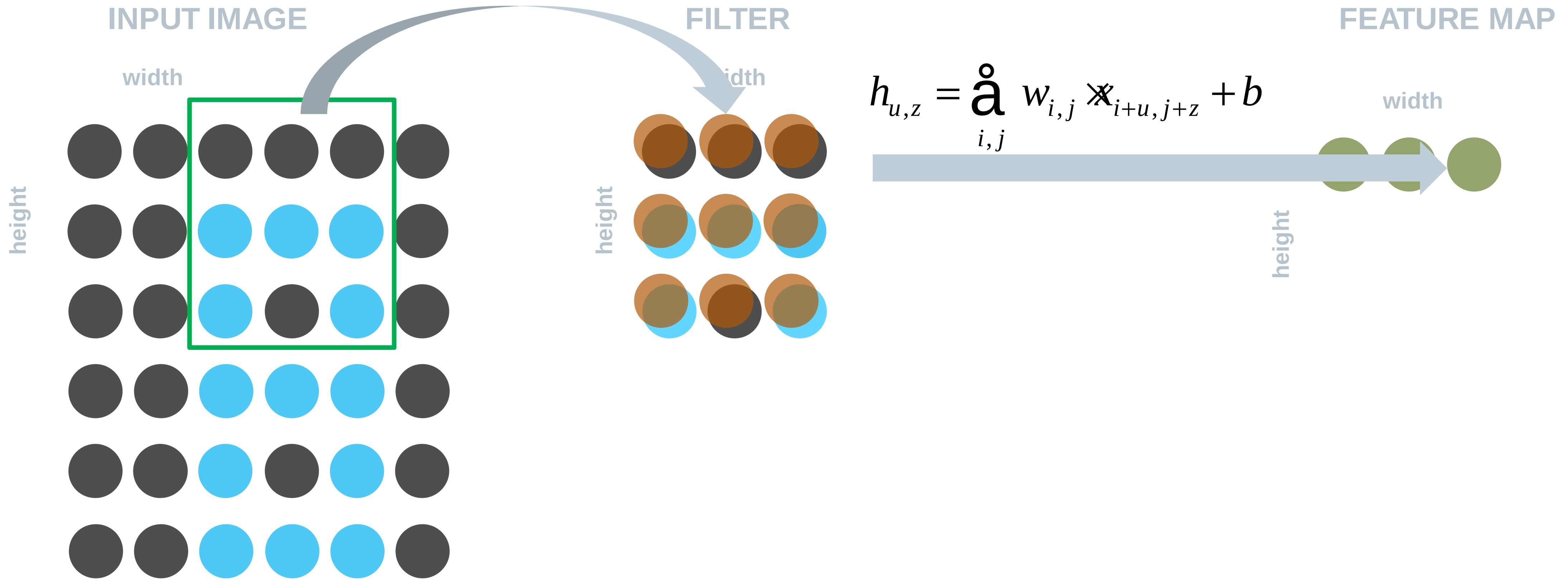
(Valid) Convolution of Filter



$$h_{u,z} = \sum_{i,j} w_{i,j} \cdot x_{i+u,j+z} + b$$

Basic Building Blocks

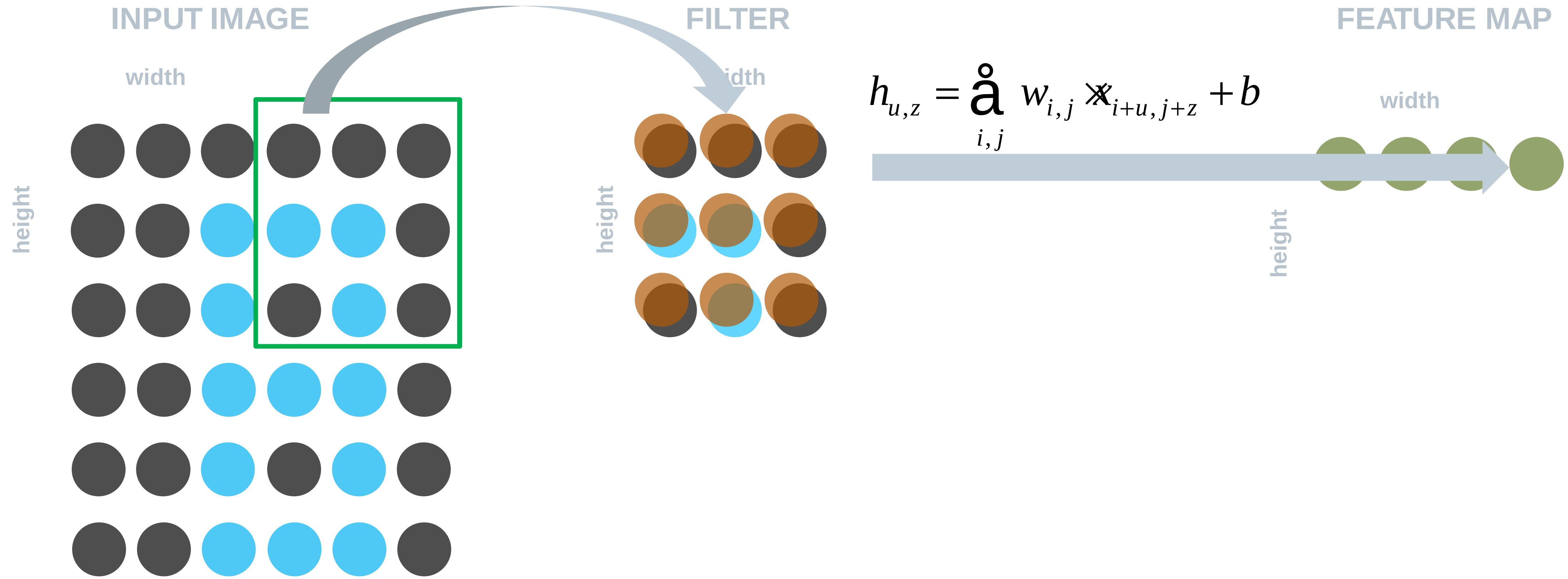
(Valid) Convolution of Filter



$$h_{u,z} = \sum_{i,j} w_{i,j} \times x_{i+u,j+z} + b$$

Basic Building Blocks

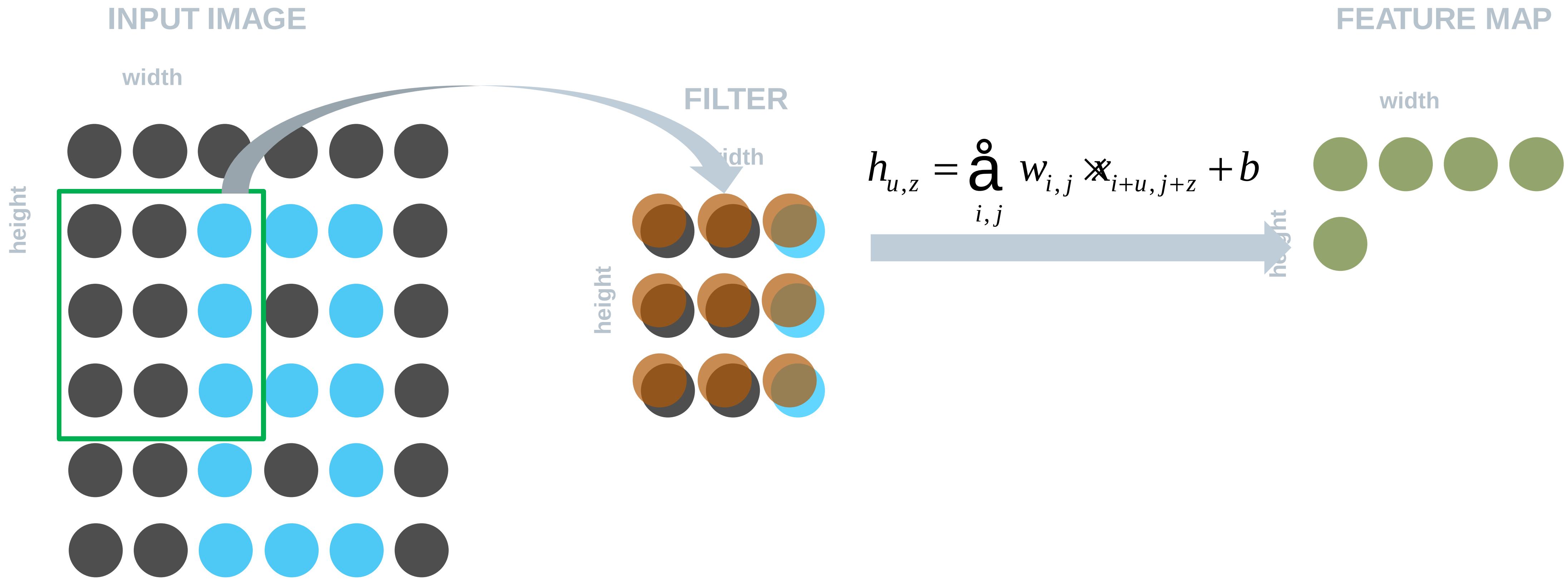
(Valid) Convolution of Filter



$$h_{u,z} = \text{å}_{i,j} w_{i,j} \times x_{i+u,j+z} + b$$

Basic Building Blocks

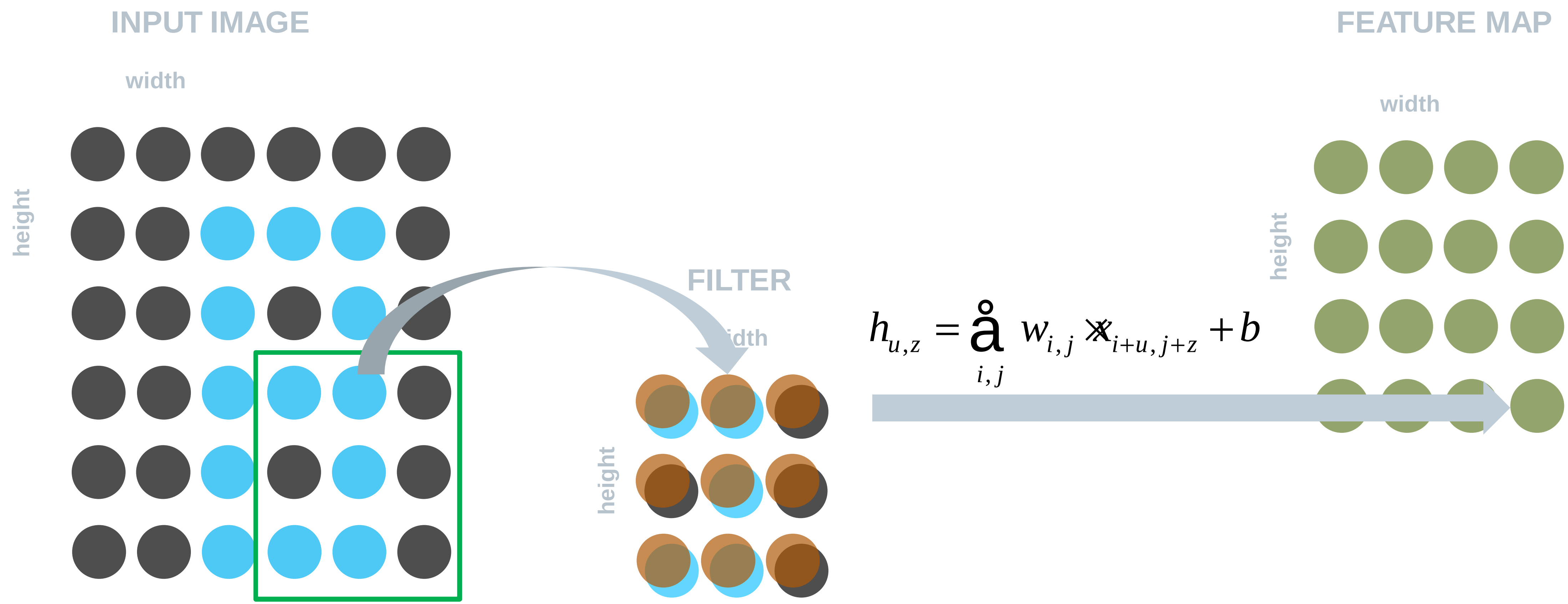
(Valid) Convolution of Filter



$$h_{u,z} = \underset{i,j}{\mathring{a}}\; w_{i,j} \times x_{i+u,j+z} + b$$

Basic Building Blocks

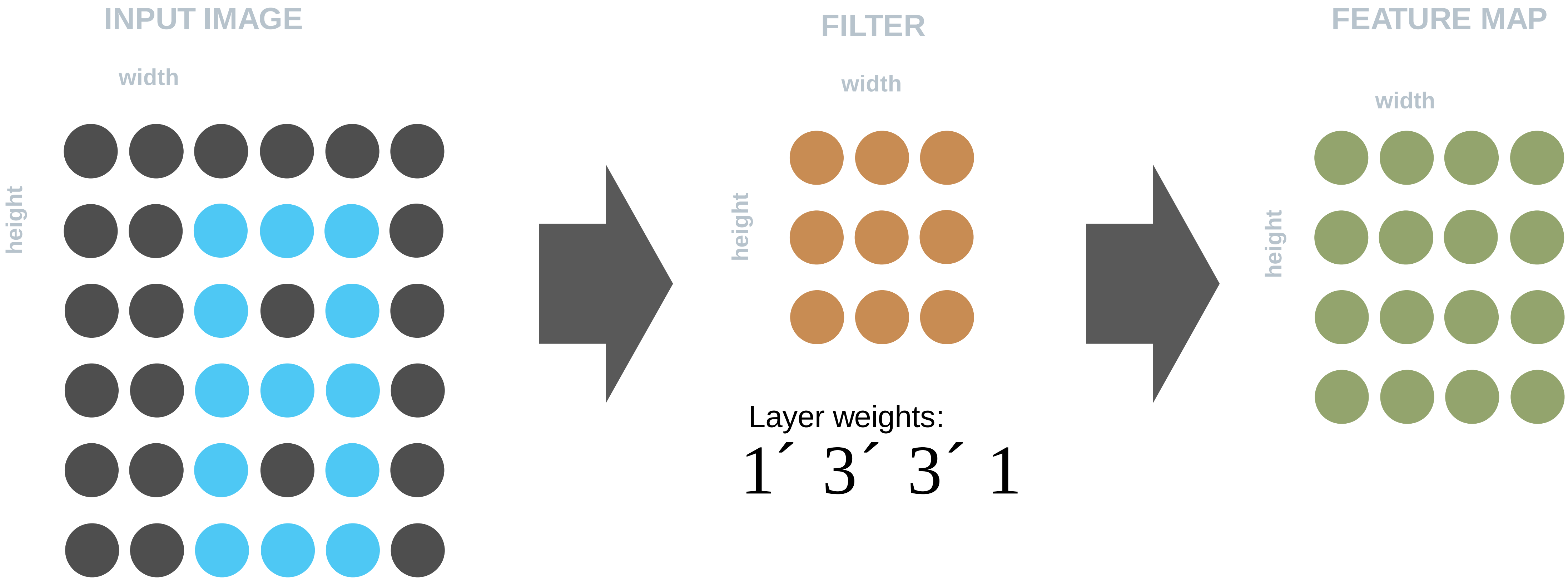
(Valid) Convolution of Filter



$$h_{u,z} = \sum_{i,j} w_{i,j} \times x_{i+u,j+z} + b$$

Basic Building Blocks

Convolutional Layer – Single Filter

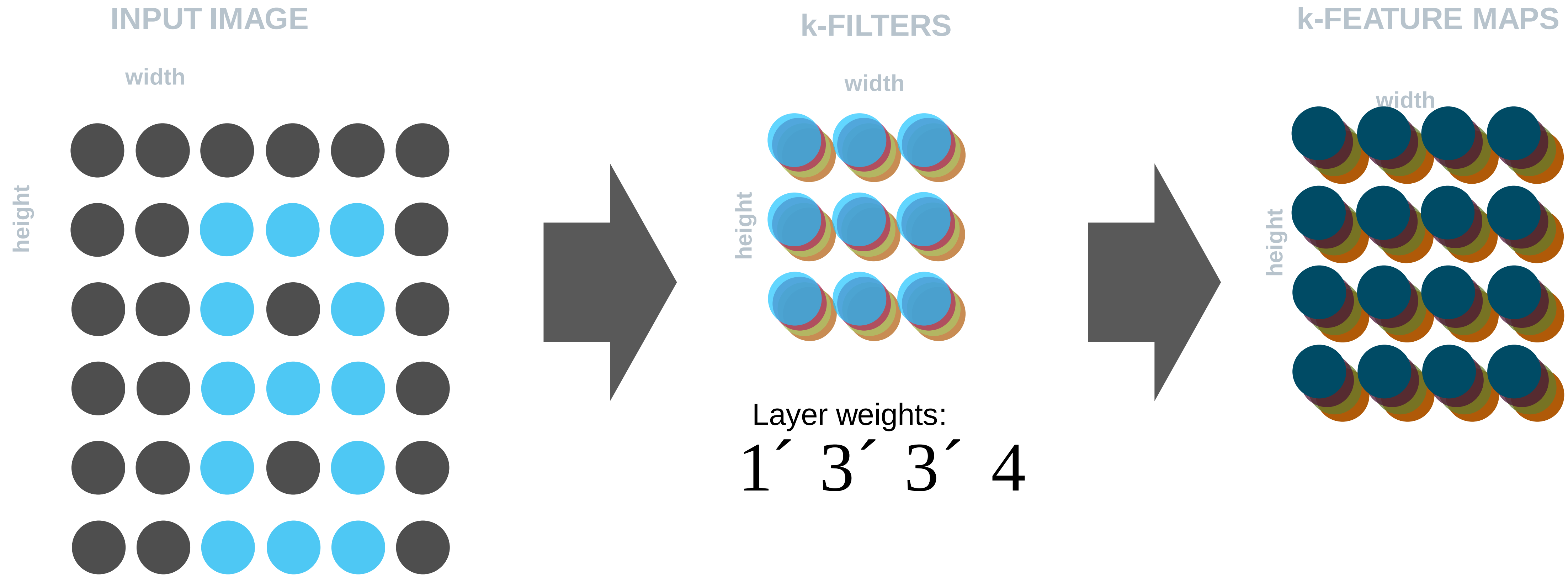


Layer weights:

1´ 3´ 3´ 1

Basic Building Blocks

Convolutional Layer – Multiple Filters

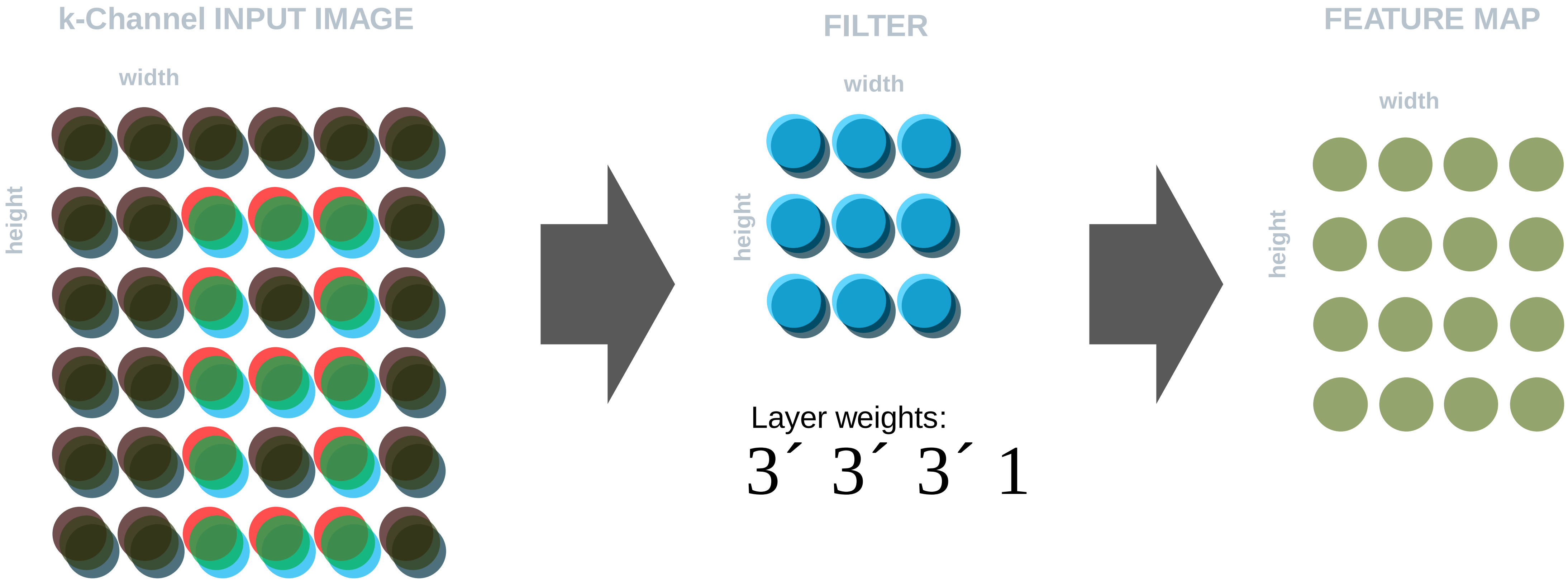


Layer weights:

1´ 3´ 3´ 4

Basic Building Blocks

Convolutional Layer – Multi-Channel Input



k-Channel INPUT IMAGE

width

height

FILTER

width

height

FEATURE MAP

width

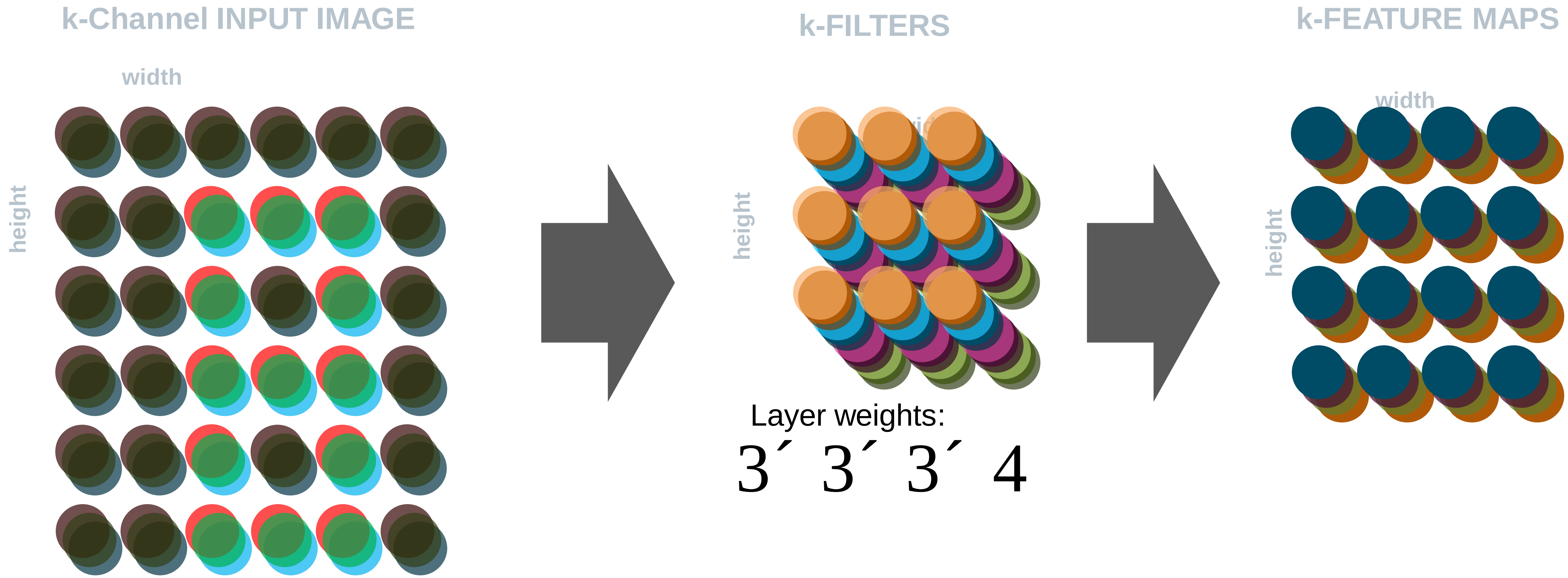
height

Layer weights:

3´ 3´ 3´ 1

Basic Building Blocks

Convolutional Layer – Multi-Channel Input

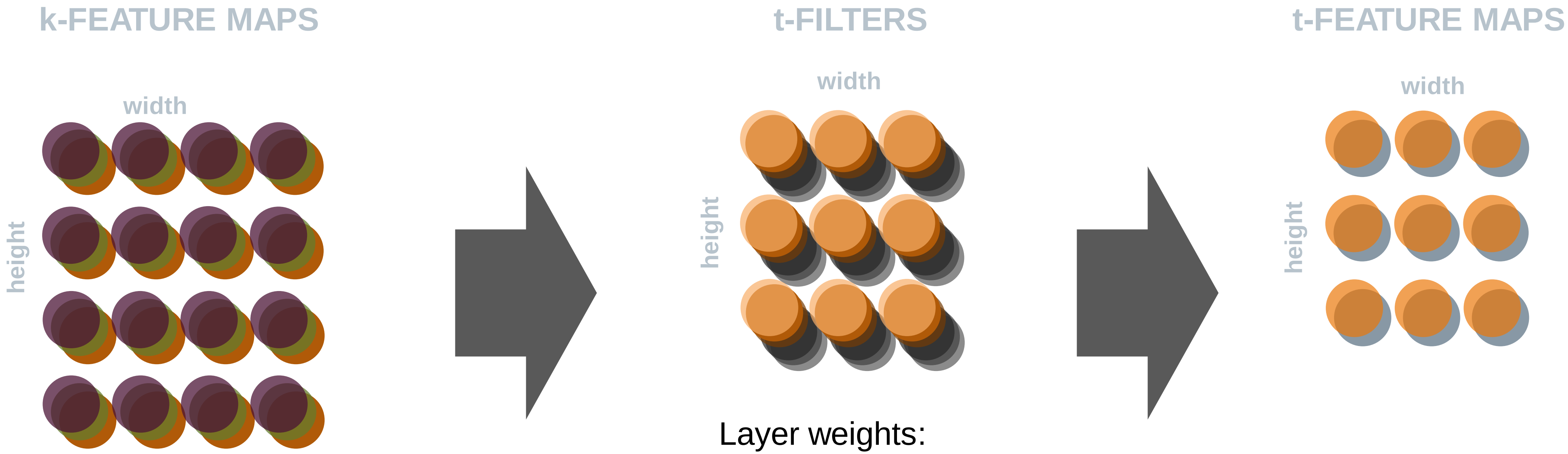


Layer weights:

3´ 3´ 3´ 4

Basic Building Blocks

Convolutional Layer - Stacking



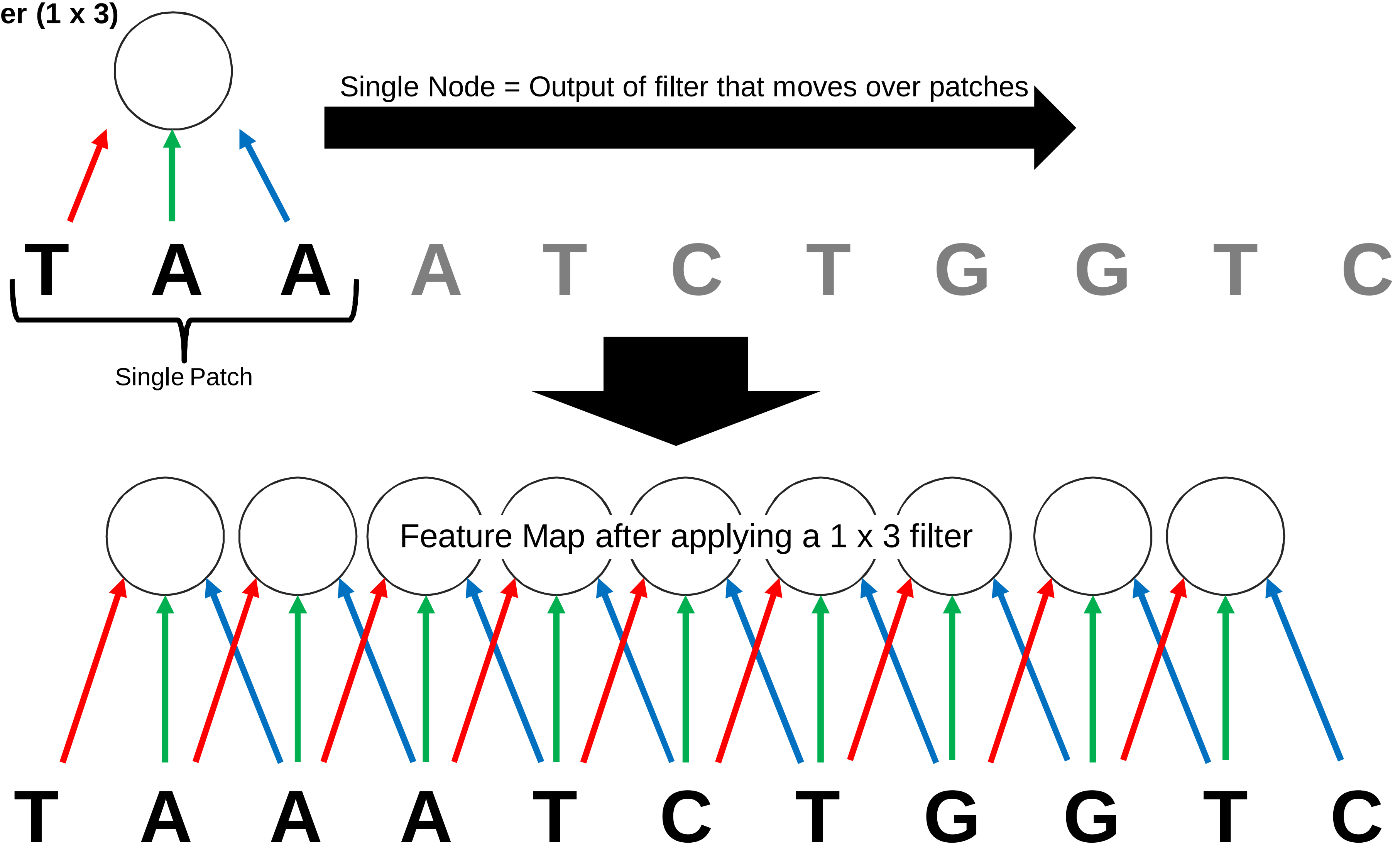
Layer weights:

k´ 3´ 3´ *t*

Basic Building Blocks

Convolutional Layer – Receptive Field Expansion

Convolutional Filter (1 x 3)

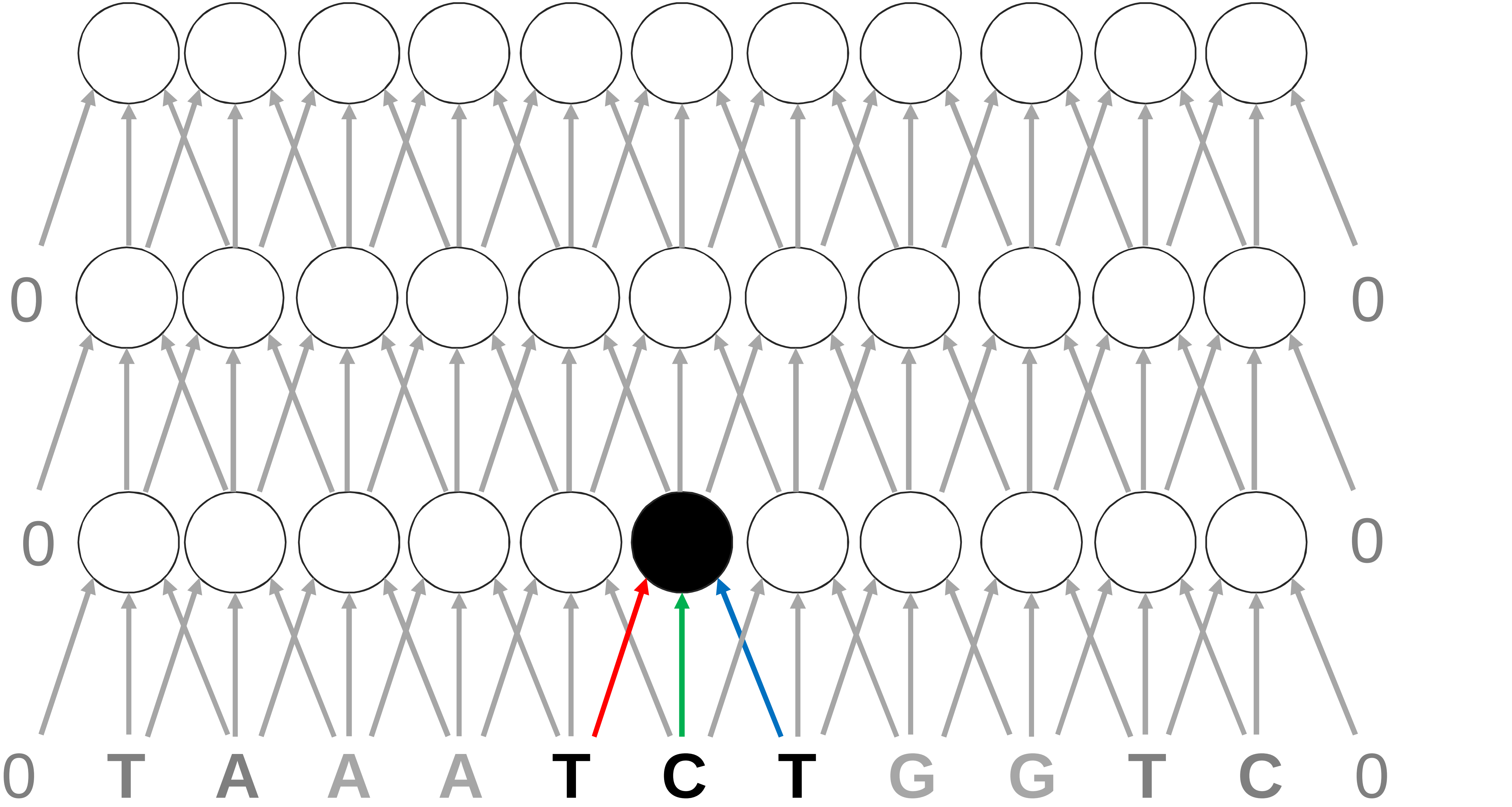


Single Node = Output of filter that moves over patches

Single Patch

Feature Map after applying a 1 x 3 filter

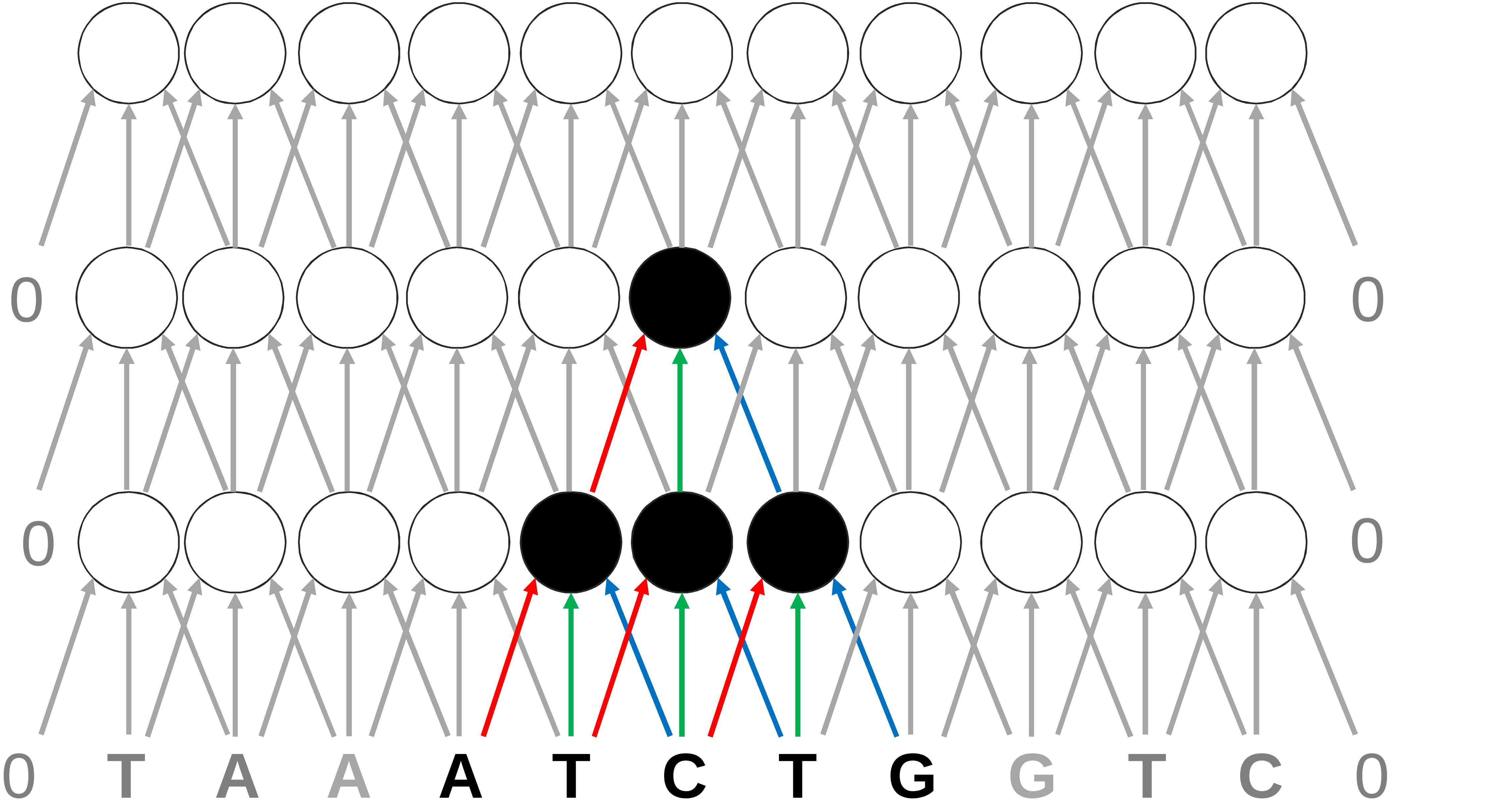
Convolutional Layer – Receptive Field Expansion.



Expansion of the receptive field for a 1 x 3 filter: 2i +1

2016/06/01

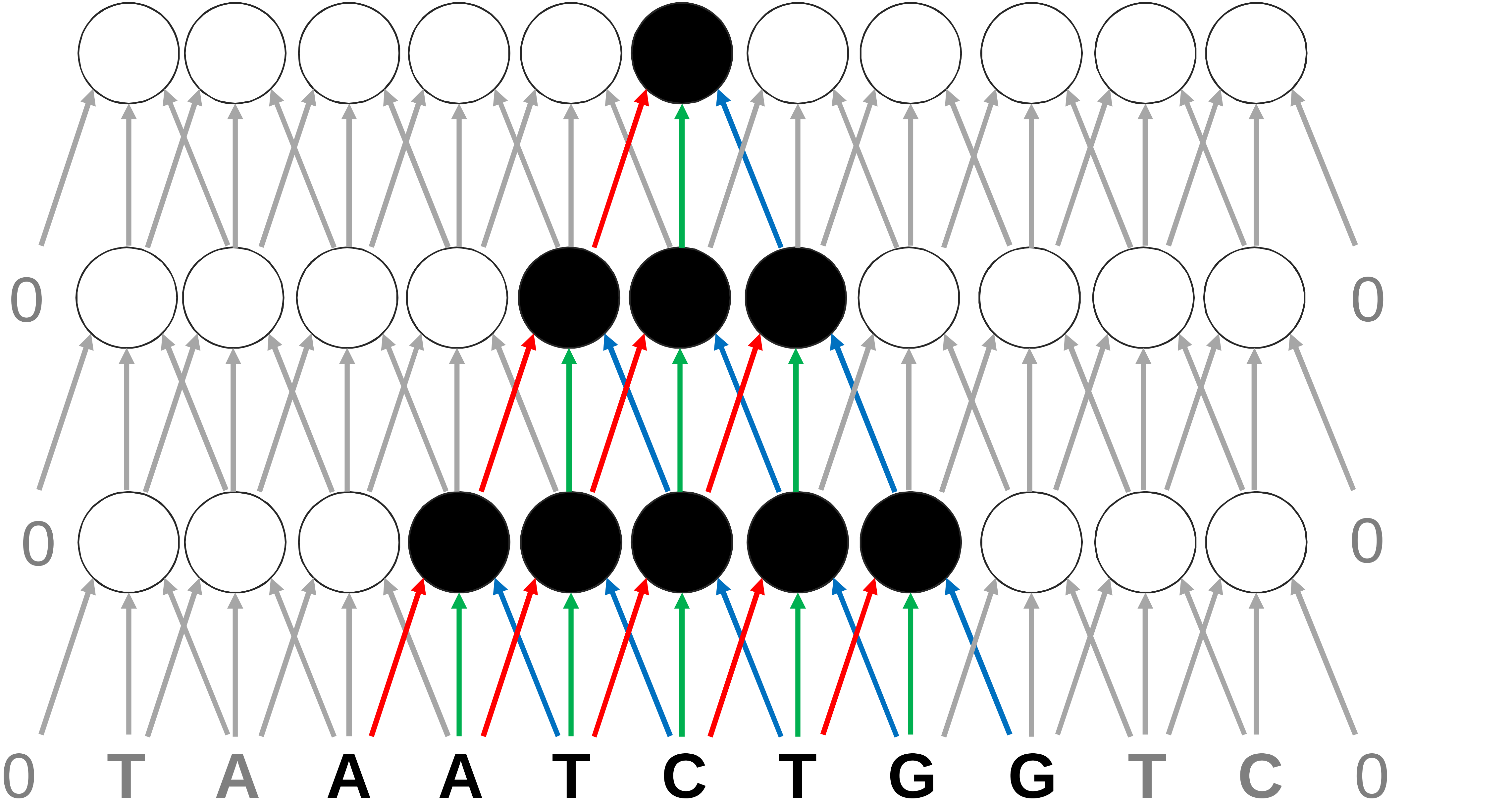
Convolutional Layer – Receptive Field Expansion.



Expansion of the receptive field for a 1 x 3 filter: 2i +1

2016/06/01

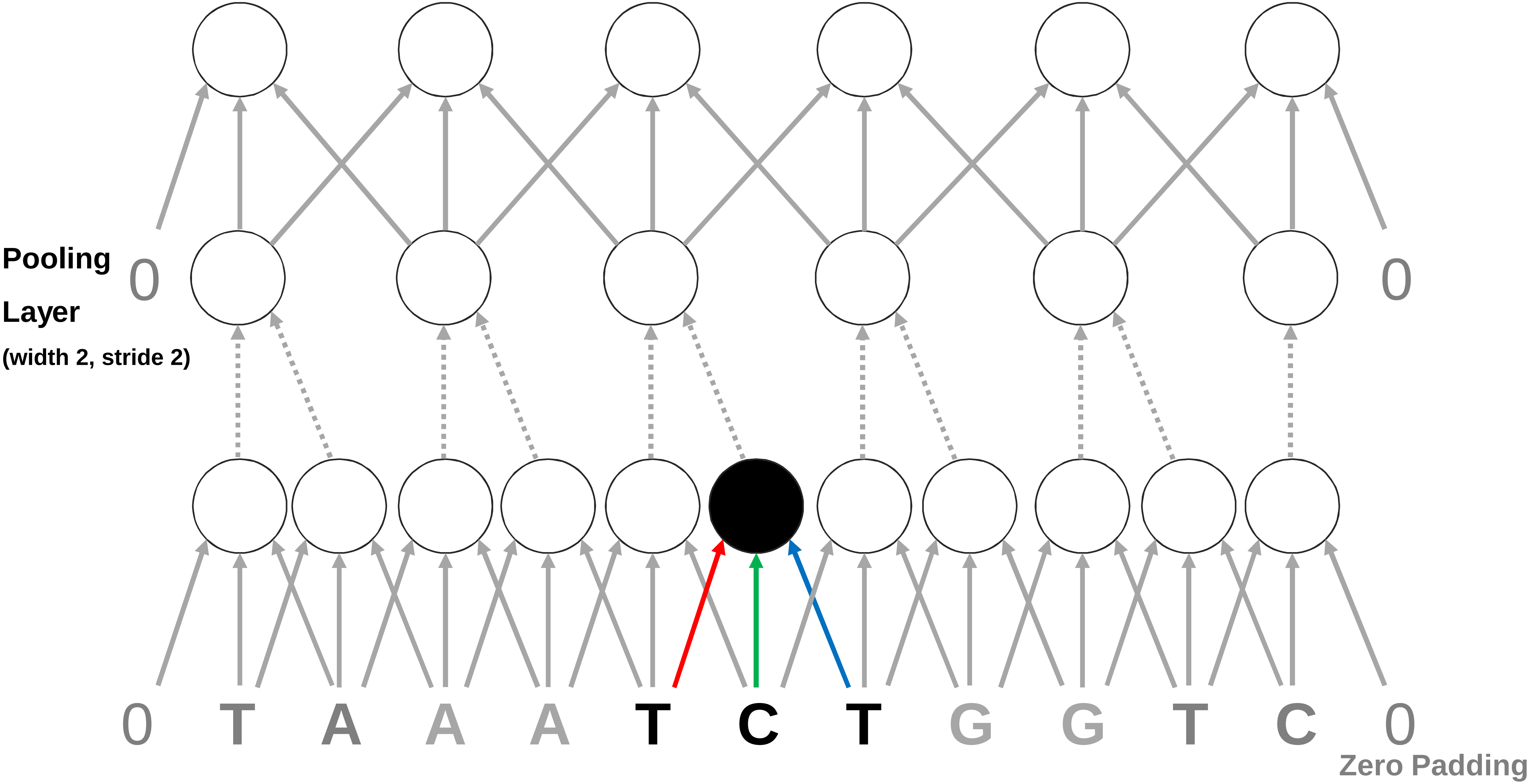
Convolutional Layer – Receptive Field Expansion.



Expansion of the receptive field for a 1 x 3 filter: 2i +1

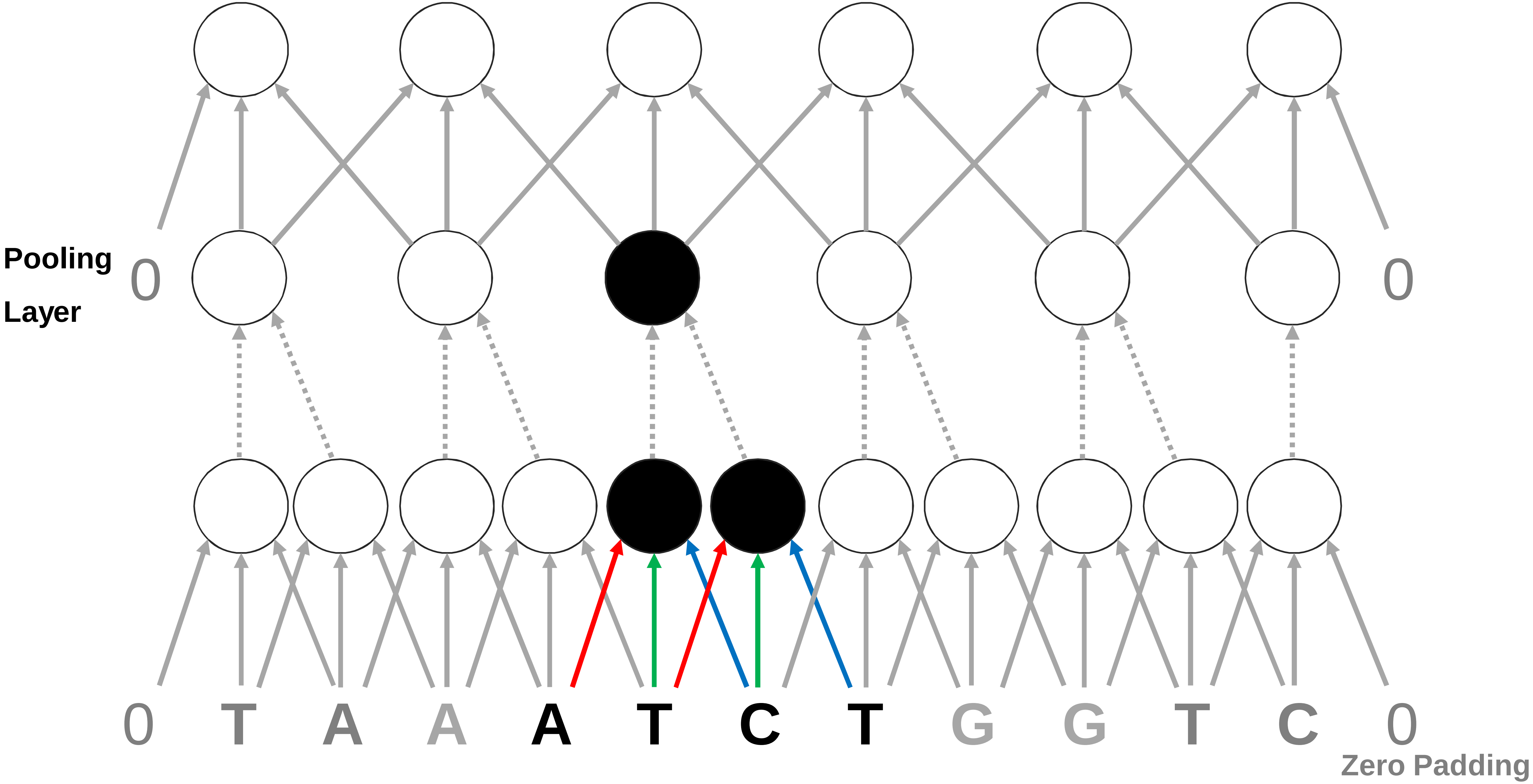
2016/06/01

Convolutional Layer – Receptive Field Expansion with Pooling Layer.



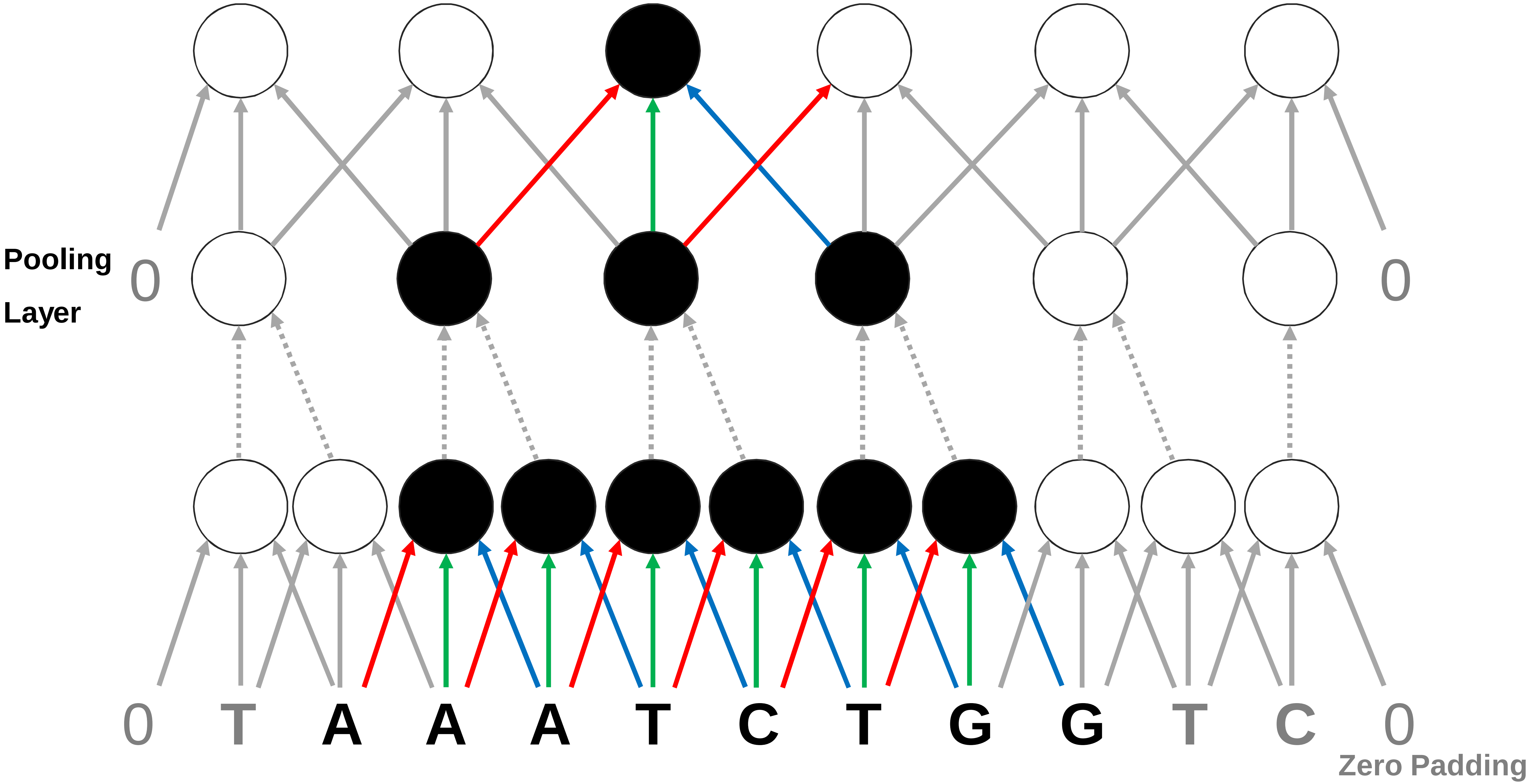
2016/06/01

Convolutional Layer – Receptive Field Expansion with Pooling Layer.



2016/06/01

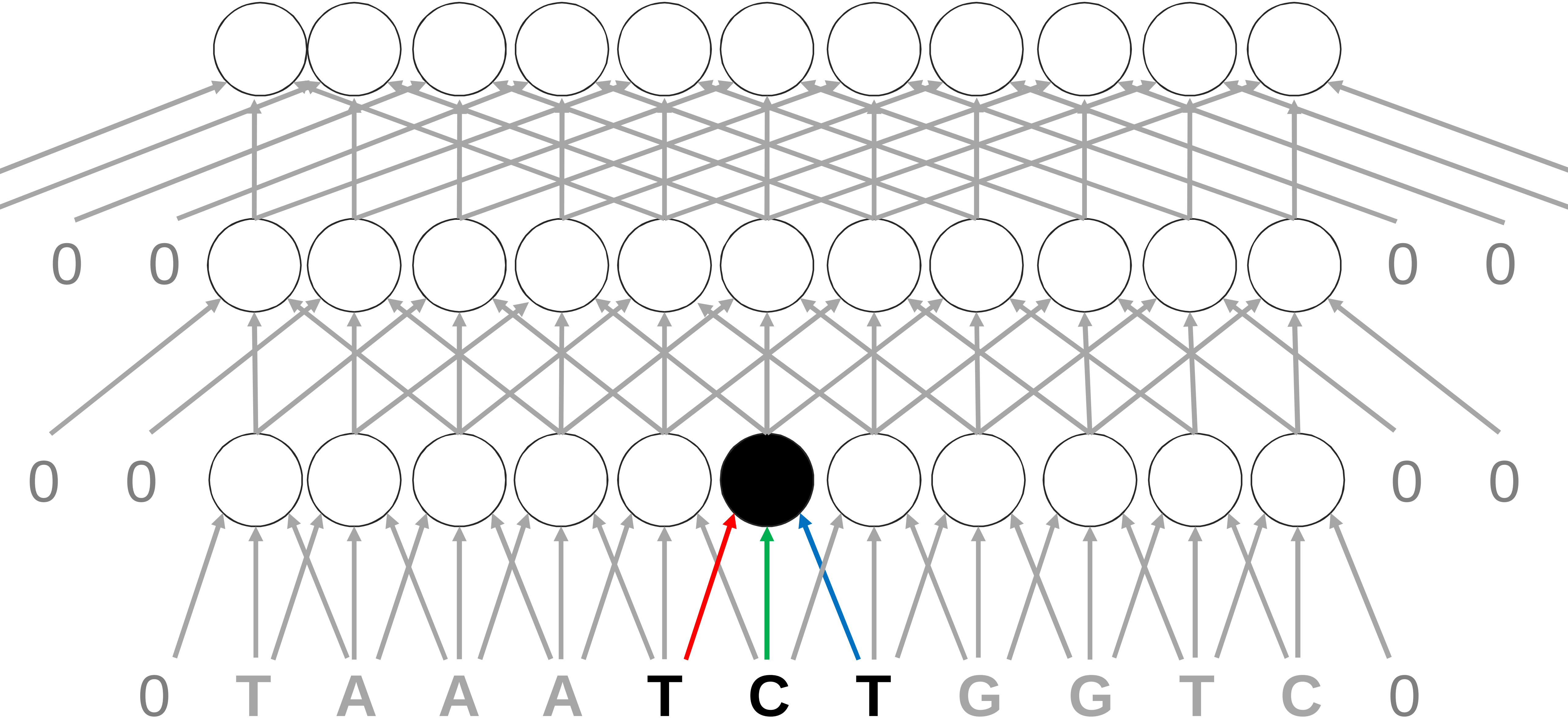
Convolutional Layer - Receptive Field Expansion with Pooling Layer.



2016/06/01

Convolutional Layer – Receptive Field Expansion with Dilation

Paper https://arxiv.org/pdf/1511.07122.pdf

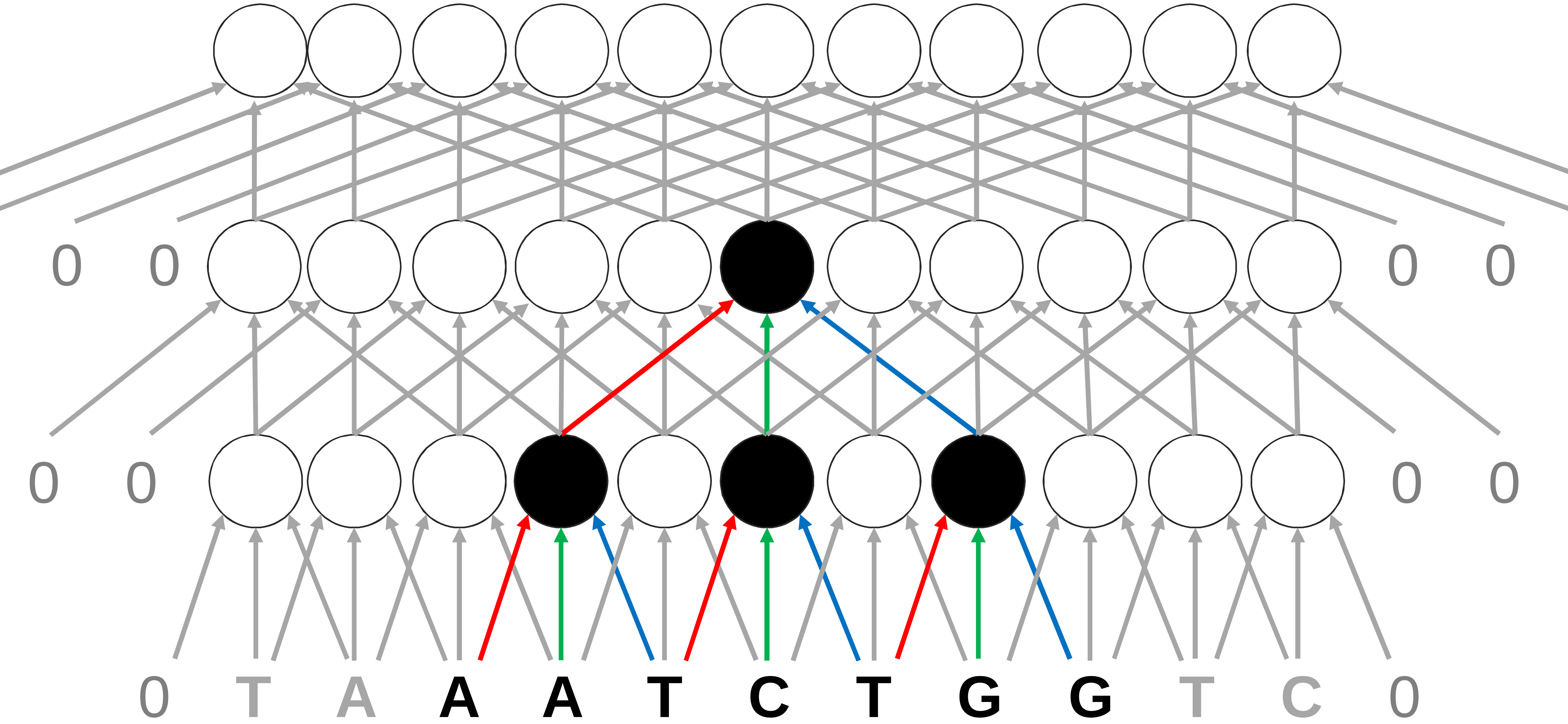


Zero Padding

Expansion of the receptive field for a 1 x 3 filter: 2^{i+1} - 1

2016/06/01

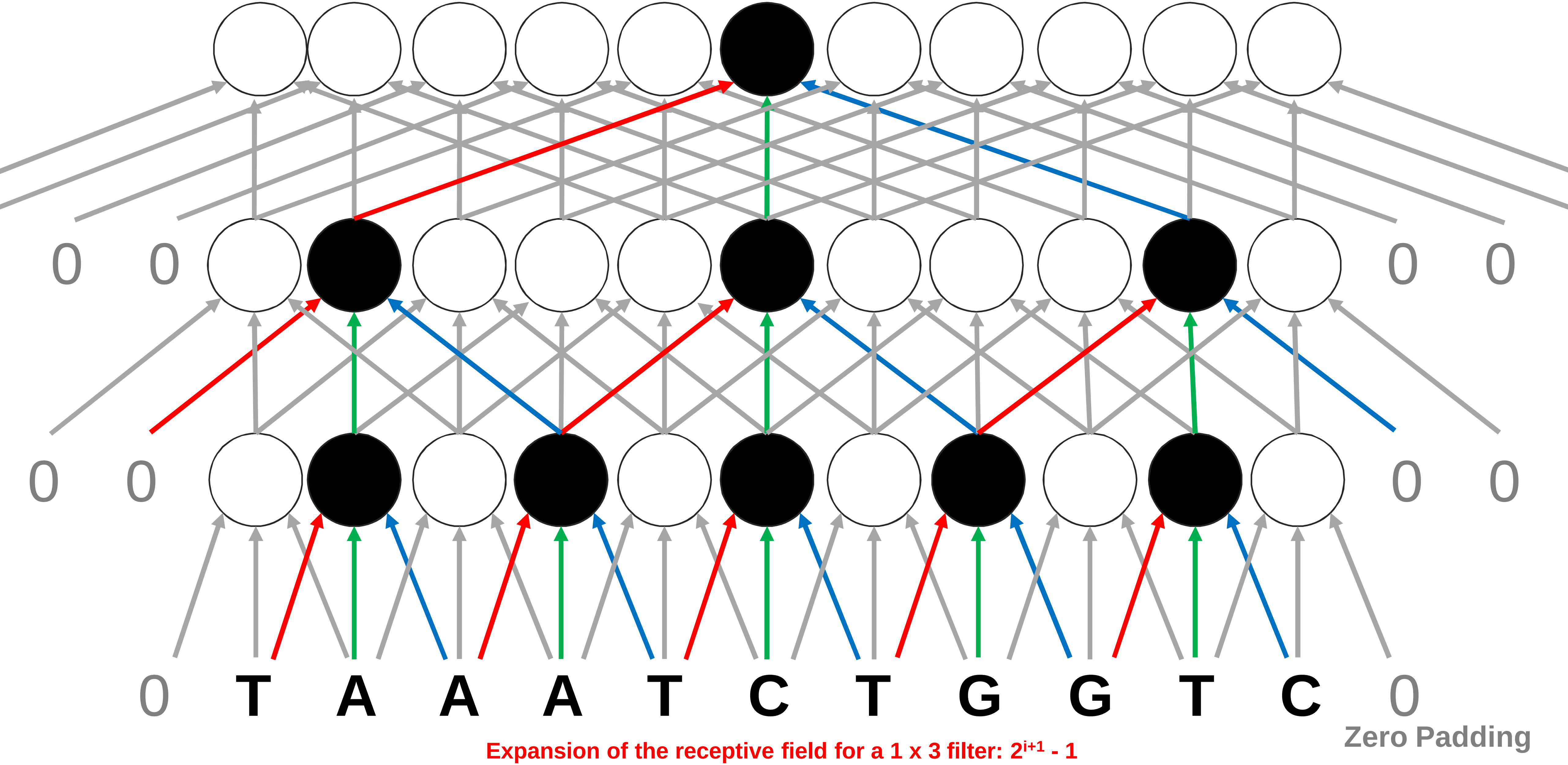
Convolutional Layer - Receptive Field Expansion with Dilation



Expansion of the receptive field for a 1 x 3 filter: $2^{i+1} - 1$

2016/06/01

Convolutional Layer - Receptive Field Expansion with Dilation

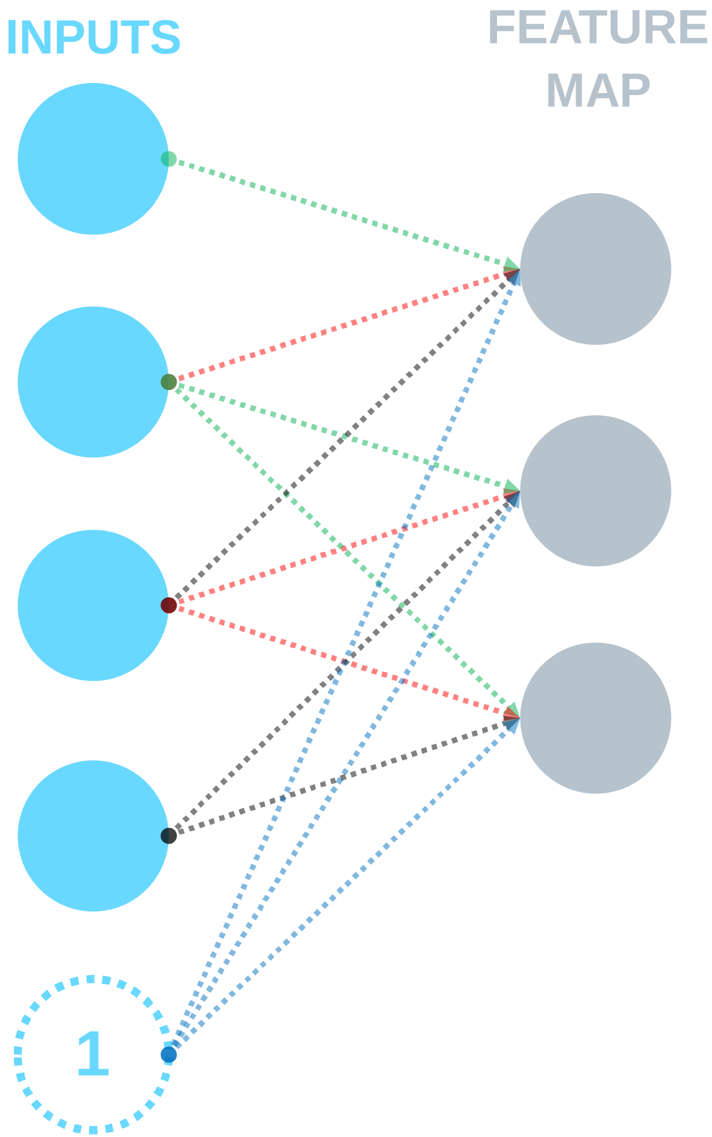


Expansion of the receptive field for a 1 x 3 filter: $2^{i+1} - 1$

2016/06/01

Basic Building Blocks

Convolutional Layer – Exploiting Neighborhood Relations

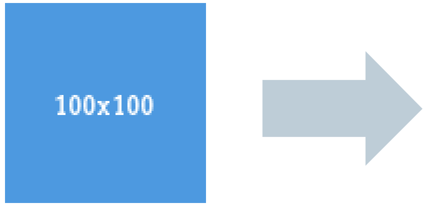


Convolutional layer:

- Exploits neighborhood relations of the inputs (e.g. spatial).
- Applies small fully connected layers to small patches of the input.
  - ØVery efficient!
  - ØWeight sharing
  - ØNumber of free parameters

    #input channels´ filter height´ filter width´ #filters
- •The receptive field can be increased by stacking multiple layers
- •Should only be used if there is a notion of neighborhood in the input:
  - •Text, images, sensor time-series, videos, …

Example:



RGB image of shape
100 x 100 x 3

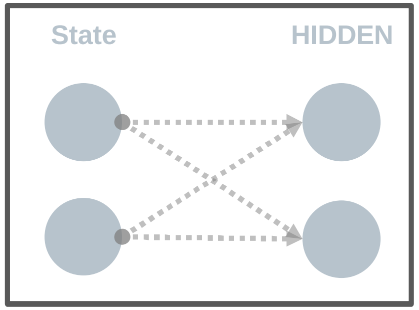
2,700 free parameters for a convolutional layer with 100 hidden units (filters) with a filter size of 3 x 3!

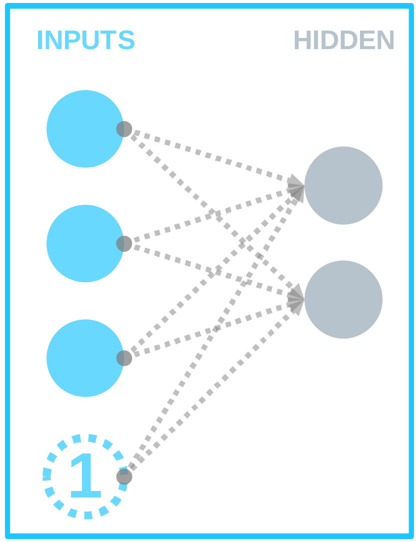
Basic Building Blocks

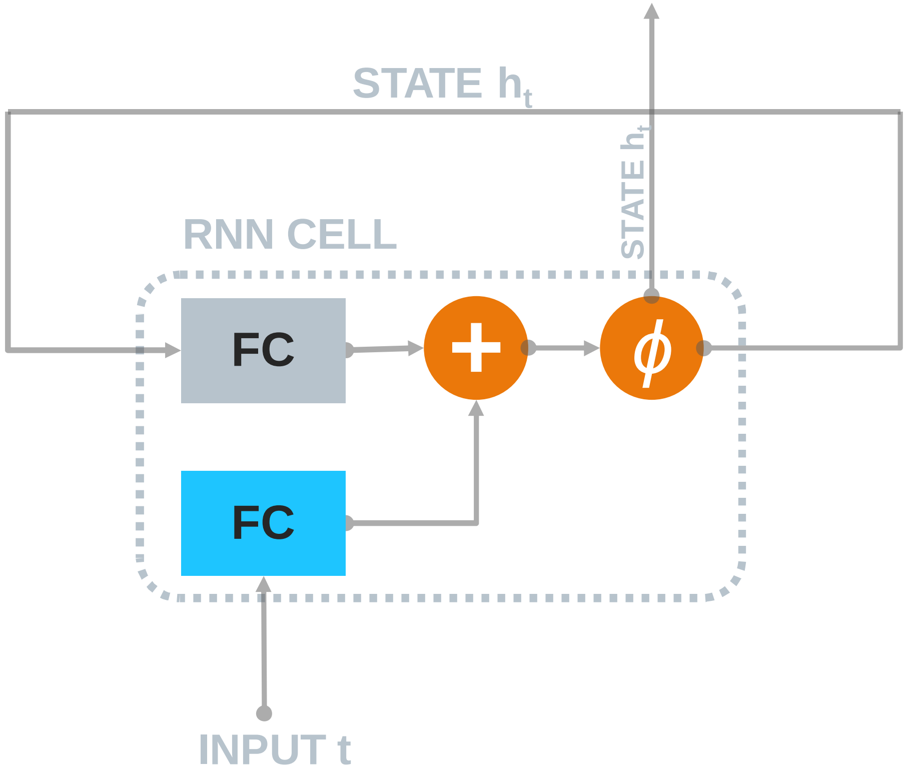
Recurrent Neural Network Layer – The RNN cell

FC = Fully connected layer
+ = Addition
Φ = Activation function

$$h_t = f\,(\boxed{Uh_{t-1}} + \boxed{Wx_t + b})$$



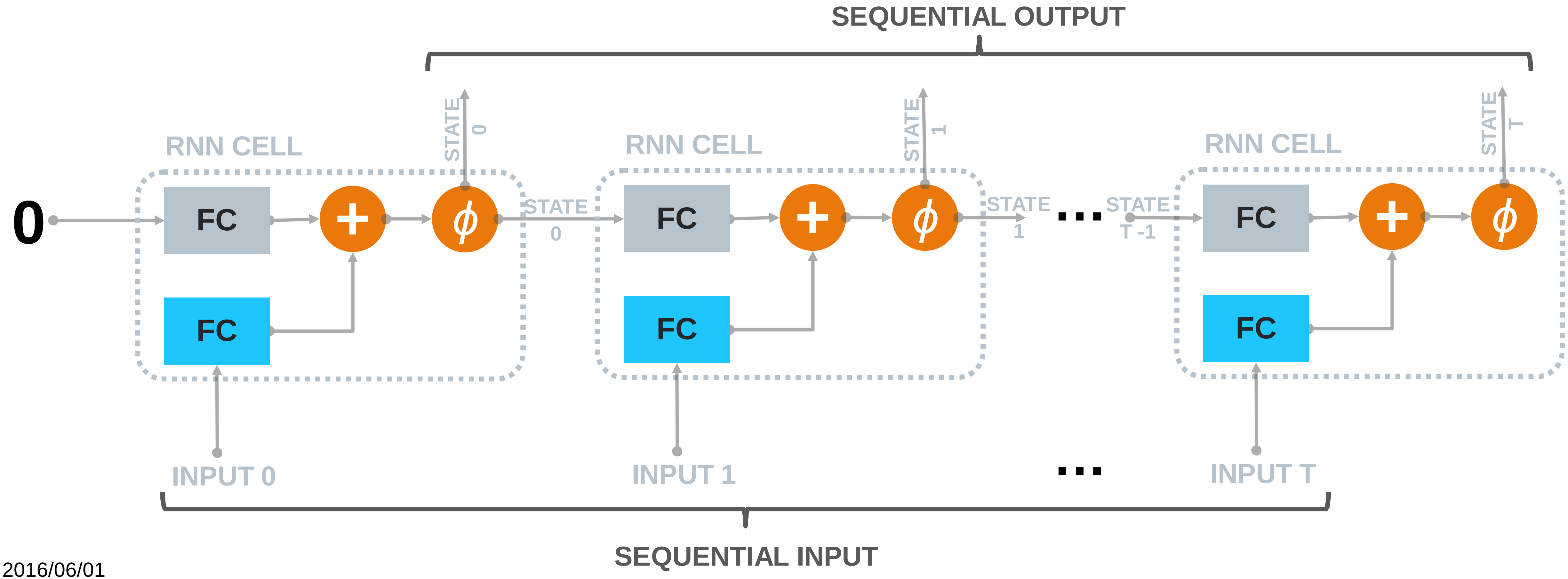




Basic Building Blocks

Recurrent Neural Network layer – Unfolded

FC = Fully connected layer
+ = Addition
Φ = Activation function



2016/06/01

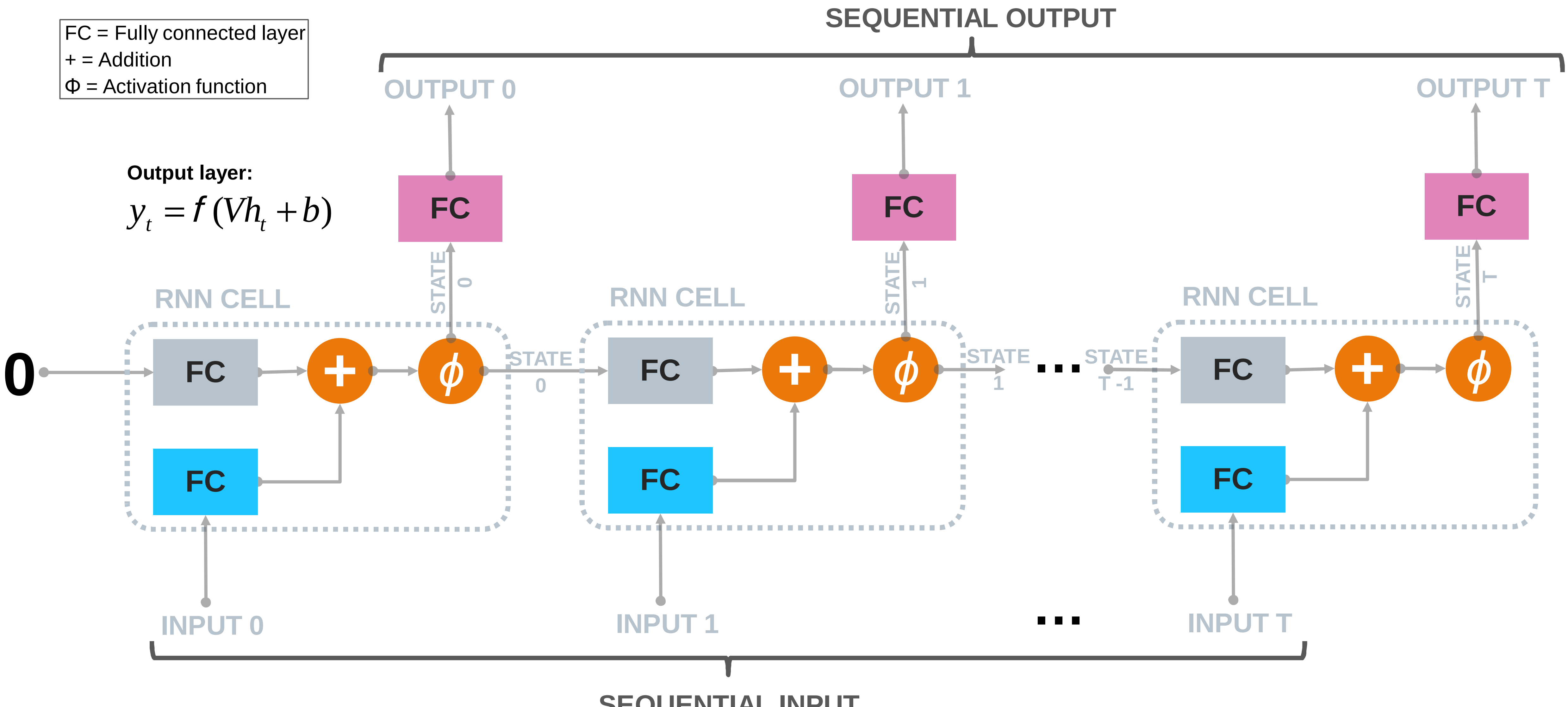
Basic Building Blocks

Vanilla Recurrent Neural Network (unfolded)

FC = Fully connected layer
+ = Addition
Φ = Activation function

Output layer:

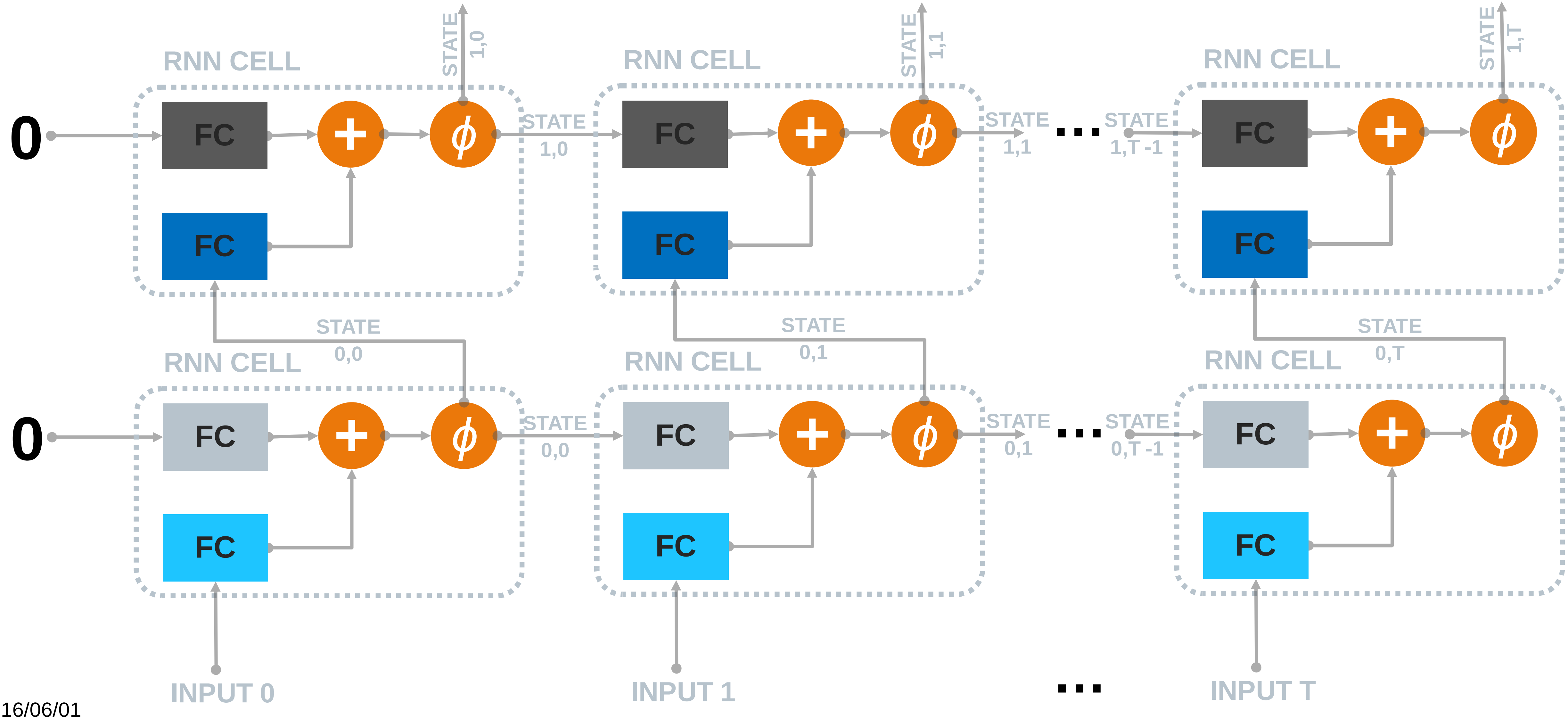
$$y_t = f\,(Vh_t + b)$$



2016/06/01

Basic Building Blocks

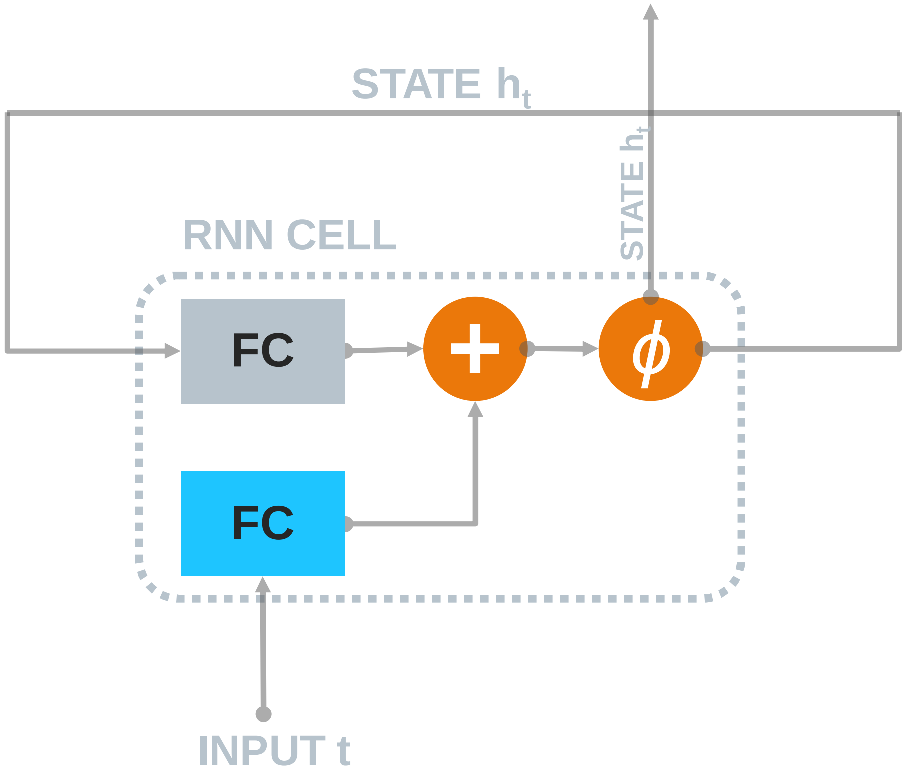
Recurrent Neural Network Layer – Stacking



2016/06/01

Basic Building Blocks

Recurrent Neural Network Layer – Exploiting Sequential Relations



RNN layer:

• Exploits sequential dependencies (Next prediction might depend on things that were observed earlier).

•Applies the same (via parameter sharing) fully connected layer to each step in the input data and combines it with collected information from the past (hidden state).

ØDirectly learns sequential (e.g. temporal) dependencies.

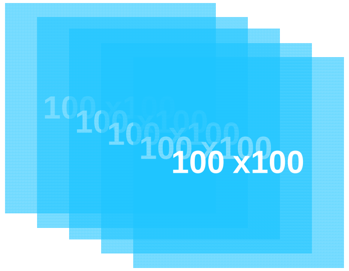
•Stacking can help to learn deeper hierarchical state representations.

•Should only be used if sequential sweeping of the data makes sense: Text, sensor time-series, (videos, images)…

•Vanilla RNN is not able to capture long-time dependencies!

•Use with care since it can also quickly over-parameterize the model

ØCan lead to degenerated solutions.



e.g. Videos of frames of shape100 x 100 x 3

2016/06/01

Deep Learning

Thinking in Macro Structures

Thinking in Macro Structures

Remember the Important Things – And Move On

Important things:

- Purpose
- Weaknesses
- General usage
- Tweaks

Fully Connected Layer	**FC**	+	In case no assumptions on the input data can be exploited. (Treat all inputs as independent)
Convolutional Layer	**CNN**	+	Good for exploiting spatial/sequential dependencies in the data.
Recurrent Neural Network Layer	**RNN**	+	Good for modeling sequential data with no long term dependencies

With these three basic building blocks, we are already able to do amazing stuff!

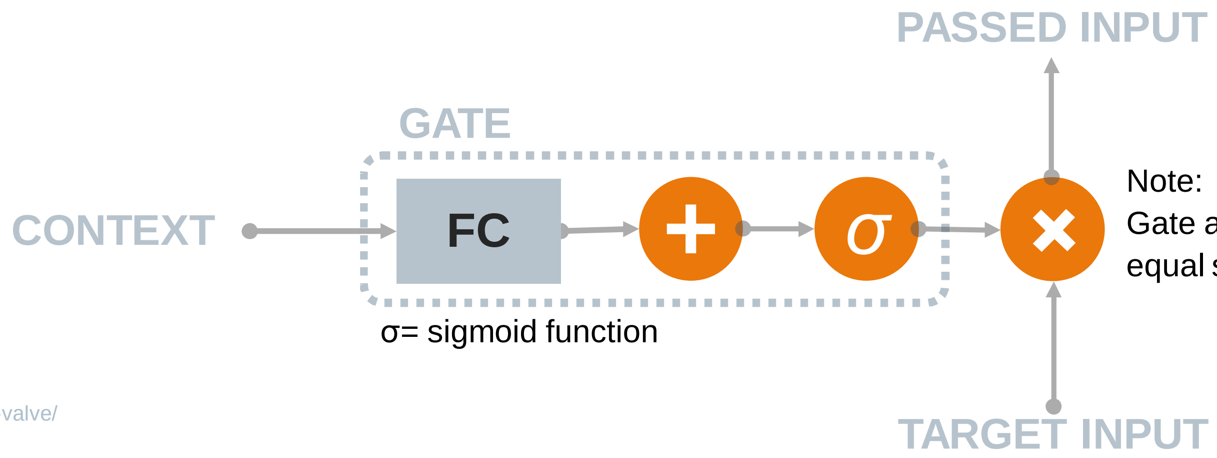
Thinking in Macro Structures
Mixing Things Up – Generating Purpose Modules.

- Given the basic building blocks introduced in the last section:
  - We can construct modules that address certain sub-task within the model that might be beneficial for reaching the actual target goal.
    - E.g. Gating, Attention, Hierarchical feature extraction, …
  - These modules can further be combined to form even larger modules serving a more complex purpose
    - LSTMs, Residual Units, Fractal Nets, Neural memory management …
  - Finally all things are further mixed up to form an architecture with many internal mechanisms that enables the model to learn very complex tasks end-to-end.
    - Text translation, Caption generation, Neural Computer…

Thinking in Macro Structures

Controlling the Information Flow – Gating

Image: https://au.pinterest.com/explore/tap-valve/



σ= sigmoid function

Note:
Gate and input have equal shapes

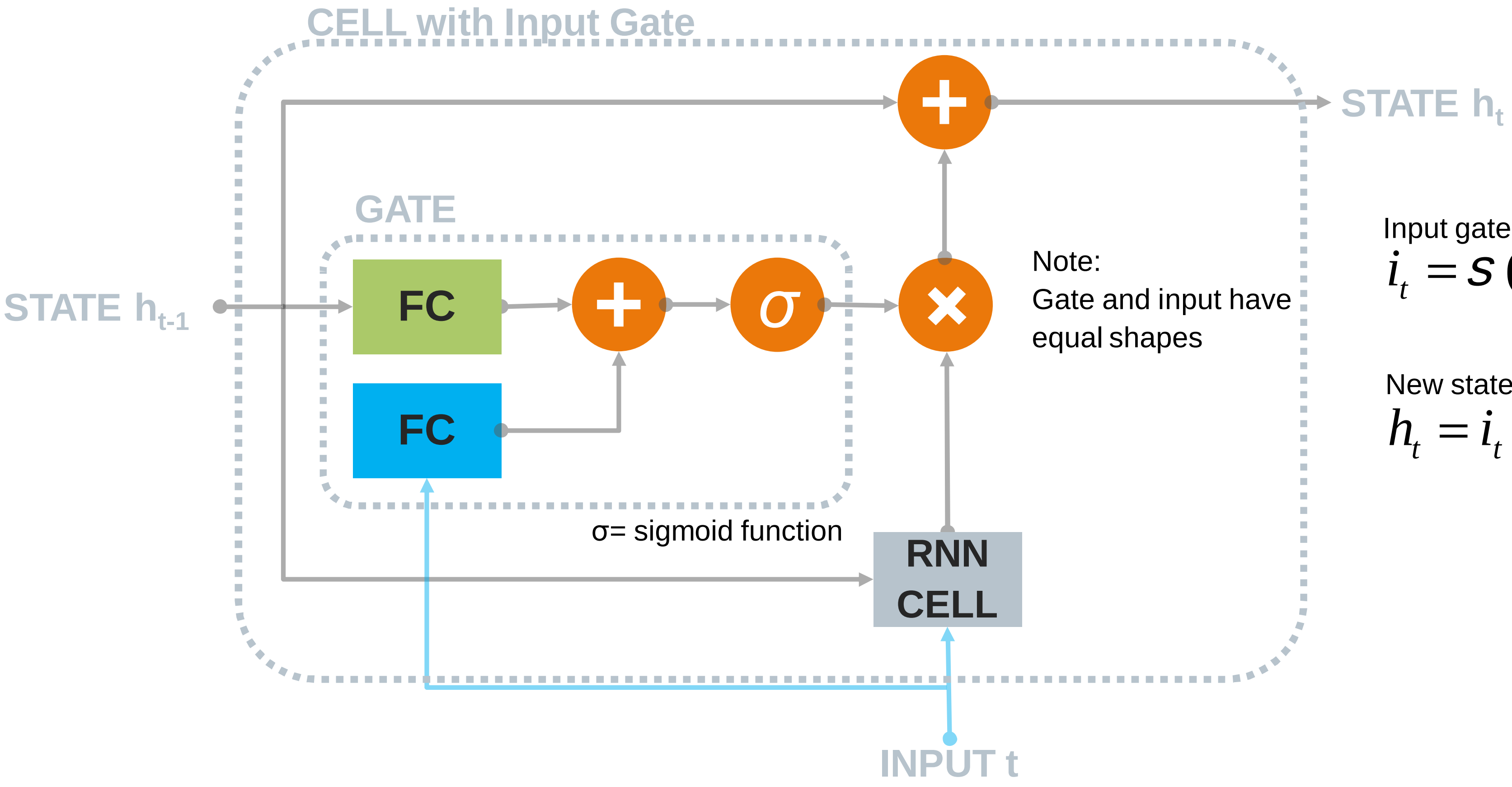
2016/06/01

Thinking in Macro Structures

Controlling the Information Flow – Gating in Recurrent Neural Network Cells

Goal: Control how much information from the current input impacts the hidden state representation.

Note: This made up example cell shows only the principle but would not work in practice since we would also need to control the information flow from the previous state representation to the next (forget gate).



Note:
Gate and input have equal shapes

σ= sigmoid function

Input gate:

$$i_t = s\,(W_i x_t + U_i h_{t-1} + b_i)$$

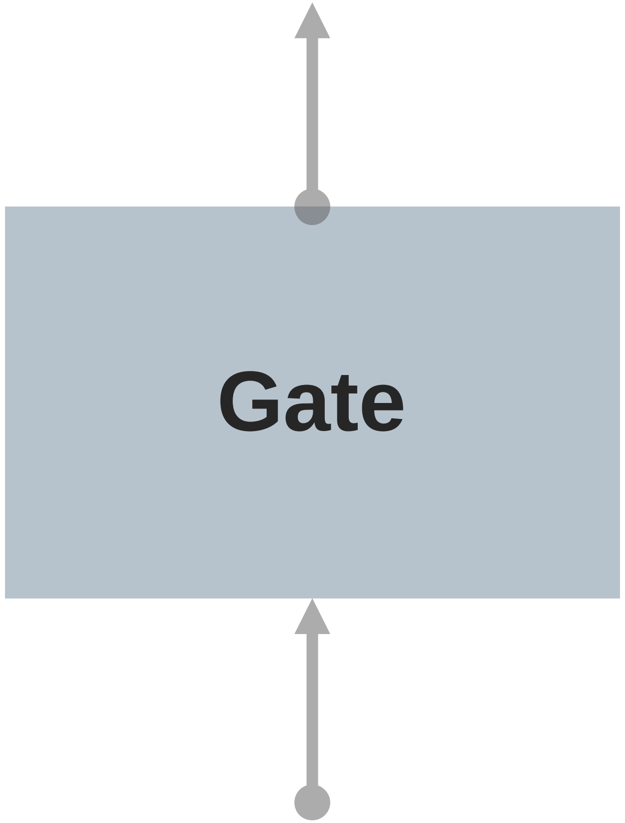
New state representation

$$h_t = i_t \times f\,(W_{RNN}\, x_t + U_{RNN}\, h_{t-1} + b_{RNN})$$

2016/06/01

Thinking in Macro Structures

Remember the Important Things – And Move On.



+ Good for controlling information flow in a network

Thinking in Macro Structures

Learning Long-Term Dependencies – The LSTM Cell

Forget gate $f_t = s\,(W_t x_t + U_f h_{t-1} + b_f)$

Input gate $i_t = s\,(W_t x_t + U_i h_{t-1} + b_i)$

$c_t = f_t \times c_{t-1} + i_t \times f\,(W_c x_t + U_c h_{t-1} + b_c)$

Output gate $o_t = s\,(W_o x_t + U_o h_{t-1} + b_o)$

$h_t = o_t \times c_t$

Thinking in Macro Structures

Learning Long-Term Dependencies – The LSTM Cell

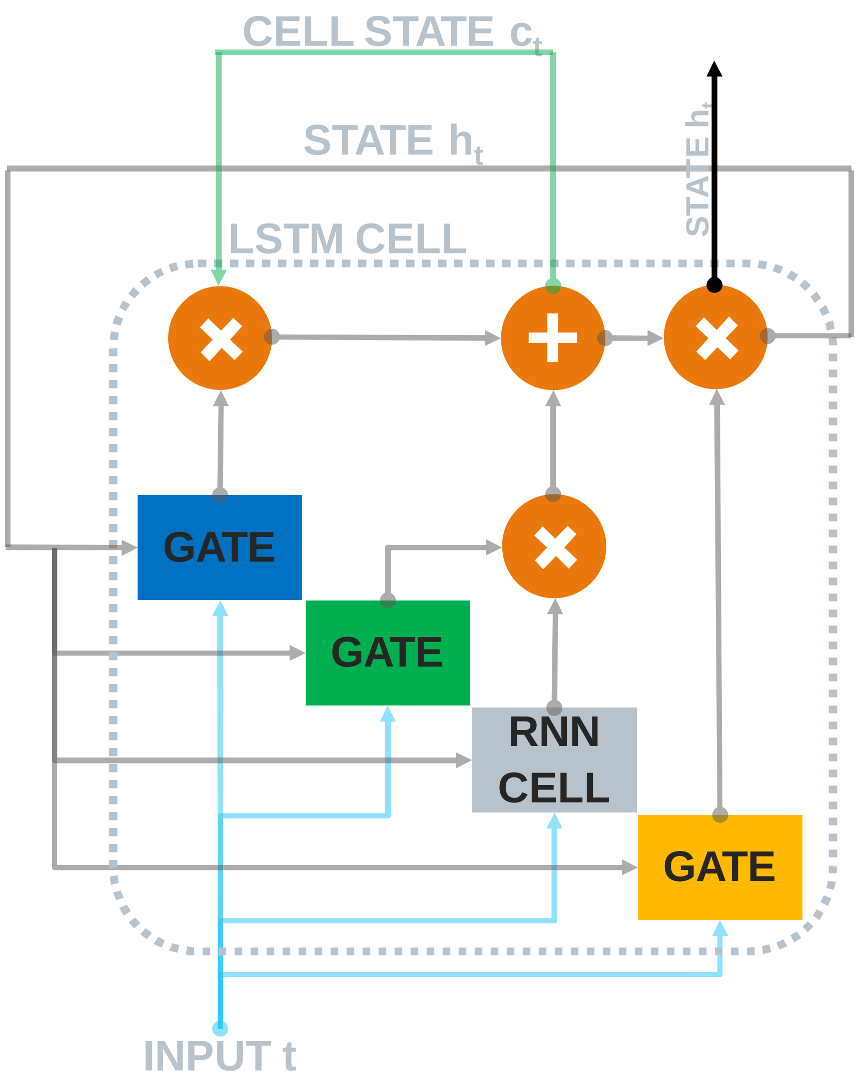
Forget gate $f_t = \sigma(W_t x_t + U_f h_{t-1} + b_f)$

Input gate $i_t = \sigma(W_t x_t + U_i h_{t-1} + b_i)$

$c_t = f_t \times c_{t-1} + i_t \times \phi(W_c x_t + U_c h_{t-1} + b_c)$

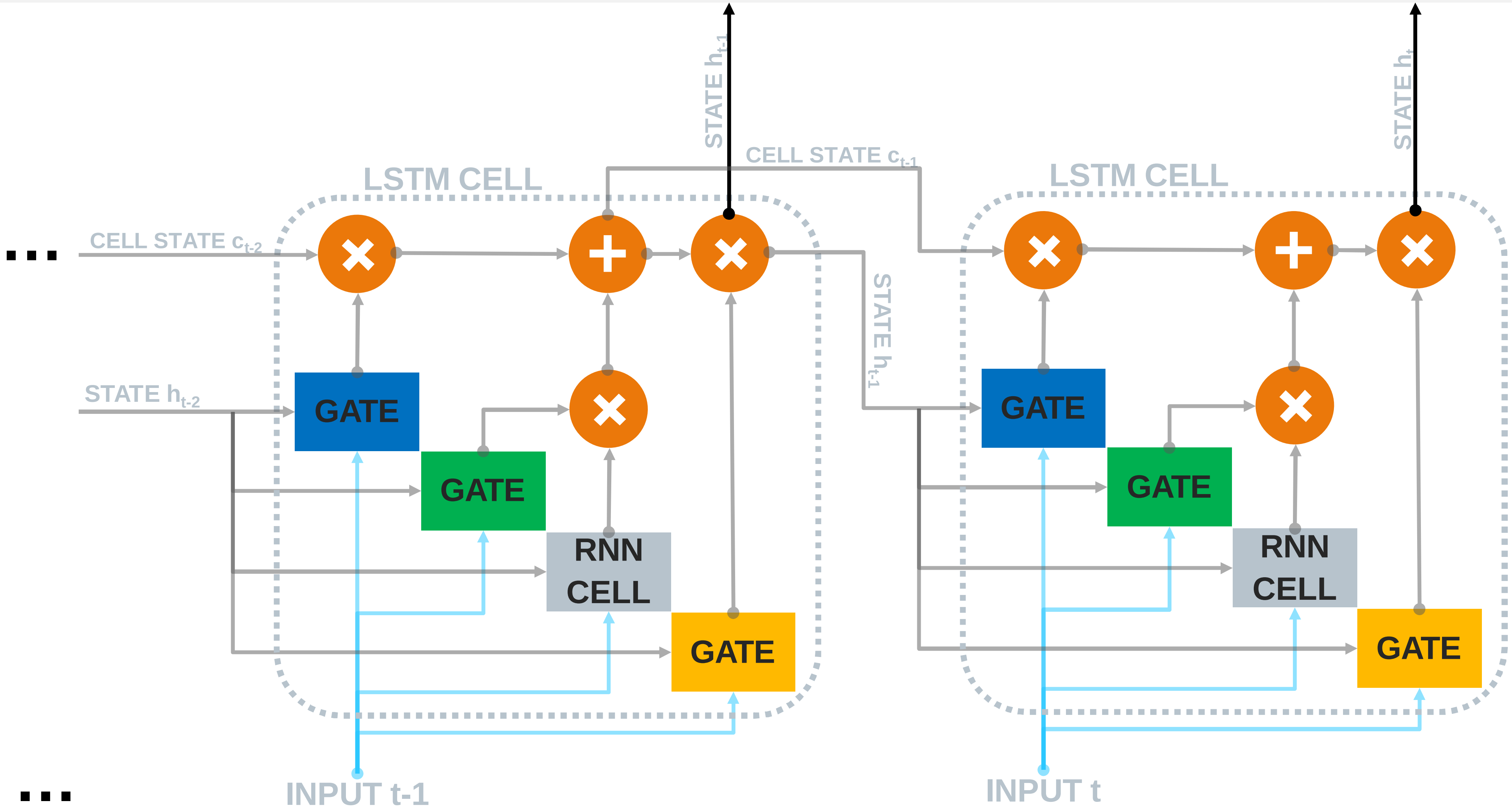
Output gate $o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$

$h_t = o_t \times c_t$



Thinking in Macro Structures

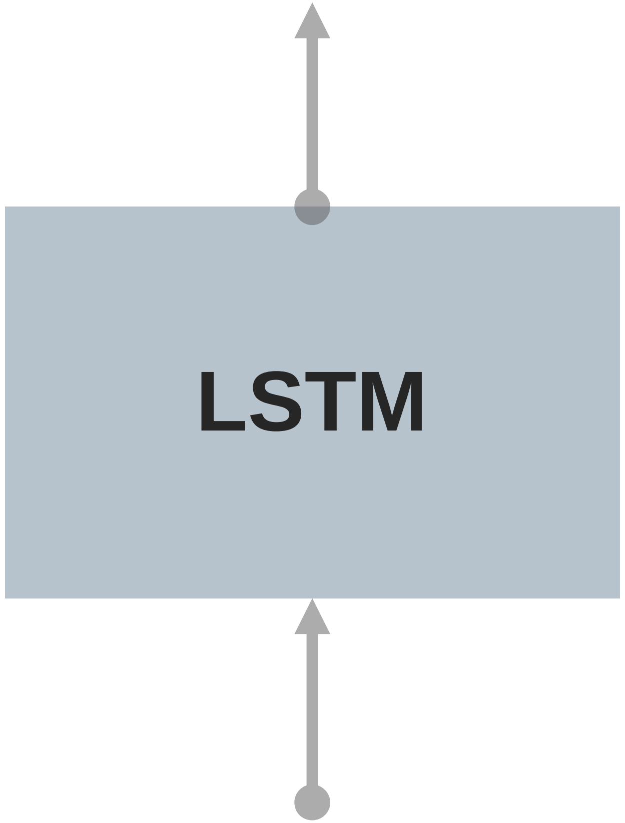
Learning Long-Term Dependencies – The LSTM Cell



2016/06/01

Thinking in Macro Structures

Remember the Important Things – And Move On.



LSTM

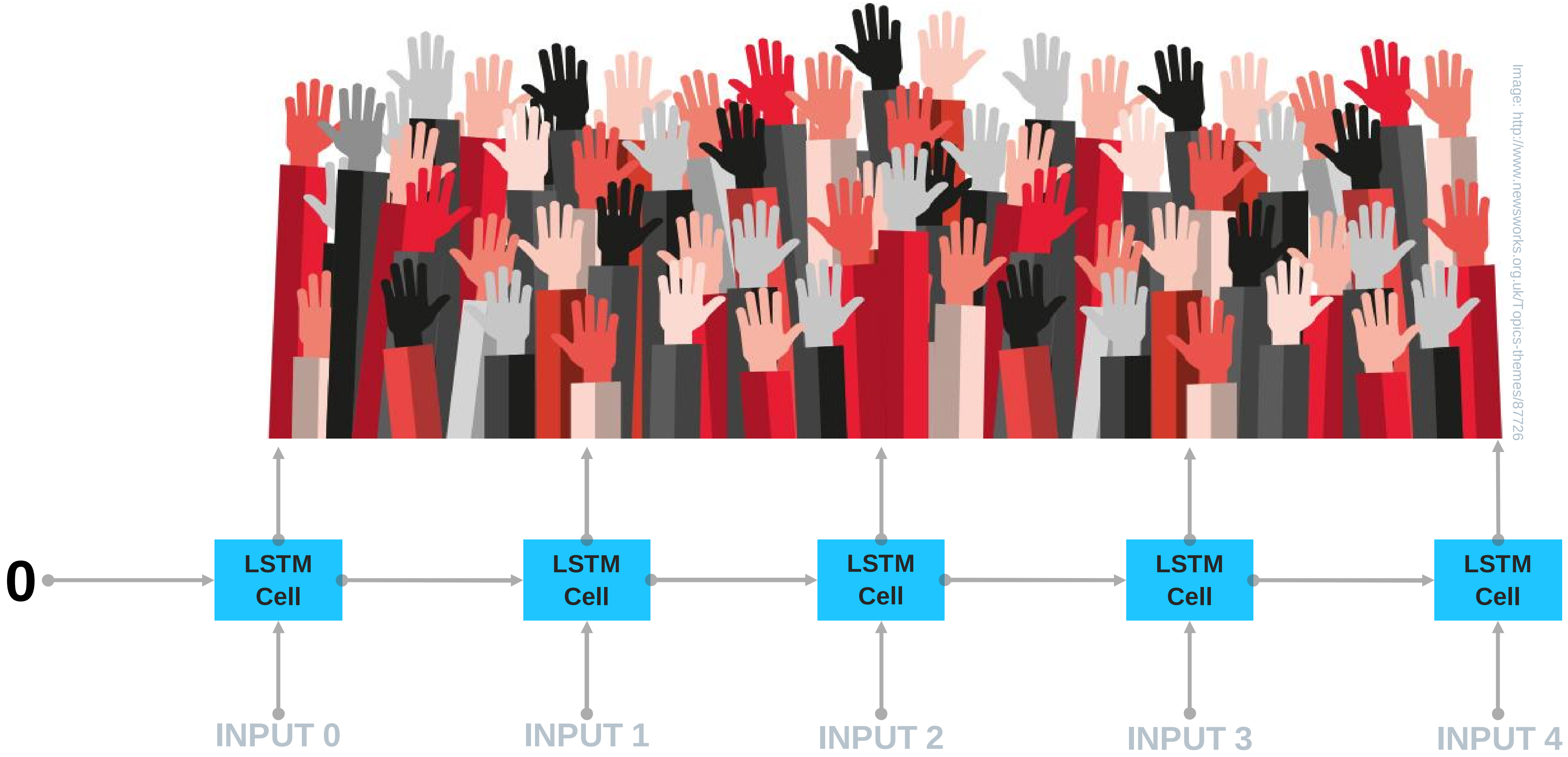
+ Good for modeling long term dependencies in sequential data

PS: Same accounts for **Gated Recurrent Units**

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

Thinking in Macro Structures

Learning to Focus on the Important Things – Attention



0

LSTM Cell

LSTM Cell

LSTM Cell

LSTM Cell

LSTM Cell

INPUT 0

INPUT 1

INPUT 2

INPUT 3

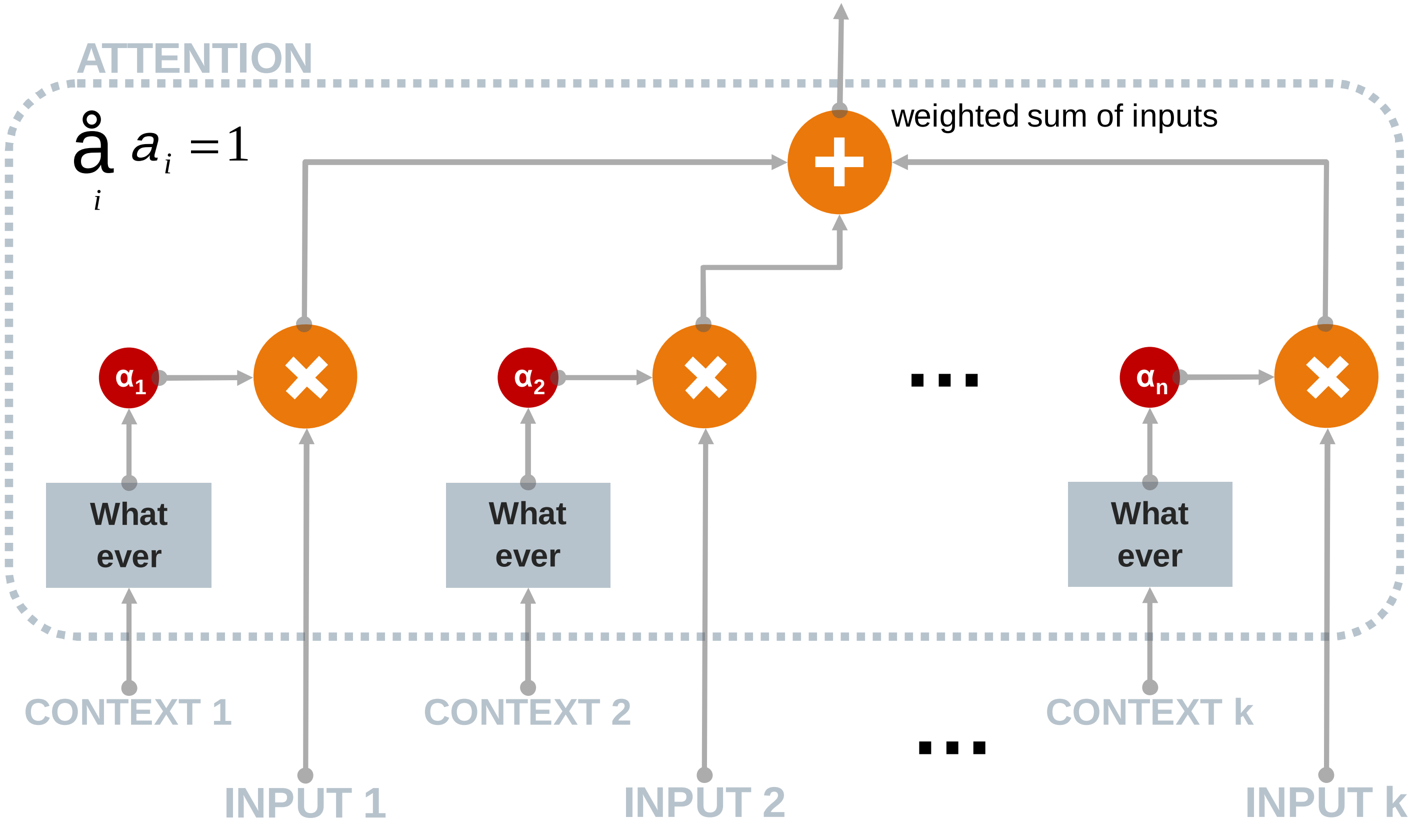
INPUT 4

Image: http://www.newsworks.org.uk/Topics-themes/87726

Thinking in Macro Structures

Learning to Focus on the Important Things – Attention

What ever = any function that maps some input to a scalar. Often a multi layer neural network that is learned with the rest.

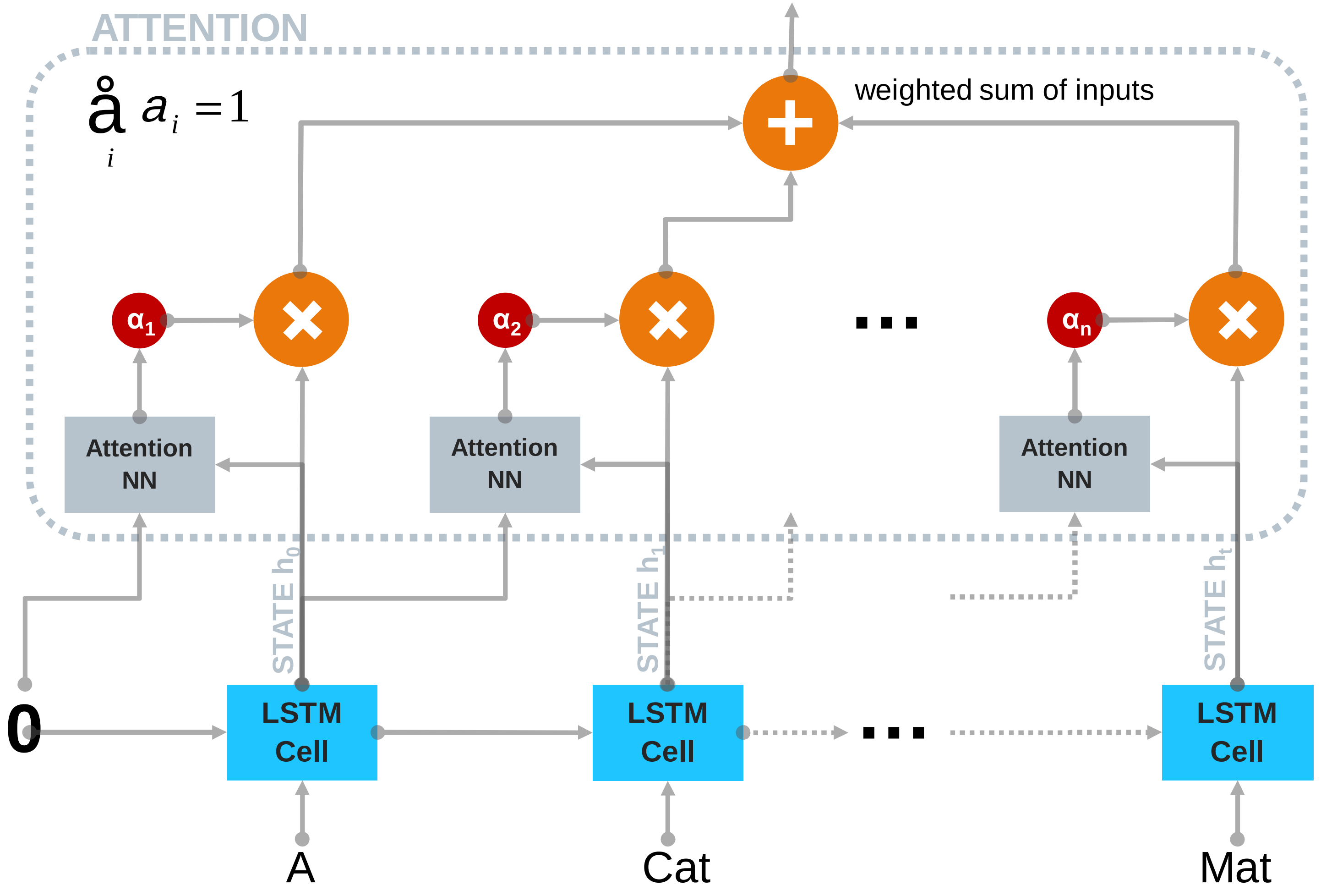


2016/06/01

Thinking in Macro Structures

Learning to Focus on the Important Things – Attention

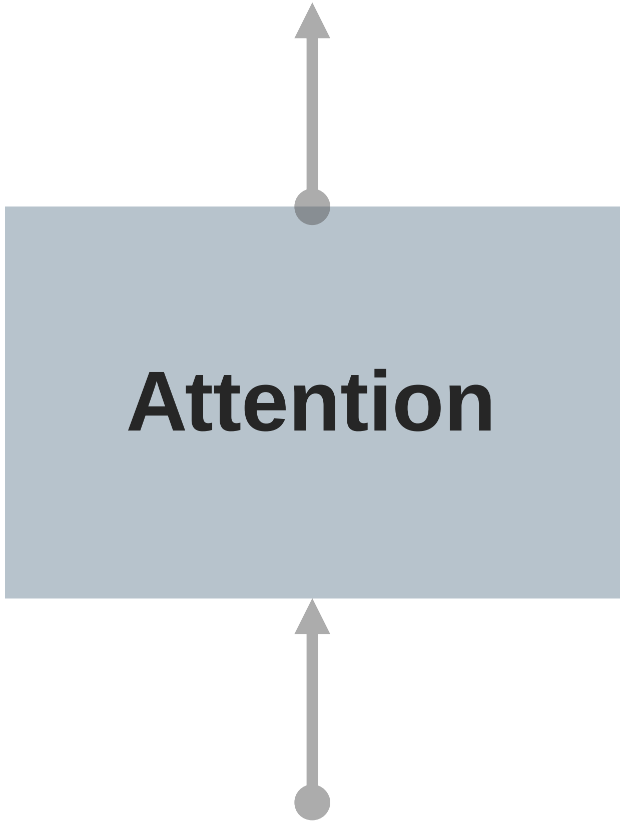
Goal: Filter out unimportant words for the target task.



Expectation:
Learns to measure the difference between the previous and current state representation:
Low difference = nothing new or important => low weight α

Thinking in Macro Structures

Remember the Important Things – And Move On.



+ Good for learning a context sensitive selection process

Interactive explanation: http://distill.pub/2016/augmented-rnns/

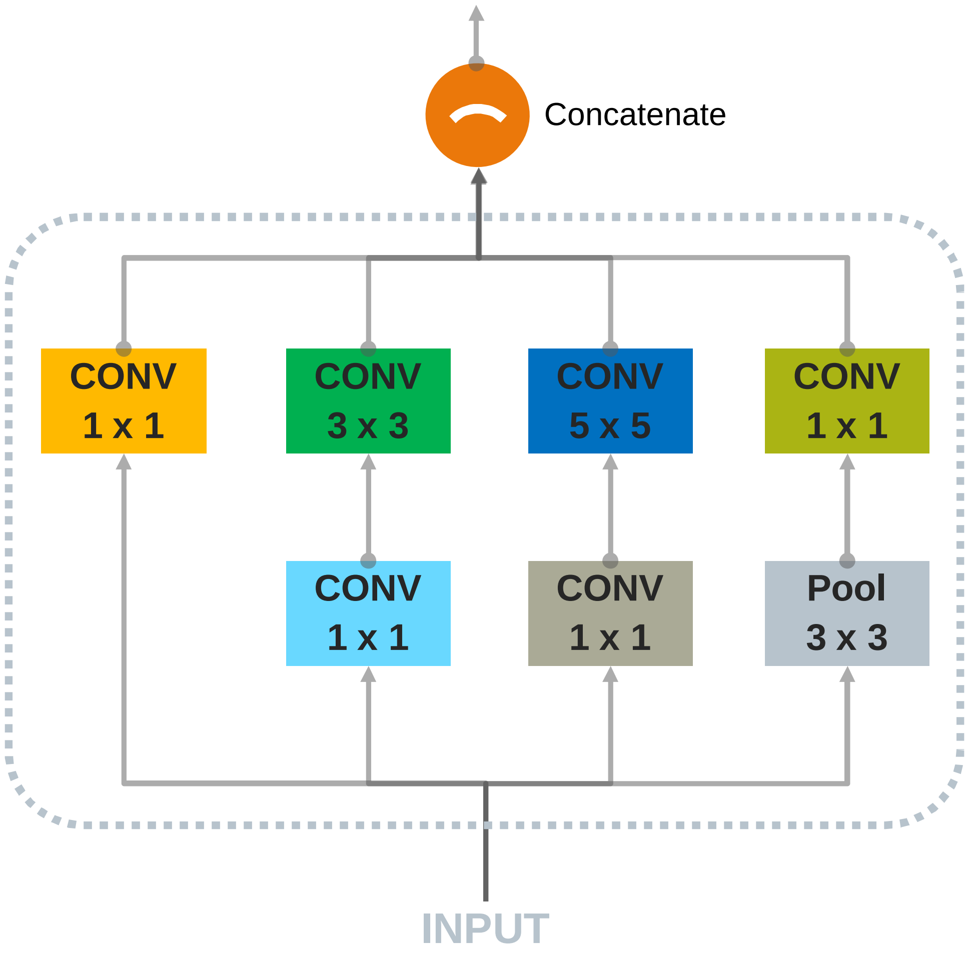
2016/06/01

Thinking in Macro Structures

Dynamic Receptive Fields – The Inception Architecture

- Provides the model with a choice of various filter sizes.

- Allows the model to combine different filter sizes.

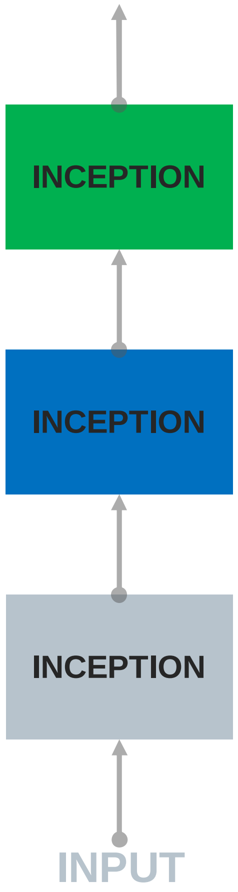


Thinking in Macro Structures

Dynamic Receptive Fields – The Inception Architecture

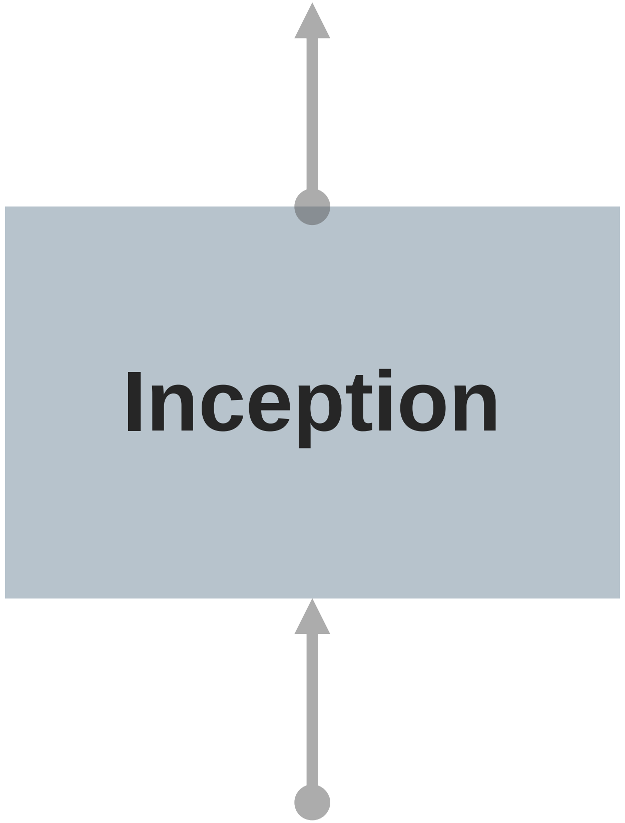
- Allows model to explicitly learn its “own” receptive field expansion.

- Allows the model to more explicitly learn different levels of receptive field expansion at the same time:
  - ØMight result in a more diverse set of hierarchical features available in each layer



Thinking in Macro Structures

Remember the Important Things – And Move On.



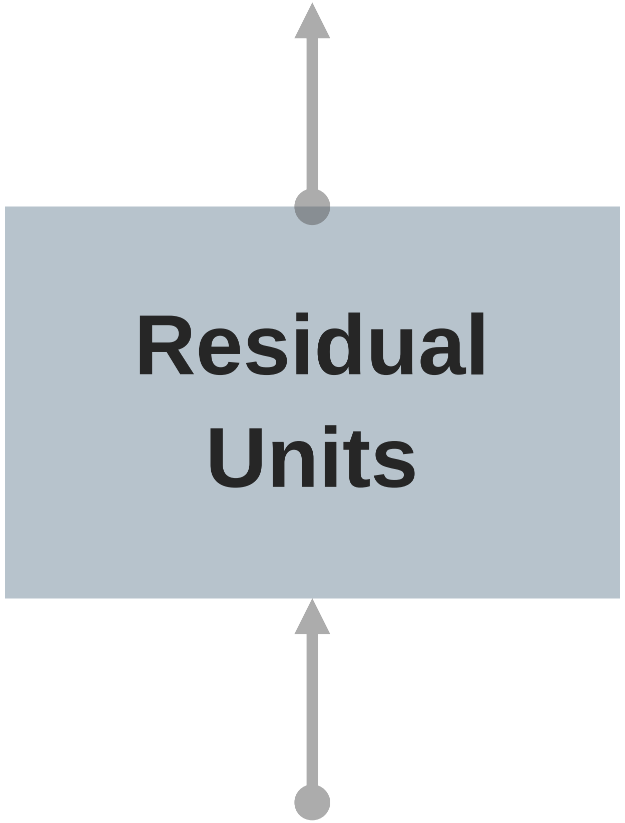
+ Good for learning complex and dynamic receptive field expansion

Paper https://arxiv.org/abs/1409.4842

2016/06/01

Thinking in Macro Structures

Remember the Important Things – And Move On.



+ Good for learning very deep networks

Paper: https://arxiv.org/pdf/1603.05027.pdf

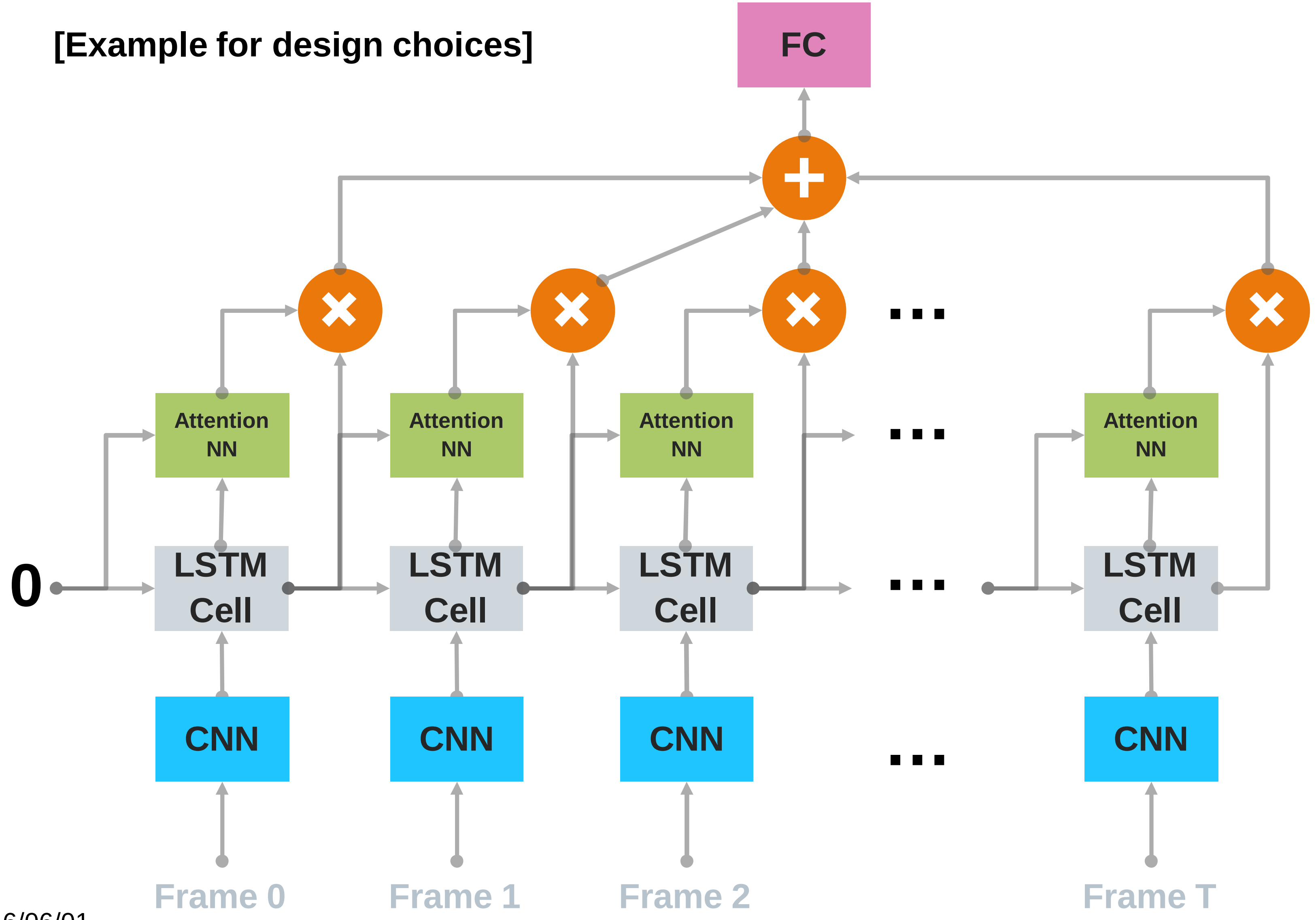
Deep Learning

End-to-End Model Design

End-to-End Model Design

Design Choices - Video Classification Example

[Example for design choices]



Classification

Remove unimportant frames

Capturing temporal dependencies

Hierarchical feature extraction
and input compression

2016/06/01

End-to-End Model Design

Real Examples - Deep Face

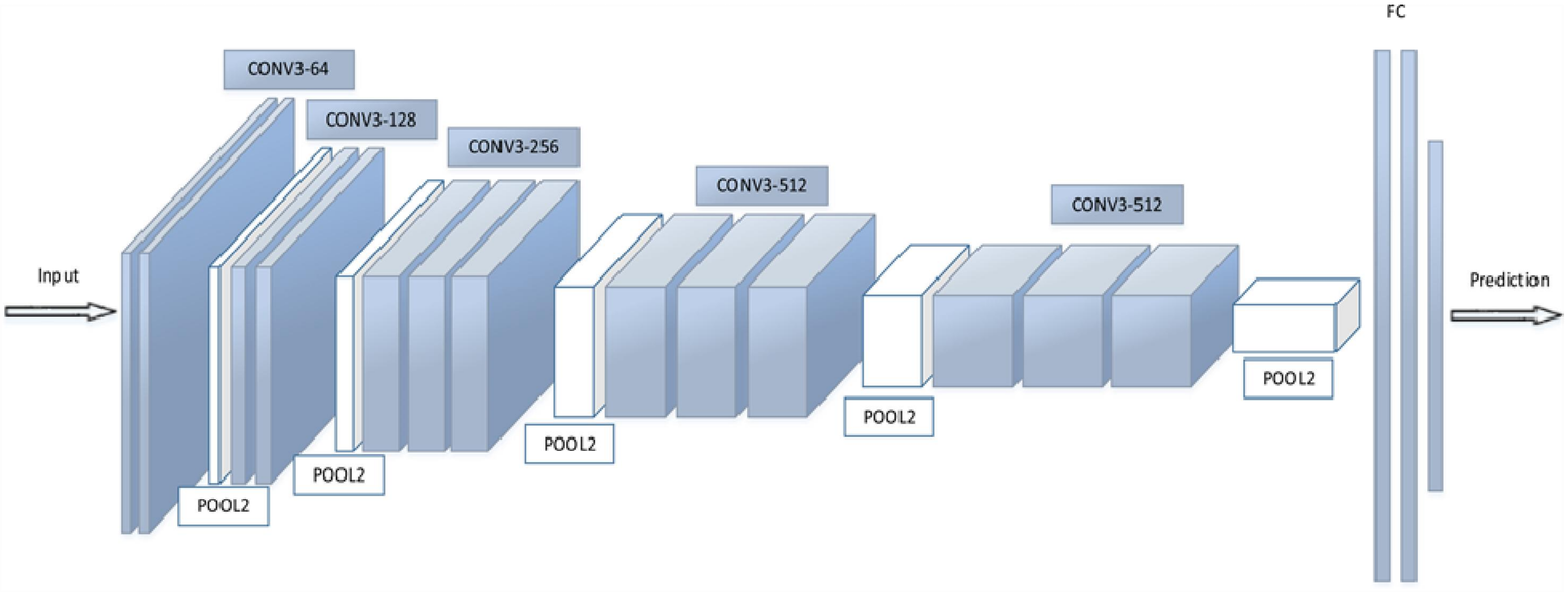


Image:

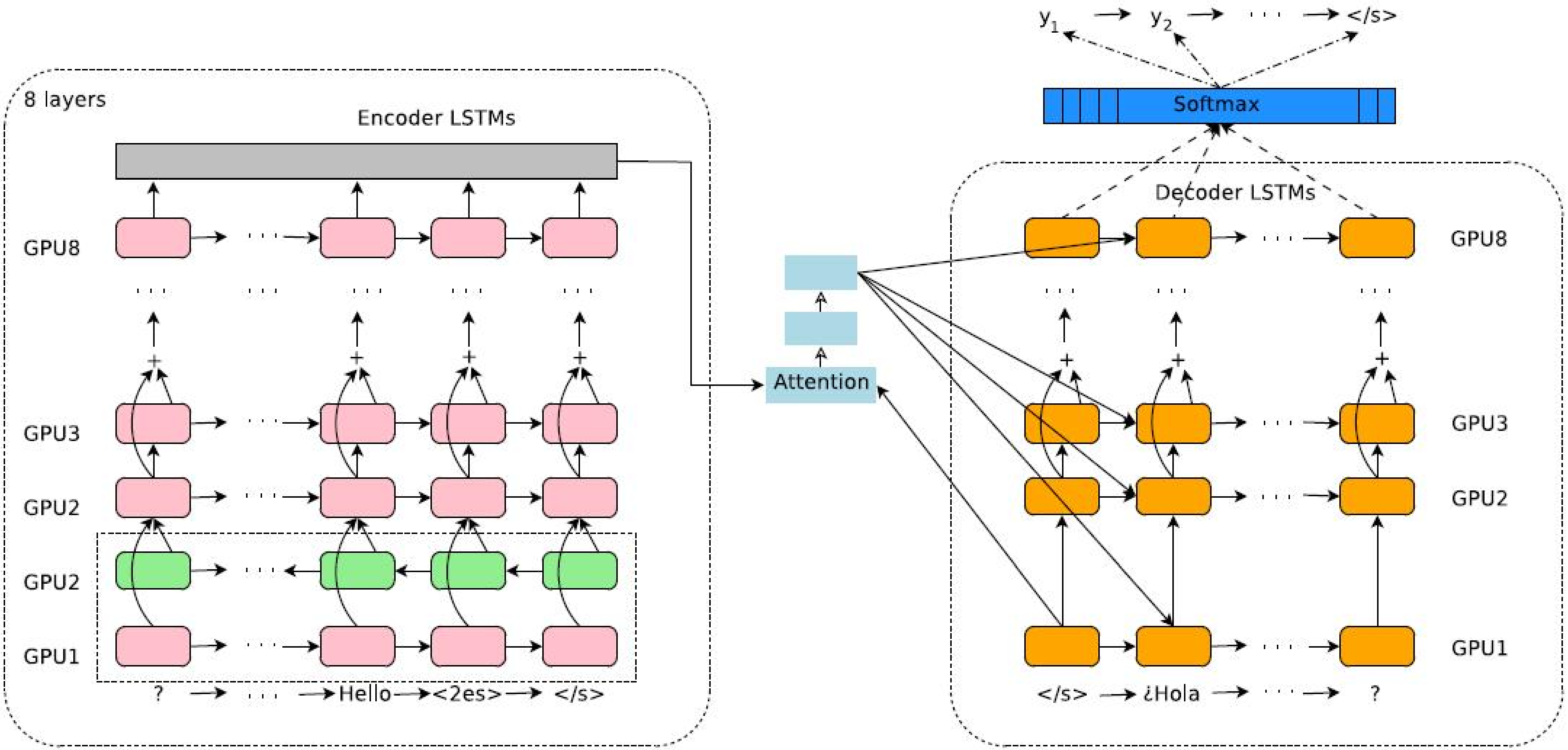
Hachim El Khiyari, Harry Wechsler

Face Recognition across Time Lapse Using Convolutional Neural Networks

Journal of Information Security, 2016.

End-to-End Model Design

Real Example - Multi-Lingual Neural Machine Translation

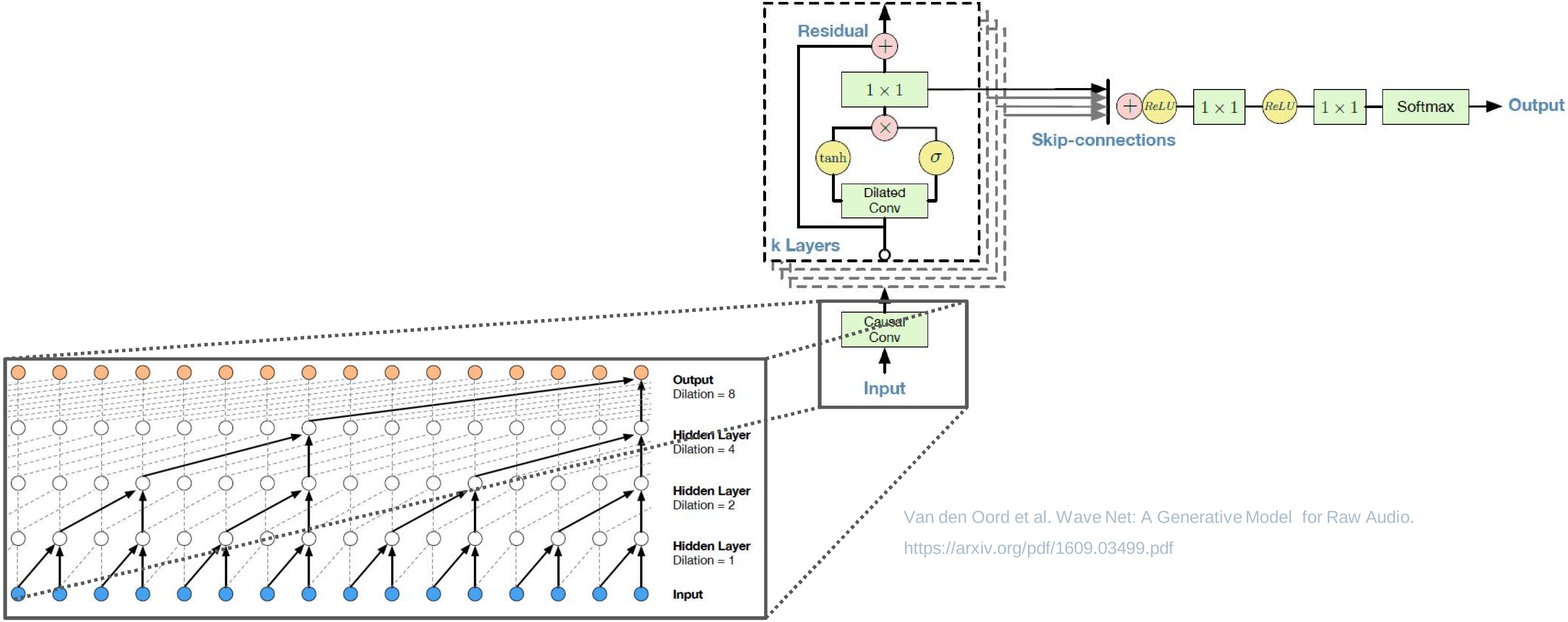


Yonghui Wu, et. al. **Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. https://arxiv.org/abs/1609.08144. 2016**

2016/06/01

End-to-End Model Design

Real Examples – Wave Net



Van den Oord et al. Wave Net: A Generative Model for Raw Audio.

https://arxiv.org/pdf/1609.03499.pdf

2016/06/01

Deep Learning

Part II

Deep Learning Model Training

Part II – Training Deep Learning Models

Loss Function Design

- Basic Loss functions
- Multi-Task Learning

Optimization

- Optimization in Deep Learning
- Work-horse Stochastic Gradient Descent
- Adaptive Learning Rates

Regularization

- Weight Decay
- Early Stopping
- Dropout
- Batch Normalization

Distributed Training

- Not covered, but I included a link to a good overview.

Deep Learning

Loss Function Design

Loss Function Design

Regression

Mean Squared Loss

$$f_{loss}^{R}(Y, X, \theta) = \frac{1}{n}\sum_{i}^{n}\left(y_i - \boxed{f_\theta(x_i)}\right)^2$$

Network output can be anything:

ØUse no activation function in output layer!

Example ID	Target Value (y_i)	Prediction ($f_\theta(x_i)$)	Example Error
1	4.2	4.1	0.01
2	2.4	0.4	4
3	-2.9	-1.4	2.25
…	…	…	…
n	0	1.0	1.0

Loss Function Design

Binary Classification

Binary Cross Entropy (also called Log Loss)

$$f^{BC}_{loss}(Y, X, \theta) = \frac{1}{n}\sum_{i}^{n} -\left[y_i \cdot \log(\boxed{f_\theta(x_i)}) + (1 - y_i) \cdot \log(1 - \boxed{f_\theta(x_i)})\right]$$

Network output needs to be between 0 and 1:

- Use sigmoid activation function in the output layer!
- Note: Sometimes there are optimized functions available that operate on the raw outputs (logits)

Example ID	Target Value (y_i)	Prediction ($f_\theta(x_i)$)	Example Error
1	0	0.211	0.237
2	1	0.981	0.019
3	0	0.723	1.284
…	…	…	…
n	0	0.134	0.144

Loss Function Design

Multi-Class Classification

Cross Entropy (Essentially the same as Perplexity in NLP)

$$f_{loss}^{MCC}(Y, X, \theta) = \frac{1}{n} \sum_{i}^{n} \sum_{j}^{c} - y_{i,j} \cdot \log(\boxed{f_\theta(x_i)_{i,j}})$$

Network output needs to represent a probability distribution over c classes: $\sum_{j}^{c} f_\theta(x_i)_{i,j} = 1$

- Use softmax activation function in the output layer!
- Note: Sometimes there are optimized functions available that operate on the raw outputs (logits)

Example ID	Target Value (y_i)	Prediction ($f_\theta(x_i)$)	Example Error
1	[0, 0, 1]	[0.2, 0.2, 0.6]	0.511
2	[1, 0, 0]	[0.3, 0.5, 0.2]	1.20
3	[0, 1, 0]	[0.1, 0.7, 0.3]	0.511
…	…	…	…
n	[0, 0, 1]	[0.0, 0.01, 0.99]	0.01

Loss Function Design

Multi-Label Classification

Multi-Label classification loss function (Just sum of Log Loss for each class)

$$f_{loss}^{MLC}(Y, X, \theta) = -\frac{1}{n}\sum_{i}^{n}\sum_{j}^{c} y_{i,j}\cdot\log(\boxed{f_\theta(x_i)_{i,j}}) + (1-y_{i,j})\cdot\log(1-\boxed{f_\theta(x_i)_{i,j}})$$

Each network output needs to be between 0 and 1:

- Use sigmoid activation function on each network output!
- Note: Sometimes there are optimized functions available that operate on the raw outputs (logits)

Example ID	Target Value (y_i)	Prediction ($f_\theta(x_i)$)	Example Error
1	[0, 0, 1]	[0.2, 0.4, 0.6]	1.245
2	[1, 0, 1]	[0.3, 0.9, 0.2]	5.116
3	[0, 1, 0]	[0.1, 0.7, 0.1]	0.567
…	…	…	…
n	[1, 1, 1]	[0.8, 0.9, 0.99]	0.339

Loss Function Design
Multi-Task Learning

Additive Cost Function

$$f_{loss}^{MT}\left(\left[Y_0,\ldots,Y_K\right],\left[X_0,\ldots,X_K\right],q\right)=\overset{K}{\underset{k}{\text{å}}}\; l_k\, f_{loss_k}\left(Y_k,X_k,q\right)$$

Each network output has associated input and target data and an associated loss metric:

ØUse proper output activation for each of the k output layer!

ØThe weighting λ$_k$ of each task in the cost function is derived from prior knowledge/assumptions or by trial and error.

ØNote that we could learn multiple tasks from the same data. This can be represented by copies of the corresponding data in the formula above. When implementing this, we would of course not copy the data.

Examples:

- Auxiliary heads for counteracting vanishing gradient (Google LeNet, https://arxiv.org/abs/1409.4842)
- Artistic style transfer (Neural Artistic Style Transfer, https://arxiv.org/abs/1508.06576)
- Instance segmentation (Mask R-NN, https://arxiv.org/abs/1703.06870)
- …

Deep Learning

Optimization

Optimization

Learning the Right Parameters in Deep Learning

- Neural networks are composed of differentiable building blocks
- Training a neural network means minimization of some non-convex differentiable loss function using iterative gradient-based optimization methods
- The simplest but mostly used optimization algorithm is “gradient descent”

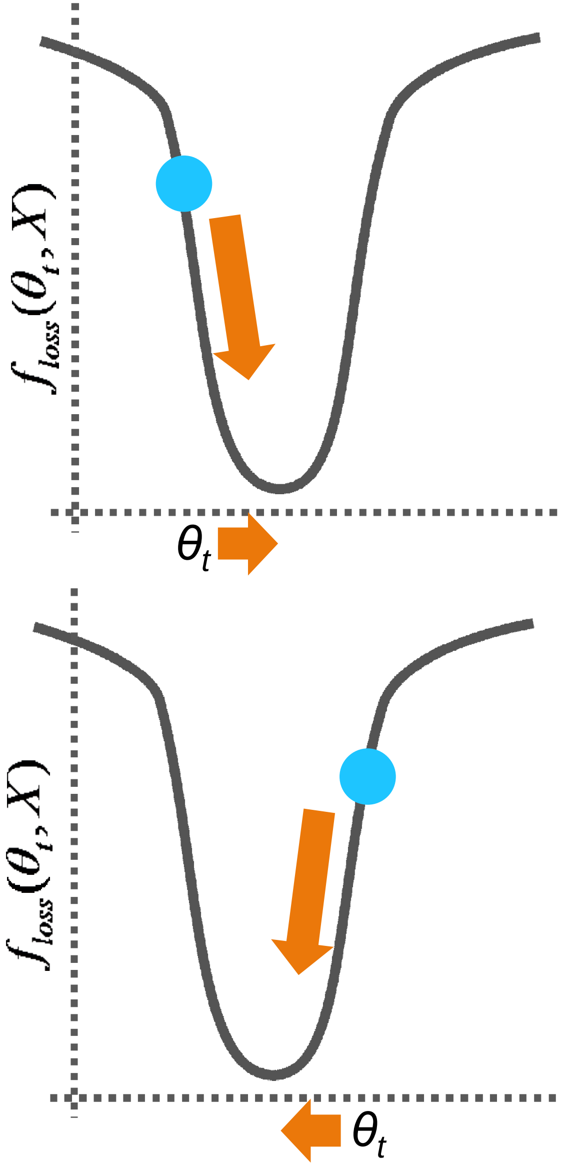
Optimization

Gradient Descent

Negative Gradient

You can think of the gradient as the local slope with respect to each parameter θ_i at step t.

Positive Gradient



We update the parameters a little bit in the direction where the error gets smaller

$$q_t = q_{t-1} - h › g_t$$

Gradient with respect to the model parameters θ

$$\text{with } g_t = \tilde{N}_q f_{loss}(Y, X, q_{t-1})$$

Optimization

Work-Horse Stochastic Gradient Descent

Stochastic Gradient Descent is Gradient Descent on samples (Mini-Batches) of data:

- Increases variance in the gradients
  - ØSupposedly helps to jump out of local minima

- But essentially, it is just super efficient and it works!

We update the parameters a little bit in the direction where the error gets smaller

$$q_t = q_{t-1} - h \times g_t^{(s)}$$

Gradient with respect to the model parameters θ

$$\text{with } g_t^{(s)} = \tilde{N}_q f_{loss}(Y^{(s)}, X^{(s)}, q_{t-1})$$

In the following we will omit the superscript s and X will always represent a mini-batch of samples from the data.

Optimization

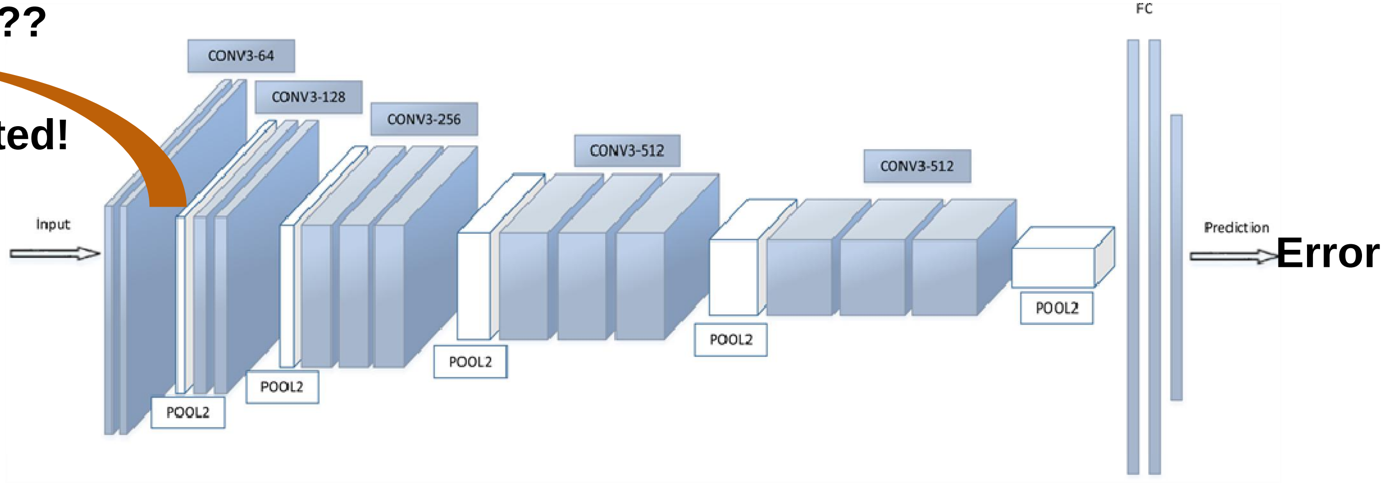
Computing the Gradient

I have to compute the gradient of that???

Sounds complicated!

$$\theta_t = \theta_{t-1} - \eta \cdot g_t$$

$$\text{with } g_t = \nabla_\theta f_{loss}(Y, X, \theta_{t-1})$$



Error

Image:

Hachim El Khiyari, Harry Wechsler

Face Recognition across Time Lapse Using Convolutional Neural Networks

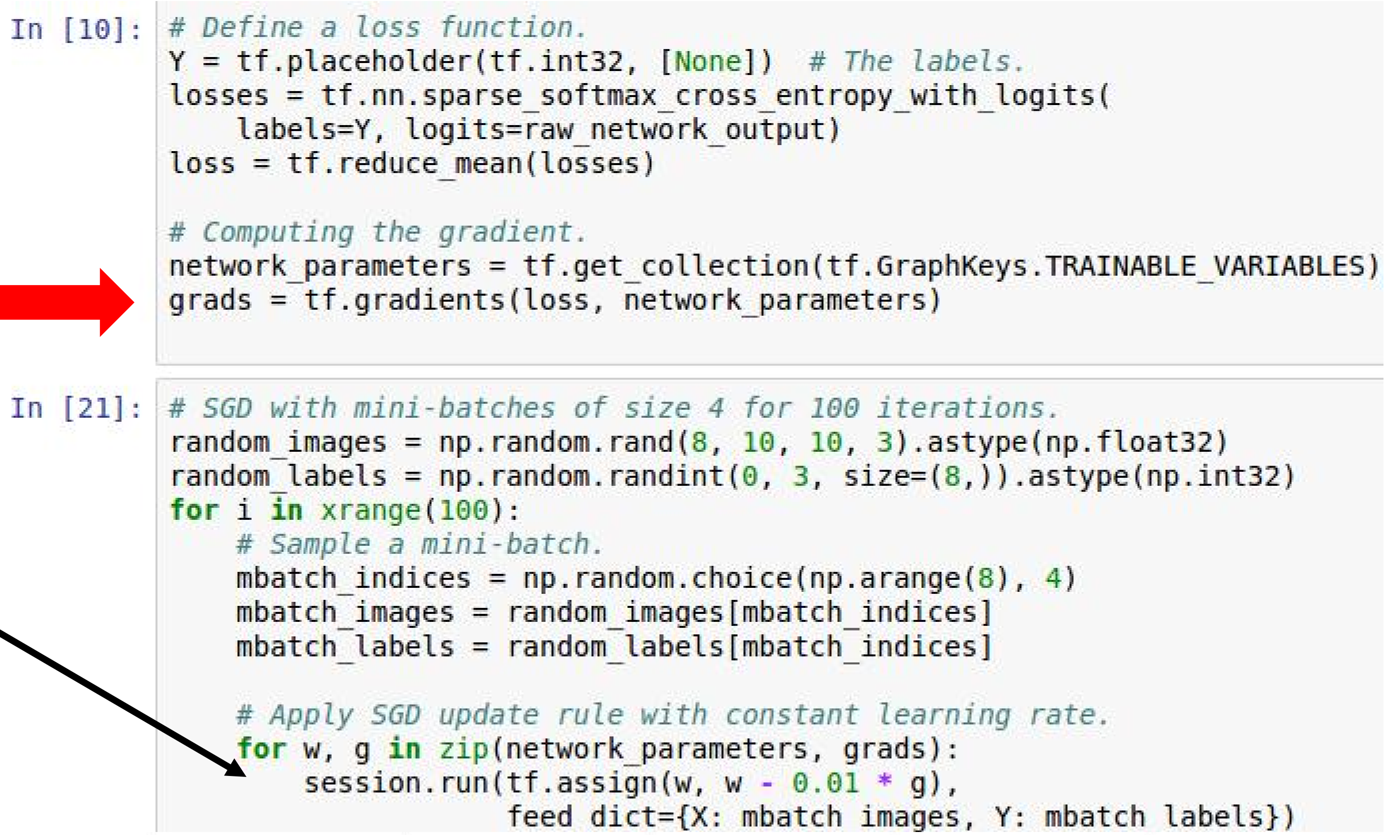
Journal of Information Security, 2016.

Optimization

Automatic Differentiation

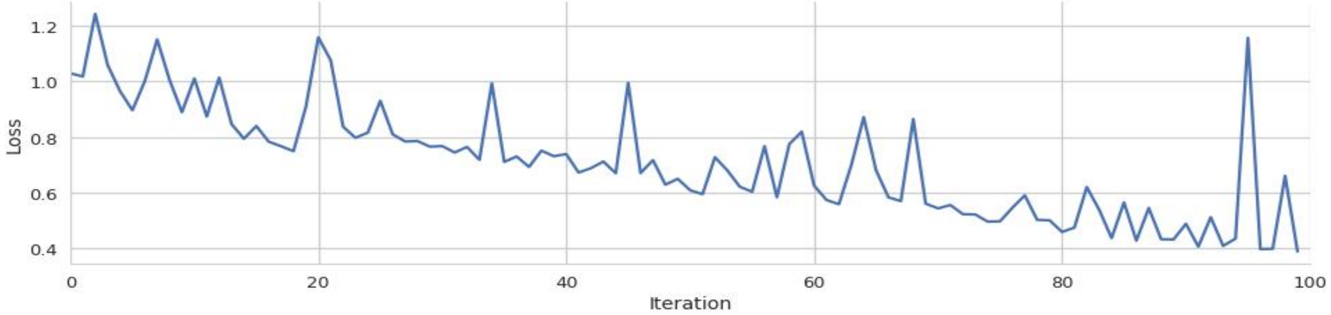
$$q_t = q_{t-1} - h › g_t$$

$$\text{with } g_t = \tilde{N}_q f_{loss}(Y, X, q_{t-1})$$



```
In [10]: # Define a loss function.
         Y = tf.placeholder(tf.int32, [None])  # The labels.
         losses = tf.nn.sparse_softmax_cross_entropy_with_logits(
             labels=Y, logits=raw_network_output)
         loss = tf.reduce_mean(losses)

         # Computing the gradient.
         network_parameters = tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES)
         grads = tf.gradients(loss, network_parameters)
```

```
In [21]: # SGD with mini-batches of size 4 for 100 iterations.
         random_images = np.random.rand(8, 10, 10, 3).astype(np.float32)
         random_labels = np.random.randint(0, 3, size=(8,)).astype(np.int32)
         for i in xrange(100):
             # Sample a mini-batch.
             mbatch_indices = np.random.choice(np.arange(8), 4)
             mbatch_images = random_images[mbatch_indices]
             mbatch_labels = random_labels[mbatch_indices]

             # Apply SGD update rule with constant learning rate.
             for w, g in zip(network_parameters, grads):
                 session.run(tf.assign(w, w - 0.01 * g),
                             feed_dict={X: mbatch_images, Y: mbatch_labels})
```



Optimization

Automatic Differentiation

AUTOMATIC DIFFERENTIATION

IS AN

EXTREMELY POWERFUL FEATURE

FOR DEVELOPING MODELS WITH

DIFFERENTIABLE OPTIMIZATION OBJECTIVES

theano TensorFlow™ torch

Optimization

Wait a Minute, I thought Neural Networks are Optimized via Backpropagation

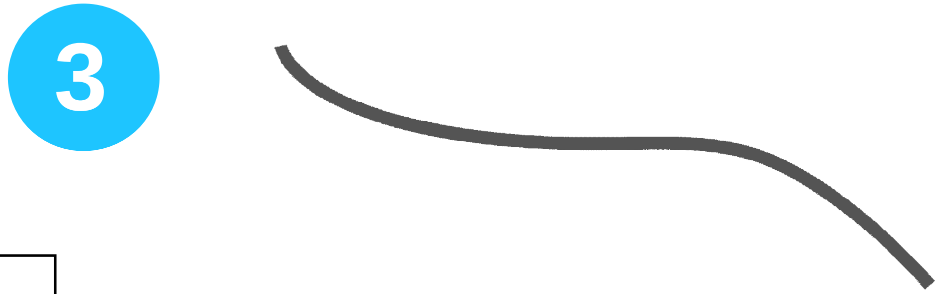
Backpropagation is just a fancy name for applying the chain rule to compute the gradients in neural networks!

Optimization

Stochastic Gradient Descent – Problems with Constant Learning Rates

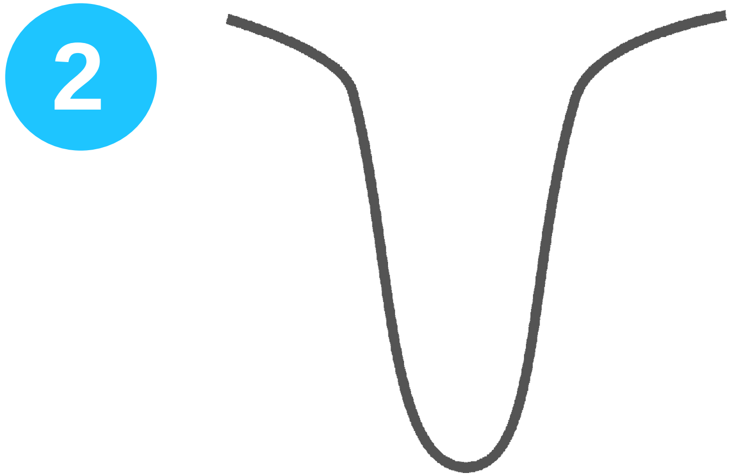
Low gradient

Flat gradient

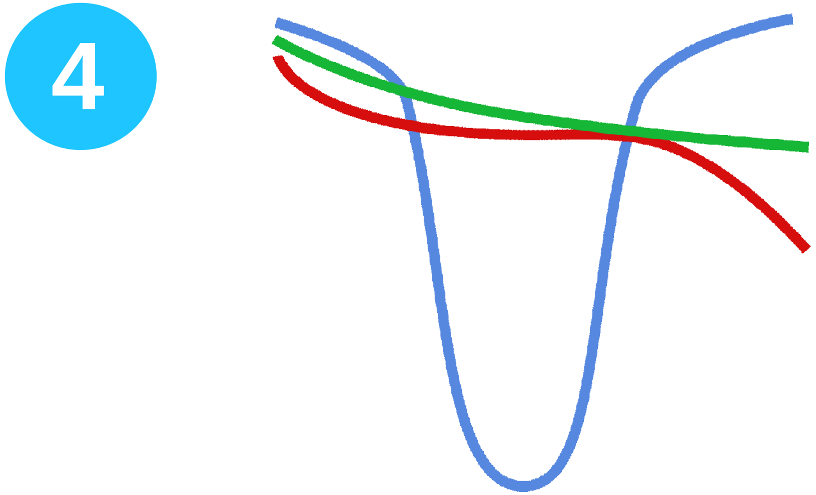


$$\boxed{q_t = q_t - h \times g_t}$$

Steep gradient



Gradients of different parameters vary



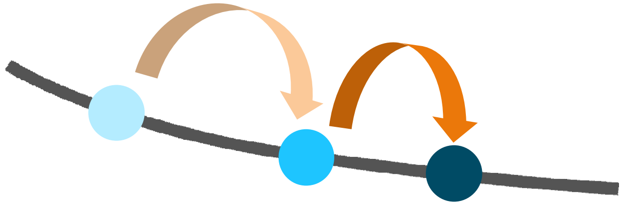
Excellent Overview and Explanation: http://sebastianruder.com/optimizing-gradient-descent/

Optimization

Stochastic Gradient Descent – Problems with Constant Learning Rates

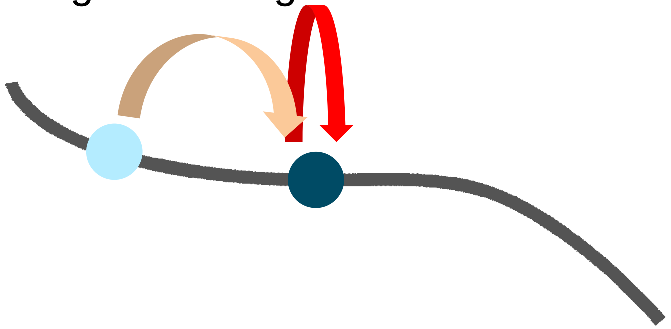
Learning rate too small

✖ 1



Get stuck in zero gradient regions

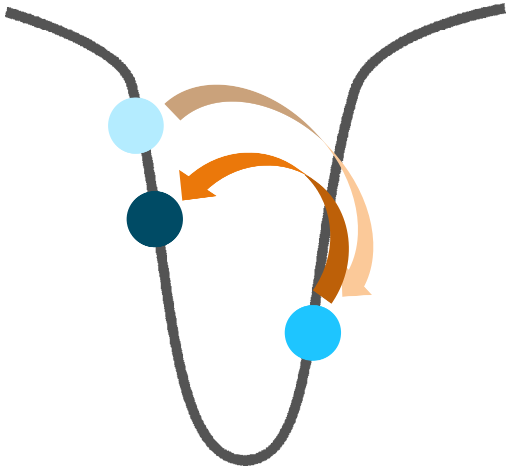
✖ 3



$$\boxed{q_t = q_t - h \times g_t}$$

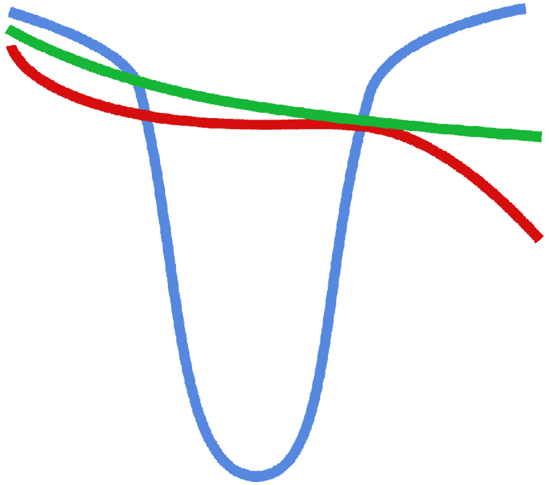
Learning rate too large

✖ 2



Learning rate can be parameter specific

✖ 4

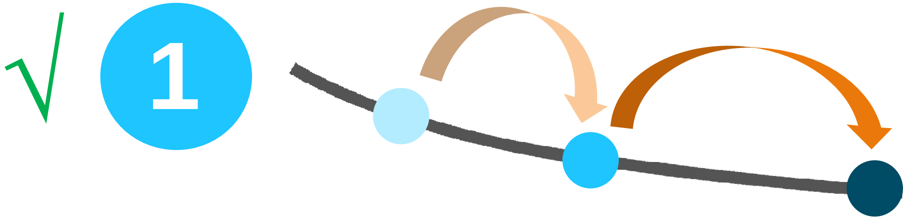


Excellent Overview and Explanation: http://sebastianruder.com/optimizing-gradient-descent/

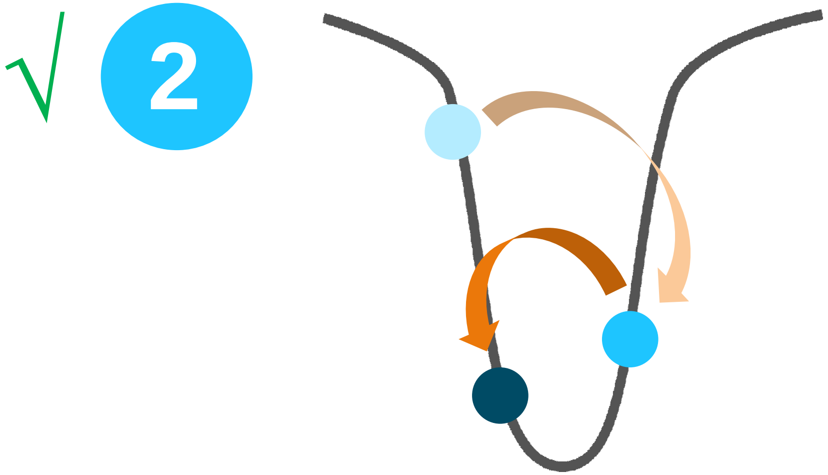
Optimization

Stochastic Gradient Descent – Adding Momentum

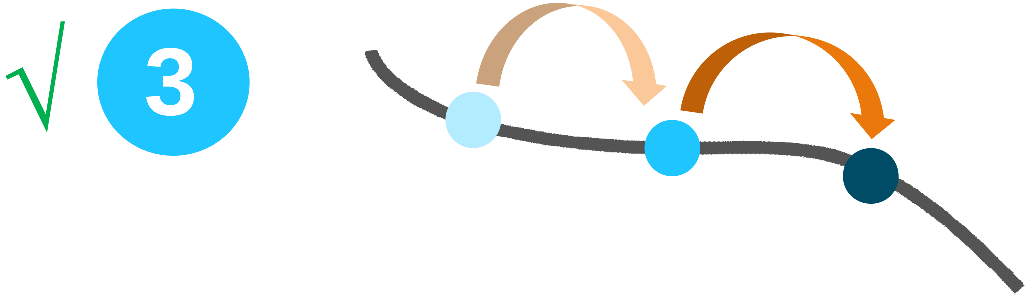
Step size can accumulates momentum if successive gradients have same direction



Step size decreases fast if the direction of the gradients changes



Momentum only decays slowly and does not stop immediately



Adding the previous step size can lead to acceleration

Decay (“friction”)

Constant learning rate

$$v_t = l\, v_{t-1} + hg_t$$

$$q_t = q_{t-1} - v_t$$

$$\text{with } g_t = \tilde{N}_q f_{loss}(Y, X, q_{t-1})$$

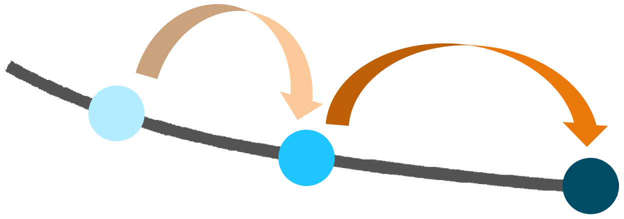
Excellent Overview and Explanation: http://sebastianruder.com/optimizing-gradient-descent/

Optimization

Stochastic Gradient Descent – Adaptive Learning Rate (RMS Prop)

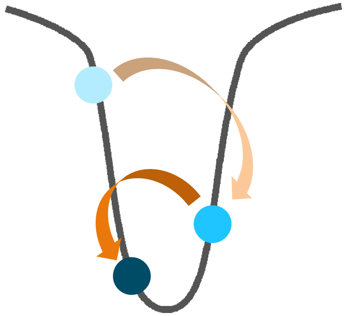
Continuously low gradient will increase the learning rate

√ 1



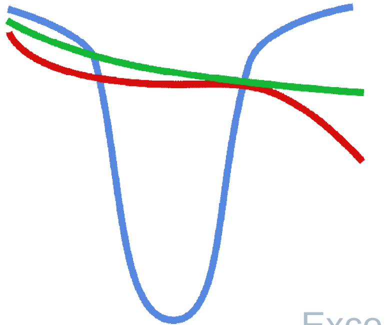
Continuously large gradients will result in a decrease of the learning rate

√ 2



For each parameter an individual learning rate is computed

√ 4



Update rule with an individual learning rate for each parameter *θ*$_i$

$$q_{t,i} = q_{t,i} - h_i \text{¢} \times g_{t,i}$$

The learning rate is adapted by a decaying mean of past updates

$$E[g_i^2]_t = b \times E[g_i^2]_{t-1} - (1- b) \times g_{t,i}^2$$

The correction of the (constant) learning rate for each parameter.
The epsilon is only for numerical stability

$$h_i \text{¢} = \frac{h}{\sqrt{E[g_i^2]_t + e}}$$

Excellent Overview and Explanation: http://sebastianruder.com/optimizing-gradient-descent/

Optimization

Stochastic Gradient Descent – Overview Common Step Rules

	Constant Learning Rate	Constant Learning Rate with Annealing	Momentum	Nesterov	AdaDelta	RMSProp	RMSProp + Momentum	ADAM
1	✖	✖	√	√	√	√	√	√
2	✖	√	√	√	√	√	√	√
3	✖	✖	√	√	✖	✖	√	√
4	✖	✖	✖	✖	√	√	√	√

This does not mean that it cannot make sense to use only a constant learning rate!

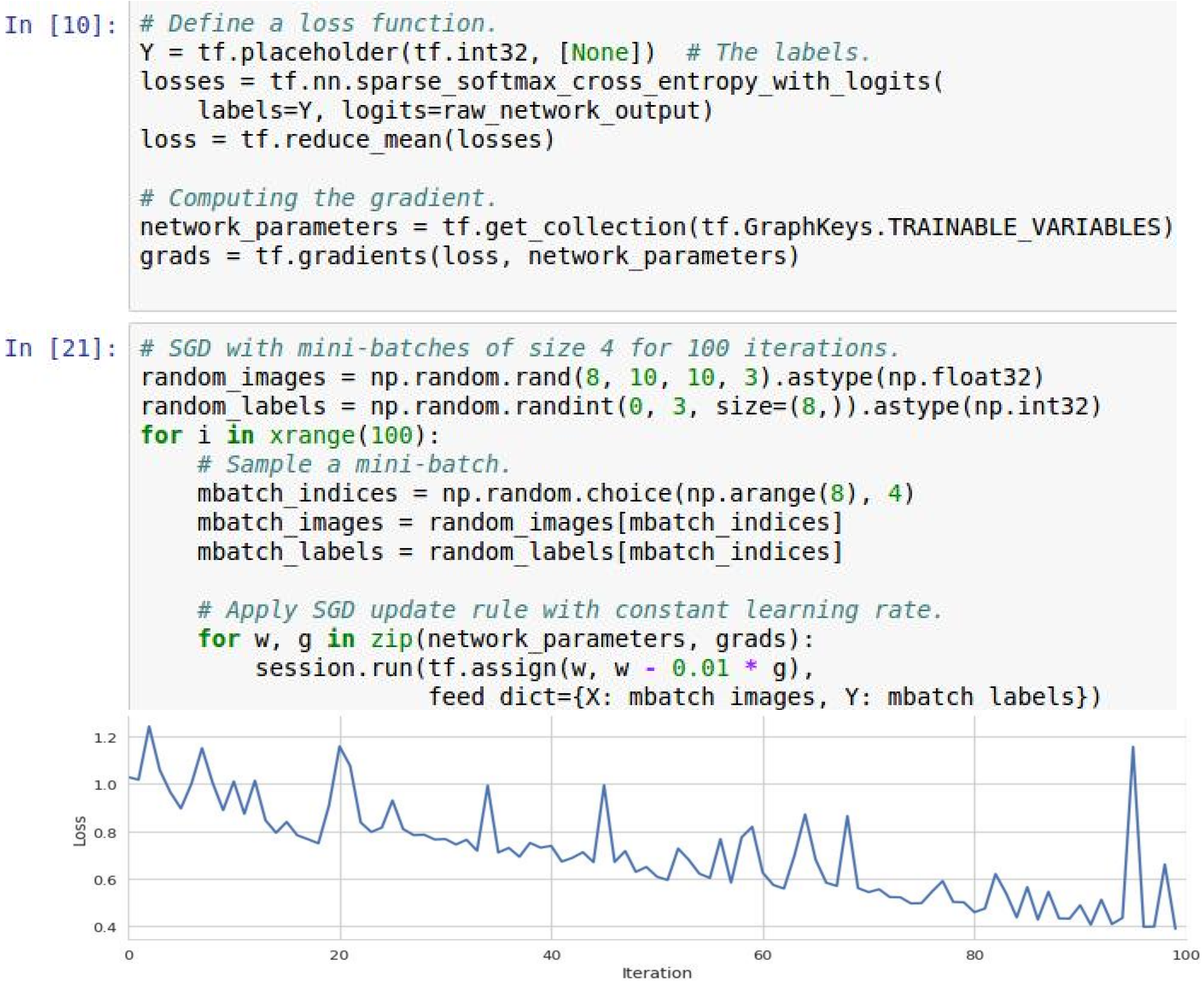
Excellent Overview and Explanation: http://sebastianruder.com/optimizing-gradient-descent/

Optimization

Something feels terribly wrong here, can you see it?

$$q_t = q_{t-1} - h \text{›} g_t$$

$$\text{with } g_t = \tilde{N}_q f_{loss}(Y, X, q_{t-1})$$



```
In [10]: # Define a loss function.
         Y = tf.placeholder(tf.int32, [None])  # The labels.
         losses = tf.nn.sparse_softmax_cross_entropy_with_logits(
             labels=Y, logits=raw_network_output)
         loss = tf.reduce_mean(losses)

         # Computing the gradient.
         network_parameters = tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES)
         grads = tf.gradients(loss, network_parameters)
```

```
In [21]: # SGD with mini-batches of size 4 for 100 iterations.
         random_images = np.random.rand(8, 10, 10, 3).astype(np.float32)
         random_labels = np.random.randint(0, 3, size=(8,)).astype(np.int32)
         for i in xrange(100):
             # Sample a mini-batch.
             mbatch_indices = np.random.choice(np.arange(8), 4)
             mbatch_images = random_images[mbatch_indices]
             mbatch_labels = random_labels[mbatch_indices]

             # Apply SGD update rule with constant learning rate.
             for w, g in zip(network_parameters, grads):
                 session.run(tf.assign(w, w - 0.01 * g),
                             feed_dict={X: mbatch_images, Y: mbatch_labels})
```

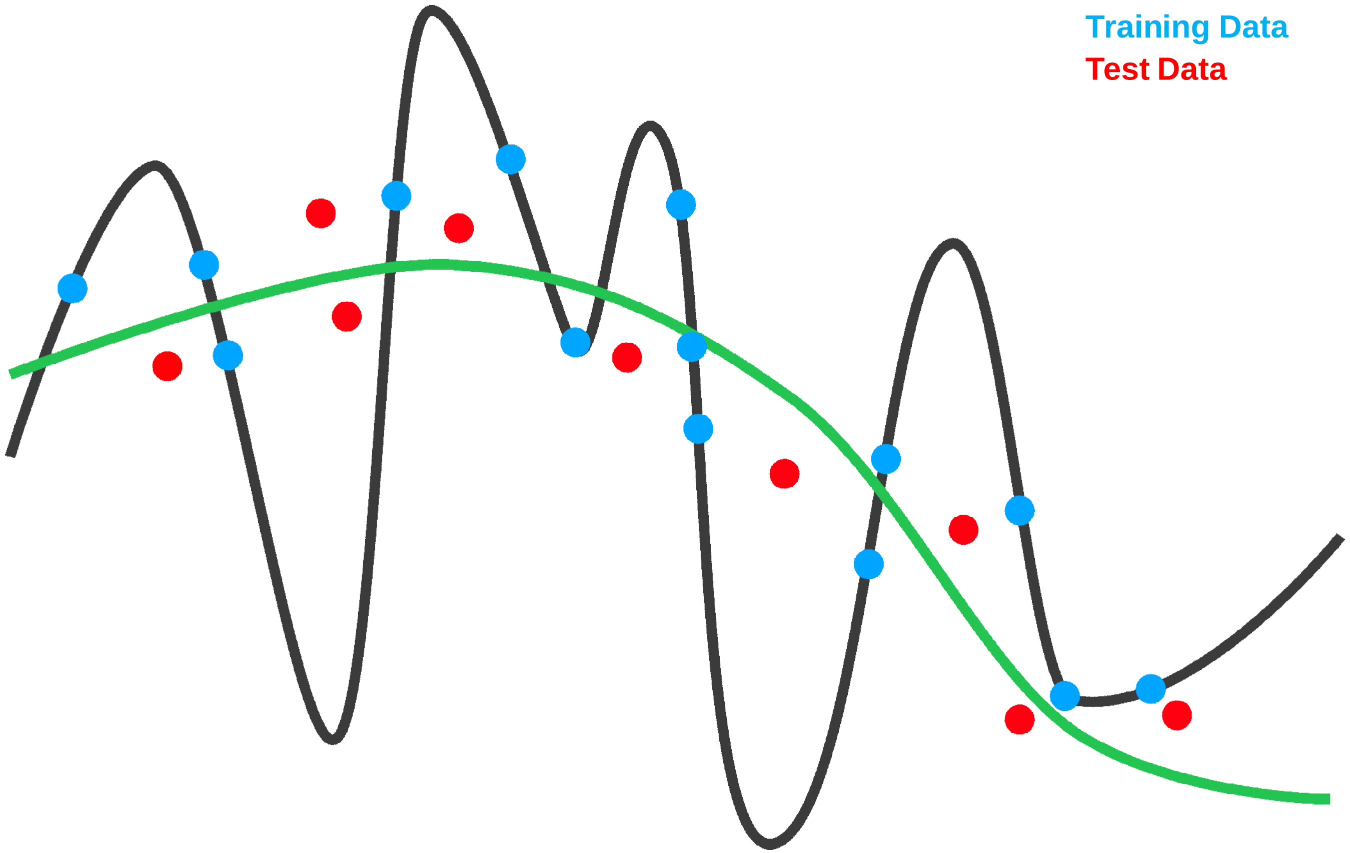
Deep Learning

Regularization

Regularization

Why Regularization is Important

- The goal of learning is not to find a solution that explain the training data perfectly.
- The goal of learning is to find a solution that generalizes well on unseen data points.
- Regularization tries to prevent the model to just fit the training data in an arbitrary way (overfitting).



Regularization
Weight Decay – Constraining Parameter Values

Intuition:

- Discourage the model for choosing undesired values for parameters during learning.

General Approach:

- Putting prior assumptions on the weights. Deviations from these assumptions get penalized.

Examples:

L2 –Regularization (Squared L2 norm or Gaussian Prior)

$$\|\theta\|_2^2 = \sum_{i,j} (\theta_{i,j})^2$$

L1-Regularization

$$\|\theta\|_1 = \sum_{i,j} |\theta_{i,j}|$$

The regularization term is just added to the cost function for the training.

$$f_{loss}^{total}(Y, X, \theta) = f_{loss}(Y, X, \theta) + \lambda \cdot \|\theta\|_2^2$$

λ is a tuning parameter that determines how strong the regularization affects learning.

Regularization

Early Stopping – Stop Training Just in Time.

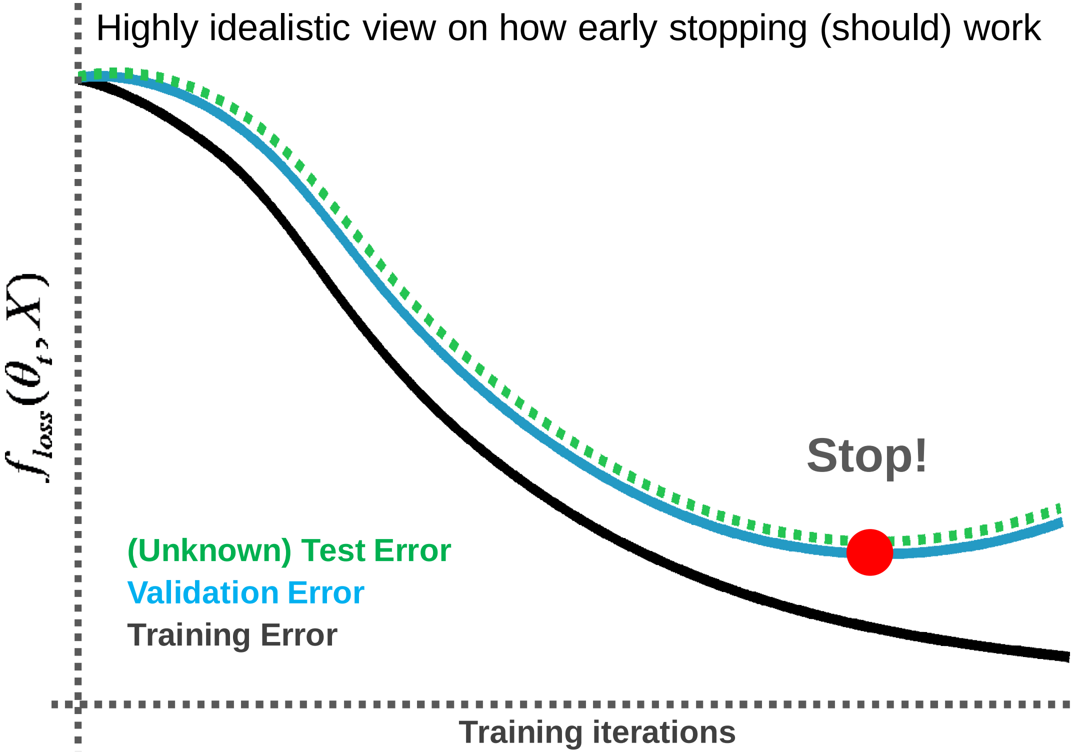
Problem

- There might be a point during training where the model starts to overfit the training data at the cost of generalization.

Approach

- Separate additional data from the training data and consistently monitor the error on this validation dataset.
- Stop the training if the error on this dataset does not improve or gets worse over a certain amount of training iterations.

- It is assumed that the validation set approximates the models generalization error (on the test data).



Regularization

Dropout – Make Nodes Expendable

Problem

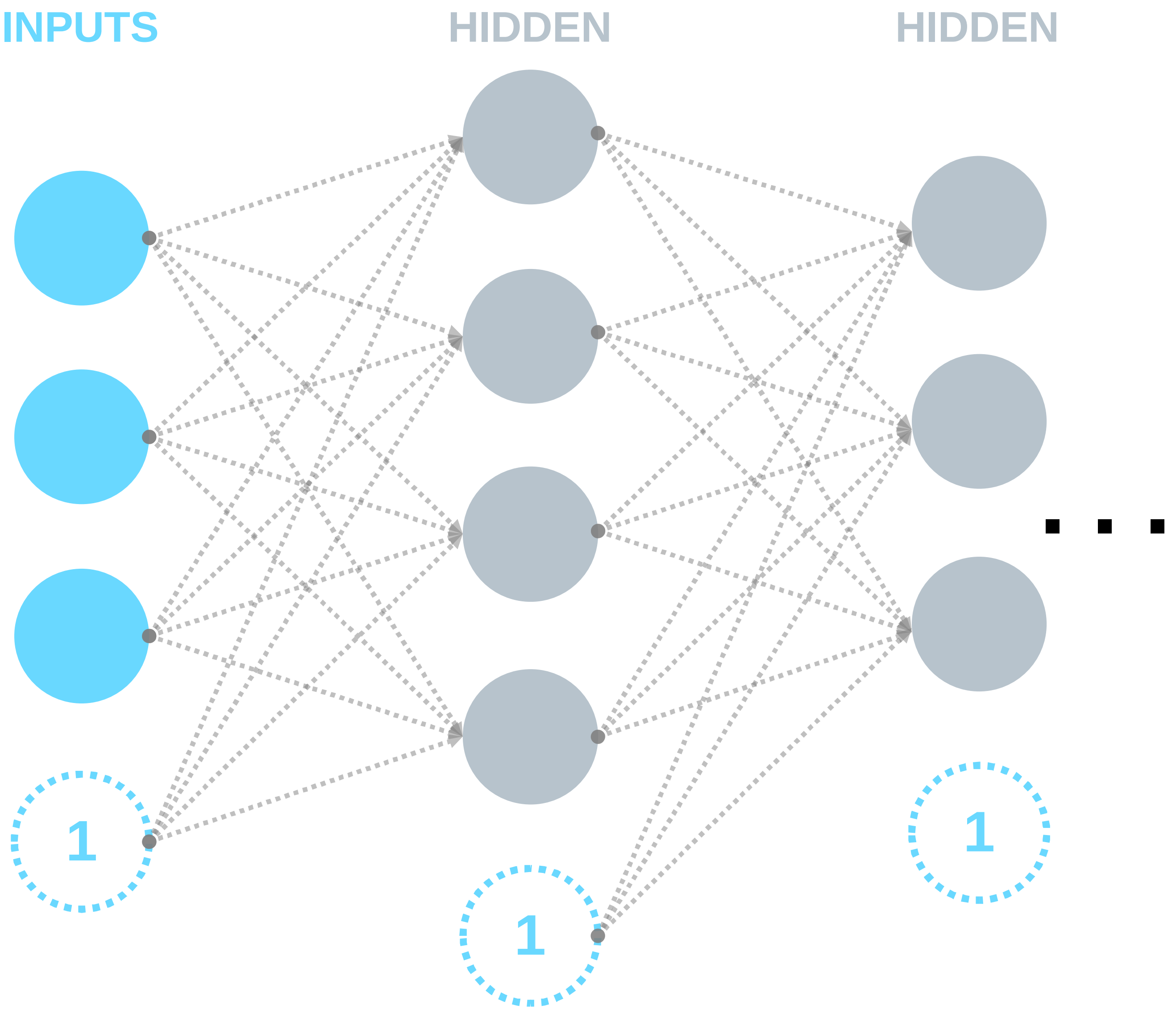
- Deep learning models are often highly over parameterized which allows the model to overfit on or even memorize the training data.

Approach

- Randomly set output neurons to zero
  - ØTransforms the network into an ensemble with an exponential set of weaker learners whose parameters are shared.

Usage

- Primarily used in fully connected layers because of the large number of parameters
- Rarely used in convolutional layers
- Rarely used in recurrent neural networks (if at all between the hidden state and output)



2016/06/01

Regularization

Dropout – Make Nodes Expendable

Problem

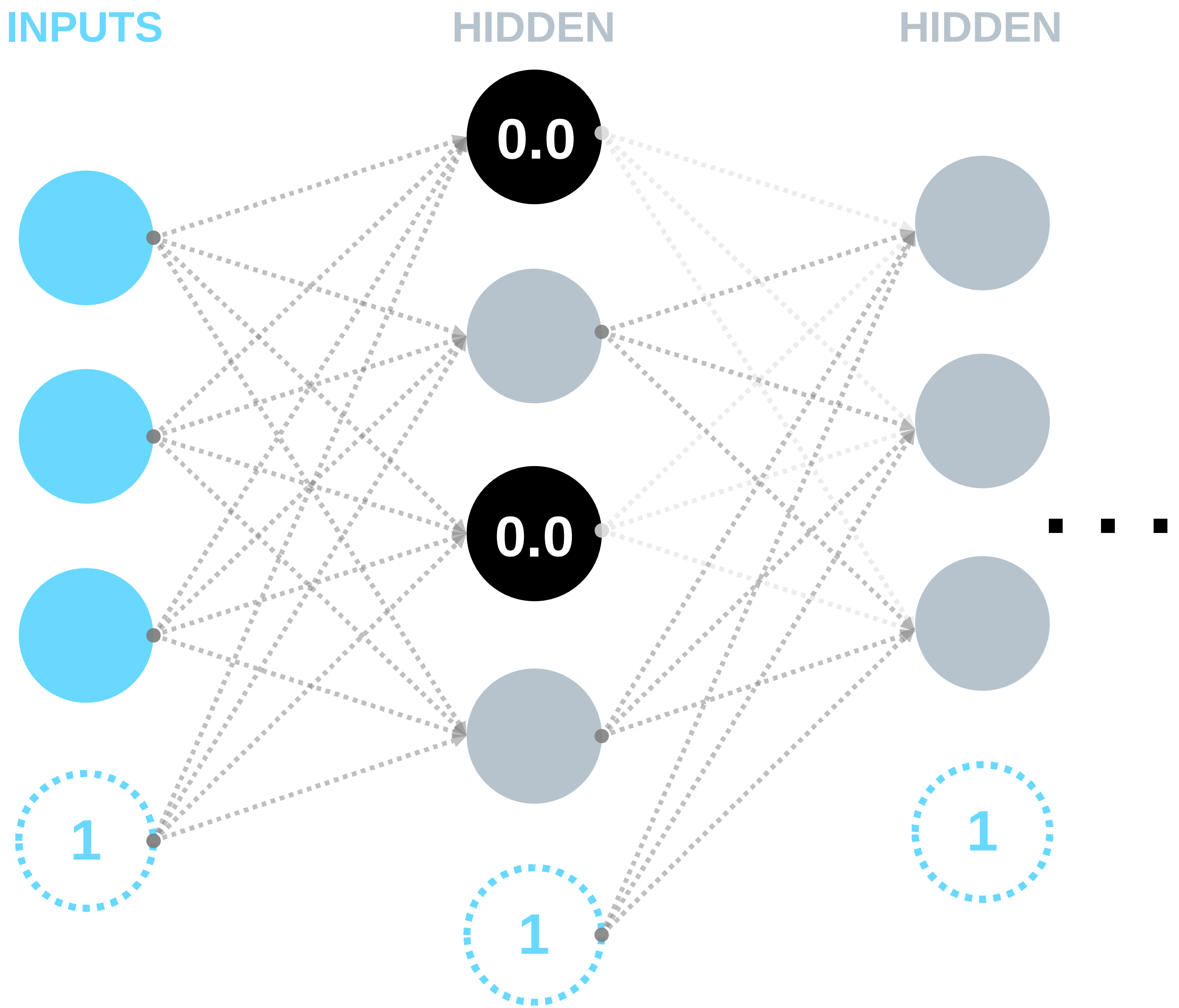
- Deep learning models are often highly over parameterized which allows the model to overfit on or even memorize the training data.

Approach

- Randomly set output neurons to zero
    - ØTransforms the network into an ensemble with an exponential set of weaker learners whose parameters are shared.

Usage

- Primarily used in fully connected layers because of the large number of parameters
- Rarely used in convolutional layers
- Rarely used in recurrent neural networks (if at all between the hidden state and output)



Regularization

Dropout – Make Nodes Expendable

Problem

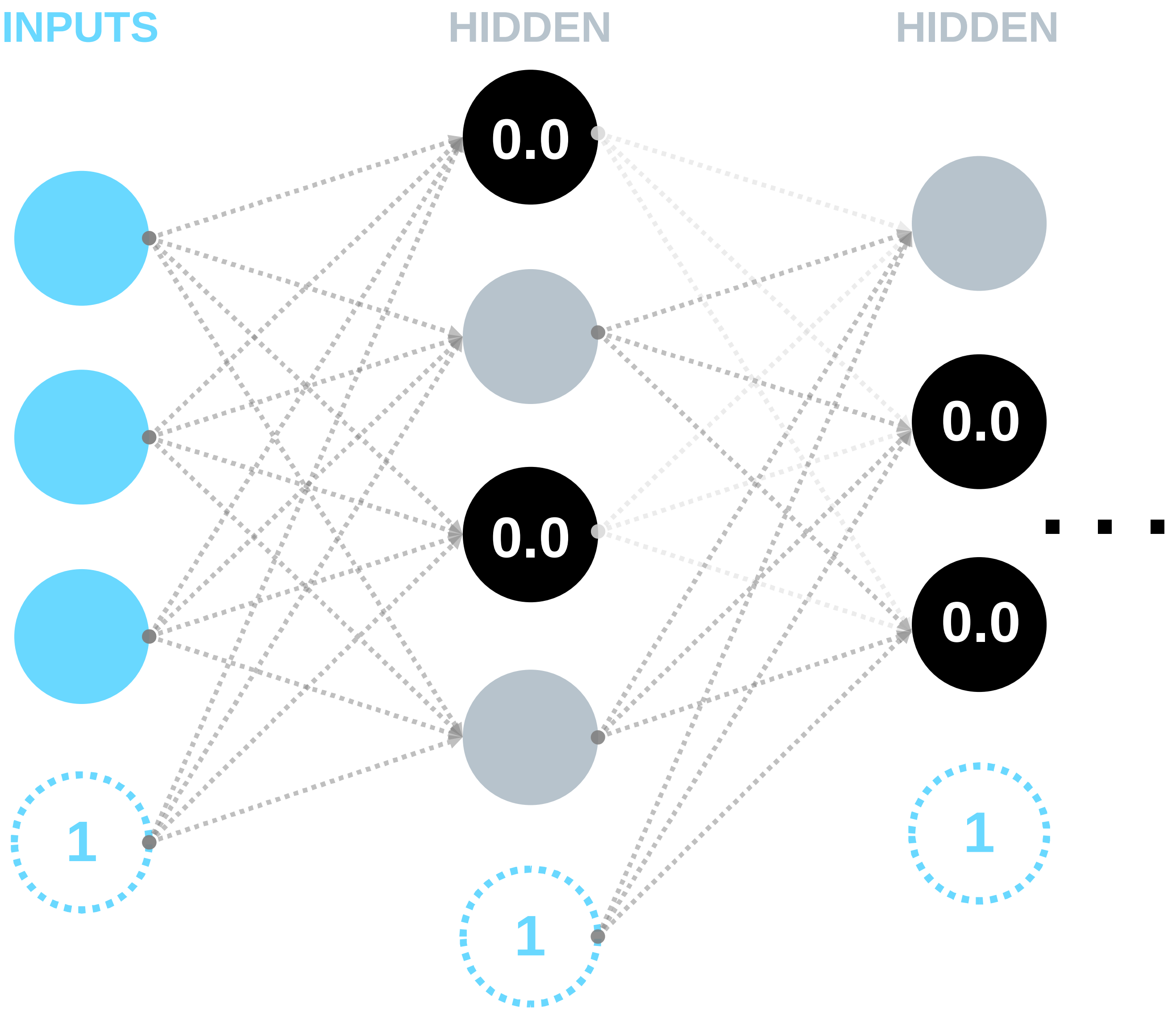
- Deep learning models are often highly over parameterized which allows the model to overfit on or even memorize the training data.

Approach

- Randomly set output neurons to zero
  - ØTransforms the network into an ensemble with an exponential set of weaker learners whose parameters are shared.

Usage

- Primarily used in fully connected layers because of the large number of parameters
- Rarely used in convolutional layers
- Rarely used in recurrent neural networks (if at all between the hidden state and output)



2016/06/01

Regularization

Batch Normalization – Avoiding Covariate Shift

Problem

- Deep neural networks suffer from internal covariate shift which makes training harder.

Approach

- Normalize the inputs of each layer (zero mean, unit variance)
  - Regularizes because the training network is no longer producing deterministic values in each layer for a given training example

Usage

- Can be used with all layers (FC, RNN, Conv)
- With Convolutional layers, the mini-batch statistics are computed from all patches in the mini-batch.

Normalize the input X of layer k by the mini-batch moments:

$$\hat{X}^{(k)} = \frac{X^{(k)} - m_X^{(k)}}{s_X^{(k)}}$$

The next step gives the model the freedom of learning to undo the normalization if needed:

$$\tilde{X}^{(k)} = g^{(k)} \hat{X}^{(k)} + b^{(k)}$$

The above two steps in one formula.

$$\tilde{X}^{(k)} = g^{(k)} \times \frac{X^{(k)}}{s_X^{(k)}} + b^{(k)} - g^{(k)} \times \frac{m_X^{(k)}}{s_X^{(k)}}$$

Note: At inference time, an unbiased estimate of the mean and standard deviation computed from all seen mini-batches during training is used.

Deep Learning

Distributed Training

http://engineering.skymind.io/distributed-deep-learning-part-1-an-introduction-to-distributed-training-of-neural-networks

Deep Learning

Part III

Deep Learning and Artificial (General) Intelligence

Part III – Deep Learning and Artificial (General) Intelligence

Deep Reinforcement Learning

- Brief introduction to the problem setting.
- End-to-End models for control
- Resources

Deep Learning as Building Block for Artificial Intelligence

- Think it over - Not all classifications happen in an blink of an eye.
- Store and retrieve important information dynamically – Managing explicit memories
- Considering long-term consequences - Simulating before acting
- Being a multi talent – Multi-task learning and transfer learning

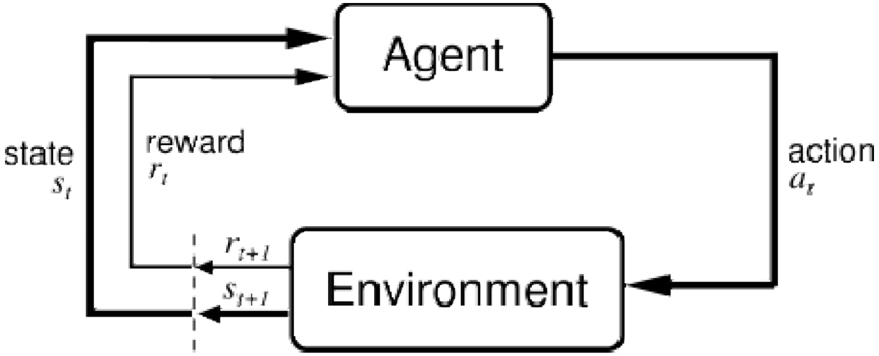
Deep Learning + Reinforcement Learning

=

Deep Reinforcement Learning

Deep Reinforcement Learning

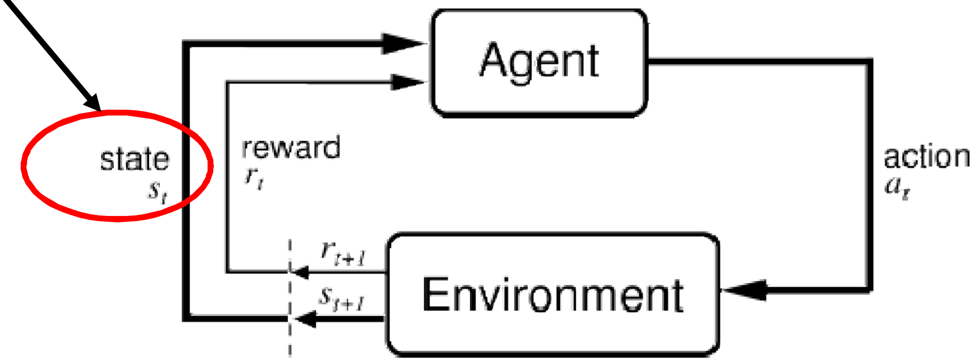
The Reinforcement Learning Setting



Deep Reinforcement Learning

The Reinforcement Learning Setting

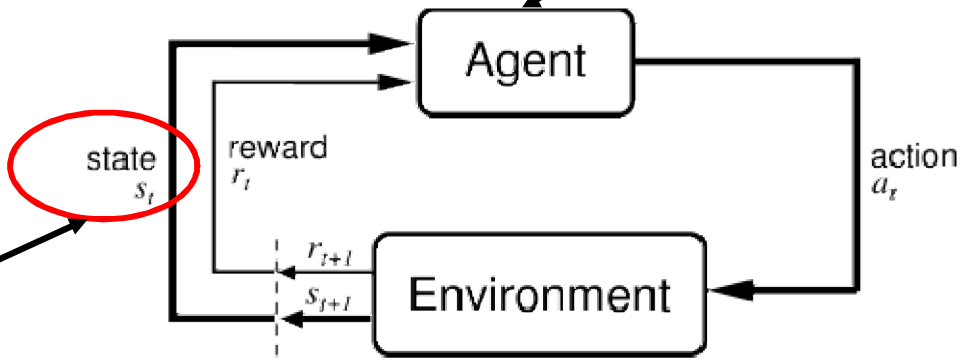
Carefully and often manually designed state representation



Deep Reinforcement Learning

Model Free Deep Reinforcement Learning

Use deep learning to automatically extract meaningful features from the state representation.



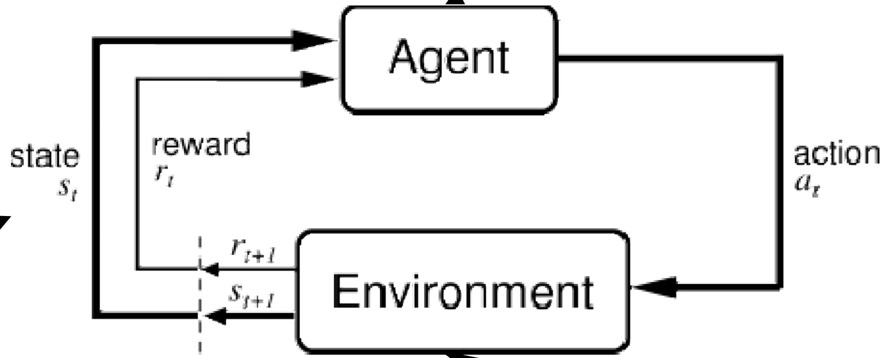
State representation consists of ‘raw’ observations from the environment.

Note: ‚Raw‘ is not meant literally here, you still have to preprocess the data in a reasonable way.

Deep Reinforcement Learning

Model Based Deep Reinforcement Learning

Use deep learning to automatically extract meaningful features from the state representation.



State representation consists of ‘raw’ observations from the environment.

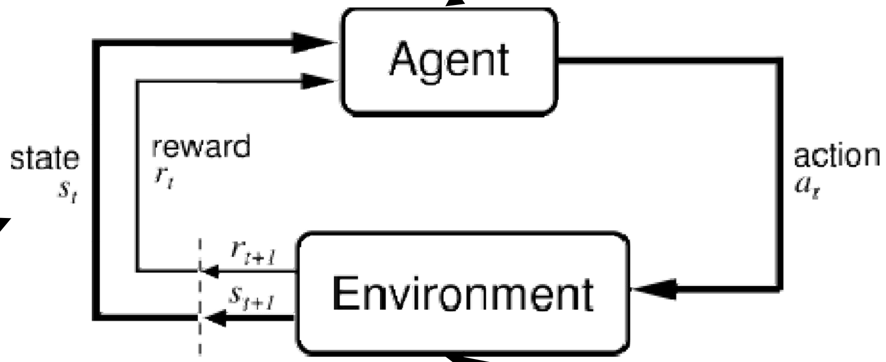
Note: ‚Raw‘ is not meant literally here, you still have to preprocess the data in a reasonable way.

Use Deep Learning to learn a simulator for the environment

Deep Reinforcement Learning

Model Based Deep Reinforcement Learning

Use deep learning to perform planning.



State representation consists of ‘raw’ observations from the environment.

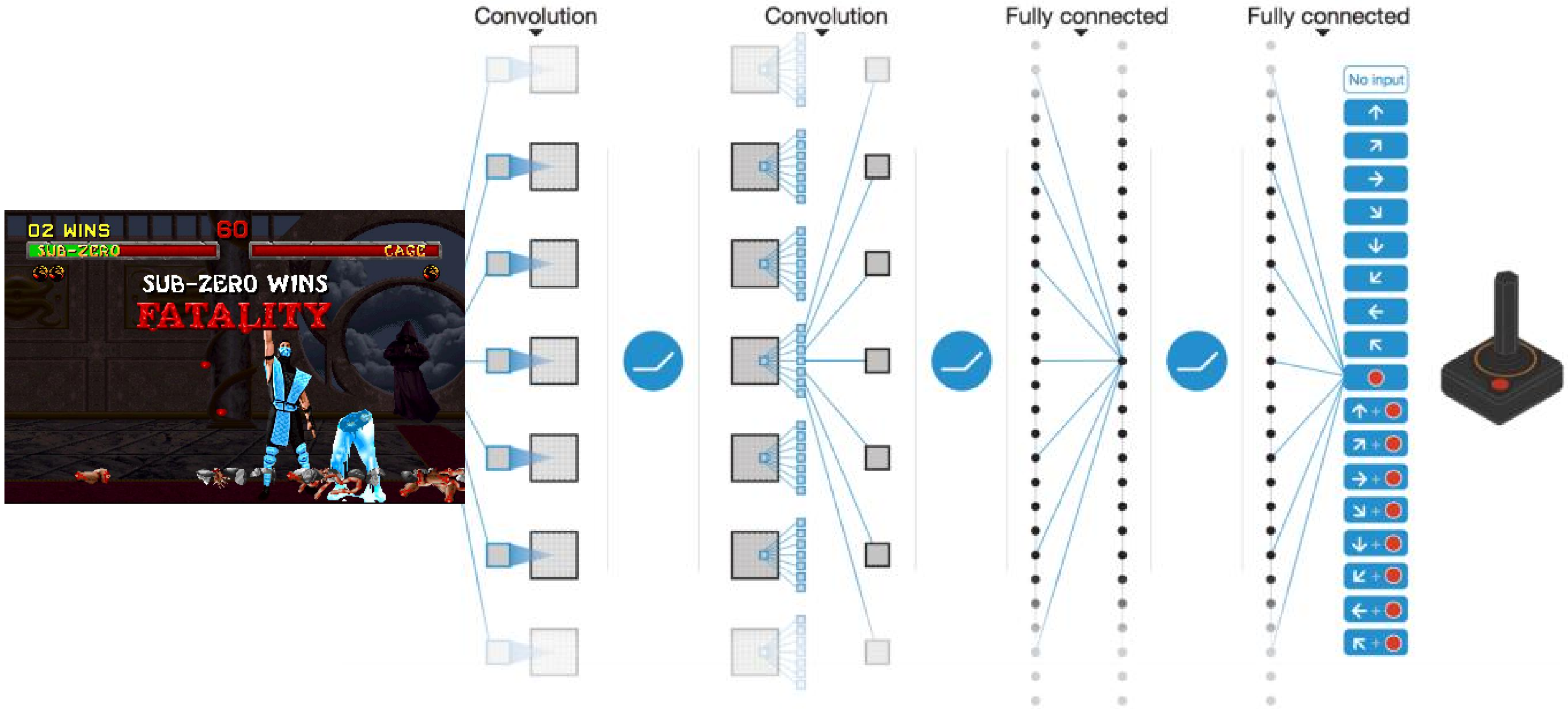
Note: ‚Raw‘ is not meant literally here, you still have to preprocess the data in a reasonable way.

Use Deep Learning to learn a simulator for the environment

Deep Reinforcement Learning

End-to-End Models for Control

Moving away from feeding carefully extracted manual (state) features into the models.

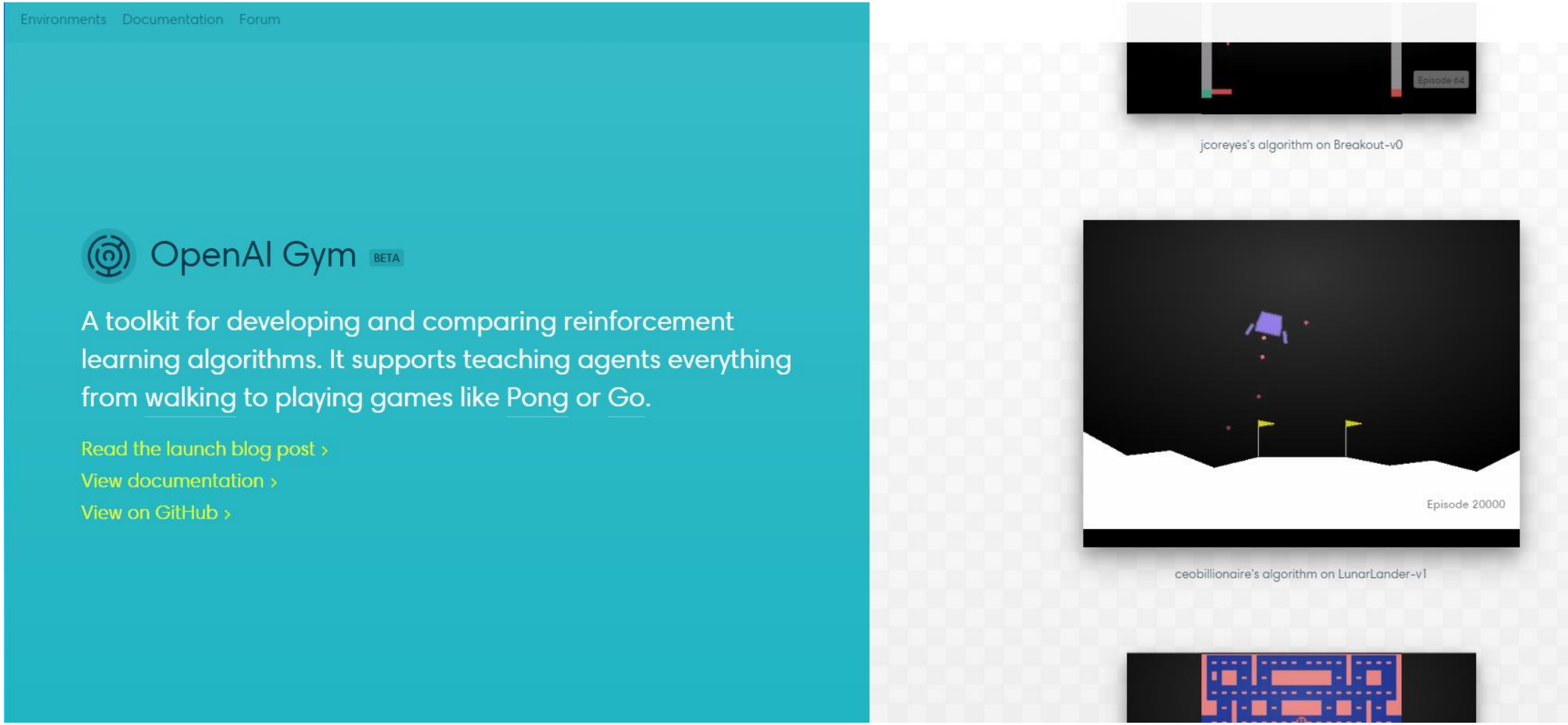


See also: Facebook and Intel reign supreme in 'Doom' AI deathmatch.

https://www.engadget.com/2016/09/22/facebook-and-intel-reign-supreme-in-doom-ai-deathmatch/

Deep Reinforcement Learning

Resources



Environments Documentation Forum

OpenAI Gym BETA

A toolkit for developing and comparing reinforcement learning algorithms. It supports teaching agents everything from walking to playing games like Pong or Go.

Read the launch blog post ›

View documentation ›

View on GitHub ›

Episode 64
jcoreyes's algorithm on Breakout-v0
Episode 20000
ceobillionaire's algorithm on LunarLander-v1

https://gym.openai.com/

2016/06/01

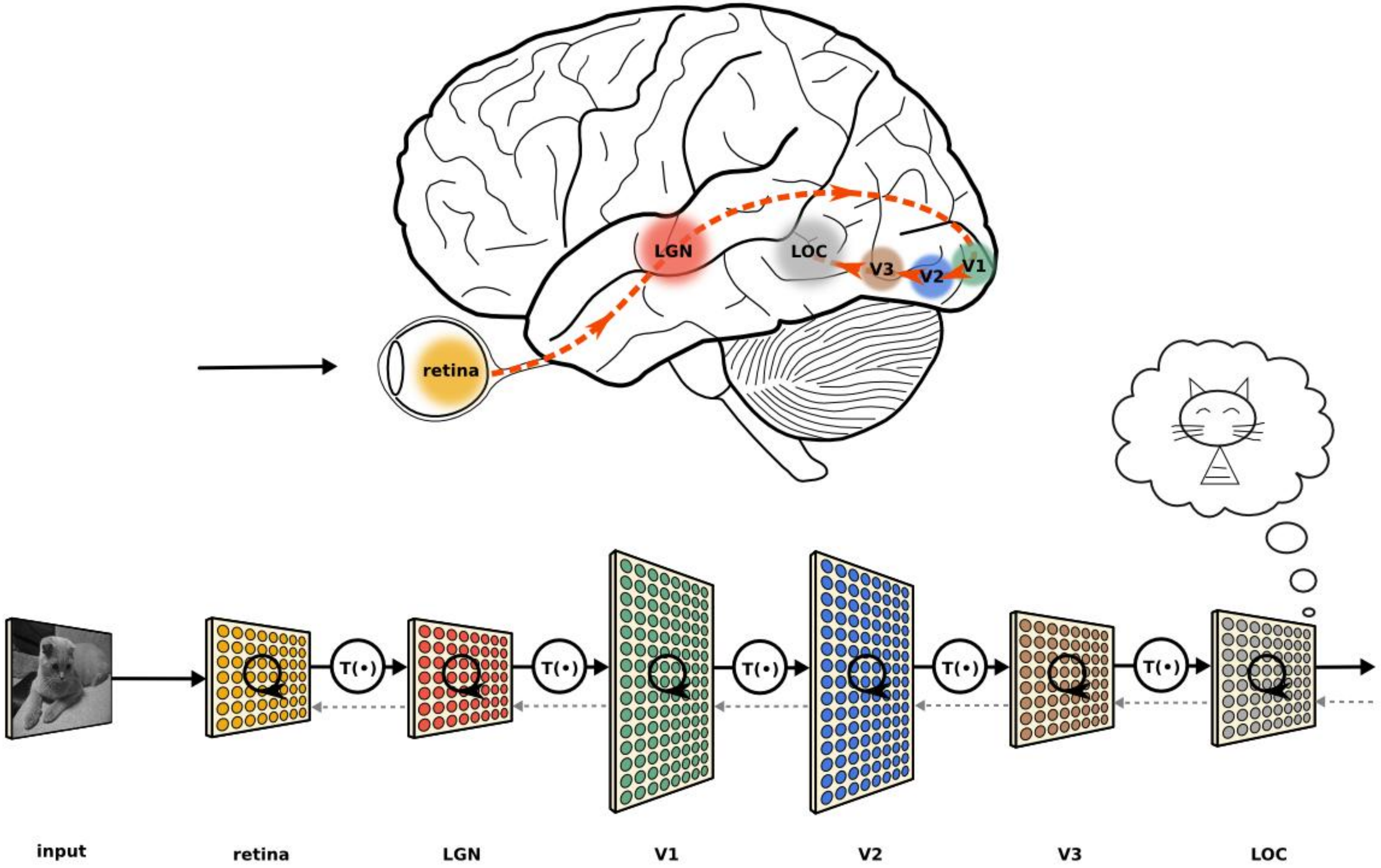
Deep Learning as Building Block for

Artificial Intelligence

Deep Learning as Building Block for AI

Think it over: Not All Decisions Happen in an Blink of An Eye.

Hierarchical feature extraction in the visual cortex.



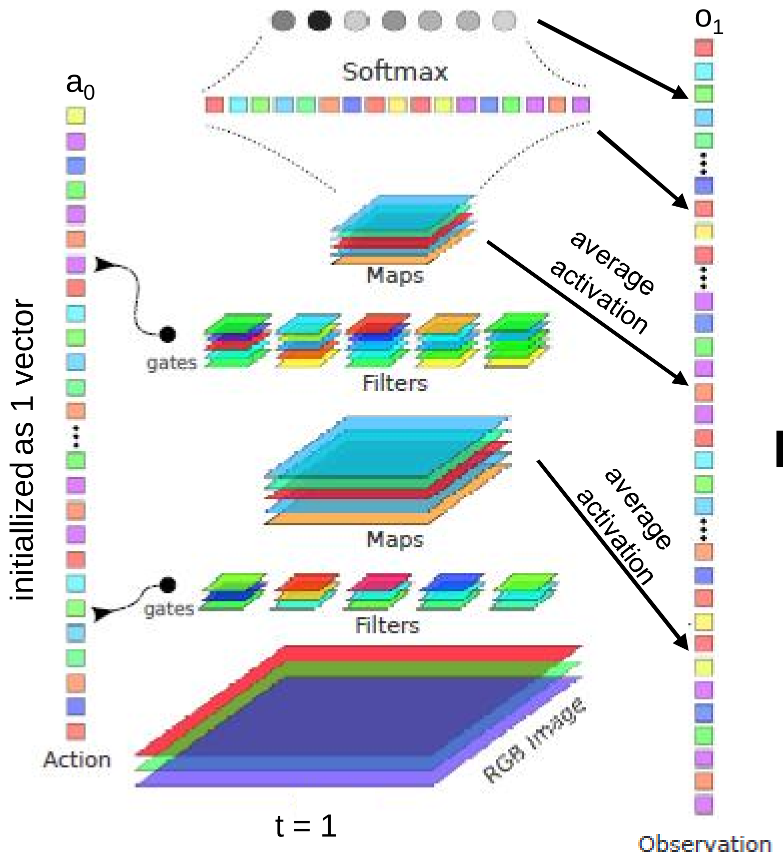
“For some types of tasks (e.g. for images presented briefly and out of context), it is thought that visual processing in the brain is hierarchical–one layer feeds into the next, computing progressively more complex features. This is the inspiration for the “layered” design of modern feed-forward neural networks.” Image (c) Jonas Kubilias

2016/06/01

Deep Learning as Building Block for AI

Non-Stationary Feed-forward Passes in Deep Neural Networks.

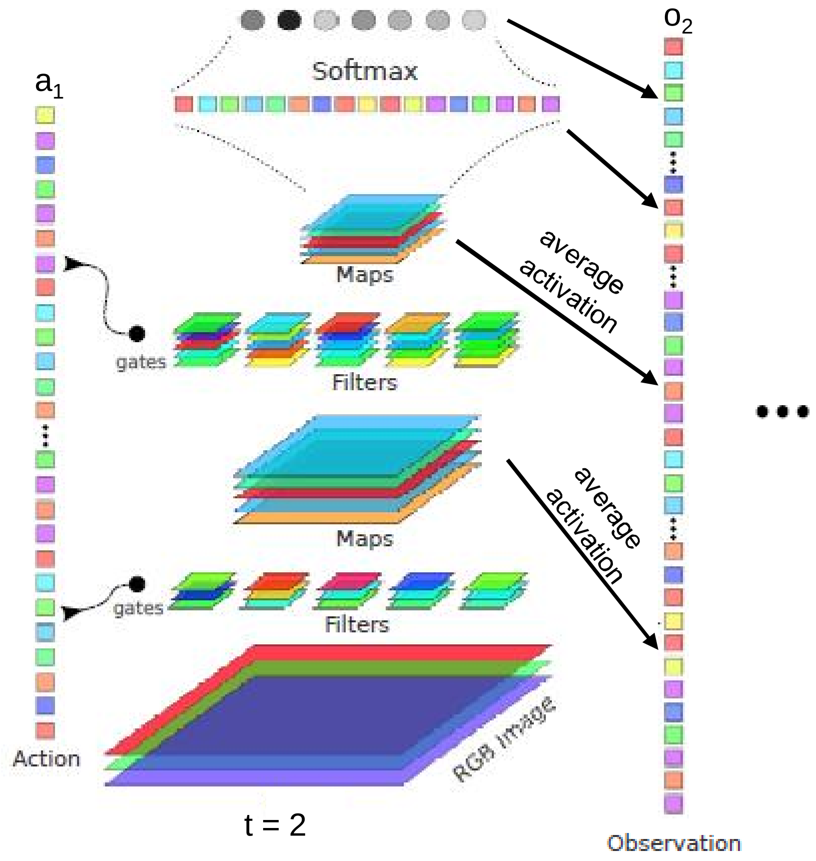
Multiple passes of an image through a network to reevaluate the final decision



Softmax

$$\pi_\theta(\mathbf{o}) = dim(A)\sigma(\boldsymbol{\theta}\ \mathbf{o_t}) = \mathbf{a_t}$$

Sampled from a Gaussian distribution, which is learned during training.



Deep Networks with Internal Selective Attention through Feedback Connections

Marijn Stolenga, Jonathan Masci, Faustino Gomez, Juergen Schmidhuber. https://arxiv.org/abs/1407.3068. 2014

Deep Learning as Building Block for AI

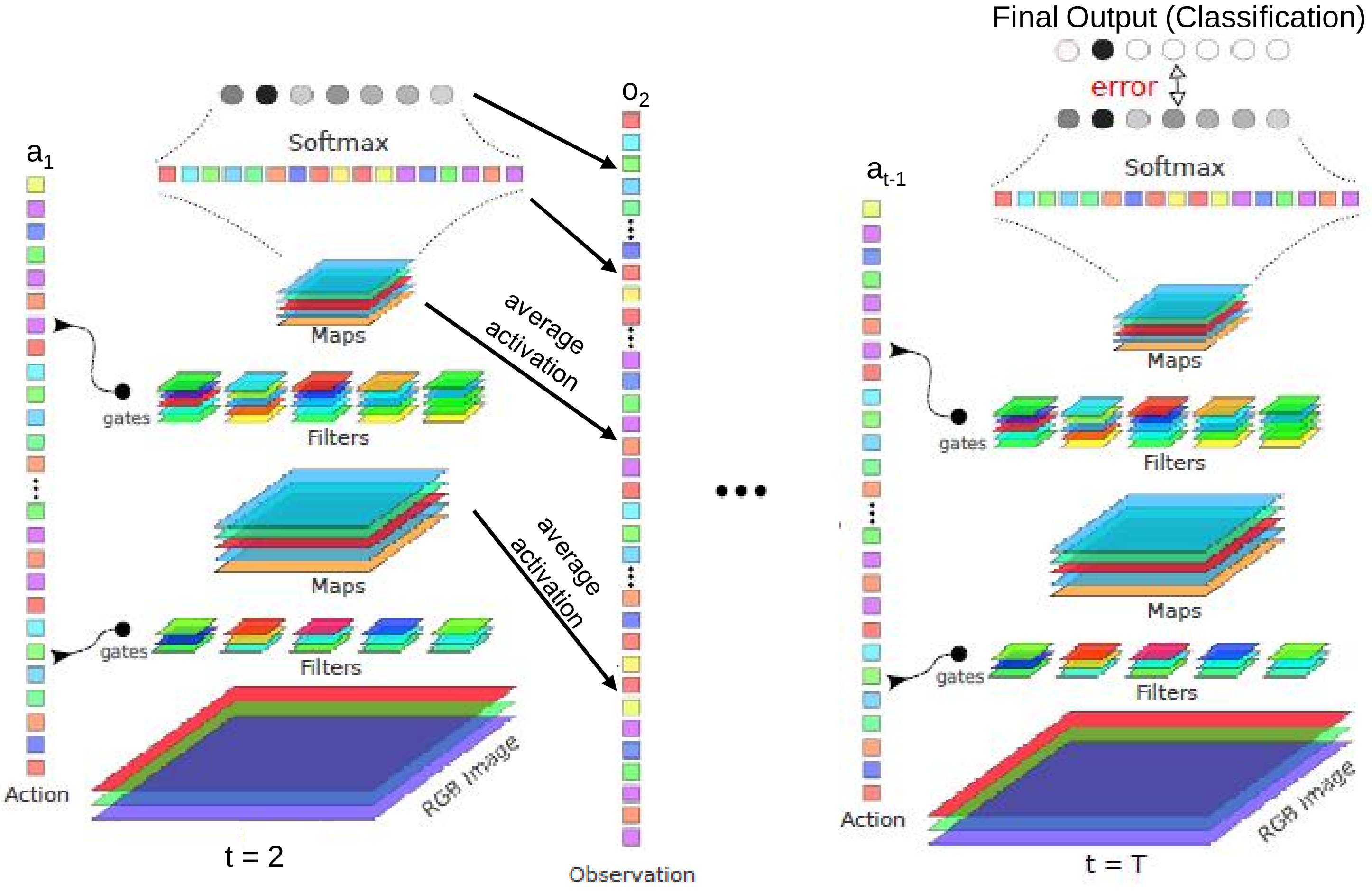
Non-Stationary Feed-Forward Passes in Deep Neural Networks.

Softmax

$$\pi_\theta\ (\mathbf{o}) = dim(A)\sigma(\theta\ \mathbf{o_t}) = \mathbf{a_t}$$

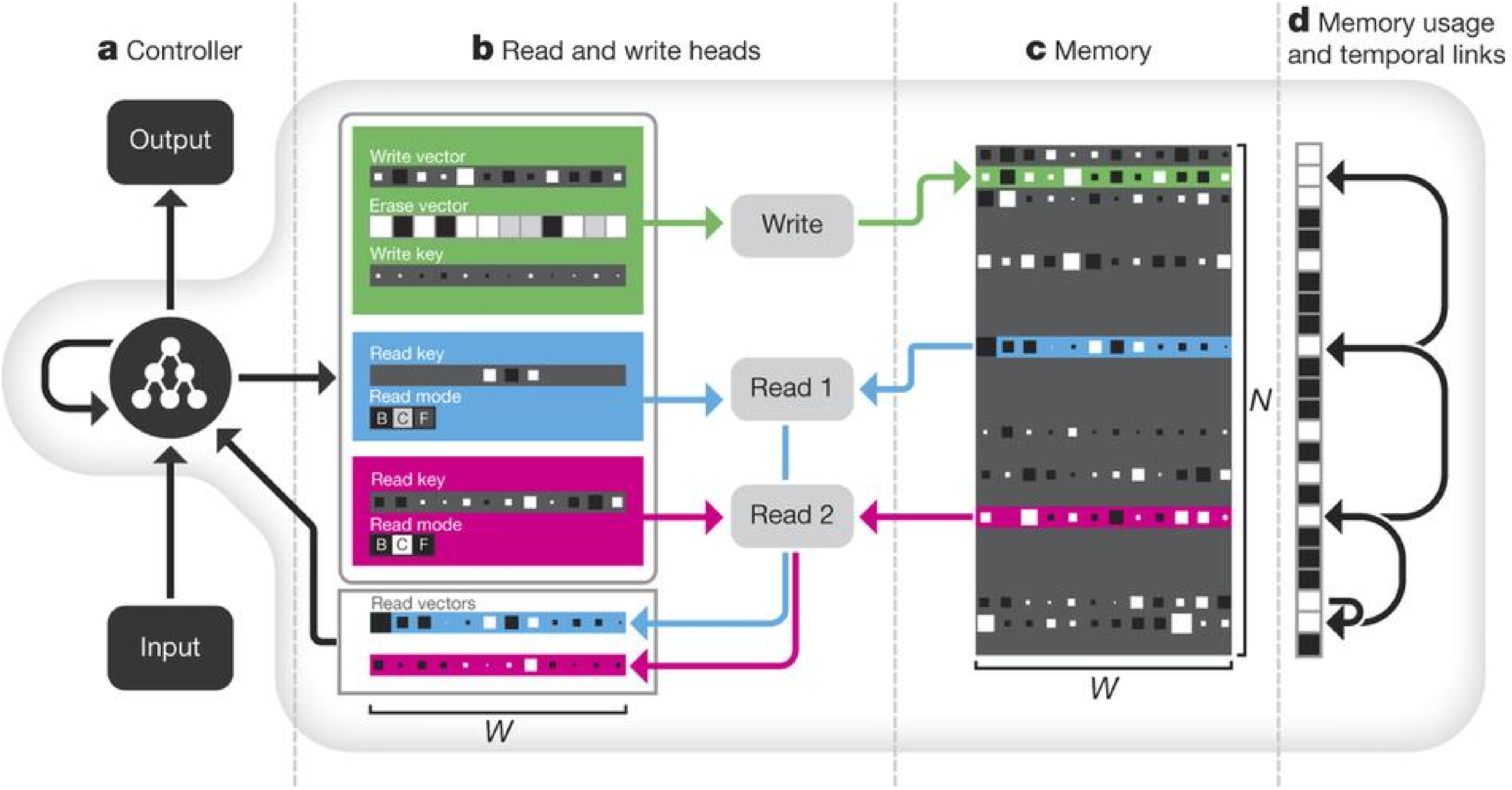
Sampled from a Gaussian distribution, which is learned during training

Multiple passes of an image through a network to reevaluate the final decision



Deep Learning as Building Block for AI

Store and Retrieve Important Information Dynamically – Managing Explicit Memories



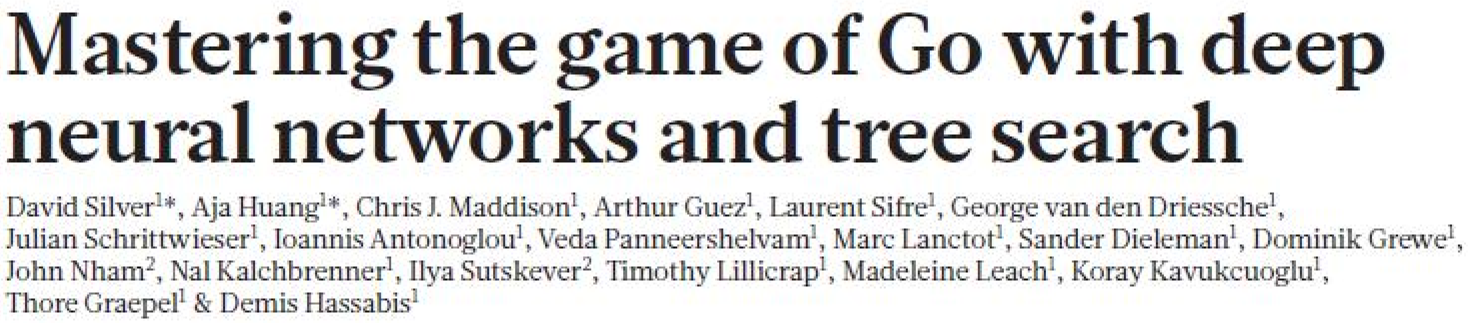
Hybrid computing using a neural network with dynamic external memory

Alex Graves et. Al. (Nature 2016)

2016/06/01

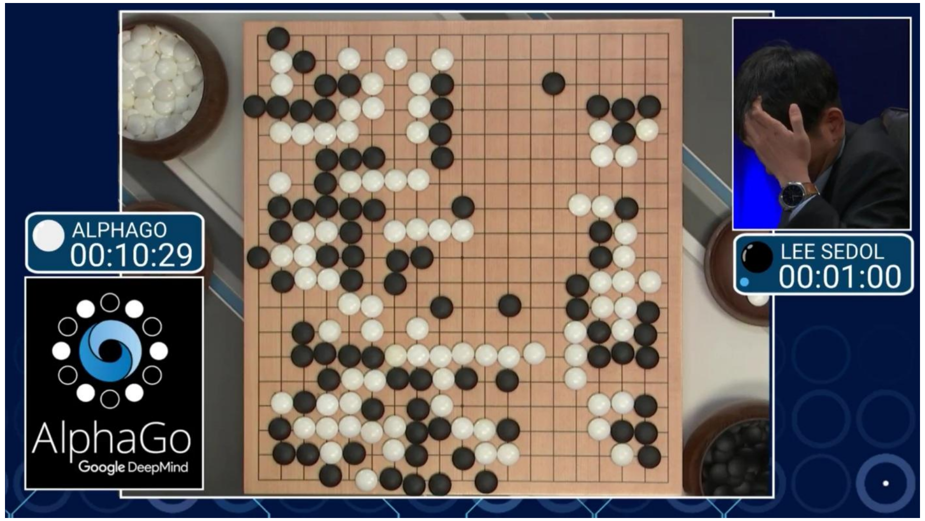
Deep Learning as Building Block for AI

Considering Long-Term Consequences – Simulating Before Acting



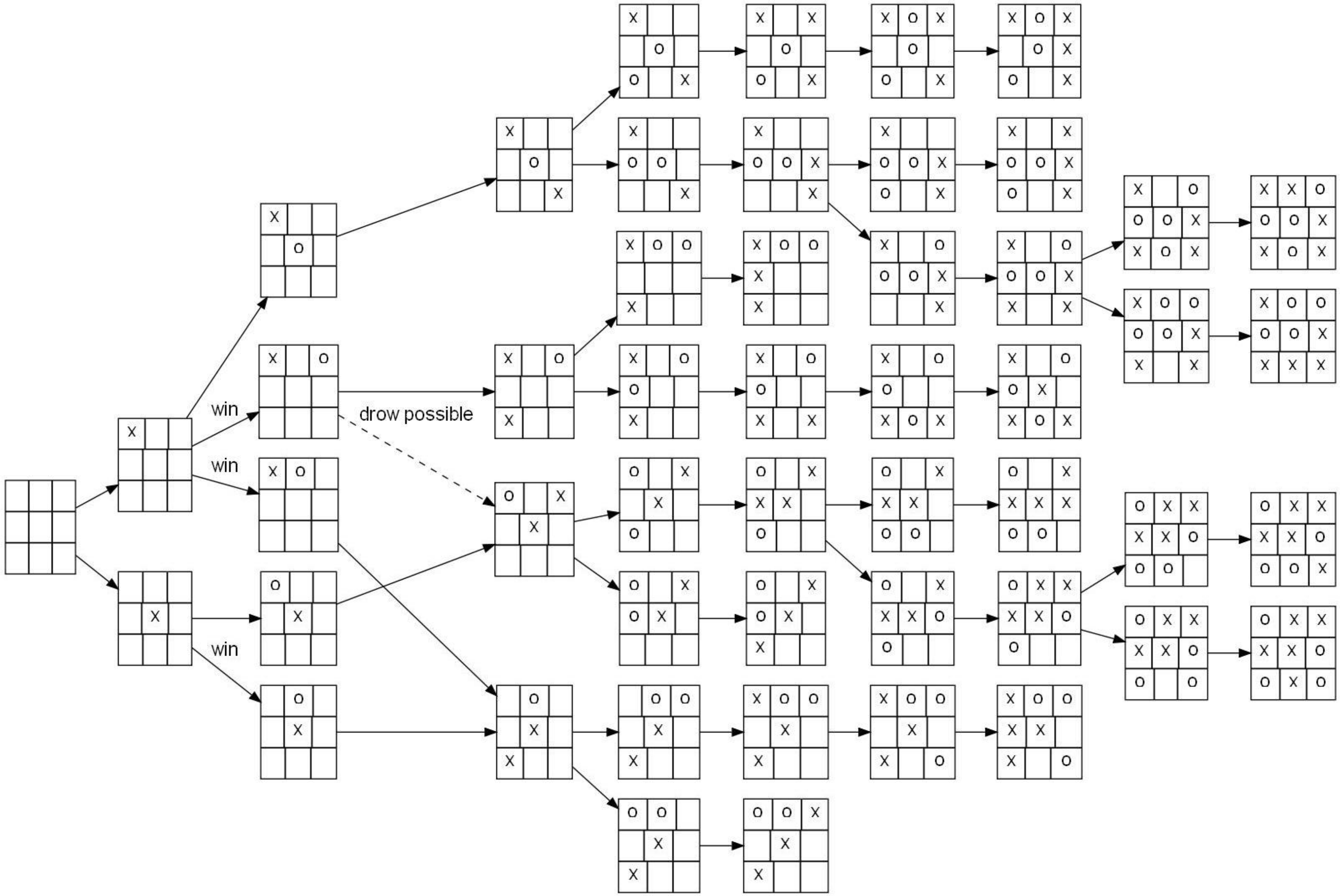
Mastering the game of Go with deep neural networks and tree search

David Silver[1]*, Aja Huang[1]*, Chris J. Maddison[1], Arthur Guez[1], Laurent Sifre[1], George van den Driessche[1], Julian Schrittwieser[1], Ioannis Antonoglou[1], Veda Panneershelvam[1], Marc Lanctot[1], Sander Dieleman[1], Dominik Grewe[1], John Nham[2], Nal Kalchbrenner[1], Ilya Sutskever[2], Timothy Lillicrap[1], Madeleine Leach[1], Koray Kavukcuoglu[1], Thore Graepel[1] & Demis Hassabis[1]



Deep Learning as Building Block for AI

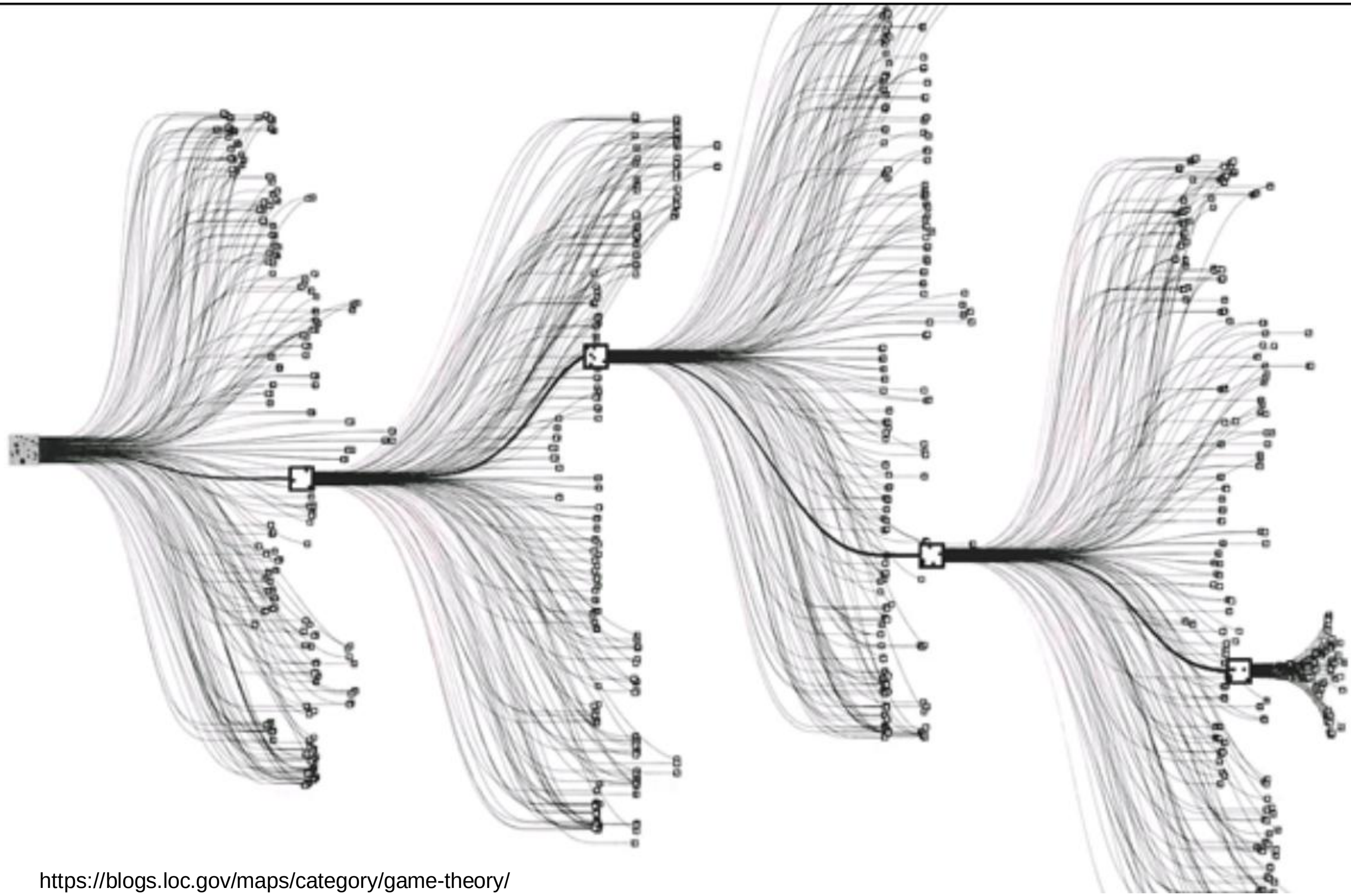
Planning in Perfect Information Games



2016/06/01

Deep Learning as Building Block for AI

Dealing with Intractable Many Game States



https://blogs.loc.gov/maps/category/game-theory/

http://paulomenin.github.io/go-presentation/images/goban.png

As in many real life settings, the whole game tree cannot be explored. For this reason we need automated methods that help to explore the game tree in a reasonable way!

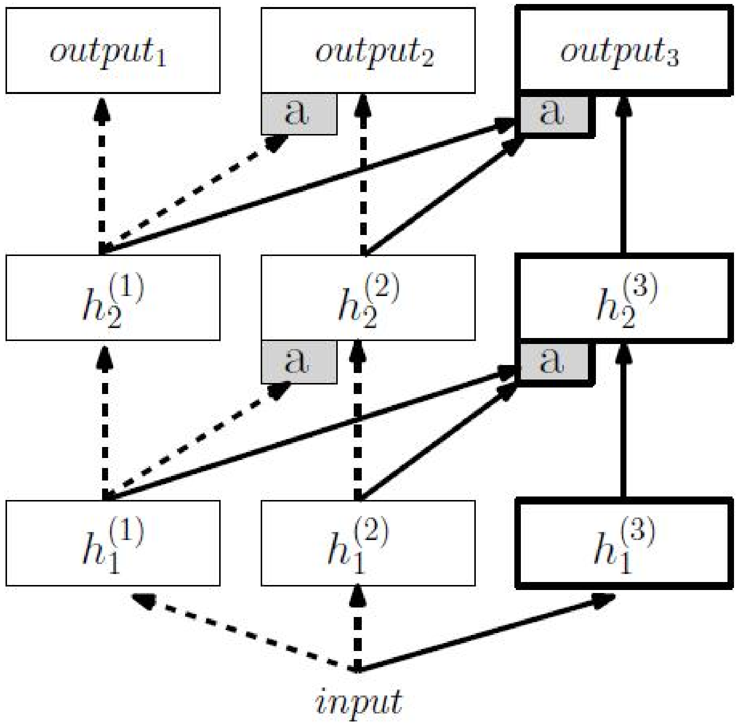
=> Deep Learning

Deep Learning as Building Block for AI

Being a Multi-Talent – Multi Task Learning and Transfer learning

Challenge:
Today, we are able to train systems that sometimes show super human performance in very complex tasks. (E.g AlphaGO)

However, the same systems fail miserably when directly applied to any other (much simpler task).



Progressive Neural Networks

Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, Raia Hadsell

arXiv:1606.04671, 2016

Things we did not cover (not complete…)

Neural Artistic Style Transfer

Benchmark datasets

Encoder-Decoder Networks

Variational Approaches

Fractal Networks

Siamese Networks

Pre-Training

Mask R-CNN

Sequence Generation

Neural Question Answering

Initialization

Sequence Generation

Learning to learn

Maxout Networks

Vanishing/Exploding Gradient

Highway Networks

Dealing with Variable Length Inputs and Outputs

Transfer Learning

Mechanism for training ultra deep networks

Hessian-free optimization

Pixel RNN/CNN

Recursive Neural Networks

(Unsupervised) pre-training

Evolutionary Methods for Model Training

Weight Normalization

Distributed Training

Deep Q-Learning

Layer Compression (e.g. Tensor-Trains)

Speech Modeling

Weight Sharing

Character Level Neural Machine Translation

Hyper-parameter tuning

Multi-Lingual Neural Machine Translation

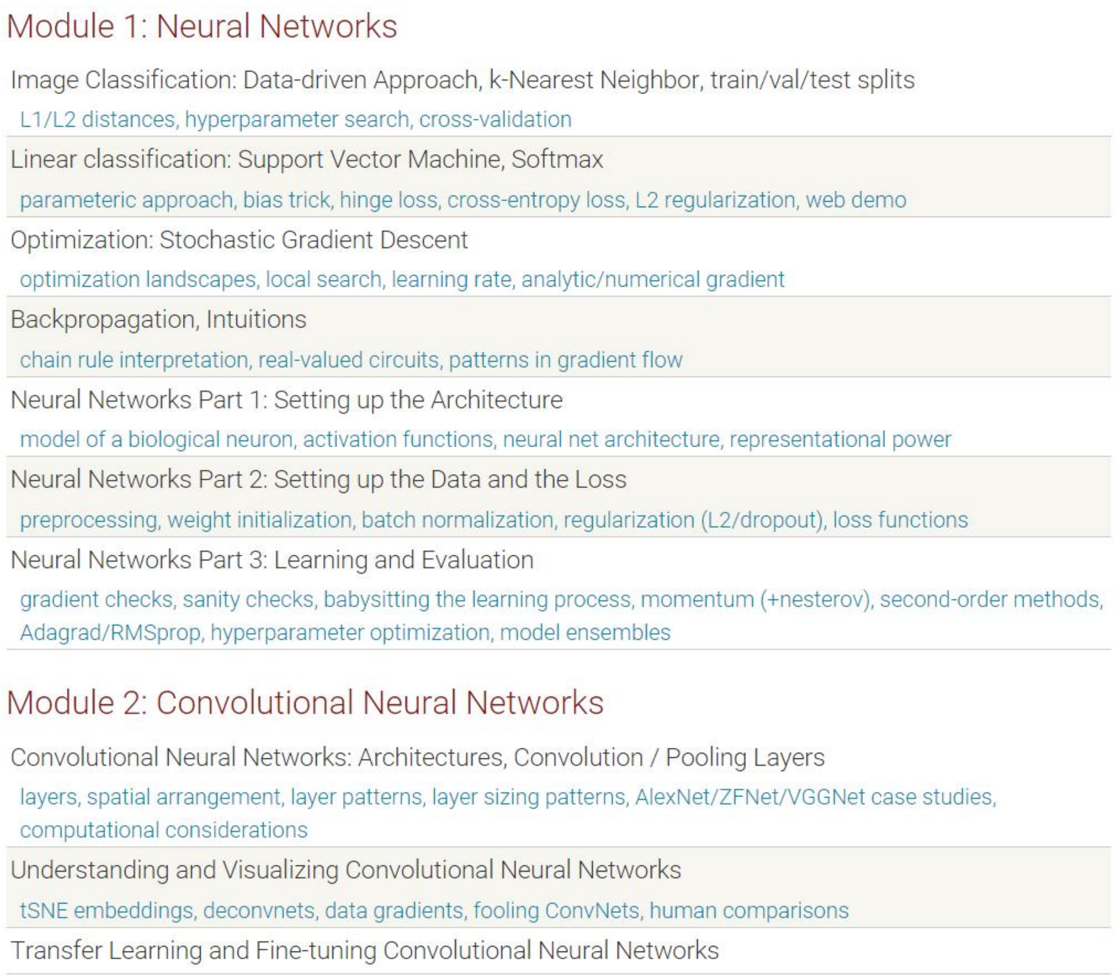
More loss functions

Generative adversarial networks

Deep Learning

Because it Works

Recommended Material



Module 1: Neural Networks

Image Classification: Data-driven Approach, k-Nearest Neighbor, train/val/test splits

L1/L2 distances, hyperparameter search, cross-validation

Linear classification: Support Vector Machine, Softmax

parameteric approach, bias trick, hinge loss, cross-entropy loss, L2 regularization, web demo

Optimization: Stochastic Gradient Descent

optimization landscapes, local search, learning rate, analytic/numerical gradient

Backpropagation, Intuitions

chain rule interpretation, real-valued circuits, patterns in gradient flow

Neural Networks Part 1: Setting up the Architecture

model of a biological neuron, activation functions, neural net architecture, representational power

Neural Networks Part 2: Setting up the Data and the Loss

preprocessing, weight initialization, batch normalization, regularization (L2/dropout), loss functions

Neural Networks Part 3: Learning and Evaluation

gradient checks, sanity checks, babysitting the learning process, momentum (+nesterov), second-order methods, Adagrad/RMSprop, hyperparameter optimization, model ensembles

Module 2: Convolutional Neural Networks

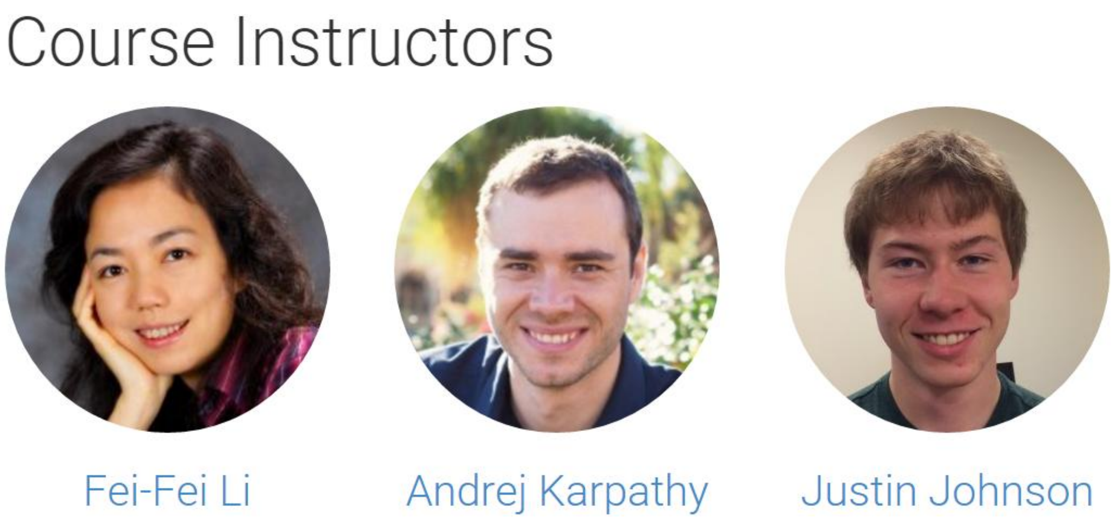
Convolutional Neural Networks: Architectures, Convolution / Pooling Layers

layers, spatial arrangement, layer patterns, layer sizing patterns, AlexNet/ZFNet/VGGNet case studies, computational considerations

Understanding and Visualizing Convolutional Neural Networks

tSNE embeddings, deconvnets, data gradients, fooling ConvNets, human comparisons

Transfer Learning and Fine-tuning Convolutional Neural Networks



Course Instructors

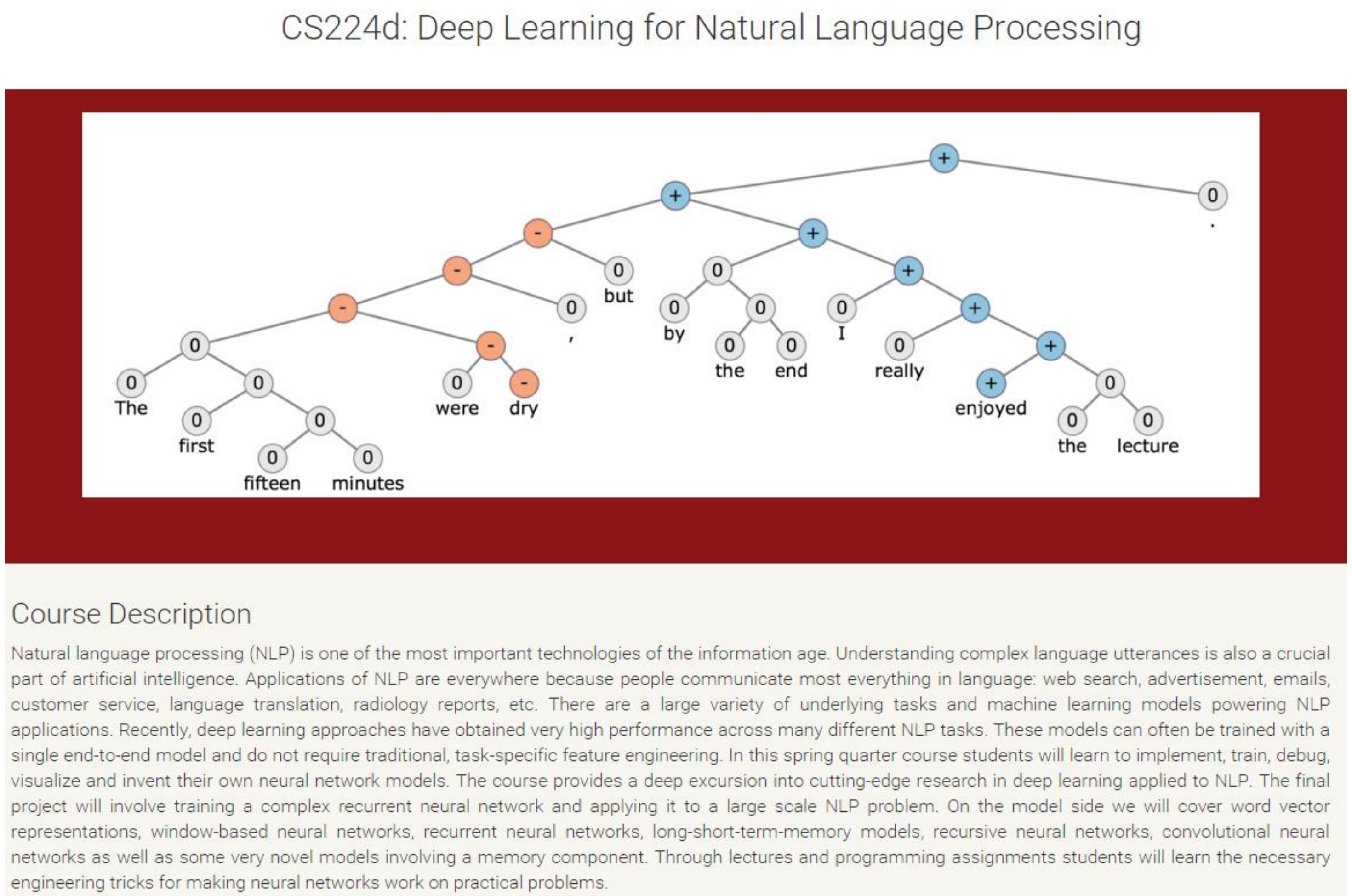
Fei-Fei Li

Andrej Karpathy

Justin Johnson

http://cs231n.stanford.edu/
http://cs231n.github.io

Recommended Material



CS224d: Deep Learning for Natural Language Processing

Course Description

Natural language processing (NLP) is one of the most important technologies of the information age. Understanding complex language utterances is also a crucial part of artificial intelligence. Applications of NLP are everywhere because people communicate most everything in language: web search, advertisement, emails, customer service, language translation, radiology reports, etc. There are a large variety of underlying tasks and machine learning models powering NLP applications. Recently, deep learning approaches have obtained very high performance across many different NLP tasks. These models can often be trained with a single end-to-end model and do not require traditional, task-specific feature engineering. In this spring quarter course students will learn to implement, train, debug, visualize and invent their own neural network models. The course provides a deep excursion into cutting-edge research in deep learning applied to NLP. The final project will involve training a complex recurrent neural network and applying it to a large scale NLP problem. On the model side we will cover word vector representations, window-based neural networks, recurrent neural networks, long-short-term-memory models, recursive neural networks, convolutional neural networks as well as some very novel models involving a memory component. Through lectures and programming assignments students will learn the necessary engineering tricks for making neural networks work on practical problems.

Course Instructor

Richard Socher

http://cs224d.stanford.edu/

Recommended Material

INTRODUCTION

- Tutorial on Neural Networks (Deep Learning and Unsupervised Feature Learning): http://deeplearning.stanford.edu/wiki/index.php/UFLDL_Tutorial
- Deep Learning for Computer Vision lecture: http://cs231n.stanford.edu (http://cs231n.github.io)
- Deep Learning for NLP lecture: http://cs224d.stanford.edu (http://cs224d.stanford.edu/syllabus.html)
- Deep Learning for NLP (without magic) tutorial: http://lxmls.it.pt/2014/socher-lxmls.pdf (Videos from NAACL 2013: http://nlp.stanford.edu/courses/NAACL2013)
- Bengio's Deep Learning book: http://www.deeplearningbook.org

Recommended Material

PARAMETER INITIALIZATION

- Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." *International Conference on Artificial Intelligence and Statistics*. 2010.
- He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *Proceedings of the IEEE International Conference on Computer Vision* (pp. 1026-1034).

BATCH NORMALIZATION

- Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of The 32nd International Conference on Machine Learning* (pp. 448-456).
- Cooijmans, T., Ballas, N., Laurent, C., & Courville, A. (2016). Recurrent Batch Normalization. *arXiv preprint arXiv:1603.09025*.

DROPOUT

- Hinton, Geoffrey E., et al. "Improving neural networks by preventing co-adaptation of feature detectors." *arXiv preprint arXiv:1207.0580* (2012).
- Srivastava, Nitish, et al. "Dropout: A simple way to prevent neural networks from overfitting." *The Journal of Machine Learning Research* 15.1 (2014): 1929-1958.

Recommended Material

OPTIMIZATION & TRAINING

- Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, *12*, 2121-2159.
- Zeiler, M. D. (2012). ADADELTA: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.
- Tieleman, T., & Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, *4*, 2.
- Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning (ICML)* (pp. 1139-1147).
- Kingma, D., & Ba, J. (2014). Adam: A method for stochastic optimization.*arXiv preprint arXiv:1412.6980*.
- Martens, J., & Sutskever, I. (2012). Training deep and recurrent networks with hessian-free optimization. In *Neural networks: Tricks of the trade* (pp. 479-535). Springer Berlin Heidelberg.

Recommended Material

COMPUTER VISION

- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems* (pp. 1097-1105).
- Taigman, Y., Yang, M., Ranzato, M. A., & Wolf, L. (2014). DeepFace: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 1701-1708).
- Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 1-9).
- Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.
- Jaderberg, M., Simonyan, K., & Zisserman, A. (2015). Spatial transformer networks. In *Advances in Neural Information Processing Systems* (pp. 2008-2016).
- Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems* (pp. 91-99).
- Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., ... & Bengio, Y. (2015). Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. In *Proceedings of The 32nd International Conference on Machine Learning* (pp. 2048-2057).
- Johnson, J., Karpathy, A., & Fei-Fei, L. (2015). DenseCap: Fully Convolutional Localization Networks for Dense Captioning. *arXiv preprint arXiv:1511.07571*.

Recommended Material

NATURAL LANGUAGE PROCESSING

- Bengio, Y., Schwenk, H., Senécal, J. S., Morin, F., & Gauvain, J. L. (2006). Neural probabilistic language models. In *Innovations in Machine Learning* (pp. 137-186). Springer Berlin Heidelberg.
- Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (2011). Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, *12*, 2493-2537.
- Mikolov, T. (2012). *Statistical language models based on neural networks* (Doctoral dissertation, PhD thesis, Brno University of Technology. 2012.)
- Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.
- Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems* (pp. 3111-3119).
- Mikolov, T., Yih, W. T., & Zweig, G. (2013). Linguistic Regularities in Continuous Space Word Representations. In *HLT-NAACL* (pp. 746-751).
- Socher, R. (2014). *Recursive Deep Learning for Natural Language Processing and Computer Vision* (Doctoral dissertation, Stanford University).
- Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.
- Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.