NEURAL NETWORKS

Deep learning = Deep neural networks = neural networks

DNNs (Deep Neural Networks)

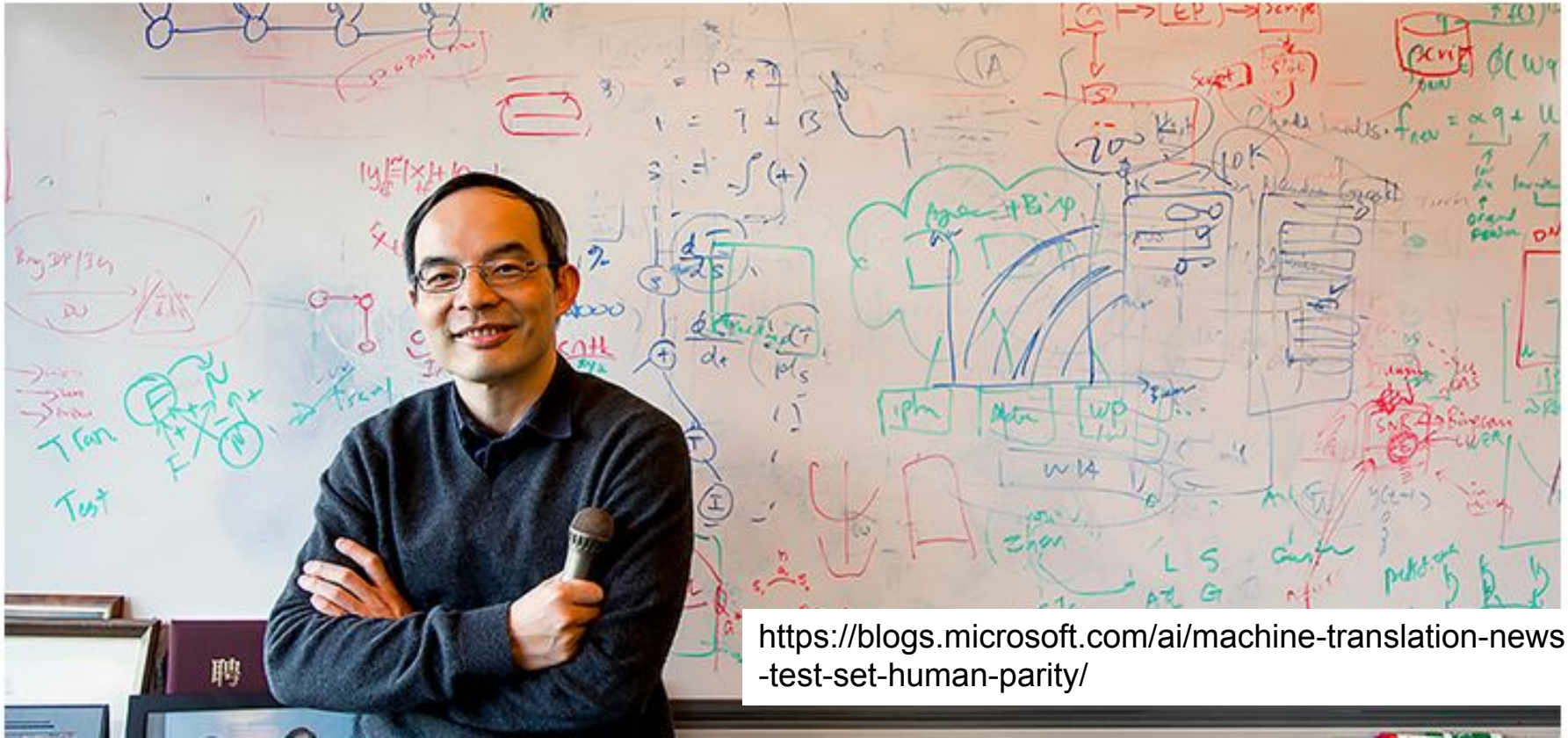
- Why deep learning?
- Greatly improved performance in ASR and other tasks (Computer Vision, Robotics, Machine Translation, NLP, etc.)
- Surpassed human performance in many tasks

Task	Previous state-of-the-art	Deep learning (2012)	Deep learning (2017)
TIMIT	24.4%	20.0%	17.0%
Switchboard	23.6%	16.1%	5.5%
Google voice search	16.0%	12.3%	4.9%

Geoffrey Hinton, Deep Neural Networks for Acoustic Modeling in Speech Recognition, 2012

Microsoft reaches a historic milestone, using AI to match human performance in translating news from Chinese to English

Mar 14, 2018 | Allison Linn



https://blogs.microsoft.com/ai/machine-translation-news-test-set-human-parity/

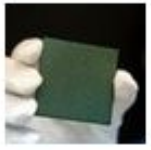
Google's AlphaGo Defeats Chinese Go Master in Win for A.I.

点击查看本文中文版

By PAUL MOZUR MAY 23, 2017

RELATED COVERAGE


A.I. I
Repla


China
FEB. 3,


THE FU
The F

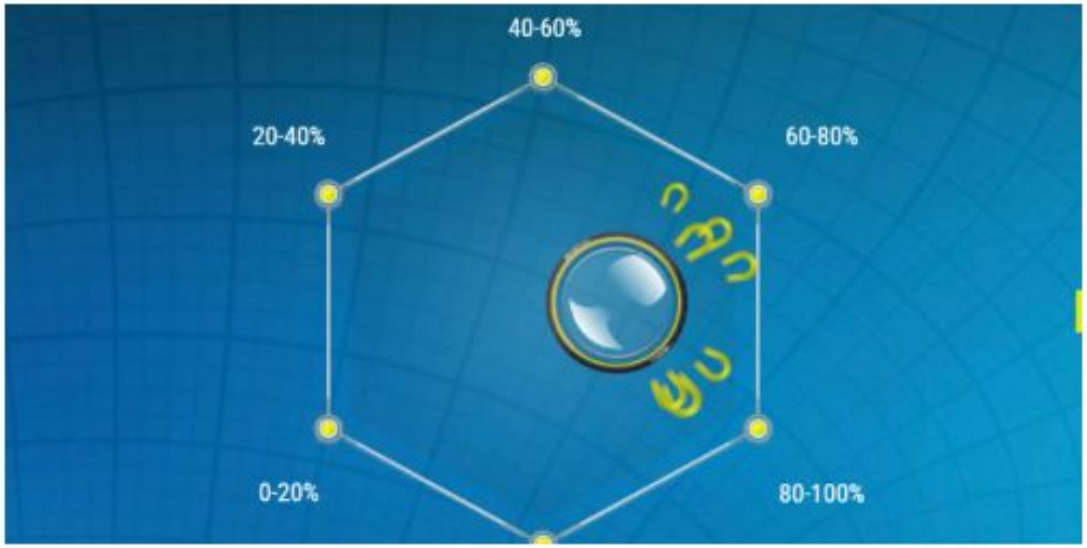

Maste
Goog

https://www.nytimes.com/2017/05/23/business/google-deepmind-alphago-go-champion-defeat.html

The Stanford Daily

Artificial swarm intelligence diagnoses pneumonia better than individual computer or doctor

by Fan Liu — September 27, 2018 0



Courtesy of Unanimous AI

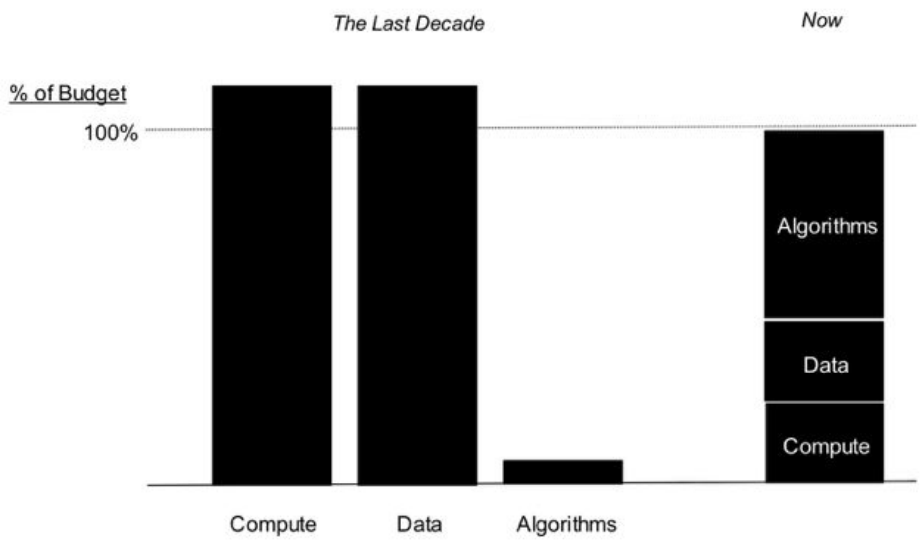
Popular Posts

Artificial swarm intelligence diagnoses pneumonia better than individual computer or

https://www.stanforddaily.com/2018/09/27/artificial-swarm-intelligence-diagnoses-pneumonia-better-than-individual-computer-or-doctor/

Why now

- Neural Networks has been around since 1990s
- Big data – DNN can take advantage of large amounts of data better than other models
- GPU – Enable training bigger models possible
- Deep – Easier to avoid bad local minima when the model is large



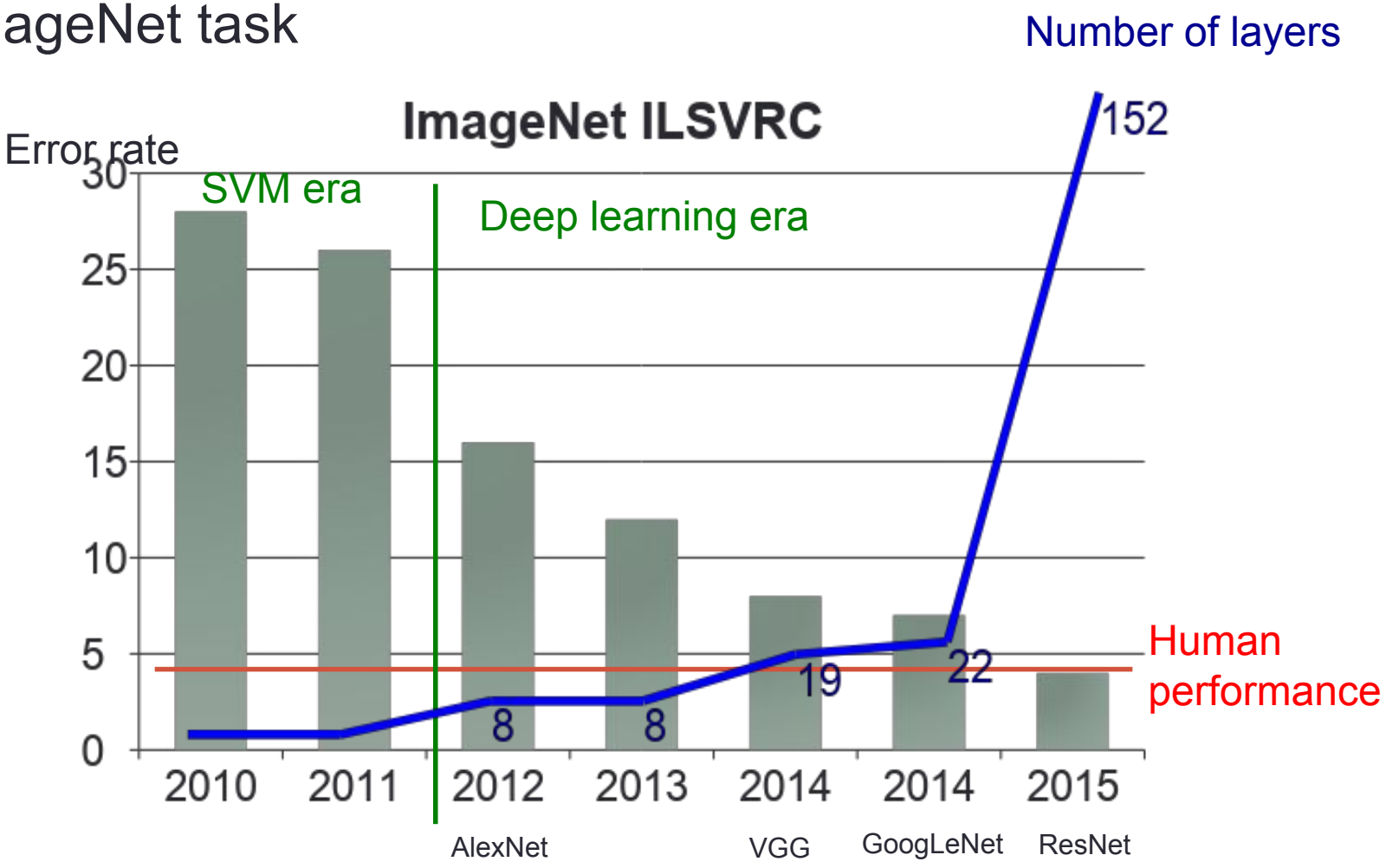
http://www.kdnuggets.com/2017/06/practical-guide-machine-learning-understand-differentiate-apply.html

ImageNet - Object classification

Alex, Krizhevsky, Imagenet classification with deep convolutional neural networks, 2012

Wider and deeper networks

- ImageNet task



Olga Russakovsky, ImageNet Large Scale Visual Recognition Challenge, 2014 https://arxiv.org/abs/1409.0575

Statistics > Machine Learning

Dynamical Isometry and a Mean Field Theory of CNNs: How to Train 10,000-Layer Vanilla Convolutional Neural Networks

Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel S. Schoenholz, Jeffrey Pennington

(Submitted on 14 Jun 2018)

In recent years, state-of-the-art methods in computer vision have utilized increasingly deep convolutional neural network architectures (CNNs), with some of the most successful models employing hundreds or even thousands of layers. A variety of pathologies such as vanishing/exploding gradients make training such deep networks challenging. While residual connections and batch normalization do enable training at these depths, it has remained unclear whether such specialized architecture designs are truly necessary to train deep CNNs. In this work, we demonstrate that it is possible to train vanilla CNNs with ten thousand layers or more simply by using an appropriate initialization scheme. We derive this initialization scheme theoretically by developing a mean field theory for signal propagation and by characterizing the conditions for dynamical isometry, the equilibration of singular values of the input-output Jacobian matrix. These conditions require that the convolution operator be an orthogonal transformation in the sense that it is norm-preserving. We present an algorithm for generating such random initial orthogonal convolution kernels and demonstrate empirically that they enable efficient training of extremely deep architectures.

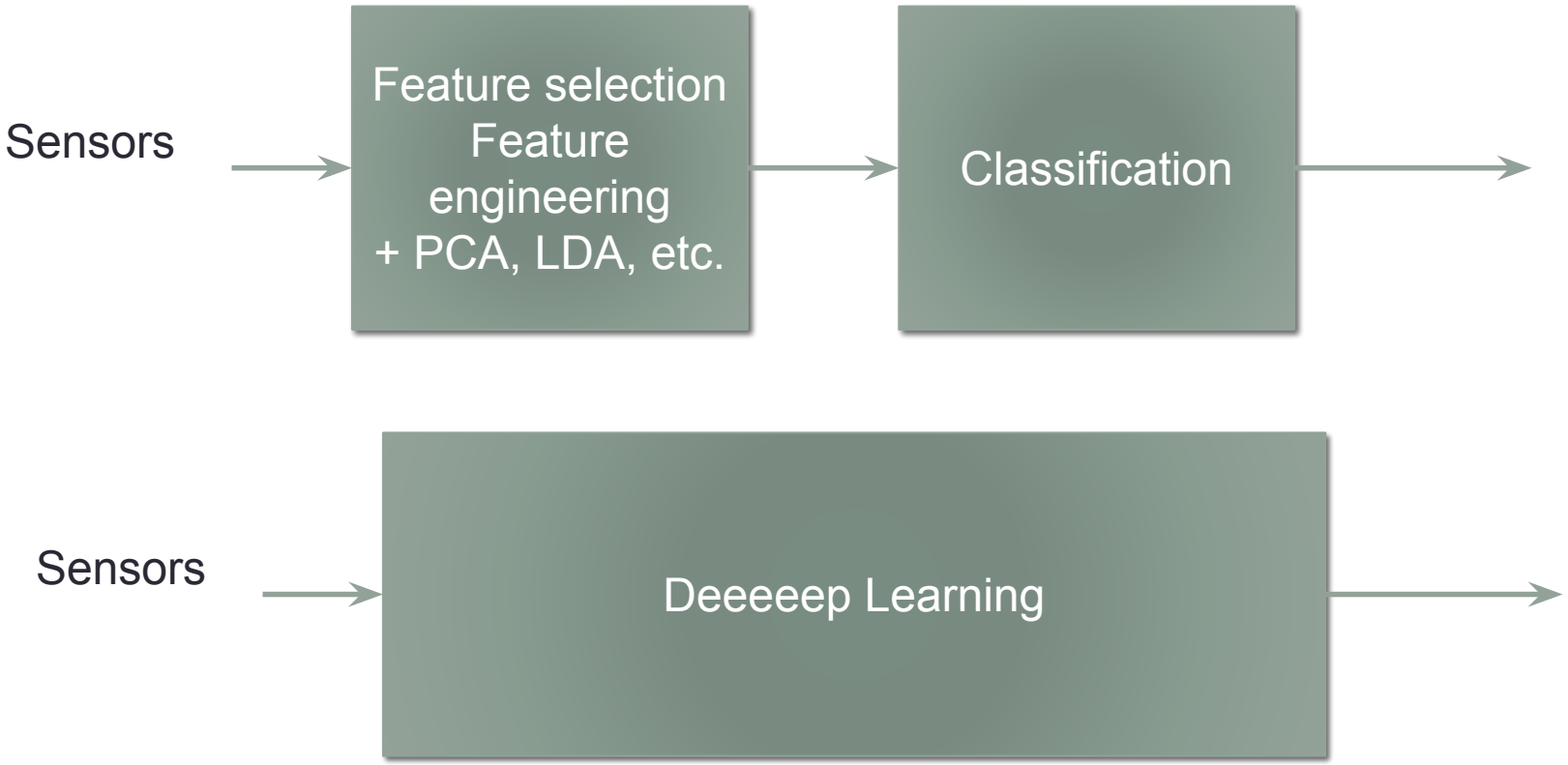
Comments: ICML 2018 Conference Proceedings
Subjects: **Machine Learning (stat.ML)**; Machine Learning (cs.LG)
Cite as: arXiv:1806.05393 **[stat.ML]**
(or **arXiv:1806.05393v1 [stat.ML]** for this version)

Submission history

From: Samuel Schoenholz [view email]
[v1] Thu, 14 Jun 2018 07:04:15 GMT (6734kb,D)

Which authors of this paper are endorsers? | Disable MathJax (What is MathJax?)

Traditional VS Deep learning

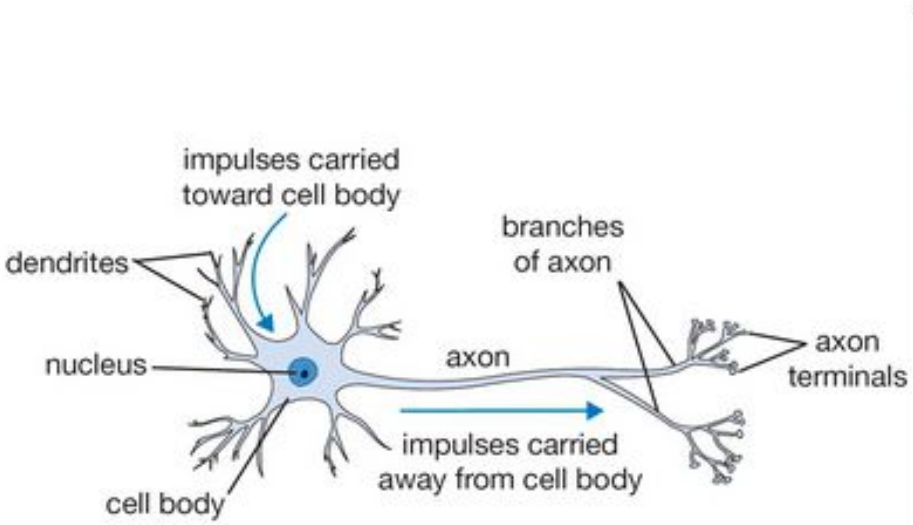


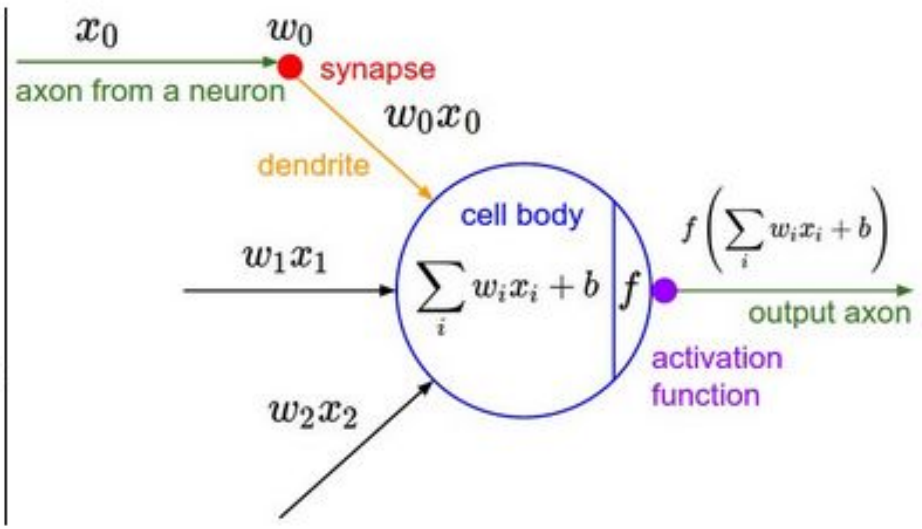
Neural networks

- Fully connected networks
  - Neuron
  - Non-linearity
  - Softmax layer
- DNN training
  - Loss function and regularization
  - SGD and backprop
  - Learning rate
  - Overfitting – dropout
- CNN
- RNN, LSTM, GRU

Fully connected networks

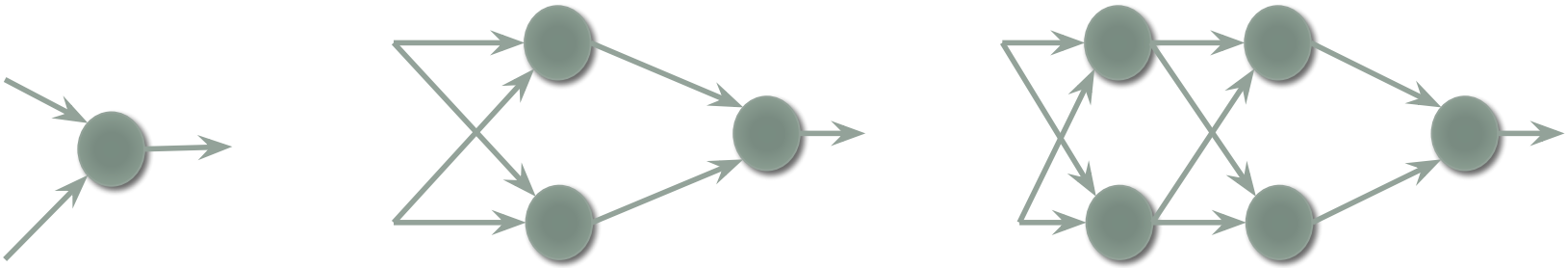
- Many names: feed forward networks or deep neural networks or multilayer perceptron or artificial neural networks
- Composed of multiple neurons

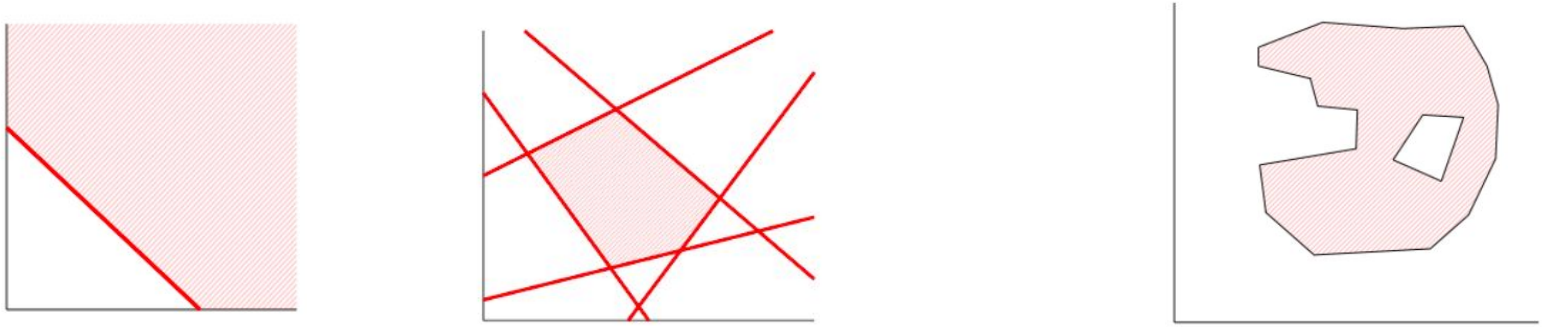




Combining neurons

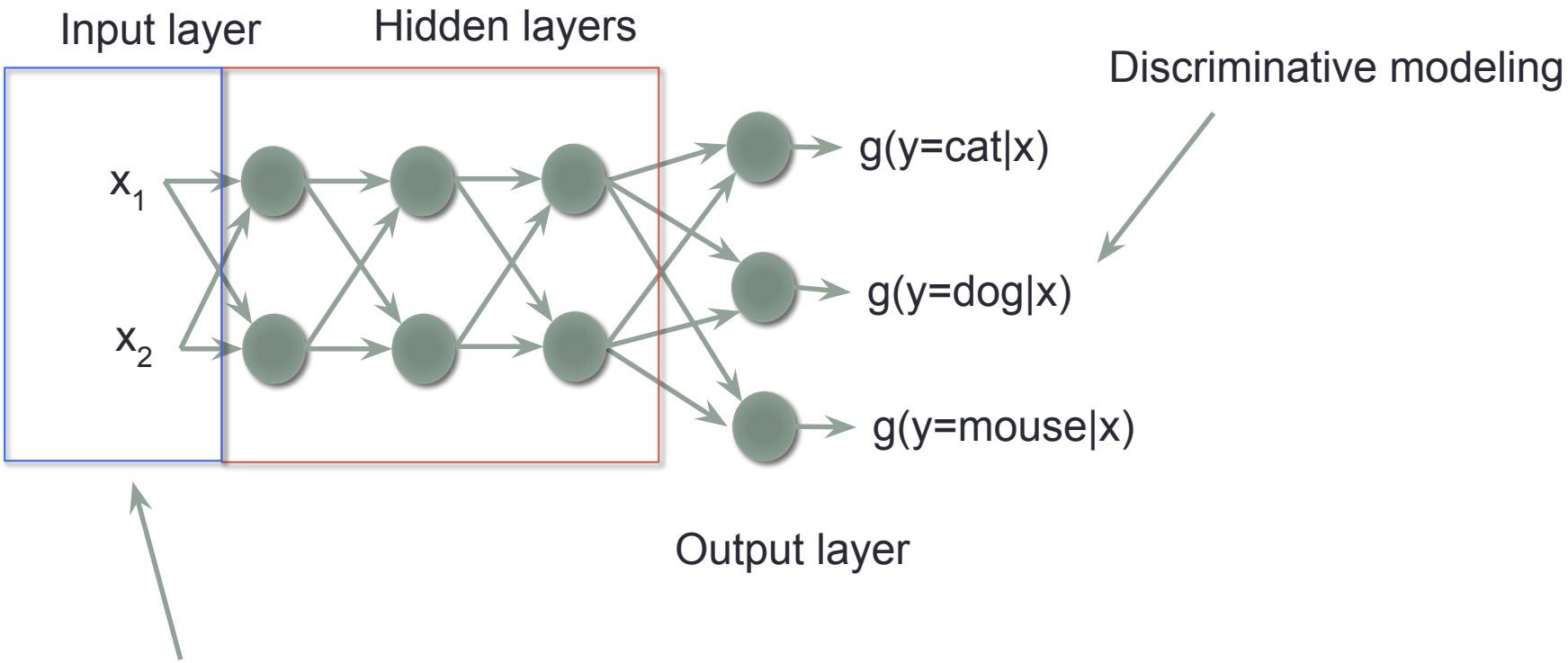
- Each neuron splits the feature space with a hyperplane
- Stacking neuron creates more complicated decision boundaries
- More powerful but prone to overfitting





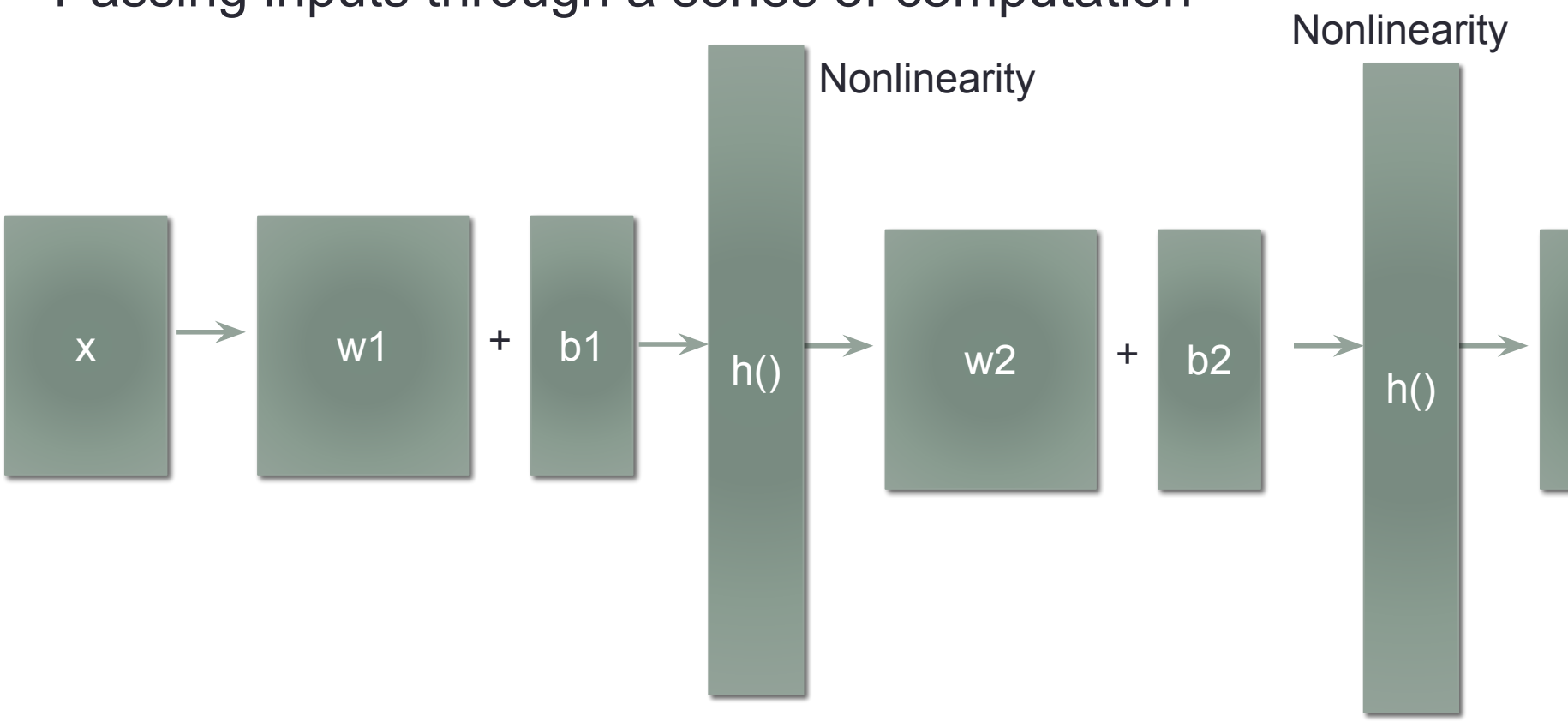
Terminology

Deep in Deep neural networks means many hidden layers



Computation graph

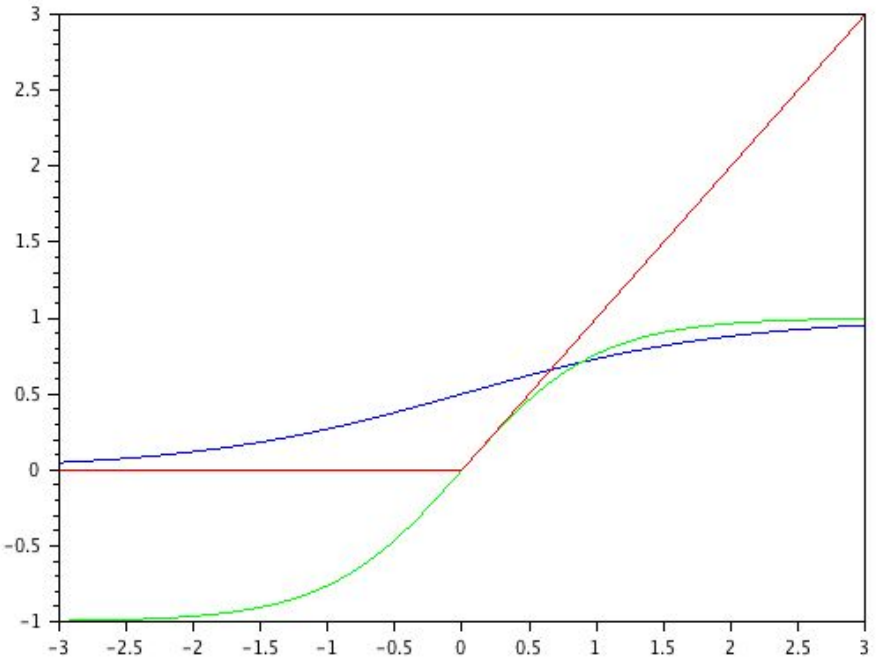
- Passing inputs through a series of computation



$h(W_2^T\, h(W_1^T X + \mathbf{b_1}) + \mathbf{b_2})$

Non-linearity

- The Non-linearity is important in order to stack neurons
- Sigmoid or logistic function
- tanh
- Rectified Linear Unit (ReLU)
- Swish (new!)
- Most popular is ReLU and its variants (Fast to train, and more stable)



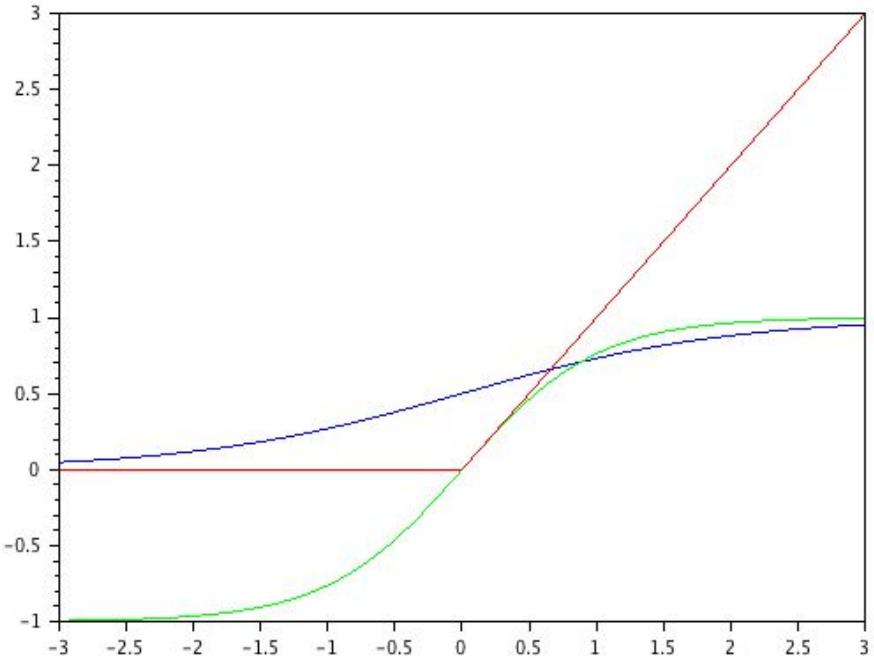
Non-linearity

- Sigmoid $\frac{1}{1+e^{-x}}$
- tanh

$$tanh(x)$$

- Rectified Linear Unit (ReLU)

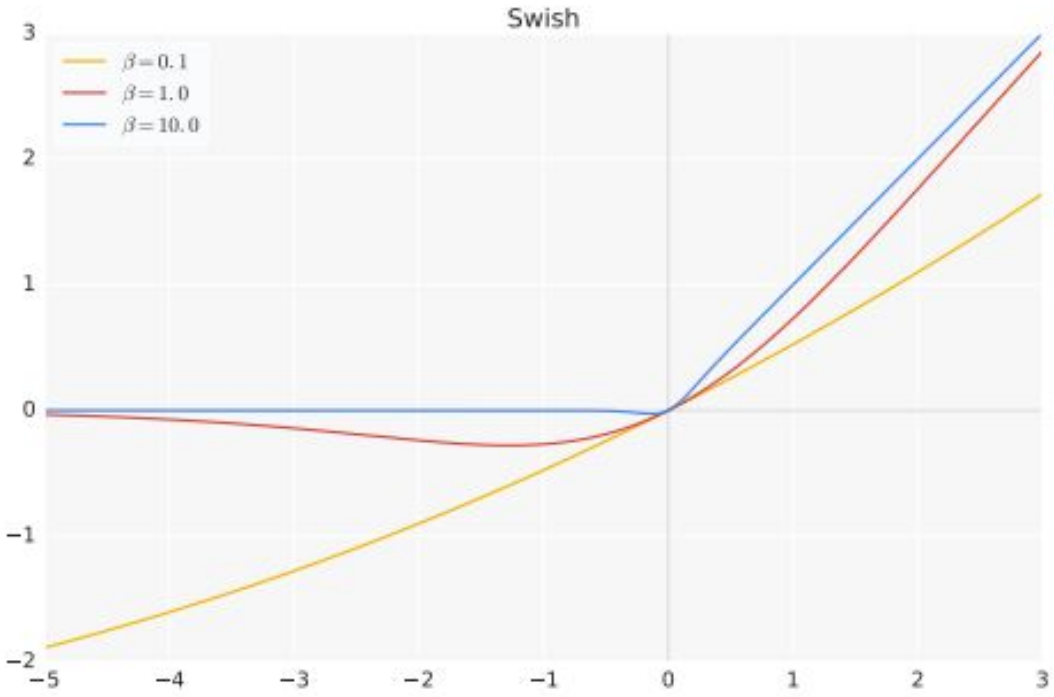
$$max(0, x)$$



Swish

Found through reinforcement learning to be the best general non-linearity

$$x \cdot sig(\beta x)$$



sig refers to a sigmoid function
Beta is a learnable parameter or can be set to 1 for slightly worse performance

Beta -> inf, then Swish -> ReLu

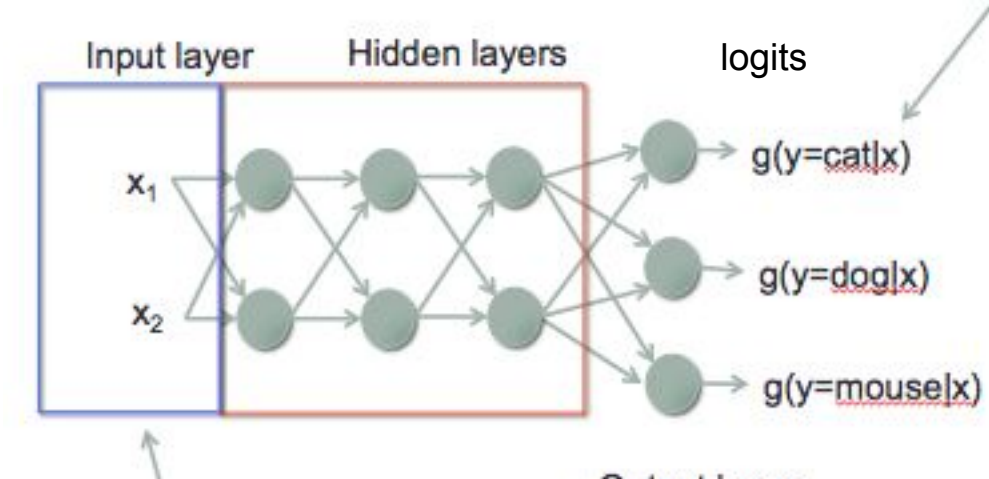
Searching for Activation Functions - arXiv

Proven theoretically to be optimal

Expectation propagation: a probabilistic view of Deep Feed Forward Networks

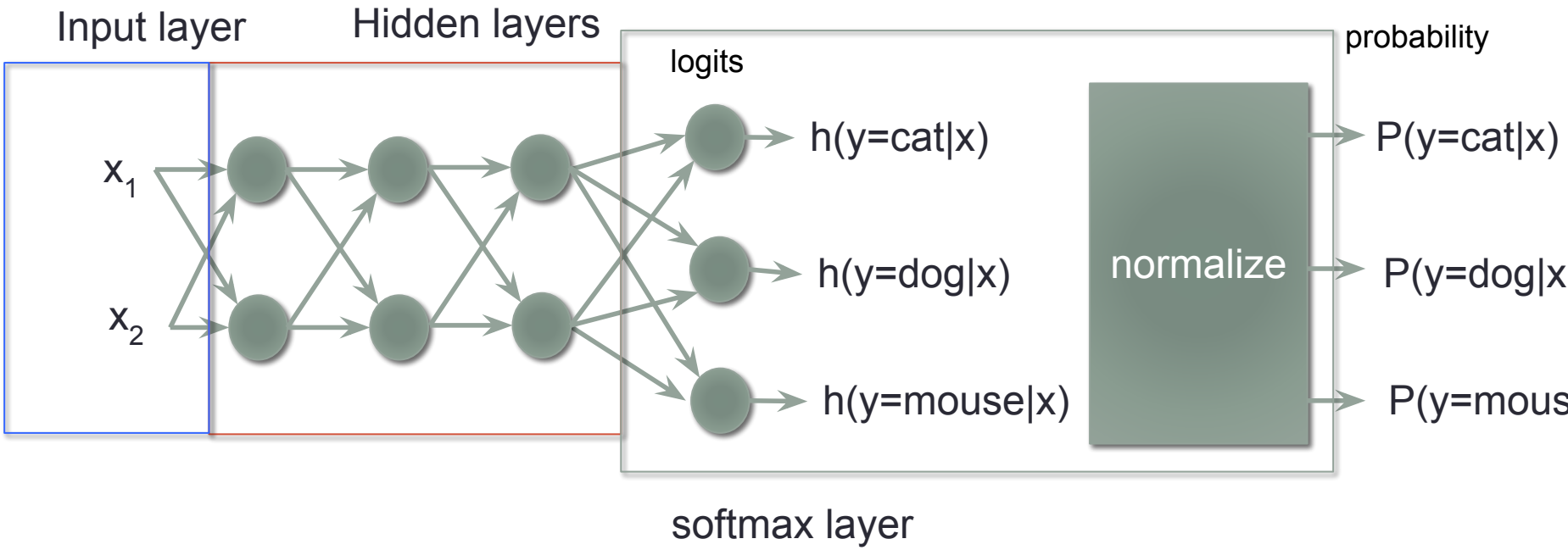
Output layer – Softmax layer

- We usually wants the output to mimic a probability function (0<=P<=1,sums to 1)
- Current setup has no such constraint
- The current output should have highest value for the correct class.
  - Value can be positive or negative number
- Takes the exponent
- Add a normalization



Softmax layer

$$P(y=j|x) = \frac{e^{h(y=j|x)}}{\sum_y e^{h(y|x)}}$$



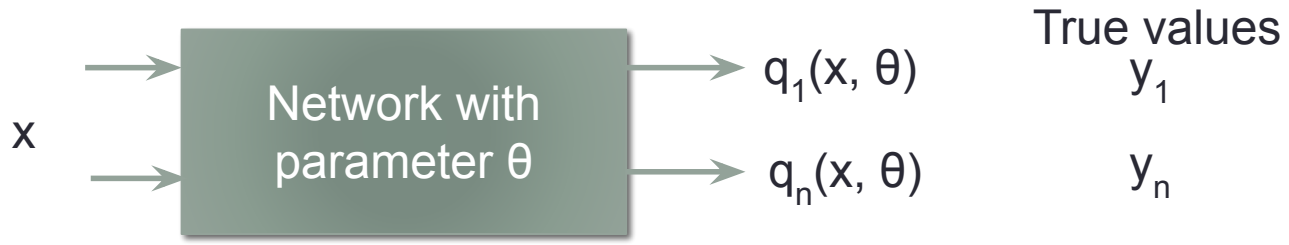
Neural networks

- Fully connected networks
  - Neuron
  - Non-linearity
  - Softmax layer
- DNN training
  - Loss function and regularization
  - SGD and backprop
  - Learning rate
  - Overfitting – dropout
- CNN
- RNN, LSTM, GRU

Objective function (Loss function)

- Can be any function that summarizes the performance into a single number
- Cross entropy

- Sum of squared errors



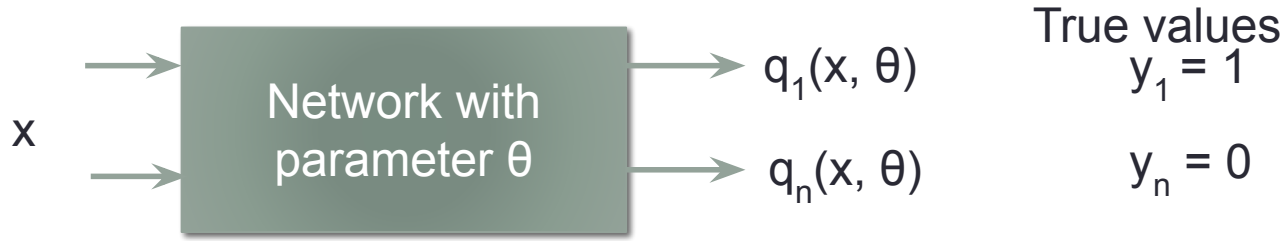
Cross entropy loss

- Used for softmax outputs (probabilities), or classification tasks

$$L = -\Sigma_n y_n log q_n(x, \theta)$$

- Where y_n is 1 if data x comes from class n
  0 otherwise

- L only has the term from the correct class
- L is non negative with highest value when the output matches the true values, a “loss” function

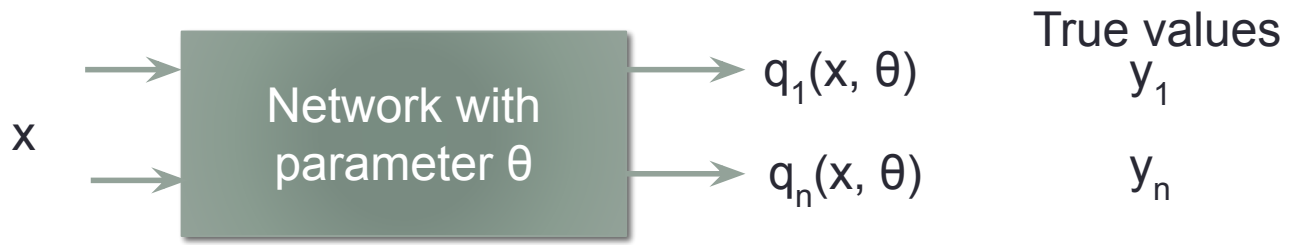


Sum of squared errors (MSE)

- Used for any real valued outputs such as regression

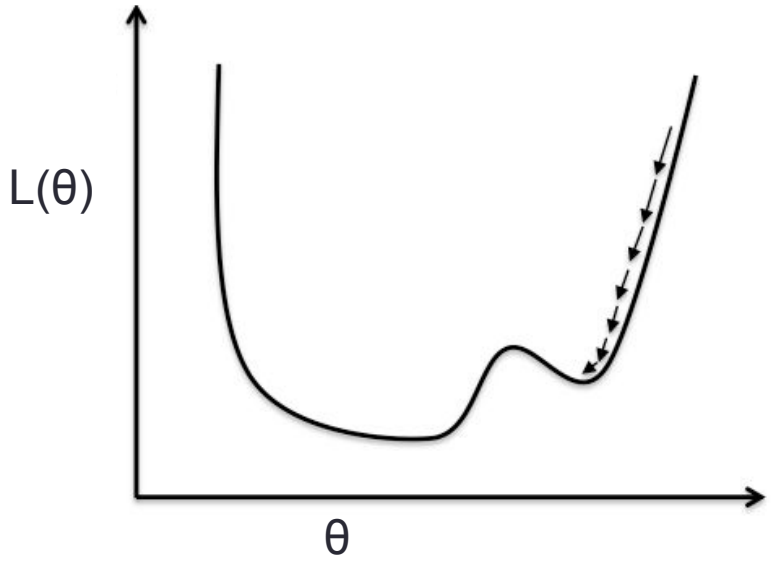
$$L = \frac{1}{2}\Sigma_n(y_n - q_n(x,\theta))^2$$

- Non negative, the better the lower the loss



Minimization using gradient descent

- We want to minimize L with respect to θ (weights and biases)
  - Differentiate with respect to θ
  - Gradients passes through the network by Back Propagation



Differentiating a neural network model

- We want to minimize loss by gradient descent
- A model is very complex and have many layers! How do we differentiate this!!?

Back propagation

- Forward pass
  - Pass the value of the input until the end of the network
- Backward pass
  - Compute the gradient starting from the end and passing down gradients using chain rule

Examples to read

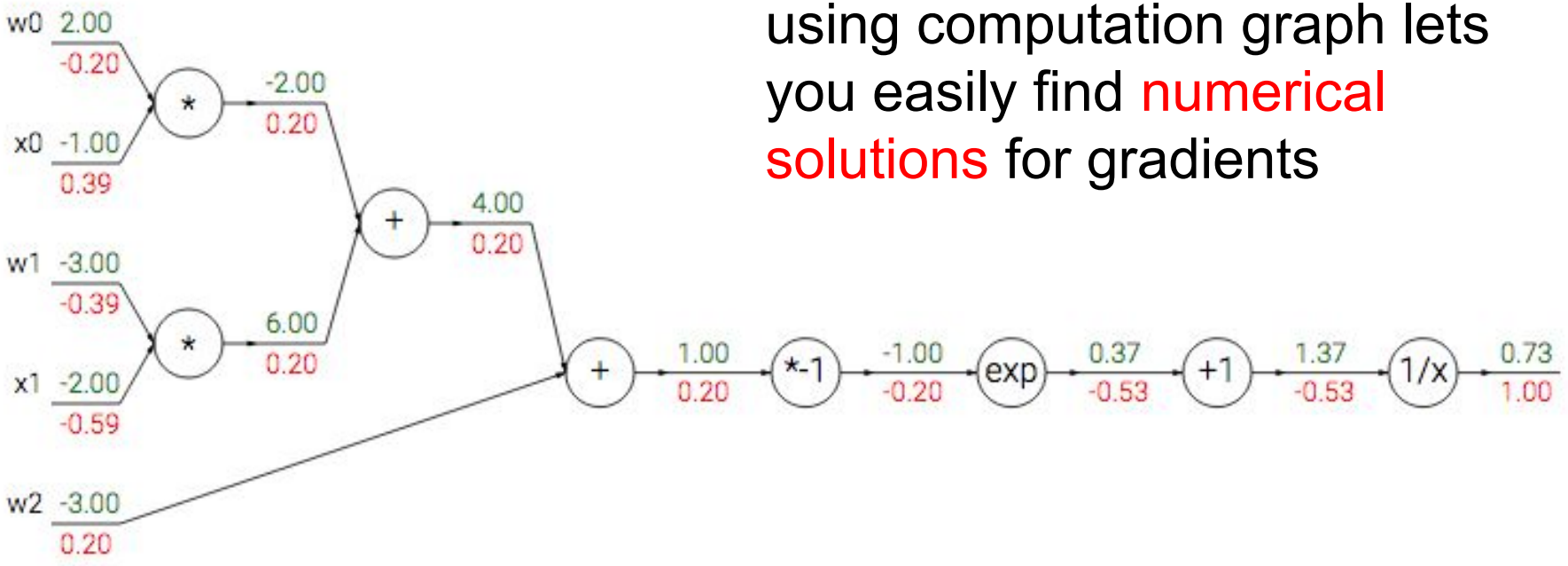
https://alonalj.github.io/2016/12/10/What-is-Backpropagation/

https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Numerical gradient flow

- Let's find the gradient of $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

Computation graph



Doing backprop (chain rule) by using computation graph lets you easily find numerical solutions for gradients

http://cs231n.github.io/optimization-2/

Gradient and non-linearities

We can now talk about how good a non-linearity is by looking at the gradients.

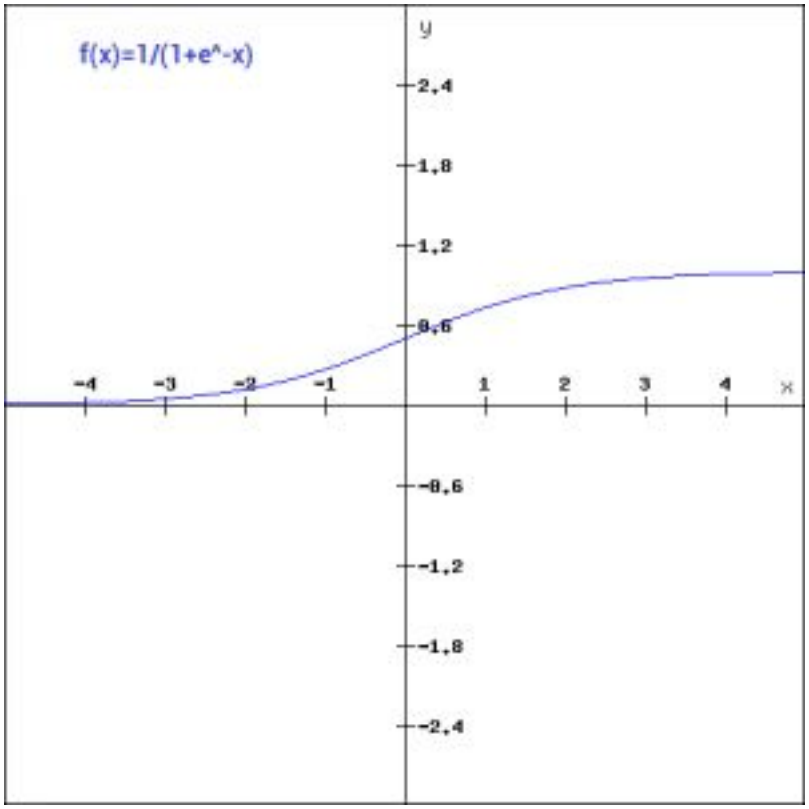
We want

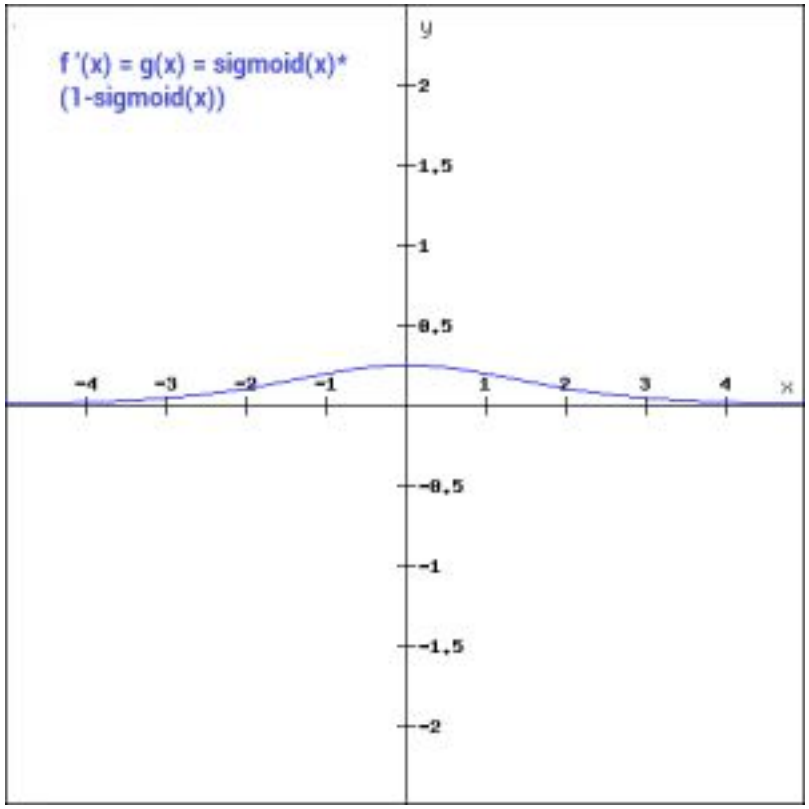
- Something that is differentiable numerically
- Cheap to compute
- Big gradients at every point

Notes on non-linearity

- Sigmoid

  Models get stuck if fall go far away from 0. Output always positive



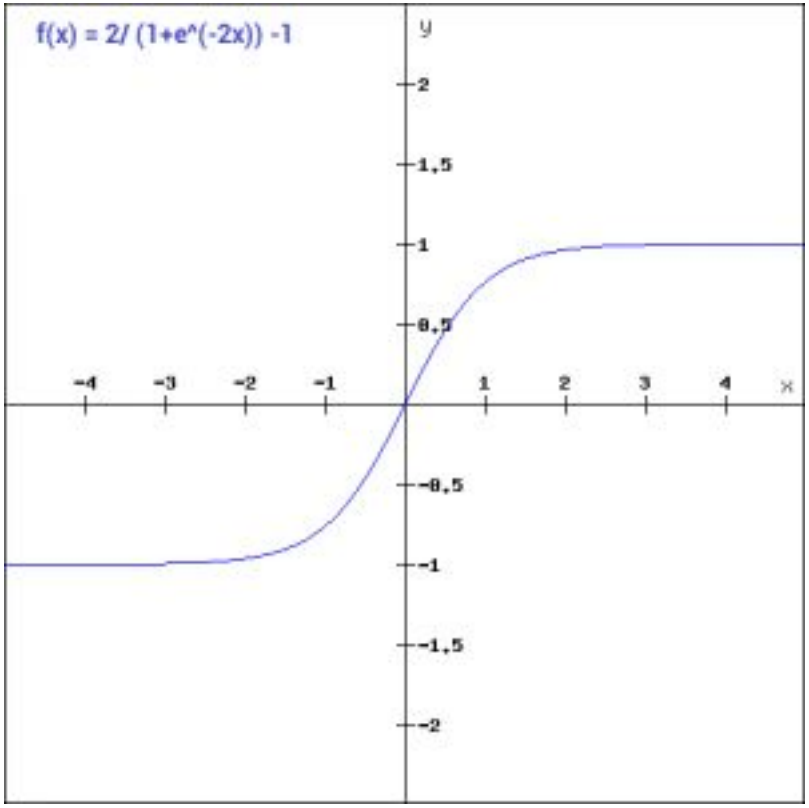


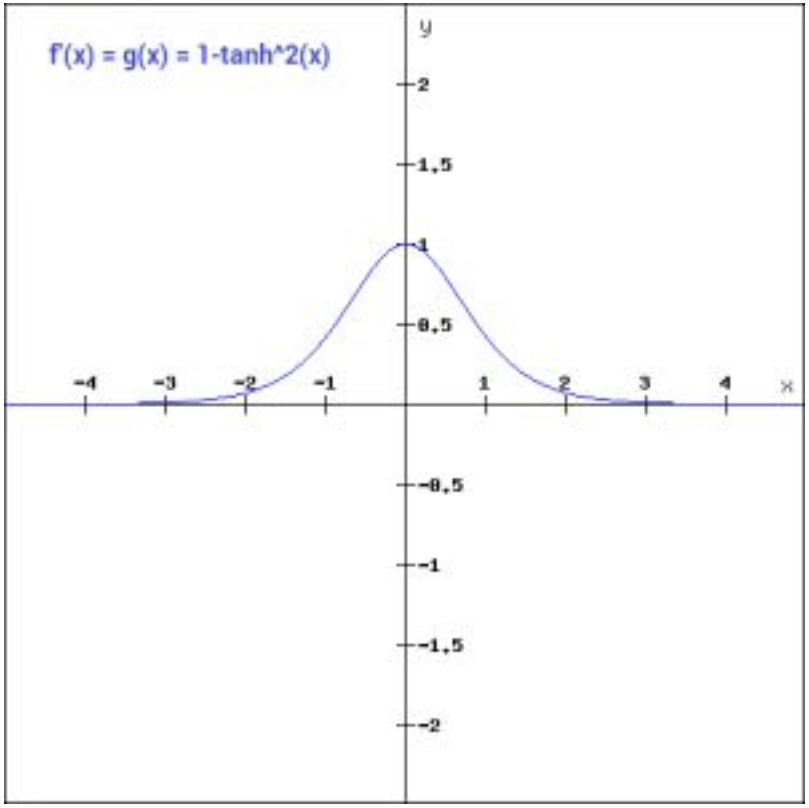
https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions-when-to-use

Notes on non-linearity

- Tanh

Output can be +-. Models get stuck if far away from 0



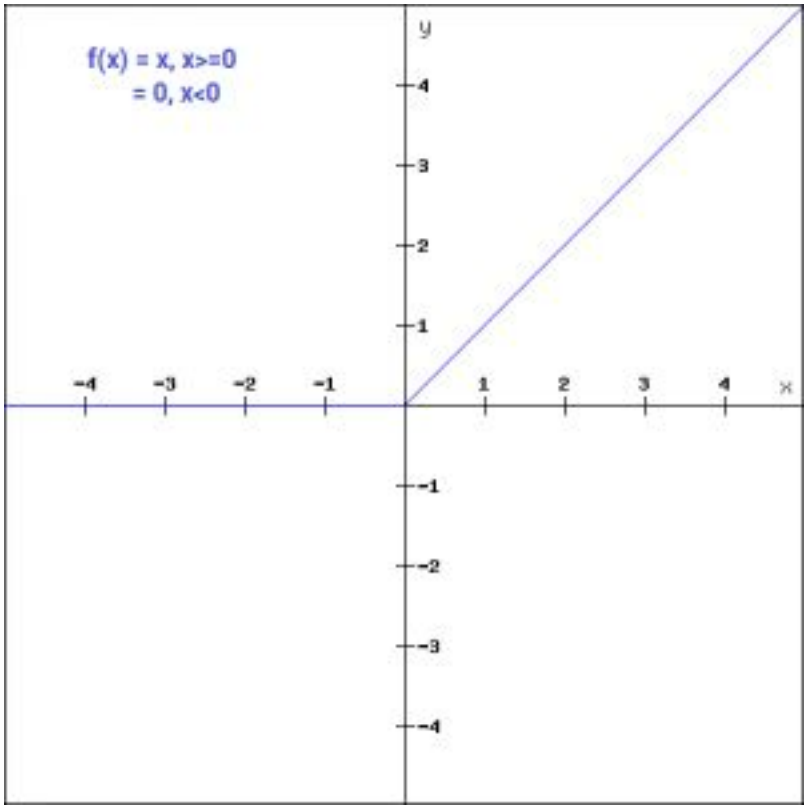


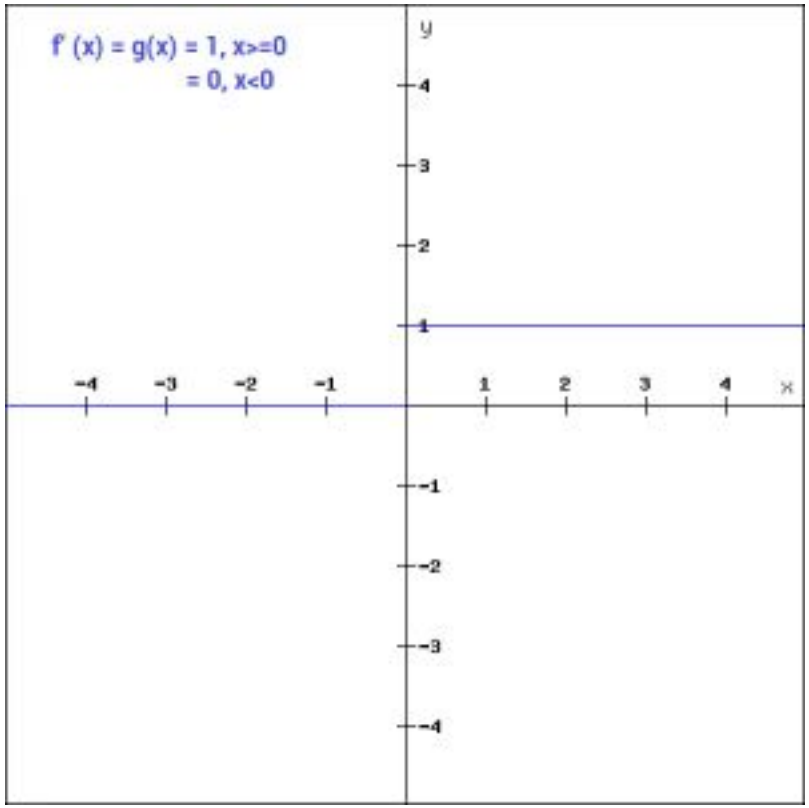
https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions-when-to-use

Notes on non-linearity

- ReLU

  High gradient in positive. Fast compute. Gradient doesn't move in negative



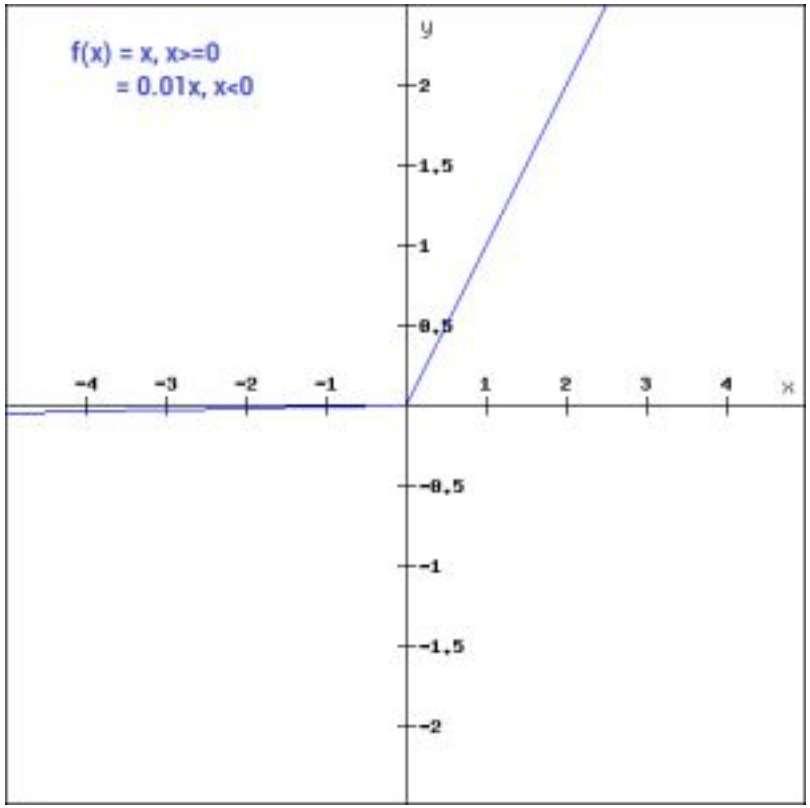


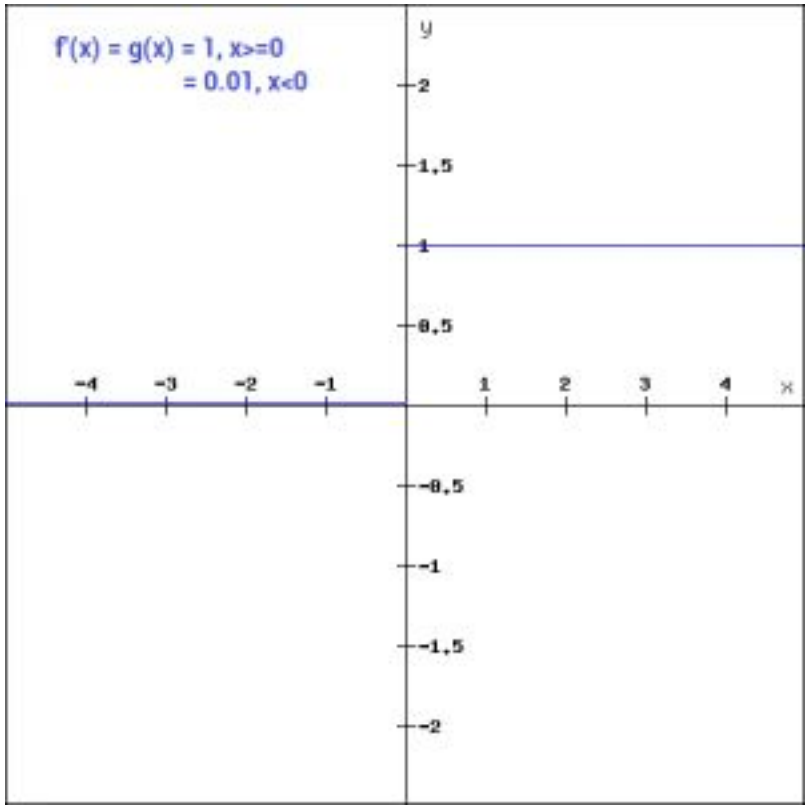
https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions-when-to-use

Notes on non-linearity

- Leaky ReLU

  Negative part now have some gradient. Small improvements depending on tasks





https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions-when-to-use

Initialization

- The starting point of your descent
- Important due to local minimas
- Not as important with large networks AND big data
- Now usually initialized randomly
  - One strategy (Xavier initialization)

$$W \sim \text{Uniform}(0, \frac{1}{\sqrt{\text{FanIn} + \text{FanOut}}})$$

  - For ReLUs

```
w = np.random.randn(n) * sqrt(2.0/n)
```

  - Or use a pre-trained network as initialization

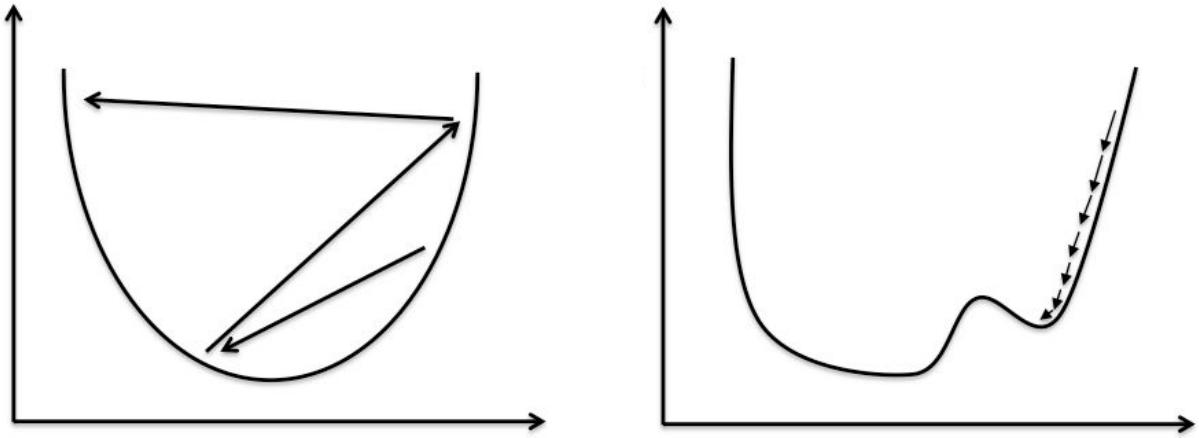
X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. 2010

Stochastic gradient descent (SGD)

- Consider you have one million training examples
  - Gradient descent computes the objective function of all samples, then decide direction of descent
    - Takes too long
  - SGD computes the objective function on subsets of samples
    - The subset should not be biased and properly randomized to ensure no correlation between samples
- The subset is called a mini-batch
- Size of the mini-batch determines the training speed and accuracy
  - Usually somewhere between 32-1024 samples per mini-batch
- Definition: 1 batch vs 1 epoch

Learning rate

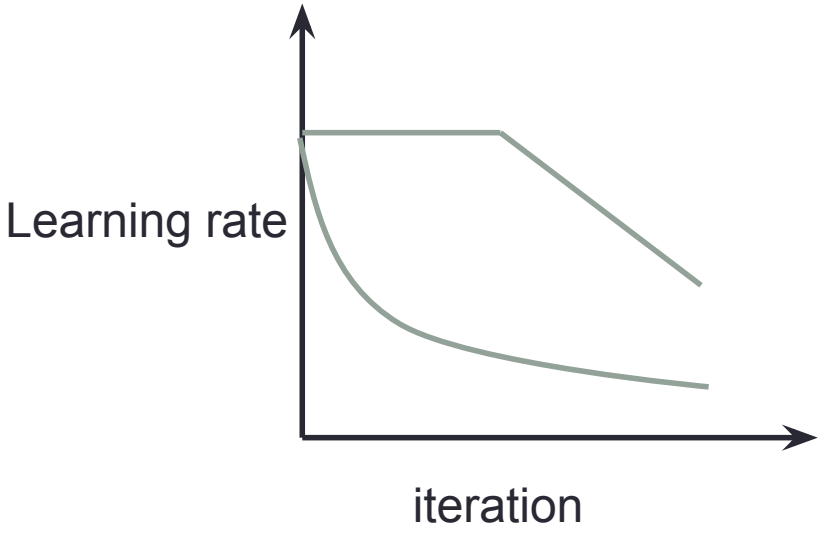
- How fast to go along the gradient direction is controlled by the learning rate
- Too large models diverge
- Too small the model get stuck in local minimas and takes too long to train



http://sebastianraschka.com/Articles/2015_singlelayer_neurons.html

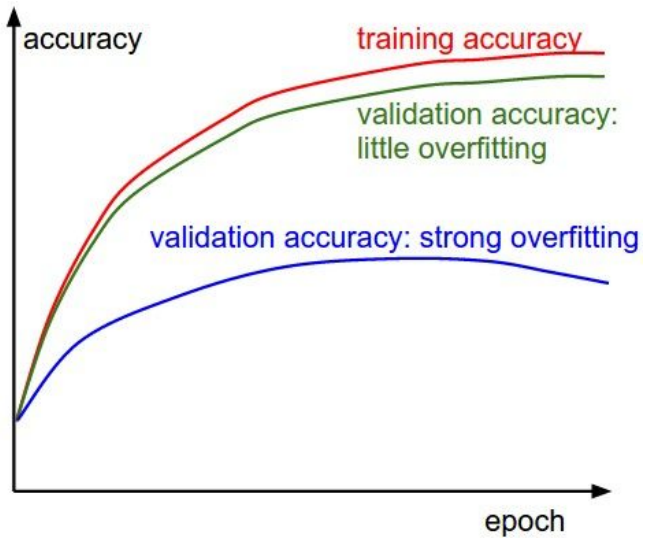
Learning rate scheduling

- Usually starts with a large learning rate then gets smaller later
- Depends on your task
- Automatic ways to adjust the learning rate : Adagrad, Adam, etc. (still need scheduling still)



Overfitting

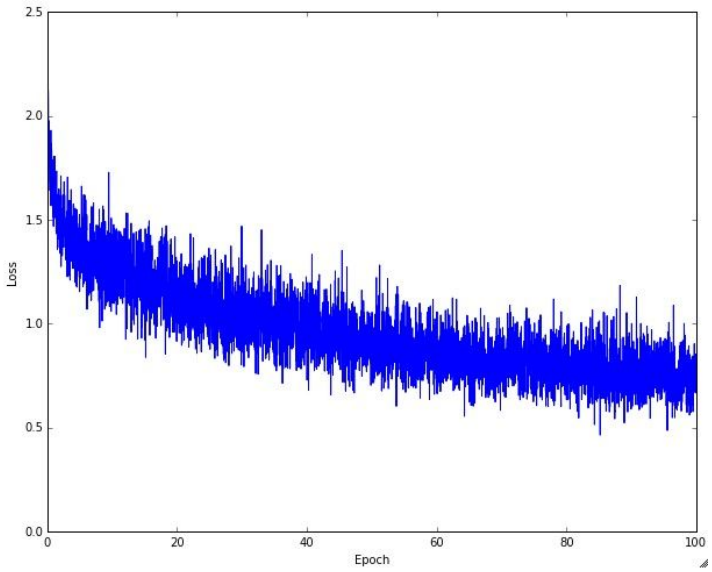
- You can keep doing back propagation forever!
- The training loss will always go down
- But it overfits
- Need to monitor performance on a held out set
- Stop or decrease learning rate when overfit happens



http://cs231n.github.io/neural-networks-3/

Monitoring performance

- Monitor performance on a dev/validation set
  - This is NOT the test set
- Can monitor many criterions
  - Loss function
  - Classification accuracy
- Sometimes these disagree
- Actual performance can be noisy, need to see the trend



http://cs231n.github.io/neural-networks-3/

Dropout

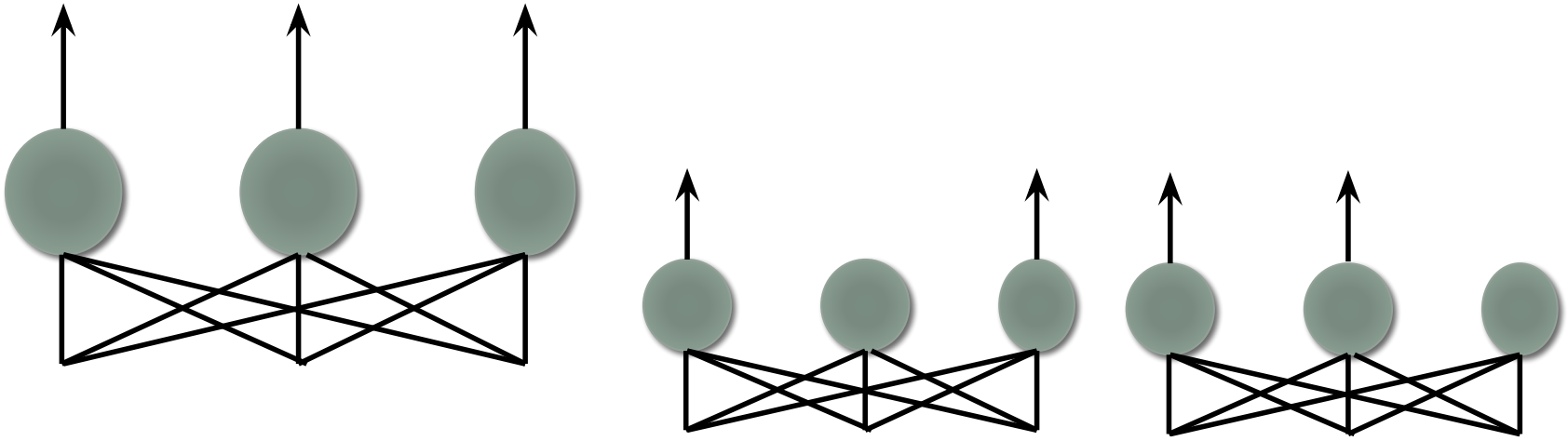
A implicit regularization technique for reducing overfitting

Randomly turn off different subset of neurons during training

Network no longer depend on any particular neuron

Force the model to have redundancy – robust to any corruption in input data

A form of performing model averaging (ensemble of experts)



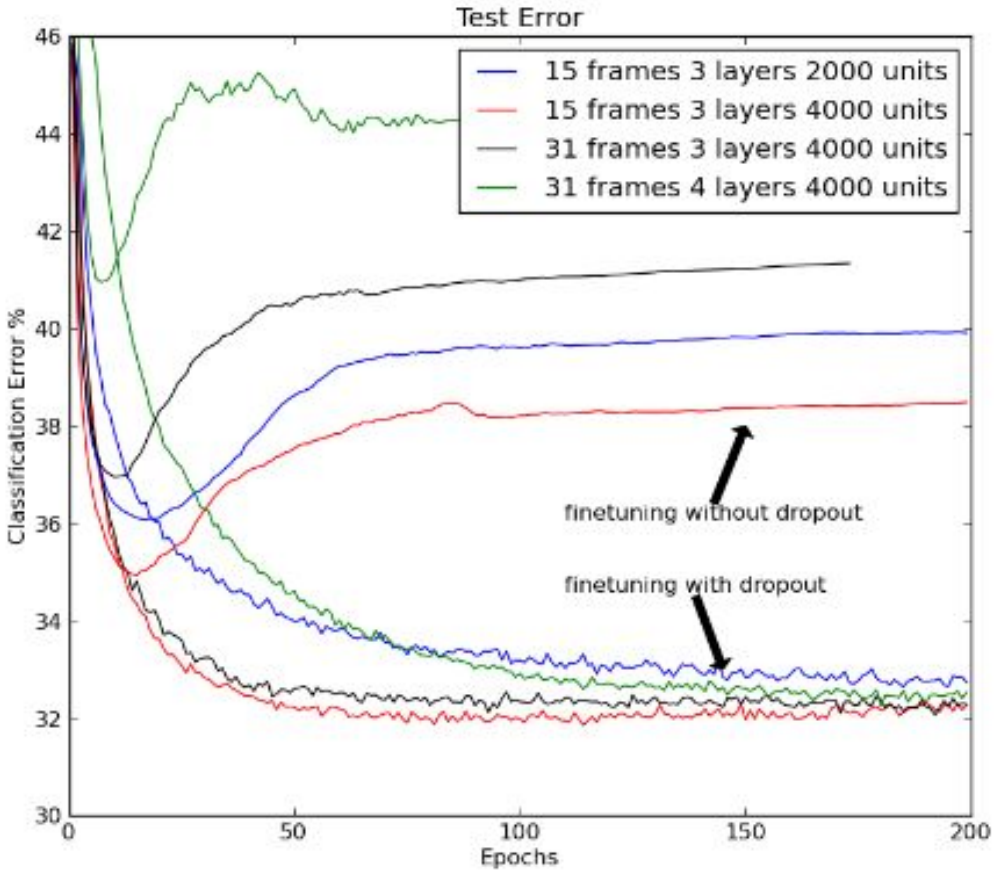
Model

Dropout rate of 0.33

Hinton, Geoffrey "Improving neural networks by preventing co-adaptation of feature detectors" 2012

Dropout on TIMIT

- A phoneme recognition task



Hinton, Geoffrey "Improving neural networks by preventing co-adaptation of feature detectors" 2012

Neural networks

- Fully connected networks
  - Neuron
  - Non-linearity
  - Softmax layer
- DNN training
  - Loss function and regularization
  - SGD and backprop
  - Learning rate
  - Overfitting – dropout
- CNN
- RNN, LSTM, GRU

Convolutional Neural Networks (CNNs)

- Consider an image of a cat. DNNs need different neurons to learn every possible location a cat can be

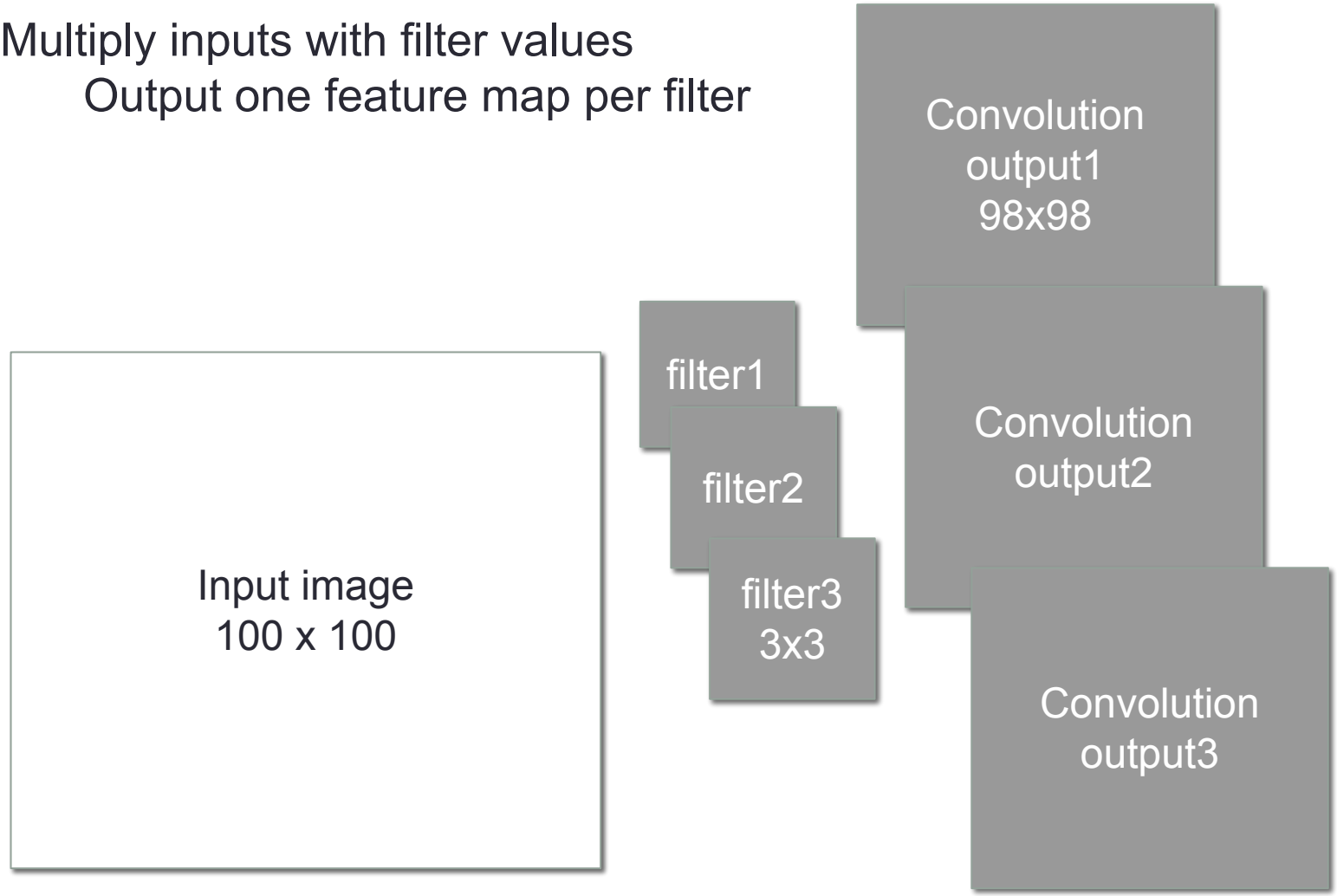
- Can we use the same parameters to learn that a cat exists regardless of location?

- 2 parts: convolutional layer and pooling layer

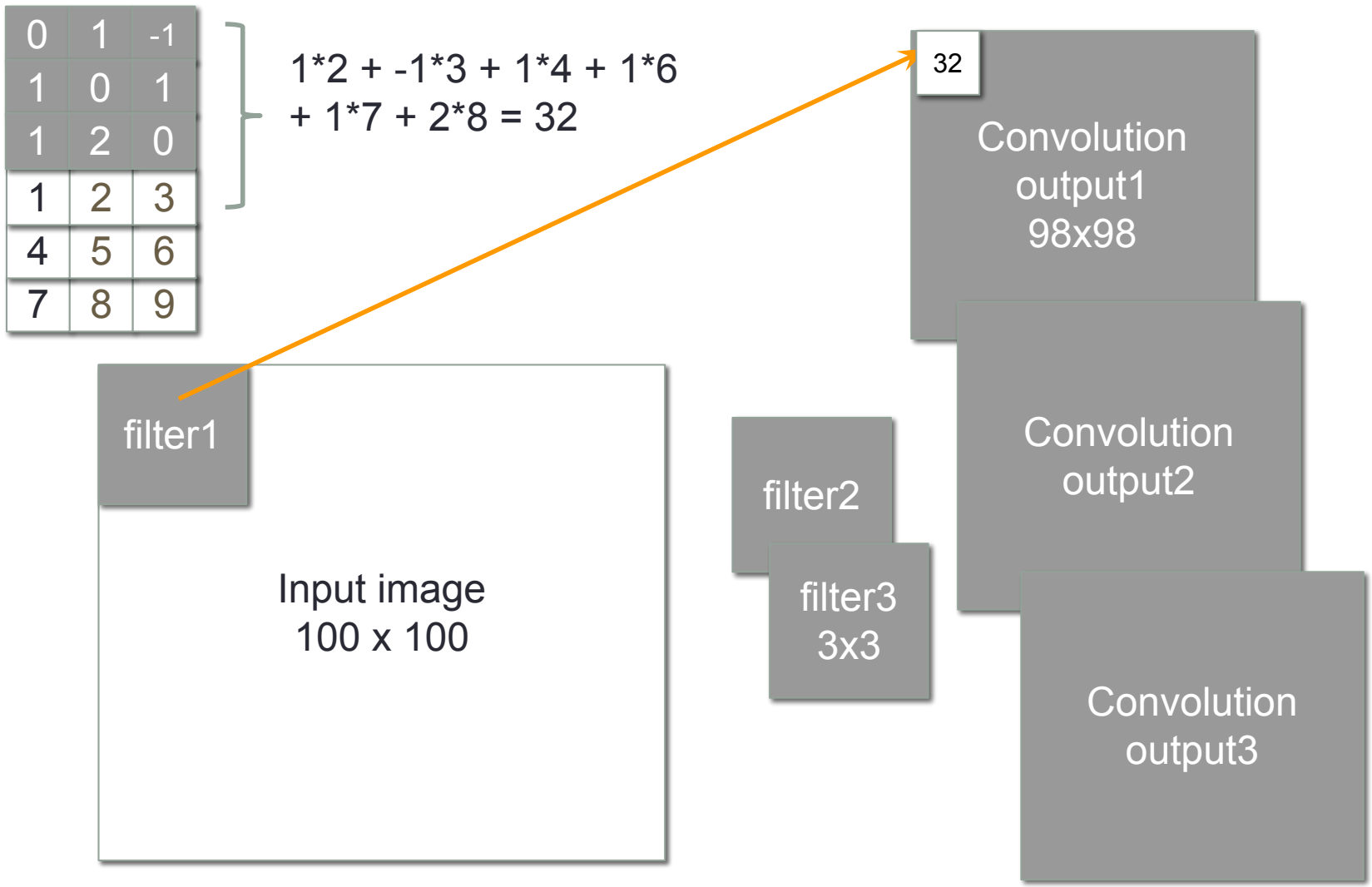
Convolutional filters

Multiply inputs with filter values

Output one feature map per filter

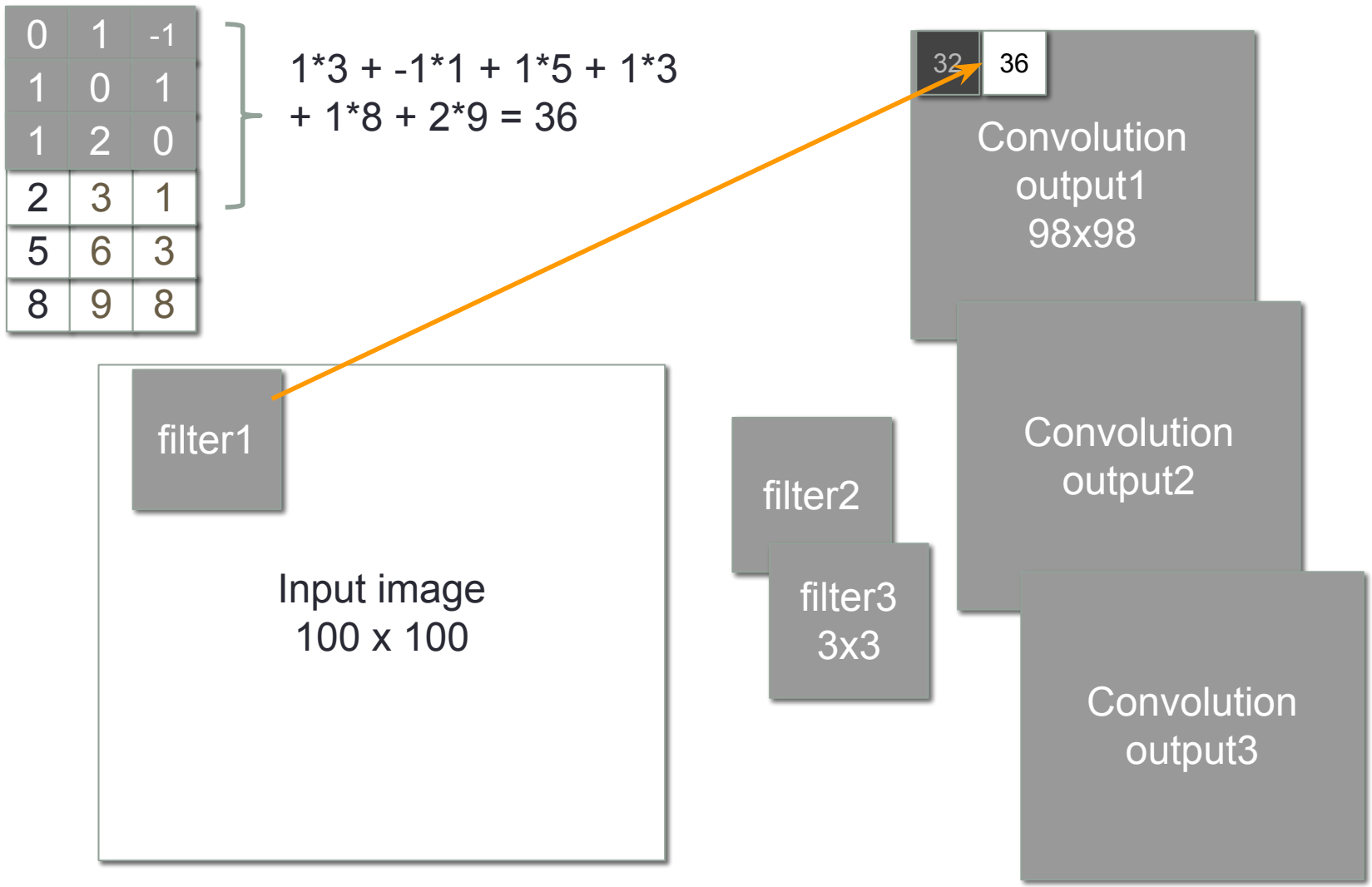


Convolutional filters

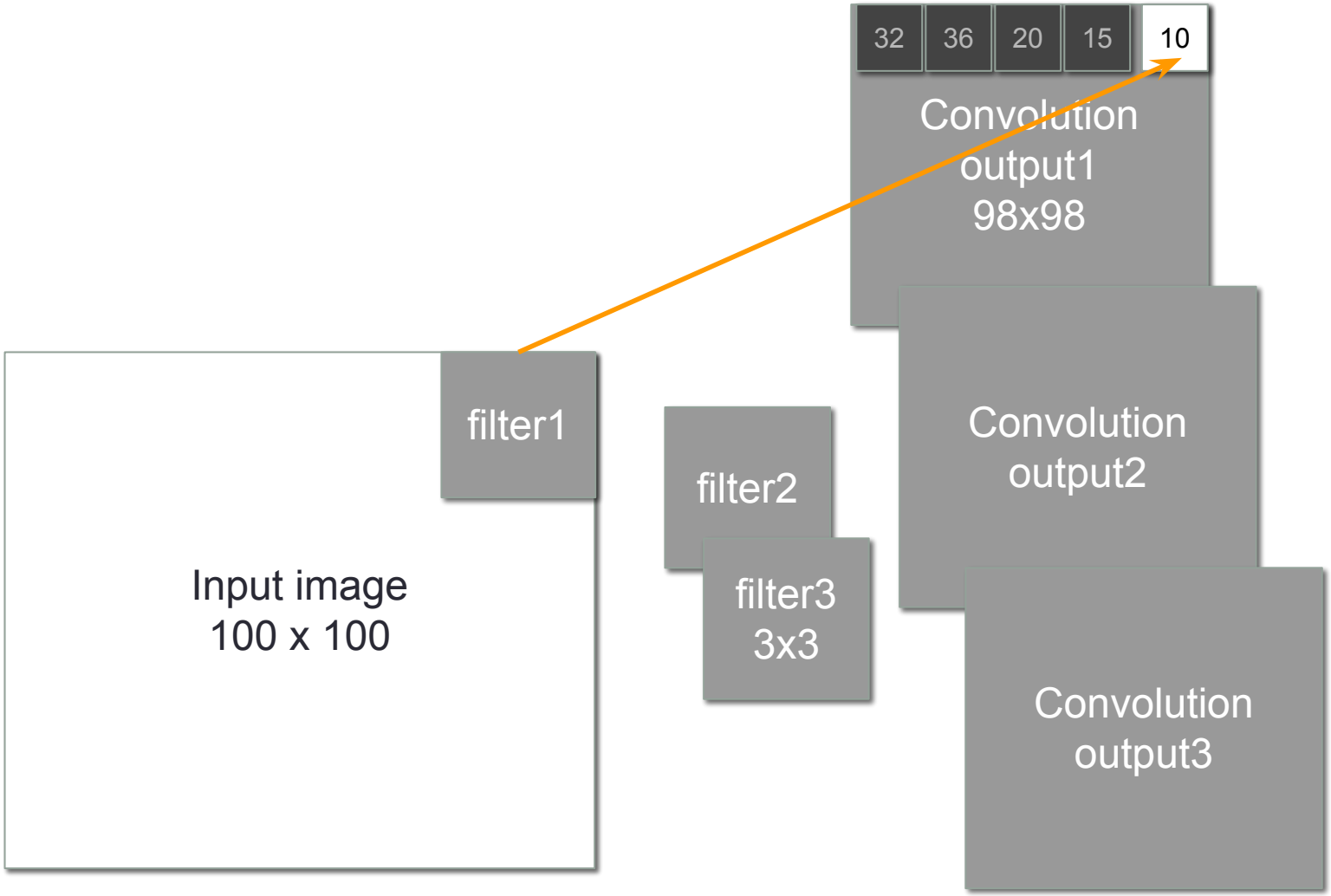


Convolutional filters

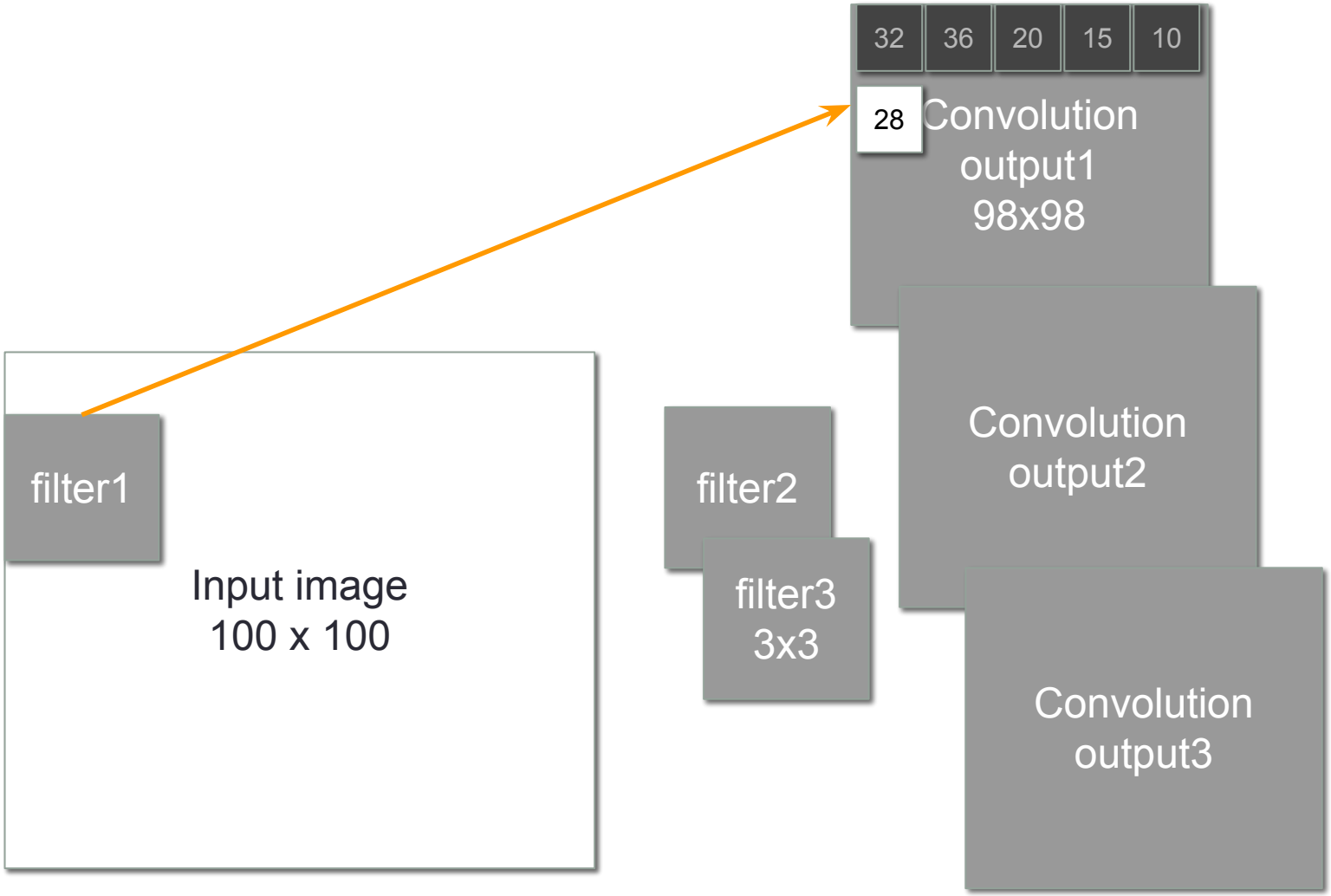
Stride of 1



Convolutional filters

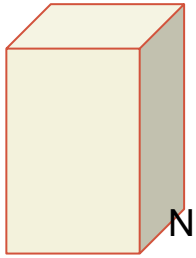


Convolutional filters



Convolutional filters

N filters means N feature maps
You get a 3 dimensional output



32	36	20	15	10
28	72	0	12	50
32	36	18	9	2
17	6	2	17	1

16

filter2

filter1

filter3
3x3

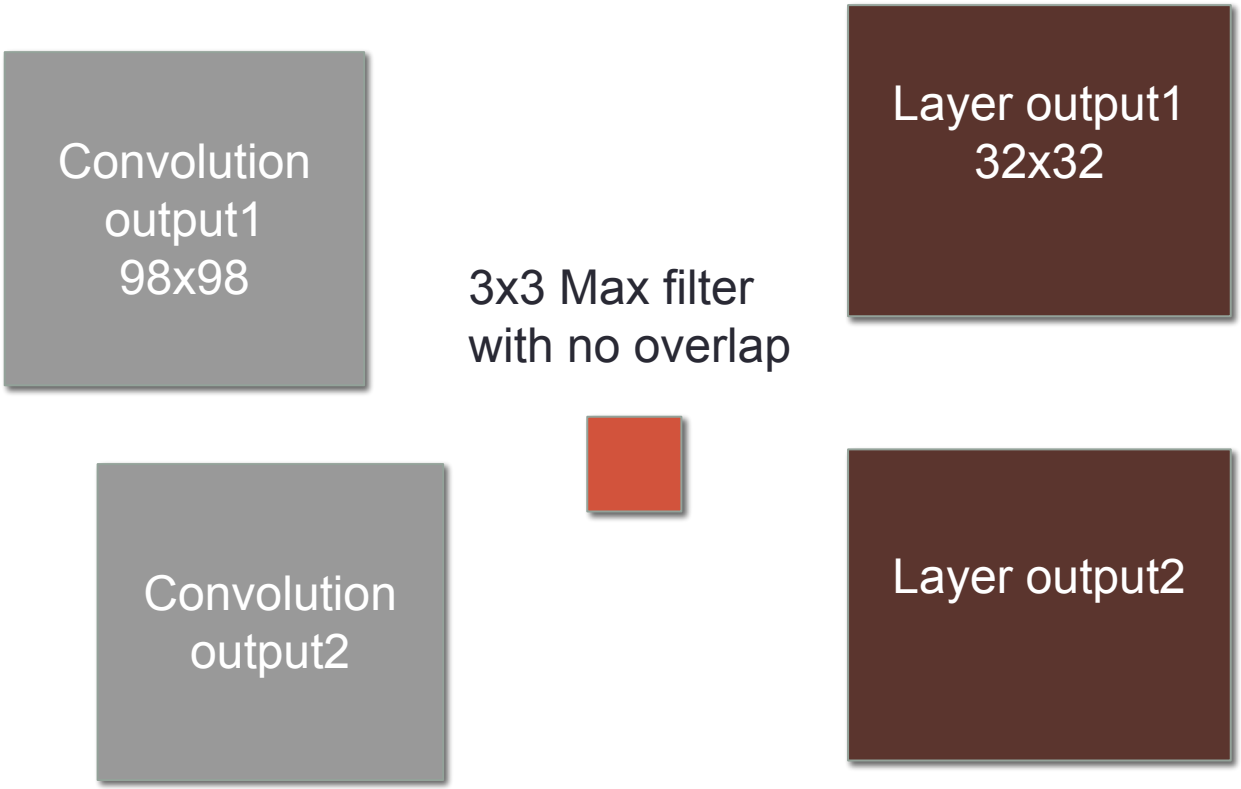
Input image
100 x 100

Convolution
output2

Convolution
output3

Pooling/subsampling

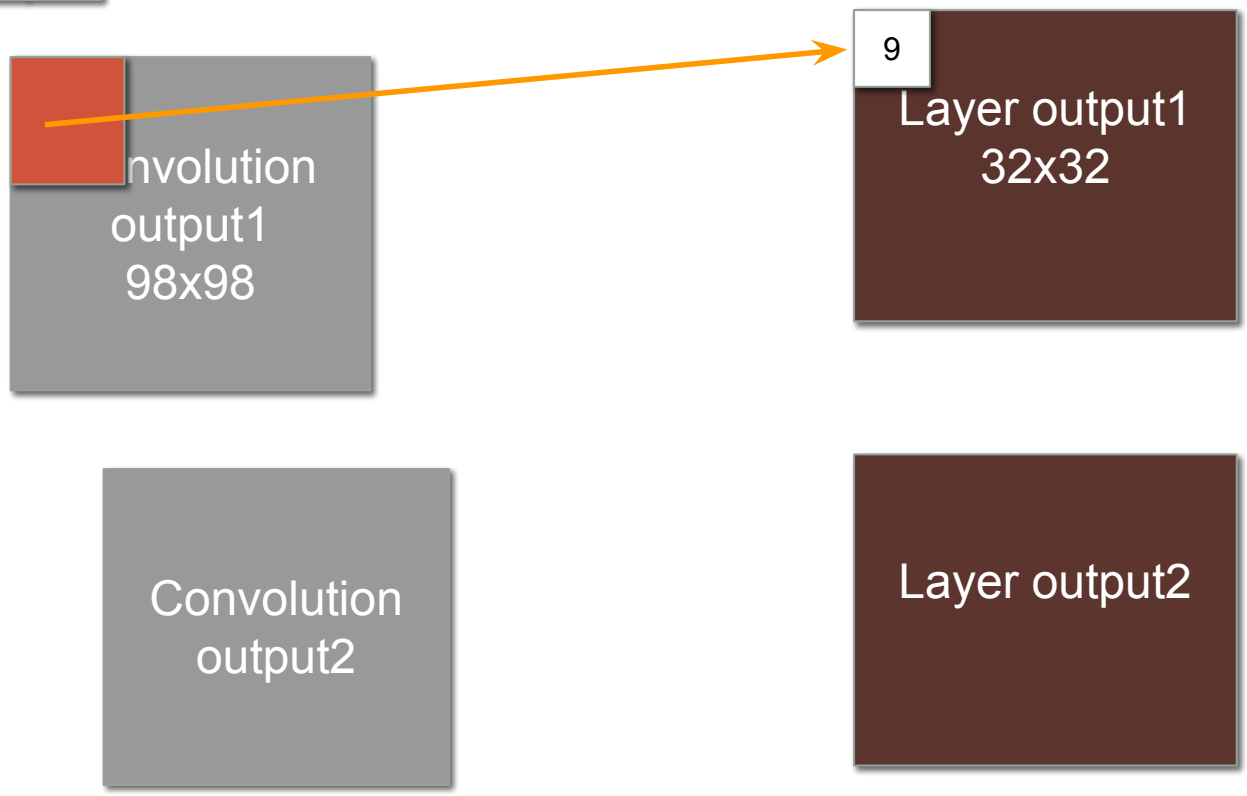
Reduce dimension of the feature maps



Pooling/subsampling

1	2	3
4	5	6
7	8	9

Max = 9

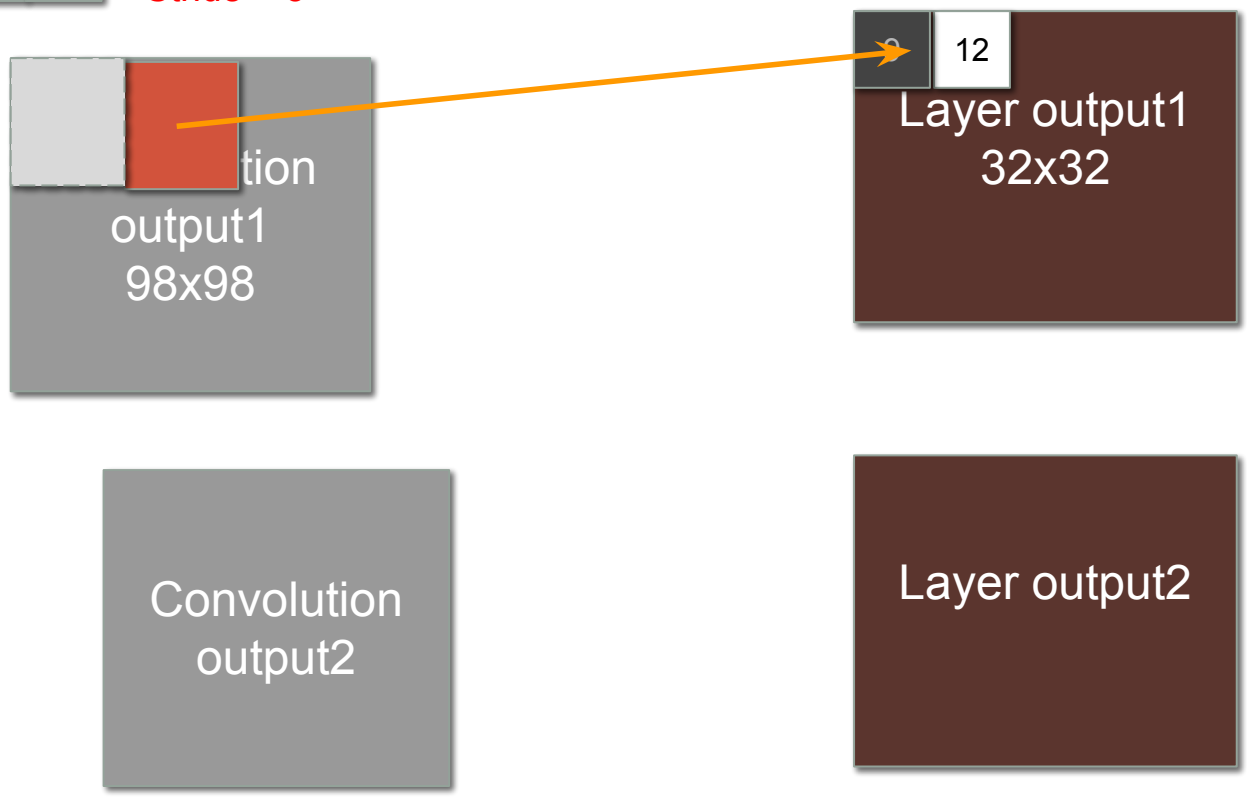


Pooling/subsampling

5	2	1
5	7	1
9	5	12

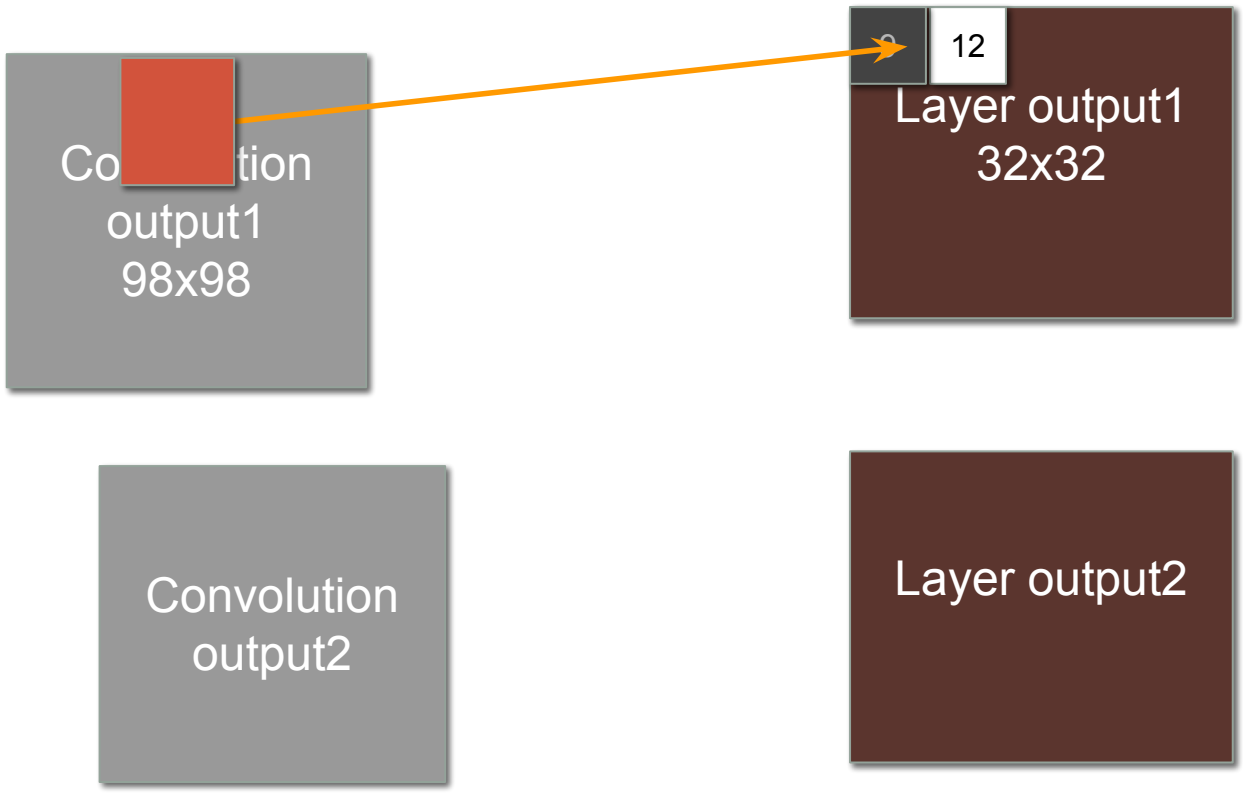
Max = 12

Stride = 3



Pooling/subsampling

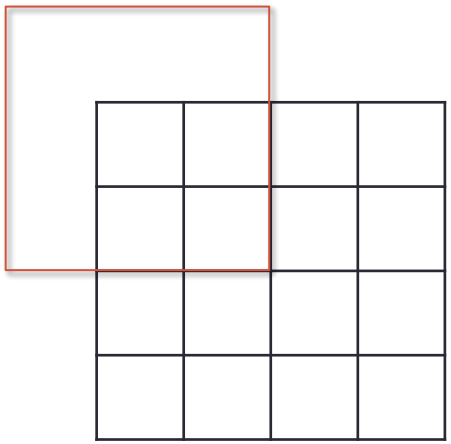
Can use other functions besides max
Example, average



Convolution puzzle

5 filters 3x3 filter pad, stride 1, pad 1

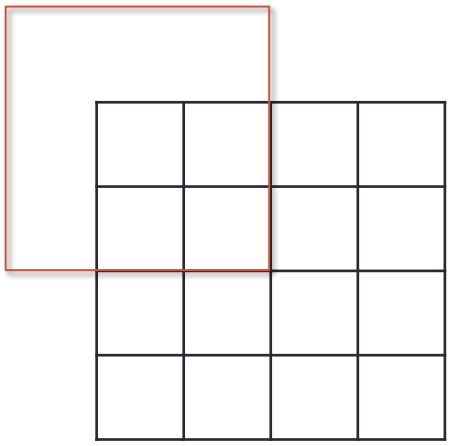
What is the output size?

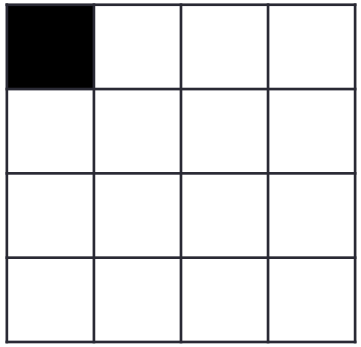


Convolution puzzle

5 filters 3x3 filter pad, stride 1, pad 1

What is the output size?

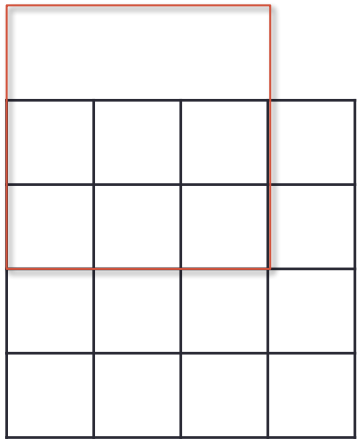


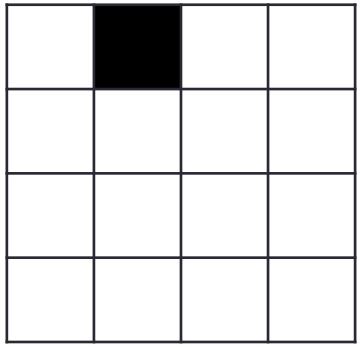


Convolution puzzle

5 filters 3x3 filter pad, stride 1, pad 1

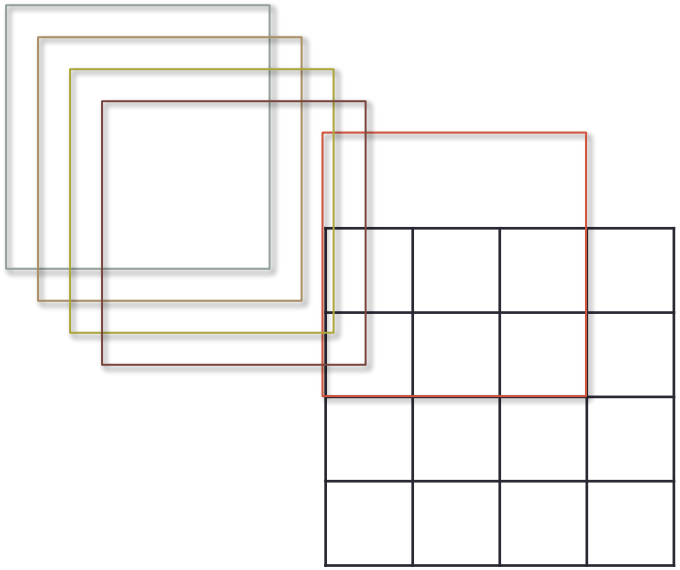
What is the output size?

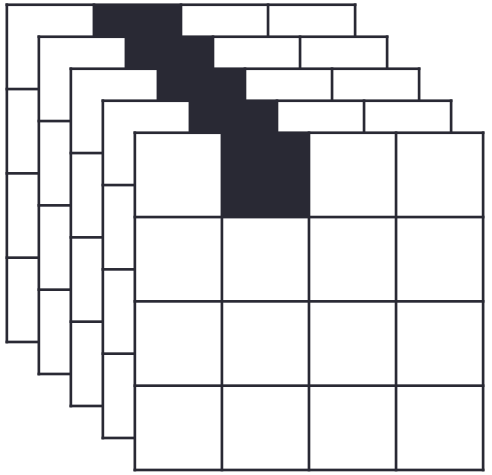




Convolution puzzle

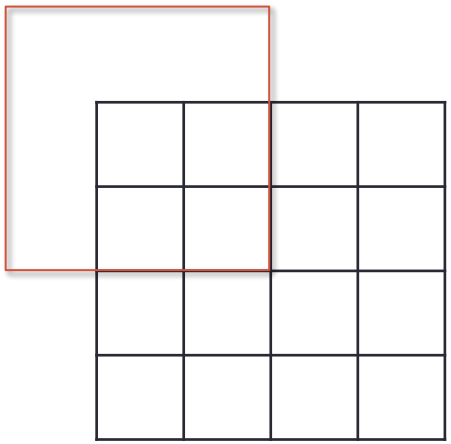
5 filters 3x3 filter pad, stride 1, pad 1





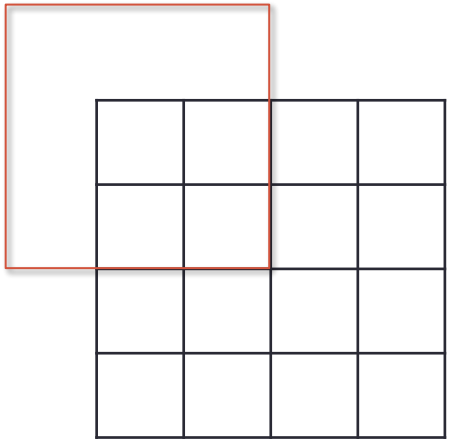
Convolution puzzle

3x3 filter pad, stride 2, pad 1



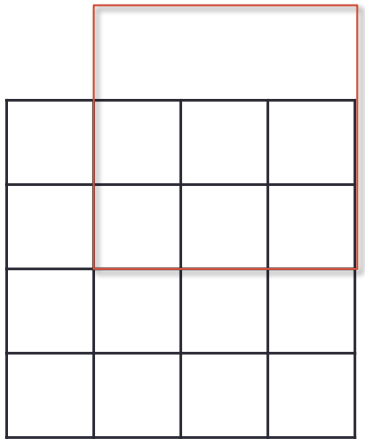
Convolution puzzle

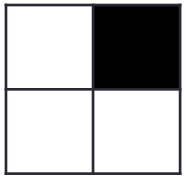
3x3 filter pad, stride 2, pad 1



Convolution puzzle

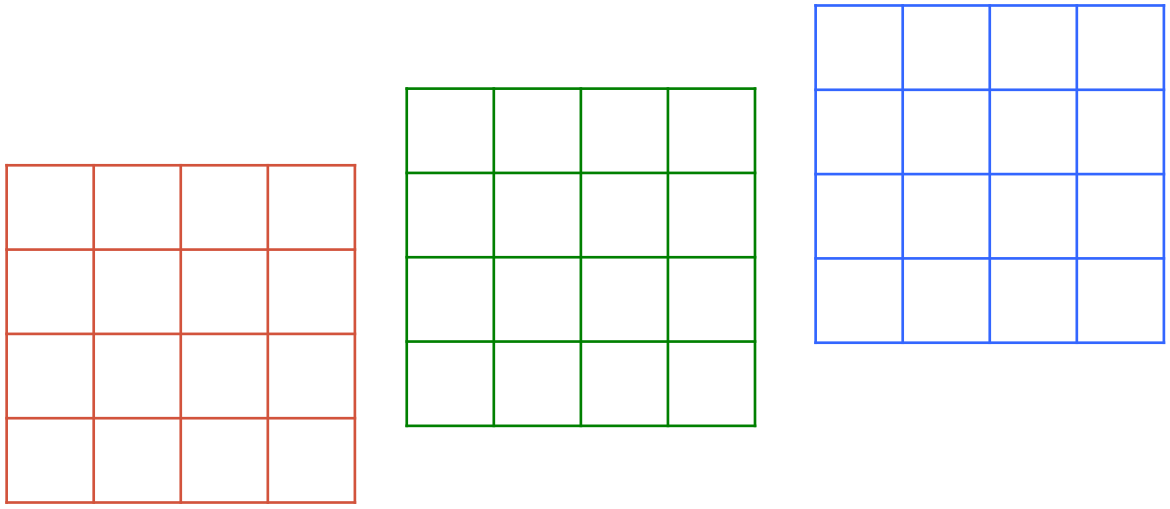
3x3 filter pad, stride 2, pad 1





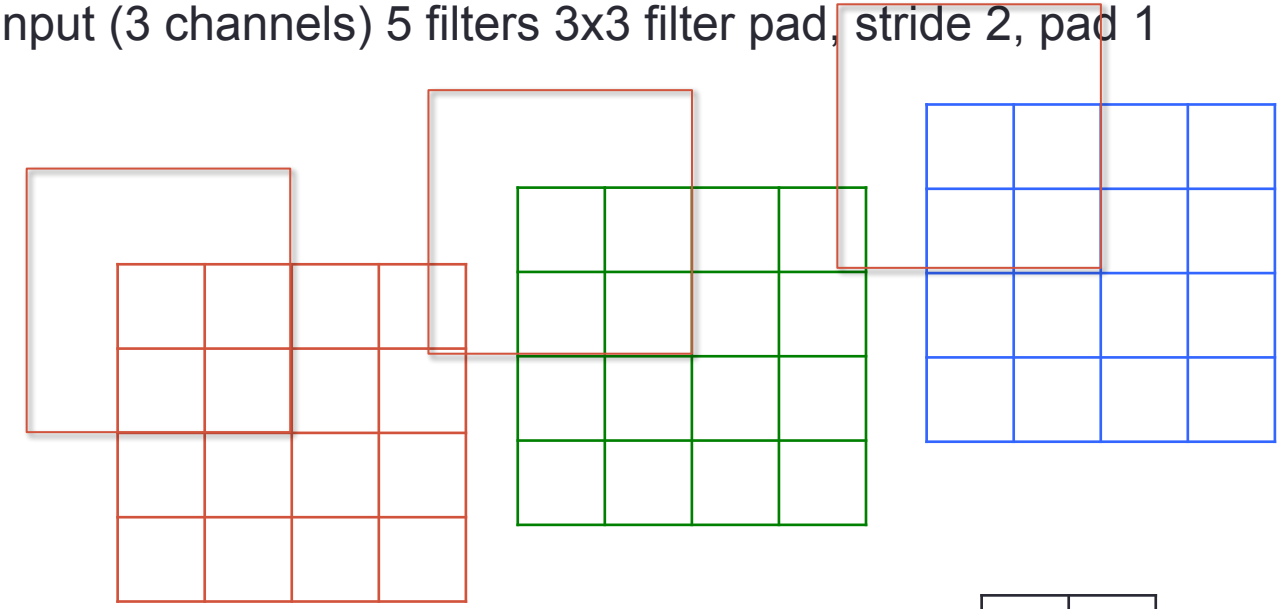
Convolution puzzle

RGB input (3 channels) 5 filters 3x3 filter pad, stride 2, pad 1

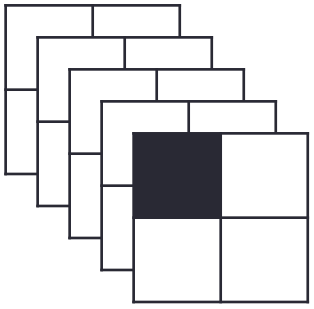


Convolution puzzle

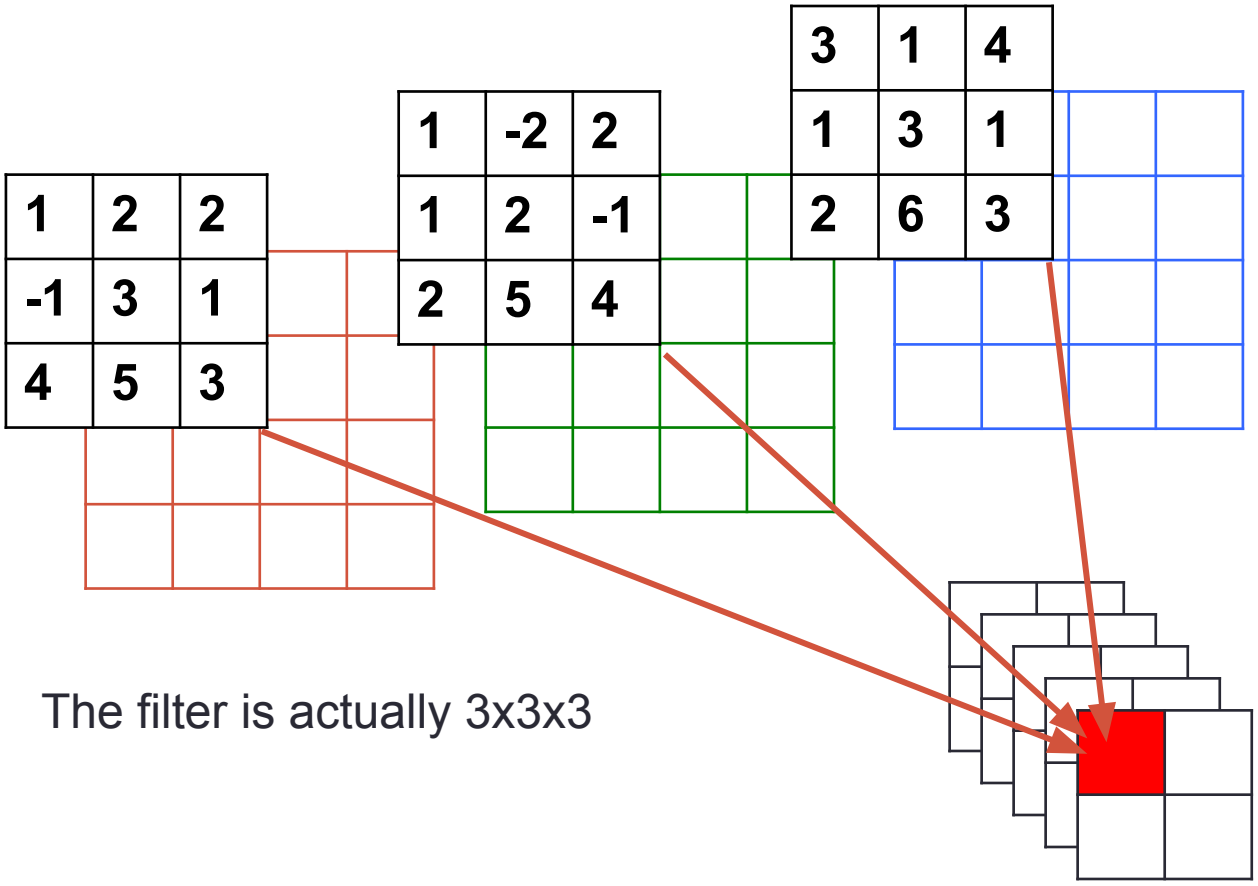
RGB input (3 channels) 5 filters 3x3 filter pad, stride 2, pad 1



The filter is actually 3x3x3



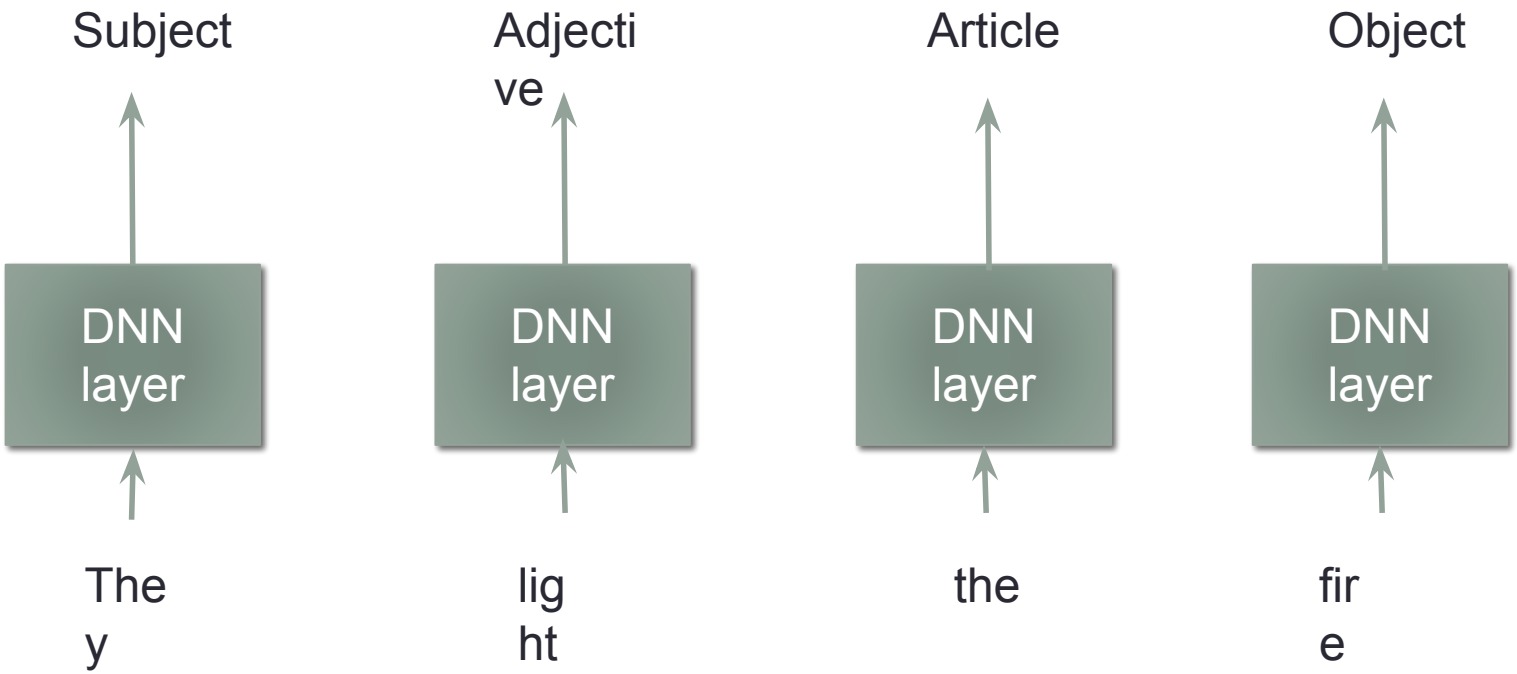
Convolution puzzle



The filter is actually 3x3x3

Recurrent neural network (RNN)

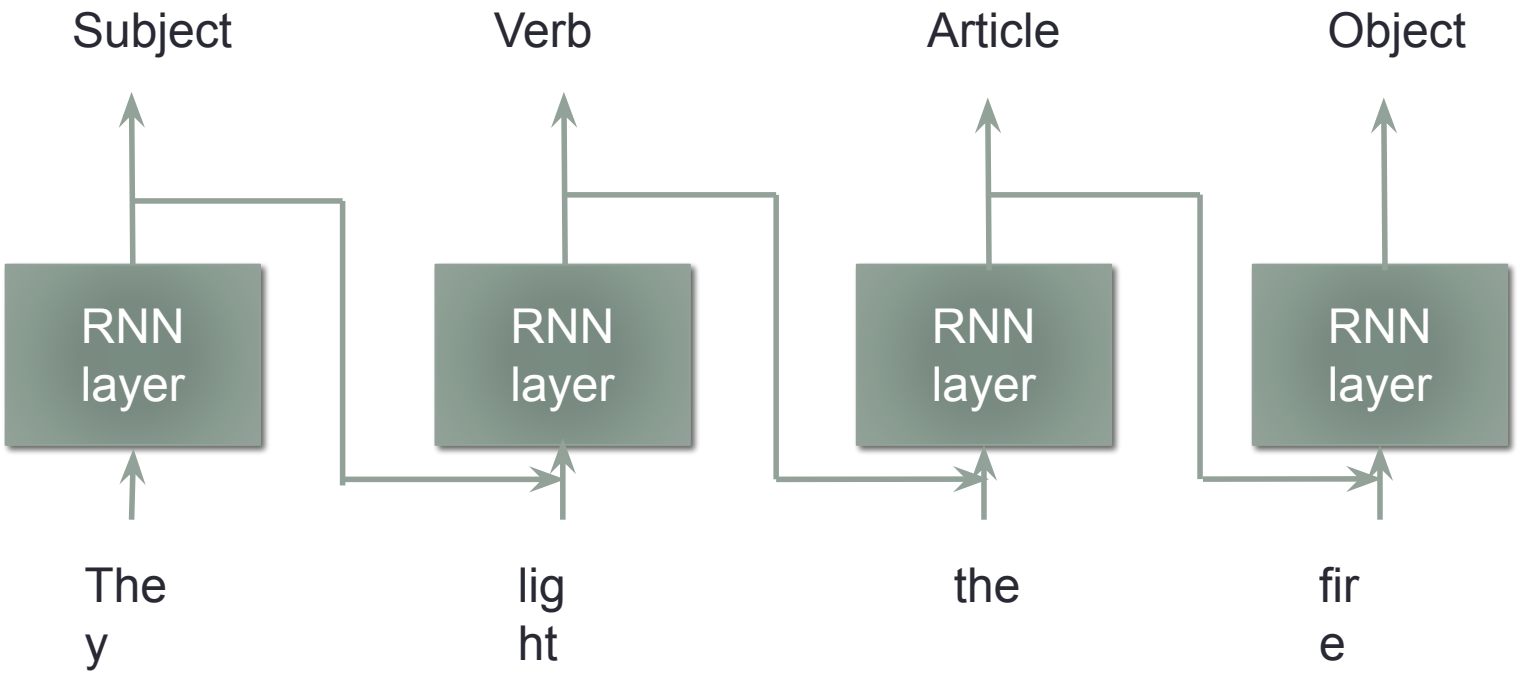
- DNN framework



Problem 1: need a way to remember the past

Recurrent neural network (RNN)

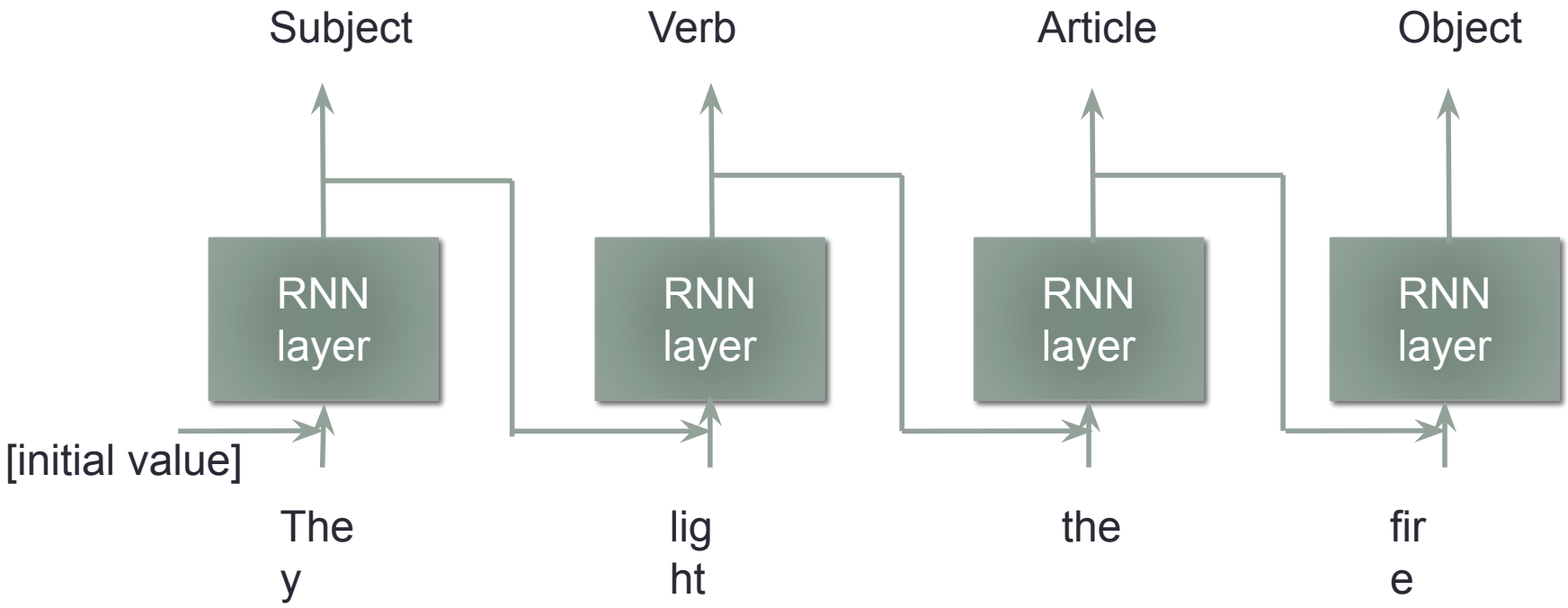
- RNN framework



Output of the layer encodes something meaningful about the past

Recurrent neural network (RNN)

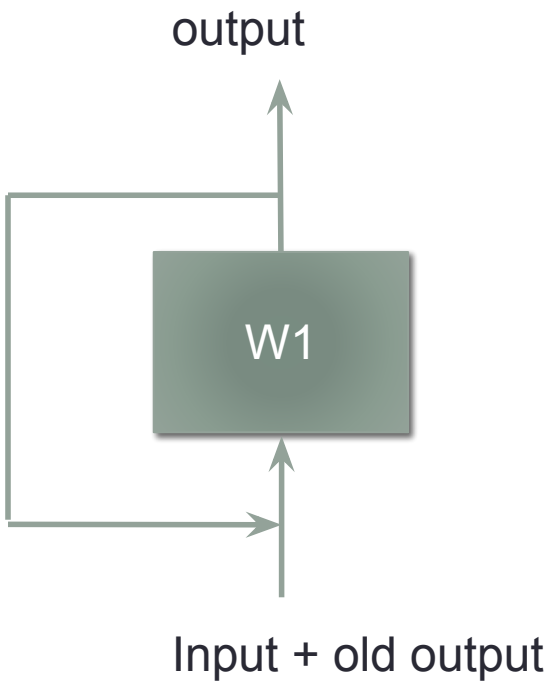
- RNN framework



New input feature = [original input feature, output of the layer at previous time step]

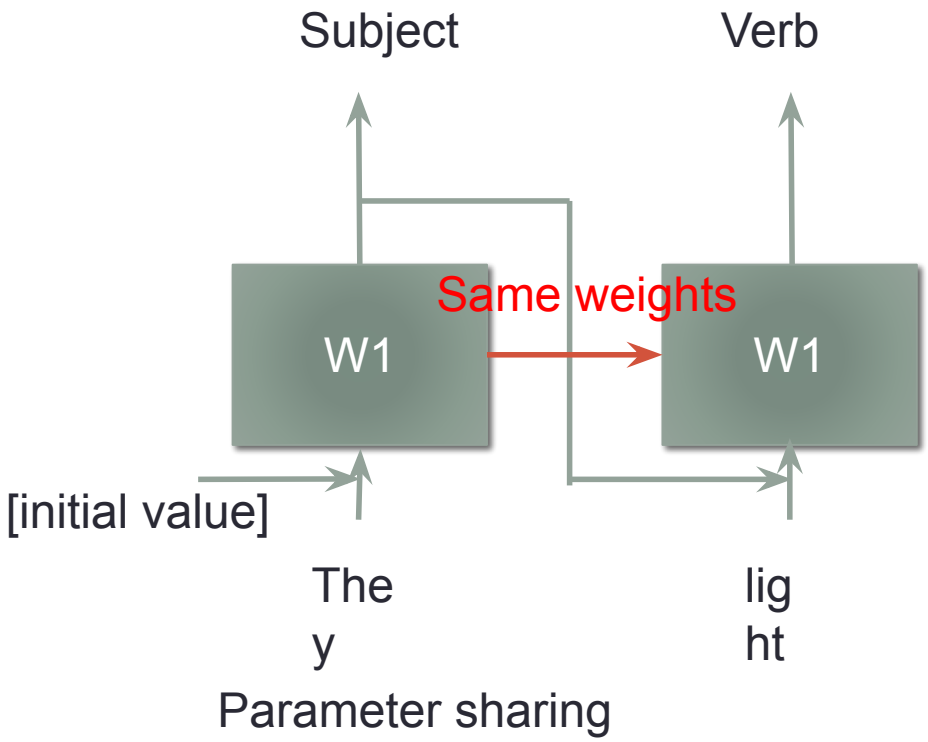
Recurrent neural network (RNN)

- Unrolling of a recurrent layer.



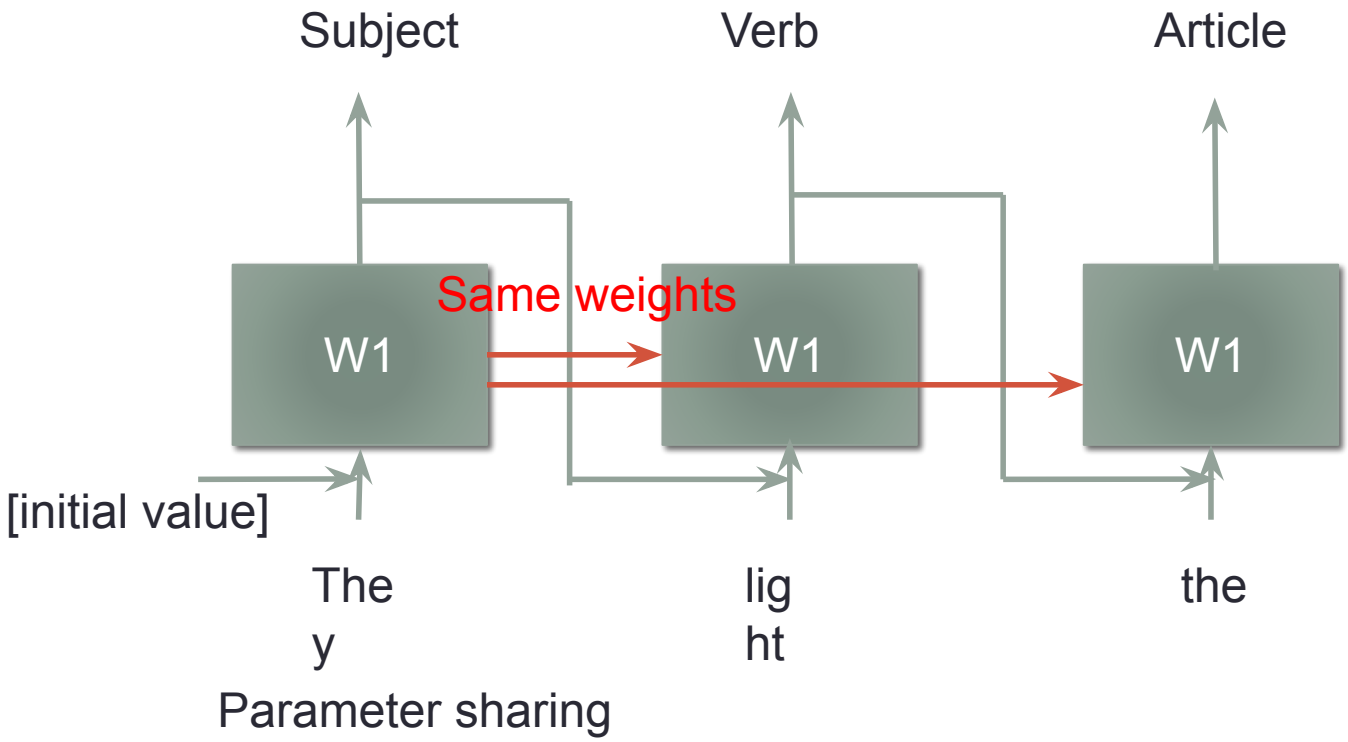
Recurrent neural network (RNN)

- Unrolling of a recurrent layer.



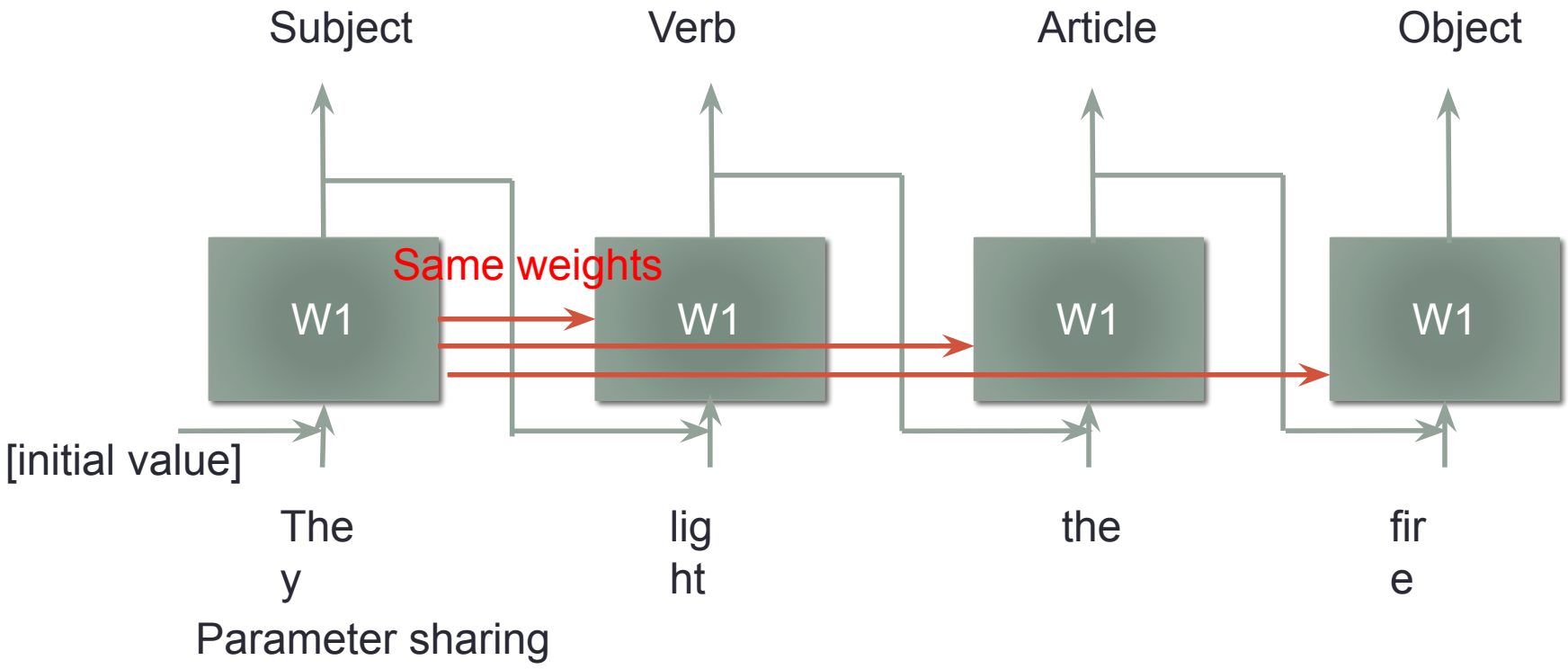
Recurrent neural network (RNN)

- Unrolling of a recurrent layer.



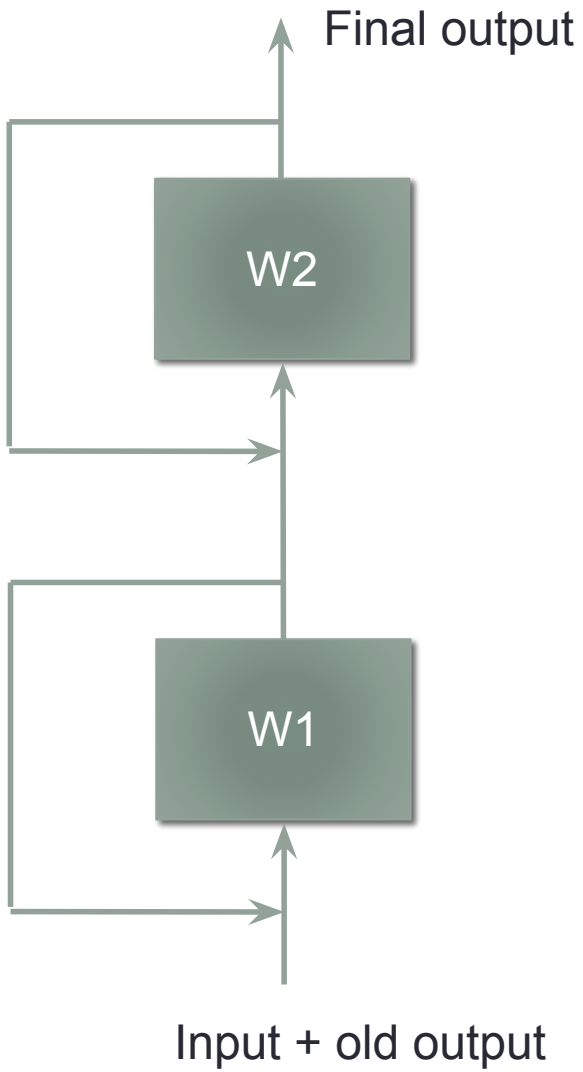
Recurrent neural network (RNN)

- Unrolling of a recurrent layer.

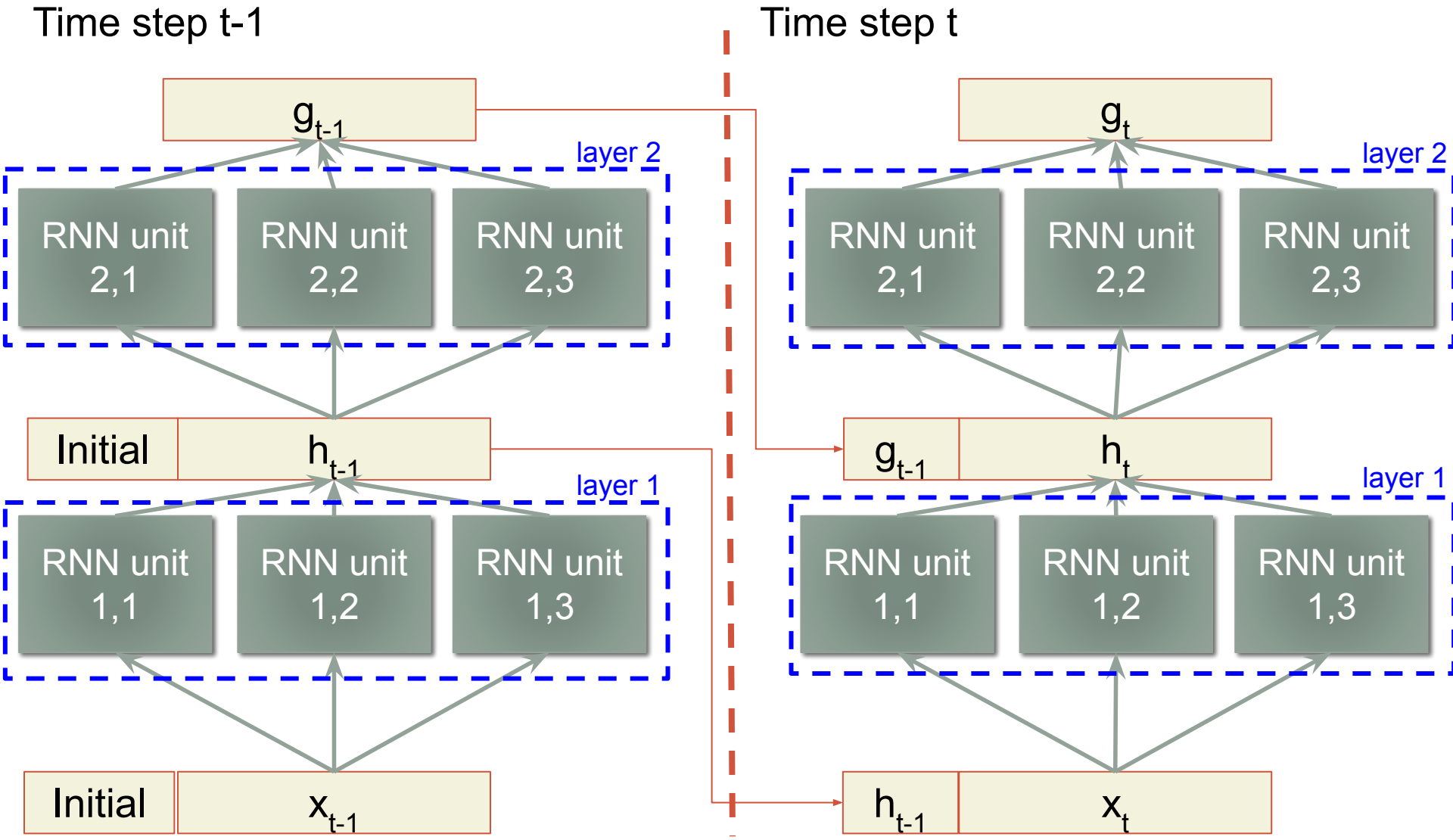


Recurrent neural network (RNN)

- Stacks of recurrent layer



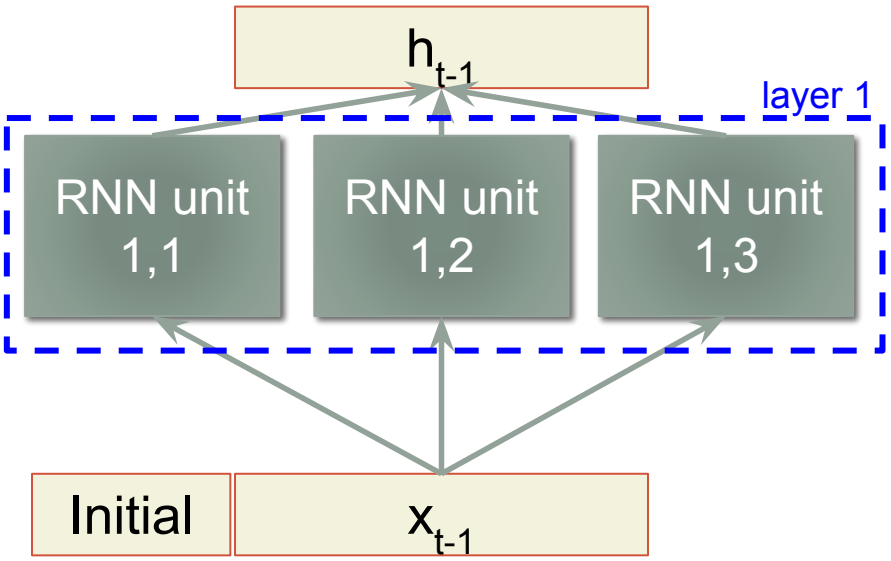
RNN layers (expanded in time)



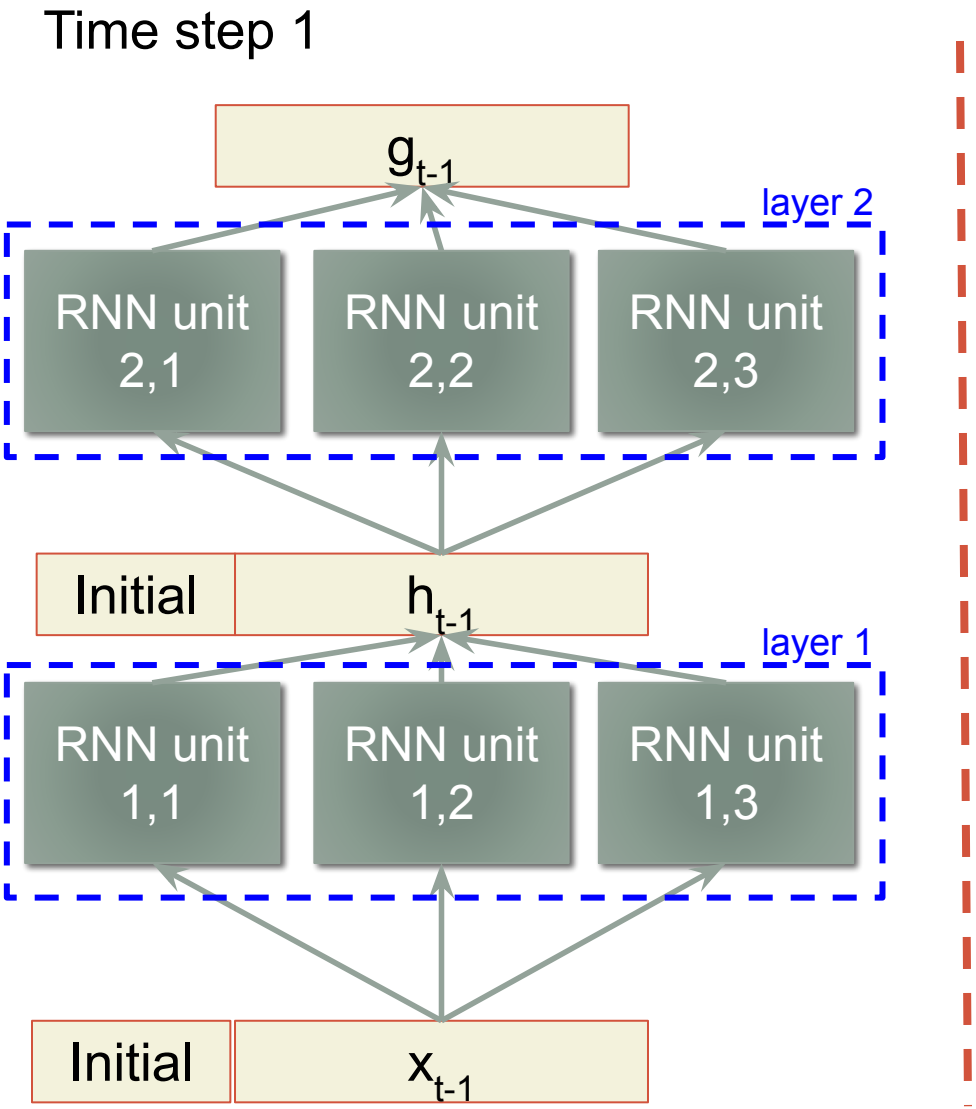
RNN layers (expanded in time)

Time step 1

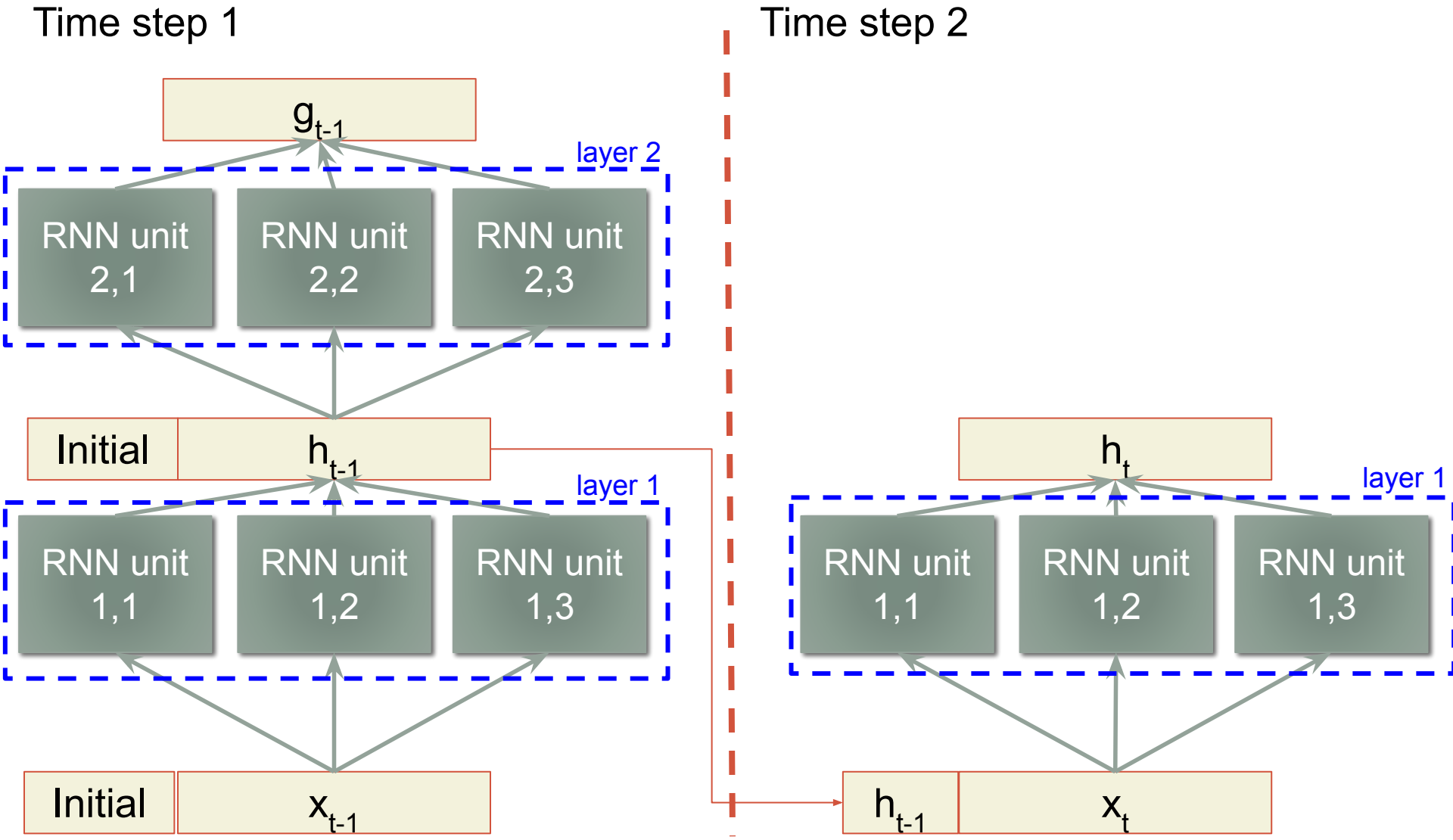
Time step 2



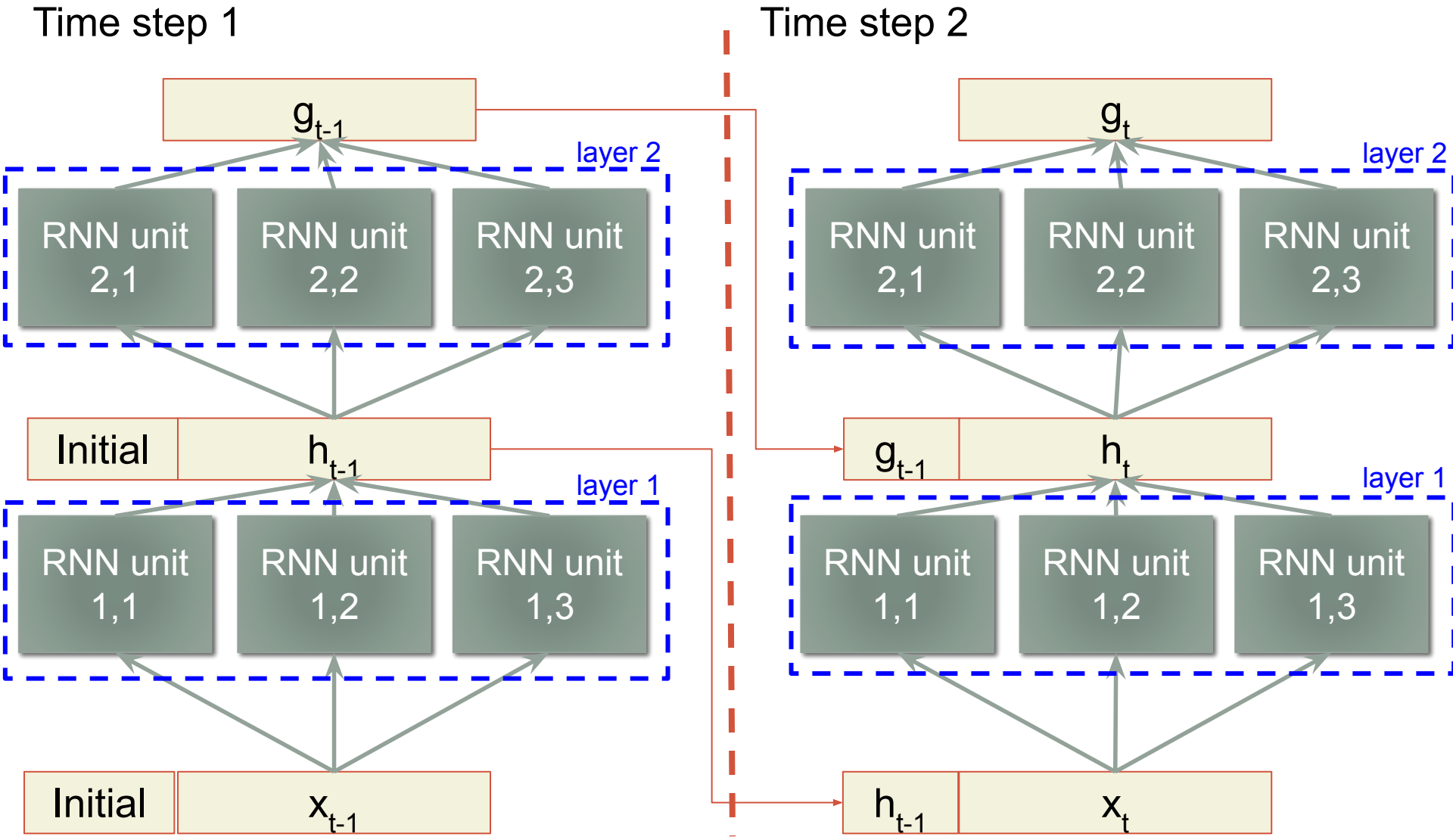
RNN layers (expanded in time)



RNN layers (expanded in time)

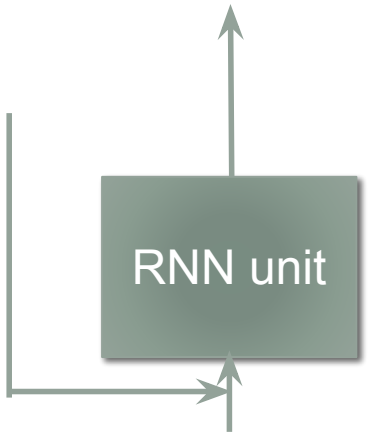


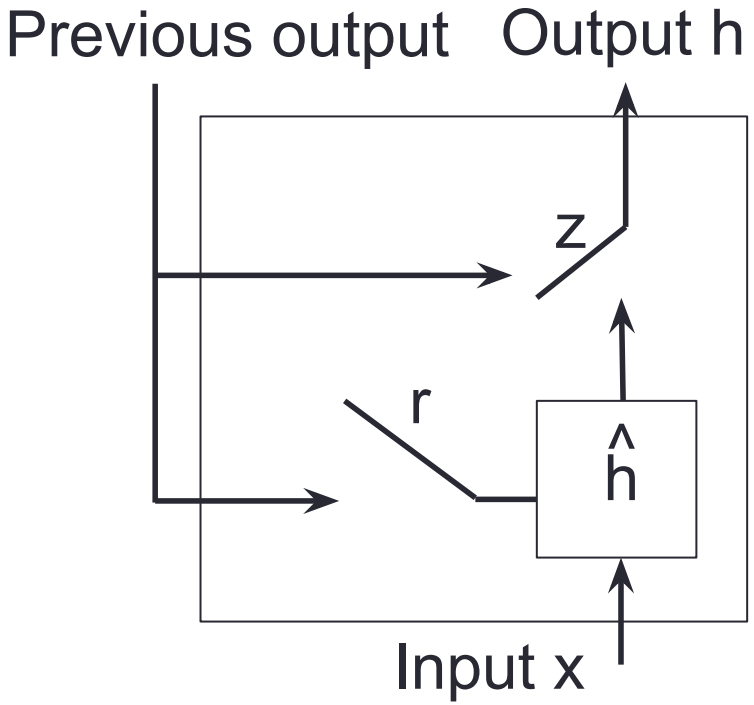
RNN layers (expanded in time)



Gated Recurrent Unit (GRU)

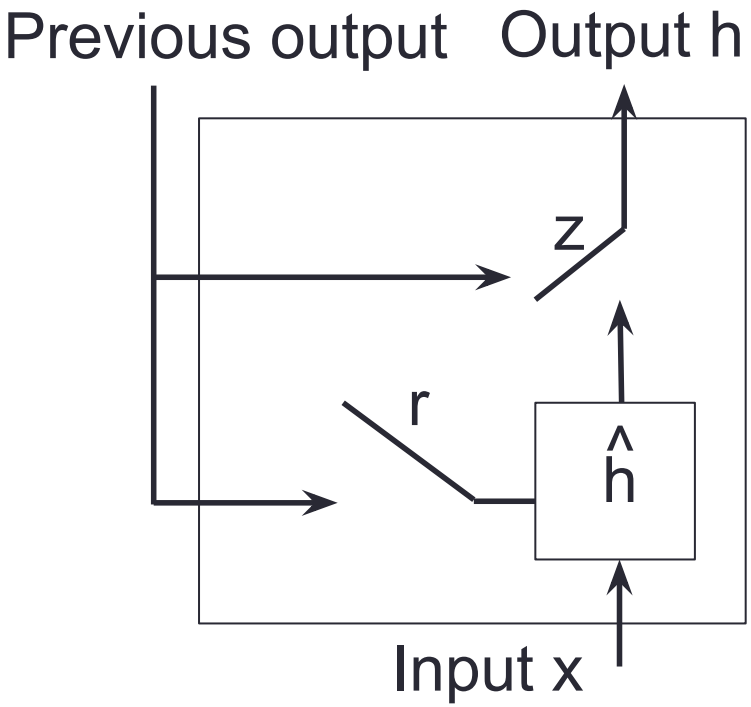
- Forms a Gated Recurrent Neural Networks (GRNN)
- Add gates that can choose to reset (r) or update (z)

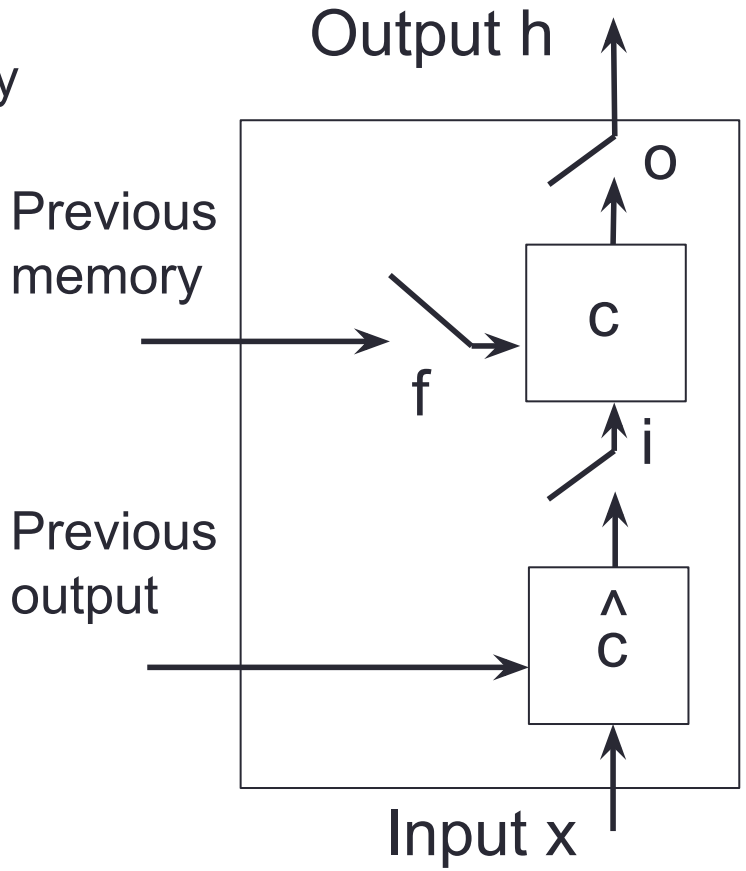




Long Short-Term Memory (LSTM)

- Have 3 gates, forget (f), input (i), output (o)
- Has an explicit memory cell (c)
  - Does not have to output the memory



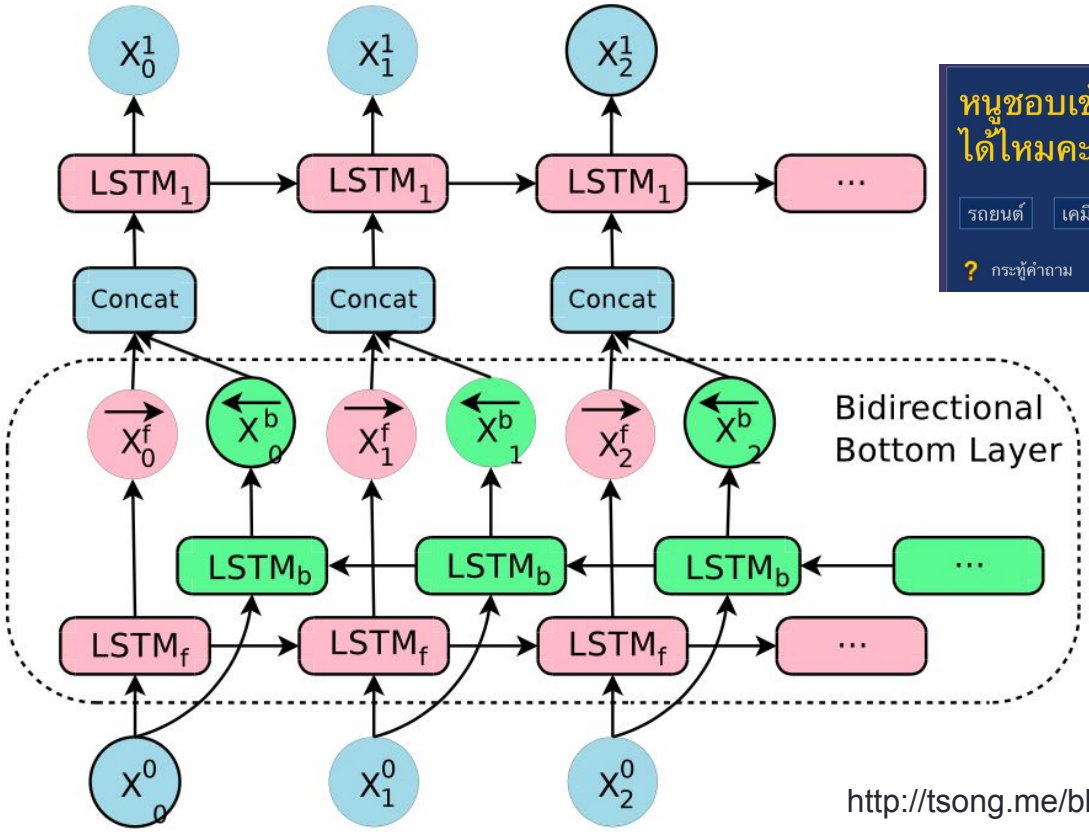


GRU vs LSTM

- GRU and LSTM offers the same performance with large dataset
  - GRU better for smaller dataset (less parameters)
  - GRU faster to train and faster runtime (smaller model)
- Use GRUs!

Bi-directional LSTM

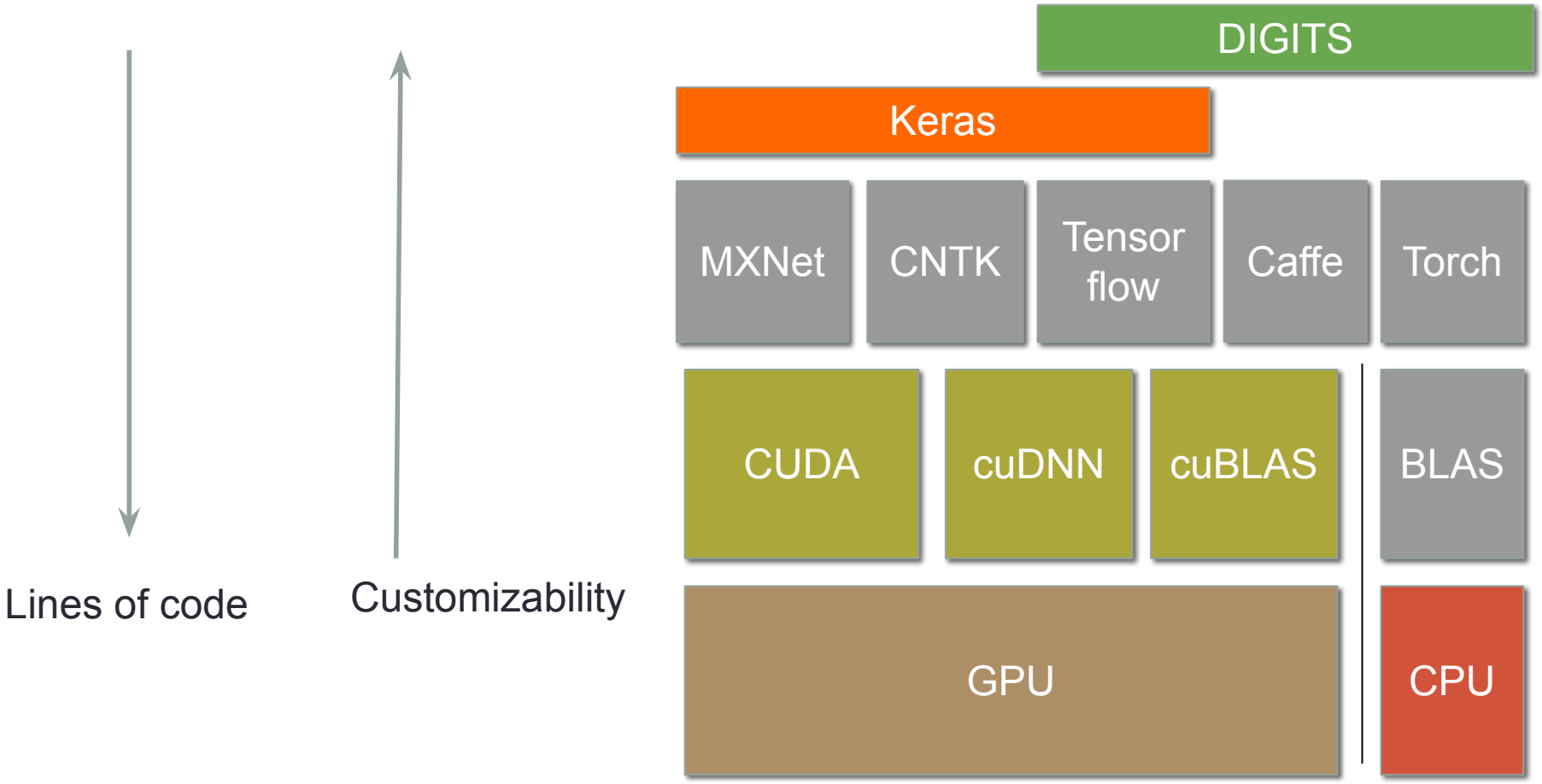
- The previous GRU/LSTM only goes backward in time (uni-directional)
- Most of the time information from the future is useful for predicting the current output



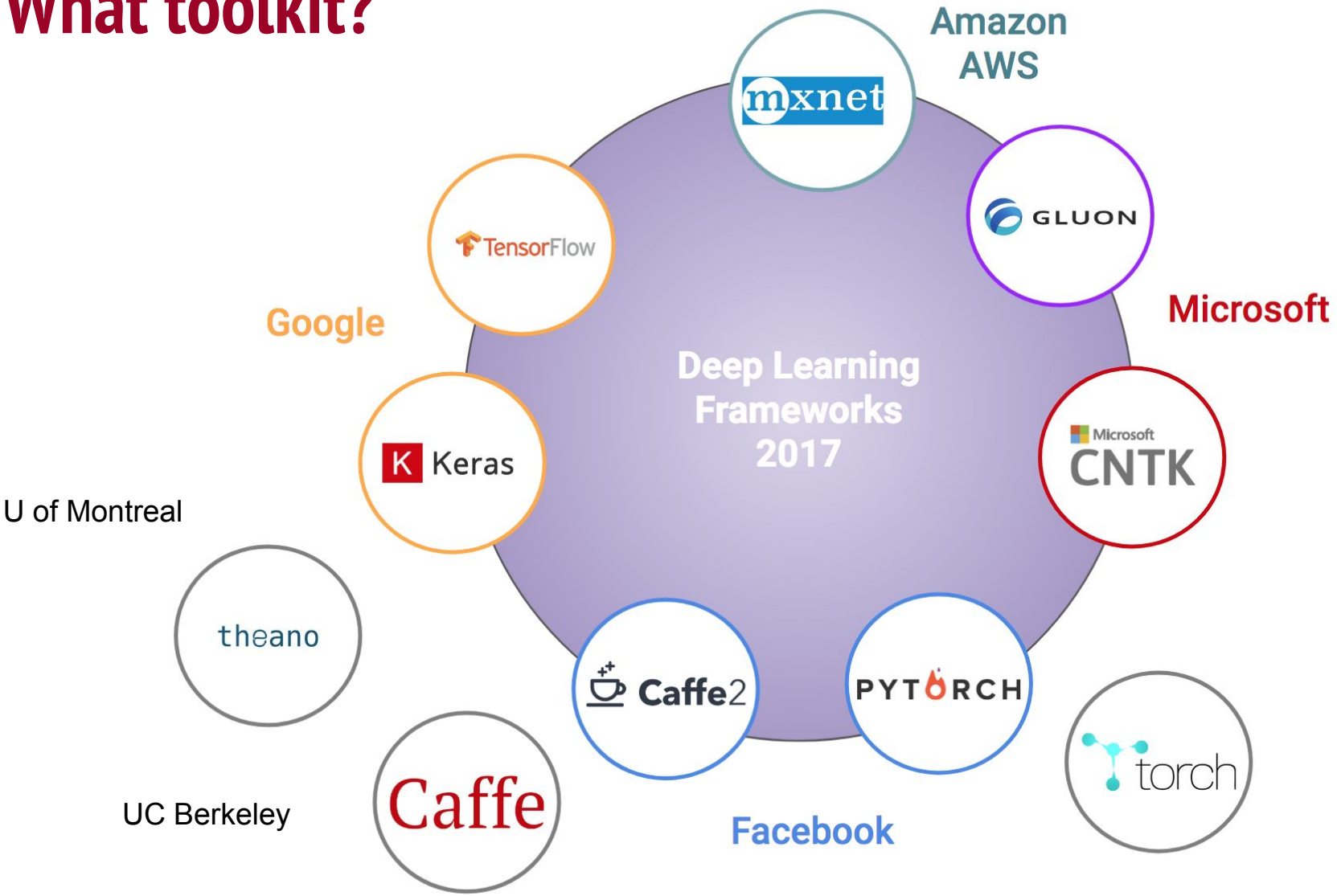
http://tsong.me/blog/google-nmt/

What toolkit

Tradeoff between customizability and ease of use



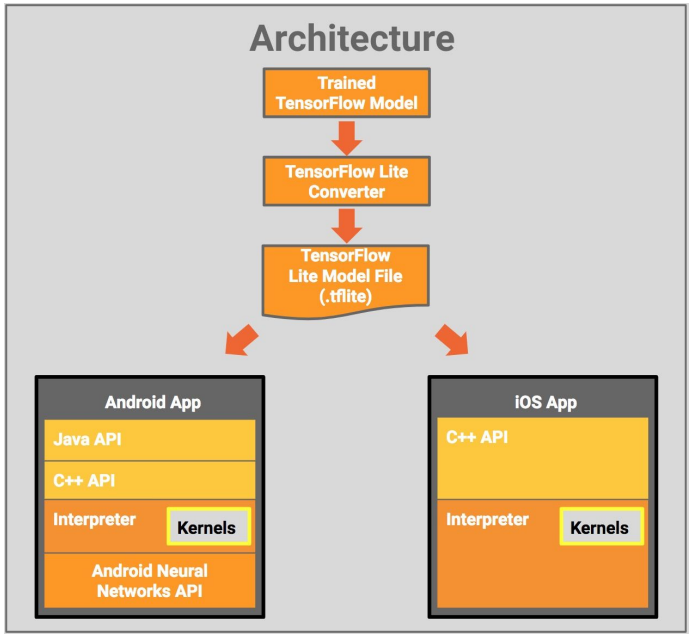
What toolkit?



https://towardsdatascience.com/battle-of-the-deep-learning-frameworks-part-i-cff0e3841750

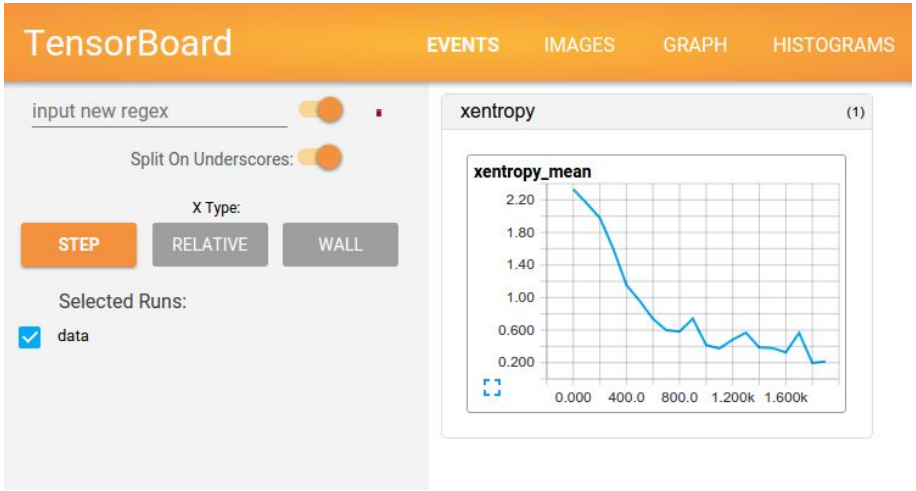
Which?

- Easiest to use and play with deep learning: Keras
- Easiest to use and tweak: pytorch
- Easiest to deploy: tensorflow
  - Tensorflow lite for mobile
  - TensorRT support
- Best tools: TensorFlow
  - Tensorboard



Which?

- Easiest to use and play with deep learning: Keras
- Easiest to use and tweak: pytorch
- Easiest to deploy: tensorflow
  - Tensorflow lite for mobile
  - TensorRT support
- Best tools: TensorFlow
  - Tensorboard
- Community: TensorFlow



Keras steps

- Define the network
- Compile the network
- Fit the network

```
def get_feedforward_nn():
  input1 = Input(shape=(21,))
  x = Dense(100, activation='relu')(input1)
  x = Dense(100, activation='relu')(x)
  x = Dense(100, activation='relu')(x)
  out = Dense(1, activation='sigmoid')(x)

  model = Model(inputs=input1, outputs=out)
  model.compile(optimizer=Adam(),
                loss='binary_crossentropy',
                metrics=['acc'])
  return model
```

```
  model_feedforward_nn.fit(x_train_char, y_train, epochs=epochs, batch_size=batch_size, verbose=ve
rbose,
                           callbacks=callbacks_list_feedforward_nn,
                           validation_data=(x_val_char, y_val))
```

Keras is easy!

Dense [source]

```
keras.layers.Dense(units, activation=None, use_bias=True, kernel_initializer='glorot_uniform', bias_initializ
```

Just your regular densely-connected NN layer.

`Dense` implements the operation: `output = activation(dot(input, kernel) + bias)` where `activation` is the element-wise activation function passed as the `activation` argument, `kernel` is a weights matrix created by the layer, and `bias` is a bias vector created by the layer (only applicable if `use_bias` is `True`).

- **Note**: if the input to the layer has a rank greater than 2, then it is flattened prior to the initial dot product with `kernel`.

Example

```
# as first layer in a sequential model:
model = Sequential()
model.add(Dense(32, input_shape=(16,)))
# now the model will take as input arrays of shape (*, 16)
# and output arrays of shape (*, 32)

# after the first layer, you don't need to specify
# the size of the input anymore:
model.add(Dense(32))
```

Dropout [source]

```
keras.layers.Dropout(rate, noise_shape=None, seed=None)
```

Applies Dropout to the input.

Dropout consists in randomly setting a fraction `rate` of input units to 0 at each update during training time, which helps prevent overfitting.

Arguments

- **rate**: float between 0 and 1. Fraction of the input units to drop.
- **noise_shape**: 1D integer tensor representing the shape of the binary dropout mask that will be multiplied with the input. For instance, if your inputs have shape `(batch_size, timesteps, features)` and you want the dropout mask to be the same for all timesteps, you can use `noise_shape=(batch_size, 1, features)`.
- **seed**: A Python integer to use as random seed.

Conv1D

[source]

Number of filters

```
keras.layers.Conv1D(filters, kernel_size, strides=1, padding='valid', dilation_rate=1, activation=None, use_b
```

Size of filter

1D convolution layer (e.g. temporal convolution).

This layer creates a convolution kernel that is convolved with the layer input over a single spatial (or temporal) dimension to produce a tensor of outputs. If `use_bias` is True, a bias vector is created and added to the outputs. Finally, if `activation` is not `None`, it is applied to the outputs as well.

When using this layer as the first layer in a model, provide an `input_shape` argument (tuple of integers or `None`, e.g. `(10, 128)` for sequences of 10 vectors of 128-dimensional vectors, or `(None, 128)` for variable-length sequences of 128-dimensional vectors.

Arguments

- **filters**: Integer, the dimensionality of the output space (i.e. the number output of filters in the convolution).
- **kernel_size**: An integer or tuple/list of a single integer, specifying the length of the 1D convolution window.
- **strides**: An integer or tuple/list of a single integer, specifying the stride length of the convolution. Specifying any stride value != 1 is incompatible with specifying any `dilation_rate` value != 1.
- **padding**: One of `"valid"`, `"causal"` or `"same"` (case-insensitive). `"valid"` means "no padding". `"same"` results in padding the input such that the output has the same length as the original input. `"causal"` results in causal (dilated) convolutions, e.g. output[t] does not depend on input[t+1:]. Useful when modeling temporal data where the model should not violate the temporal order. See WaveNet: A Generative Model for Raw Audio, section 2.1.