NEURAL NETWORKS

Deep learning = Deep neural networks =
neural networks

Material:

DNNs (Deep Neural Networks)

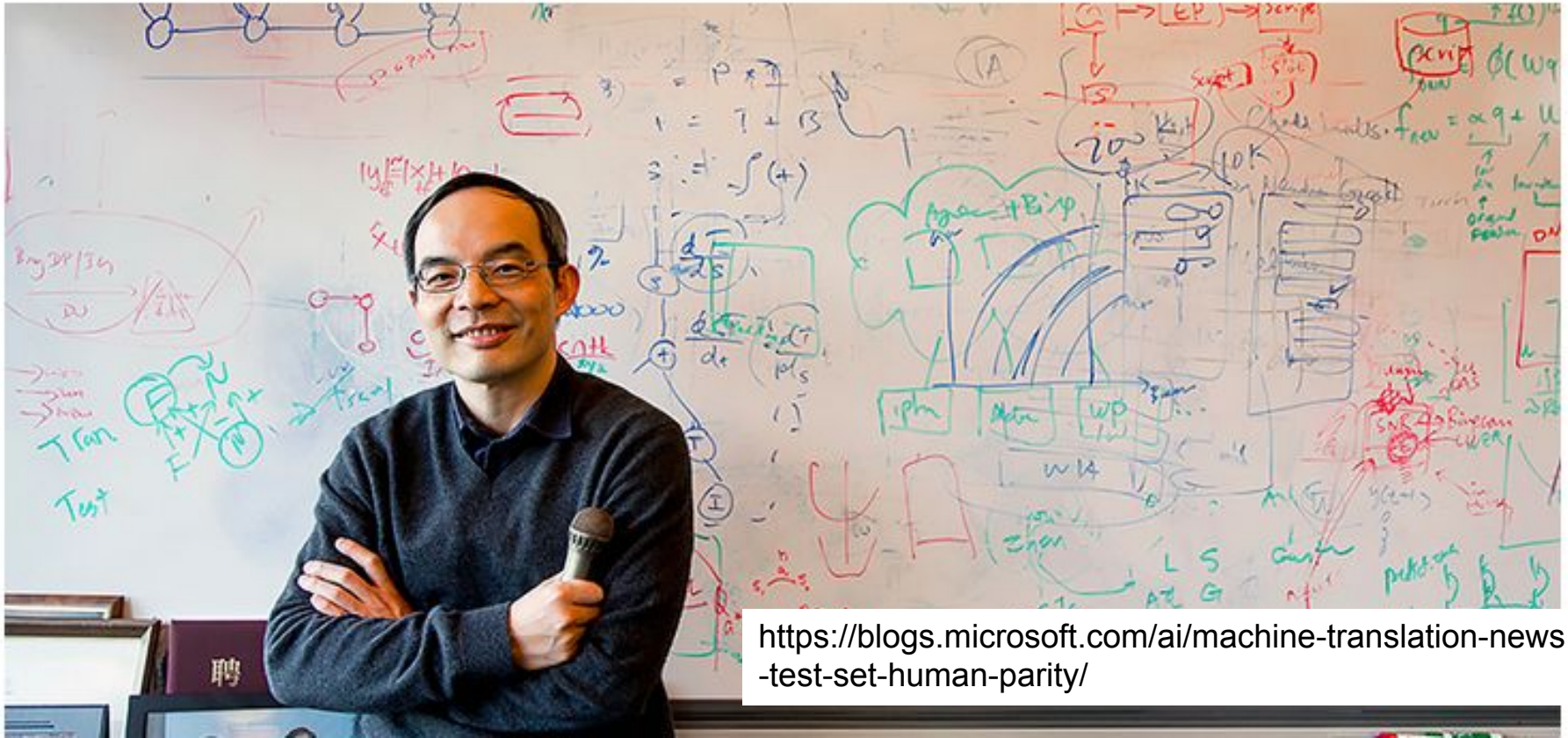
- Why deep learning?
- Greatly improved performance in ASR and other tasks (Computer Vision, Robotics, Machine Translation, NLP, etc.)
- Surpassed human performance in many tasks

Task	Previous state-of-the-art	Deep learning (2012)	Deep learning (2017)
IMAGENET	26%	16%	4%
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/

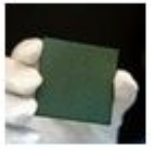
Google's AlphaGo Defeats Chinese Go Master in Win for A.I.

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

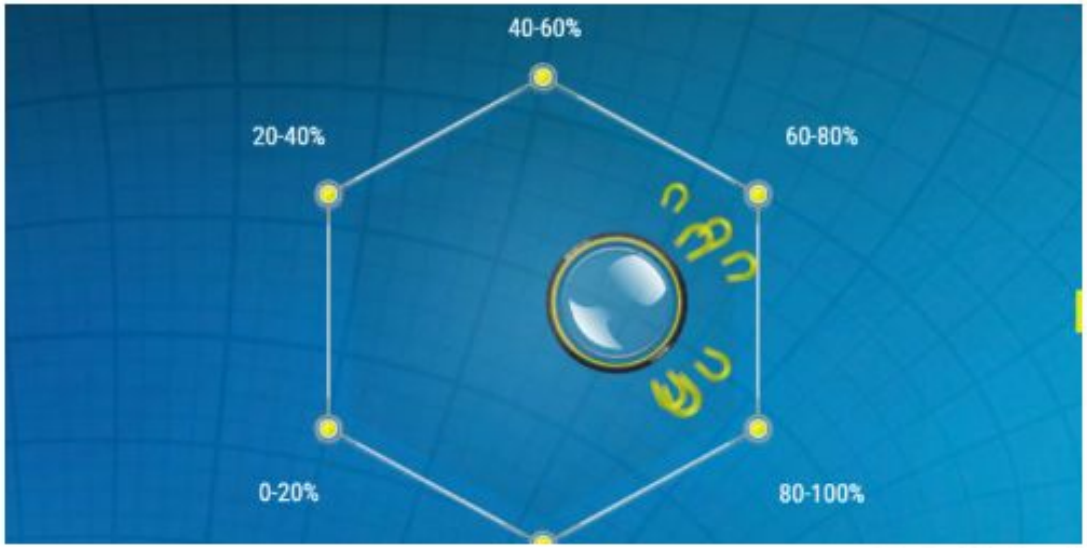

Maste
Goog

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

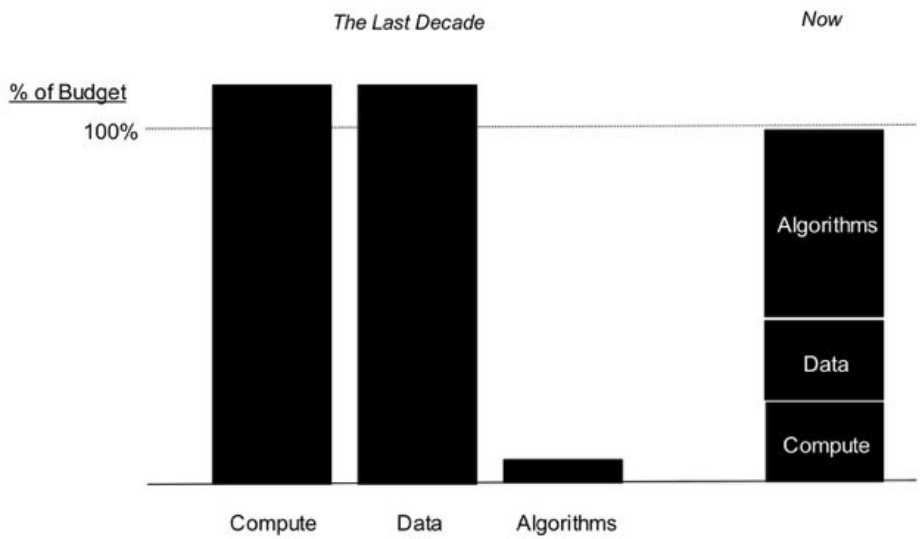
Popular Posts

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

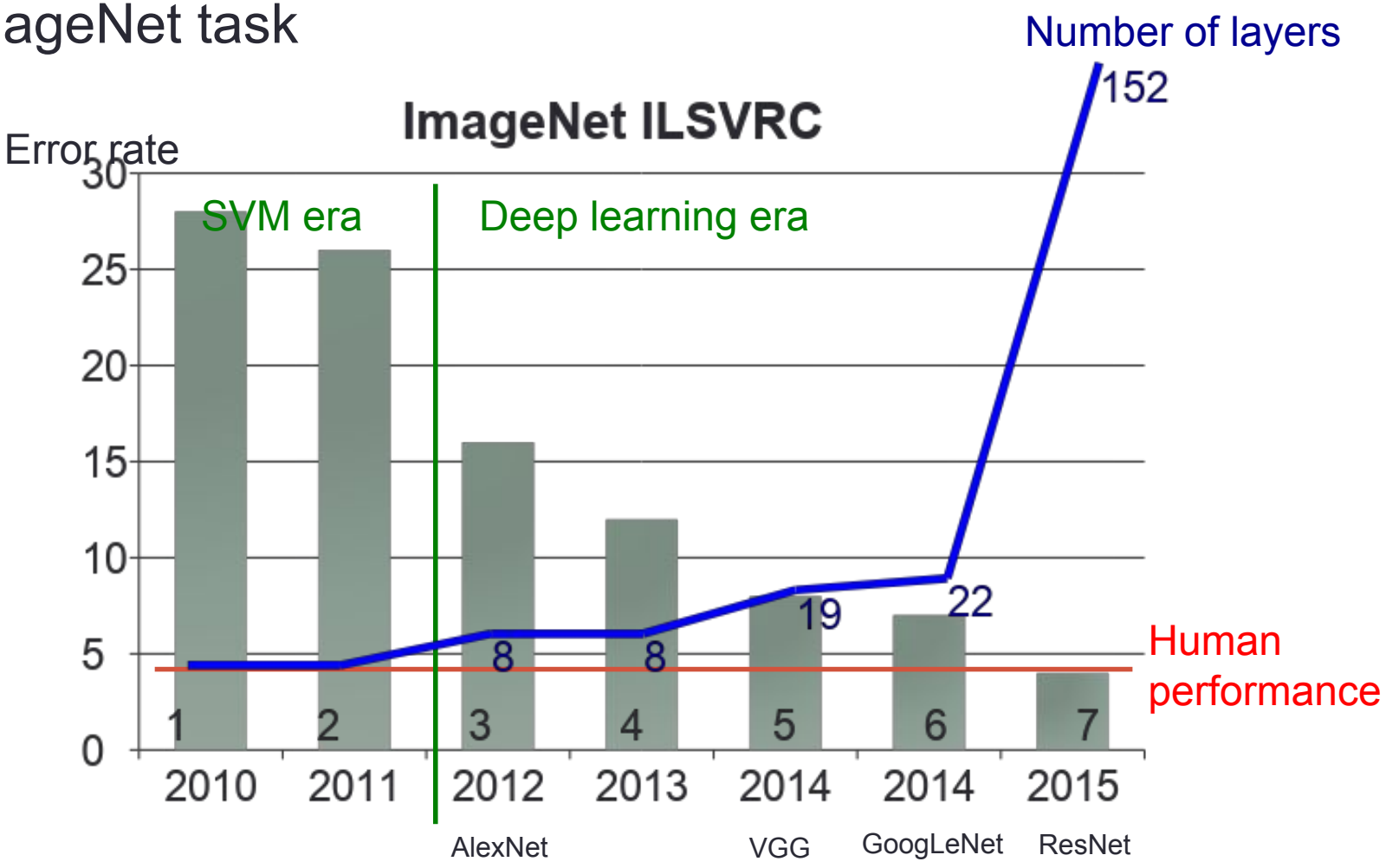
- Neural Networks has been around since 1990s
- Big data – DNN can take advantage of large amounts of data better than other models
- GPU – Enable training bigger models possible
- Deep – Easier to avoid bad local minima when the model is large



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

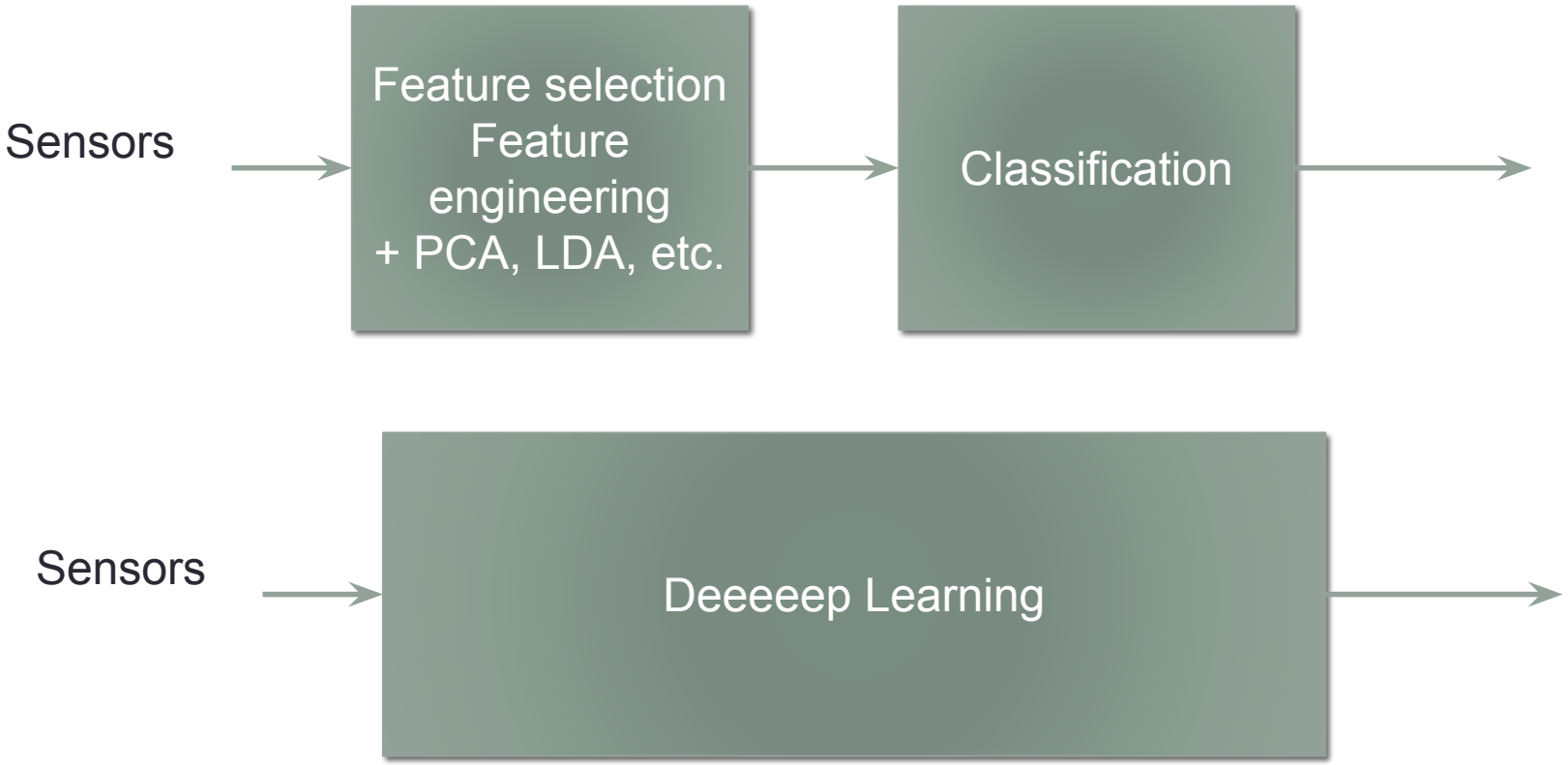
Wider and deeper networks

- ImageNet task



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

Traditional VS Deep learning

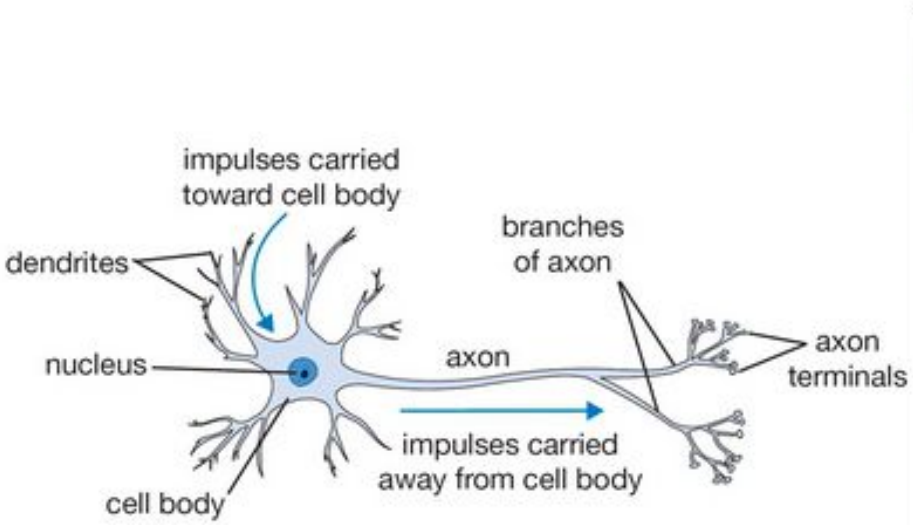


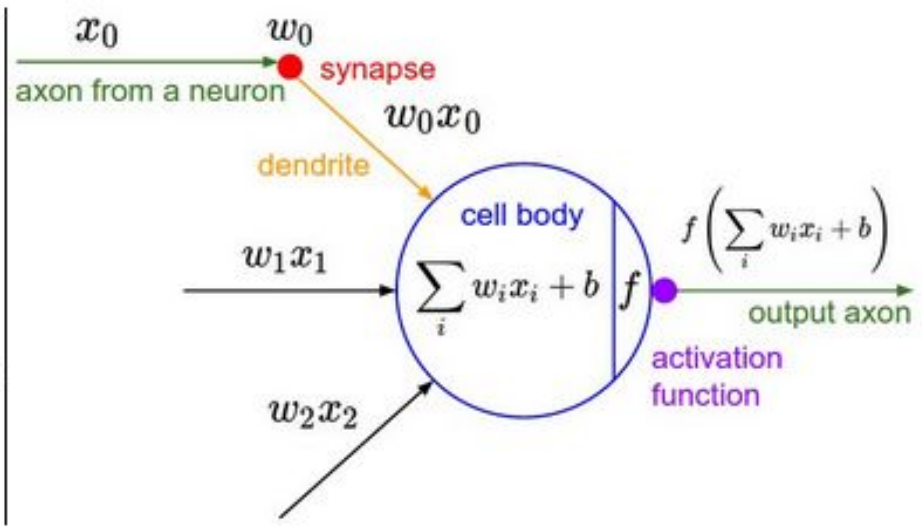
Neural networks

- Fully connected networks
  - Neuron
  - Non-linearity
  - Softmax layer
- DNN training
  - Loss function
  - SGD and backprop
  - Learning rate
  - Overfitting
- 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



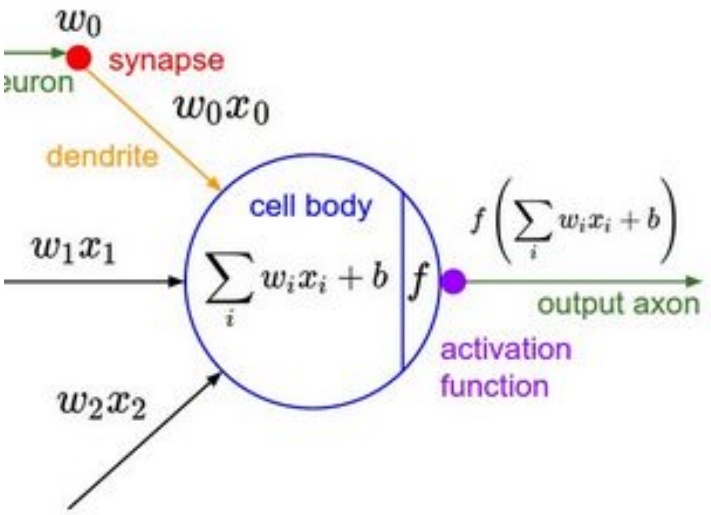


Comparing with linear regression

$h_\theta(\mathbf{x}_1) = \theta_0 + \theta_1 x_{1,1} + \theta_2 x_{1,2} + \theta_3 x_{1,3} + \theta_4 x_{1,4} + \theta_5 x_{1,5}$

We can rewrite

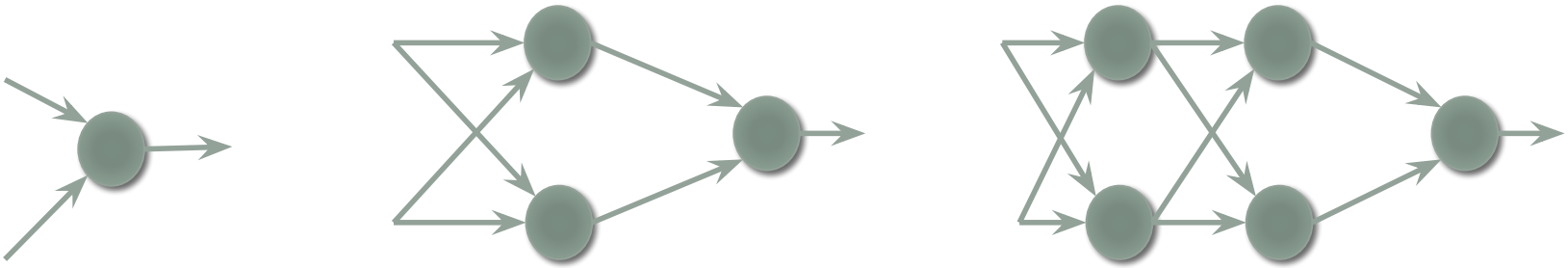
$$h_\theta(\mathbf{x}_i) = \sum_{j=0}^{n} \theta_j x_{i,j}$$

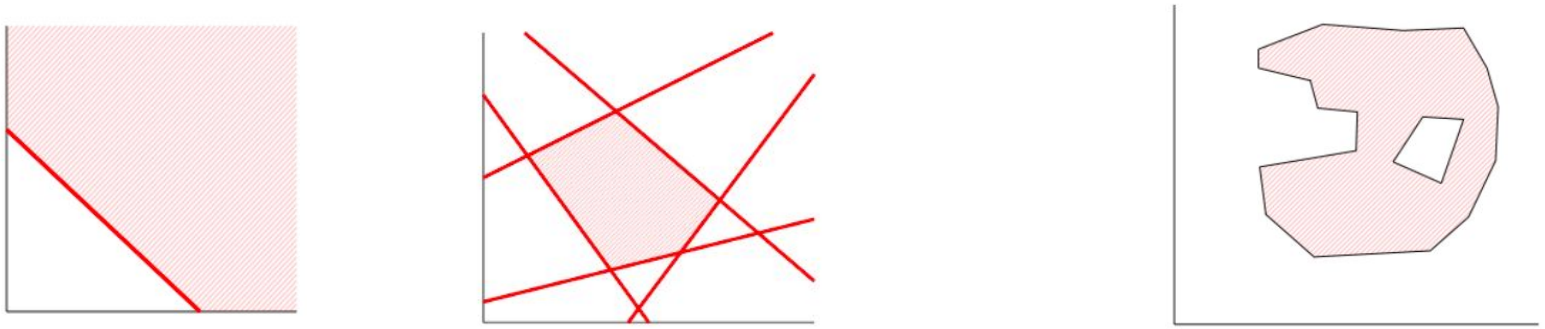


- One neuron is like linear regression but with no linearity

Combining neurons

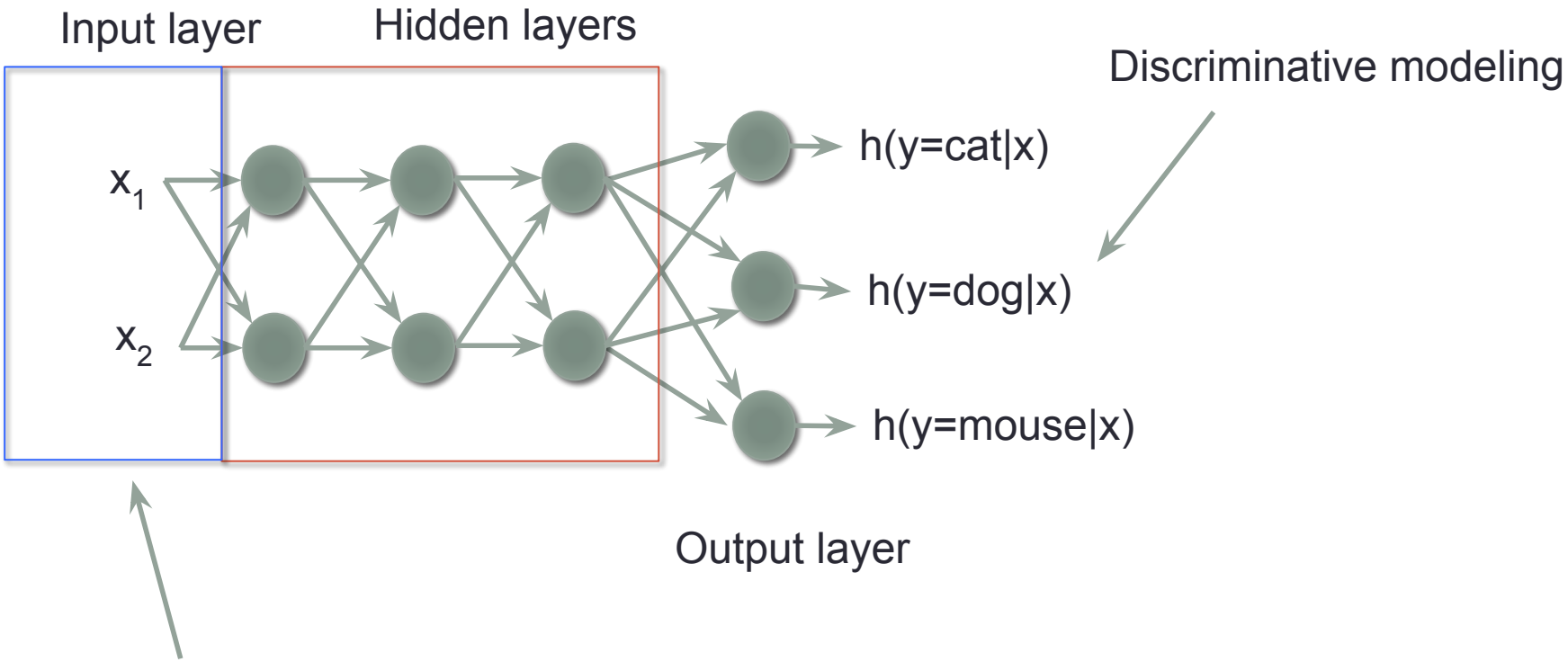
- Each neuron splits the feature space with a hyperplane
- Stacking neuron creates more complicated decision boundaries
- More powerful but prone to overfitting





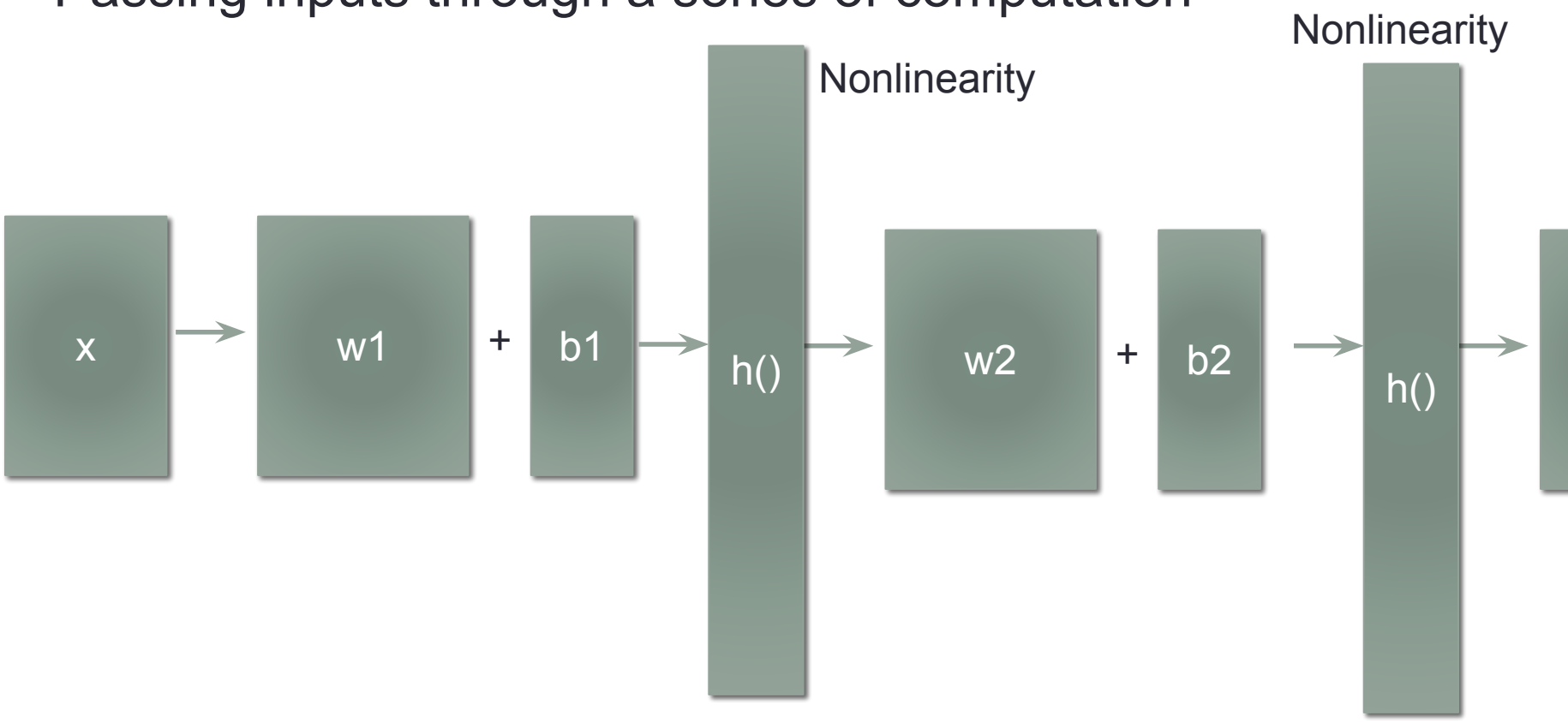
Terminology

Deep in Deep neural networks means many hidden layers



Computation graph

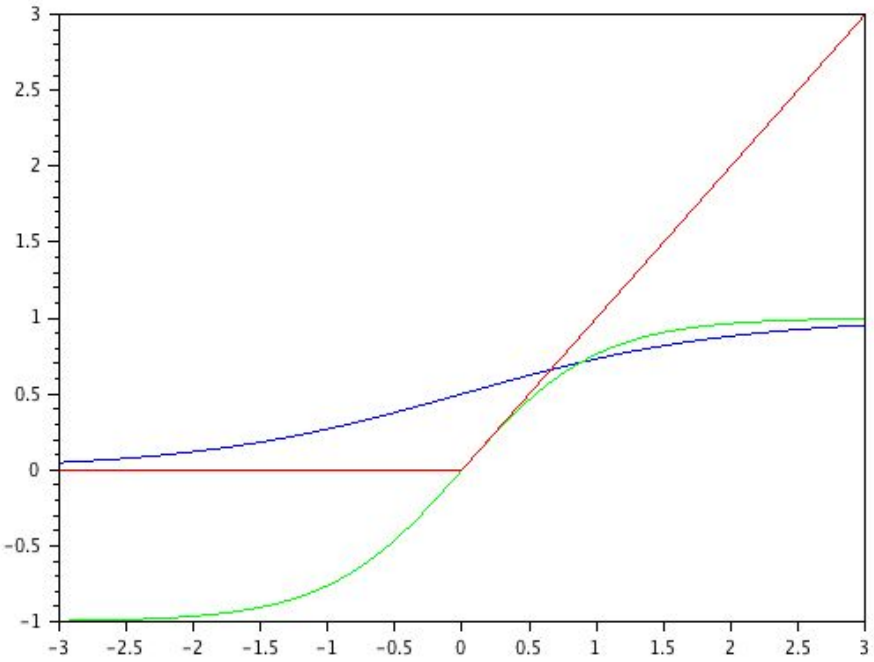
- Passing inputs through a series of computation



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

Non-linearity

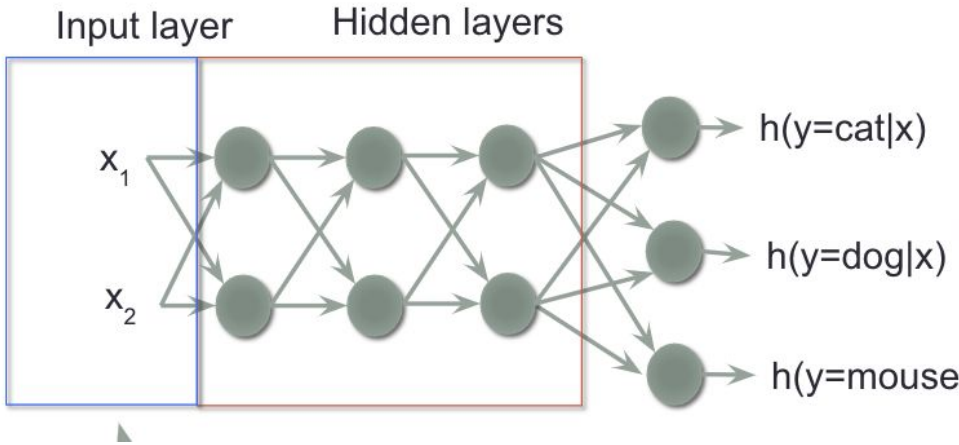
- The Non-linearity is important in order to stack neurons
  - If F is linear, a multi layered network can be collapsed as a single layer (by just multiplying weights together)
- Sigmoid or logistic function
- tanh
- Rectified Linear Unit (ReLU)
- Most popular is ReLU and its variants (Fast to train, and more stable)



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

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



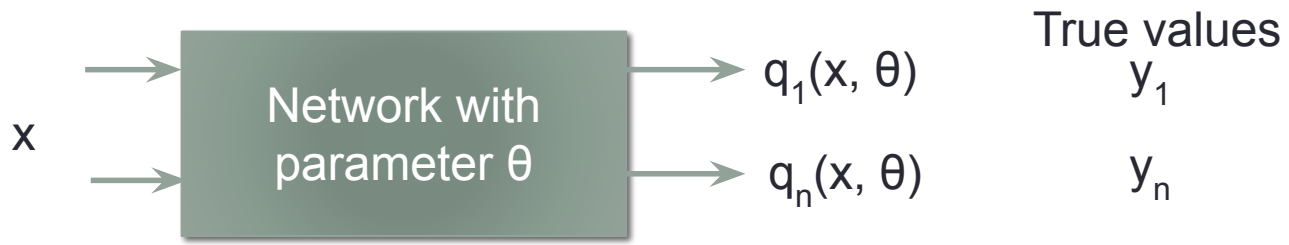
Neural networks

- Fully connected networks
  - Neuron
  - Non-linearity
  - Softmax layer
- DNN training
  - Loss function
  - SGD and backprop
  - Learning rate
  - Overfitting
- 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

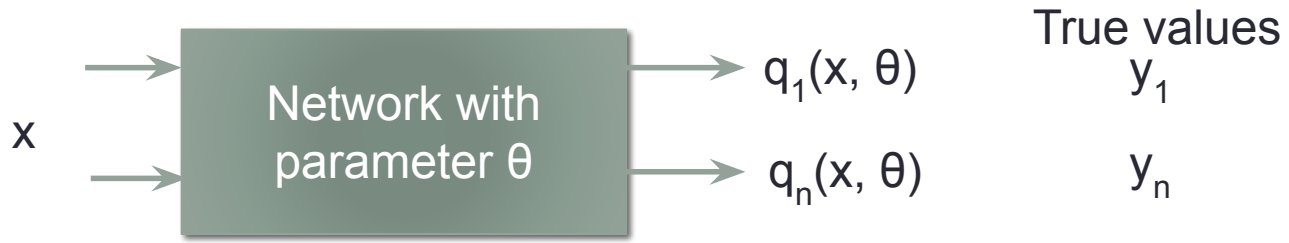


Sum of squared errors

- 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



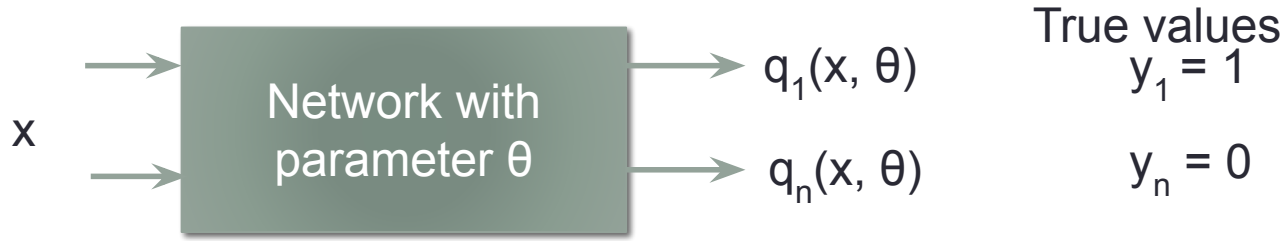
Cross entropy loss

- 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

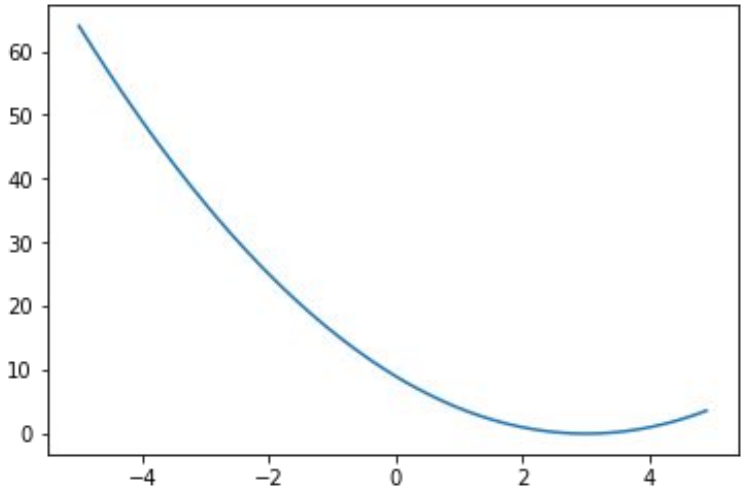
- L only has the term from the correct class
- L is non negative with highest value when the output matches the true values, a “loss” function



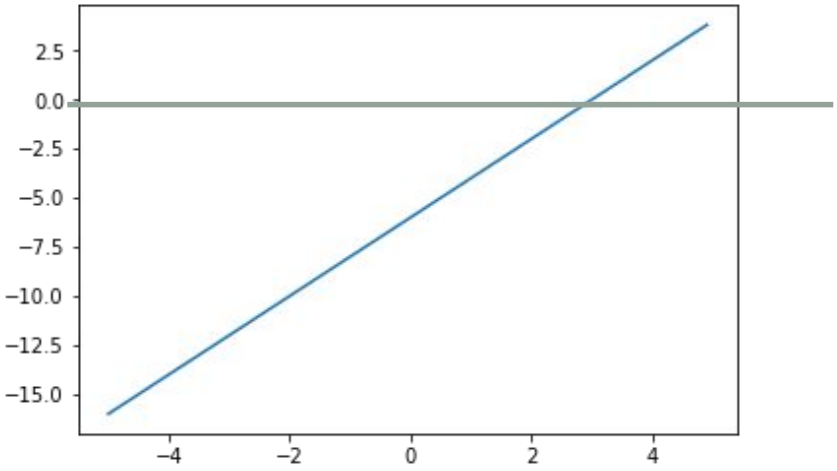
Minimizing the loss

- You have a function
  - $y = (x - a)^2$
- You want to minimize Y with respect to x
  - $dy/dx = 2x - 2a$
  - Take the derivative and set the derivative to 0
    - (And maybe check if it's a minima, maxima or saddle point)
- We can also go with an iterative approach

Gradient descent



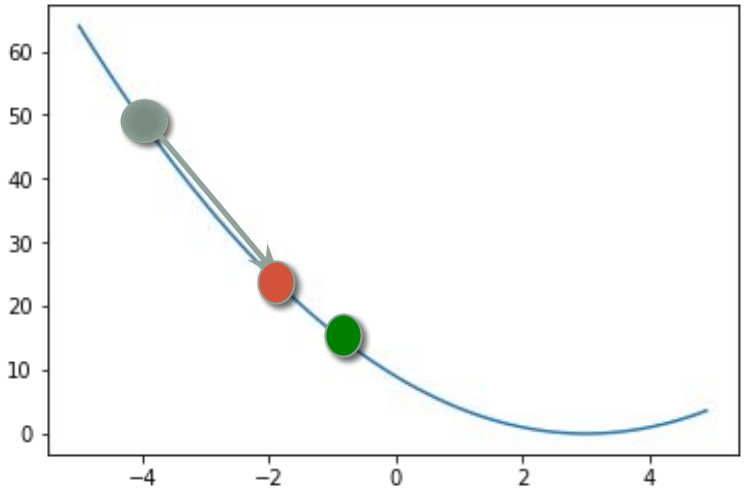
y



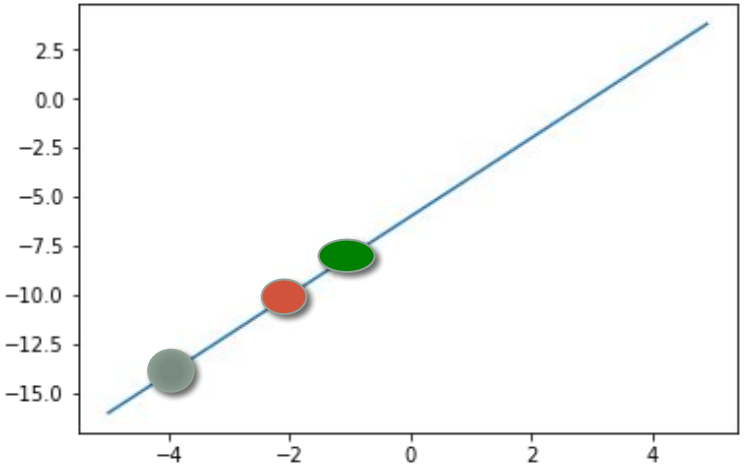
dy/dx

First what does dy/dx means?

Gradient descent



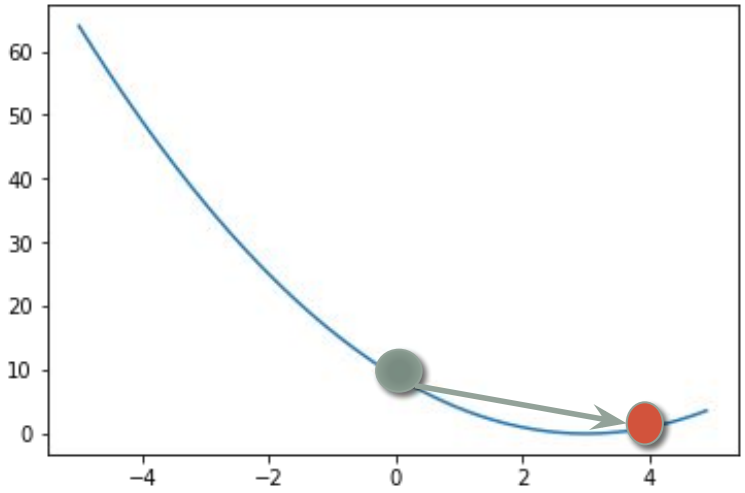
y



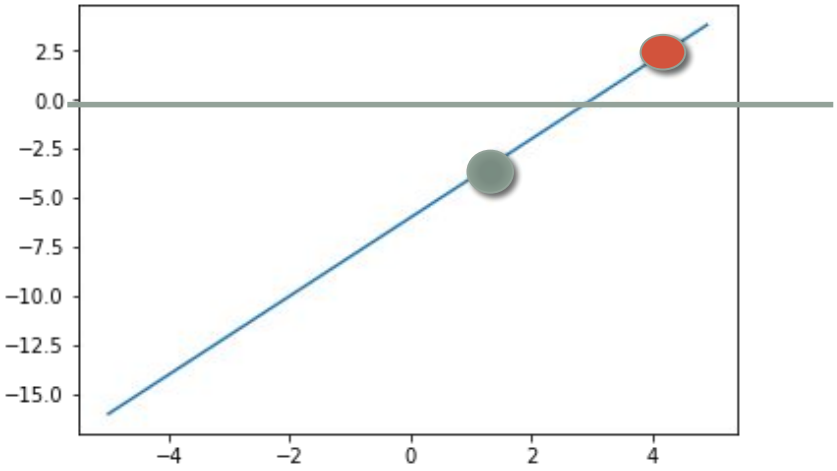
dy/dx

Move along the negative direction of the gradient
The bigger the gradient the bigger step you move

Gradient descent



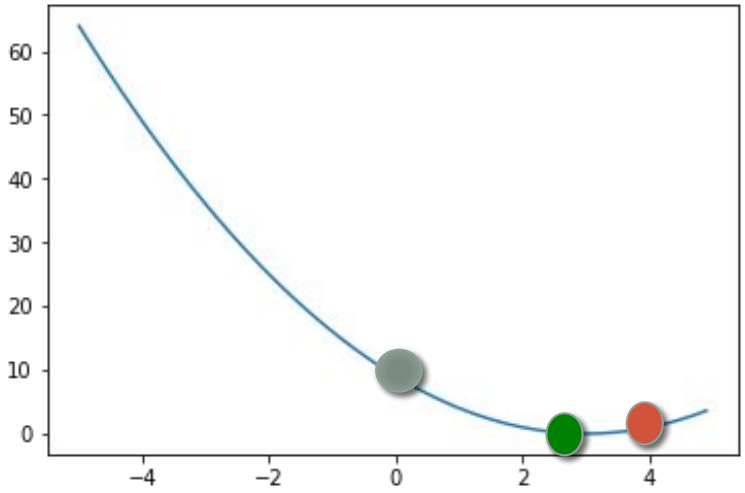
y



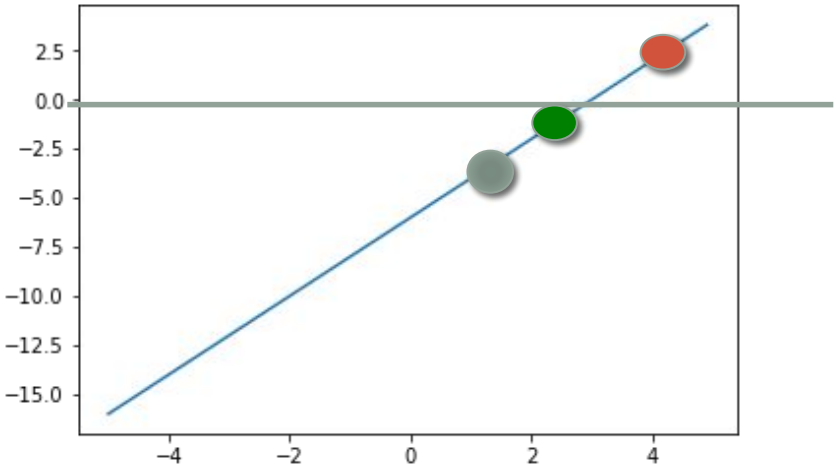
dy/dx

What happens when you overstep?

Gradient descent



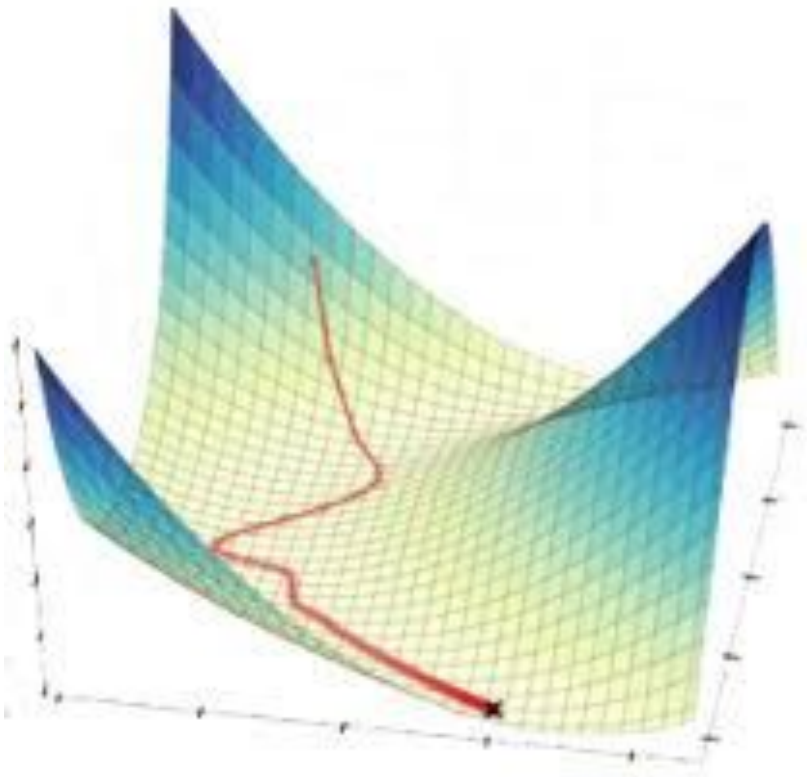
y



dy/dx

If you over step you can move back

Gradient descent in 3d



http://dsdeepdive.blogspot.com/2016/03/optimizations-of-gradient-descent.html

Formal definition

- $y = f(x)$

- Pick a starting point x_0

- Moves along -dy/dx
- $x_{n+1} = x_n - r * dy/dx$
- Repeat till convergence
- r is the learning rate
  - Big r means you might overstep
  - Small r and you need to take more steps

Mini-batch

If we want to minimize the loss

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

One training example gives one loss value -> on gradient

This can be noisy.

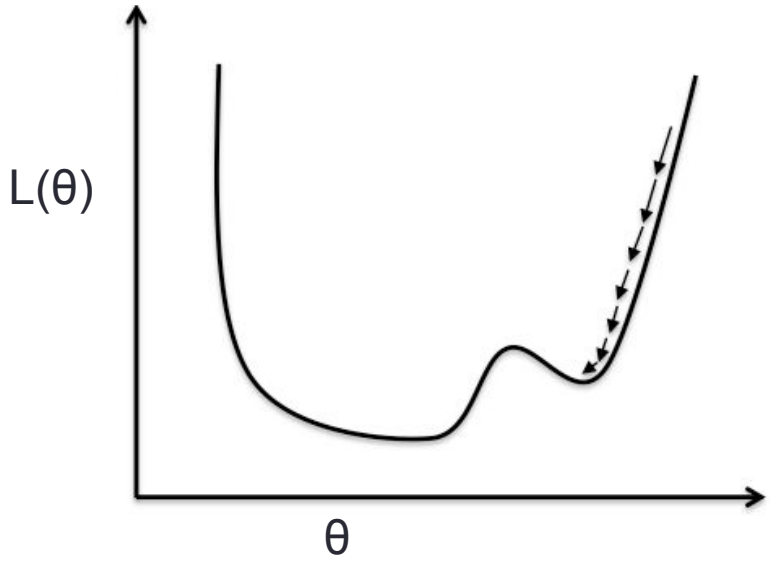
Update using the average gradient from a mini-batch training examples. This is called Stochastic Gradient Descent

Standard mini-batch sizes 16, 32, 64, 128

One entire data pass = one epoch

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

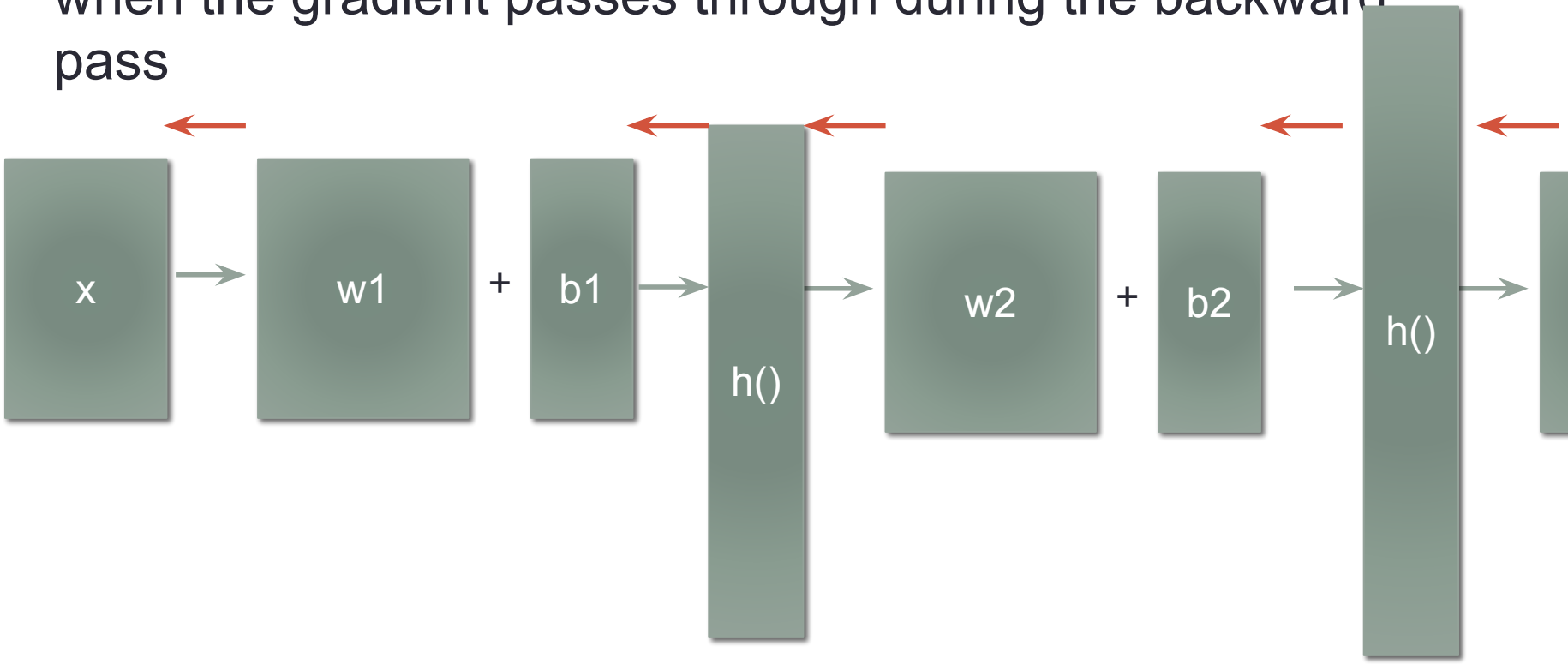
Examples to read

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

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

Backprop and computation graph

- We can also define what happens to a computing graph when the gradient passes through during the backward pass



This lets us to build any neural networks without having to redo all the derivation as long as we define a forward and backward computation for the block.

Initialization

- The starting point of your descent
- Important due to local minimas
- Not as important with large networks AND big data (>100 hours for ASR)
- Now usually initialized randomly (Protip: just use default)
  - One strategy (Xavier initialization)

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

  - For ReLUs (He initialization)

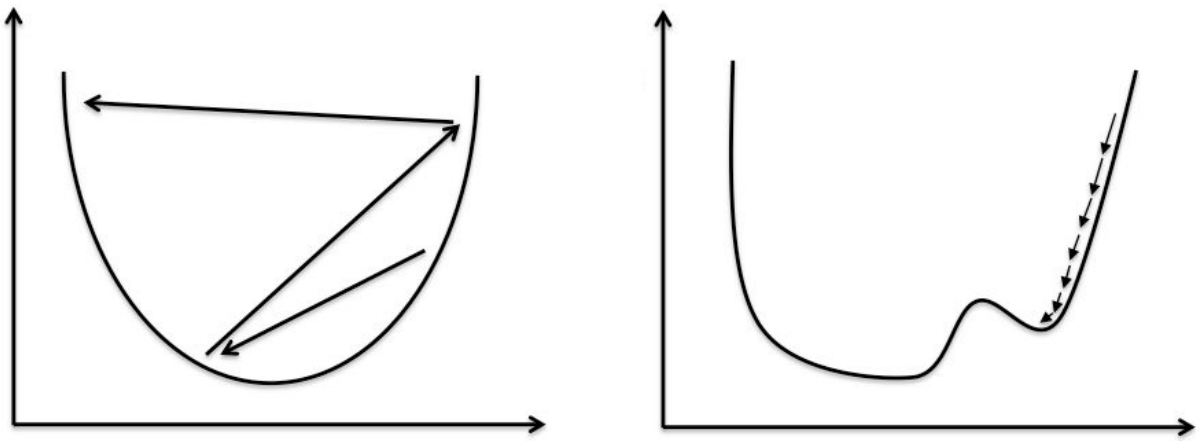
```
w = np.random.randn * sqrt(2.0/FanIn)
```

  - Or use a pre-trained network as initialization

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

Learning rate

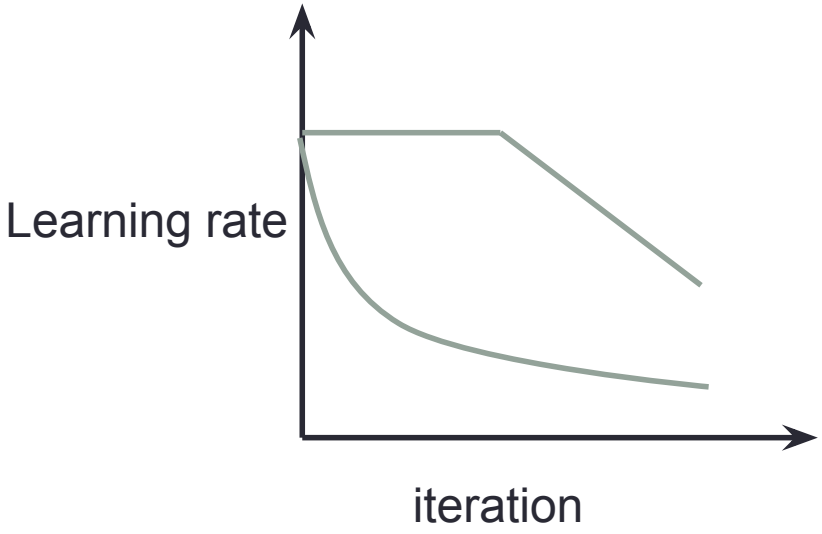
- How fast to go along the gradient direction is controlled by the learning rate
- Too large models diverge
- Too small the model get stuck in local minimas and takes too long to train



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

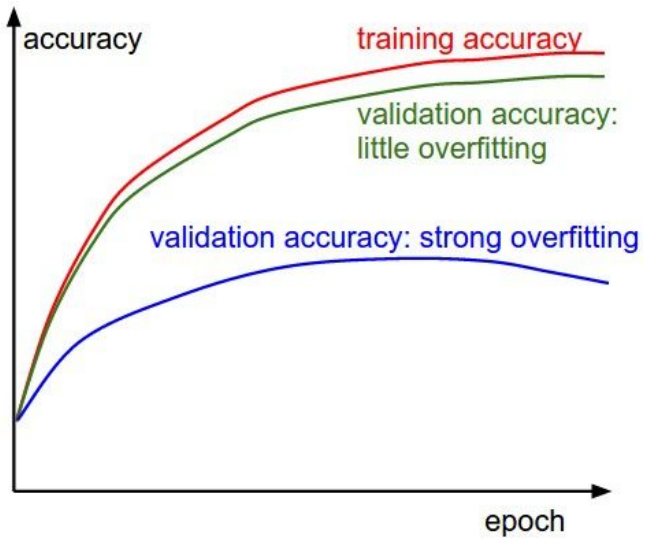
Learning rate scheduling

- 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

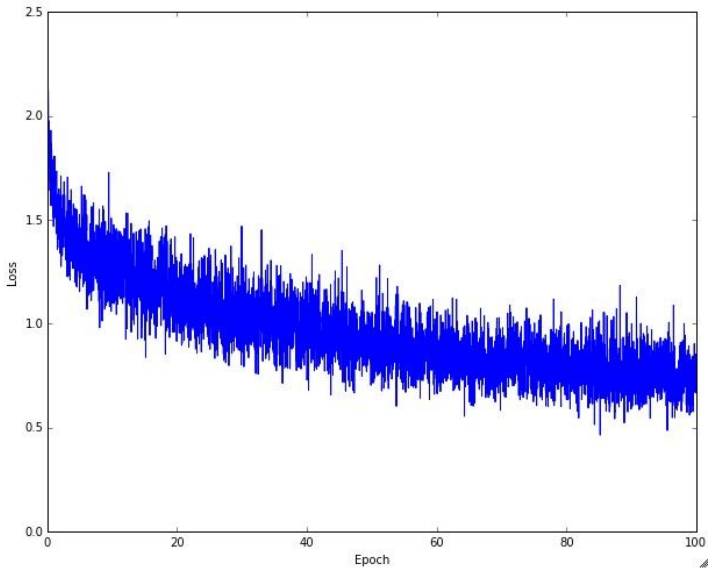
- You can keep doing back propagation forever!
- The training loss will always go down
- But it overfits
- Need to monitor performance on a held out set
- Stop or decrease learning rate when overfit happens



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



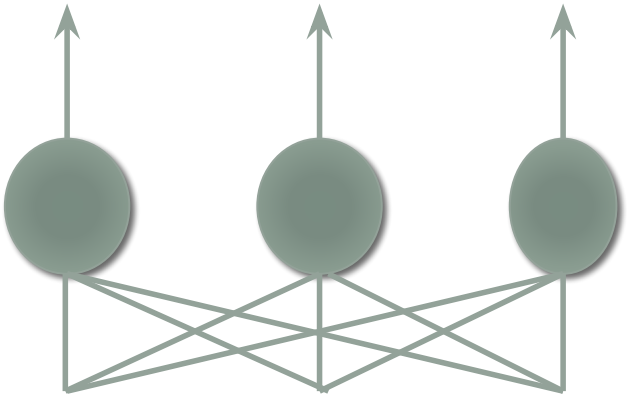
http://cs231n.github.io/neural-networks-3/

Reducing overfitting - dropout

- A 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)
- Now a standard technique

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

Dropout visualized

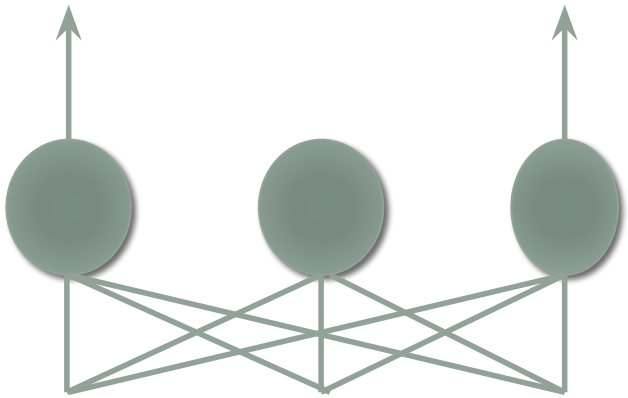


Model

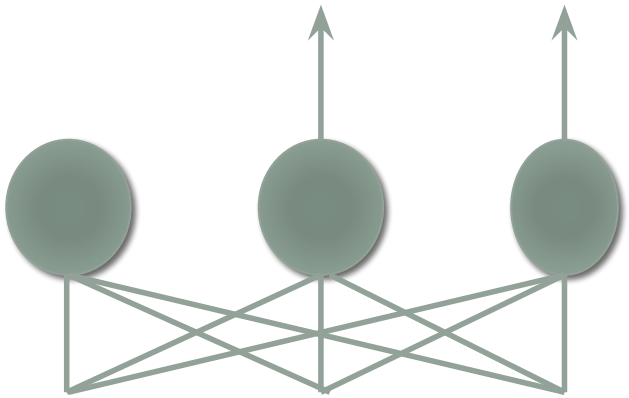
Dropout training time

Drop out rate 0.33

Randomly removed outputs
for each training sample



x1

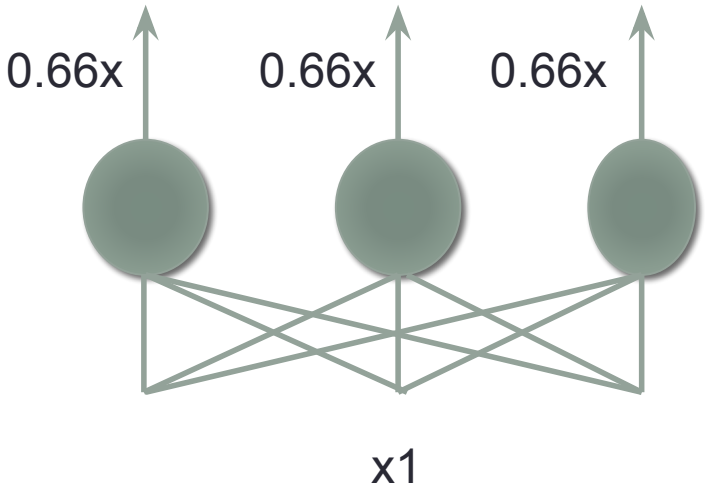


x2

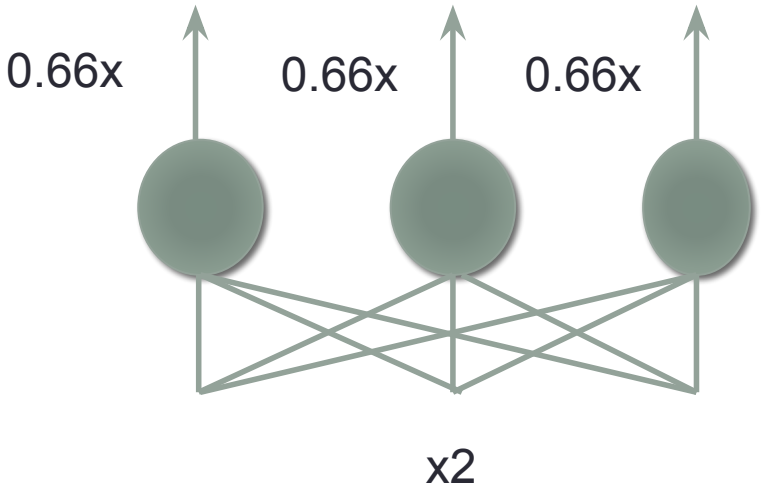
Dropout test time

Dropout rate 0.33

Scale outputs so the output contribution is around the same

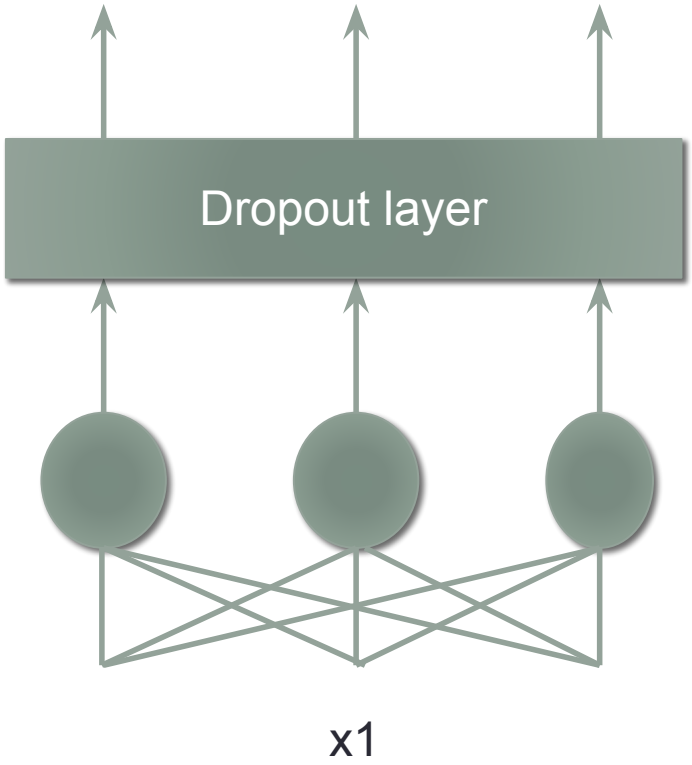


x1



x2

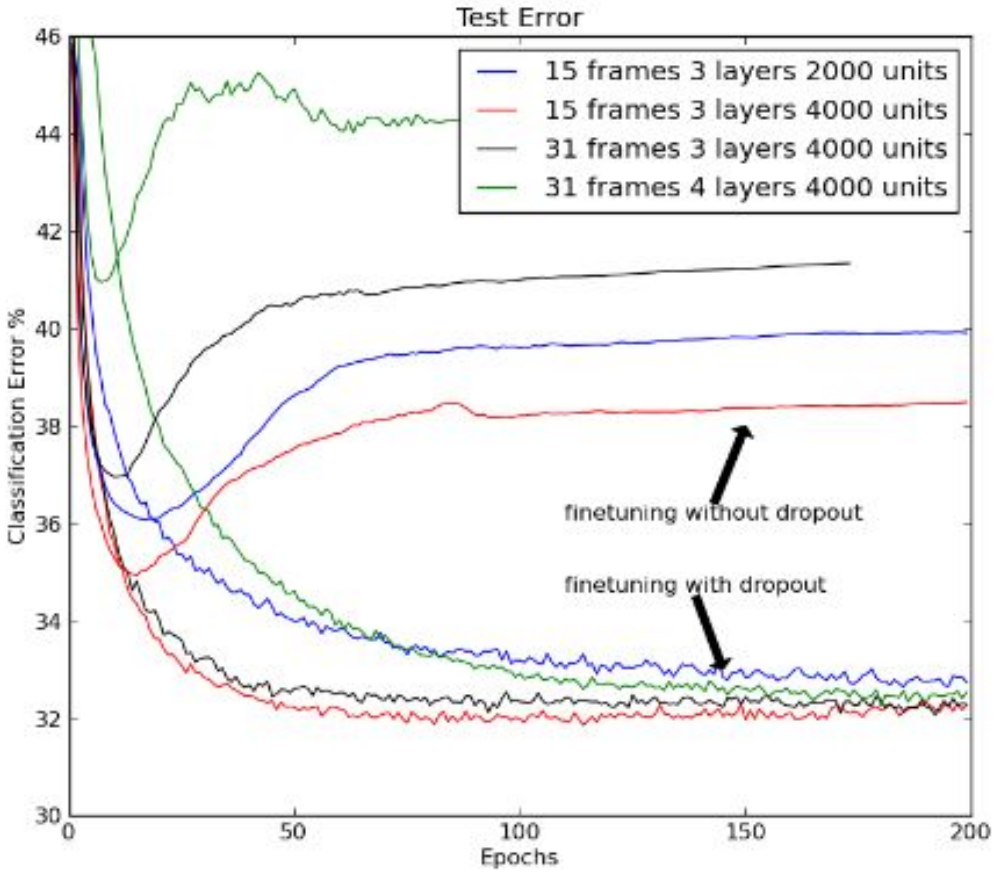
Dropout implementation



Dropout layer
Just another layer that drops inputs

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
  - SGD and backprop
  - Learning rate
  - Overfitting
- 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

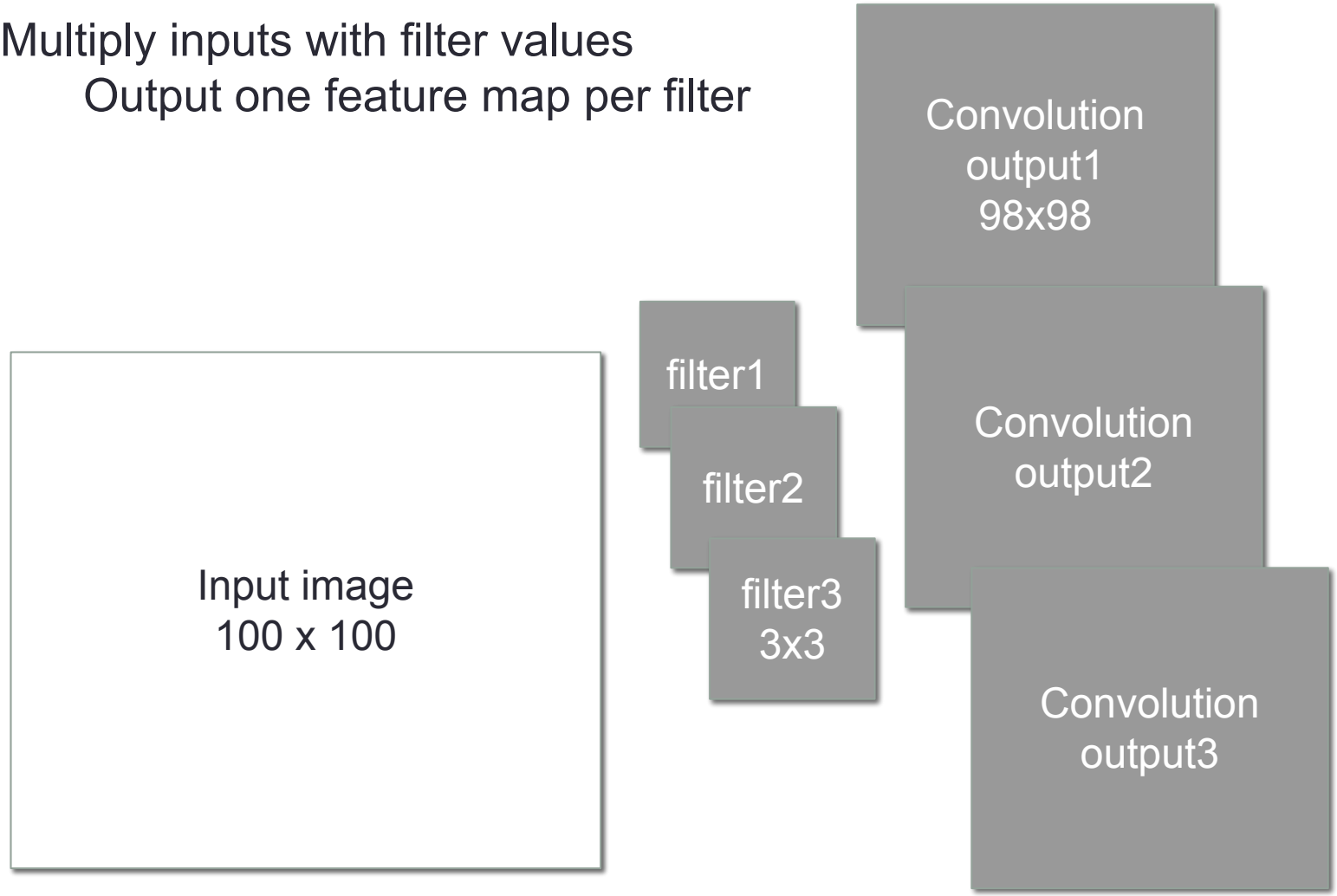
- Can we use the same parameters to learn that a cat exists regardless of location?

- 2 parts: convolutional layer and pooling layer

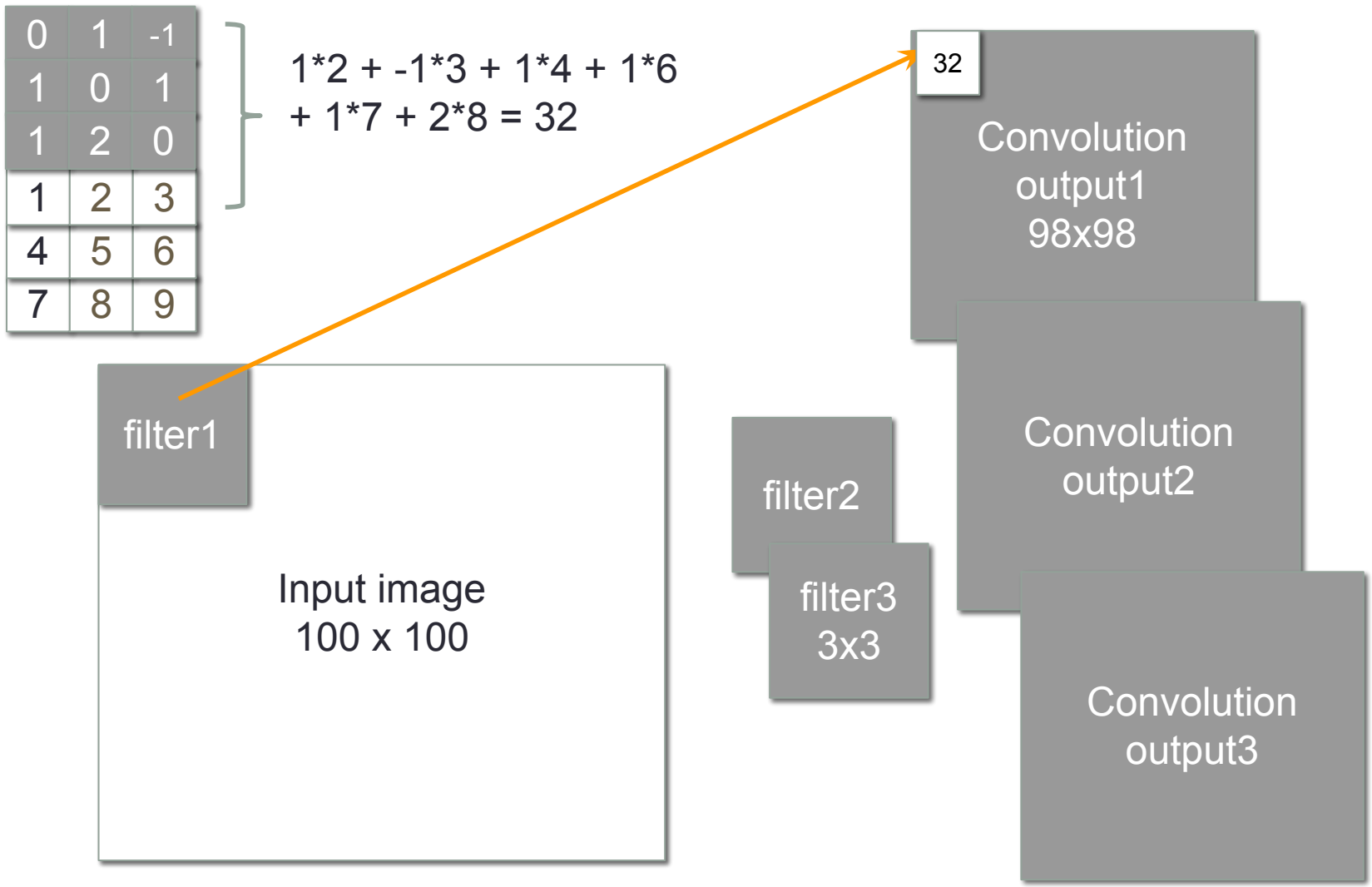
Convolutional filters

Multiply inputs with filter values

Output one feature map per filter

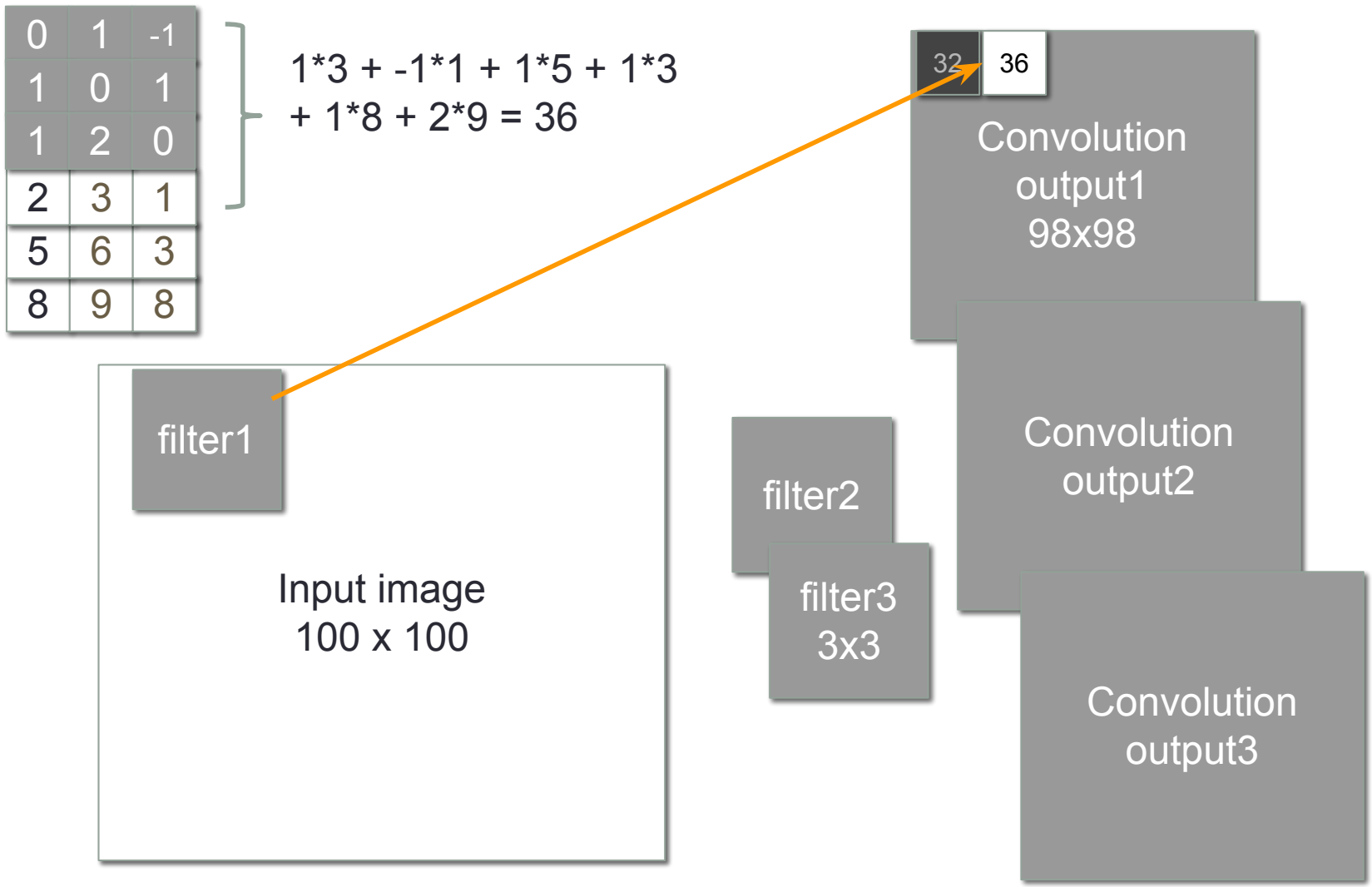


Convolutional filters

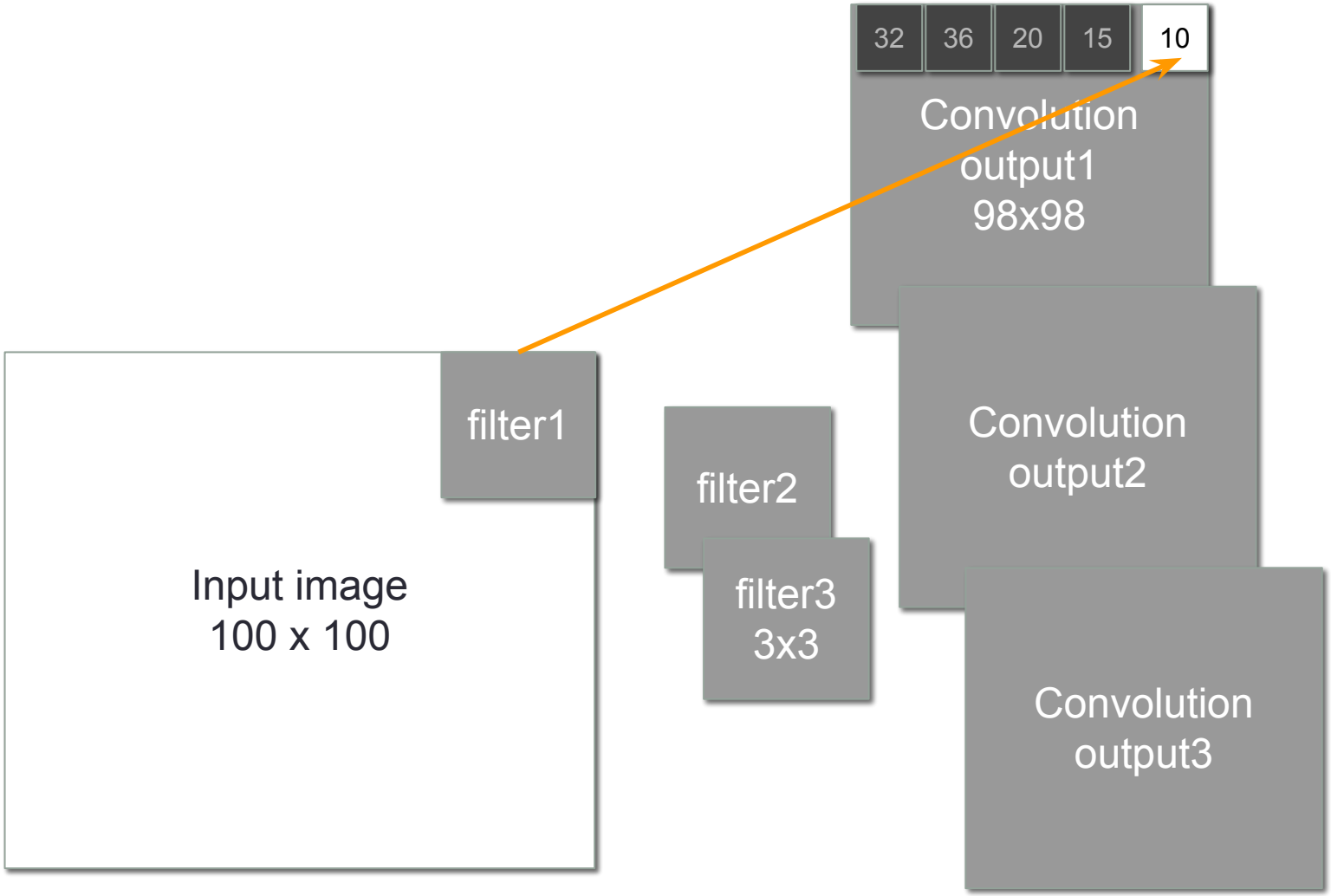


Convolutional filters

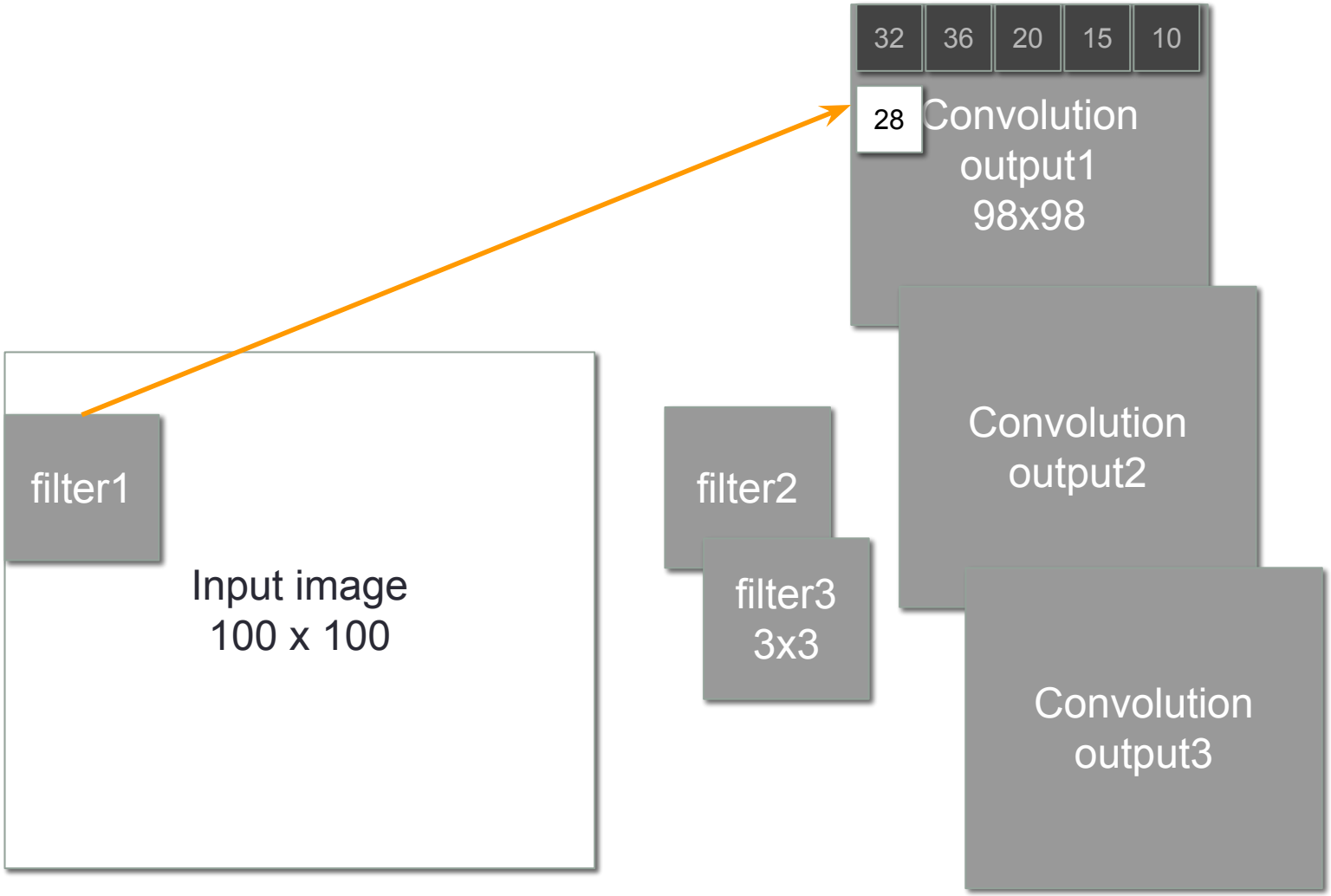
Stride of 1



Convolutional filters

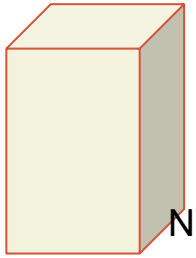


Convolutional filters



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

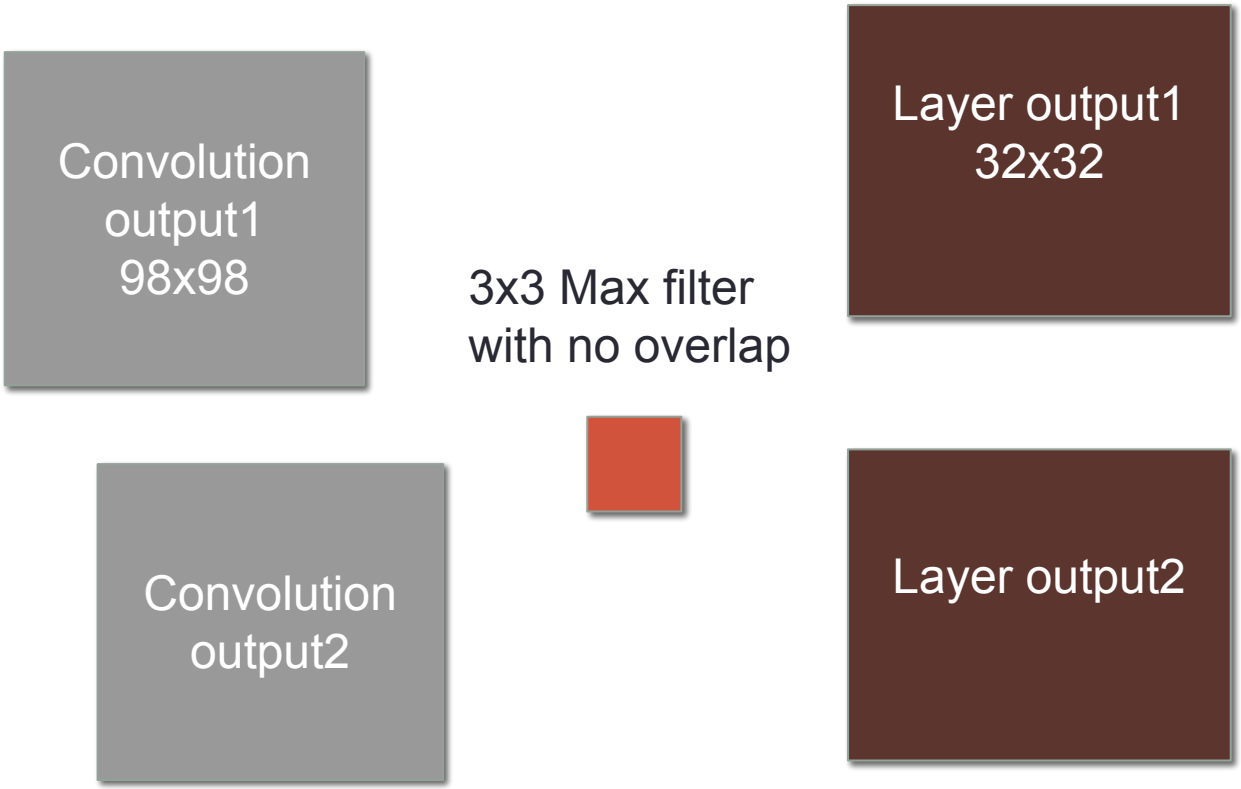
Input image
100 x 100

Convolution
output2

Convolution
output3

Pooling/subsampling

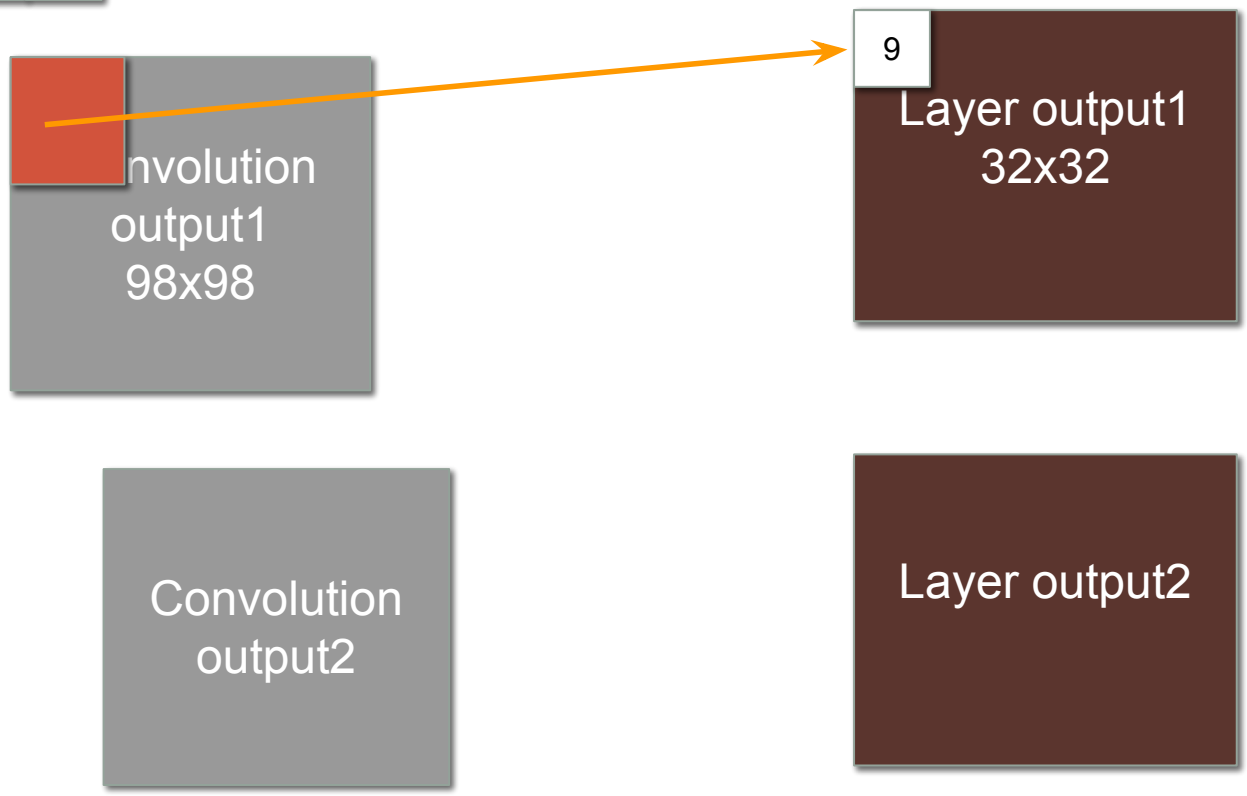
Reduce dimension of the feature maps



Pooling/subsampling

1	2	3
4	5	6
7	8	9

Max = 9

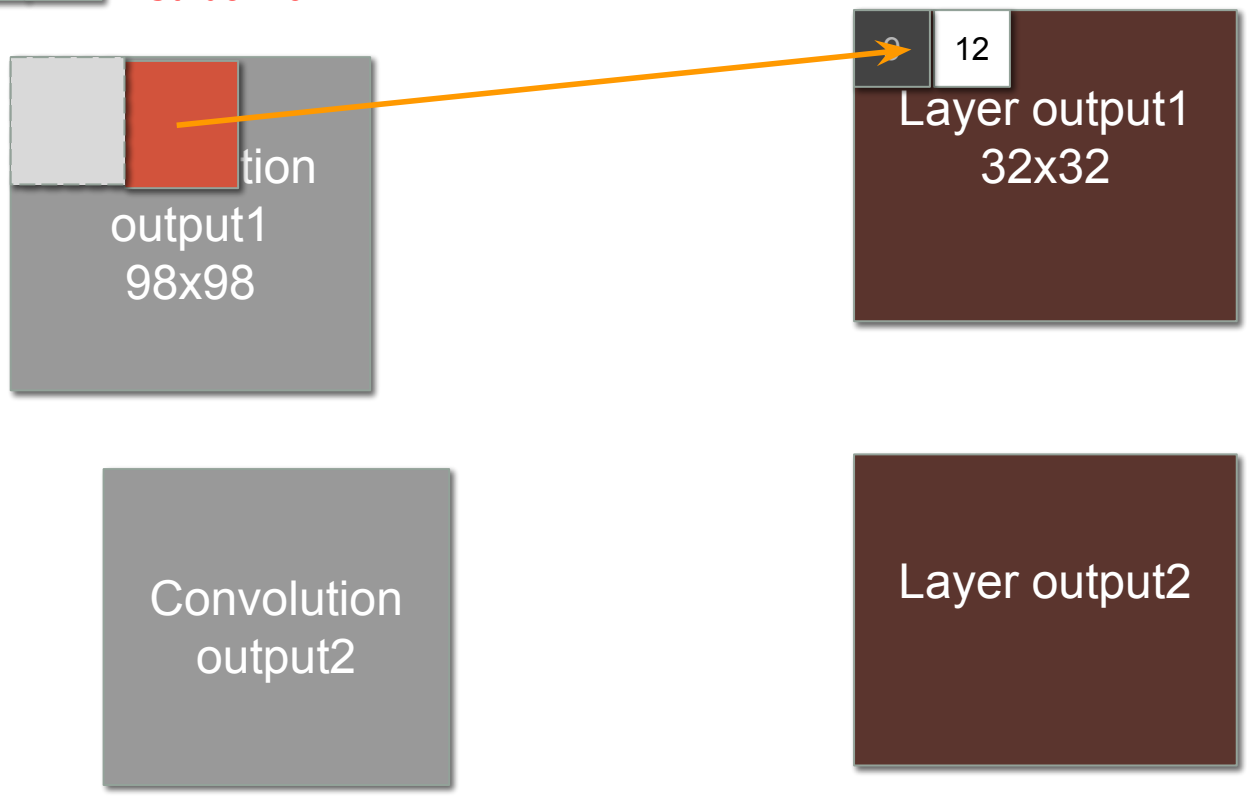


Pooling/subsampling

5	2	1
5	7	1
9	5	12

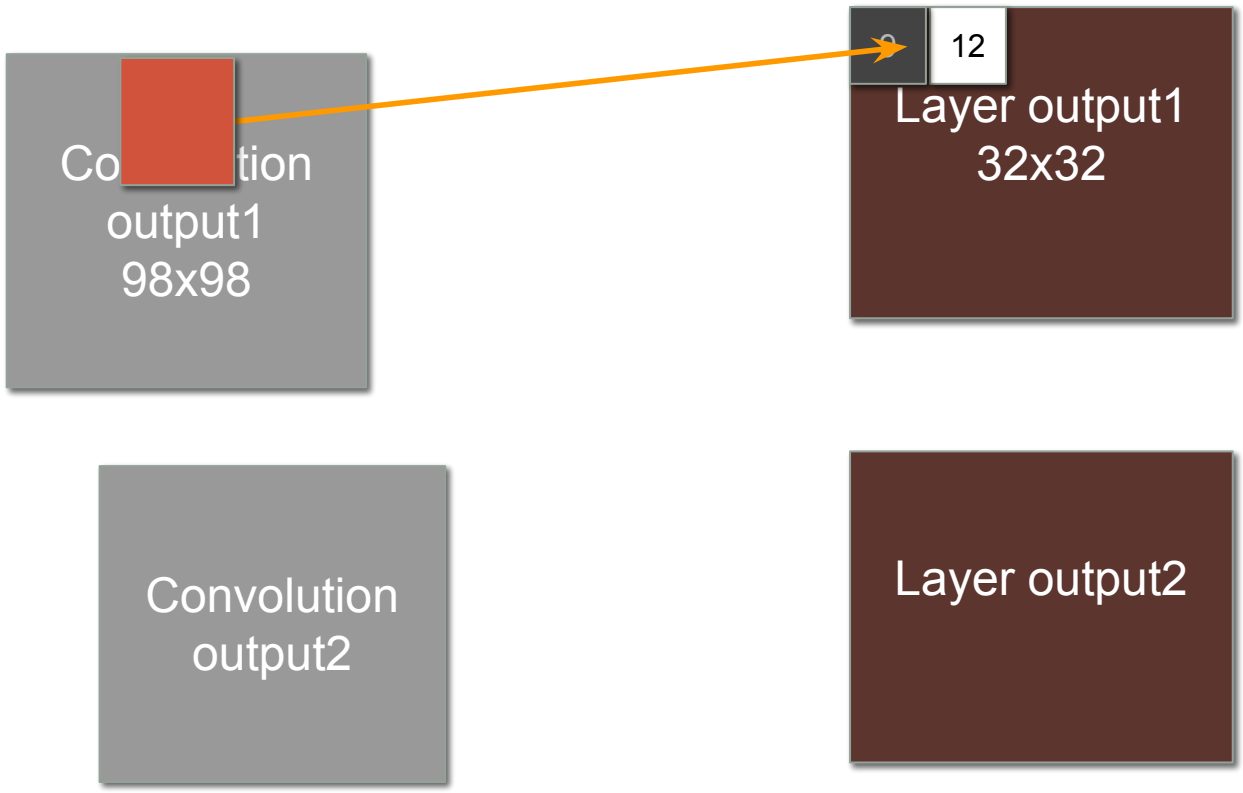
Max = 12

Stride = 3



Pooling/subsampling

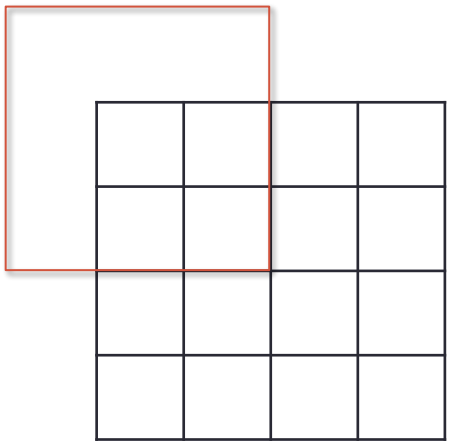
Can use other functions besides max
Example, average



Convolution puzzle

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

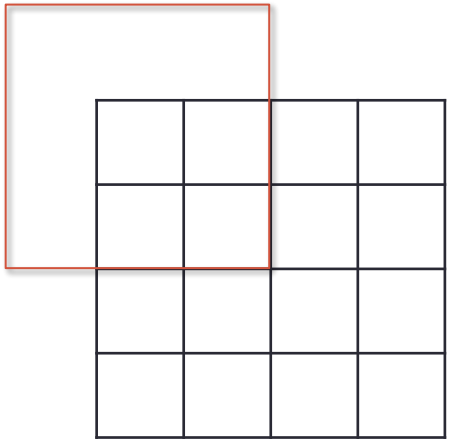
What is the output size?

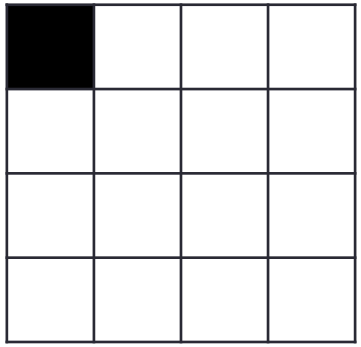


Convolution puzzle

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

What is the output size?

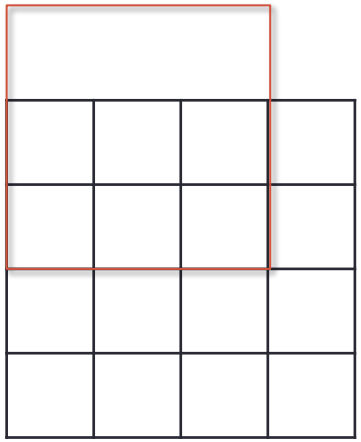


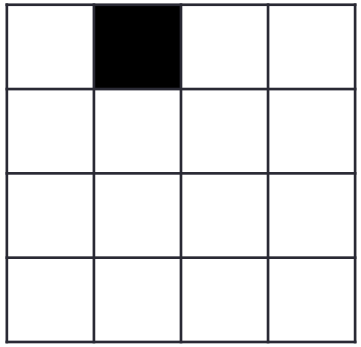


Convolution puzzle

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

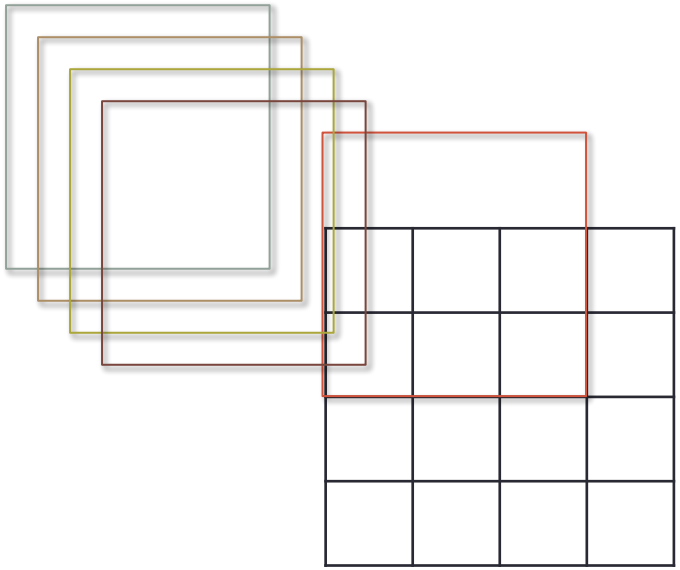
What is the output size?

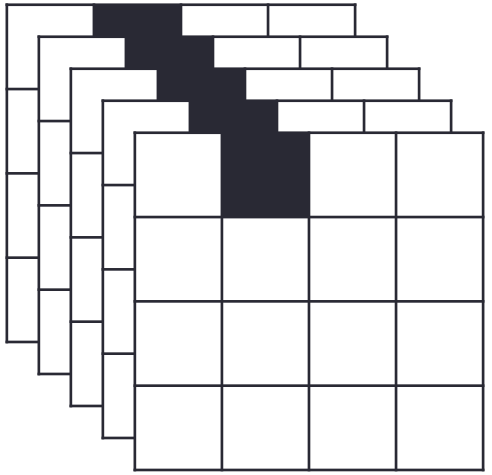




Convolution puzzle

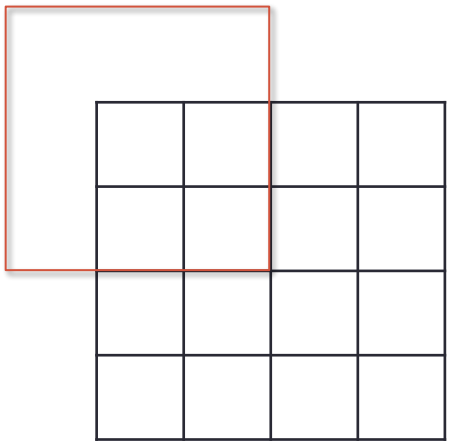
5 filters 3x3 filter pad, stride 1, pad 1





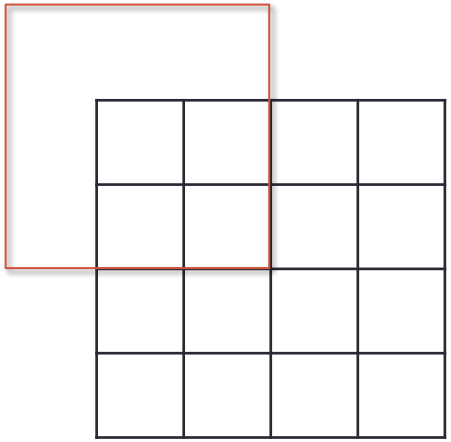
Convolution puzzle

3x3 filter pad, stride 2, pad 1



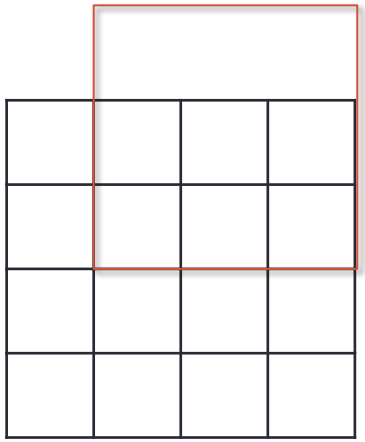
Convolution puzzle

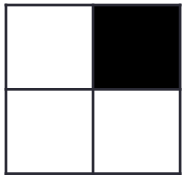
3x3 filter pad, stride 2, pad 1



Convolution puzzle

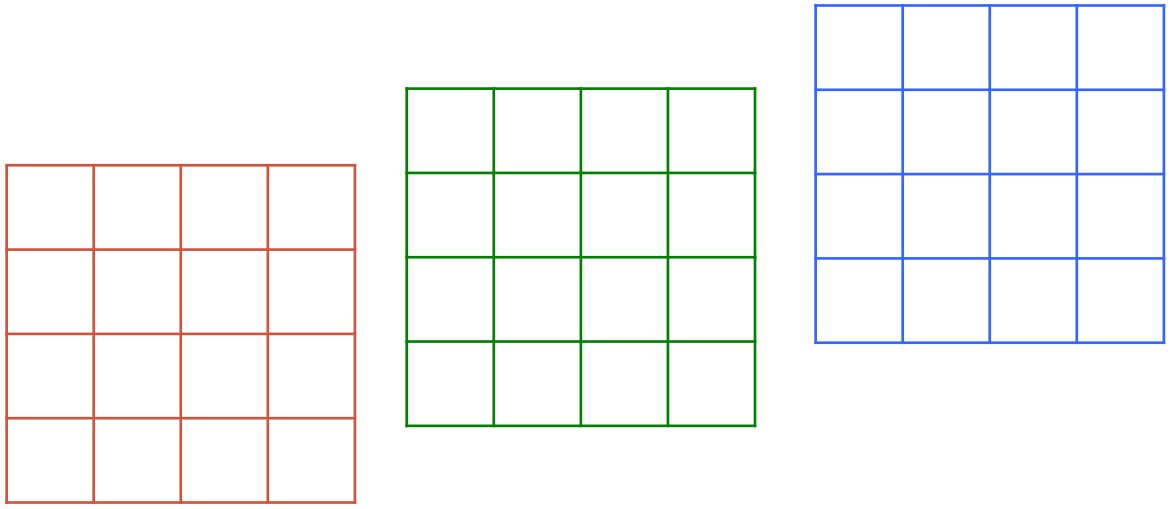
3x3 filter pad, stride 2, pad 1





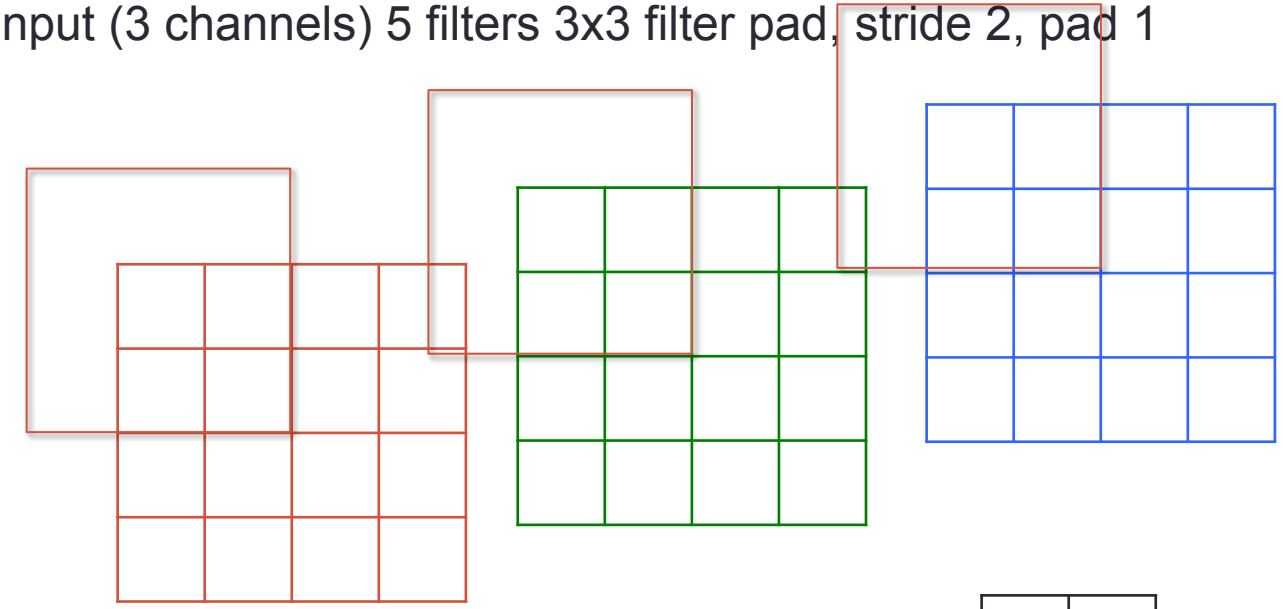
Convolution puzzle

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

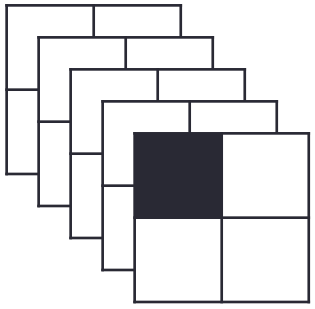


Convolution puzzle

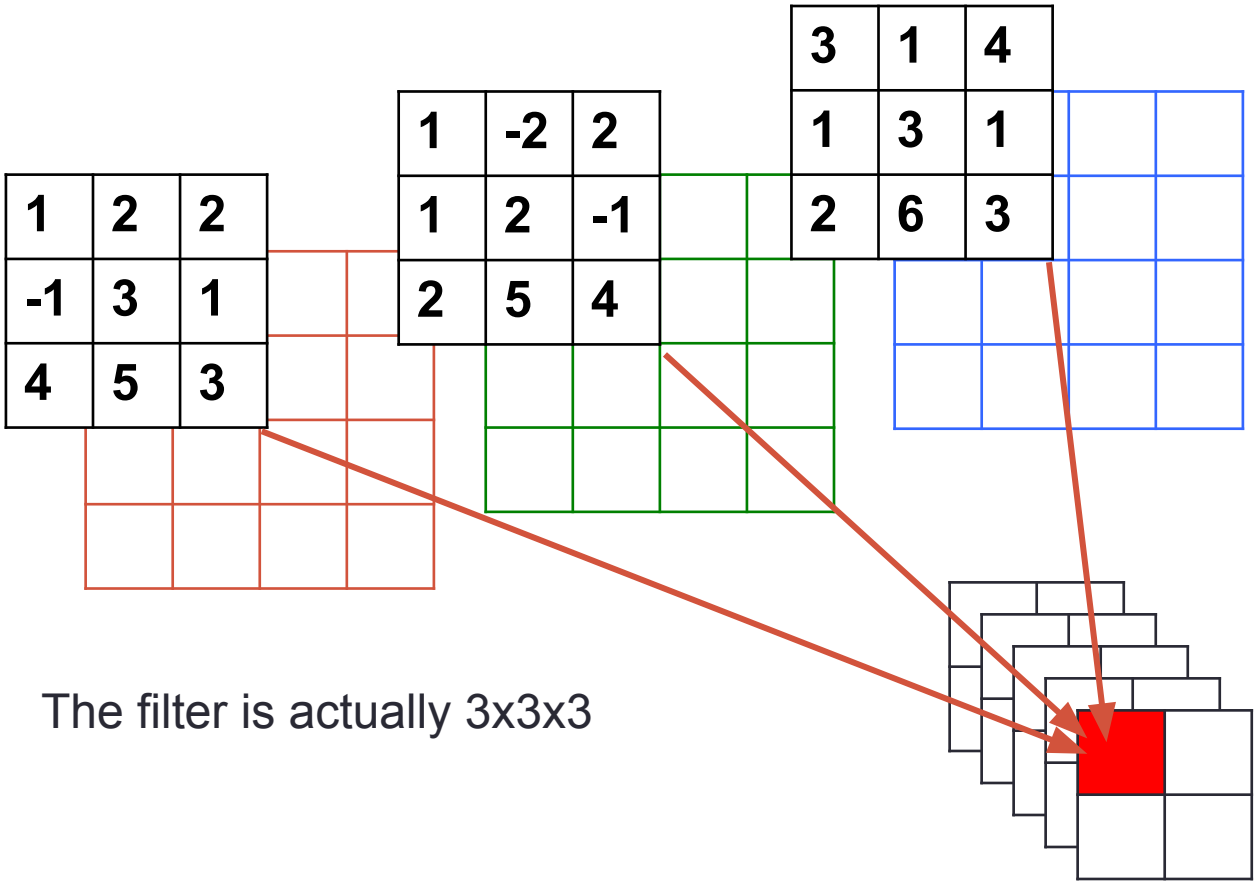
RGB input (3 channels) 5 filters 3x3 filter pad, stride 2, pad 1



The filter is actually 3x3x3



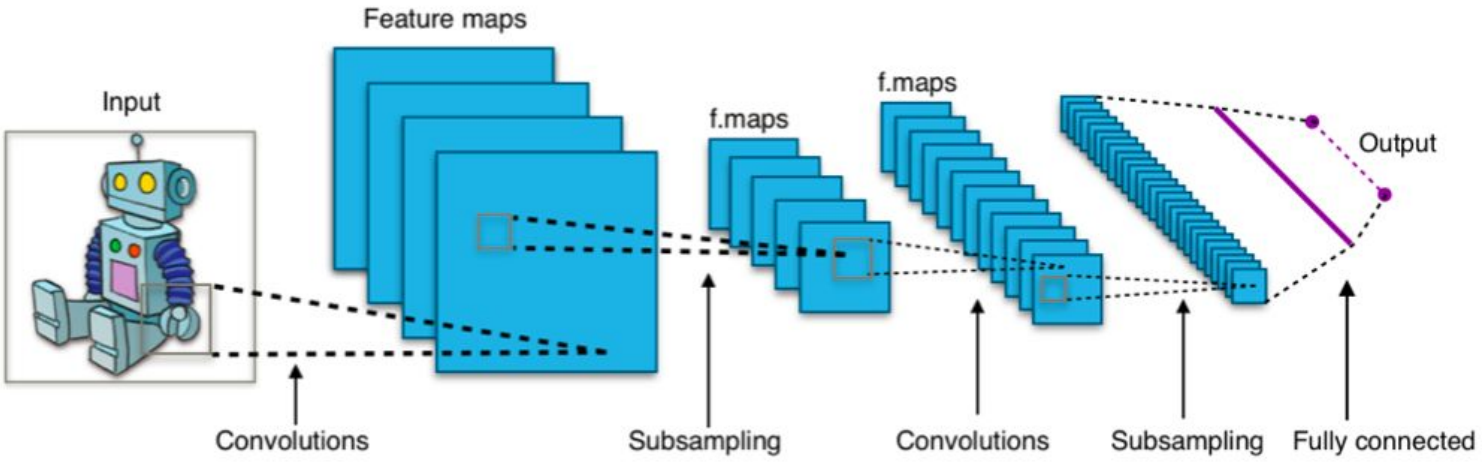
Convolution puzzle



The filter is actually 3x3x3

CNN overview

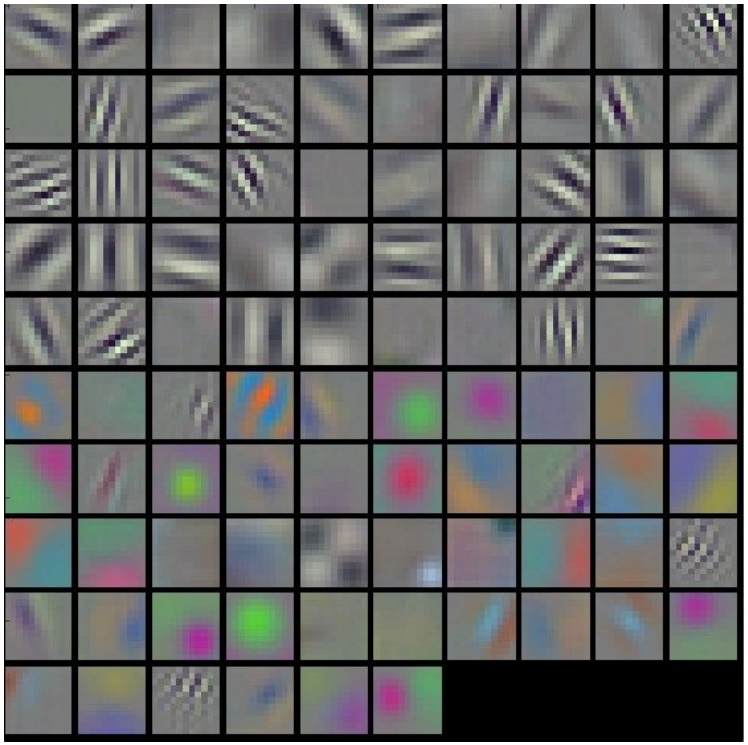
- Filter size, number of filters, filter shifts, and pooling rate are all parameters
- Usually followed by a fully connected network at the end
  - CNN is good at learning low level features
  - DNN combines the features into high level features and classify



https://en.wikipedia.org/wiki/Convolutional_neural_network#/media/File:Typical_cnn.png

Visualizing convolutional layers

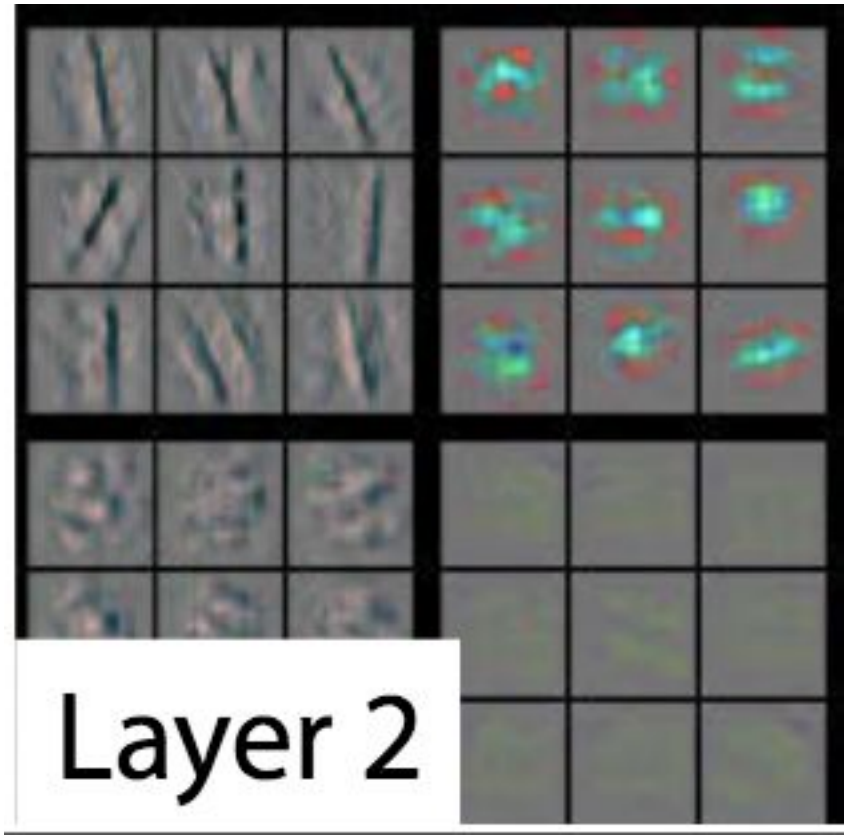
- We can visualize the weights of the filter



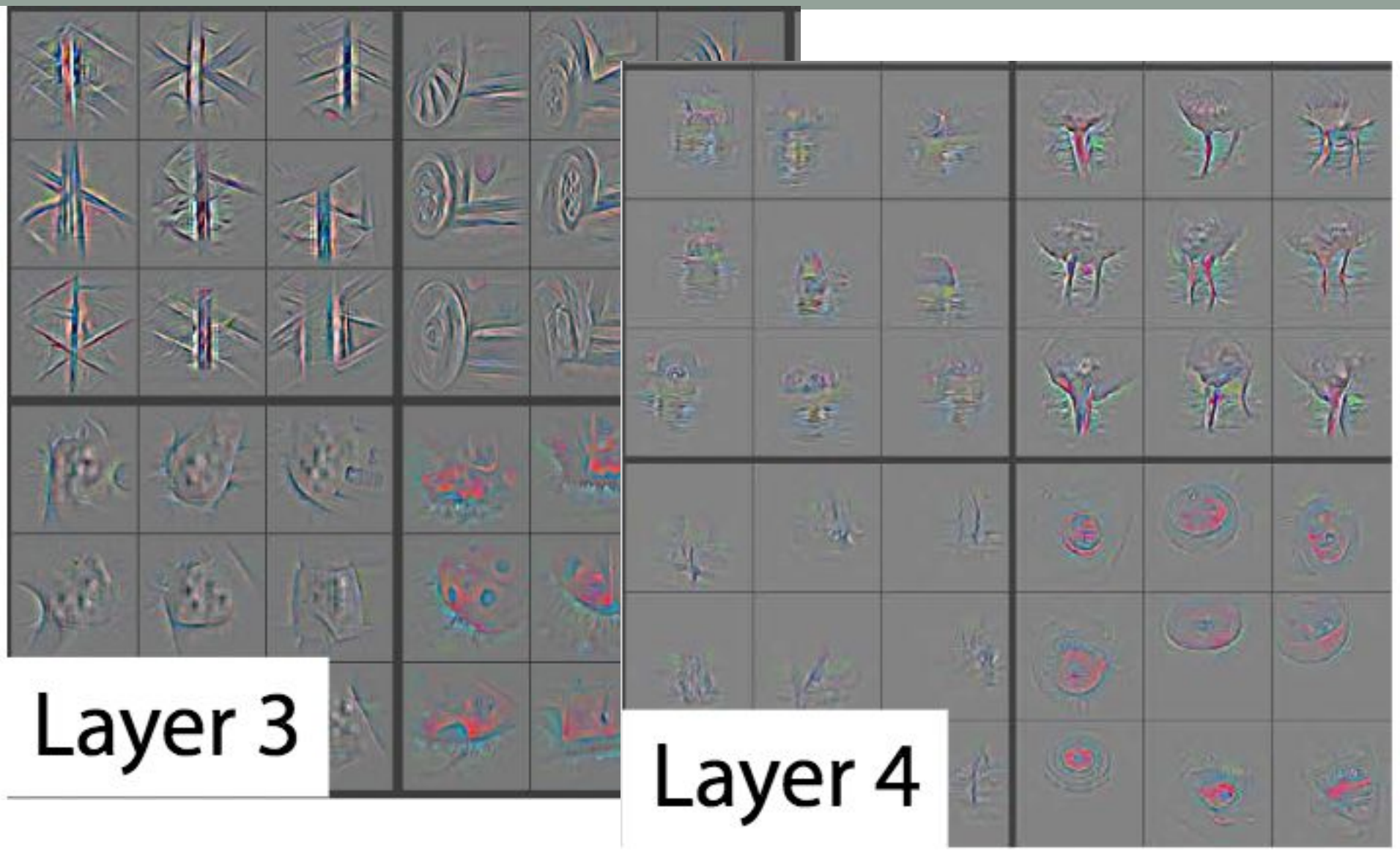
http://cs231n.github.io/understanding-cnn/

Higher layer captures higher-level concepts

Layer 1



https://arxiv.org/pdf/1311.2901.pdf


Layer 3
Layer 4

CNN

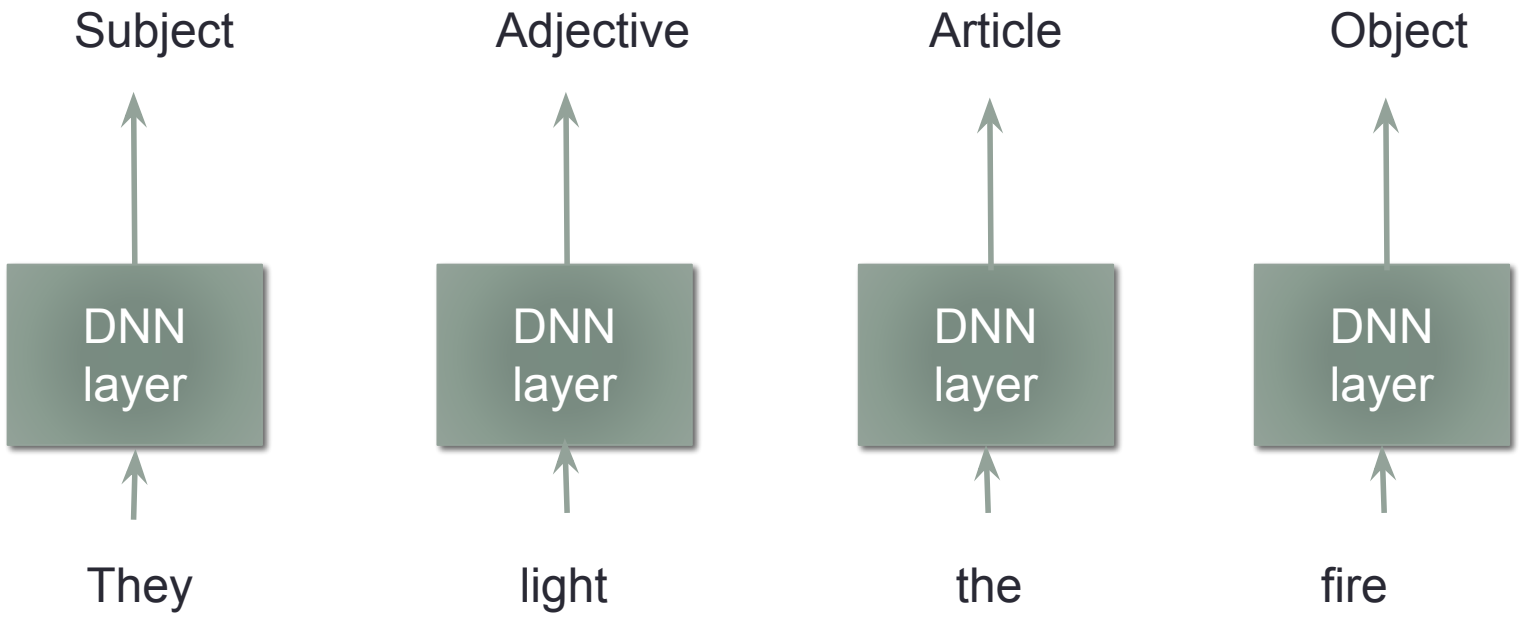
- Convolutional layer
- Subsampling

- Sharing of parameters in space

- Sharing of parameters in time?

Recurrent neural network (RNN)

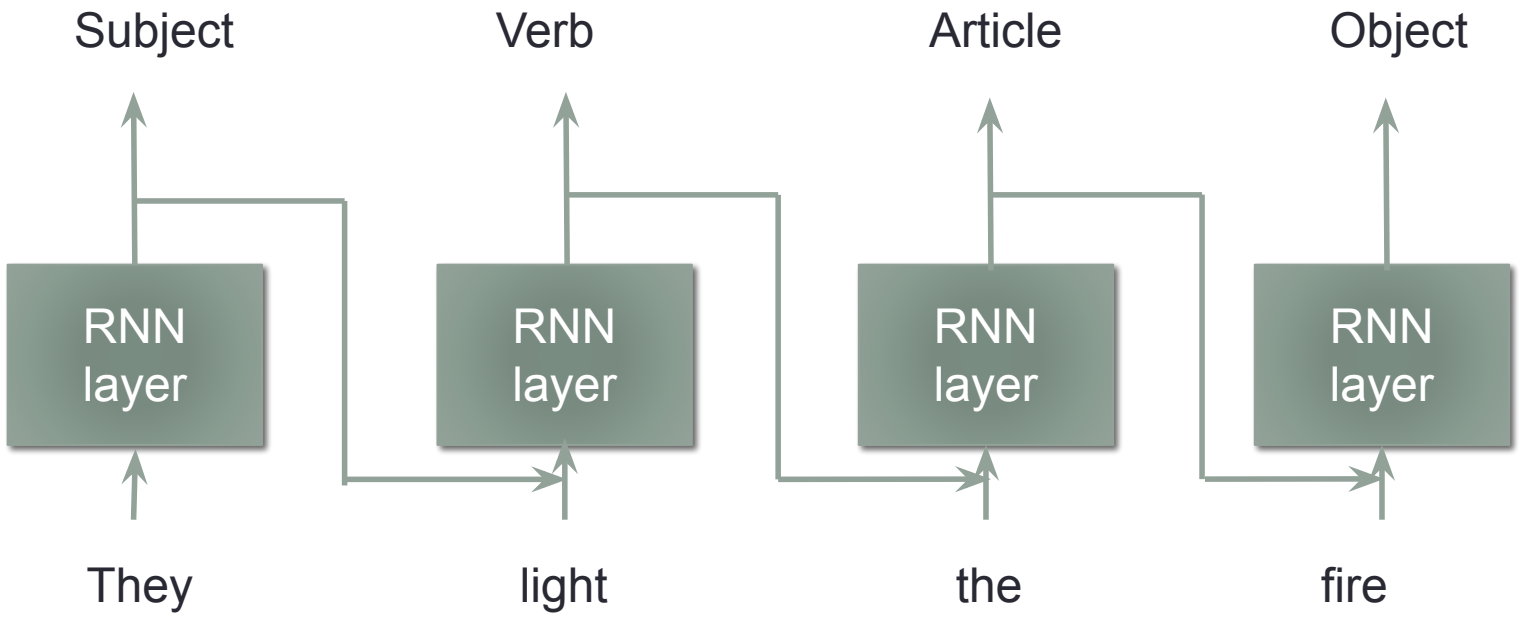
- DNN framework



Problem1: need a way to remember the past

Recurrent neural network (RNN)

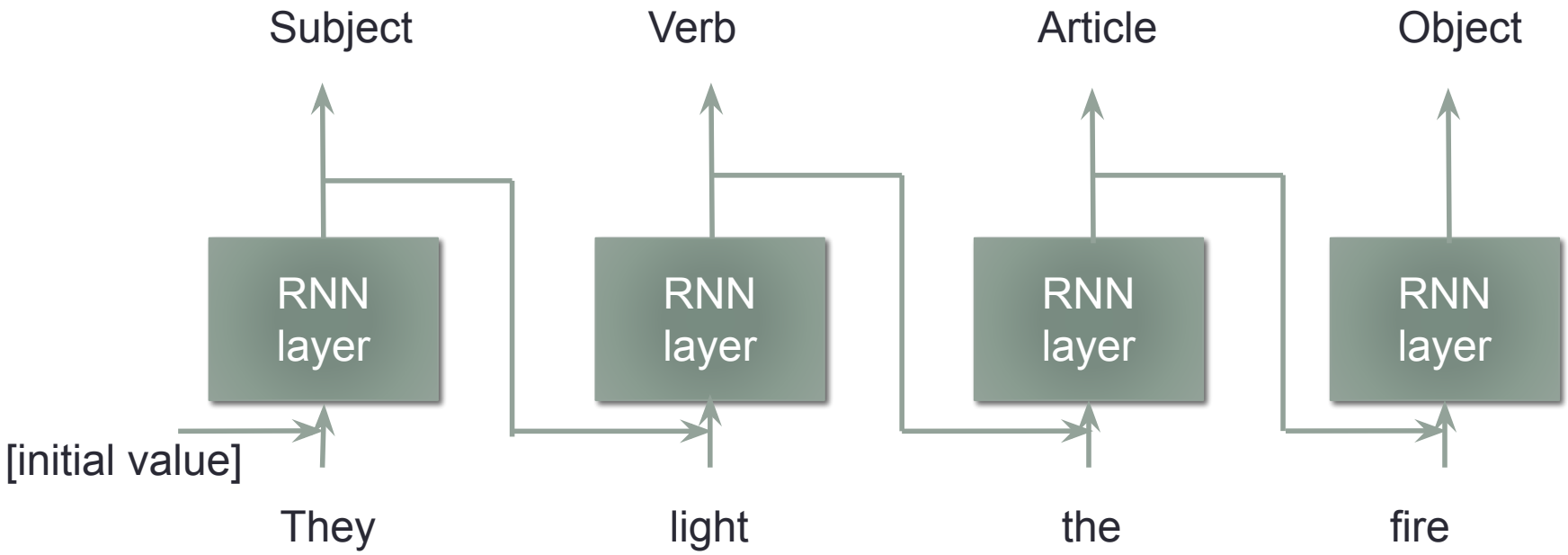
- RNN framework



Output of the layer encodes something meaningful about the past

Recurrent neural network (RNN)

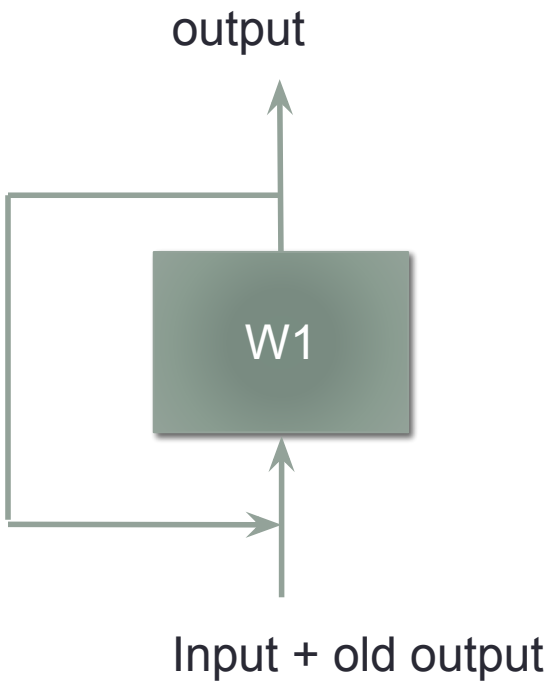
- RNN framework



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

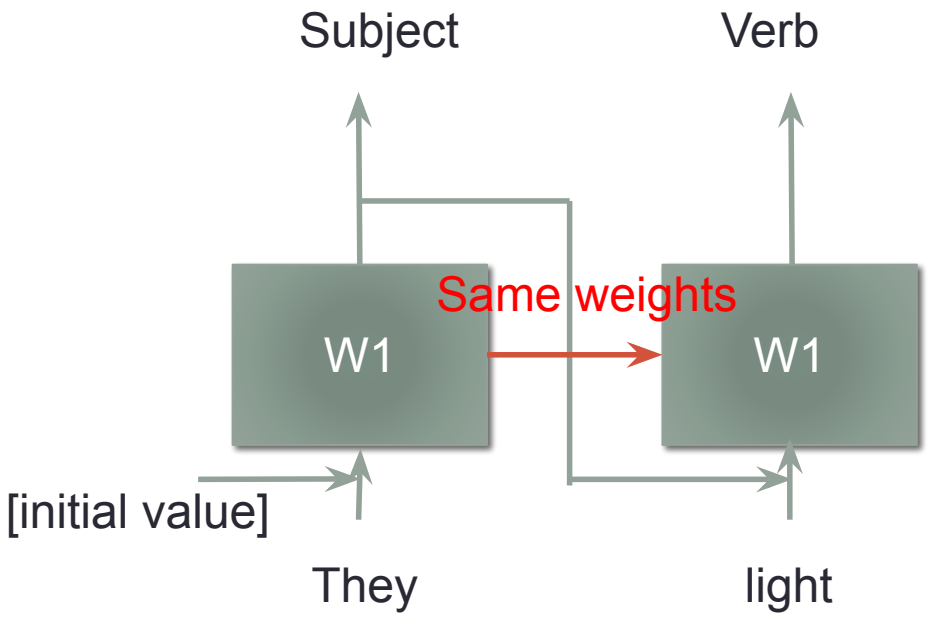
Recurrent neural network (RNN)

- Unrolling of a recurrent layer.



Recurrent neural network (RNN)

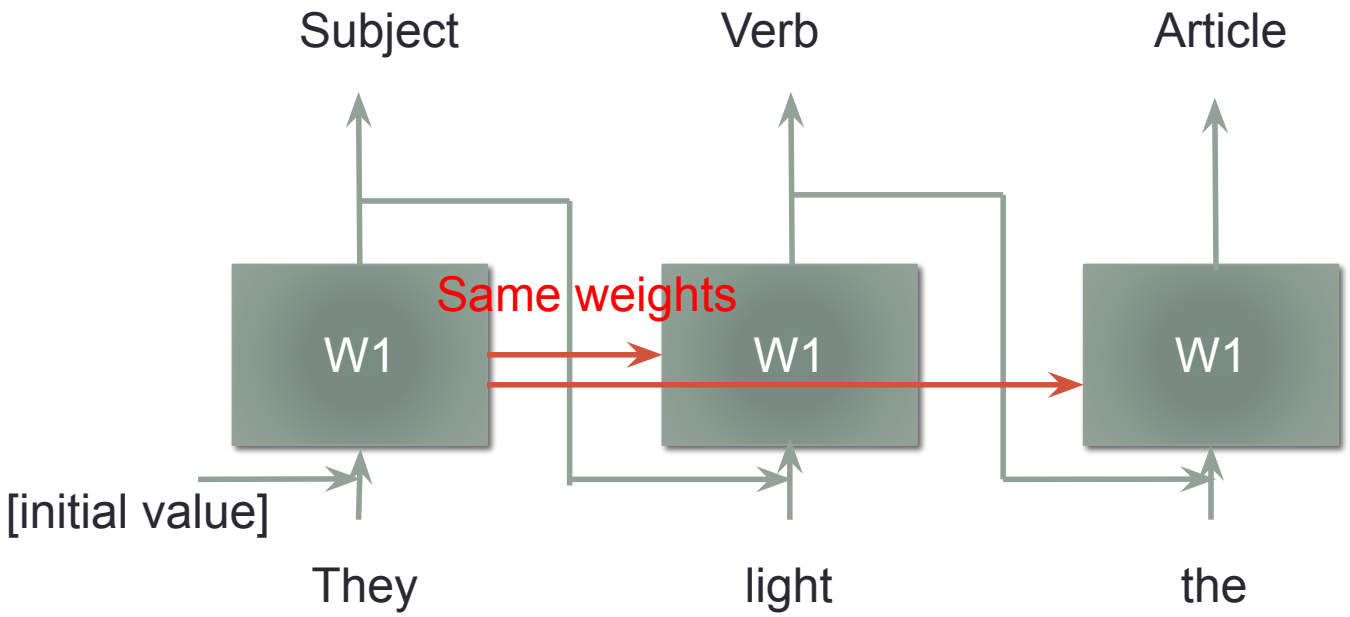
- Unrolling of a recurrent layer.



Parameter sharing

Recurrent neural network (RNN)

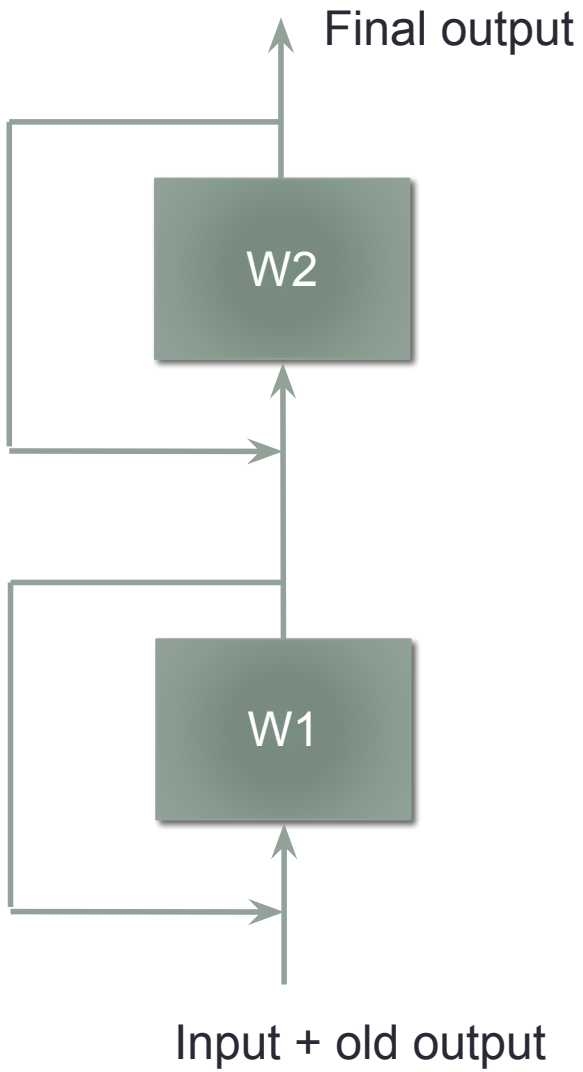
- Unrolling of a recurrent layer.



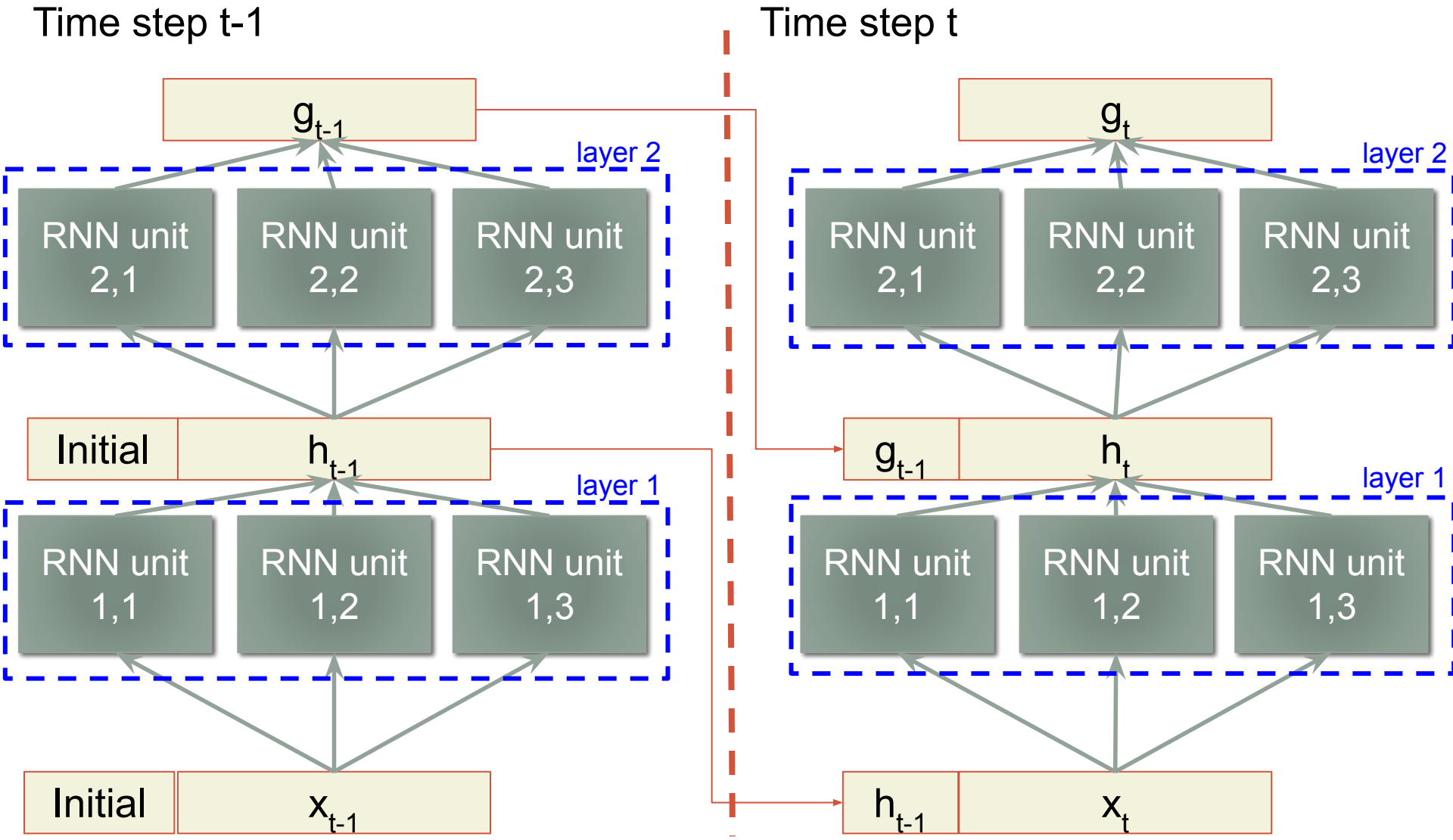
Parameter sharing

Recurrent neural network (RNN)

- Stacks of recurrent layer

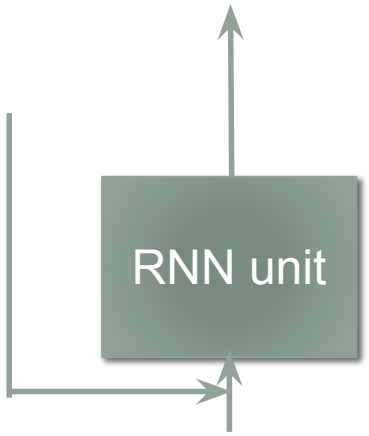


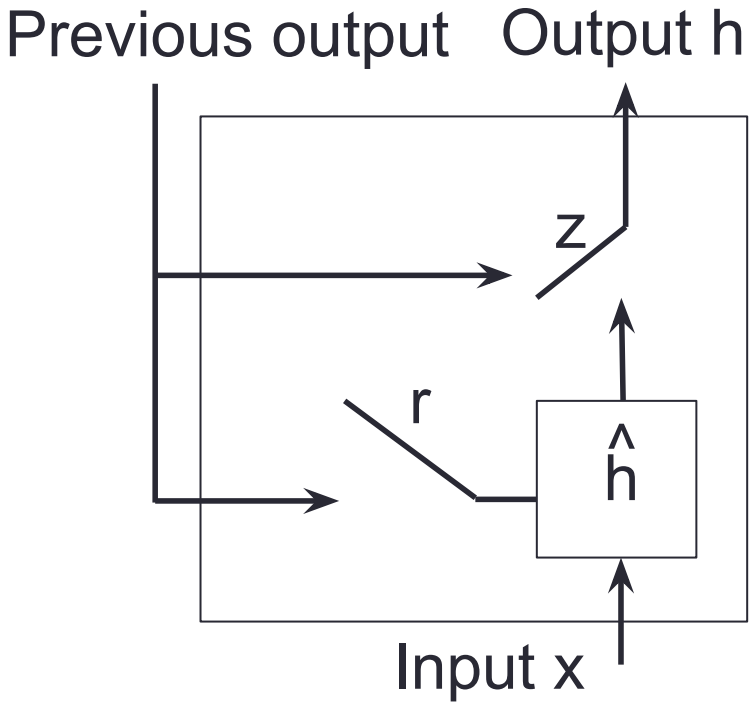
RNN layers (expanded in time)



Gated Recurrent Unit (GRU)

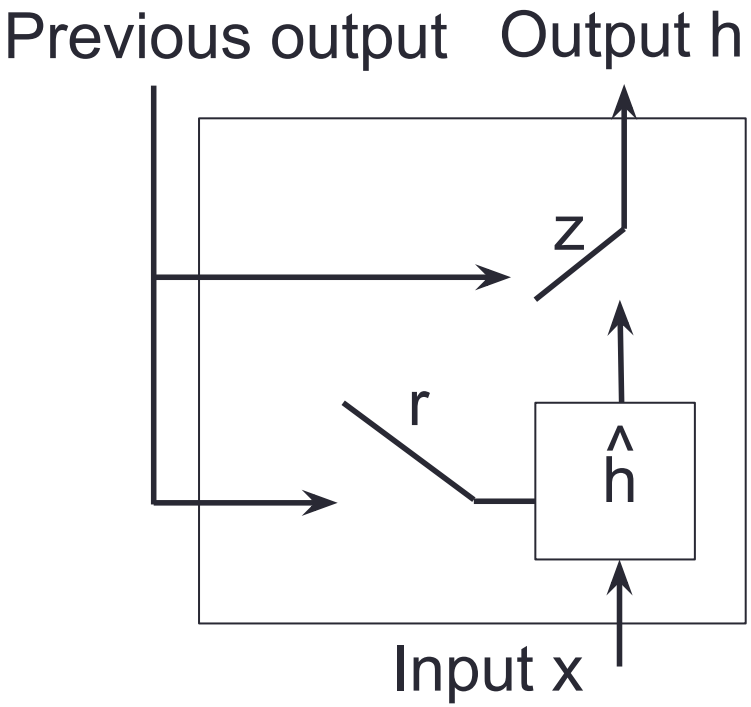
- Forms a Gated Recurrent Neural Networks (GRNN)
- Add gates that can choose to reset (r) or update (z)

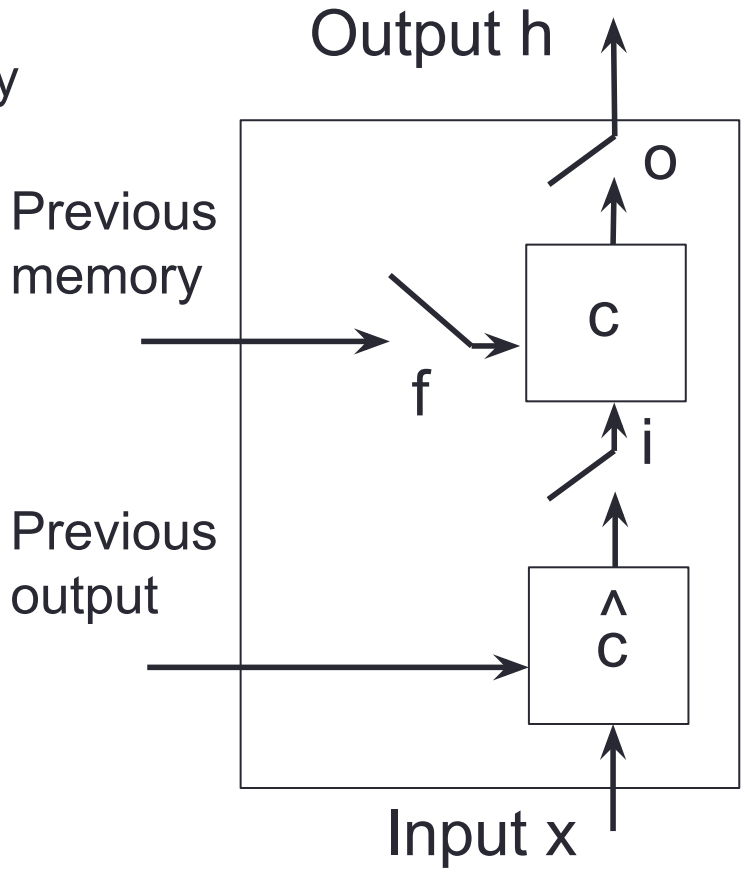




Long Short-Term Memory (LSTM)

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



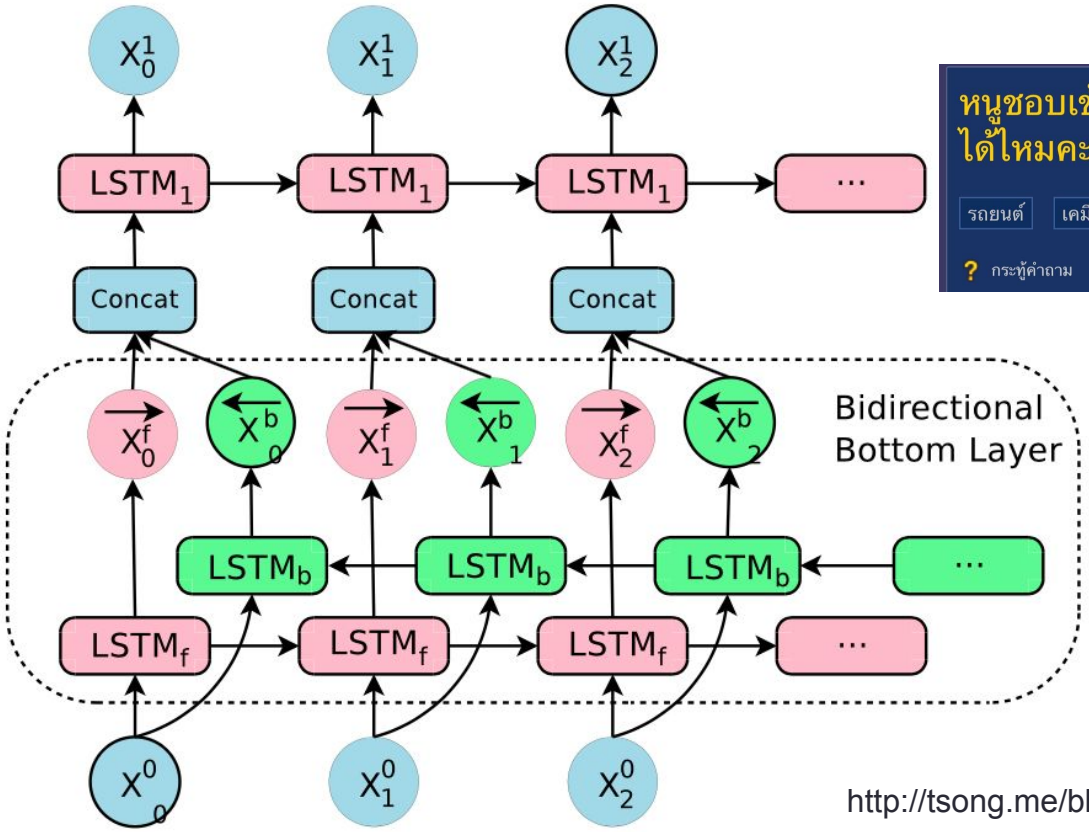


GRU vs LSTM

- 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/

LSTM remembers meaningful things

Cell sensitive to position in line:

```
The sole importance of the crossing of the Berezina lies in the fact
that it plainly and indubitably proved the fallacy of all the plans for
cutting off the enemy's retreat and the soundness of the only possible
line of action--the one Kutuzov and the general mass of the army
demanded--namely, simply to follow the enemy up. The French crowd fled
at a continually increasing speed and all its energy was directed to
reaching its goal. It fled like a wounded animal and it was impossible
to block its path. This was shown not so much by the arrangements it
made for crossing as by what took place at the bridges. When the bridges
broke down, unarmed soldiers, people from Moscow and women with children
who were with the French transport, all--carried on by vis inertiae--
pressed forward into boats and into the ice-covered water and did not,
surrender.
```

Cell that turns on inside quotes:

```
"You mean to imply that I have nothing to eat out of.... On the
contrary, I can supply you with everything even if you want to give
dinner parties," warmly replied Chichagov, who tried by every word he
spoke to prove his own rectitude and therefore imagined Kutuzov to be
animated by the same desire.

Kutuzov, shrugging his shoulders, replied with his subtle penetrating
smile: "I meant merely to say what I said."
```

Andrej Karpathy, Visualizing and Understanding Recurrent Networks, 2015,

https://arxiv.org/abs/1506.02078

Embeddings

- A way to encode information to a lower dimensional space
  - PCA
  - We learn about this lower dimensional space through data

One hot encoding

- Categorical representation is usually represented by one hot encoding

- Categorical representations examples:
  - Words in a vocabulary, characters in Thai language

Apple -> 1 -> [1, 0, 0, 0, …]

Bird -> 2 -> [0, 1, 0, 0, …]

Cat -> 3 -> [0, 0, 1, 0, …]

- Sparse representation
  - Spare means most dimension are zero

One hot encoding

- Sparse – but lots of dimension
  - Curse of dimensionality
- Does not represent meaning.

Apple -> 1 -> [1, 0, 0, 0, …]

Bird -> 2 -> [0, 1, 0, 0, …]

Cat -> 3 -> [0, 0, 1, 0, …]

|Apple – Bird| = |Bird – Cat|

Getting meaning into the feature vectors

- You can add back meanings by hand-crafted rules
- Old-school NLP is all about feature engineering
- Word segmentation example:
  - Cluster Numbers
  - Cluster letters
- Concatenate them
- 1 = [0 0 0 0 1 0 0 0, 1, 0]
- ก = [0 0 0 1 0 0 0 0, 0, 1]
- า = [1 0 0 0 0 0 0 0, 0, 2]
- Which rules to use?
  - Try as many as you can think of, and do feature selection or use models that can do feature selection

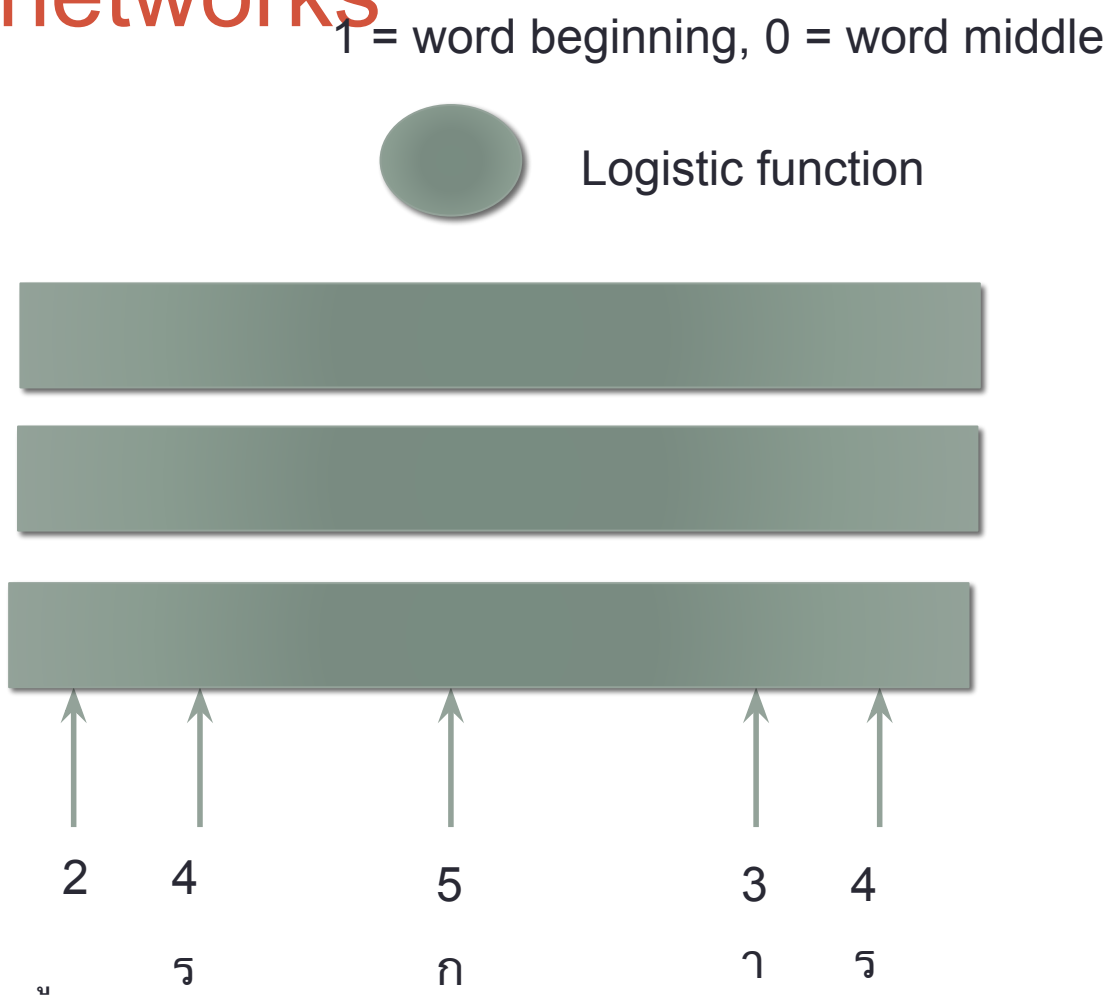
Dense representation

- We can encode sparse representation into a lower dimensional space
  - F: R^N -> R^M, where N>M

Apple -> 1 -> [1, 0, 0, 0, …] -> [2.3, 1.2]
Bird -> 2 -> [0, 1, 0, 0, …] -> [-1.0, 2.4]
Cat -> 3 -> [0, 0, 1, 0, …] -> [-3.0, 4.0]

- We can do this by using an embedding layer

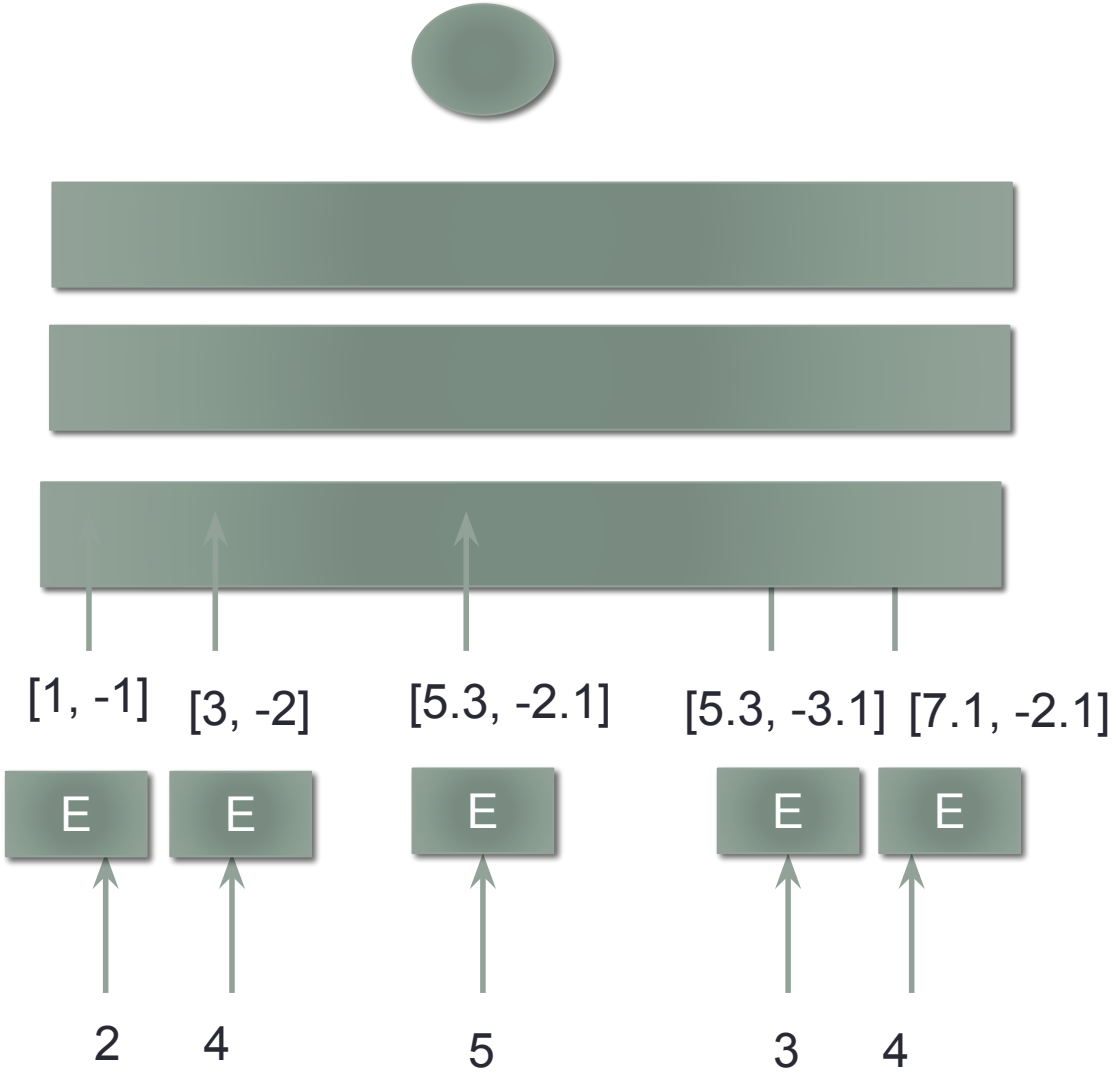
Word segmentation with fully connected networks



Adding embedding layer

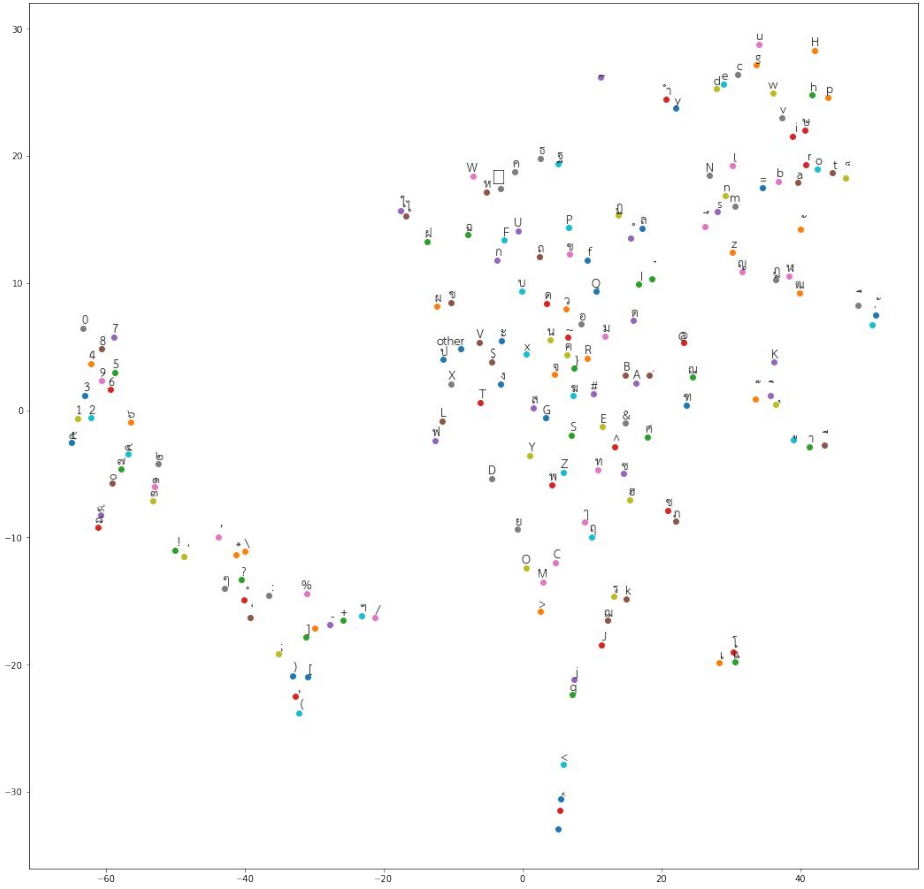
Embedding layer shares the same weights

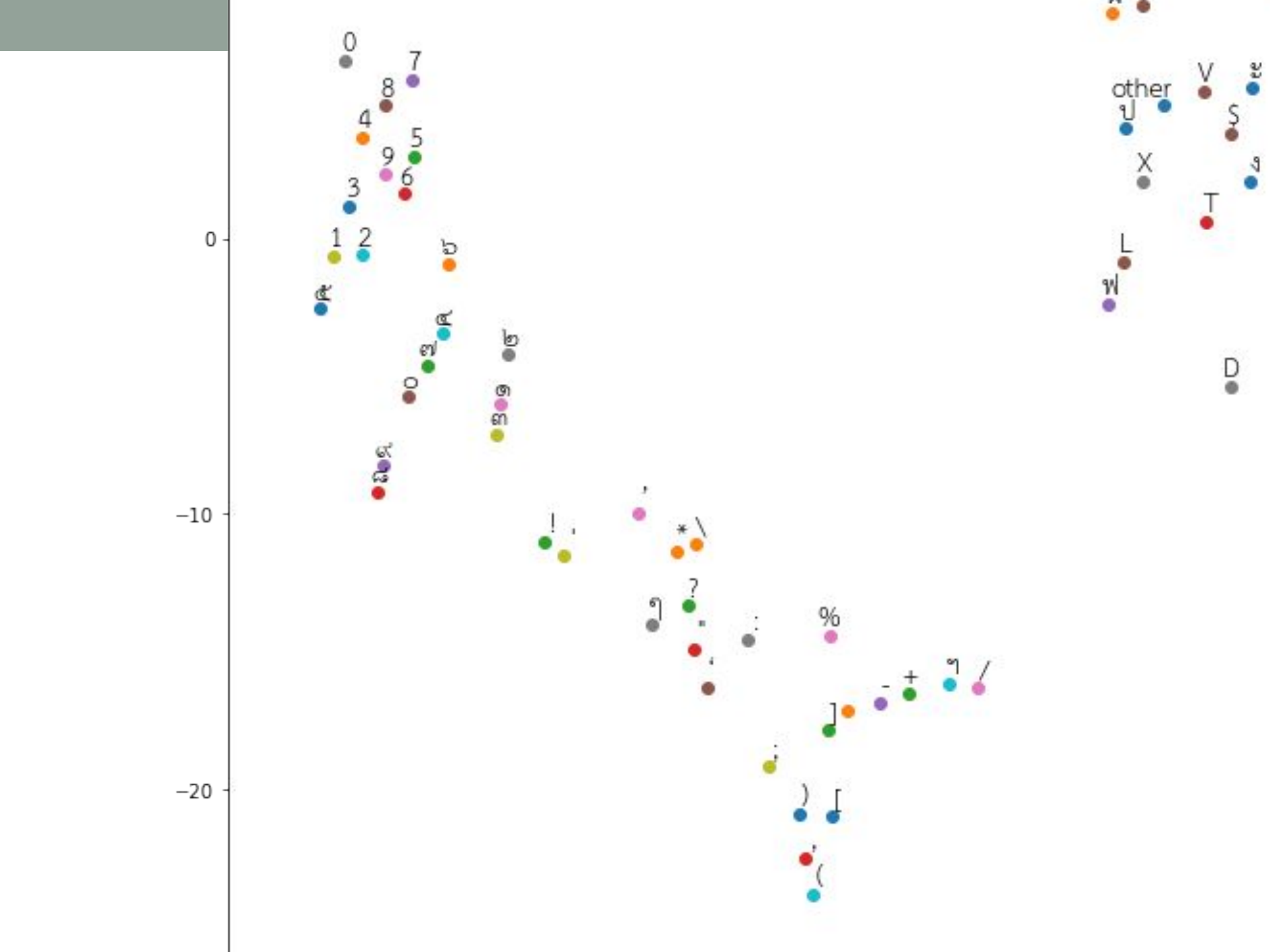
Parameter sharing!

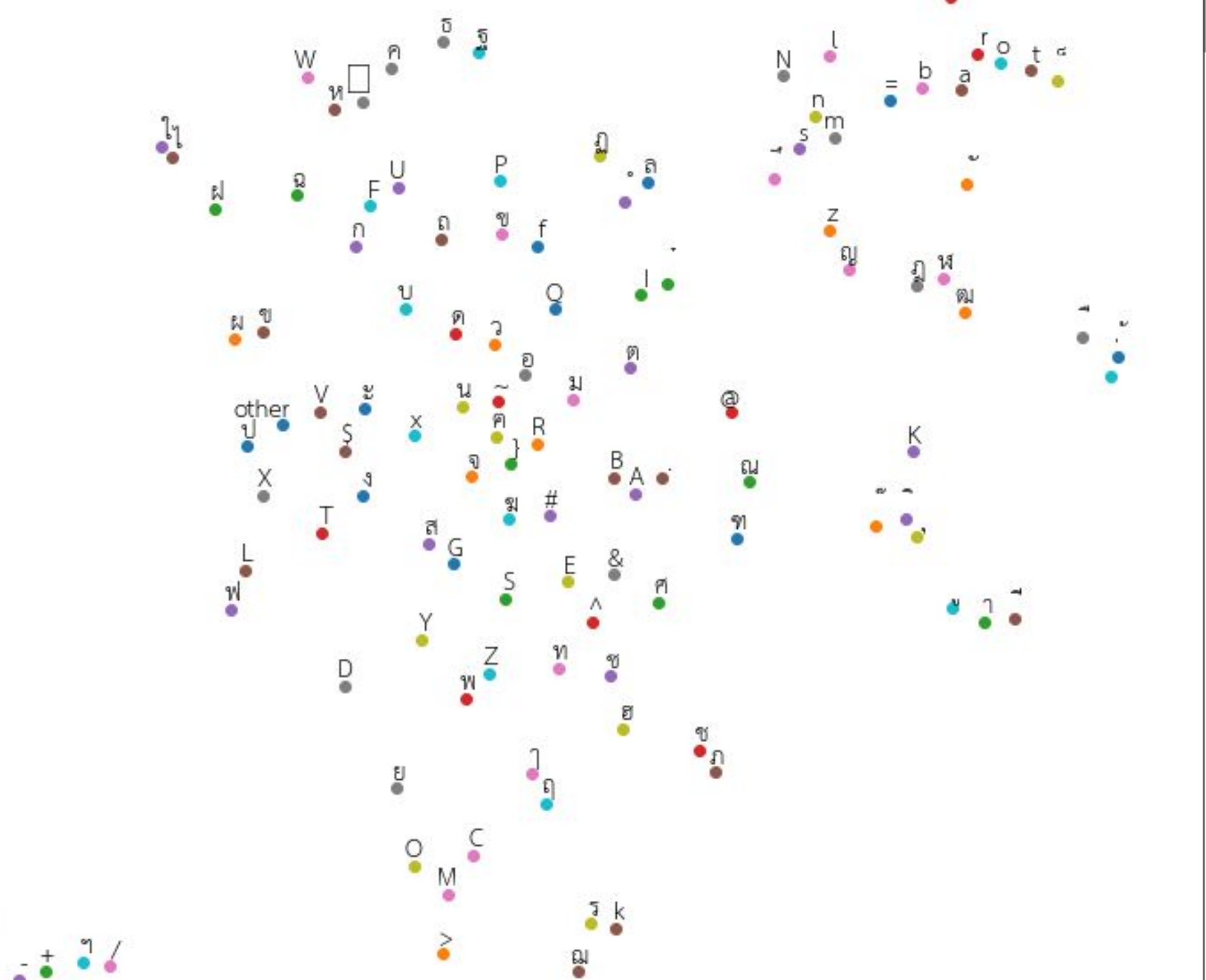


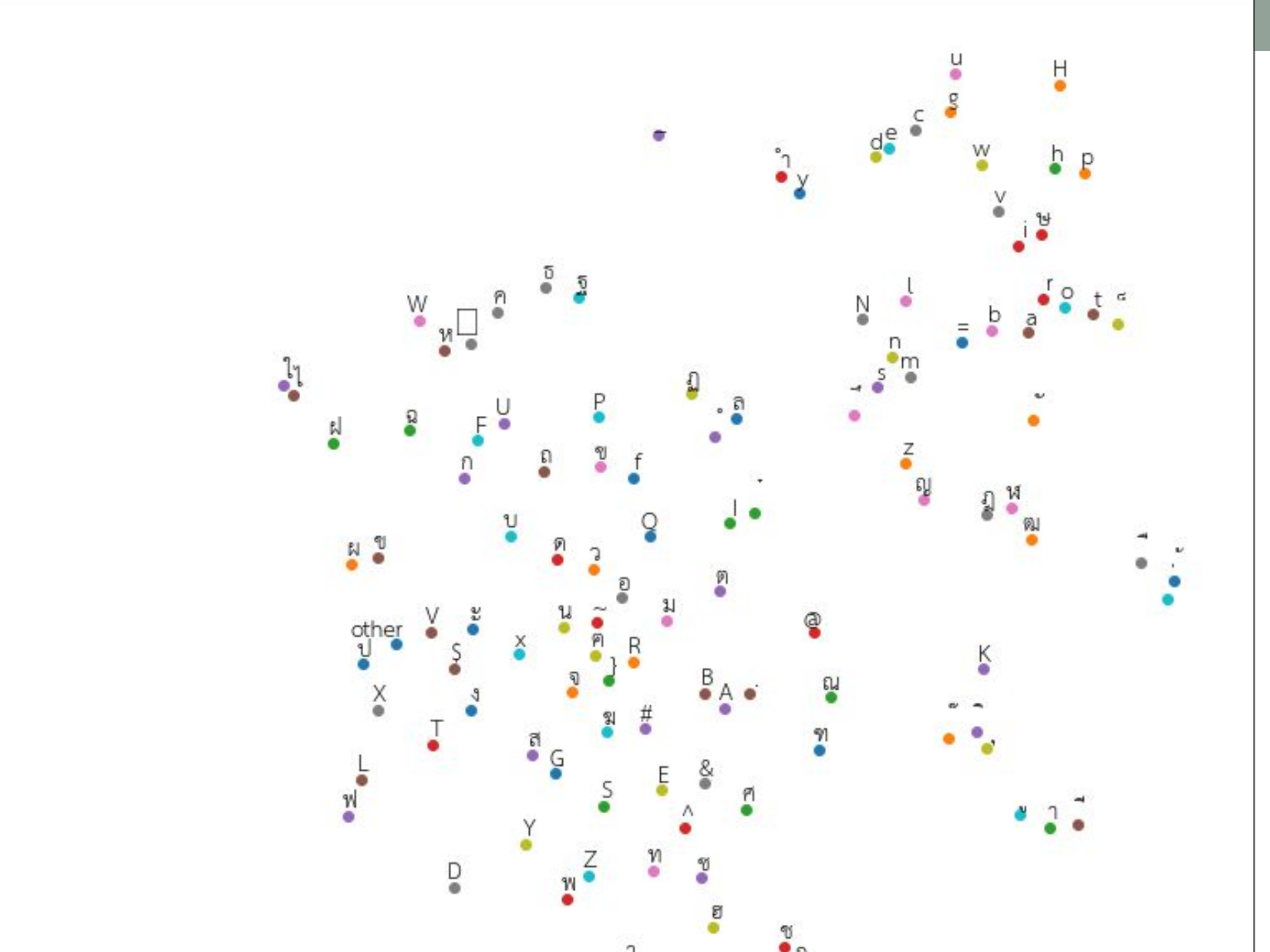
Embedding and meaning (semantics)

- Meaning is inferred from the task
- Embedding of 32 dimensions -> t-SNE into 2 dimension for visualization
- Automatically!



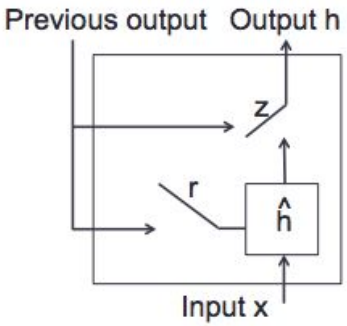

0
−10
−20
other

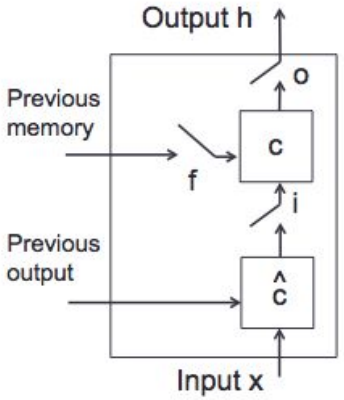

other

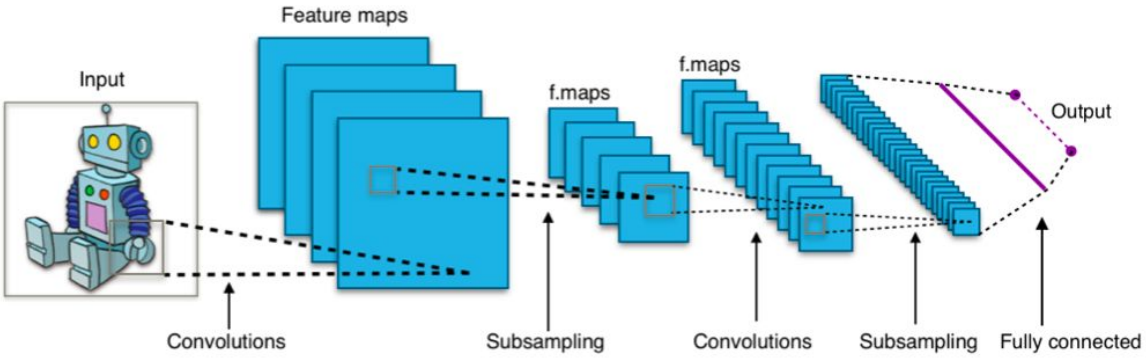

other

Neural networks

- Fully connected networks
  - SGD, backprop
- CNN
- RNN, LSTM, GRU





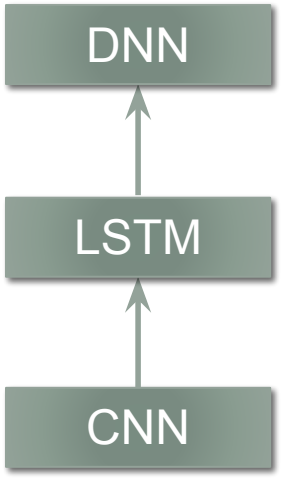


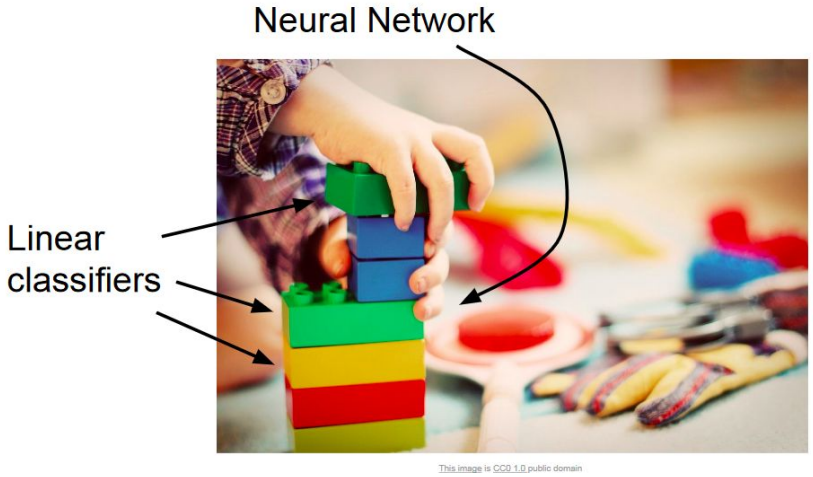
Attention modeling
Recursive neural networks

← Future lectures

DNN Legos

- Typical models now consists of all 3 types
  - CNN: local structure in the feature. Used for feature learning.
  - LSTM: remembering longer term structure or across time
  - DNN: Good for mapping features for classification. Usually used in final layers





https://www.analyticsvidhya.com/blog/2017/08/10-advanced-deep-learning-architectures-data-scientists/

Back to tokenization…

TABLE II
RESULTS OF THE SIX BEST TEAMS

Type of participants	F-Measure (%)	Time (mm:ss)
Non-Students[a]	97.94937	00:47
Non-Students	97.84097	02:46
Non-Students	97.18822	00:26
Bachelor Students[b]	95.78162	01:08
Master Students	95.56670	12:14
PhD+Master Students	92.02067	02:28

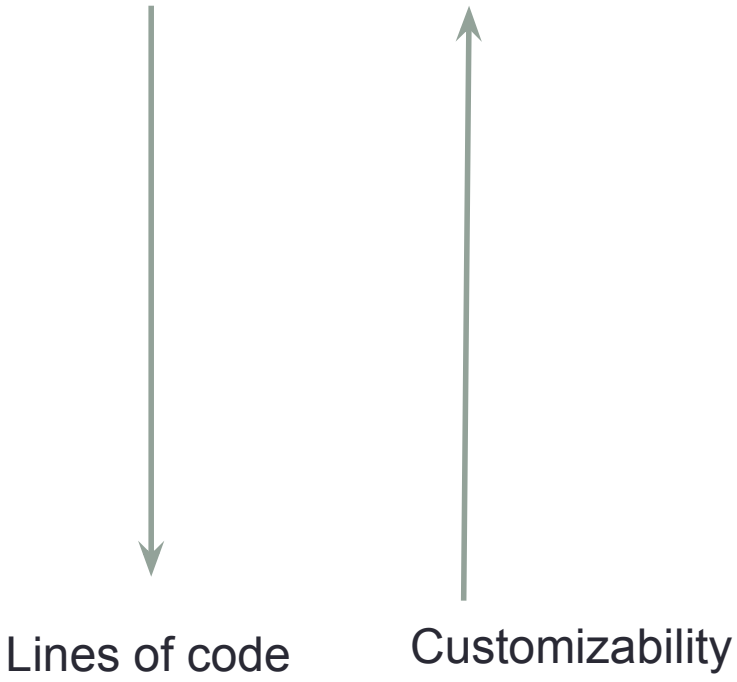
[a]Best of the BEST 2009 Award Winner
[b]BEST Student 2009 Award Winner

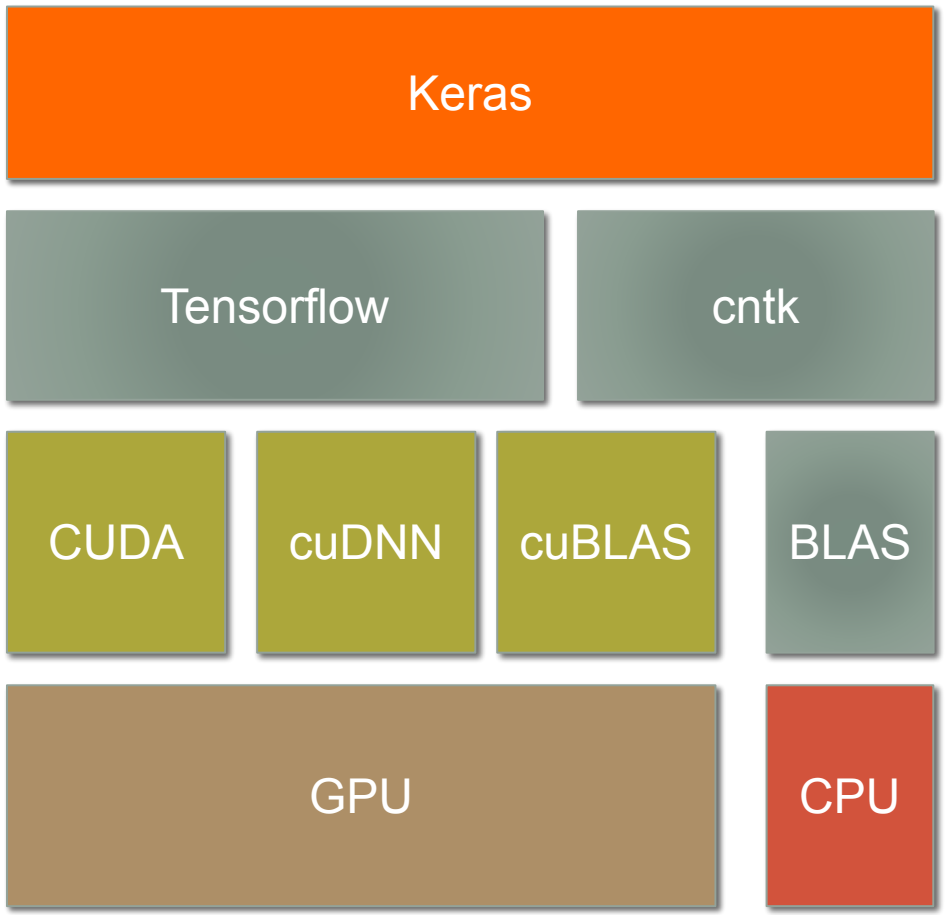
Pattern recognition course homework best F-score: 97.91
Sertis' model: 99.18

BEST 2009 : Thai word segmentation software contest
http://ieeexplore.ieee.org/document/5340941/
https://sertiscorp.com/thai-word-segmentation-with-bi-directional_rnn/

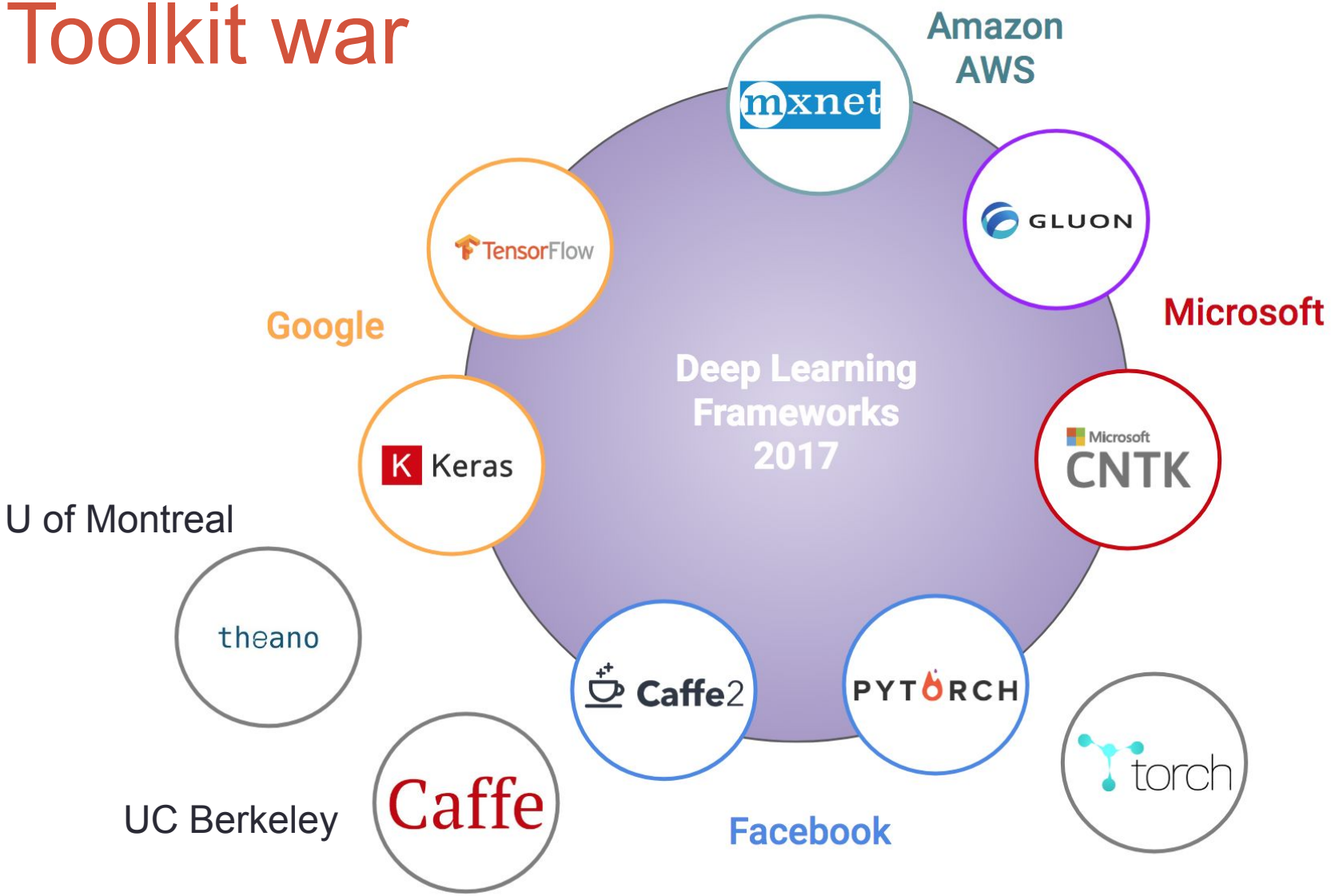
What toolkit

- Tensorflow – lower level
- Keras – higher level





Toolkit war



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

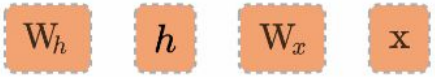
Which?

- Easiest to use and play with deep learning: Keras (now in tf2)
- Easiest to use and tweak: pytorch

A graph is created on the fly

```
from torch.autograd import Variable

x = Variable(torch.randn(1, 10))
prev_h = Variable(torch.randn(1, 20))
W_h = Variable(torch.randn(20, 20))
W_x = Variable(torch.randn(20, 10))
```

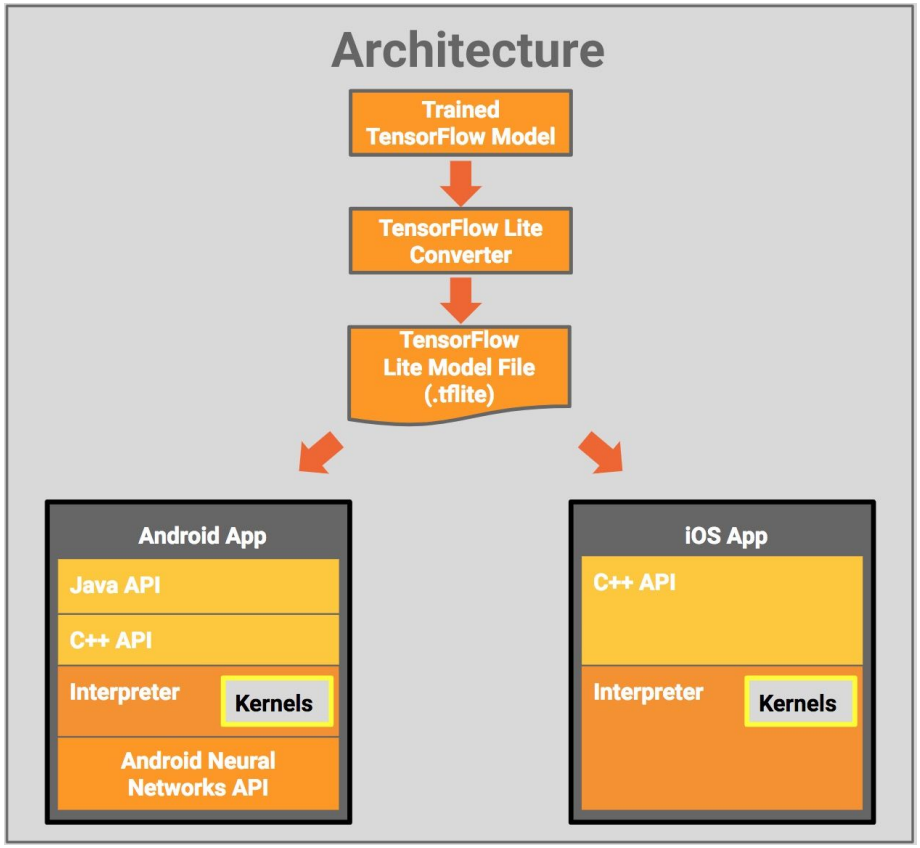


Dynamic computation graph in pytorch. Good for when size of and structure of input data varies. Example parse tree.

http://pytorch.org/about/

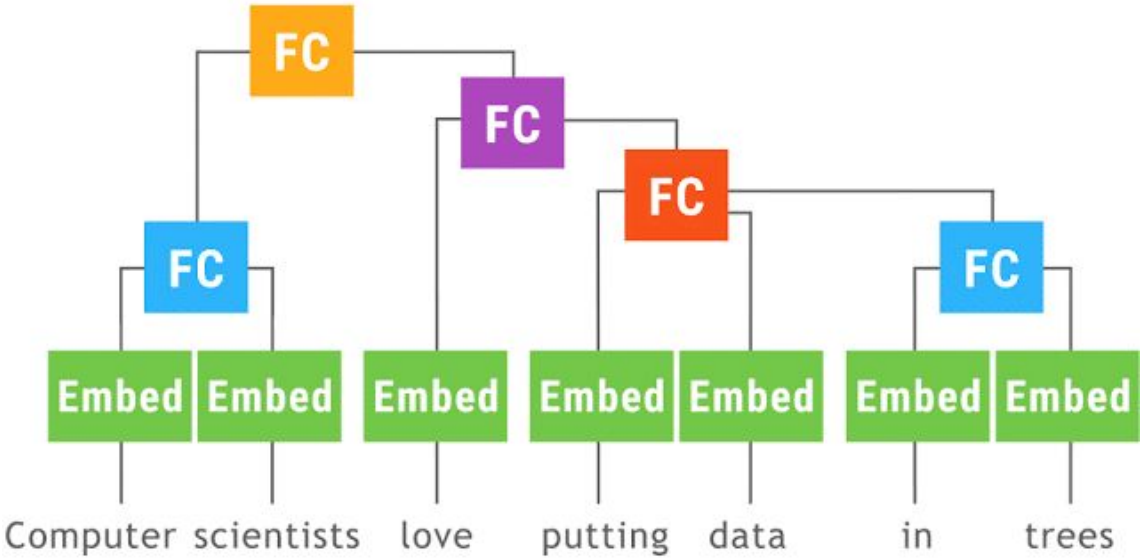
Which?

- Easiest to use and play with deep learning: Keras
- Easiest to use and tweak: pytorch
- Easiest to deploy: tensorflow
  - Tensorflow lite for mobile



Which?

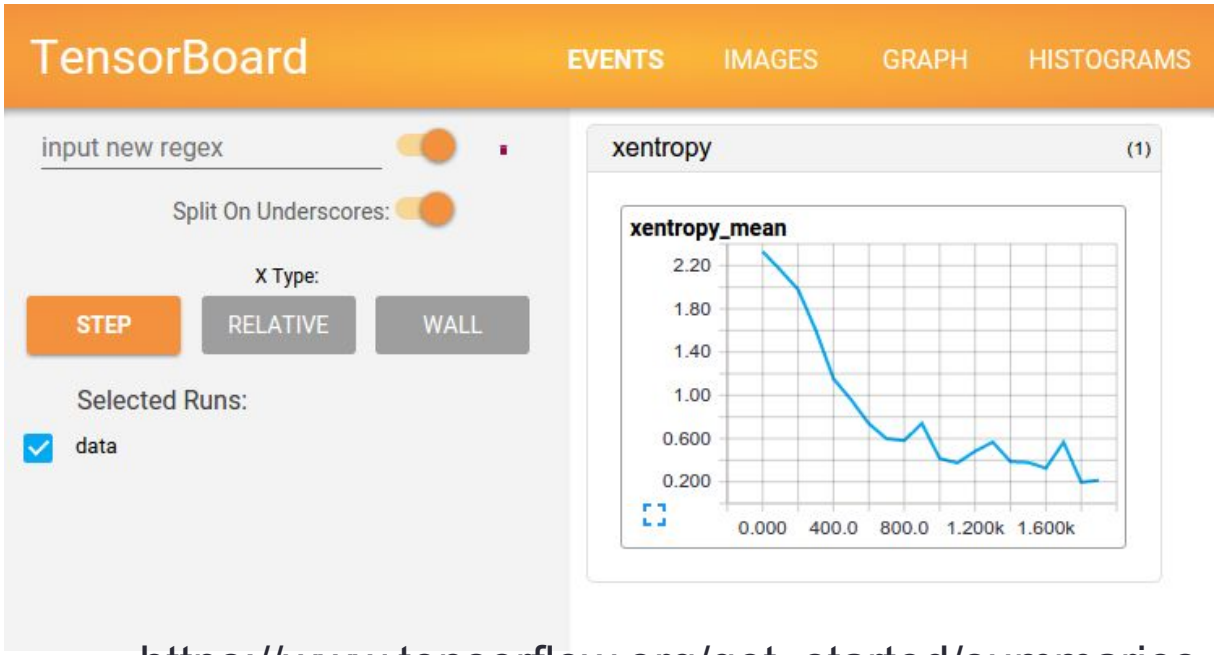
- Easiest to use and play with deep learning: Keras
- Easiest to use and tweak: PyTorch
- Easiest to deploy: TensorFlow
  - TensorFlow Lite for mobile
  - Now support dynamic graph: eager execution mode (default in tf2)



https://www.tensorflow.org/guide/eager

Which?

- Easiest to use and play with deep learning: Keras
- Easiest to use and tweak: PyTorch
- Easiest to deploy: TensorFlow
- Best tools: TensorFlow Tensorboard visualization



https://www.tensorflow.org/get_started/summaries_and_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
  - Now support dynamic graph: eager execution mode
- Biggest community: TensorFlow
  - Github repositories & pre-trained models (Tensorflow Hub)
- TPUs: Google

Lab

- Keras

- Something to read
  - https://keras.io/getting-started/functional-api-guide/
  - Or the tf2 docs (somewhat incomplete)
    - https://www.tensorflow.org/beta/guide/keras

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.