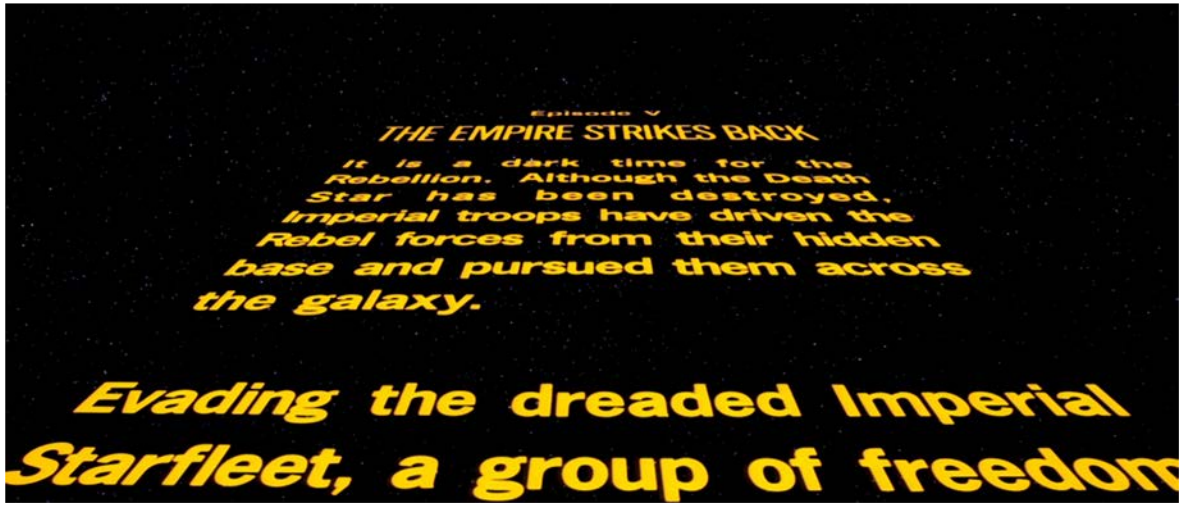
TOKENIZATION

Intro to neural networks

The need for segmentation

- Text as a stream of characters



- We need a way to understand the meaning of text
  - Break into words (assign meaning to word) ← Tokenization
  - Break into sentences (put word meanings back to sentence meaning)

http://starwars.wikia.com/wiki/Opening_crawl

Tokenization

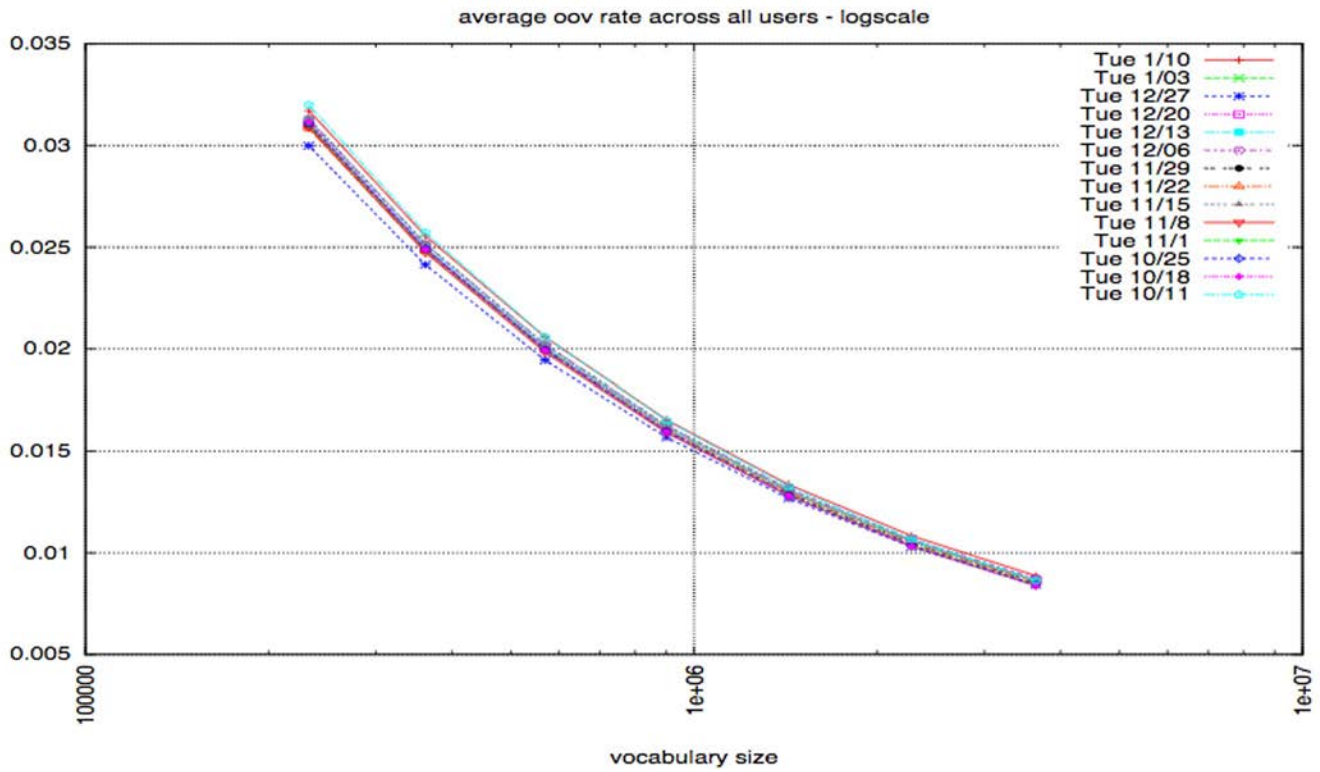
- A token should
  - 1. Linguistically significant
  - 2. Methodologically useful

Dictionary-based vs Machine-learning-based

- Dictionary-based
  - Longest matching
  - Maximal matching
- Machine-learning-based

Dictionary-based drawbacks

- Cannot handle words outside of the dictionary (Out-of-Vocabulary, OOV words)
- Performs worse than machine-learning-based approach



Maryam Kamvar, Ciprian Chelba, OPTIMAL SIZE, FRESHNESS AND TIME-FRAME FOR VOICE SEARCH VOCABULARY

NEURAL NETWORKS

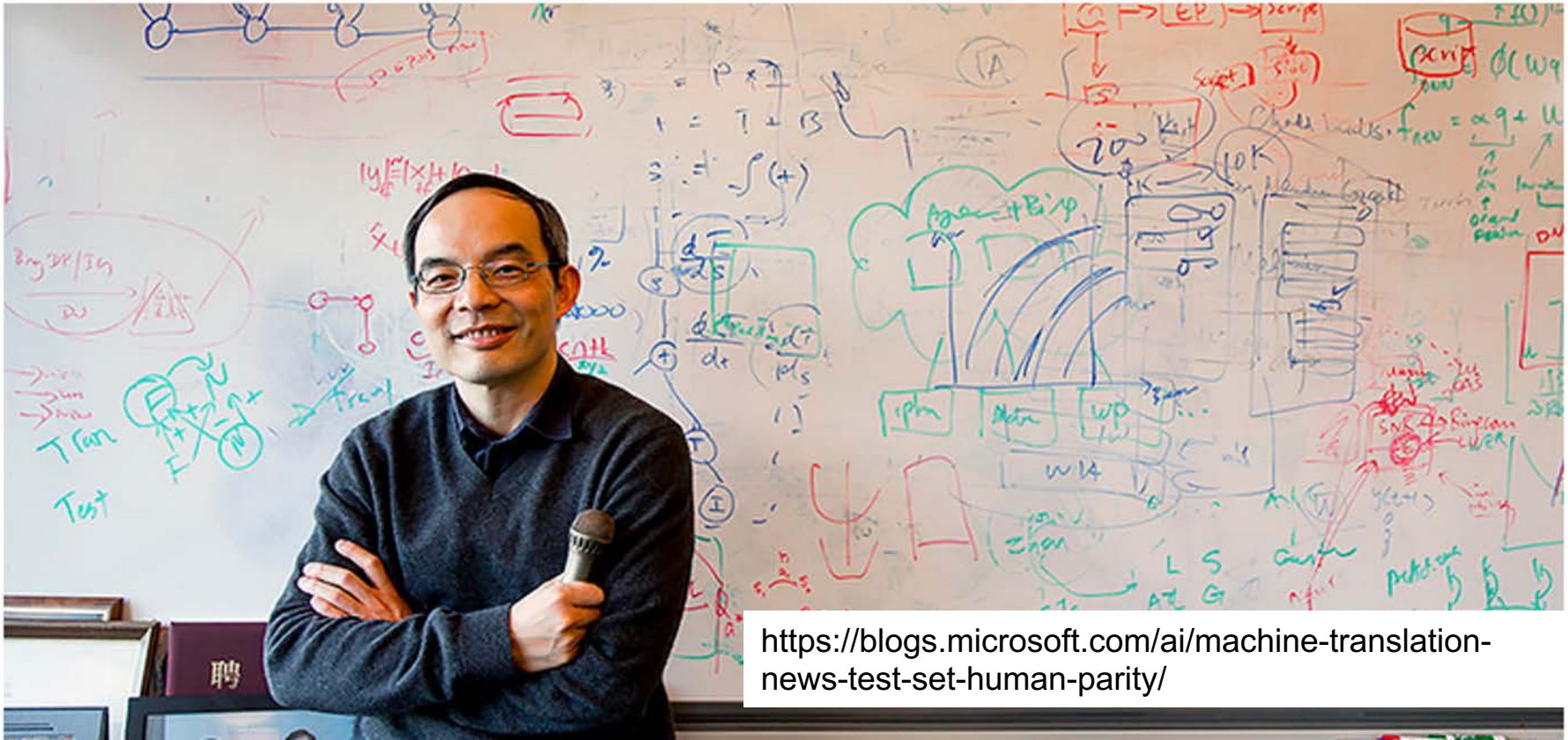
Deep learning = Deep neural networks = neural networks

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

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/

Deep learning in NLP

Easy task modest gains

	Traditional ML	Deep learning
Sentiment (th)	72%	76%
Topic classification (th)	67%	70%
PoS (th)	96%	97%

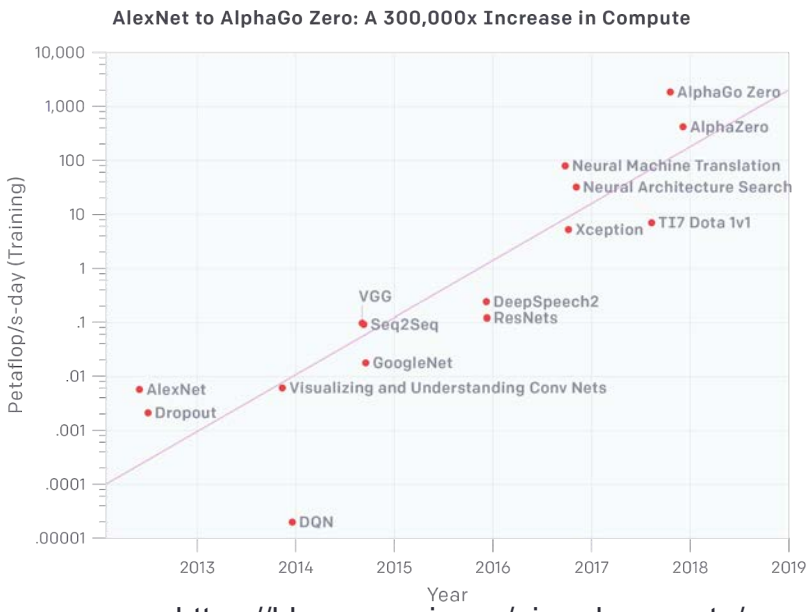
Harder task larger gains

	Traditional ML	Deep learning
QA	51%*	90%
Creating image from text	???	very good

wangchanbert
https://www.aclweb.org/anthology/D16-1264/
https://openai.com/blog/dall-e/

Why now

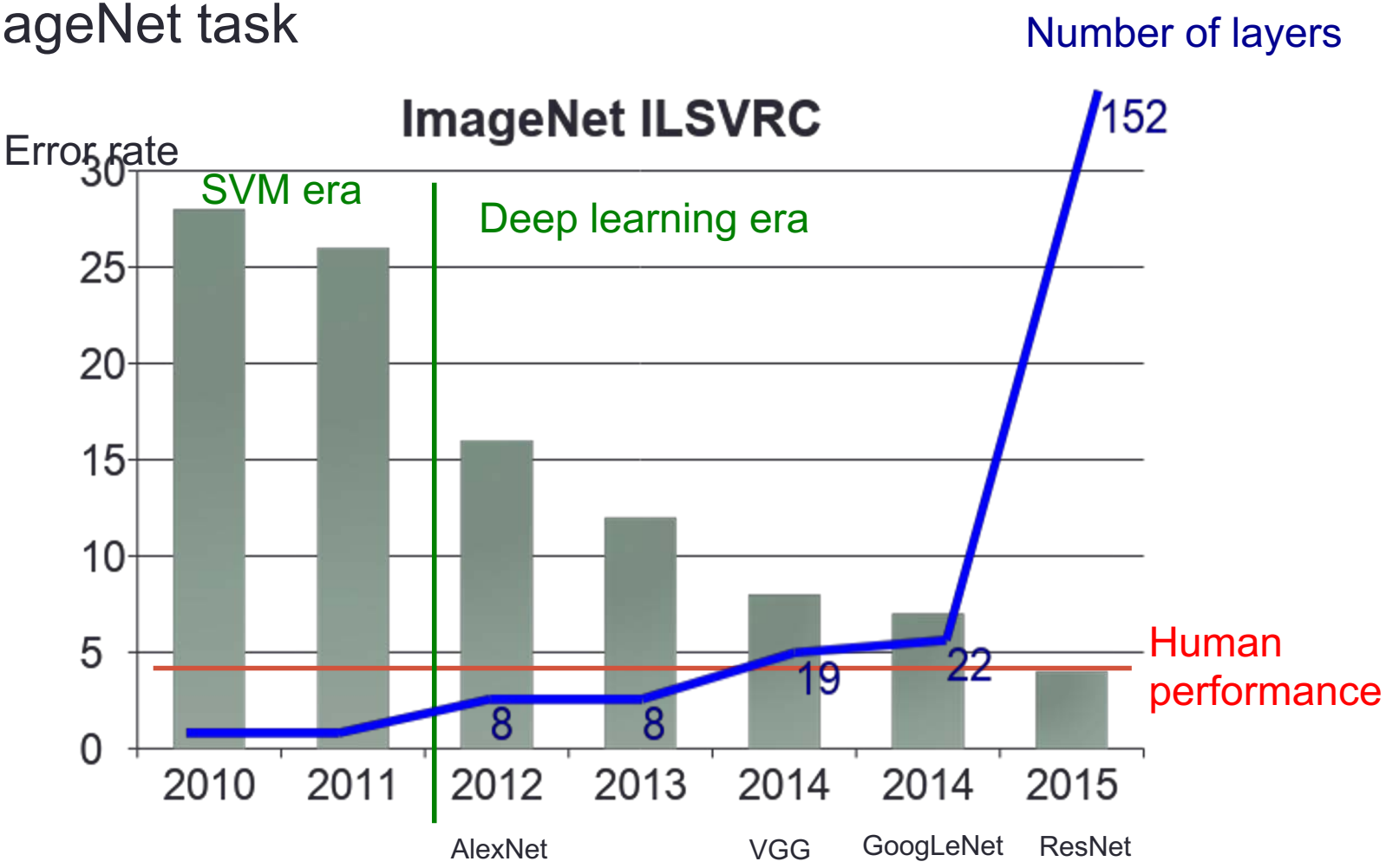
- Neural Networks has been around since 1990s
- Big data – DNN can take advantage of large amounts of data better than other models
- GPU – Enable training bigger models possible
- Deep – Easier to avoid bad local minima when the model is large



https://blog.openai.com/ai-and-compute/

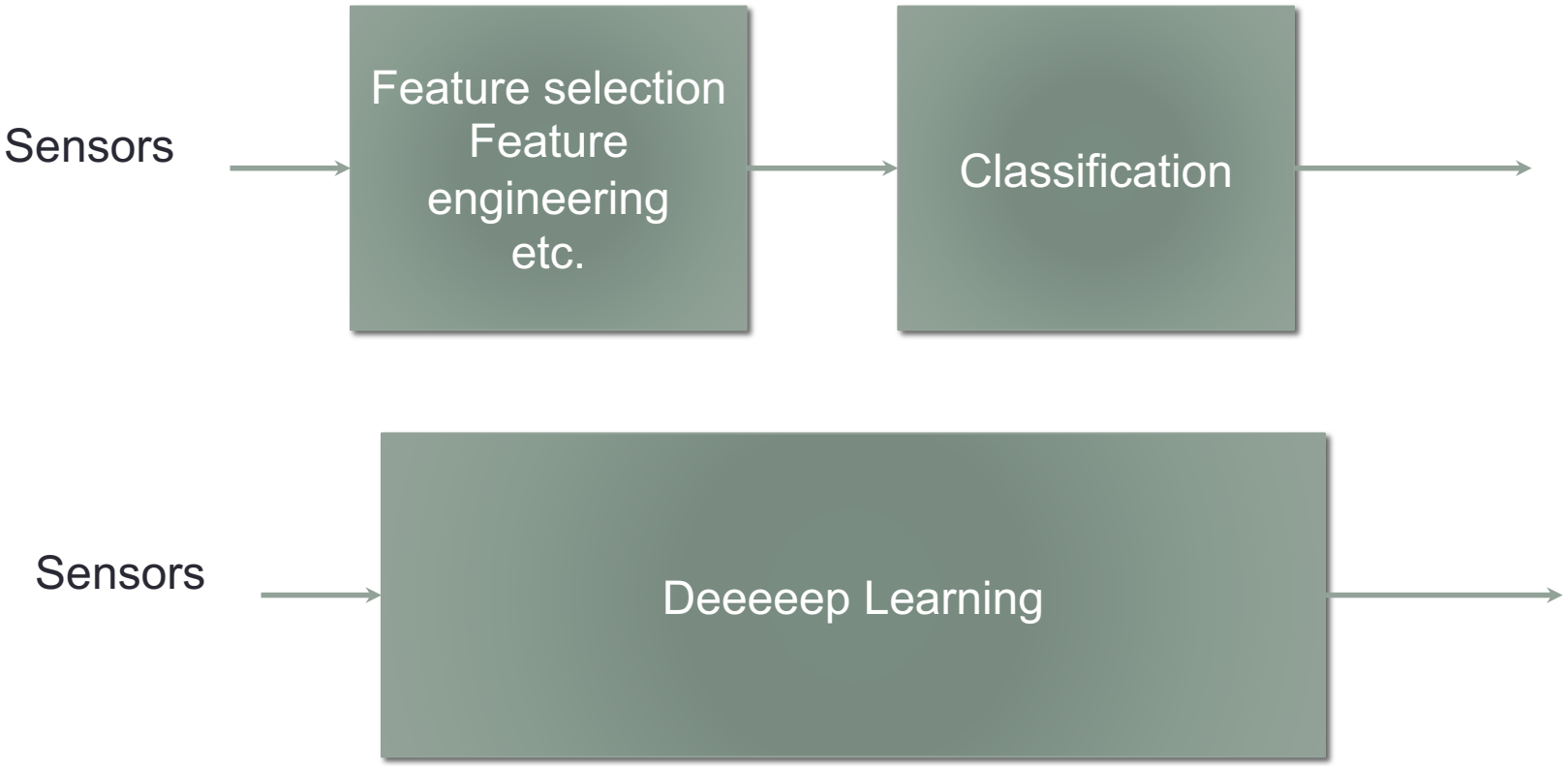
Wider and deeper networks

- ImageNet task



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

Traditional VS Deep learning

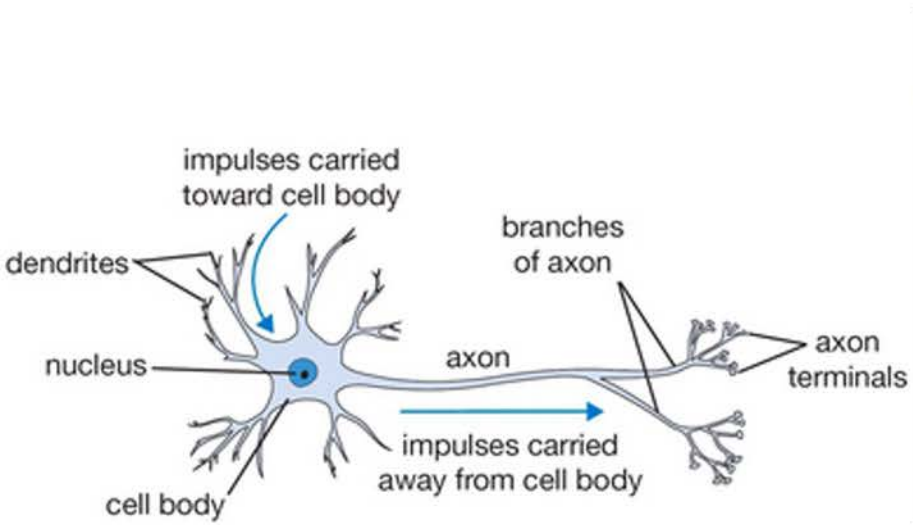


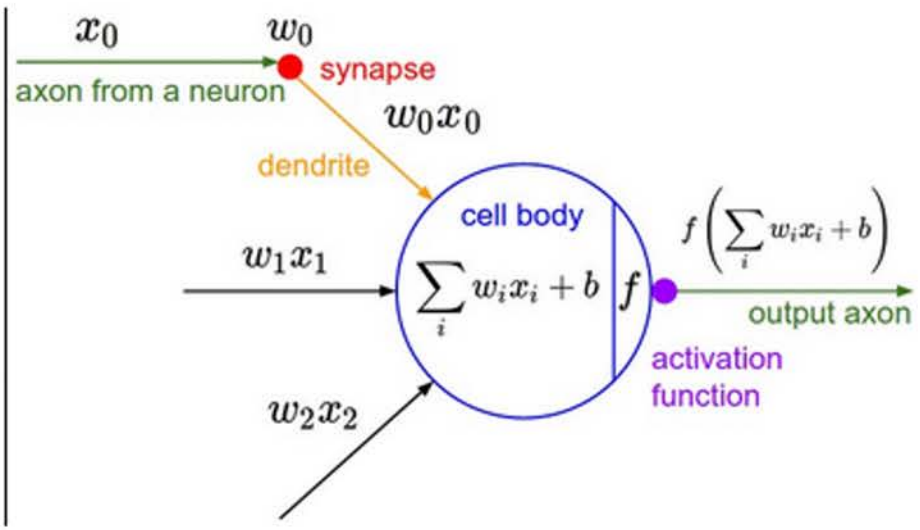
Neural networks

- 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

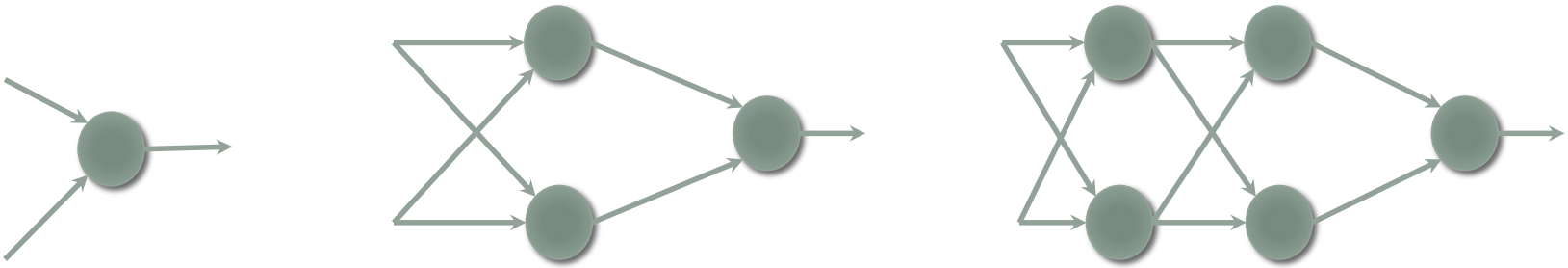


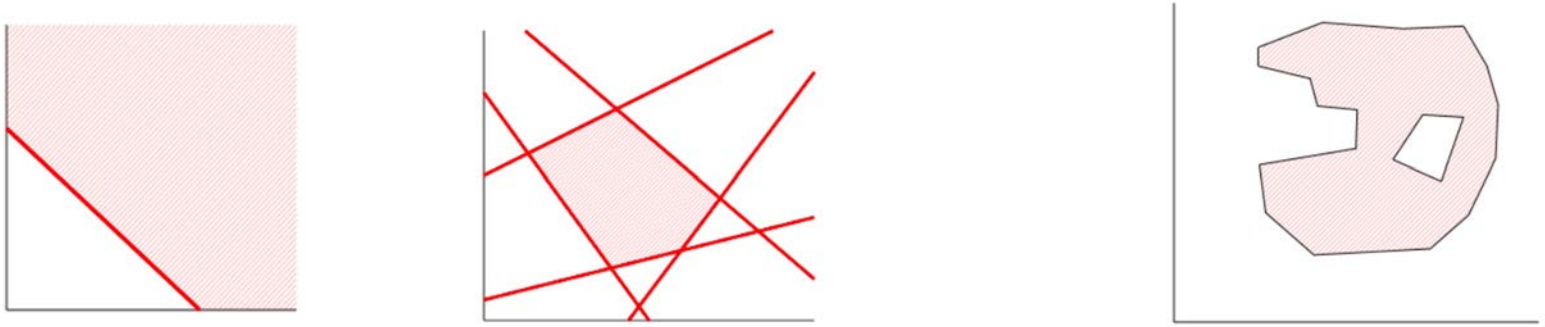


http://cs231n.github.io/neural-networks-1/

Combining neurons

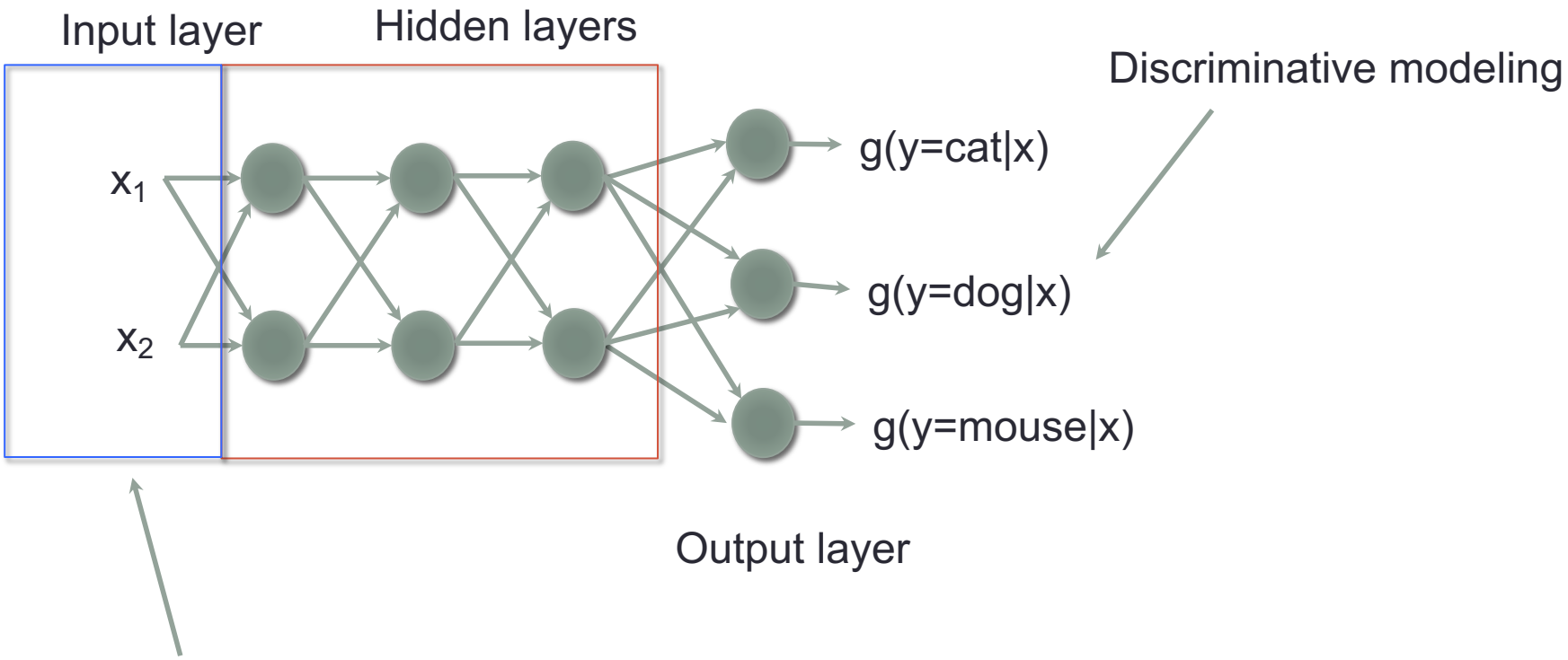
- Each neuron splits the feature space with a hyperplane
- Stacking neuron creates more complicated decision boundaries
- More powerful but prone to overfitting





Terminology

Deep in Deep neural networks means many hidden layers



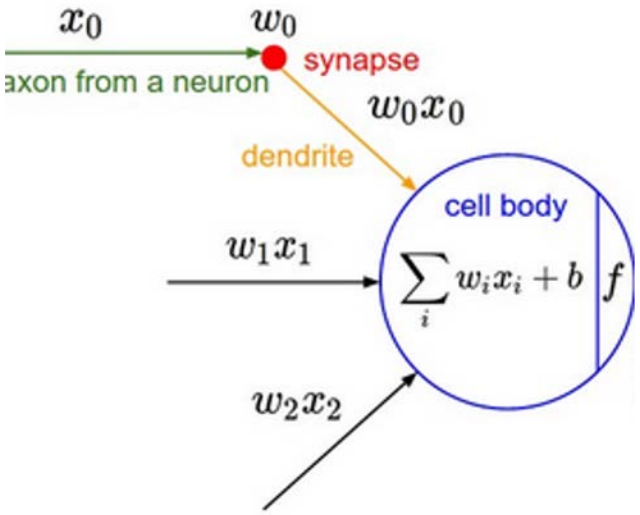
Matrices

- Inputs

features

x_1
x_2
x_3

X



W

weights

w_1
w_2
w_3

+ b

$W^T X + b$

w_1 w_2 w_3

x_1
x_2
x_3

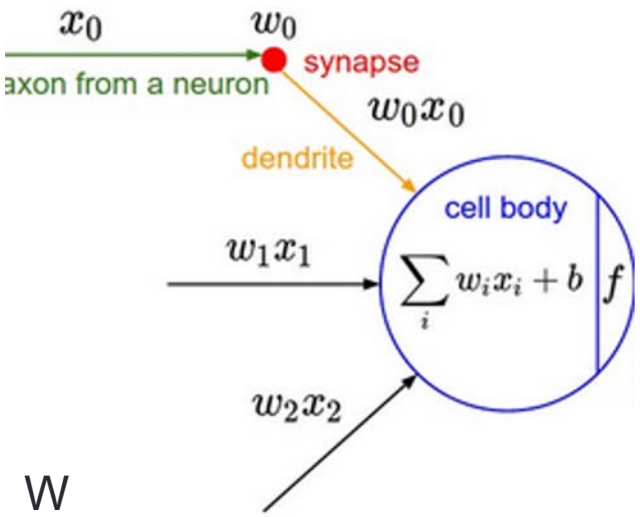
Matrices

- Inputs

samples

features

$$X = \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \end{bmatrix}$$



$$W = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

weights

$W^T X + b$

$$\begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \end{bmatrix} + \begin{bmatrix} b \end{bmatrix}$$

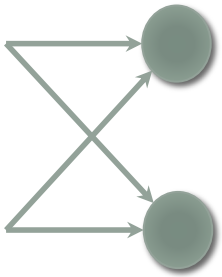
Matrices

- Inputs

samples

features

$$X = \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \end{bmatrix}$$



neorons

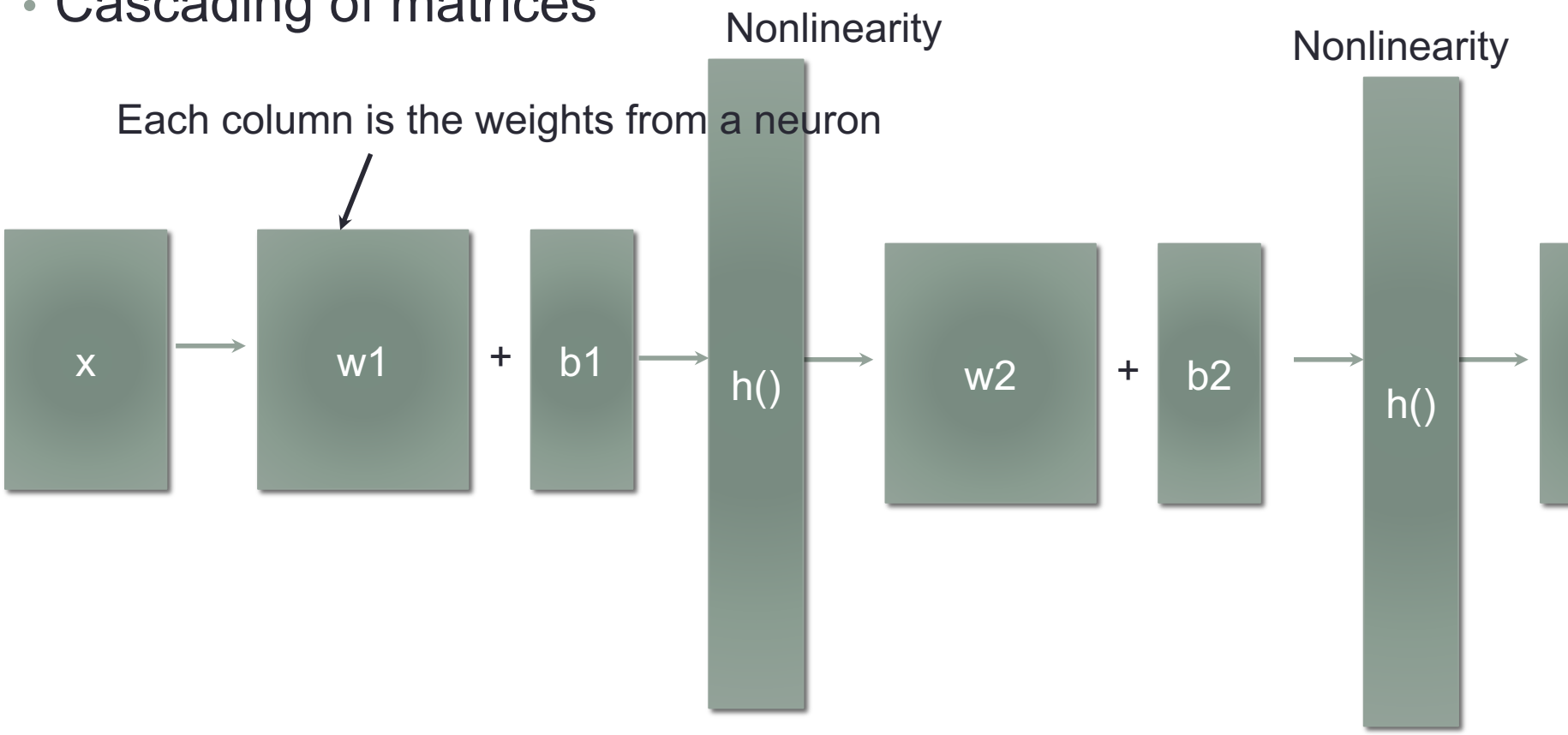
weights

$$W = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix}$$

$$W^T X + b = \begin{bmatrix} w_{11} & w_{21} & w_{31} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

More linear algebra

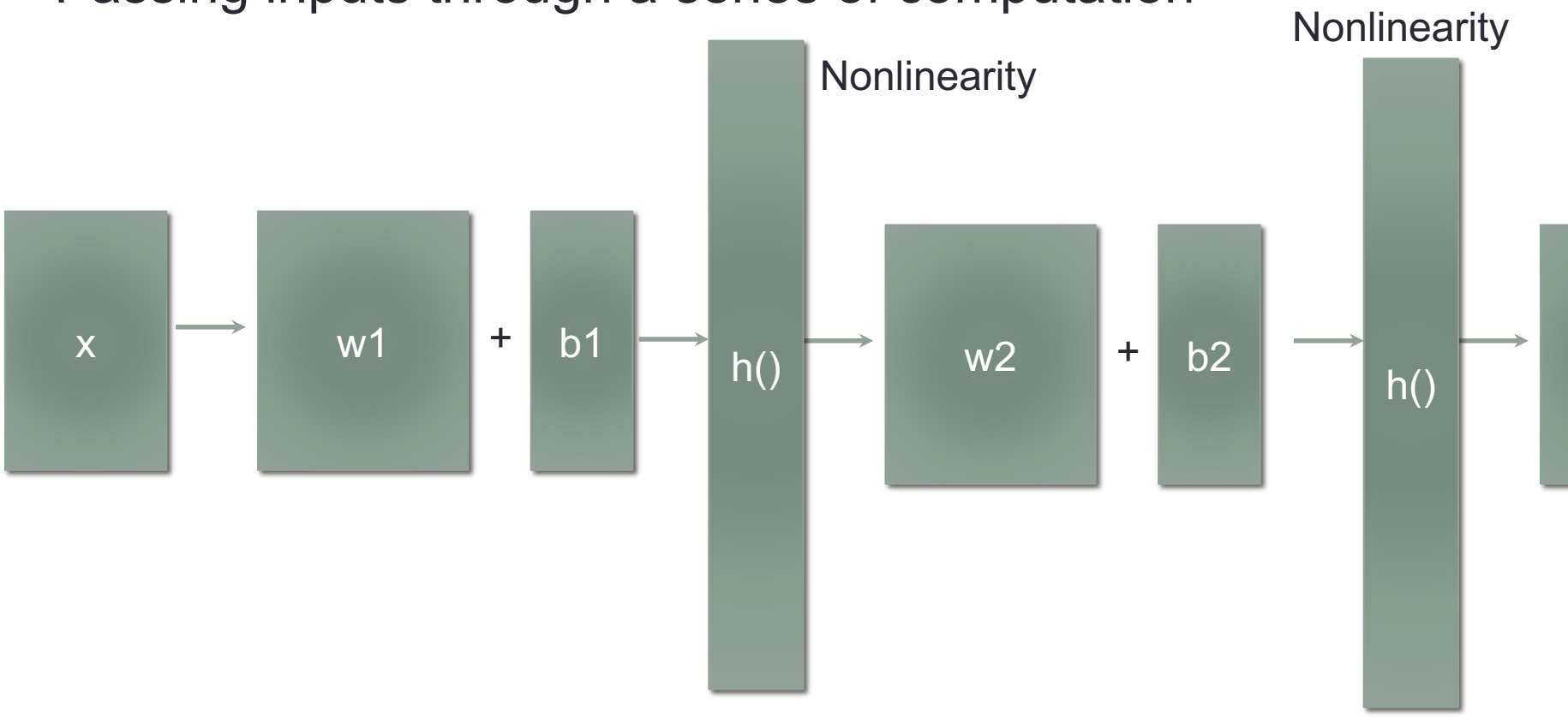
- Cascading of matrices



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

Computation graph

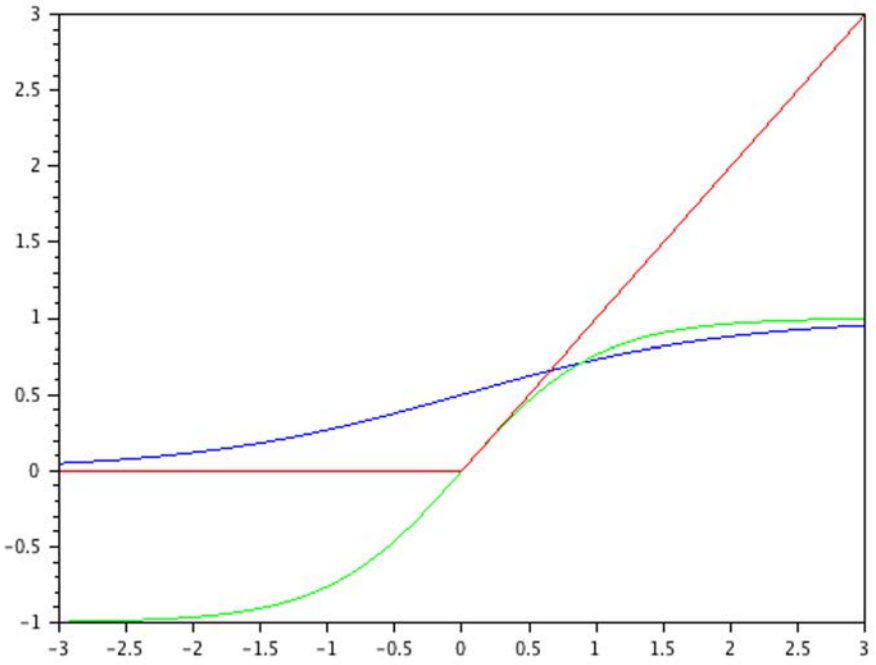
- Passing inputs through a series of computation



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

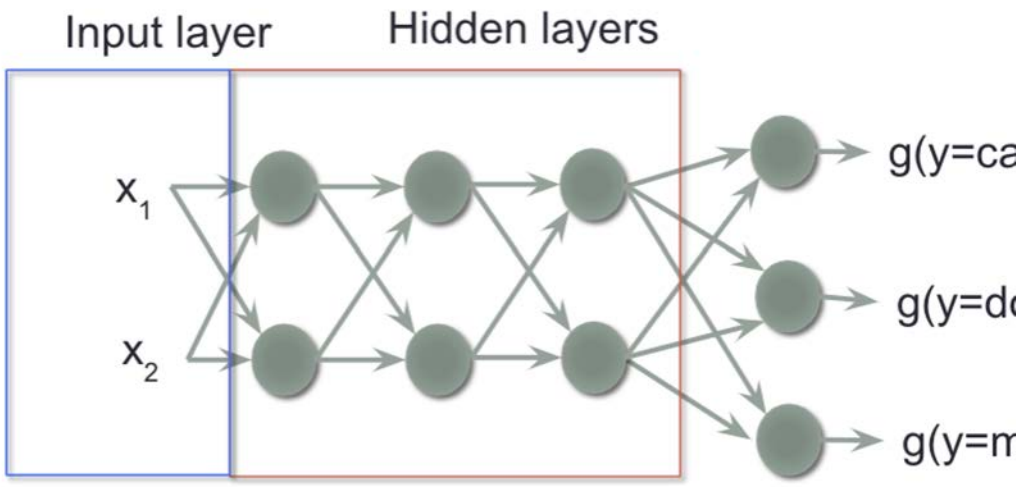
Non-linearity

- 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)
  - LeakyReLU, ELU, PreLU
- Sigmoid Linear Units (SiLU)
  - Swish, Mish, GELU
- Most popular is ReLU and its variants (Fast to train, and more stable)



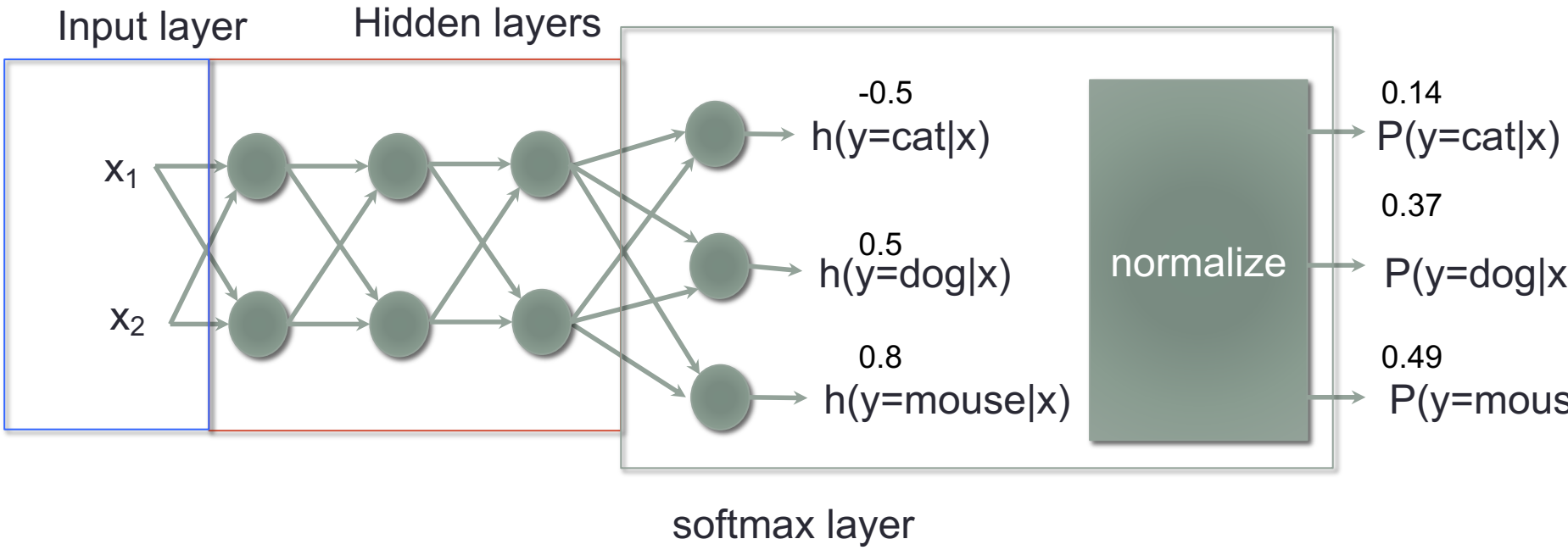
Output layer – Softmax layer

- 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



Softmax layer

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



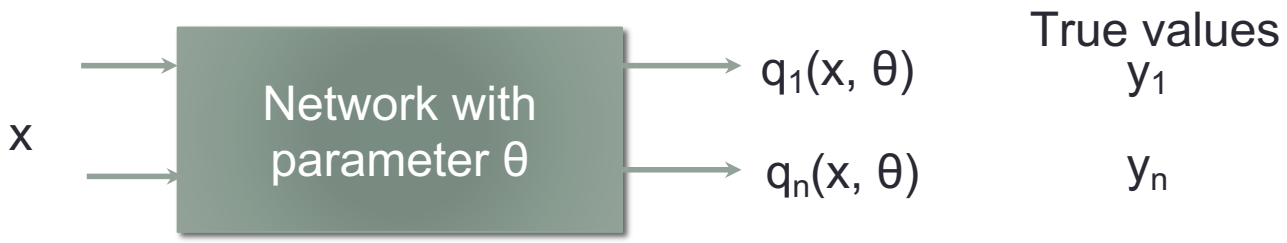
Neural networks

- Fully connected networks
  - Neuron
  - Non-linearity
  - Softmax layer
- DNN training
  - Loss function
  - 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



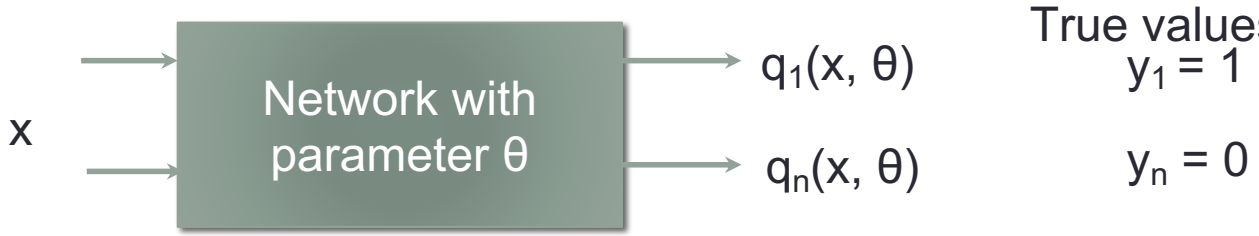
Cross entropy loss

- 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

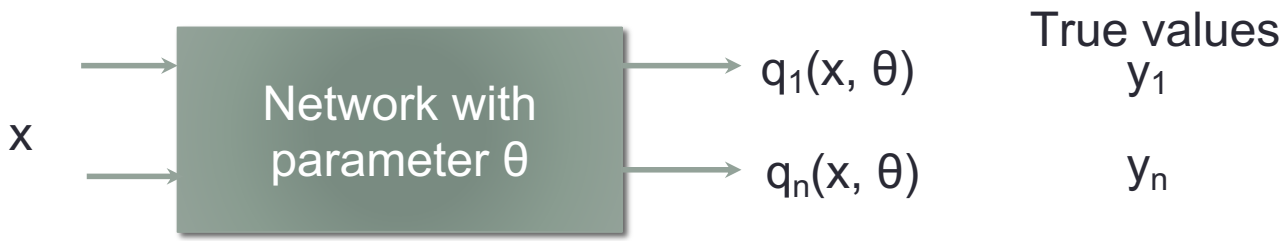


Sum of squared errors

- Used for any real valued outputs such as regression

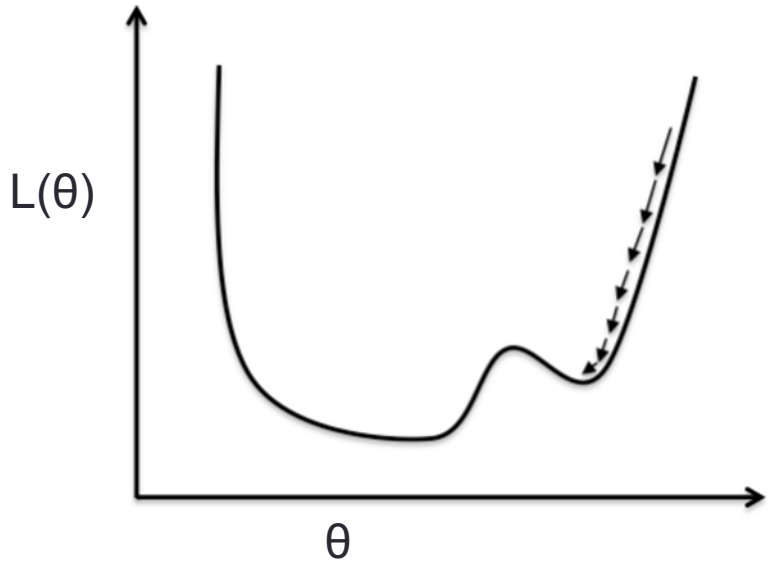
$$L = \frac{1}{2}\Sigma_n(y_n - q_n(x,\theta))^2$$

- 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

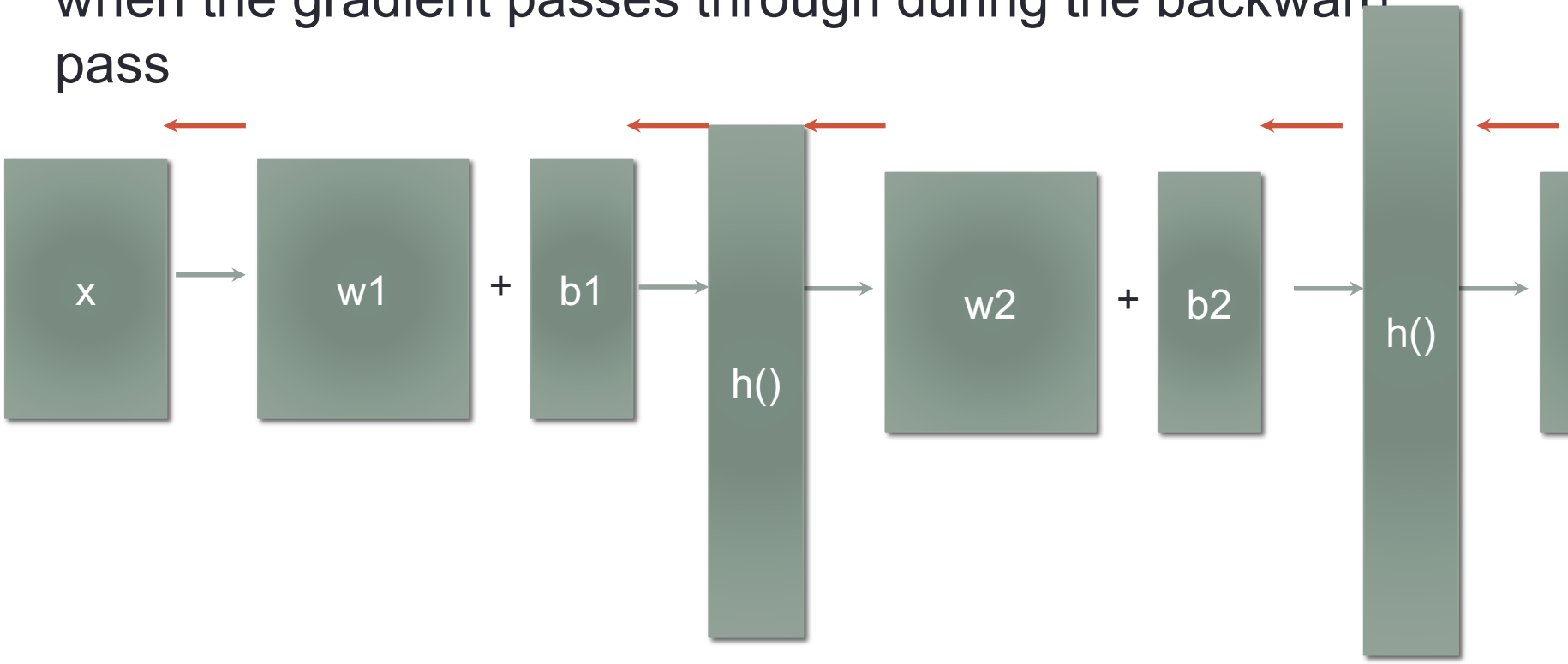
Examples to read

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

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

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
- Now usually initialized randomly
  - One strategy (Xavier init)

    $$var(w) = 2/(fan\_in + fan\_out)$$
  - For ReLUs (He init)

    $$var(w) = 2/(fan\_in)$$

  - Or use a pre-trained network as initialization

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. 2010
Kaiming He, Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. 2015

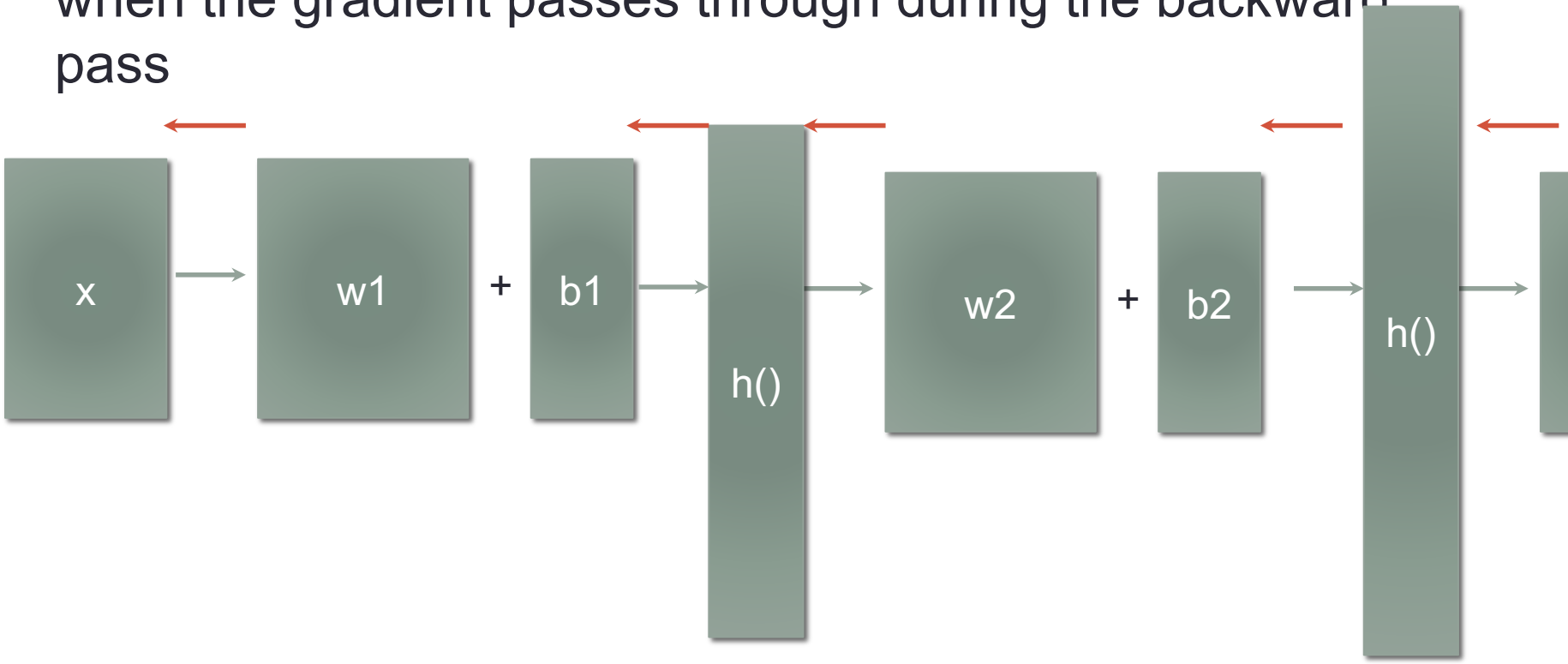
Stochastic gradient descent (SGD)

- 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

Note: one can read relationship between batch size and learning rate here
https://arxiv.org/abs/1711.00489

Backprop and computation graph

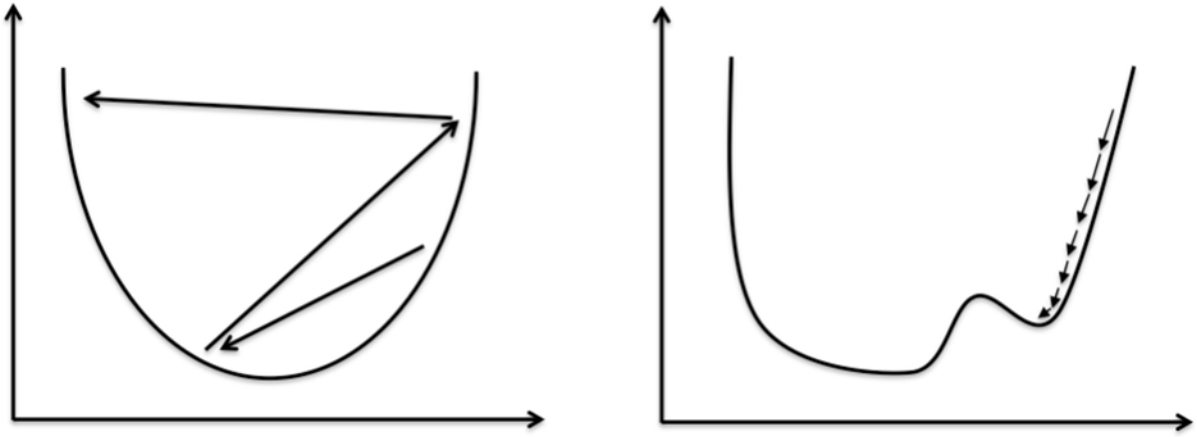
- We can also define what happens to a computing graph when the gradient passes through during the backward pass



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.

Learning rate

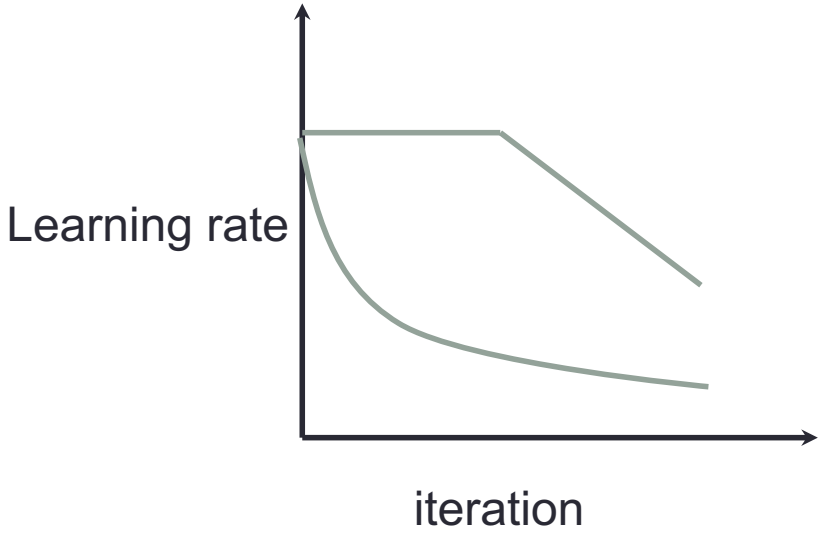
- How fast to go along the gradient direction is controlled by the learning rate
- Too large models diverge
- Too small the model get stuck in local minimas and takes too long to train



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)

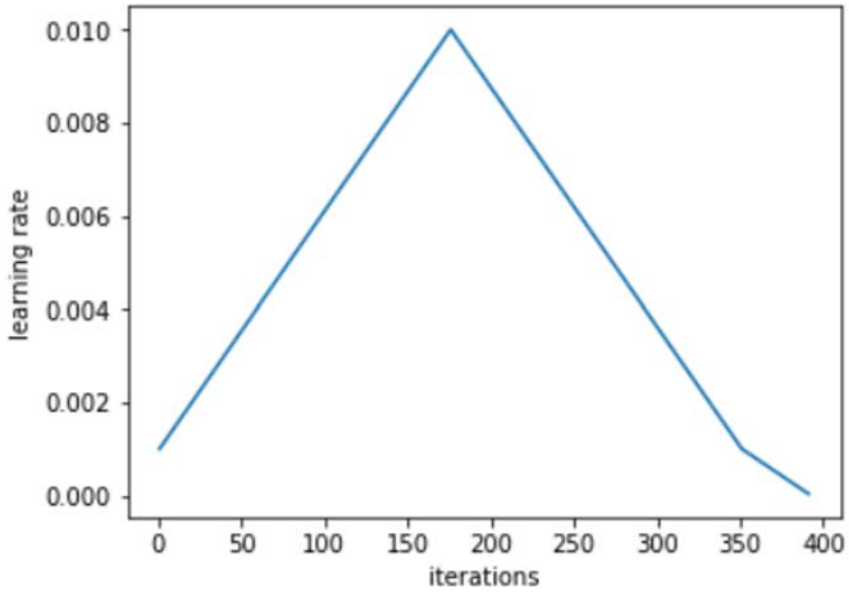


Learning rate warm up

Initial point of the network can be at a bad spot.

Try not to go to fast - has a warm up period.

Useful for large datasets, or adaption (transfer learning)



Potentially leads to faster convergence and better accuracy

See links below for methods to select the shape of the triangle

https://sgugger.github.io/the-1cycle-policy.html#the-1cycle-policy

Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour

Cyclical Learning Rates for Training Neural Networks

Optimizers

- Besides learning rate scheduling (coarse grain) we can do finer (and automatic) control of the learning rate

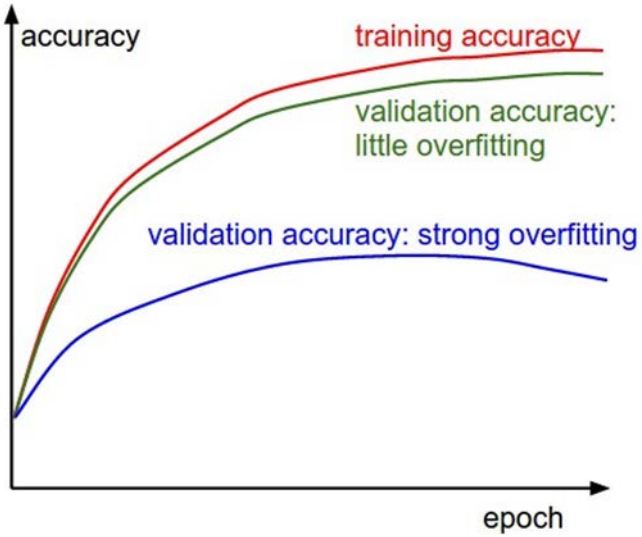
- RMSprop
  - Faster than SGD but slower than Adam
  - More stable than Adam
- Adam & variants (AdamW)
  - Most popular for its ease of use

People find simple SGD with momentum and decay to perform better (with proper tuning)

More details see
http://ruder.io/optimizing-gradient-descent/index.html#whichoptimizertochoose
https://towardsdatascience.com/why-adamw-matters-736223f31b5d

Overfitting

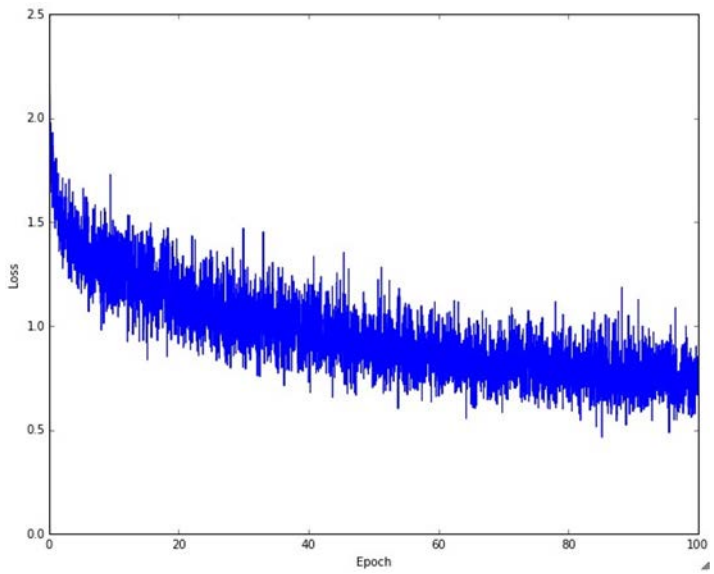
- You can keep doing back propagation forever!
- The training loss will always go down
- But it overfits
- Need to monitor performance on a held out set
- Stop or decrease learning rate when overfit happens



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



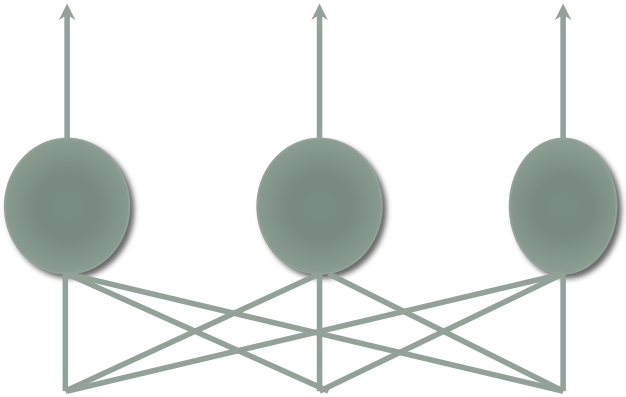
http://cs231n.github.io/neural-networks-3/

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

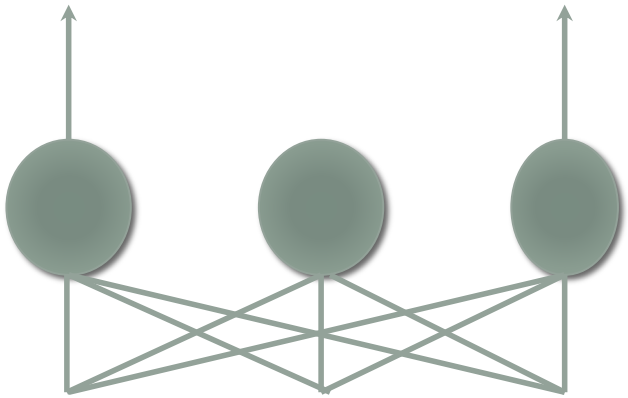


Model

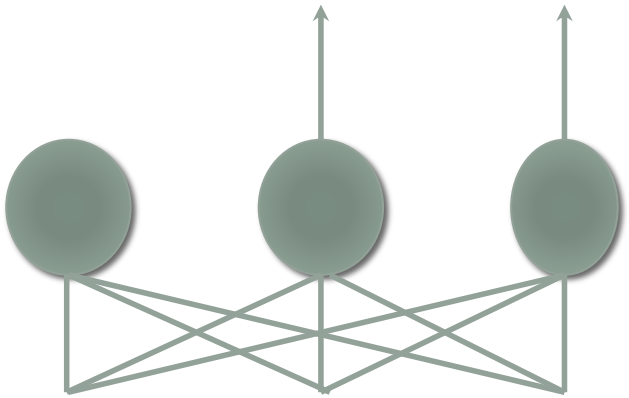
Dropout training time

Drop out rate 0.33

Randomly removed outputs
for each training sample



x1

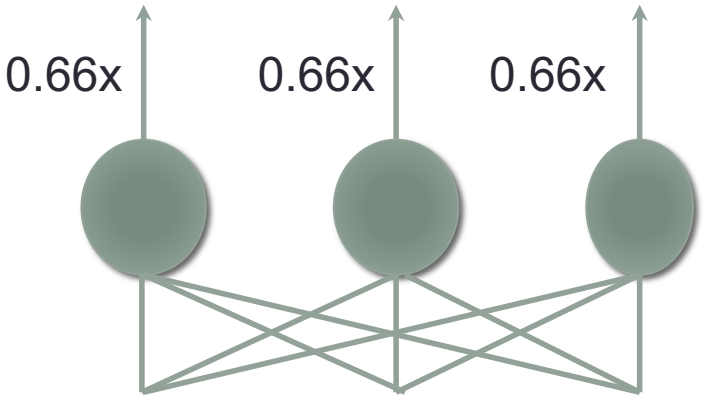


x2

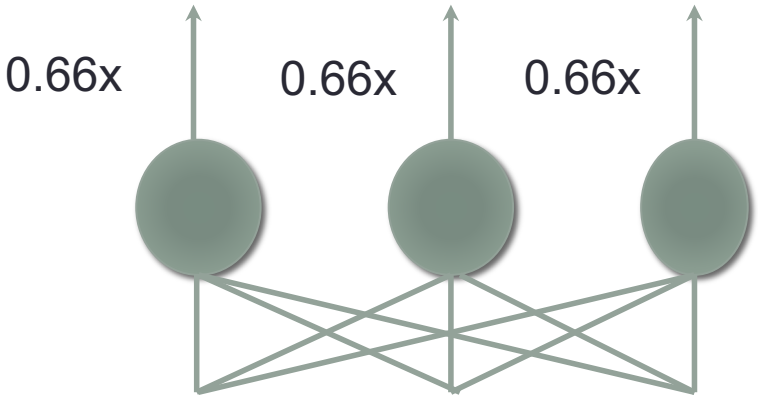
Dropout test time

Dropout rate 0.33

Scale outputs so the output contribution is around the same

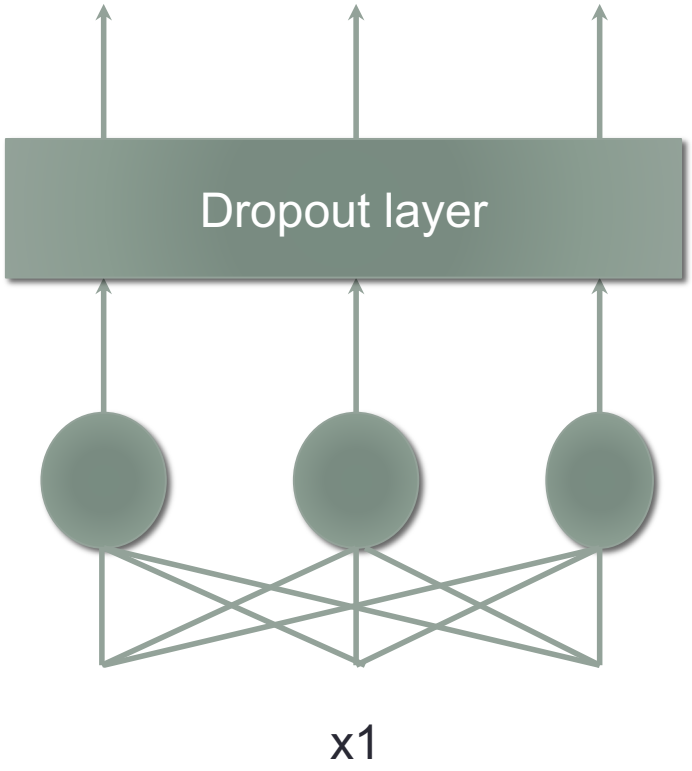


x1



x2

Dropout implementation

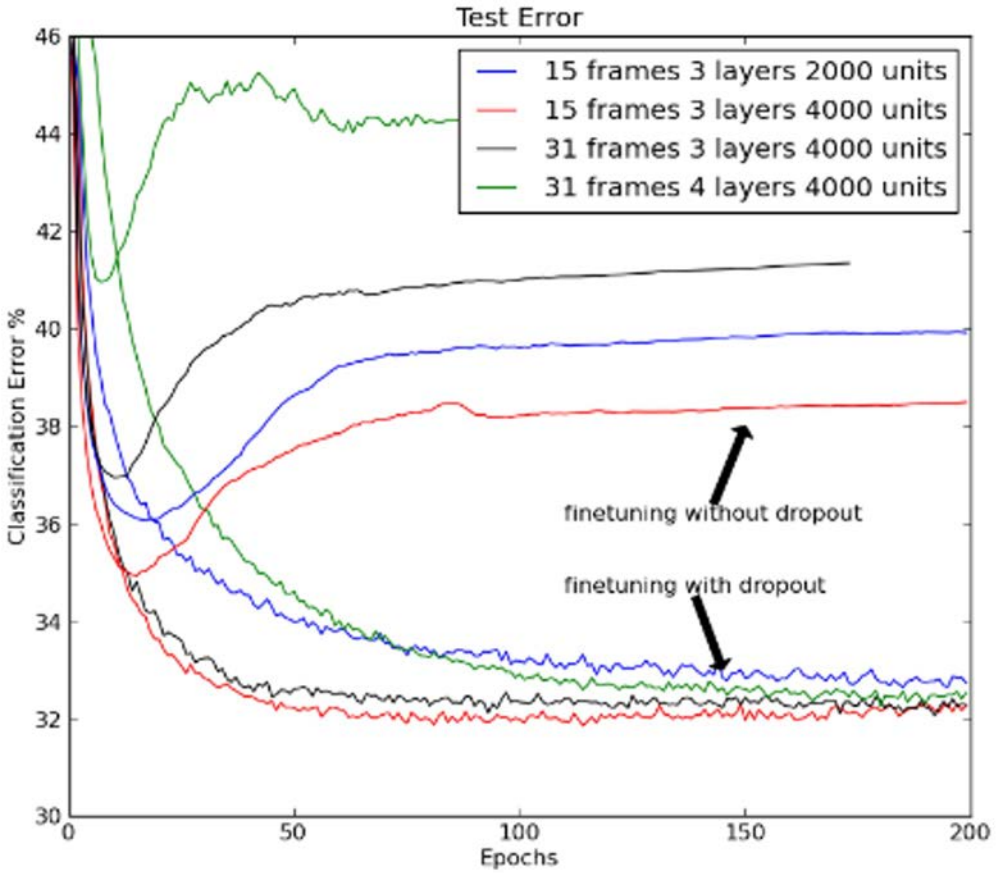


Dropout layer
Just another layer that drops inputs

Inverted dropout
A variant of dropout that scales the dropout at training time, so that you don't have to scale at test time.

Dropout on TIMIT

- A phoneme recognition task



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

Batch normalization

- Recent technique for (implicit) regularization
- Normalize every mini-batch at various batch norm layers to standard Gaussian (different from global normalization of the inputs)
- Place batch norm layers before non-linearities
- Faster training and better generalizations

For each mini-batch that goes through batch norm

1. Normalize by the mean and variance of the mini-batch for each dimension
2. Shift and scale by learnable parameters

Replaces dropout in some networks

https://arxiv.org/abs/1502.03167

$$\hat{x} = \frac{x - \mu_b}{\sigma_b}$$

$$y = \alpha\hat{x} + \beta$$

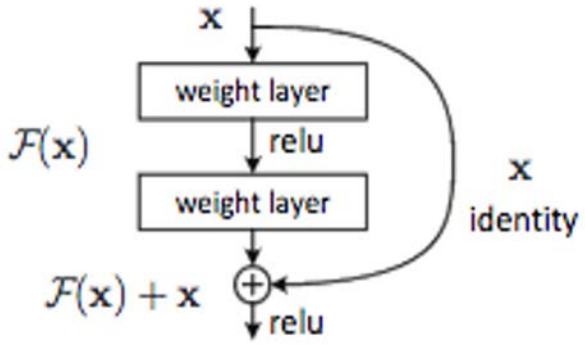
Dropout vs batchnorm

- You can add dropout in the hidden layers (0-0.5)
- Or input layers (0-0.2 is typical)
  - “Noising” the inputs, data augmentation
- Dropout in computer vision
  - use batchnorm instead
- Dropout in NLP
  - Usually works better than Batchnorm for NLP in simple architectures
  - Drop full words at the embedding
  - Recurrent dropout http://arxiv.org/abs/1512.05287
  - Recent works (transformers) find batchnorm to be better than dropout
  - For seq2seq models, layer norm is popular

Vanishing/Exploding gradient

- Backprop introduces many multiplications down chain
- The gradient value gets smaller and smaller
  - The deeper the network the smaller the gradient in the lower layers
  - Lower layers changes too slowly (or not at all)
  - Hard to train very deep networks (>6 layers)
- The opposite can also be true. The gradient explodes from repeated multiplication
  - Put a maximum value for the gradient (Gradient clipping)

- How to deal with this?
  - Residual connection



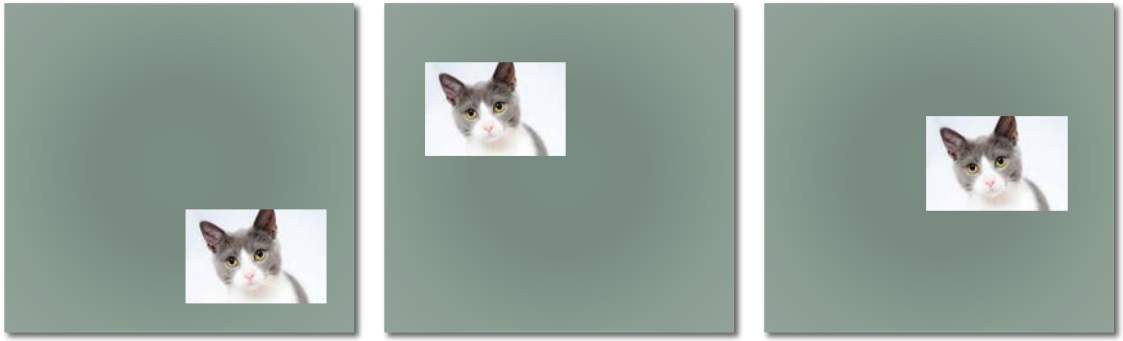
https://arxiv.org/pdf/1512.03385.pdf

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



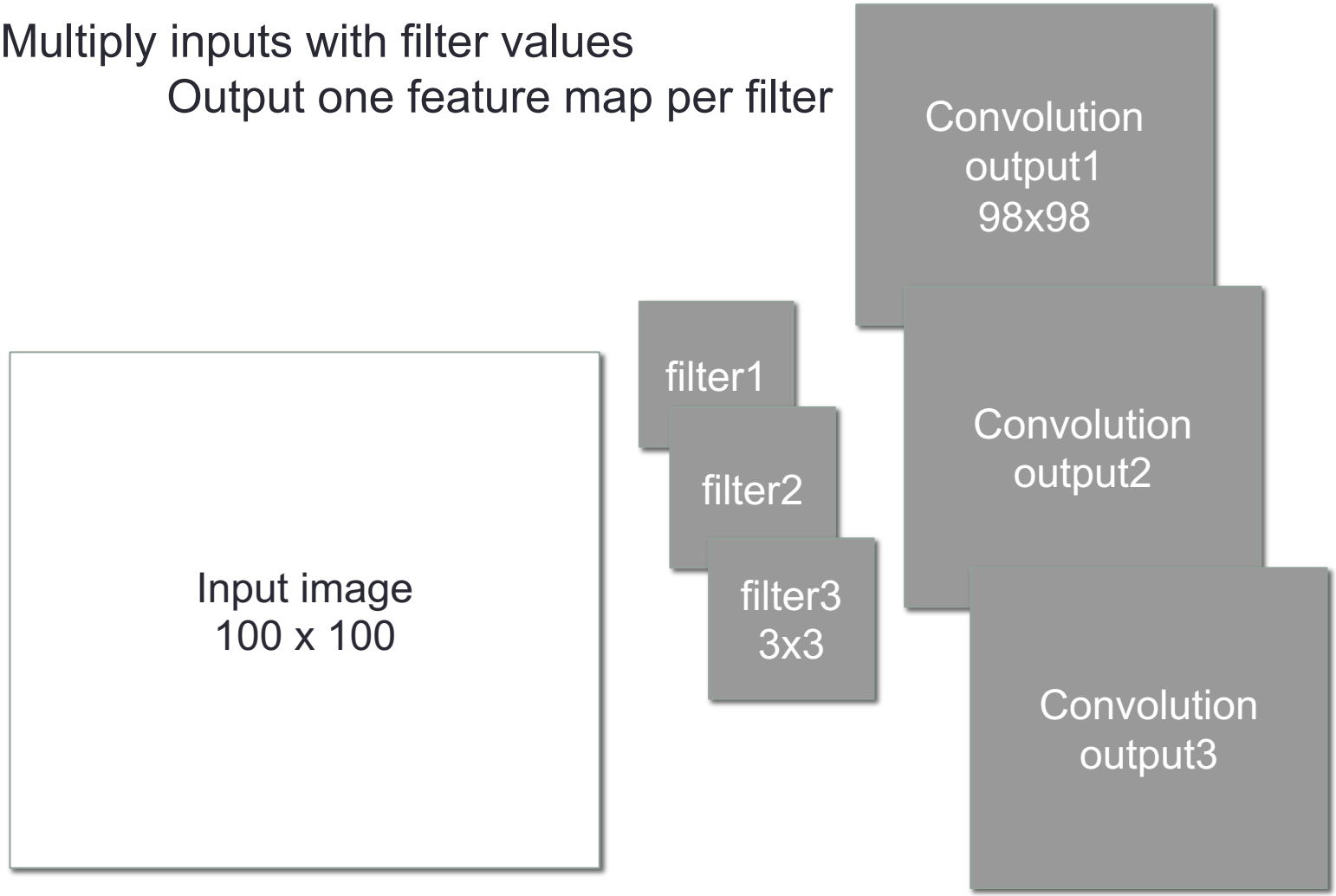
- Can we use the same parameters to learn that a cat exists regardless of location?

- 2 parts: convolutional layer and pooling layer

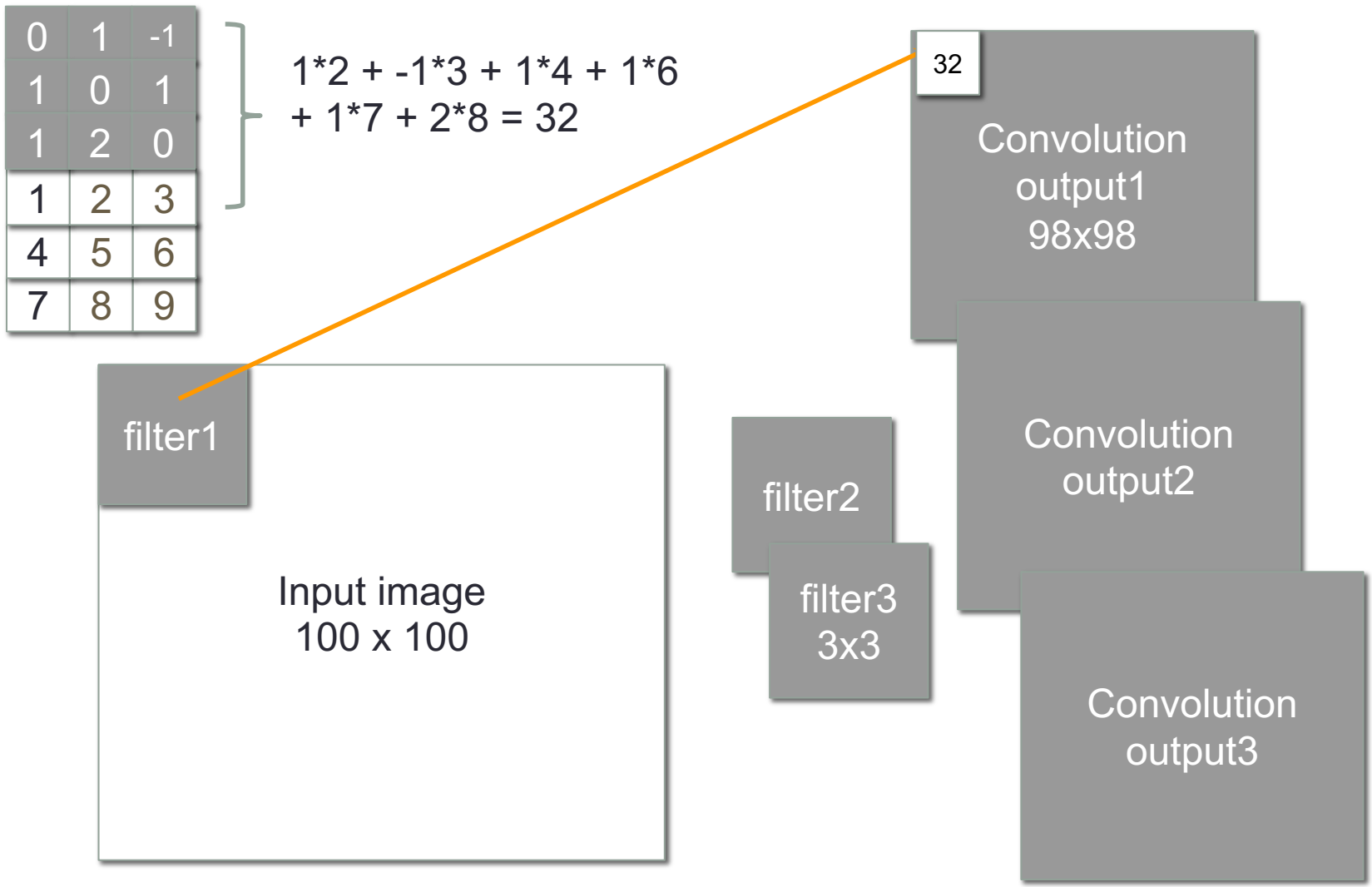
Convolutional filters

Multiply inputs with filter values

Output one feature map per filter

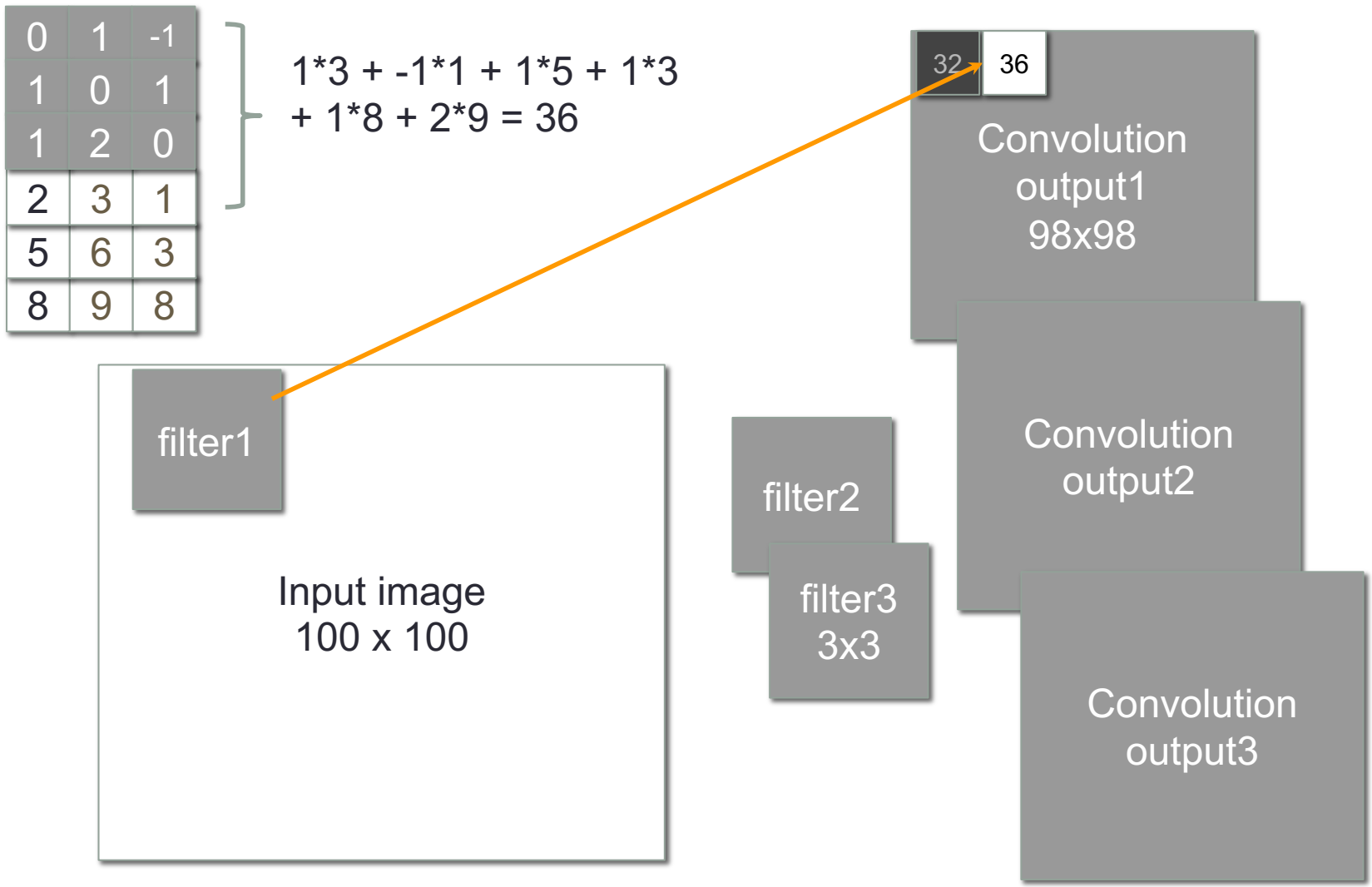


Convolutional filters

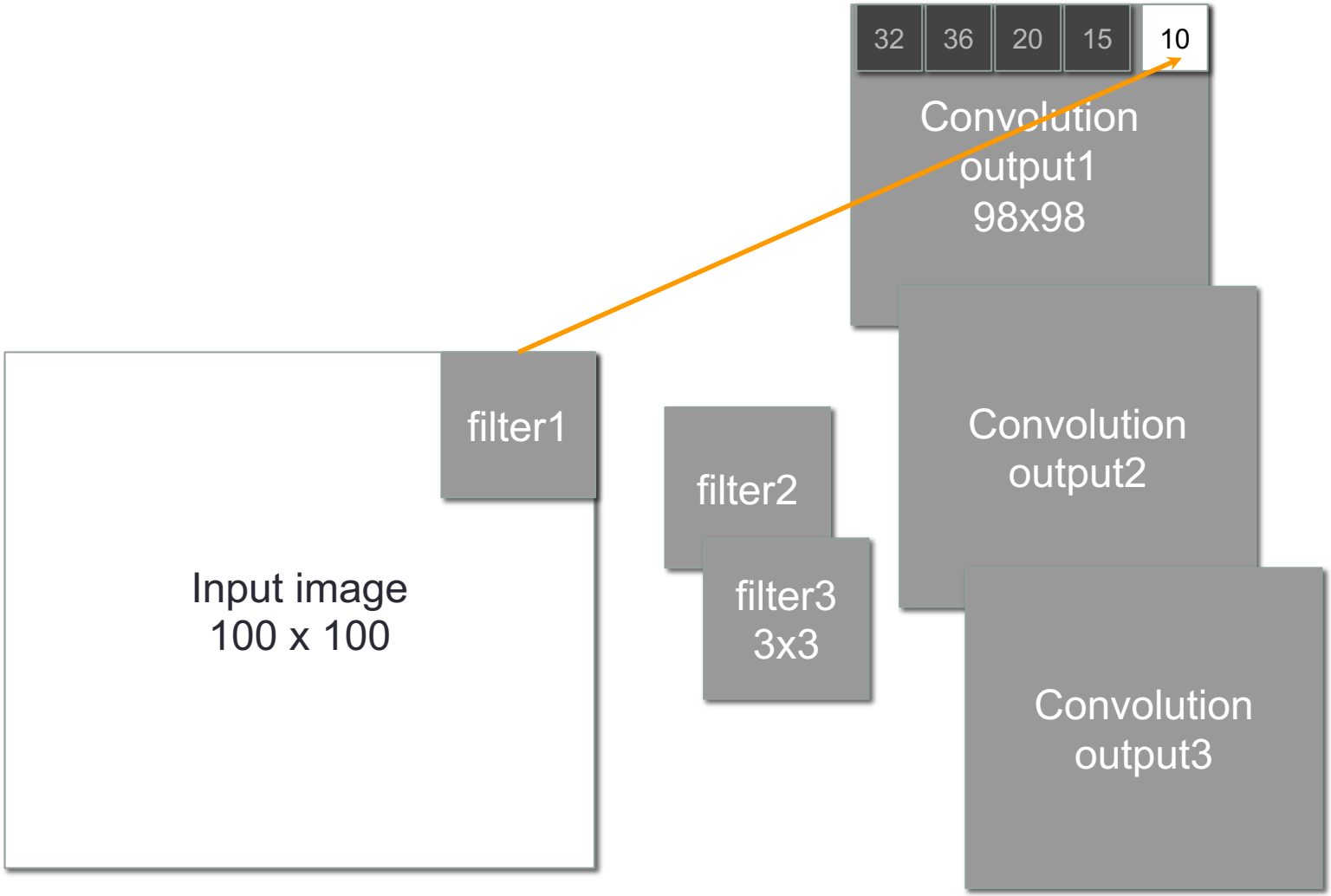


Convolutional filters

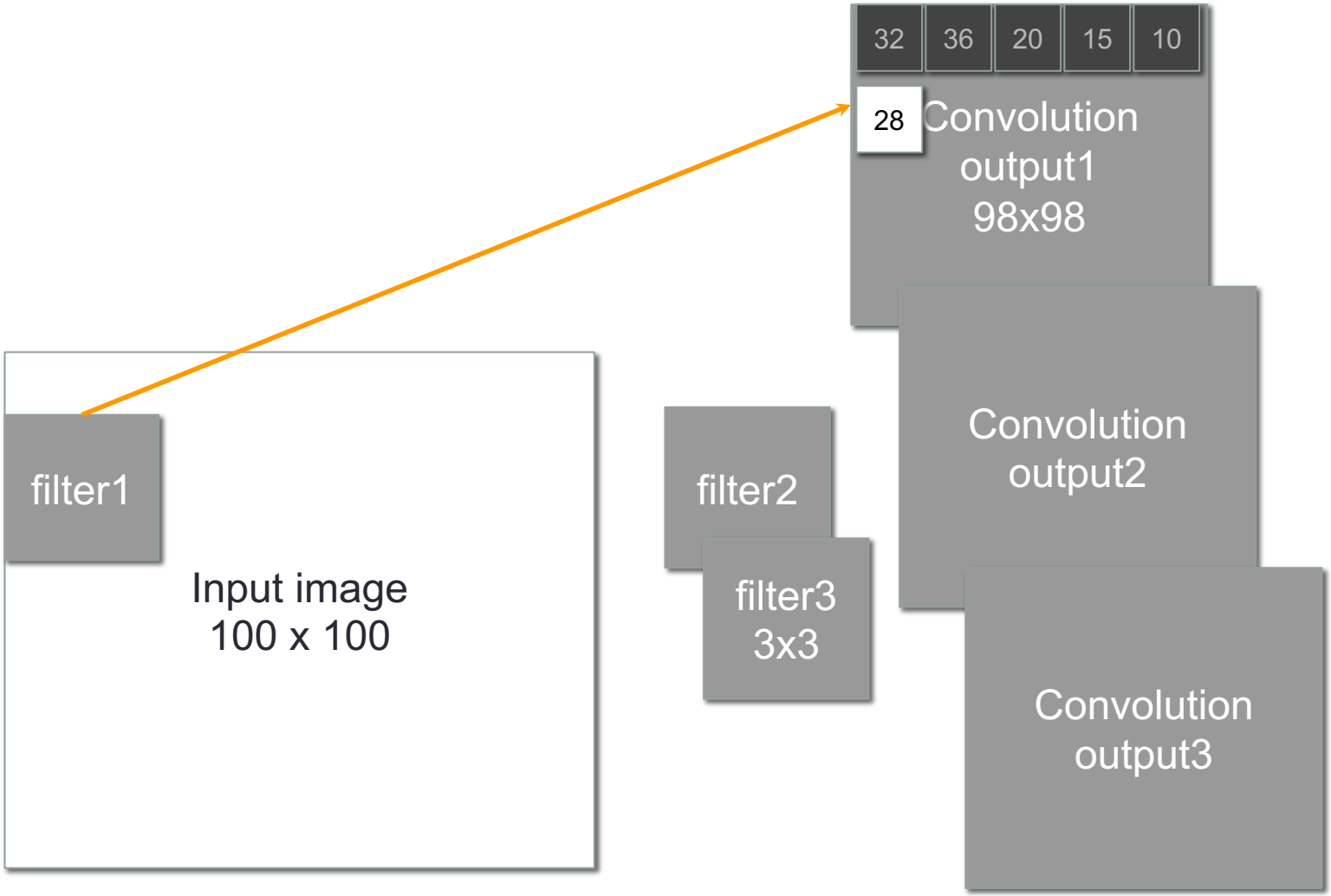
Stride of 1



Convolutional filters

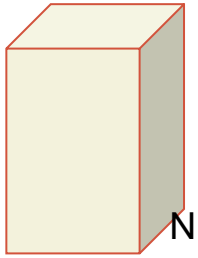


Convolutional filters



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

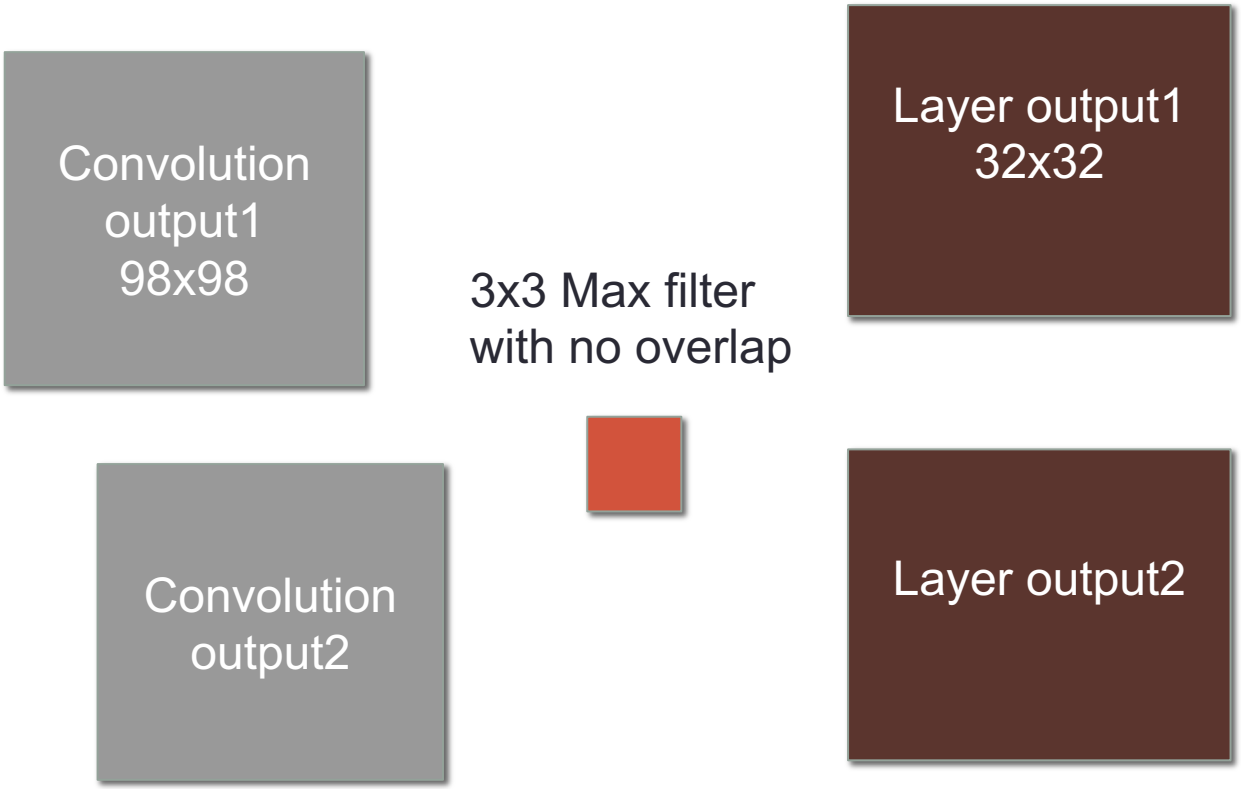
Input image
100 x 100

Convolution
output2

Convolution
output3

Pooling/subsampling

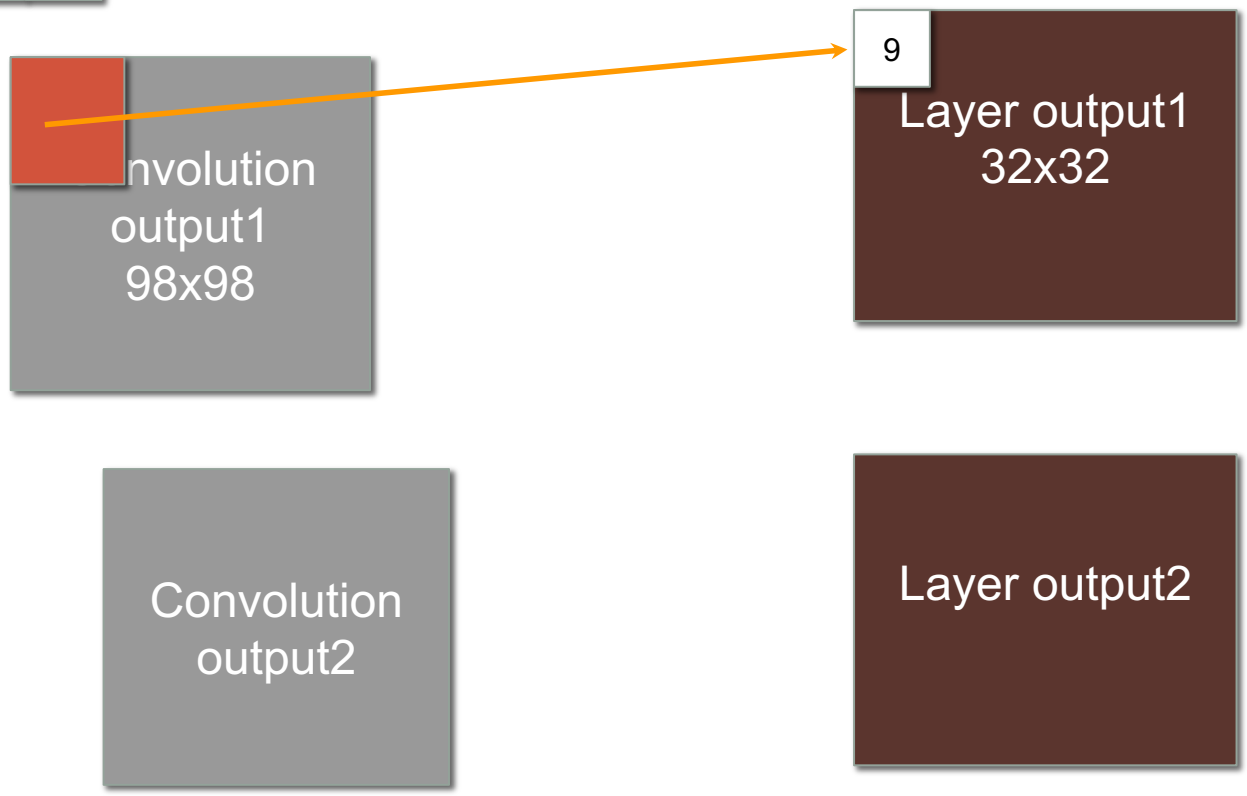
Reduce dimension of the feature maps



Pooling/subsampling

1	2	3
4	5	6
7	8	9

Max = 9

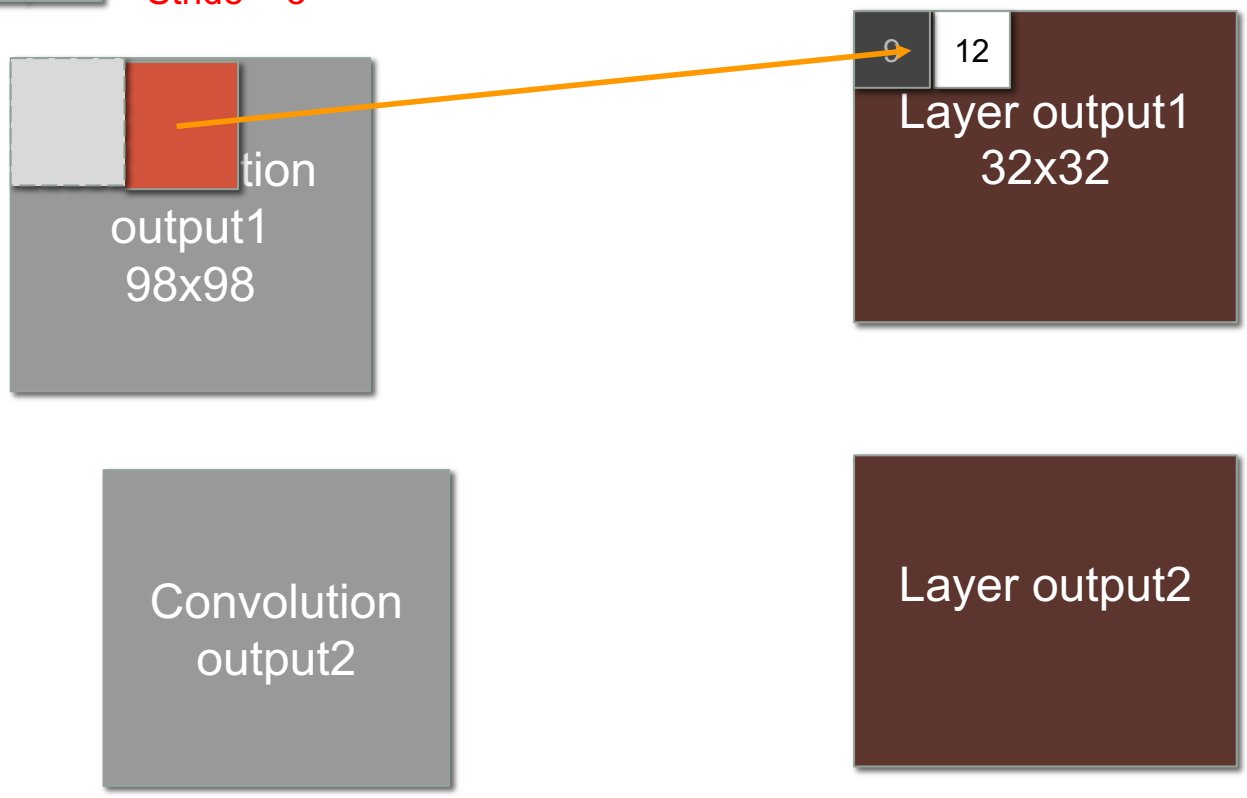


Pooling/subsampling

5	2	1
5	7	1
9	5	12

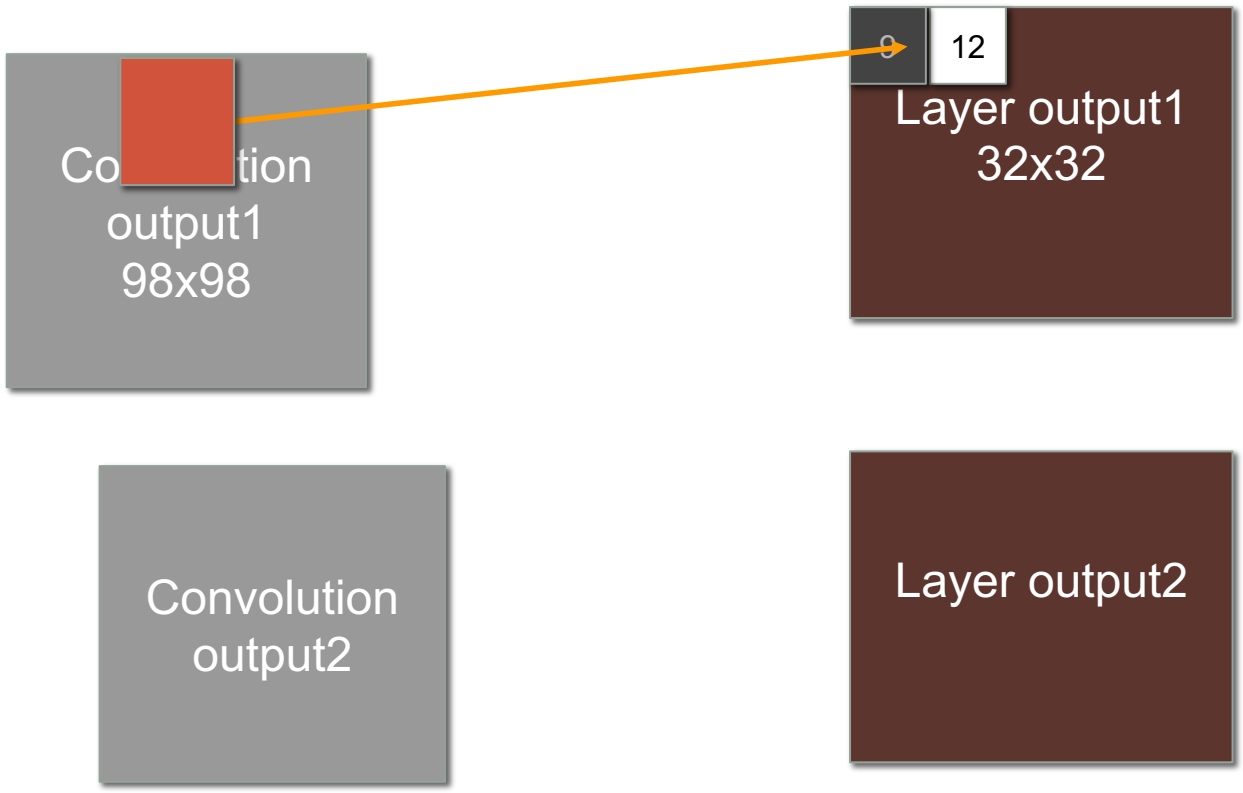
Max = 12

Stride = 3



Pooling/subsampling

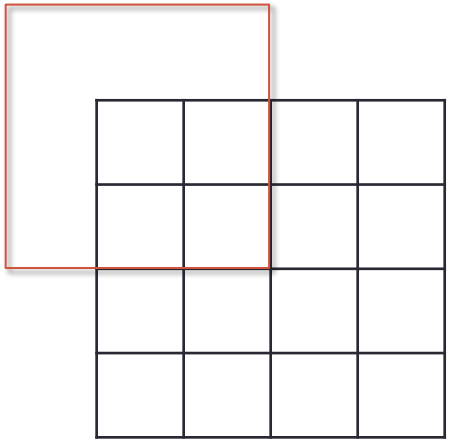
Can use other functions besides max
Example, average



Convolution puzzle

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

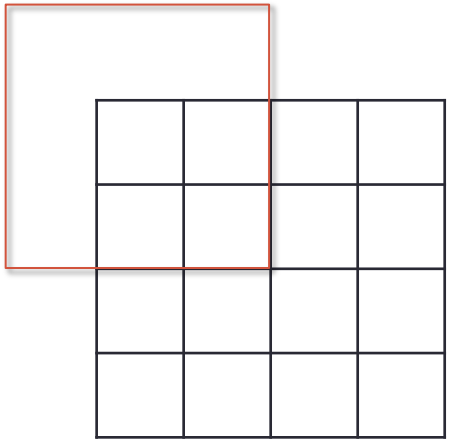
What is the output size?

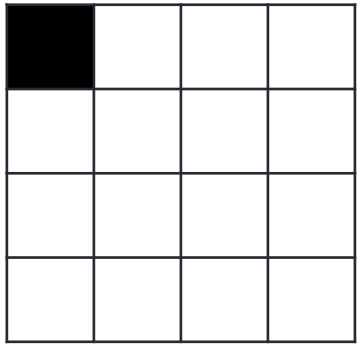


Convolution puzzle

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

What is the output size?

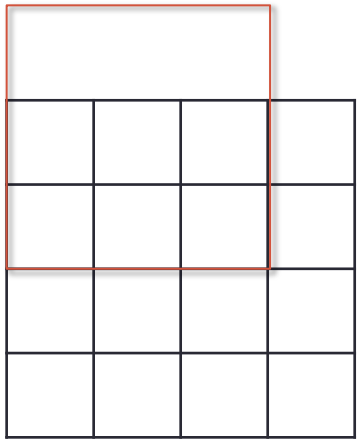


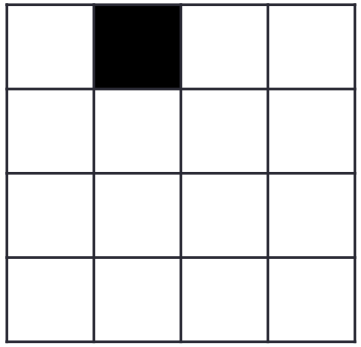


Convolution puzzle

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

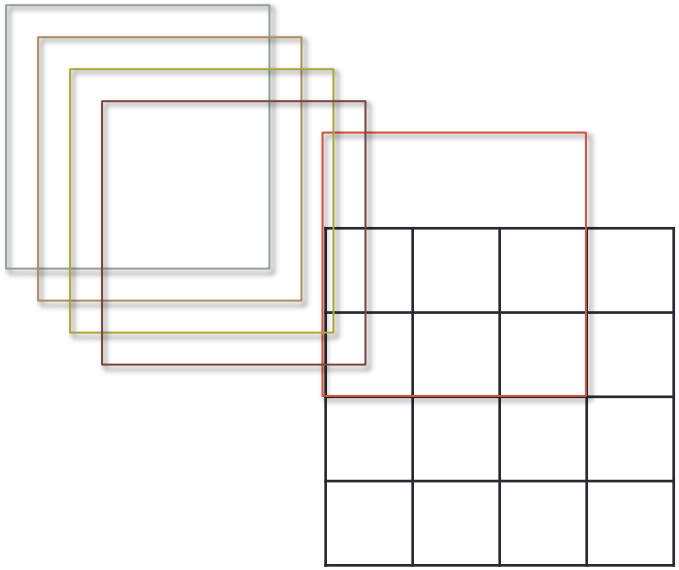
What is the output size?

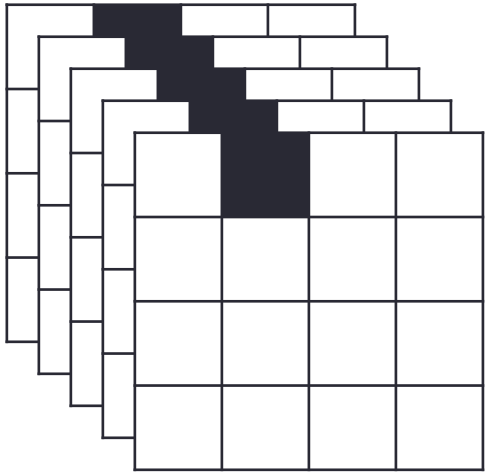




Convolution puzzle

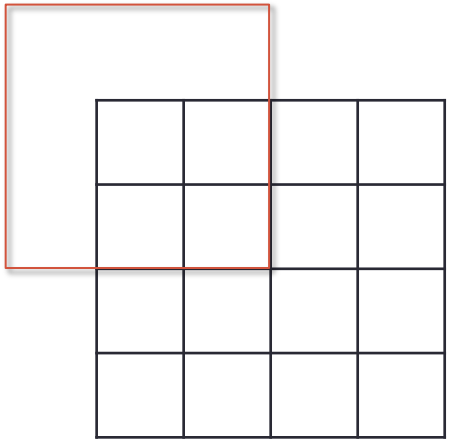
5 filters 3x3 filter pad, stride 1, pad 1





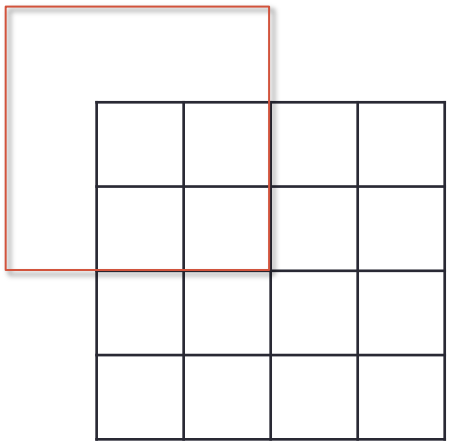
Convolution puzzle

3x3 filter pad, stride 2, pad 1



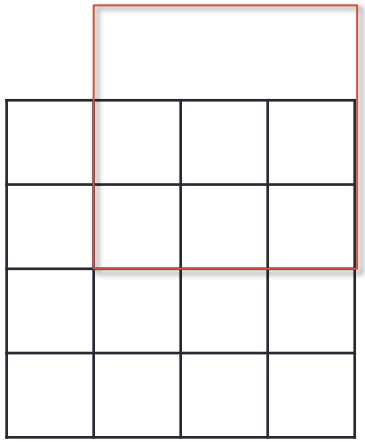
Convolution puzzle

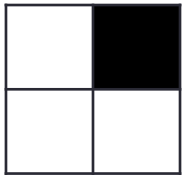
3x3 filter pad, stride 2, pad 1



Convolution puzzle

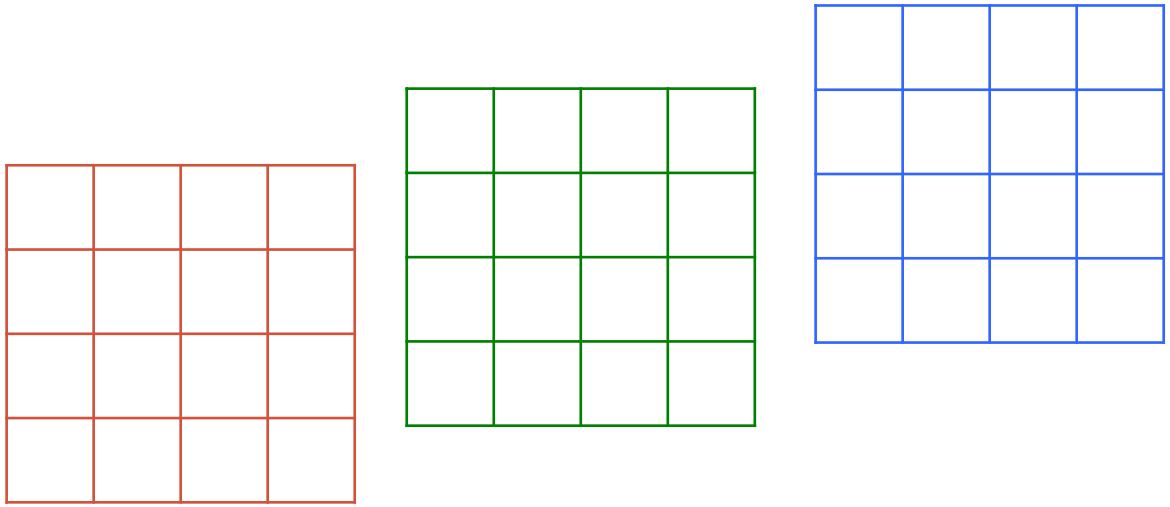
3x3 filter pad, stride 2, pad 1





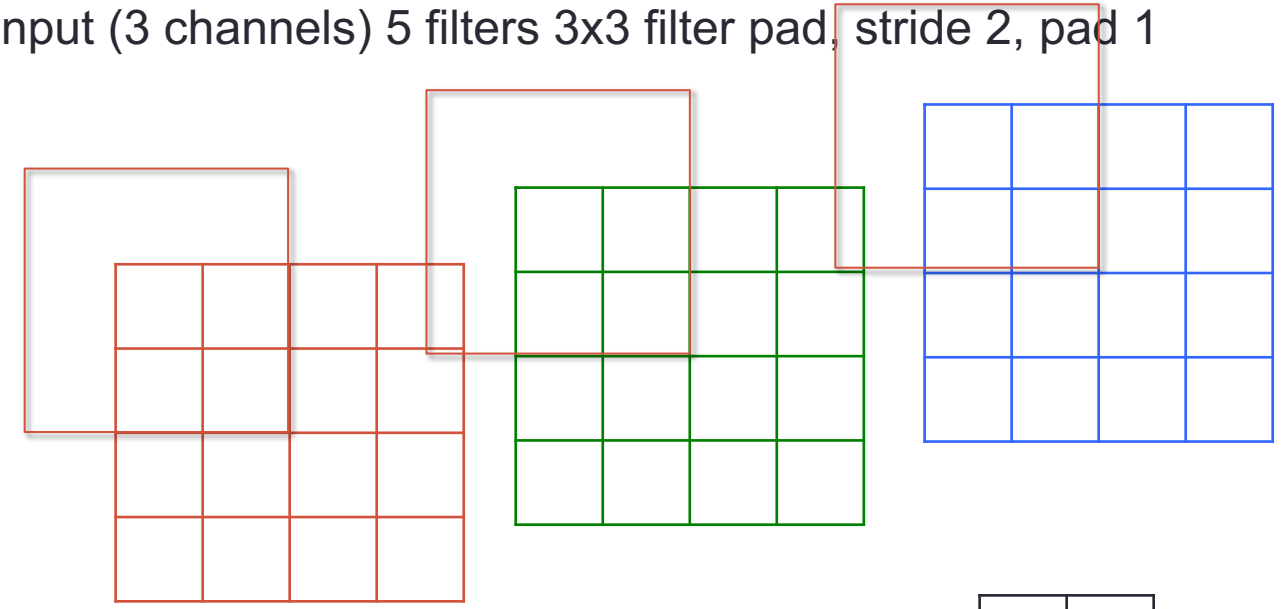
Convolution puzzle

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

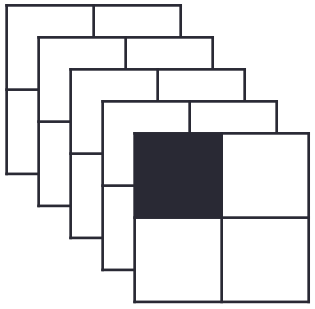


Convolution puzzle

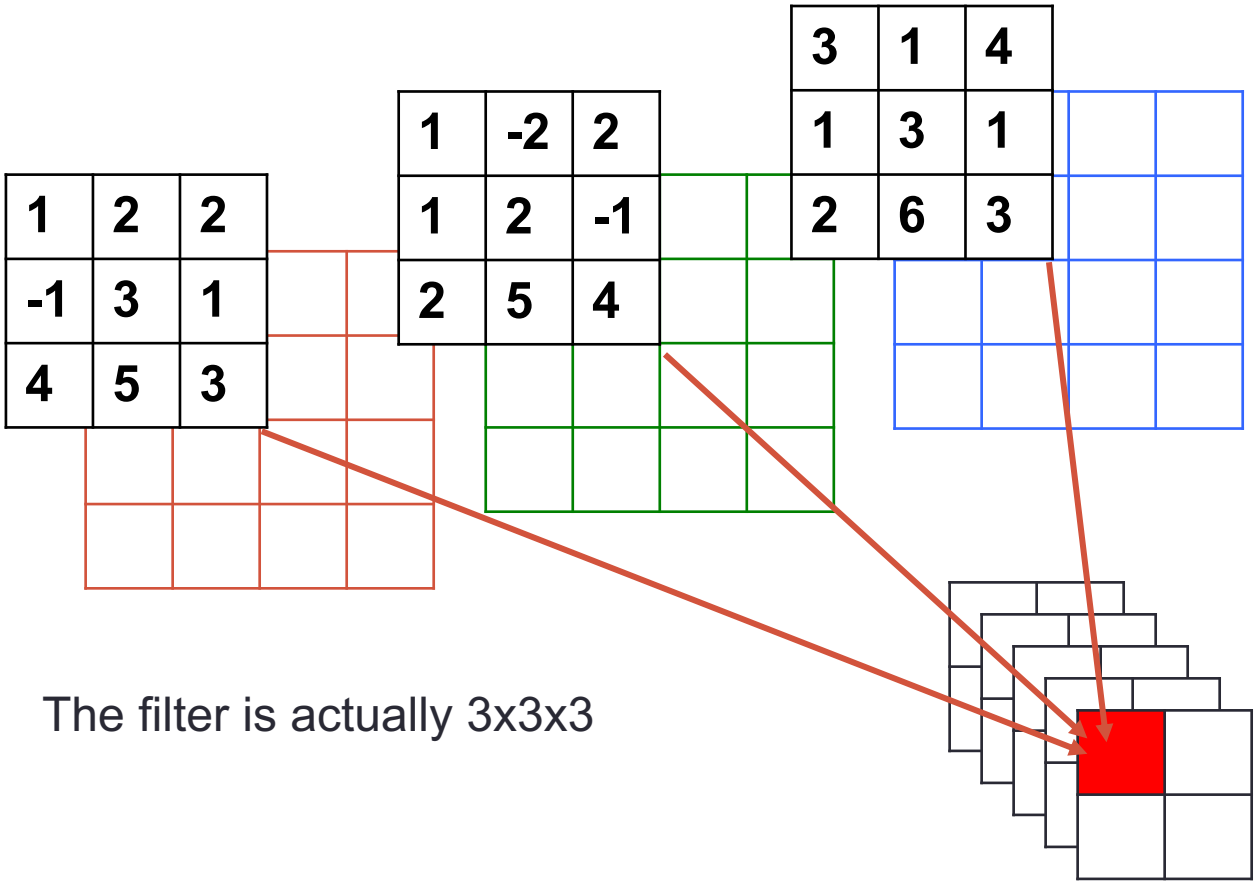
RGB input (3 channels) 5 filters 3x3 filter pad, stride 2, pad 1



The filter is actually 3x3x3



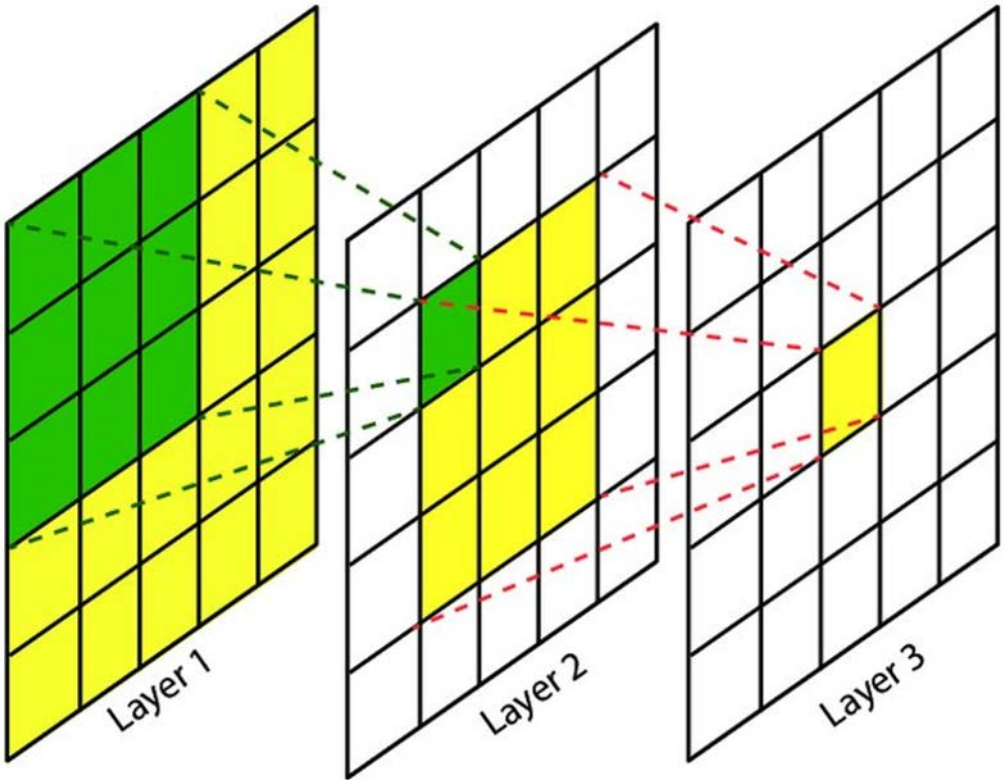
Convolution puzzle



The filter is actually 3x3x3

Receptive field

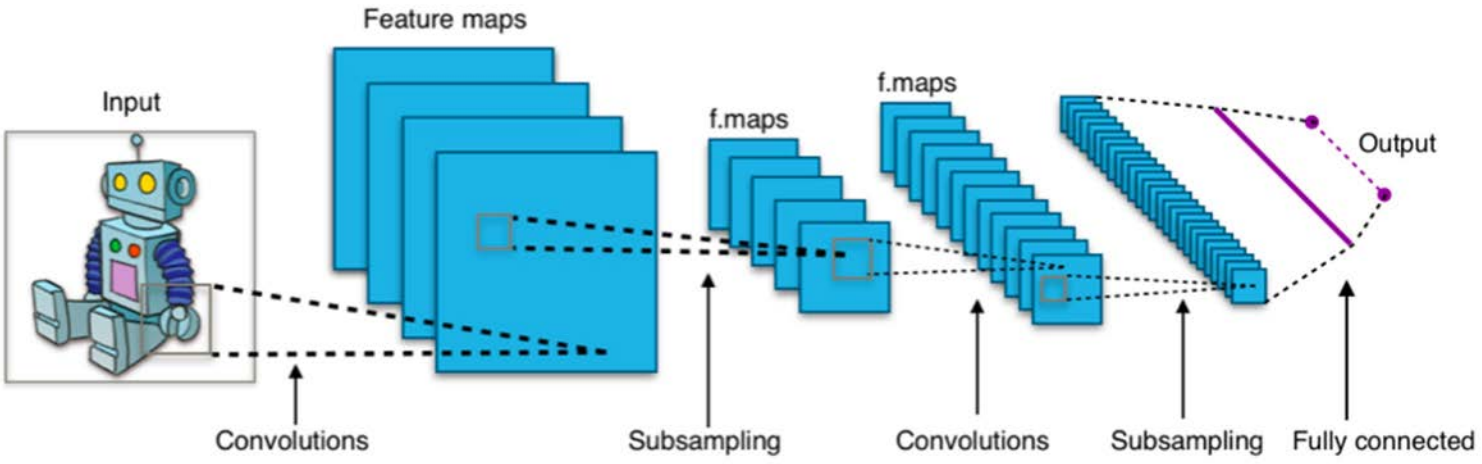
You might want to consider about how large is your pattern when designing your network



source image

CNN overview

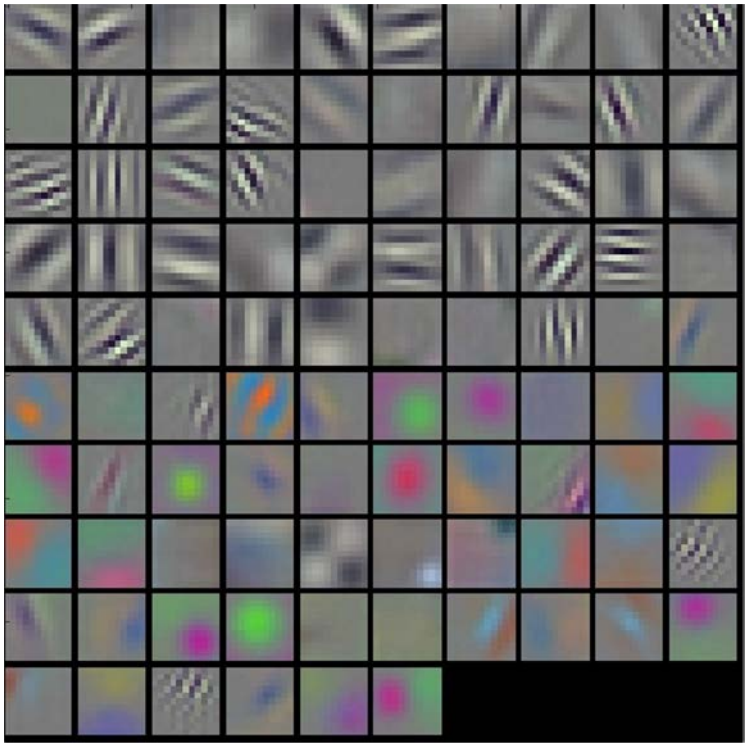
- Filter size, number of filters, filter shifts, and pooling rate are all parameters
- Usually followed by a fully connected network at the end
  - CNN is good at learning low level features
  - DNN combines the features into high level features and classify



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

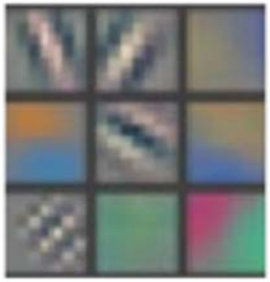
Visualizing convolutional layers

- We can visualize the weights of the filter
- “Matched filters”

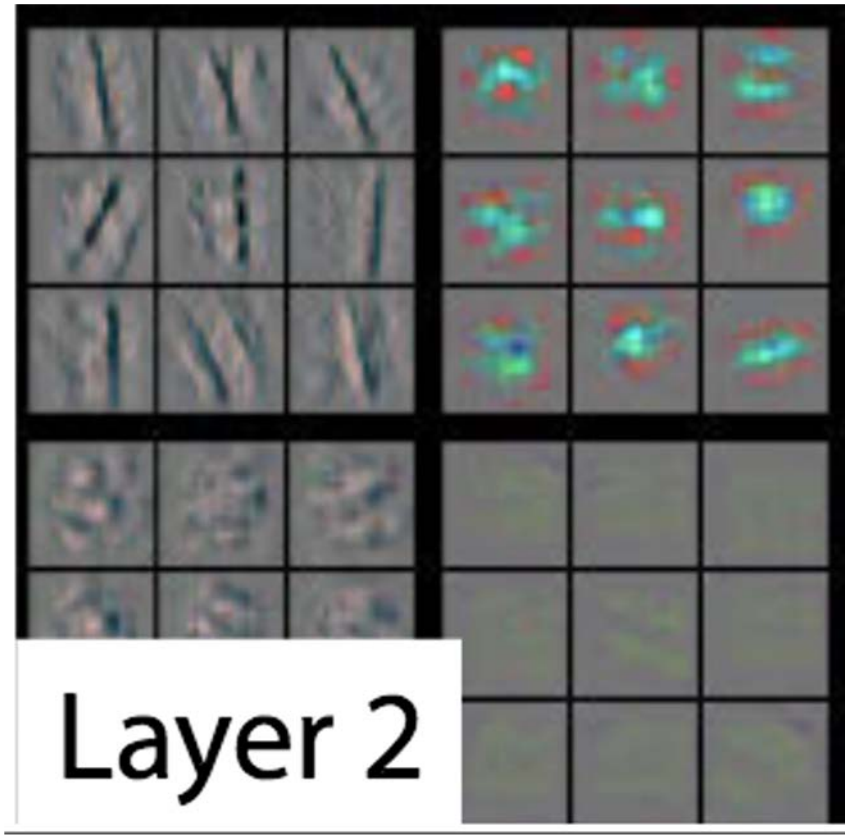


http://cs231n.github.io/understanding-cnn/

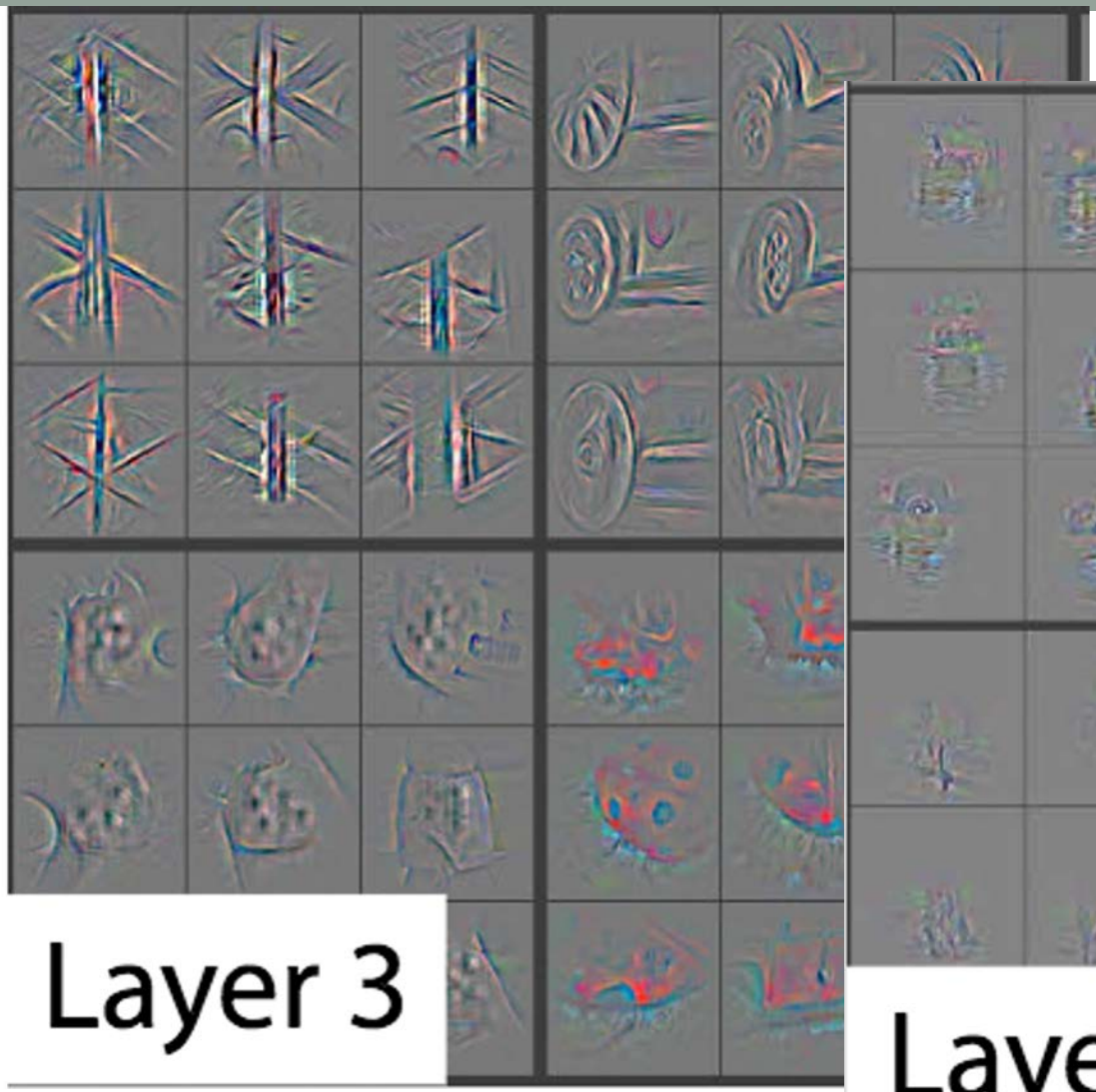
Higher layer captures higher-level concepts

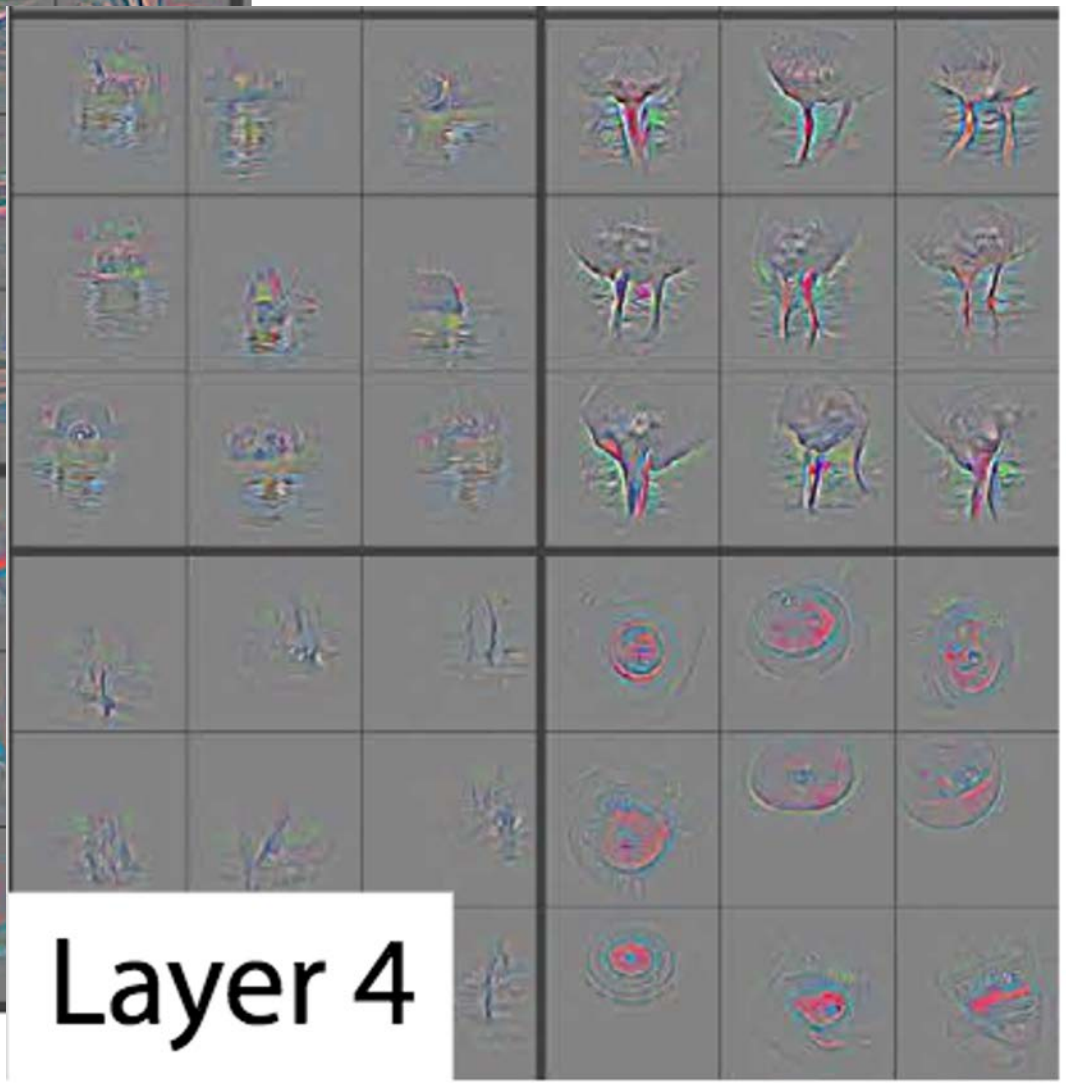


Layer 1



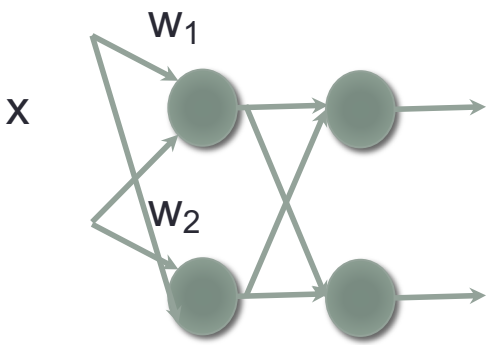
https://arxiv.org/pdf/1311.2901.pdf


Layer 3

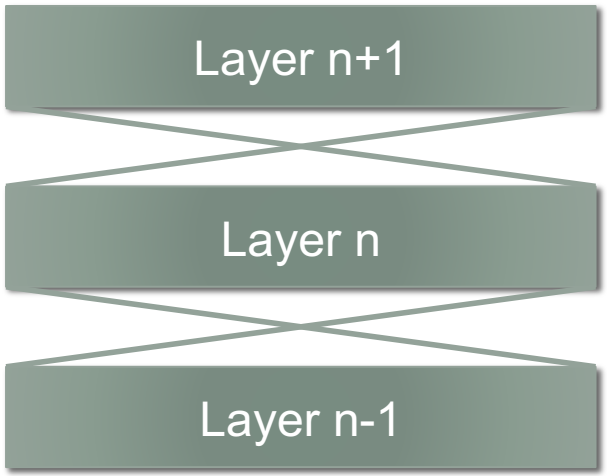

Layer 4

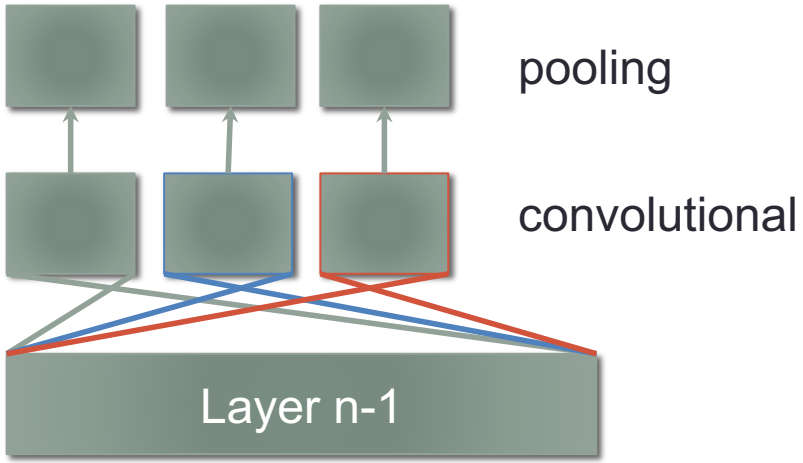
Parameter sharing in convolution neural networks

- $W^T x$



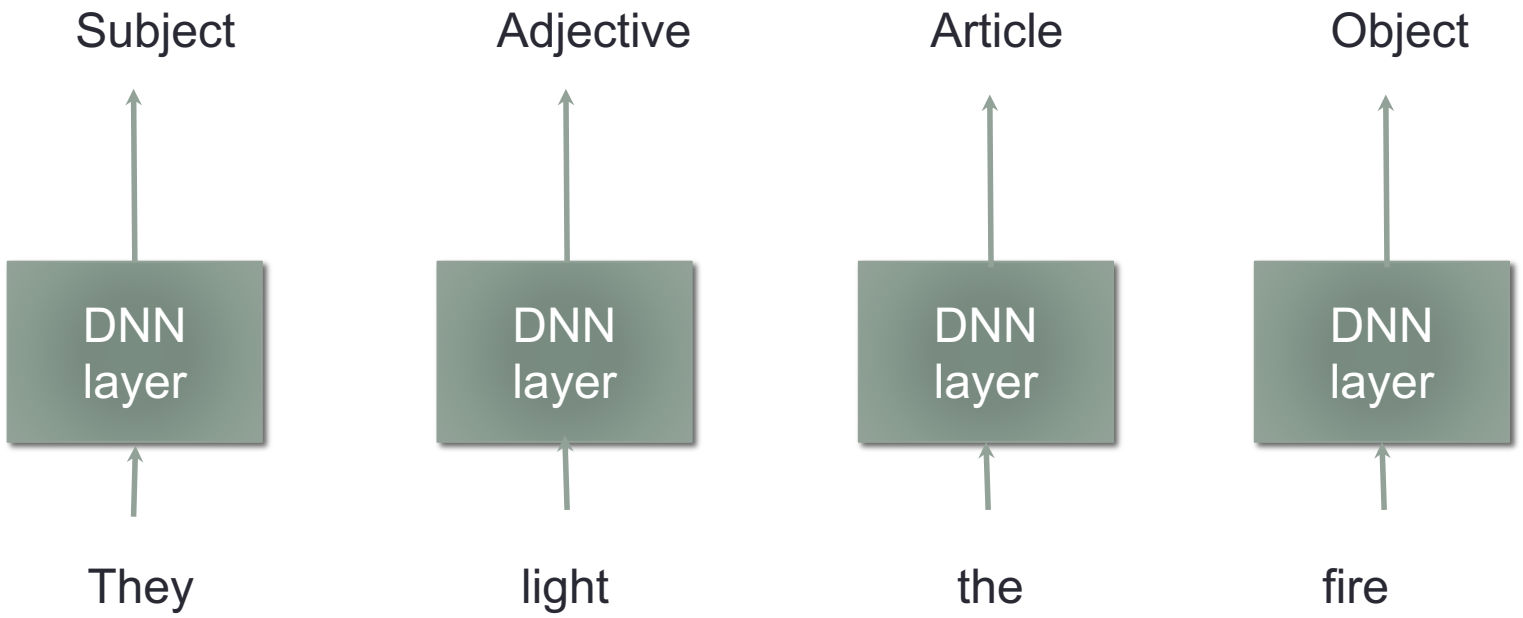
- Cats at different location might need two neurons for different locations in fully connect NNs.
- CNN shares the parameters in 1 filter
- The network is no longer fully connected





Recurrent neural network (RNN)

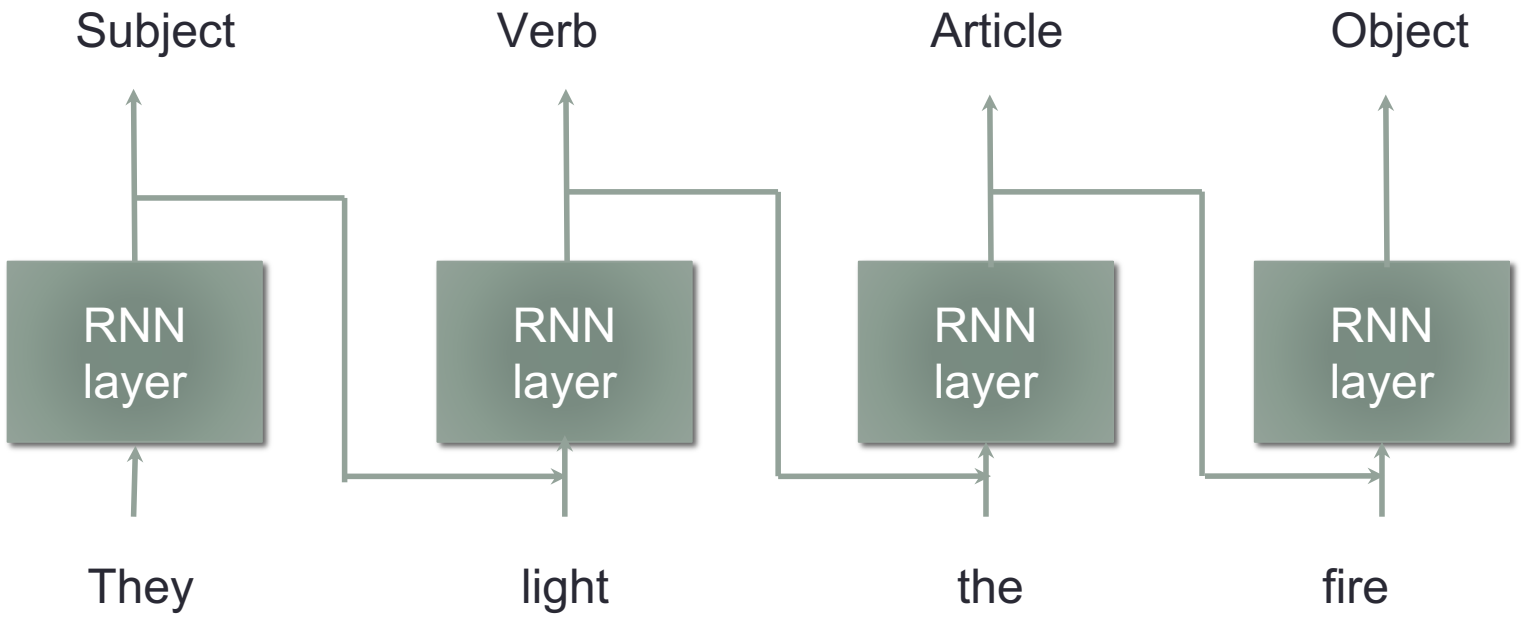
- DNN framework



Problem1: need a way to remember the past

Recurrent neural network (RNN)

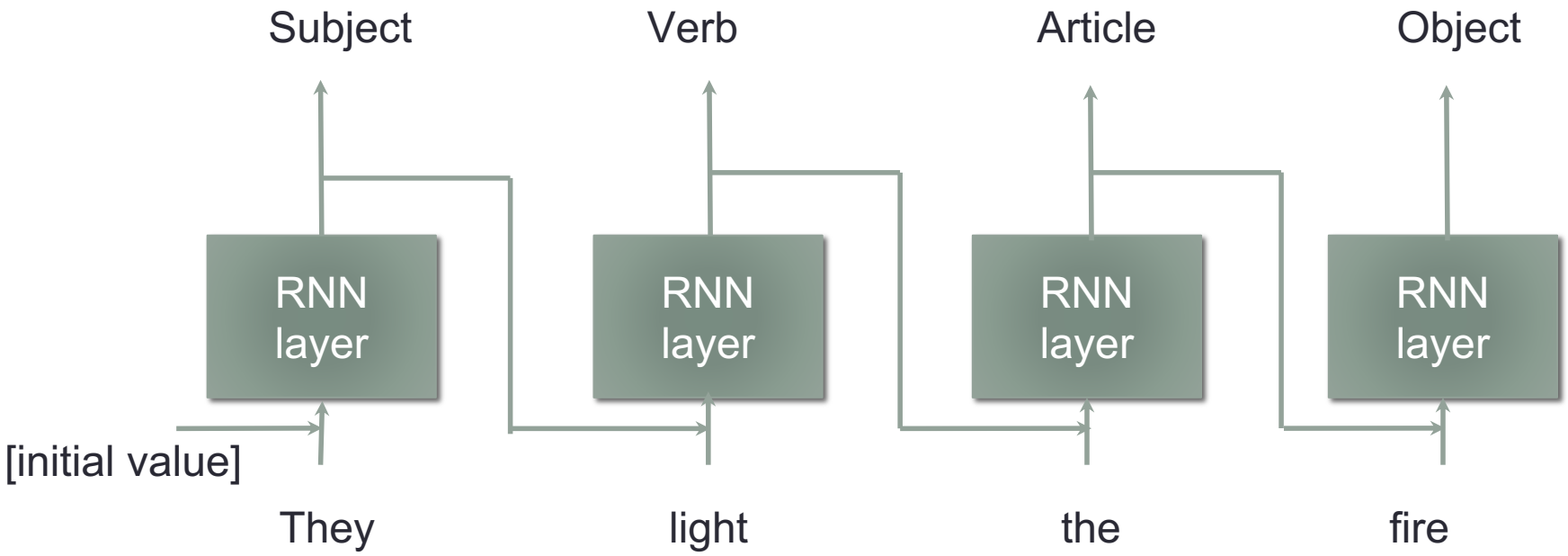
- RNN framework



Output of the layer encodes something meaningful about the past

Recurrent neural network (RNN)

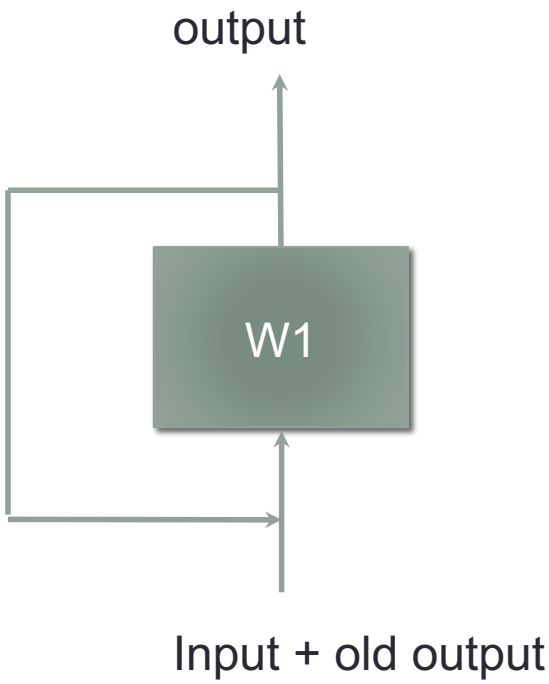
- RNN framework



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

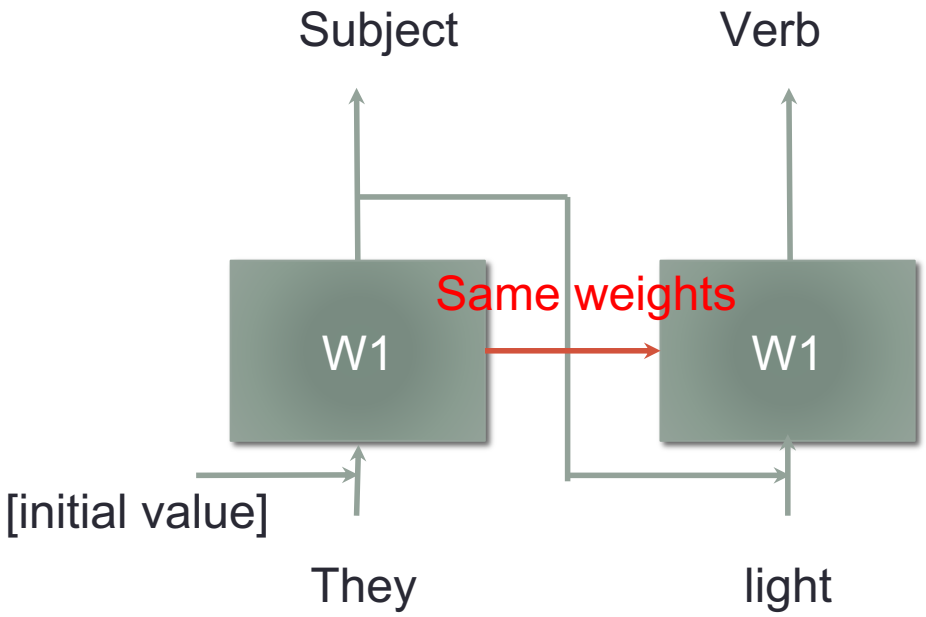
Recurrent neural network (RNN)

- Unrolling of a recurrent layer.



Recurrent neural network (RNN)

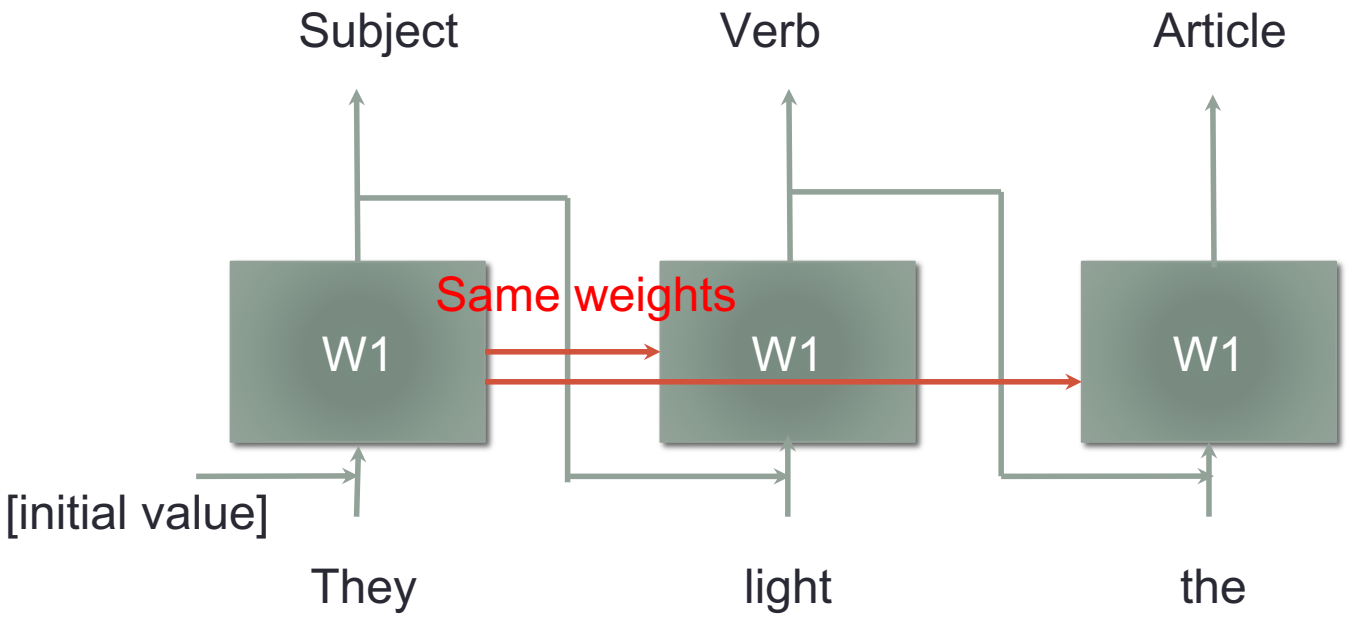
- Unrolling of a recurrent layer.



Parameter sharing

Recurrent neural network (RNN)

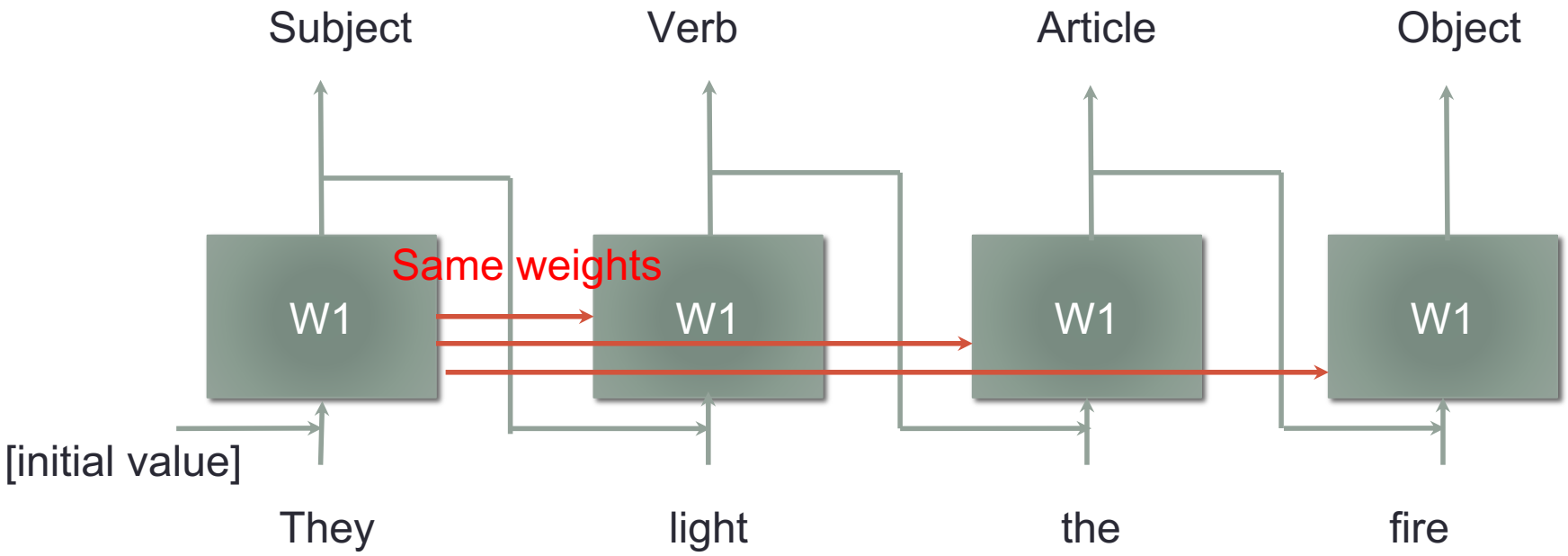
- Unrolling of a recurrent layer.



Parameter sharing

Recurrent neural network (RNN)

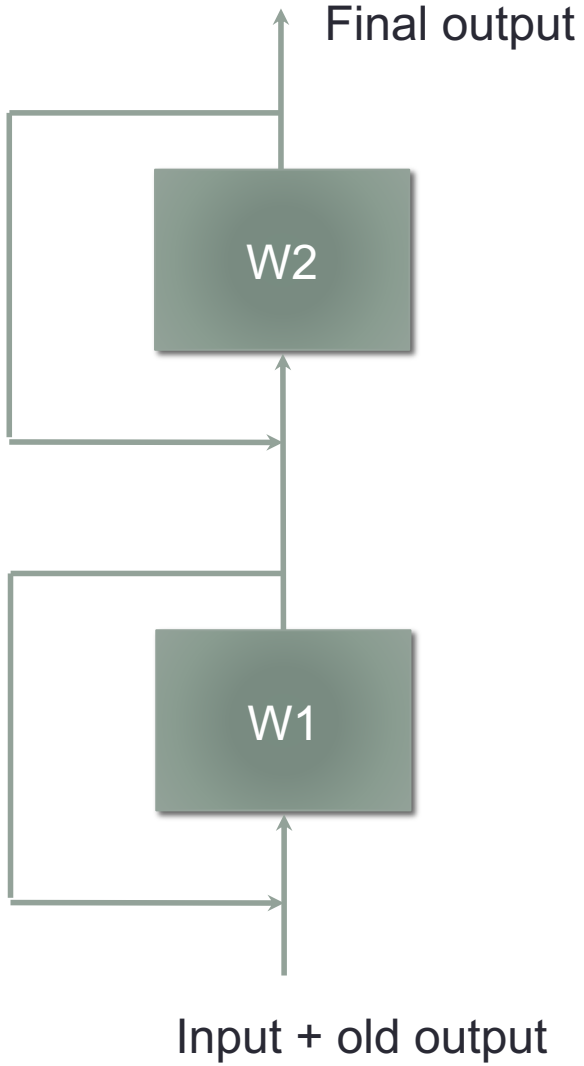
- Unrolling of a recurrent layer.



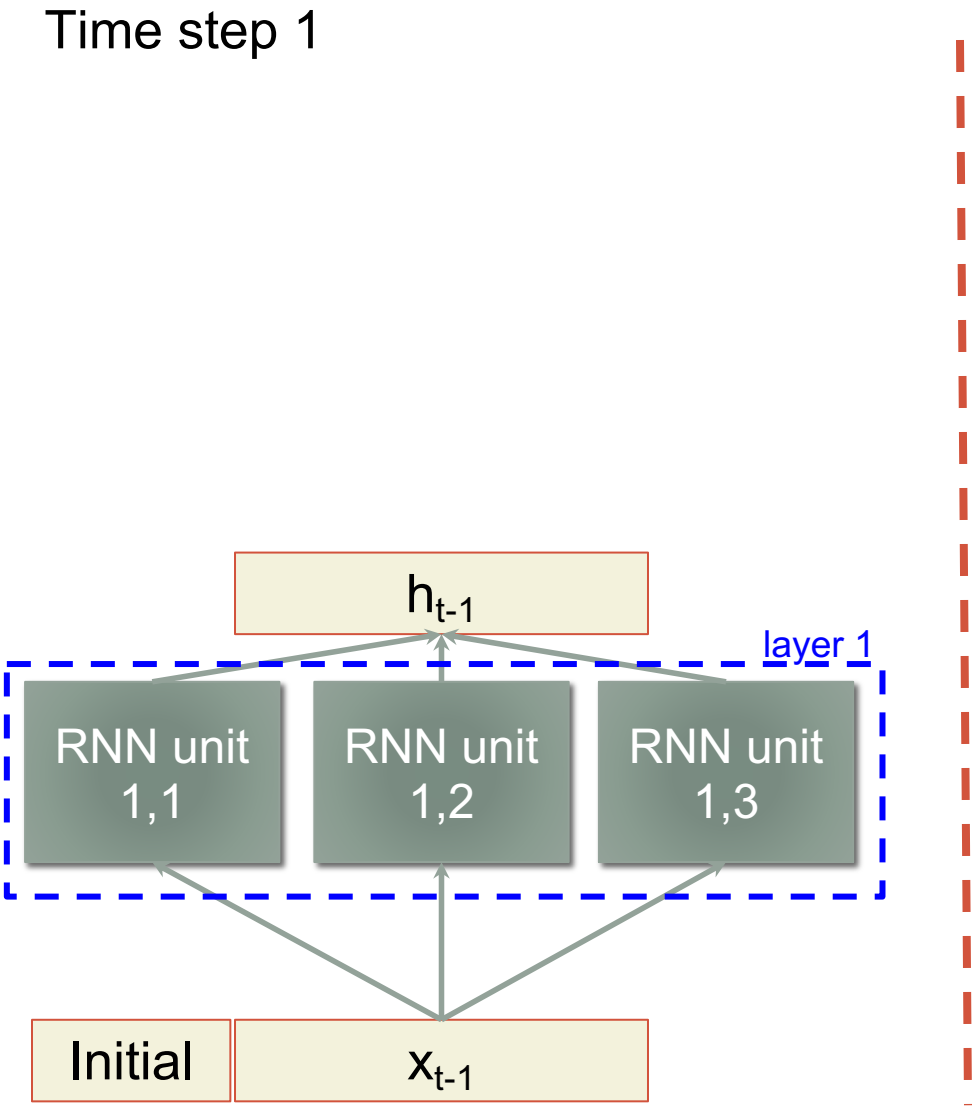
Parameter sharing

Recurrent neural network (RNN)

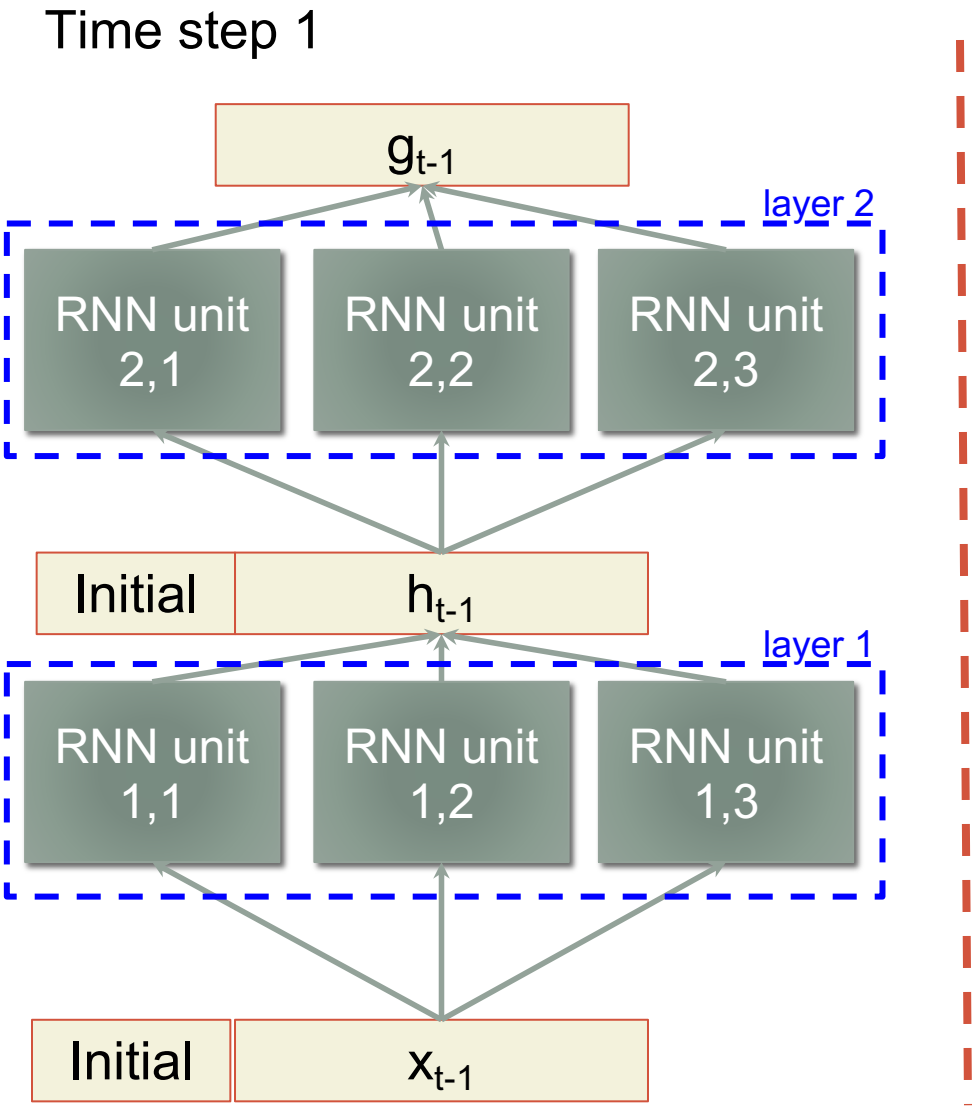
- Stacks of recurrent layer



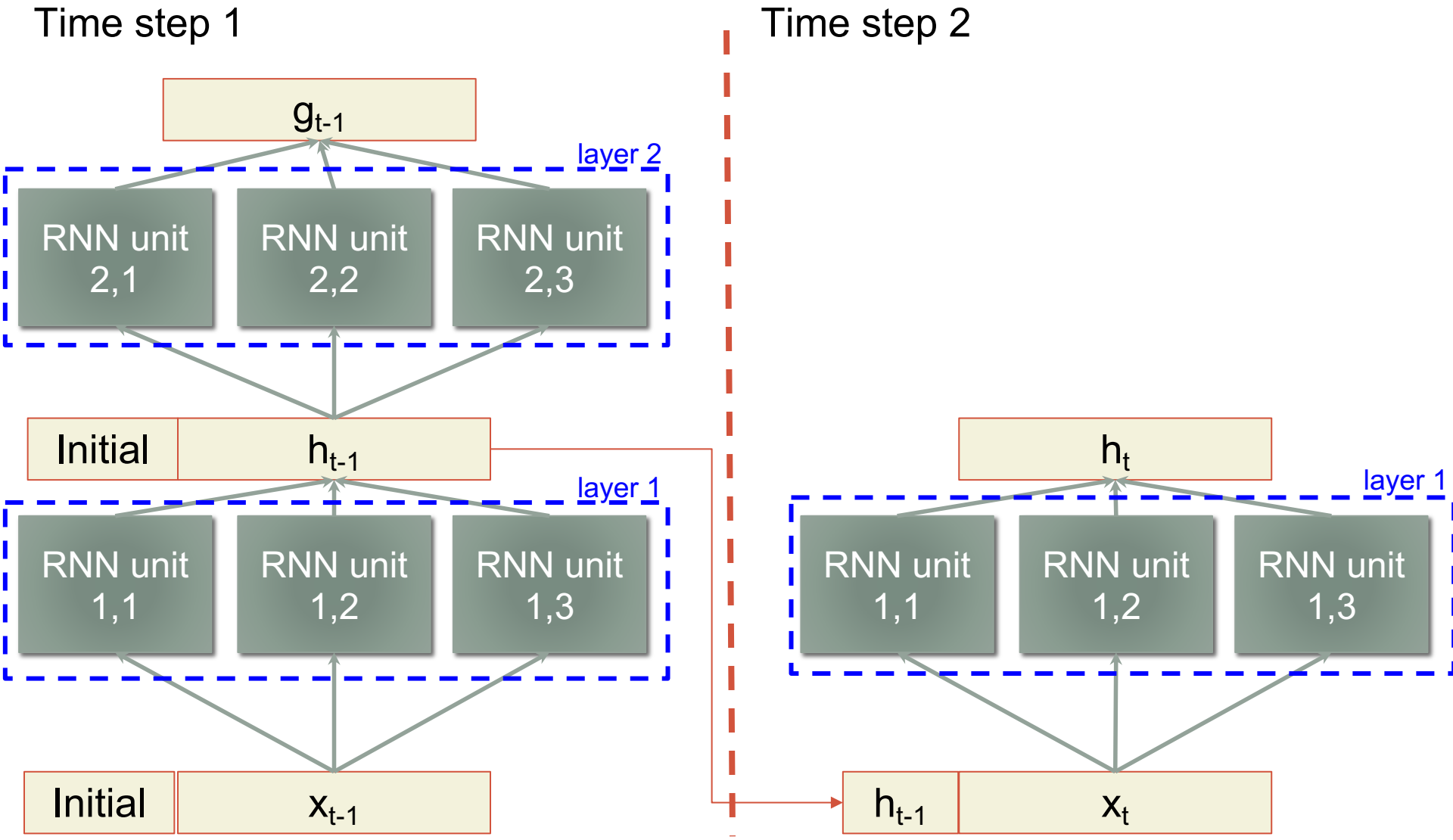
RNN layers (expanded in time)



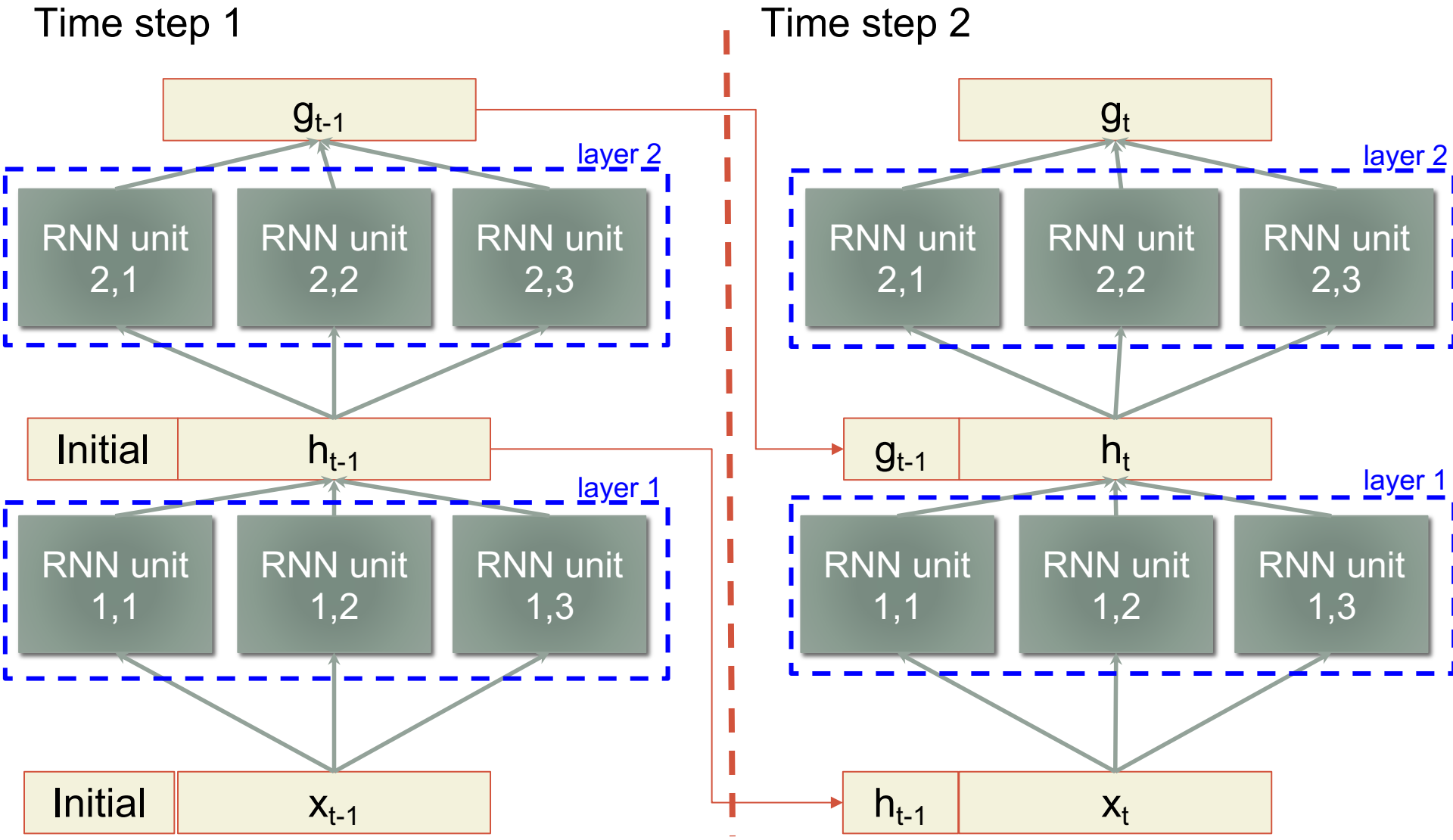
RNN layers (expanded in time)



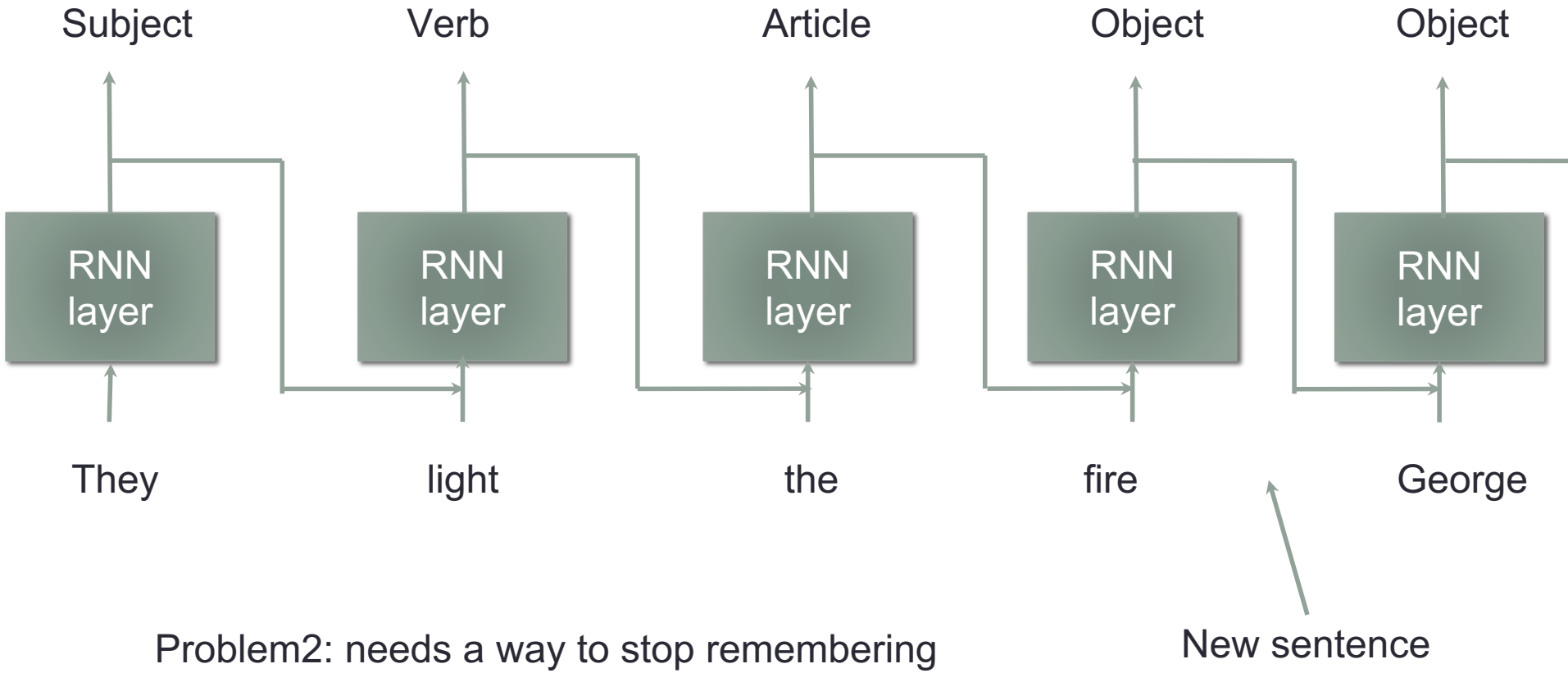
RNN layers (expanded in time)



RNN layers (expanded in time)



Recurrent neural network (RNN)

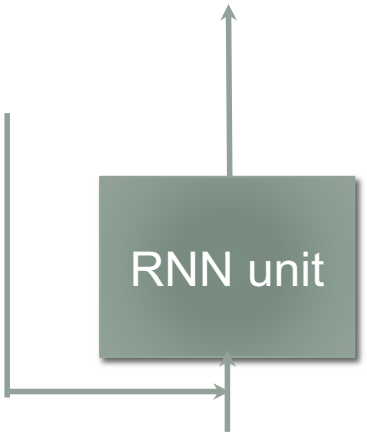


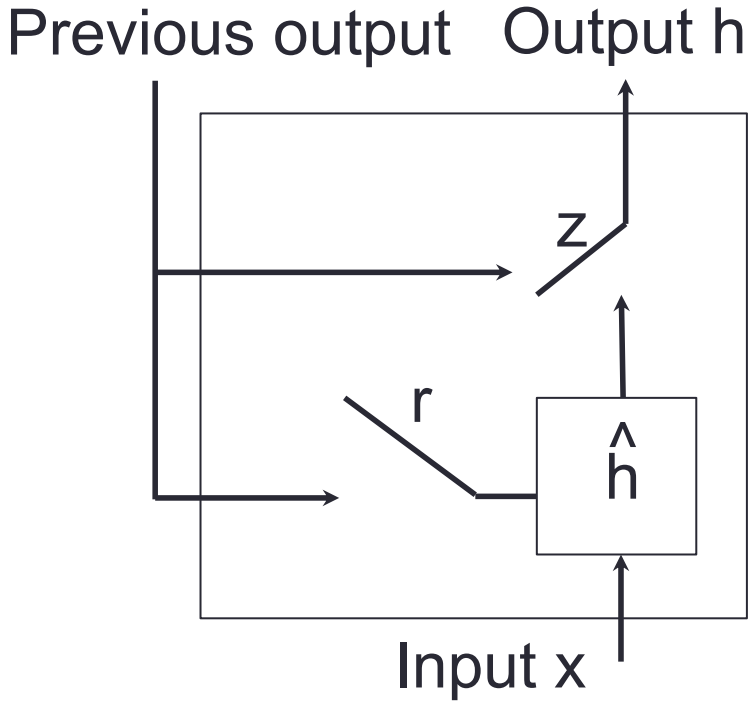
Problem2: needs a way to stop remembering

Can the network learn when to start and stop remembering things?

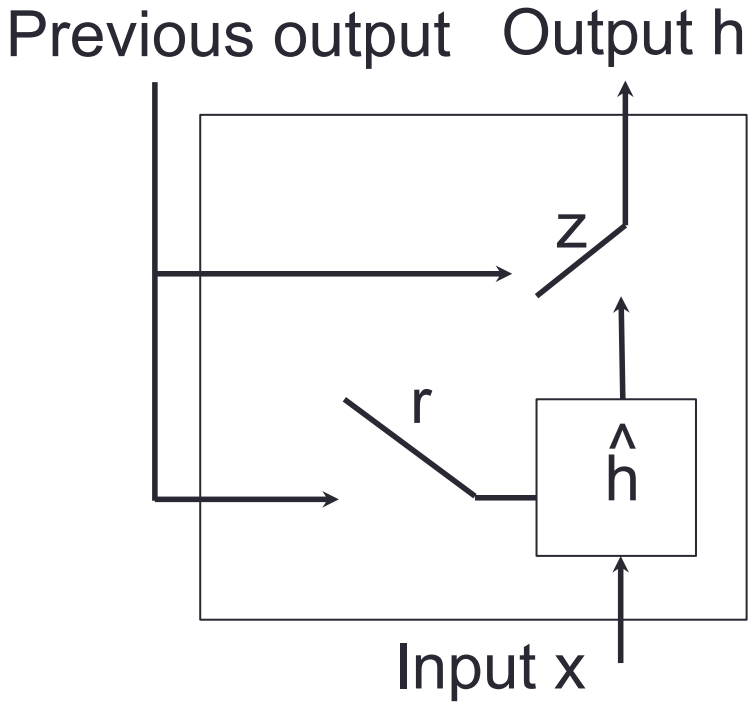
Gated Recurrent Unit (GRU)

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





Gated Recurrent Unit (GRU)

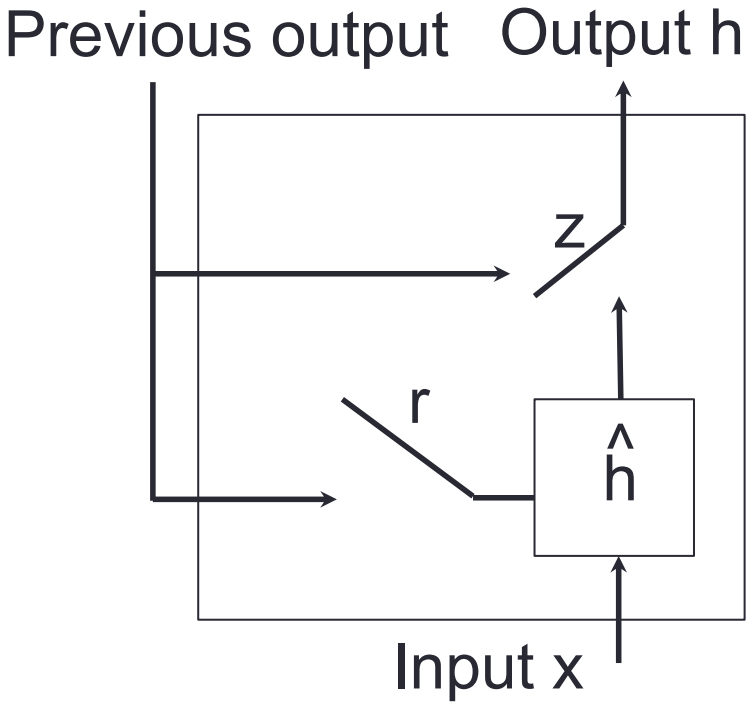


Neuron index

$$h_t^j = (1 - z_t^j) h_{t-1}^j + z_t^j \hat{h}_t^j$$

time index

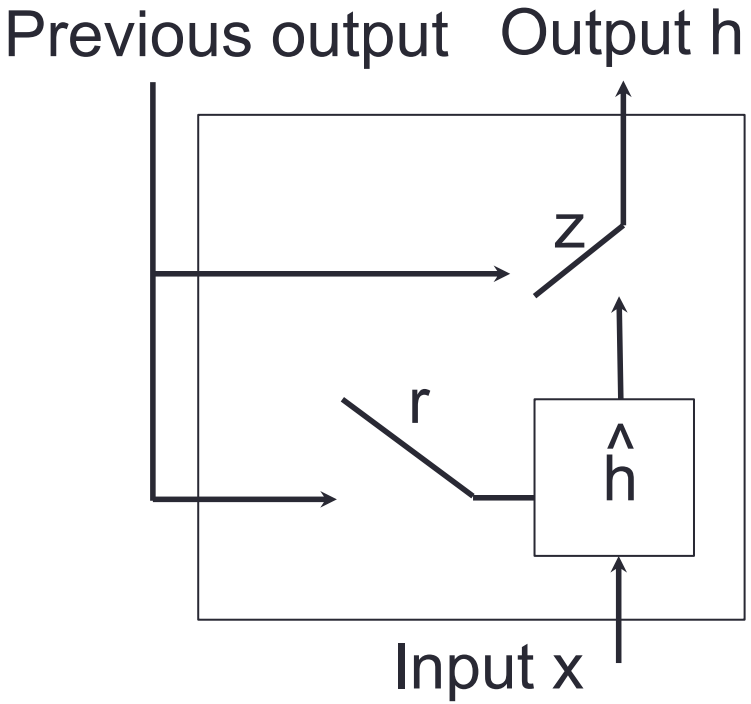
Gated Recurrent Unit (GRU)



$$h_t^j = (1 - z_t^j) h_{t-1}^j + z_t^j \hat{h}_t^j$$

One GRU neuron output (scalar)

Gated Recurrent Unit (GRU)



$$h_t^j = (1 - z_t^j) h_{t-1}^j + z_t^j \hat{h}_t^j$$

$$\hat{h}_t^j = tanh^j(W\mathbf{x}_t + U(\mathbf{r}_t \odot \mathbf{h}_{t-1}))$$

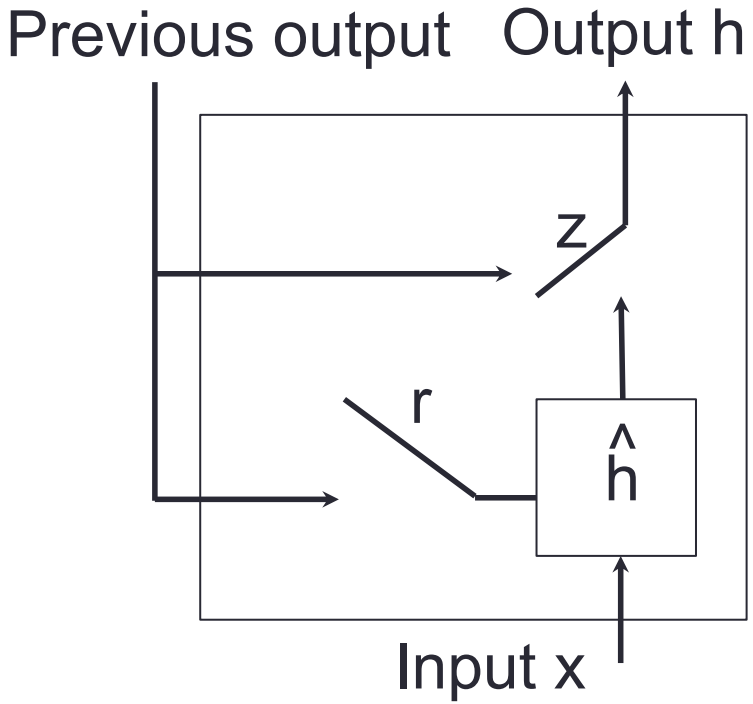
Element-wise product

Linear transform with matrix multiply

Vector (each value from each GRU unit in the previous layer)

$x_t^j = h_t^j$

Gated Recurrent Unit (GRU)



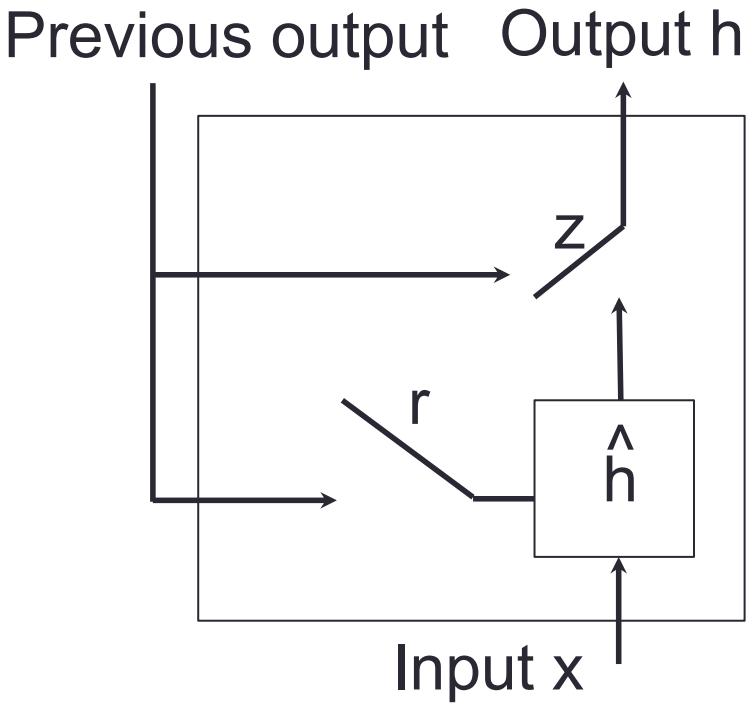
$$h_t^j = (1 - z_t^j) h_{t-1}^j + z_t^j \hat{h}_t^j$$

$$\hat{h}_t^j = tanh^j(W\mathbf{x}_t + U(\mathbf{r}_t \odot \mathbf{h}_{t-1}))$$

Takes the j-th element

Bounds the output

Gated Recurrent Unit (GRU)



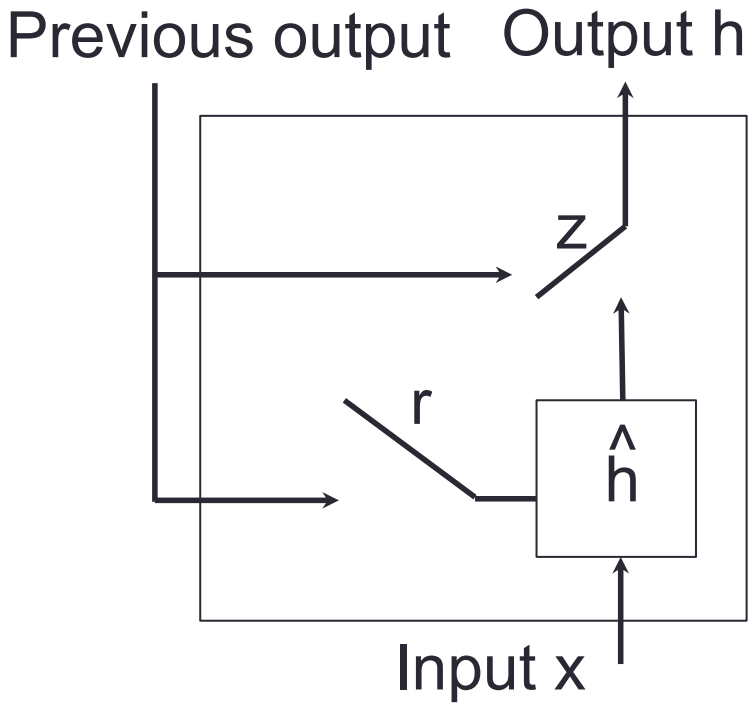
$$h_t^j = (1 - z_t^j) h_{t-1}^j + z_t^j \hat{h}_t^j$$

$$\hat{h}_t^j = tanh^j(W\mathbf{x}_t + U(\mathbf{r}_t \odot \mathbf{h}_{t-1}))$$

$$z_t^j = sigmoid^j(W_z\mathbf{x}_t + U_z\mathbf{h}_{t-1})$$

Indicates a different set of weights

Gated Recurrent Unit (GRU)



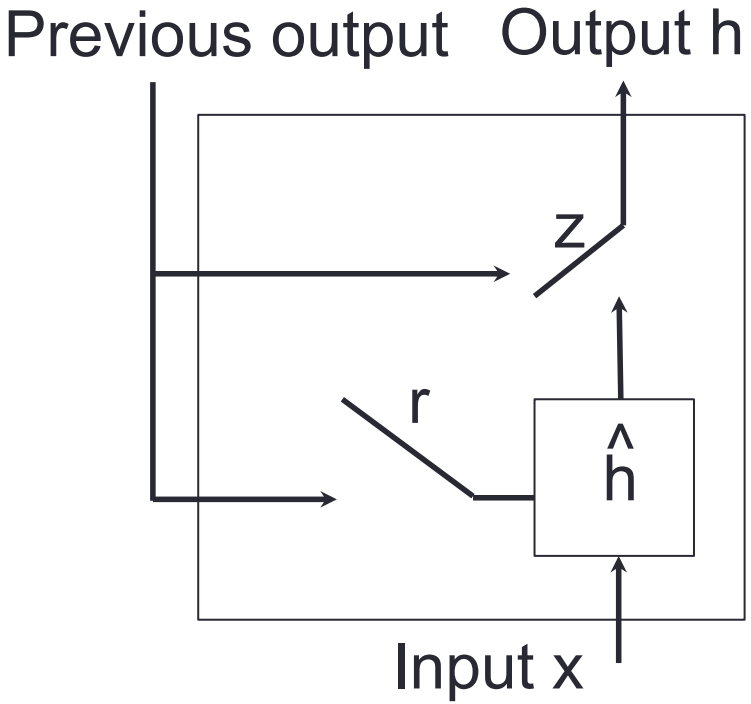
$$h_t^j = (1 - z_t^j) h_{t-1}^j + z_t^j \hat{h}_t^j$$

$$\hat{h}_t^j = tanh^j(W\mathbf{x}_t + U(\mathbf{r}_t \odot \mathbf{h}_{t-1}))$$

$$z_t^j = sigmoid^j(W_z\mathbf{x}_t + U_z\mathbf{h}_{t-1})$$

Bounds the output to 0 to 1 for interpolation

Gated Recurrent Unit (GRU)



$$h_t^j = (1 - z_t^j) h_{t-1}^j + z_t^j \hat{h}_t^j$$

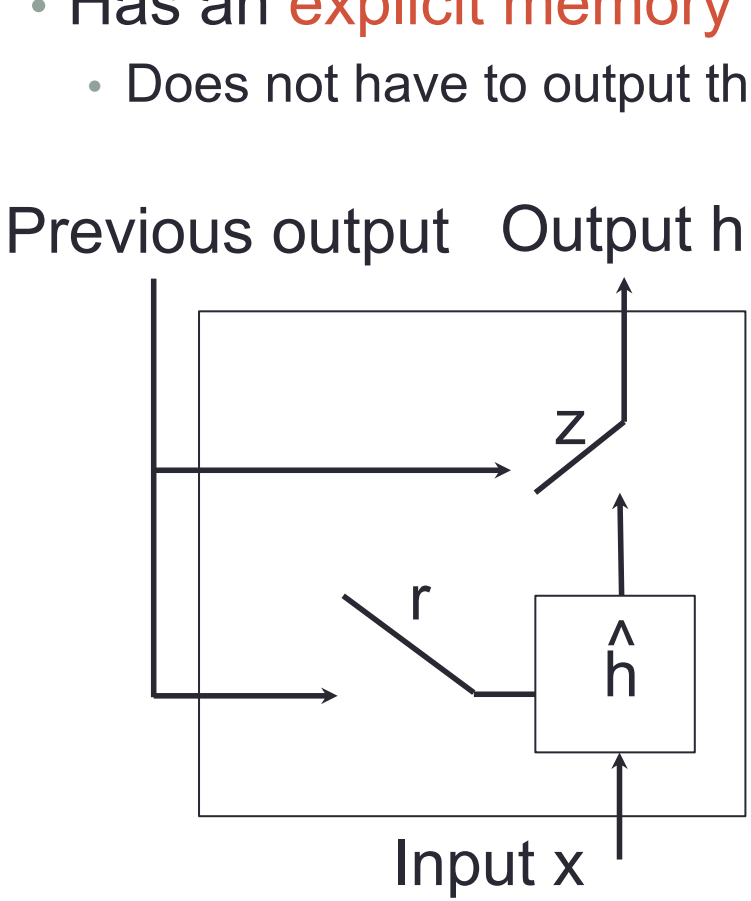
$$\hat{h}_t^j = tanh^j(W\mathbf{x}_t + U(\mathbf{r}_t \odot \mathbf{h}_{t-1}))$$

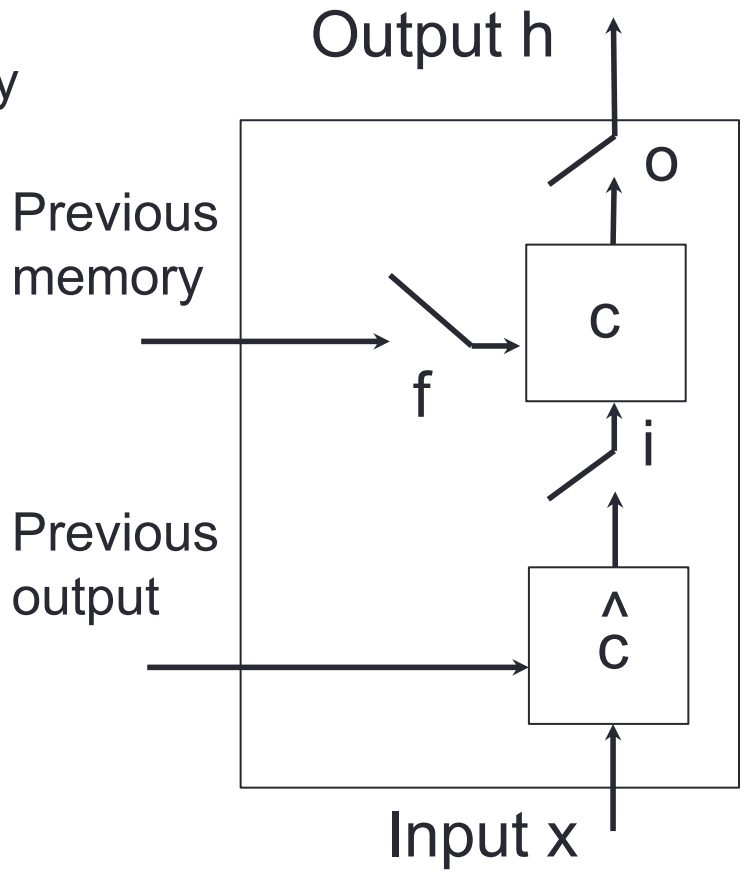
$$z_t^j = sigmoid^j(W_z\mathbf{x}_t + U_z\mathbf{h}_{t-1})$$

$$r_t^j = sigmoid^j(W_r\mathbf{x}_t + U_r\mathbf{h}_{t-1})$$

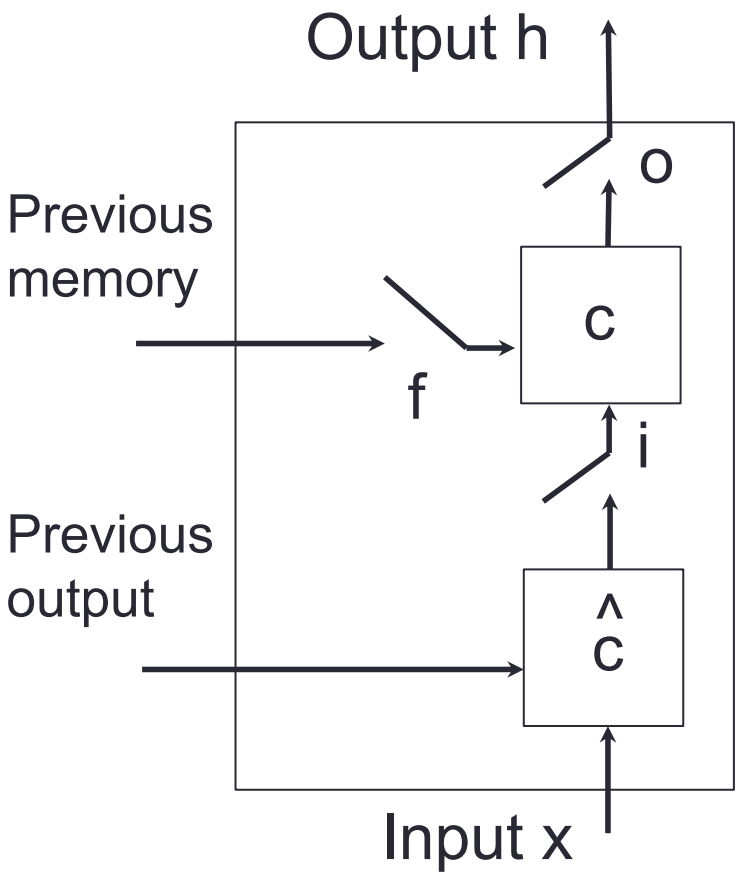
Long Short-Term Memory (LSTM)

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





Long Short-Term Memory (LSTM)

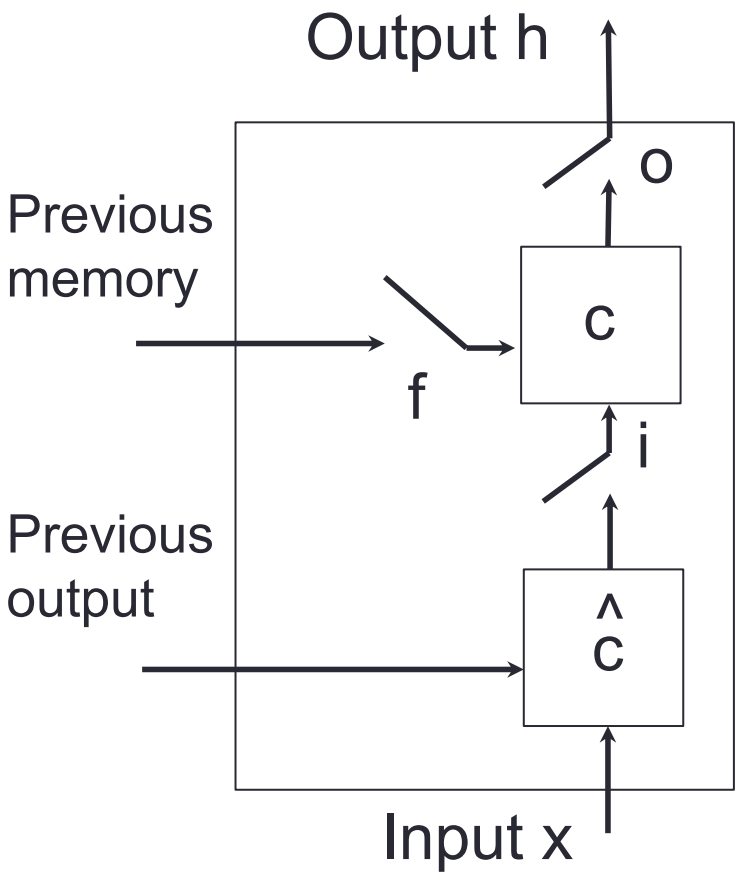


$$i_t^j = F^j(W_i \mathbf{x}_t + U_i \mathbf{h}_{t-1} + V_i \mathbf{c_{t-1}})$$
$$o_t^j = F^j(W_o \mathbf{x}_t + U_o \mathbf{h}_{t-1} + V_o \mathbf{c_t})$$
$$f_t^j = F^j(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1} + V_j \mathbf{c_{t-1}})$$

Contribution from memory "Peephole connection"

Vs are diagonal matrices(Each cell can only see its own memory)

Long Short-Term Memory (LSTM)



$$i_t^j = F^j(W_i \mathbf{x}_t + U_i \mathbf{h}_{t-1} + V_i \mathbf{c_{t-1}})$$

$$o_t^j = F^j(W_o \mathbf{x}_t + U_o \mathbf{h}_{t-1} + V_o \mathbf{c_t})$$

$$f_t^j = F^j(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1} + V_j \mathbf{c_{t-1}})$$

$$h_t^j = o_t^j tanh(c_t^j)$$

$$c_t^j = f_t^j c_{t-1}^j + i_t^j \hat{c}_t^j$$

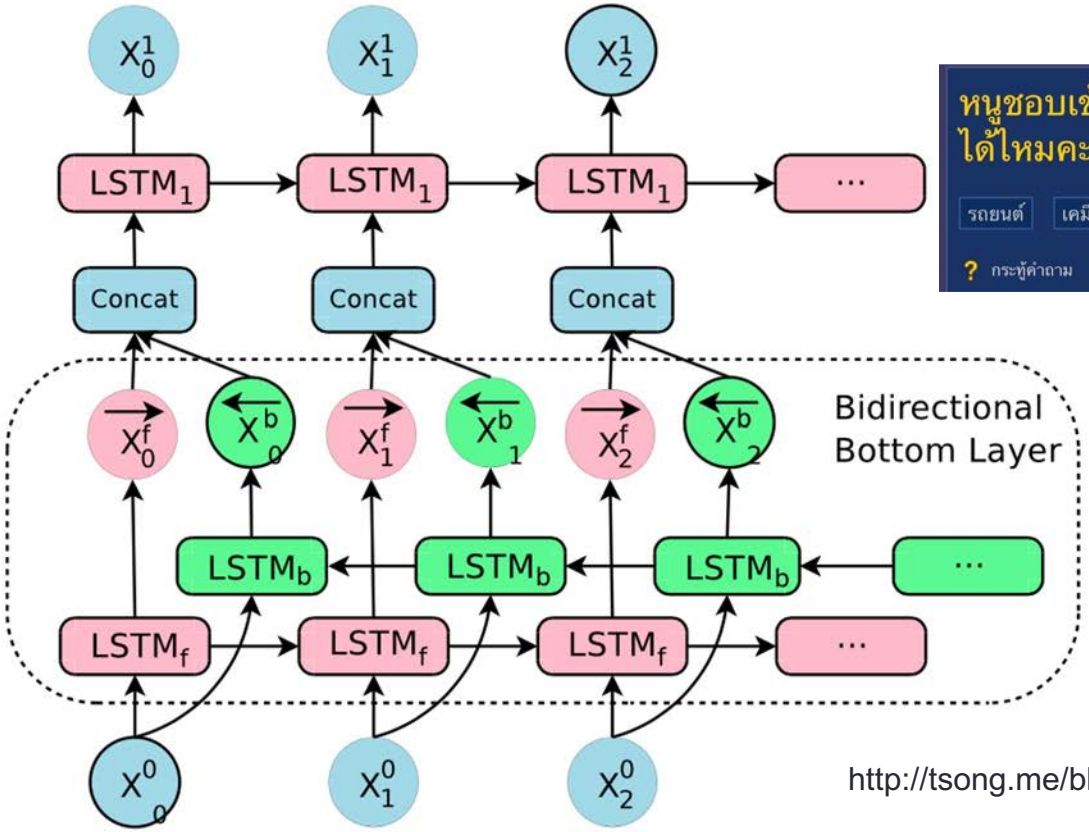
$$\hat{c}_t^j = tanh^j(W_c \mathbf{x}_t + U_c \mathbf{h}_{t-1})$$

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)
- Advantage of LSTM over GRU is for when you want to use the memory values

Bi-directional LSTM

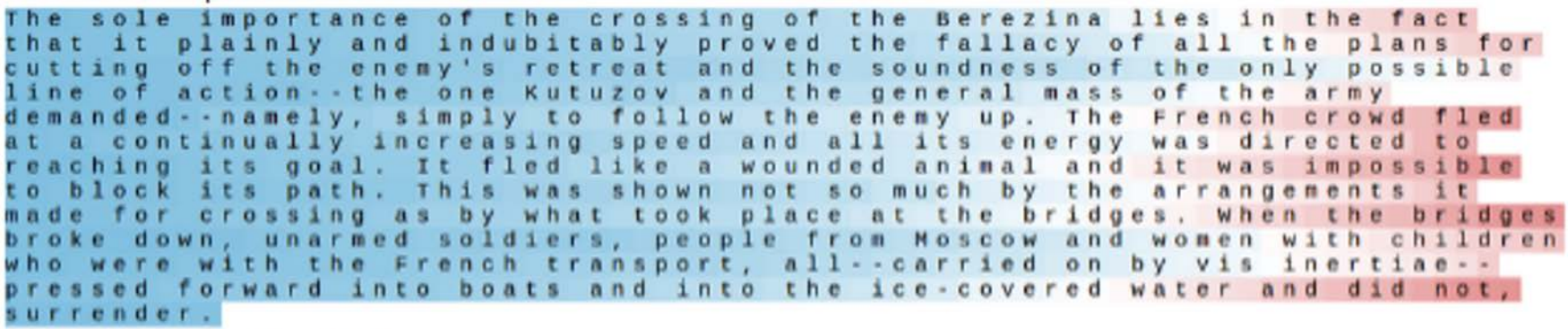
- The previous GRU/LSTM only goes backward in time (uni-directional)
- Most of the time information from the future is useful for predicting the current output



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

LSTM remembers meaningful things

Cell sensitive to position in line:



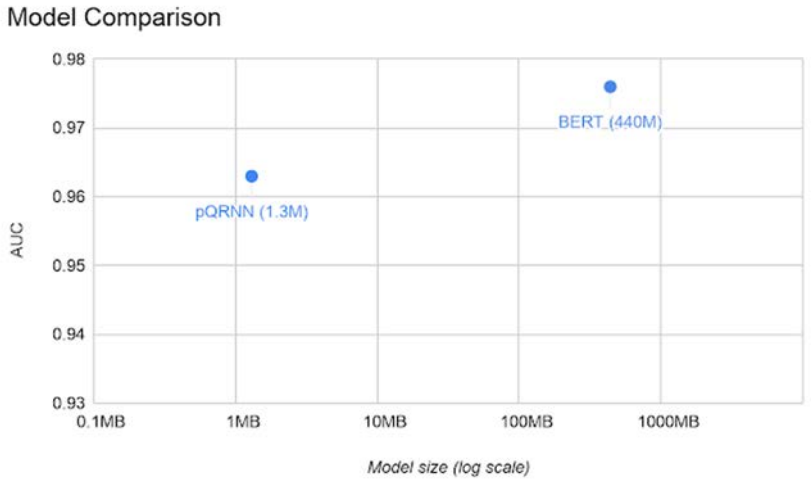
Cell that turns on inside quotes:

Andrej Karpathy, Visualizing and Understanding Recurrent Networks, 2015, https://arxiv.org/abs/1506.02078

Notes on RNNs

Considered obsolete by some due to speed and performance

- Improved speed using QRNN https://arxiv.org/abs/1611.01576
- Can be as good as BERTs on simple tasks



https://ai.googleblog.com/2020/09/advancing-nlp-with-efficient-projection.html

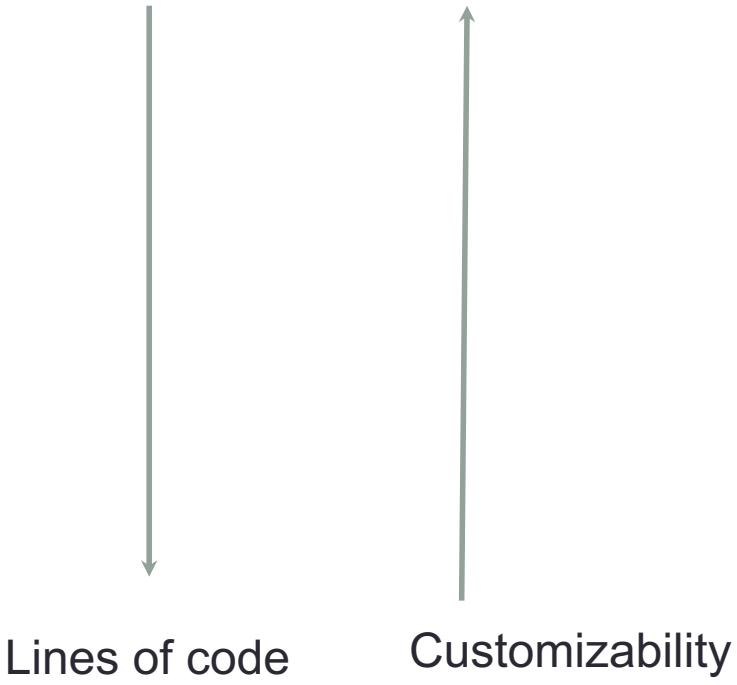
Easier to train than attention-based models (easier to tune and **faster convergence**). Less memory requirements.

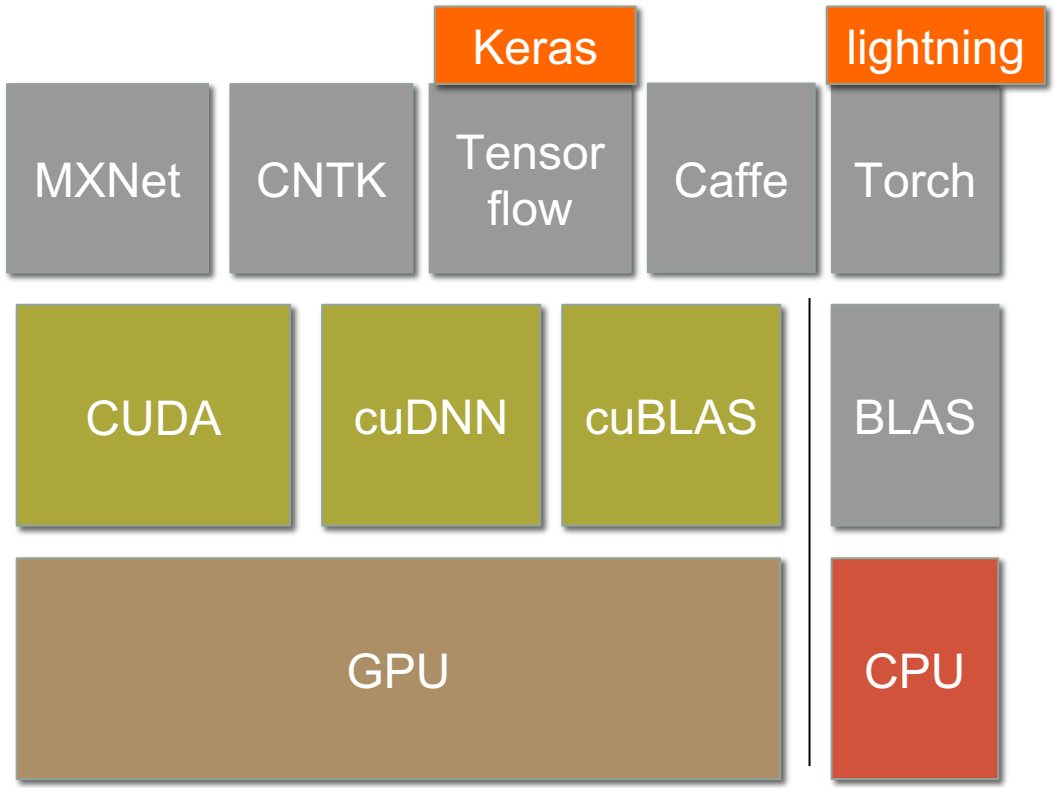
Tips to tune

- Feeling and experience ¯\(°_o)/¯
- Take numbers from papers
- Grid search
  - Heuristic search
  - Random search
  - Genetic Algorithm
- Picking the right type of model is more important than pikcing the right number of neurons
  - Inductive bias

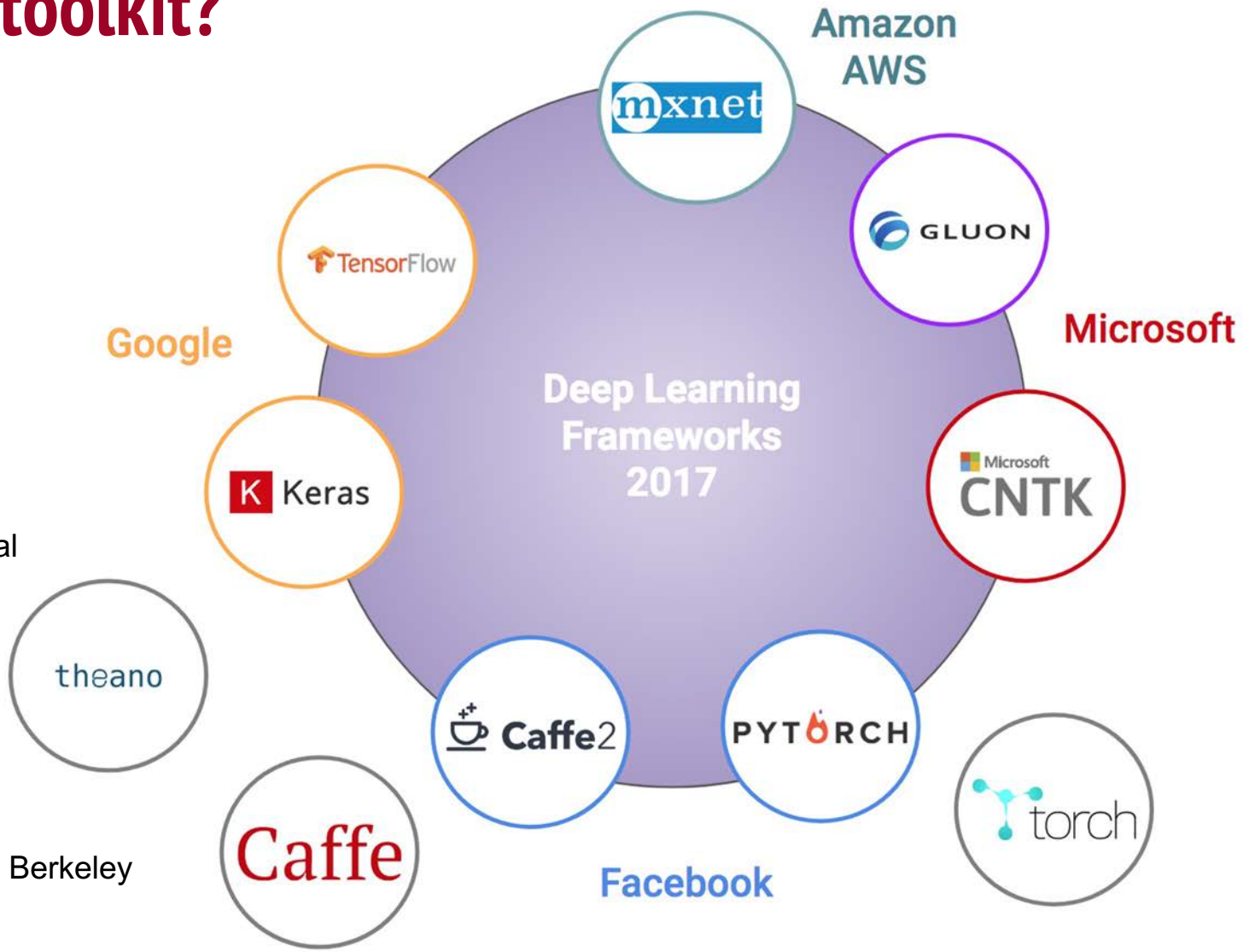
What toolkit

Tradeoff between customizability and ease of use



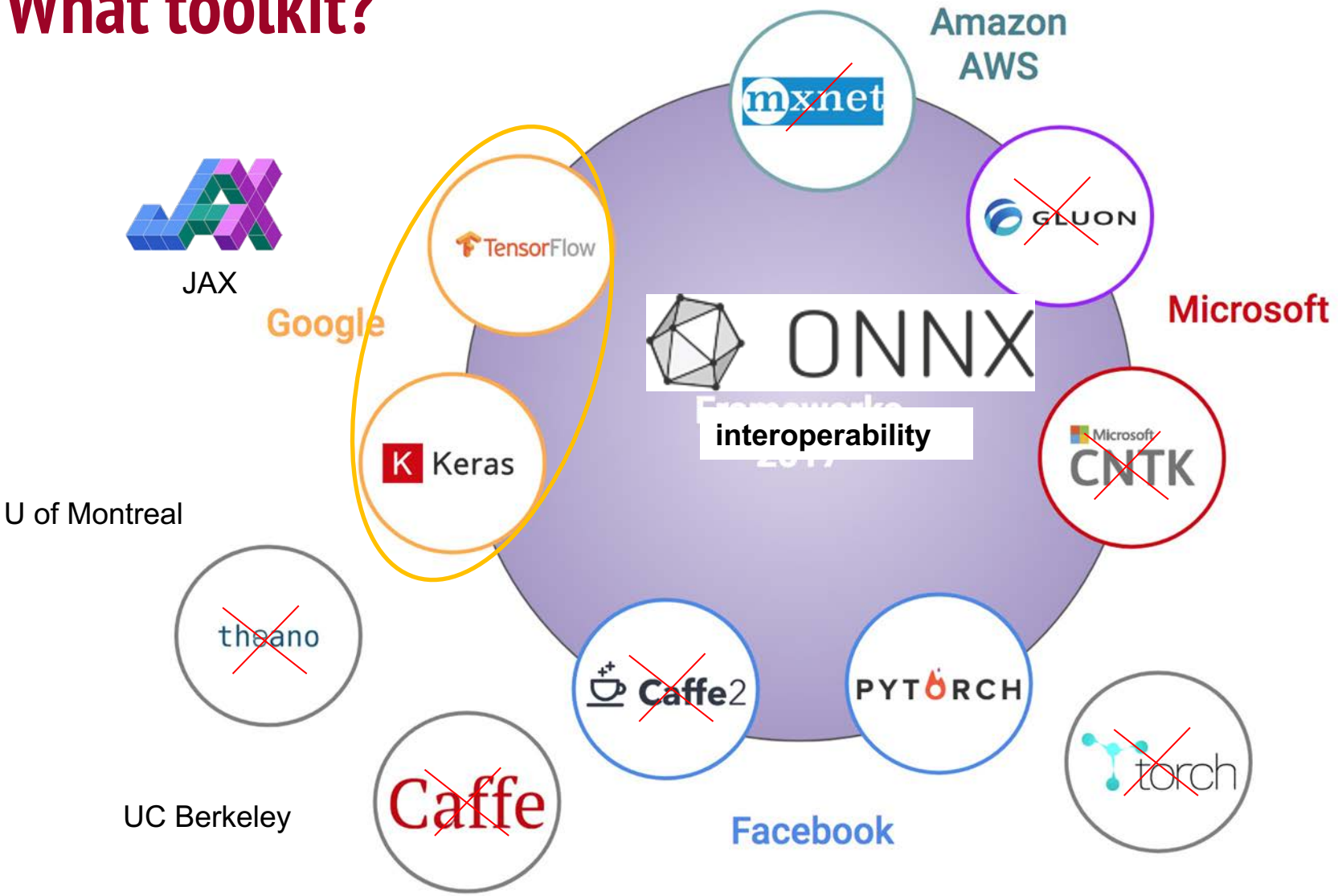


What toolkit?



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

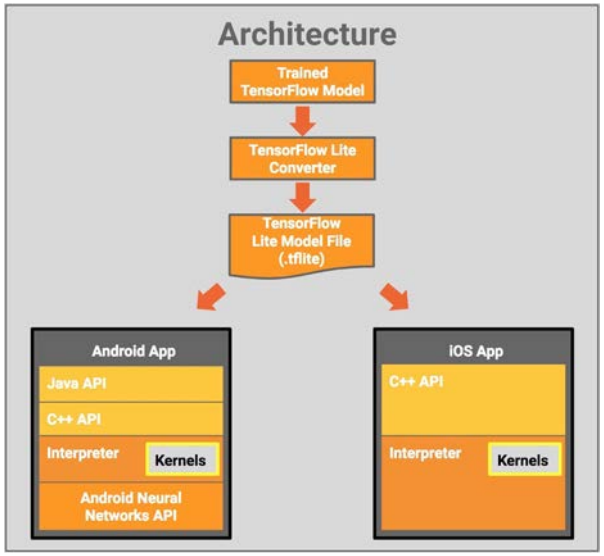
What toolkit?

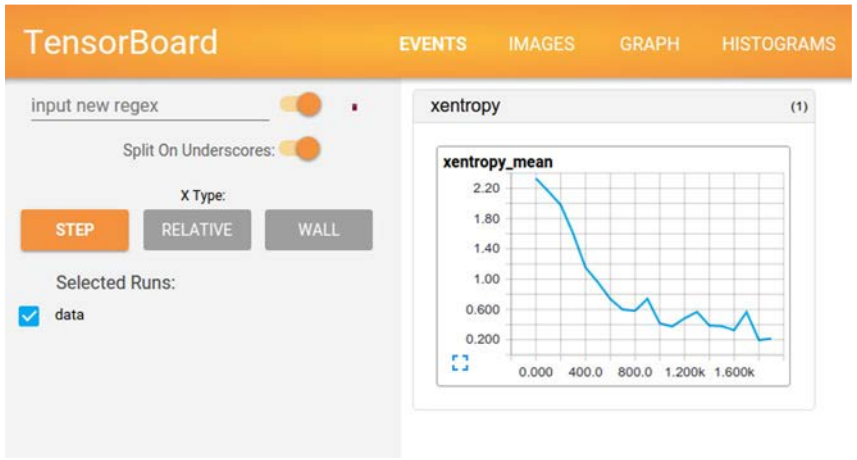


https://towardsdatascience.com/battle-of-the-deep-learning-frameworks-part-i-cff0e3841750

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
  - Javaruntime support
- Best tools: increasingly pytorch
- Community: increasingly pytorch
- In the end you should know some tf and pytorch





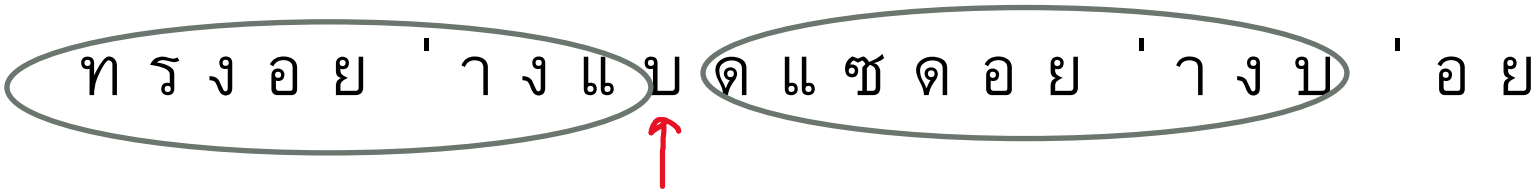
Pytorch steps

- Setting up dataloader
  - Gives minibatch
- Define a network
  - Init weights
  - Define computation graph
- Setup optimization method
  - Pick LR scheduler
  - Pick optimizer
- Training loop
  - Forward (compute Loss)
  - Backward (compute gradient and apply gradient)

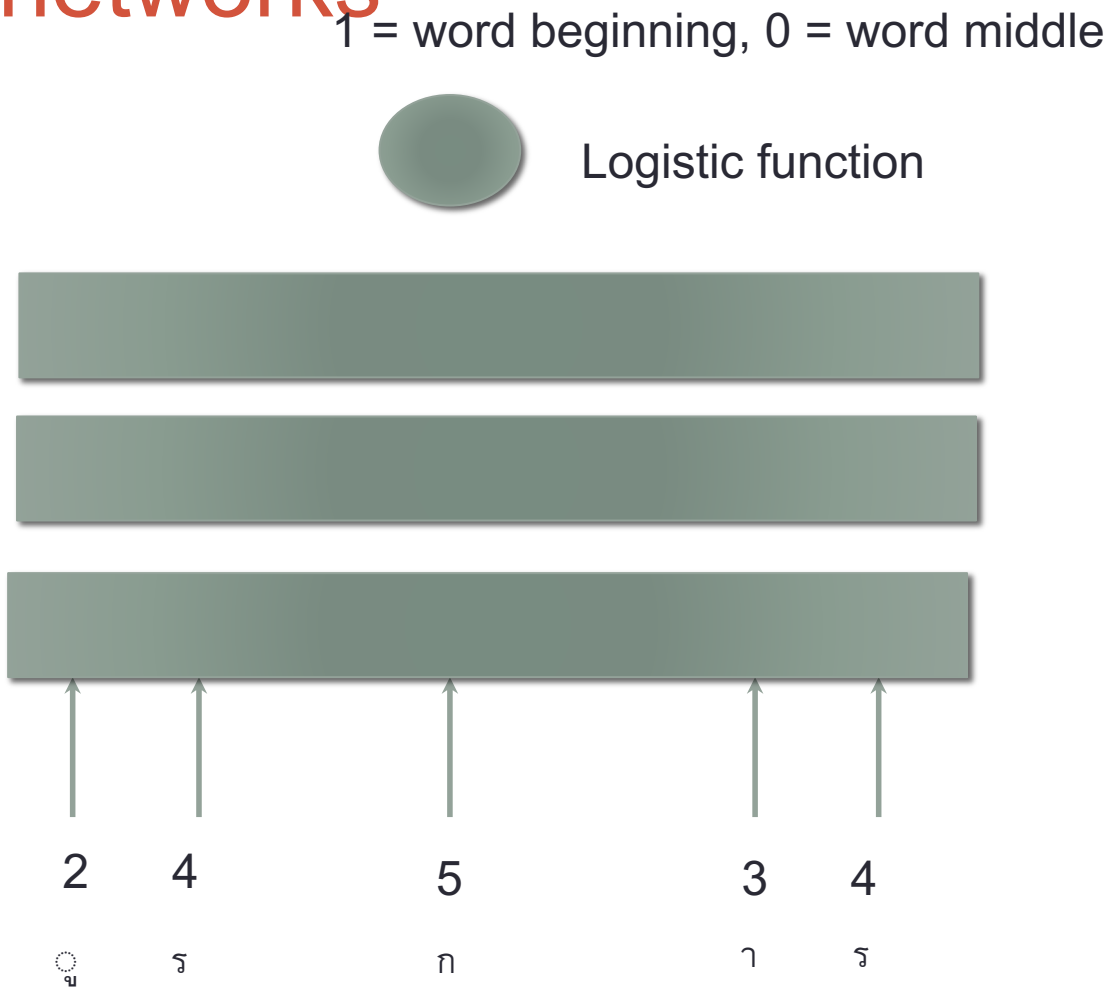
- Let’s demo

Lab/HW

- Word segmentation using pytorch
- Given a letter with 10 letters before and after, determine whether it's a start of a word



Word segmentation with fully connected networks

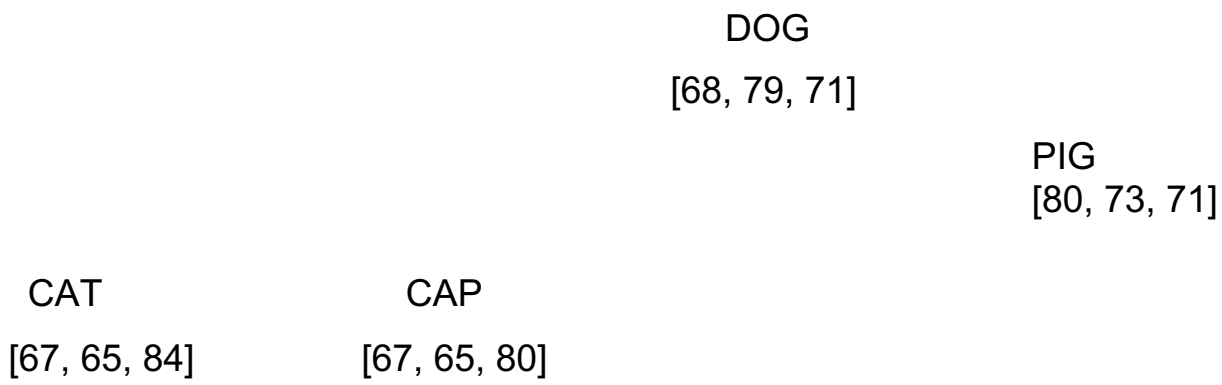


Debugging guide

- https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/guide3/Debugging_PyTorch.html
  has list of common errors and best practices
- http://karpathy.github.io/2019/04/25/recipe/
  has guide for end-to-end model building (start simple and go more advance)

Embeddings

- A way to encode information to a lower dimensional space
  - We can 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

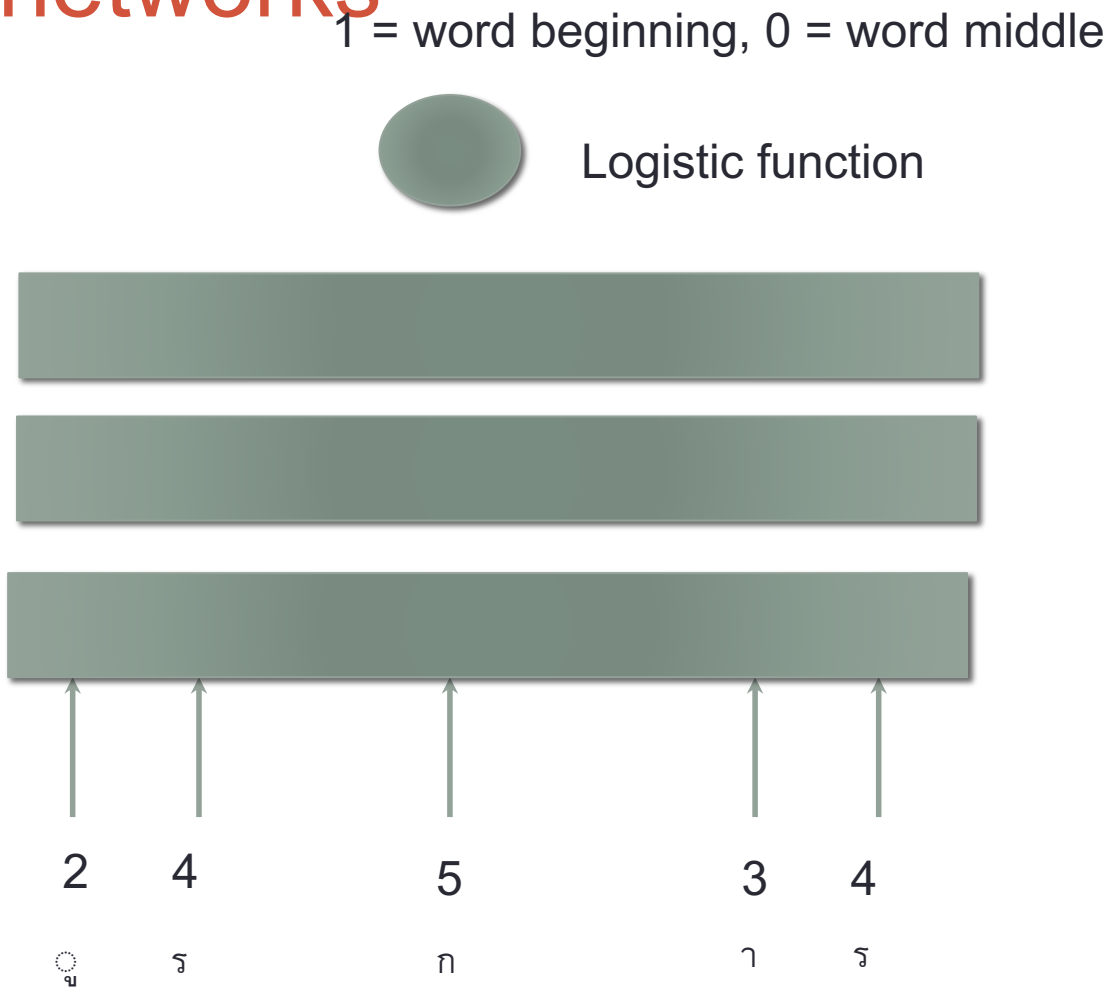
Dense representation

- 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

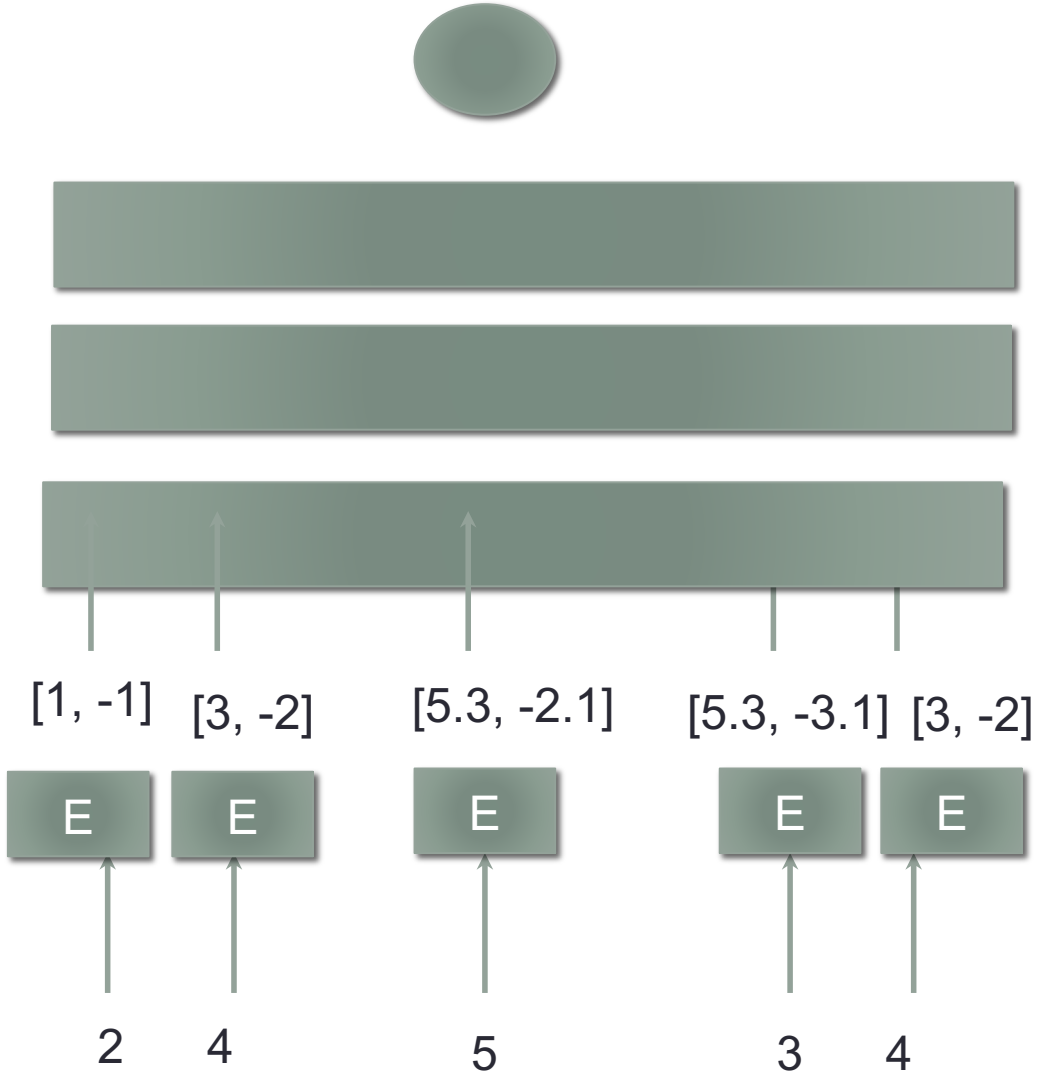
Word segmentation with fully connected networks



Adding embedding layer

Embedding layer shares the same weights

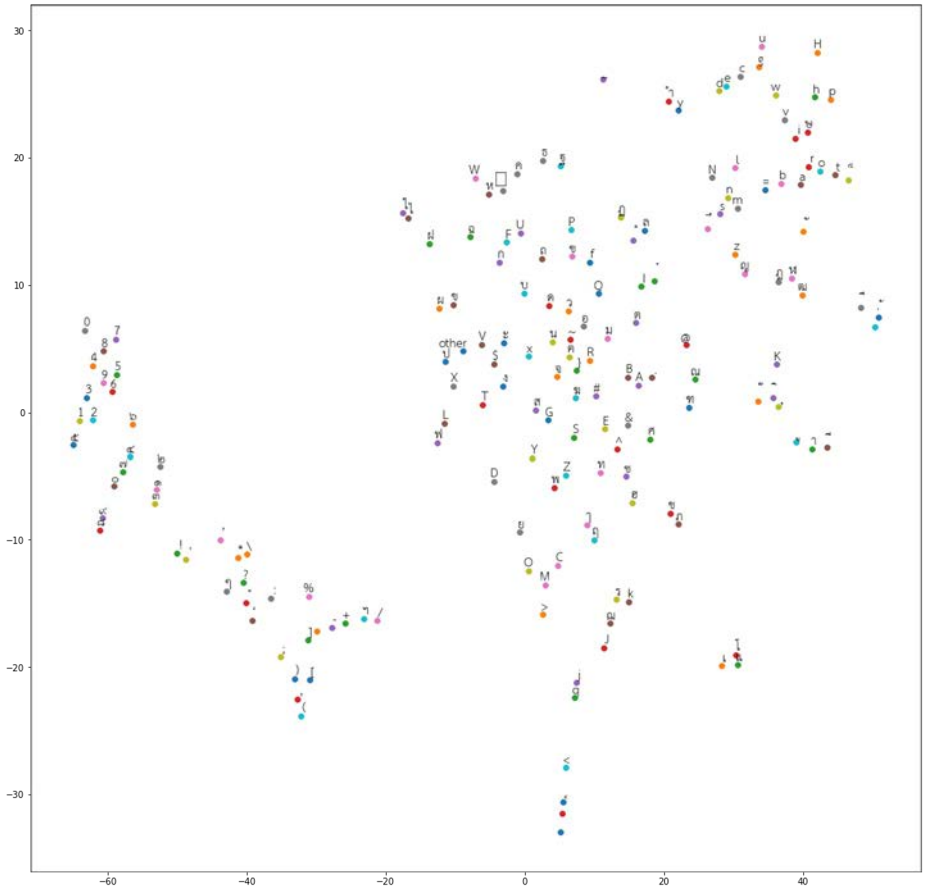
Parameter sharing!

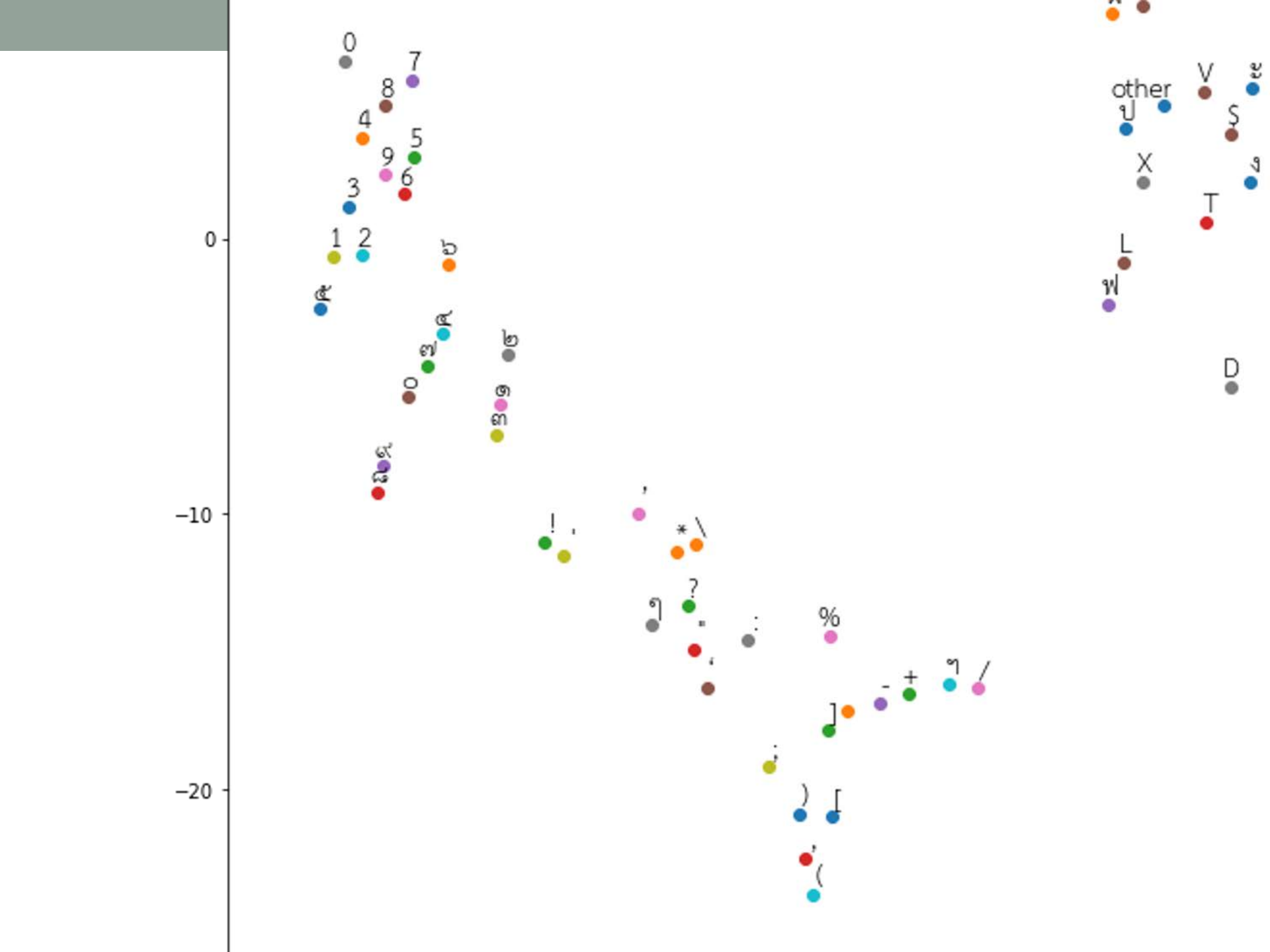


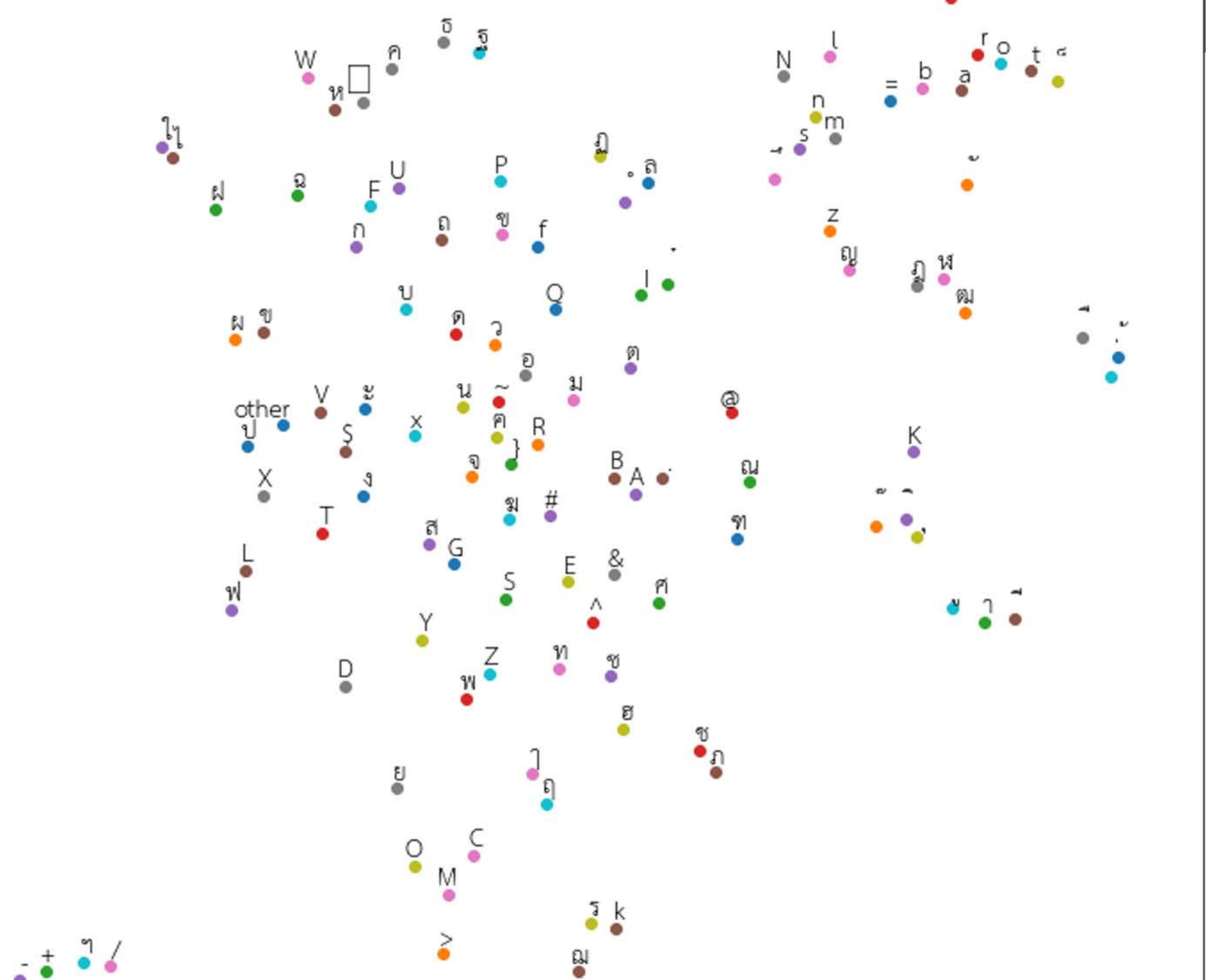
More on embeddings in the next two lectures!

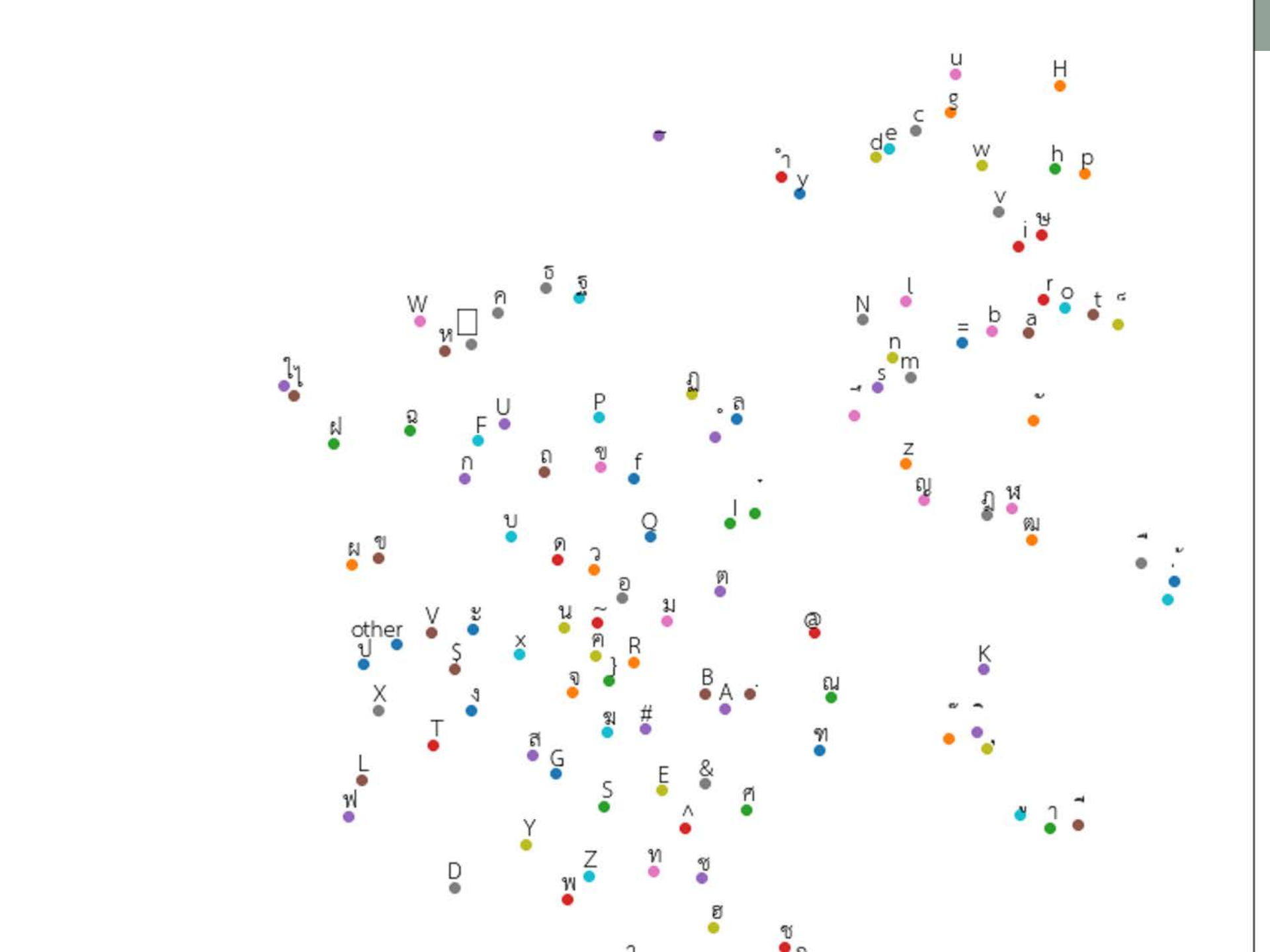
Embedding and meaning (semantics)

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



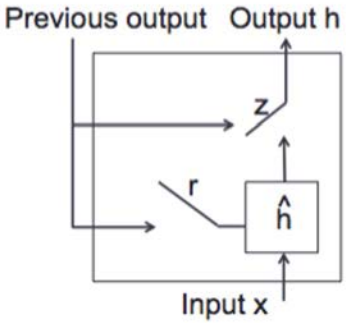

0
−10
−20
other

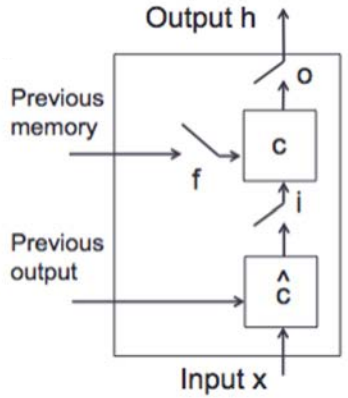

other

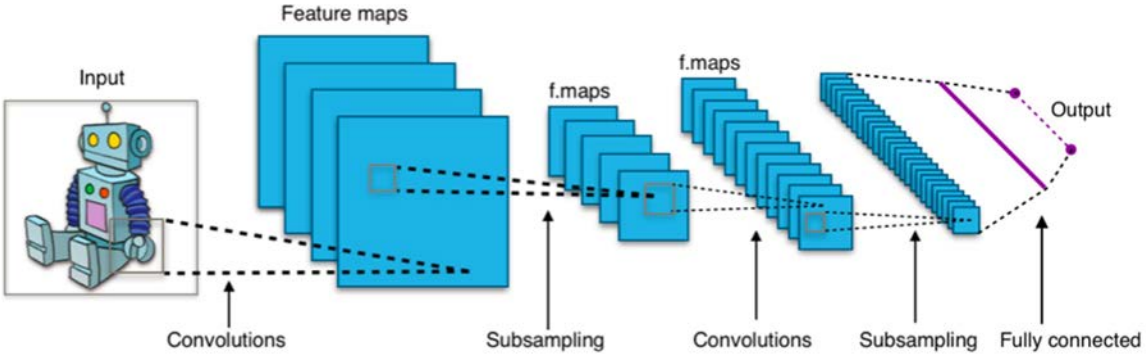

other

Neural networks

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





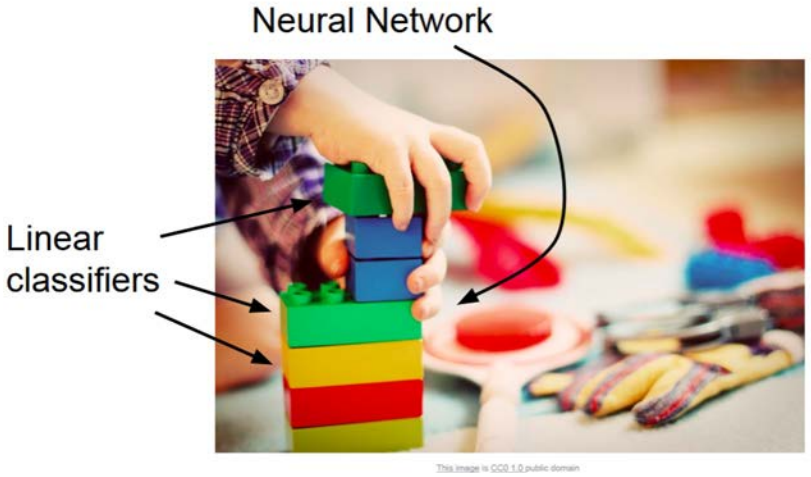


Attention modeling
Recursive neural networks

← 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/

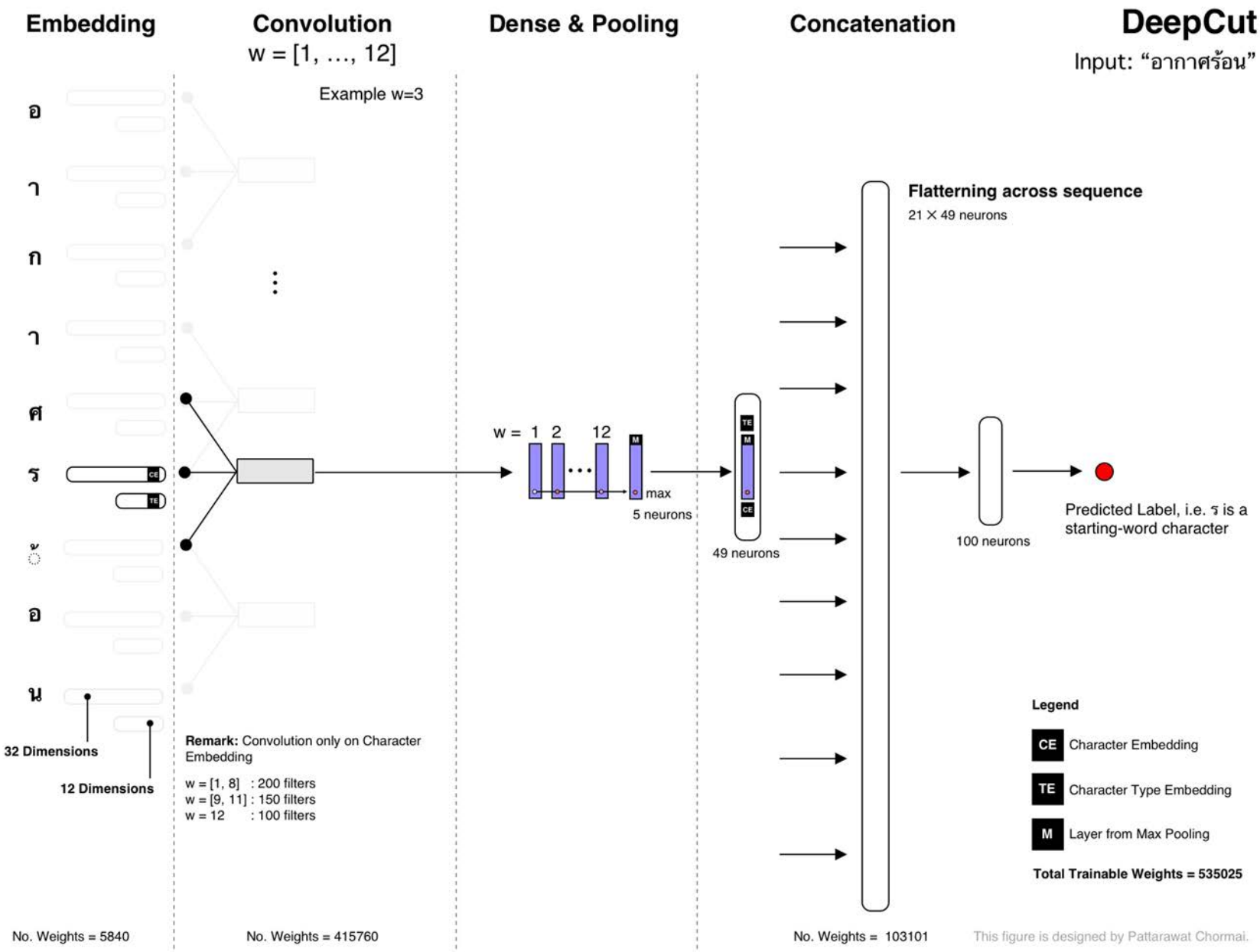
Back to tokenization…

TABLE II
RESULTS OF THE SIX BEST TEAMS

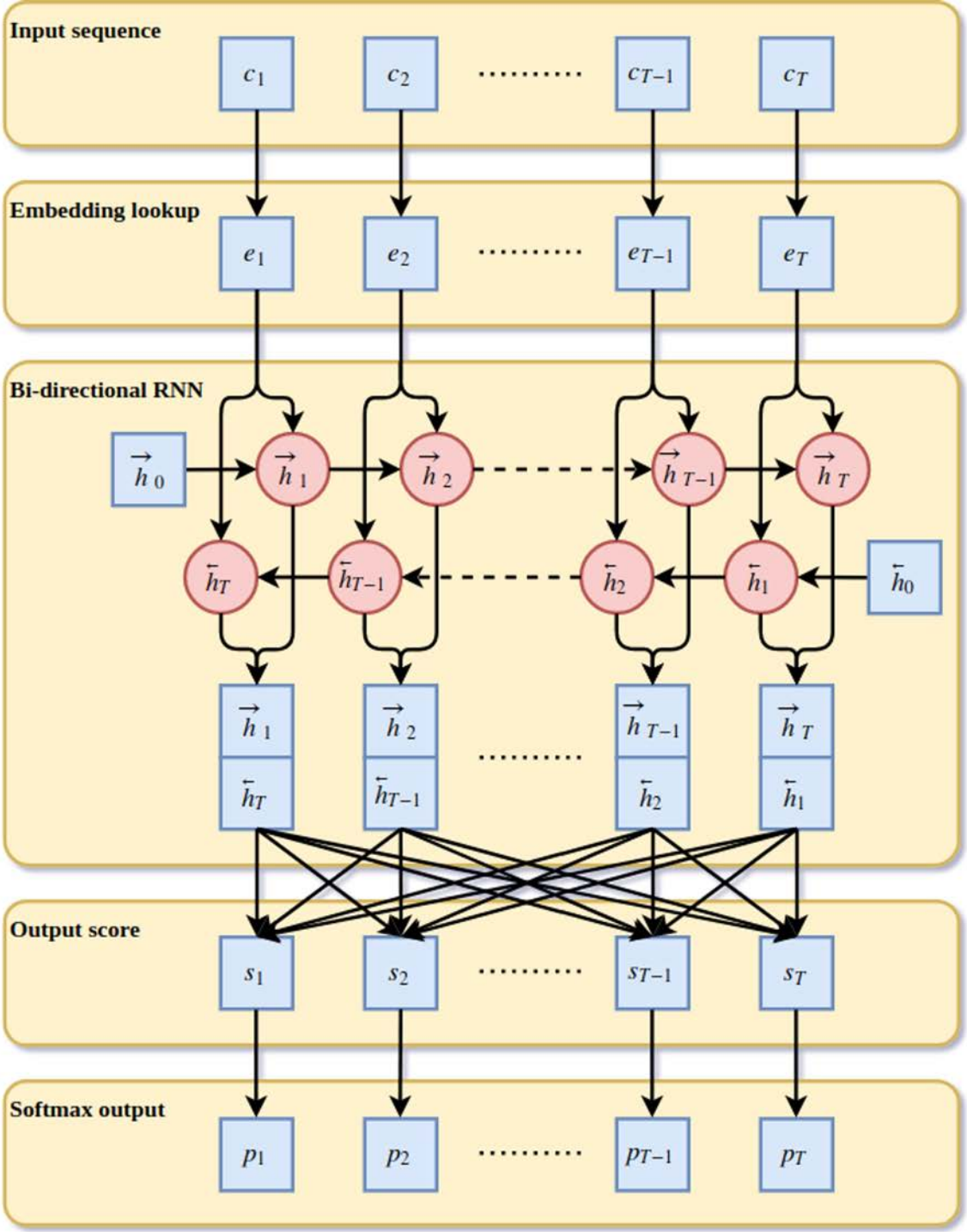
Type of participants	F-Measure (%)	Time (mm:ss)
Non-Students[a]	97.94937	00:47
Non-Students	97.84097	02:46
Non-Students	97.18822	00:26
Bachelor Students[b]	95.78162	01:08
Master Students	95.56670	12:14
PhD+Master Students	92.02067	02:28

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

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



https://github.com/rkcosmos/deepcut



https://www.sertiscorp.com/november-20-2017

$$e_t = W_c c_t$$

$$z_t = \text{sigmoid}\left(W_z e_t + U_z h_{t-1} + b_z\right)$$

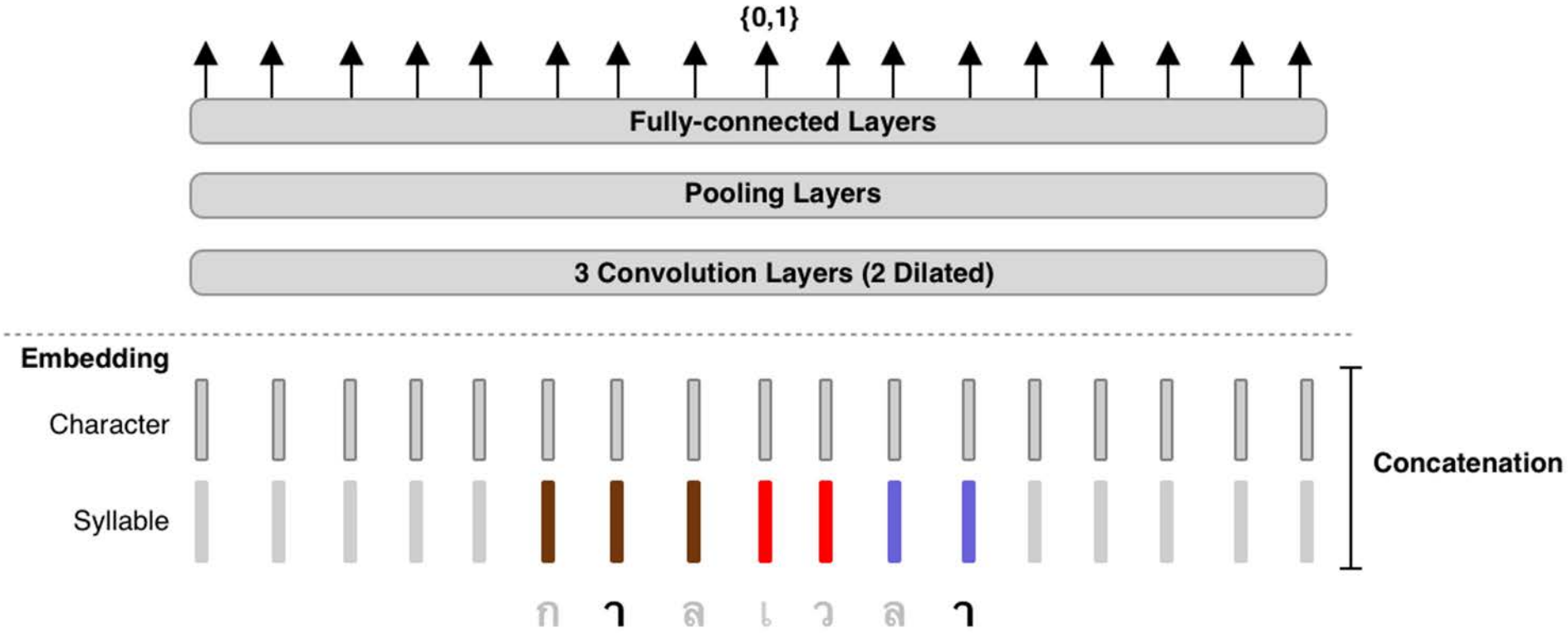
$$r_t = \text{sigmoid}\left(W_r e_t + U_r h_{t-1} + b_r\right)$$

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tanh\left(W_h e_t + U_h\left(r_t \odot h_{t-1}\right) + b_h\right)$$

$$H_t = \left[\overrightarrow{h}_t, \overleftarrow{h}_{T-t+1}\right]$$

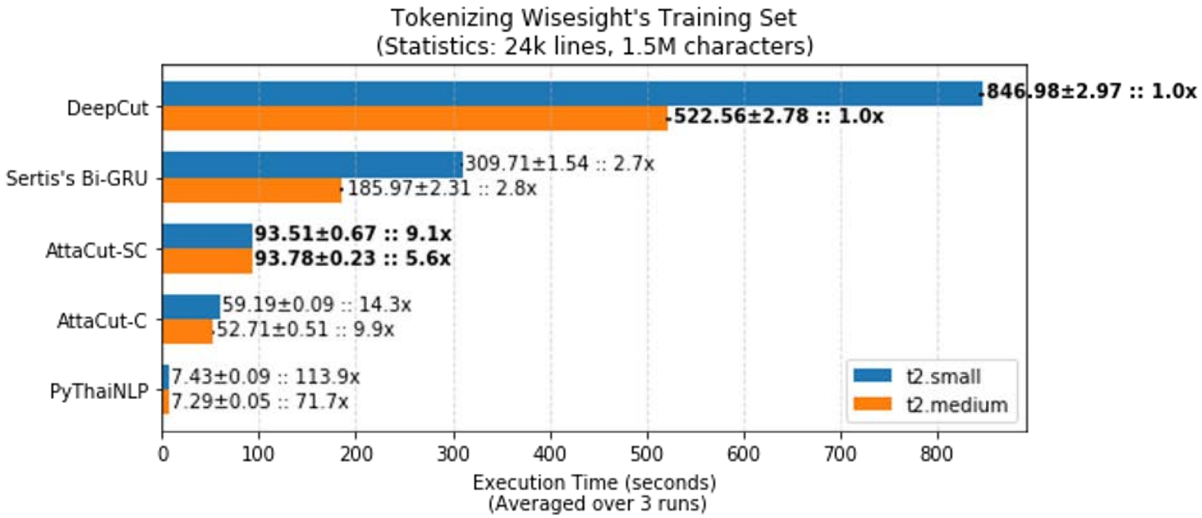
$$s_t = H_t W_s + b_s^{\top}$$

$$p_t = \frac{\exp(s_t)}{\exp(s_{t1}) + \exp(s_{t2})}$$



https://github.com/PyThaiNLP/attacut

		Others			Ours	
Last Updated: 29/08/2019		**PyThaiNLP newmm**	**Sertis Bi-GRU**	**DeepCut**	**AttaCut-C**	**AttaCut-SC**
BEST Validation Set						
Character-Level	precision	0.94±0.11	0.95±0.10	0.99±0.05	0.97±0.07	0.98±0.05
	recall	0.83±0.09	0.99±0.02	0.99±0.03	0.98±0.04	0.99±0.03
	f1	0.88±0.08	0.97±0.07	**0.99±0.04**	0.98±0.05	0.99±0.04
Word-Level	precision	0.73±0.16	0.91±0.14	0.97±0.07	0.94±0.10	0.96±0.08
	recall	0.65±0.16	0.94±0.10	0.97±0.07	0.94±0.09	0.97±0.08
	f1	0.68±0.15	0.93±0.12	**0.97±0.07**	0.94±0.10	**0.97±0.08**
BEST Test Set						
Character-Level	precision	0.91±0.15	0.92±0.11	0.96±0.08	0.94±0.10	0.95±0.09
	recall	0.85±0.09	0.98±0.04	0.98±0.04	0.98±0.04	0.98±0.04
	f1	0.86±0.11	0.95±0.08	**0.97±0.06**	0.96±0.07	0.96±0.07
Word-Level	precision	0.70±0.19	0.85±0.18	0.92±0.14	0.88±0.17	0.91±0.15
	recall	0.64±0.18	0.90±0.14	0.93±0.12	0.91±0.14	0.92±0.13
	f1	0.67±0.19	0.87±0.16	**0.93±0.13**	0.89±0.16	0.91±0.14



https://github.com/PyThaiNLP/attacut

ผมเห็นคนวงการนี้เมื่อ 20-30 ปีที่แล้วทำเรื่องตัดคำ วงการนี้มันไม่ไปไหนเลยใช่ไหมเนี่ย

มิตรสหาย Business Development ท่านนึง

Words of caution

Statistical tokenizers fail on mismatched data

A tokenizer trained on social text might not be able to cut simple words like

มะม่วง มะละกอ

https://www.aclweb.org/anthology/2020.emnlp-main.315/

	WS160	TNHC
Deepcut	93.8	93.5
Attacut	93.5	80.8

Statistical tokenizers fails unpredictably

หมูกรอบ => |หมู|กรอบ|
ข้าวผัดคะน้าหมูกรอบหนึ่งจาน => |ข้าวผัด|คะน้า|หมูก|รอบ|หนึ่ง|จาน|

Might need rule-based to override (Deepcut has this)

For speed, maximal matching (newmm) is reliable.

- drawbacks?

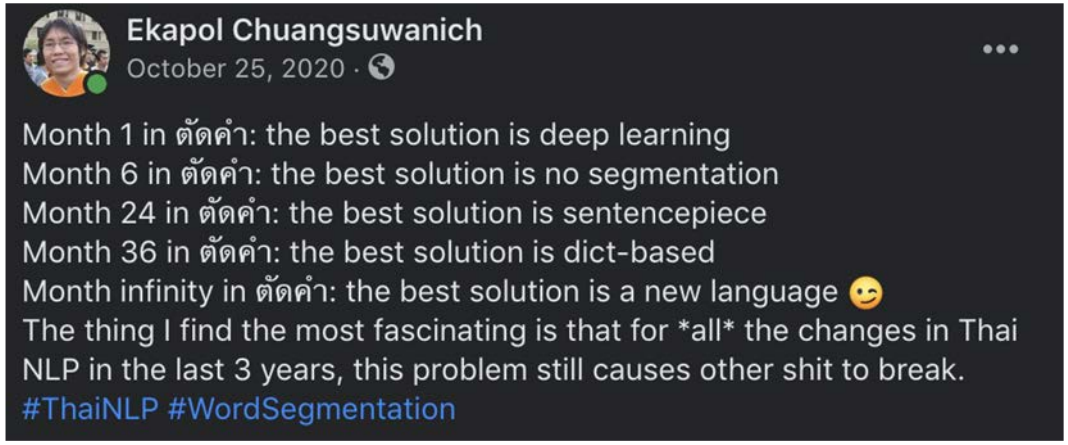
Words of caution

Tokenization performance effects downstream task performance

Can be small (1%) or large (10%)

Specialized tokenizer can help your downstream task

Example: e-commerce search |หู|ฟัง| |ต่าง|หู|



Words of caution

Be careful of what tokenization you used to train the model.

If there's a mismatch in training and testing tokenization, the results can be devastating.

TrueVoice

Training	Testing tokenization			
Deepcut	Deepcut	Longest matching+ noise0.1	Longest matching+ noise0.4	Longest matching+ noise0.7
	76.8	60.4	50.9	42

Wisesight1000

Training	Testing			
Manual	Manual	Longest matching+ noise0.1	Longest matching+ noise0.4	Longest matching+ noise0.7
	52.1	48.2	38.1	32.7

https://ieeexplore.ieee.org/document/9836268

Tokenization - English

- Even English has tokenization issues!
  - Space is usually not enough
  - aren't
    - are + n't
    - aren't
    - arent
    - aren t
    - are + not
  - San Francisco

- Usually includes the text normalization step
- This depends on application
  - "aren't" might be different from "are not" for sentiment analysis

Other English issues - hyphens

- “*the New York-based co-operative was fine- tuning forty-two K-9-like models.* “

- Lexical vs Sentential hyphens

Tokenization of non-standard text

- Twitter

@SentimentSymp: can't wait for the Nov 9 #Sentiment talks! YAAAAAAY!!! >:-D http://sentimentsymposium.com/.

Needs to correctly tokenize

- Emoticons
- Twitter markup (# and @)
- Capitalization (and html tags for bold, etc.)
- Lengthening
- Dates

http://sentiment.christopherpotts.net/tokenizing.html

Hand-crafted for tweets

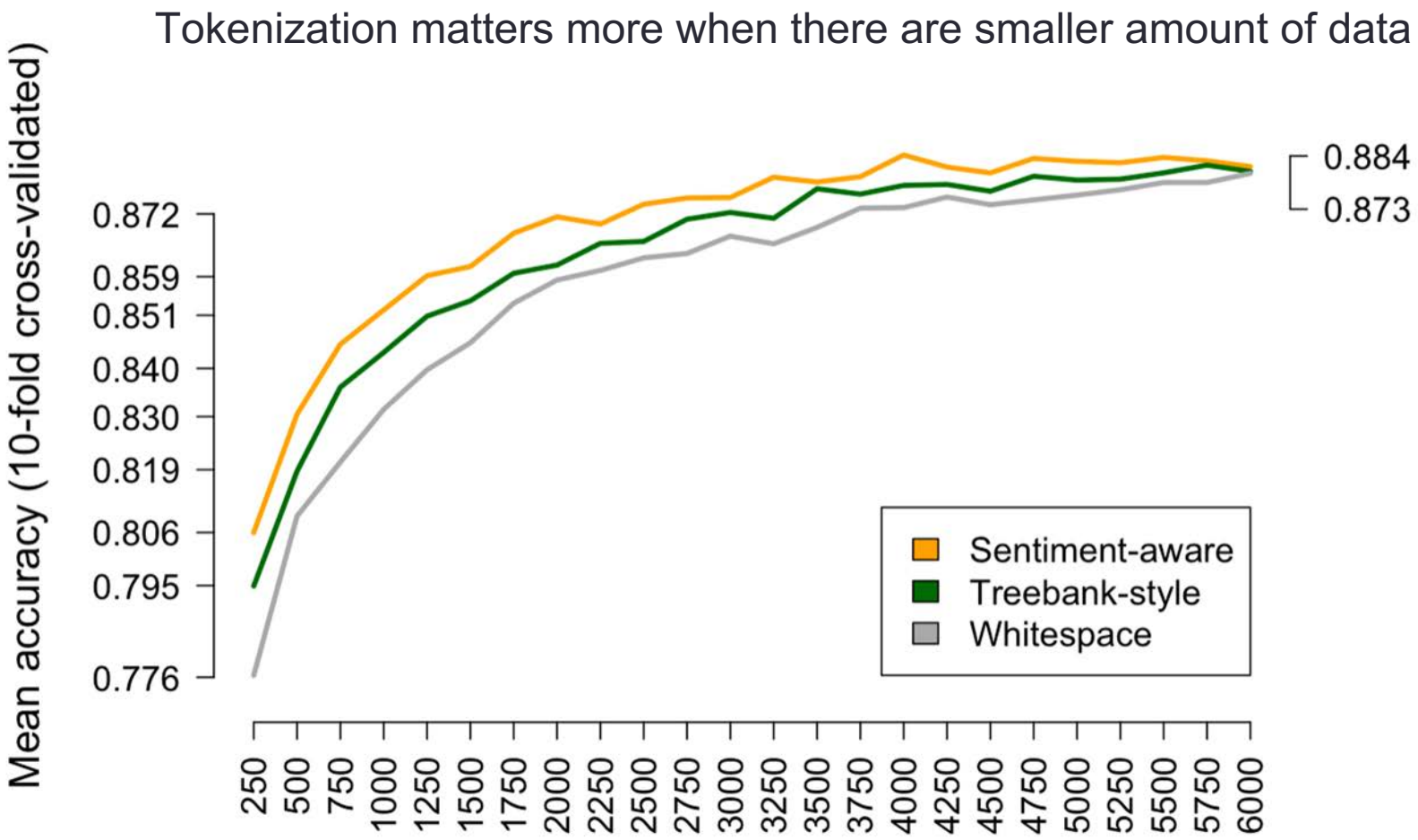
```
@sentimentsymp
:
can't
wait
for
the
Nov_09
#sentiment
talks
!
YAAAY
!
!
!
>:-D
http://sentimentsymposium.com/
.
```

Standard tokenizer
(Stanford tokenizer)

```
@
SentimentSymp
:
ca
n't
wait
for
the
Nov
9
#
Sentiment
talks
!
YAAAAAAY
!
!
!
&gt;
;
:
-D
http
:
//sentimentsymposium.com/
.
```

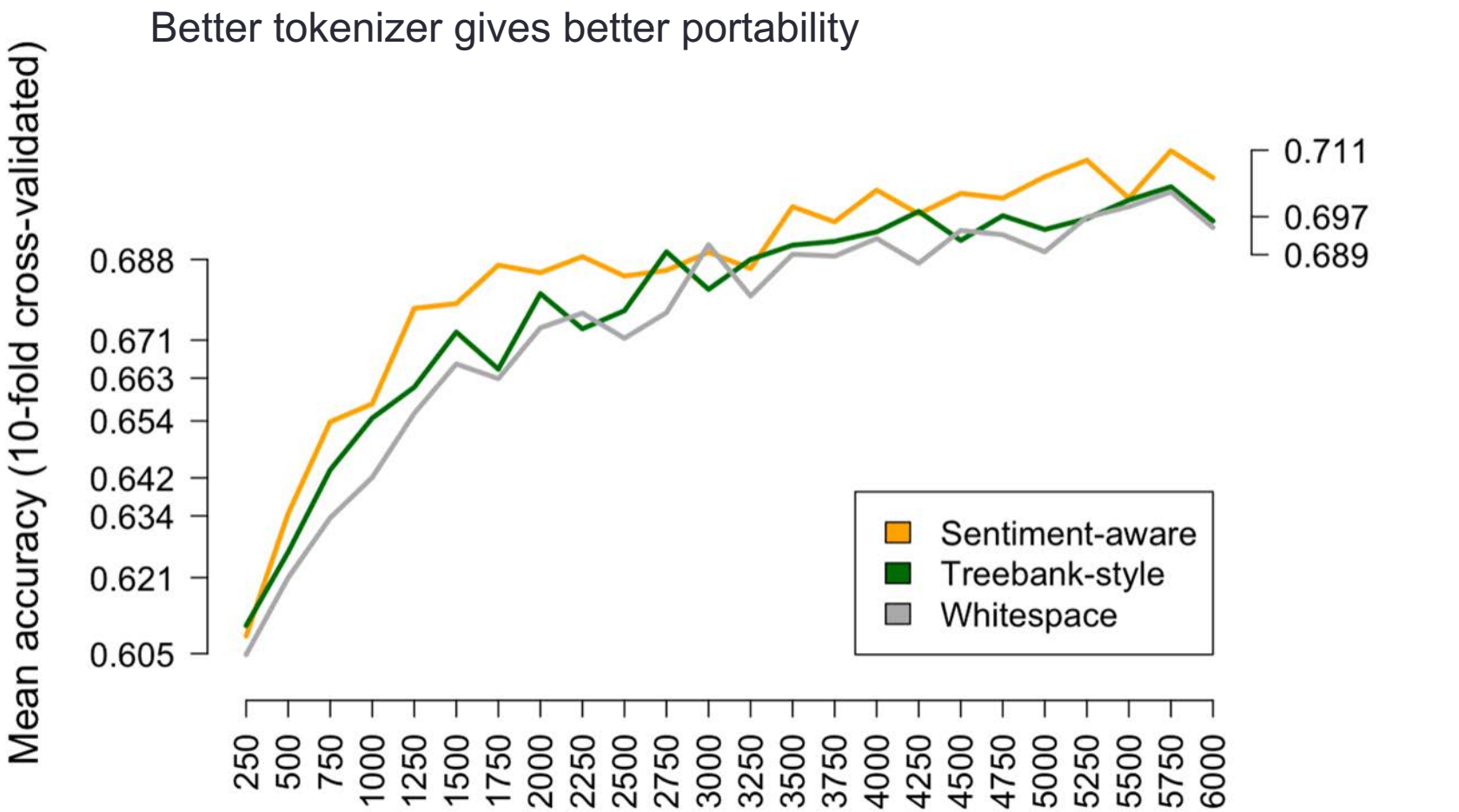
http://sentiment.christopherpotts.net/tokenizing.html

Sentiment analysis (in-domain)



http://sentiment.christopherpotts.net/tokenizing.html

Sentiment analysis (out-of-domain)



http://sentiment.christopherpotts.net/tokenizing.html

End-to-end models

- Classical machine learning systems usually break the problem into smaller subtasks
  - Self-driving:
    - Image -> objects detection -> path finding -> steering
  - Speech2speech translation:
    - Speech A -> text A -> text B -> Speech B
- End-to-end models use one large neural networks process the input and generate the desired output
  - Image -> steering
  - Speech A -> Speech B

End-to-end NLP?

Discourse

Semantics

Syntax: Constituents

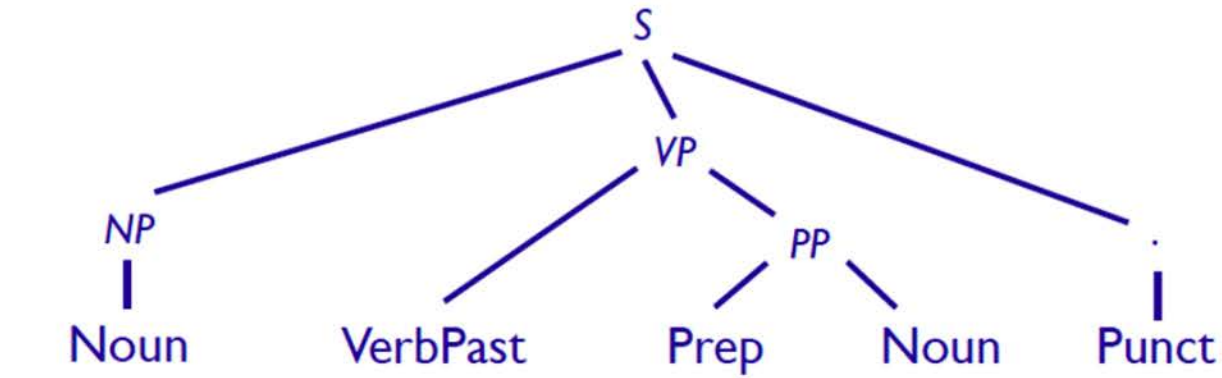
Syntax: Part of Speech

Words

Morphology

Characters

CommunicationEvent(e)
Agent(e, Alice)
Recipient(e, Bob)
SpeakerContext(s)
TemporalBefore(e, s)



talk -ed [VerbPast]

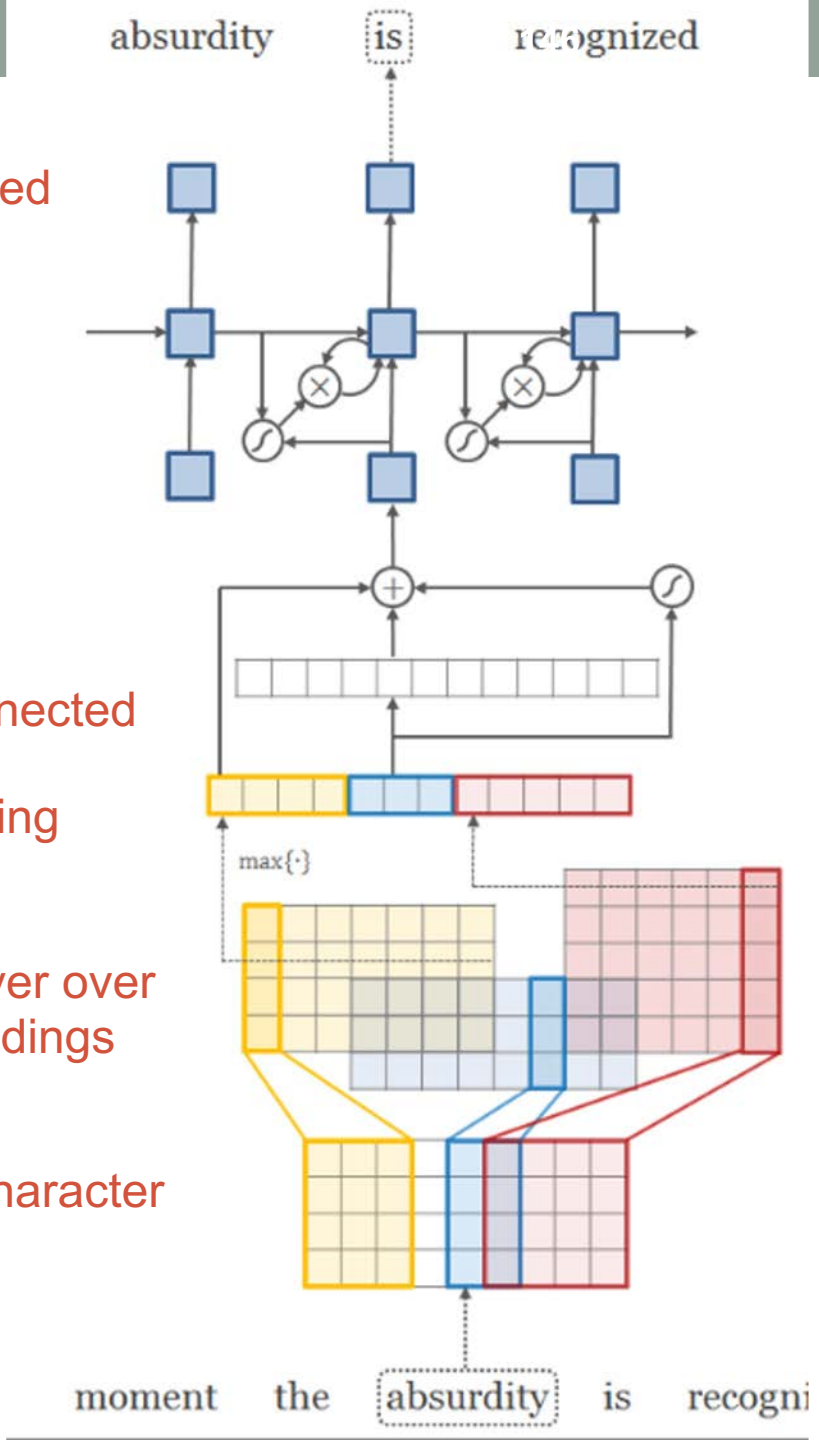
Alice talked to Bob.

Ref: Prof. Brendan O'Connor, CS 585 Intro to NLP, @UMass

Towards no tokenization? :Character-aware neural language models

- Input: previous characters
- Output: next word

CNN over characters capture character sequence patterns



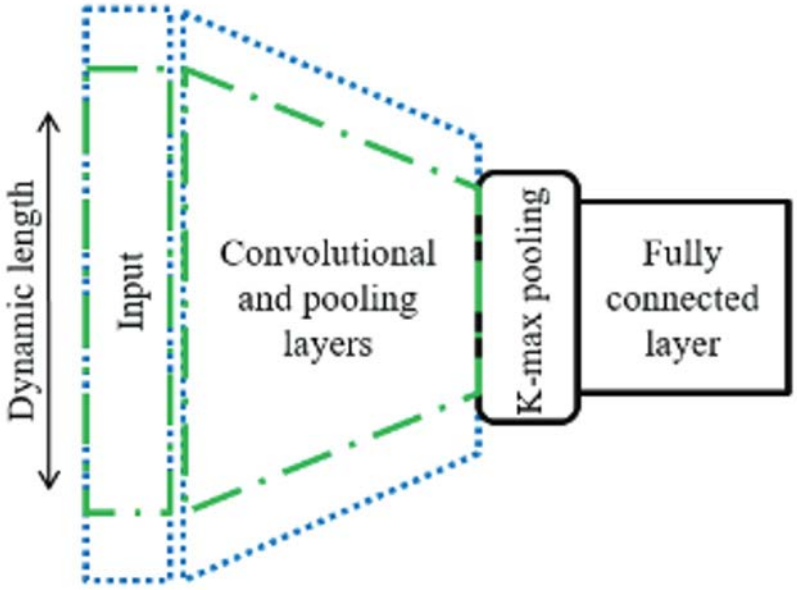
https://arxiv.org/pdf/1508.06615.pdf

Towards no tokenization

- Text classification using charCNN on Thai

Method	Accuracy (%)	F_1 (%)
Naïve Bayes, BoW	87.2	87.1
Naïve Bayes, TF-IDF	89.0	88.9
Logistic Regression, BoW	94.8	94.8
Logistic Regression, TF-IDF	94.7	94.7
SVM, BoW	93.7	93.7
SVM, TF-IDF	95.2	95.2
DCNN (Kalchbrenner et al., 2014)	**95.9**	**95.9**
Proposed Char-CNN	95.4	95.4

Word-based methods



A character-level convolutional neural network with dynamic input length for Thai text categorization

http://ieeexplore.ieee.org/document/7886102/

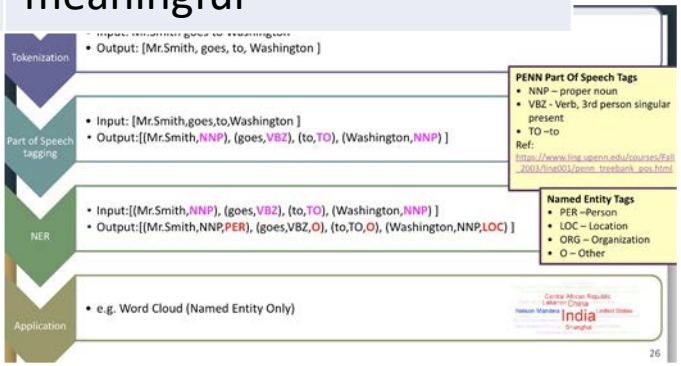
Caveats of end-to-end models

- Requires lots of data for the specific task
- Hard to fix specific mistakes by the model

Things to consider when thinking about tokenization

Know your use cases

Word	Subword	Character
Large vocabulary	Medium vocabulary	Small vocabulary
Can use simpler model	Moderate complexity in modeling	Needs a powerful model to learn long range influences
High OOVs	Few OOVs	No OOVs
Individual tokens are meaningful	Individual tokens might be meaningful	Individual tokens are not meaningful



Conclusion

- Neural network is cool
  - Pick the kind of layers that suit the nature of your task
- Tokenization is far from solved but don't let this discourage you
  - No tokenization is perfect
  - Pick one that is suited for your task
    - Speed
    - Robustness to misspelling and unseen words
    - Consistency
    - Certain tools assume you are using a particular type of tokenization, check!