NEURAL NETWORKS

TrueVoice Training

INTRO TO LINEAR REGRESSION

Predicting amount of rainfall

https://esan108.com/%E0%B8%9E%E0%B8%A3%E0%B8%B0%E0%B9%82%E0%B8%84%E0%B8%81%E0%B8%B4%E0%B8%99%E0%B8%AD%E0%B8%B0%E0%B9%84%E0%B8%A3-%E0%B8%AB%E0%B8%A1%E0%B8%B2%E0%B8%A2%E0%B8%96%E0%B8%B6%E0%B8%87.html

Predicting amount of rainfall

Cloth	Corn	Grass	Water	Beer	Rainfall
4	6	3	10	0	**76950**
5	1	0	0	7	**30234**
6	0	3	5	7	**123456**
5	0	3	12	0	**89301**
4	3	0	6	7	**?**

We assume the input features have some correlation with the amount of rainfall.

Can we create a model that predict the amount of rainfall?

What is the output?

What is the input (features)?

Predicting the amount of rainfall

Cloth	Corn	Grass	Water	Beer	Rainfall
4	6	3	10	0	**76950**
5	1	0	0	7	**30234**
6	0	3	5	7	**123456**
5	0	3	12	0	**89301**
4	3	0	6	7	**?**

- $h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4 + \theta_5 x_5$
- Where θs are the parameter of the model
- Xs are values in the table

(Linear) Regression

- $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4 + \theta_5 x_5$
- θs are the parameter (or weights)
- We can rewrite

Assume x_0 is always 0

$$h_\theta(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T \mathbf{x}$$

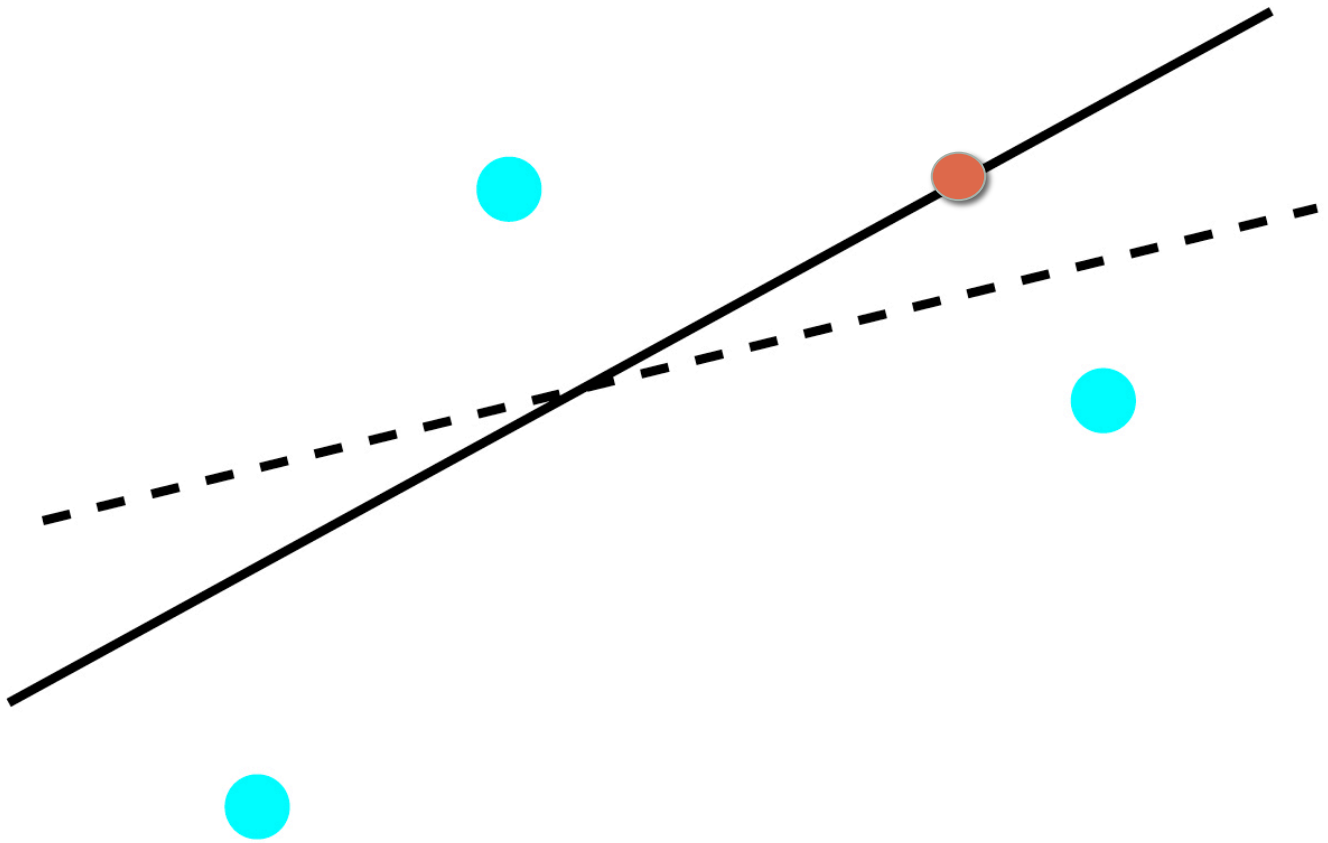
- Notation: vectors are bolded
- Notation: vectors are column vectors

Picking **θ**

- In order to pick **θ** we need to quantify performance.
- How to quantify best performance?

$$h_\theta(x) = \Sigma_{i=0}^{n} \theta_i x_i = \theta^T \mathbf{x}$$

$$h_\theta(x) = \Sigma_{i=0}^{n} \theta_i x_i = \theta^T \mathbf{x}$$



https://chem.libretexts.org/Textbook_Maps/Analytical_Chemistry_Textbook_Maps/Map%3A_Analytical_Chemistry_2.0_(Harvey)/05_Standardizing_Analytical_Methods/5.4%3A_Linear_Regression_and_Calibration_Curves

Cost function (Loss function)

- Let's use the mean square error (MSE)

$$J(\theta) = \frac{1}{m}\Sigma_{i=1}^{m}(y_i - \theta^T \mathbf{x_i})^2$$

We want to pick **θ** that minimize the loss

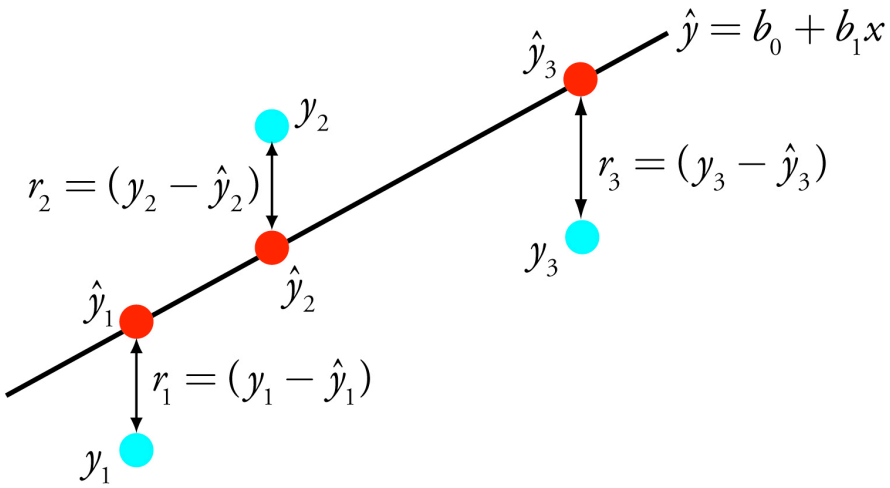
m is number of training data points

Cost function (Loss function)

- Let's use the mean square error (MSE)

$$J(\theta) = \frac{1}{m}\Sigma_{i=1}^{m}(y_i - \theta^T \mathbf{x_i})^2$$

We want to pick **θ** that minimize the loss



Cost function (Loss function)

- Let's use the mean square error (MSE)

$$J(\theta) = \frac{1}{m}\Sigma_{i=1}^{m}(y_i - \theta^T \mathbf{x_i})^2$$

We want to pick **θ** that minimize the loss

Scaling

$$\frac{m}{2}J(\theta) = \frac{1}{2}\Sigma_{i=1}^{m}(y_i - \theta^T \mathbf{x_i})^2$$

Same **θ** minimizes both function

Picking $\boldsymbol{\theta}$

- How to quantify best performance?
  - Square error loss function

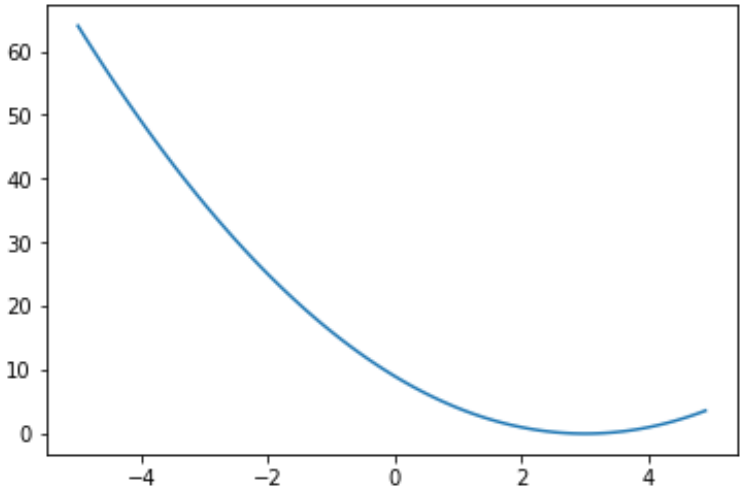
$$J(\theta) = \frac{1}{2}\Sigma_{i=1}^{m}(y_i - \theta^T \mathbf{x_i})^2$$

- How to pick $\boldsymbol{\theta}$ to minimize J
  - Random until you get the best performance?
  - Can we do better than random chance?

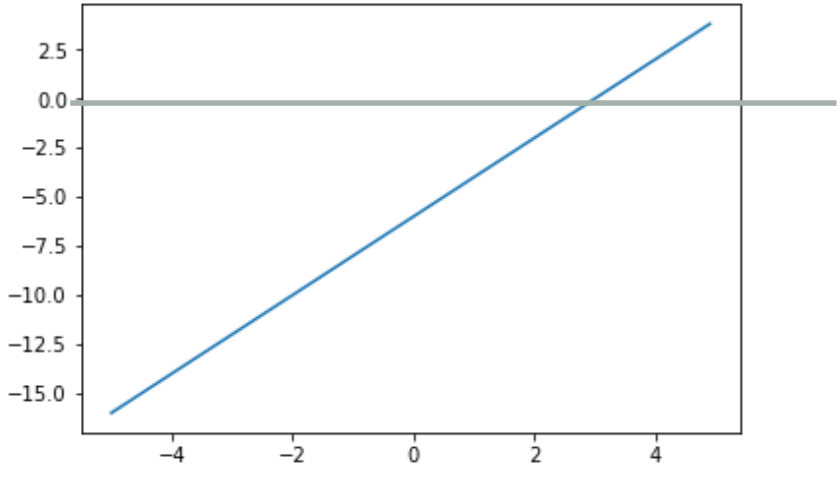
Minimizing a function

- 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



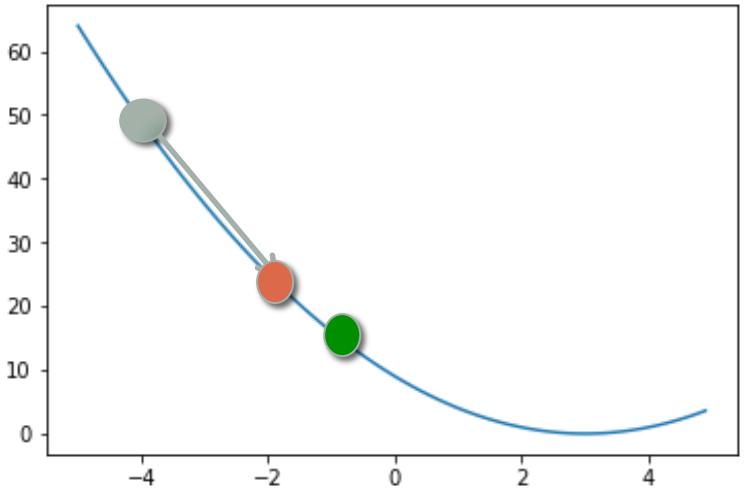
y



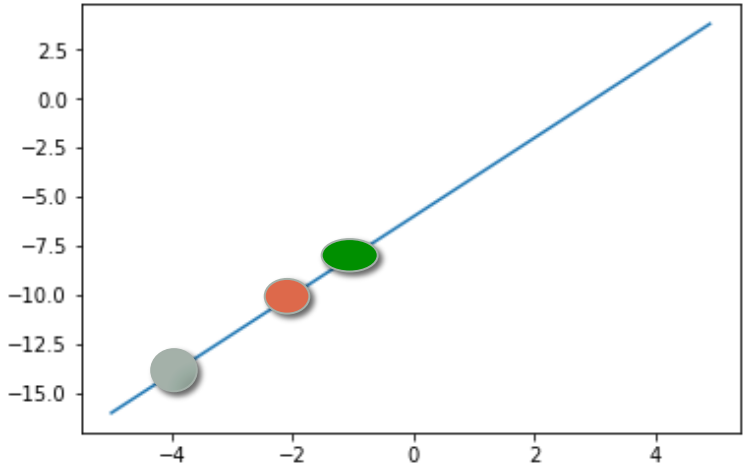
dy/dx

First what does dy/dx means?

Gradient descent



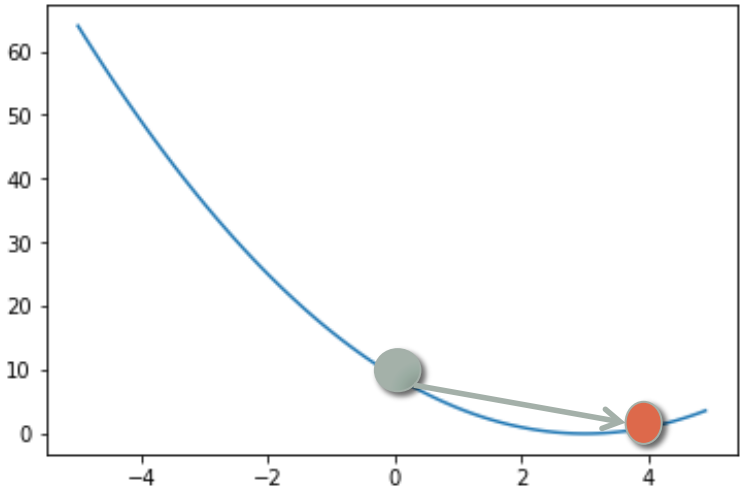
y



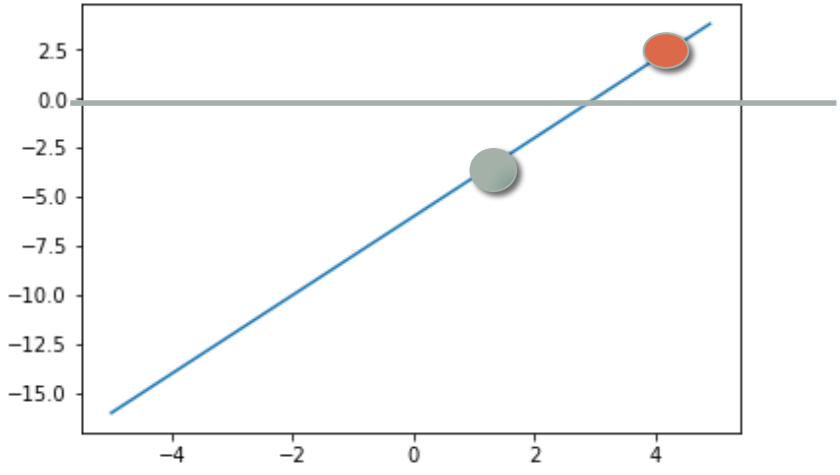
dy/dx

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

Gradient descent



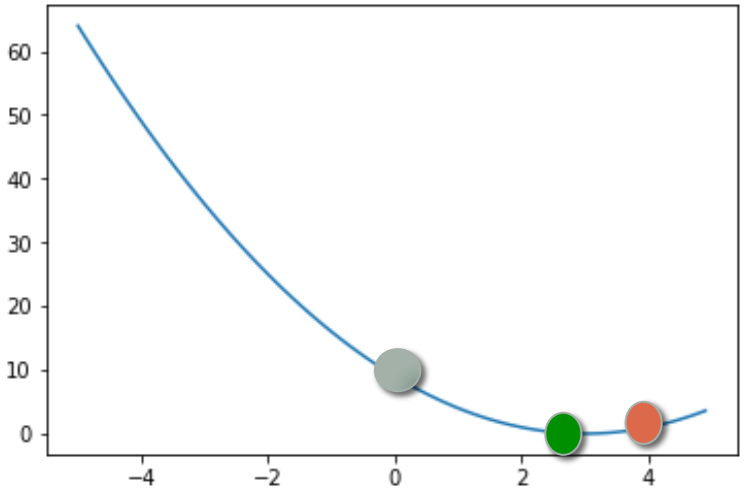
y



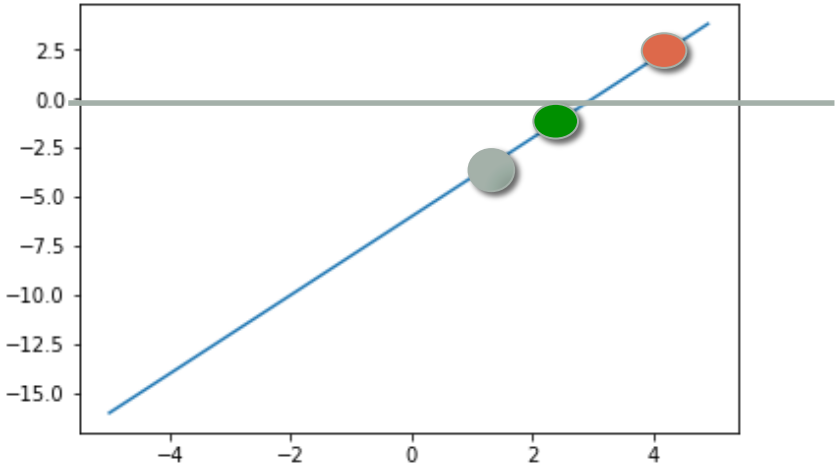
dy/dx

What happens when you overstep?

Gradient descent



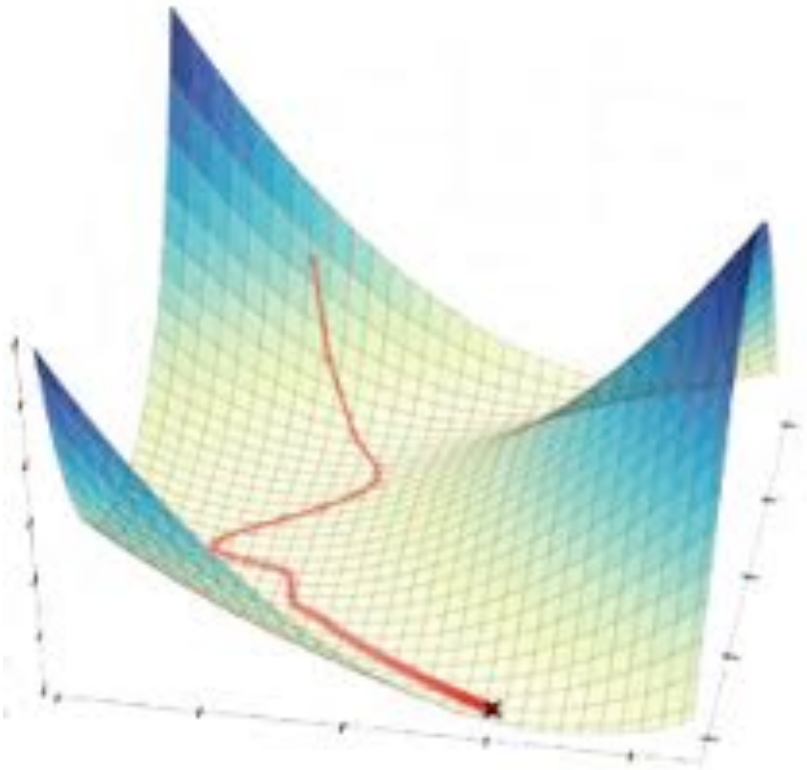
y



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

Picking **θ**

- How to quantify best performance?
  - Square error loss function

$$J(\theta) = \frac{1}{2}\Sigma_{i=1}^{m}(y_i - \theta^T \mathbf{x_i})^2$$

- How to pick **θ** to minimize J
  - Gradient descent

LMS regression with gradient descent

$$\frac{\partial J}{\partial \theta_j} = -\Sigma_{i=1}^{m}(y_i - \theta^T \mathbf{x}_i)x_i^{(j)}$$

$$\theta_j \Leftarrow \theta_j + r\Sigma_{i=1}^{m}(y_i - \theta^T \mathbf{x}_i)x_i^{(j)}$$

m = number of training example
i = index of training example
j = index of feature dimension

$\mathbf{x}_i$

$x^{(1)}$
$x^{(2)}$
$x^{(3)}$
…
$x^{(j)}$
…
$x^{(J)}$

Batch updates vs mini-batch

$$\theta_j \Leftarrow \theta_j - r\Sigma_{i=1}^{m}(y_i - \theta^T \mathbf{x}_i)x_i^{(j)}$$

- Batch updates (considering the whole training data) estimate the Loss function precisely
  - Can takes a long time if m is large
- Updates with a subset of m
  - We now have an estimate of the loss function
    - This can lead to a wrong direction, but we get faster updates
  - Called Stochastic Gradient Descent (SGD) or incremental gradient descent

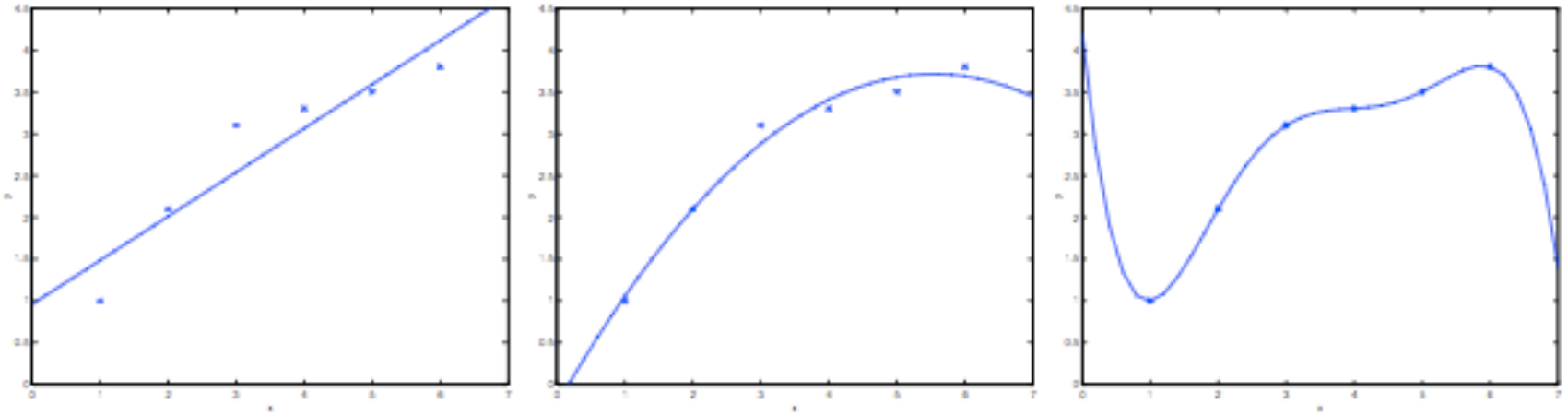
Regression with non-linear features

- If we add extra features that are non-linear
  - For example x^2

Cloth	Corn	Grass	Water	Beer	Rainfall
4	6	3	10	0	**76950**
5	1	0	0	7	**30234**
6	0	3	5	7	**123456**
5	0	3	12	0	**89301**
4	3	0	6	7	**?**

- $h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4 + \theta_5 x_5 + \theta_6 x_1^2 + \ldots$
- These can be considered as additional features
- We can now have a line that is non-linear

Overfitting Underfitting



Adding more non-linear features makes the line more curvy
(Adding more features also means more model parameters)

The curve can go directly to the outliers with enough parameters.
We call this effect overfitting
For the opposite case, having not enough parameters to model the data is called underfitting

Predicting floods

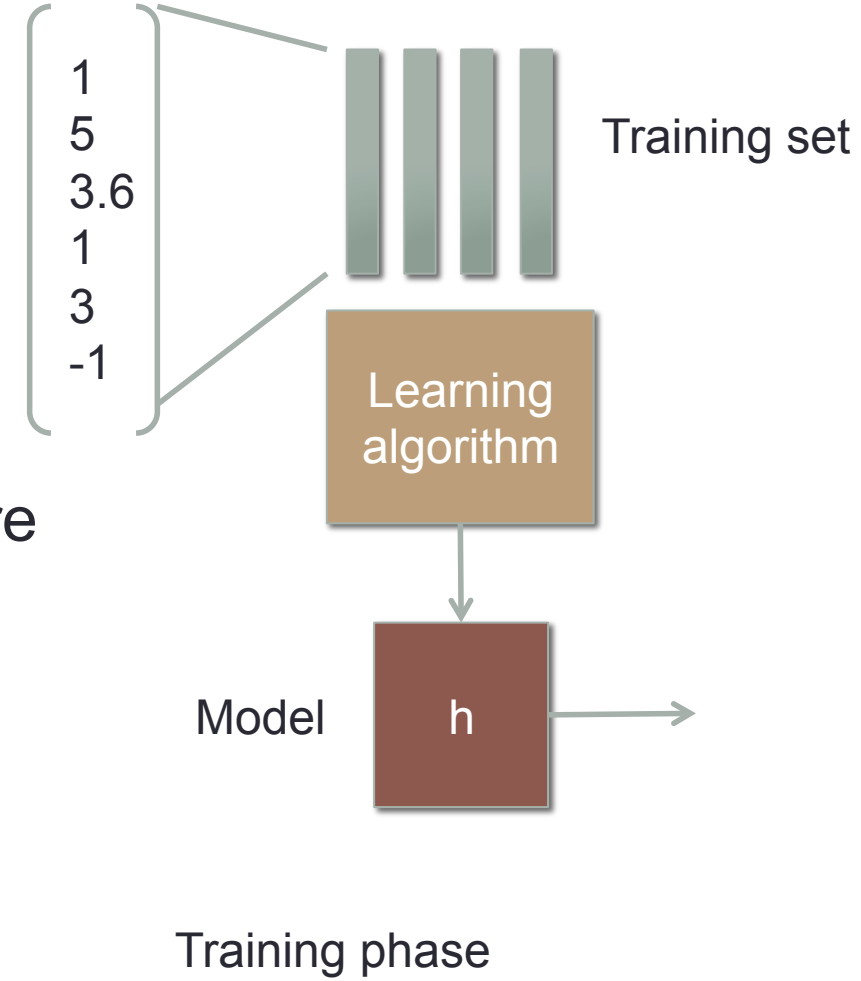
Cloth	Corn	Grass	Water	Beer	Flood?
4	6	3	10	0	**yes**
5	1	0	0	7	**yes**
6	0	3	5	7	**no**
5	0	3	12	0	**yes**
4	3	0	6	7	**?**

So far we talk about predicting an amount what if we want to do classification

Let’s start with a binary choice. Flood or no flood

Flood or no flood

- What would be the output?
- y = 0 if not flooded
- y = 1 if flooded
- Anything in between is a score for how likely it is to flood



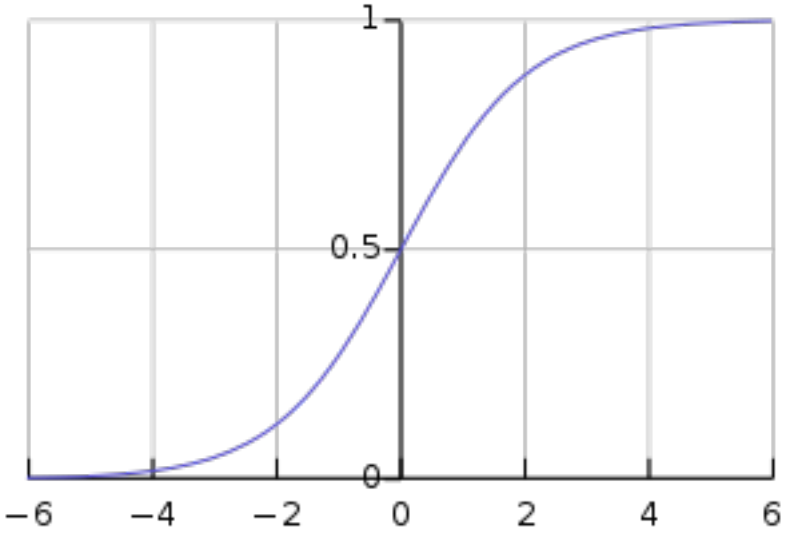
Can we use regression?

- Yes
- $h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4 + \theta_5 x_5$

- But
- What does it mean when h is higher than 1?
- Can h be negative? What does it mean to have a negative flood value?

Logistic function

- Let's force h to be between 0 and 1 somehow
- Introducing the logistic function (sigmoid function)



$$f(x) = \frac{1}{1+e^{-x}} = \frac{e^{x}}{1+e^{x}}$$

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

Logistic Regression

- Pass $\theta^T \mathbf{x}$ through the logistic function

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

Logistic Regression update rule

$$\theta_j \Leftarrow \theta_j + r\Sigma_{i=1}^{m}(y_i - h_\theta(x_i))x_i^{(j)}$$

Update rule for linear regression

$$\theta_j \Leftarrow \theta_j + r\Sigma_{i=1}^{m}(y_i - \theta^T \mathbf{x}_i)x_i^{(j)}$$

Making a prediction

- Pass $\theta^T \mathbf{x}$ through the logistic function

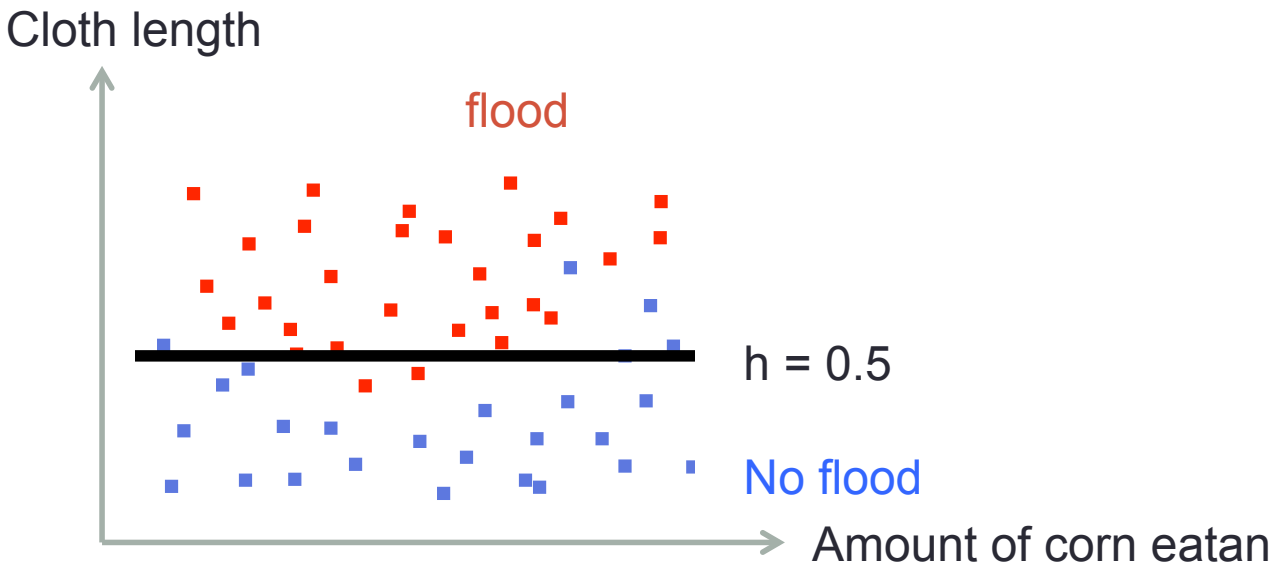
$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

- For a h(x) which value will we say there's a flood
  - Recall our training data
  - y = 0 if not flooded
  - y = 1 if flooded

Decision boundary

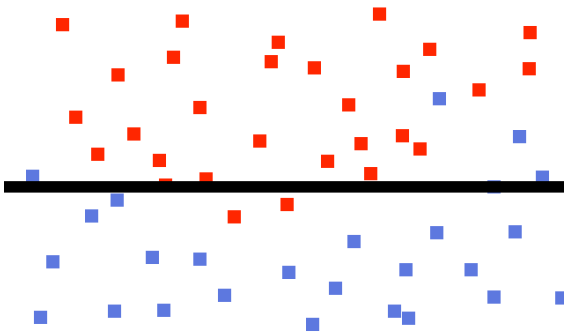
- The line where our decision flips
  - h > 0.5

y = 0 if not flooded
y = 1 if flooded

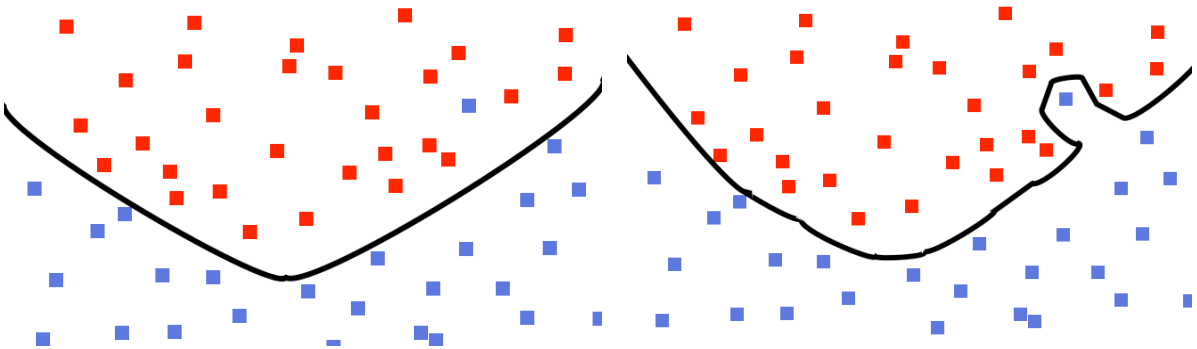


Decision boundary & overfitting

- Usually the more complex the model, the more complex the decision boundary
- One way to gauge the complexity of the model is the number of parameters we need to learn
  - Linear regression with 2 features – 3 features
  - Linear regression with 2 features and x^2 features – 6 features



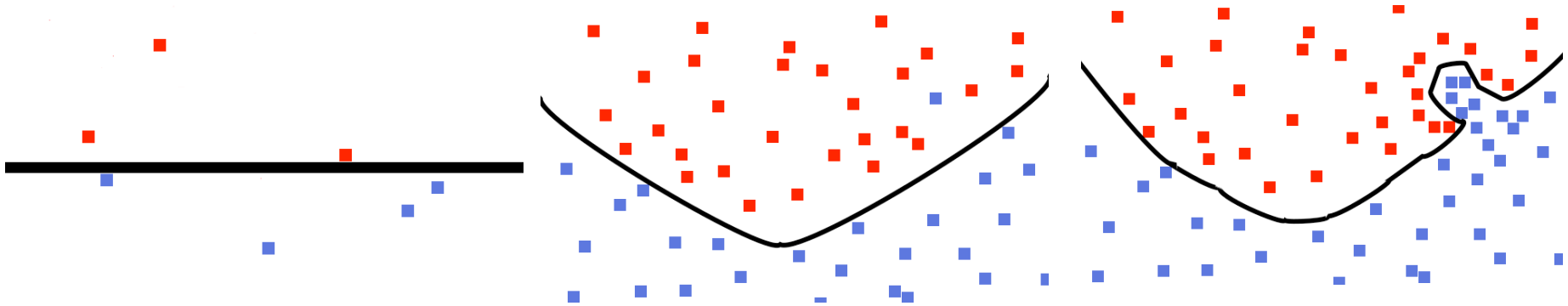
Logistic regression



Logistic regression with non-linear features

Overfitting vs Underfitting

- It is hard to know beforehand how which model is the one that does not overfit or underfit
- The best model complexity depends on amount of data
- The best way to combat overftting is to evaluate multiple models on the dev set
- Data limitation is also known as the data sparsity problem

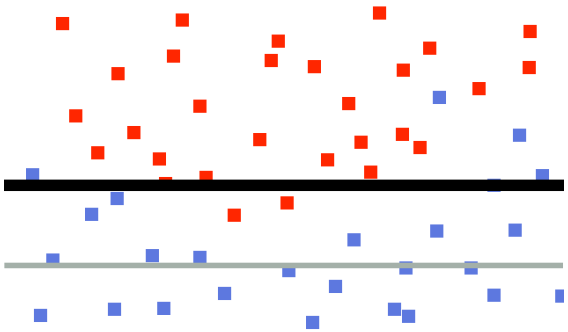


Small amount of training data, pick simpler models

The threshold

- Prediction
- If h > 0.5 flood
- If h < 0.5 no flood

- Do we need to use 0.5?



If we choose 0.3, we will predict more floods. More false alarms. But less likely to miss a non-flood year

With 0.3, all years with floods are predict correctly.

NEURAL NETWORKS

DNNs (Deep Neural Networks)

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

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

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

NLP tasks and Neural networks

	Without Deep Learning	With Deep Learning
Parsing	92.89	**94.61**
Machine Translation	37.0	**41.16**
Sentiment Analysis	81.3	**88.6**
Natural Language Inference	78.2	**89.9**

Source: MIT's NLP course 2017

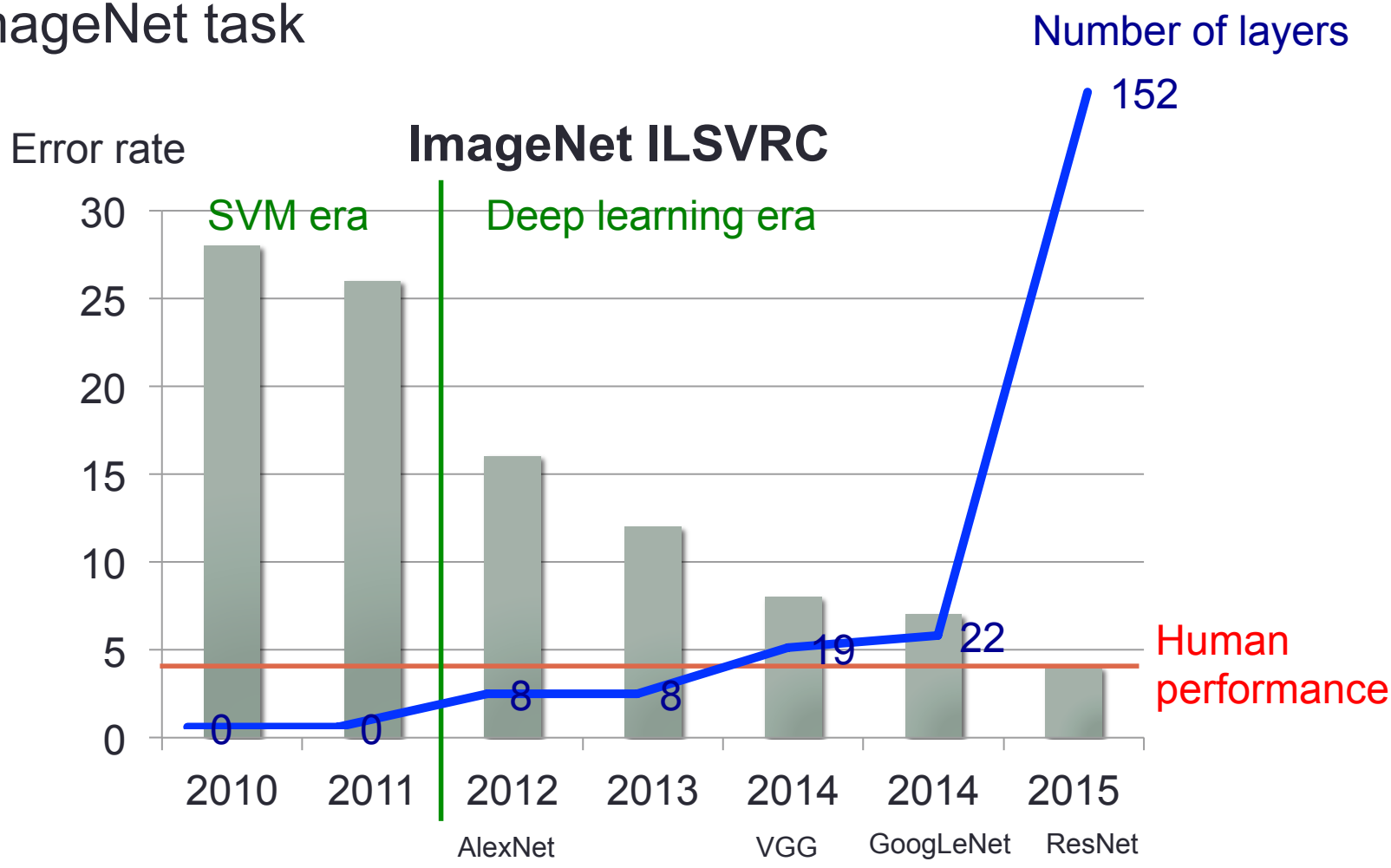
Why now

- 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

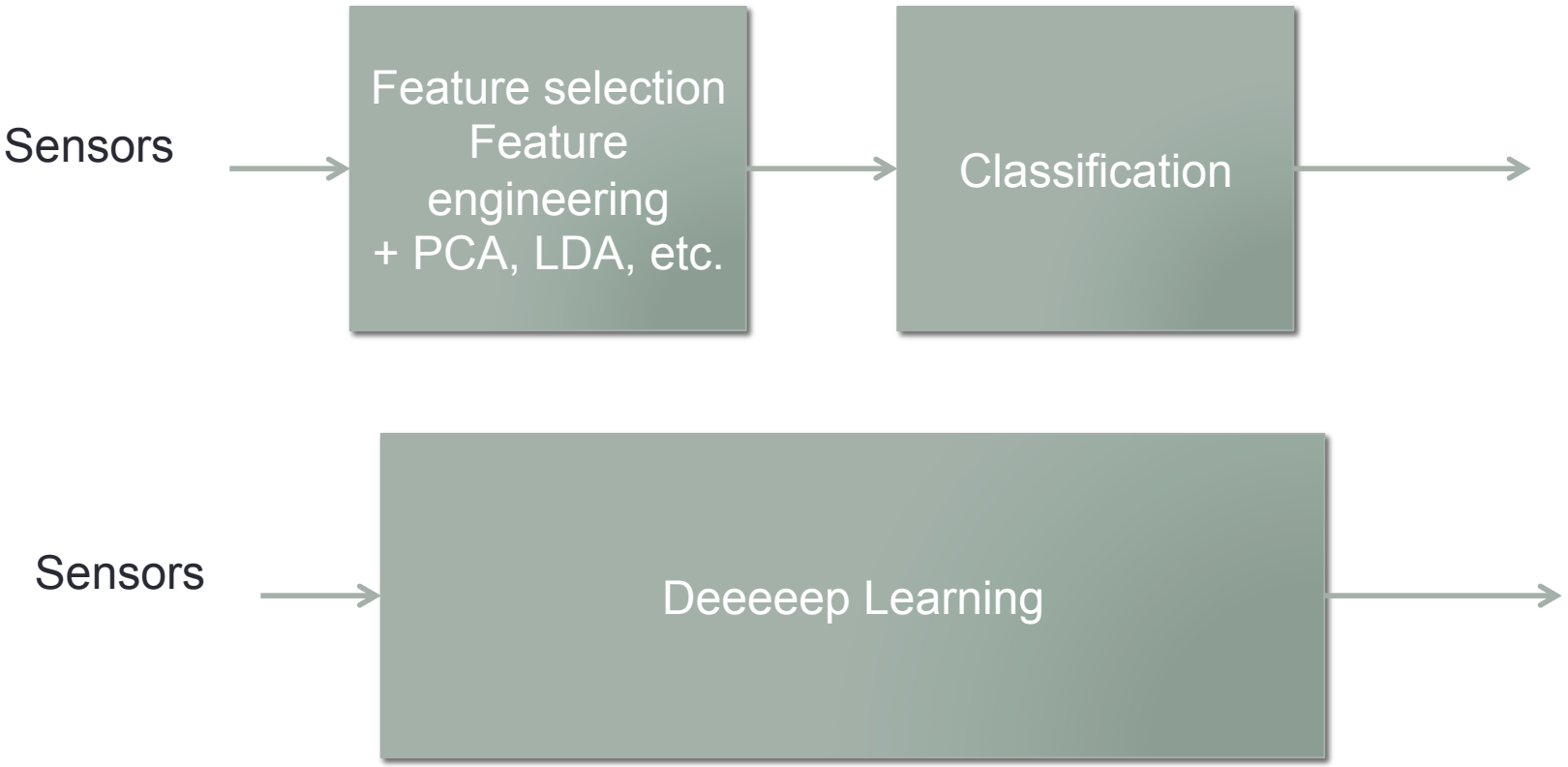


Olga Russakovsky, ImageNet Large Scale Visual Recognition Challenge, 2014 https://arxiv.org/abs/1409.0575

Why is deep learning good

- Traditional machine learning approaches need feature engineering
  - Features are based on human understanding of the phenomena
  - Have some simplifying assumptions
- Human knowledge captures known knowns, and Known unknowns NOT unknown unknowns
- **Deep learning combines features engineering and modeling into one single optimization problem**

Traditional VS Deep learning

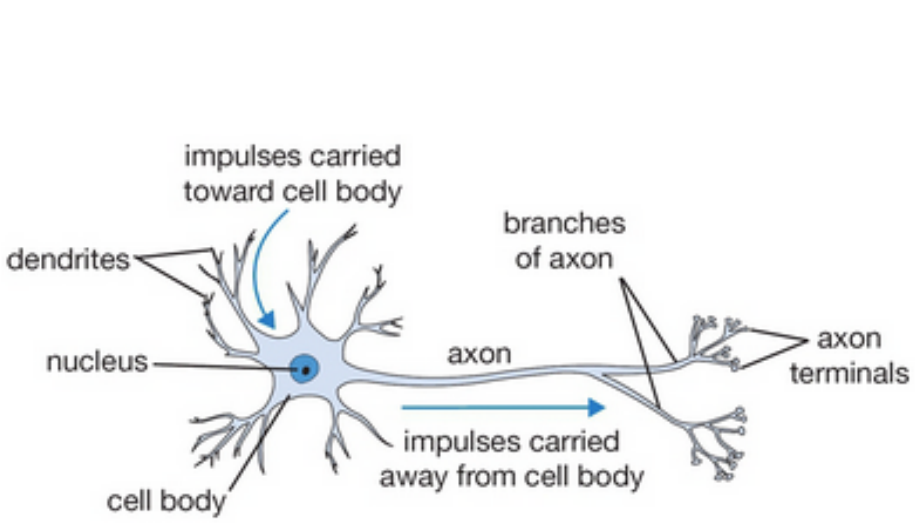


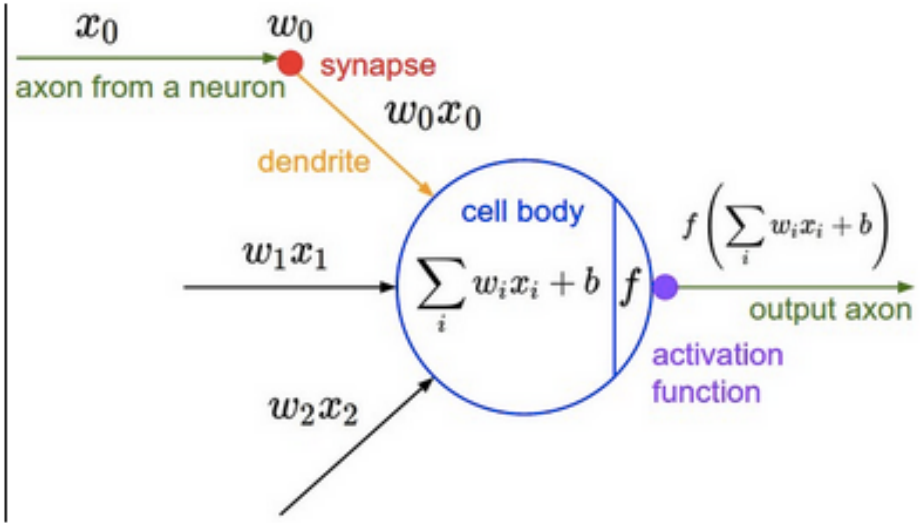
Neural networks

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

Fully connected networks

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

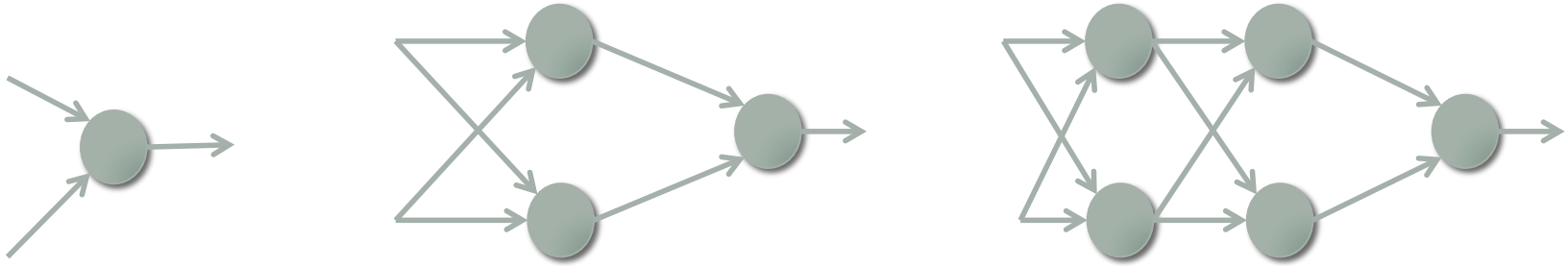


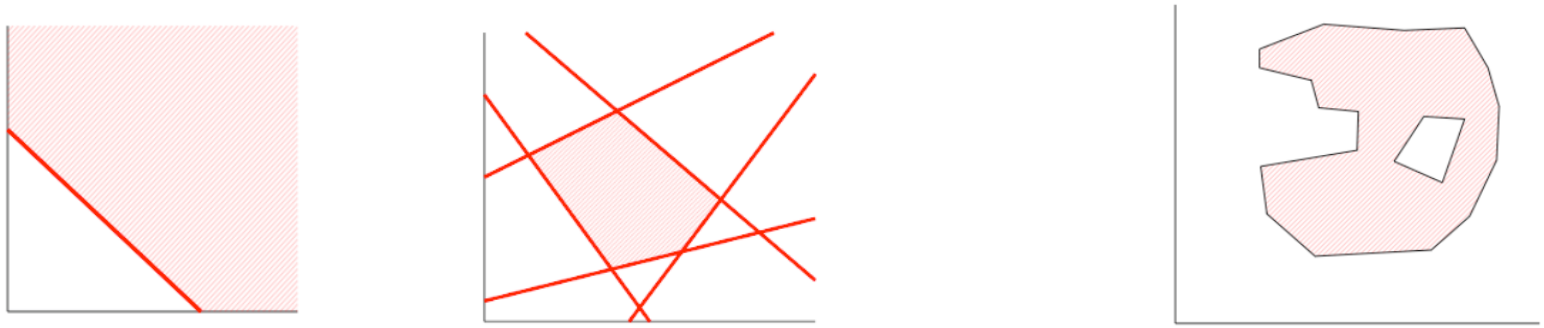


http://cs231n.github.io/neural-networks-1/

Combining neurons

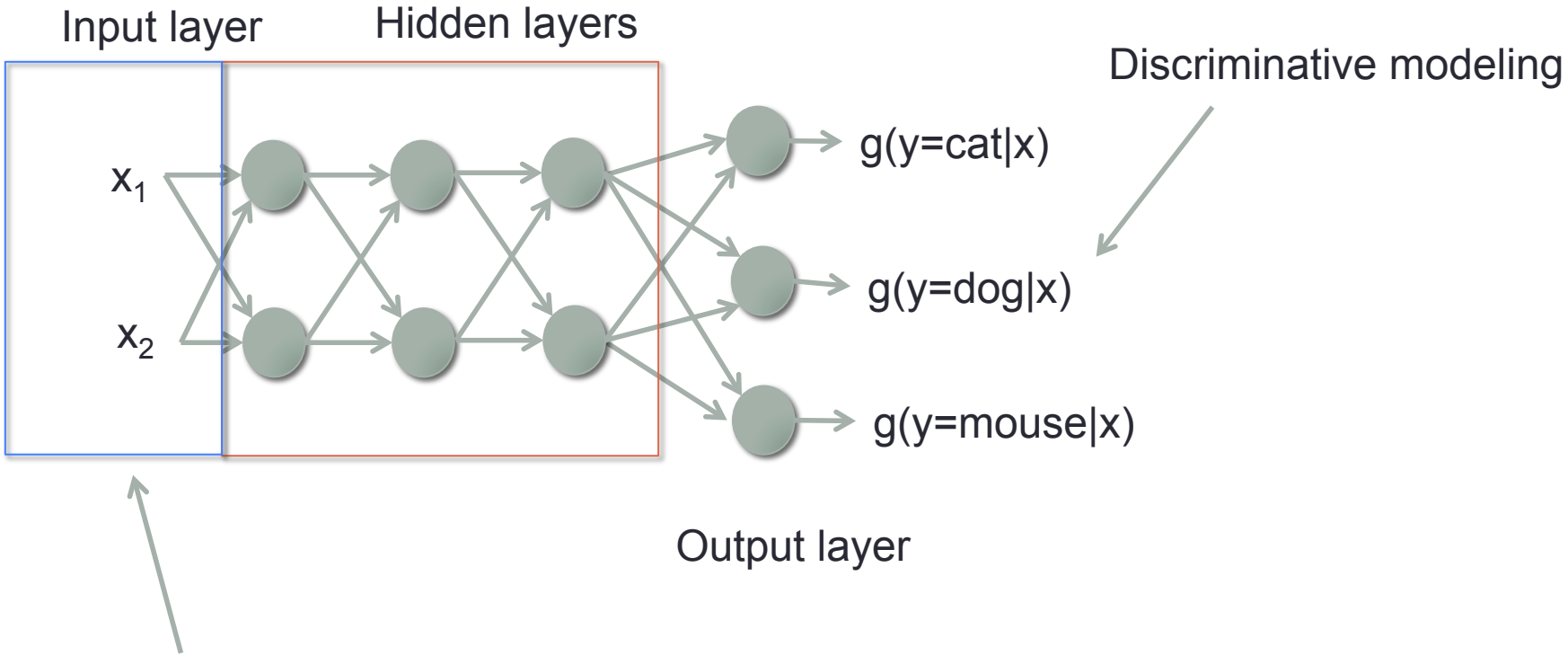
- Each neuron splits the feature space with a hyperplane
- Stacking neuron creates more complicated decision boundaries
- More powerful but prone to overfitting





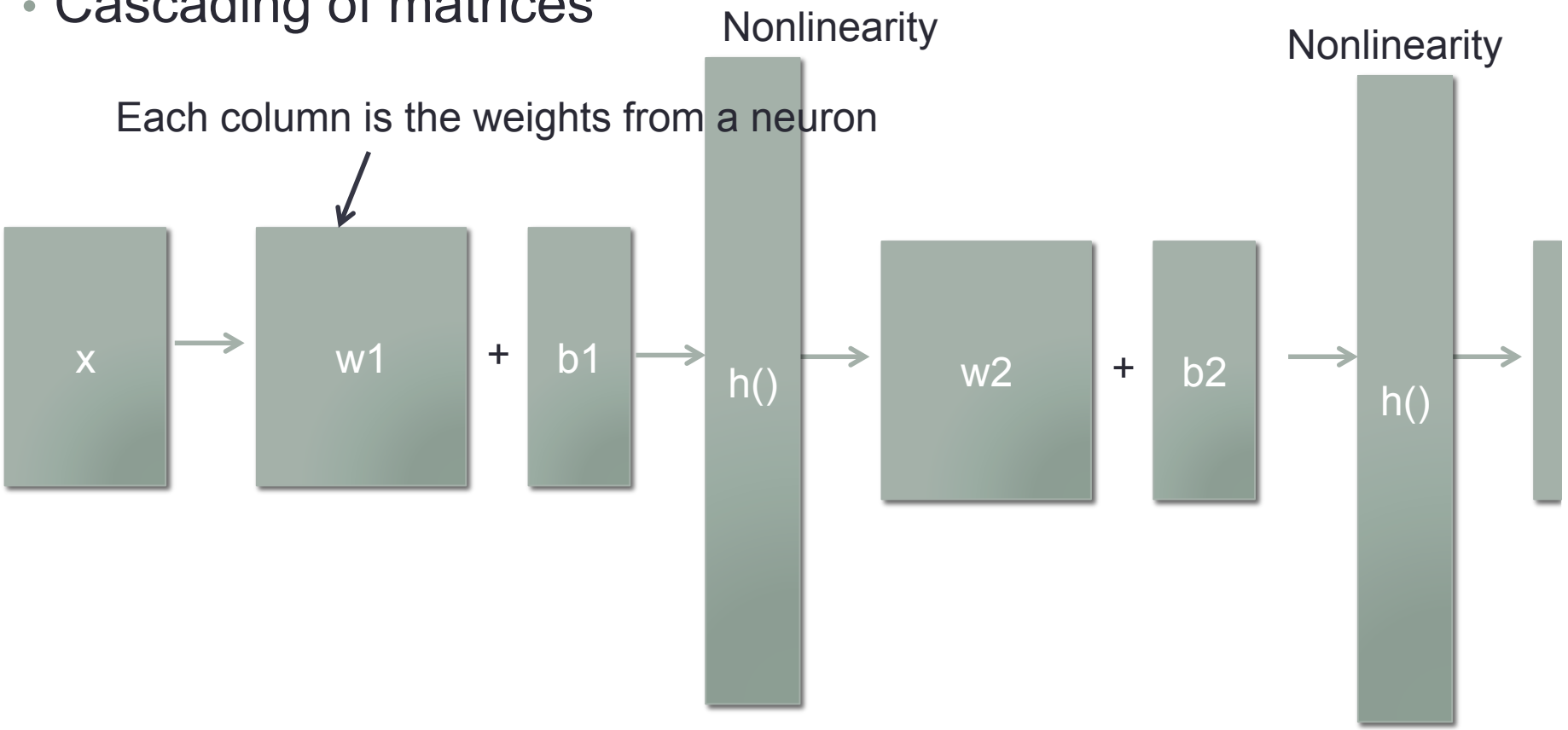
Terminology

Deep in Deep neural networks means many hidden layers



More linear algebra

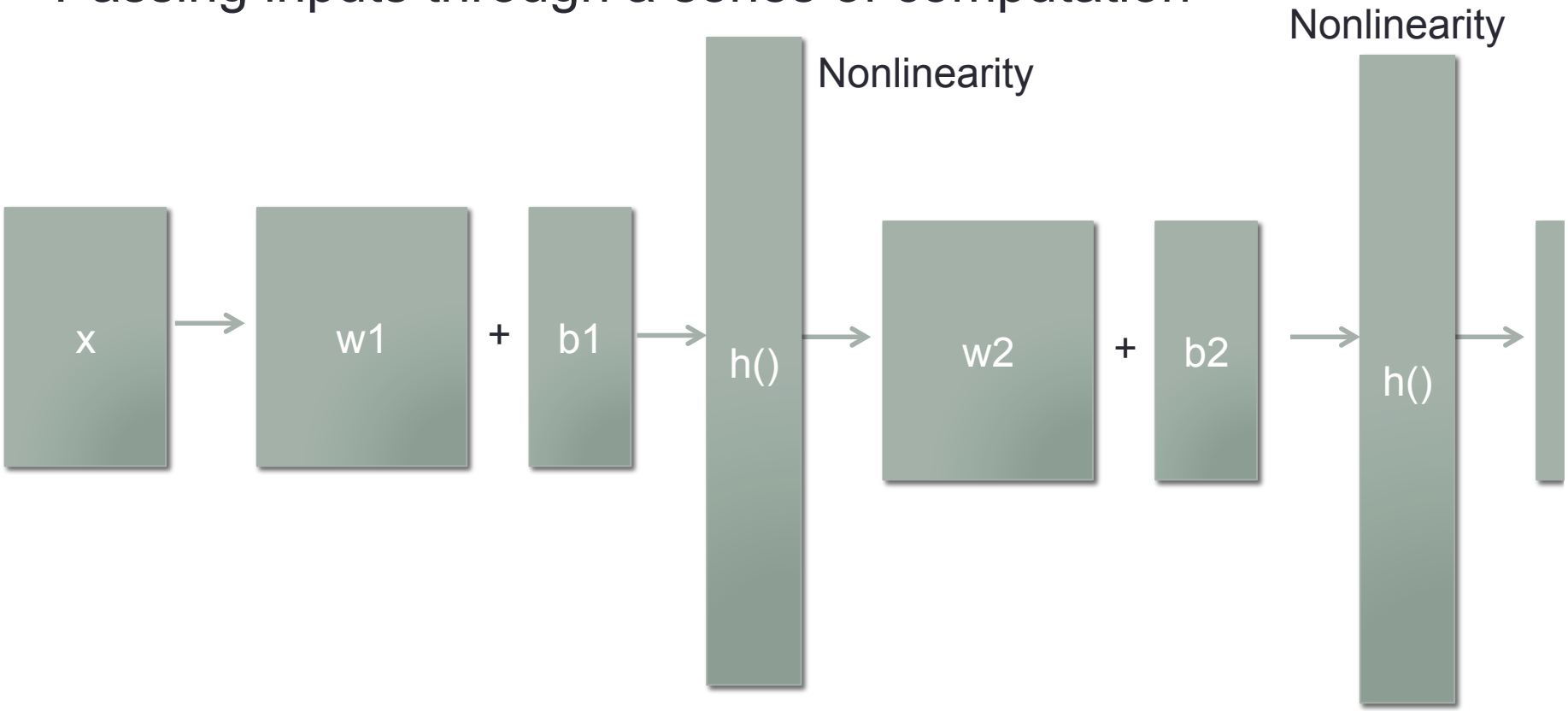
- Cascading of matrices



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

Computation graph

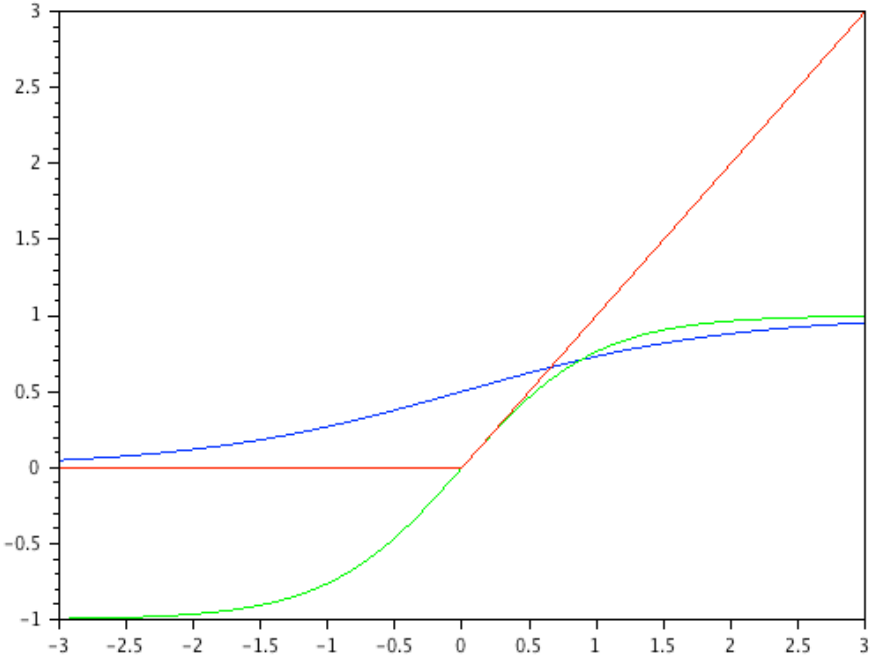
- Passing inputs through a series of computation



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

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)
- Most popular is ReLU and its variants (Fast to train, and more stable)



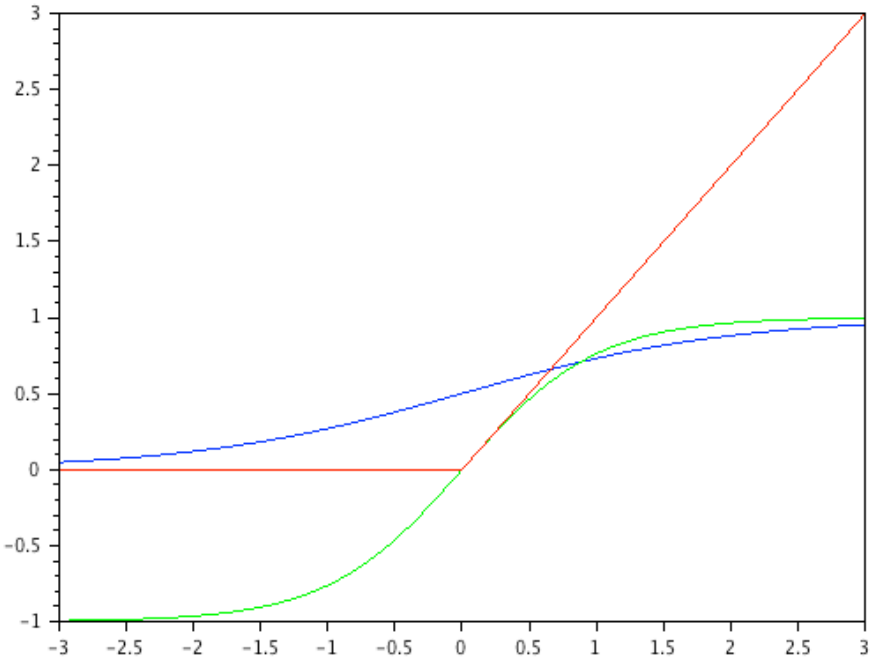
Non-linearity

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

$$tanh(x)$$

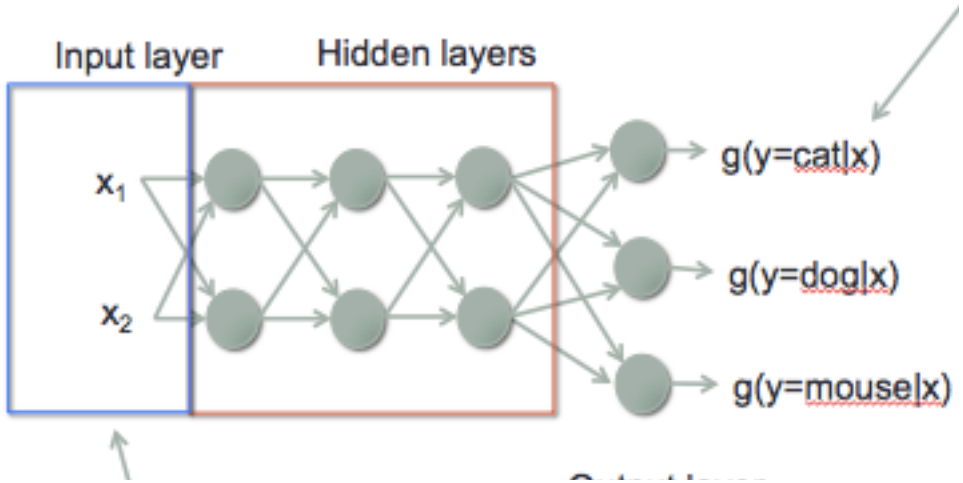
- Rectified Linear Unit (ReLU)

$$max(0, x)$$



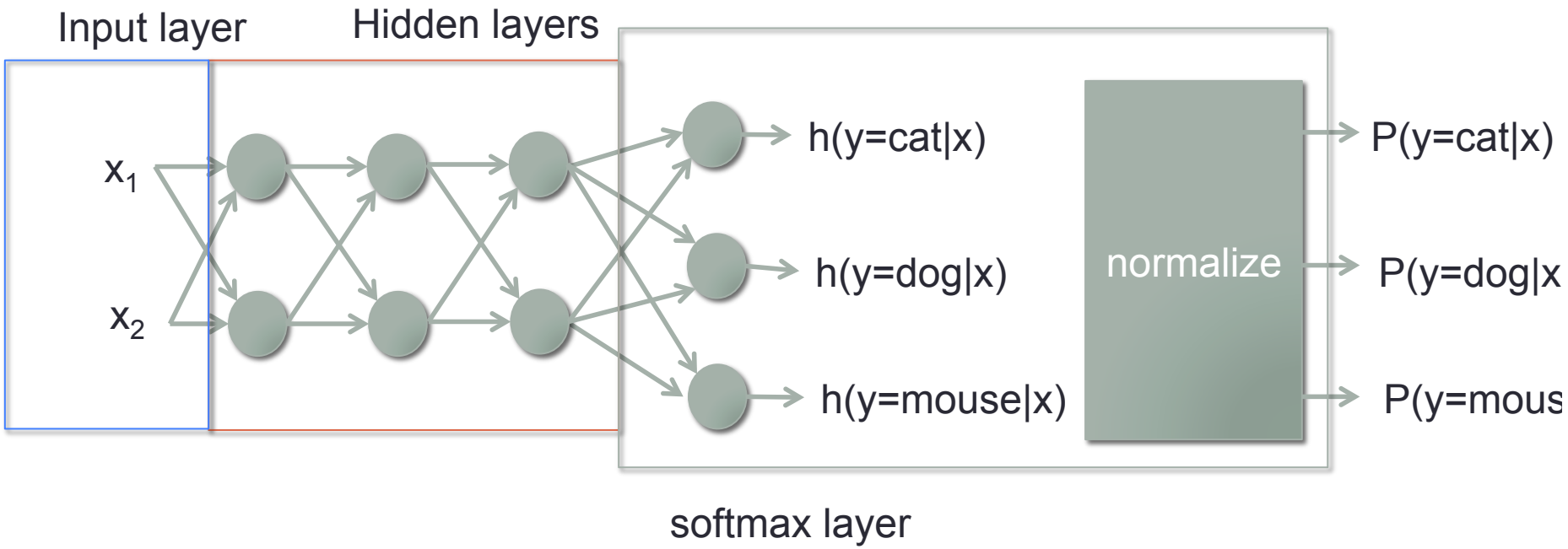
Output layer – Softmax layer

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



Softmax layer

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

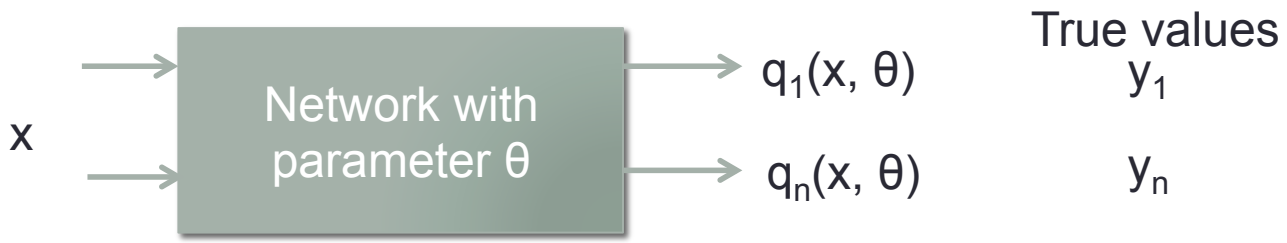


Neural networks

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

Objective function (Loss function)

- Can be any function that summarizes the performance into a single number
- Two main loss functions
  - Cross entropy
  - Sum of square errors



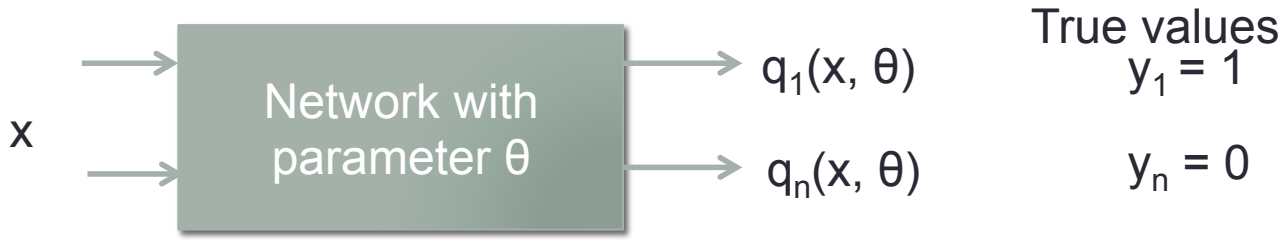
Cross entropy loss

- 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

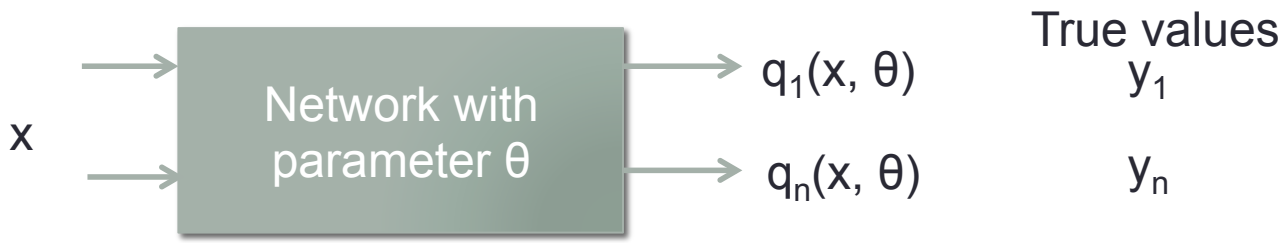


Sum of square errors

- 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



Regularization terms

- The current loss function is good but it is too good
- Easy to overfit

Regularization in one slide

- What?
  - Regularization is a method the reduce overfitting of the data
- Why?
  - Gives more generalizability
  - Better for lower amounts of data
- How?
  - Introducing regularizing terms in the original loss function
    - Can be anything that make sense

Regularization in neural networks L2

- We want to improve generalization somehow.
- Observation, models are better when the weights are spread out (no peaky weights). [0.3 -0.2 1.9]
  - Try to use every part of the model. [0.3 -0.2 -01]
- Add a cost if we put some value to the weights

- Regularized loss = Original loss + 0.5 $C\Sigma w^2$

- we sum the square of weights of the whole model
- 0.5 is for prettiness when we take derivative
- C is a hyperparameter weighting the regularization term

Regularization in neural networks

- We want to improve generalization somehow.
- Observation, models behave better when we force the weights to be sparse.
  - Sparse means many weights are zero or close to zero
  - Force the model to focus on only important parts
  - Less prone to noise
- Add a cost if we put some value to the weights
- Regularized loss = Original loss + 0.5 $C\Sigma|w|$

[0.3 -0.2 1.9]

[0.3 0.0 0.001]

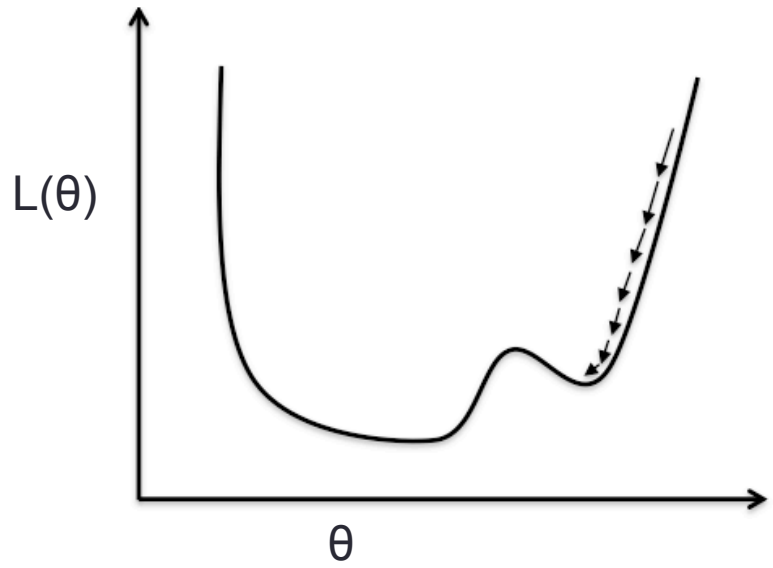
- we sum the absolute weights of the whole model
- 0.5 is for prettiness when we take derivative
- C is a hyperparameter weighting the regularization term

L1 L2 regularization notes

- Can use both at the same time
  - People claim L2 is superior
- I found them useless in practice for deep neural networks
  - Maybe there are some tasks out there that benefit from this? I don't know
- Other regularization methods exist (we will go over these later)

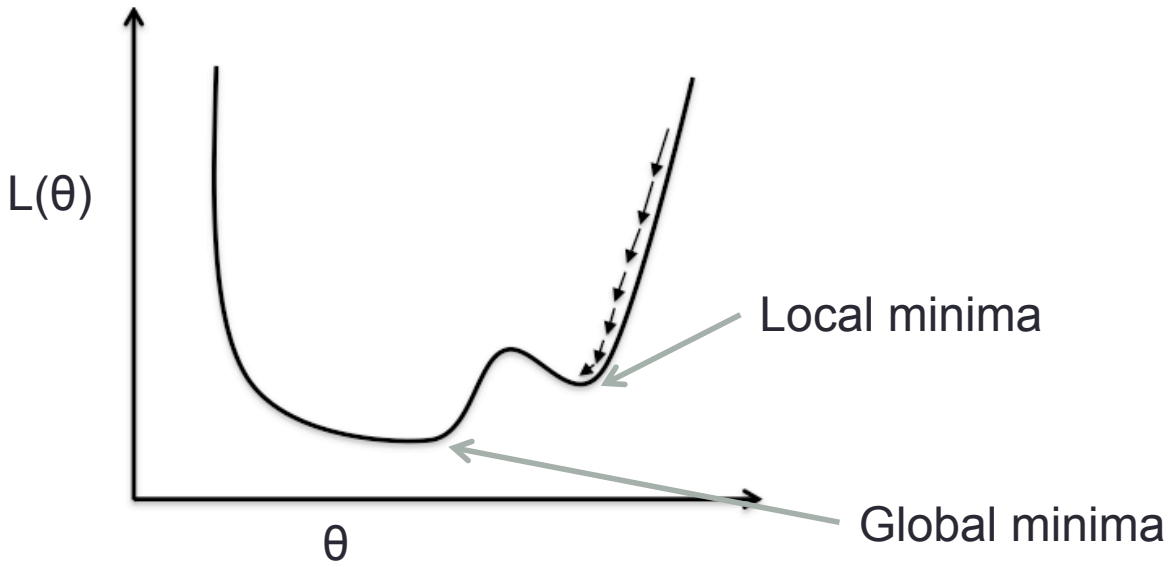
Minimization using gradient descent

- 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



Deep vs Shallow

- If deep, you can avoid most local minima



Kenji Kawaguchi,Deep Learning without Poor Local Minima, 2016 https://arxiv.org/abs/1605.07110

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

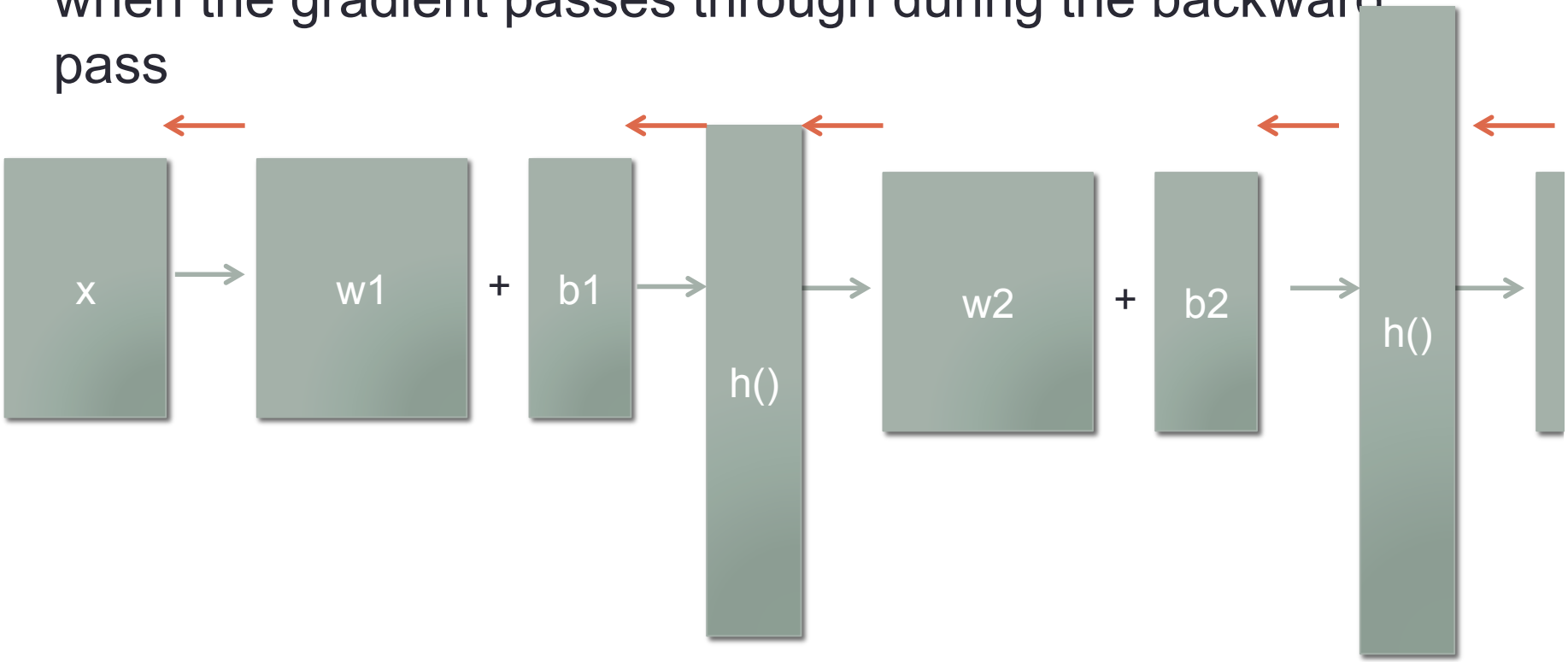
Examples to read

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

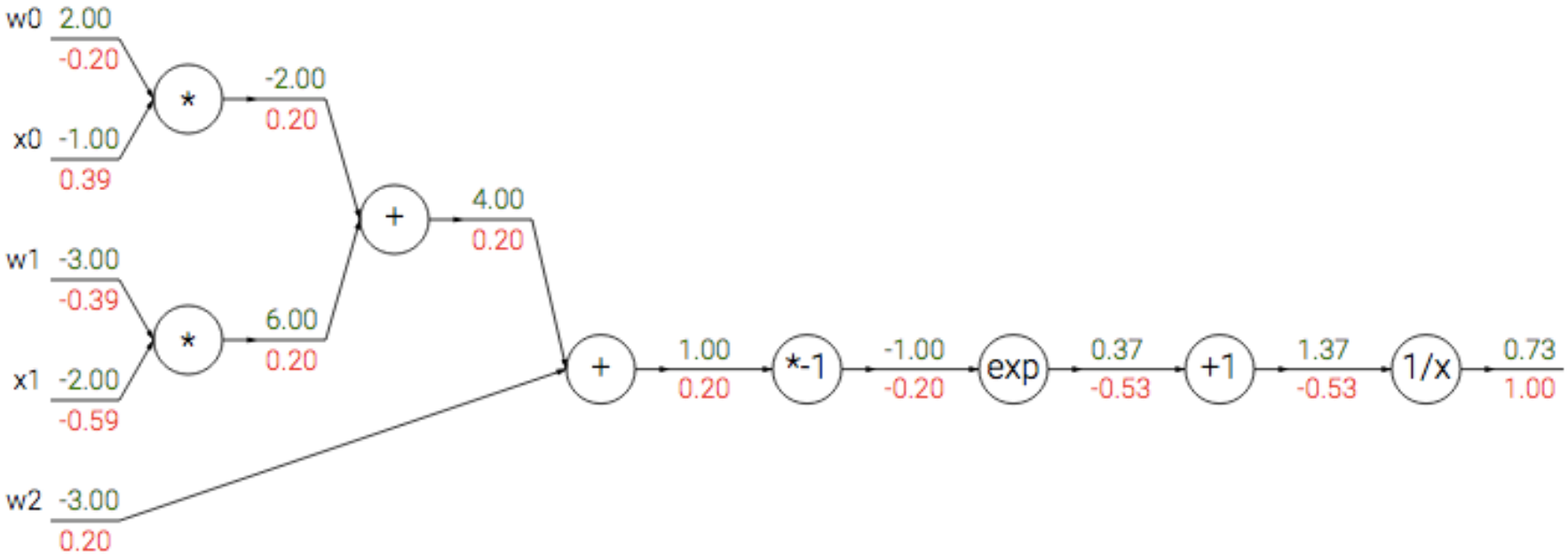


This lets us to build any neural networks without having to redo all the derivation as long as we define a forward and backward computation for the block.

Gradient flow in code

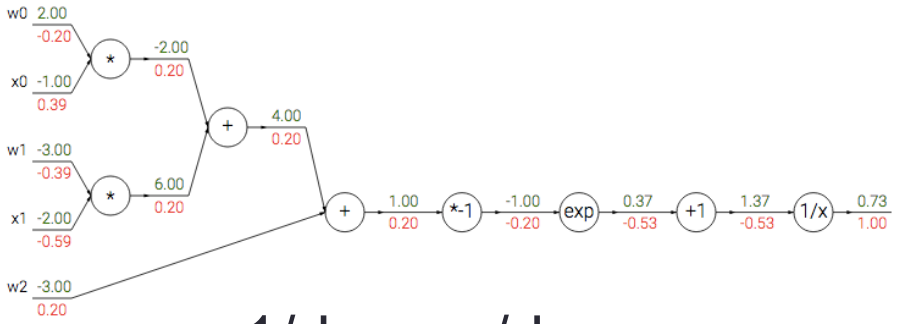
- Let's find the gradient of $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

Computation graph



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

$$f(w,x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

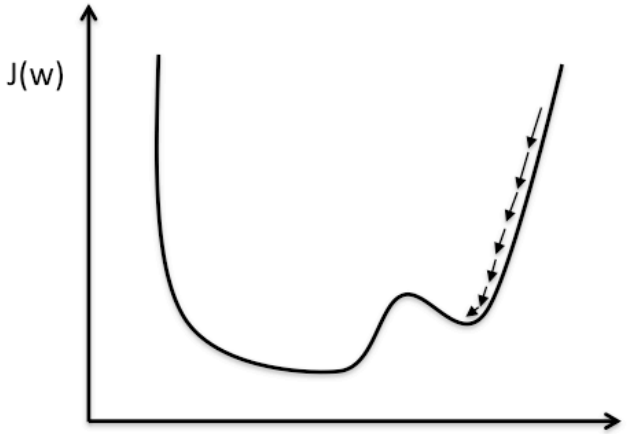


- w = [0, -3, -3]
- x = [-1, -2]
- t0 = w[0]*x[0]
- t1 = w[1]*x[1]
- t01 = t0+t1
- t012 = t01+w[2]
- nt = -t012
- e = exp(nt)
- denom = e+1
- f = 1/denom

- ddenom = -1/denom/denom
- de = 1*ddenom
- dnt = exp(nt)*de
- dt012 = -dnt
- dw2 = 1*dt012
- dt01 = 1*dt012
- dt0 = 1*dt01
- dt1 = 1*dt01
- dw1 = x[1]dt1
- dx1 = w[1]dt1
- dw0 = x[0]dt0; dx0 = w[0]dt0

Perform backward pass in reverse order. No need to explicitly find overall derivative

Initialization



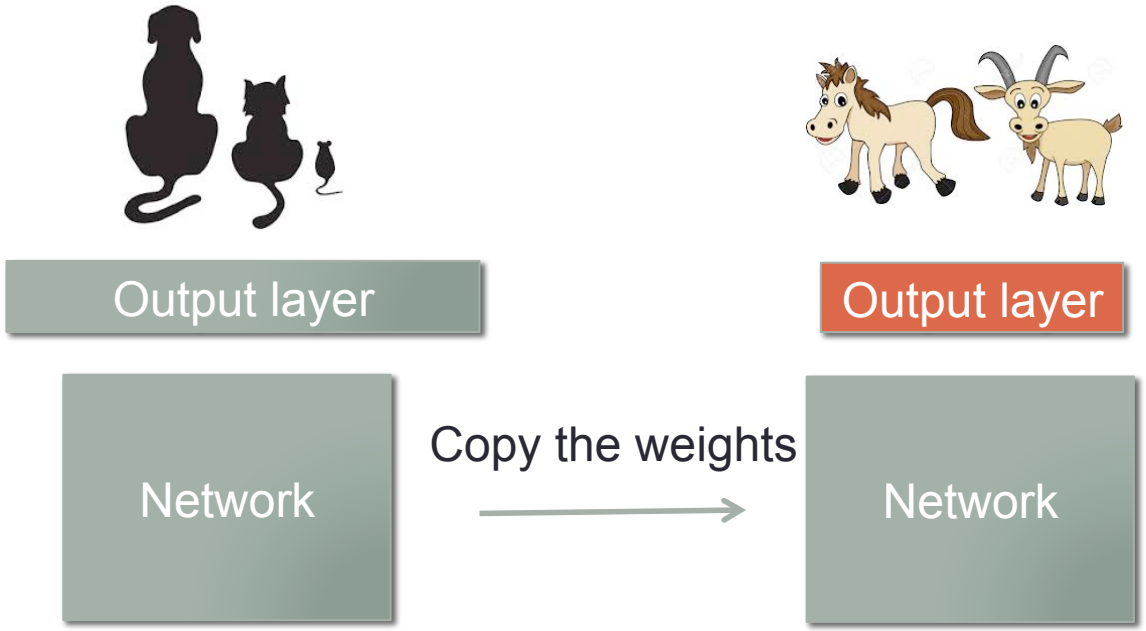
- The starting point of your descent
- Important due to local minimas
- Not as important with large networks AND big data

- Now usually initialized randomly
- Or use transfer learning

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

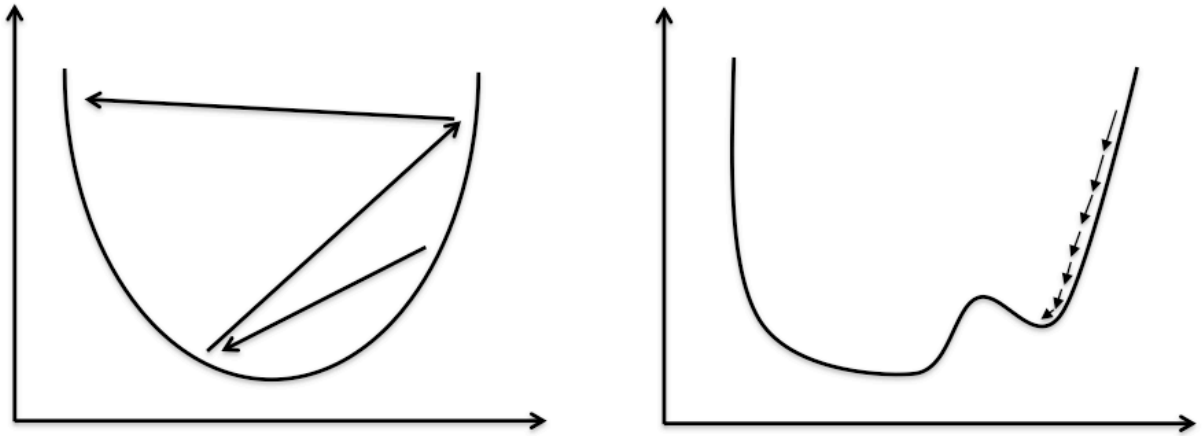
Transfer learning

- Initialize weights by using parameters from another network
- Neural networks learn features!
- Features should be useful in related task



Learning rate

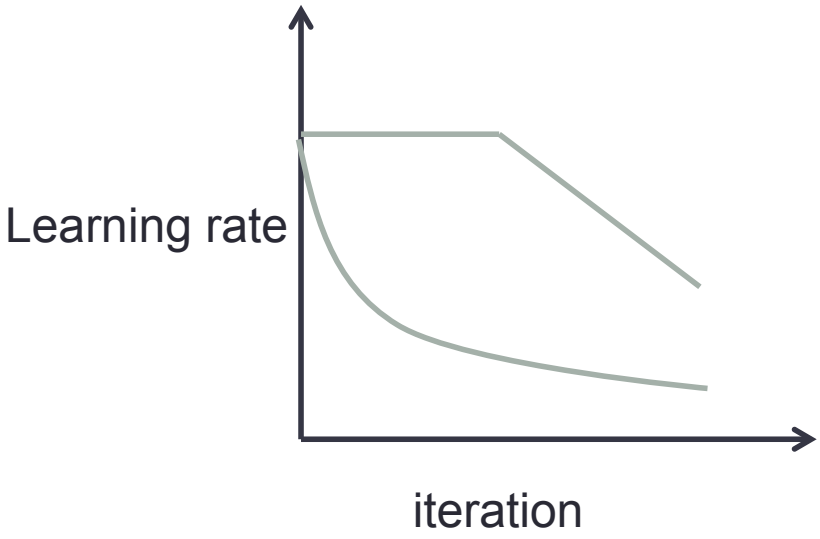
- How fast to go along the gradient direction is controlled by the learning rate
- Too large models diverge
- Too small the model get stuck in local minimas and takes too long to train



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)

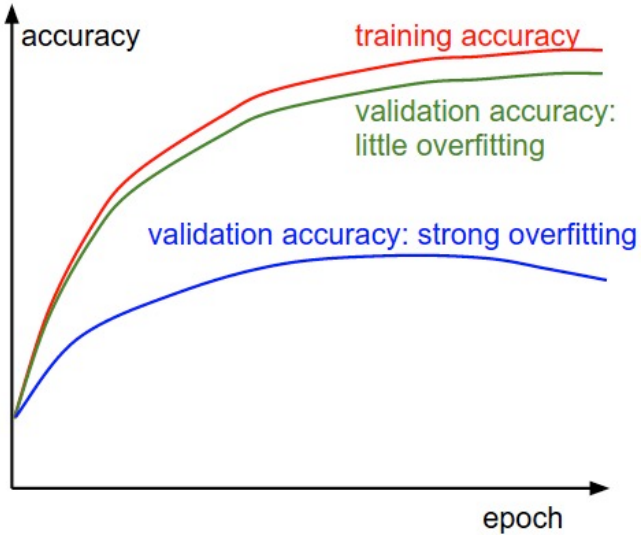


Learning rate strategies (annealing)

- Step decay
  - reduce learning rate by x after y epochs
- New bob method
  - half learning rate every time the validation error goes up. Only plausible in larger tasks
- Exponential decay
  - multiplies the learning rate by exp(-rate*epoch number)

Overfitting

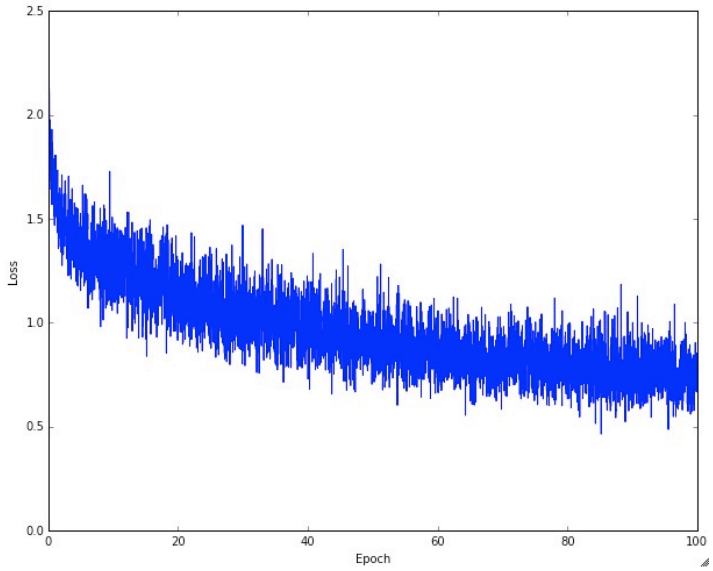
- You can keep doing back propagation forever!
- The training loss will always go down
- But it overfits
- Need to monitor performance on a held out set
- Stop or decrease learning rate when overfit happens



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/

Corpus segmentation

- Train – learn our parameters
  - Something we learn from data
  - Weights of neural network
- Test
  - Simulate the unknown real usage of the model

- Tune/Validation/Dev set – learn our hyper-parameters
  - Something we decide
  - Number of neurons, learning rate, how to decay, when to stop training.
  - We need a separate set to try out different parameters.

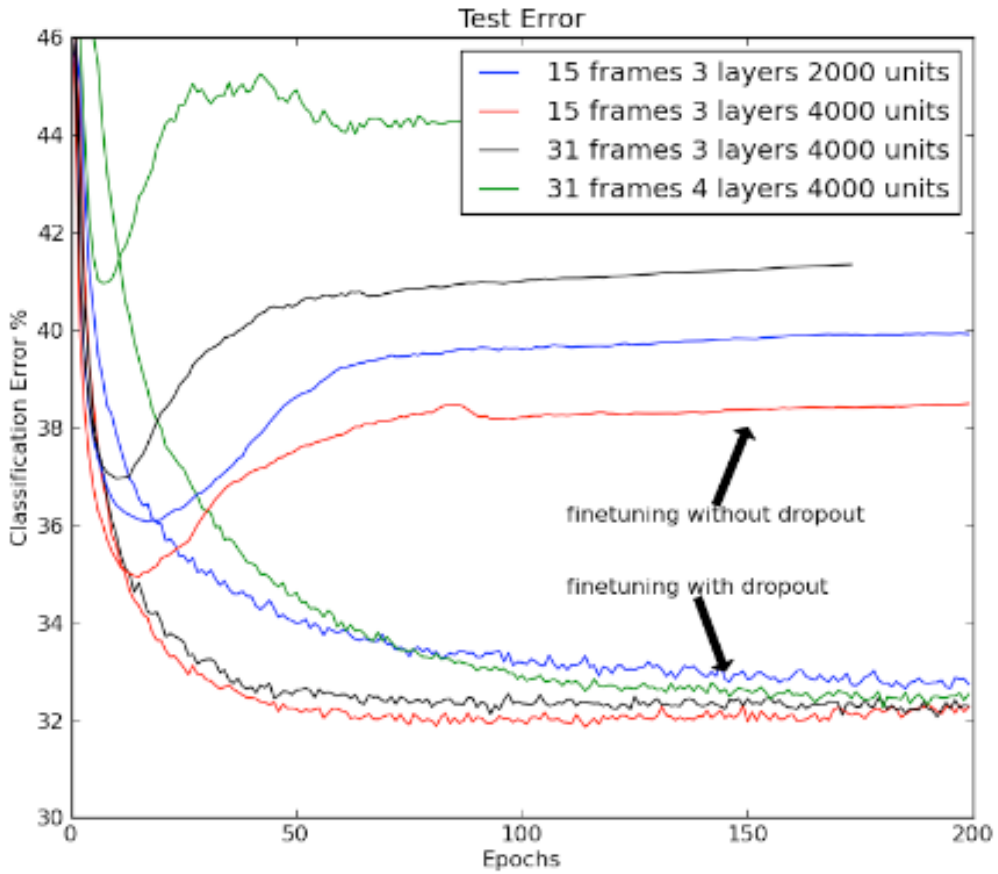
Reducing overfitting - dropout

- A recent (2012) 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 (assemble of experts)
- Now a standard technique

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

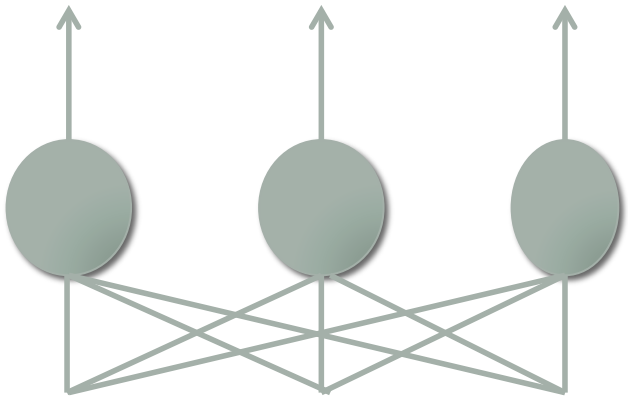
Dropout on TIMIT

- A phoneme recognition task



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

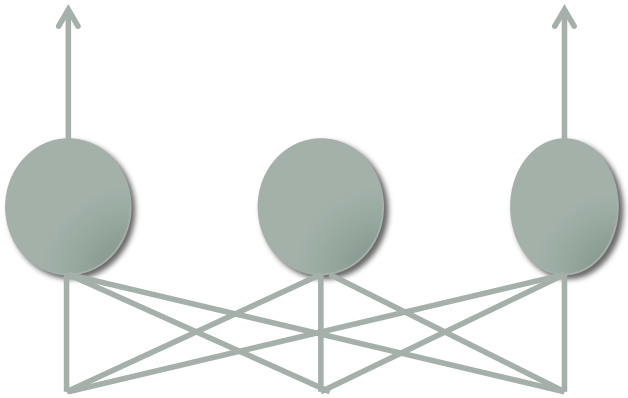
Dropout visualized



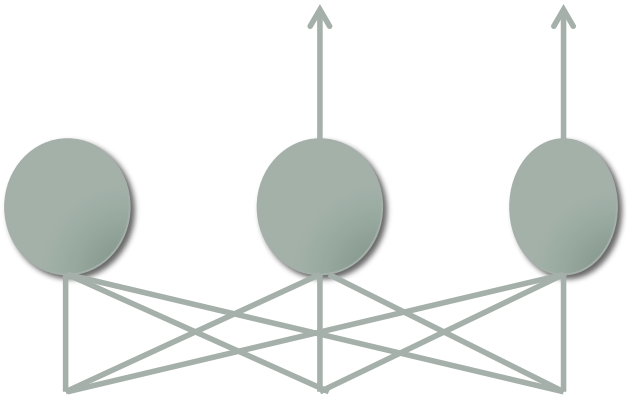
Model

Dropout visualized

Randomly removed outputs
for each training sample

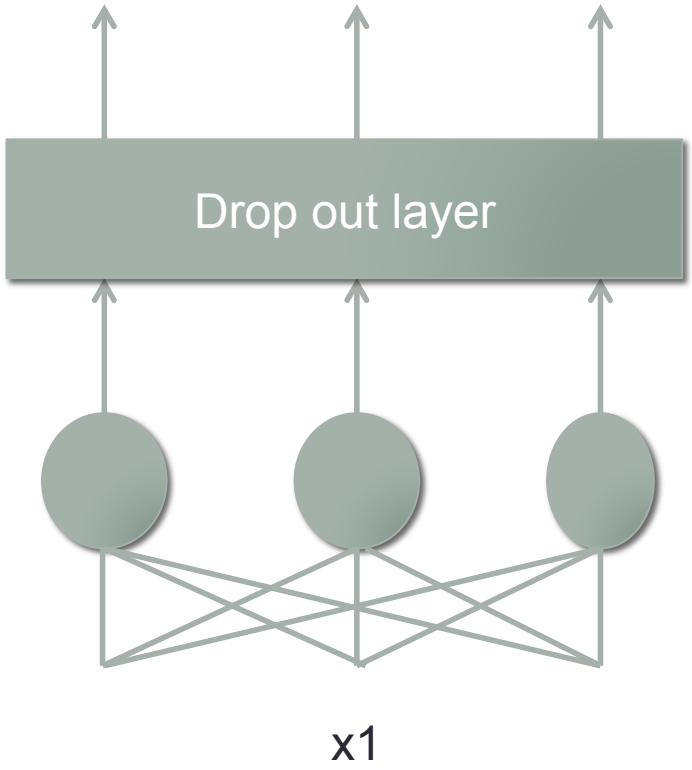


x1



x2

Dropout implementation



Randomly drop inputs and pass through other inputs

Neural networks

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

Convolutional Neural Networks (CNNs)

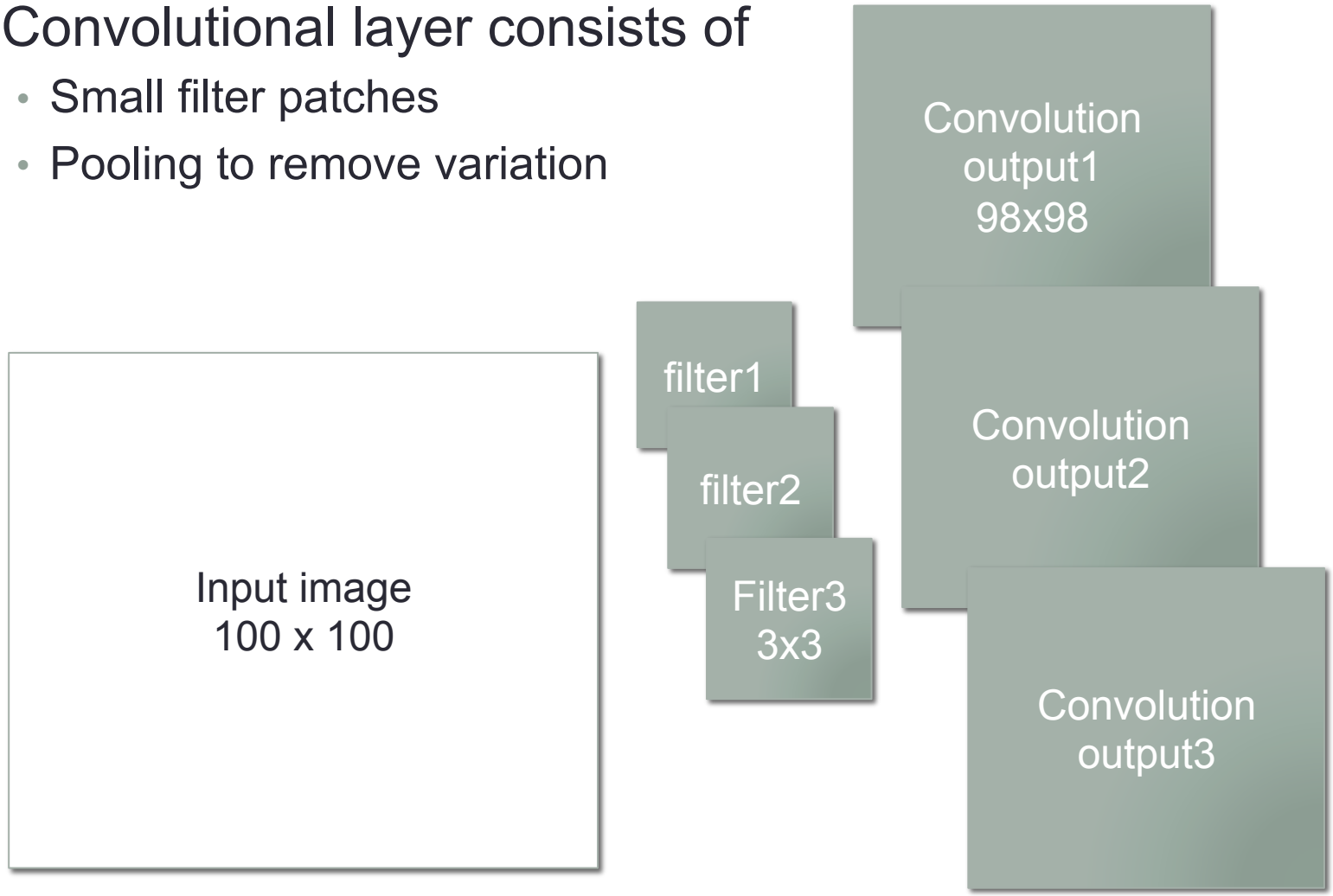
- Consider an image of a cat. DNNs need different neurons to learn every possible location a cat can be

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

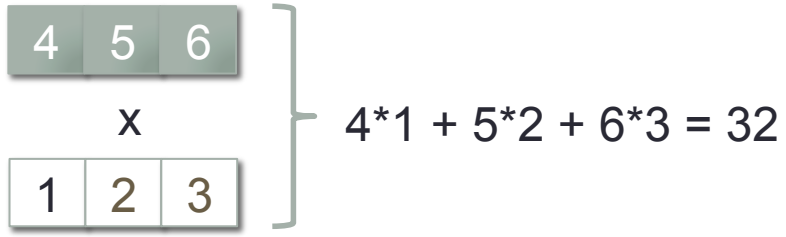
- 2 parts: convolutional layer and pooling layer

Convolutional filters

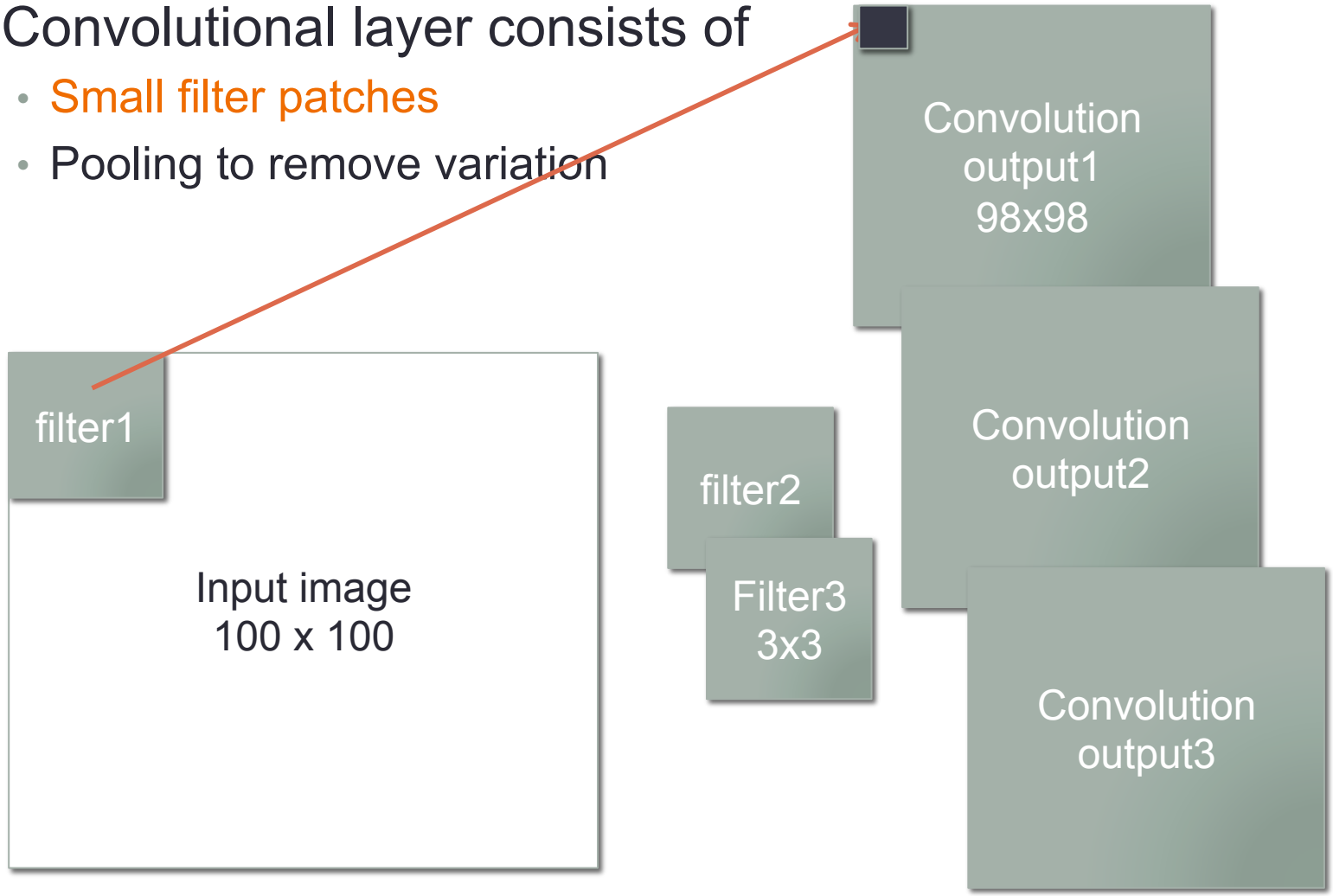
- Convolutional layer consists of
  - Small filter patches
  - Pooling to remove variation



Convolutional filters

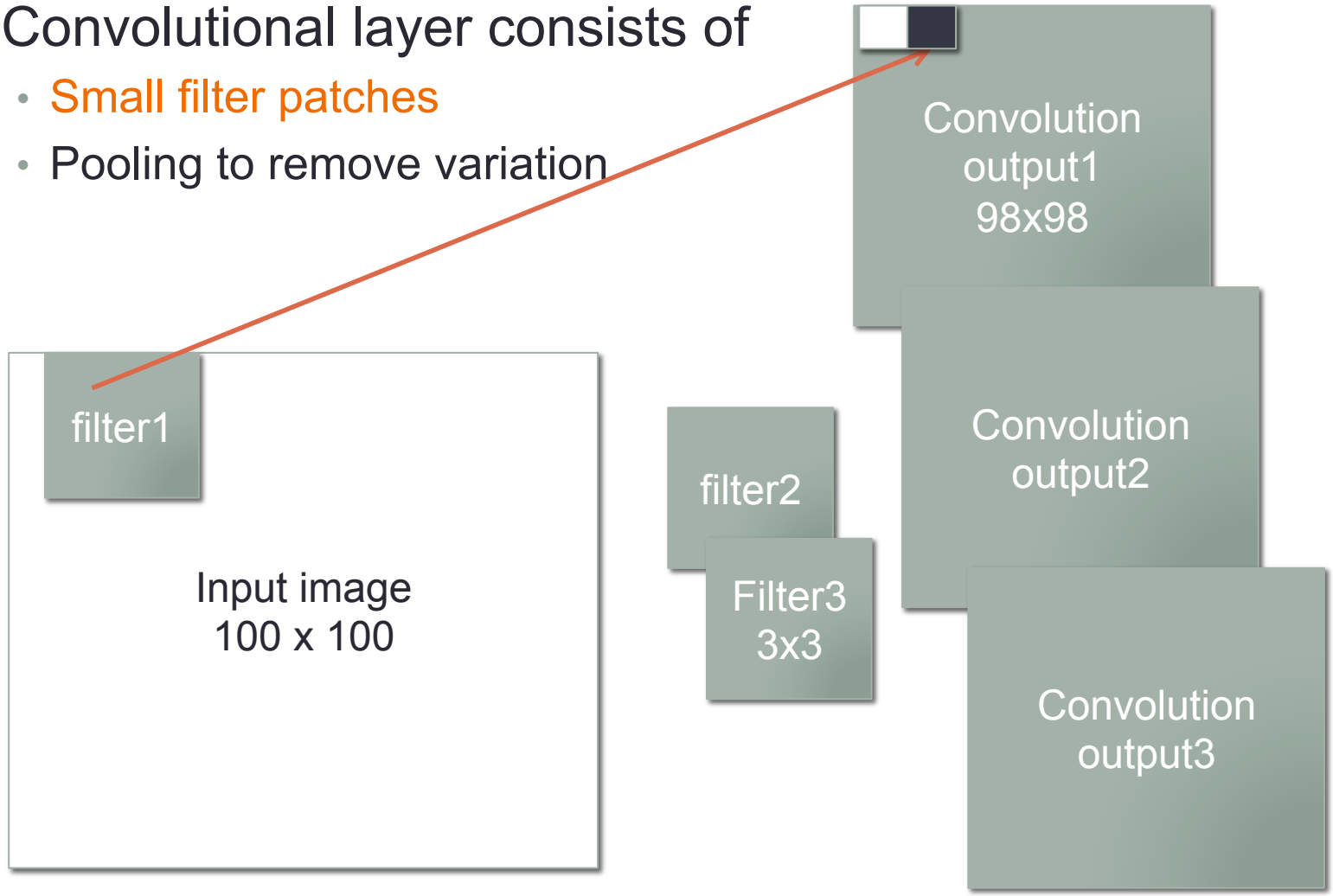


- Convolutional layer consists of
  - Small filter patches
  - Pooling to remove variation



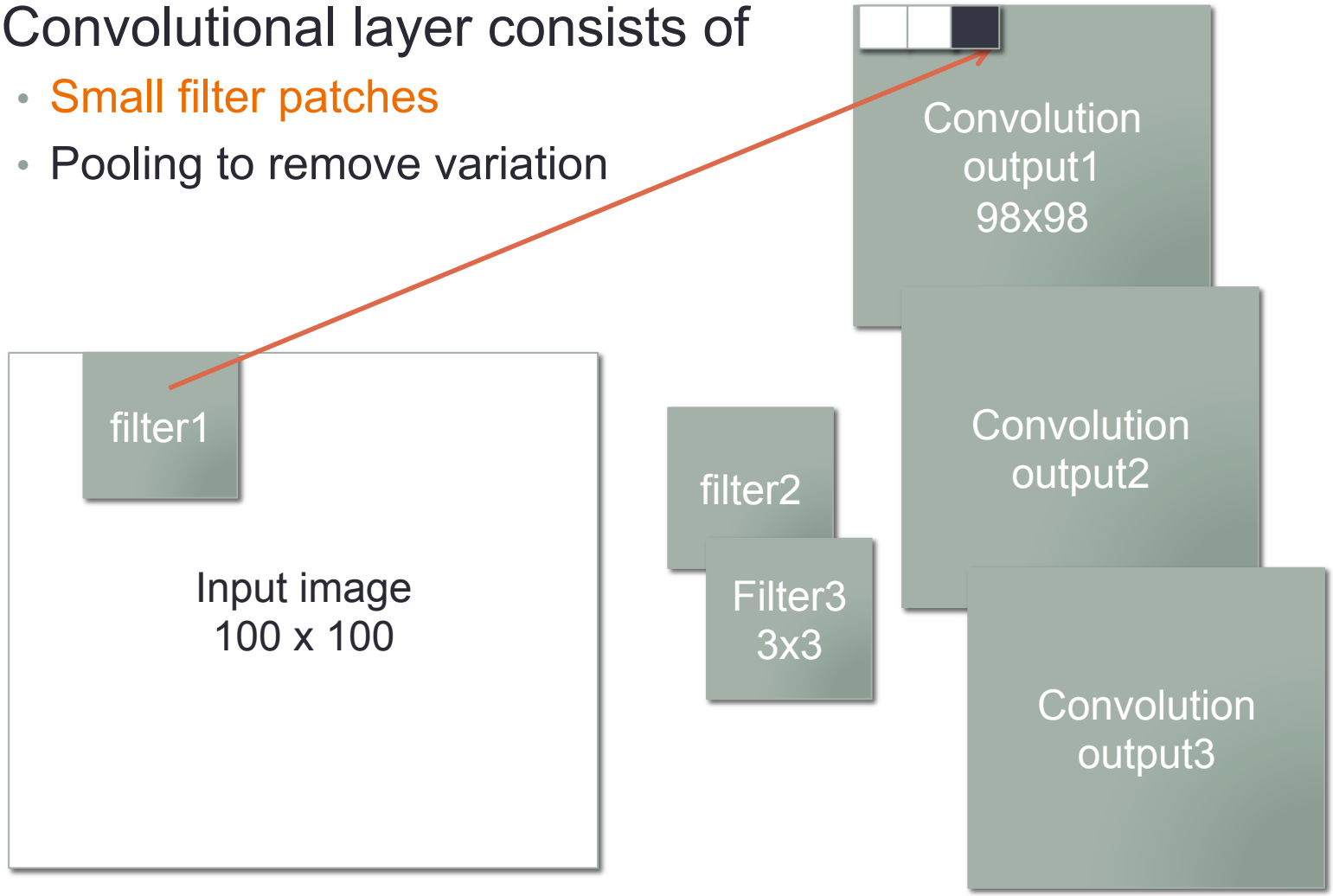
Convolutional filters

- Convolutional layer consists of
  - Small filter patches
  - Pooling to remove variation



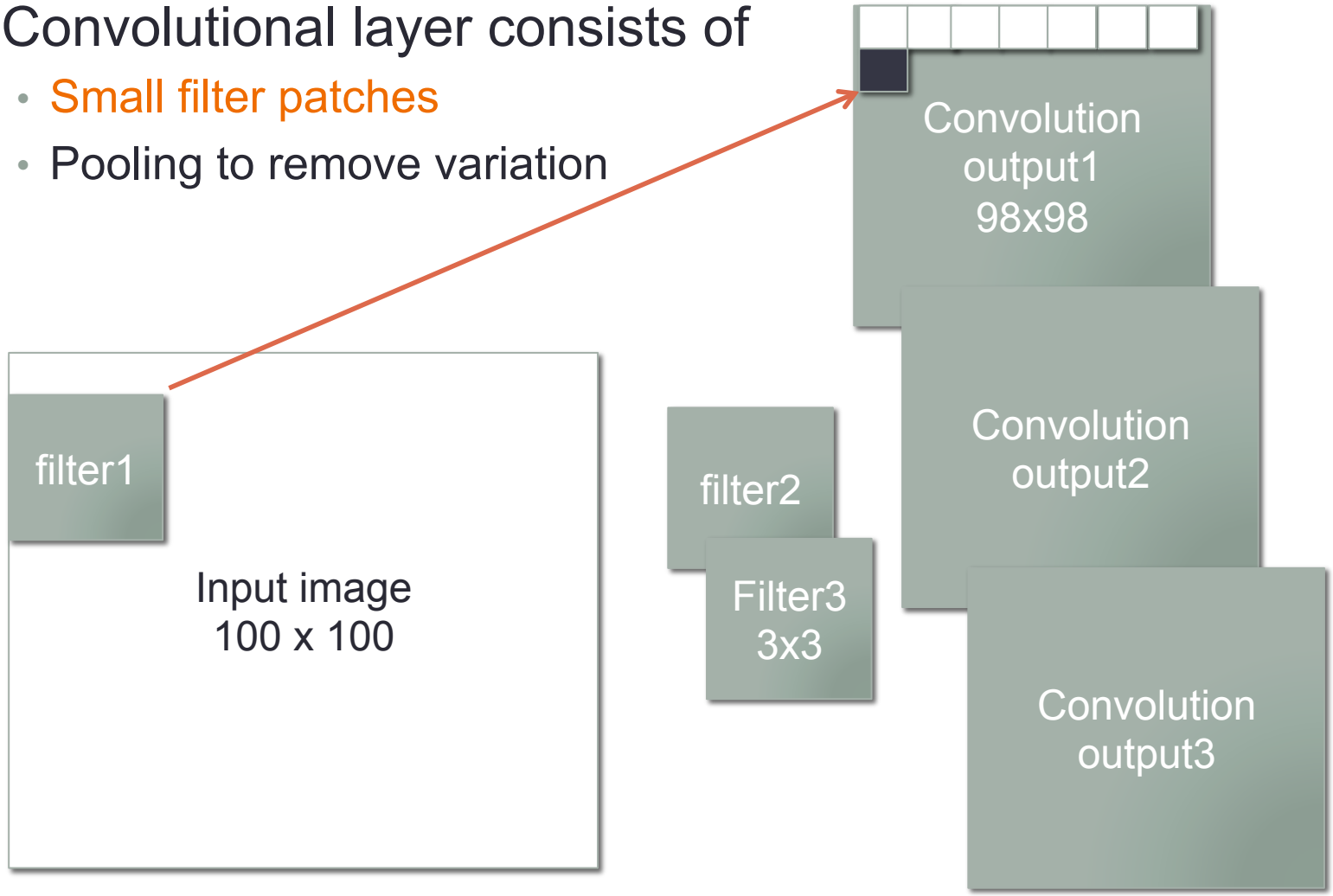
Convolutional filters

- Convolutional layer consists of
  - Small filter patches
  - Pooling to remove variation



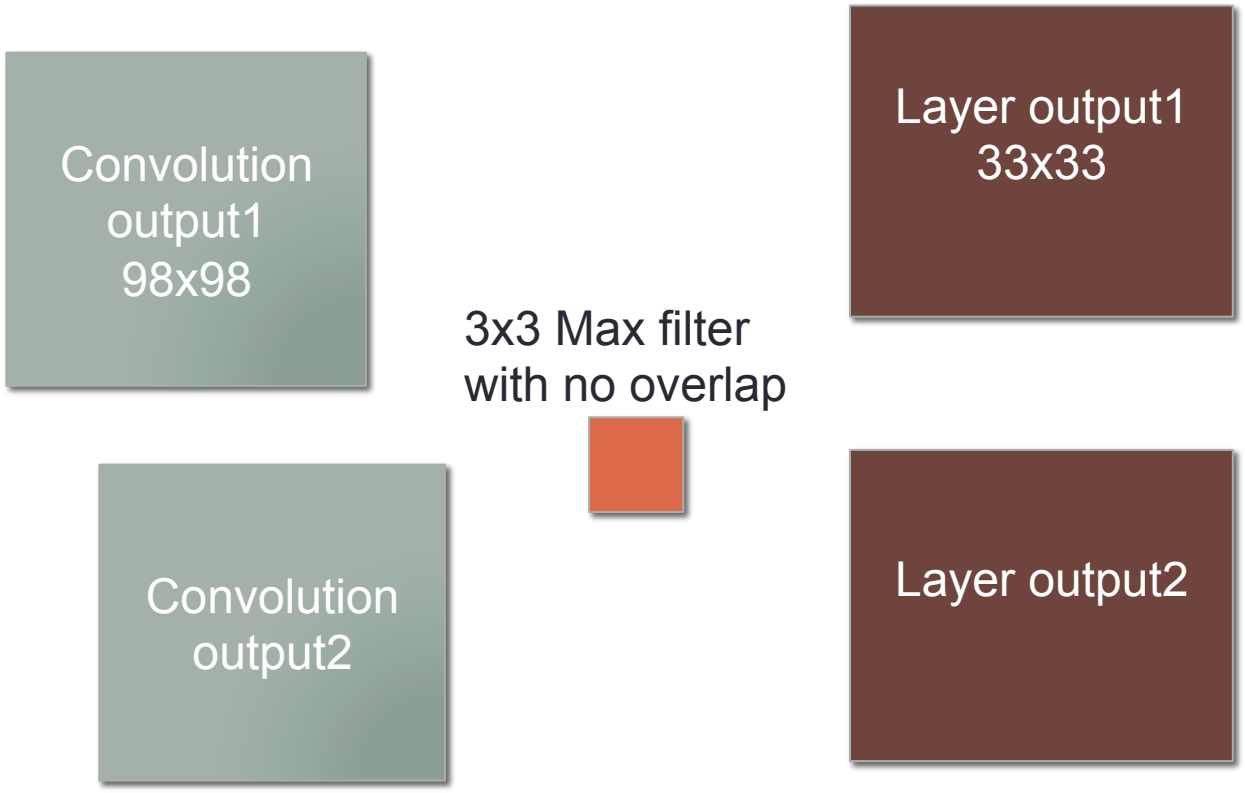
Convolutional filters

- Convolutional layer consists of
  - Small filter patches
  - Pooling to remove variation



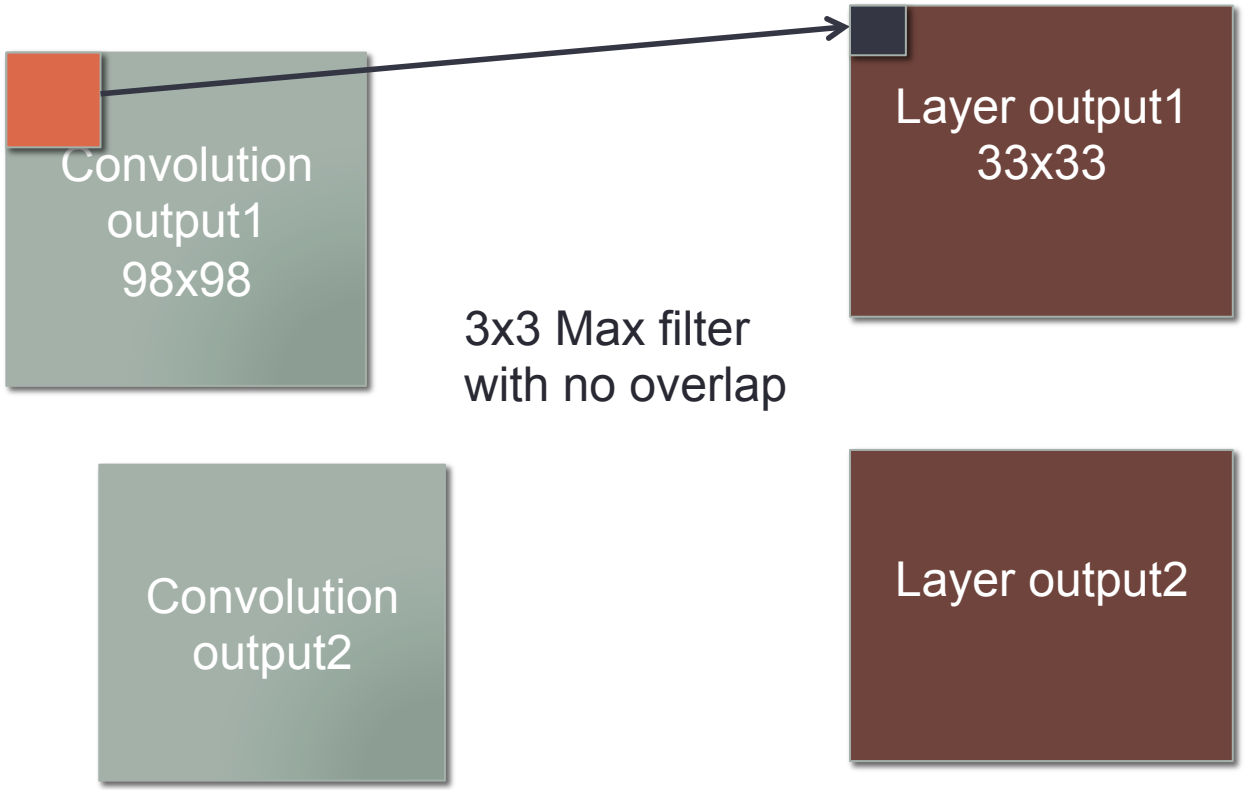
Pooling/subsampling

- Convolutional layer consists of
  - Small filter patches
  - Pooling to remove variation



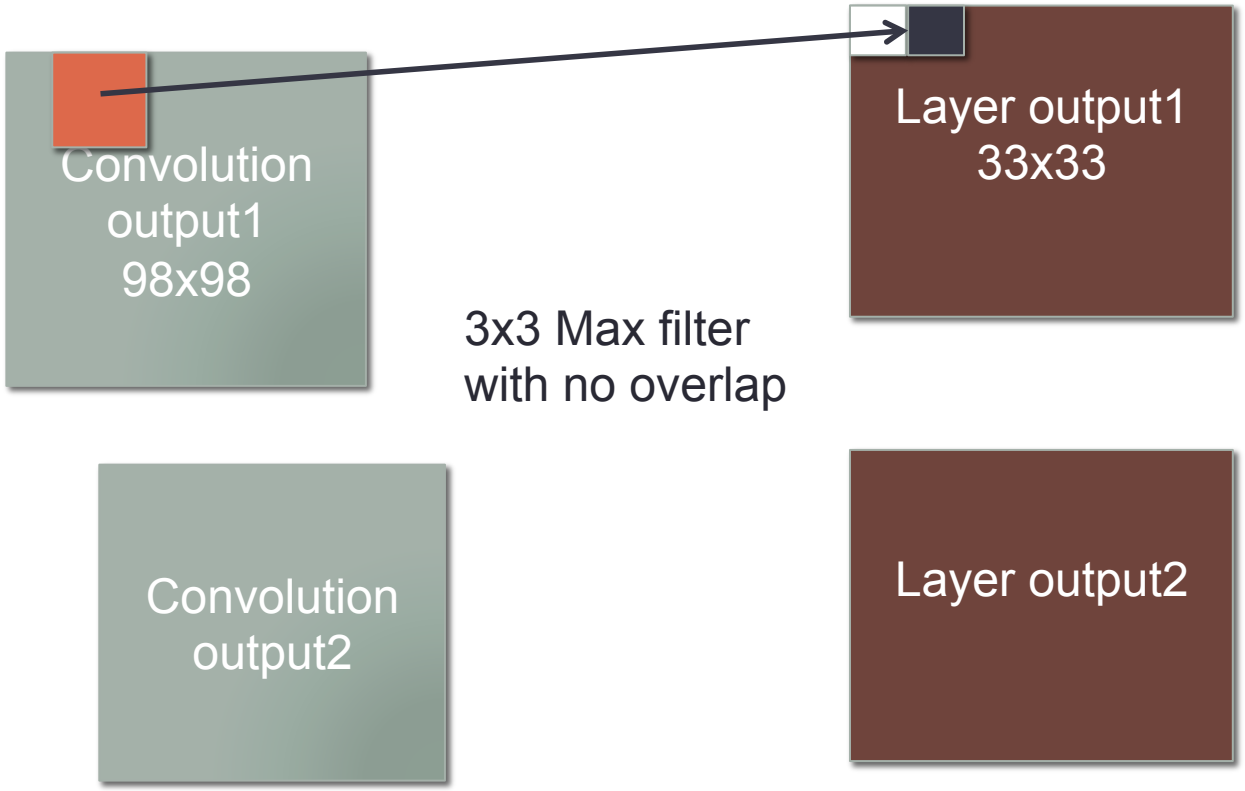
Pooling/subsampling

- Convolutional layer consists of
  - Small filter patches
  - Pooling to remove variation



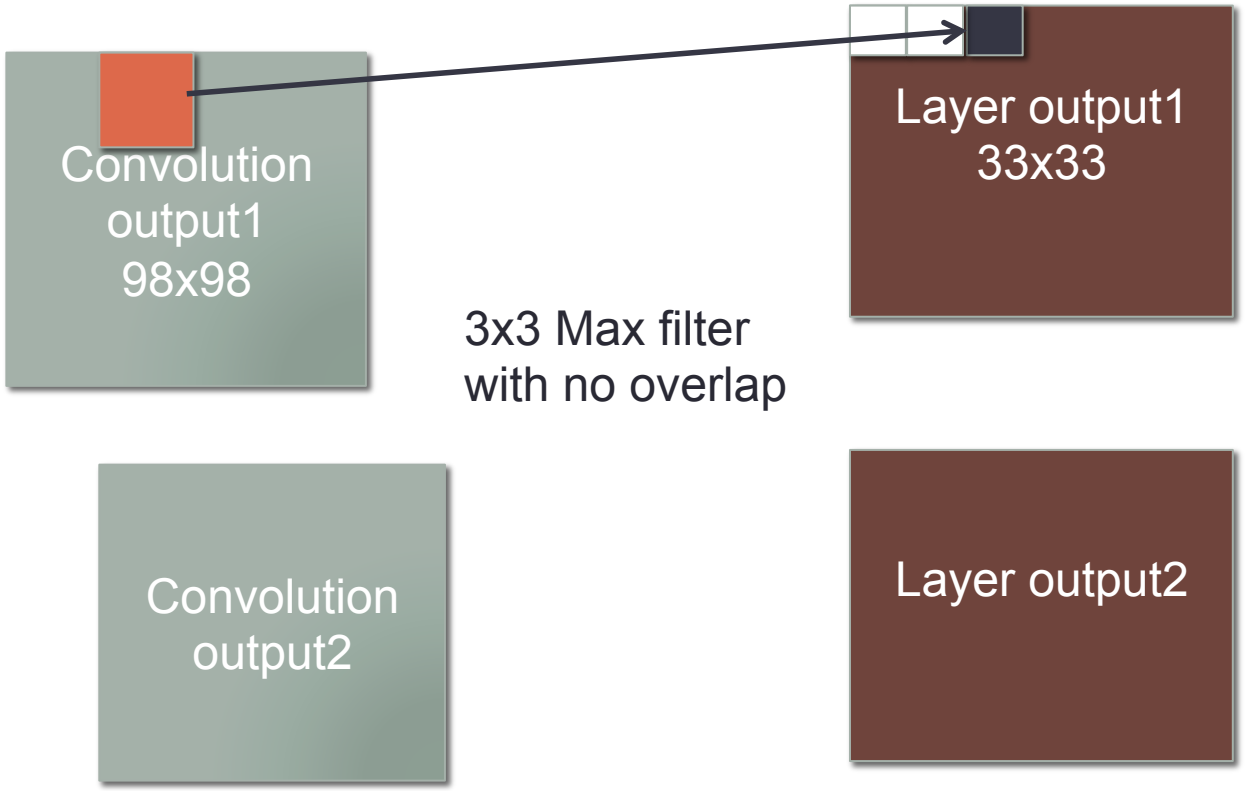
Pooling/subsampling

- Convolutional layer consists of
  - Small filter patches
  - Pooling to remove variation



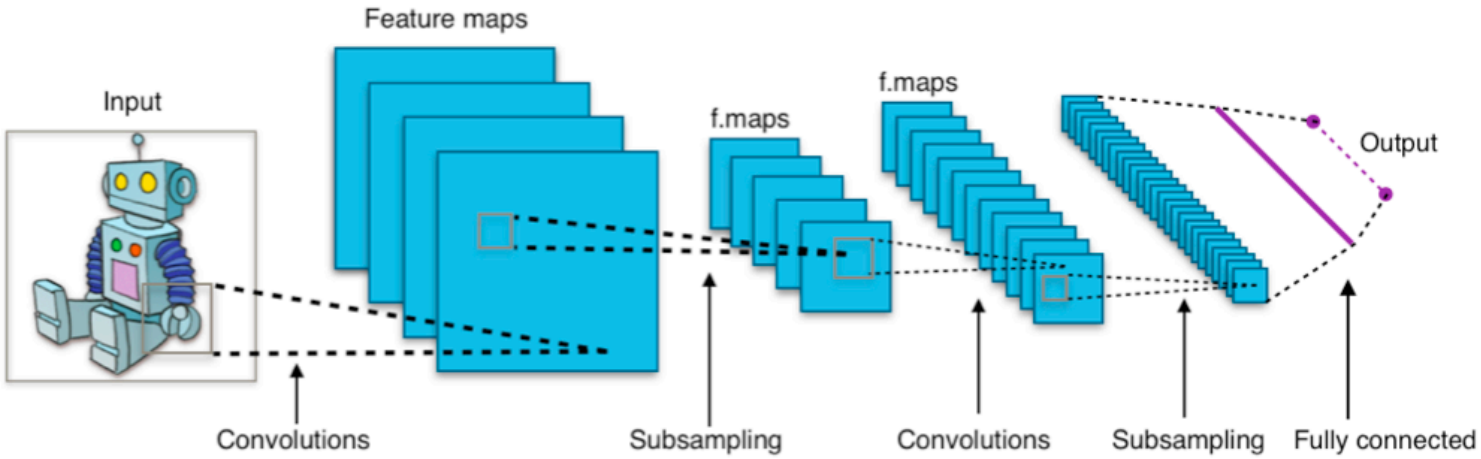
Pooling/subsampling

- Convolutional layer consists of
  - Small filter patches
  - Pooling to remove variation



CNN overview

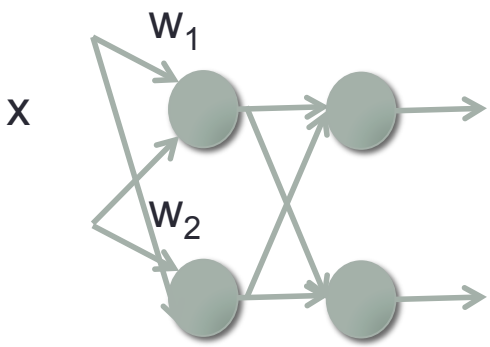
- Filter size, number of filters, filter shifts, and pooling rate are all parameters
- Usually followed by a fully connected network at the end
  - CNN is good at learning low level features
  - DNN combines the features into high level features and classify



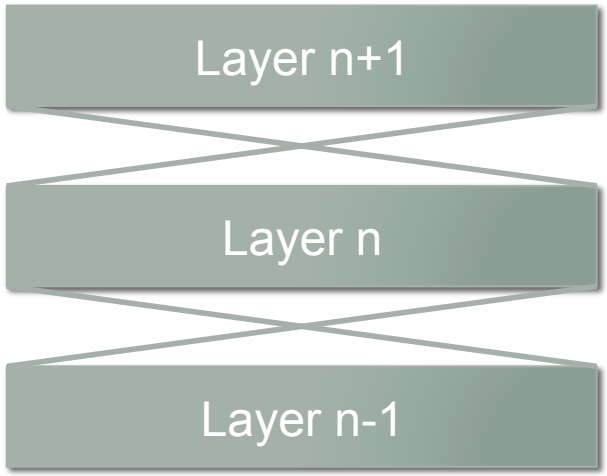
https://en.wikipedia.org/wiki/Convolutional_neural_network#/media/File:Typical_cnn.png

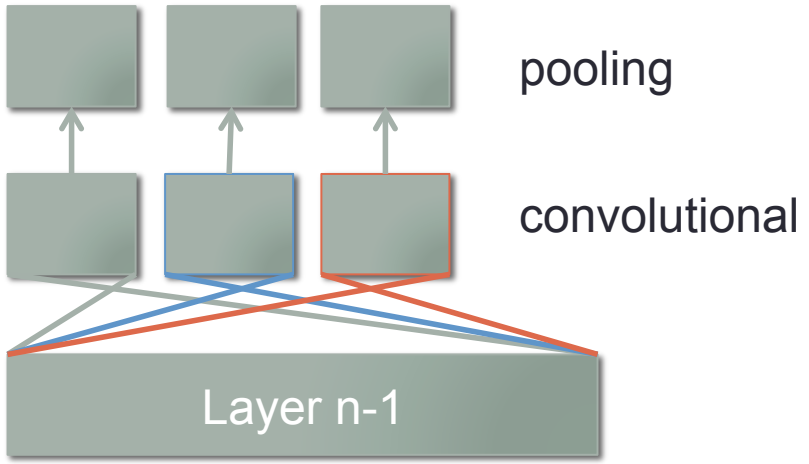
Parameter sharing in convolution neural networks

- $W^T x$



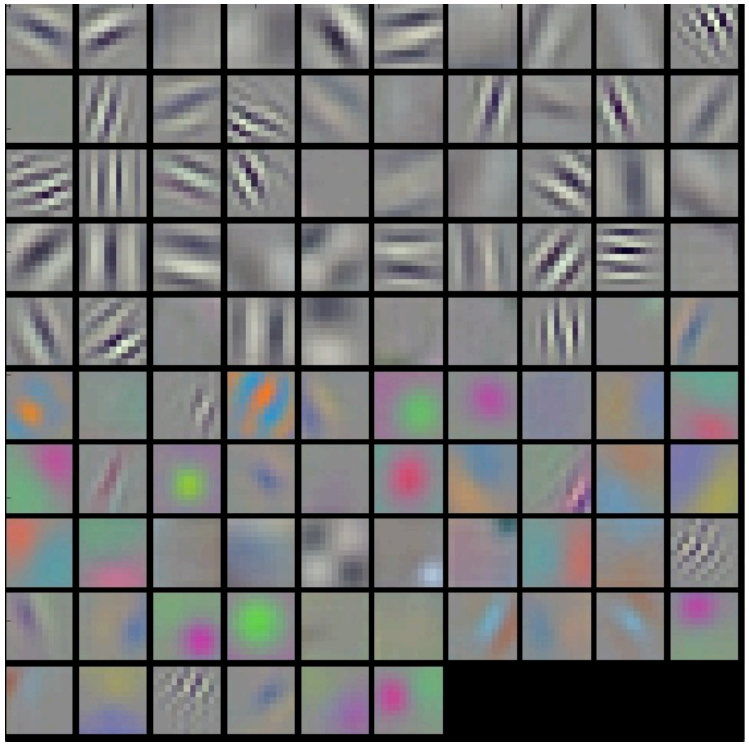
- Cats at different location might need two neurons for different locations in fully connect NNs.
- CNN shares the parameters in 1 filter
- The network is no longer fully connected





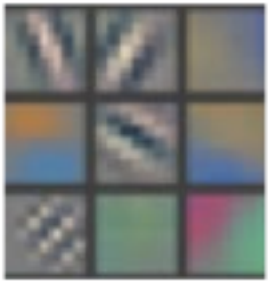
Visualizing convolutional layers

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

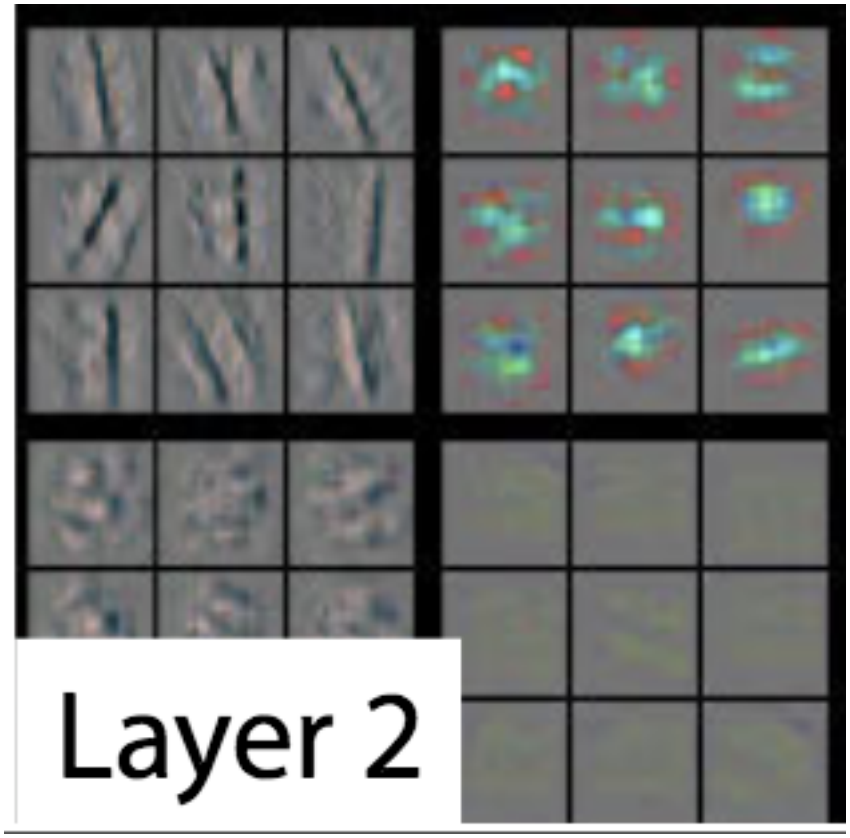


http://cs231n.github.io/understanding-cnn/

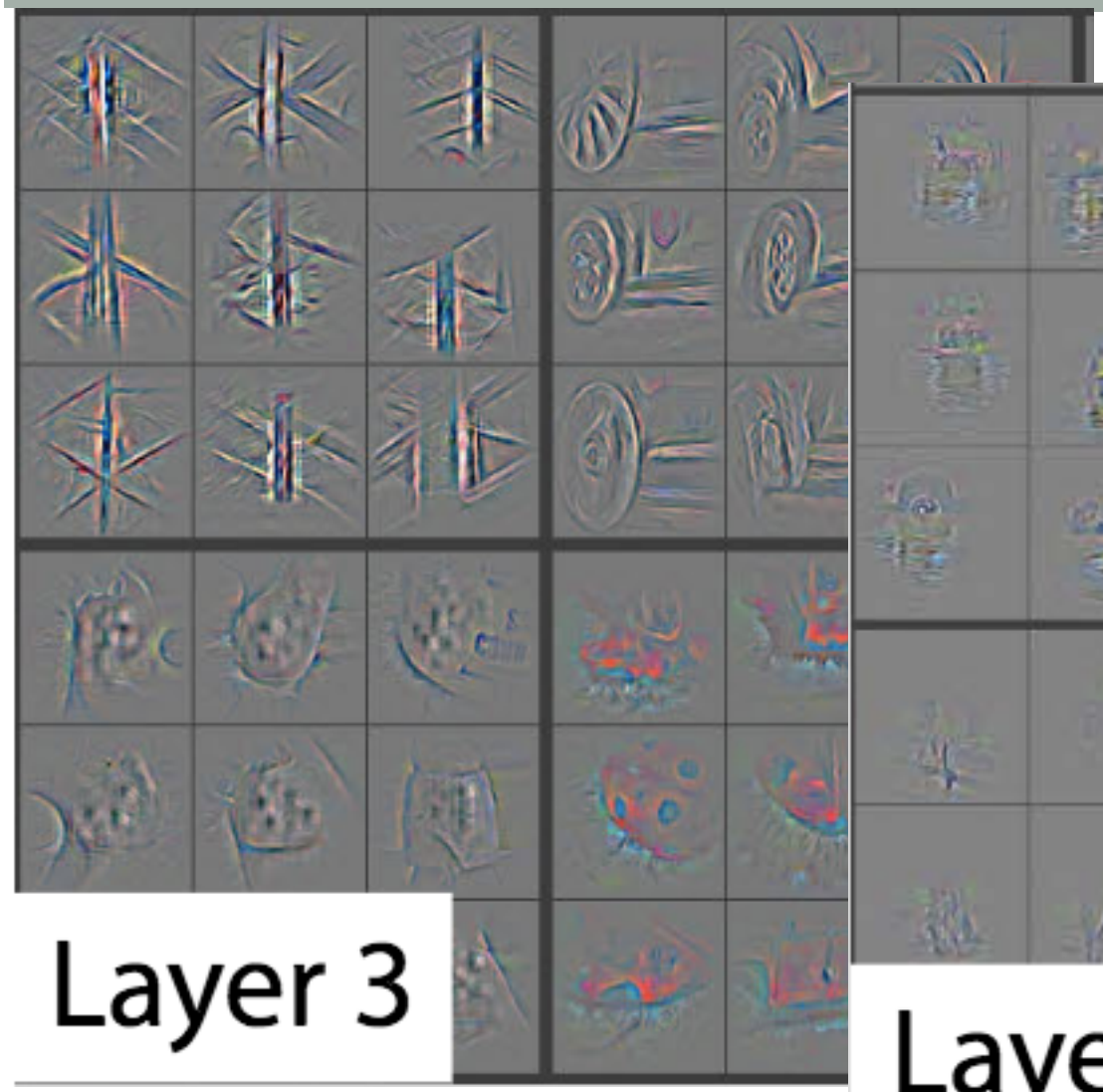
Higher layer captures higher-level concepts

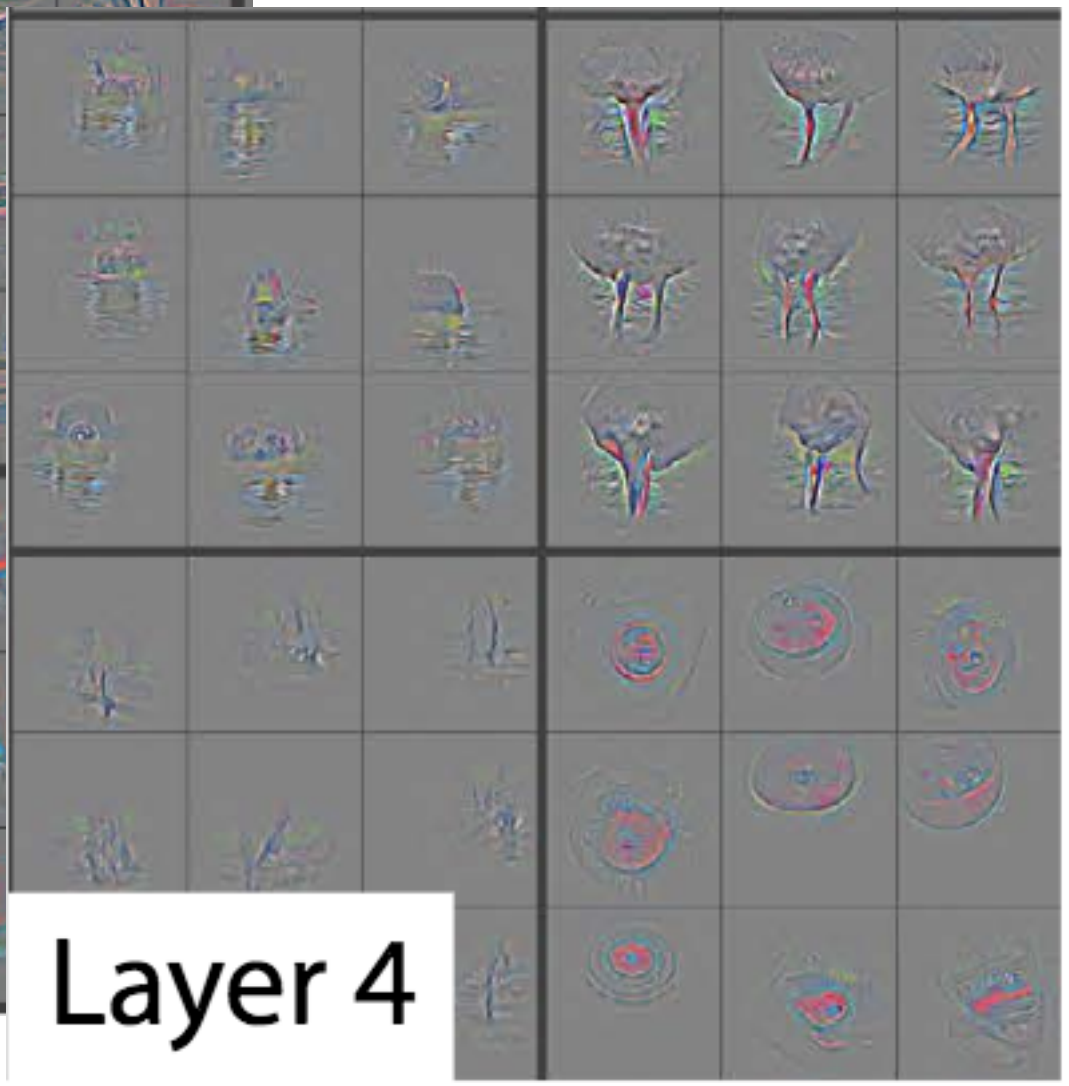


Layer 1



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


Layer 3


Layer 4

CNN

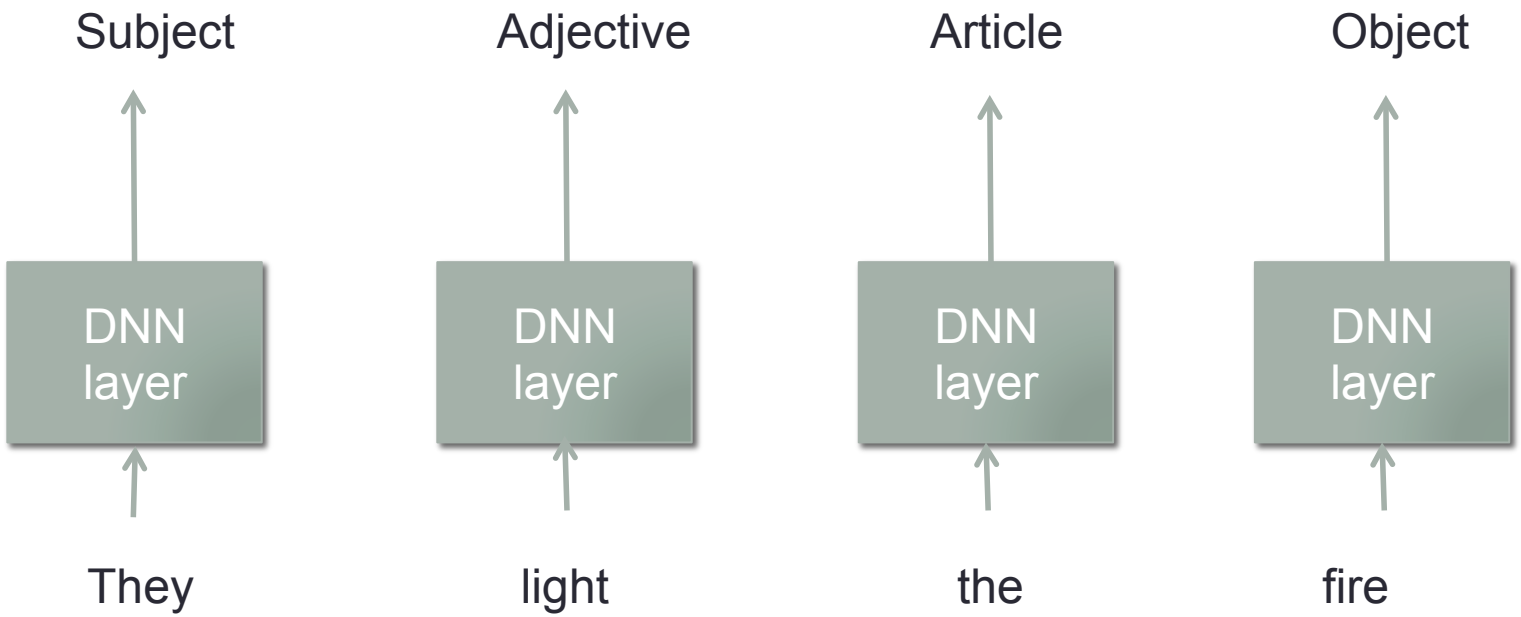
- Convolutional layer
- Subsampling

- Sharing of parameters in space

- Sharing of parameters in time?

Recurrent neural network (RNN)

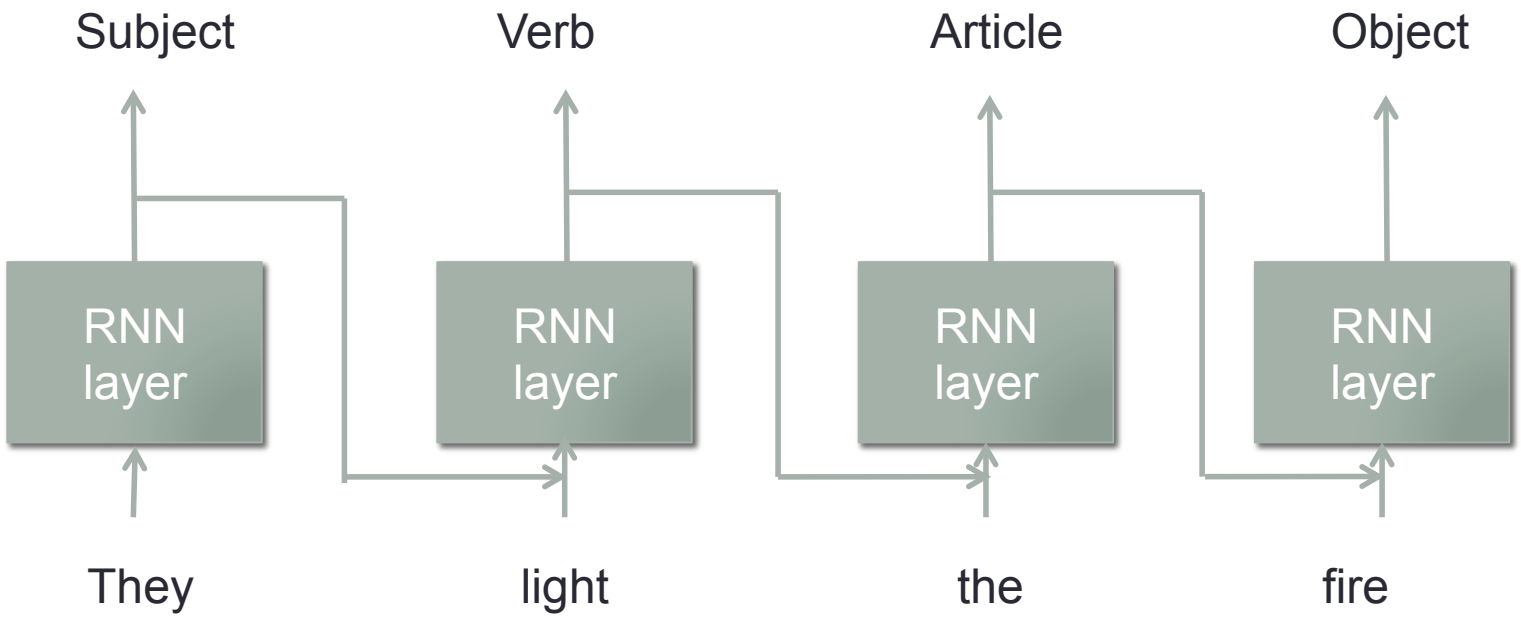
- DNN framework



Problem1: need a way to remember the past

Recurrent neural network (RNN)

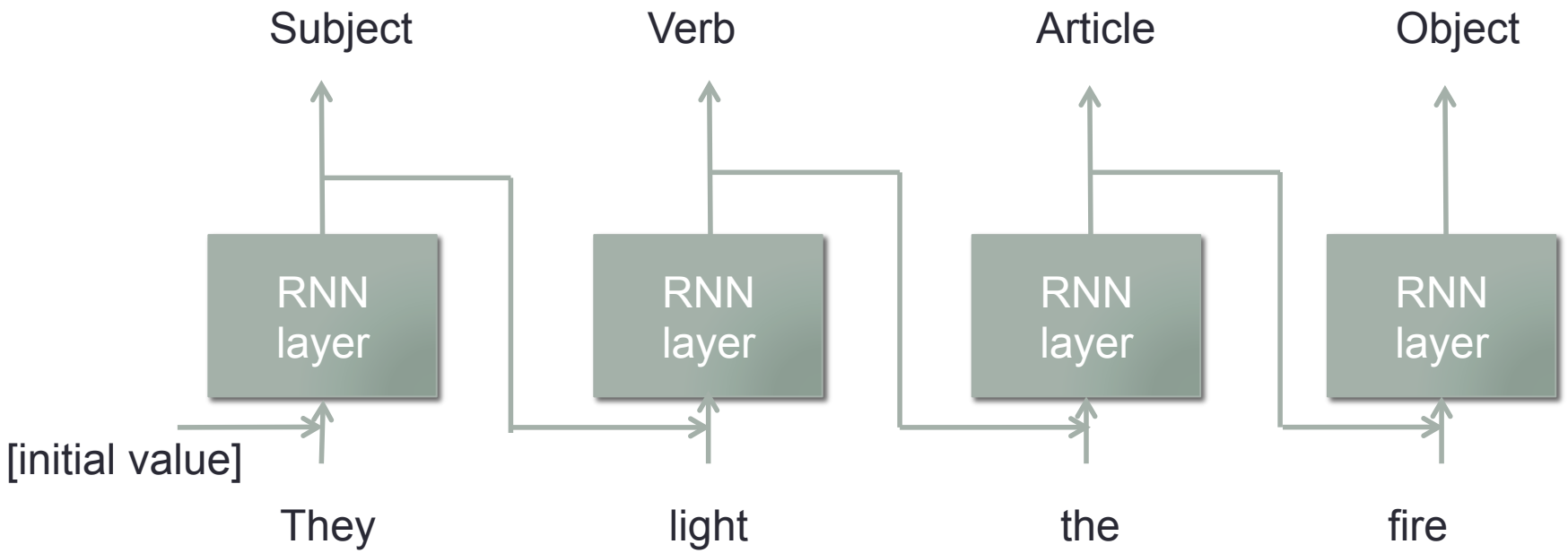
- RNN framework



Output of the layer encodes something meaningful about the past

Recurrent neural network (RNN)

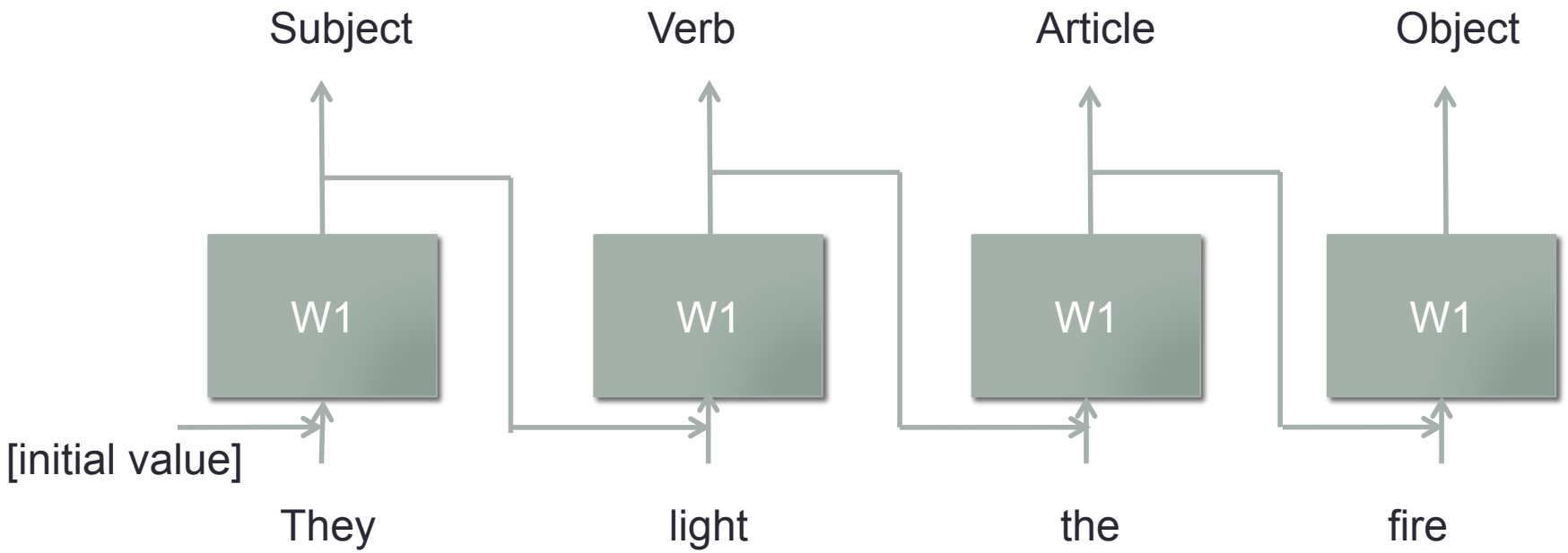
- RNN framework



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

Training a recurrent neural network

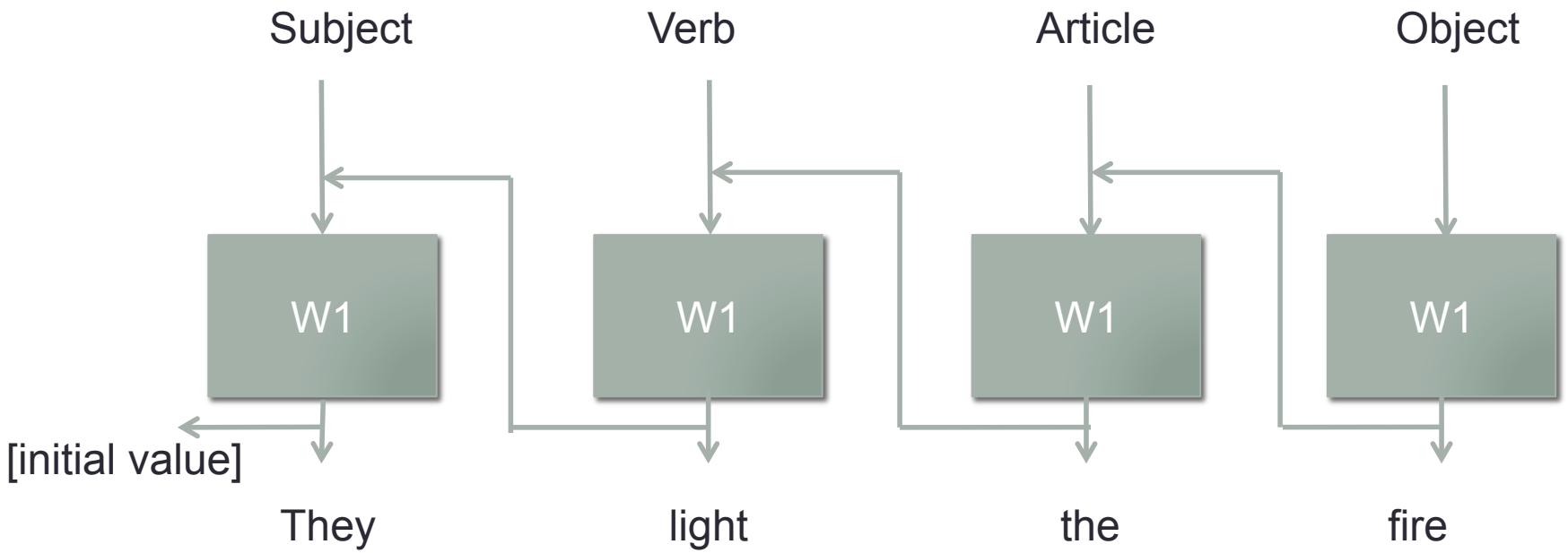
- Computation graph



Parameter sharing

Training a recurrent neural network

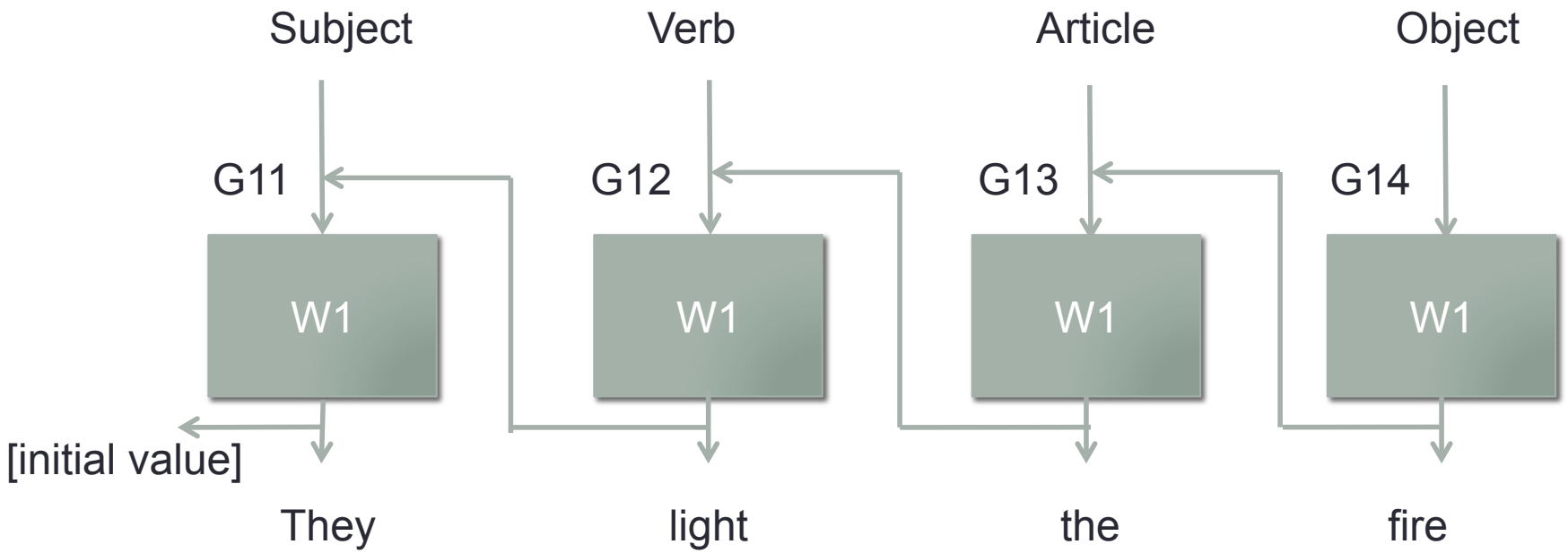
- Backward Computation graph



Backpropagation through time (BPTT)

BPTT

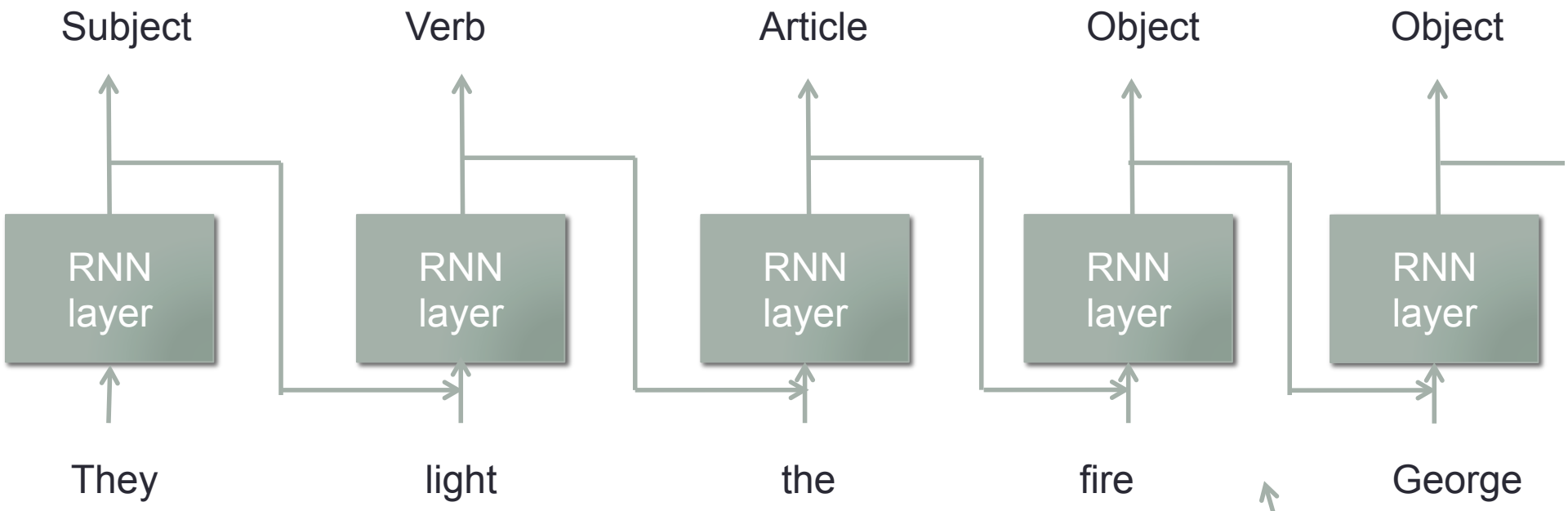
- Backward Computation graph



W1 <- W1 + G11 + G12 + G13 + G14

Cannot deal with infinitely long recurrent
Gradient explosion, vanishing

Recurrent neural network (RNN)

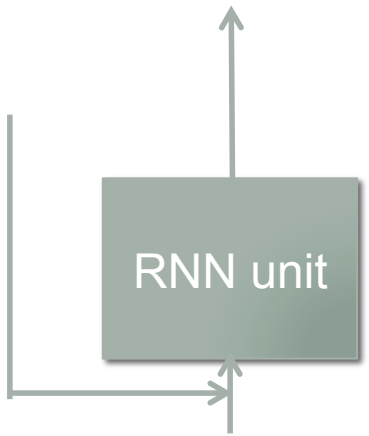


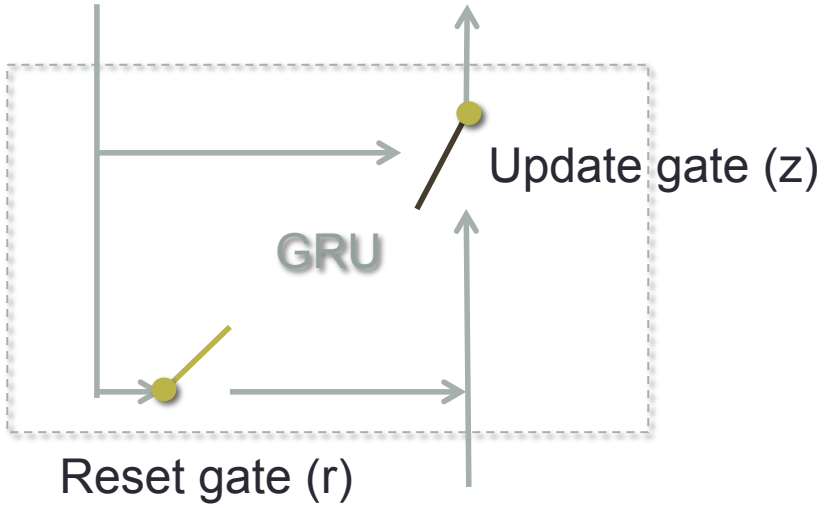
Problem2: needs a way to stop remembering

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

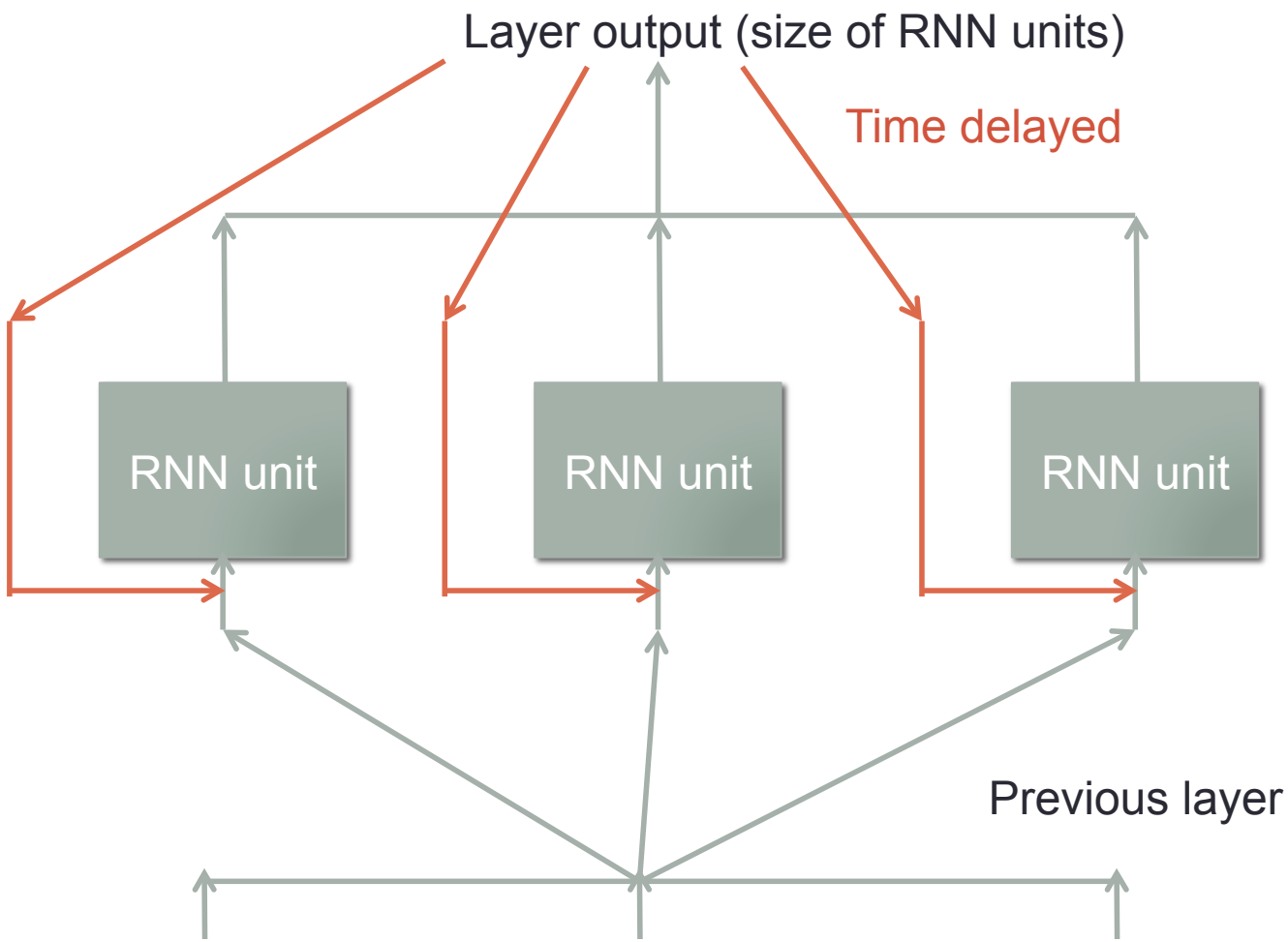
Gated Recurrent Unit (GRU)

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



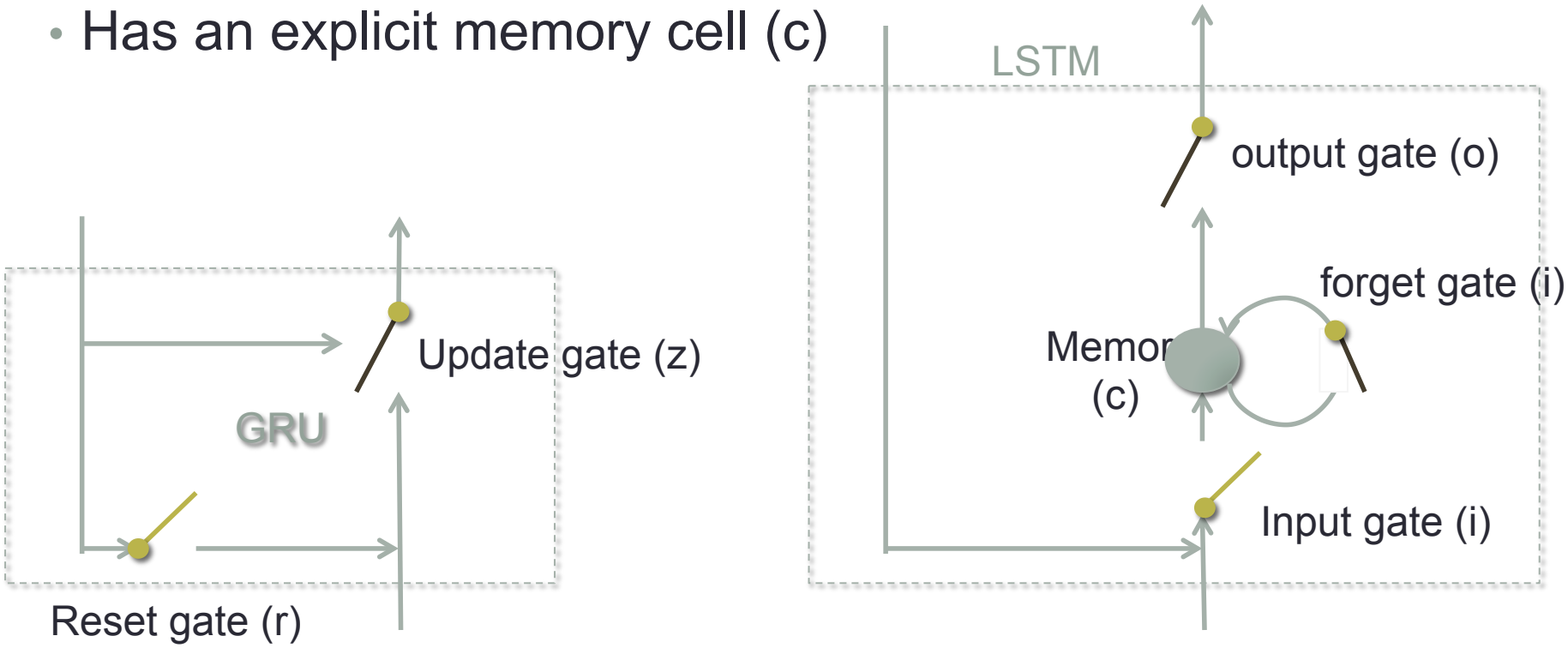


Gated Recurrent Unit (GRU) layer



Long Short-Term Memory (LSTM)

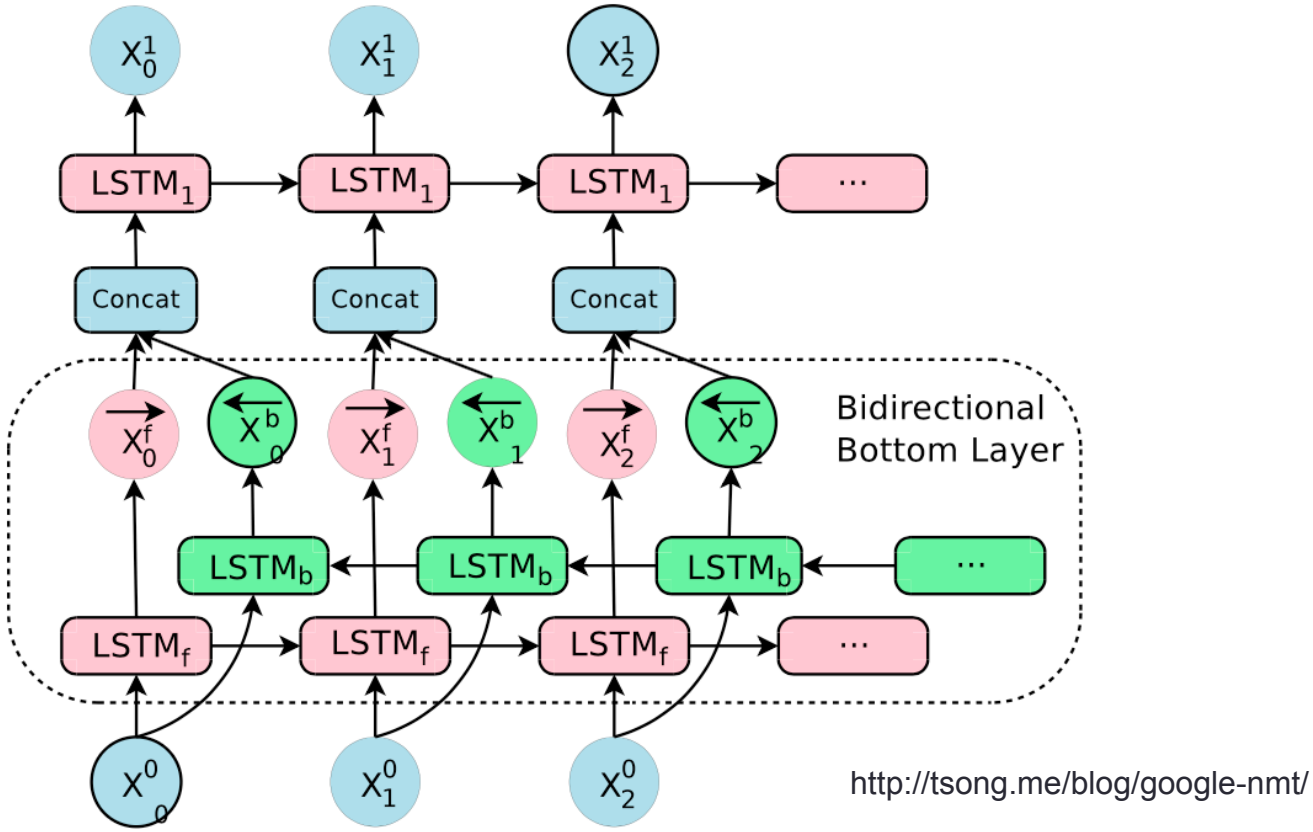
- Have 3 gates, forget (f), input (i), output (o)
- Has an explicit memory cell (c)



Both works for data with time dependency. Try GRU first

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



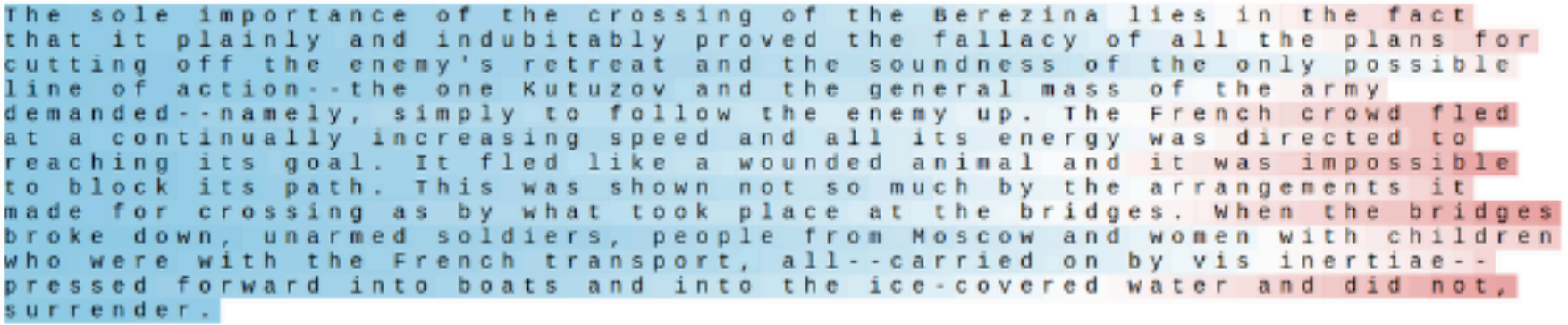
http://tsong.me/blog/google-nmt/

Real time bi-directional LSTM

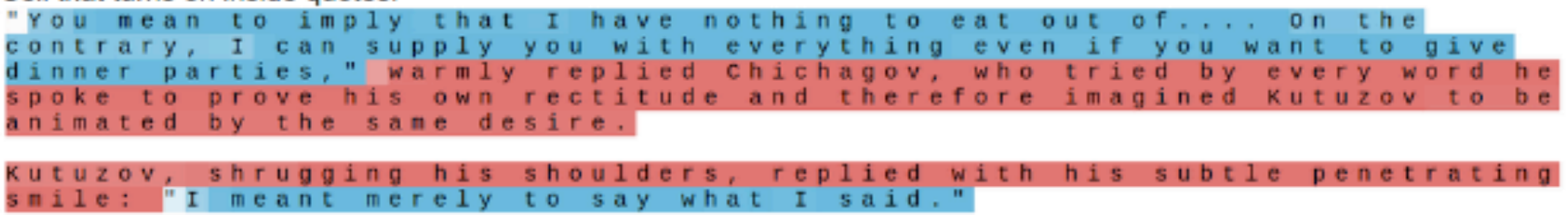
- For real time applications (speech), only “look ahead” a certain amount of time steps
- For text, bi-directional is easily applied

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

Encoder-decoder

- We know neural networks can learn representations internally
- Can we use those internal representations?

One hot encoding

- Categorical representation is usually represented by one hot encoding

- Categorial 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

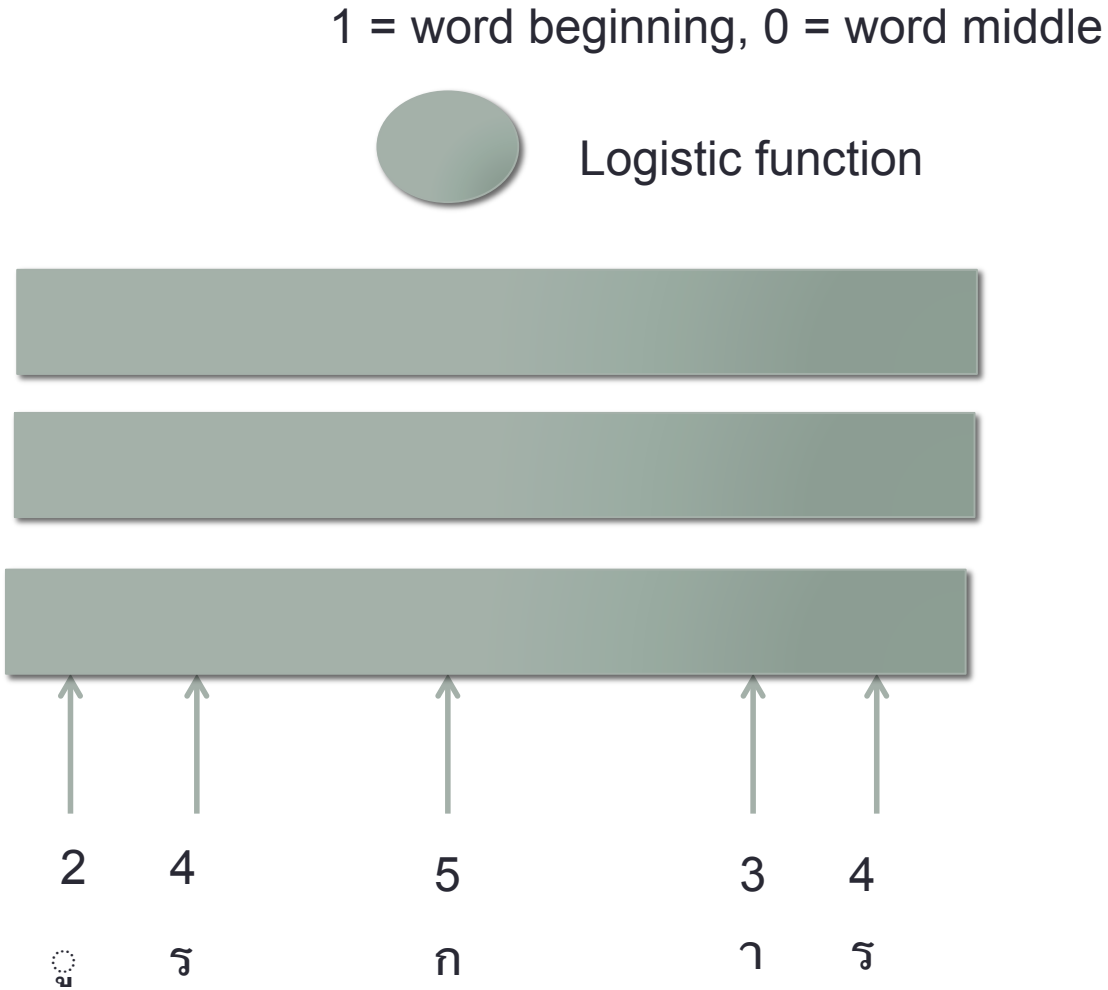
Dense representation

- 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

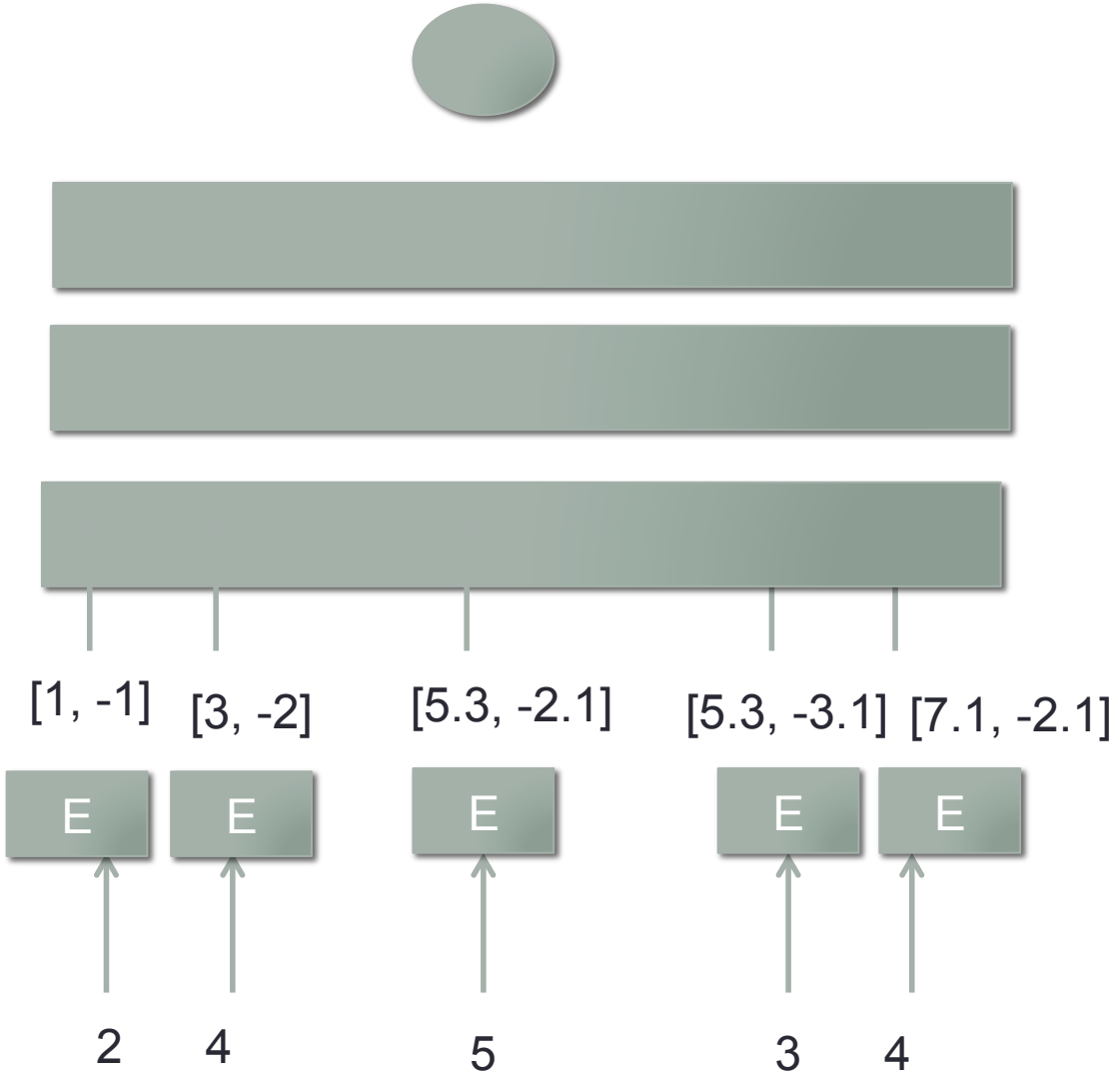
Word segmentation with fully connected networks



Adding embedding layer

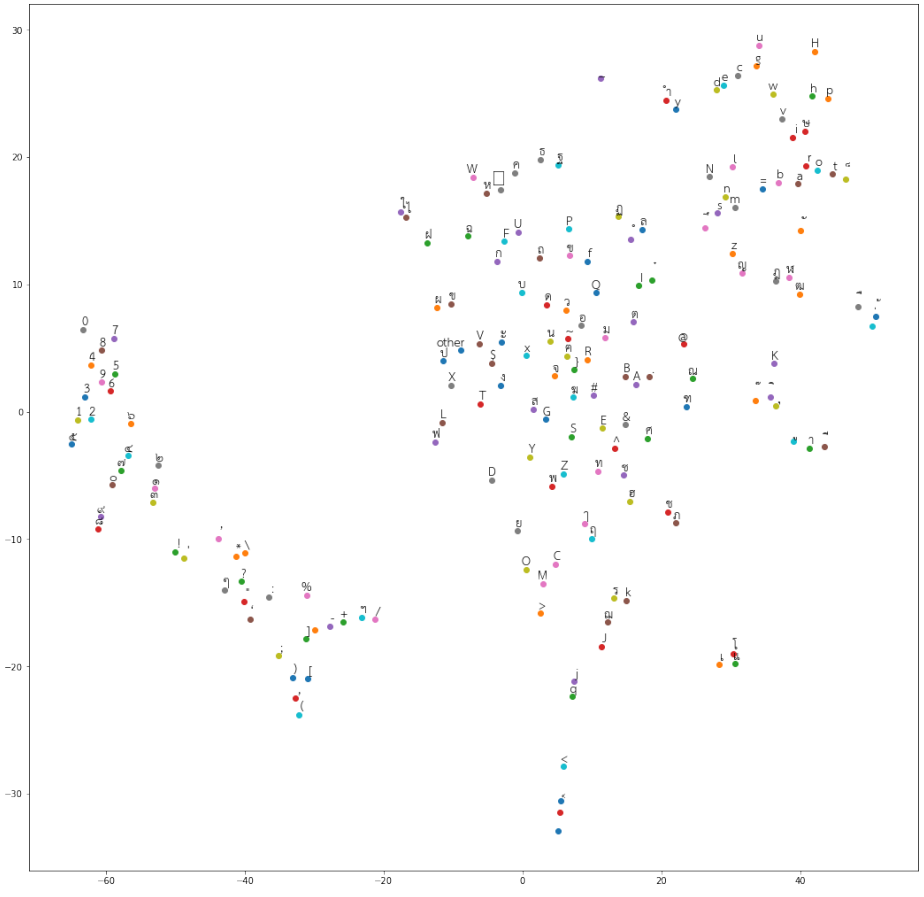
Embedding layer shares the same weights

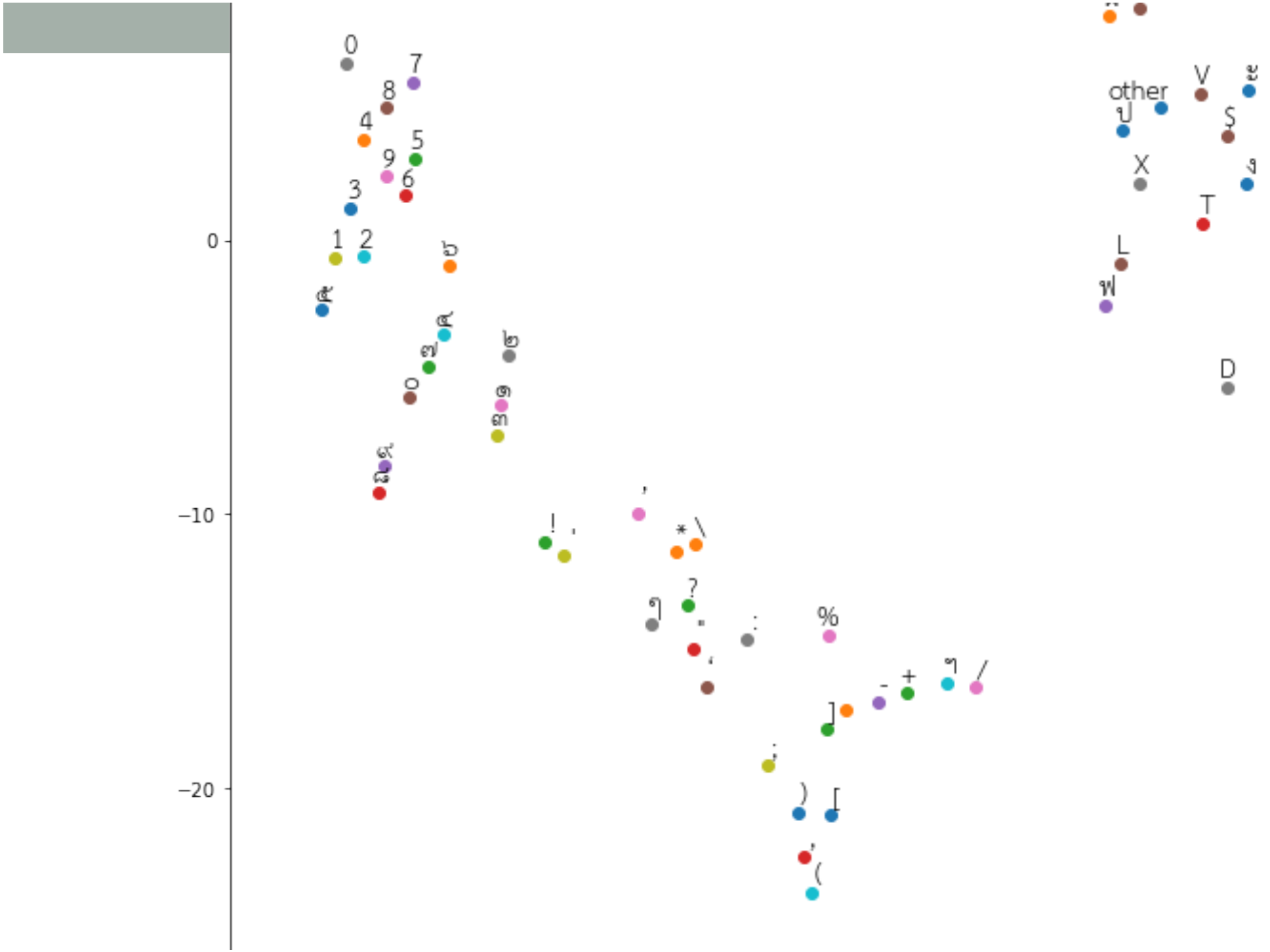
Parameter sharing!

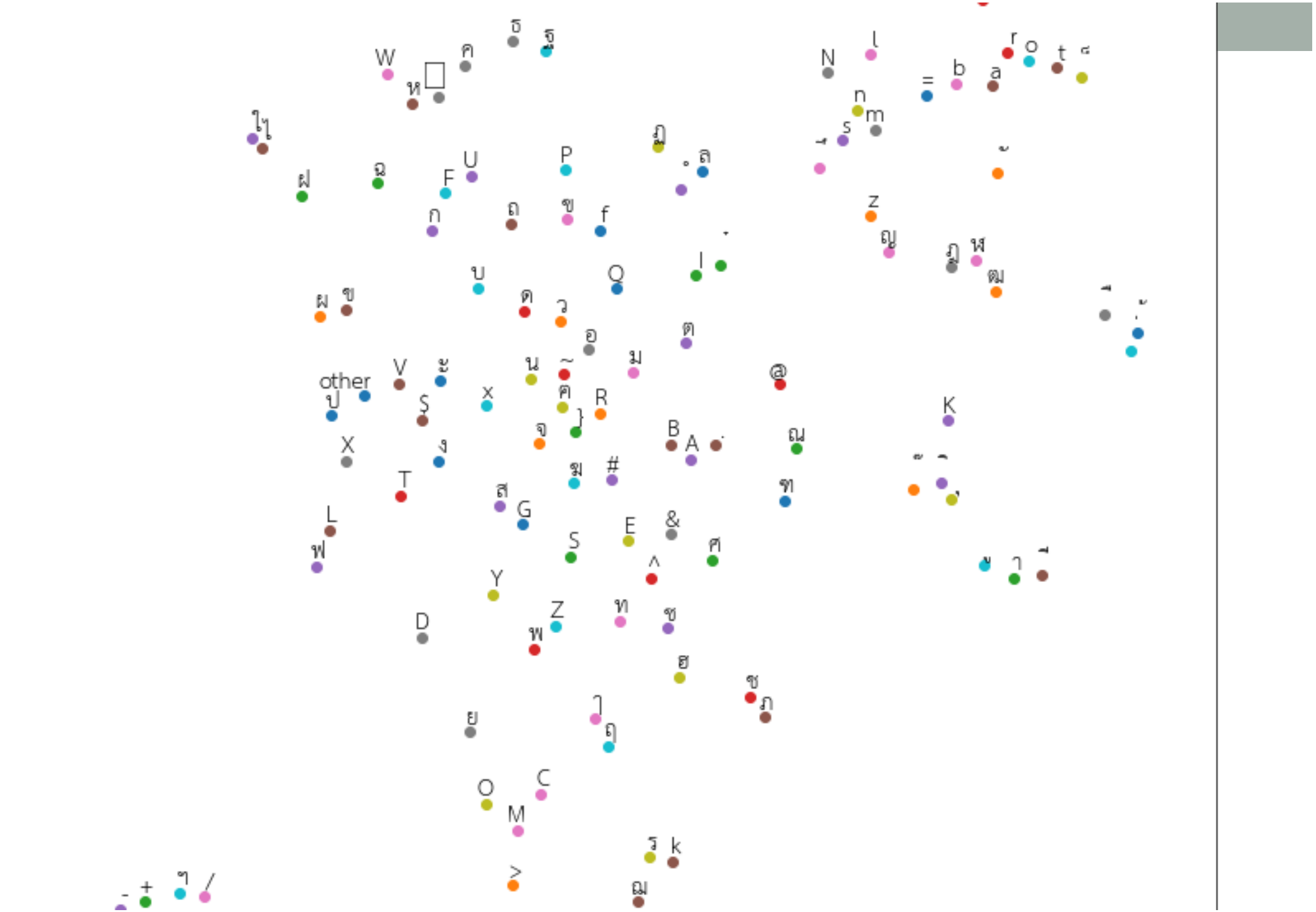


Embedding and meaning (semantics)

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

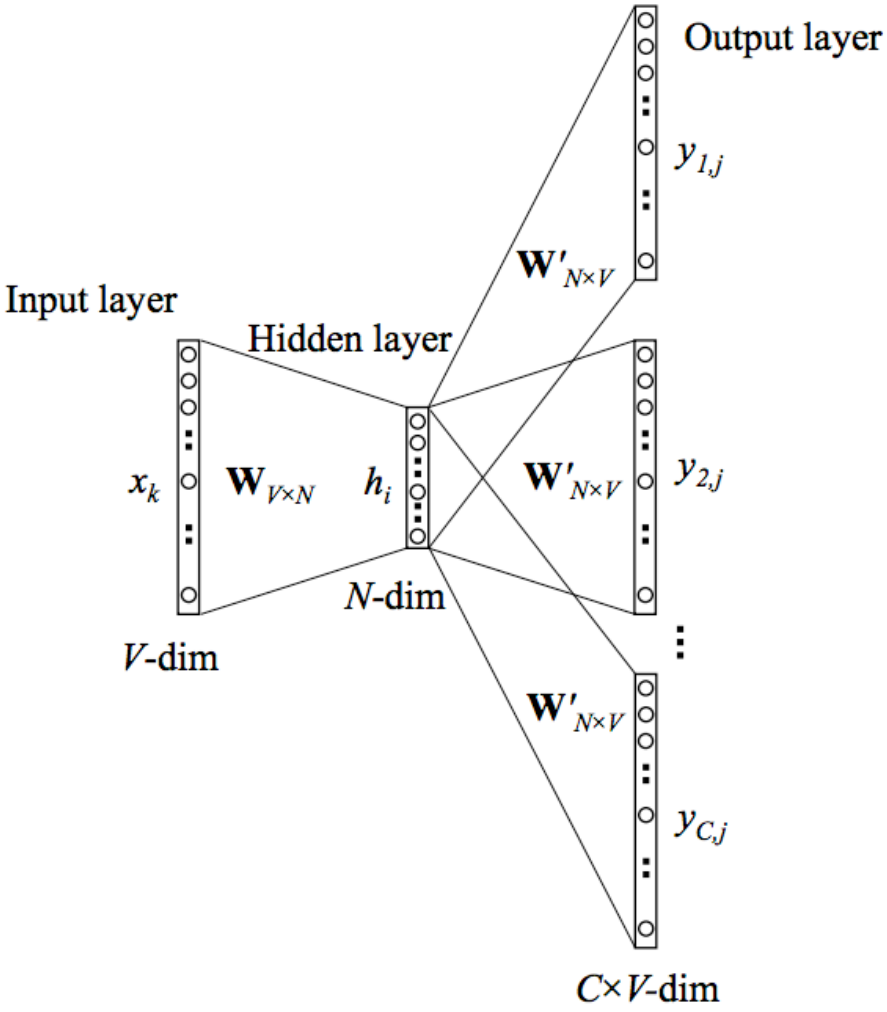



0
−10
−20
other


other

Word2Vec

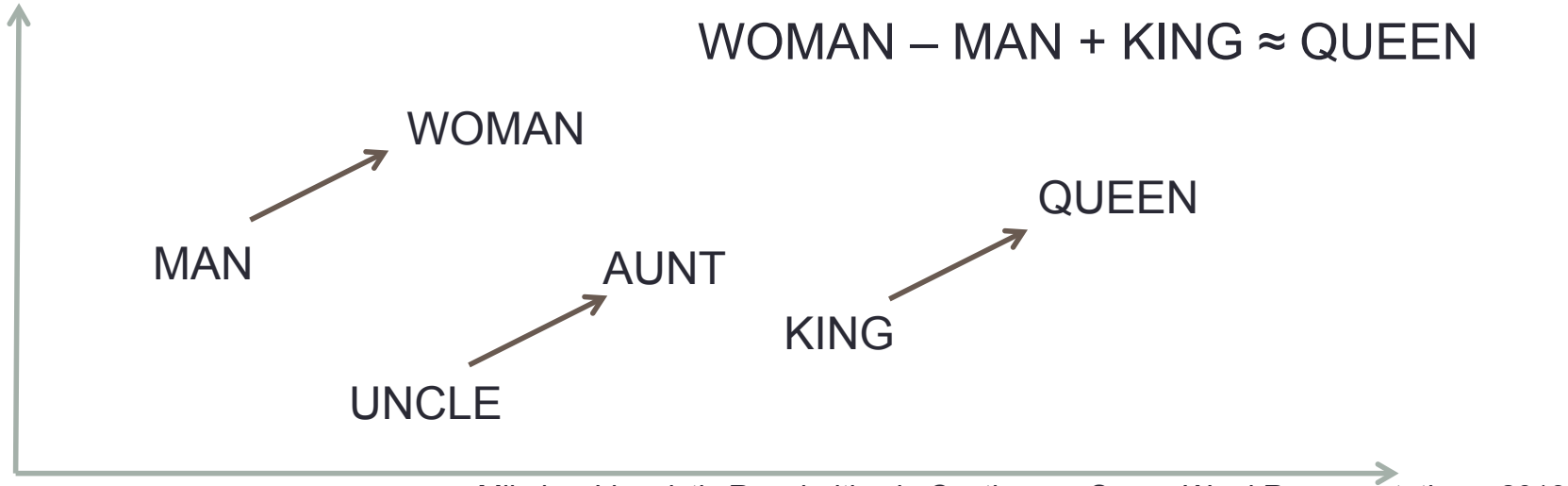
- Maps a word to a vector representation using neural networks
- Input word, predicts word around it
  - Words meaning can be captured by the context words
- Mary ___ an apple.



Mikolov, Linguistic Regularities in Continuous Space Word Representations, 2013

Word2Vec

- Vectors from similar words are near each other
- You can also add and subtract meanings just like numbers!

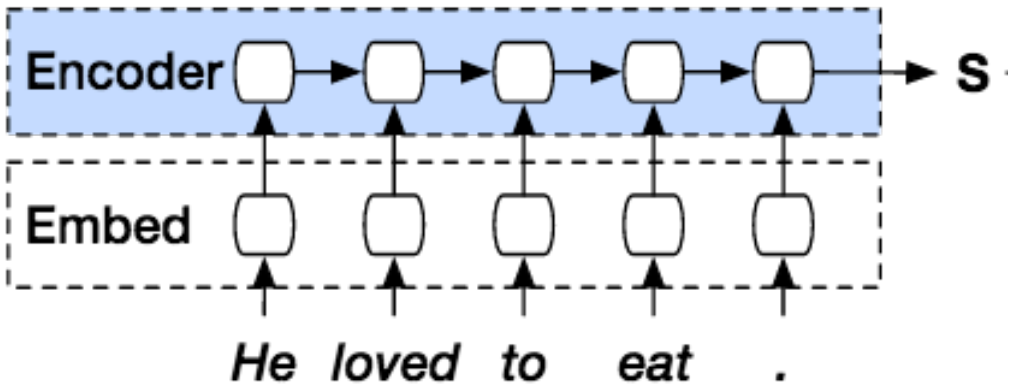


Mikolov, Linguistic Regularities in Continuous Space Word Representations, 2013

Example code for Thai: https://github.com/fooljames/tf-text-workshop

Sentence encoder

- How can we represent sentences?
- Similarly, use the internal states as the representation
- LSTMs/GRUs are good for sequence task

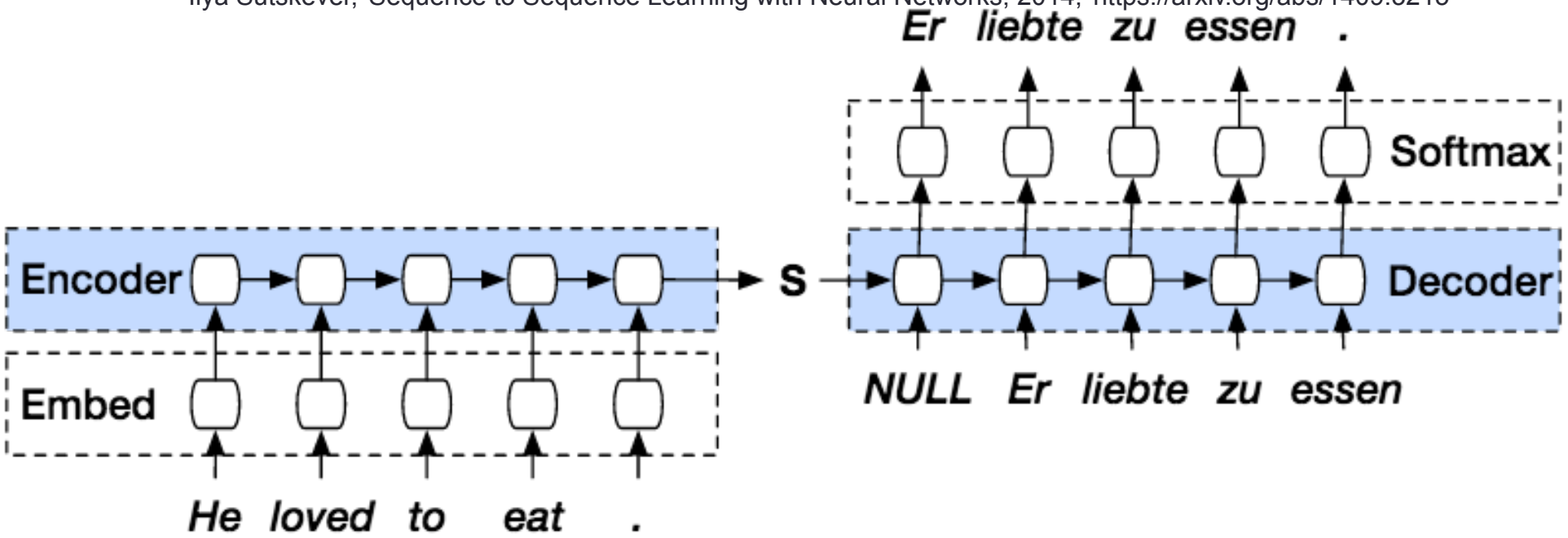


https://smerity.com/articles/2016/google_nmt_arch.html

Machine translation using encoder-decoder

- Use another LSTM as the decoder
- Input to the LSTM is the encoded sentence and the previously output word

Ilya Sutskever, Sequence to Sequence Learning with Neural Networks, 2014, https://arxiv.org/abs/1409.3215

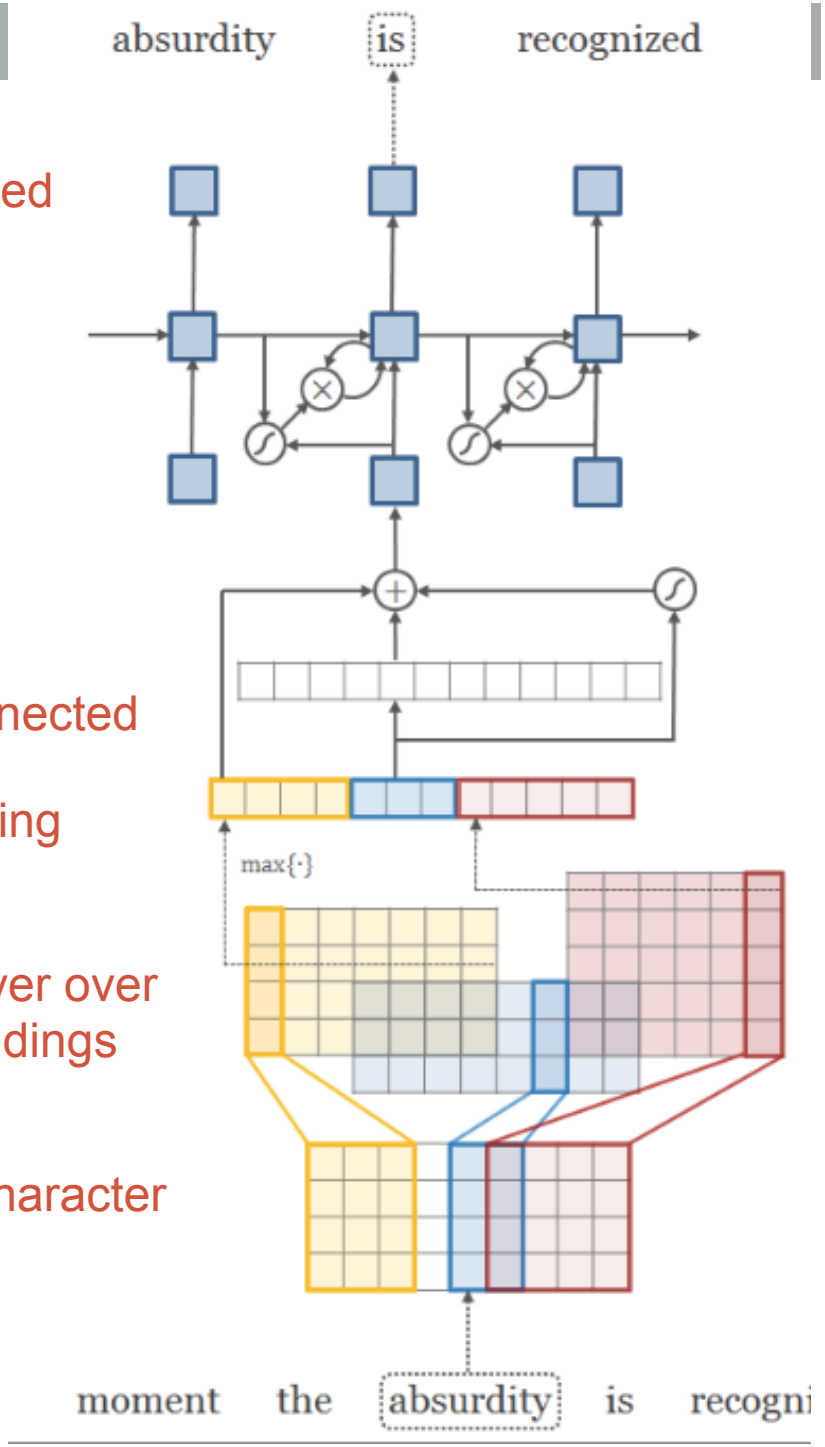


DNN Legos

- 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

Case study: CharCNN

- Input: previous characters
- Output: next word

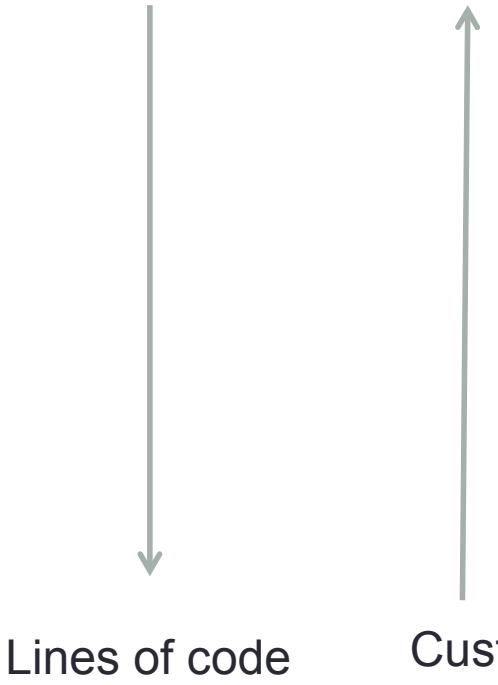


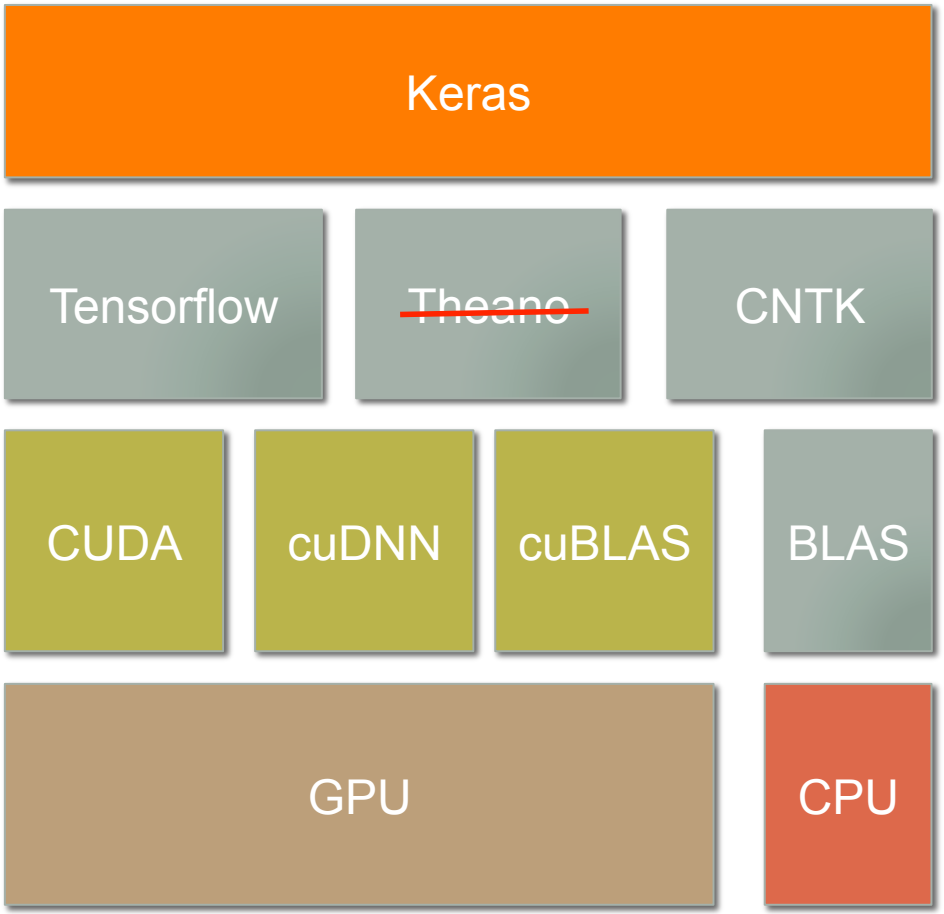
CNN over characters capture character sequence patterns

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

What toolkit

- Tensorflow – lower level
- Keras – higher level





Summary

- Linear regression
  - Model with parameters
  - Minimizing loss function
- Fully connected neural networks
  - Fully connected as cascading of linear regression + non-linearity
- Convolutional neural networks
  - Convolutional layer
  - Pooling layer
- Recurrent neural networks
  - Remembering the past
  - GRU/LSTM

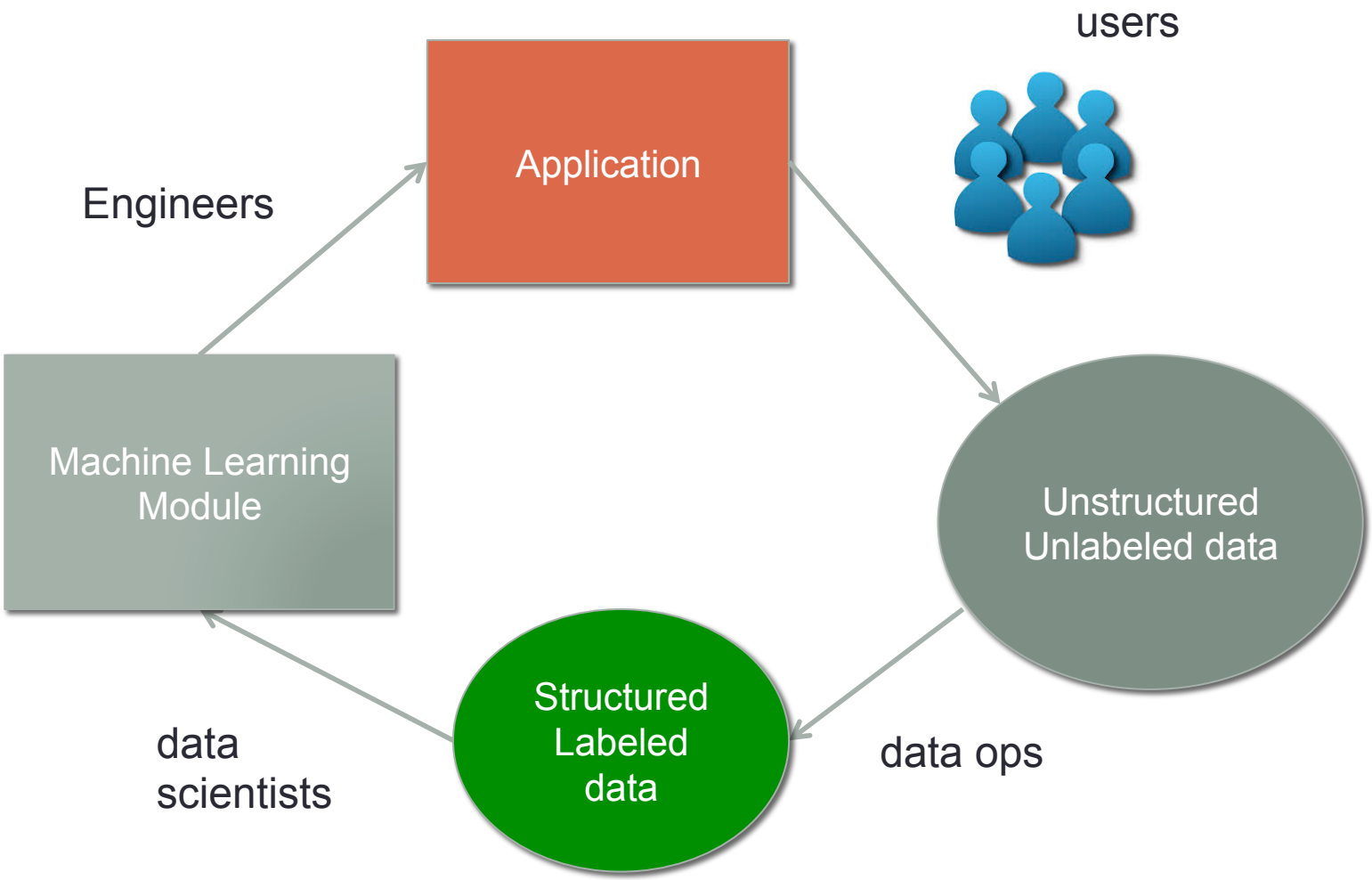
Cautionary notes

- “There is no free lunch.”
- The **“No Free Lunch”** theorem states that there is no one model that works best for every problem.
- Depends on
  - Nature of the task
  - Nature of the data
  - Amount of data
- Which model is the best?
  - Try it on your problem.

“Deep learning is not magical.”

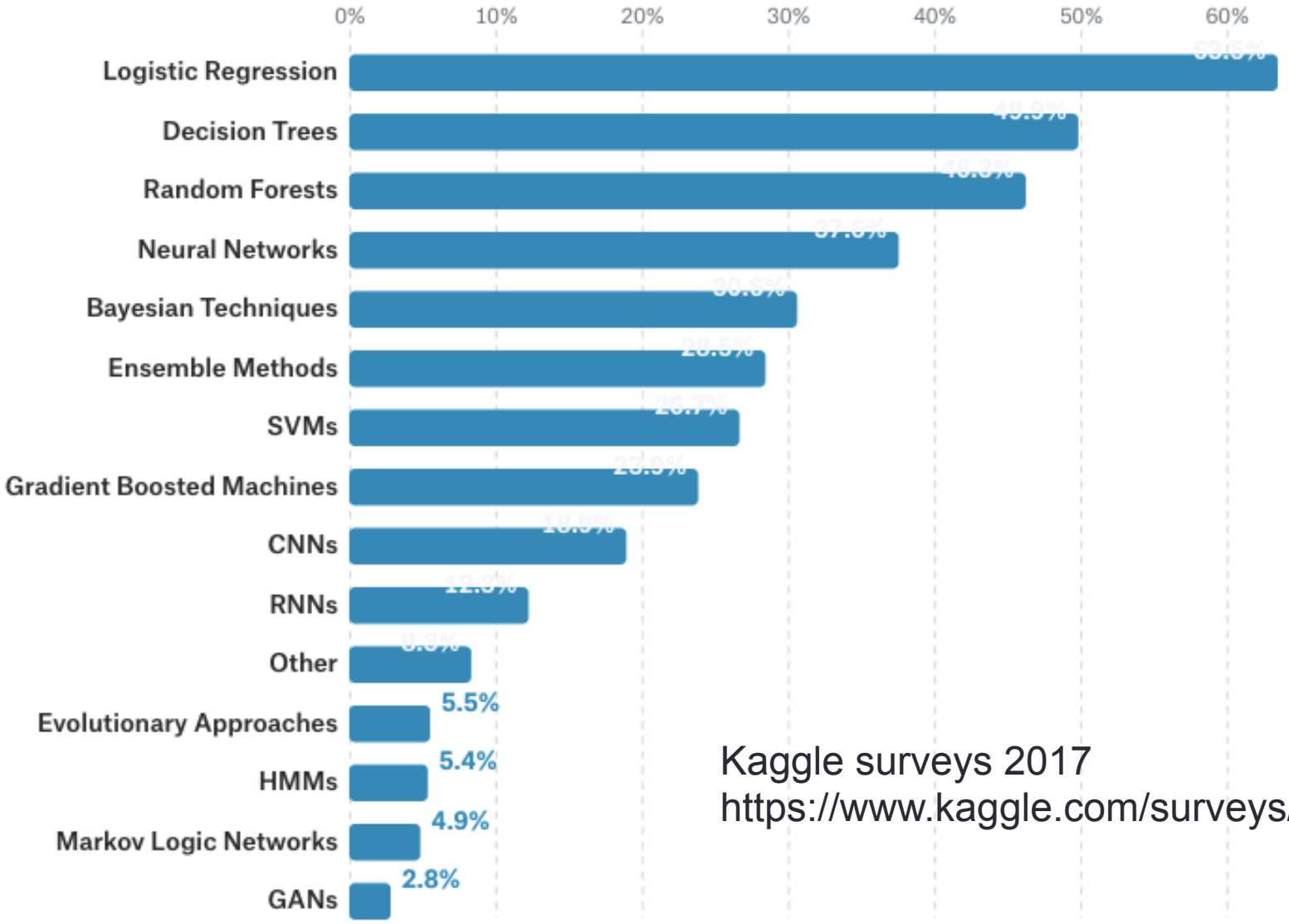
Next class we will learn other models that can augment or be used instead of deep learning in other scenarios

Getting into the data cycle



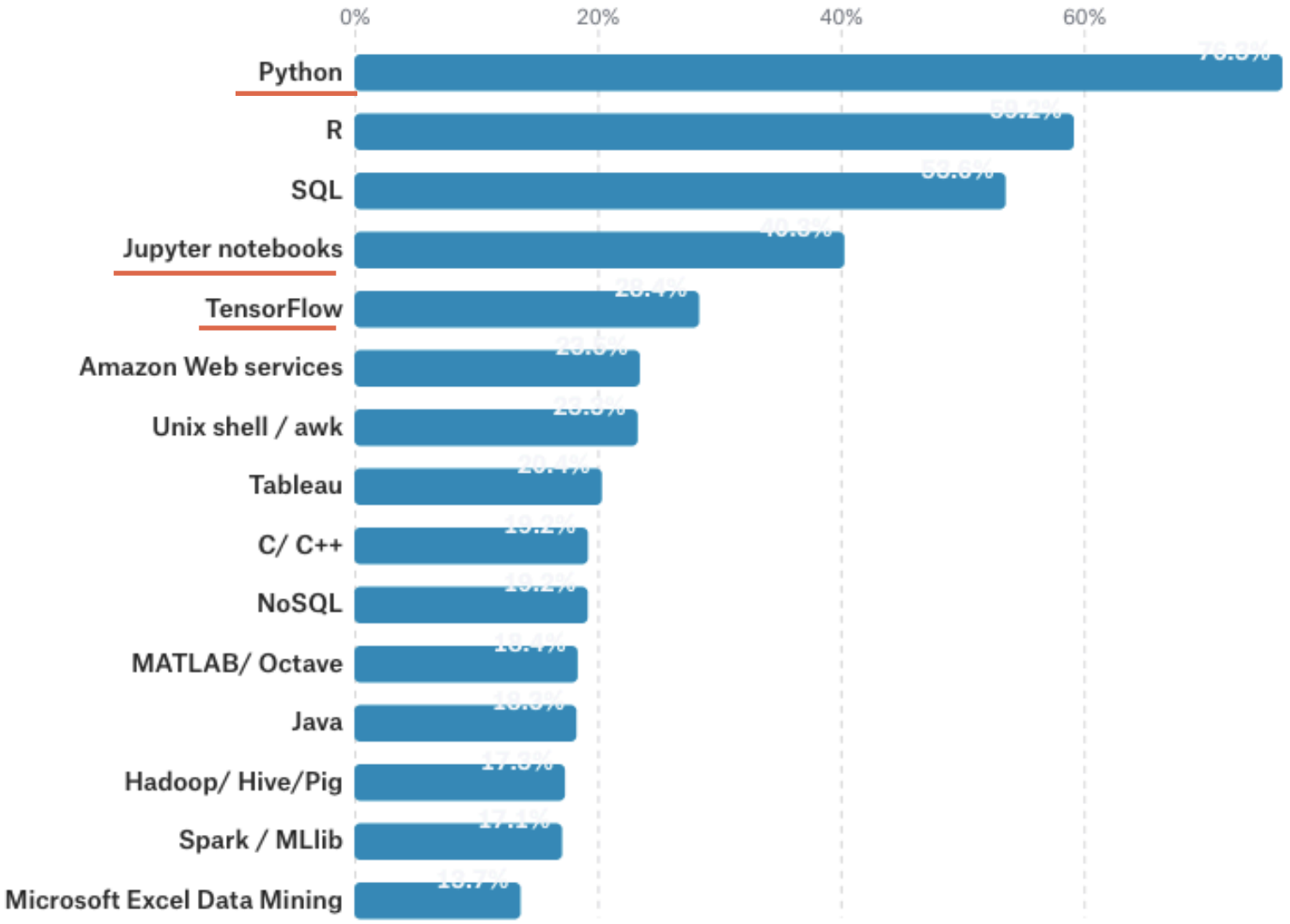
Launch fast, and iterate

What models do data scientist use?



Kaggle surveys 2017
https://www.kaggle.com/surveys/2017

What tools?


0%
20%
40%
60%
Python
R
SQL
Jupyter notebooks
TensorFlow
Amazon Web services
Unix shell / awk
Tableau
C/ C++
NoSQL
MATLAB/ Octave
Java
Hadoop/ Hive/Pig
Spark / MLlib
Microsoft Excel Data Mining

Jupyter demo

- Jupyter is a python IDE
- How to install
  - http://jupyter.readthedocs.io/en/latest/install.html
  - Use the anaconda method
- Server-client model, access via a browser

- Numpy – math library (matrix manipulation)
- Scipy – library that augments numpy, higher level tasks
- Matplotlib – plotting library
- Panda – library for playing with data
- Scikit-learn – machine learning libraries for everything besides deep learning

Homework

- Install jupyter and play with numpy_tutorial and titanic_tutorial.

- https://github.com/ekapolc/TrueVoice-Training/tree/master/tutorials