NEURAL NETWORKS

At VISTEC

INTRO TO MACHINE LEARNING

ML vs PR vs DM vs KDD

- “The short answer is: None. They are … concerned with the same question: how do we learn from data?”

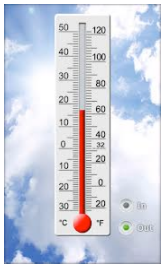
  Larry Wasserman – CMU Professor

- Nearly identical tools and subject matter

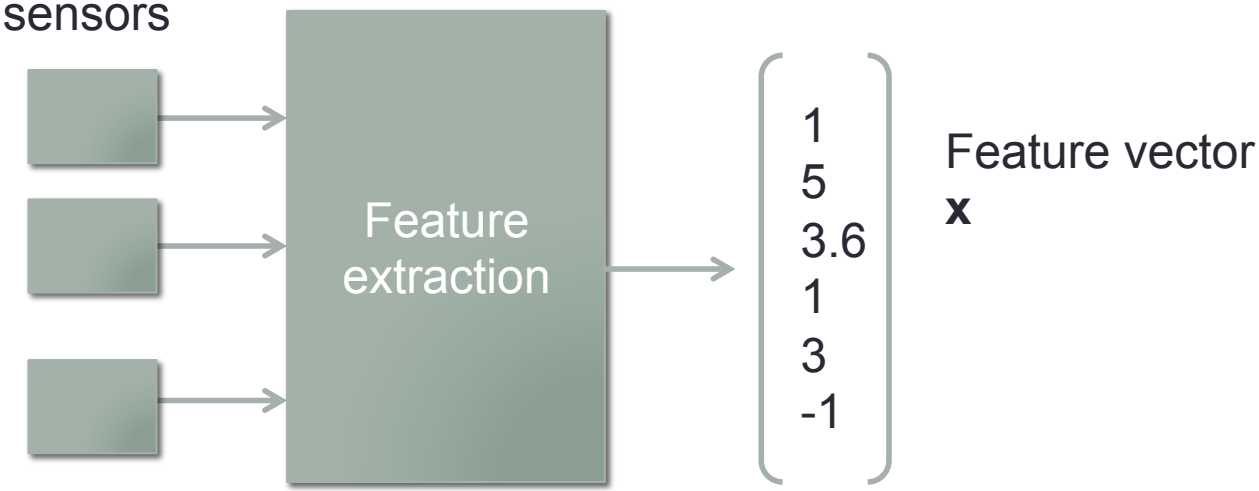
How do we learn from data?

- The typical workflow

Real world observations

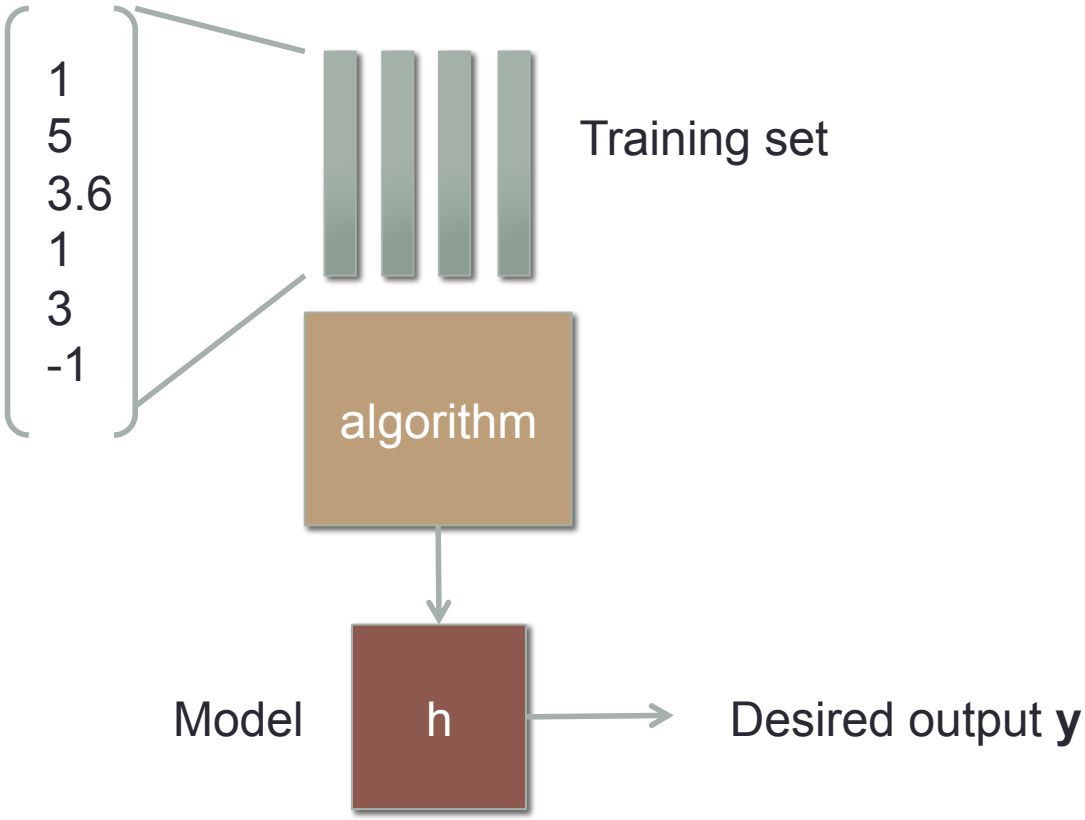


sensors



Feature vector
x

How do we learn from data?

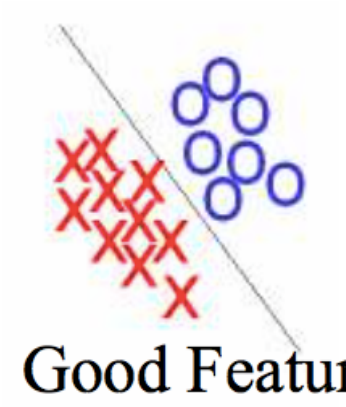


Regression task

Classification task

Feature properties

- The quality of the feature vector is related to its ability to discriminate samples from different classes

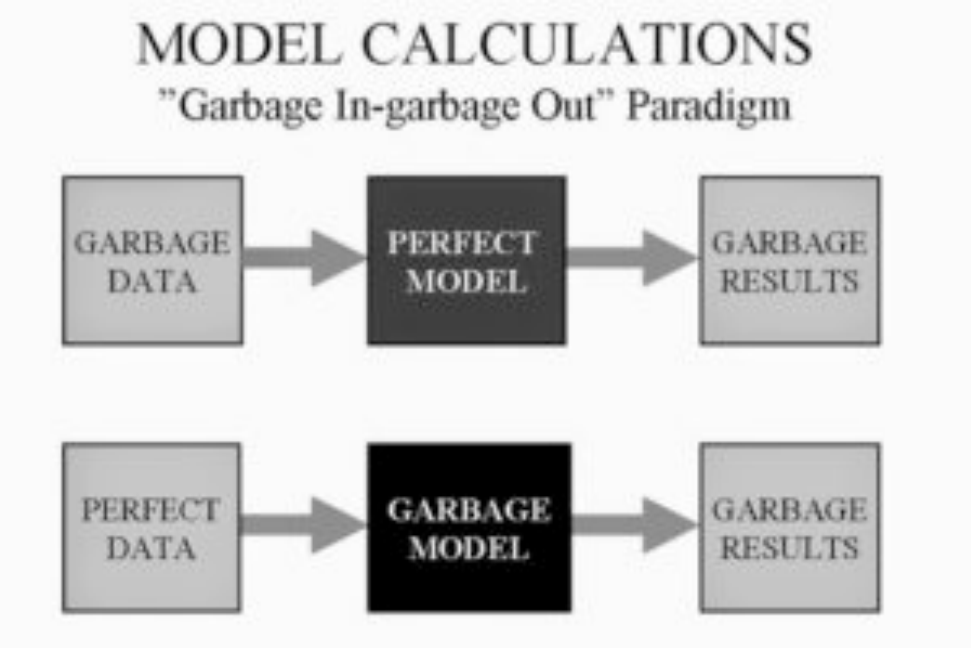


Good Features

Bad Features

Garbage in Garbage out

- The machine is as intelligent as the data/features we put in
- “Garbage in, Garbage out”
- Data cleaning is often done to reduce unwanted things



https://precisionchiroco.com/garbage-in-garbage-out/

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

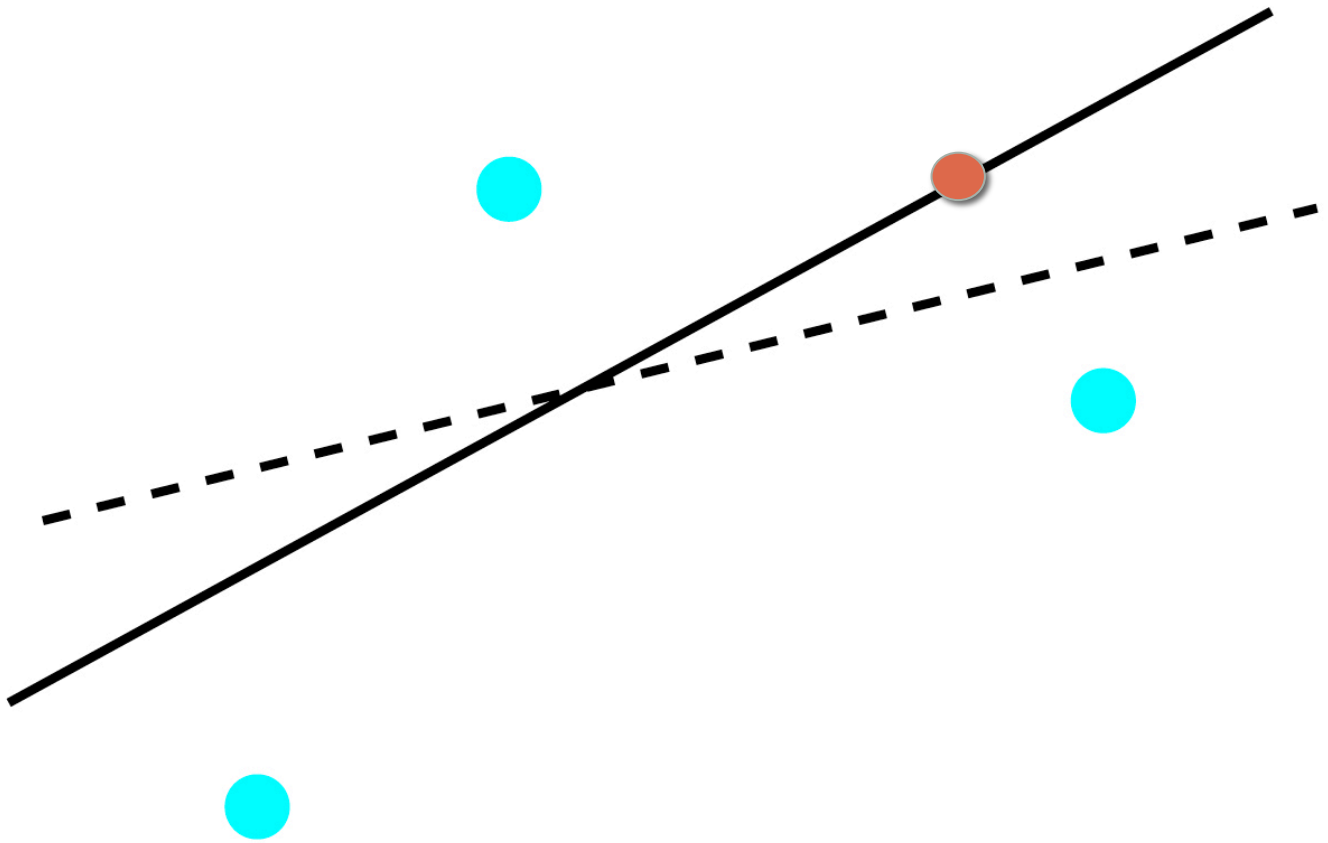
Picking θ

- Random until you get the best 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)

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

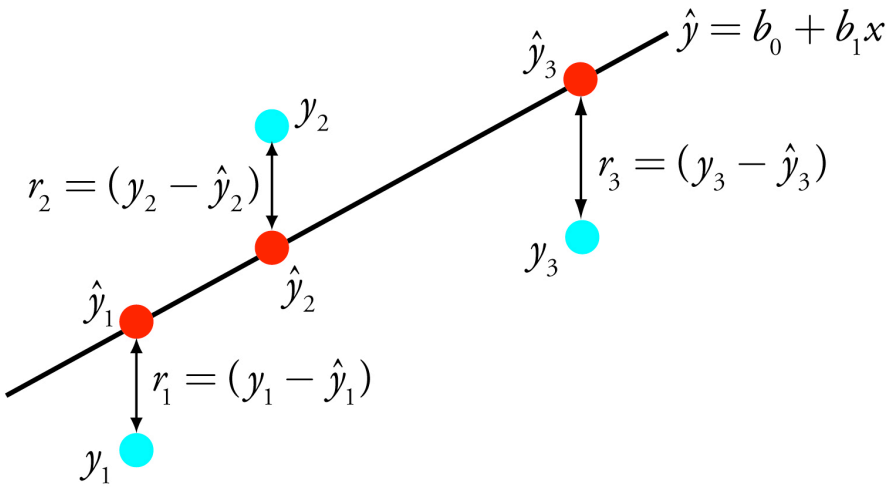
We want to pick **θ** that minimize the loss

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

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

Picking θ

- Random until you get the best performance?
  - Can we do better than random chance?

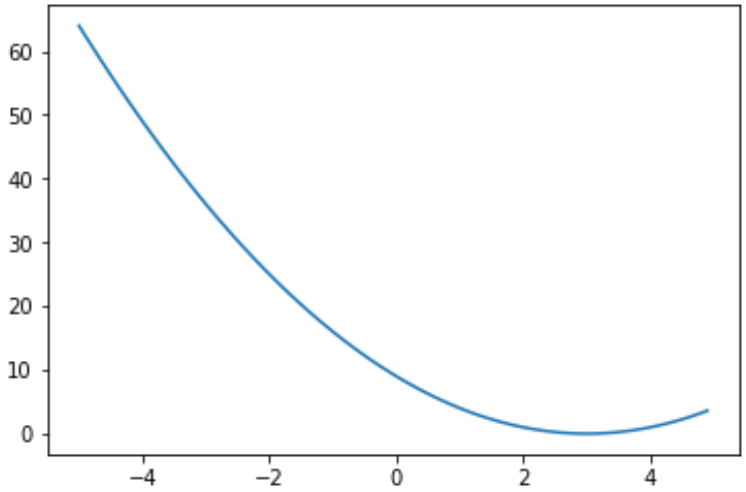
- How to quantify best performance?

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

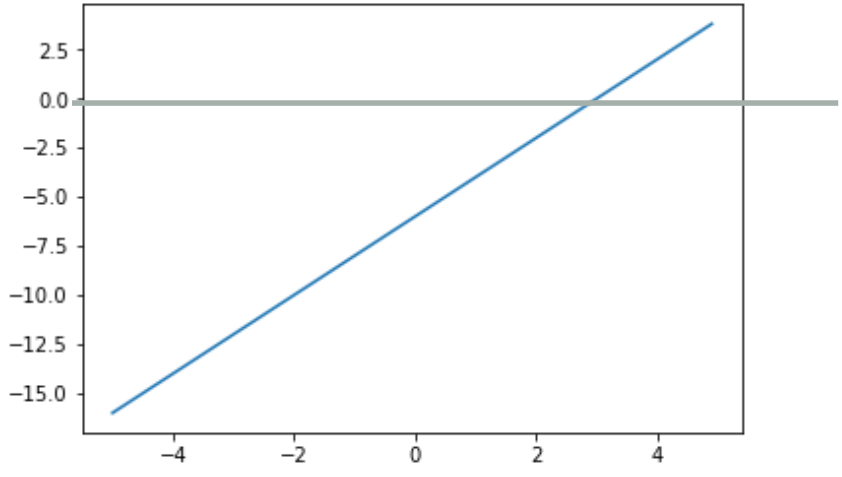
Minimizing a function

- 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



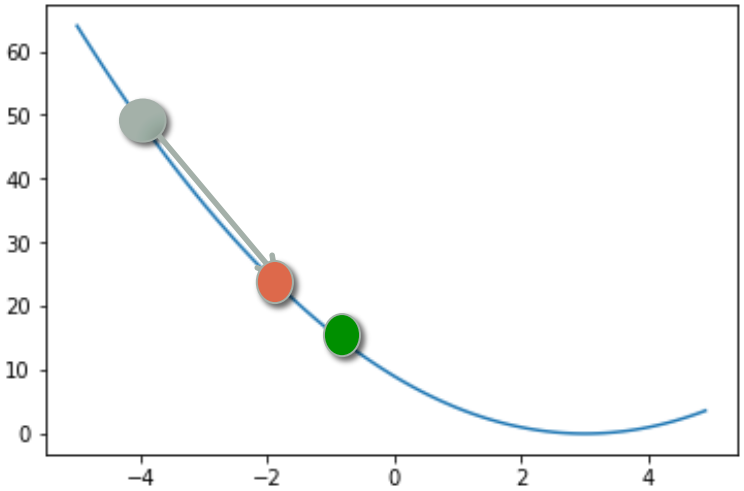
y



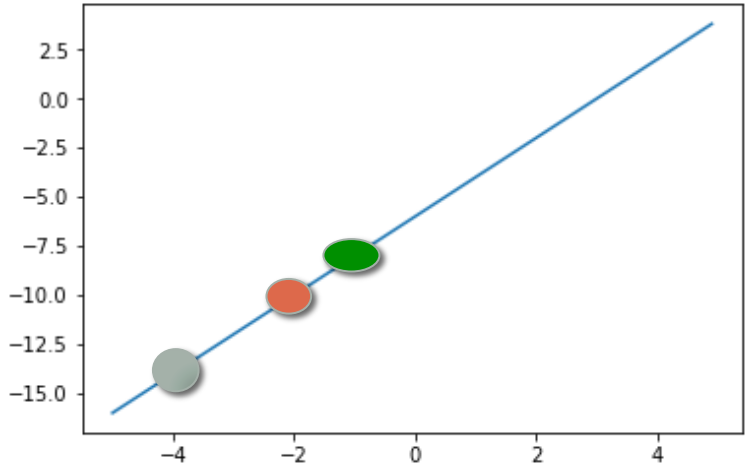
dy/dx

First what does dy/dx means?

Gradient descent



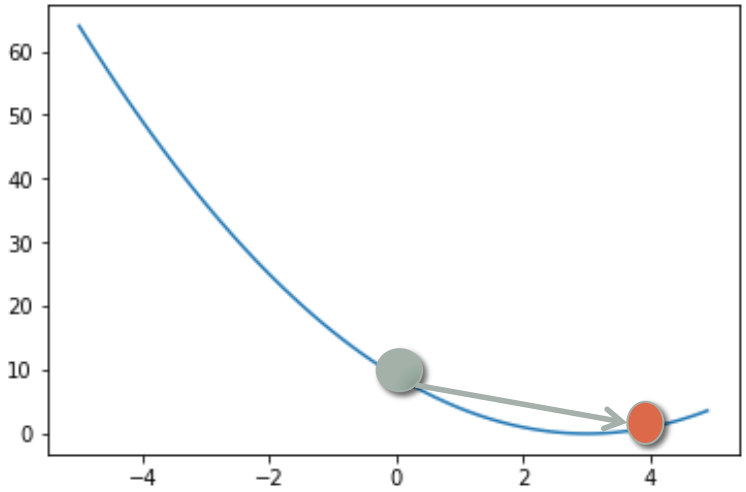
y



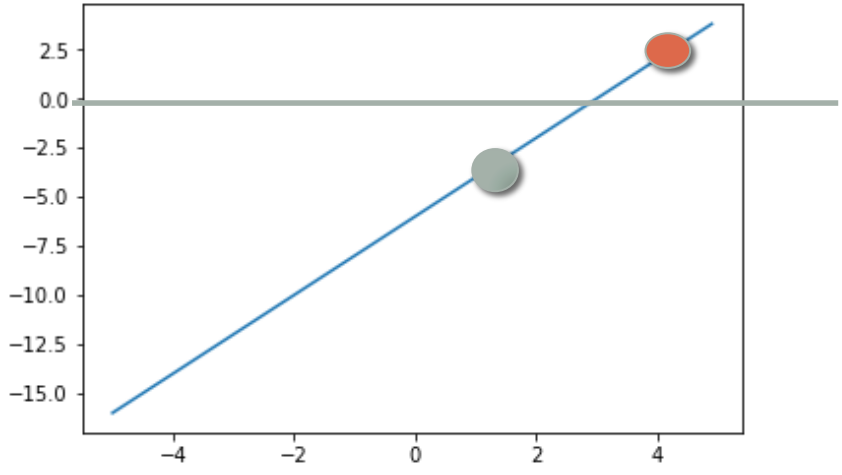
dy/dx

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

Gradient descent



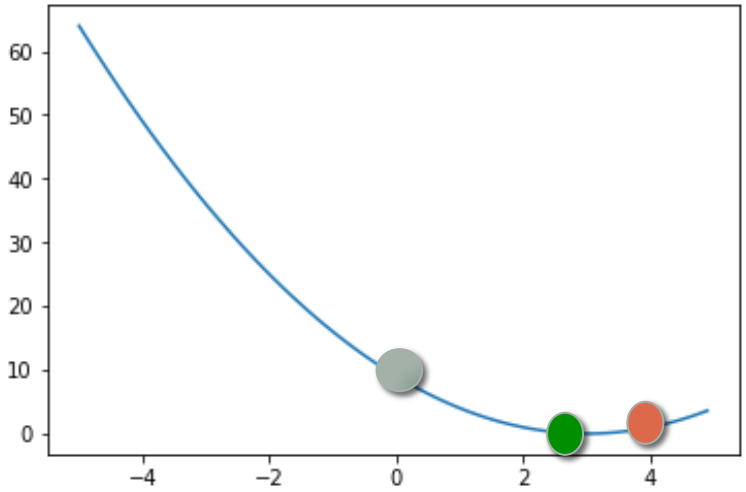
y



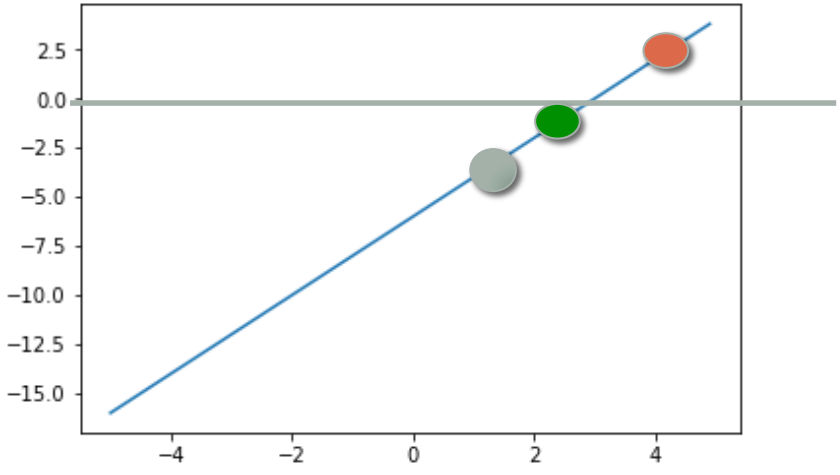
dy/dx

What happens when you overstep?

Gradient descent



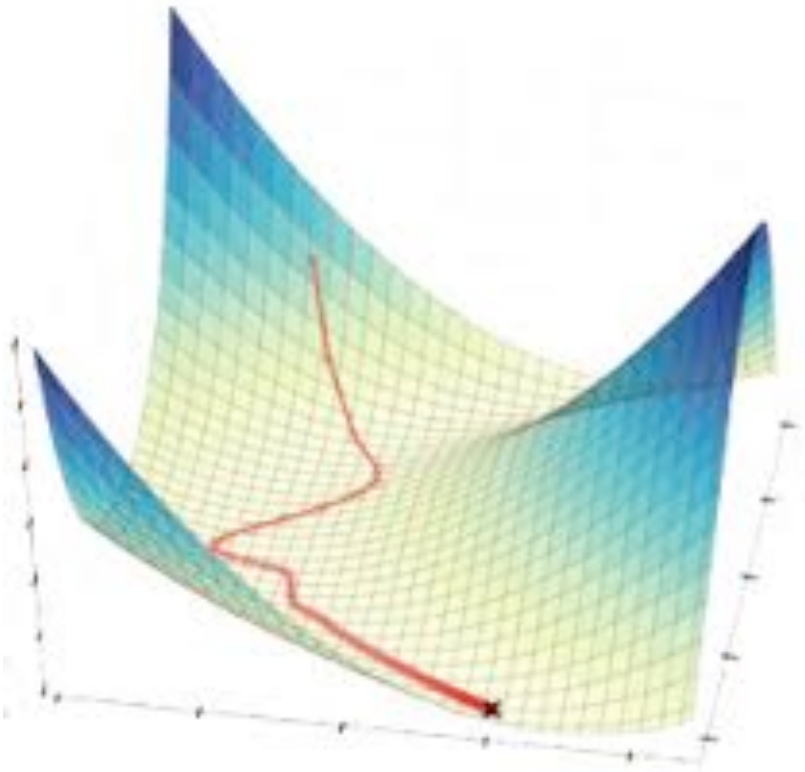
y



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

- Random until you get the best performance?
  - Can we do better than random chance?
    - Gradient descent (a better guess!)

- How to quantify best performance?

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

LMS regression with gradient descent

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

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

Interpretation?

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

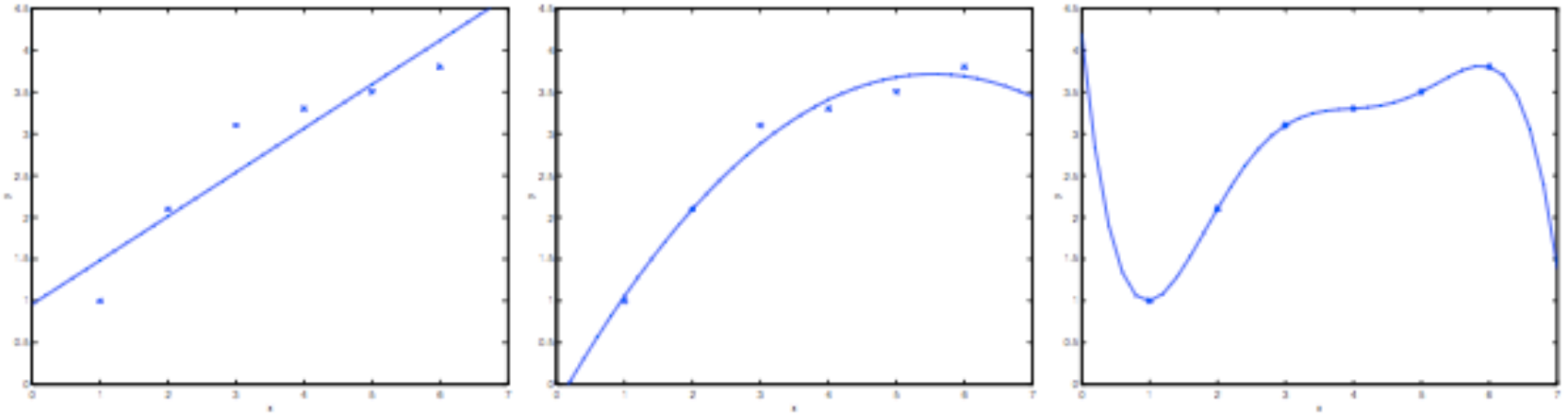
Regression with non-linear features

- 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

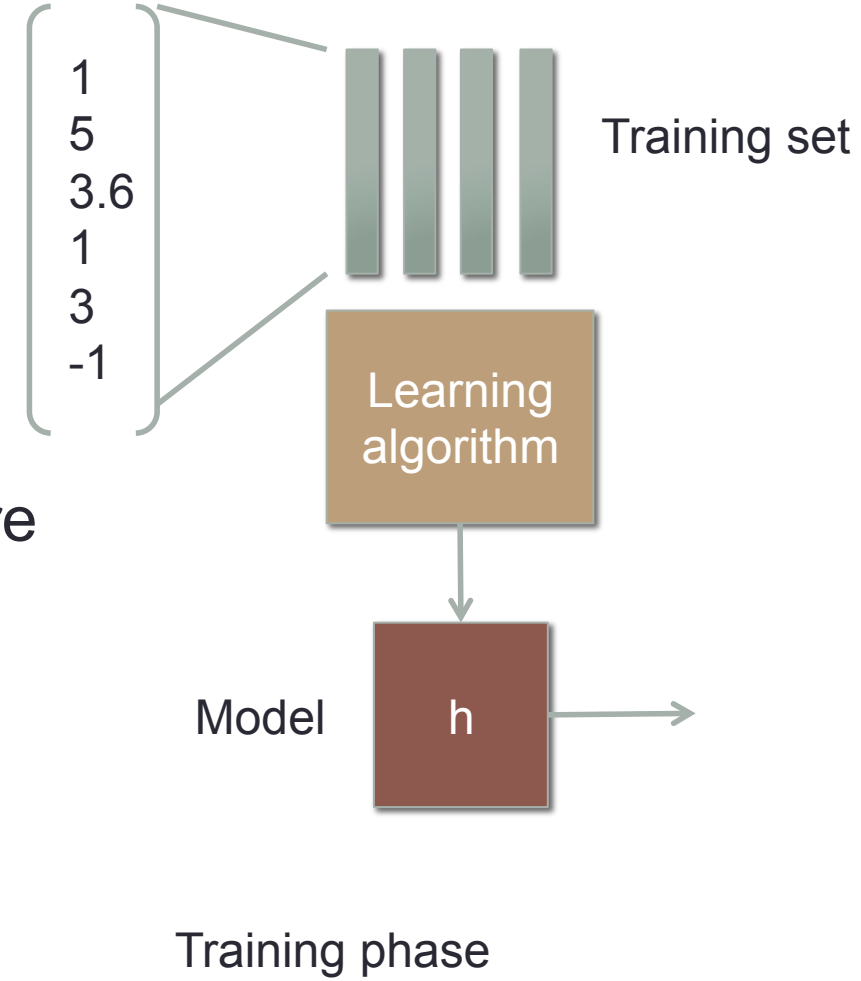
Cloth	Corn	Grass	Water	Beer	Flood?
4	6	3	10	0	**yes**
5	1	0	0	7	**yes**
6	0	3	5	7	**no**
5	0	3	12	0	**yes**
4	3	0	6	7	**?**

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?

- Not that great but we'll use it for now. (Real answer: logistic regression)

Can we use regression?

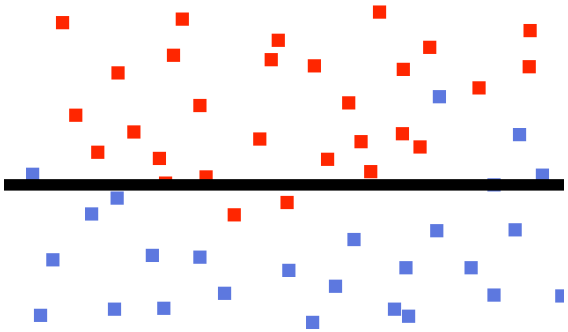
- Yes
- $h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4 + \theta_5 x_5$

- Training data
- y = 0 if not flooded
- y = 1 if flooded

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

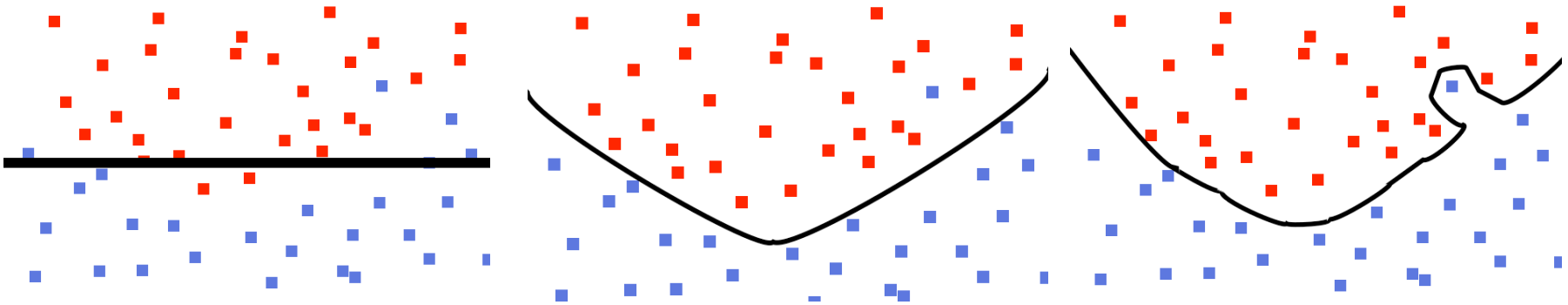
Decision boundary

- The line where our decision flips
  - h = 0



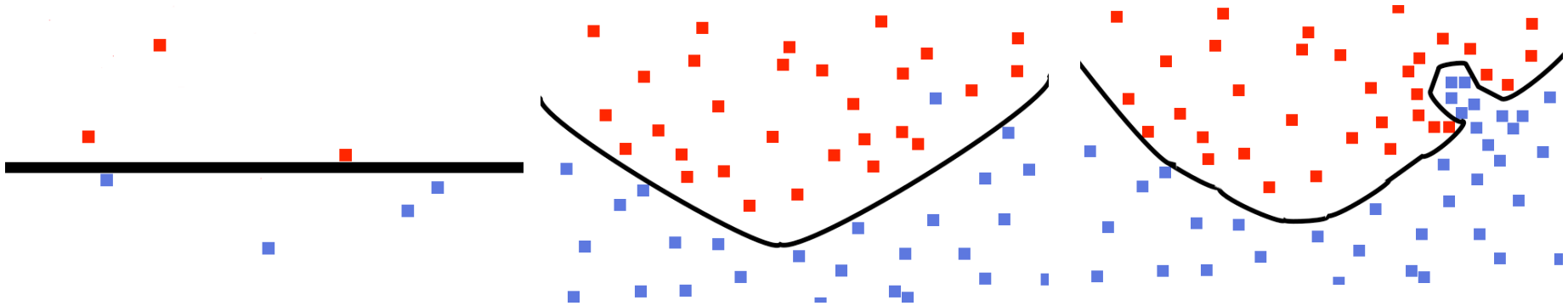
Decision boundary & overfitting

- 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



Overfitting vs Underfitting

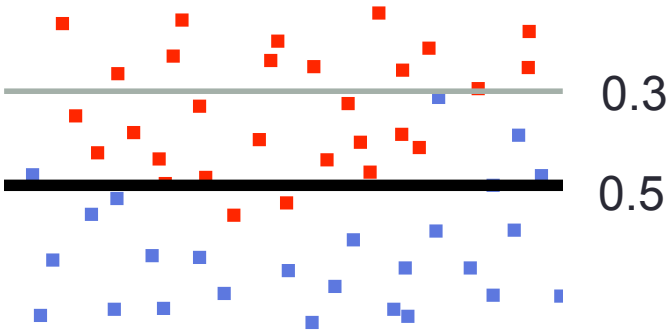
- It is hard to know beforehand how which model is the one that does not overfit or underfit
- The best model complexity depends on amount of data
- The best way to combat overftting is to evaluate multiple models on the dev set
- Data limitation is also known as the data sparsity problem



The threshold

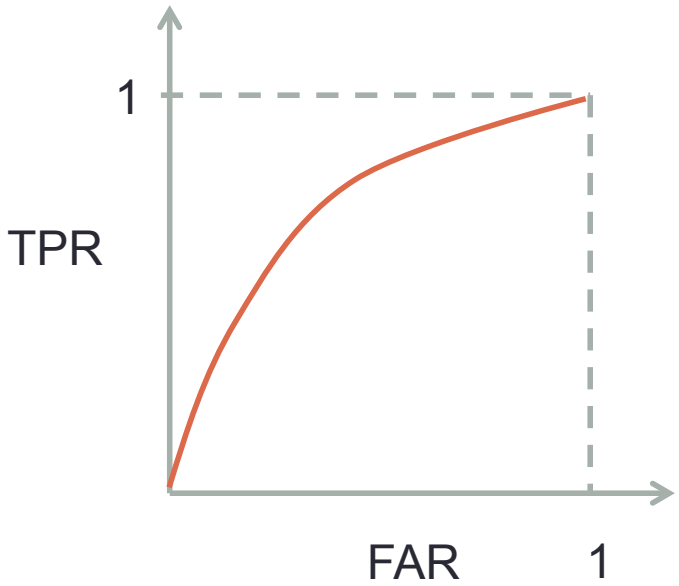
- Prediction
- If h > 0.5 flood
- If h < 0.5 no flood

- Do we need to use 0.5?



Threshold - False alarm vs True positive

- Increase threshold -> +True positive, -Miss detection, +False alarm
- Decrease threshold -> -True positive, +miss detection, -False alarm



All classifiers will have some inherent bias.
You might not want the 0.5 threshold.

NEURAL NETWORKS

DNNs (Deep Neural Networks)

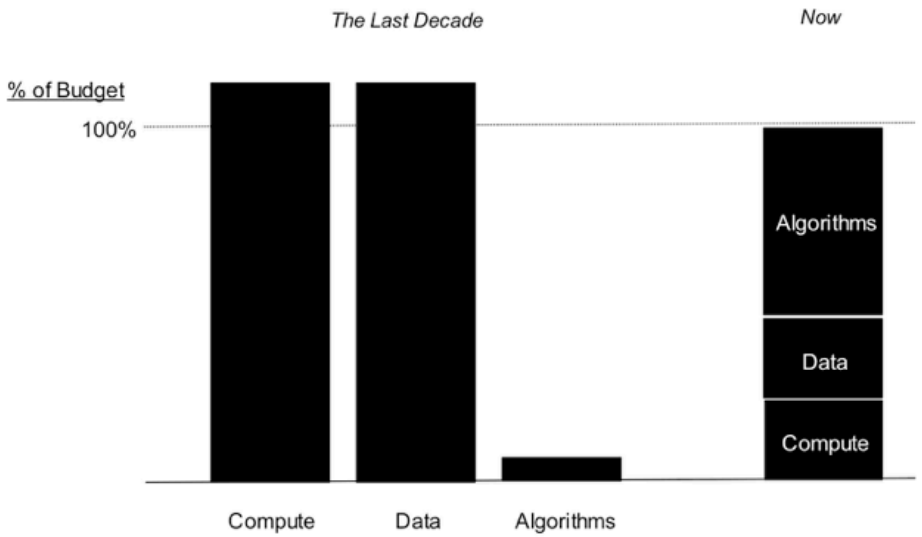
- Why deep learning?
- Greatly improved performance in ASR and other tasks (Computer Vision, Robotics, Machine Translation, NLP, etc.)
- Surpassed human performance in many tasks

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

Why now

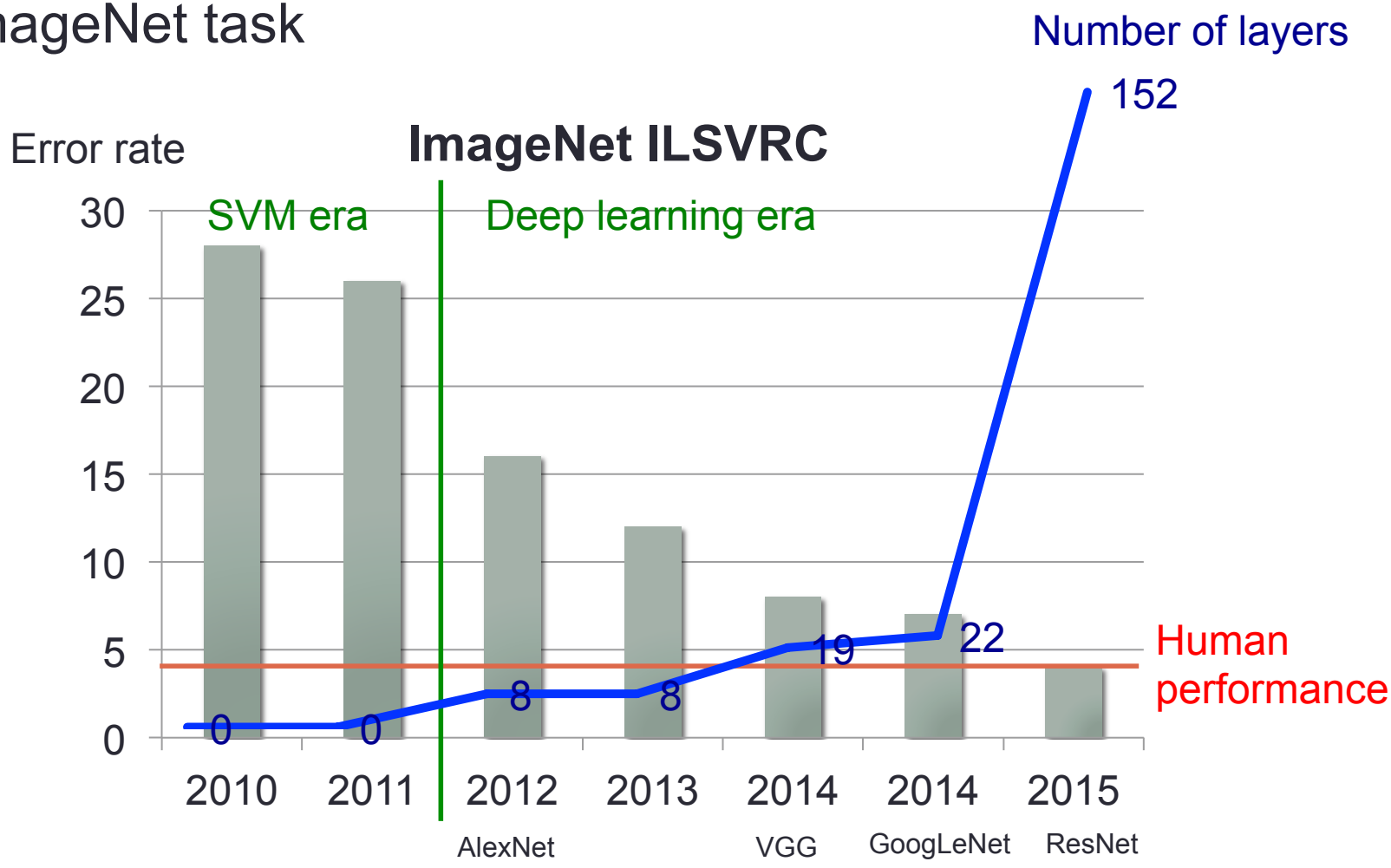
- Neural Networks has been around since 1990s
- Big data – DNN can take advantage of large amounts of data better than other models
- GPU – Enable training bigger models possible
- Deep – Easier to avoid bad local minima when the model is large



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

Wider and deeper networks

- ImageNet task

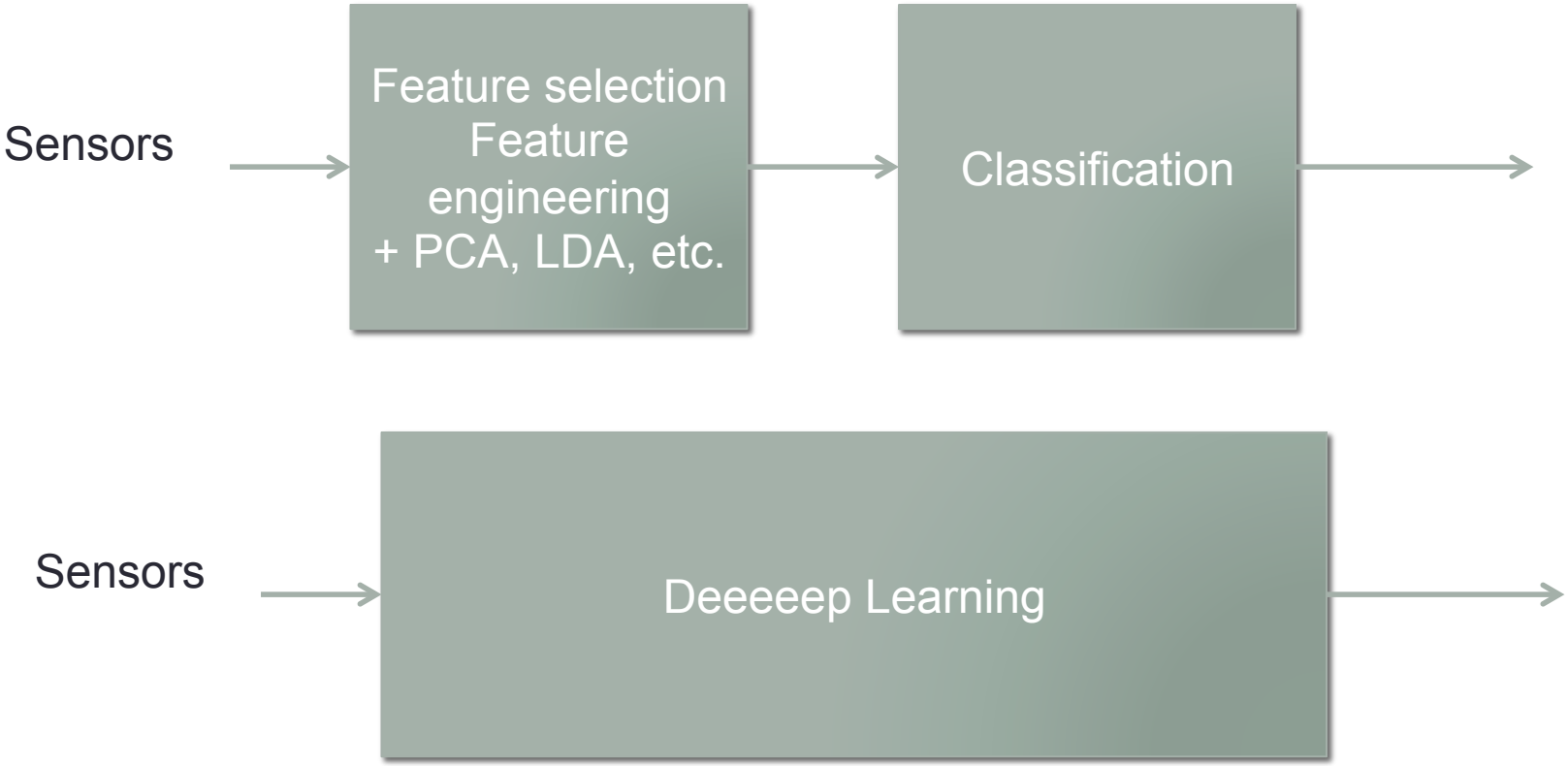


Olga Russakovsky, ImageNet Large Scale Visual Recognition Challenge, 2014 https://arxiv.org/abs/1409.0575

Why is deep learning good

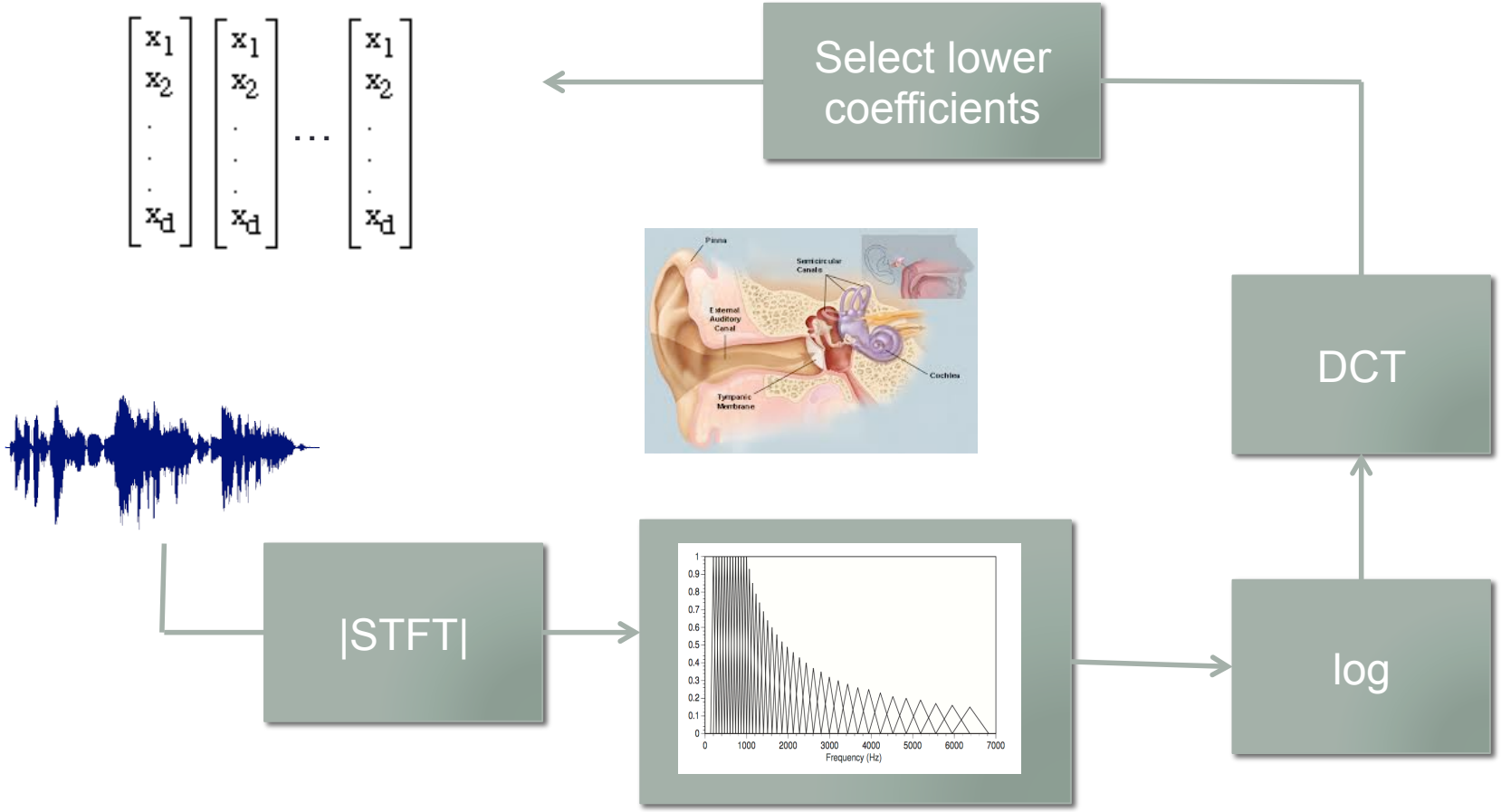
- Traditional machine learning approaches need feature engineering
  - Features are based on human understanding of the phenomena
  - Have some simplifying assumptions
- Human knowledge captures known knowns, and Known unknowns NOT unknown unknowns
- Deep learning has enough parameters and model freedom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

Traditional VS Deep learning

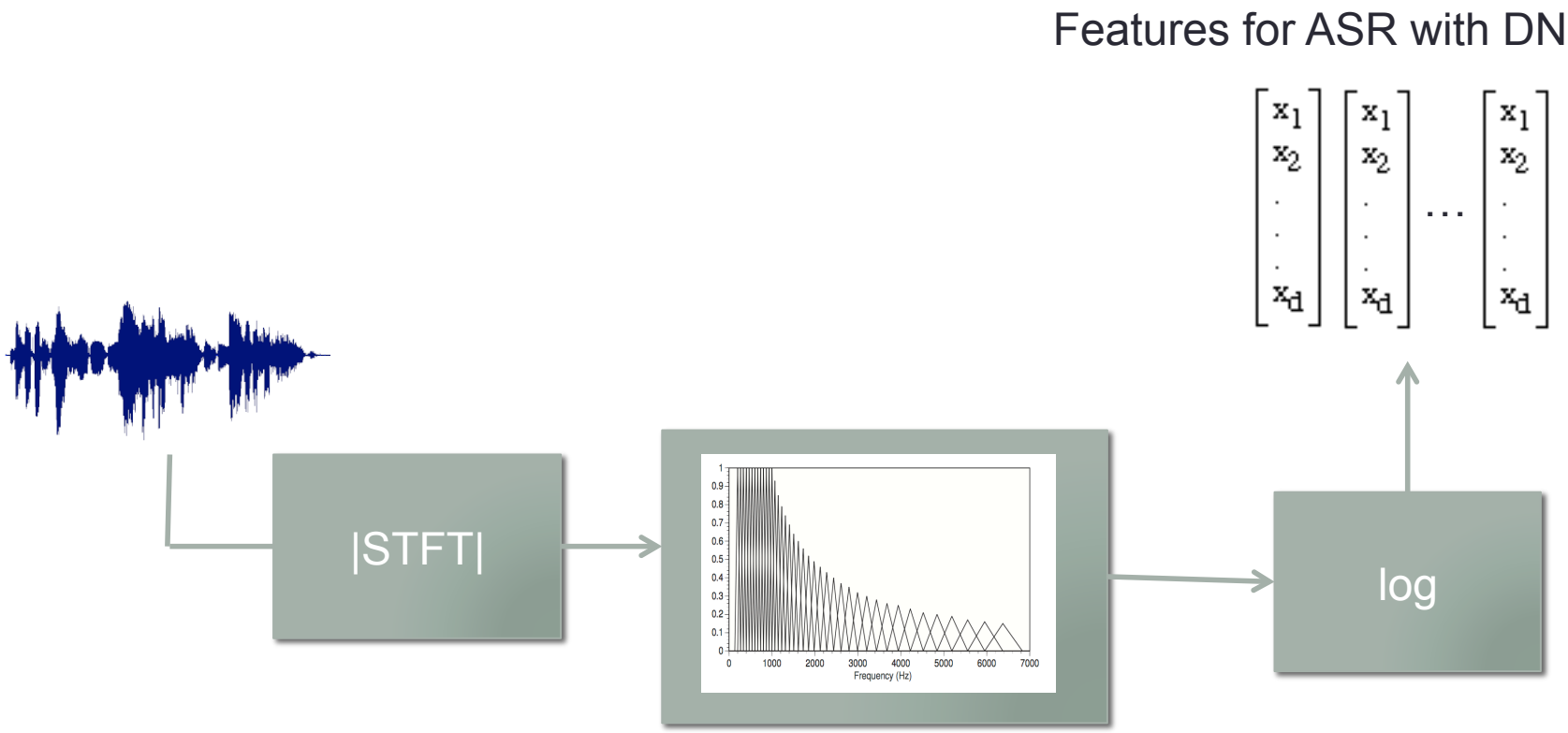


Feature engineering in Speech recognition

Features for traditional ASR



DNN features

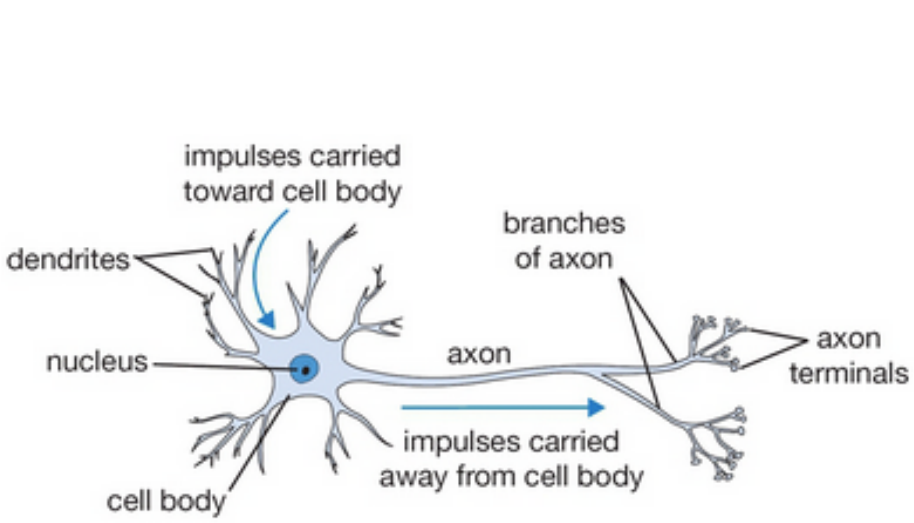


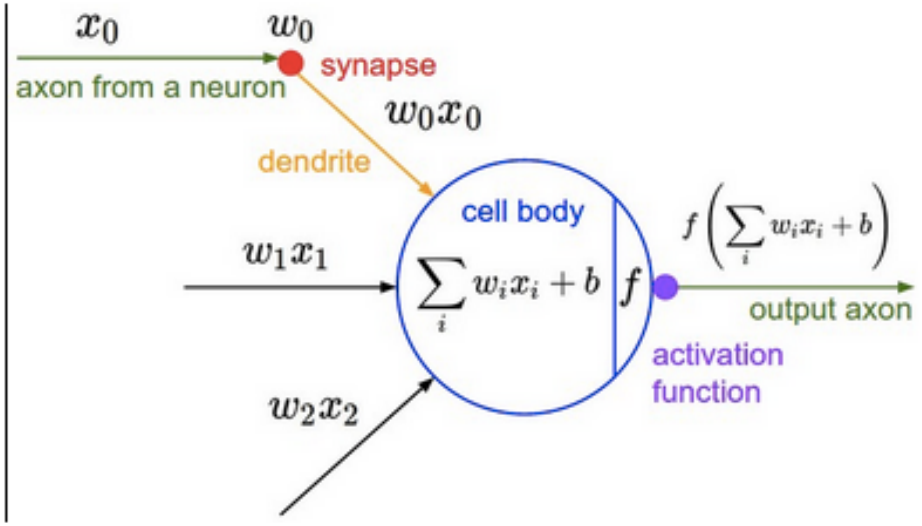
Neural networks

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

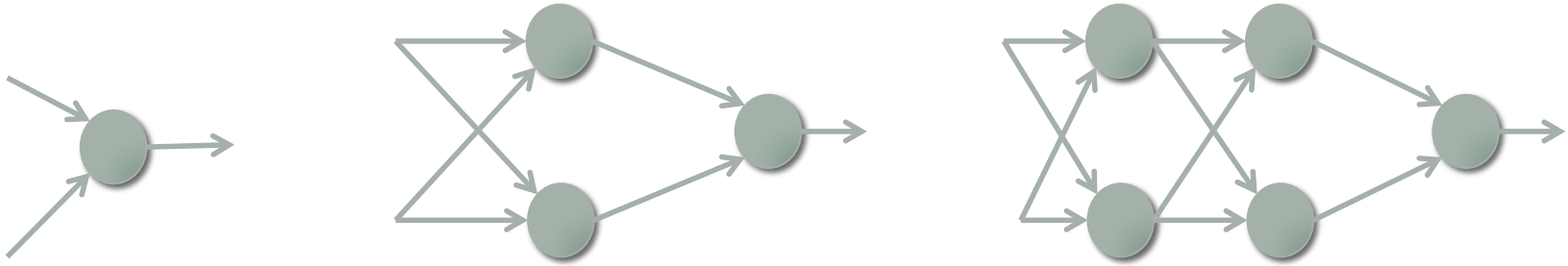


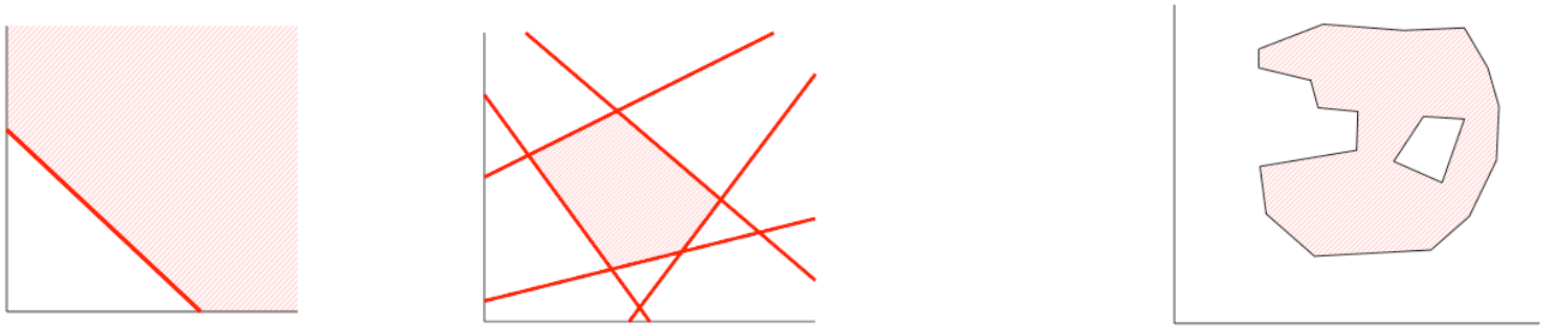


http://cs231n.github.io/neural-networks-1/

Combining neurons

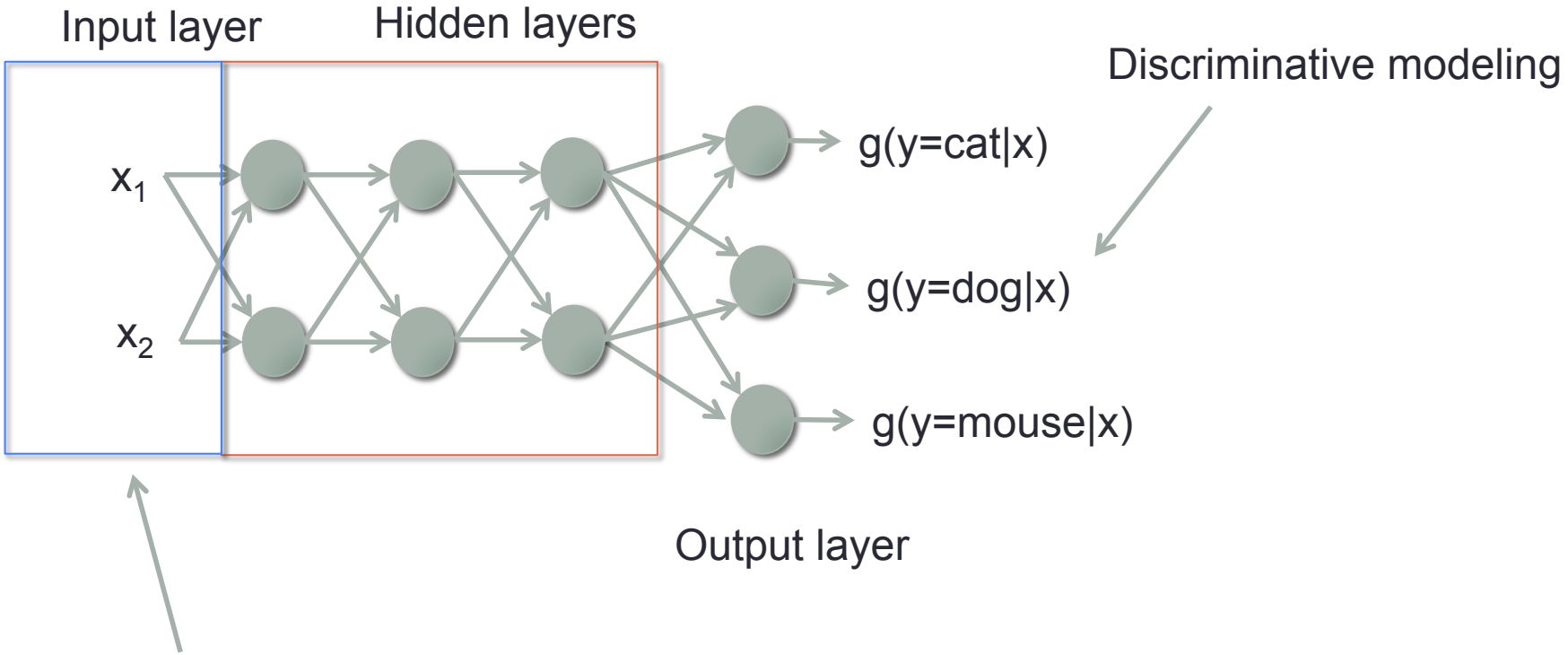
- Each neuron splits the feature space with a hyperplane
- Stacking neuron creates more complicated decision boundaries
- More powerful but prone to overfitting





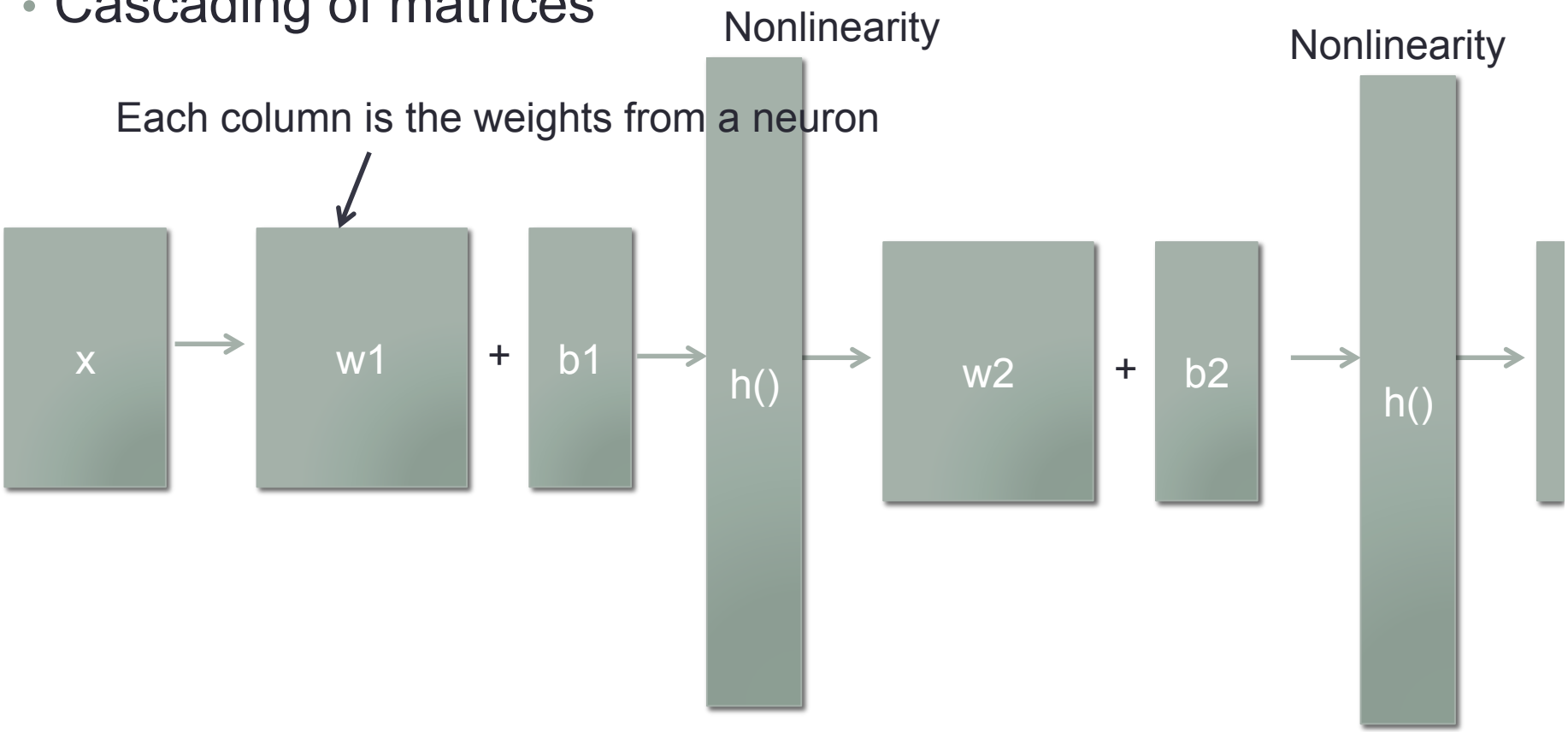
Terminology

Deep in Deep neural networks means many hidden layers



More linear algebra

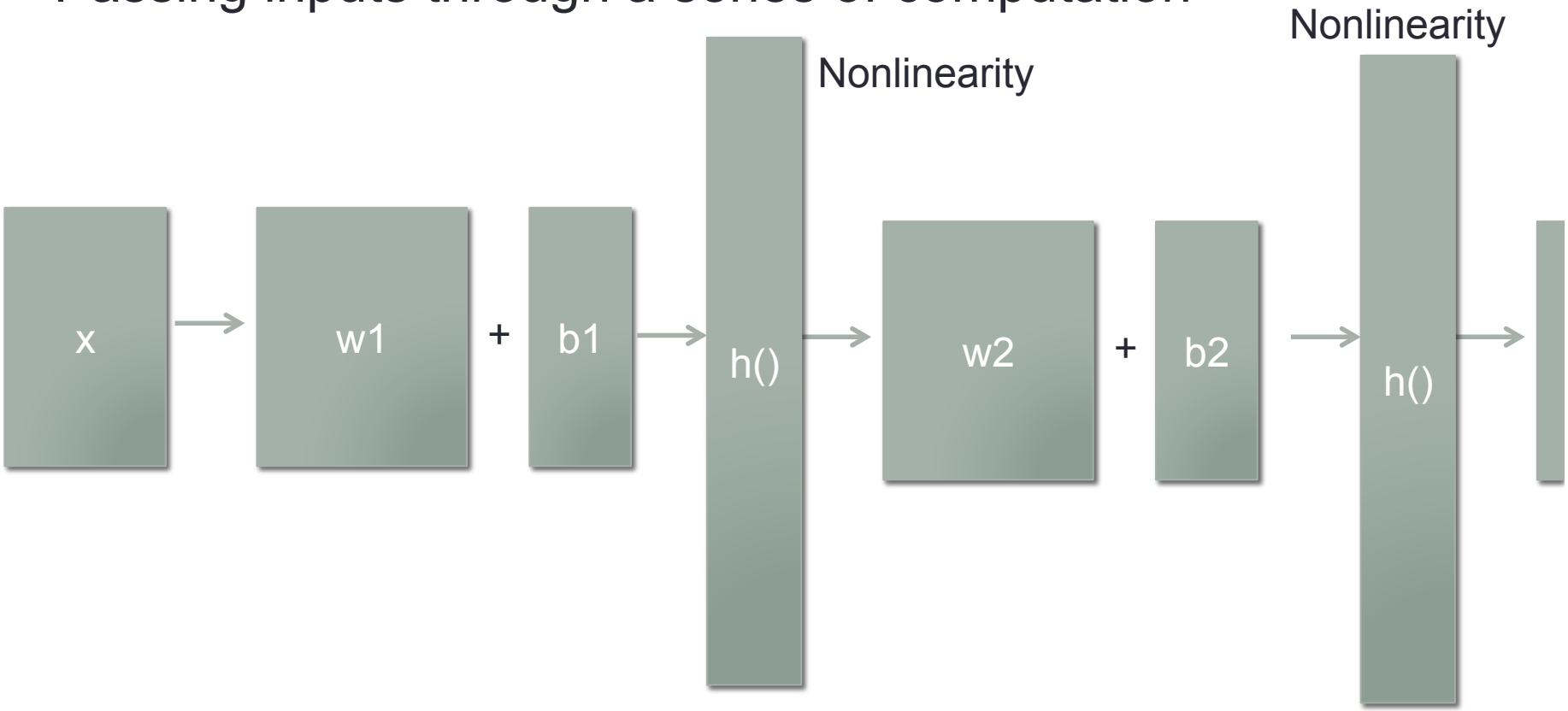
- Cascading of matrices



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

Computation graph

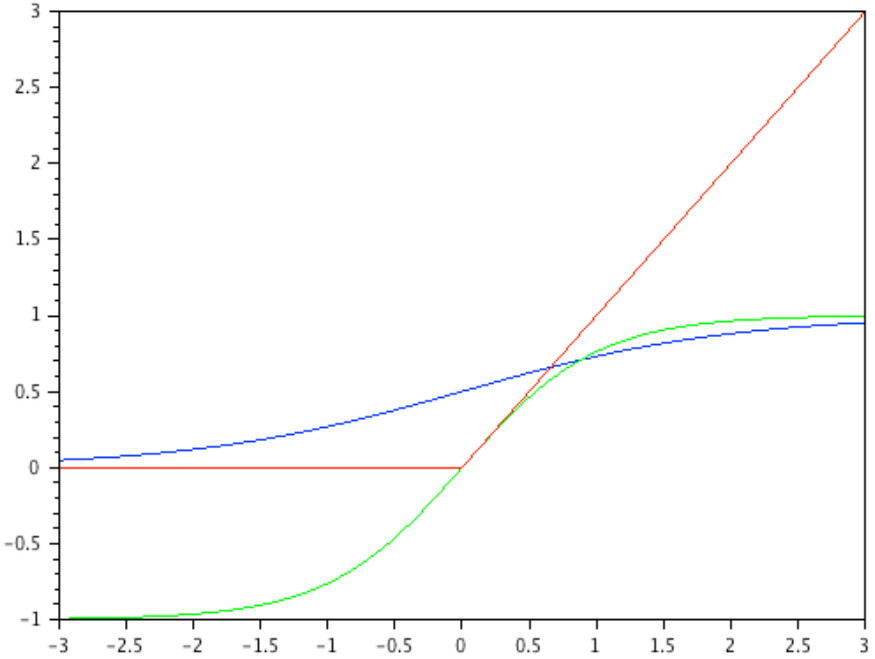
- Passing inputs through a series of computation



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

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)



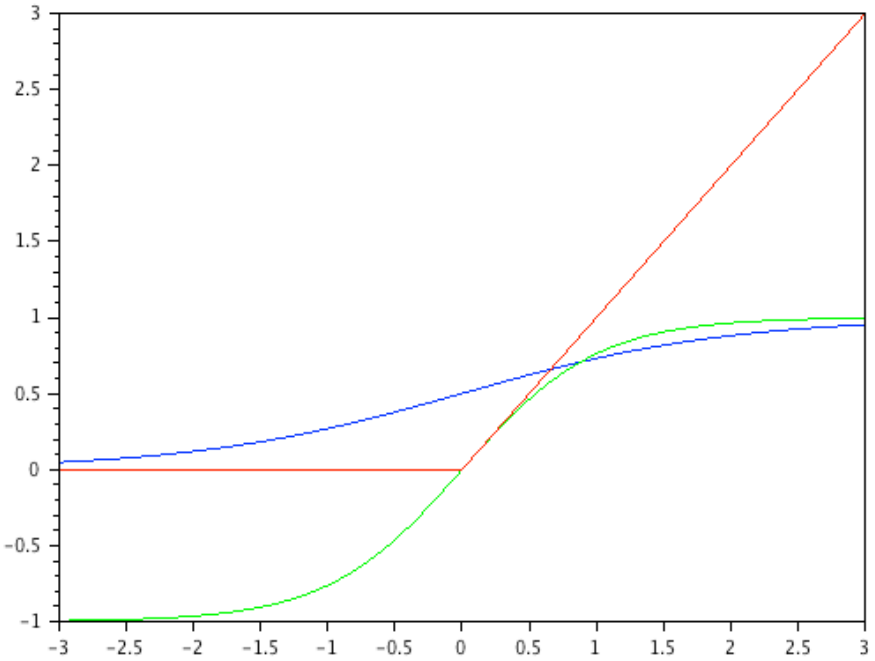
Non-linearity

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

$$tanh(x)$$

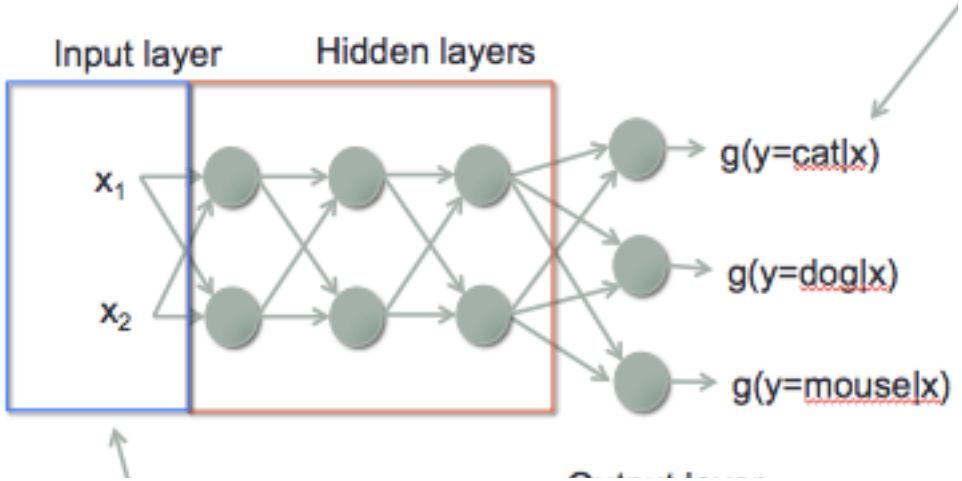
- Rectified Linear Unit (ReLU)

$$max(0, x)$$



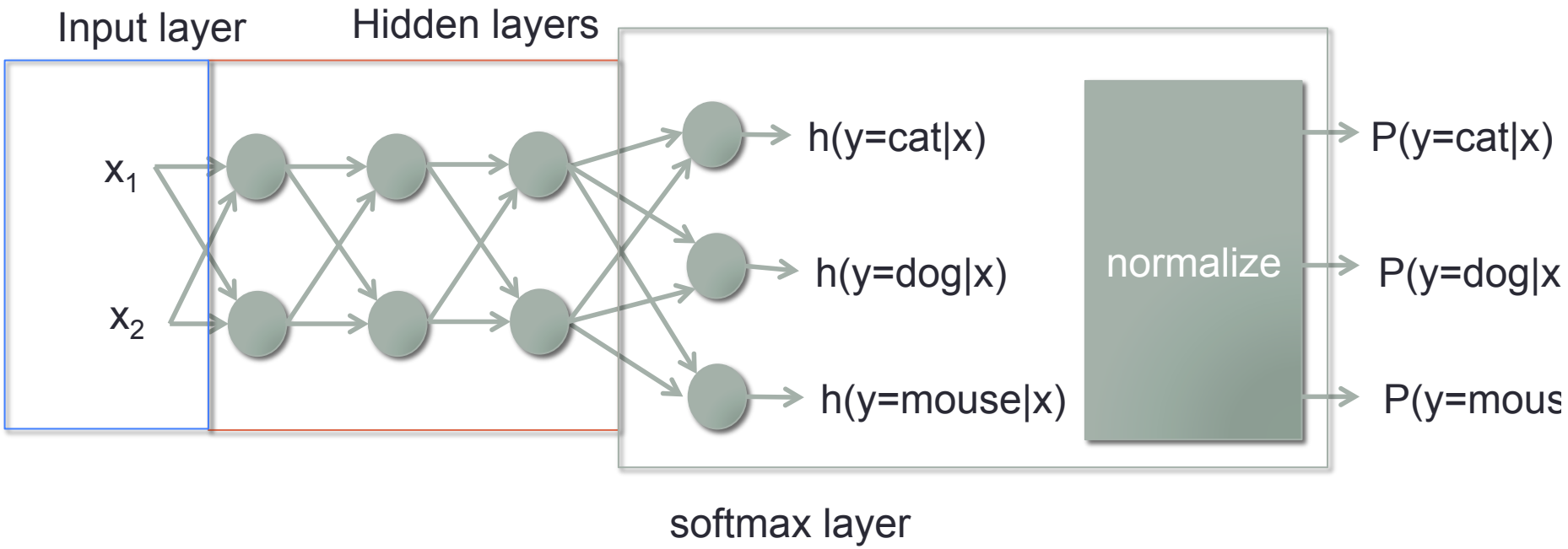
Output layer – Softmax layer

- 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

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



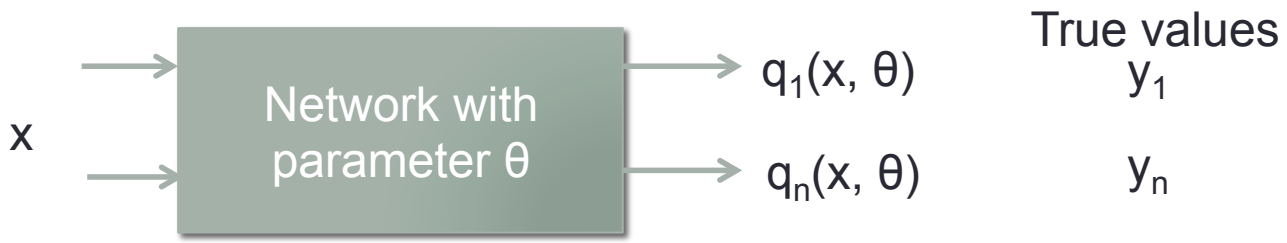
Neural networks

- Fully connected networks
  - Neuron
  - Non-linearity
  - Softmax layer
- DNN training
  - Loss function and regularization
  - SGD and backprop
  - Learning rate and momentum
  - Overfitting – dropout, batchnorm
- t-SNE
- Demos
  - Tensorflow, Gcloud, Keras
- CNN, RNN, LSTM, GRU <- Next class

Objective function (Loss function)

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

- Sum of square errors



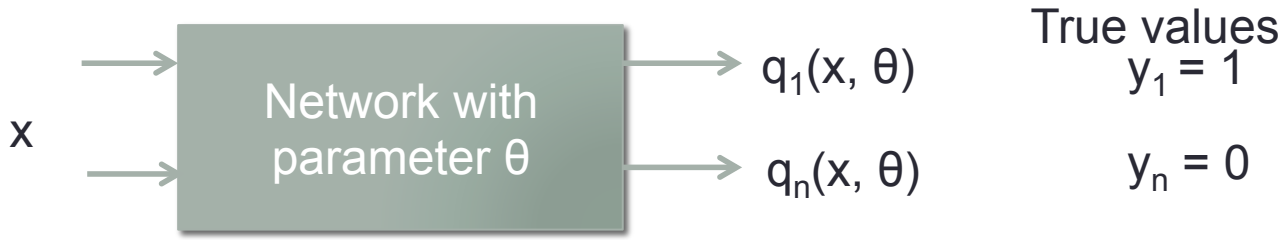
Cross entropy loss

- 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

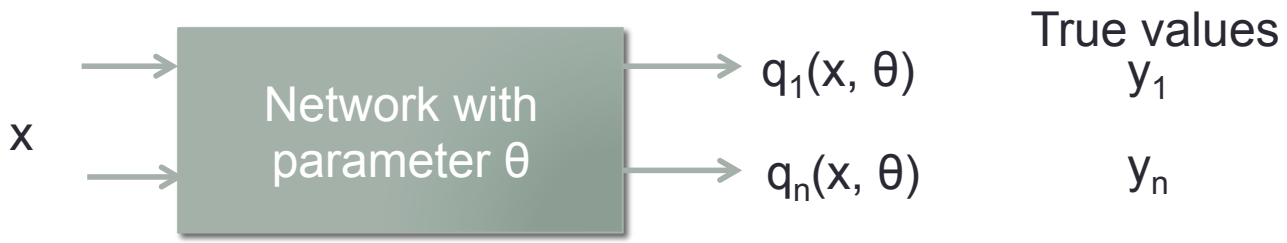


Sum of square errors

- 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



Average Loss per data sample

- Loss is usually computed for each mini-batch

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

- Bigger loss if bigger mini-batch
  - Higher gradient (change mini-batch size and you have to change the learning rate)

- Use average loss per data sample instead
  - L = L/N

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).
  - Try to use every part of the model.
- 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 + $0.5\ C\Sigma|w|$

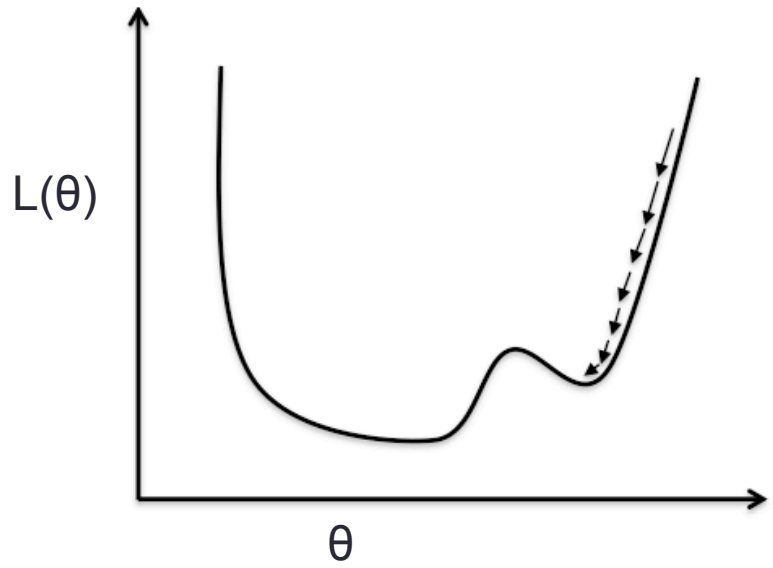
- we sum the absolute weights of the whole model
- 0.5 is for prettiness when we take derivative
- C is a hyperparameter weighting the regularization term

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)

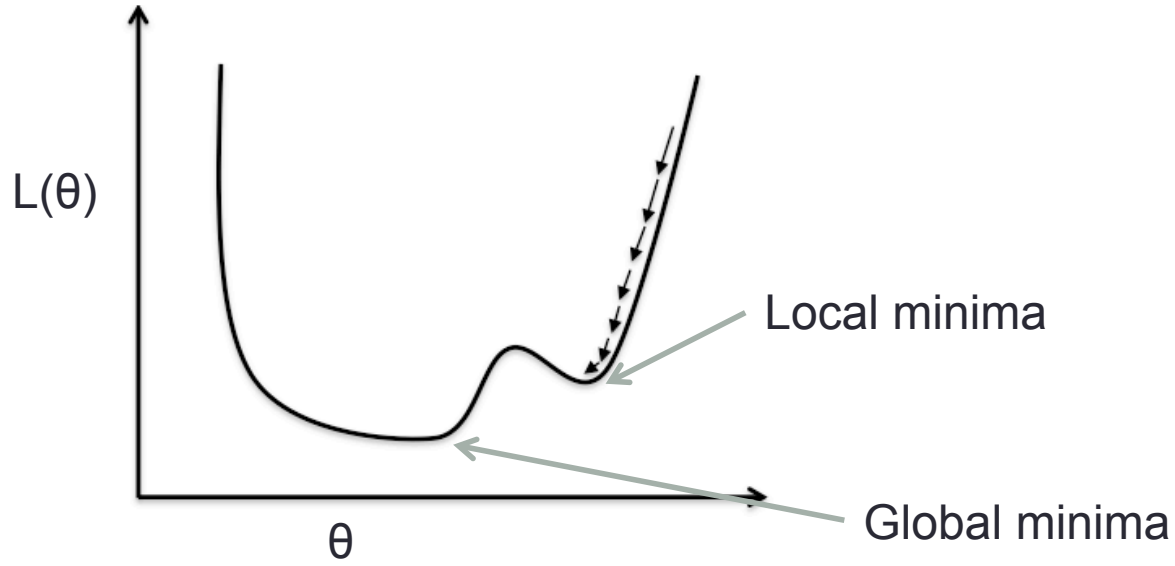
Minimization using gradient descent

- 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



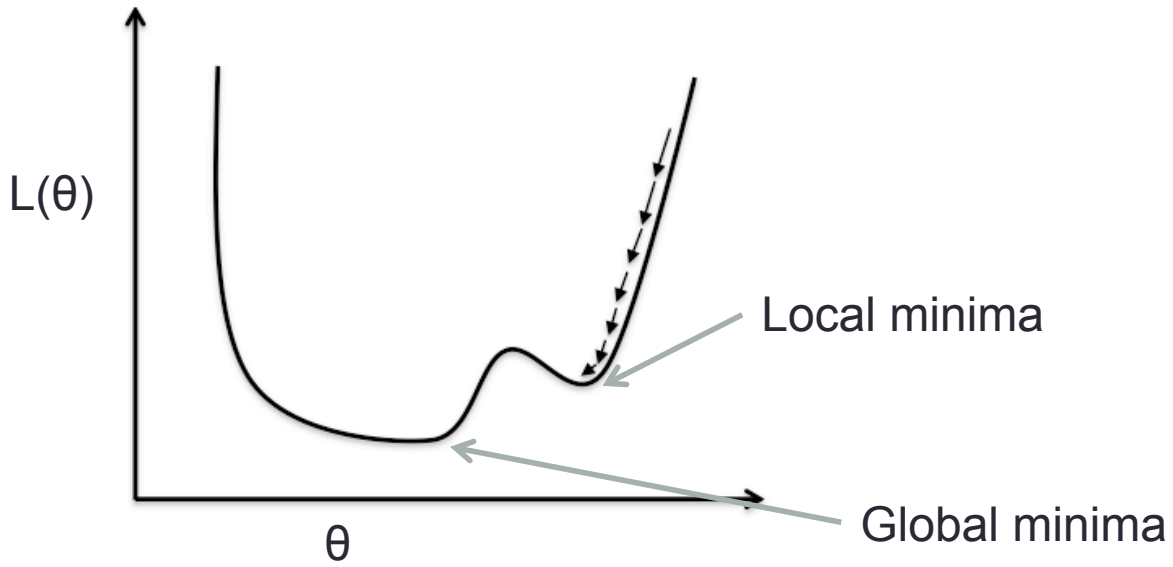
Deep vs Shallow

- The loss function of neural network is non-convex (and non-concave)
  - Local minimas can be avoided with convexity
    - Linear regression, SVM are convex optimization
  - Convexity gives easier training
    - Does not imply anything about the generalization of the model
    - The loss is optimized by the training set



Deep vs Shallow

- If deep, most local minimas are the global minima!
  - Always a way to lower the loss in the network with millions of paramters
  - Enough parameters to remember every training examples
  - Does not imply anything about generalization



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

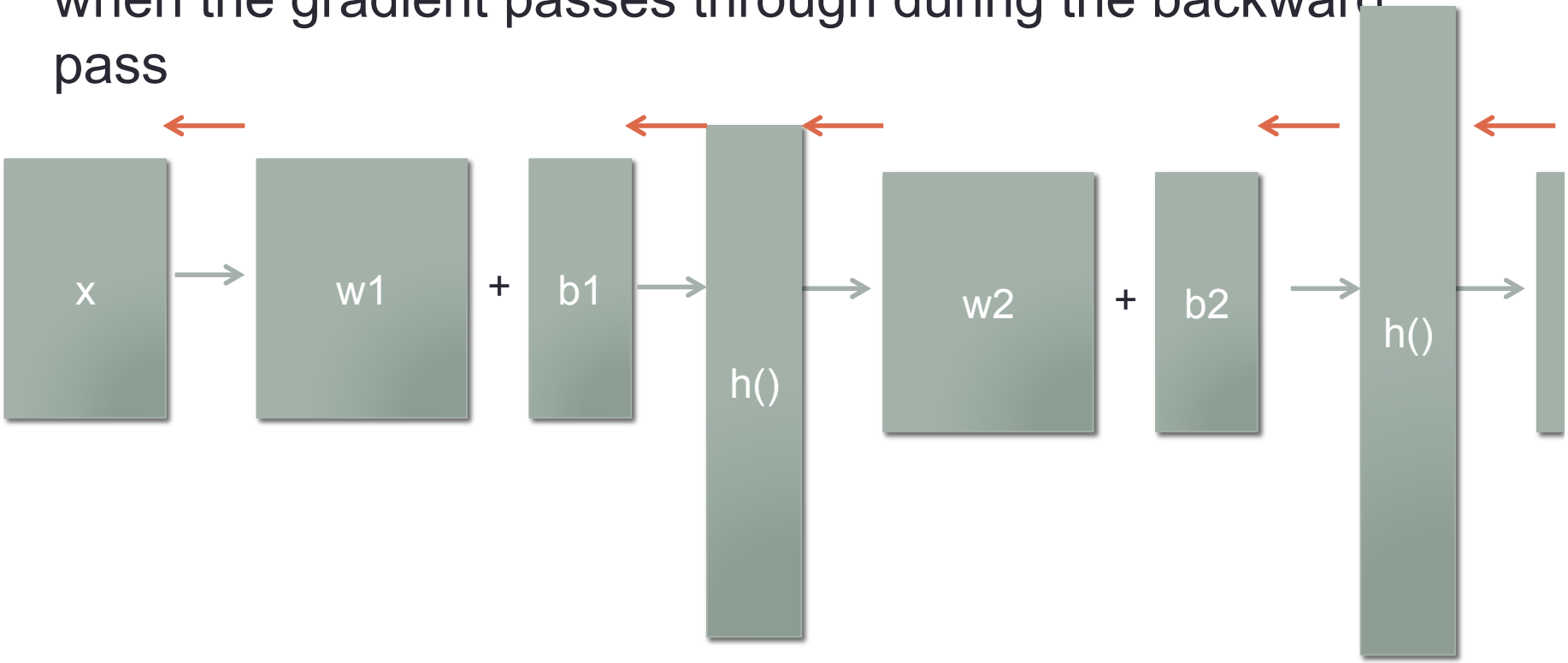
Examples to read

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

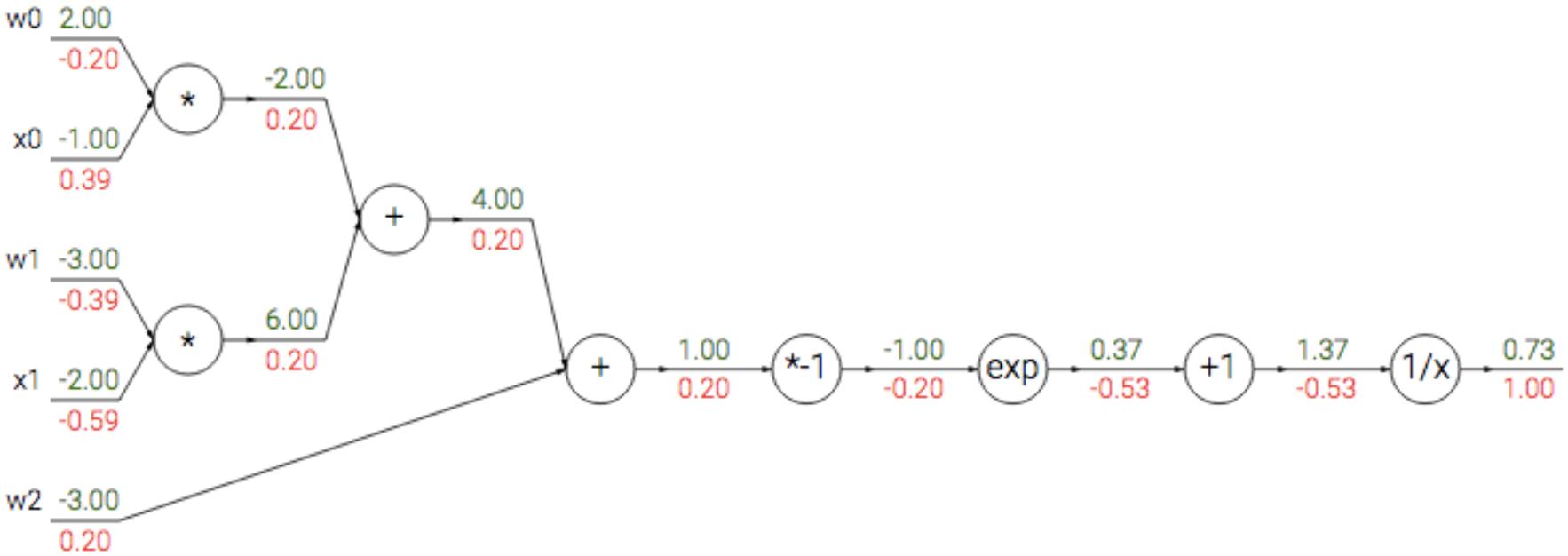


This lets us to build any neural networks without having to redo all the derivation as long as we define a forward and backward computation for the block.

Gradient flow in code

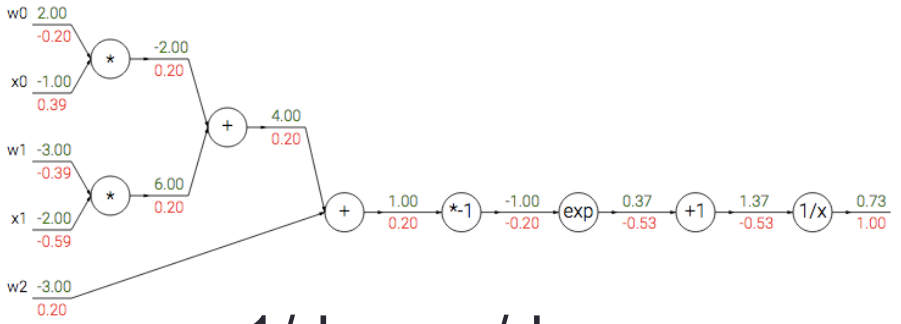
- Let's find the gradient of $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

Computation graph



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

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

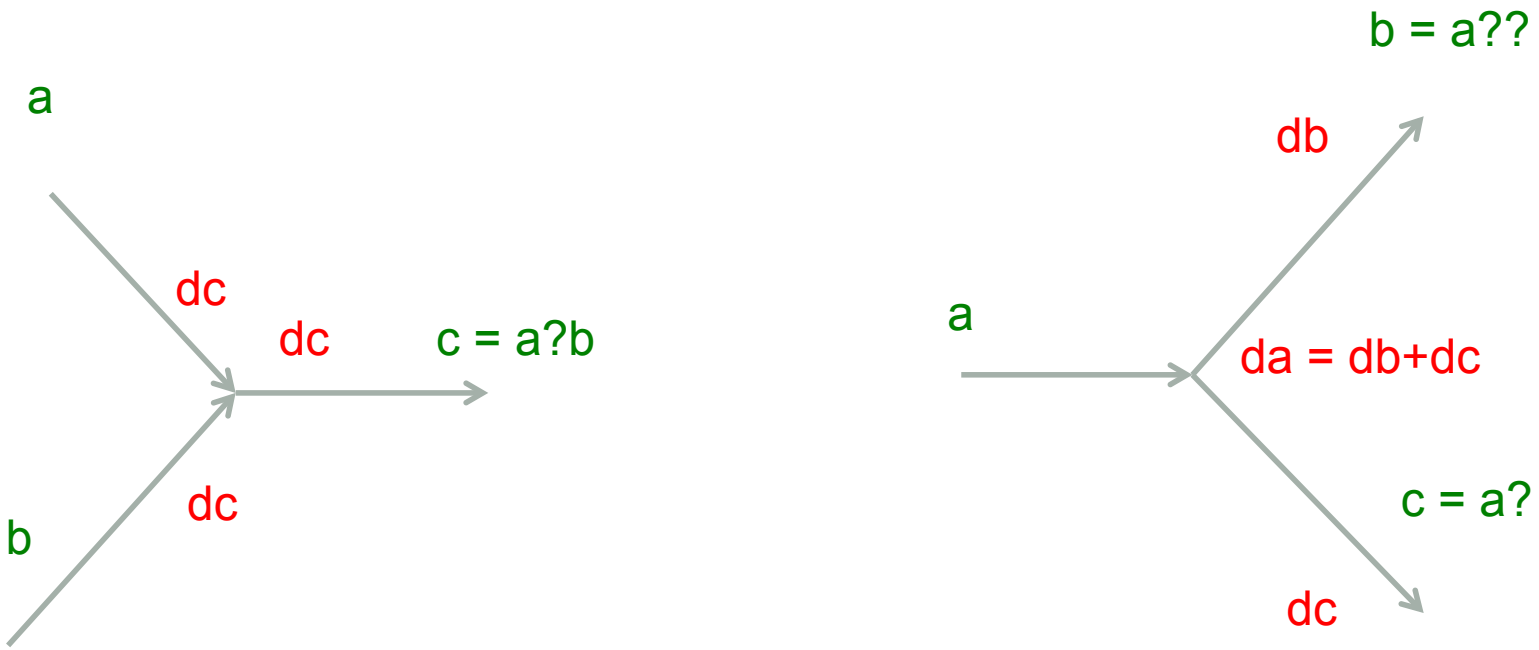


- w = [0, -3, -3]
- x = [-1, -2]
- t0 = w[0]*x[0]
- t1 = w[1]*x[1]
- t01 = t0+t1
- t012 = t01+w[2]
- nt = -t012
- e = exp(nt)
- denom = e+1
- f = 1/denom

- 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

Gradient flow at forks



Forward and backward pass acts differently at forks

Initialization

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

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

  - For ReLUs

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

  - Or use a pre-trained network as initialization

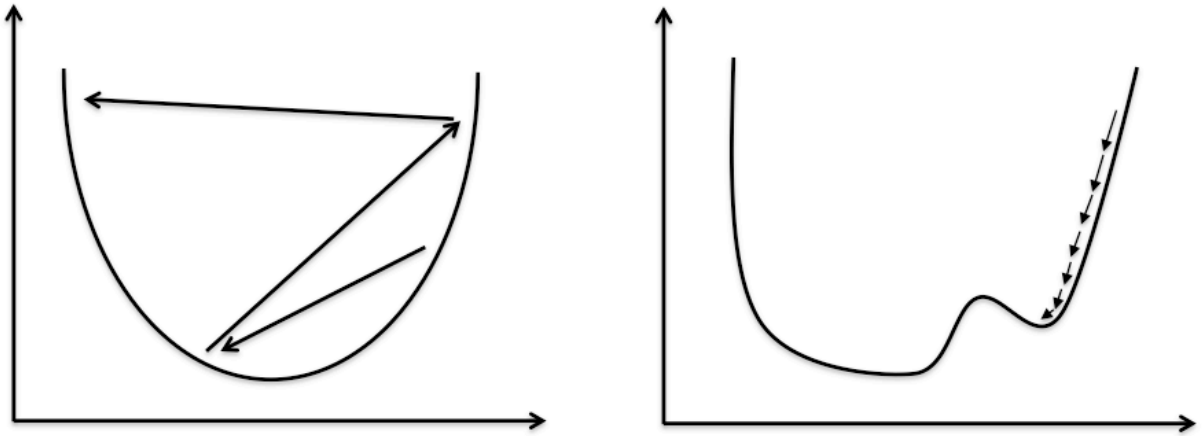
X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. 2010

Initialization

- Bias
  - All Zeros is fine

Learning rate

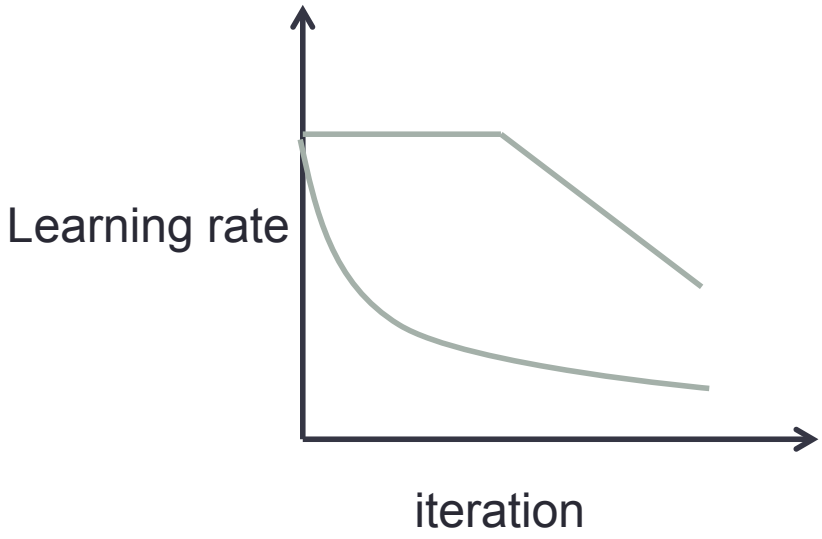
- How fast to go along the gradient direction is controlled by the learning rate
- Too large models diverge
- Too small the model get stuck in local minimas and takes too long to train



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)

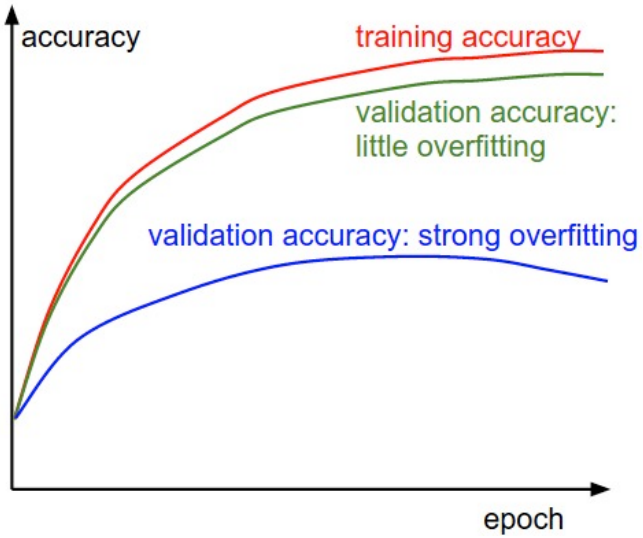


Learning rate strategies (annealing)

- 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

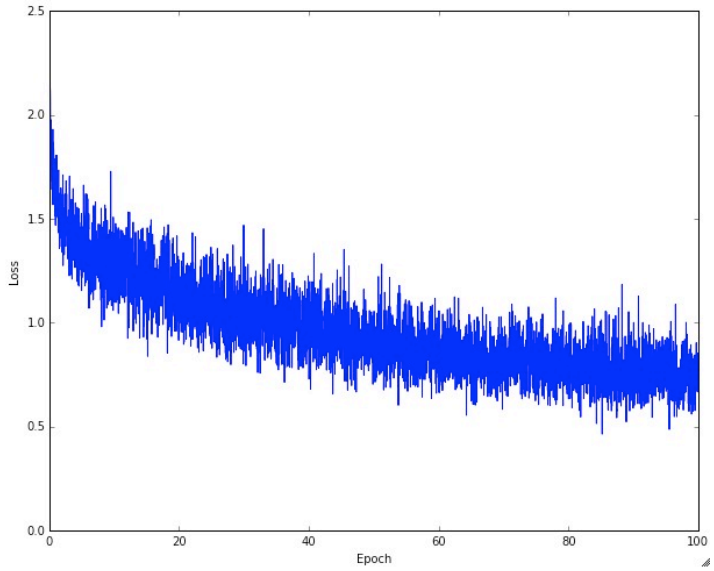
- You can keep doing back propagation forever!
- The training loss will always go down
- But it overfits
- Need to monitor performance on a held out set
- Stop or decrease learning rate when overfit happens



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.

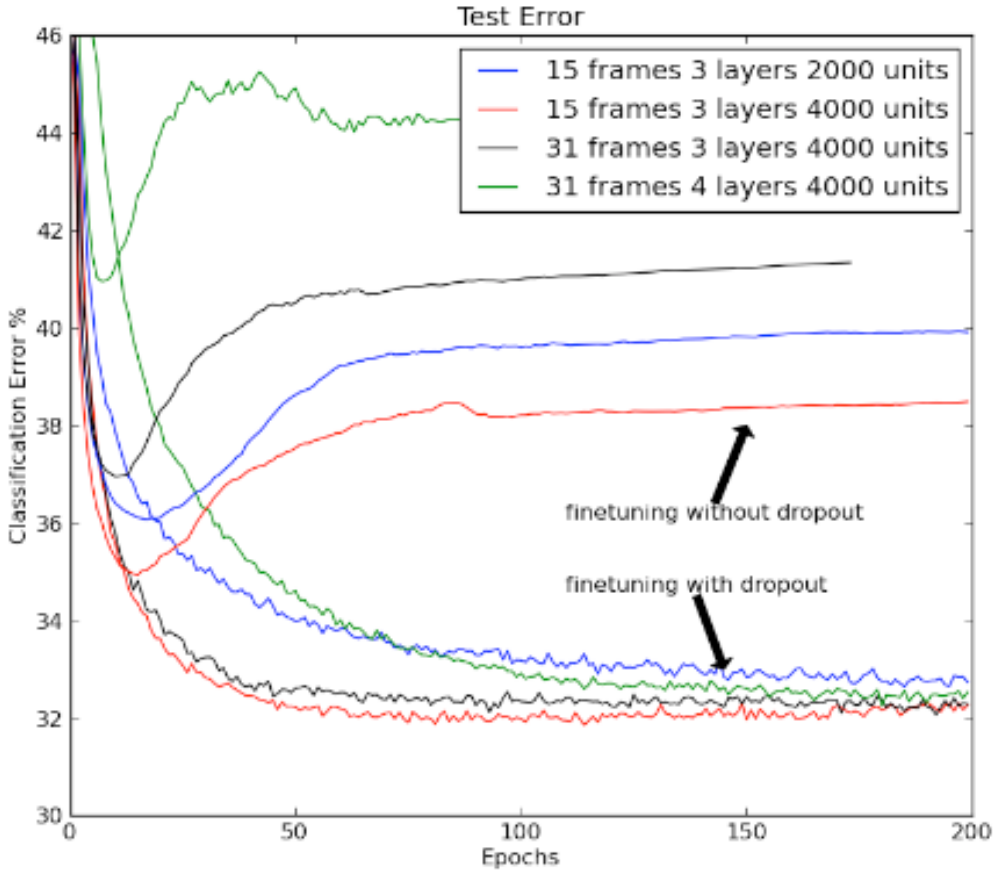
Reducing overfitting - dropout

- 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

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

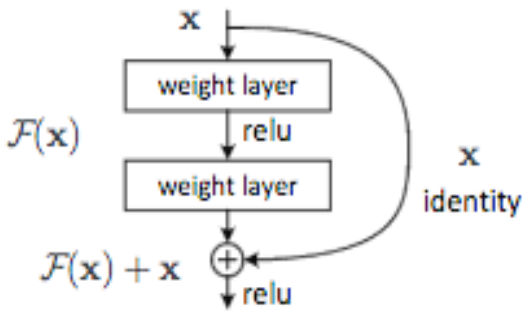
https://arxiv.org/abs/1502.03167

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 and regularization
  - SGD and backprop
  - Learning rate
  - Overfitting – dropout, batchnorm
- CNN, RNN, LSTM, GRU

Convolutional Neural Networks (CNNs)

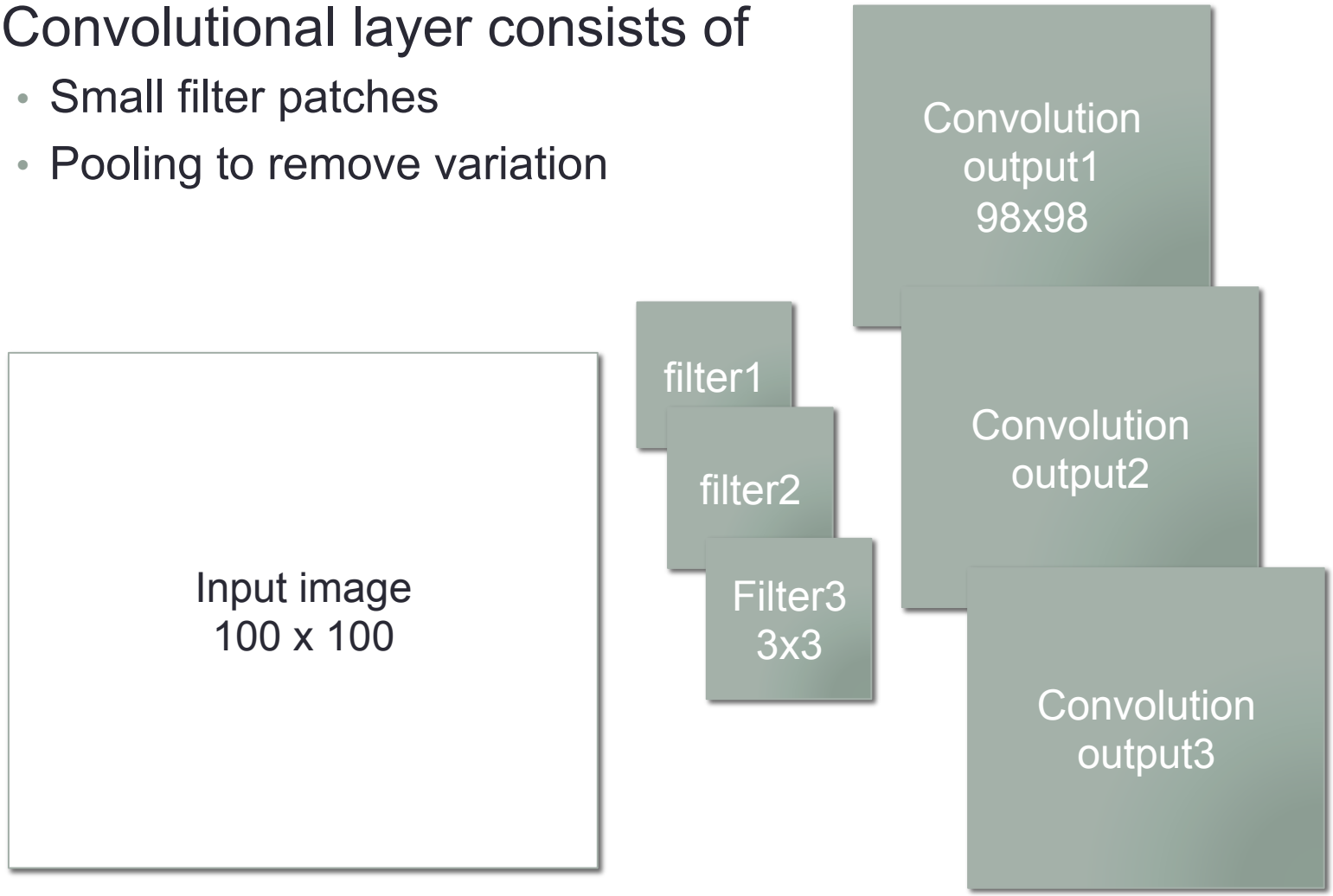
- Consider an image of a cat. DNNs need different neurons to learn every possible location a cat can be

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

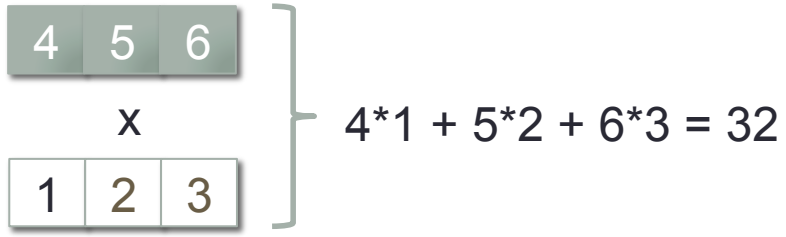
- 2 parts: convolutional layer and pooling layer

Convolutional filters

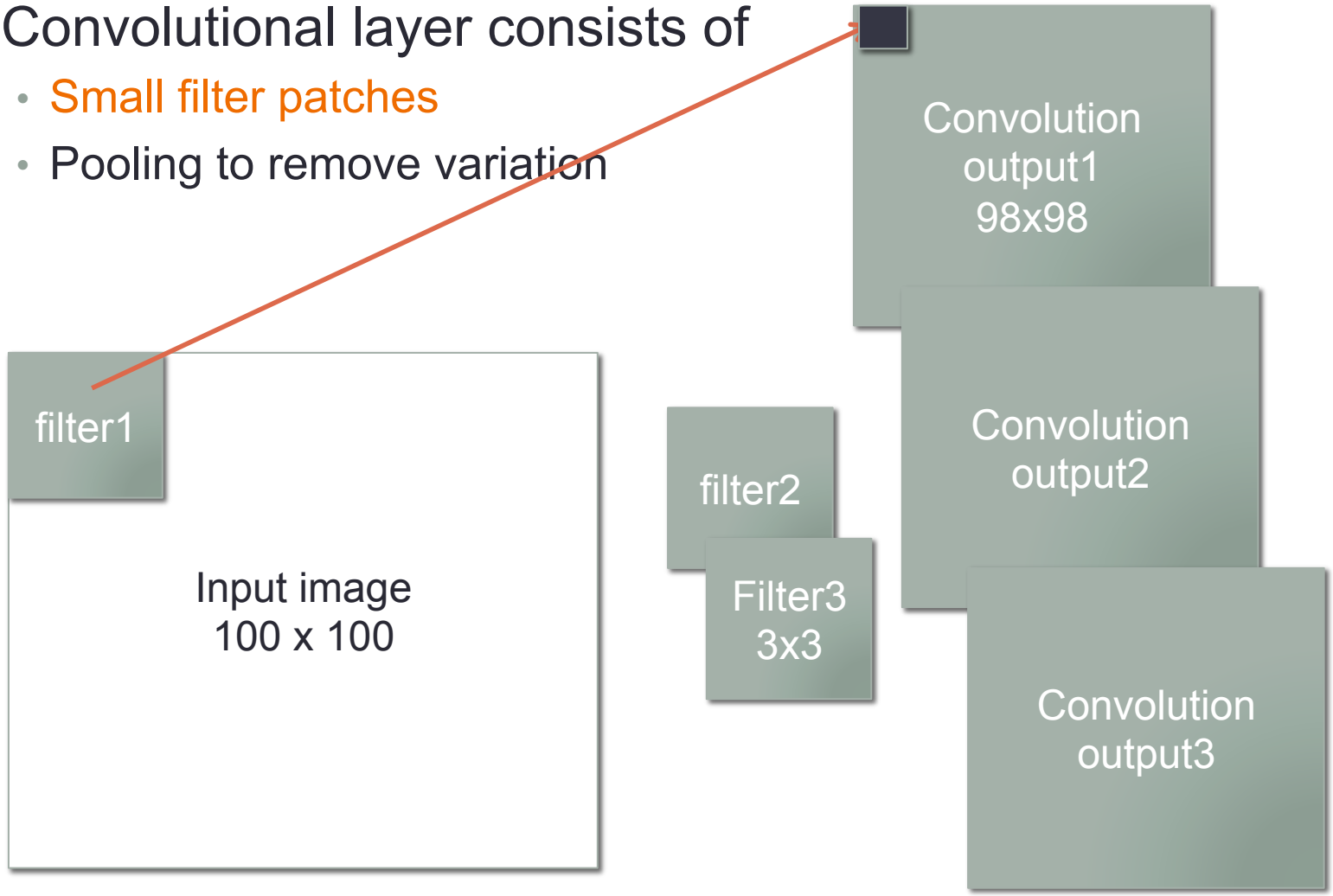
- Convolutional layer consists of
  - Small filter patches
  - Pooling to remove variation



Convolutional filters

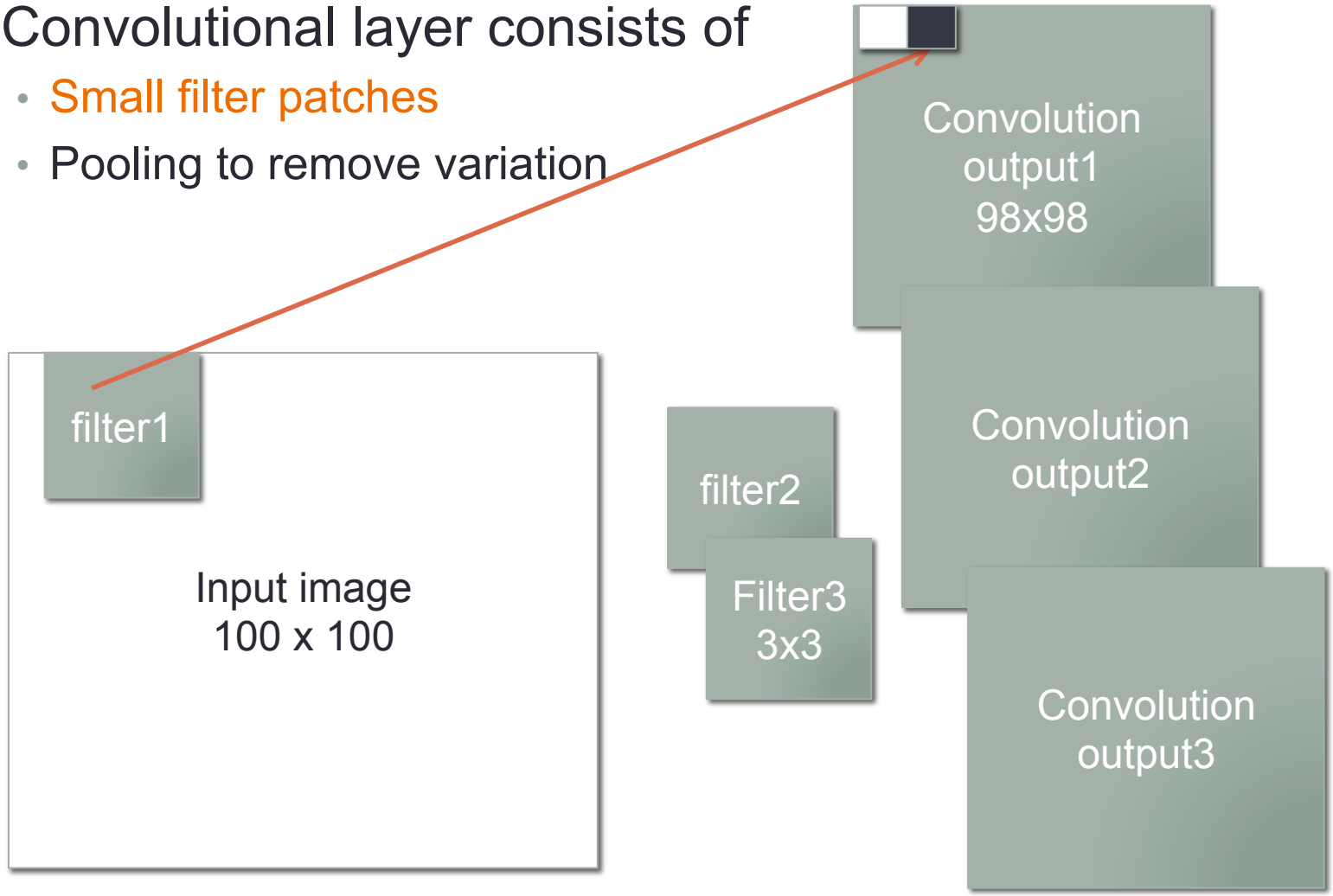


- Convolutional layer consists of
  - Small filter patches
  - Pooling to remove variation



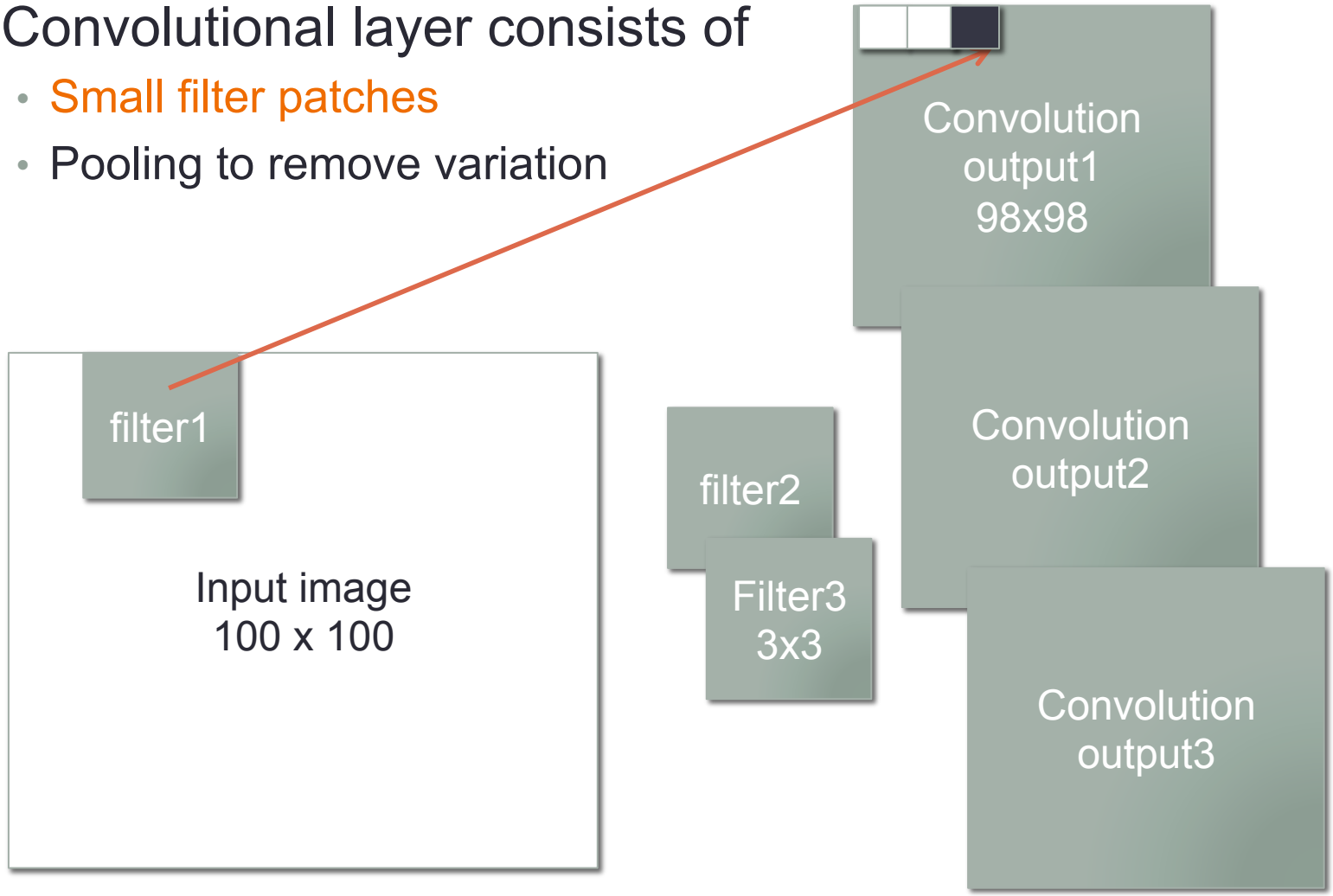
Convolutional filters

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



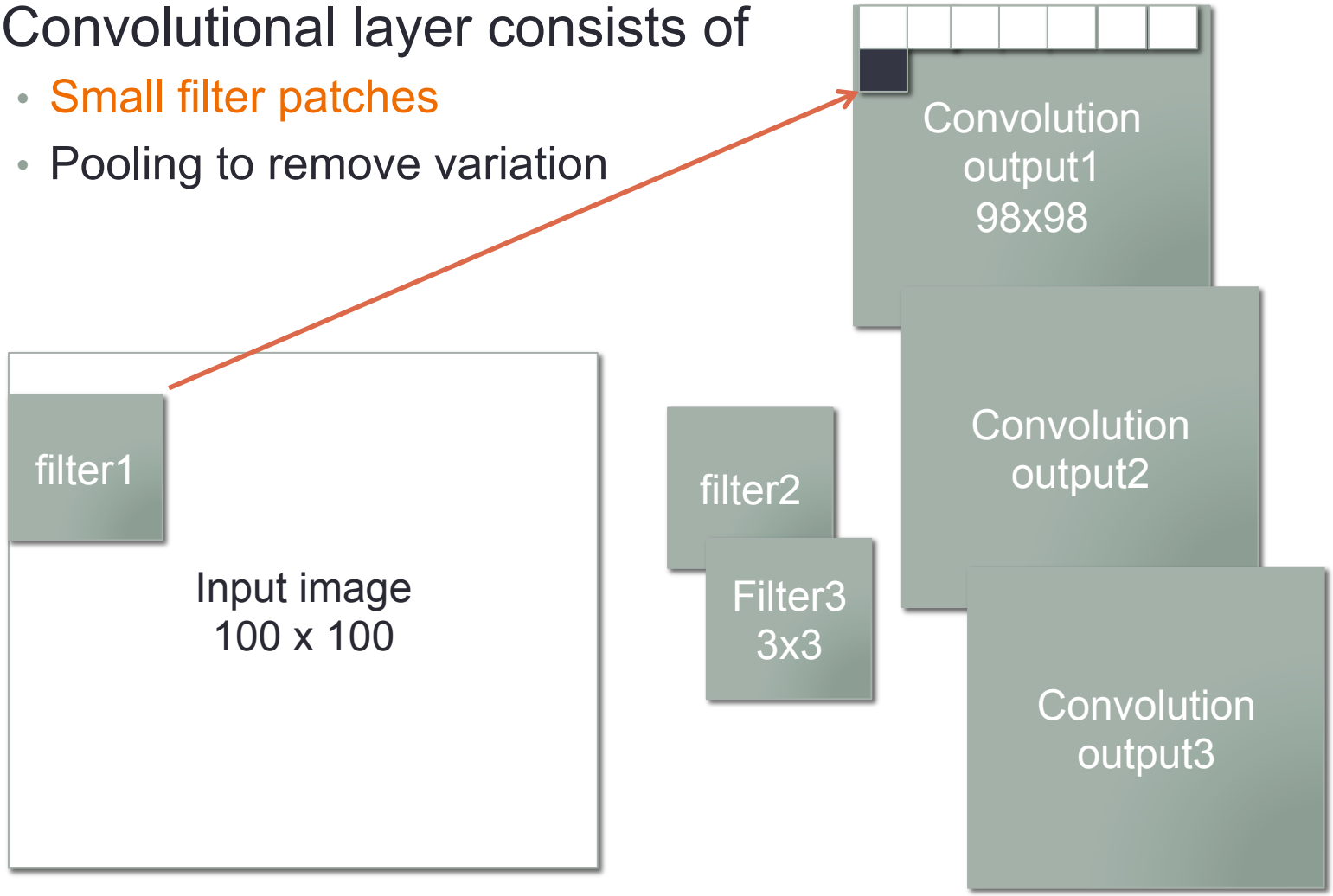
Convolutional filters

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



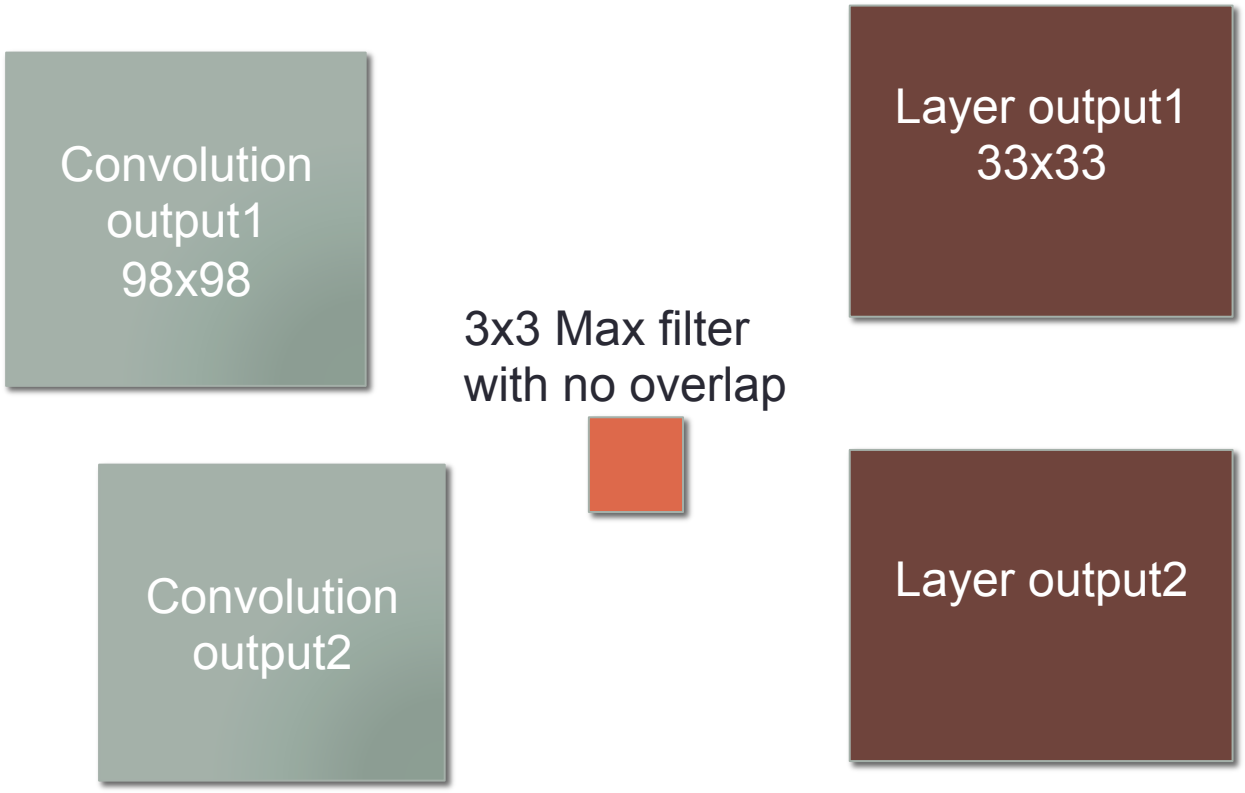
Convolutional filters

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



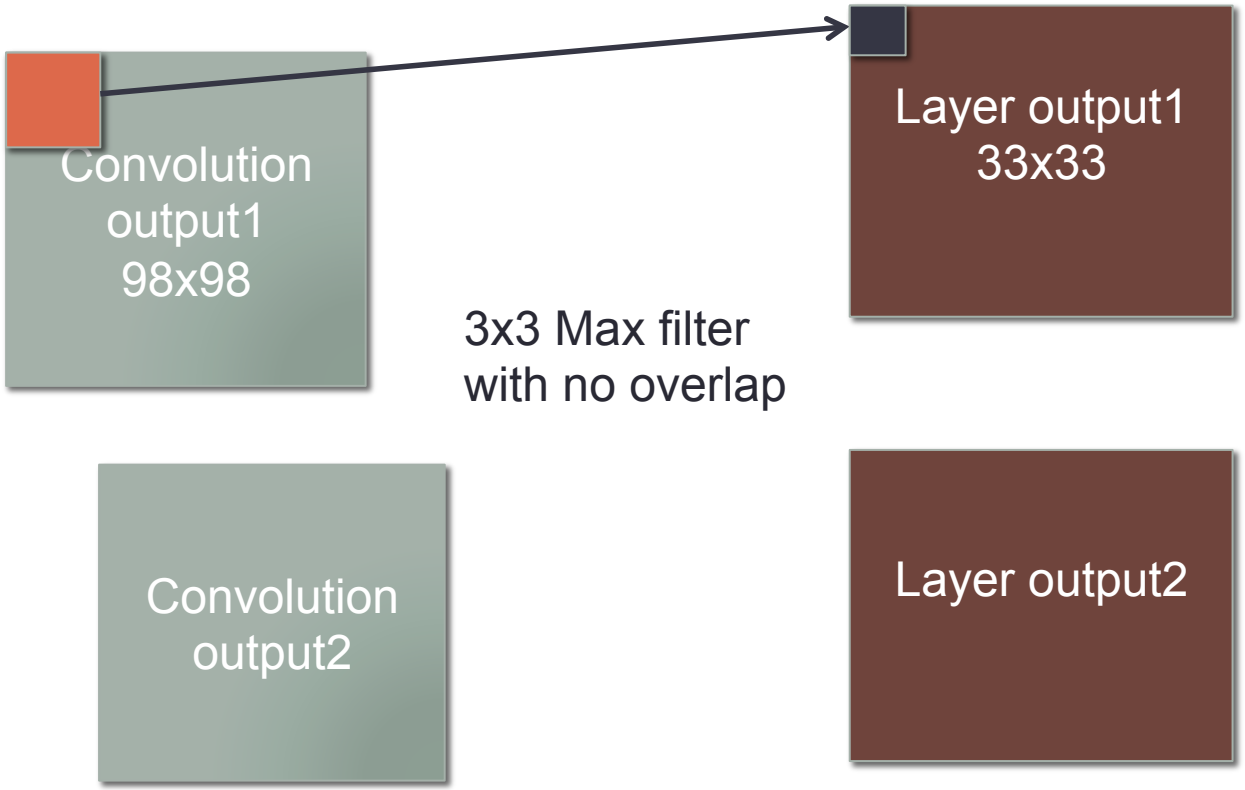
Pooling/subsampling

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



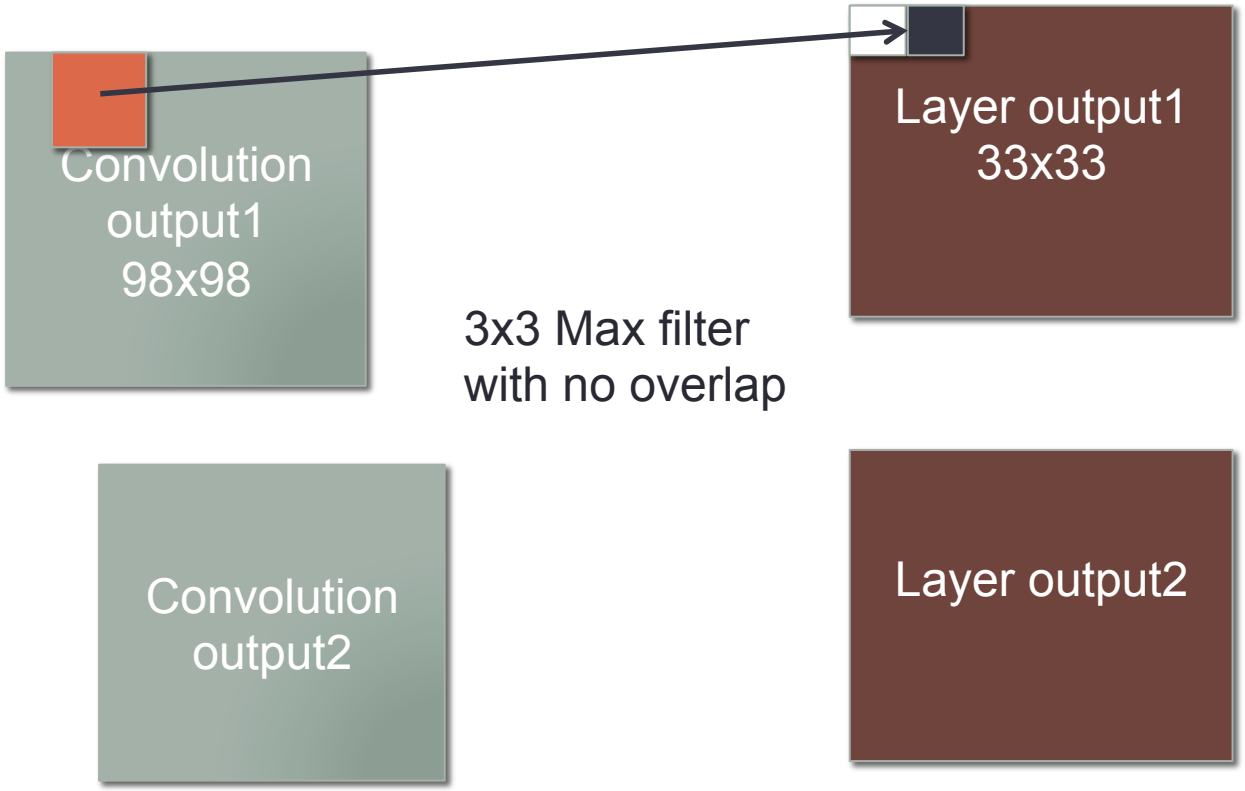
Pooling/subsampling

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



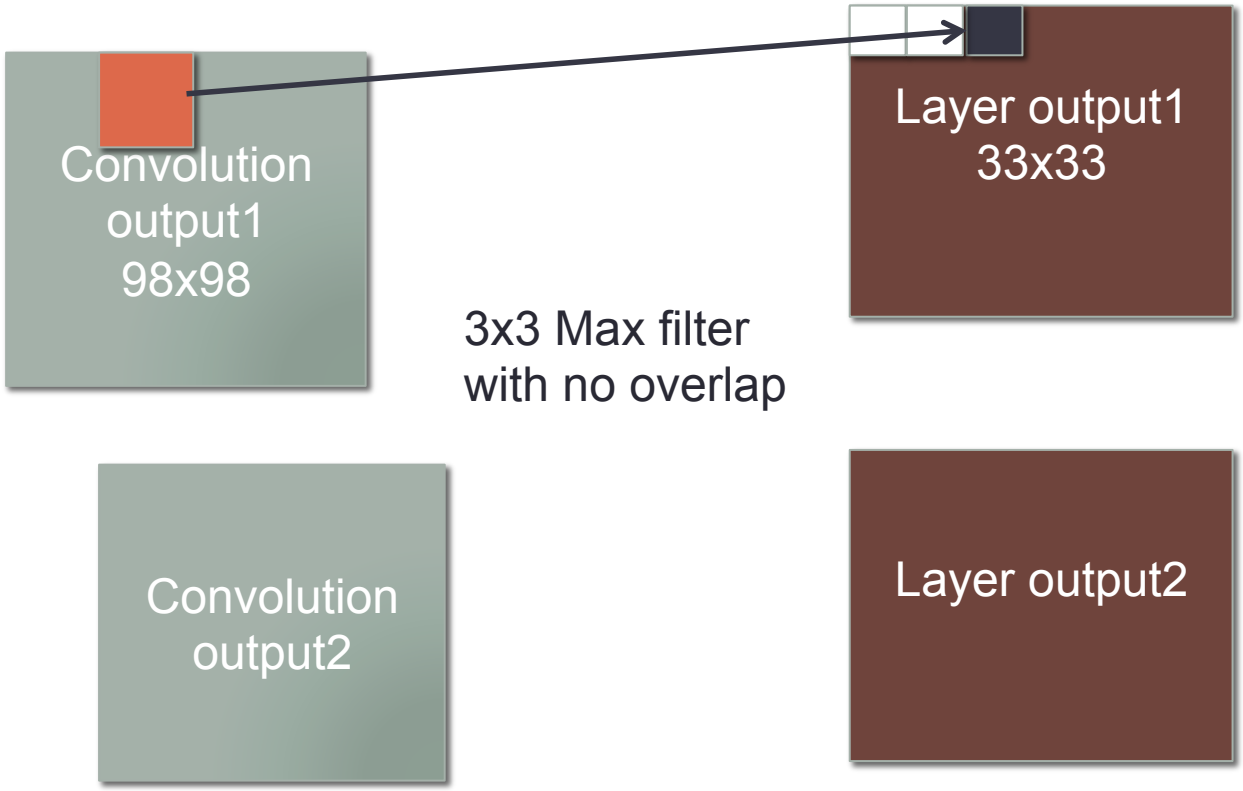
Pooling/subsampling

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



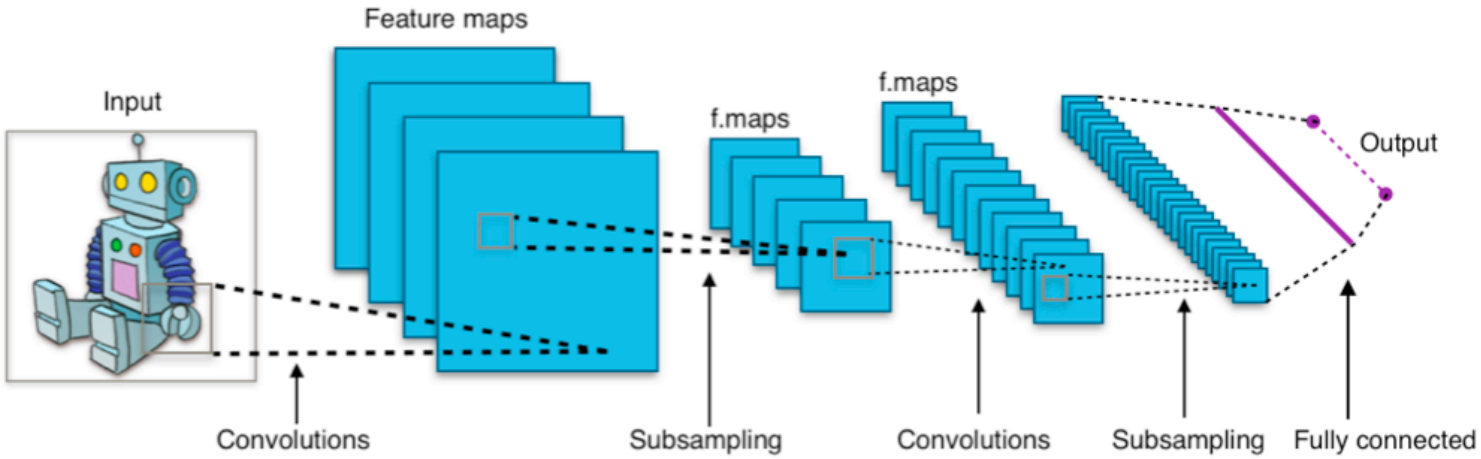
Pooling/subsampling

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



CNN overview

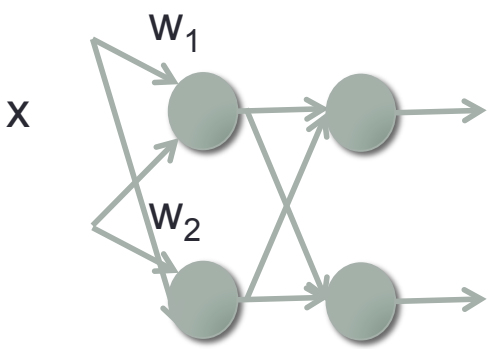
- Filter size, number of filters, filter shifts, and pooling rate are all parameters
- Usually followed by a fully connected network at the end
  - CNN is good at learning low level features
  - DNN combines the features into high level features and classify



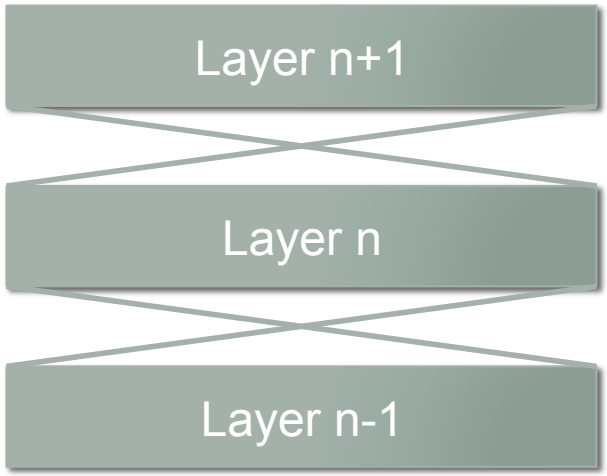
https://en.wikipedia.org/wiki/Convolutional_neural_network#/media/File:Typical_cnn.png

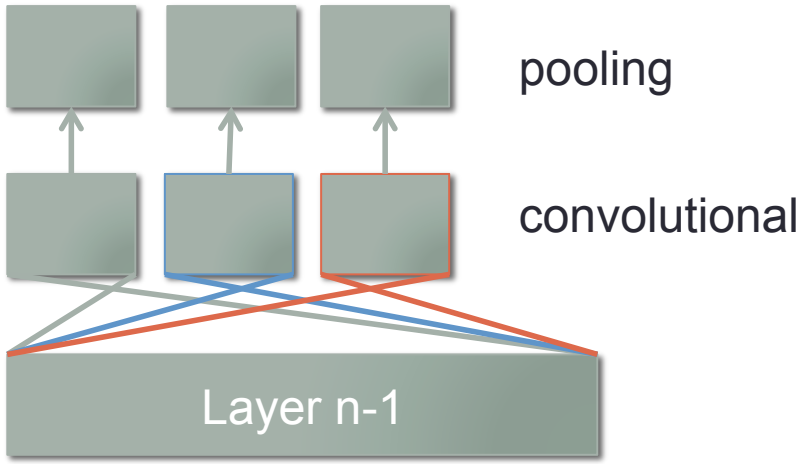
Parameter sharing in convolution neural networks

- $W^T x$



- 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



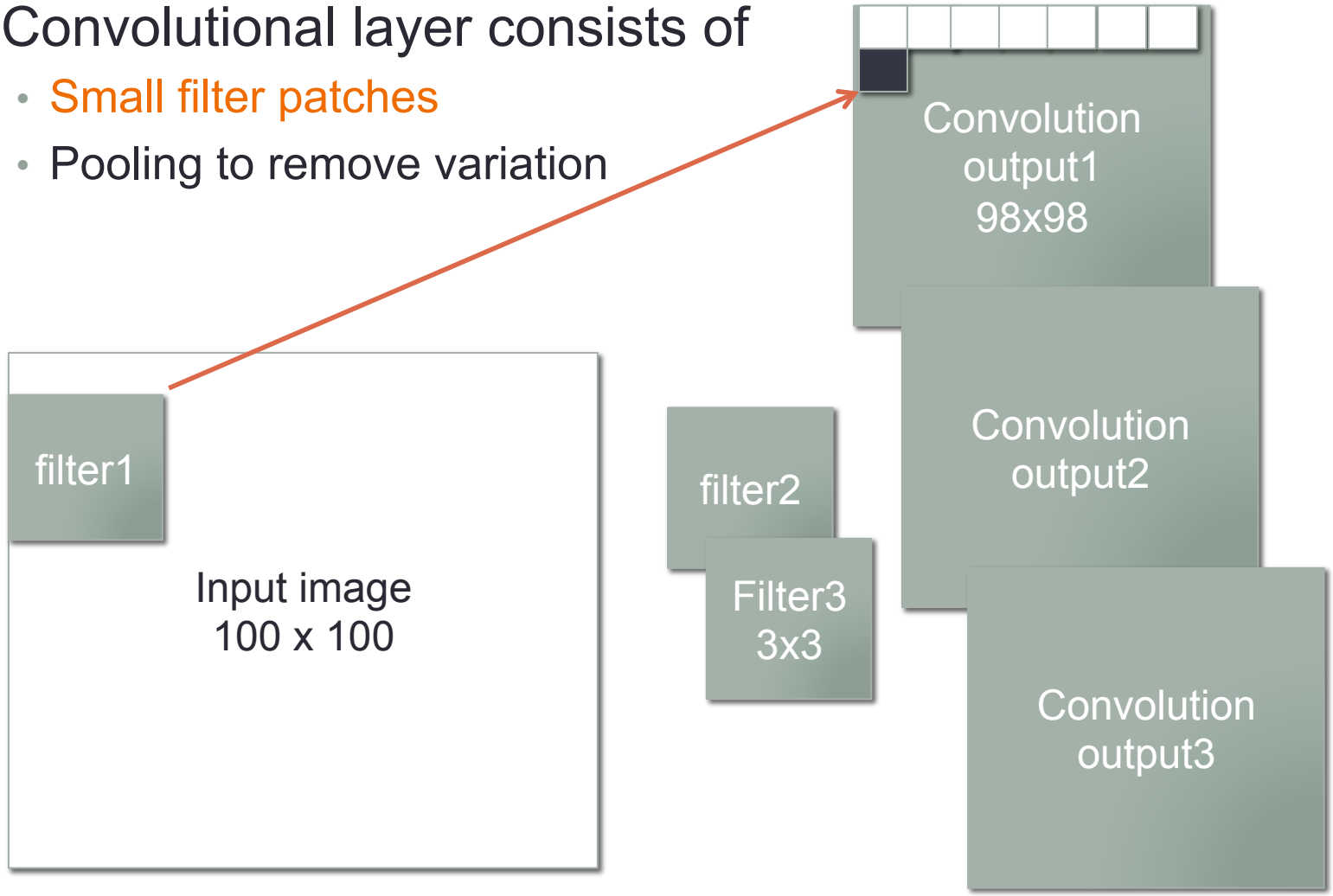


Pooling/subsampling

- Max filter -> Maxout
  - Backward pass?
    - Gradient pass through the maximum location, 0 otherwise
- P-norm filter
- Fully connected layer – (1x1 convolutions)
- Recently, people care less about the meaning of pooling as way to introduce a shift invariance, but more as a dimension reduction (since conv layers usually has a higher dimension than the input)

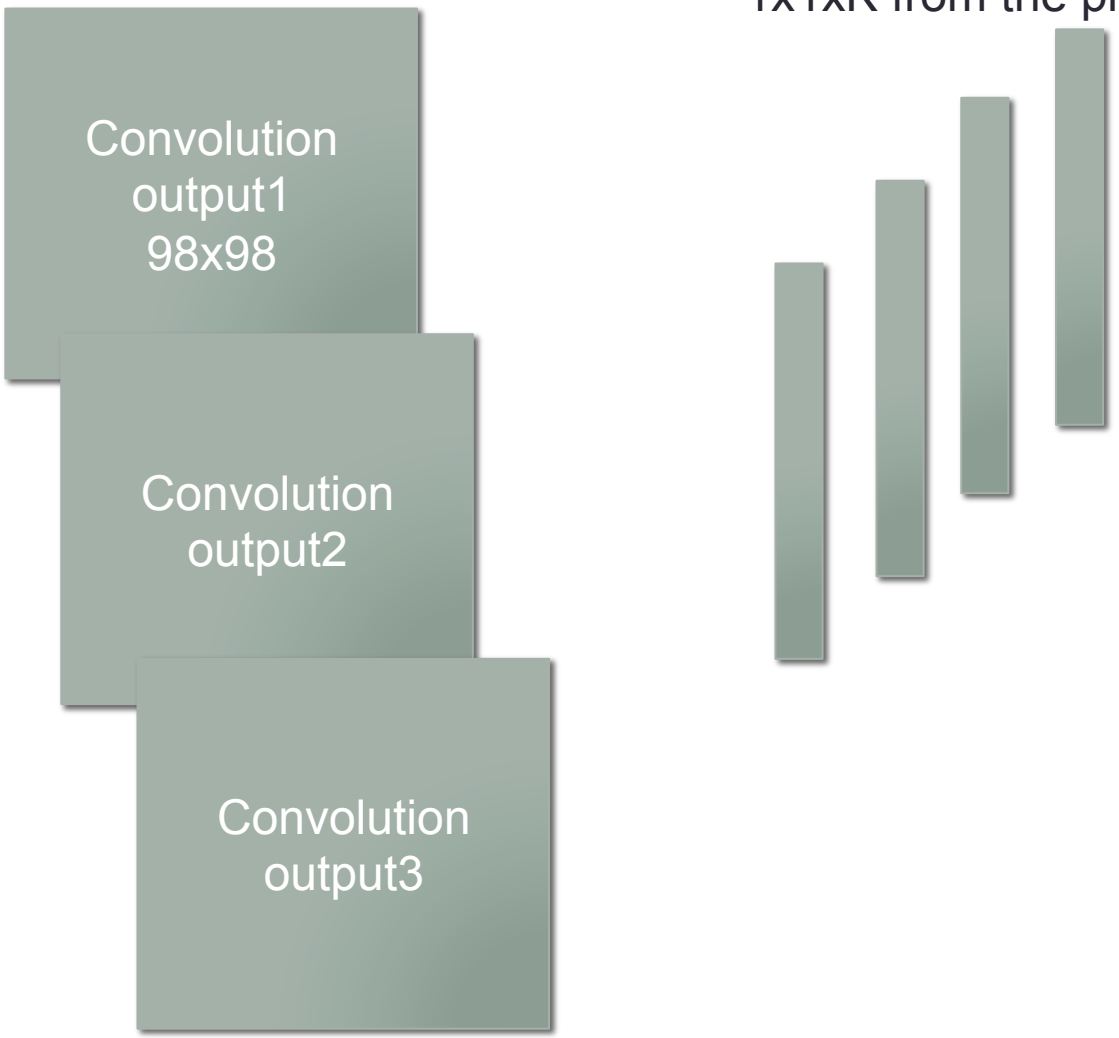
1x1 Convolutions

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

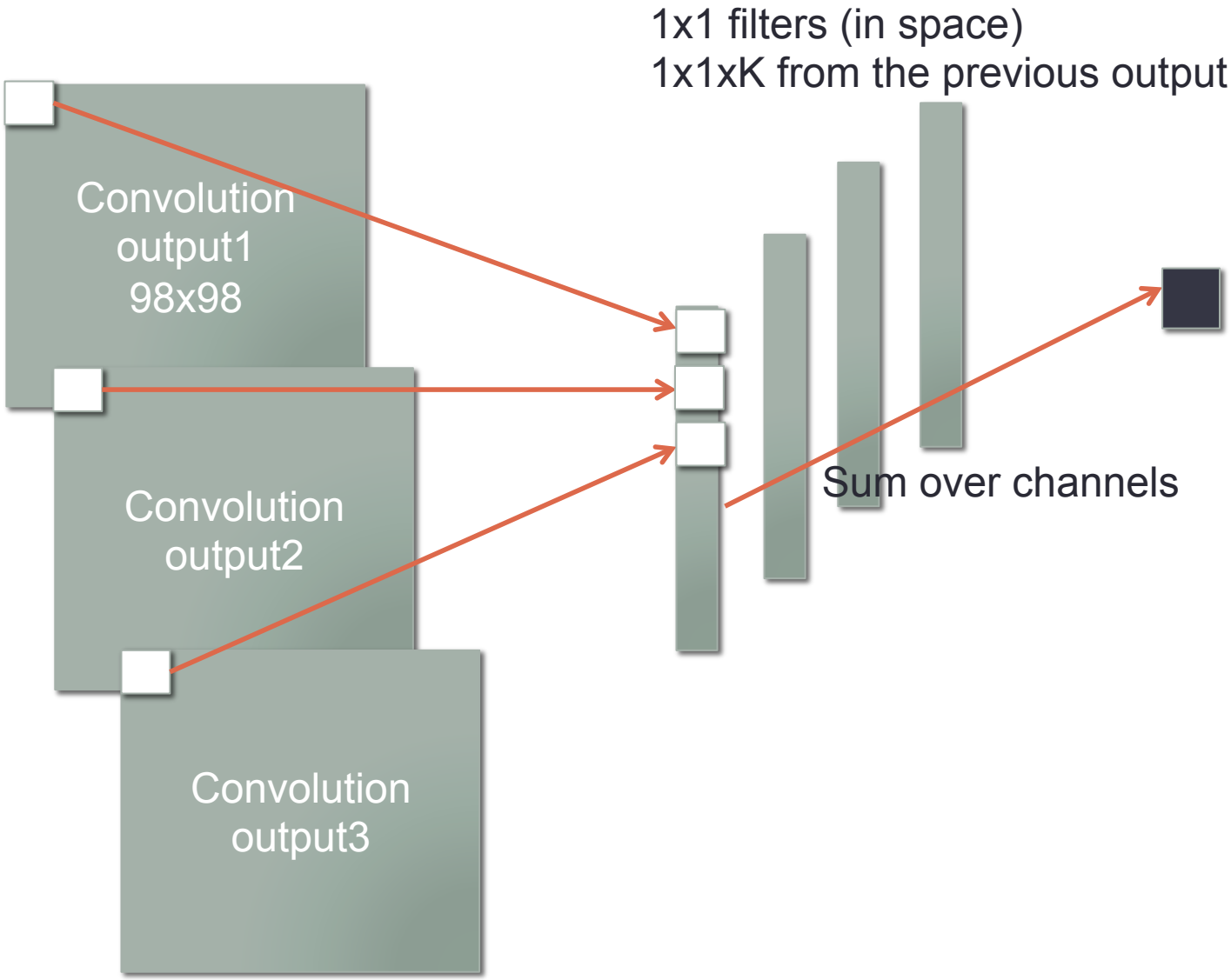


1x1 Convolutions

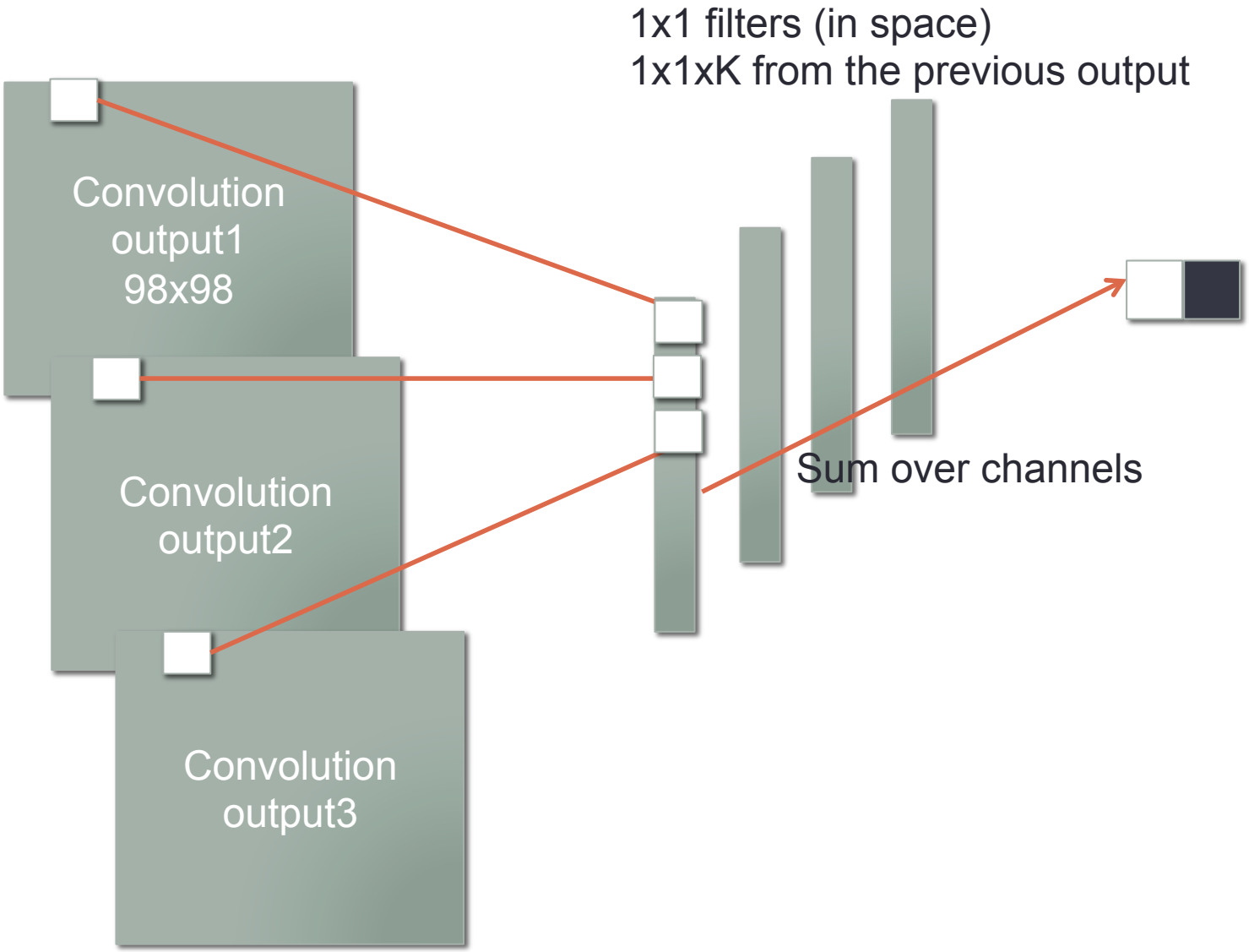
1x1 filters (in space)
1x1xK from the previous output



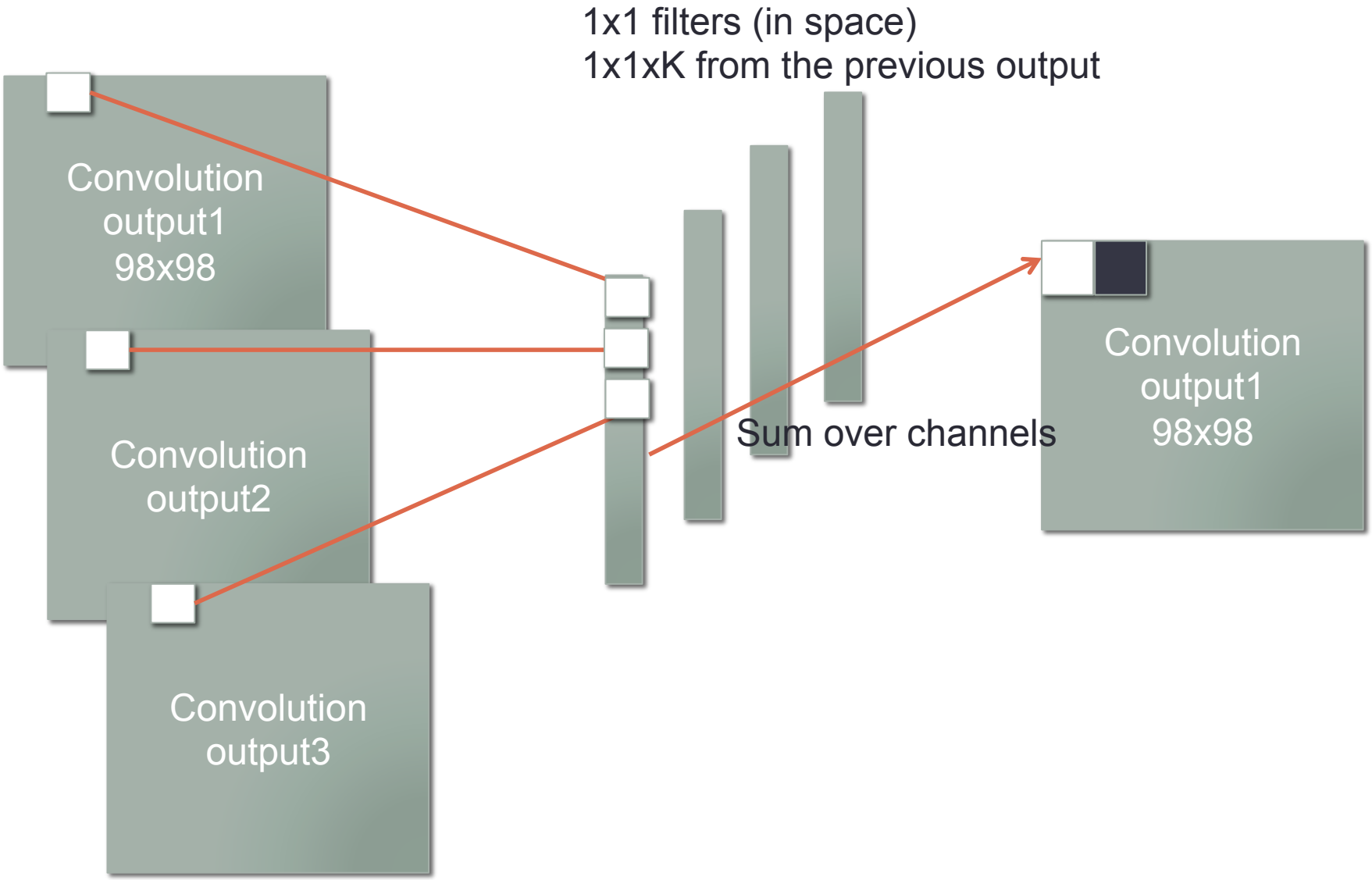
1x1 Convolutions



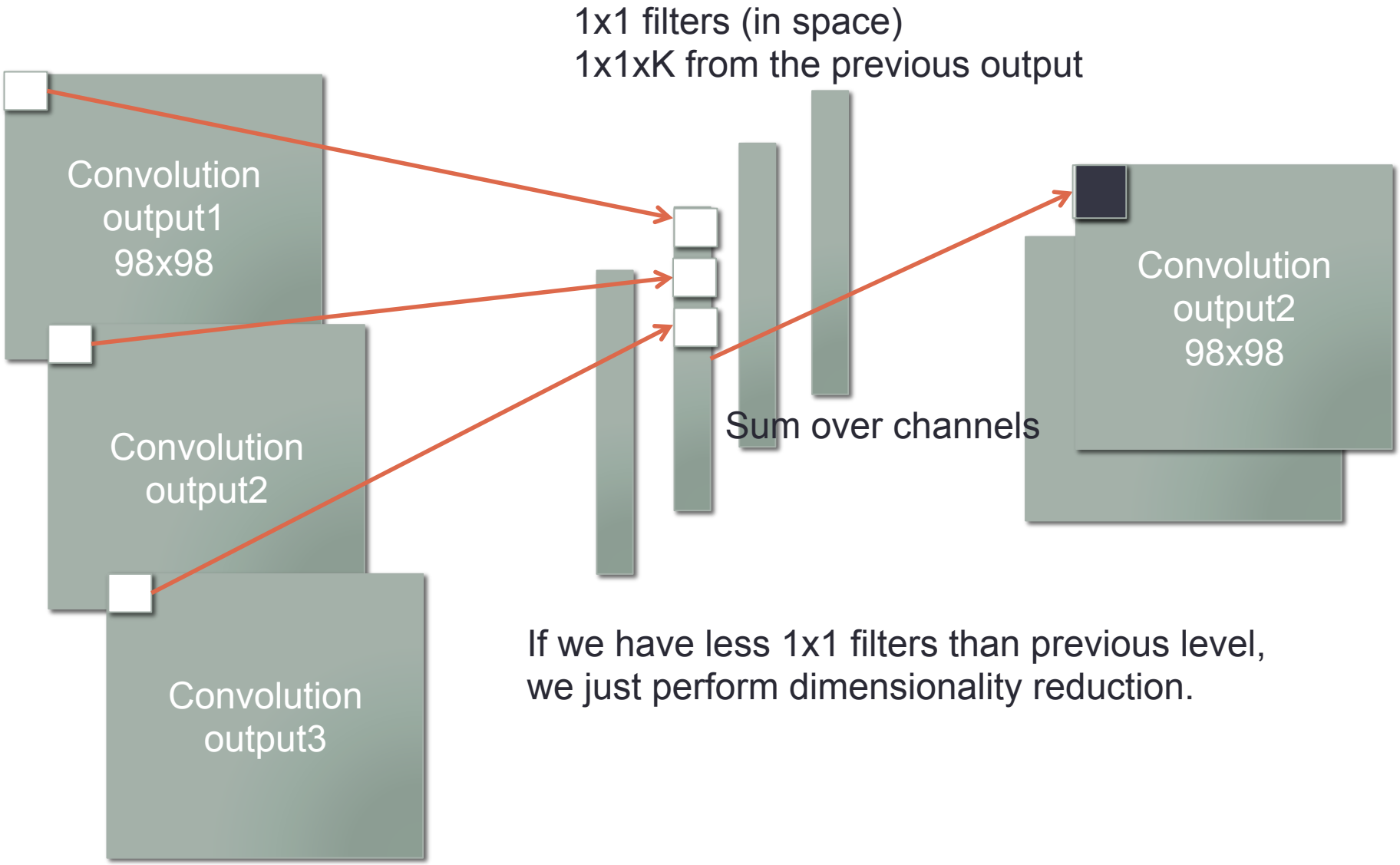
1x1 Convolutions



1x1 Convolutions



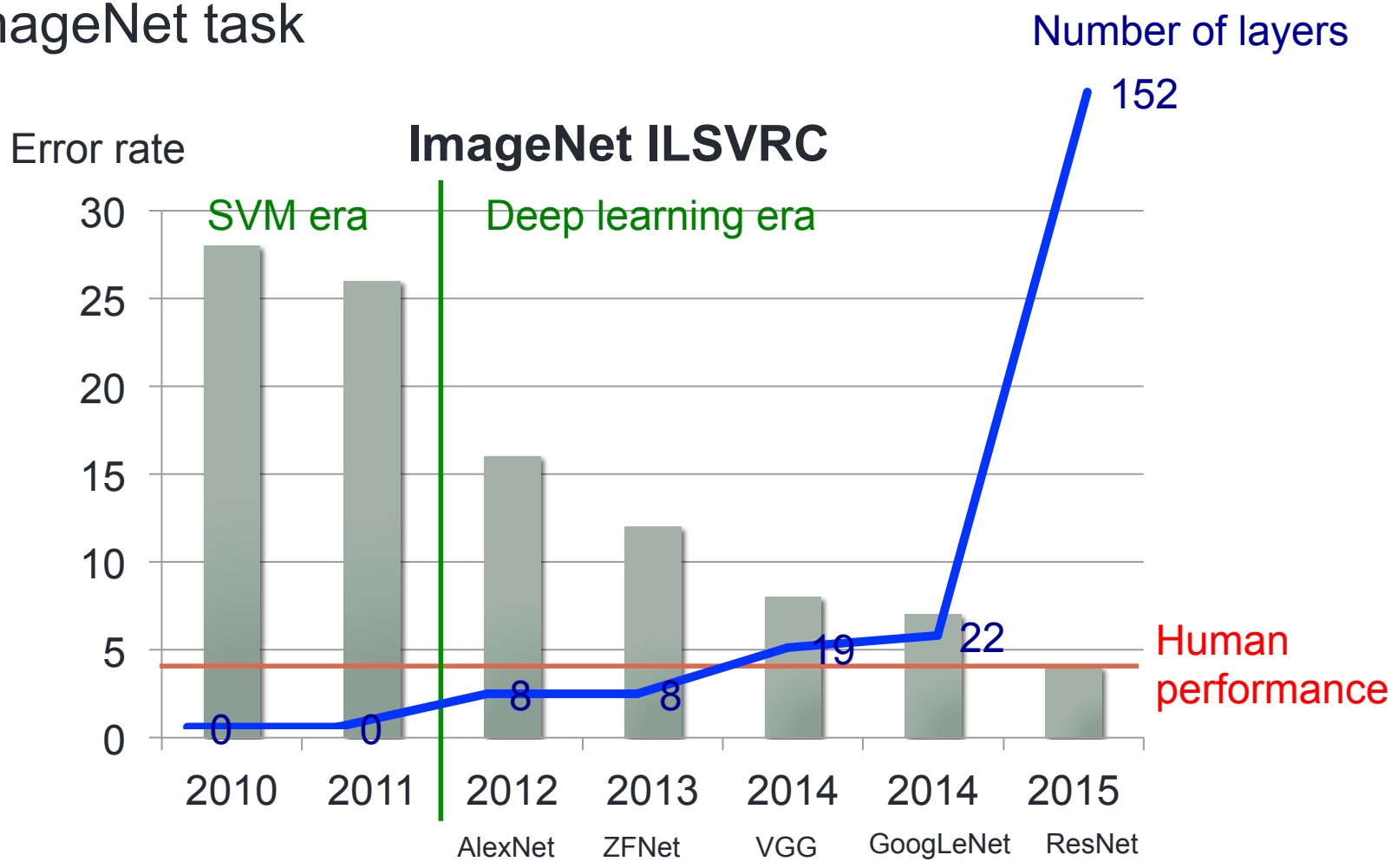
1x1 Convolutions



If we have less 1x1 filters than previous level, we just perform dimensionality reduction.

Wider and deeper networks

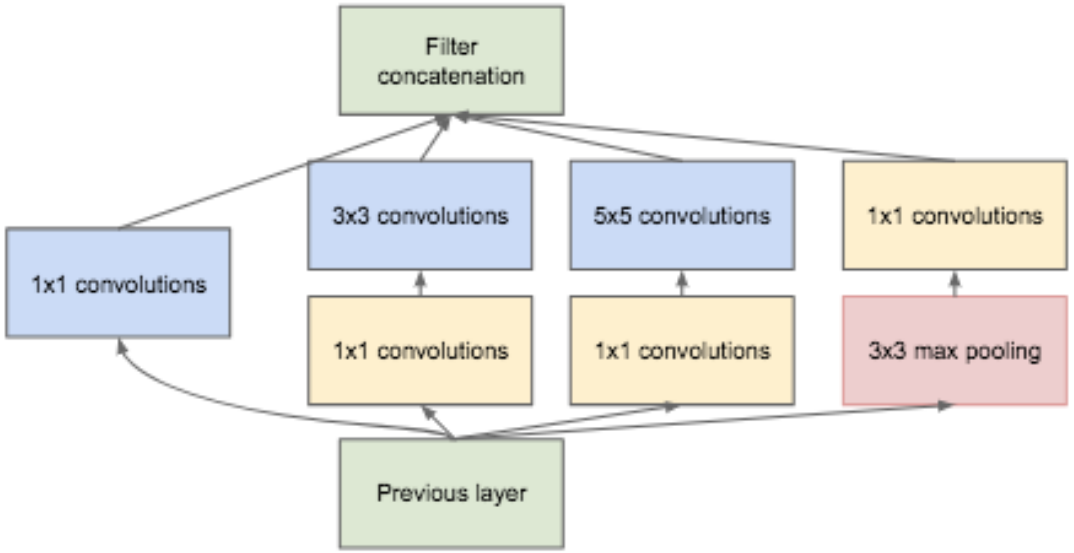
- ImageNet task



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

ImageNET prize winner

- AlexNet – first deep model based on LeNET (simple CNN from 1990s)
- ZFNet – AlexNET just deeper, and better tuned hyperparameters
- GoogLeNet (Inception Model)



ImageNET prize winner 2

- VGGNet - just a bigger network. Notable mention because model available
- ResNet – Next class

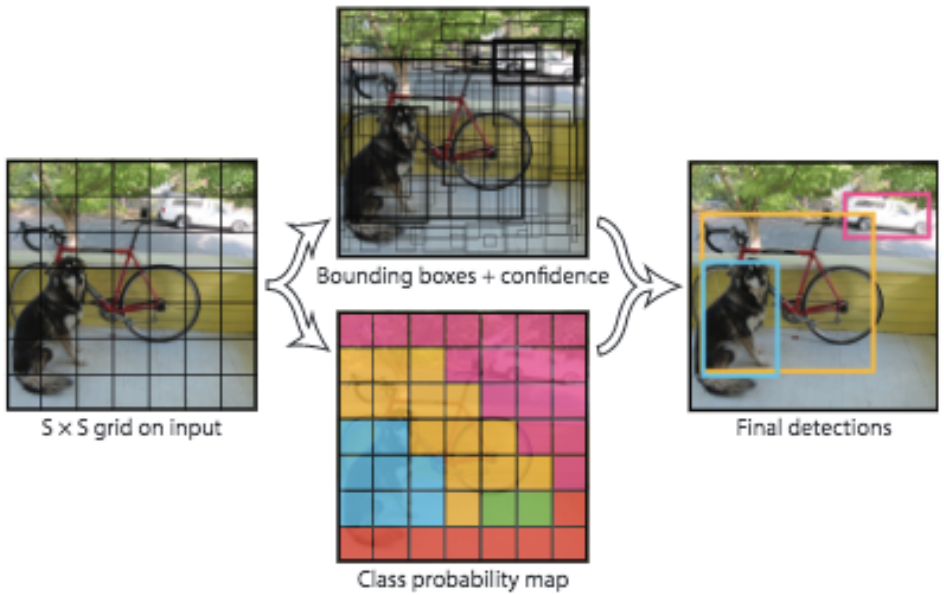
Other models

- YOLO: Frame object classification to regression
  - Regression: coordinates + class probabilities
  - Traditional object detector segments the original images into patches and scan the patches – each different patch is scanned many times

https://pjreddie.com/media/files/papers/yolo_1.pdf

YOLO

- YOLO: Frame object classification to regression
  - Regression: coordinates + class probabilities
  - You Only Look Once: output possible bounding boxes and class
  - A post-processing step is used to merge the bounding boxes
  - Fast model for real time object detectors
  - Model is just VGG



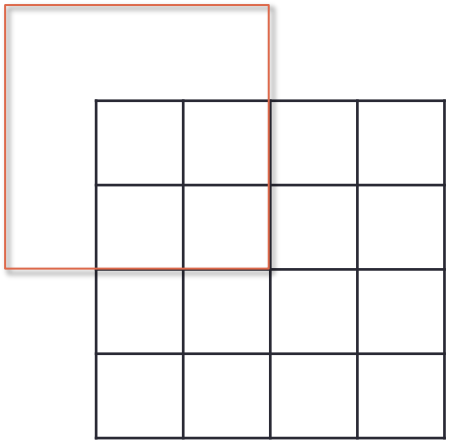
https://pjreddie.com/media/files/papers/yolo_1.pdf

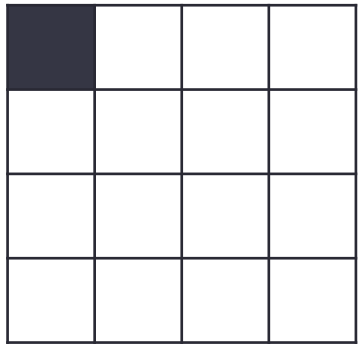
De-convolution layers

- Yet another abused of notation made by vision folks
  - Conventional meaning:
    - A method to reverse an effect of a filter
    - Blurred image -> de-convolution -> Good image
  - Neural network meaning:
    - Something that reverse the order of convolution computation
    - Backward pass of a convolutional layer
- Used for upsampling

Convolution

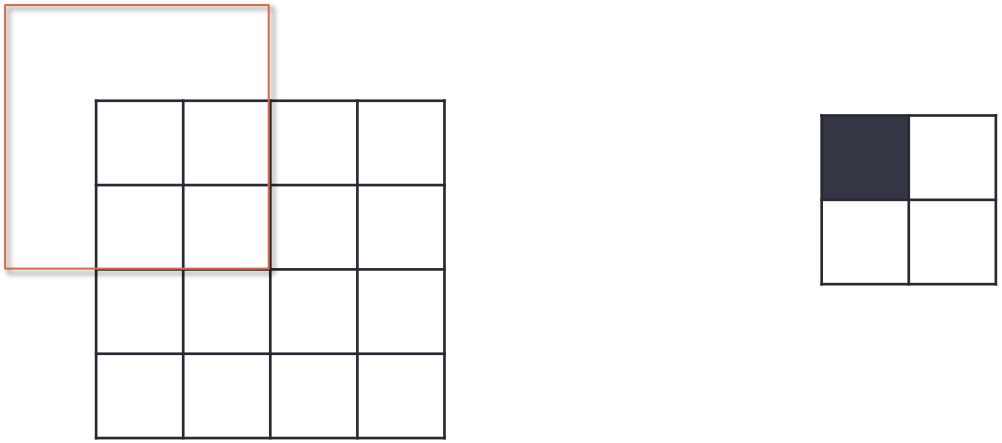
- 3x3 filter pad, stride 1, pad 1





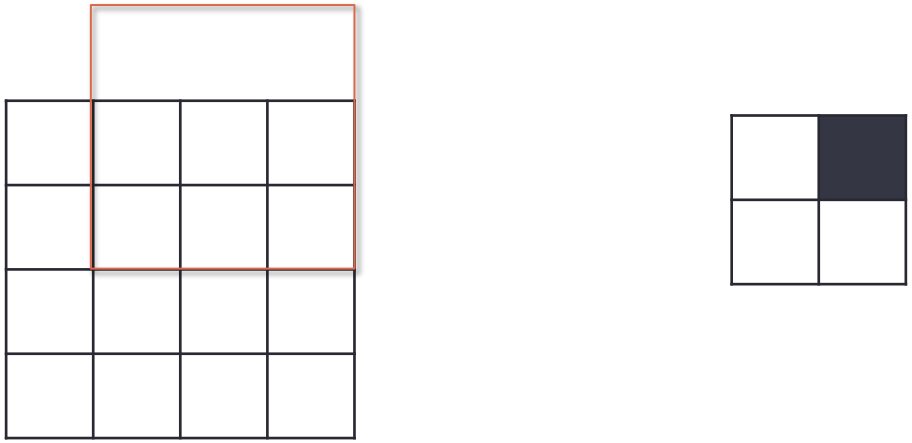
Convolution

- 3x3 filter pad, stride 2, pad 1



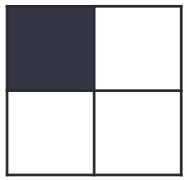
Convolution

- 3x3 filter pad, stride 2, pad 1

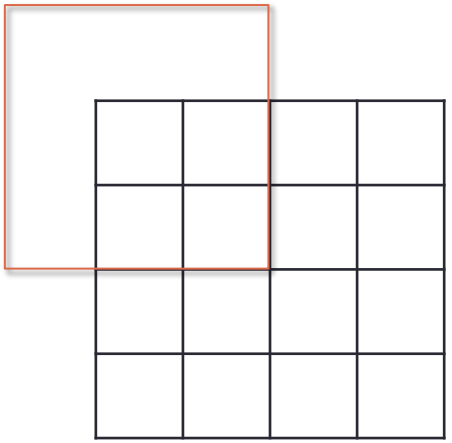


De-convolution

- 3x3 de-convolution filter, stride 2, pad 1



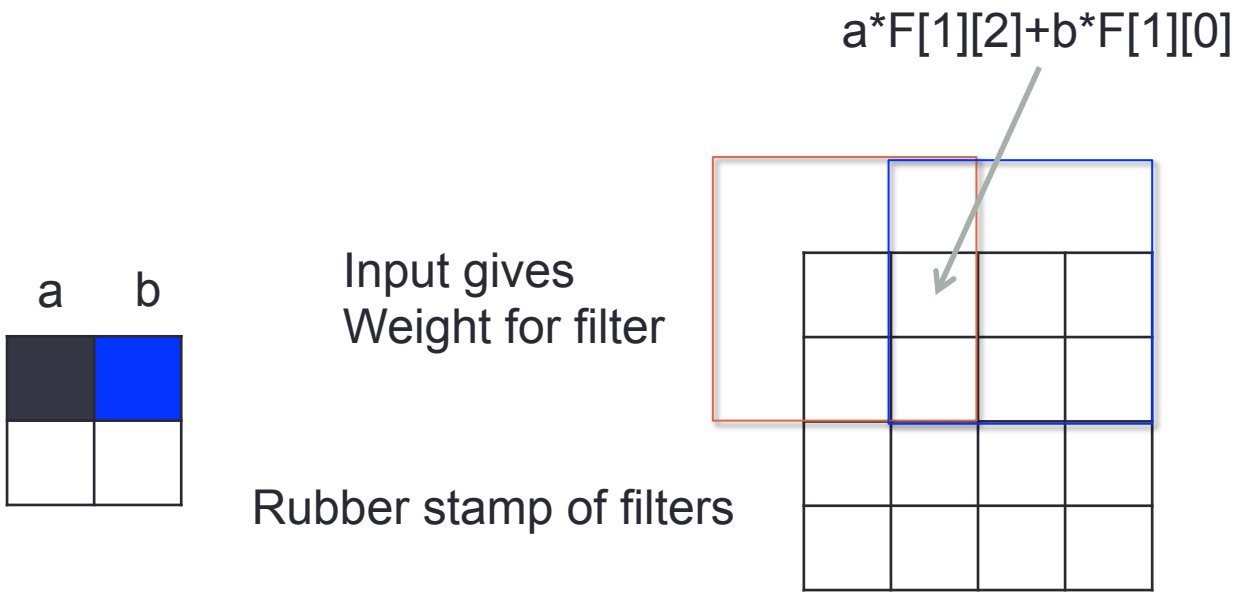
Input gives
Weight for filter



Rubber stamp

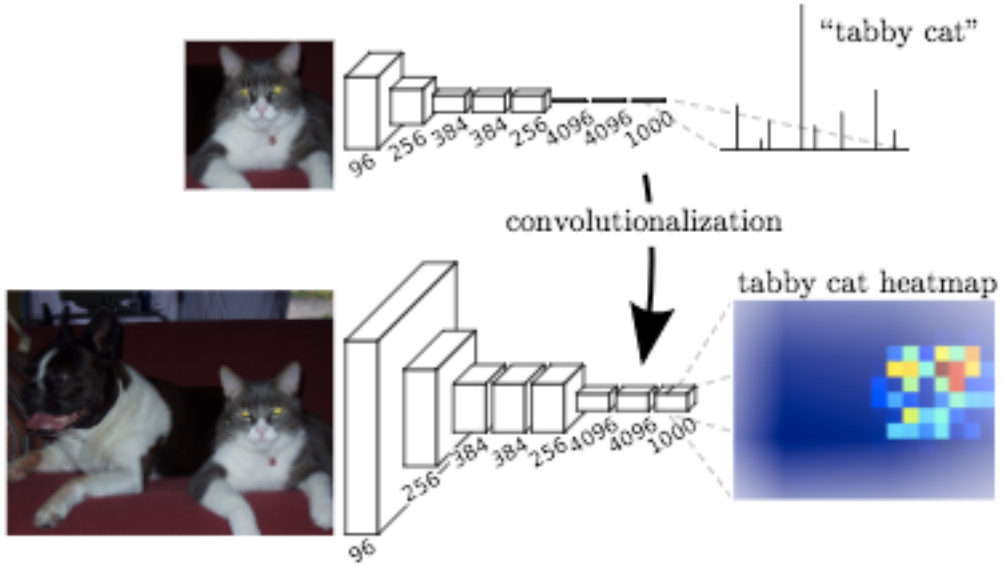
De-convolution

- 3x3 deconvolution filter, stride 2, pad 1



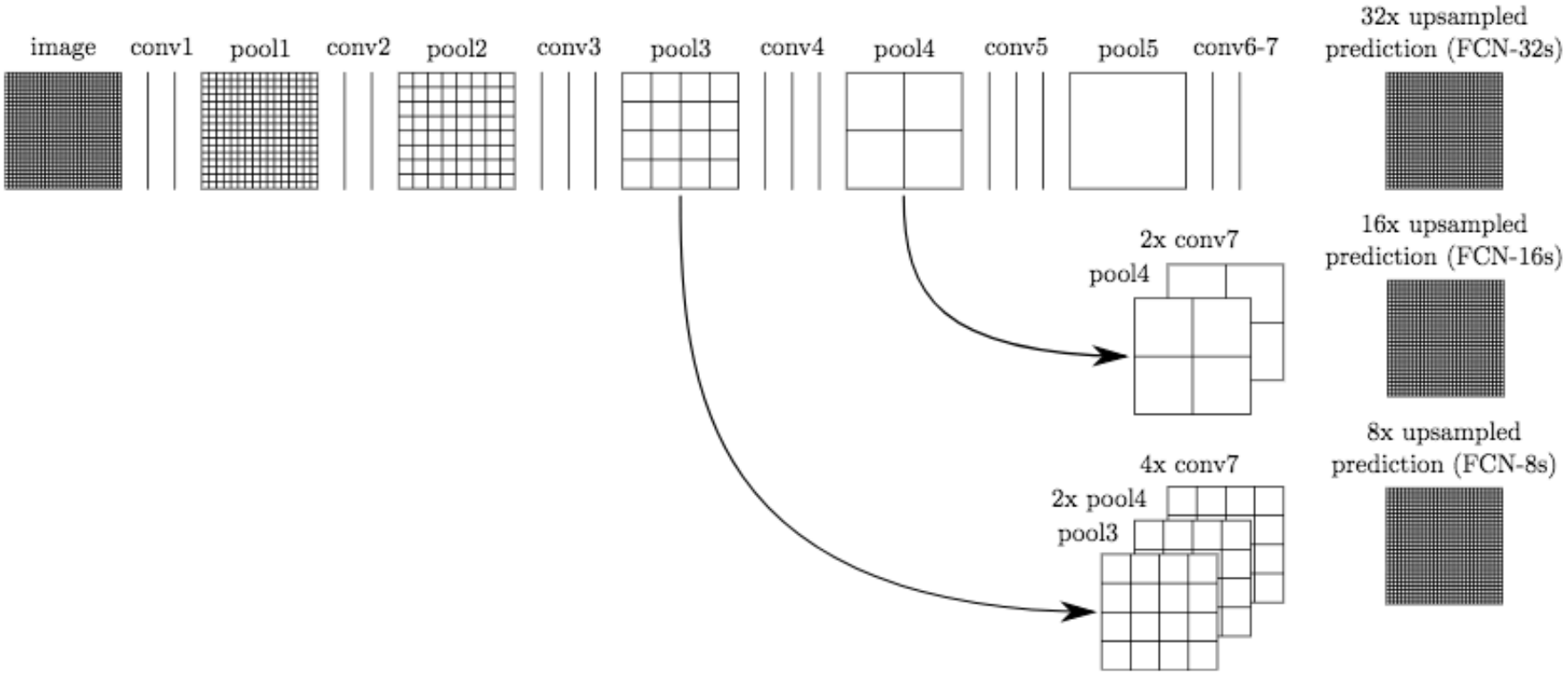
Other names because this name sucks (for me)
- Convolution transpose, upconvolution, backward strided convolution

De-convolution for segmentation



https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf

De-convolution for segmentation



https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf

De-convolution for segmentation

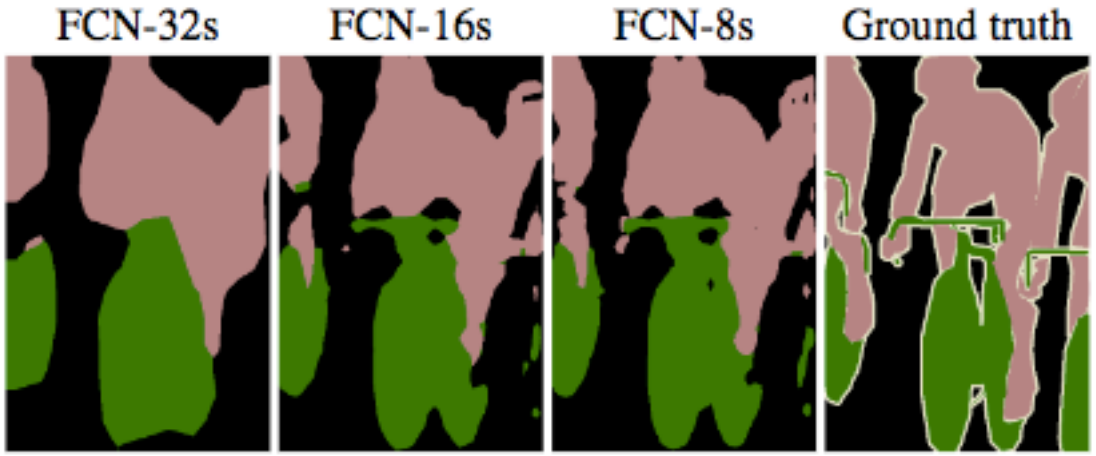
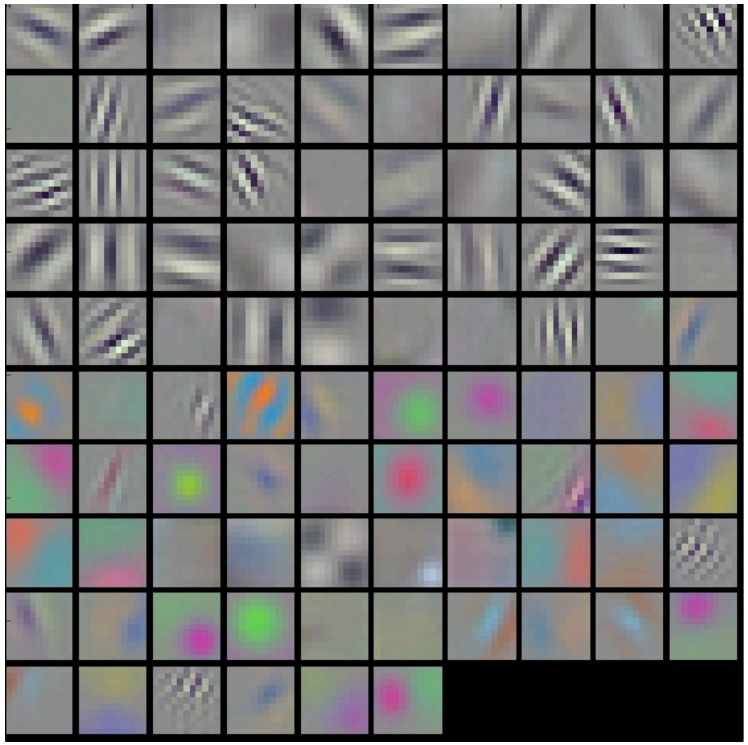


Figure 4. Refining fully convolutional nets by fusing information from layers with different strides improves segmentation detail. The first three images show the output from our 32, 16, and 8 pixel stride nets (see Figure 3).

https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf

Visualizing convolutional layers

- Just like PCA, we can visualize the weights of a transform
- “Matched filters”



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

CNN

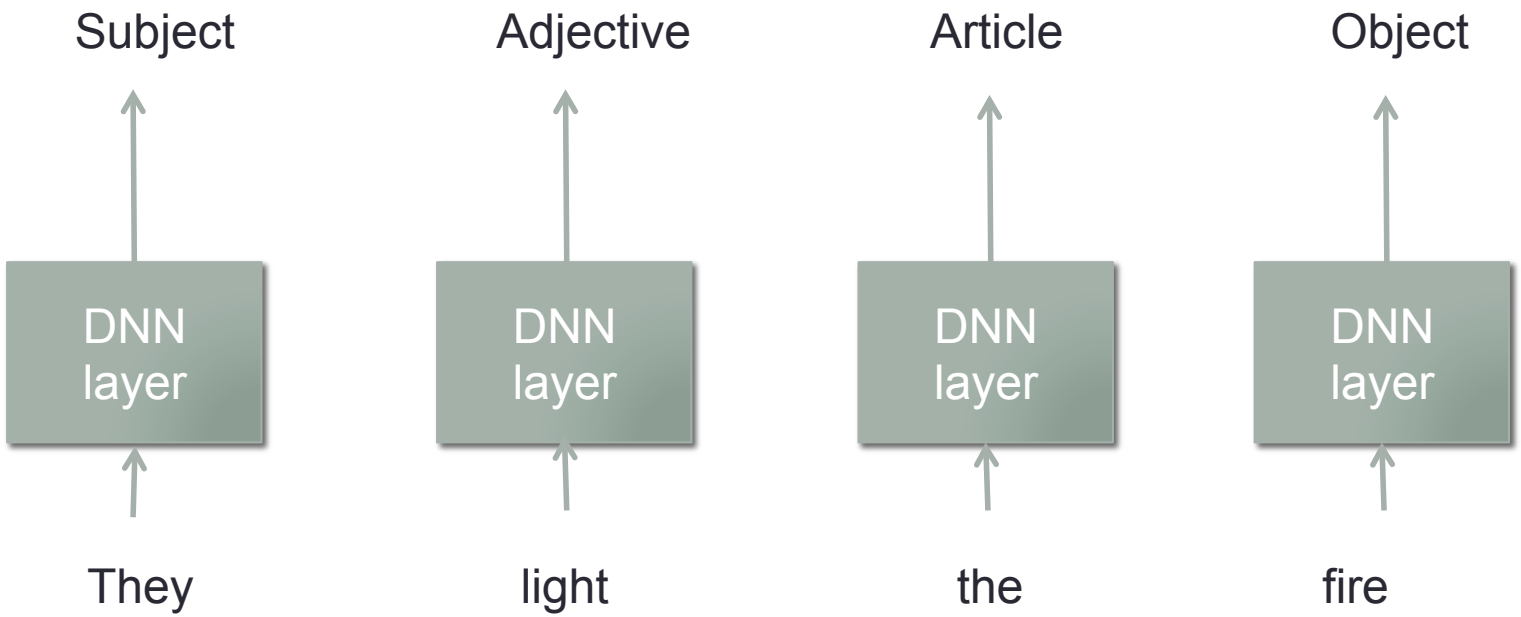
- Convolutional layer
- Subsampling

- Sharing of parameters in space

- Sharing of parameters in time?

Recurrent neural network (RNN)

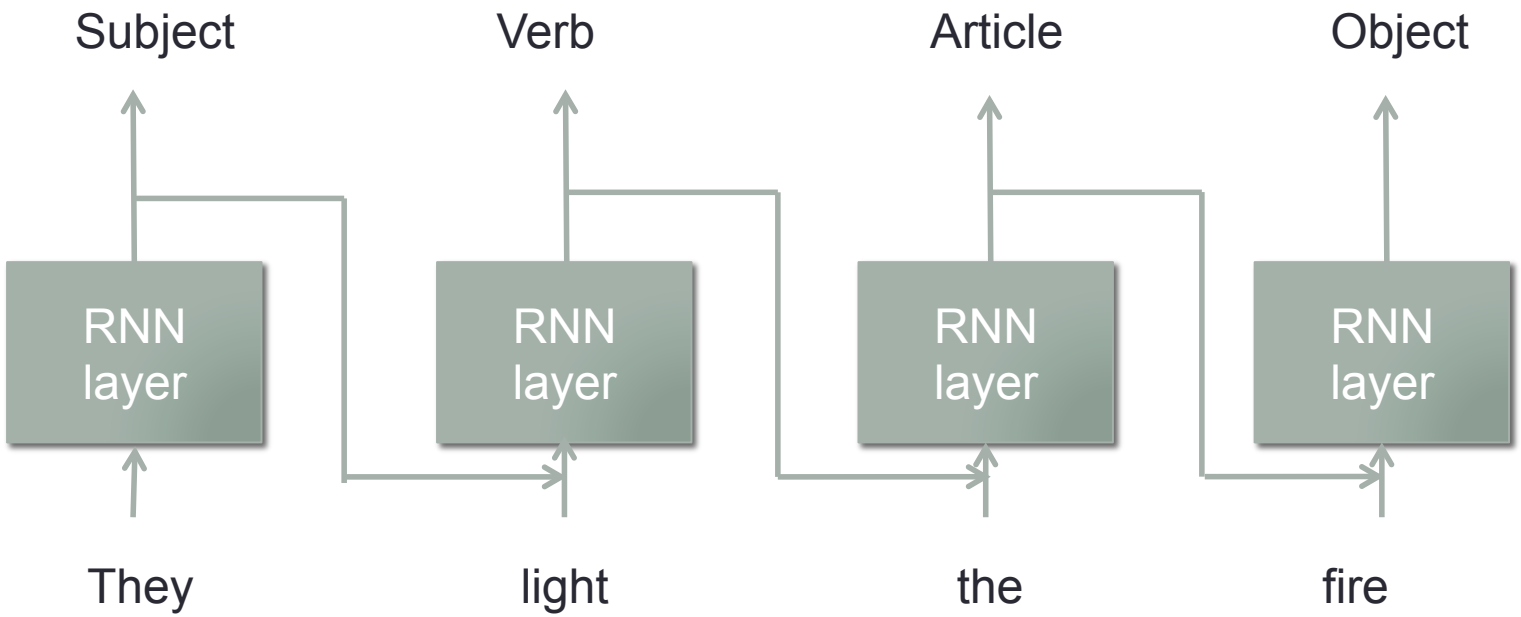
- DNN framework



Problem: need a way to remember the past

Recurrent neural network (RNN)

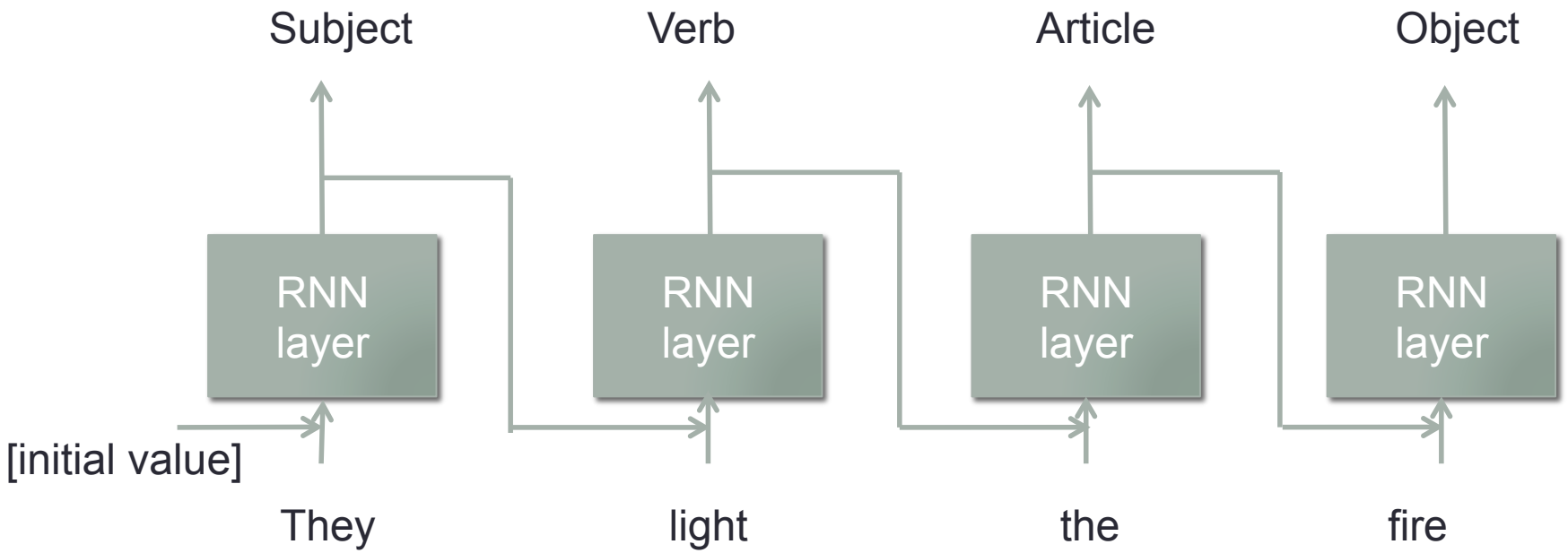
- RNN framework



Output of the layer encodes something meaningful about the past

Recurrent neural network (RNN)

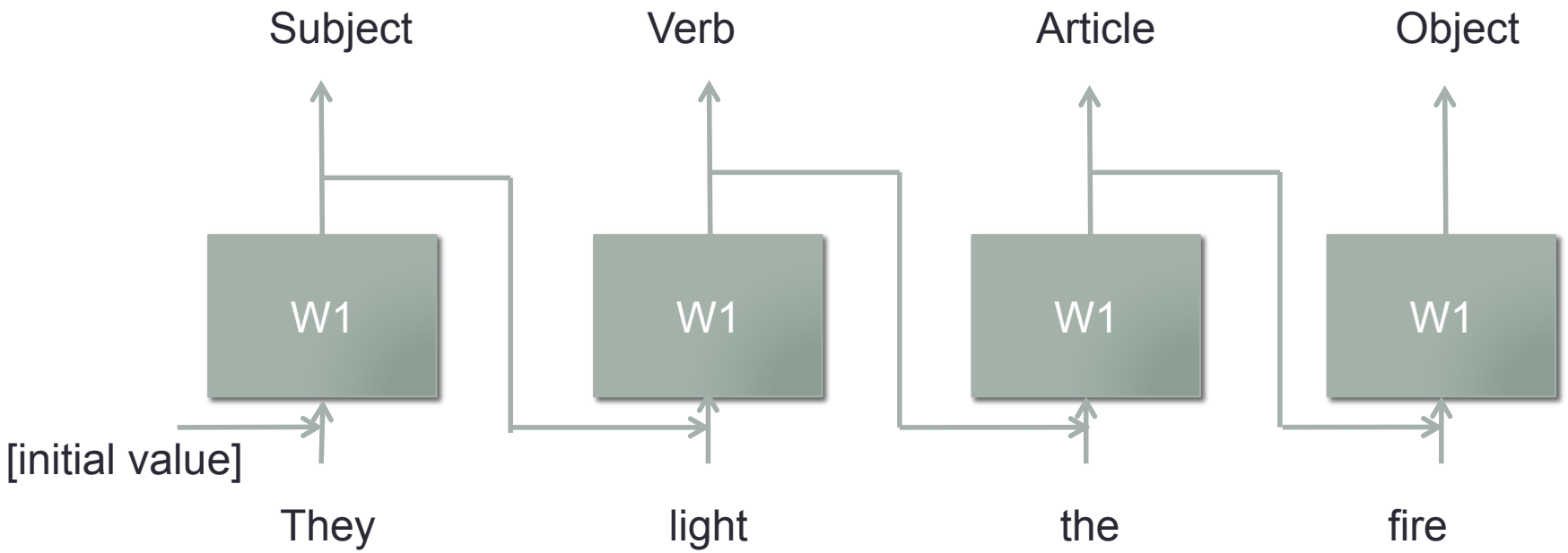
- RNN framework



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

Training a recurrent neural network

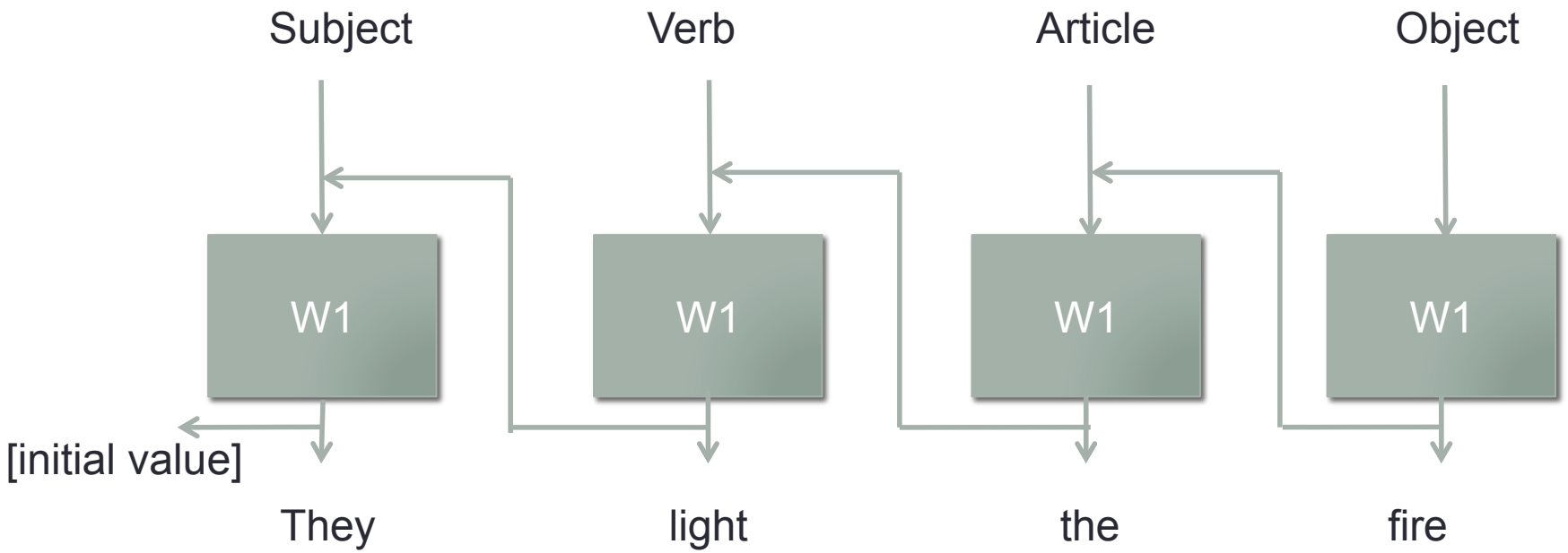
- Computation graph



Parameter sharing

Training a recurrent neural network

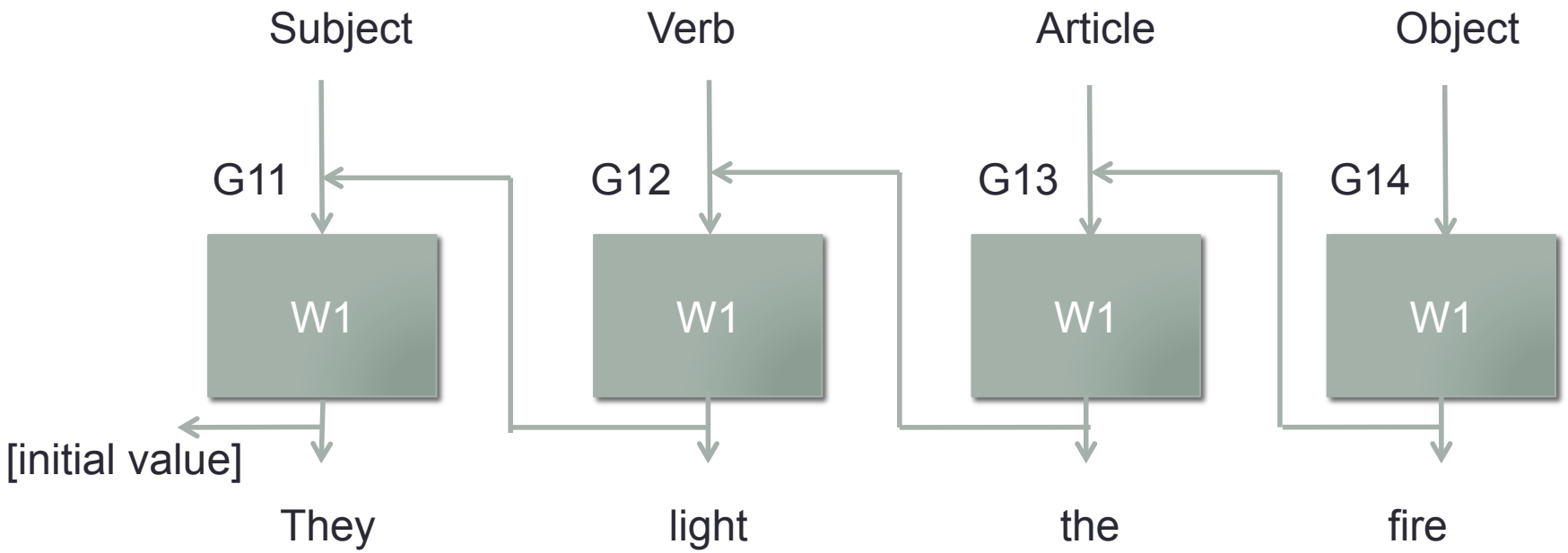
- Backward Computation graph



Backpropagation through time (BPTT)

BPTT

- Backward Computation graph

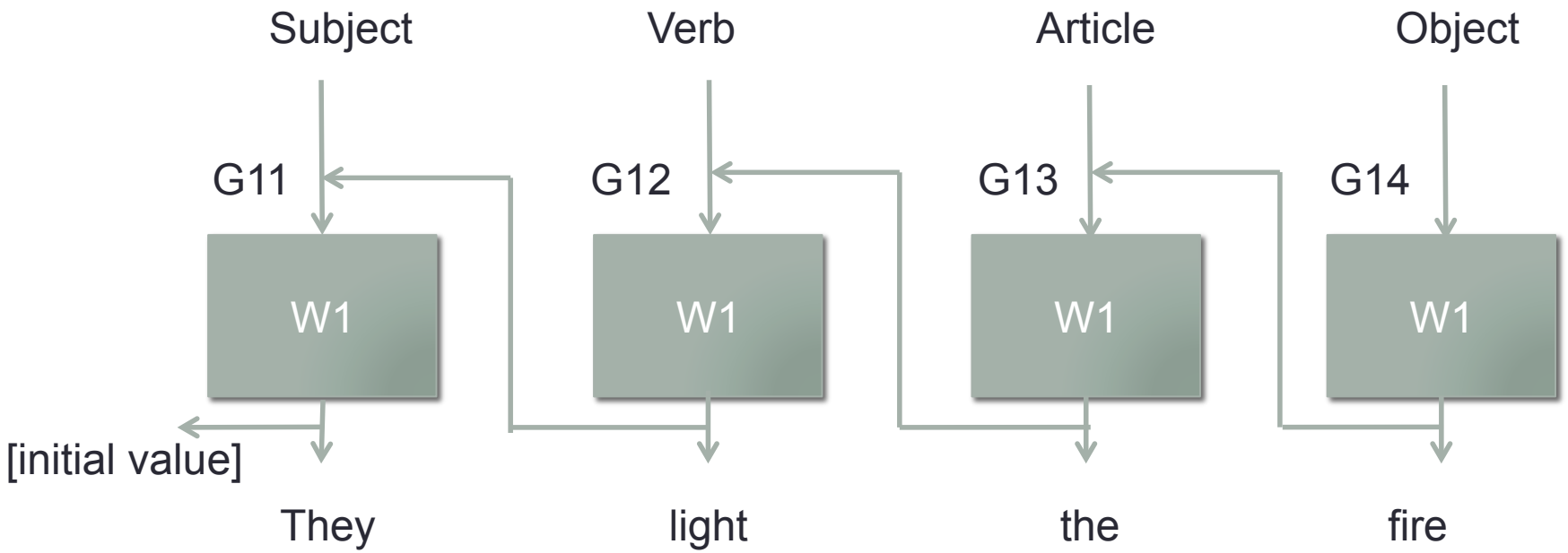


W1 <- W1 + G11 + G12 + G13 + G14

Cannot deal with infinitely long recurrent
Gradient explosion, vanishing

Truncated BPTT

- Backward Computation graph

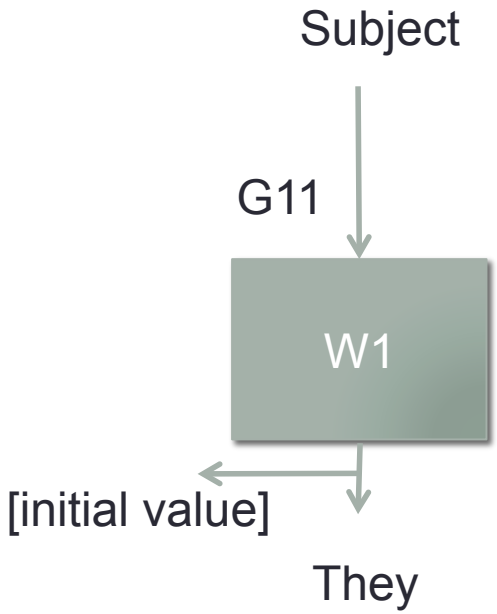


W1 <- W1 + G13 + G14

Pick a maximum number of time steps and go backwards that much

Truncated BPTT

- Backward Computation graph

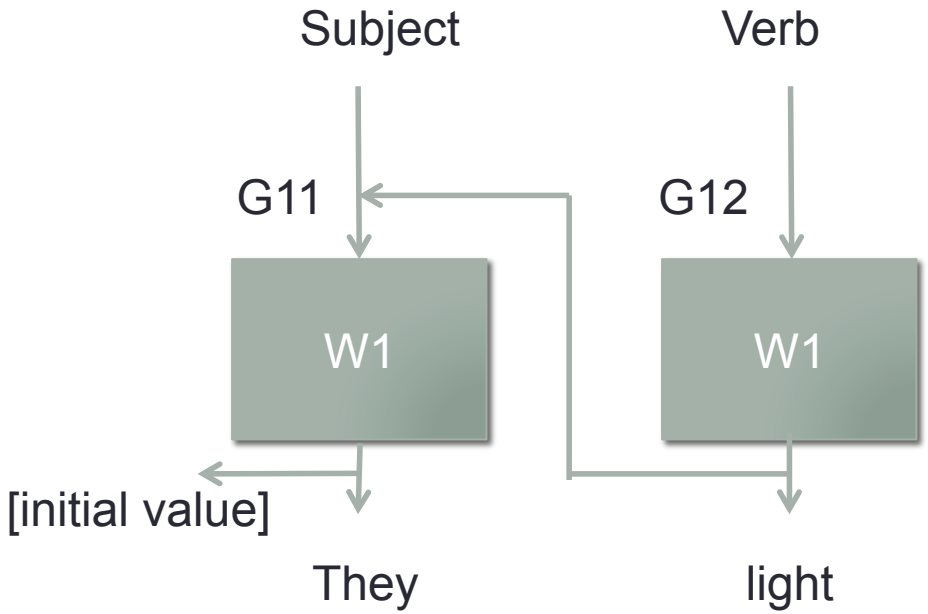


W1 <- W1 + G11

Pick a maximum number of time steps and go backwards that much

Truncated BPTT

- Backward Computation graph

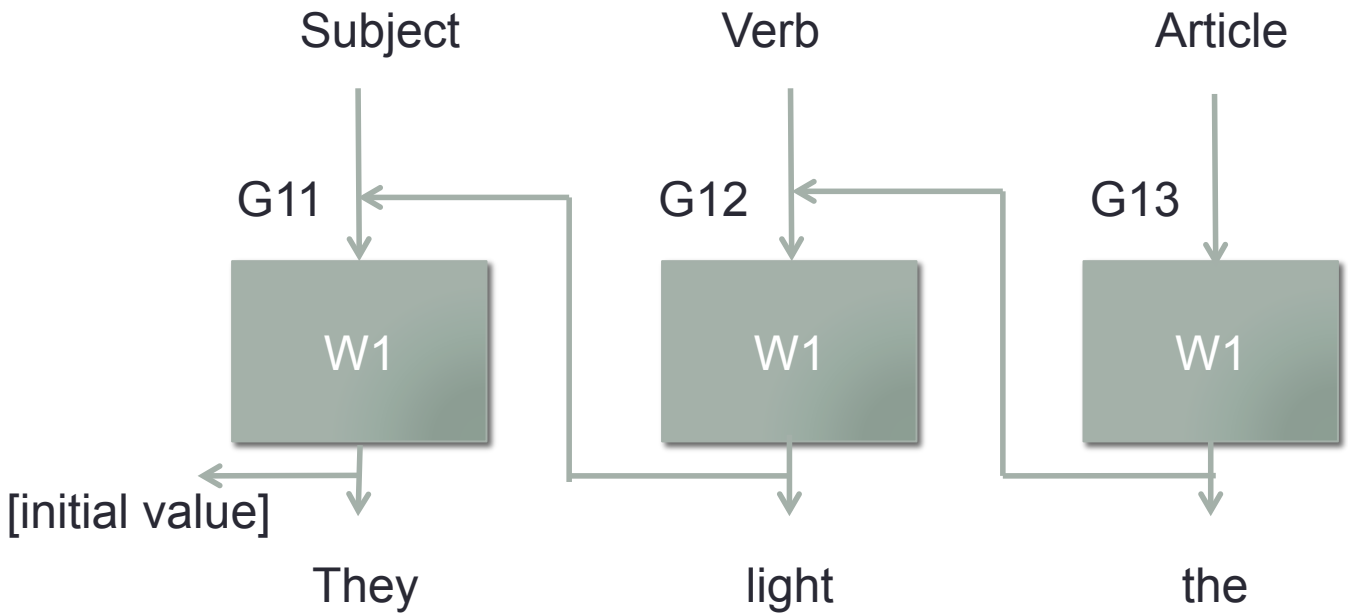


W1 <- W1 + G11 + G12

Pick a maximum number of time steps and go backwards that much

Truncated BPTT

- Backward Computation graph

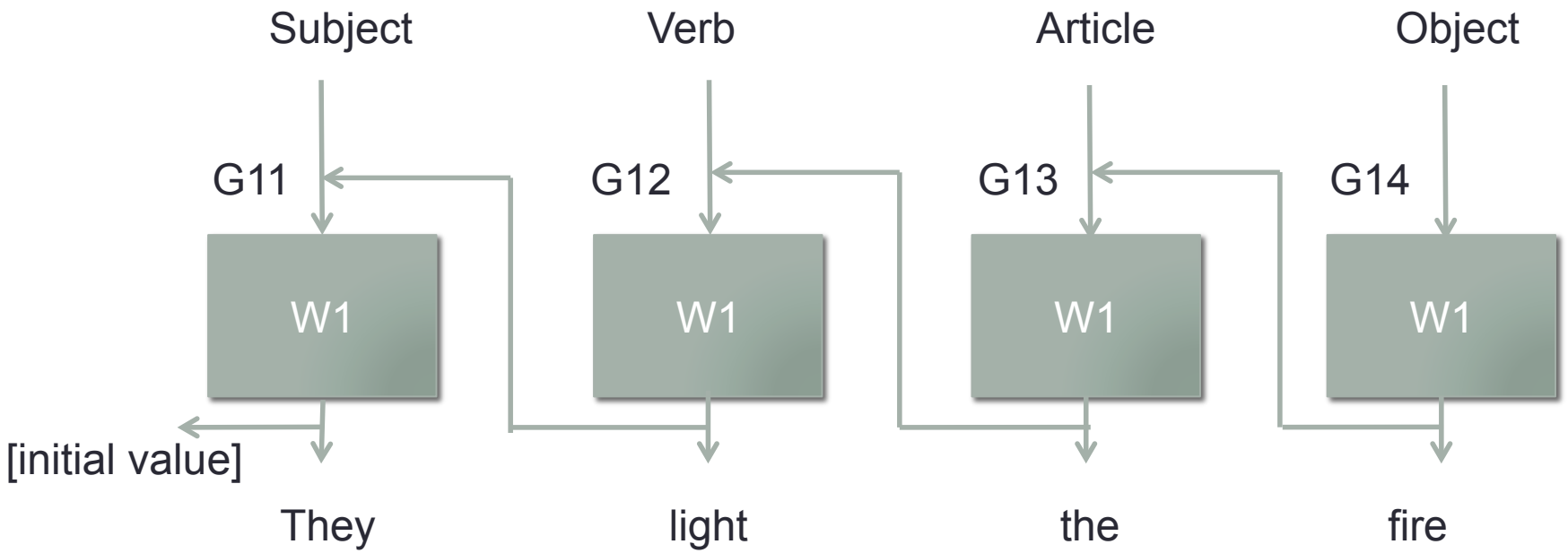


W1 <- W1 + G12 + G13

Pick a maximum number of time steps and go backwards that much

Truncated BPTT

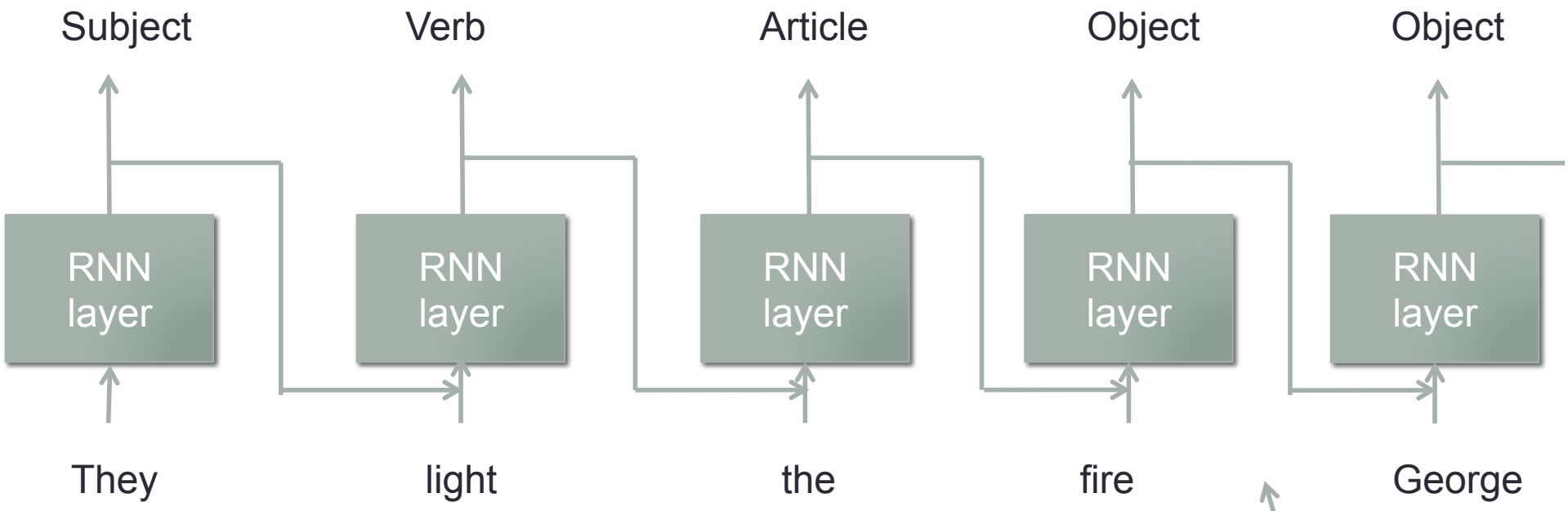
- Backward Computation graph



W1 <- W1 + G13 + G14

Pick a maximum number of time steps and go backwards that much

Recurrent neural network (RNN)

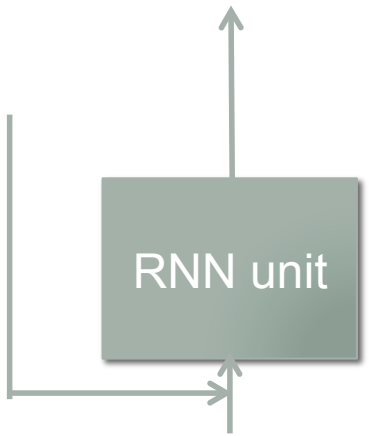


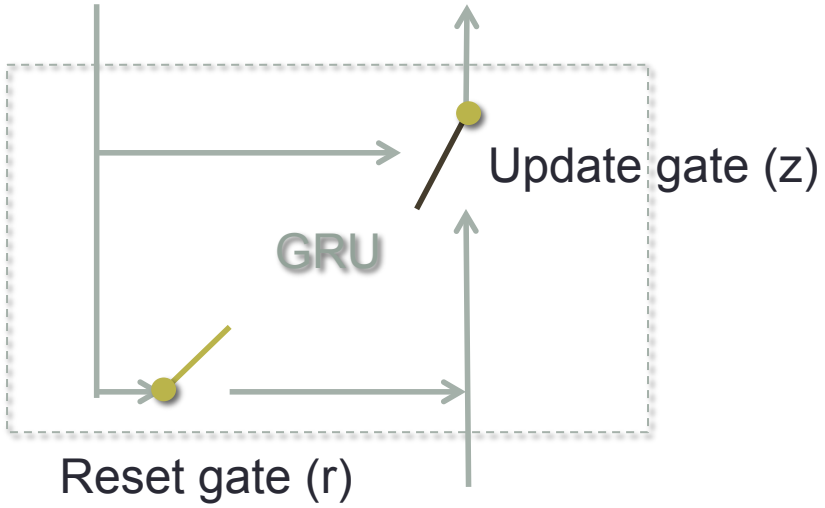
Problem2: needs a way to stop remembering

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)

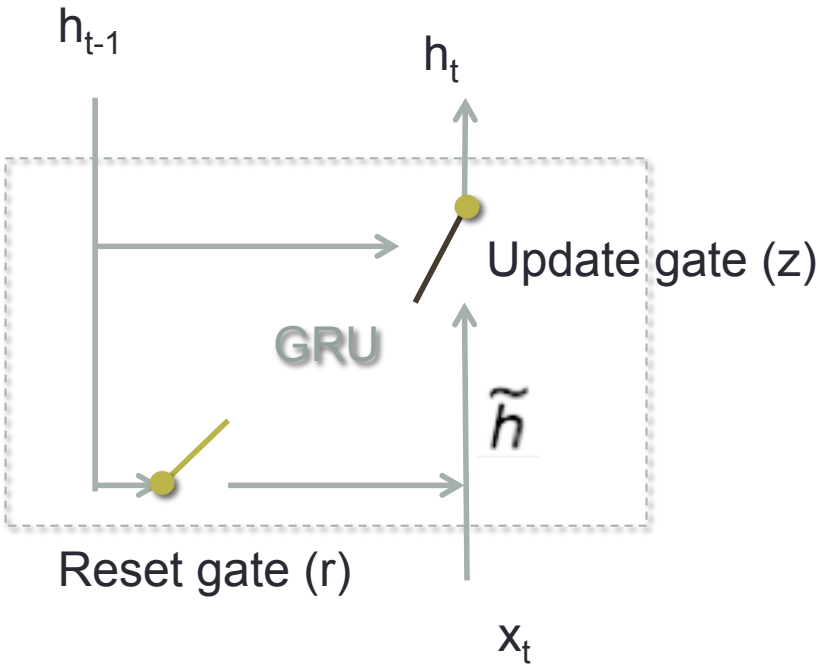




Gated Recurrent Unit (GRU)

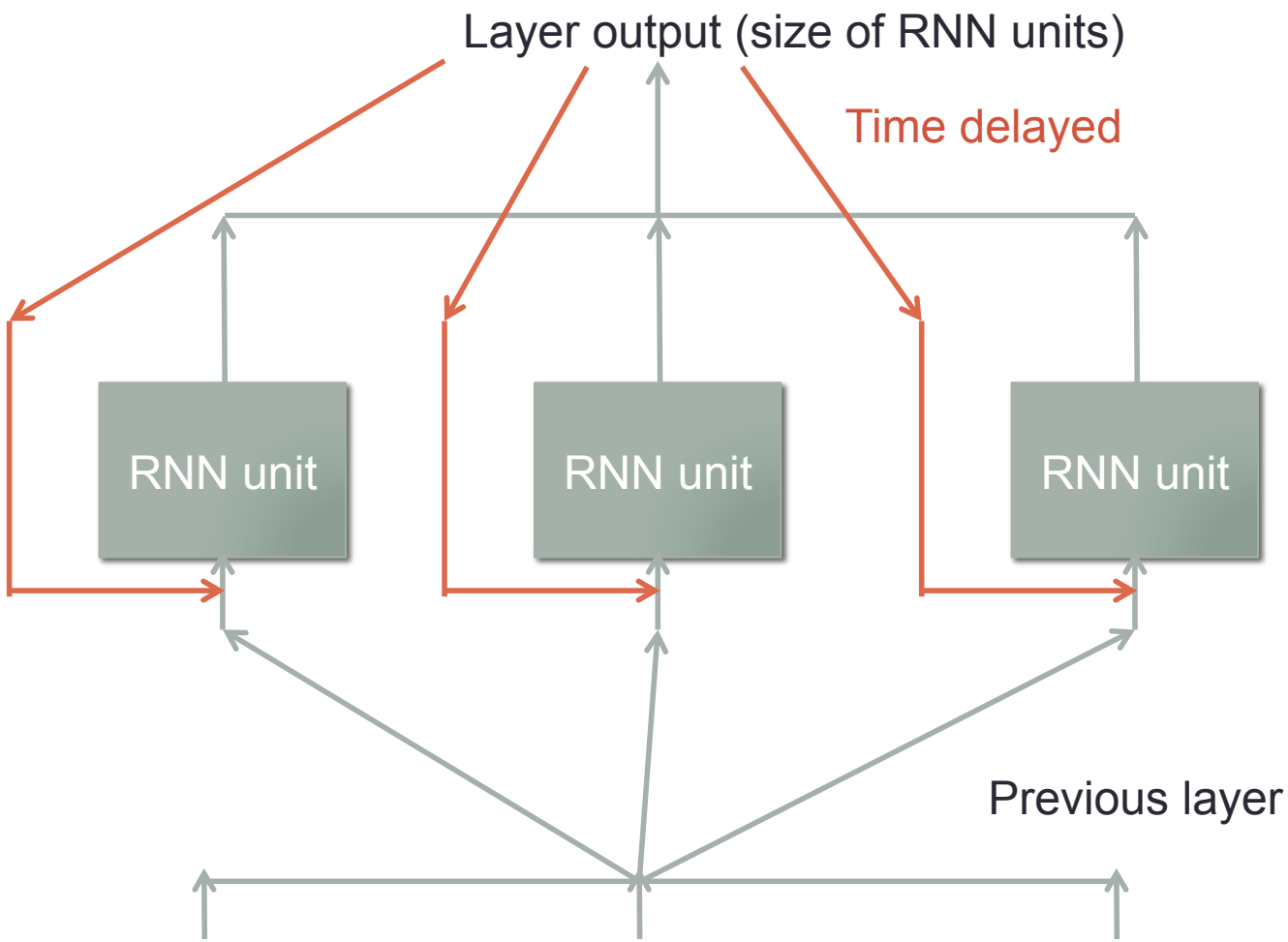
$$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z)$$
$$r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r)$$
$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \sigma_h(W_h x_t + U_h(r_t \circ h_{t-1}) + b_h)$$

$\tilde{h}$



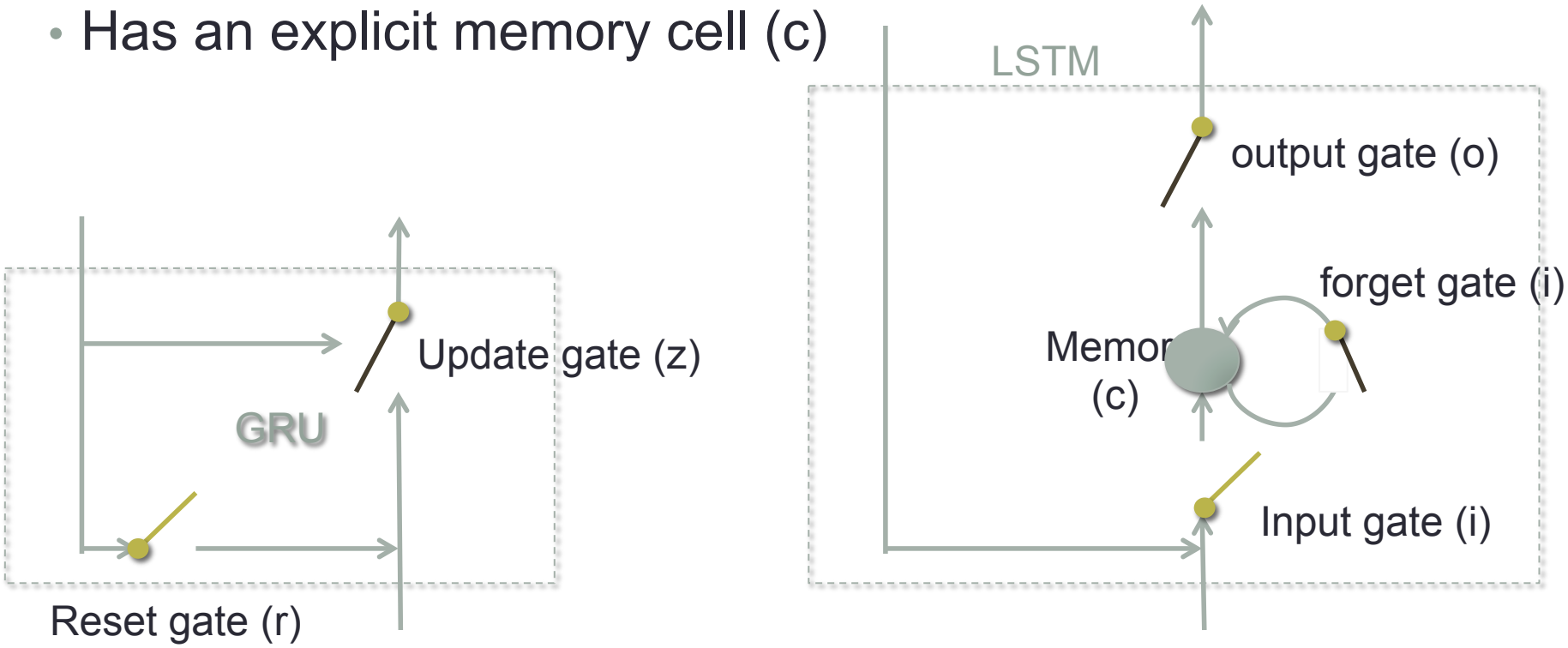
Note: non-linearity of GRU and LSTM are usually sigmoid/tanh – not ReLU

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

Long Short-Term Memory (LSTM)

j is the index of the LSTM cell

$$o_t^j = \sigma\left(W_o \mathbf{x}_t + U_o \mathbf{h}_{t-1} + V_o \mathbf{c}_t\right)^j$$

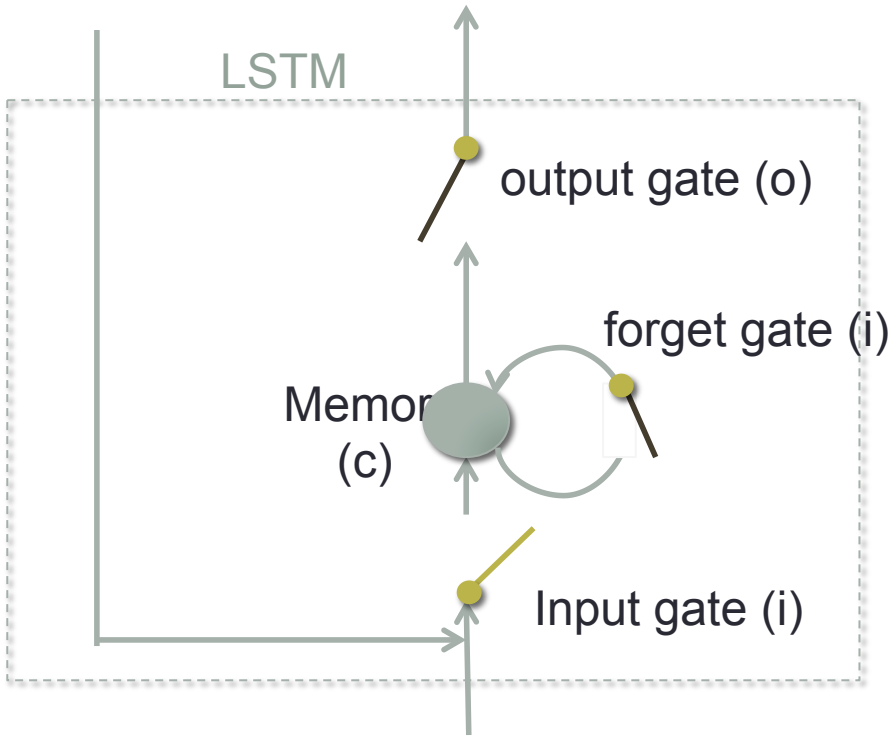
$$f_t^j = \sigma\left(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1} + V_f \mathbf{c}_{t-1}\right)^j$$

$$i_t^j = \sigma\left(W_i \mathbf{x}_t + U_i \mathbf{h}_{t-1} + V_i \mathbf{c}_{t-1}\right)^j .$$

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

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

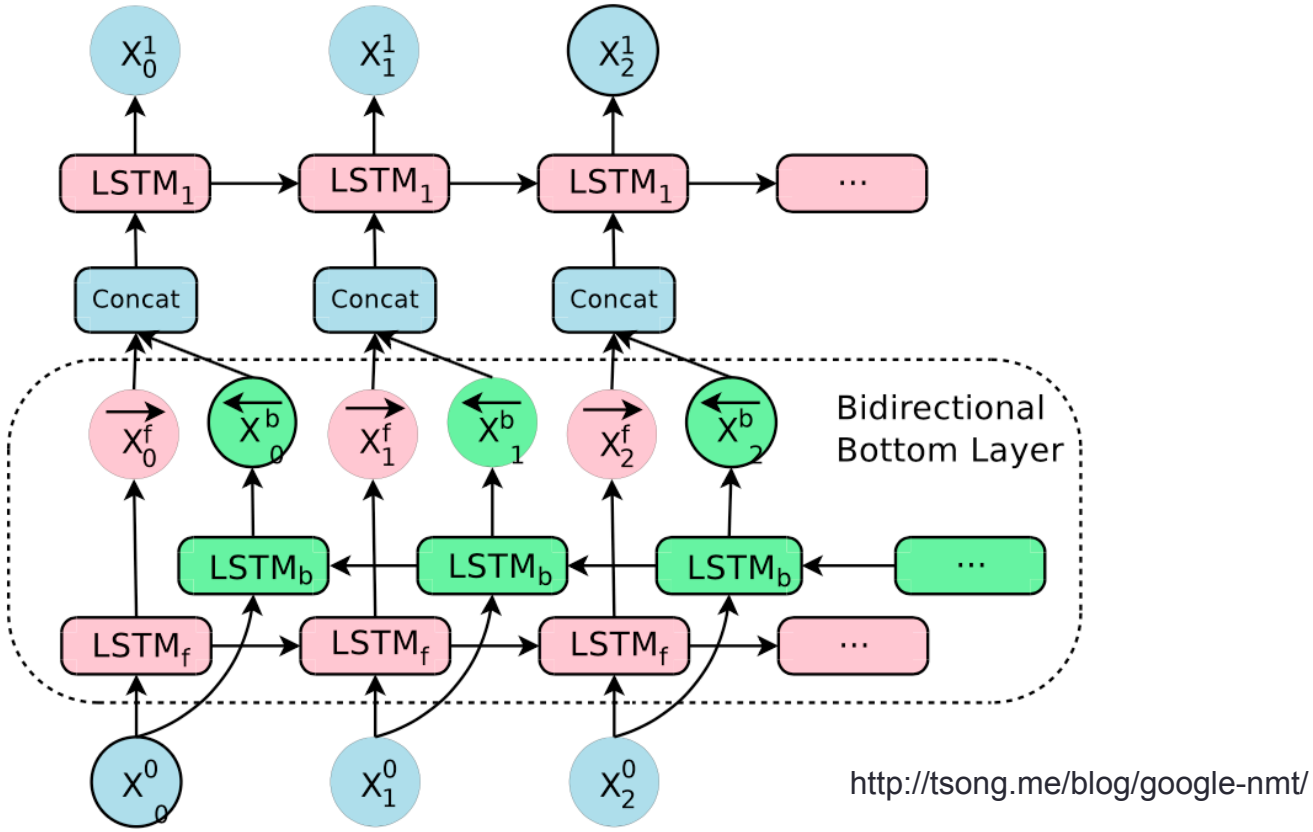
$$h_t^j = o_t^j \tanh\left(c_t^j\right)$$



Note V is diagonal (output of the cell is related to the memory of its own cell)

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



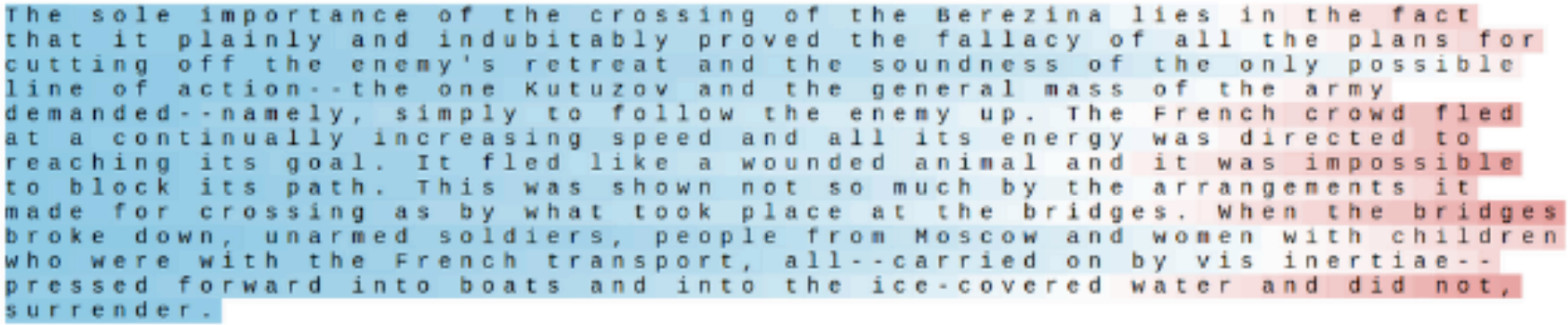
http://tsong.me/blog/google-nmt/

Real time bi-directional LSTM

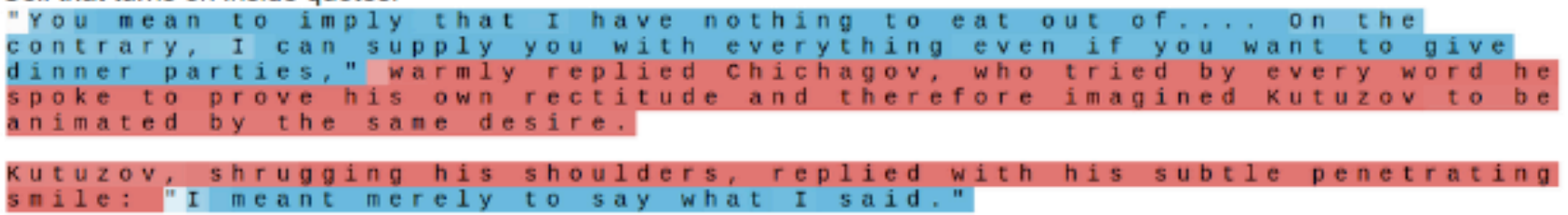
- For real time applications, only “look ahead” a certain amount of time steps
- Still helpful

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

DNN Legos

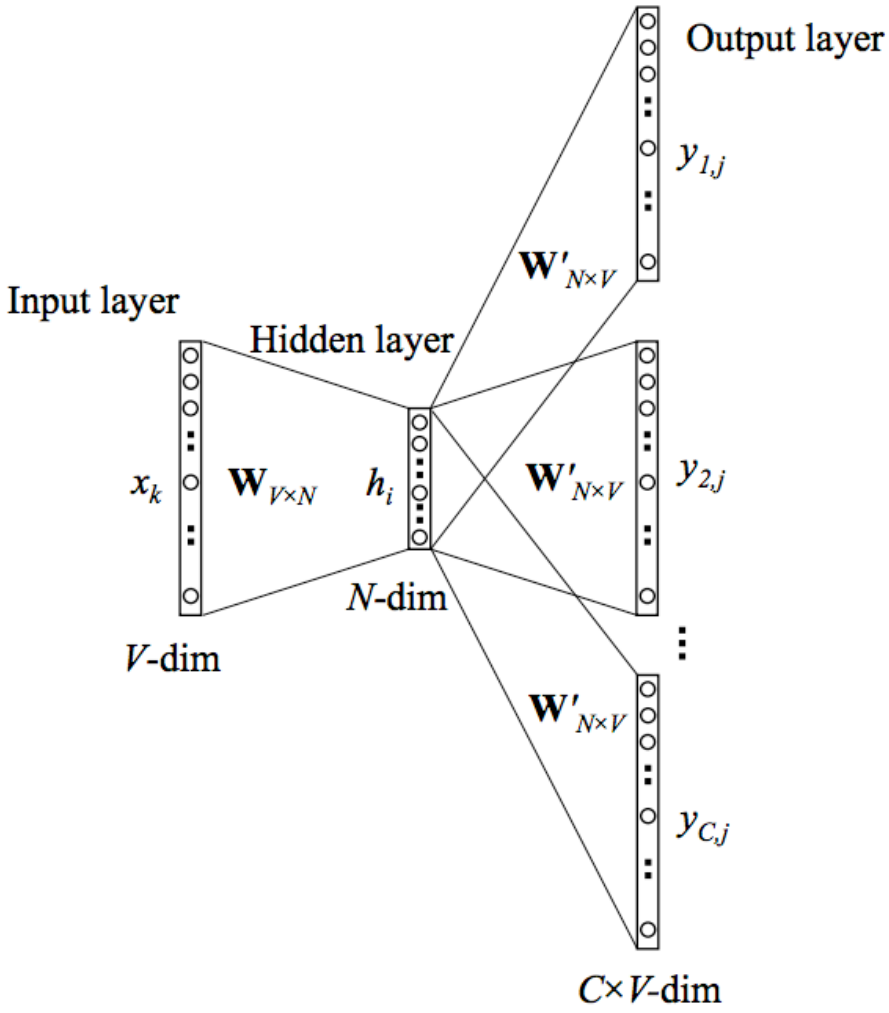
- Typical models now consists of all 3 types
  - CNN: local structure in the feature. Used for feature learning.
  - LSTM: remembering longer term structure or across time
  - DNN: Good for mapping features for classification. Usually used in final layers

Encoder-decoder

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

Word2Vec

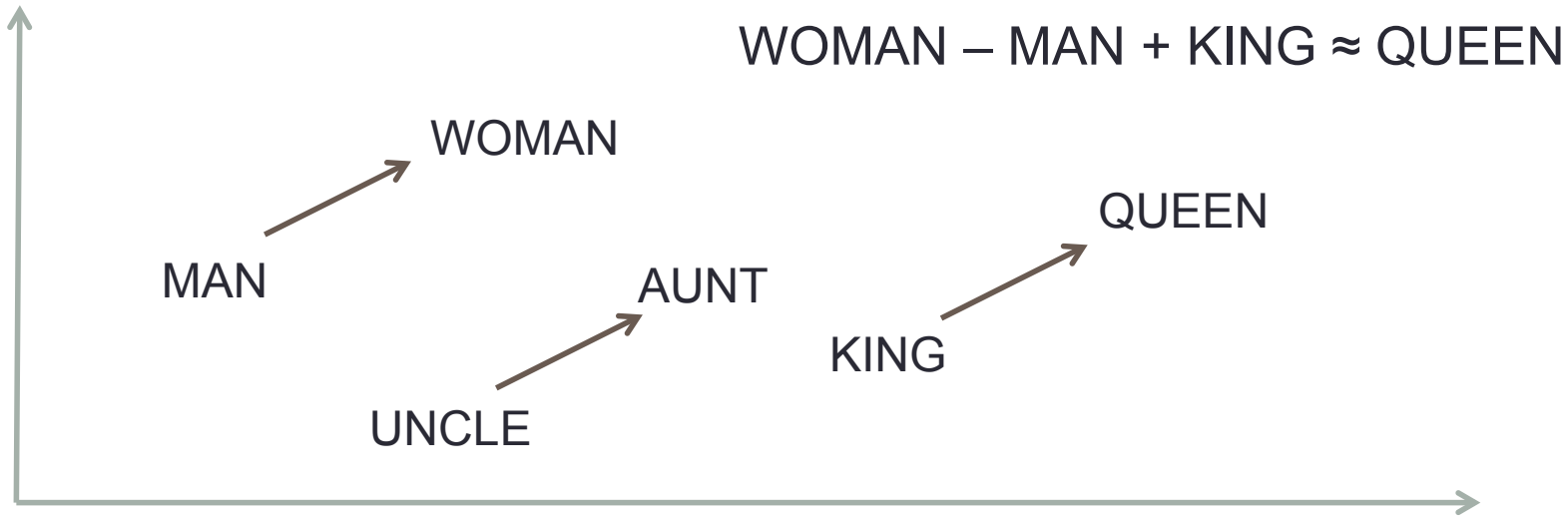
- Maps a word to a vector representation using neural networks



Mikolov, Linguistic Regularities in Continuous Space Word Representations, 2013

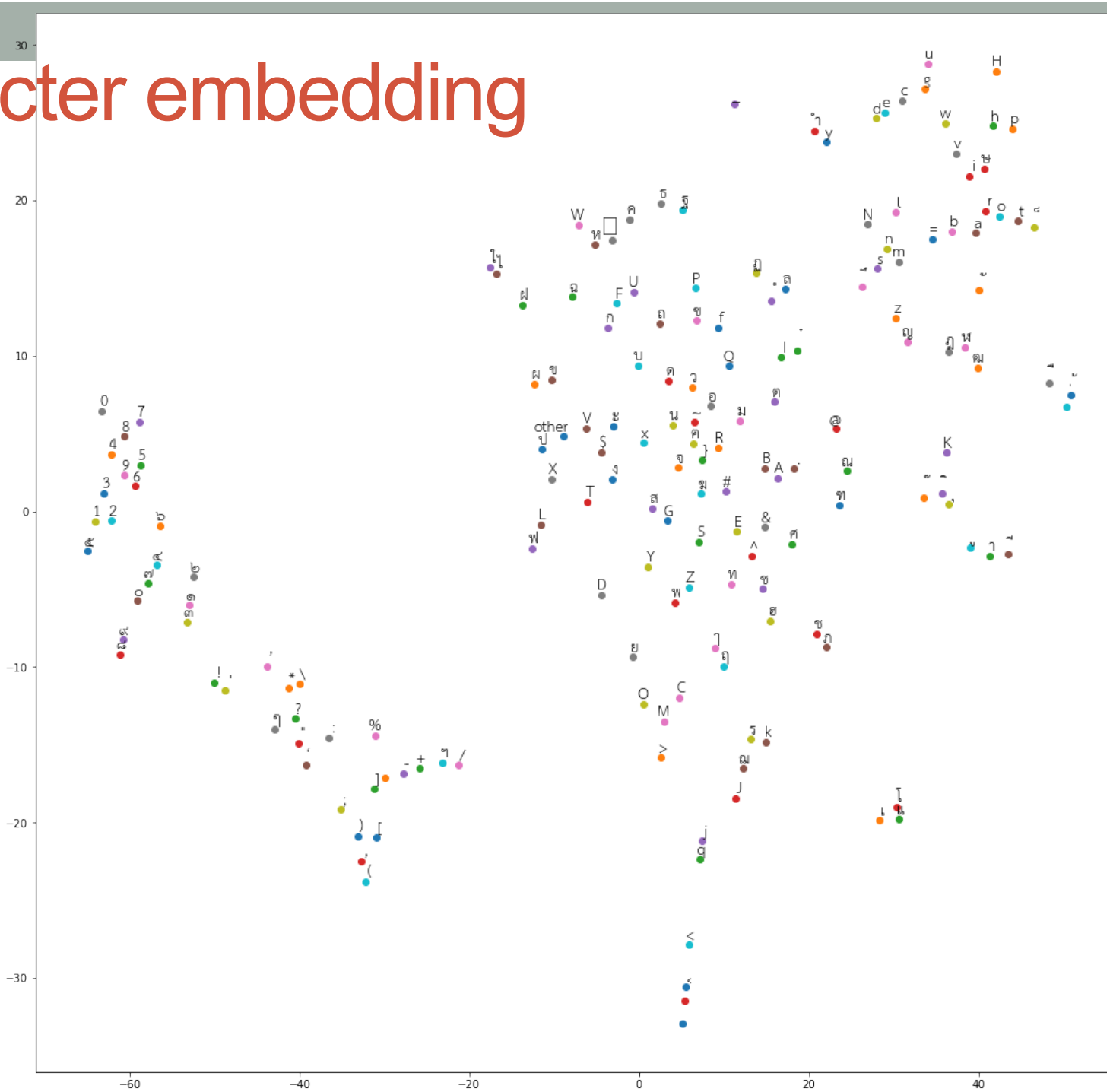
Word2Vec

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



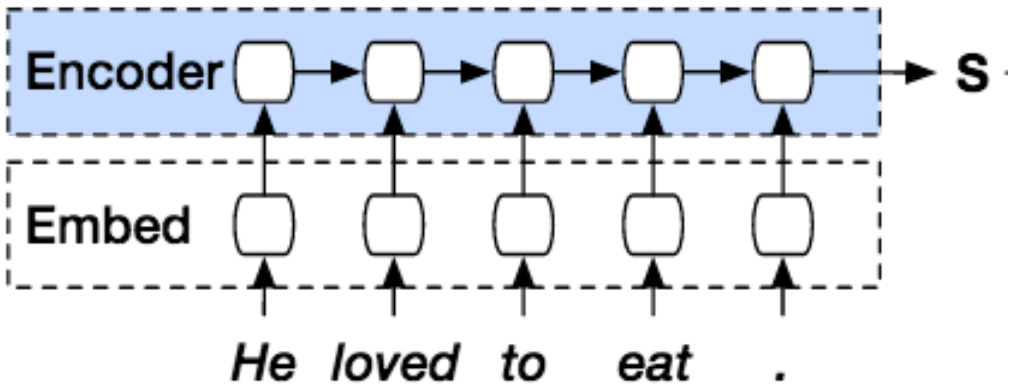
Mikolov, Linguistic Regularities in Continuous Space Word Representations, 2013

Thai character embedding



Sentence encoder

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

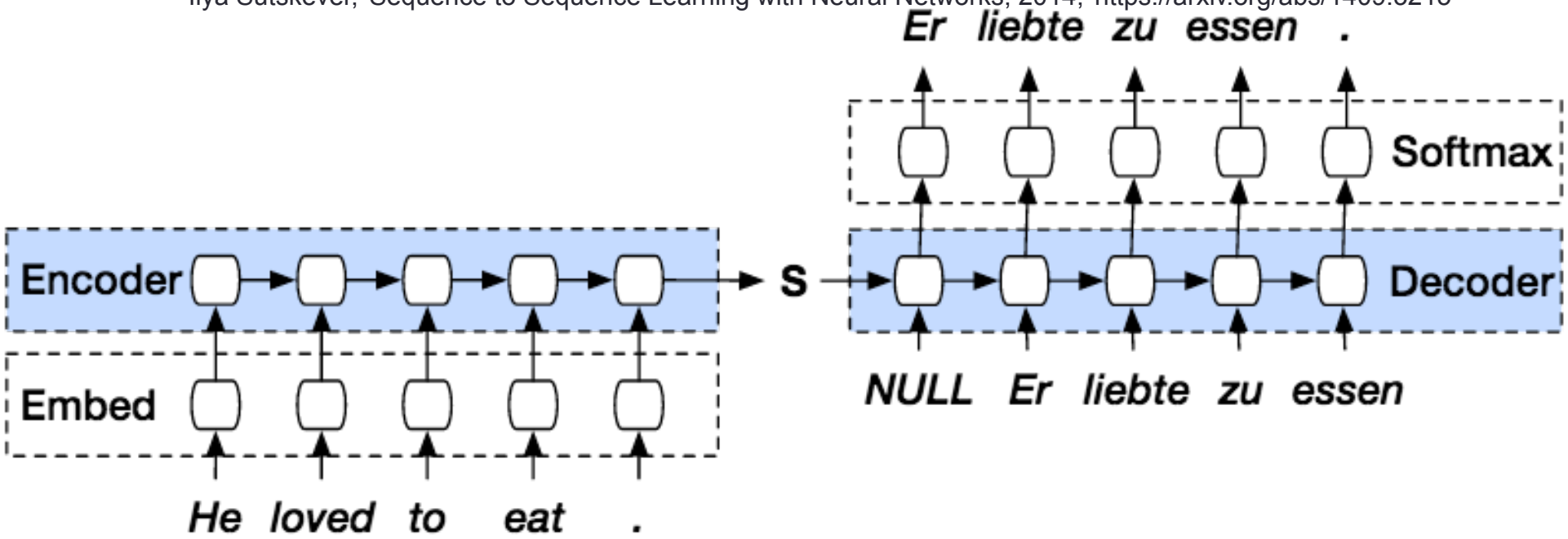


https://smerity.com/articles/2016/google_nmt_arch.html

Machine translation using encoder-decoder

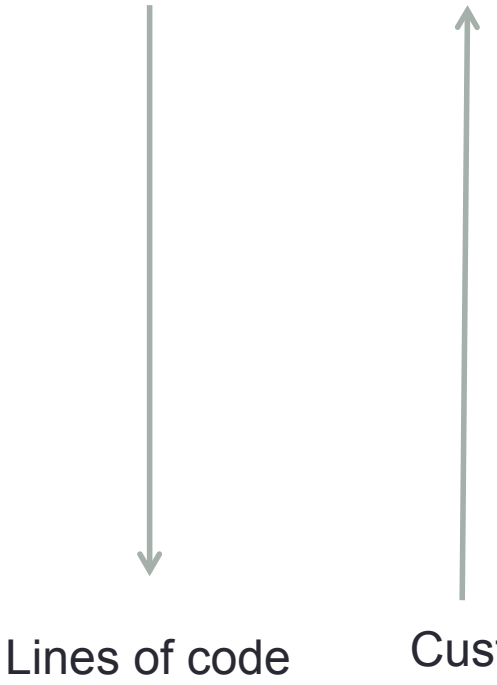
- Use another LSTM as the decoder
- Input to the LSTM is the encoded sentence and the previously output word

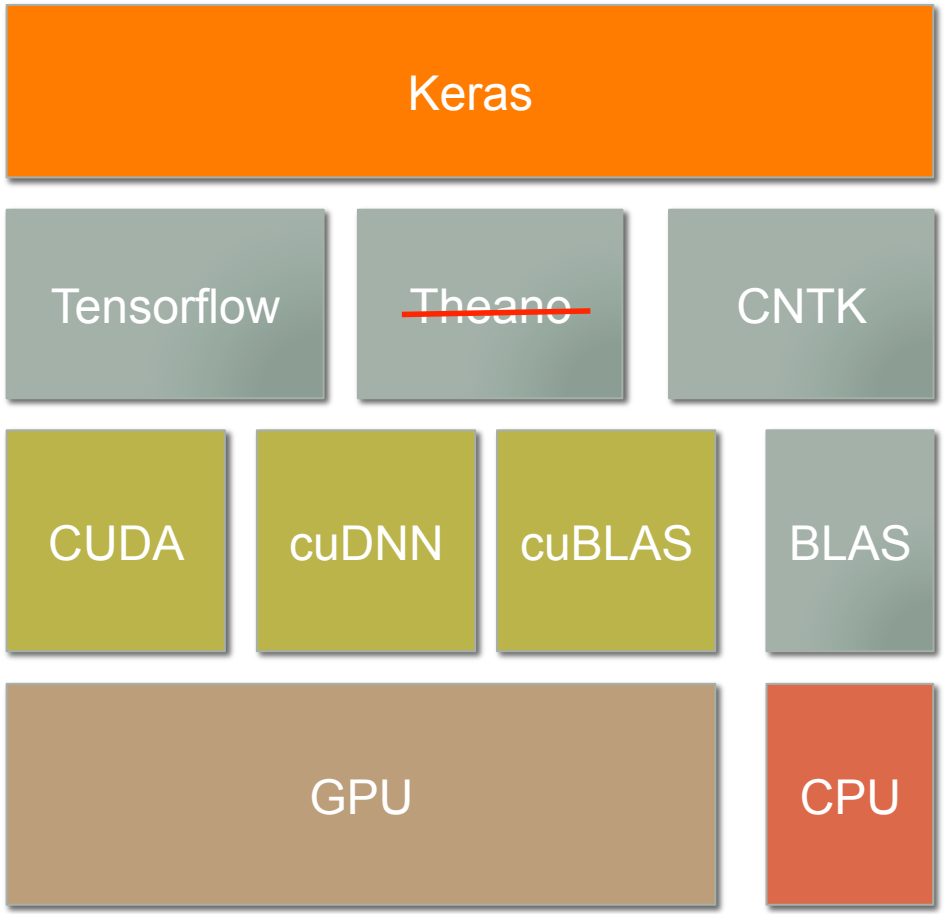
Ilya Sutskever, Sequence to Sequence Learning with Neural Networks, 2014, https://arxiv.org/abs/1409.3215



What toolkit

- Tensorflow – lower level
- Keras – higher level





My machine learning course

- https://www.youtube.com/playlist?list=PLcBOyD1N1T-OQd0a6mqjY6gWOuIl_stuv&disable_polymer=true

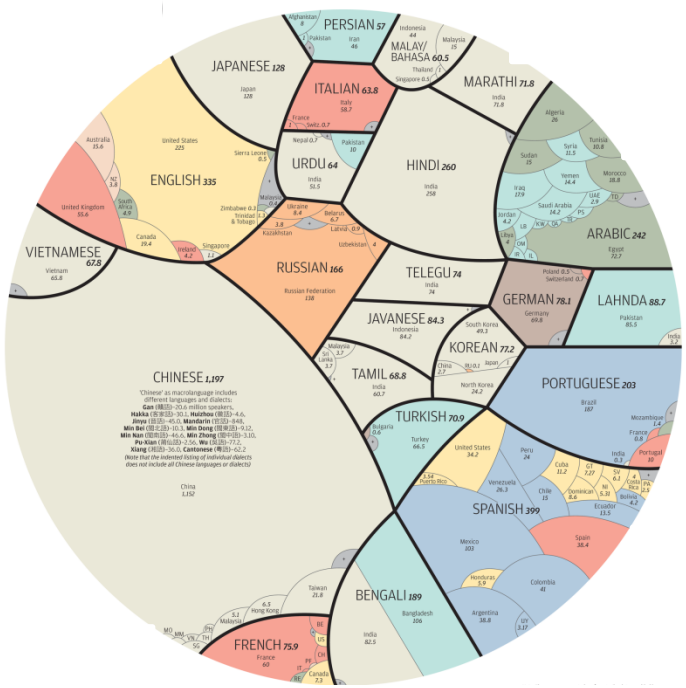
Lab

- https://github.com/ekapolc/cattern/tree/master/TUMKUD

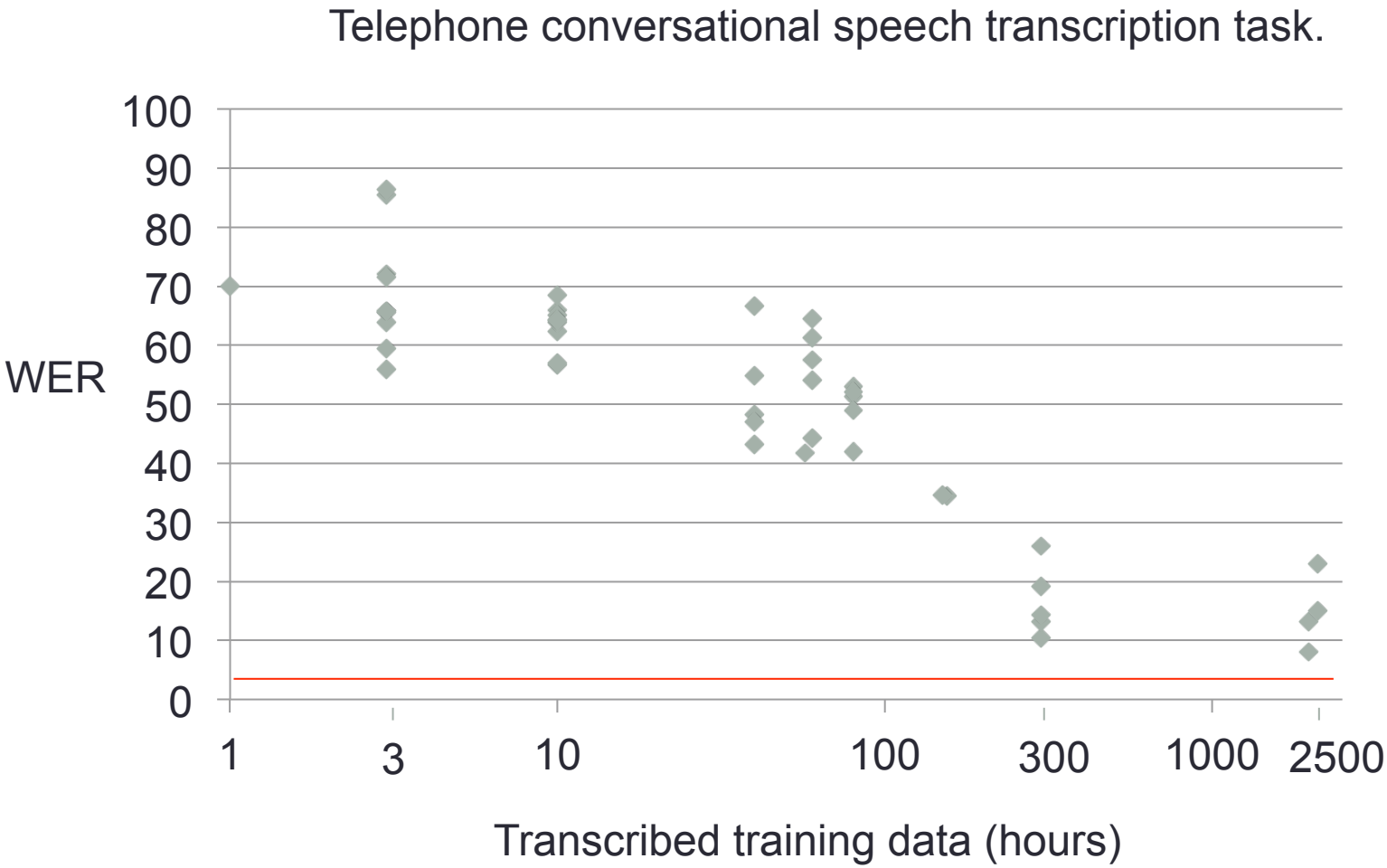
NEURAL NETWORKS TRANSFER LEARNING AND ASR

The lack of ASR capabilities

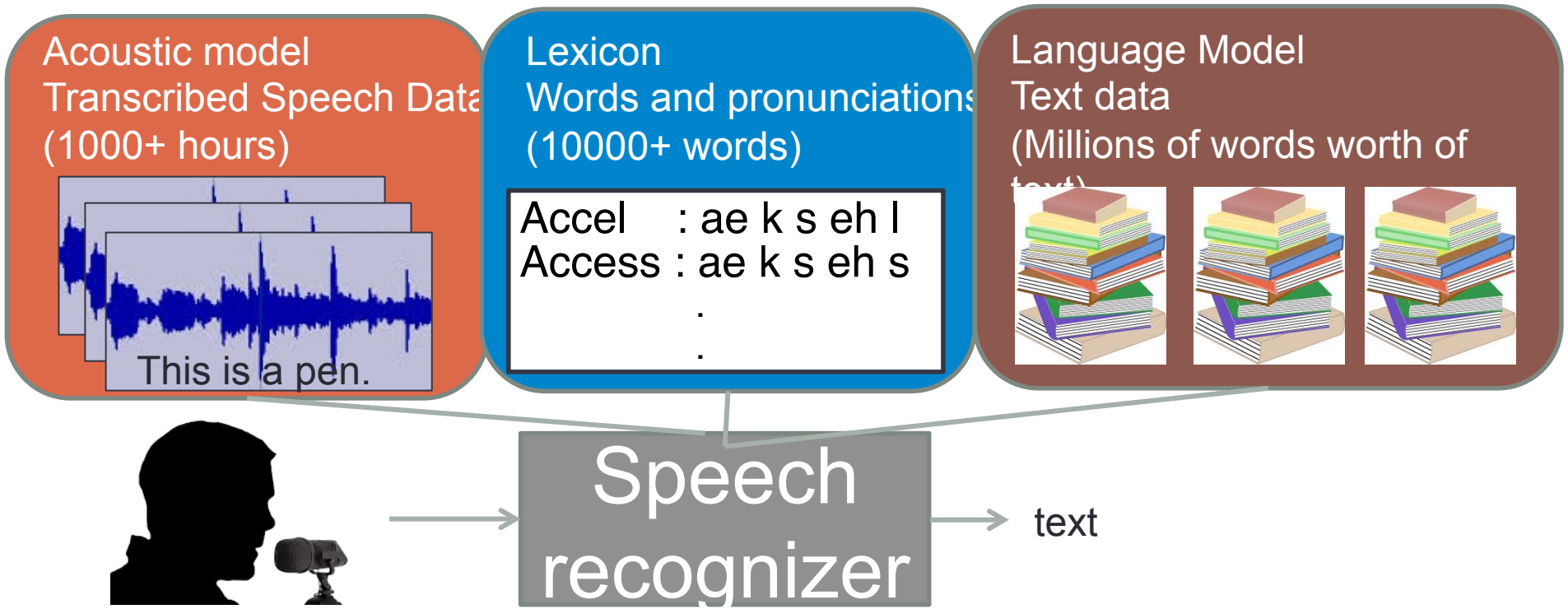
- Over 7000 languages over the world
  - ~400 with more than 1 million speakers
  - YouTube offers automatic captioning for 10 languages
  - Google speech API currently supports ~80 languages



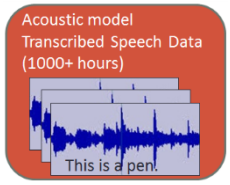
Effect of acoustic training data on ASR performance



The speech recognizer



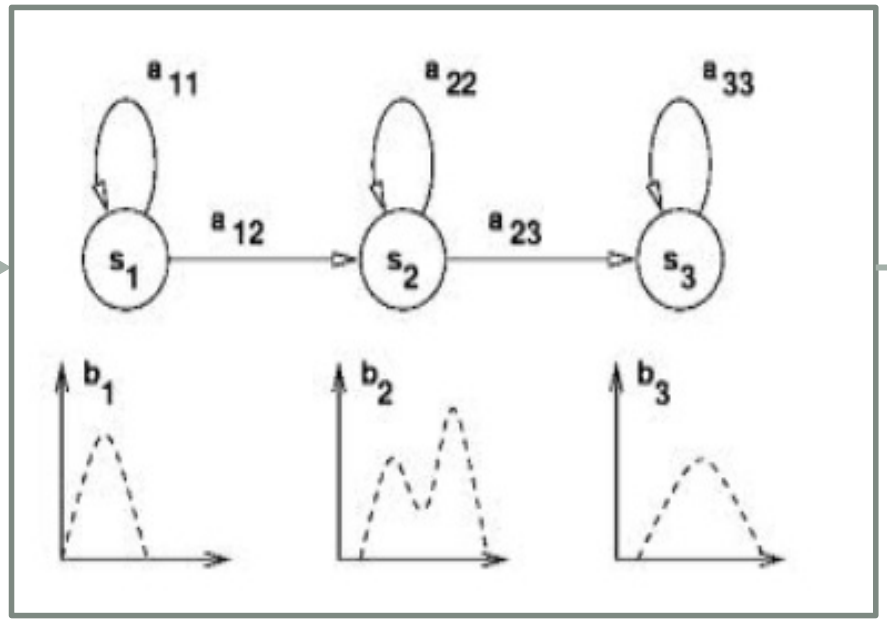
The Acoustic Model



PXS

Feature extraction

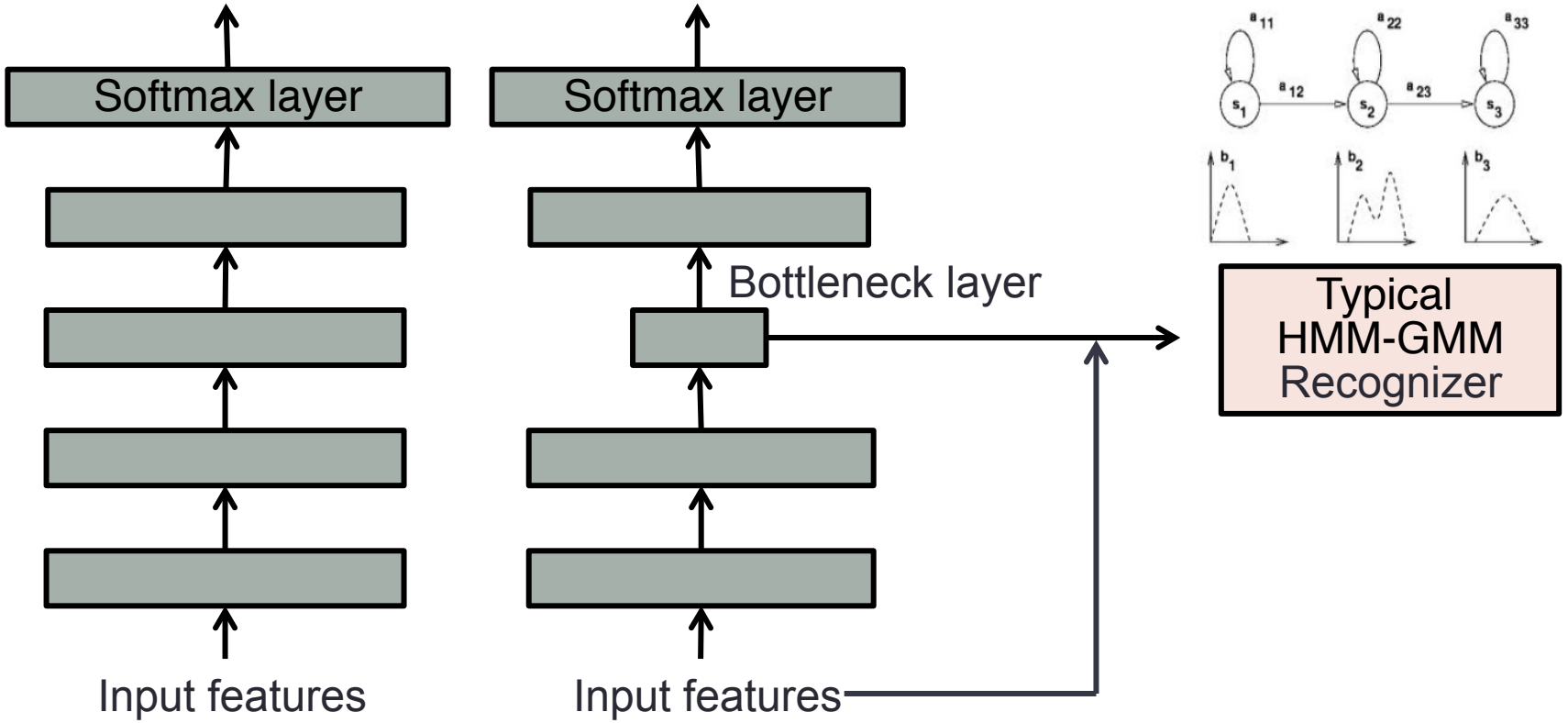
X



PXS

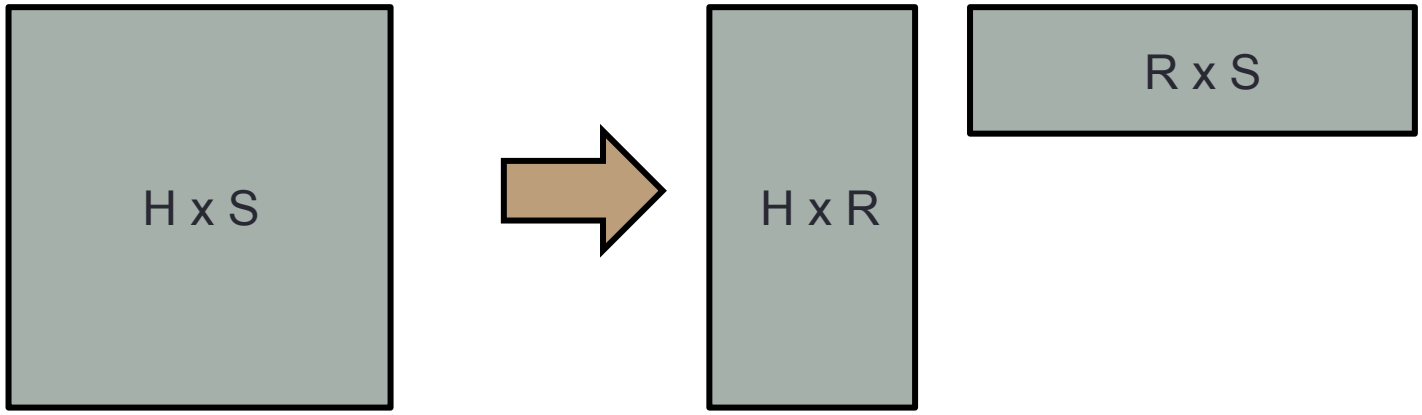
Tandem approach

- Use the DNN to generate good features to feed into the general GMM-HMM framework.
- Typically done by placing a narrow hidden layer in the network.



Low-rank matrix approximation

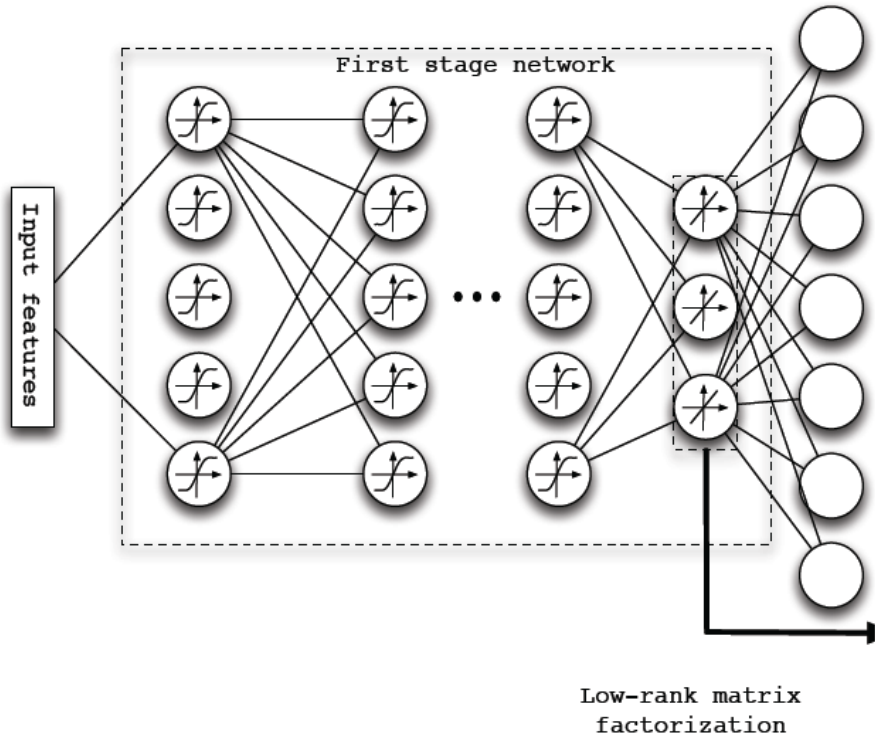
- Sainath et al. [2012] shows that the softmax layer of the DNN exhibits low-rank properties.
  - The softmax weights can be approximated by two matrixes of lower rank R



  - Reduces the amount of parameters from H x S to R x (H+S).
  - Depending on the value of R, the performance can remain the same.

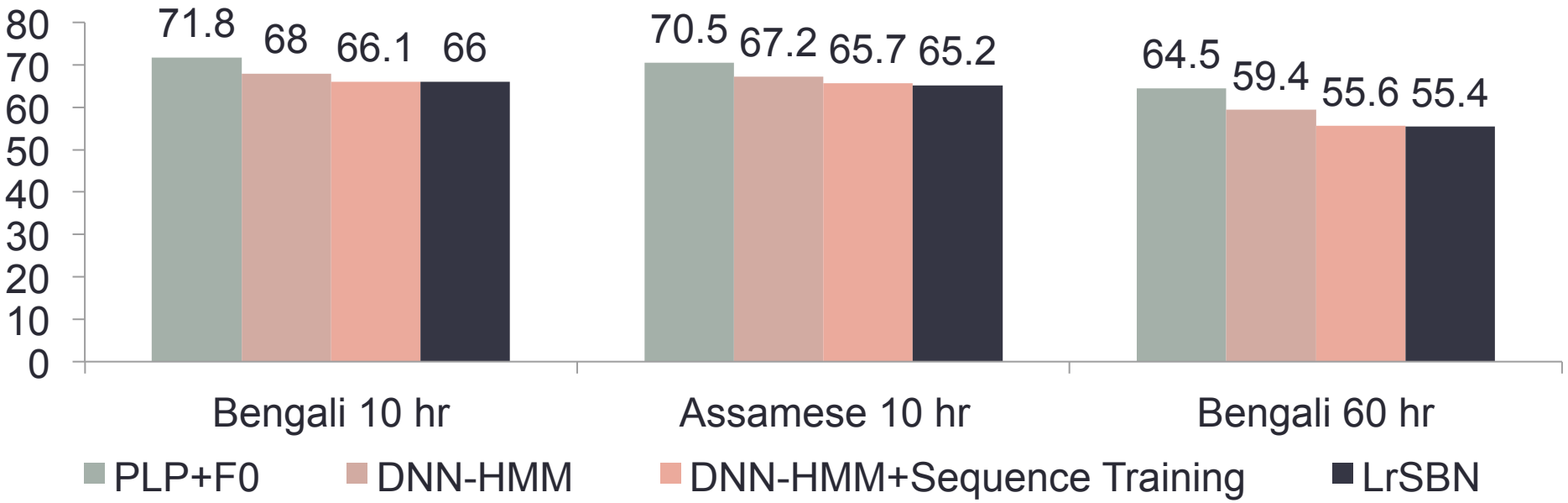
Low-rank Stacked Bottleneck Features (LrSBN)

- To incorporate longer context information we cascade two DNNs.



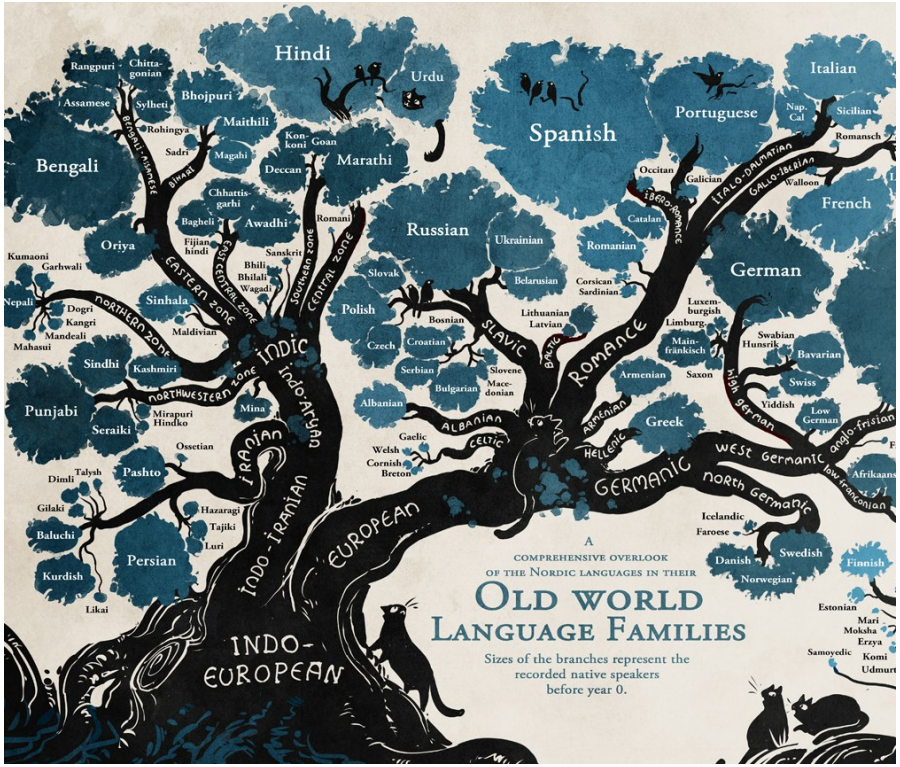
LrSBN results

- LrSBN models outperform hybrid DNN-HMM models on all tasks.



Leveraging multilingual and out of domain data

- We have existing resources from other domains or languages
- Languages are related
  - Sounds produced by human
  - Languages share phonemes
- Knowledge about channels, recording characteristics, and noise can be useful
- BN features offer a great way to share this information



Previous works on multilingual ASR

- Pool resources to train a recognizer
  - Experiments are limited in either the variability of the languages or the amount of data per language
  - Most use IPA mappings. Does not work that well and requires linguistic knowledge
- Which are the best resources to use?

A case study in Assamese and Bengali

- Both spoken in India
- Same writing script, with some shared vocabulary
- Bengali : 33 consonants 21 vowels
- Assamese : 30 consonants 20 vowels
- 24-28 shared consonants 15 shared vowels

Bengali

Assamese

Transfer learning using bottleneck features

- BN features learns from DNN offers somewhat language independent features
- Use the DNN learned on Assamese to extract features for Bengali. Train GMM-HMM on Bengali data only.

Bengali	WER
PLP+F0 (10 hr Bengali)	71.8%
PLP+F0 (60 hr Bengali)	64.5%
LrSBN (10 hr Assamese -> 10 hr Bengali)	66.0%
LrSBN (60 hr Assamese -> 10 hr Bengali)	64.6%
+ Adaptation	63.7%
LrSBN (60 hr Bengali -> 10 hr Bengali)	61.6%

Results on more language pairs

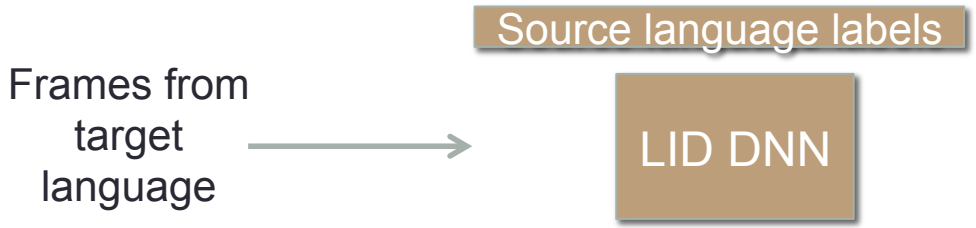
- Learn LrSBN on 60 hours of data, and apply on languages with 10 hours of data
- Numbers in blue is for 10->10 hours (baseline)

		Learn LrSBN on			
		Bengali	Assamese	Lao	Turkish
Use LrSBN on	Bengali	66.0	63.8	65.1	64.2
	Assamese	61.2	65.2	62.9	62.1
	Lao	59.8	60.1	62.3	60.0
	Turkish	61.8	63.1	63.3	63.9

- Transfer learning always improves performance.
- Similar languages perform better on transfer learning
- Can we identify this automatically?

Language Identification for data selection

- Train a classifier (LID) on source languages to predict the language given input frames
- Compute posteriors of the target language data using that classifier
- The best language for the target language should have the highest average posterior score

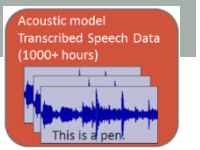


Predicting the best language

		Learn LrSBN on			
		Bengali	Assamese	Lao	Turkish
Use LrSBN on	Bengali	66.0	63.8	65.1	64.2
	Assamese	61.2	65.2	62.9	62.1
	Lao	59.8	60.1	62.3	60.0
	Turkish	61.8	63.1	63.3	63.9

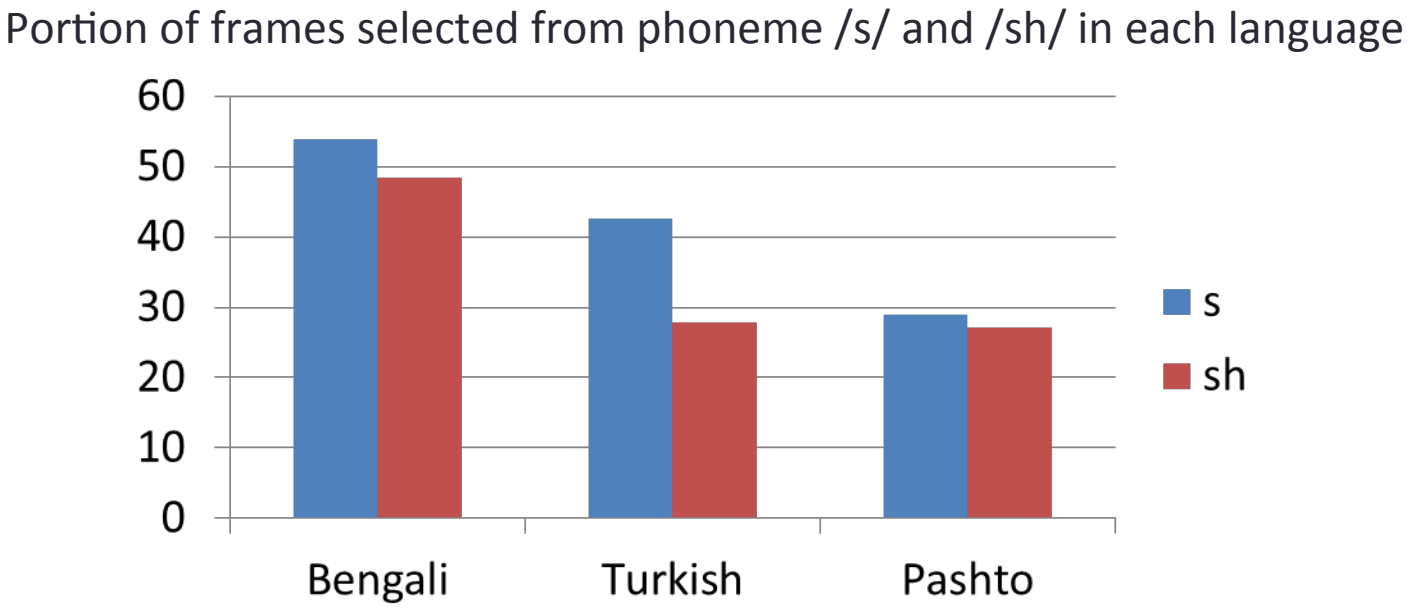
		Averaged predicted posterior			
		Bengali	Assamese	Lao	Turkish
Input frames	Bengali	0.57	0.21	0.09	0.13
	Assamese	0.21	0.57	0.11	0.11
	Lao	0.08	0.11	0.71	0.10
	Turkish	0.13	0.12	0.10	0.65

The LID scores correspond to the best language to use most of the time.

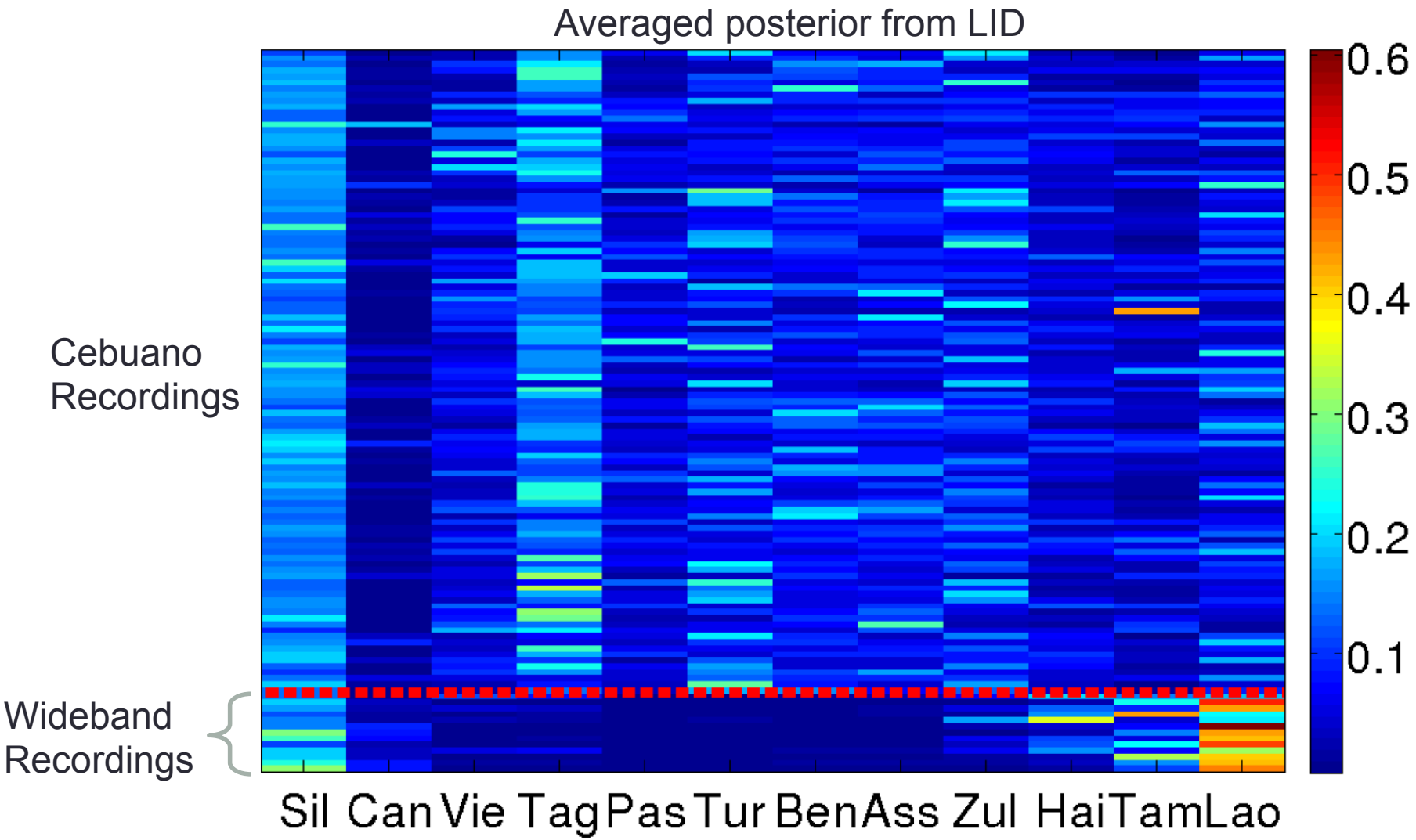


LID captures linguistic information

- Cebuano has /s/ but not /sh/
- High scoring frames for Cebuano favor /s/ over /sh/



LID also captures channel information



Data selection for transfer learning

- Selecting parts of many language vs selecting the best language

System (LrSBNs)	Cebuano (3hr)	Telugu (3hr)	Swahili (3hr)
	MTWV	MTWV	MTWV
All language (500 hr)	0.2259	0.1269	0.3983
Closest language (50 hr)	0.2526	0.1682	0.4225
Parts (100 hr)	0.2513	0.1711	**0.4244**
Parts (200 hr)	**0.2531**	**0.1756**	0.4233

Summary

- DNN can learn cross-language features for ASR
- Transfer learning can benefit from data selection

- Not from this talk
  - These features for ASR has been used successfully for other task
    - Language ID
    - Speaker ID