Data-driven Intelligent Systems

Lecture 12
Deep Neural Networks

KNOWLEDGE
TECHNOLOGY

http://www.informatik.uni-hamburg.de/WTM/

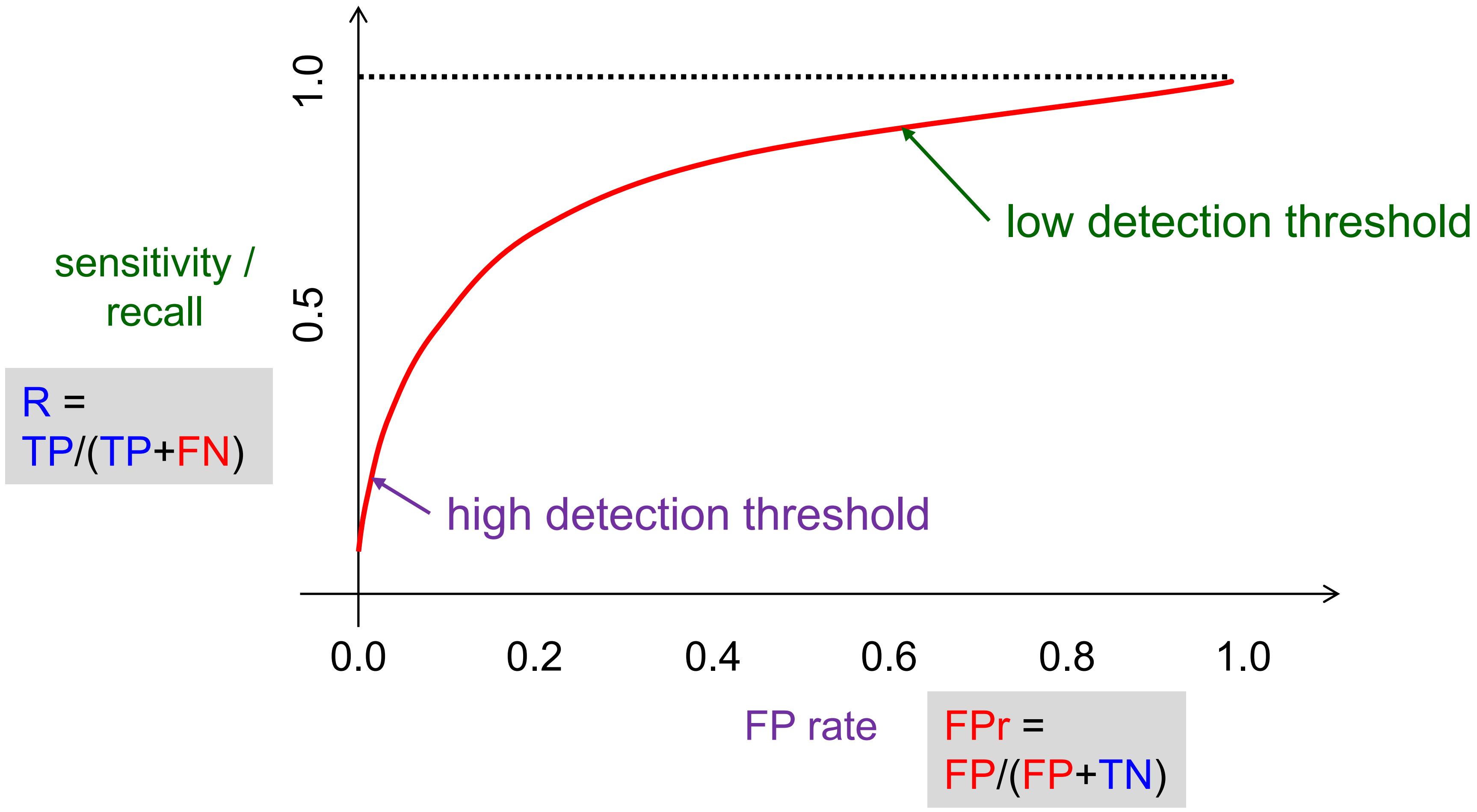
Outline

1. Evaluation of Classifiers: ROC Curves
2. Deep Neural Networks
   - Computational Graphs
   - Typical Deep NNs
   - Better Than Human Performance

Receiver Operating Characteristic (ROC)

- The confusion matrix quantifies the performance of only ***one*** trained binary classifier model
- Most models can be tuned to handle the tradeoff between
  - high sensitivity/recall, i.e. detecting most positives (this asks for a low detection threshold)
  - low false positive rate (~ high specificity and high precision), i.e. not detecting negatives as positive (this asks for a high detection threshold)
- A ROC curve shows overall model performance
  - It plots sensitivity over false positive rate for ***many threshold settings***

Receiver Operating Characteristic (ROC)



- Shown curve is an idealization
  - actual graph may not be continuous

How to Construct ROC Curve? (1)

- A model is often tunable to different thresholds
- ROC plots sensitivity and $\underbrace{\text{specificity}}_{1\text{-}FPr}$ for all possible thresholds

Extremes:

- If threshold = maximum
  - $FP = 0$ $(TP = 0)$
  - sensitivity = 0; FPr = 0
- If threshold = minimum
  - $FN = 0$ $(TN = 0)$
  - sensitivity = 1; FPr ≈ 1

	actual 1	actual 0
predicted 1	TP	FP
predicted 0	FN	TN

How to Construct ROC Curve? (2)

- Suppose we use a low threshold of ***0.5*** for our classifier…

	Actual Outcome	
Predicted Outcome	1	0
1	8	3
0	0	9

Sensitivity: 8/(8+0)

FP rate: 3/(3+9)

How to Construct ROC Curve? (3)

- Suppose we use a higher threshold of ***0.8***

	Actual Outcome	
Predicted Outcome	1	0
1	6	2
0	2	10

Sensitivity: 6/(6+2)

bad: lower sensitivity

FP rate: 2/(2+10)

good: lower false positive rate

How to Construct ROC Curve – Automatization

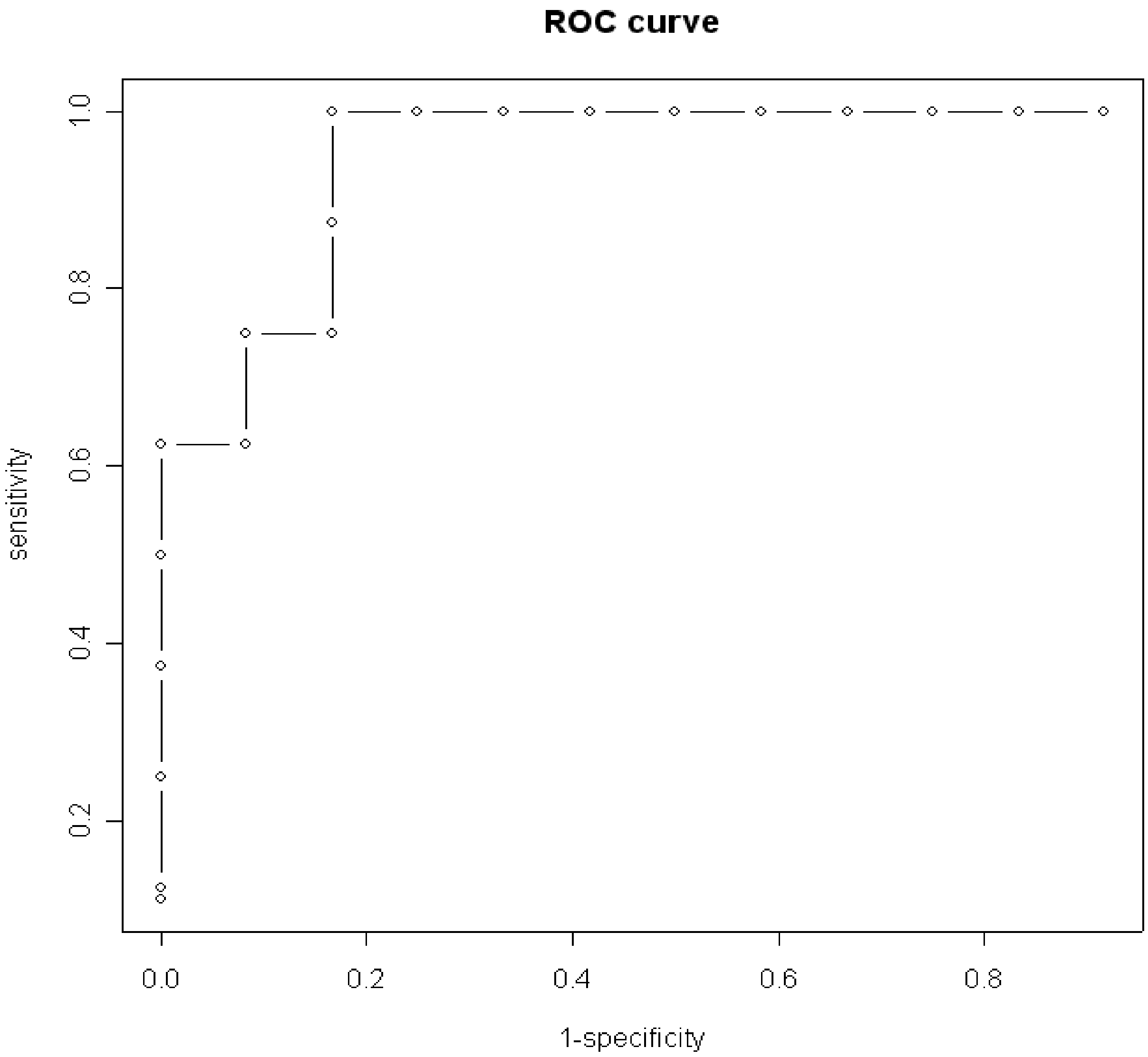
threshold

A1

	A	C	D	E	F	G	H	I
1			**a**	**b**	**c**	**d**	**sensitivity**	**specificity**
2	0	0.005694	8	11	0	1	1	0.083333
3	0	0.009911	8	10	0	2	1	0.166667
4	0	0.025475	8	9	0	3	1	0.25
5	0	0.039375	8	8	0	4	1	0.333333
6	0	0.070495	8	7	0	5	1	0.416667
7	0	0.080184	8	6	0	6	1	0.5
8	0	0.099051	8	5	0	7	1	0.583333
9	0	0.346722	8	4	0	8	1	0.666667
10	0	0.493576	8	3	0	9	1	0.75
11	0	0.635592	8	2	0	10	1	0.833333
12	1	0.705922	7	2	1	10	0.875	0.833333
13	1	0.753097	6	2	2	10	0.75	0.833333
14	0	0.88035	6	1	2	11	0.75	0.916667
15	1	0.92832	5	1	3	11	0.625	0.916667
16	0	0.970674	5	0	3	12	0.625	1
17	1	0.97985	4	0	4	12	0.5	1
18	1	0.983794	3	0	5	12	0.375	1
19	1	0.984132	2	0	6	12	0.25	1
20	1	0.99631	1	0	7	12	0.125	1
21	1	0.999876	1	0	8	12	0.111111	1
22								
23								

```
sens<-c(1,1,1,1,1,1,1,1,1,1,0.875,0.75,0.75,0.625,0.625,0.5,0.375,0.25,0.125,0.11111
spec<-c(0.083333333,0.166666667,0.25,0.333333333,0.416666667,0.5,0.583333333,0.66666
33333,0.916666667,0.916666667,1,1,1,1,1,1)
plot(1-spec,sens,type="b",xlab="1-specificity",ylab="sensitivity",main="ROC curve")
```

How to Construct ROC Curve – Automatization



"Area under the ROC curve" is a common measure of predictive performance.

ROC (Receiver Operating Characteristic)

- ***ROC Space:*** Each classifier is represented by plotting its (FP rate, TP rate) pair

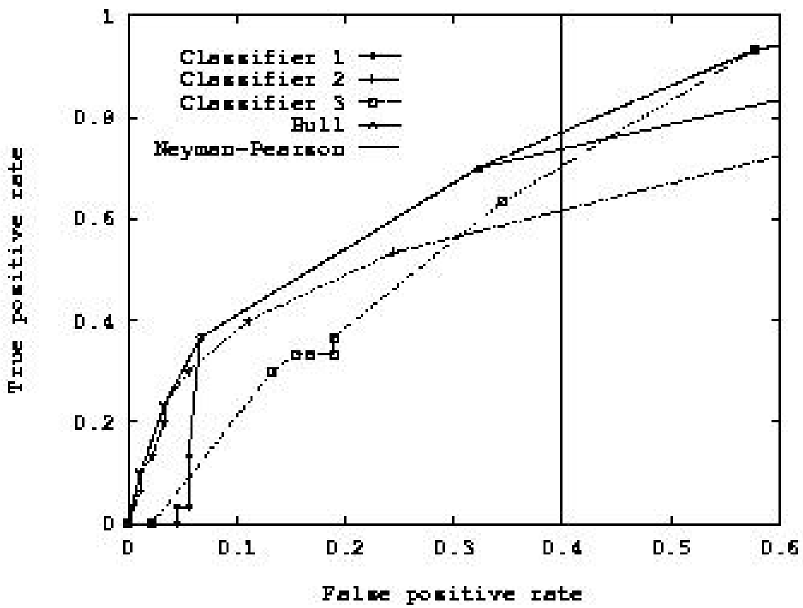


Figure 4: The ROC Convex Hull used to select a classifier under the Neyman-Pearson criterion

- ***Good model:*** maximizing AUC (Area Under Curve)
- ***Interpolation:*** a good model extends the ROC Convex Hull.

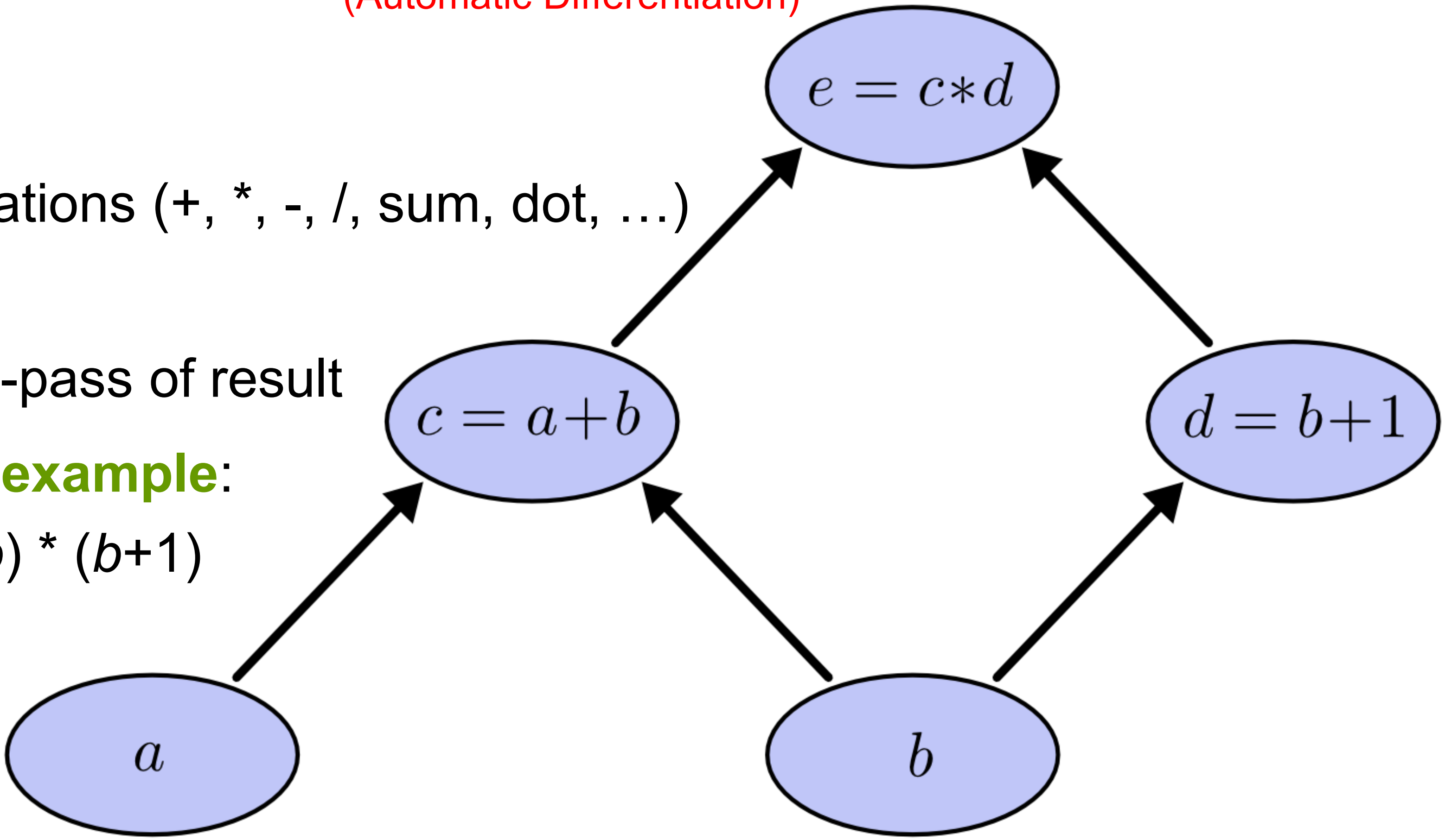
Outline

1. Evaluation of Classifiers: ROC Curves
2. Deep Neural Networks
   - Computational Graphs
   - Typical Deep NNs
   - Better Than Human Performance

Computational Graphs

(Automatic Differentiation)

- **Nodes**:
  Computations (+, *, -, /, sum, dot, …)
- **Edges**:
  Forward-pass of result
- **Shown example**:
  $e = (a+b) * (b+1)$



http://colah.github.io/posts/2015-08-Backprop

Computational Graphs

(Automatic Differentiation)

- **Forward pass**: Set input values

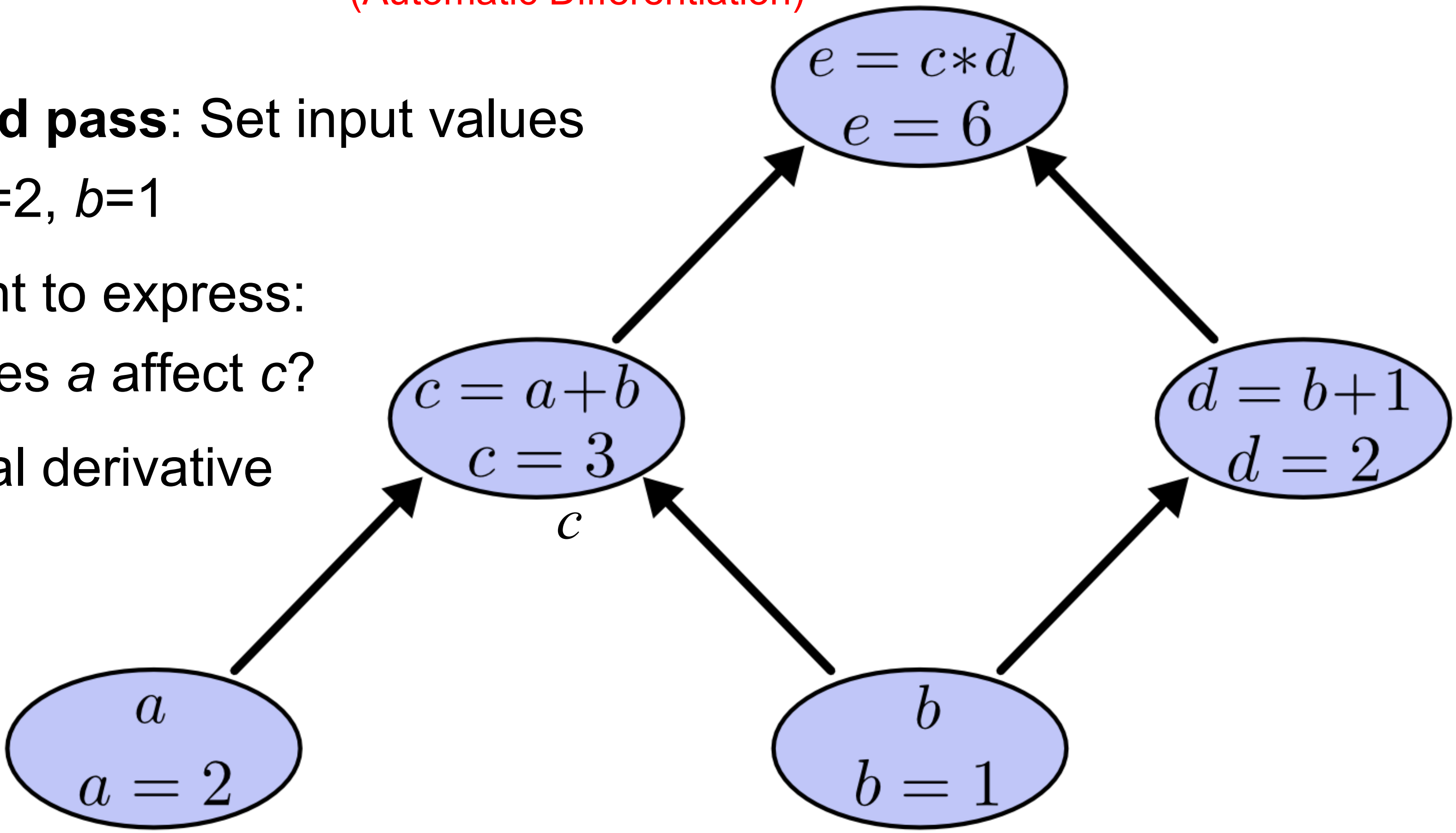
  e.g.: *a*=2, *b*=1

- We want to express:

  How does *a* affect *c*?

  → partial derivative

  $$\frac{\partial c}{\partial a}$$



With sum rule:

$$\frac{\partial c}{\partial a} = \frac{\partial}{\partial a}(a+b) = \frac{\partial a}{\partial a} + \frac{\partial b}{\partial a} = 1 + 0$$

Computational Graphs

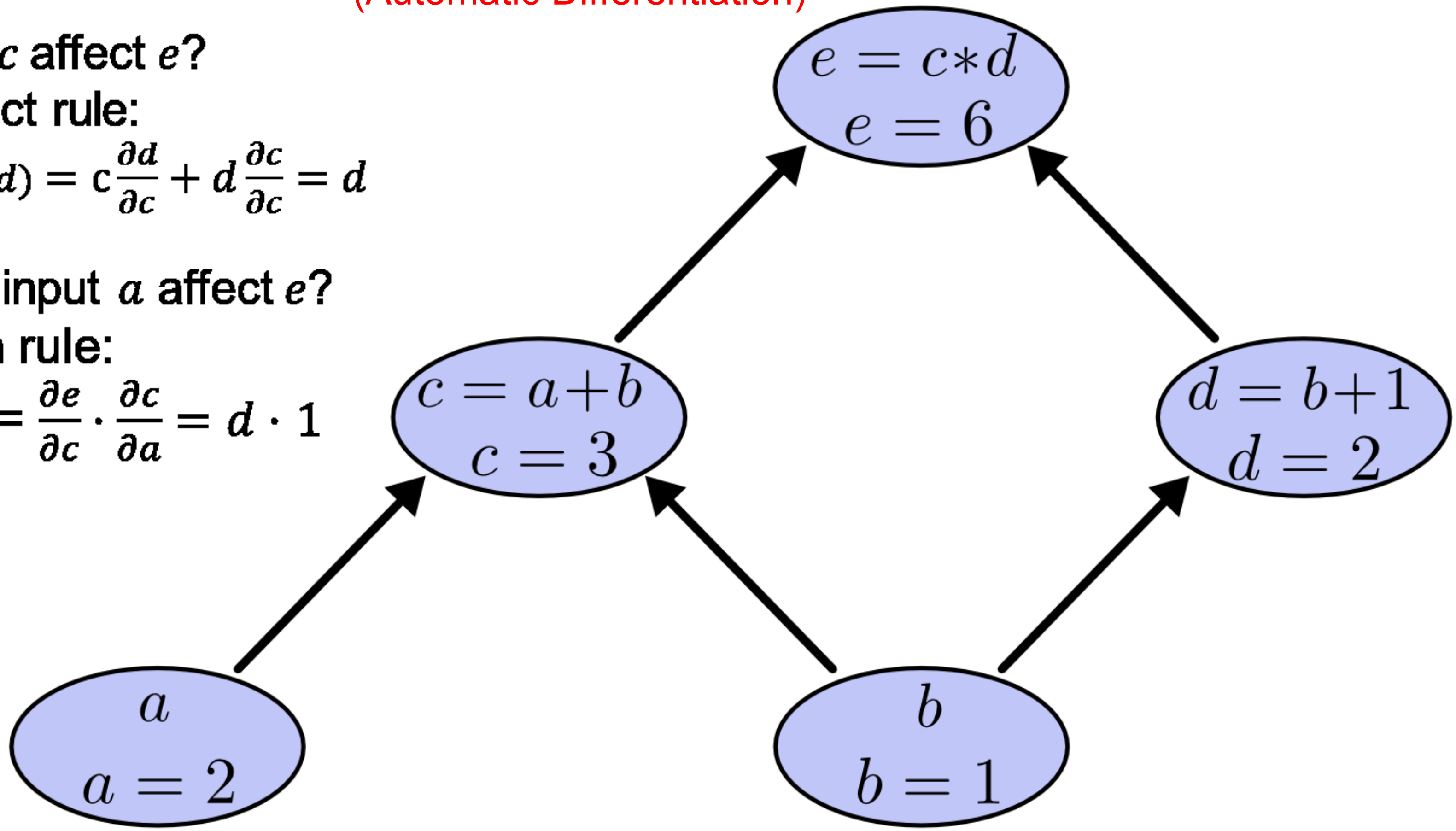
(Automatic Differentiation)

- How does c affect e?
- With product rule:

$$\frac{\partial e}{\partial c} = \frac{\partial}{\partial c}(c \cdot d) = c\frac{\partial d}{\partial c} + d\frac{\partial c}{\partial c} = d$$

- How does input a affect e?
- With chain rule:

$$\frac{\partial}{\partial a}e(c(a)) = \frac{\partial e}{\partial c} \cdot \frac{\partial c}{\partial a} = d \cdot 1$$

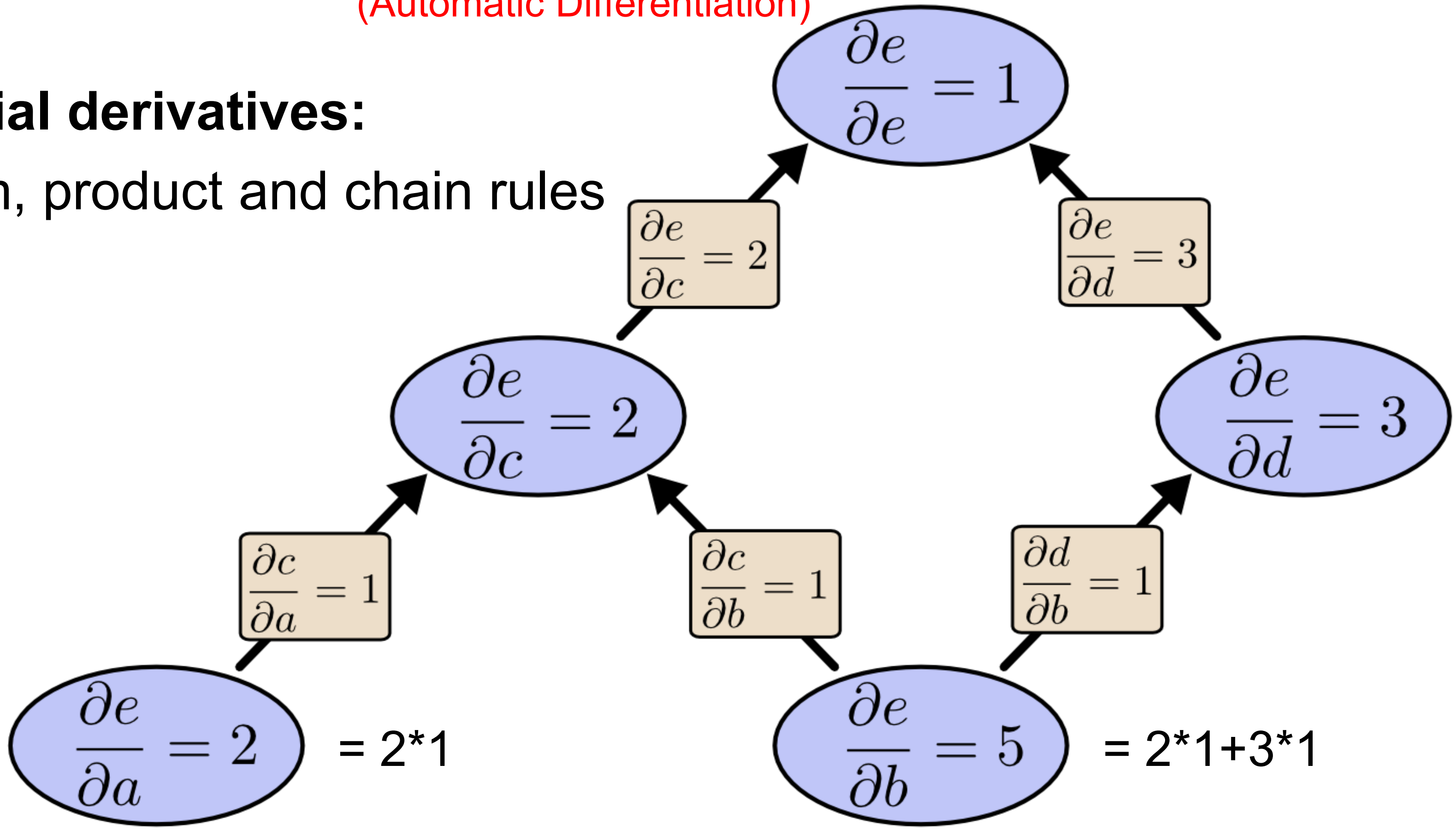


Computational Graphs

(Automatic Differentiation)

- **All partial derivatives:**

  Use sum, product and chain rules



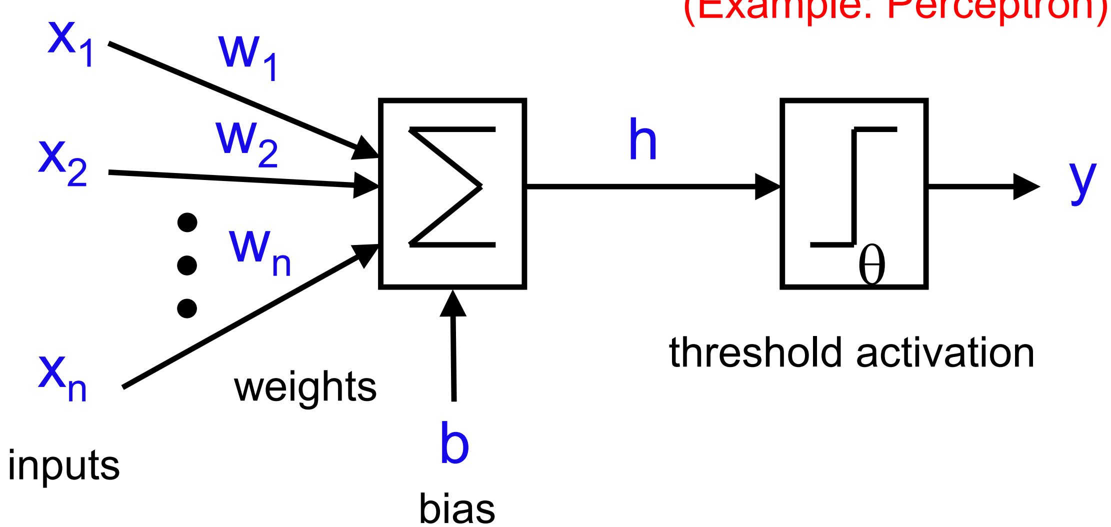
Reverse-mode differentiation:

Factor the paths backwards starting at *e*.

Gives derivative of *e* with respect to every node!

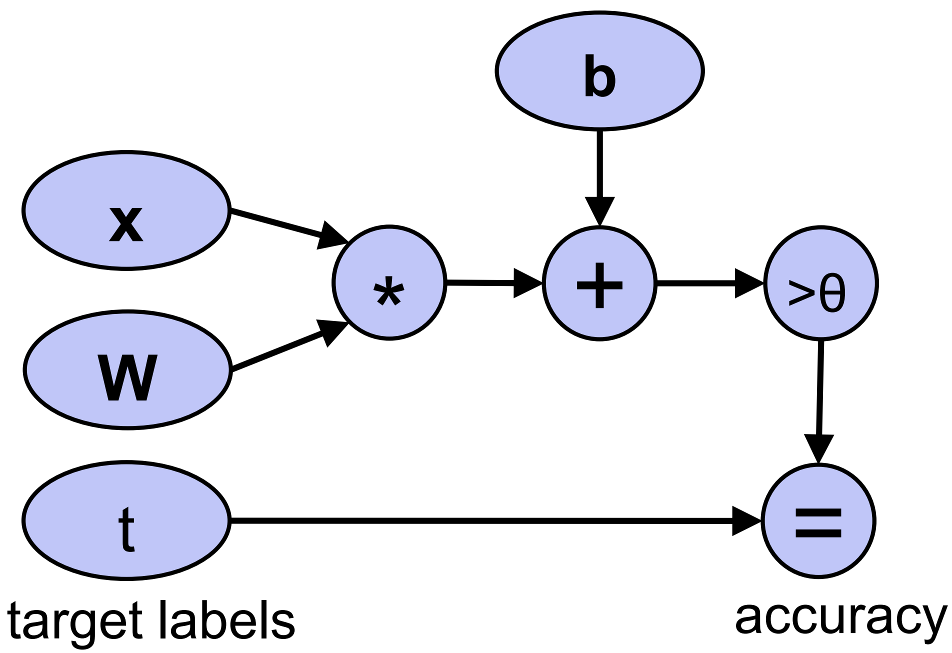
Computational Graphs

(Example: Perceptron)



$$h = \sum_i x_i w_i + b$$

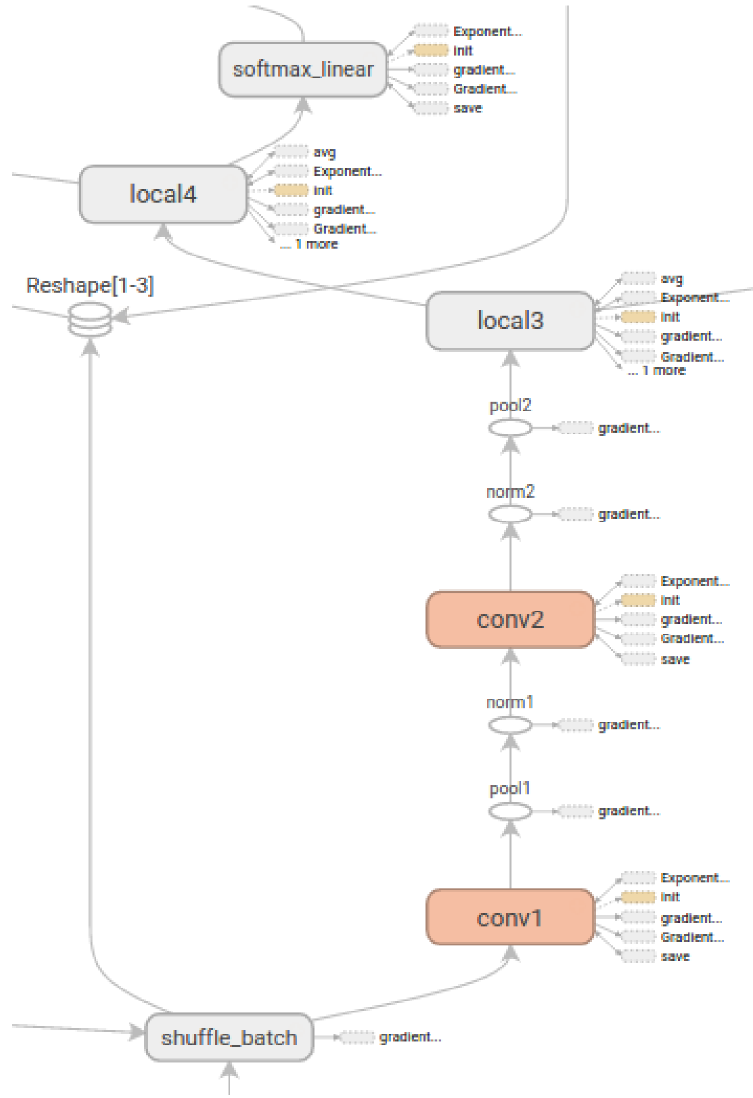
$$y = \begin{cases} 1 \text{ if } & h \geq \theta \\ 0 \text{ otherwise} \end{cases}$$



Computational Graphs

Why use computational graphs?

- Gradients “for free“
- Easy parallelization: Computation naturally segmented
- Computational model (graph) close to conceptual model (neural network)



Challenge:

- Symbolic programming requires rethinking

Deep Learning Frameworks

Advantages:

- Easily build big computational graphs
- Automatically compute all gradients
- Use GPU

Examples:

- Tensorflow + high-level wrapper: Keras
- PyTorch

Tensorflow Example Code (Trivial)

```
import tensorflow as tf
d = 2.0
x = tf.placeholder(tf.float32)              # target for feed, below
w = tf.constant(3.0, dtype=tf.float32)
z = tf.add(x, w)                            # create computational graph
sess = tf.Session()                         # launch graph
result = sess.run(z, feed_dict={x: d})      # run (part of) graph
print(result)
```

Tensorflow Example Code (with Optimization)

```
import tensorflow as tf
d = [2.0]
x = tf.placeholder(tf.float32, shape=[1])
w = tf.Variable(tf.random_normal([1]))          # variable to be optimized
z = tf.add(w,x)                                  # simple graph: z = w+x
cost = tf.reduce_mean(tf.nn.l2_loss([5.0]-z))    # minimize (5-z)^2
optimizer = \
  tf.train.GradientDescentOptimizer(learning_rate=0.3).minimize(cost)
sess = tf.Session()                              # launch graph
sess.run(tf.global_variables_initializer())      # initialize the variable
for epoch in range(20):
    sess.run(optimizer, feed_dict={x: d})        # use autograd
    cost_c = sess.run(cost, feed_dict={x: d})    # get current loss
    w_c = sess.run(w)                            # get current variable
    print("cost=", "{:.9f}".format(cost_c), “w_c=", "{:.3f}".format(w_c[0]))
```

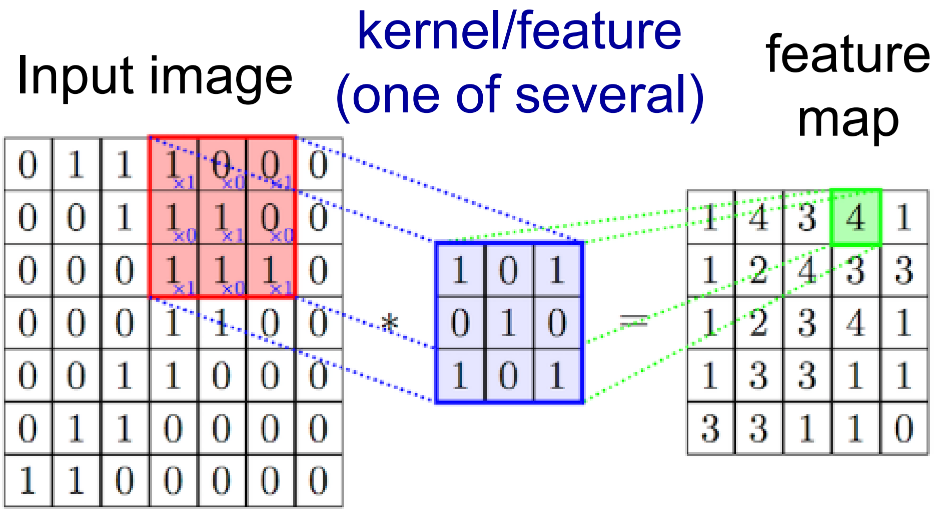
PyTorch Example Code (with Optimization)

```
import torch
from torch.autograd import Variable
x = Variable(2.0 * torch.ones(1).type(torch.FloatTensor), \
      requires_grad=False)
w = Variable(torch.randn(1).type(torch.FloatTensor), \
      requires_grad=True)                        # variable to be optimized
learning_rate=0.3
for epoch in range(20):
    z = x + w                                    # simple graph: z = w+x
    loss = (5.0 - z).pow(2).sum()                # minimize (5-z)^2
    loss.backward()                              # use autograd
    w.data –= learning_rate * w.grad.data        # update
    w.grad.data.zero_()                          # set gradient to zero again
    print(“w=", w.data, "cost=", "{:.9f}".format(loss.data[0]))
            # w.data holds the value     # loss.data holds the cost
```

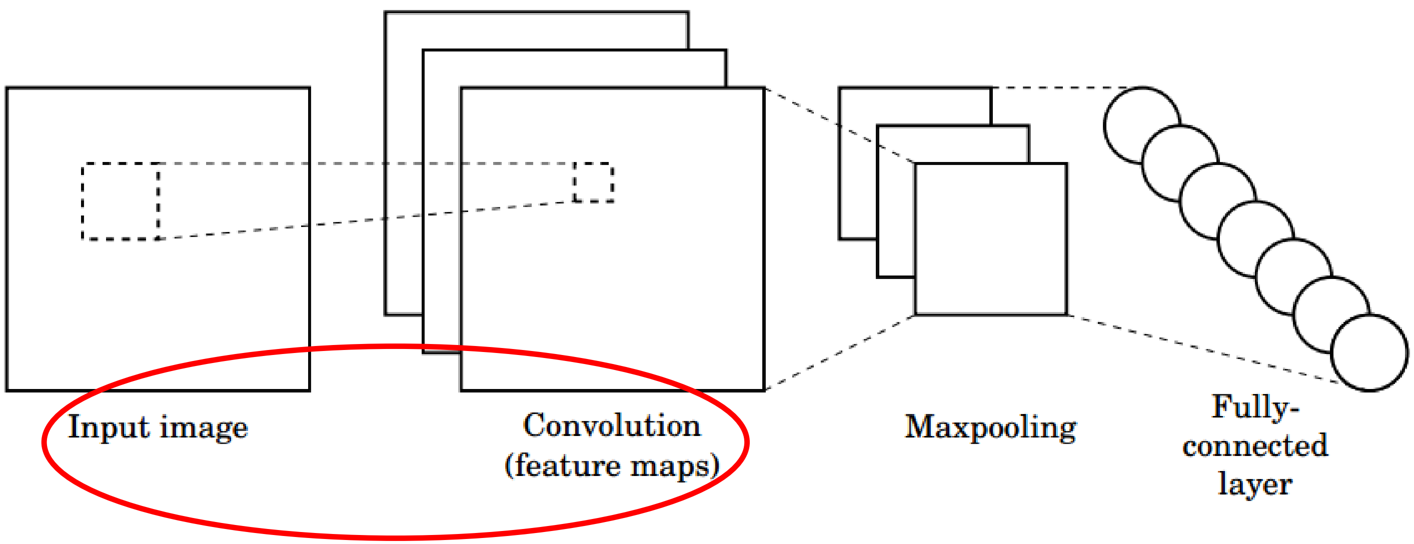
Typical Settings in Deep Learning

- more layers than traditional MLPs
- many more neurons
- for image input (also: sound, others):
  - convolutional layers
  - pooling layers
- other transfer functions, e.g.
  - rectified linear units, ReLU (instead of sigmoidal function)
  - classification: softmax with cross entropy loss on output layer
- Adam Optimizer (instead of Gradient Descent Optimizer)
- smaller learning rate

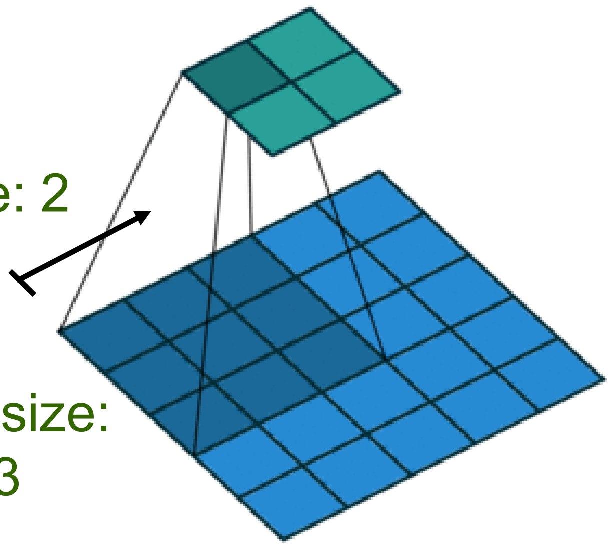
Convolution



- Local processing of image
- Small “kernels” instead of full connectivity
- Reuse of kernels/weights
- Convolution typical for lower layers



Parameters of a Convolution (here, 2D)



Size of input vector: 25

Number of weights: 9

Channels: 1

Size of output vector: 4

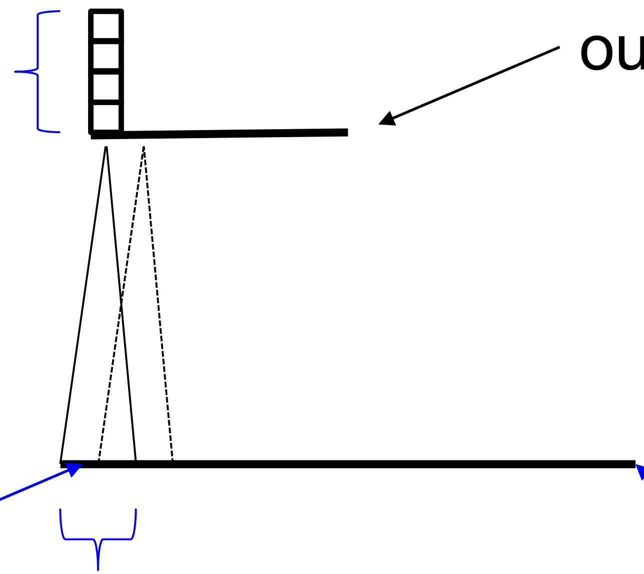
full connectivity would yield 25×4=100 weights

- **kernel size**: input region the neuron connects to
- **stride**: step size of kernel shift
- typically, multiple kernels (of same size and stride) are used → multiple output **channels** (feature maps)

Output layer size is determined by input size, kernel size, stride and number of channels.

https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d

Example, 1D Convolution:



output channels = 4

output layer

weights = ?

output layer size = ?

stride = 5

kernel size = 10

input layer dimension = 500

→ # weights = 10 × 4

output layer size = (500 – 10) / 5 × 4 = 98 × 4

Pooling

- Reduction of layer size
- Invariance to small shifts
- Convolution and max-pooling often used in pairs on lower layers

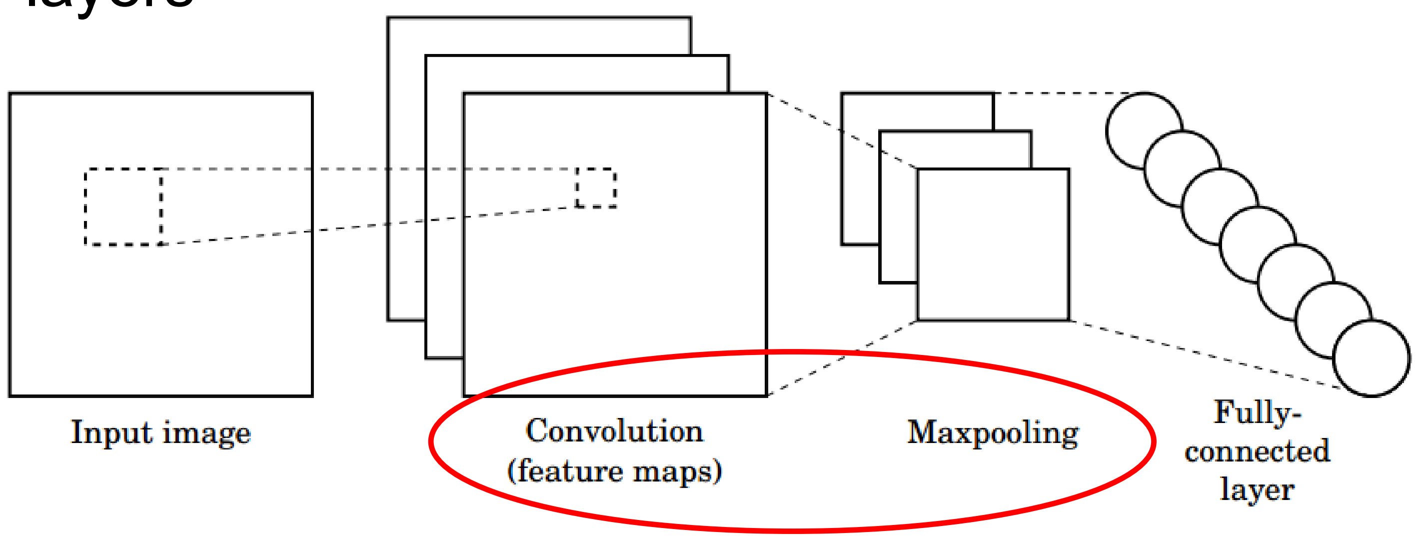
Typical: no overlap of pooled regions

12	20	30	0
8	12	2	0
34	70	37	4
112	100	25	12

Example: 2×2 max-pooling

20	30
112	37

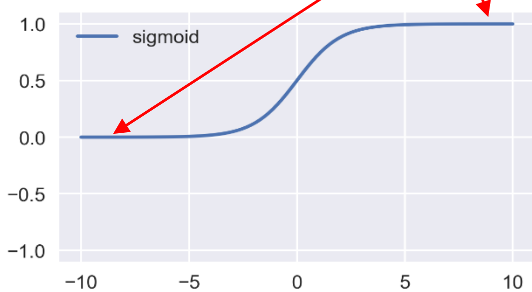
Reduction of layer size by 75%

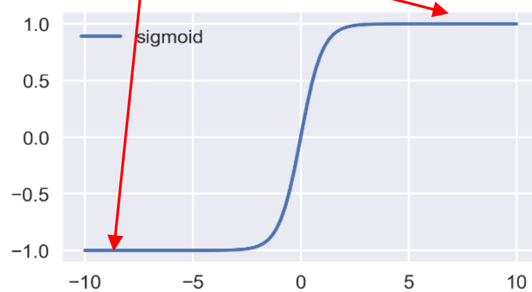


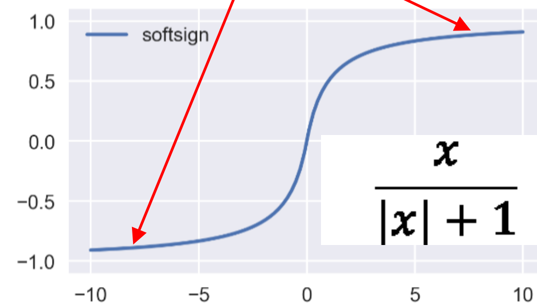
Transfer Functions

vanishing gradient

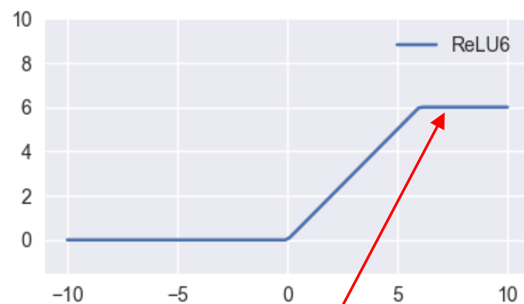
better gradient

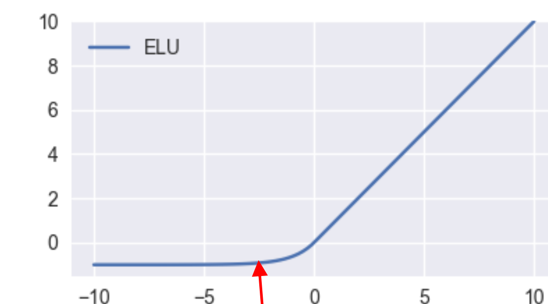






Rectified Linear Unit





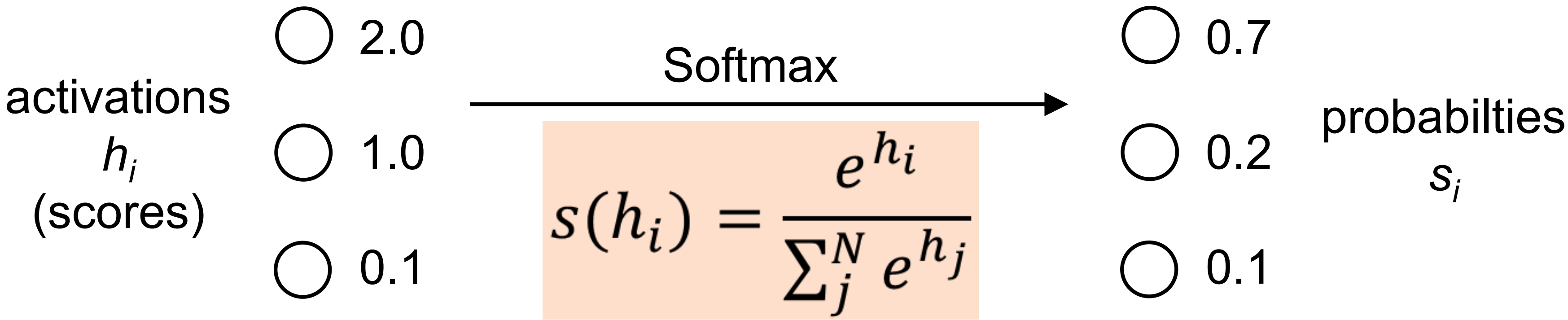
“active” gradient

saturation
(against divergence)

differentiable with
negative activity

http://yagami12.hatenablog.com/entry/2017/09/17/111935#ID_1-4-1

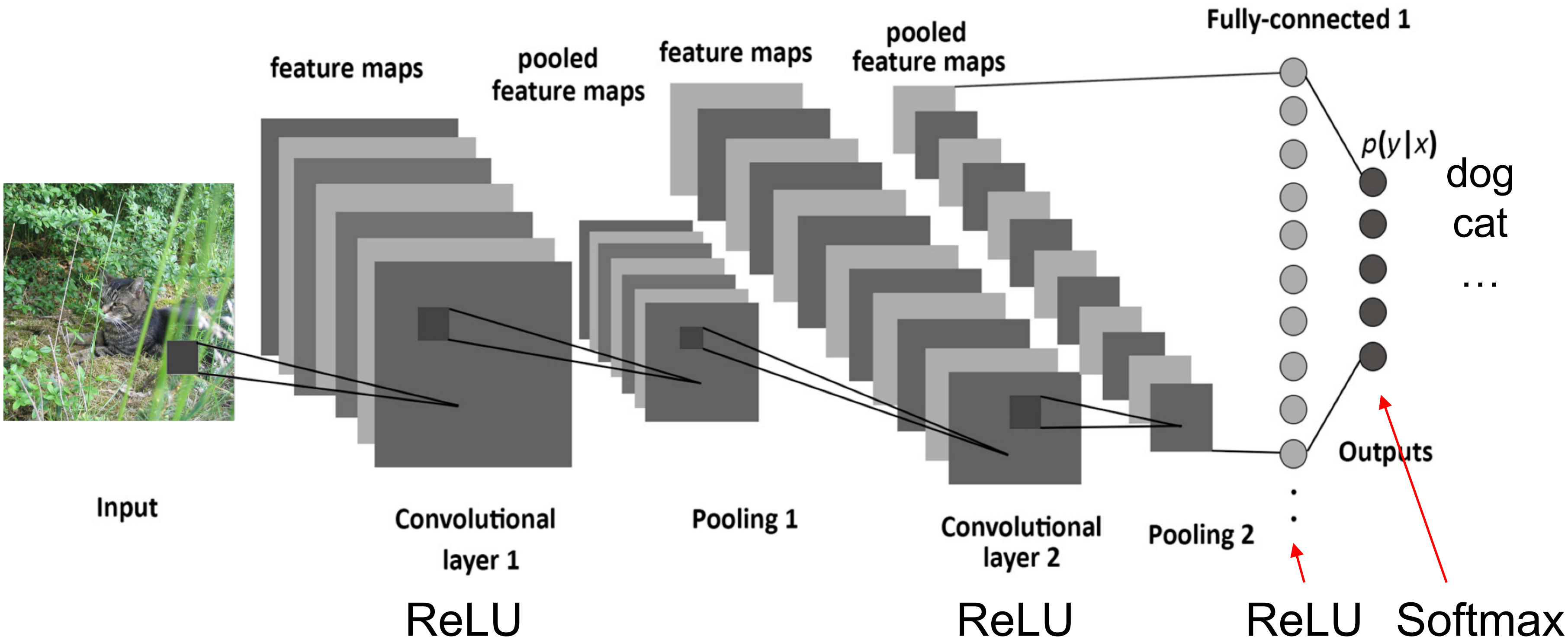
Softmax Transfer Function



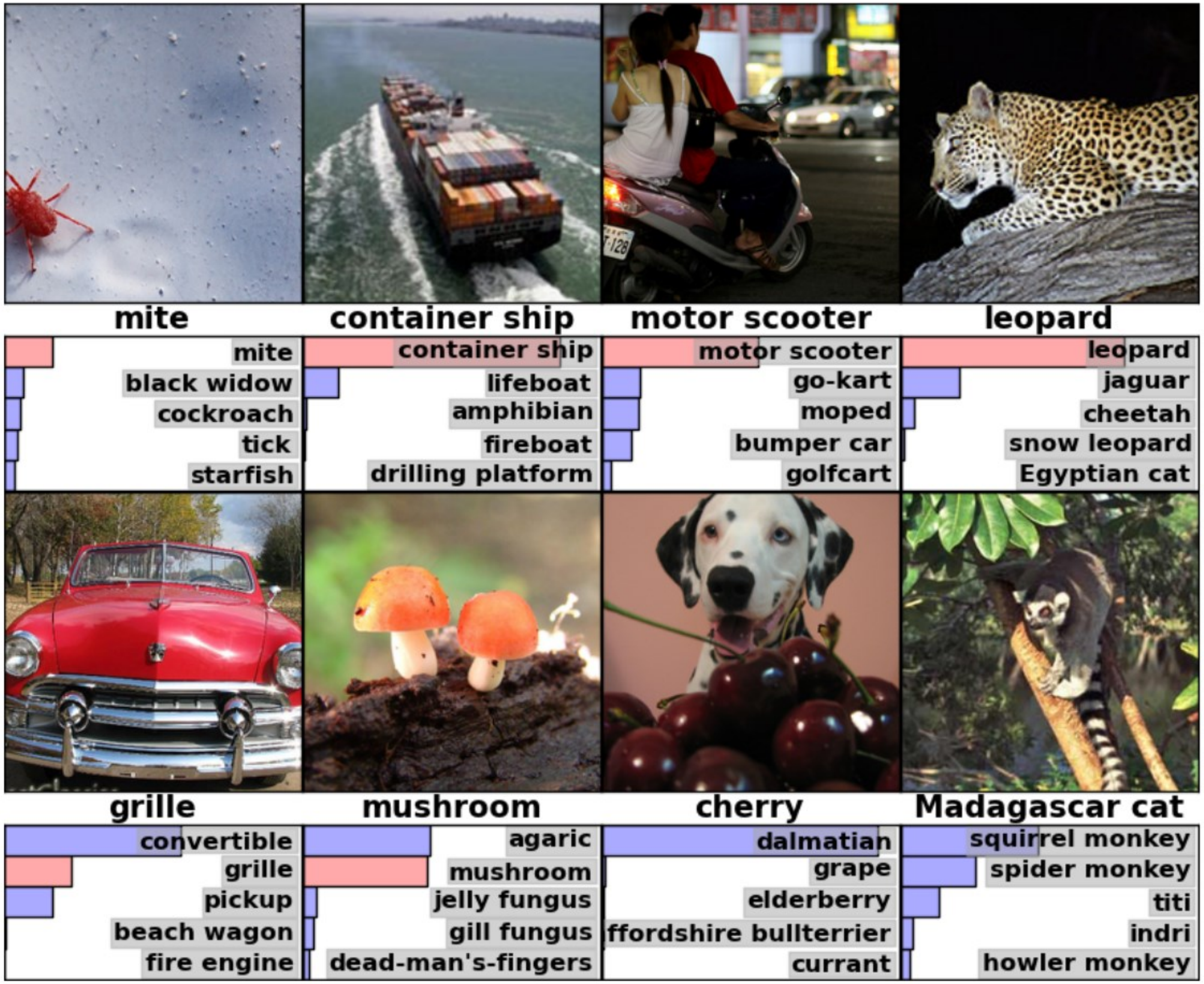
$$s(h_i) = \frac{e^{h_i}}{\sum_j^N e^{h_j}}$$

- Softmax typically used for 1-of-N classification problems
  - N output units with Softmax
  - s_i = probability of class i
  - Not a neuron-local transfer function
    (cf. winner-take-all in SOMs)
- Now L2 loss function not optimal.
  Better: cross-entropy loss.

Typical Convolutional NN (CNN)



Deep Learning Multi-Class Classification



ImageNet Classification with Deep Convolutional NNs. Krizhevsky, Sutskever, Hinton. NIPS, 2012

Deep Learning – Better than Human Performance

Previously, computers beat humans at several tasks:

- Pocket calculator
- Backgammon (BKG 9.8,1979)
- Chess (Deep Blue, 1996)
- Scrabble (Quackle, 2007)
- Poker (CFR$^+$, 2015)
- …

Deep learning added several more tasks: …

Better than Human: Image Classification

- Traffic sign recognition
  - Multi-column CNN (Ciresan, Meier, Schmidhuber, 2011)

- ImageNet data classification
  - Parametric ReLU (He et al., 2015)

GT: horse cart	GT: birdhouse	GT: forklift
1: horse cart	1: birdhouse	1: forklift
2: minibus	2: sliding door	2: garbage truck
3: oxcart	3: window screen	3: tow truck
4: stretcher	4: mailbox	4: trailer truck
5: half track	5: pot	5: go-kart

- Apparent age estimation from faces
  - Pretrained CNN (ImageNet data) + fine tuning (Rothe 2015)

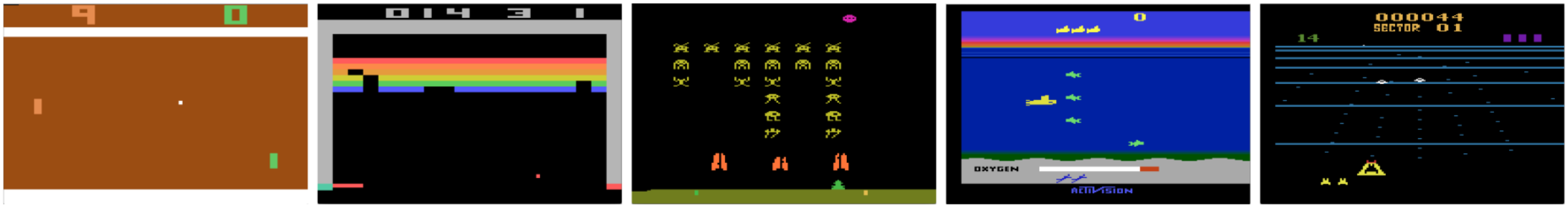
real / apparent age

59 / 57 37 / 35 65 / 51 20 / 29

Better than Human: Lip Reading & Speech

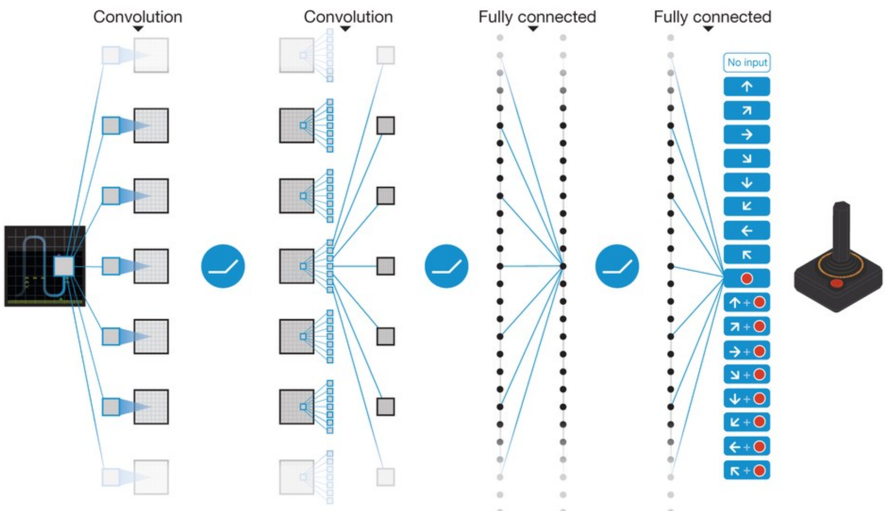
- Lip reading
  - CNN + recurrent LSTM network (Chung et al, 2016)
  - Attention mechanism aligns video frames with outputs
  - Outputs are characters
    - Characters are far fewer than words
    - Word/language model needed, can be trained on outputs
  - Beats human experts
- Conversational Speech Recognition (Xiong et al., 2016)
  - CNN + recurrent LSTM network
  - “Highway” connections allow ~50 layers
    - a linear transform of each layer’s input to the layer’s output
  - Achieves human parity

Better than Human: Atari Playing



- Multiple Atari Games (at some games still worse than humans)
  - Deep Reinforcement Learning (RL) (Mnih et al., 2015)

input (screen)

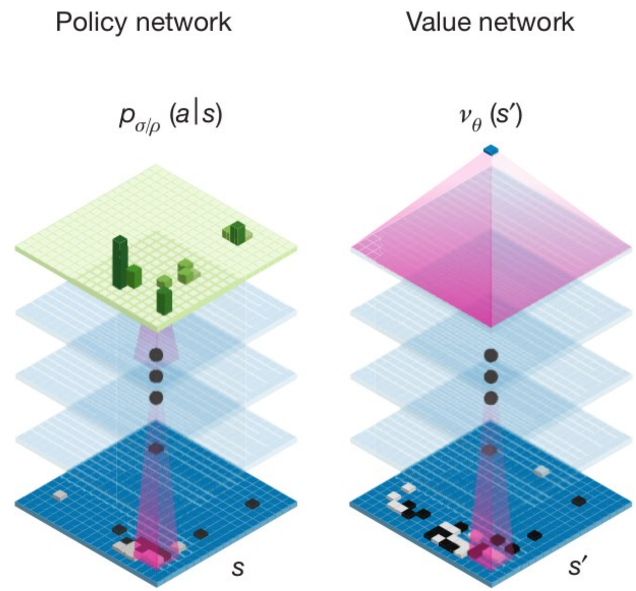


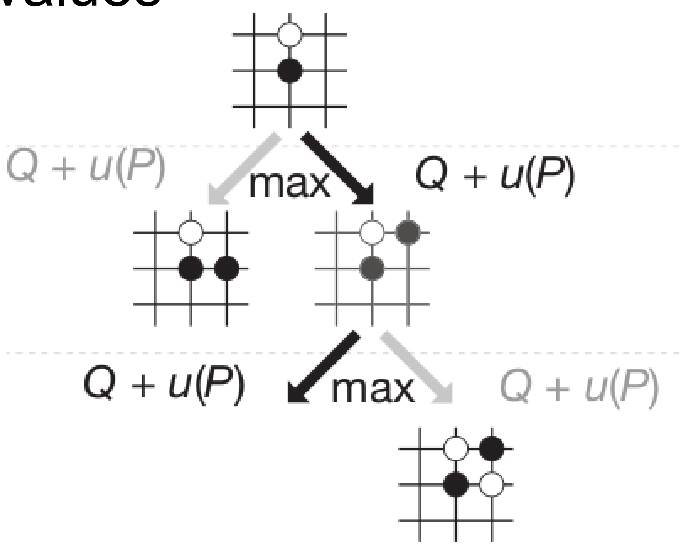
output (values of discrete actions, determined by RL)

CNN with ReLU

Better than Human: Go Game

- AlphaGo (Silver et al. 2016)
  - CNN pre-trained supervised from human players
  - CNN then trained by RL via self-play to predict policy- and state values
  - Tree search into branches of large values





Better than Human: Question Answering

- Stanford Question Answering Dataset (Wikipedia-based)
  - Answers contained in text (exact match)
  - Ensembles are best

In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under **gravity**. The main forms of precipitation include drizzle, rain, sleet, snow, **graupel** and hail... Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals **within a cloud**. Short, intense periods of rain in scattered locations are called "showers".

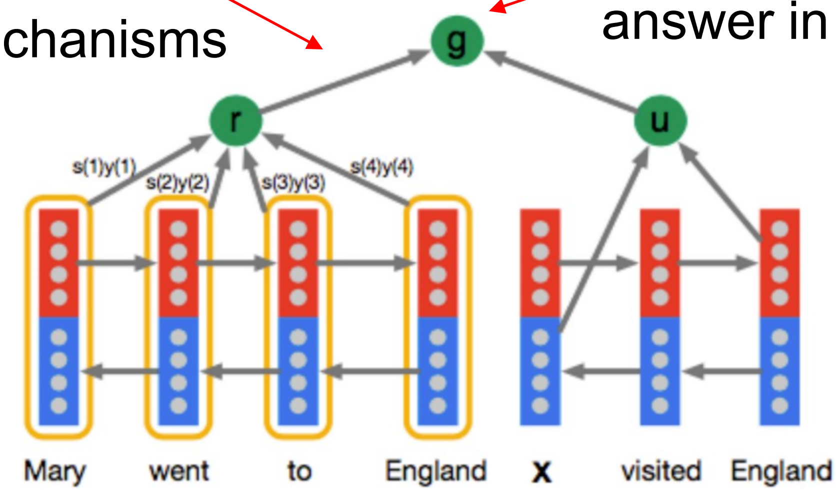
What causes precipitation to fall?
gravity

What is another main form of precipitation besides drizzle, rain, snow, sleet and hail?
graupel

Where do water droplets collide with ice crystals to form precipitation?
within a cloud

attention mechanisms

pointer to answer in text



document

query

bi-directional RNN

word embedding vectors

Further Reading

- LeCun: Efficient Backprop

  http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf

- Tensorflow, examples with MNIST data:

  https://www.tensorflow.org/tutorials/

  https://keras.io

- PyTorch:

  http://pytorch.org/tutorials/

  https://arxiv.org/abs/1912.01703
  (paper describing PyTorch principles)