Machine Learning

Lecture 17: Neural networks continued

Stefan Harmeling

11. December 2017

Was haben Sie nicht verstanden?

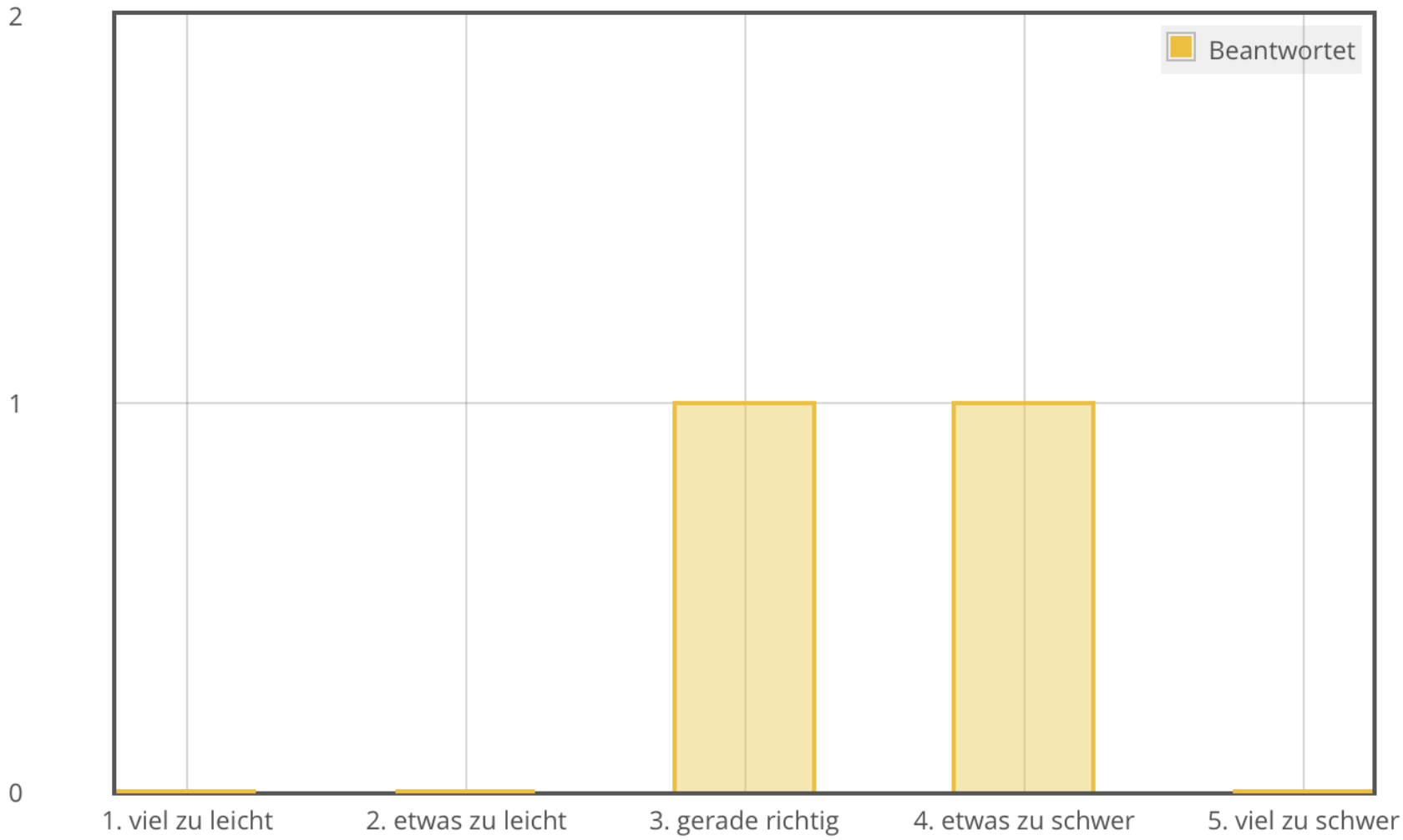
- Unterscheiden sich ein einstufiges neuronales Netz und eine SVM nur durch die Loss-Funktion?

Was hat Ihnen gefallen?

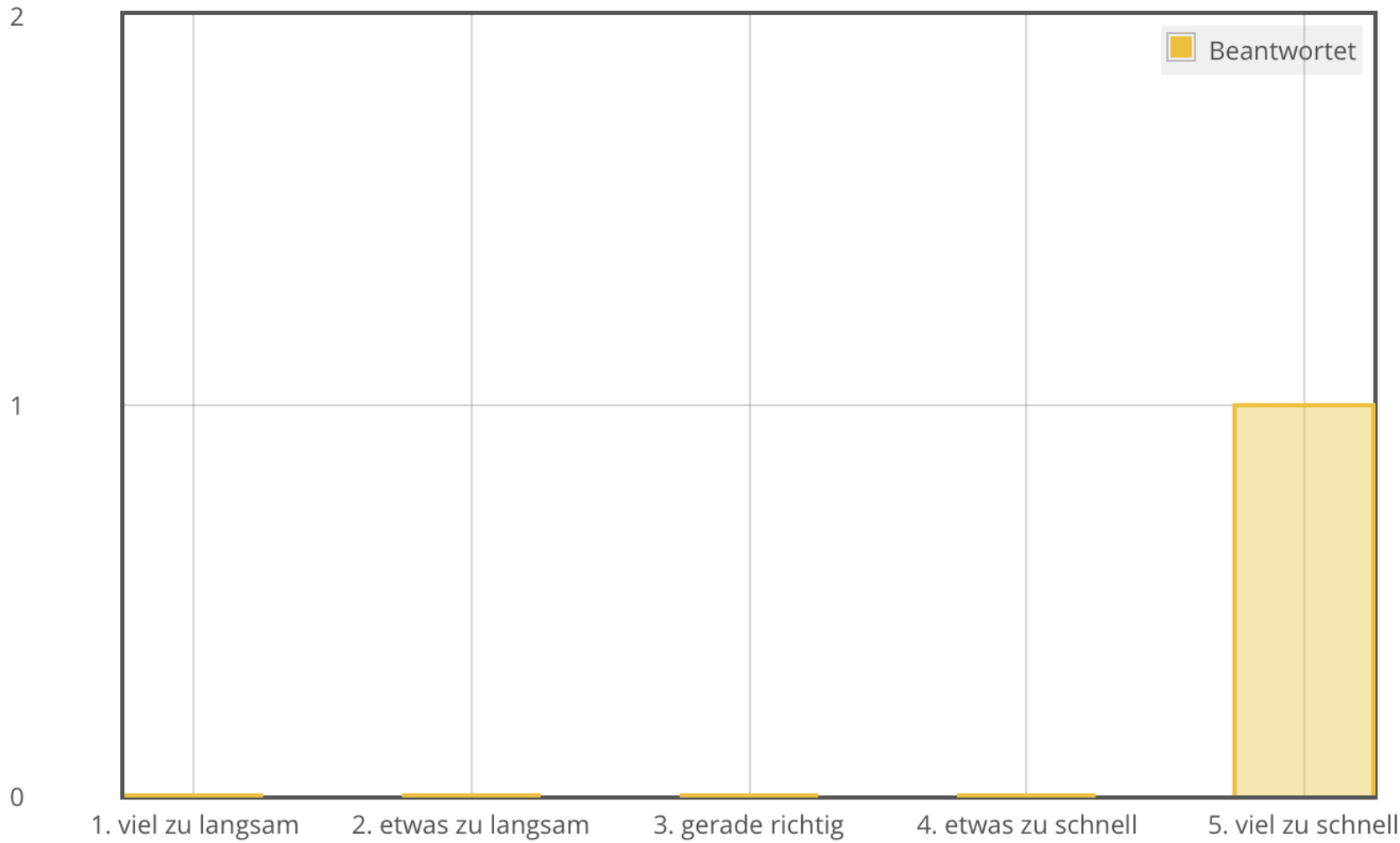
- Das Experiment

Was hat Ihnen nicht gefallen?

War die Vorlesung zu leicht oder zu schwer?



War die Vorlesung zu schnell oder zu langsam?



Haben Sie weitere Anmerkungen?

- Einige Themen sind kompliziert aber Prof. Dr. Stefan Harmeling ist der Beste !
- Was denken Sie über Bitcoin ?

What is back propagation?

- Consider simple neural network with scalars

$$y = f_3(w_3, f_2(w_2, f_1(w_1, x)))$$

  x and y are scalars, parameters w_3, w_2, w_1 are also scalars. We don't specify f_1, f_2, f_3.
- For a data point x with target value t we define the loss function

$$E = \frac{1}{2}(y - t)^2$$

- Goal: learn parameters that minimize the loss

$$\min_{w_3, w_2, w_1} E$$

- *Back-propagation* is an efficient way to calculate the derivatives of the loss wrt the parameters:

$$\frac{\partial E}{\partial w_3} \qquad \frac{\partial E}{\partial w_2} \qquad \frac{\partial E}{\partial w_1}$$

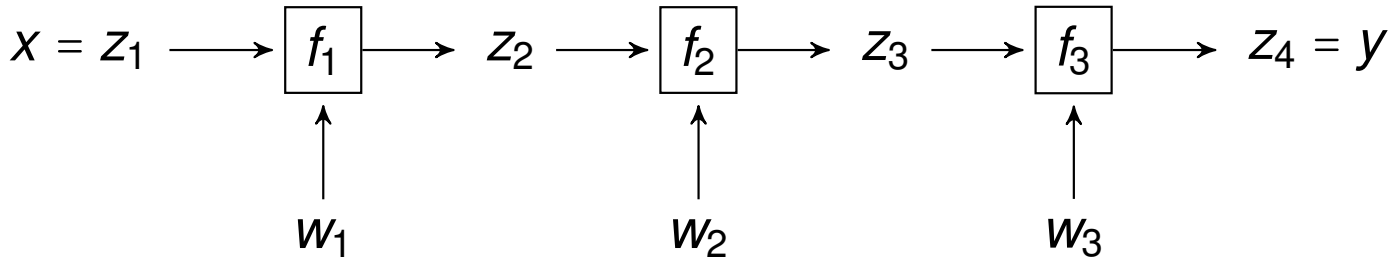
Rewrite the network into layers

$$
\begin{aligned}
z_1 &= x && \text{input layer} \\
z_2 &= f_1(w_1, z_1) && \text{second layer (hidden)} \\
z_3 &= f_2(w_2, z_2) && \text{third layer (hidden)} \\
y = z_4 &= f_3(w_3, z_3) && \text{output layer} \\
E &= \tfrac{1}{2}(y - t)^2 && \text{squared error (the loss)}
\end{aligned}
$$

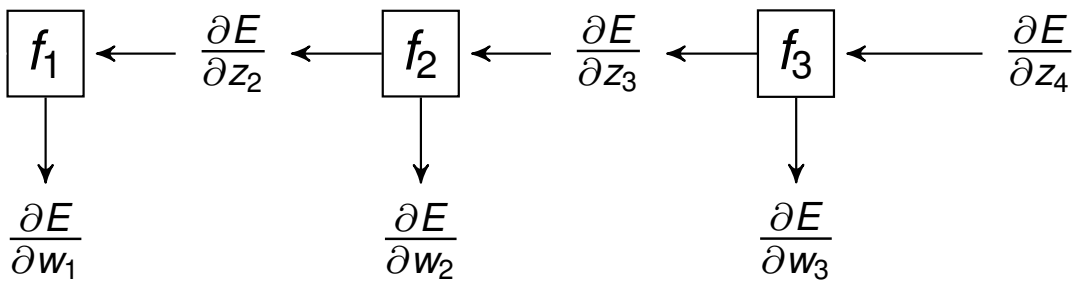
Applying chain rule

$$
\begin{aligned}
\frac{\partial E}{\partial w_3} &= \frac{\partial E}{\partial z_4}\frac{\partial z_4}{\partial w_3} \\
\frac{\partial E}{\partial w_2} &= \frac{\partial E}{\partial z_4}\frac{\partial z_4}{\partial z_3}\frac{\partial z_3}{\partial w_2} \\
\frac{\partial E}{\partial w_1} &= \frac{\partial E}{\partial z_4}\frac{\partial z_4}{\partial z_3}\frac{\partial z_3}{\partial z_2}\frac{\partial z_2}{\partial w_1}
\end{aligned}
$$

Propagation (feed forward)



Back propagation



Back-propagation step by step

1. At the loss:
   - sends $\frac{\partial E}{\partial z_4}$ to f_3
2. At f_3:
   - receives $\frac{\partial E}{\partial z_4}$
   - calculates $\frac{\partial E}{\partial w_3} = \frac{\partial E}{\partial z_4}\frac{\partial z_4}{\partial w_3}$ with $\frac{\partial z_4}{\partial w_3} = \frac{\partial}{\partial w_3} f_3(w_3, z_3)$
   - updates $w_3 = w_3 - \eta \frac{\partial E}{\partial w_3}$
   - sends $\frac{\partial E}{\partial z_3} = \frac{\partial E}{\partial z_4}\frac{\partial z_4}{\partial z_3}$ to f_2 with $\frac{\partial z_4}{\partial z_3} = \frac{\partial}{\partial z_3} f_3(w_3, z_3)$
3. At f_2:
   - receives $\frac{\partial E}{\partial z_3}$
   - calculates $\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial z_3}\frac{\partial z_3}{\partial w_2}$ with $\frac{\partial z_3}{\partial w_2} = \frac{\partial}{\partial w_2} f_2(w_2, z_2)$
   - updates $w_2 = w_2 - \eta \frac{\partial E}{\partial w_2}$
   - sends $\frac{\partial E}{\partial z_2} = \frac{\partial E}{\partial z_3}\frac{\partial z_3}{\partial z_2}$ to f_1 with $\frac{\partial z_3}{\partial z_2} = \frac{\partial}{\partial z_2} f_2(w_2, z_2)$
4. At f_1:
   - receives $\frac{\partial E}{\partial z_2}$
   - calculates $\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial z_2}\frac{\partial z_2}{\partial w_1}$ with $\frac{\partial z_2}{\partial w_1} = \frac{\partial}{\partial w_1} f_1(w_1, z_1)$
   - updates $w_1 = w_1 - \eta \frac{\partial E}{\partial w_1}$

Notation!

- Scalars only (EASY)

$$\underbrace{\frac{\partial E}{\partial w}}_{\in \mathbb{R}} = \underbrace{\frac{\partial E}{\partial y}}_{\in \mathbb{R}} \underbrace{\frac{\partial y}{\partial w}}_{\in \mathbb{R}}$$

- Scalars and vectors (BE CAREFUL)

$$\underbrace{\frac{\partial E}{\partial x}}_{\in \mathbb{R}^{1\times n}} = \underbrace{\frac{\partial E}{\partial y}}_{\in \mathbb{R}^{1\times m}} \underbrace{\frac{\partial y}{\partial x}}_{\in \mathbb{R}^{m\times n}}$$

where $E \in \mathbb{R}$, $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$.

- Scalars, vectors, and matrices (DANGER!)

$$\underbrace{\frac{\partial E}{\partial W}}_{\in \mathbb{R}^{n\times m}} = \underbrace{\frac{\partial E}{\partial y}}_{\in \mathbb{R}^{1\times m}} \underbrace{\frac{\partial y}{\partial W}}_{\in \mathbb{R}^{m\times m\times n}}$$

Here, good notation is difficult!

where $E \in \mathbb{R}$, $W \in \mathbb{R}^{m\times n}$, $y \in \mathbb{R}^m$.

Building blocks for neural networks

see Bottou, Gallinari, *A Framework for the Cooperation of Learning Algorithms*, 1991

In a nutshell

Neural networks are sequences of computational blocks (or bricks).

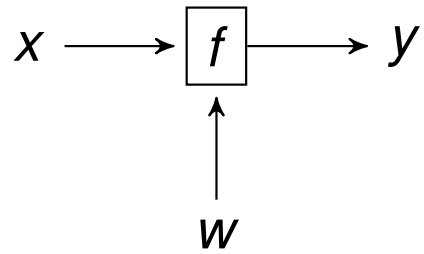
More general

Neural networks are directed acyclic graphs (DAGs)
of computational blocks (or bricks).

Question

What kind of blocks?

General block with input x, output y, parameters w



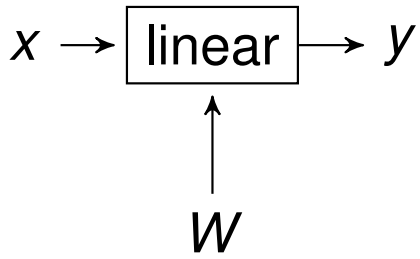
Propagation (feed forward)

$$y = f(w, x)$$

Back propagation

$$\underbrace{\frac{\partial E}{\partial w} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial w}}_{\text{to update } w} \qquad \underbrace{\frac{\partial E}{\partial x} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial x}}_{\text{to send to previous block}}$$

Linear block:



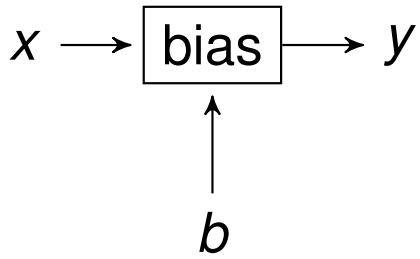
Propagation (feed forward)

$$y = Wx$$

Back propagation

$$\frac{\partial E}{\partial W} = x\frac{\partial E}{\partial y} \qquad \frac{\partial E}{\partial x} = \frac{\partial E}{\partial y}W$$

Bias block:



Propagation (feed forward)

$$y = x + b$$

Back propagation

$$\frac{\partial E}{\partial b} = \frac{\partial E}{\partial y} \qquad \frac{\partial E}{\partial x} = \frac{\partial E}{\partial y}$$

Tanh block (component-wise):

Propagation (feed forward)

$$y = \tanh(x)$$

Back propagation

$$\frac{\partial E}{\partial x} = \frac{\partial E}{\partial y} \operatorname{Diag}(1 - y \odot y) = \frac{\partial E}{\partial y} \odot (1 - y \odot y)^{\mathsf{T}}$$

Logistic block (component-wise):

$$x \rightarrow \boxed{\text{logistic}} \rightarrow y$$

Propagation (feed forward)

$$y = \frac{1}{1 + e^{-x}}$$

Back propagation

$$\frac{\partial E}{\partial x} = \frac{\partial E}{\partial y} \text{Diag}(y \odot (1 - y)) = \frac{\partial E}{\partial y} \odot (y \odot (1 - y))^{\mathsf{T}}$$

ReLU block (rectified linear unit, component-wise):

Propagation (feed forward)

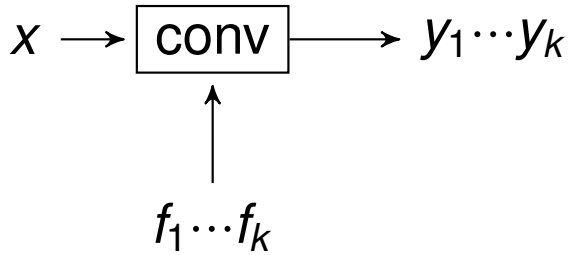
$$y = \max(x, 0) = [x > 0] \odot x$$

Back propagation

$$\frac{\partial E}{\partial x} = \frac{\partial E}{\partial y} \text{Diag}([x > 0]) = \frac{\partial E}{\partial y} \odot [x > 0]^T$$

where $[x > 0]$ is a vector of ones and zeros.

Convolutional block:



Propagation (feed forward)

$$y_i = f_i * x$$ "convolution"

Back propagation

$$\frac{\partial E}{\partial f_i} = \ldots \qquad \frac{\partial E}{\partial x} = \ldots$$

Max pooling block:

Propagation (feed forward)

$$y = Dx$$

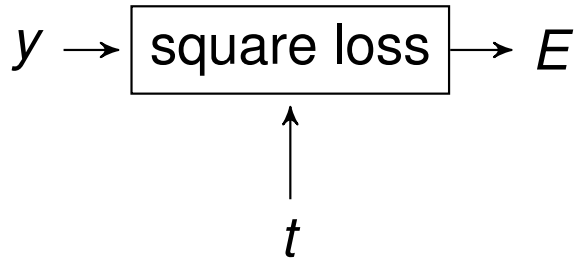
"D downsampling matrix"

Back propagation

$$\frac{\partial E}{\partial x} = \frac{\partial E}{\partial y} D$$

- pooling means “downsampling”
- alternatives: maximum or mean over region

Square loss block:



Propagation (feed forward)

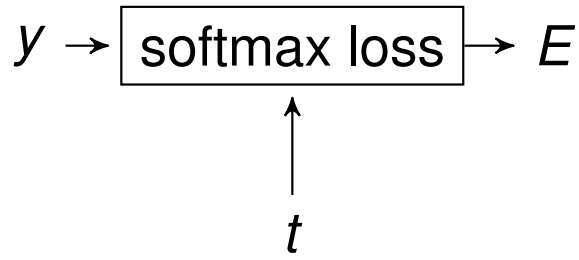
$$E = 0.5(y - t)^2$$

Back propagation

$$\frac{\partial E}{\partial y} = y - t$$

- a loss layer is for convenience, it is not part of the solution

Softmax loss block:



Propagation (feed forward)

$$E = -\log \frac{\exp(y_t)}{\sum_j \exp(y_j)}$$

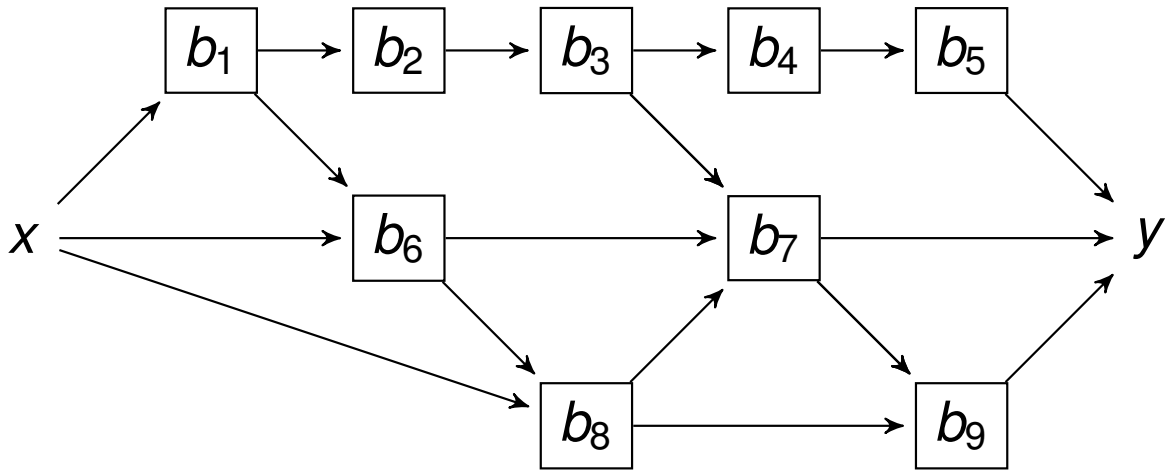
Back propagation

$$\frac{\partial E}{\partial y} = \dots$$

- loss function for multi-class problems
- assuming $y = [y_1, \dots, y_k]$ for k classes, and $t \in \{1, \dots, k\}$

Neural network as a sequence of bricks

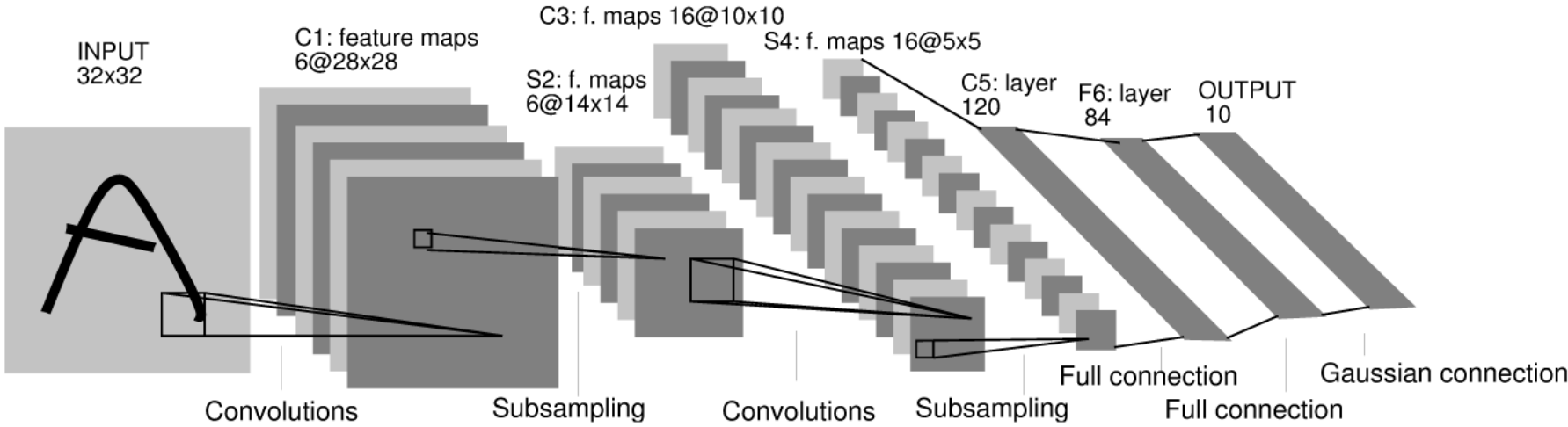
Neural network as a DAG of bricks



Famous examples

LeNet-5

Layers of LeNet-5: not its bricks

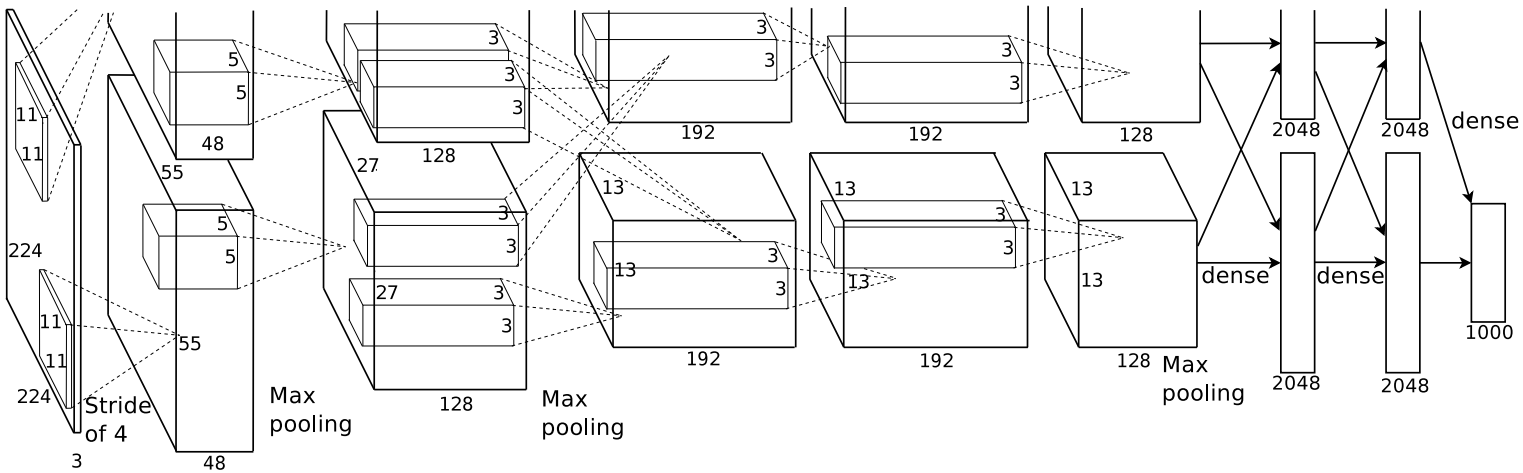


from Bottou's MLSS 2013 slides on neural networks

- LeNet-5 robustly recognizes digits.
- Developed at AT&T in the 1990s by Yann Lecun, Leon Bottou et al.
- LeNet-5 been used for a decade to read digits on about 10-20 percent of all (!) US checks.
- Demo at `http://yann.lecun.com/exdb/lenet/index.html`.

ImageNet classification

- 1.2 million images, 1000 classes
- network with 60 million parameters



from Kryzhevsky, Sutskever, Hinton, *ImageNet Classification with Deep Convolutional Neural Networks*, 2012

ImageNet classification

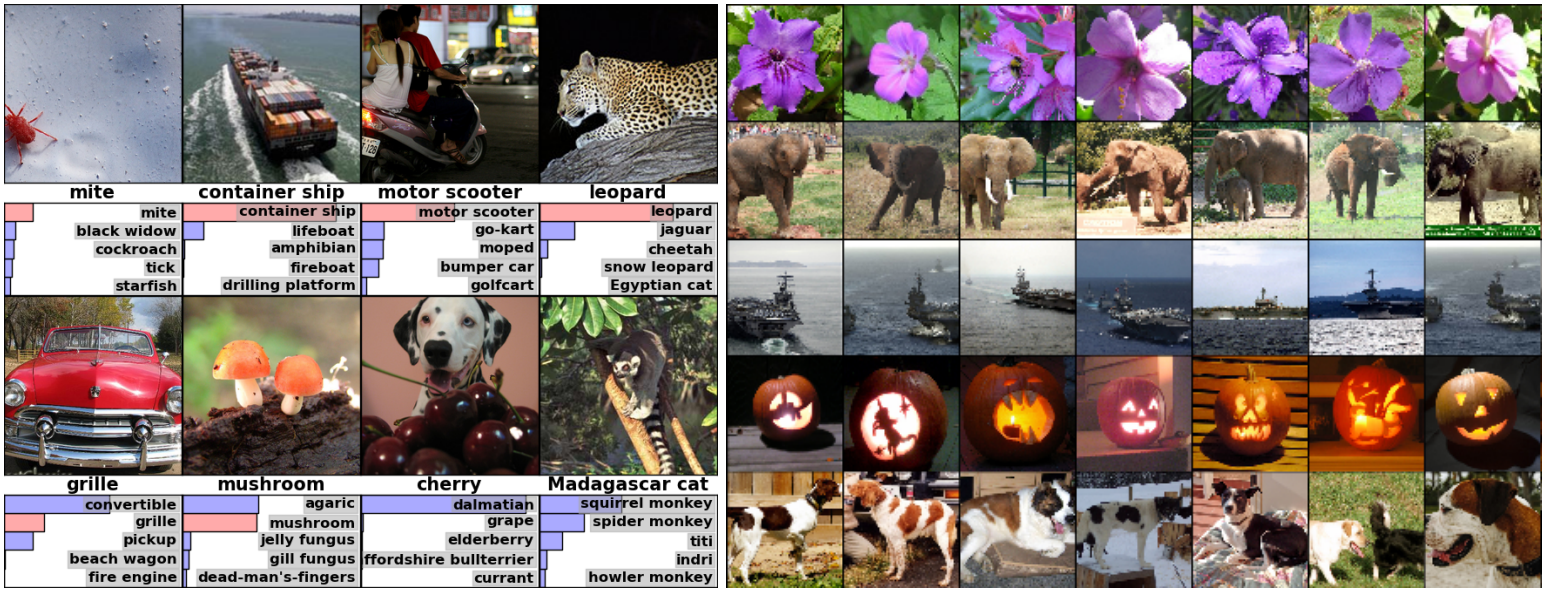


Figure 4: **(Left)** Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). **(Right)** Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.

from Kryzhevsky, Sutskever, Hinton, *ImageNet Classification with Deep Convolutional Neural Networks*, 2012

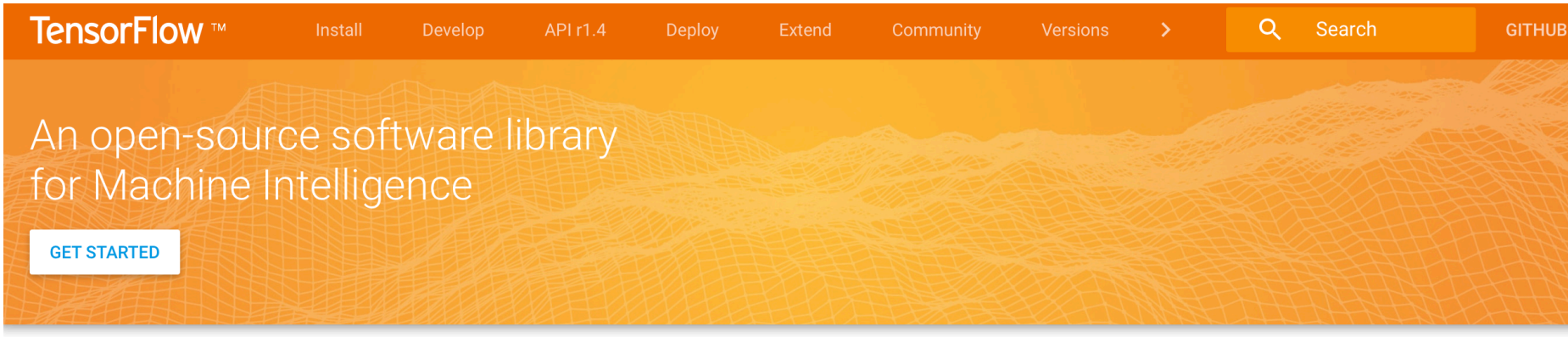
Toolboxes for neural networks

Toolboxes (somewhat randomly selected)

- Caffe – Deep learning framework
  `http://caffe.berkeleyvision.org`
  - written in C++ and cuda, interface for Python, Matlab
  - 40M images per day on GPU (from their website)
  - runs on CPU and GPU, supports NVIDIA's cuDNN
- Torch 7 `http://torch.ch`
  - written in C combined with scripting language Lua and LuaJIT
  - runs on CPU and GPU, (most likely) supports NVIDIA's cuDNN
  - contains lots of other machine learning algorithms as well
  - ...
- Mocca `http://mochajl.readthedocs.org/en/latest/`
  - written in Julia, just install it via `Pkg.add("Mocca")`
  - simple to install
  - basically this is Caffe written in Julia
  - runs on CPU and GPU, supports NVIDIA's cuDNN

Many more and benchmarks at:

- `https://github.com/soumith/convnet-benchmarks`



Eager Execution

We're announcing eager execution, an imperative, define-by-run interface to TensorFlow. Check out the README to get started today.

TensorFlow 1.4 has arrived!

We're excited to announce the release of TensorFlow 1.4! Check out the release notes for all the latest.

Announcing TensorFlow Lite

Learn more about TensorFlow's lightweight solution for mobile and embedded devices.