DEV DAY

DEV DAY

MLT2

Deep Learning: concepts, algorithms & use cases

Julien Simon
Global Evangelist, AI & Machine Learning
Amazon Web Services

@julsimon

aws

What to expect

An introduction to Deep Learning theory

- Neurons & Neural Networks
- The Training Process
- Backpropagation
- Optimizers

Common network architectures and use cases

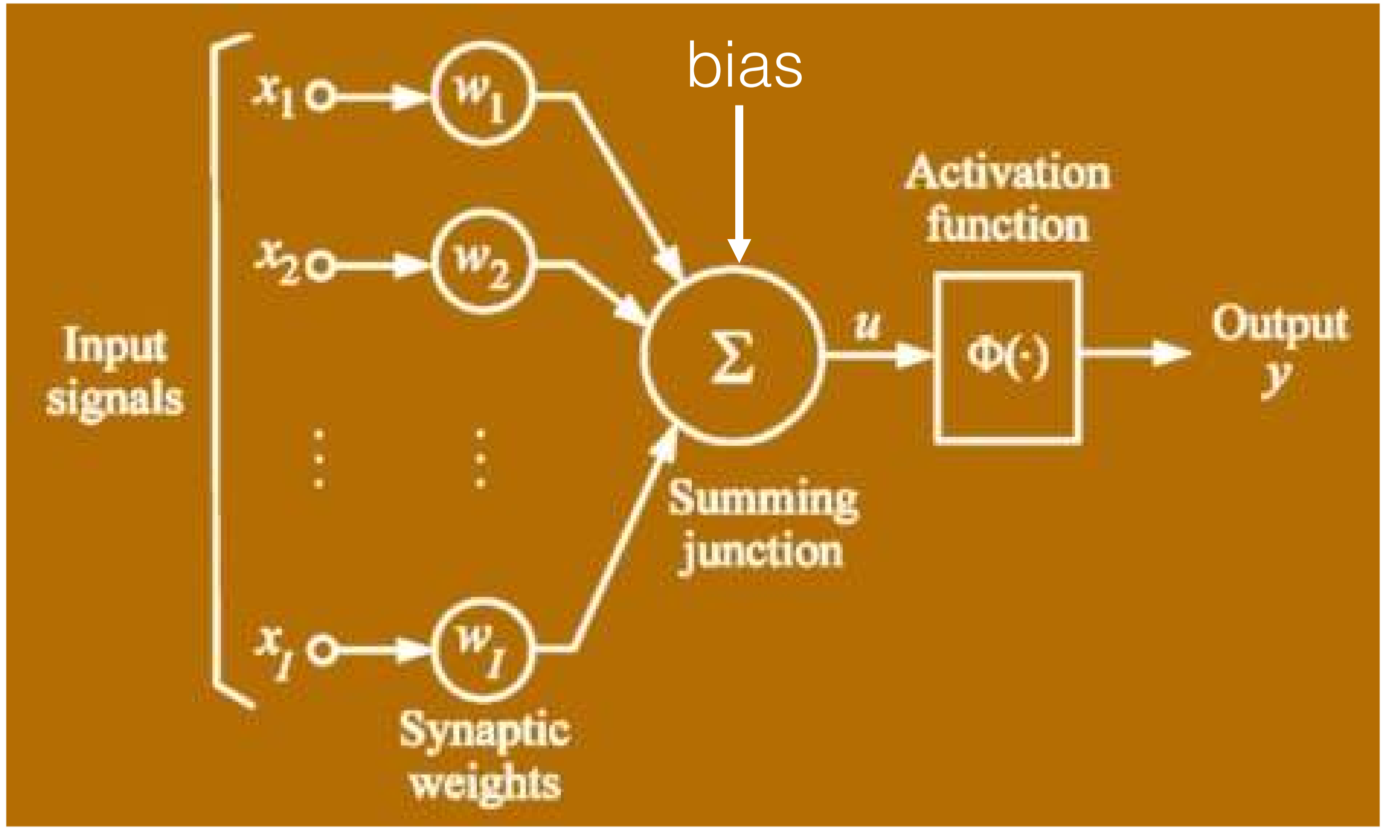
- Convolutional Neural Networks
- Recurrent Neural Networks
- Long Short Term Memory Networks
- Generative Adversarial Networks

Getting started

DEV DAY

An introduction to Deep Learning theory

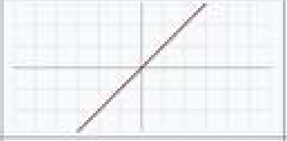
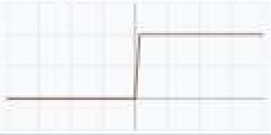
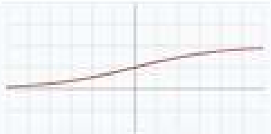
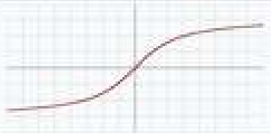
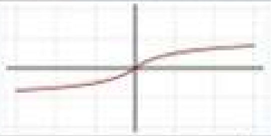
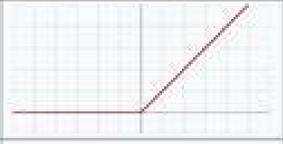
The neuron



$$\sum_{i=1}^{l} x_i * wi + b = u$$

"Multiply and Accumulate"

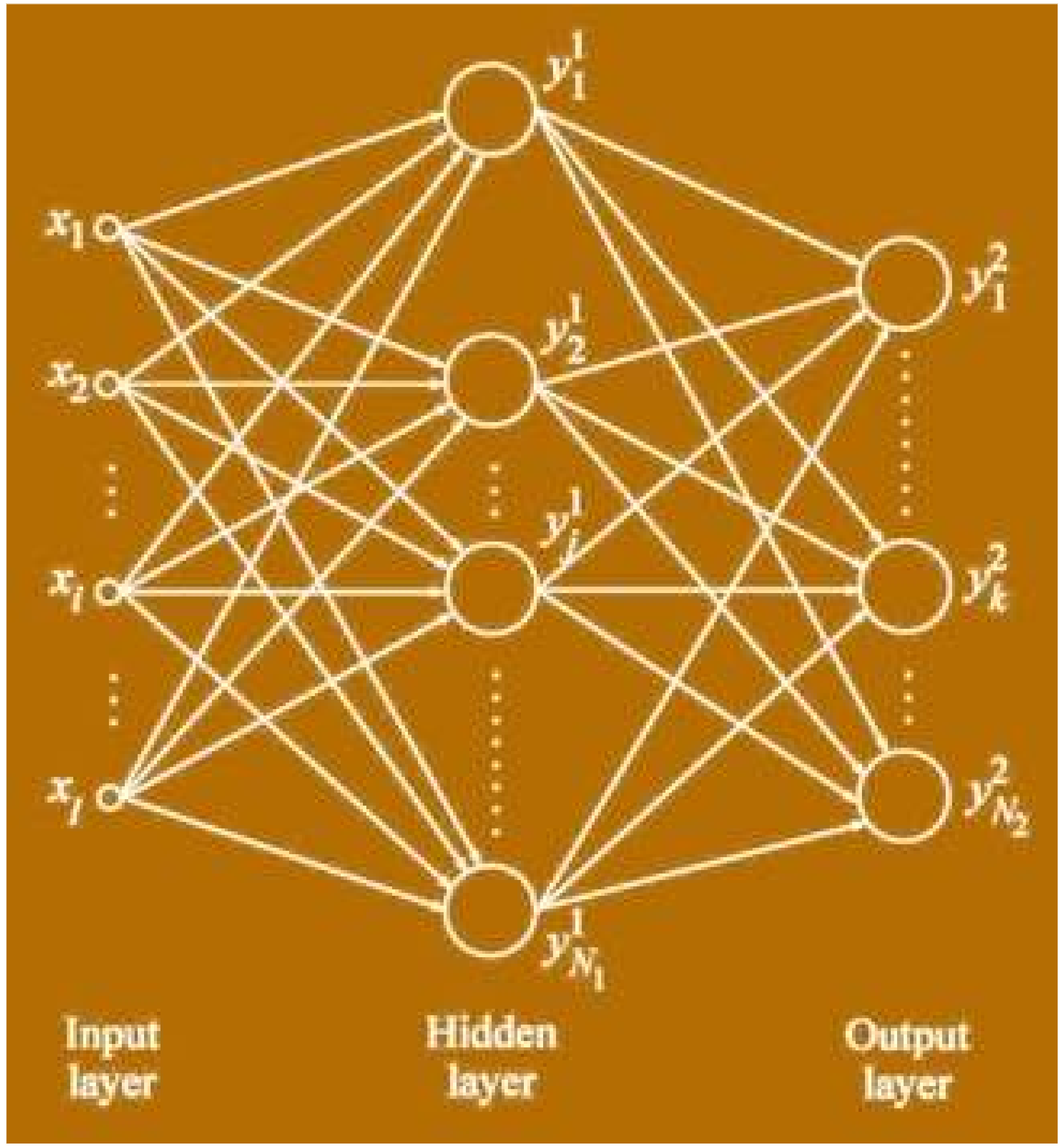
Activation functions

Name	Plot	Equation
Identity		$f(x) = x$
Binary step		$f(x) = \begin{cases} 0 & \text{for} & x < 0 \\ 1 & \text{for} & x \geq 0 \end{cases}$
Logistic (a.k.a. Soft step)		$f(x) = \frac{1}{1+e^{-x}}$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$
ArcTan		$f(x) = \tan^{-1}(x)$
Softsign [7][8]		$f(x) = \frac{x}{1+\lvert x\rvert}$
Rectified linear unit (ReLU)[9]		$f(x) = \begin{cases} 0 & \text{for} & x < 0 \\ x & \text{for} & x \geq 0 \end{cases}$

Source: Wikipedia

Neural networks

Building a simple classifier



l features

X =

$x_{11}, x_{12}, \dots\ x_{1l}$

$x_{21}, x_{22}, \dots\ x_{2l}$

… … …

$x_{m1}, x_{m2}, \dots\ x_{ml}$

m samples

y =

2
0
…
4

0,0,1,0,0,…,0

1,0,0,0,0,…, 0

…

0,0,0,0,1,…, 0

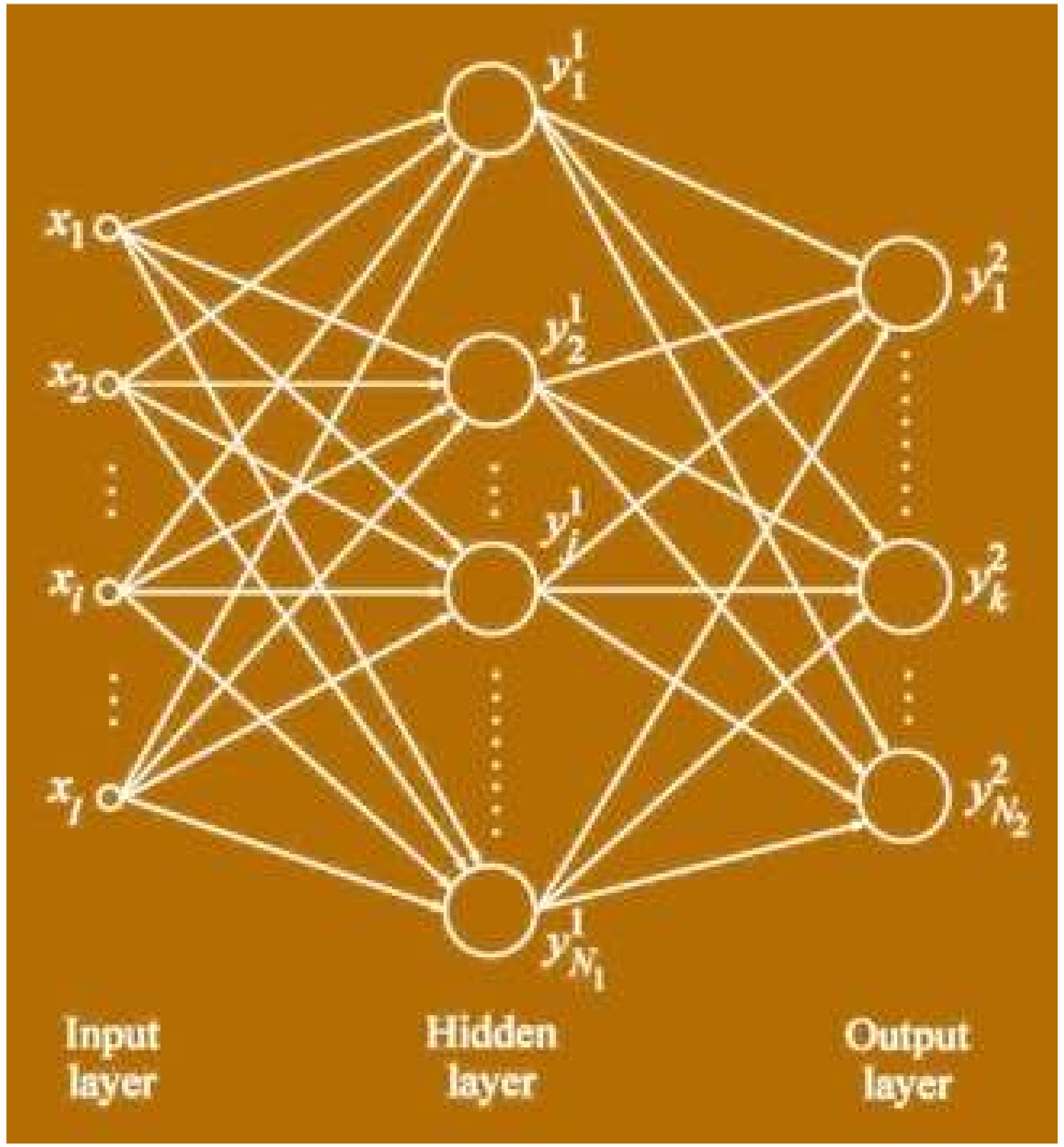
One-hot encoding

m labels, N_2 categories

Biases are ignored for the rest of this discussion

Neural networks

Building a simple classifier



l features

$x =$

x_{11}, x . x_{1l}
x_{21}, x .. x_{2l}
... ..
x_{m1}, x ..

$x_{11}, x_{12}, \ldots x_{1l}$

m samples

0,0,1,0,0,...,0

$y =$

2
0
...
4

0,0,1,0,...,
0 1,0,0,...,
0 ...
0,0,0,...,
0

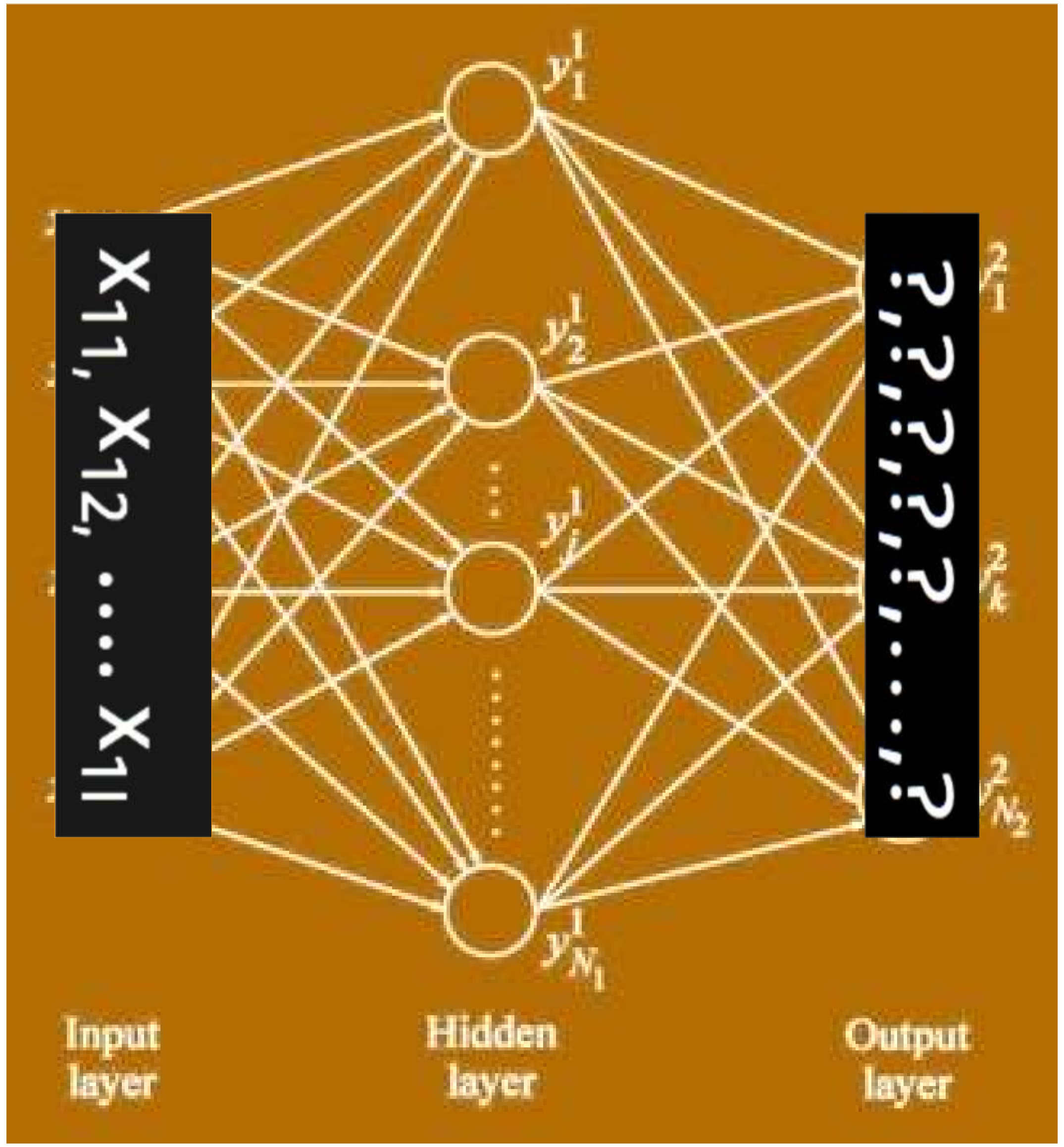
One-hot encoding

m labels, N_2 categories

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

Neural networks

Building a simple classifier



Initially, the network will not predict correctly
$f(X_1) = Y'_1$

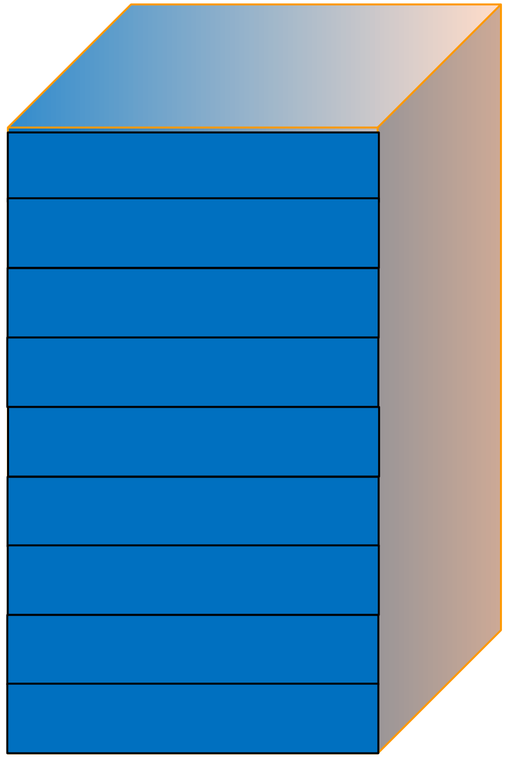
A loss function measures the difference between the real label Y_1 and the predicted label Y'_1
error = $\text{loss}(Y_1, Y'_1)$

For a batch of samples:

$$\sum_{i=1}^{\text{batch size}} \text{loss}(Y_i, Y'_i) = \textbf{batch error}$$

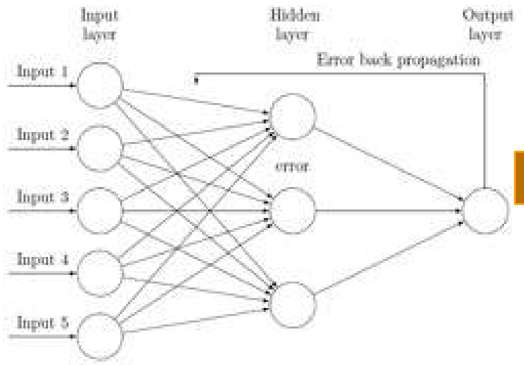
The purpose of the training process is to minimize error by gradually adjusting weights.

Mini-batch Training

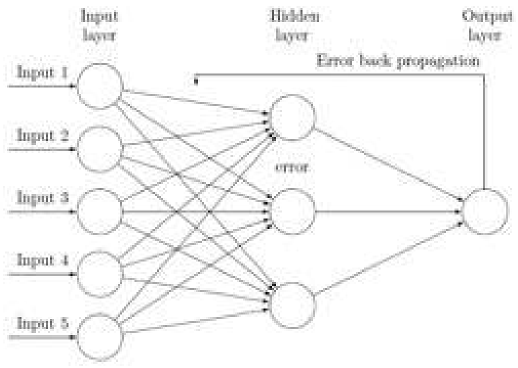


Training data set

Forward propagation



Backpropagation

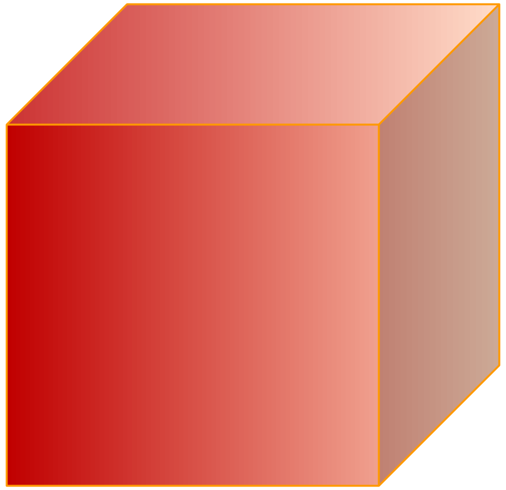


Trained
neural network

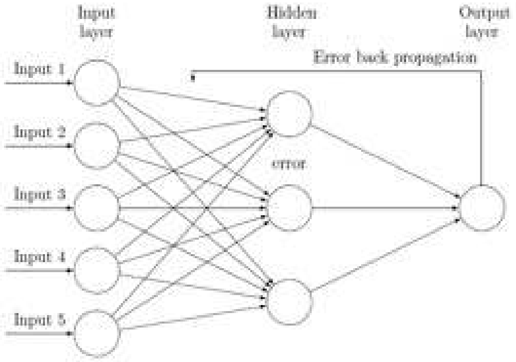
Batch size
Learning rate
Number of epochs

Hyper parameters

Validation



Validation data set (also called dev set)



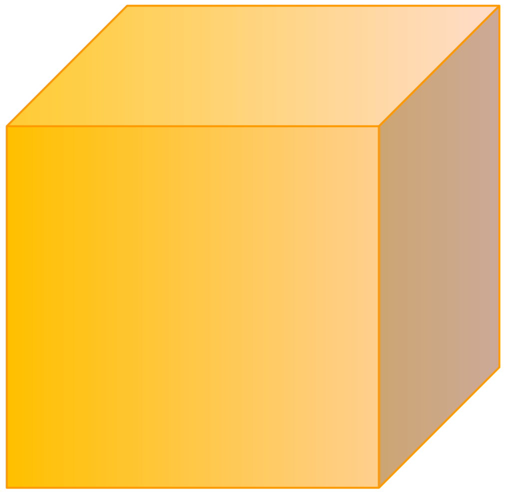
Neural network in training

Prediction at the end of each epoch

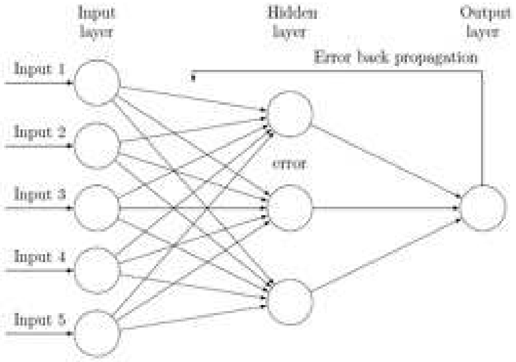
Validation accuracy

This data set must have the same distribution as real-life samples, or else validation accuracy won't reflect real-life accuracy.

Test



Test data set



Fully trained neural network

Prediction at the end of experimentation

Test accuracy

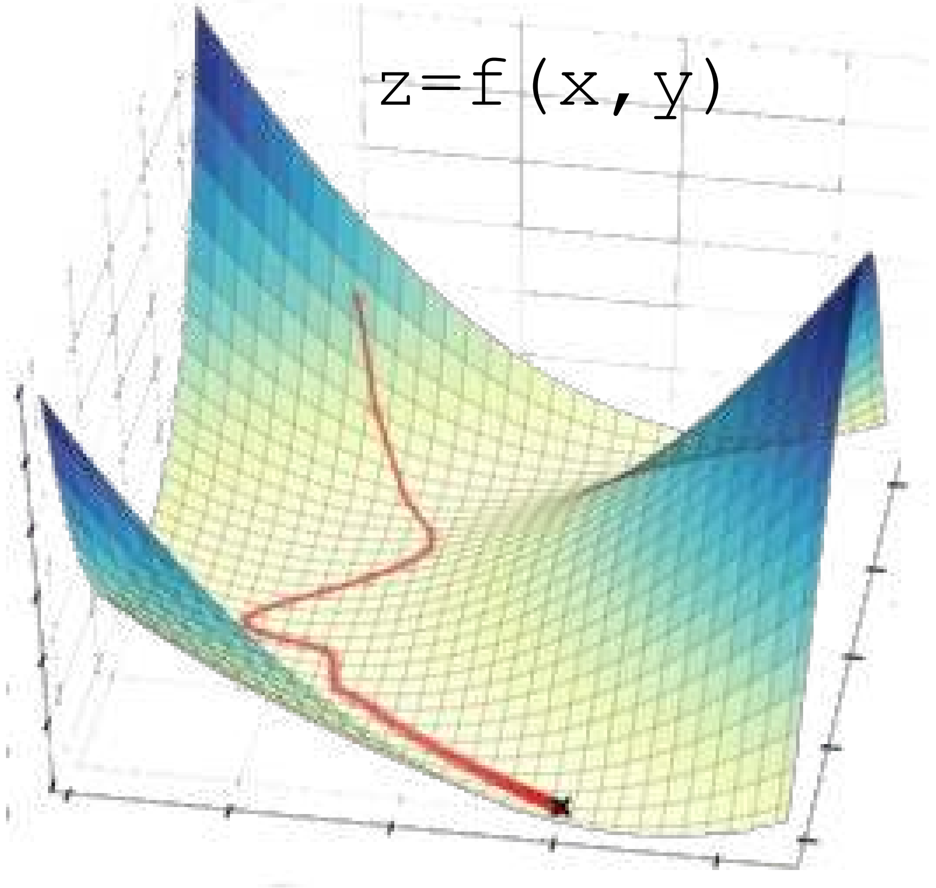
This data set must have the same distribution as real-life samples, or else test accuracy won't reflect real-life accuracy.

Stochastic Gradient Descent (1951)

Imagine you stand on top of a mountain (…). You want to get down to the valley as quickly as possible, but there is fog and you can only see your immediate surroundings. How can you get down the mountain as quickly as possible?

You look around and identify the steepest path down, go down that path for a bit, again look around and find the new steepest path, go down that path, and repeat—this is exactly what gradient descent does.

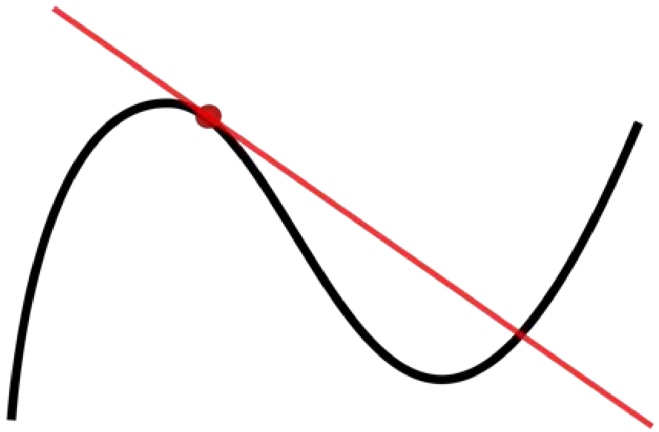
Tim Dettmers, University of Lugano, 2015

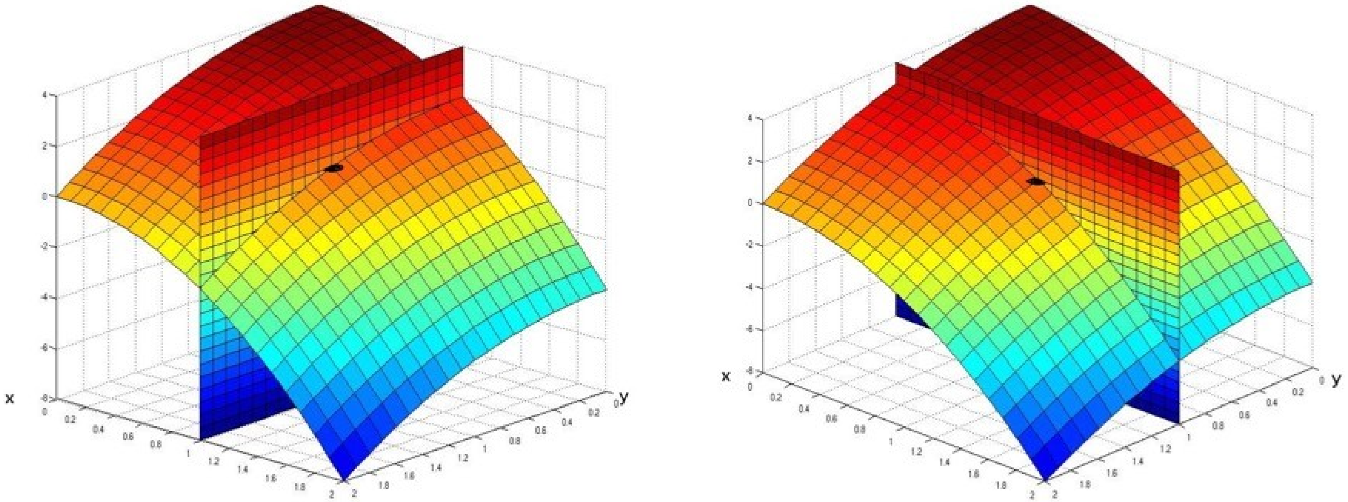


The « step size » depends on the learning rate

https://devblogs.nvidia.com/parallelforall/deep-learning-nutshell-history-training/

Finding the slope with Derivatives





Scalar-valued multivariable function

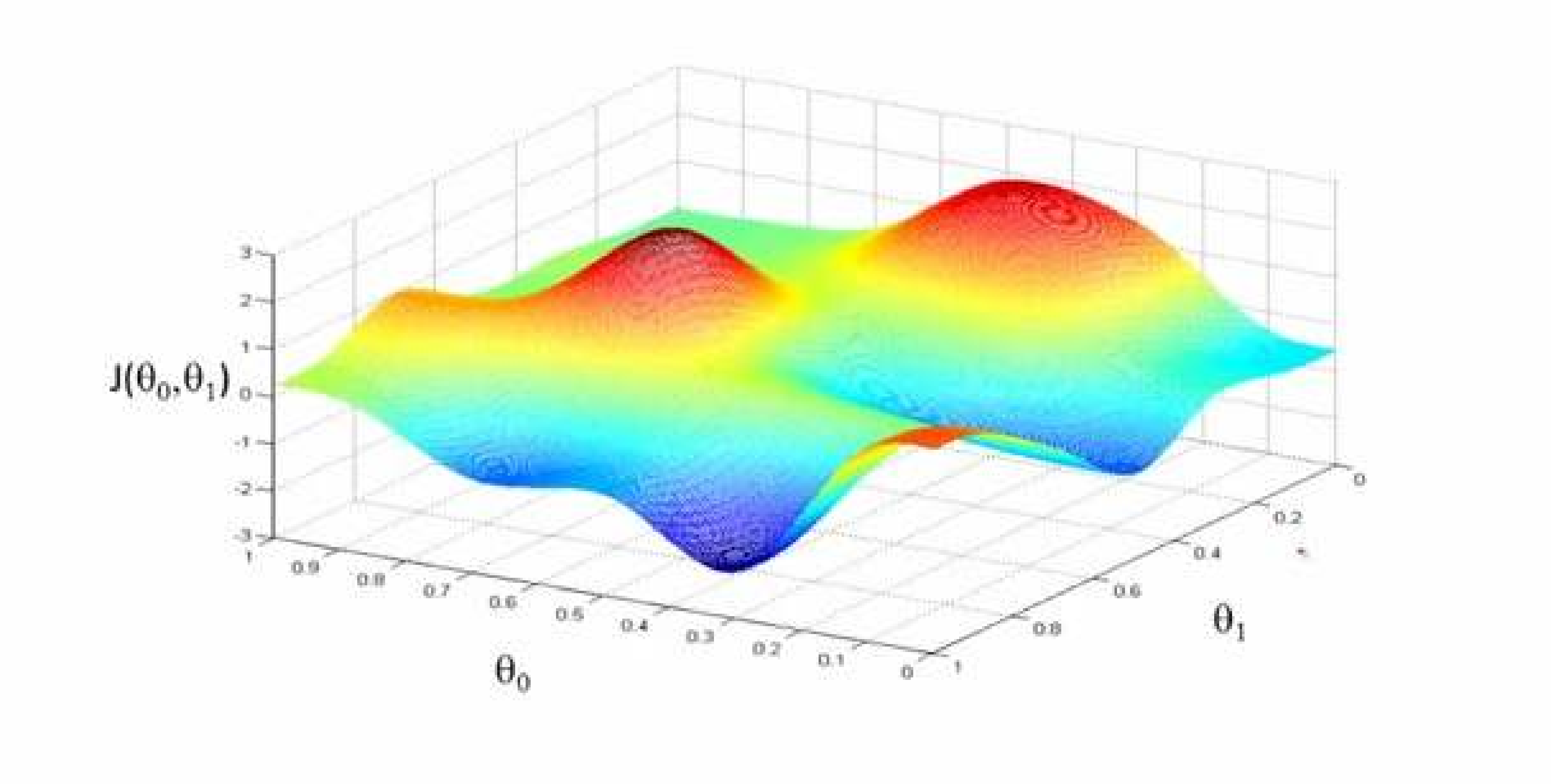
$$\nabla f(\underbrace{x_0, y_0, \dots}_{\substack{\nabla f \text{ takes the same} \\ \text{type of inputs as } f}}) = \underbrace{\begin{bmatrix} \frac{\partial f}{\partial x}(x_0, y_0, \dots) \\ \frac{\partial f}{\partial y}(x_0, y_0, \dots) \\ \vdots \end{bmatrix}}_{\substack{\nabla f \text{ outputs a vector with} \\ \text{all possible partial derivatives of } f.}}$$

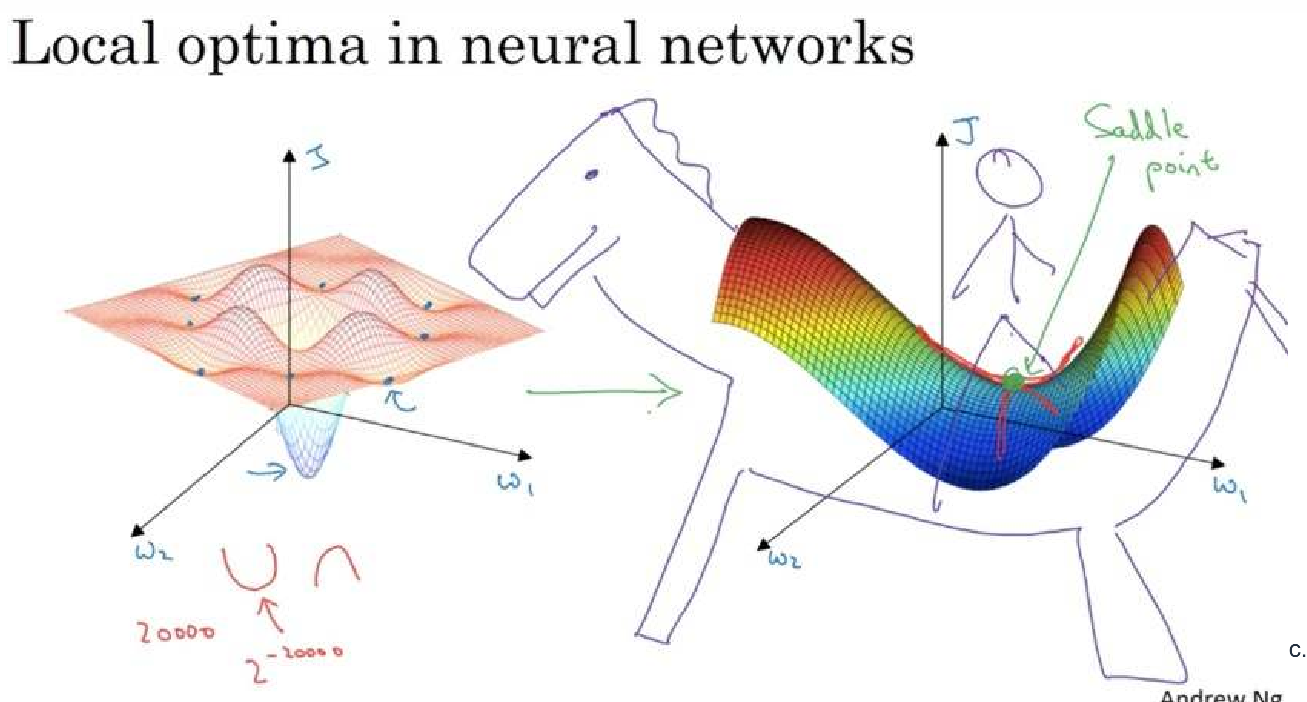
Notation for gradient, called "nabla".

End-to-end example of computing backpropagation with partial derivatives: https://mattmazur.com/2015/03/17/a-step-by-step ackpropagation-example

Source: Wikipedia, Oklahoma State University, Khan

Local minima and saddle points





> « *Do neural networks enter and escape a series of local minima? Do they move at varying speed as they approach and then pass a variety of saddle points? Answering these questions definitively is difficult, but we present evidence strongly suggesting that the answer to all of these questions is no.* »

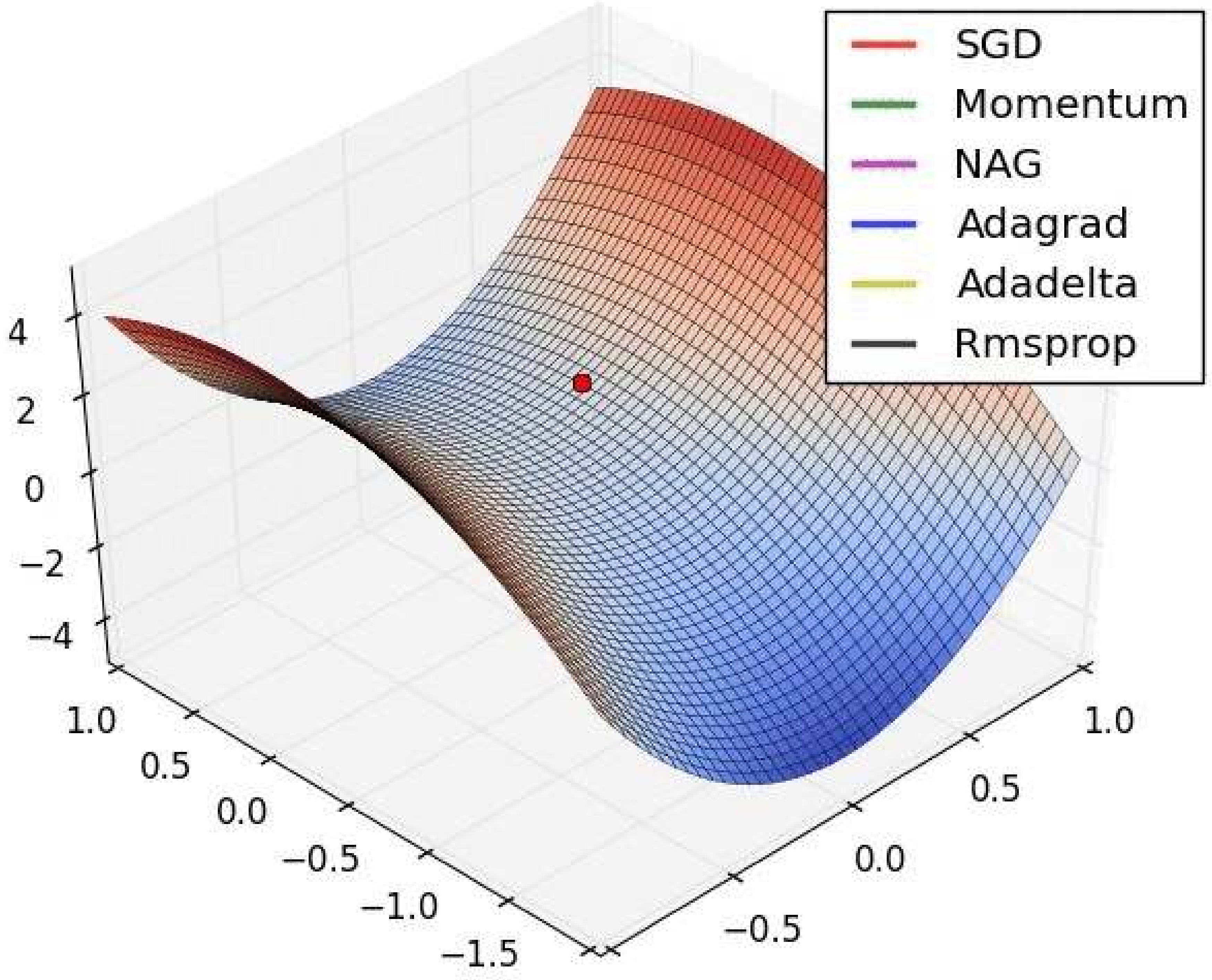
« Qualitatively characterizing neural network optimization problems », Goodfellow et al, 2015 https://arxiv.org/abs/1412.6544

Optimizers

SGD works remarkably well and is still widely used.

Adaptative optimizers use a variable learning rate.

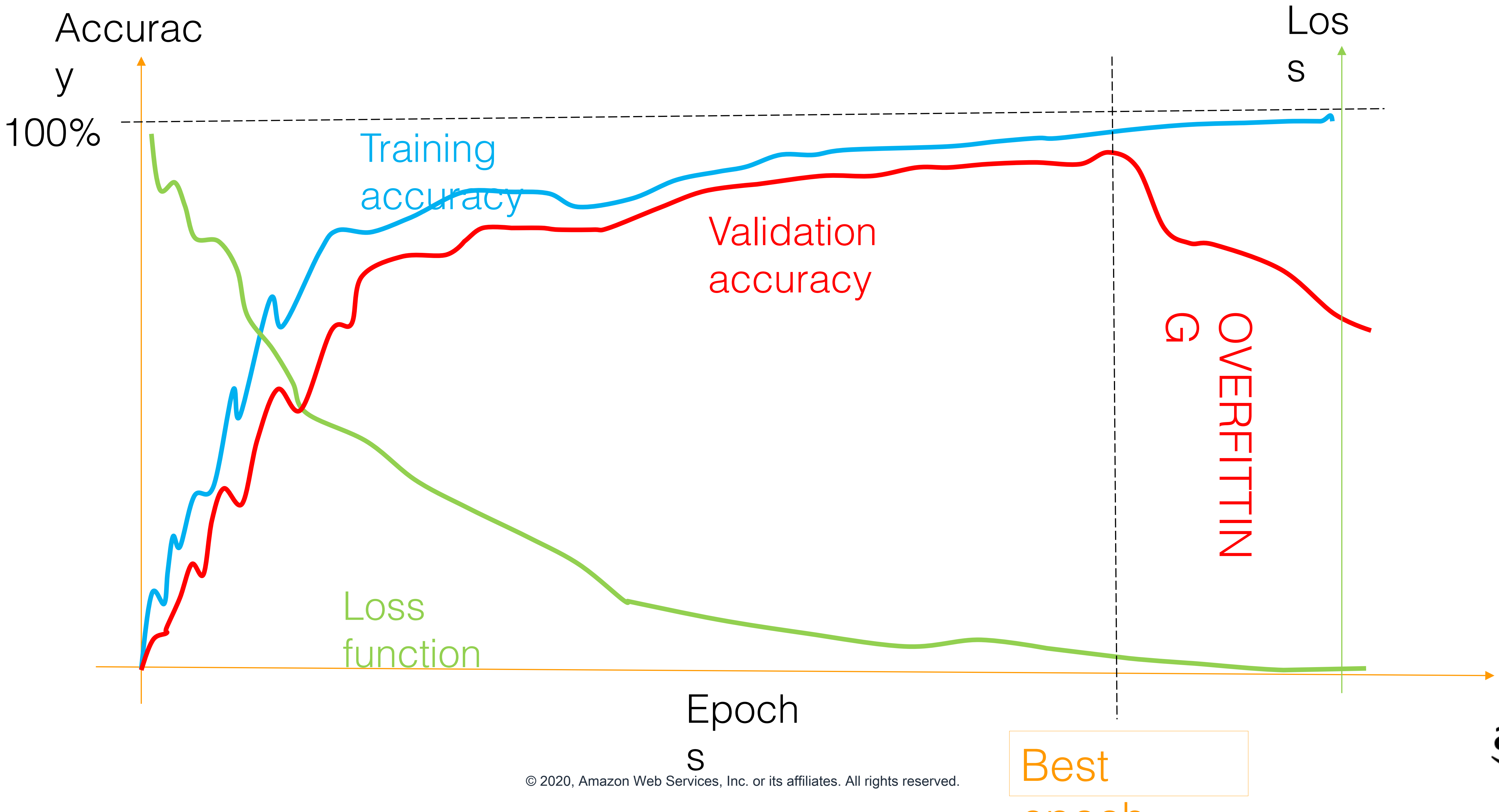
Some even use a learning rate per dimension (Adam).



https://medium.com/@julsimon/tumbling-down-the-sgd-rabbit-hole-part-1-740fa402f0d7

Summing things up

« Deep Learning ultimately is about finding a minimum that generalizes well, with bonus points for finding one fast and reliably », Sebastian Ruder



Demo

Common network architectures and use cases

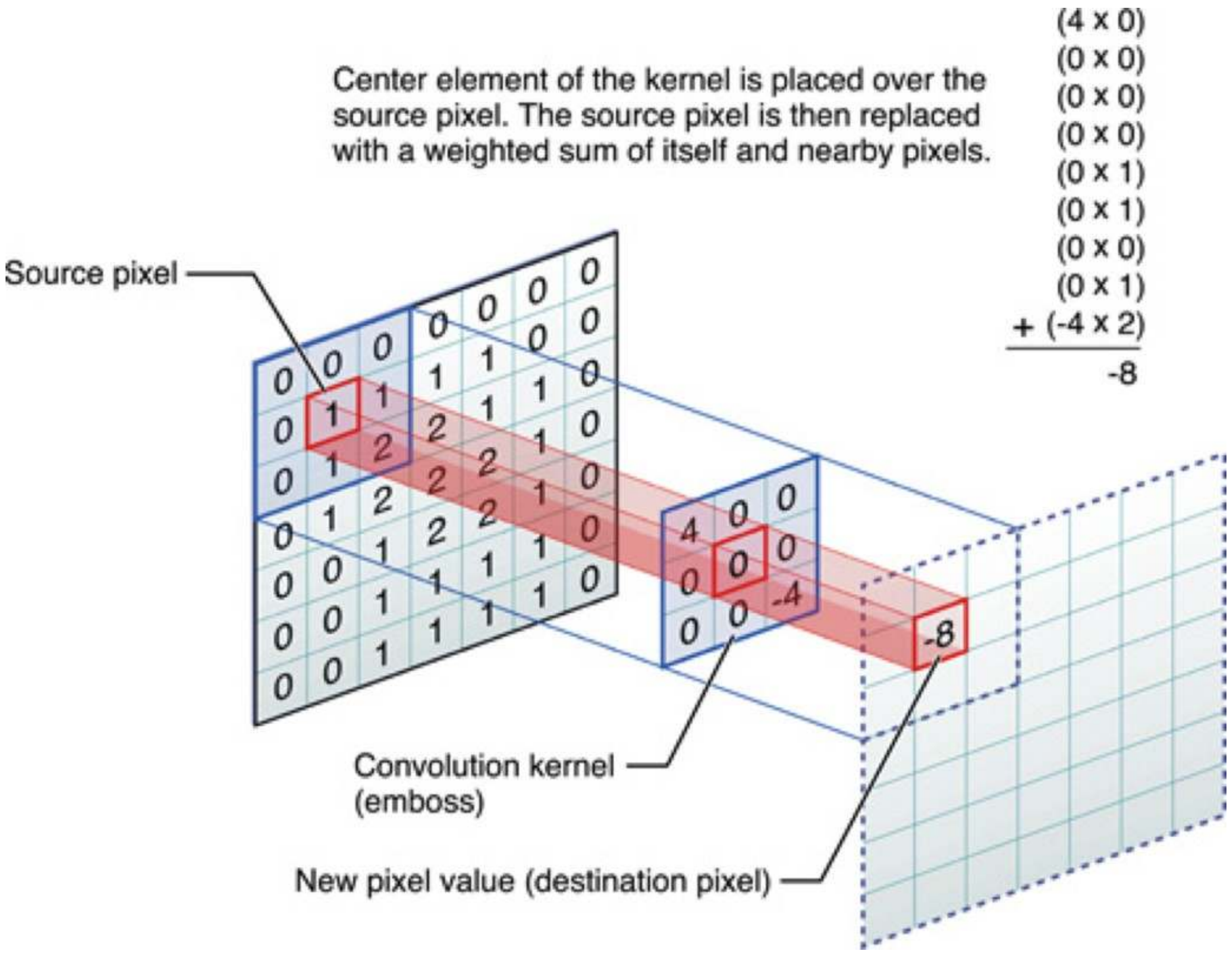
Fully Connected Networks are nice, but…

- What if we need lots of layers in order to extract complex features?
  - The number of parameters increases very quickly with the number of layers
  - Overfitting is a constant problem
- What about large data?
  - 256x256 images = 65,535 input neurons ?
- What about 2D/3D data ? Won't we lose lots of info by flattening it?
  - Images, videos, etc.
- What about sequential data, where the order of samples is important?
  - Translating text
  - Predicting time series

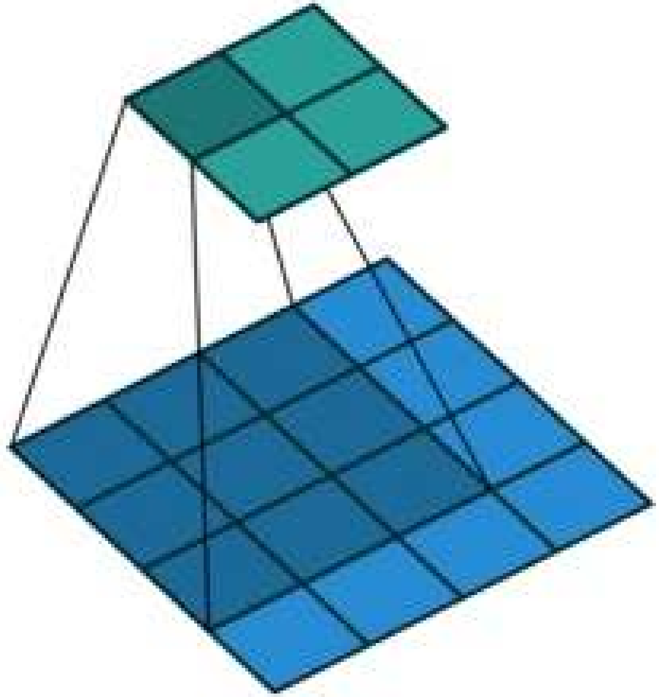
DEV DAY

Convolutional Neural Networks

The convolution operation



Source: https://ikhlestov.github.io/pages/machine-learning/convolutions-types/



Source: Theano documentation

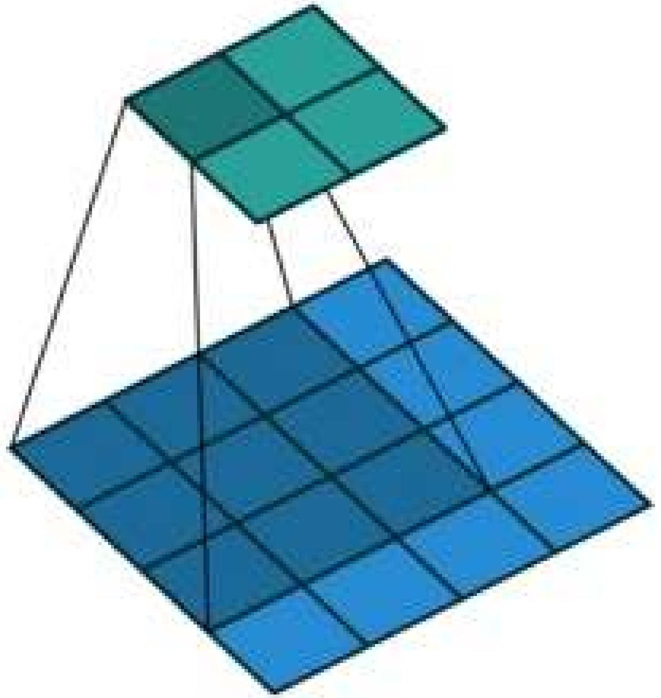
Extracting features with convolution

Input image

Convolution Kernel

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

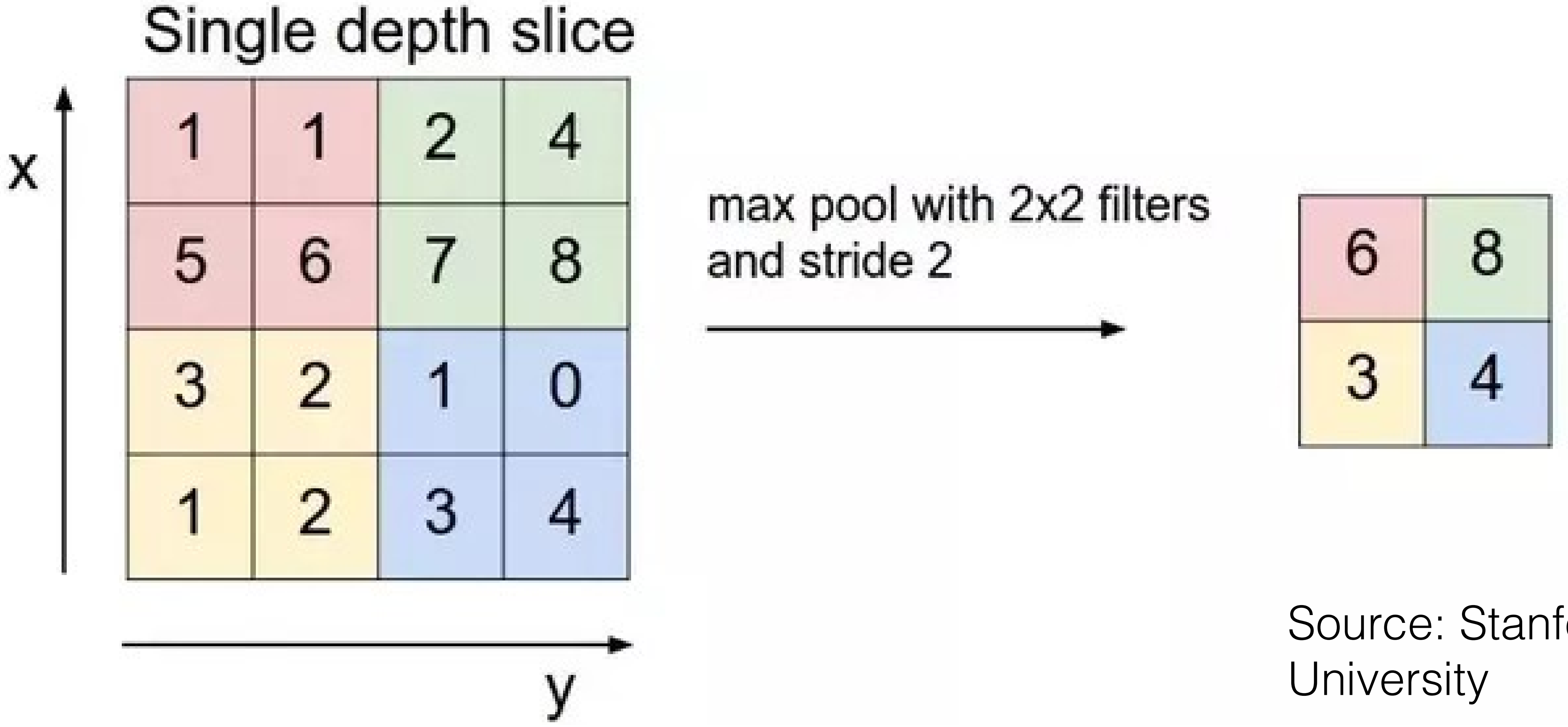
Feature map



Source: http://timdettmers.com

Convolution extracts features automatically.
Kernel parameters are learned during the training process.

Downsampling images with pooling

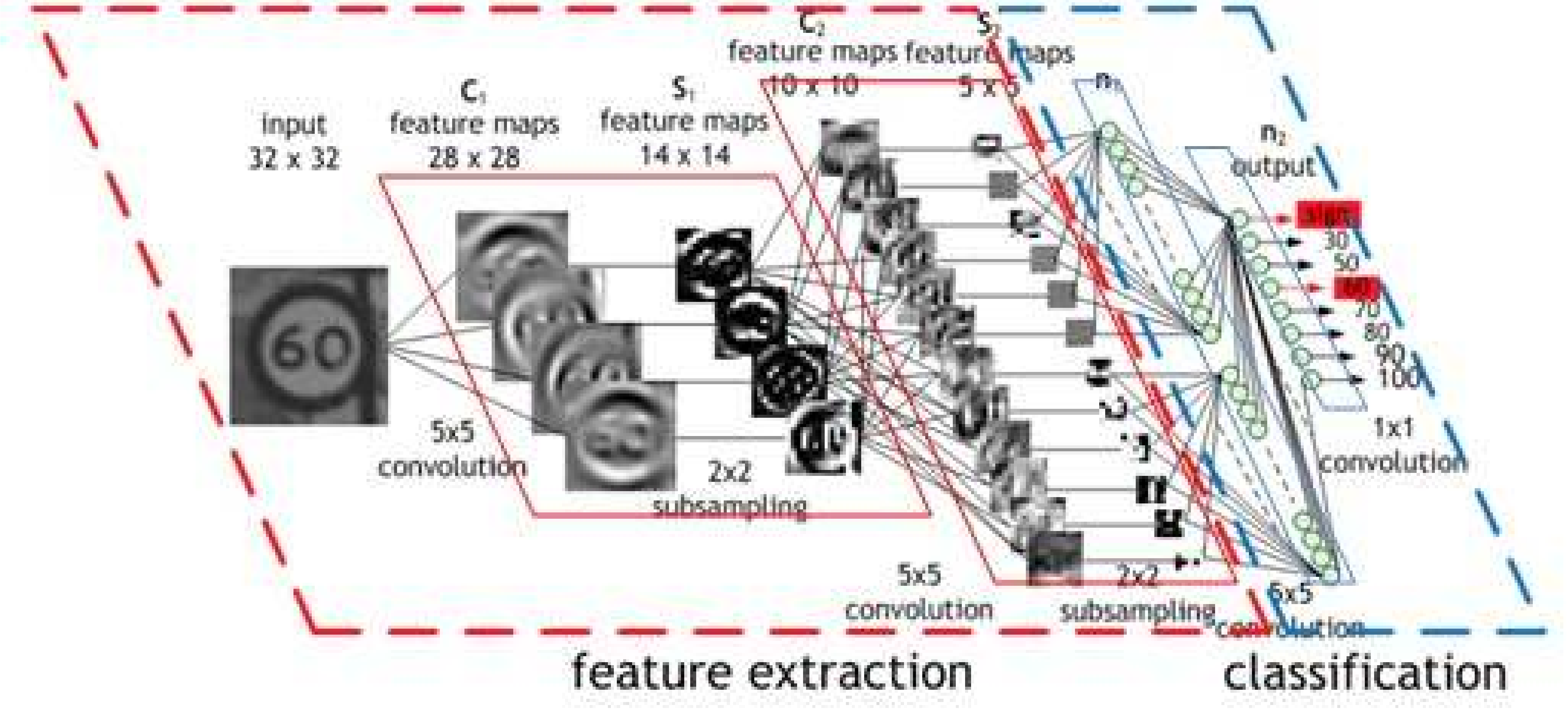


Source: Stanford University

Pooling shrinks images while preserving significant information.

Convolutional Neural Networks (CNN)

Le Cun, 1998: handwritten digit recognition, 32x32 pixels



https://devblogs.nvidia.com/parallelforall/deep-learning-nutshell-core-concepts/

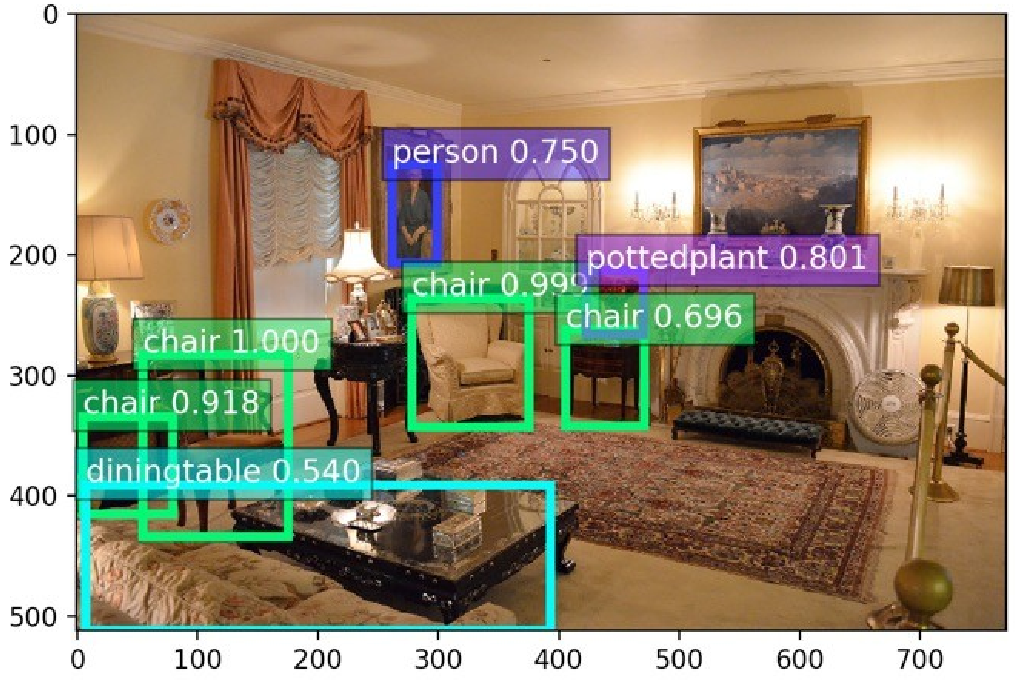
DEV DAY

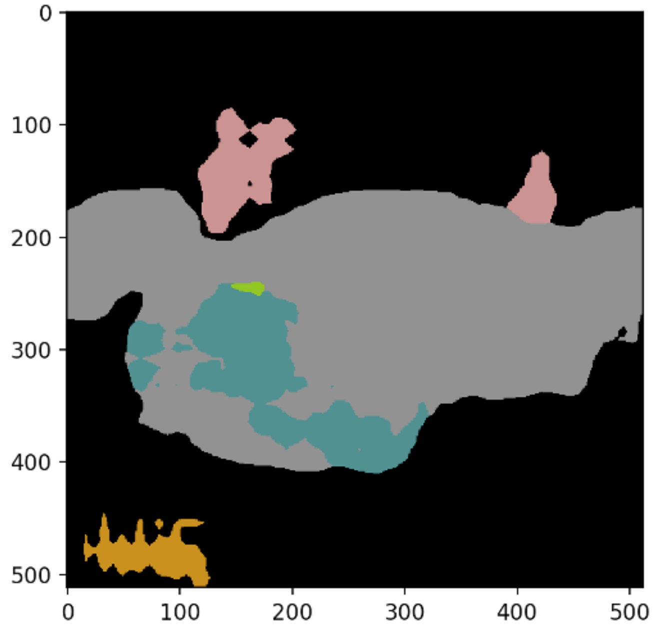
Demo

Gluon

Classification, detection, segmentation

[electric_guitar], with probability 0.671



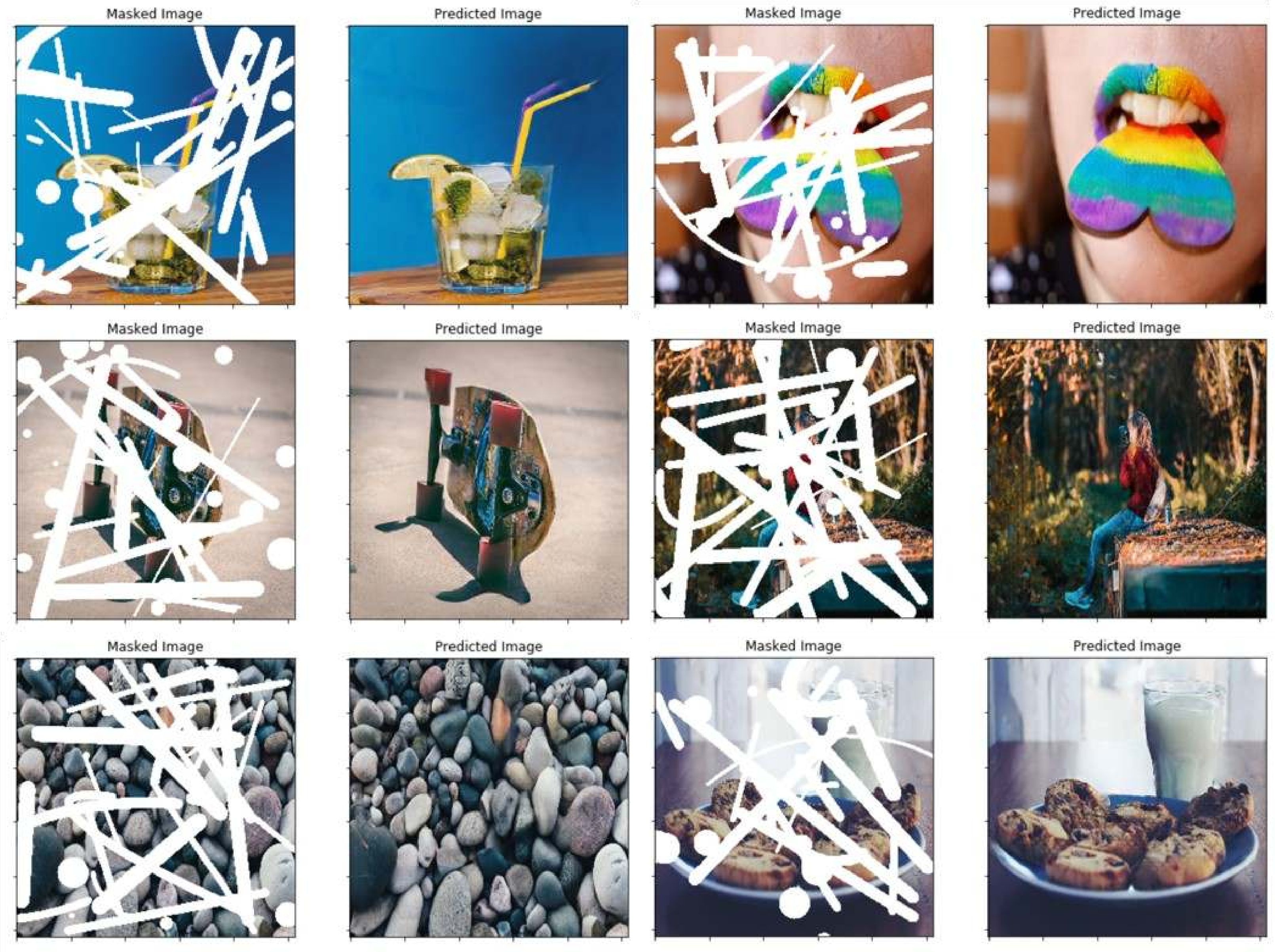


https://github.com/dmlc/gluon-cv
Based on models published in 2015-2017

Demo

Keras

Image Inpainting



https://github.com/MathiasGruber/PConv-Keras
https://arxiv.org/abs/1804.07723
April 2018

aws

Detectron2

https://github.com/facebookresearch/detectron2 November 2019

Demo on the AWS Deep Learning AMI https://www.youtube.com/watch?v=7R8-VAk0ruk

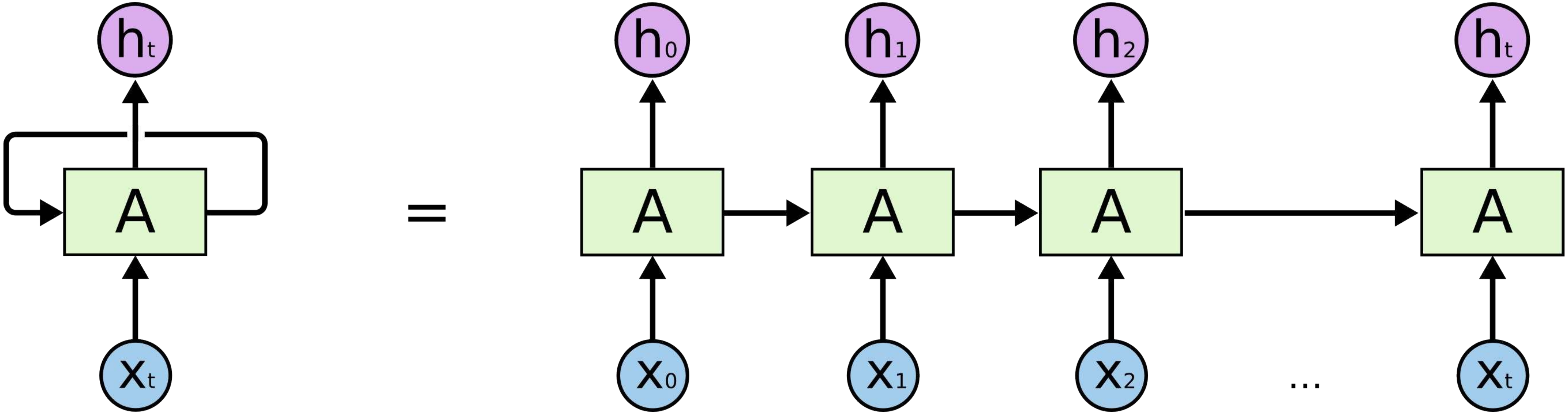
DEV DAY

Demo

DEV DAY

Recurrent Neural Networks

Recurrent Neural Networks (RNN)



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recurrent Neural Networks (RNN)

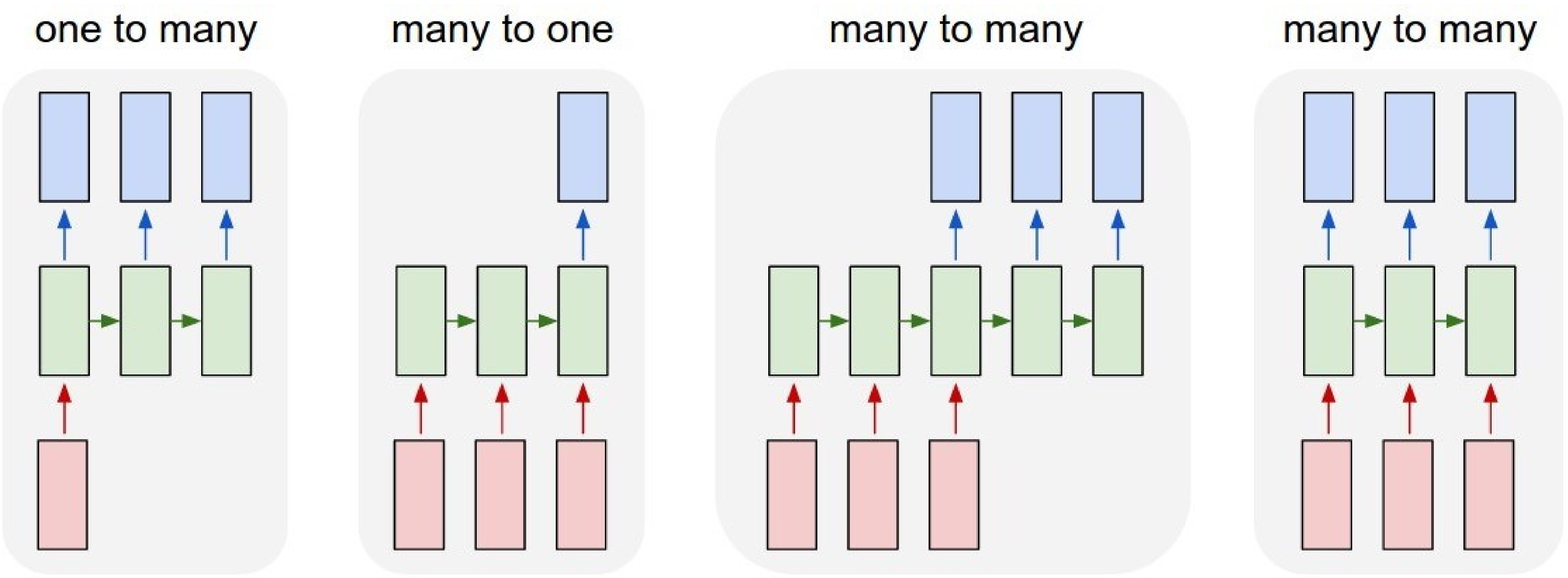


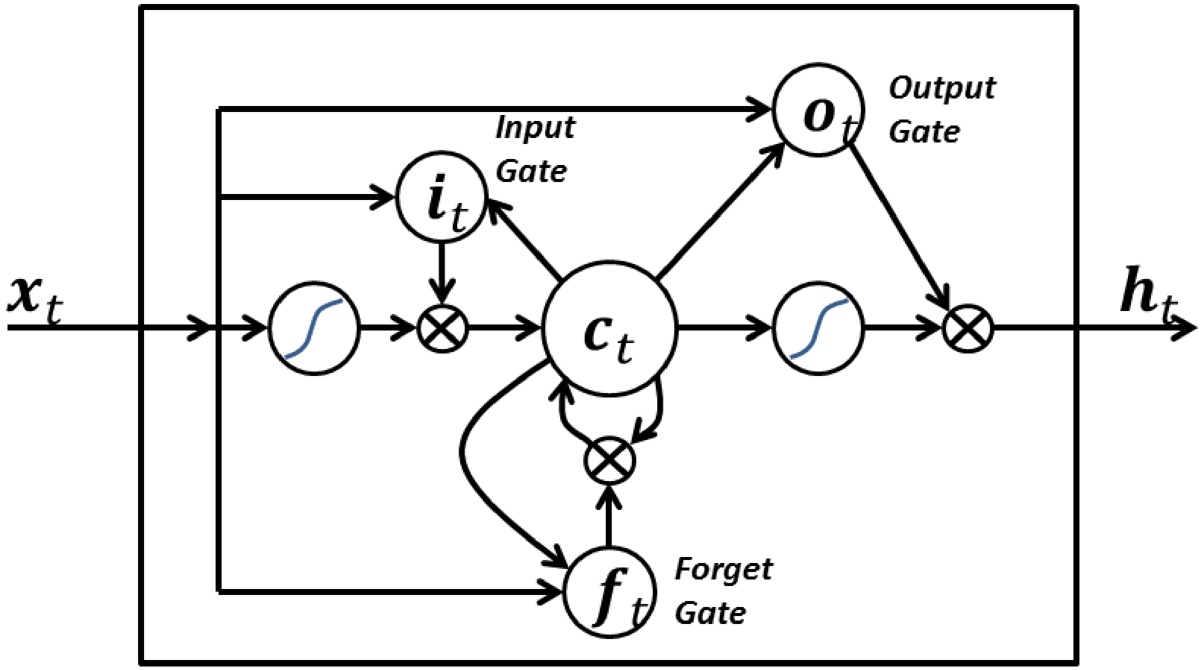
Image captionin g | Sentiment analysis | Machine translation | Video frame labeling

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Long Short Term Memory Networks (LSTM)

Hochreiter and Schmidhuber, 1997

- A LSTM neuron computes the output based on the input and a previous state
- LSTM neurons have « short-term memory »
- They do a better job than RNN at predicting longer sequences of data (i.e. hundreds of steps)
- Gated Recurrent Units aka GRU (Cho et al. 2014) are easier to train, and just as efficient.



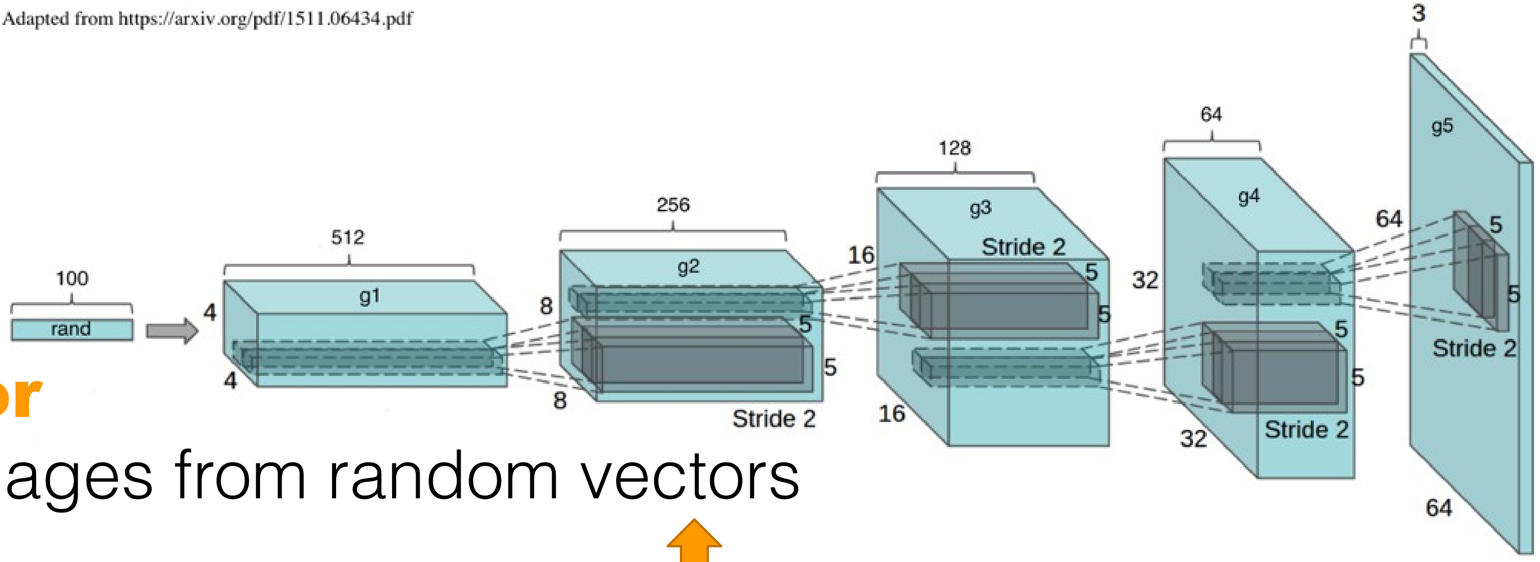
DEV DAY

Generative Adversarial Networks

Generative Adversarial Networks

Goodfellow, 2014 https://arxiv.org/abs/1406.2661

Adapted from https://arxiv.org/pdf/1511.06434.pdf



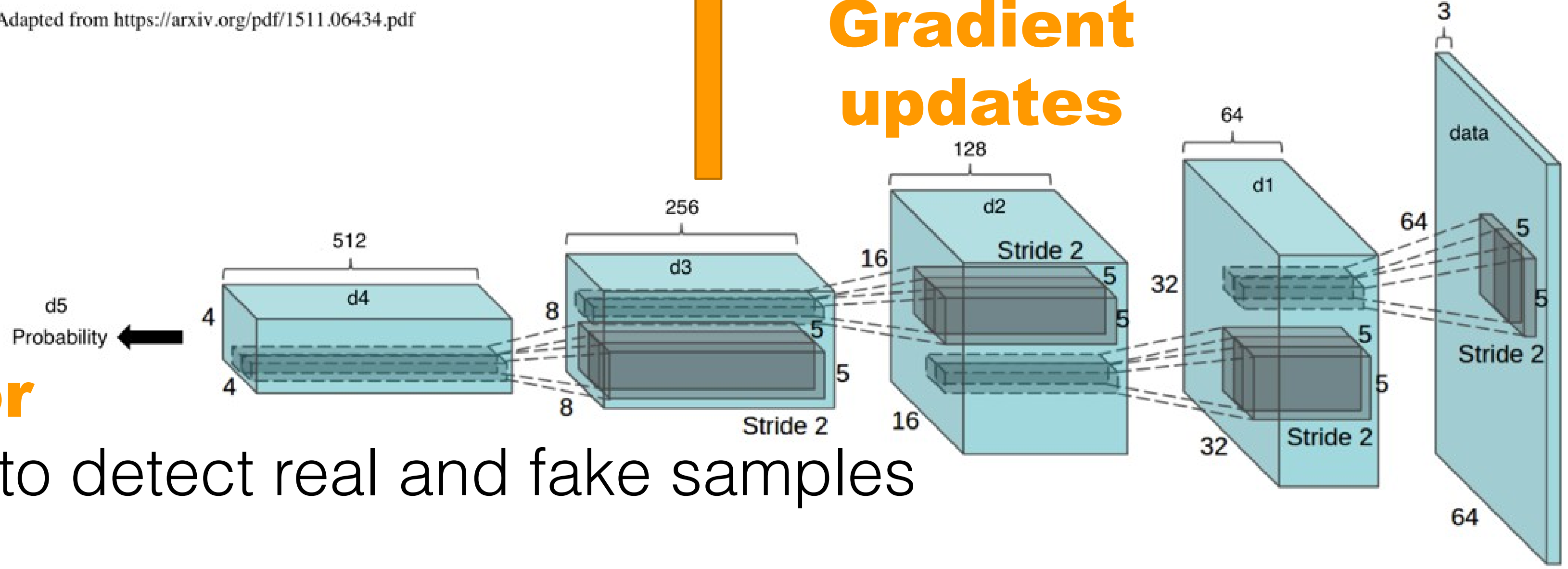
Generator

Building images from random vectors

Fake images

Gradient updates

Adapted from https://arxiv.org/pdf/1511.06434.pdf



Real images

Detector

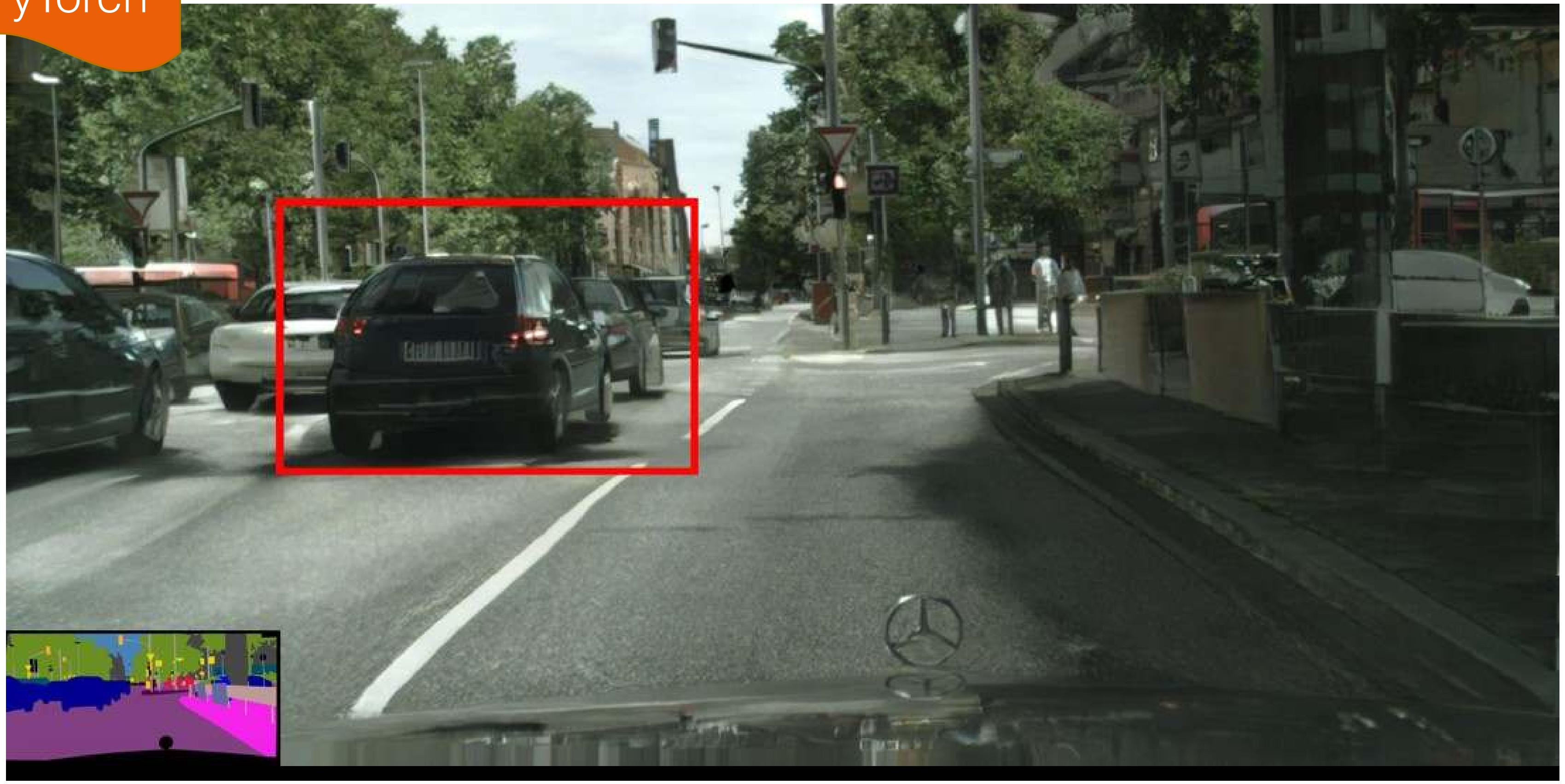
Learning to detect real and fake samples

https://medium.com/@julsimon/generative-adversarial-networks-on-apache-mxnet-part-1-b6d39e6b5df1

GAN: Welcome to the (un)real world, Neo

TF

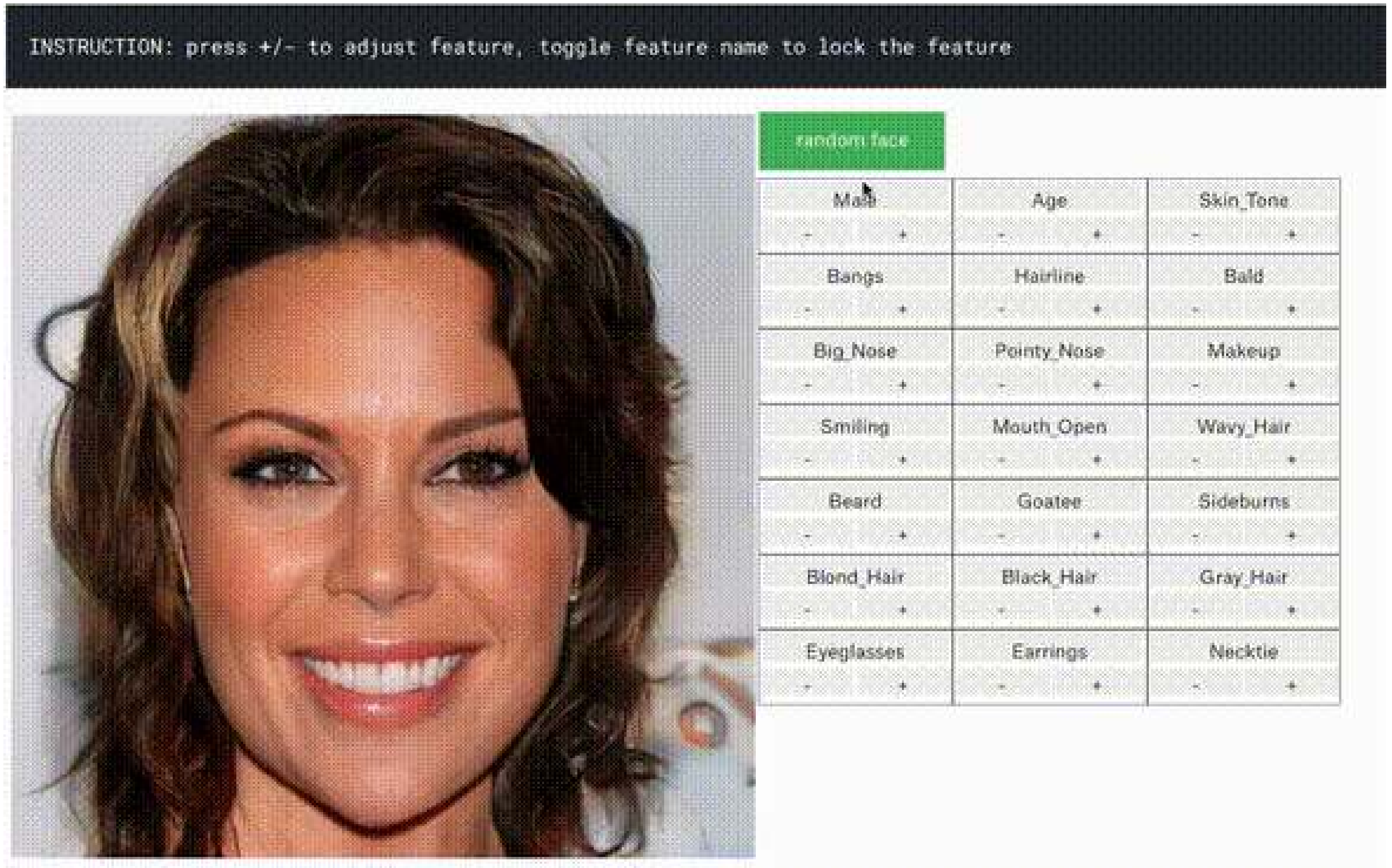
PyTorch



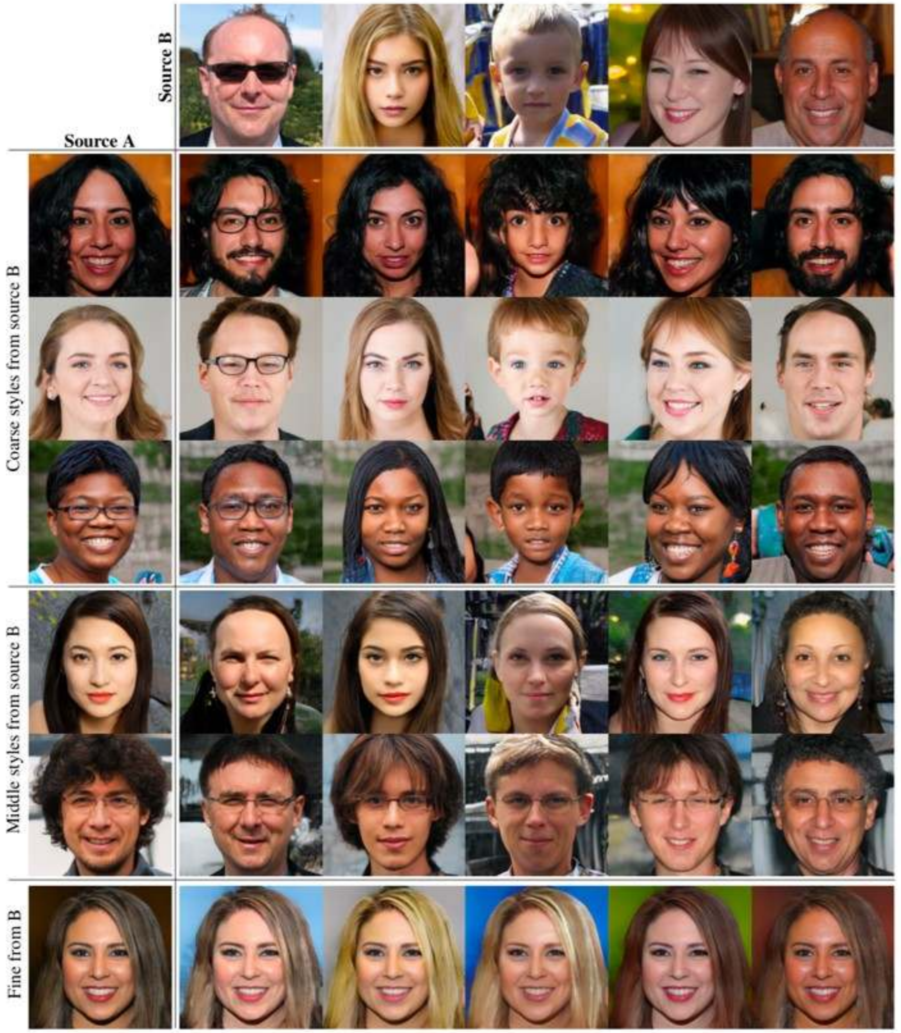
Generating new "celebrity" faceshttps://github.com/tkarras/progressive_growing_of_gans
April 2018

From semantic map to 2048x1024 picture
https://tcwang0509.github.io/pix2pixHD/
November 2017

More face generation with GANs



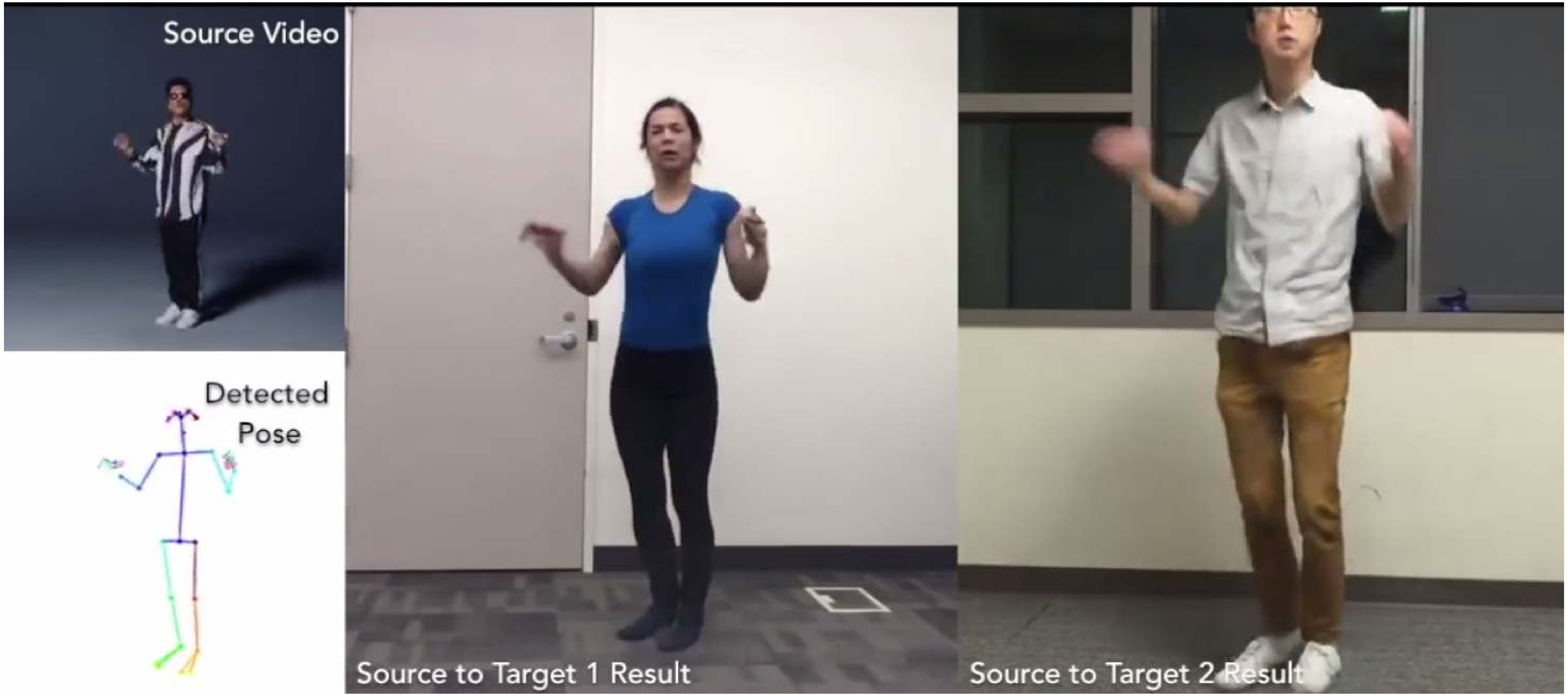
Controlled Image Generation with TL-GAN
https://github.com/SummitKwan/transparent_latent_gan
October 2018



TF

Applying the style of a face to another face https://github.com/NVlabs/stylegan
https://www.youtube.com/watch?v=kSLJriaOumA
March 2019

GAN: Everybody dance now



https://arxiv.org/abs/1808.07371
https://www.youtube.com/watch?v=PCBTZh41Ris
August 2018

DEV DAY

Getting started

Resources

https://aws.training/machinelearning
https://deeplearning.ai
https://fast.ai

http://www.deeplearningbook.org/
https://d2l.ai/

https://gluon.mxnet.io
https://keras.io

https://gitlab.com/juliensimon/dlnotebooks

Thank you!

Appendix – Apache MXNet demos

Demo – Image classification: using a pre-trained model

```
*** VGG16
[(0.46811387, 'n04296562 stage'), (0.24333163,
'n03272010 electric guitar'), (0.045918692, 'n02231487
walking stick, walkingstick, stick insect'),
(0.03316205, 'n04286575 spotlight, spot'),
(0.021694135, 'n03691459 loudspeaker, speaker, speaker
unit, loudspeaker system, speaker system')]
*** ResNet-152
[(0.8726753, 'n04296562 stage'), (0.046159592,
'n03272010 electric guitar'), (0.041658506, 'n03759954
microphone, mike'), (0.018624334, 'n04286575 spotlight,
spot'), (0.0058045341, 'n02676566 acoustic guitar')]
*** Inception v3
[(0.44991142, 'n04296562 stage'), (0.43065304,
'n03272010 electric guitar'), (0.067580454, 'n04456115
torch'), (0.012423956, 'n02676566 acoustic guitar'),
(0.0093934005, 'n03250847 drumstick')]
```

https://medium.com/@julsimon/an-introduction-to-the-mxnet-api-part-5-9e78534096db

Demo – Image classification: fine-tuning a model

CIFAR-10 data set

60,000 images in 10 classes

32x32 color images

Initial training

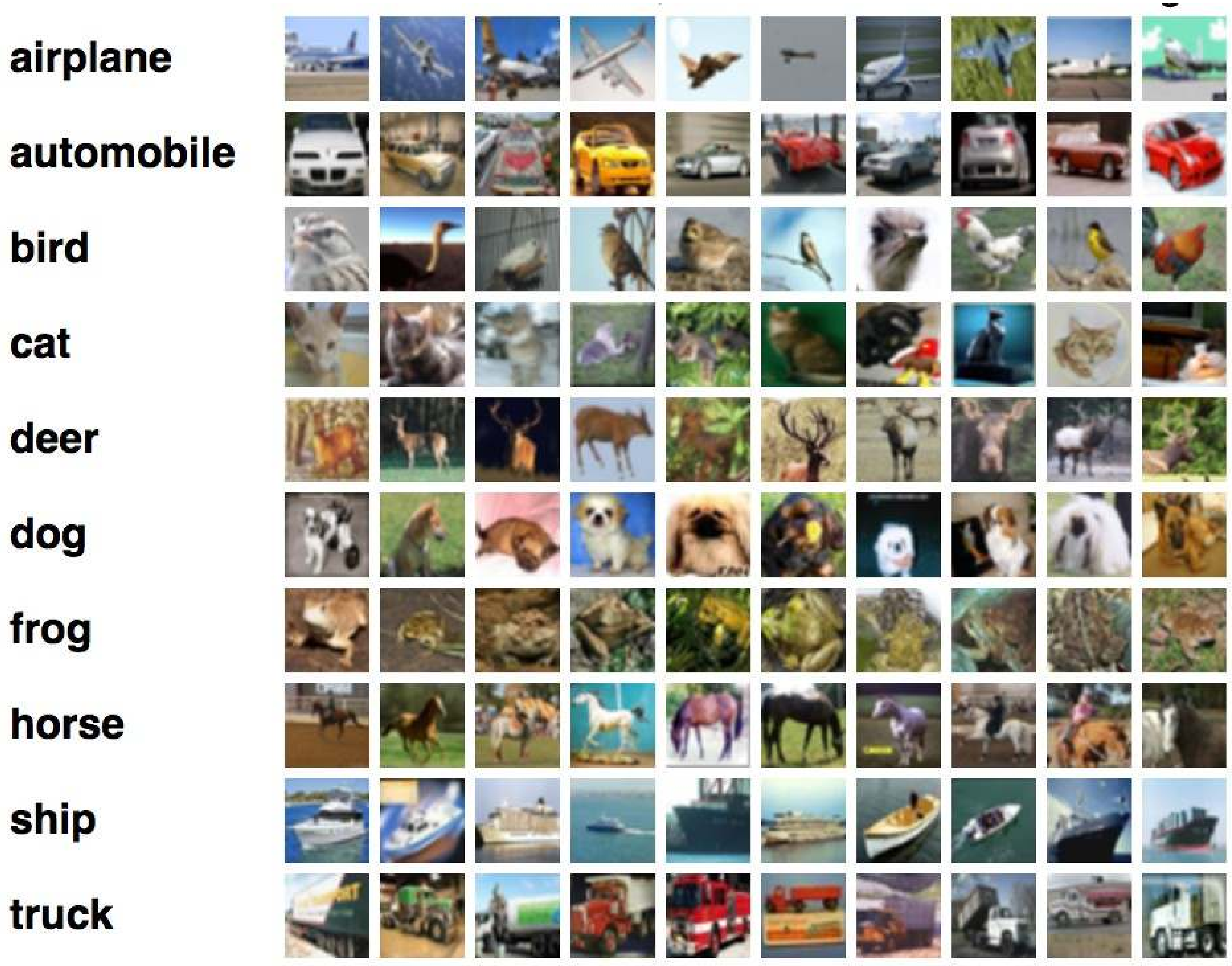
Resnet-50 CNN

200 epochs

82.12% validation

Cars vs. horses

88.8% validation accuracy



https://medium.com/@julsimon/keras-shoot-out-part-3-fine-tuning-7d1548c51a41

Demo – Image classification: fine-tuning a model

Freezing all layers but the last one

Fine-tuning on « cars vs. horses » for 10 epochs

2 minutes on 1 GPU

98.8% validation accuracy

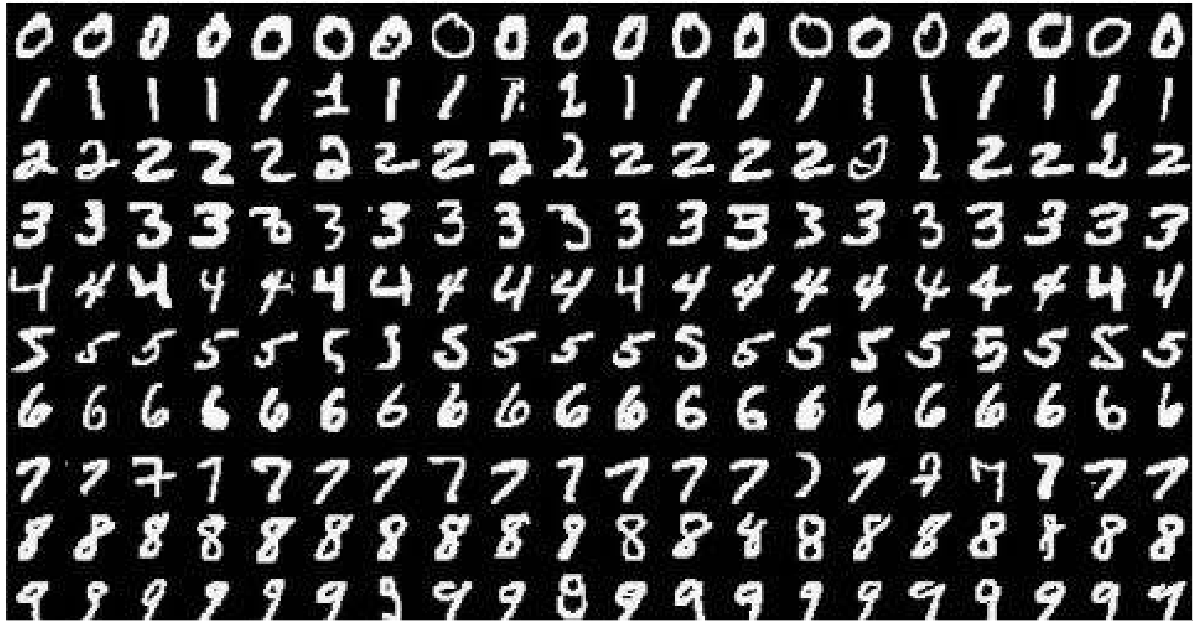
```
Epoch 10/10
10000/10000 [==============================] - 12s
loss: 1.6989 - acc: 0.9994 - val_loss: 1.7490 - val_acc: 0.9880

2000/2000 [==============================] - 2s
[1.7490020694732666, 0.98799999999999999]
```

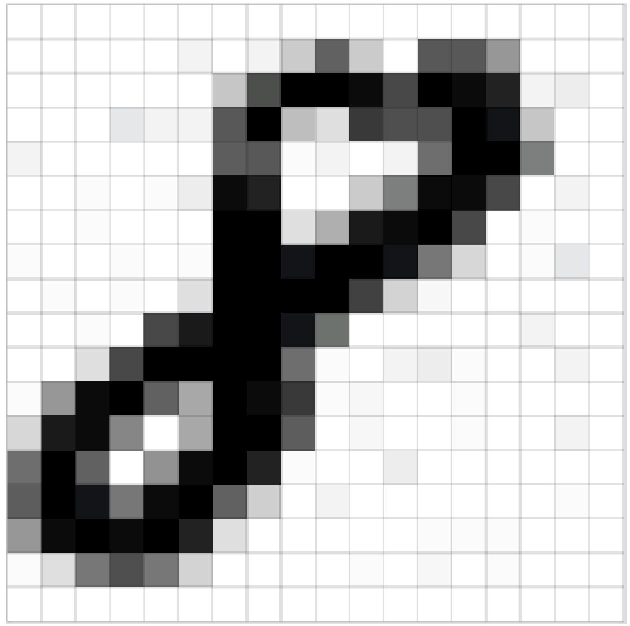
Demo – Image classification: learning from scratch

MNIST data set

70,000 hand-written digits

28x28 grayscale images



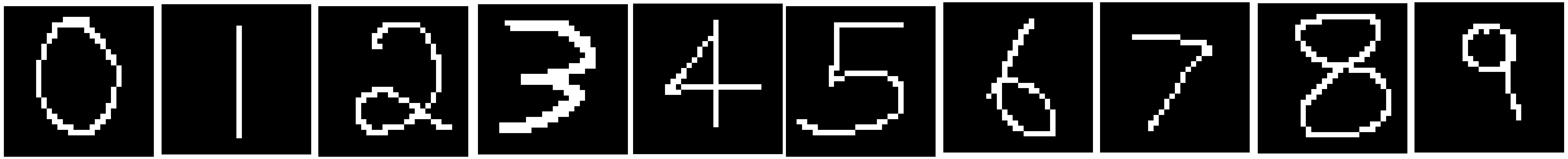


0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	12	0	11	39	137	37	0	152	147	84	0	0	0
0	0	1	0	0	0	41	160	250	255	235	162	255	238	206	11	13	0
0	0	0	16	9	9	150	251	45	21	184	159	154	255	233	40	0	0
10	0	0	0	0	0	145	146	3	10	0	11	124	253	255	107	0	0
0	0	3	0	4	15	236	216	0	0	38	109	247	240	169	0	11	0
1	0	2	0	0	0	253	253	23	62	224	241	255	164	0	5	0	0
6	0	0	4	0	3	252	250	228	255	255	234	112	28	0	2	17	0
0	2	1	4	0	21	255	253	251	255	172	31	8	0	1	0	0	0
0	0	4	0	163	225	251	255	229	120	0	0	0	0	0	11	0	0
0	0	21	162	255	255	254	255	126	6	0	10	14	6	0	0	9	0
3	79	242	255	141	66	255	245	189	7	8	0	0	5	0	0	0	0
26	221	237	98	0	67	251	255	144	0	8	0	0	7	0	0	11	0
125	255	141	0	87	244	255	208	3	0	0	13	0	1	0	1	0	0
145	248	228	116	235	255	141	34	0	11	0	1	0	0	0	1	3	0
85	237	253	246	255	210	21	1	0	1	0	0	6	2	4	0	0	0
6	23	112	157	114	32	0	0	0	0	2	0	8	0	7	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

https://medium.com/@julsimon/training-mxnet-part-1-mnist-6f0dc4210c62

Multi-Layer Perceptron vs. Handmade-Digits-From-Hell™

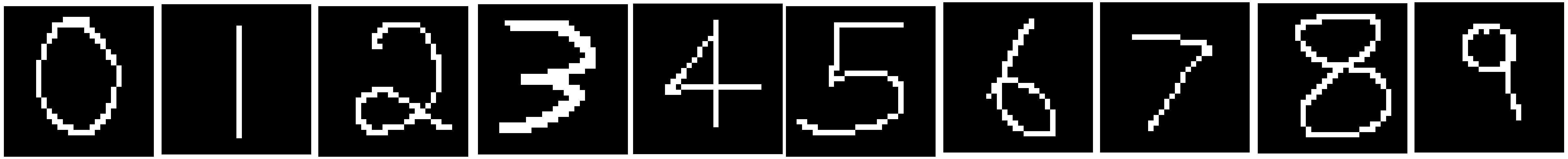
784/128/64/10, Relu, AdaGrad, 100 epochs → 97.51% validation accuracy



```
[[ 0.839   0.034   0.039   0.009   0.      0.008   0.066   0.002   0.      0.004]]
[[ 0.      0.988   0.001   0.003   0.001   0.001   0.002   0.003   0.001   0.002]]
[[ 0.006   0.01    0.95    0.029   0.      0.001   0.004   0.      0.      0.]]
[[ 0.      0.      0.      1.      0.      0.      0.      0.      0.      0.]]
[[ 0.      0.001   0.005   0.001   0.982   0.001   0.      0.007   0.      0.002]]
[[ 0.001   0.001   0.      0.078   0.      0.911   0.01    0.      0.      0.]]
[[ 0.003   0.      0.019   0.      0.005   0.004   0.863   0.      0.105   0.001]]
[[ 0.001   0.008   0.098   0.033   0.      0.      0.      0.852   0.004   0.004]]
[[ 0.001   0.      0.006   0.      0.      0.001   0.002   0.      0.991   0.]]
[[ 0.002   0.158   0.007   0.117   0.082   0.001   0.      0.239   0.17    0.224]]
```

LeNet vs. Handmade-Digits-From-Hell™

ReLu instead of tanh, 20 epochs, AdaGrad → 99.20% validation accuracy



```
[[ 1.      0.      0.     0.     0.     0.     0.     0.     0.     0.]]
[[ 0.      1.      0.     0.     0.     0.     0.     0.     0.     0.]]
[[ 0.      0.      1.     0.     0.     0.     0.     0.     0.     0.]]
[[ 0.      0.      0.     1.     0.     0.     0.     0.     0.     0.]]
[[ 0.      0.      0.001  0.     0.998  0.     0.     0.001  0.     0.]]
[[ 0.      0.      0.     0.     0.     1.     0.     0.     0.     0.]]
[[ 0.      0.      0.     0.     0.     0.     1.     0.     0.     0.]]
[[ 0.      0.      0.     0.001  0.     0.     0.     0.999  0.     0.]]
[[ 0.      0.     0.006   0.     0.     0.     0.     0.     0.994  0.]]
[[ 0.      0.      0.     0.001  0.001  0.     0.     0.001  0.001  0.996]]
```