PATTERN RECOGNITION AND MACHINE LEARNING

Slide Set 6: Neural Networks and Deep Learning

February 2017

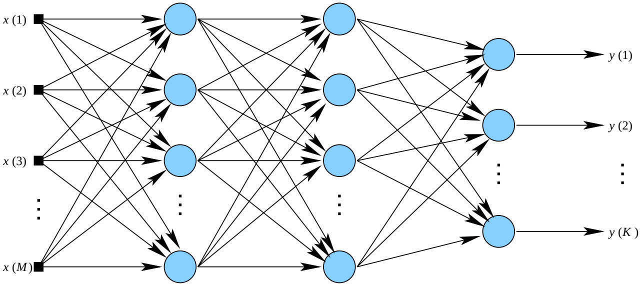
Heikki Huttunen

heikki.huttunen@tut.fi

Department of Signal Processing
Tampere University of Technology

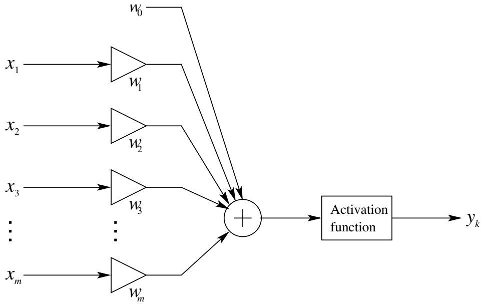
Traditional Neural Networks

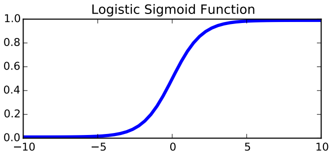
- Neural networks have been studied for decades.
- Traditional networks were *fully connected* (also called *dense*) networks consisting of typically 1-3 layers.
- Input dimensions were typically in the order of few hundred from a few dozen categories.
- Today, input may be 10k...100k variables from 1000 classes and network may have over 1000 layers.

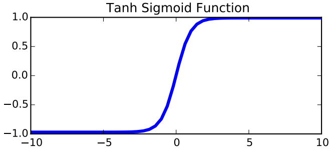


Traditional Neural Networks

- The neuron of a vanilla network is illustrated below.
- In essence, the neuron is a dot product between the inputs $\mathbf{x} = (1, x_1, \ldots, x_n)$ and weights $\mathbf{w} = (w_0, w_1, \ldots, w_n)$ followed by a nonlinearity, most often *logsig* or *tanh*.



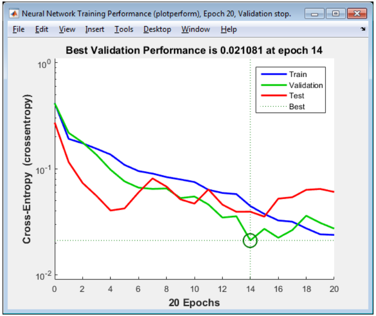




- In other words: this is *logistic regression* model, and the full net is just a stack of logreg models.

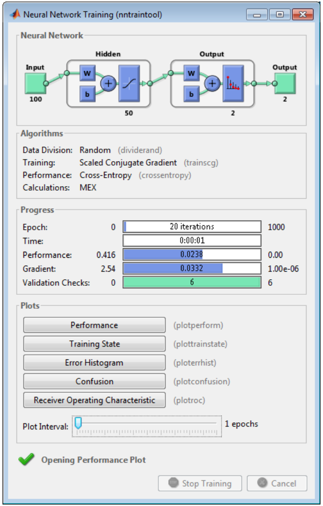
Training the Net

- Earlier, there was a lot of emphasis on training algorithms: *conjugate gradient, Levenberg-Marquardt, etc.*
- Today, people mostly use *stochastic gradient descent*.



Neural Network Training Performance (plotperform), Epoch 20, Validation stop.

Best Validation Performance is 0.021081 at epoch 14



Neural Network Training (nntraintool)

Algorithms

Data Division: Random (dividerand)

Training: Scaled Conjugate Gradient (trainscg)

Performance: Cross-Entropy (crossentropy)

Calculations: MEX

Progress

```
Epoch:               0   20 iterations   1000
Time:                    0:00:01
Performance:     0.416   0.0238          0.00
Gradient:         2.54   0.0332          1.00e-06
Validation Checks:   0   6               6
```

Plots

Plot Interval: 1 epochs

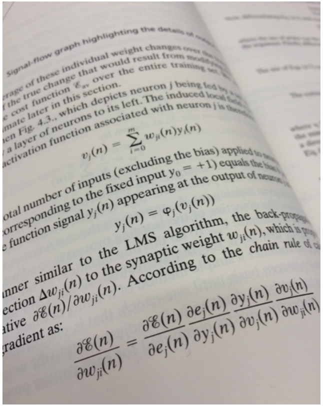
✓ Opening Performance Plot

Backpropagation

- The network is trained by adjusting the weights according to the partial derivatives

$$w_{ij} \leftarrow w_{ij} - \eta \frac{\partial \mathcal{E}}{\partial w_{ij}}$$

- In other words, the j^{th} weight of the i^{th} node steps towards the negative gradient with step size $\eta > 0$.
- In the 1990's the network structure was rather fixed, and the formulae would be derived by hand.
- Today, the same principle applies, but the exact form is computed symbolically.



Signal-flow graph highlighting the details of

erage of these individual weight changes over the

f the true change that would result from modifying

e cost function $\mathscr{E}_{av}$ over the entire training set.

mate later in this section.

en Fig. 4.3., which depicts neuron j being fed by a

a layer of neurons to its left. The induced local field

activation function associated with neuron j is there

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n)$$

otal number of inputs (excluding the bias) applied to

orresponding to the fixed input $y_0 = +1$) equals the bias

e function signal $y_j(n)$ appearing at the output of neuron

$$y_j(n) = \varphi_j(v_j(n))$$

nner similar to the LMS algorithm, the back-propaga

ection $\Delta w_{ji}(n)$ to the synaptic weight $w_{ji}(n)$, which is prop

ative $\partial\mathscr{E}(n)/\partial w_{ji}(n)$. According to the chain rule of

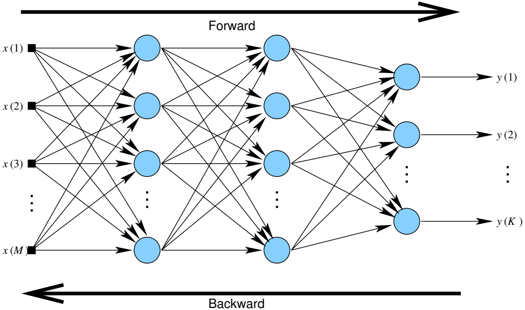
radient as:

$$\frac{\partial \mathscr{E}(n)}{\partial w_{ji}(n)} = \frac{\partial \mathscr{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)}$$

Backpropagation in Haykin: Neural networks, 1999.

Forward and Backward

- Training has two passes: *forward pass* and *backward pass*.
- The forward pass feeds one (or more) samples to the net.
- The backward pass computes the (mean) error and propagates the gradients back adjusting the weights one at a time
- When all samples are shown to the net, one *epoch* has passed. Typically the network runs for thousands of epochs.

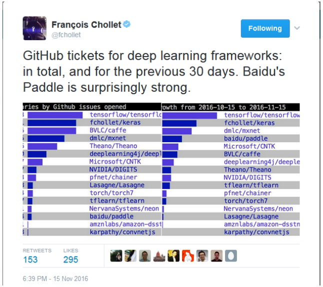


Neural Network Software

- Several packages exist:
  - **Matlab NN Toolbox**: Obsolete.
  - **Caffe**: C++ / CUDA with Python and Matlab interfaces
  - **Theano**: Python based CUDA engine; several front ends available: e.g., **Keras** and Lasagne.
  - **Torch**: Library implemented in Lua language (Facebook). Also pyTorch interface exists since Jan 2017.
  - **TensorFlow**: Google deep learning engine. Open sourced in Nov 2015. Supported by **Keras**, which will be the default interface.
  - Others: VELES (Samsung), Minerva,...
  - Most use Nvidia **cuDNN** middle layer.
- The important ones:
  - **Caffe** is very fast and default in image recognition. Good Python and Matlab interface.
  - **Torch** has a large user base and lot of momentum from Facebook.
  - **Keras** is very flexible and readable full-python interface to Theano and Tensorflow. **This is our choice for this course.**
  - **http://keras.io/** and **https://github.com/fchollet/keras**

Current Trends

- Python has become the language of machine learning.



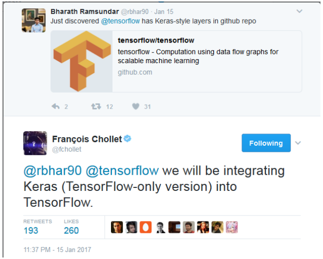
François Chollet @fchollet

Following

GitHub tickets for deep learning frameworks: in total, and for the previous 30 days. Baidu's Paddle is surprisingly strong.

RETWEETS 153 LIKES 295

6:39 PM - 15 Nov 2016



Bharath Ramsundar @rbhar90 · Jan 15

Just discovered @tensorflow has Keras-style layers in github repo

tensorflow/tensorflow

tensorflow - Computation using data flow graphs for scalable machine learning

github.com

2 12 31

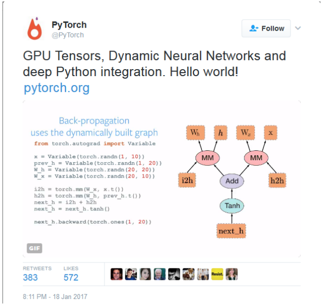
François Chollet @fchollet

Following

@rbhar90 @tensorflow we will be integrating Keras (TensorFlow-only version) into TensorFlow.

RETWEETS 193 LIKES 260

11:37 PM - 15 Jan 2017



PyTorch @PyTorch

Follow

GPU Tensors, Dynamic Neural Networks and deep Python integration. Hello world! pytorch.org

Back-propagation uses the dynamically built graph

```
from torch.autograd import Variable

x = Variable(torch.randn(1, 10))
prev_h = Variable(torch.randn(1, 20))
W_h = Variable(torch.randn(20, 20))
W_x = Variable(torch.randn(20, 10))

i2h = torch.mm(W_x, x.t())
h2h = torch.mm(W_h, prev_h.t())
next_h = i2h + h2h
next_h = next_h.tanh()

next_h.backward(torch.ones(1, 20))
```

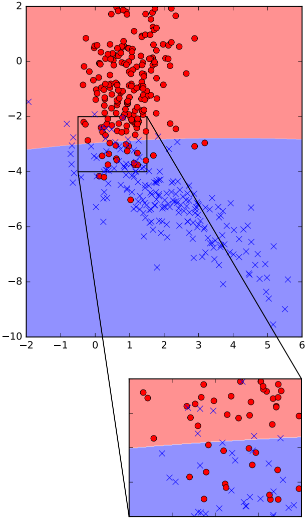
RETWEETS 383 LIKES 572

8:11 PM - 18 Jan 2017

Train a 2-layer Network with Keras

```
# Training code:
from keras.models import Sequential
from keras.layers.core import Dense, Activation

# First we initialize the model. "Sequential" means there are no loops.
clf = Sequential()

# Add layers one at the time. Each with 100 nodes.
clf.add(Dense(100, input_dim=2, activation = ’sigmoid’))
clf.add(Dense(100, activation = ’sigmoid’))
clf.add(Dense(1, activation = ’sigmoid’))

# The code is compiled to CUDA or C++
clf.compile(loss=’mean_squared_error’, optimizer=’sgd’)
clf.fit(X, y, nb_epoch=20, batch_size=16) # takes a few seconds
```

```
# Testing code:
# Probabilities
>>> clf.predict(np.array([[1, -2], [-3, -5]]))
array([[ 0.50781795],
       [ 0.48059484]])
# Classes
>>> clf.predict(np.array([[1, -2], [-3, -5]])) > 0.5
array([[ True],
       [False]], dtype=bool)
```



Deep Learning

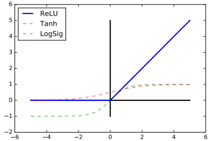
- The neural network research was rather silent after the rapid expansion in the 1990's.
- The hot topic of 2000's were, *e.g.*, the SVM and *big data*.
- However, at the end of the decade, neural networks started to gain popularity again: A group at Univ. Toronto led by Prof. Geoffrey Hinton studied unconventionally **deep** networks using *unsupervised* pretraining.
- He discovered that training of large networks was indeed possible with an unsupervised pretraining step that initializes the network weights in a layerwise manner.
- Another key factor to the success was the rapidly increased computational power brought by recent Graphics Processing Units (GPU's).

Unsupervised Pretraining

- There were two key problems why network depth did not increase beyond 2-3 layers:
  1. The error has huge **local minima areas** when the net becomes deep: Training gets stuck at one of them.
  2. The **gradient vanishes** at the bottom layers: The logistic activation function tends to decrease the gradient magnitude at each layer; eventually the gradient at the bottom layer is very small and they will not train at all.
- The former problem was corrected by **unsupervised** pretraining:
  - Train layered models that learned to *represent* the data (no class labels, no classification, just try to learn to reproduce the data).
  - Initialize the network with the weights of the unsupervised model and train in a supervised setting.
  - Common tools: *restricted Boltzmann machine* (RBM), *deep belief network* (DBN), *autoencoders*, etc.

Back to Supervised Training

- After the excitement of deep networks was triggered, the study of fully supervised approaches started as well (purely supervised training is more familiar, well explored and less scary angle of approach).
- A few key discoveries avoid the need for pretraining:
  - New activation functions that better preserve the gradient over layers; most importantly the Rectified Linear Unit[a]: $\mathrm{ReLU}(x) = \max(0, x)$.
  - Novel weight initialization techniques; *e.g.,* Glorot initialization (aka. Xavier initialization) adjusts the initial weight magnitudes layerwise[b].
  - Dropout regularization; avoid overfitting by injecting noise to the network[c]. Individual neurons are shut down at random in the training phase.



[a] Glorot, Bordes, and Bengio. "Deep sparse rectifier neural networks."

[b] Glorot and Bengio. "Understanding the difficulty of training deep feedforward neural networks."

[c] Srivastava, Hinton, Krizhevsky, Sutskever and Salakhutdinov. "Dropout: A simple way to prevent neural networks from overfitting."

Convolutional Layers

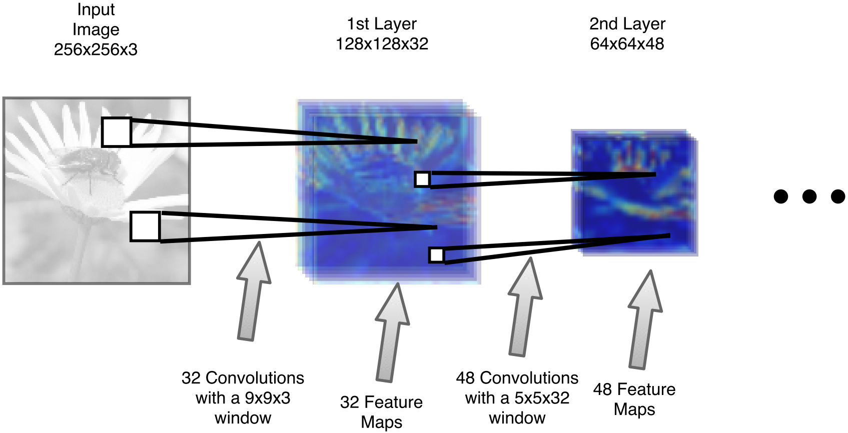
- In addition to the novel techniques for training, also new network architectures have been adopted.
- Most important of them is *convolutional layer*, which preserves also the topology of the input.
- Convolutional network was proposed already in 1989 but had a rather marginal role as long as image size was small (*e.g.*, 1990’s MNIST dataset of size 28 × 28 as compared to current ImageNet benchmark of size 256 × 256).

Convolutional Network

- The typical structure of a convolutional network repeats the following elements:
  convolution ⇒ **ReLU** ⇒ **subsampling**

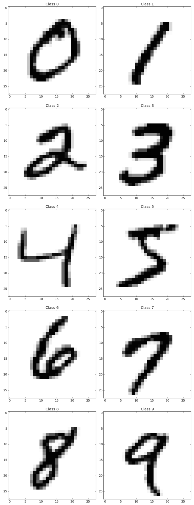
1. **Convolution** filters the input with a number of convolutional kernels. In the first layer these can be, *e.g.*, $9 \times 9 \times 3$; *i.e.*, they see the local window from all RGB layers.
    - The results are called **feature maps**, and there are typically a few dozen of those.
2. **ReLU** passes the feature maps through a pixelwise ReLU.
    - In numpy: `y = numpy.maximum(x, 0)`.
3. **Subsampling** shrinks the input dimensions by an integer factor.
    - Originally this was done by averaging each 2×2 block.
    - Nowadays, **maxpooling** is more common (take max of each 2×2 block).
    - Subsampling reduces the data size and improves spatial invariance.

Convolutional Network



Convolutional Network: Example

- Let’s train a convnet with the famous MNIST dataset.
- MNIST consists of 60000 training and 10000 test images representing handwritten numbers from US mail.
- Each image is 28 × 28 pixels and there are 10 categories.
- Generally considered an easy problem: Logistic regression gives over 90% accuracy and convnet can reach (almost) 100%.
- However, 10 years ago, the state of the art error was still over 1%.



Convolutional Network: Example

```python
# Training code (modified from mnist_cnn.py at Keras examples)
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation,
      Flatten
from keras.layers.convolutional import Convolution2D,
      MaxPooling2D

# We use the handwritten digit database "MNIST".
# 60000 training and 10000 test images of
# size 28x28
(X_train, y_train), (X_test, y_test) = mnist.load_data()

num_featmaps = 32   # This many filters per layer
num_classes = 10    # Digits 0,1,...,9
num_epochs = 50     # Show all samples 50 times
w, h = 5, 5         # Conv window size
```

```python
model = Sequential()

# Layer 1: needs input_shape as well.
model.add(Convolution2D(num_featmaps, w, h,
          input_shape=(1, 28, 28),
          activation = 'relu'))

# Layer 2:
model.add(Convolution2D(num_featmaps, w, h, activation = 'relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# Layer 3: dense layer with 128 nodes
# Flatten() vectorizes the data:
# 32x10x10 -> 3200
# (10x10 instead of 14x14 due to border effect)
model.add(Flatten())
model.add(Dense(128, activation = 'relu'))
model.add(Dropout(0.5))

# Layer 4: Last layer producing 10 outputs.
model.add(Dense(num_classes, activation='softmax'))

# Compile and train
model.compile(loss='categorical_crossentropy', optimizer='adadelta')
model.fit(X_train, Y_train, nb_epoch=100)
```

Convolutional Network: Training Log

- The code runs for about 5-10 minutes on a GPU.
- On a CPU, this would take 1-2 hours (1 epoch ≈ 500 s)

```
Using gpu device 0: Tesla K40m
Using Theano backend.
Compiling model...
Model compilation took 0.1 minutes.
Training...
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] — 31s — loss: 0.2193 — acc: 0.9322 — val_loss: 0.0519 — val_acc: 0.9835
Epoch 2/10
60000/60000 [==============================] — 31s — loss: 0.0807 — acc: 0.9758 — val_loss: 0.0398 — val_acc: 0.9863
Epoch 3/10
60000/60000 [==============================] — 31s — loss: 0.0581 — acc: 0.9825 — val_loss: 0.0322 — val_acc: 0.9898
Epoch 4/10
60000/60000 [==============================] — 31s — loss: 0.0500 — acc: 0.9851 — val_loss: 0.0276 — val_acc: 0.9913
Epoch 5/10
60000/60000 [==============================] — 31s — loss: 0.0430 — acc: 0.9872 — val_loss: 0.0287 — val_acc: 0.9906
Epoch 6/10
60000/60000 [==============================] — 31s — loss: 0.0387 — acc: 0.9882 — val_loss: 0.0246 — val_acc: 0.9922
Epoch 7/10
60000/60000 [==============================] — 31s — loss: 0.0352 — acc: 0.9897 — val_loss: 0.0270 — val_acc: 0.9913
Epoch 8/10
60000/60000 [==============================] — 31s — loss: 0.0324 — acc: 0.9902 — val_loss: 0.0223 — val_acc: 0.9928
Epoch 9/10
60000/60000 [==============================] — 31s — loss: 0.0294 — acc: 0.9907 — val_loss: 0.0221 — val_acc: 0.9926
Epoch 10/10
60000/60000 [==============================] — 31s — loss: 0.0252 — acc: 0.9922 — val_loss: 0.0271 — val_acc: 0.9916
Training (10 epochs) took 5.8 minutes.
```

Save and Load the Net

- The network can be saved and loaded to disk in a straightforward manner:
  - **Saving:**

```
model.save("my_net.h5")
```

  - **Loading:**

```
from keras.models import load_model
load_model("my_net.h5")
```

- Network is saved in HDF5 format. HDF5 is a serialization format similar to `.mat` or `.pkl` although a lot more efficient.
- Use HDF5 for data storage with h5py.

```
# Save np.array X to h5 file:
import h5py
with h5py.File("my_data.h5", "w") as h5:
   h5["X"] = X
```

```
# Load np.array X from h5 file:
import h5py
with h5py.File("my_data.h5", "r") as h5:
   X = np.array(h5["X"])

# Note: Don’t cast to numpy unless necessary.
# Data can be accessed from h5 directly.
```

Network Structure

- It is possible to look into the filters on the convolutional layers.

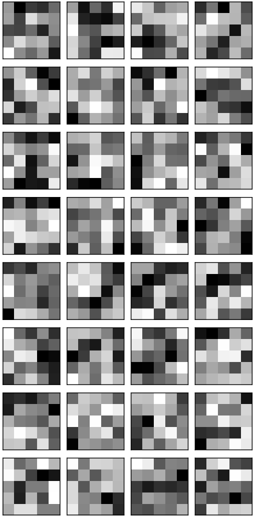
```
# First layer weights (shown on the right):
weights = model.layers[0].get_weights()[0]
```

- The second layer is difficult to visualize, because the input is 32-dimensional:

```
# Zeroth layer weights:
>>> model.layers[0].get_weights()[0].shape
(32, 1, 5, 5)
# First layer weights:
>>> model.layers[1].get_weights()[0].shape
(32, 32, 5, 5)
```

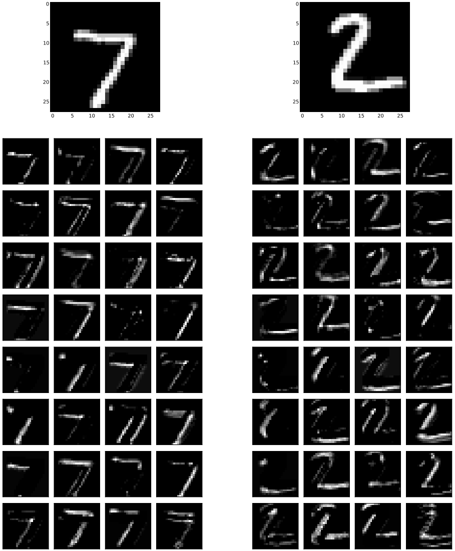
- The dense layer is the 5th (conv → conv → maxpool → dropout → flatten → dense).

```
# Fifth layer weights map 3200 inputs to 128 outputs.
# This is actually a matrix multiplication.
>>> model.layers[5].get_weights()[0].shape
(3200, 128)
```



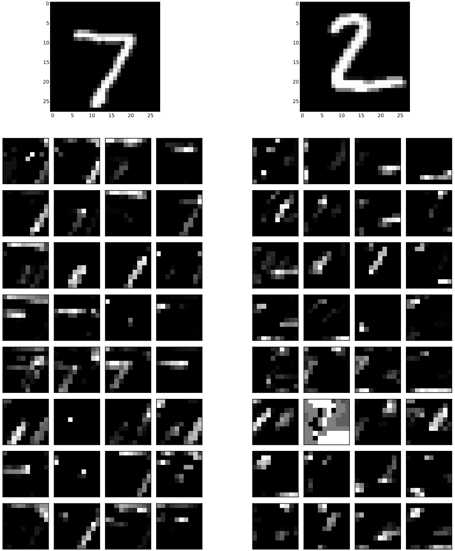
Network Activations

- The layer outputs are usually more interesting than the filters.
- These can be visualized as well.
- For details, see Keras FAQ.



Second Layer Activations

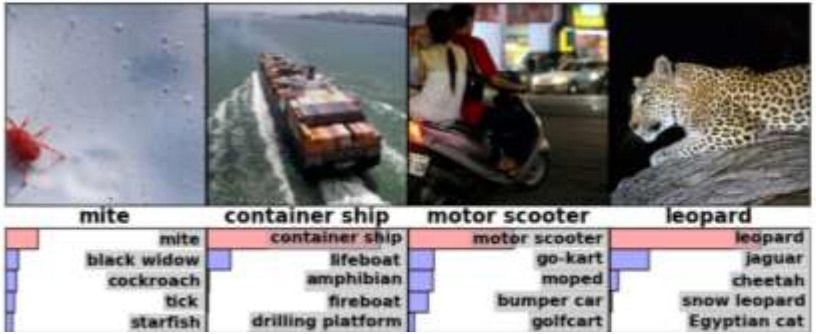
- On the next layer, the figures are downsampled to 12x12.
- This provides *spatial invariance*: The same activation results although the input would be slightly displaced.



Applications and Deep Learning State of the Art

Image Recognition

- Imagenet is the standard benchmark set for image recognition
- Classify 256x256 images into 1000 categories, such as ”person”, ”bike”, ”cheetah”, etc.
- Total 1.2M images
- Many error metrics, including top-5 error: error rate with 5 guesses



Picture from Alex Krizhevsky et al., ”ImageNet Classification with Deep Convolutional Neural Networks”, 2012

Computer Vision: Case Visy Oy

- Computer vision for logistics since 1994
- License plates (LPR), container codes,…
- How to grow in an environment with heavy competition?
  - Be agile
  - Be innovative
  - Be credible
  - Be customer oriented
  - Be technologically state-of-the-art

Kymmenistätuhansista autoista verot maksamatta – poliisin uusi laite käräytti 74 000 autoa

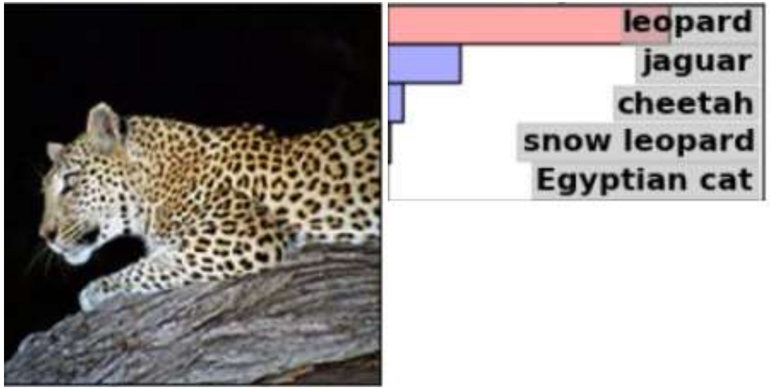
[illegible]

[illegible] HELSINGIN SANOMAT

What has changes in 20 years?

- **In 1996:**
  - Small images (*e.g.*, 10x10)
  - Few classes (< 100)
  - Small network (< 4 layers)
  - Small data (< 50K images)

- **In 2016:**
  - Large images (256x256)
  - Many classes (> 1K)
  - Deep net (> 100 kerrosta)
  - Large data (> 1M)

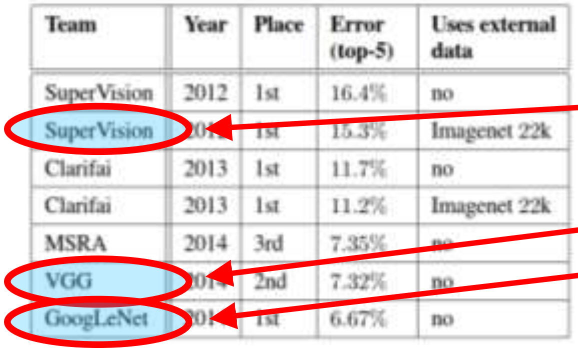


Net Depth Evolution Since 2012

ILSVRC Image Recognition Task:

- 1.2 million images
- 1 000 categories

(Prior to 2012: 25.7 %)



Team	Year	Place	Error (top-5)	Uses external data
SuperVision	2012	1st	16.4%	no
SuperVision	2012	1st	15.3%	Imagenet 22k
Clarifai	2013	1st	11.7%	no
Clarifai	2013	1st	11.2%	Imagenet 22k
MSRA	2014	3rd	7.35%	no
VGG	2014	2nd	7.32%	no
GoogLeNet	2014	1st	6.67%	no

8 layers

16 layers

22 layers

- 2015 winner: MSRA (error 3.57%)
- 2016 winner: Trimps-Soushen (2.99 %)

152 layers
(> 1200 layers tested)

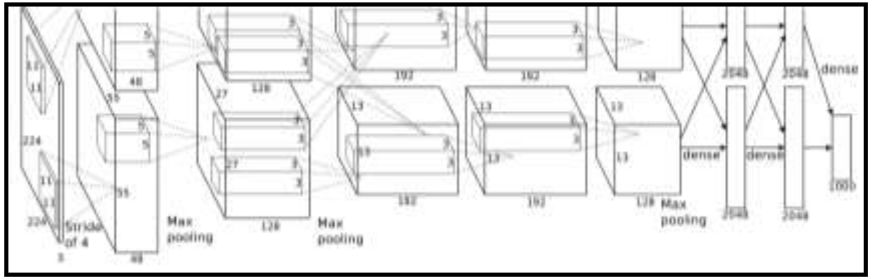
ILSVRC2012

- ILSVRC2012[1] was a game changer
- ConvNets dropped the top-5 error 26.2% → 15.3 %.
- The network is now called *AlexNet* named after the first author (see previous slide).
- Network contains 8 layers (5 convolutional followed by 3 dense); altogether 60M parameters.

[1] Imagenet Large Scale Visual Recognition Challenge

The AlexNet

- The architecture is illustrated in the figure.
- The pipeline is divided to two paths (upper & lower) to fit to 3GB of GPU memory available at the time (running on 2 GPU’s)
- Introduced many tricks for *data augmentation*
- Left-right flip
- Crop subimages (224x224)



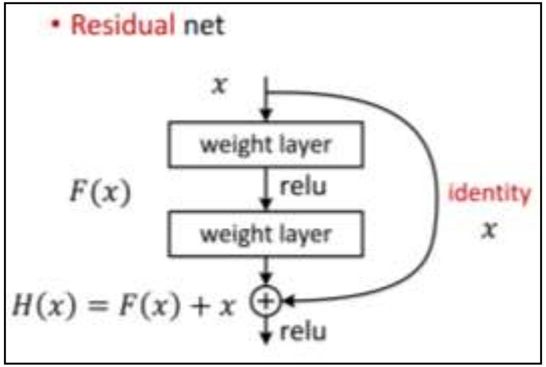
Picture from Alex Krizhevsky et al., ”ImageNet Classification with Deep Convolutional Neural Networks”, 2012

ILSVRC2014

- Since 2012, ConvNets have dominated
- 2014 there were 2 almost equal teams:
  - GoogLeNet Team with 6.66% Top-5 error
  - VGG Team with 7.33% Top-5 error
- In some subchallenges VGG was the winner
- GoogLeNet: 22 layers, only 7M parameters due to fully convolutional structure and clever *inception* architecture
- VGG: 16 layers, 144M parameters

ILSVRC2015

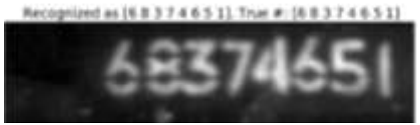
- Winner MSRA (Microsoft Research) with TOP-5 error 3.57 %
- 152 layers! 51M parameters.
- Built from residual blocks (which include the inception trick from previous year)
- Key idea is to add *identity shortcuts*, which make training easier

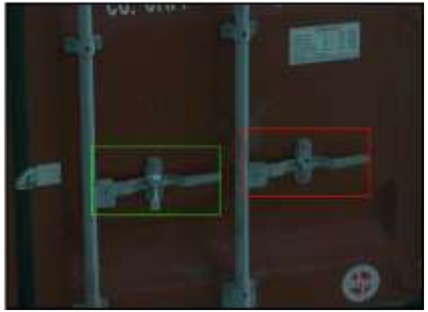


Pictures from MSRA ICCV2015 slides

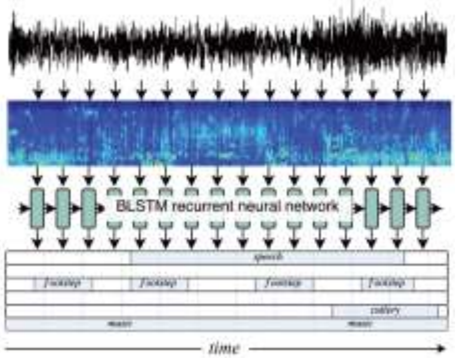
Research at TUT

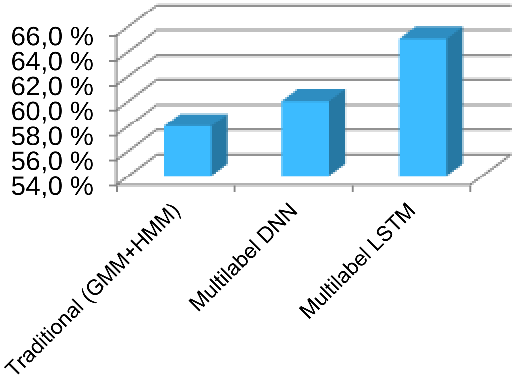
Images



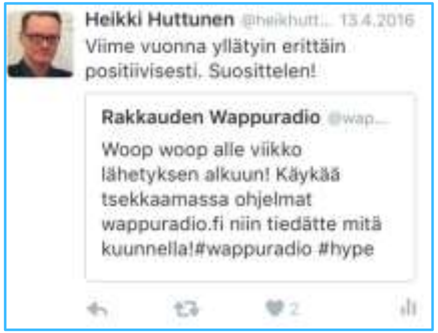


Sound





Text

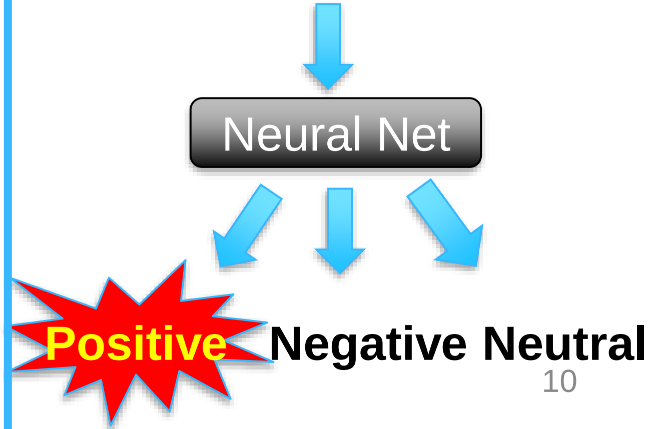


Heikki Huttunen @heikhutt... 13.4.2016

Viime vuonna yllätyin erittäin positiivisesti. Suosittelen!

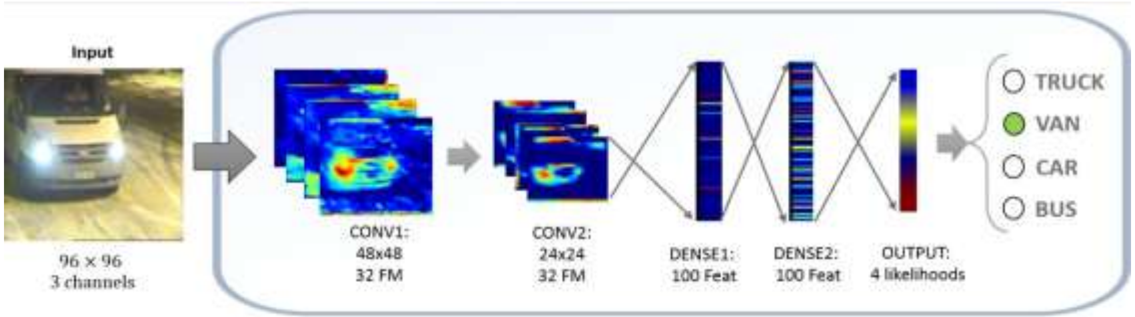
Rakkauden Wappuradio @wap...

Woop woop alle viikko lähetyksen alkuun! Käykää tsekkaamassa ohjelmat wappuradio.fi niin tiedätte mitä kuunnella!#wappuradio #hype

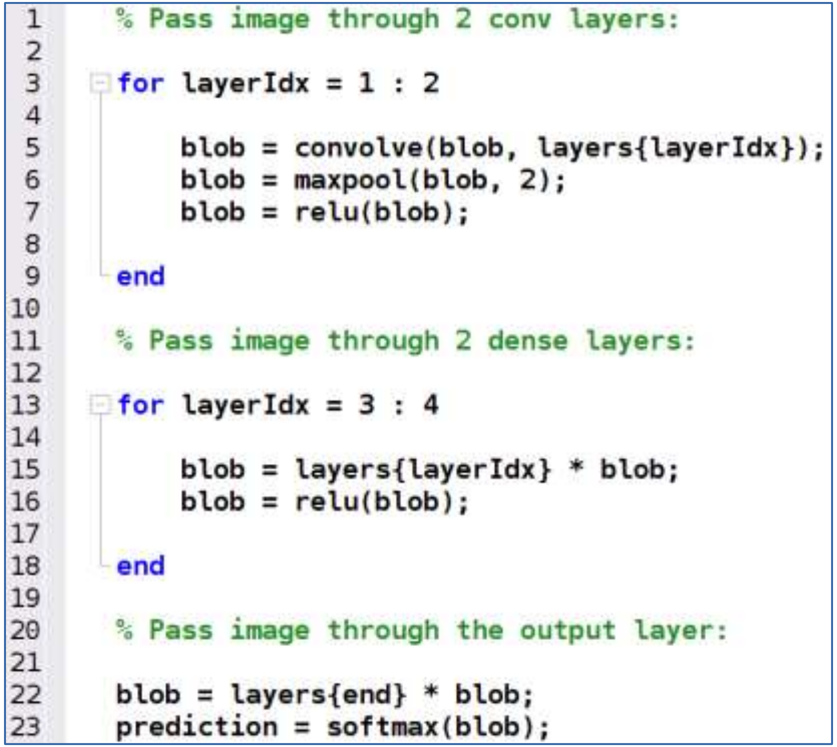


Example case

- TUT has studied shallow convolutional architectures for fast/real time detection tasks
- For example, Automatic Car Type Detection from Picture: **Van, Bus, Truck** or **Normal Vehicle**
- The network recognizes the car type (4 classes) with 98% accuracy (13 000 images).



Components of the Network

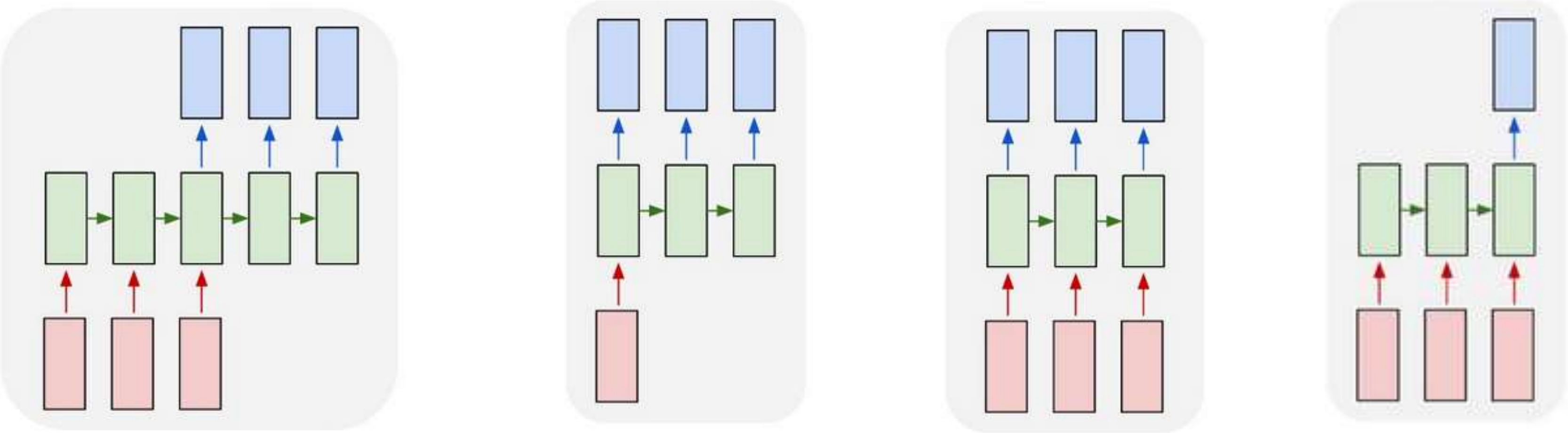


```
% Pass image through 2 conv layers:

for layerIdx = 1 : 2

    blob = convolve(blob, layers{layerIdx});
    blob = maxpool(blob, 2);
    blob = relu(blob);

end

% Pass image through 2 dense layers:

for layerIdx = 3 : 4

    blob = layers{layerIdx} * blob;
    blob = relu(blob);

end

% Pass image through the output layer:

blob = layers{end} * blob;
prediction = softmax(blob);
```

- **Convolution**: 5x5 window
- **Maxpooling**: 2x2 downsampling with the maximum
- **Relu**: max(x, 0)
- **Matrix multiplication**
- **Softmax**: $[\sigma(\mathbf{x})]_k = \frac{\exp(x_k)}{\sum \exp(x_k)}$

Recurrent Networks

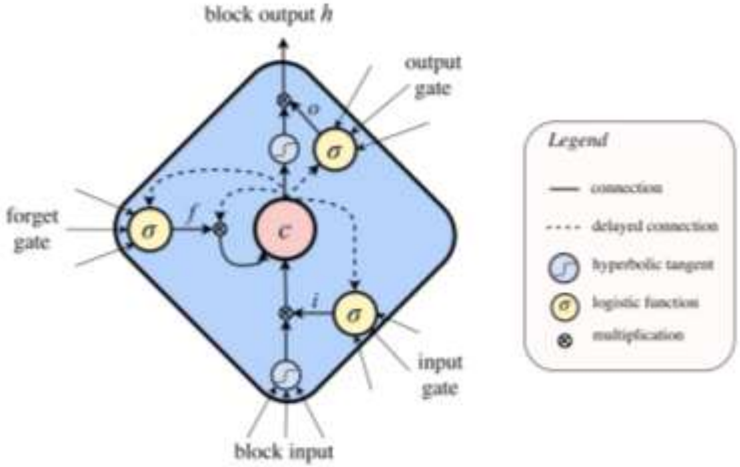
- Recurrent networks process sequences of arbitrary length; *e.g.*,
    - Sequence → sequence
    - Image → sequence
    - Sequence → class ID



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

Recurrent Networks

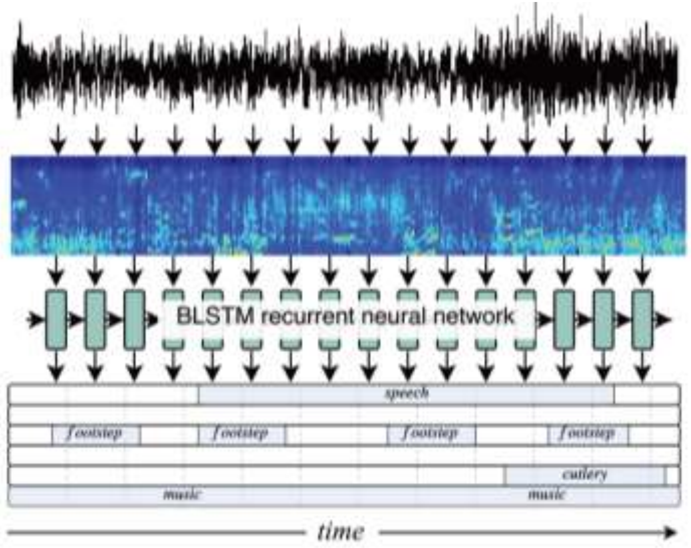
- Recurrent net consist of special nodes that remember past states.
- Each node receives 2 inputs: the data and the previous state.
- Keras implements *SimpleRNN, LSTM* and *GRU* layers.
- Most popular recurrent node type is *Long Short Term Memory* (LSTM) node.
- LSTM includes also *gates*, which can turn on/off the history and a few additional inputs.



Picture from G. Parascandolo M.Sc. Thesis, 2015.
http://urn.fi/URN:NBN:fi:tty-201511241773

Recurrent Networks

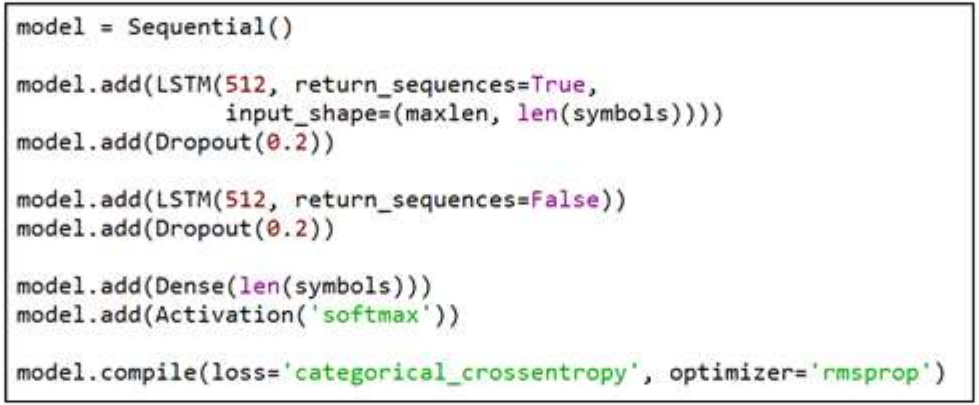
- An example of use is from our recent paper.
- We detect acoustic events within 61 categories.
- LSTM is particularly effective because it remembers the past events (or the context).
- In this case we used a *bidirectional* LSTM, which remembers also the future.
- BLSTM gives slight improvement over LSTM.



Picture from Parascandolo et al., ICASSP 2016

LSTM in Keras

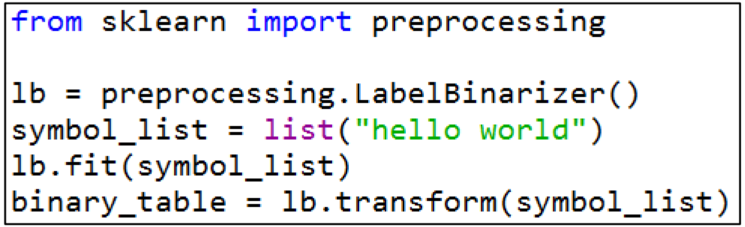
- LSTM layers can be added to the model like any other layer type.
- This is an example for natural language modeling: *Can the network predict next symbol from the previous ones?*
- Accuracy is greatly improved from N-Gram etc.



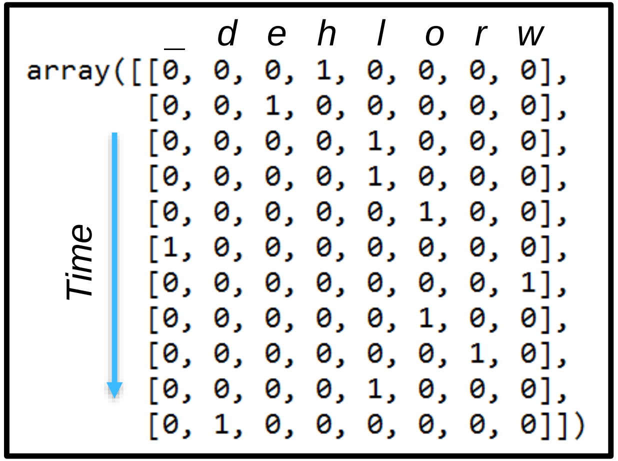
```
model = Sequential()

model.add(LSTM(512, return_sequences=True,
               input_shape=(maxlen, len(symbols))))
model.add(Dropout(0.2))

model.add(LSTM(512, return_sequences=False))
model.add(Dropout(0.2))

model.add(Dense(len(symbols)))
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy', optimizer='rmsprop')
```

Text Modeling

- The input to LSTM should be a sequence of vectors.
- For text modeling, we represent the symbols as binary vectors.



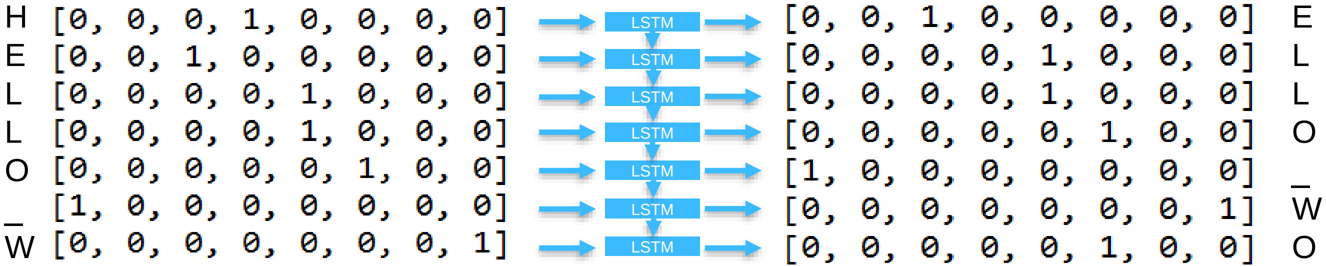
```python
from sklearn import preprocessing

lb = preprocessing.LabelBinarizer()
symbol_list = list("hello world")
lb.fit(symbol_list)
binary_table = lb.transform(symbol_list)
```



```
        _  d  e  h  l  o  r  w
array([[0, 0, 0, 1, 0, 0, 0, 0],
       [0, 0, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 1, 0, 0, 0],
       [0, 0, 0, 0, 1, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 0, 0],
       [1, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 1, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 0],
       [0, 0, 0, 0, 1, 0, 0, 0],
       [0, 1, 0, 0, 0, 0, 0, 0]])
```

Text Modeling

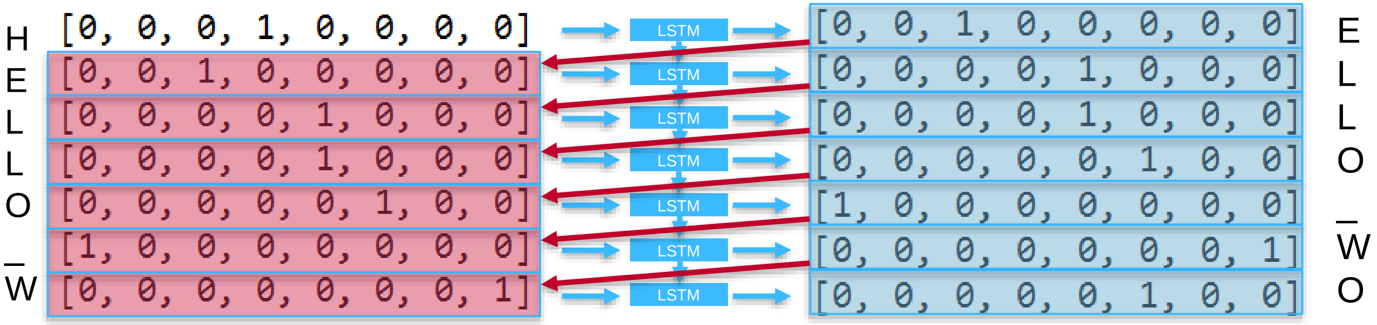
- The prediction target for the LSTM net is simply the input delayed by one step.
- For example: we have shown the net these symbols: [’h’, ’e’, ’l’, ’l’, ’o’, ’_’, ’w’]
- Then the network should predict ’o’.



```
H [0, 0, 0, 1, 0, 0, 0, 0]  →  LSTM  →  [0, 0, 1, 0, 0, 0, 0, 0]  E
E [0, 0, 1, 0, 0, 0, 0, 0]  →  LSTM  →  [0, 0, 0, 0, 1, 0, 0, 0]  L
L [0, 0, 0, 0, 1, 0, 0, 0]  →  LSTM  →  [0, 0, 0, 0, 1, 0, 0, 0]  L
L [0, 0, 0, 0, 1, 0, 0, 0]  →  LSTM  →  [0, 0, 0, 0, 0, 1, 0, 0]  O
O [0, 0, 0, 0, 0, 1, 0, 0]  →  LSTM  →  [1, 0, 0, 0, 0, 0, 0, 0]  _
_ [1, 0, 0, 0, 0, 0, 0, 0]  →  LSTM  →  [0, 0, 0, 0, 0, 0, 0, 1]  W
W [0, 0, 0, 0, 0, 0, 0, 1]  →  LSTM  →  [0, 0, 0, 0, 0, 1, 0, 0]  O
```

Text Modeling

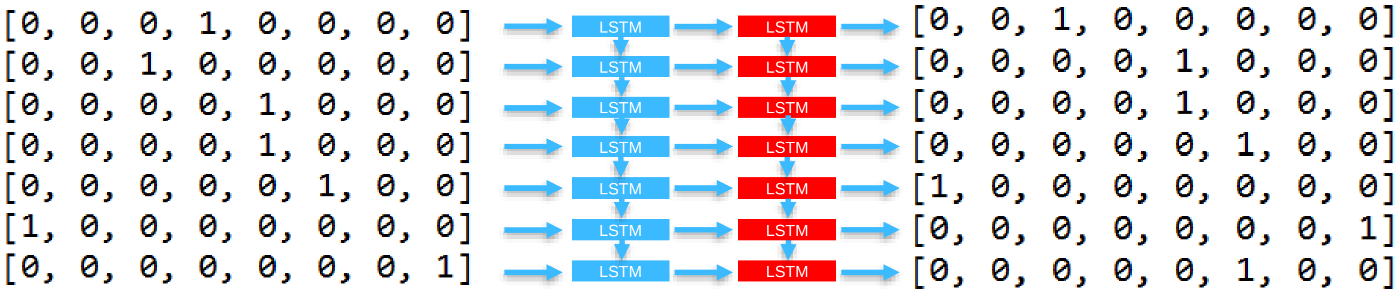
- Trained LSTM can be used as a text generator.
- Show the first character, and set the predicted symbol as the next input.
- Randomize among the top scoring symbols to avoid static loops.



H	[0, 0, 0, 1, 0, 0, 0, 0]	LSTM	[0, 0, 1, 0, 0, 0, 0, 0]	E
E	[0, 0, 1, 0, 0, 0, 0, 0]	LSTM	[0, 0, 0, 0, 1, 0, 0, 0]	L
L	[0, 0, 0, 0, 1, 0, 0, 0]	LSTM	[0, 0, 0, 0, 1, 0, 0, 0]	L
L	[0, 0, 0, 0, 1, 0, 0, 0]	LSTM	[0, 0, 0, 0, 0, 1, 0, 0]	O
O	[0, 0, 0, 0, 0, 1, 0, 0]	LSTM	[1, 0, 0, 0, 0, 0, 0, 0]	_
_	[1, 0, 0, 0, 0, 0, 0, 0]	LSTM	[0, 0, 0, 0, 0, 0, 0, 1]	W
W	[0, 0, 0, 0, 0, 0, 0, 1]	LSTM	[0, 0, 0, 0, 0, 1, 0, 0]	O

Many LSTM Layers

- A straightforward extension of LSTM is to use it in multiple layers (typically less than 5).
- Below is an example of two layered LSTM.
- Note: Each blue block is exactly the same with, *e.g.*, 512 LSTM nodes. So is each red block.



[0, 0, 0, 1, 0, 0, 0, 0] → LSTM → LSTM → [0, 0, 1, 0, 0, 0, 0, 0]

[0, 0, 1, 0, 0, 0, 0, 0] → LSTM → LSTM → [0, 0, 0, 0, 1, 0, 0, 0]

[0, 0, 0, 0, 1, 0, 0, 0] → LSTM → LSTM → [0, 0, 0, 0, 1, 0, 0, 0]

[0, 0, 0, 0, 1, 0, 0, 0] → LSTM → LSTM → [0, 0, 0, 0, 0, 1, 0, 0]

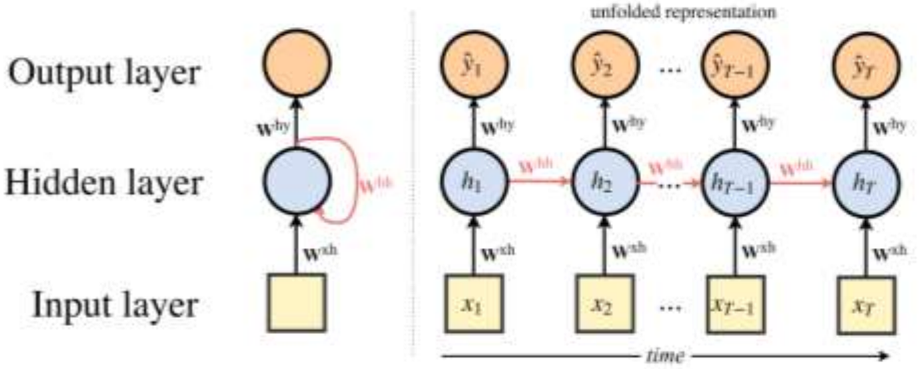
[0, 0, 0, 0, 0, 1, 0, 0] → LSTM → LSTM → [1, 0, 0, 0, 0, 0, 0, 0]

[1, 0, 0, 0, 0, 0, 0, 0] → LSTM → LSTM → [0, 0, 0, 0, 0, 0, 0, 1]

[0, 0, 0, 0, 0, 0, 0, 1] → LSTM → LSTM → [0, 0, 0, 0, 0, 1, 0, 0]

LSTM Training

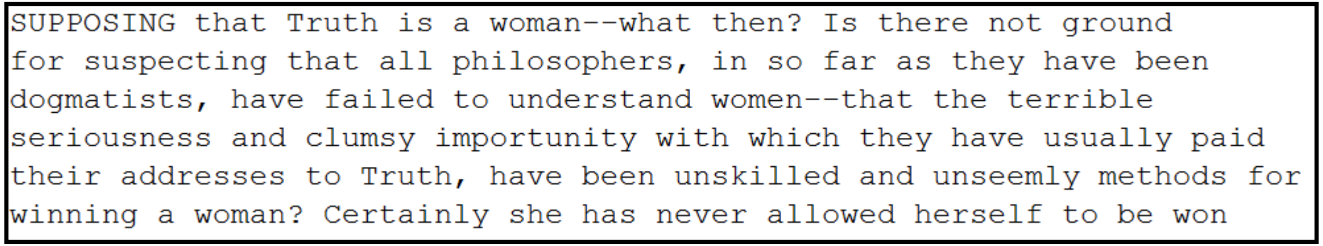
- LSTM net can be viewed as a very deep non-recurrent network.
- The LSTM net can be *unfolded* in time over a sequence of time steps.
- After unfolding, the normal gradient based learning rules apply.



Picture from G. Parascandolo M.Sc. Thesis, 2015.
http://urn.fi/URN:NBN:fi:tty-201511241773

Text Modeling Experiment

- Keras includes an example script: https://github.com/fchollet/keras/blob/master/examples/lstm_text_generation.py
- Train a 2-layer LSTM (512 nodes each) by showing Nietzche texts.
- A sequence of 600901 characters consisting of 59 symbols (uppercase, lowercase, special characters).

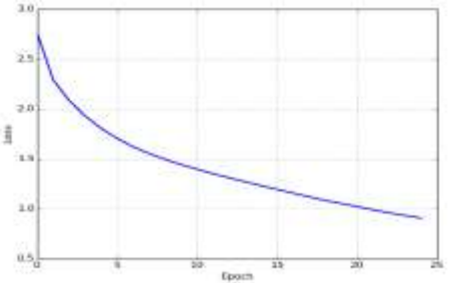


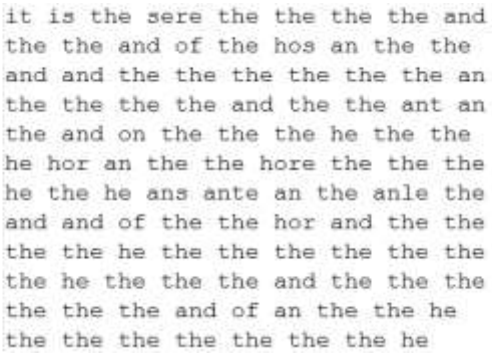
```
SUPPOSING that Truth is a woman--what then? Is there not ground
for suspecting that all philosophers, in so far as they have been
dogmatists, have failed to understand women--that the terrible
seriousness and clumsy importunity with which they have usually paid
their addresses to Truth, have been unskilled and unseemly methods for
winning a woman? Certainly she has never allowed herself to be won
```

Sample of training data

Text Modeling Experiment

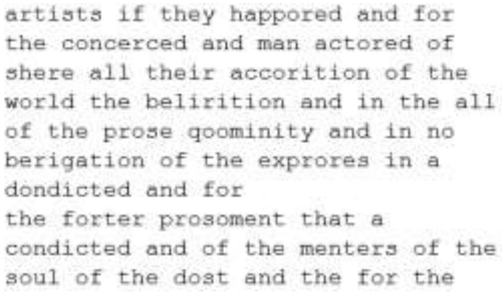
- The training runs for a few hours on a Nvidia high end GPU (Tesla K40m).
- At start, the net knows only a few words, but picks up the vocabulary rather soon.





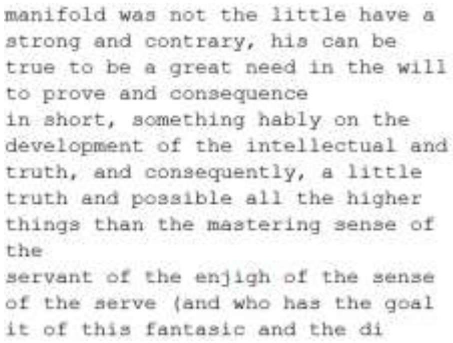
```
it is the sere the the the the and
the the and of the hos an the the
and and the the the the the the an
the the the the and the the ant an
the and on the the the he the the
he hor an the the hore the the the
he the he ans ante an the anle the
and and of the the hor and the the
the the he the the the the the the
the he the the the and the the the
the the the and of an the the he
the the the the the the the he
```

Epoch 1



```
artists if they happored and for
the concerced and man actored of
shere all their accorition of the
world the belirition and in the all
of the prose qoominity and in no
berigation of the exprores in a
dondicted and for
the forter prosoment that a
condicted and of the menters of the
soul of the dost and the for the
```

Epoch 3



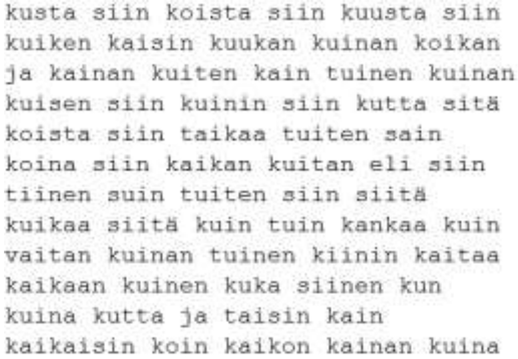
```
manifold was not the little have a
strong and contrary, his can be
true to be a great need in the will
to prove and consequence
in short, something habbly on the
development of the intellectual and
truth, and consequently, a little
truth and possible all the higher
things than the mastering sense of
the
servant of the enjigh of the sense
of the serve (and who has the goal
it of this fantasic and the di
```

Epoch 25

TAMPERE UNIVERSITY OF TECHNOLOGY

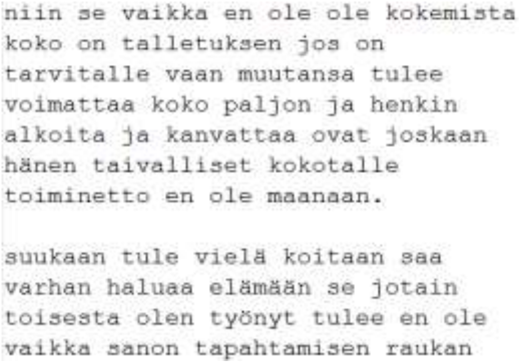
Text Modeling Experiment

- Let’s do the same thing for Finnish text: All discussions from Suomi24 forum are released for public.
- The message is nonsense, but syntax close to correct: A foreigner can not tell the difference.



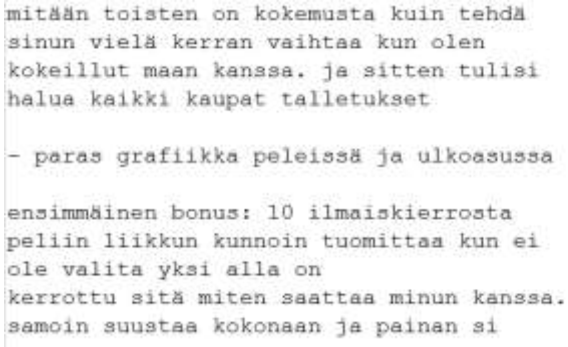
```
kusta siin koista siin kuusta siin
kuiken kaisin kuukan kuinan koikan
ja kainan kuiten kain tuinen kuinan
kuisen siin kuinin siin kutta sitä
koista siin taikaa tuiten sain
koina siin kaikan kuitan eli siin
tiinen suin tuiten siin siitä
kuikaa siitä kuin tuin kankaa kuin
vaitan kuinan tuinen kiinin kaitaa
kaikaan kuinen kuka siinen kun
kuina kutta ja taisin kain
kaikaisin koin kaikon kainan kuina
```

Epoch 1



```
niin se vaikka en ole ole kokemista
koko on talletuksen jos on
tarvitalle vaan muutansa tulee
voimattaa koko paljon ja henkin
alkoita ja kanvattaa ovat joskaan
hänen taivalliset kokotalle
toiminetto en ole maanaan.

suukaan tule vielä koitaan saa
varhan haluaa elämään se jotain
toisesta olen työnyt tulee en ole
vaikka sanon tapahtamisen raukan
```

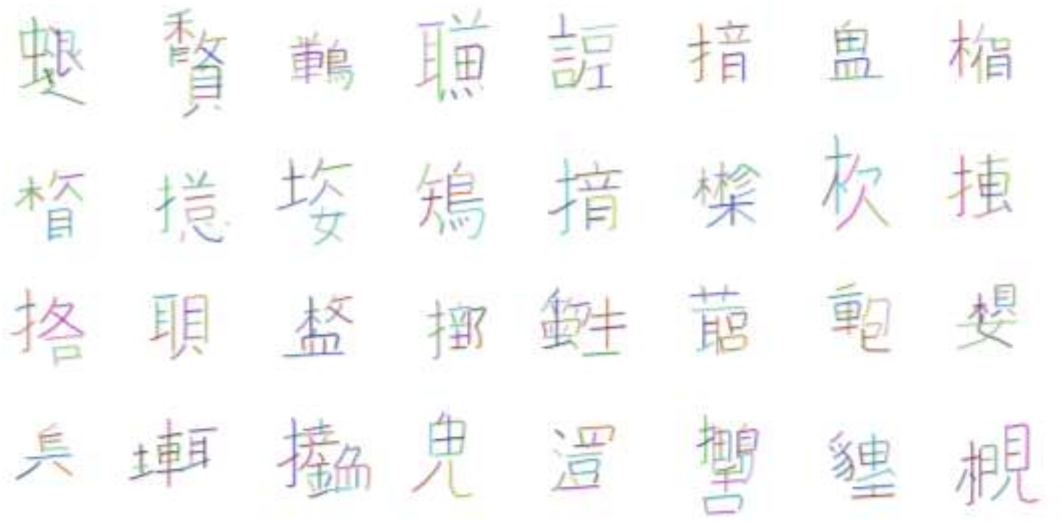
Epoch 4



```
mitään toisten on kokemusta kuin tehdä
sinun vielä kerran vaihtaa kun olen
kokeillut maan kanssa. ja sitten tulisi
halua kaikki kaupat talletukset

- paras grafiikka peleissä ja ulkoasussa

ensimmäinen bonus: 10 ilmaiskierrosta
peliin liikkun kunnoin tuomittaa kun ei
ole valita yksi alla on
kerrottu sitä miten saattaa minun kanssa.
samoin suustaa kokonaan ja painan si
```

Epoch 44

Fake Chinese Characters



http://tinyurl.com/no36azh

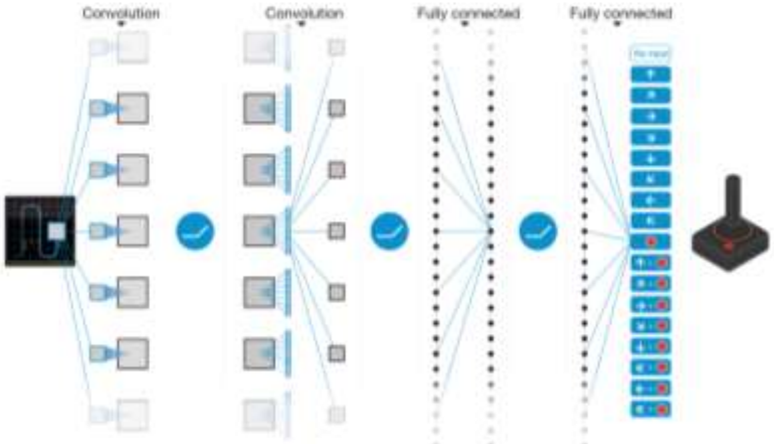
EXAMPLES

Age / Gender / Expression Recognition

- TUT age estimation demo is an example of modern computer vision
- System estimates the age in real time
- Trained using a 500 K image database
- Average error ±3 years

Deep Net Learns to Play

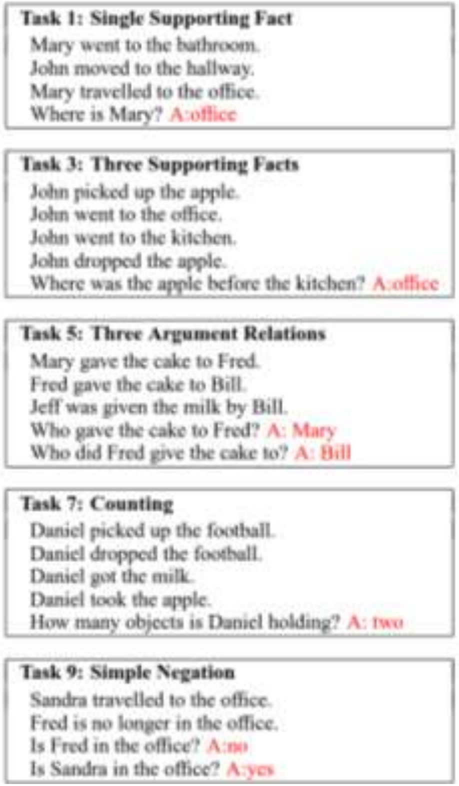
- Mnih *et al.* (Google Deepmind, 2015) trained a network to play computer games



- Better than human in many classic 1980’s games: *Pinball, Pong, Space Invaders.*

Computer and Logical Reasoning

- Logical reasoning is considered as a humans-only skill
- In this example, the computer was shown 1,000 question and answers
- In all 10 categories, the computer answers with > 95 % accuracy (except Task 7: 85 %)



Task 1: Single Supporting Fact
Mary went to the bathroom.
John moved to the hallway.
Mary travelled to the office.
Where is Mary? A:office

Task 3: Three Supporting Facts
John picked up the apple.
John went to the office.
John went to the kitchen.
John dropped the apple.
Where was the apple before the kitchen? A:office

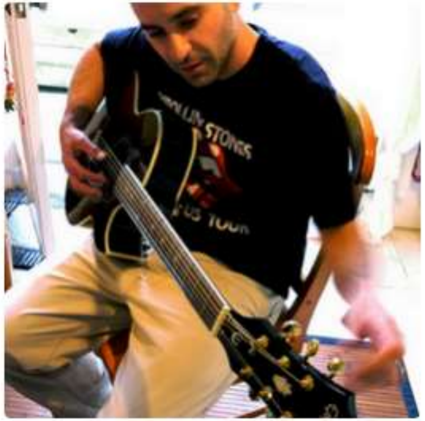
Task 5: Three Argument Relations
Mary gave the cake to Fred.
Fred gave the cake to Bill.
Jeff was given the milk by Bill.
Who gave the cake to Fred? A: Mary
Who did Fred give the cake to? A: Bill

Task 7: Counting
Daniel picked up the football.
Daniel dropped the football.
Daniel got the milk.
Daniel took the apple.
How many objects is Daniel holding? A: two

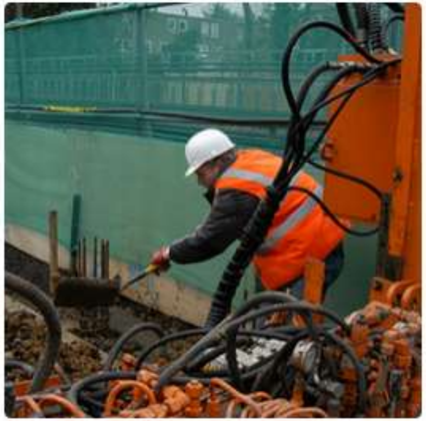
Task 9: Simple Negation
Sandra travelled to the office.
Fred is no longer in the office.
Is Fred in the office? A:no
Is Sandra in the office? A:yes

Weston et al., ”Towards AI-complete question answering”, ICLR2016.

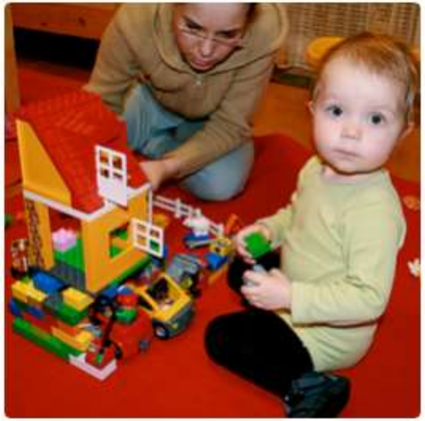
From Image to Text



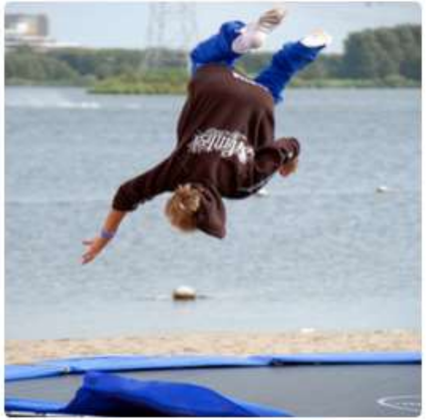
"man in black shirt is playing guitar."



"construction worker in orange safety vest is working on road."



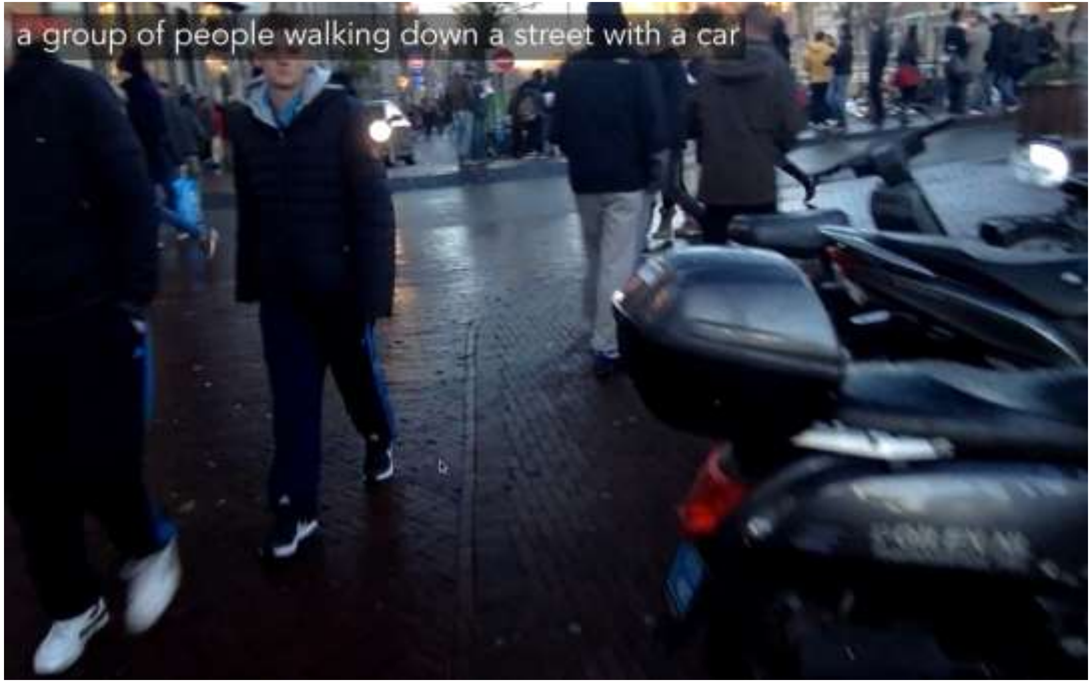
"two young girls are playing with lego toy."



"boy is doing backflip on wakeboard."

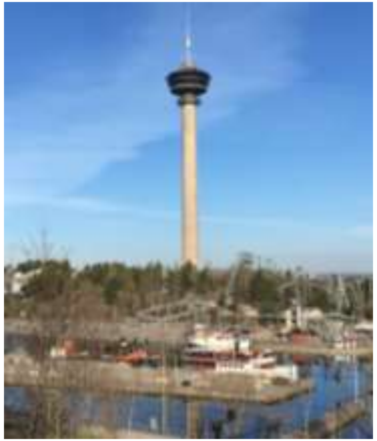
Karpathy *et al.*, ”Deep Visual-semantic Alignments for Generating Image Descriptions,” CVPR 2015, June 2015.

From Video to Text

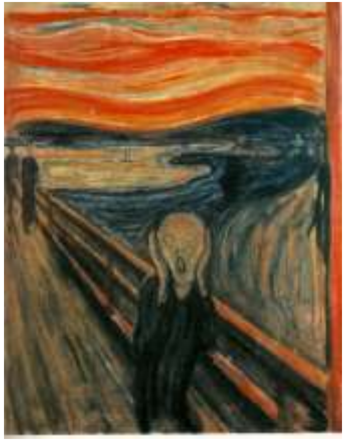


https://www.youtube.com/watch?v=8BFzu9m52sc

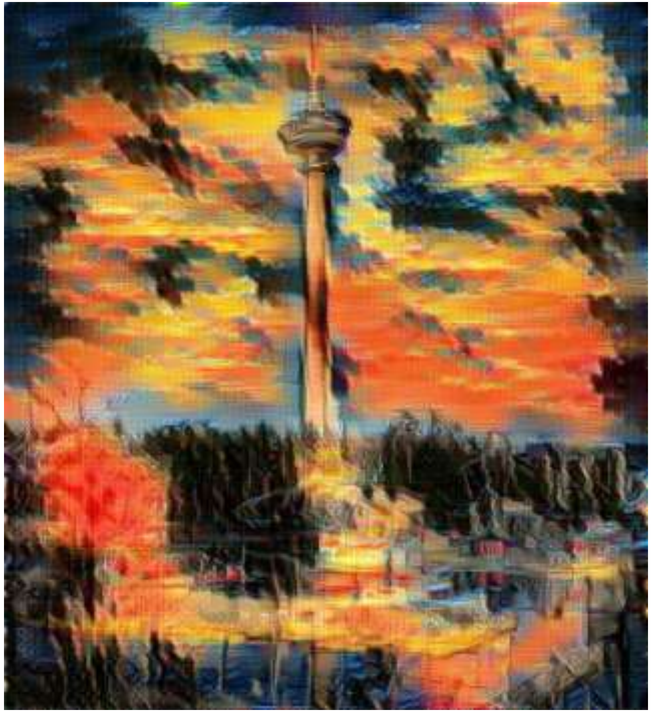
Artistic Style Transfer



+



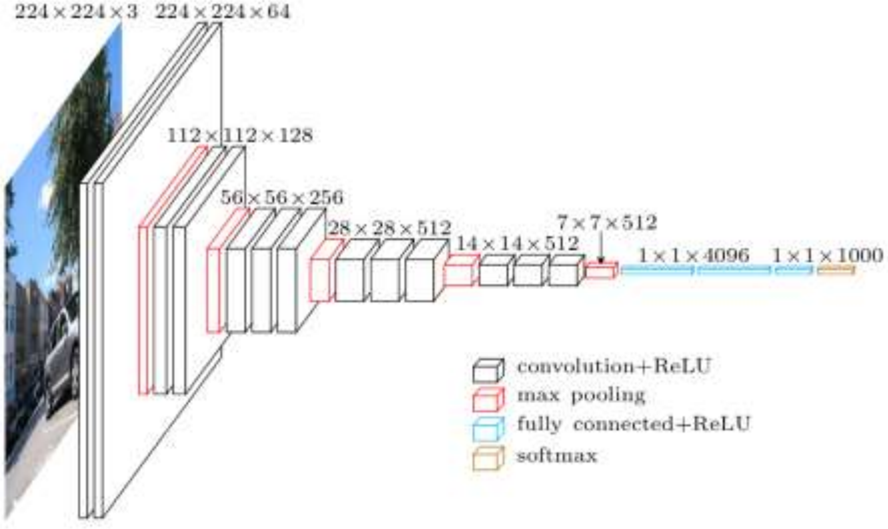
=



Check out Prisma App

Pretraining

- With small data, people often initialize the net with a *pretrained* network.
- This may be one of the imagenet winners; VGG16, ResNet, …
- See `keras.applications` for some of these.



VGG16 network
source: https://www.cs.toronto.edu/~frossard/post/vgg16/

To Conclude…

- During the last ten years, the landscape of artificial intelligence has reached a new level of maturity:
  - **Infrastructure** has been built to allow low cost access to high-performance computing.
  - **Publicity** of the results has become a standard model in dissemination of the research results.
  - **Resources** have increased: Companies are extremely active in AI research, and aggressively headhunting for the best talents in the field.
  - **Methods** have been improved and computers are increasingly able to solve human-like tasks.