VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

Deep Learning Training on Nvdidia GPUs with Tensorflow

Georg Zitzlsberger georg.zitzlsberger@vsb.cz

12-11-2019

EUROPEAN UNION
European Structural and Investment Funds
Operational Programme Research,
Development and Education

MINISTRY OF EDUCATION, YOUTH AND SPORTS

Agenda

History

Difference Machine vs. Deep Learning

Neuronal Networks

Programming

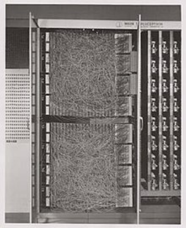
Parallelism

Questions

History

In a nutshell:

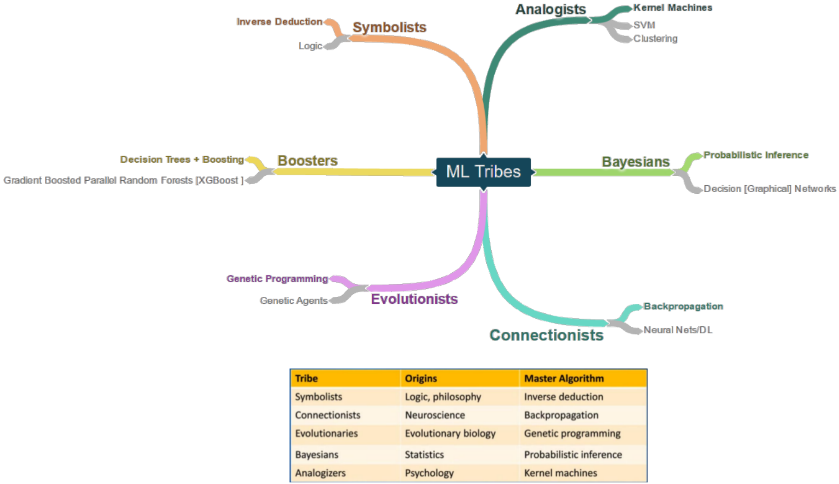
- 1950: Perceptron



(Image: Cornell Aeronautical Laboratory)

  - By Frank Rosenblatt (funded by US Office of Naval Research)
  - 20x20 input photocells
  - Electro-mechanic
  - Not capable enough for multi-class patterns (only one layer)
- First AI winter (1974-1980)
- 1989:
  Yann le Cun's *Theoretical Framework for Back-Propagation*
- Second AI winter (1987-1993)
- 2012:
  Dawn of Deep Neuronal Networks with *AlexNet*
- What's next? AI winter or Singularity?

ML Tribes



ML Tribes

Symbolists
Inverse Deduction
Logic

Analogists
Kernel Machines
SVM
Clustering

Boosters
Decision Trees + Boosting
Gradient Boosted Parallel Random Forests [XGBoost]

Bayesians
Probabilistic Inference
Decision [Graphical] Networks

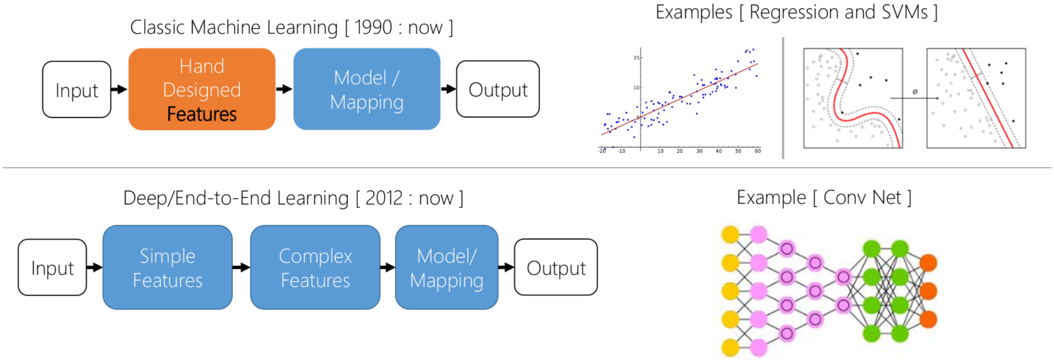
Evolutionists
Genetic Programming
Genetic Agents

Connectionists
Backpropagation
Neural Nets/DL

Tribe	Origins	Master Algorithm
Symbolists	Logic, philosophy	Inverse deduction
Connectionists	Neuroscience	Backpropagation
Evolutionaries	Evolutionary biology	Genetic programming
Bayesians	Statistics	Probabilistic inference
Analogizers	Psychology	Kernel machines

(Image: Nvidia)

Difference Machine vs. Deep Learning



Classic Machine Learning [1990 : now]

Input → Hand Designed **Features** → Model / Mapping → Output

Examples [Regression and SVMs]

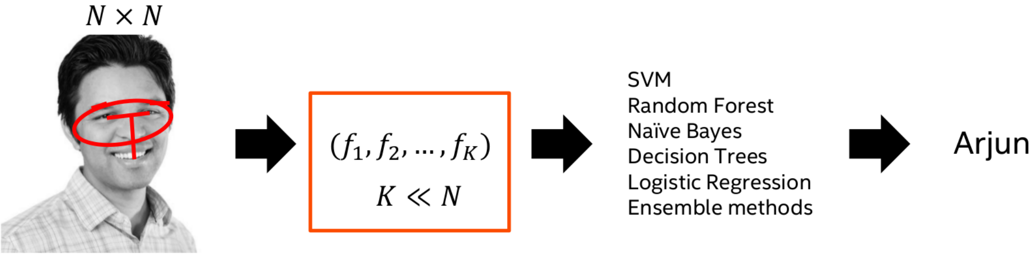
Deep/End-to-End Learning [2012 : now]

Input → Simple Features → Complex Features → Model/ Mapping → Output

Example [Conv Net]

(Image: Nvidia)

Machine Learning: Feature Engineering



$N \times N$

$(f_1, f_2, \dots, f_K)$

$K \ll N$

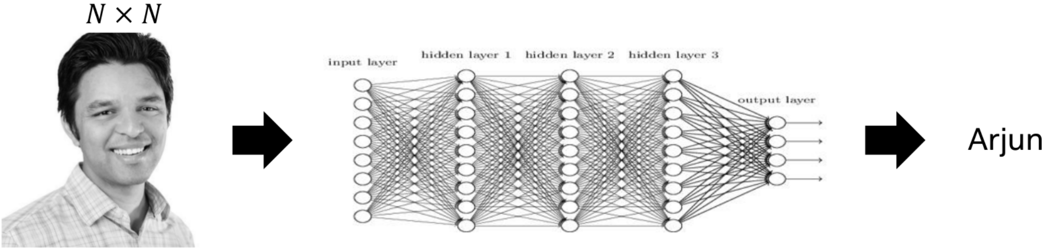
SVM
Random Forest
Naïve Bayes
Decision Trees
Logistic Regression
Ensemble methods

Arjun

!! Takes lot of time

(Image: Intel)

Deep Learning: Data Engineering



Features are discovered from data

Extract features at multiple levels of abstraction

Performance improves with more data

High degree of representational power

But old practices apply:
Data cleaning, Exploration, Data annotation, hyper-parameters, etc.

(Image: Intel)

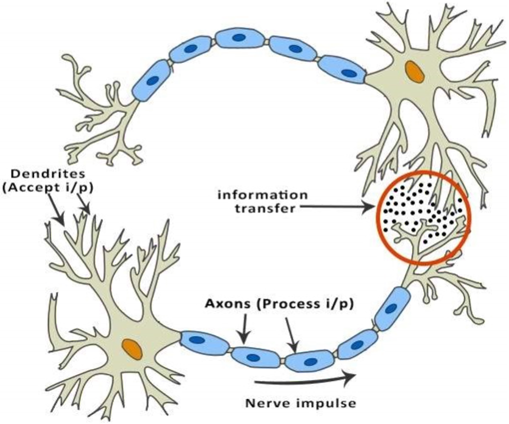
Why Now?

(Image: Nvidia)

- Big Data:
  Large amounts of data are available
- Recent Deep Network Development:
  New Deep Learning methodologies evolved (2010 onwards)
- Hardware:
  Modern systems are fast enough and have the memory needed

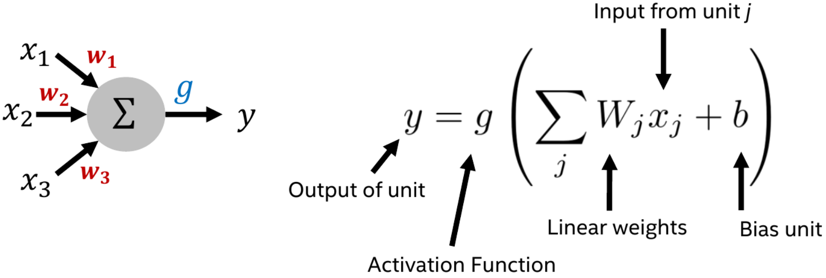
Neuronal Networks

Inspired by biology:



(Image: Intel)

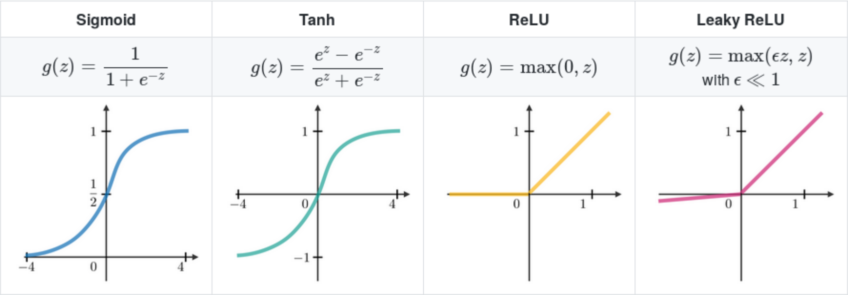
Artificial Neuronal Networks



$$y = g\left(\sum_j W_j x_j + b\right)$$

(Image: Intel)

Activation Function



Sigmoid	Tanh	ReLU	Leaky ReLU
$g(z) = \dfrac{1}{1+e^{-z}}$	$g(z) = \dfrac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$

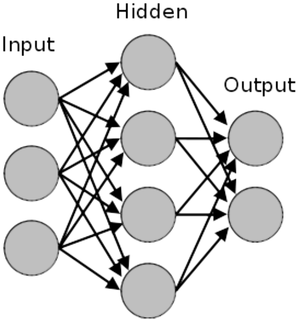
(Image: Afshine Amidi[1])

- Adds non-linearity
- ReLU is currently the most popular (est. 2010)
  - Easy to compute its derivation
  - Mitigates vanishing/exploding gradient problem (even better here: Leaky ReLU)

[1] https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-deep-learning

Deep Neural Network

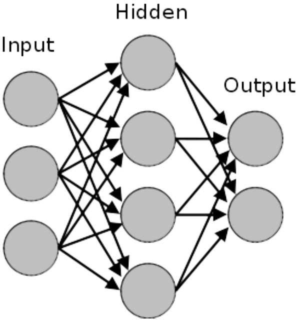
Example of a "Deep" Neural Network:



(Image: Intel)

- Layers can have different number of neurons
- Input and output formats can be arbitrary
- There can be multiple (hundreds) of hidden layers
- Typically output is combined with *softmax* function (probabilistic output)
- Example shows fully connected network, which is a special case

Deep Neural Network Operation

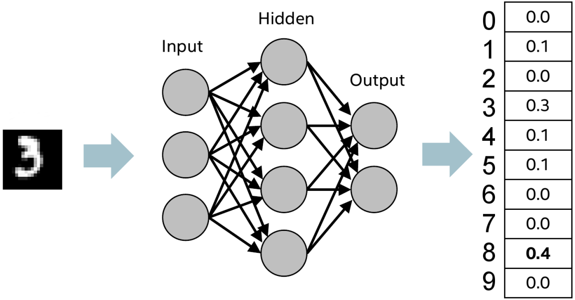


(Image: Intel)

1. Random weights
2. Get a random batch of training data
3. Forward propagation
4. Calculate cost (loss)
5. Backward propagation
6. Update weights and bias
7. Goto step 2.

Deep Neural Network: Forward Propagation

Example for one digit (image):



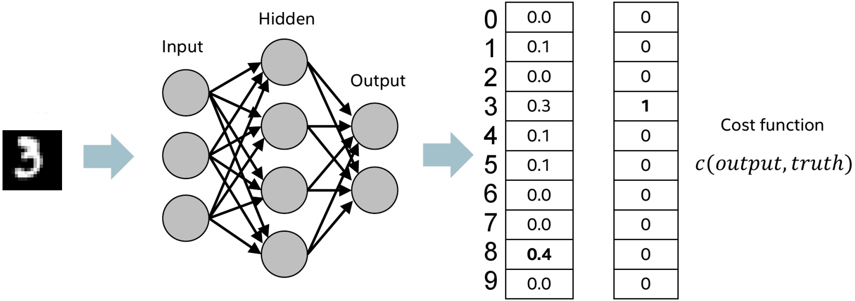
0	0.0
1	0.1
2	0.0
3	0.3
4	0.1
5	0.1
6	0.0
7	0.0
8	**0.4**
9	0.0

$$y = g\left(\sum_j W_j x_j + b\right)$$

(Image: Intel)

Deep Neural Network: Cost Function

Example of a cost (or loss) function:



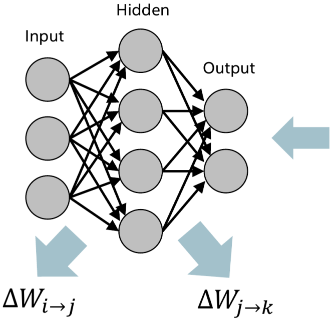
0	0.0	0
1	0.1	0
2	0.0	0
3	0.3	**1**
4	0.1	0
5	0.1	0
6	0.0	0
7	0.0	0
8	**0.4**	0
9	0.0	0

(Image: Intel)

- How far off are we from the ground truth?
- Example has labeled data (different if non-labeled data)

Deep Neural Network: Backward Propagation

How weights are updated:



0	0.0	0
1	0.1	0
2	0.0	0
3	0.3	**1**
4	0.1	0
5	0.1	0
6	0.0	0
7	0.0	0
8	**0.4**	0
9	0.0	0

Cost function

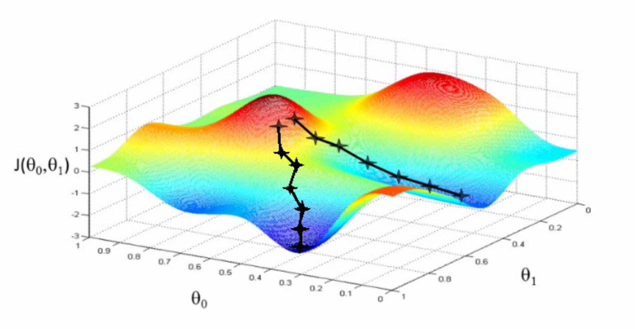
$c(output, truth)$

(Image: Intel)

- From back to front (problem: vanishing gradient for deep networks)
- Changes of the weights are usually dampened/controlled by changing the learning rate

Deep Neural Network: Stochastic Gradient Descent

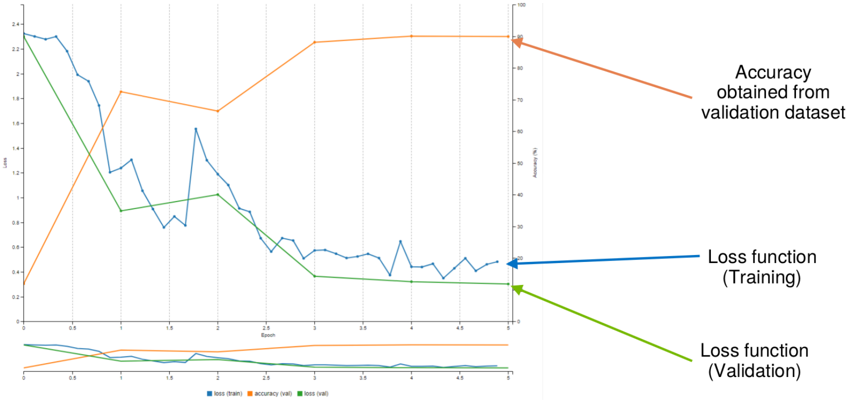
How to find the best weight updates:



(Image: blog.datumbox.com)

- Gradient descent methods, e.g.:
  - Stochastic Gradient Descent (SGD)
  - Adaptive Moment Estimation (ADAM)
- Example: only two weights (θ_1, θ_2) with cost in 3rd dimension
- Multiple (local) minima are possible

Training - Find the Best Weights



Accuracy obtained from validation dataset

Loss function (Training)

Loss function (Validation)

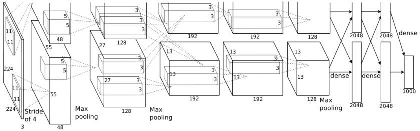
(Image: Nvidia)

- Data sets separated into training, validation, and testing sets
- Training data set is repeatedly used for training (over epochs)
- Validation data: Track the performance of the network during training
- Testing data set: Final independent performance validation

Deep Network Examples

AlexNet:

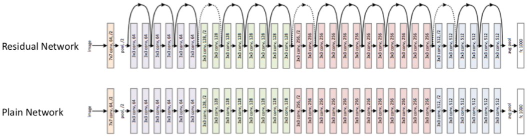
- Won ImageNet Challenge 2012
- 5 conv. + 3 fully connected layers
- 60 million parameters



(Image: Krizhevsky, et al.)

ResNet:

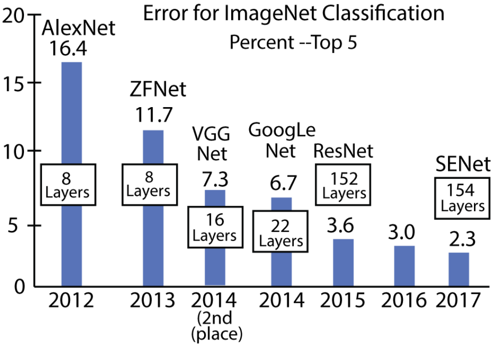
- Won ImageNet Challenge 2015
- Mitigates *vanishing gradient* problem
- 25 million parameters



(Image: He, et al.)

An overview of more CNNs can be found ▸ here

Image Classification Errors



(Image: principlesofdeeplearning.com)

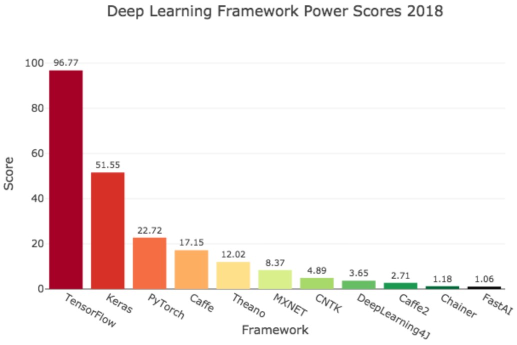
- Trend: More layers
- Error (performance) converges
- Ensemble networks were used last

Programming

How to "program" Deep Neural Networks is different:

- In two phases:
  - Training (time consuming)
  - Inference (usage)
- High quality and quantity training (and validation/testing) data is needed
- Output is probabilistic
- Programming with frameworks:
  - TensorFlow
  - CNTK
  - Theano

    } Keras
  - PyTorch
  - Caffe{2}
  - ...

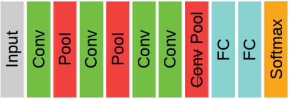


(Image: keras.io)

Programming

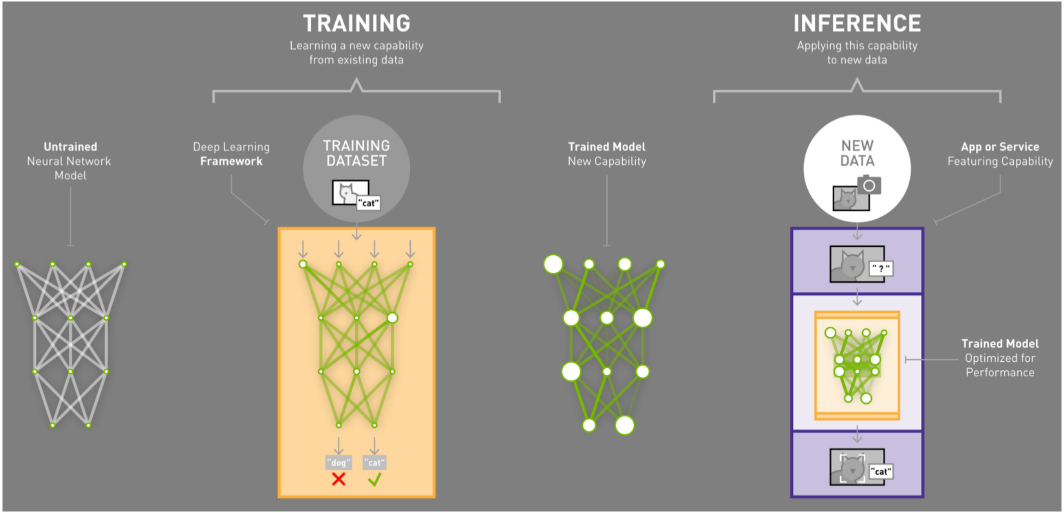
Example of AlexNet[2] with Keras:

```
from keras import Sequential
from keras.layers import Conv2D, MaxPooling2D,
  BatchNormalization, ZeroPadding2D, Dropout,
  Activation, Flatten, Dense

def alexnet(n_classes=5):
    model = Sequential()
    model.add(Conv2D(64, 11, strides=4))
    model.add(ZeroPadding2D(2))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=3,
                           strides=2))
    model.add(Conv2D(192, 5))
    model.add(ZeroPadding2D(2))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=3,

    model.add(Conv2D(384, 3))
    model.add(ZeroPadding2D(1))
    model.add(Activation('relu'))

    model.add(Conv2D(256, 3))
    model.add(ZeroPadding2D(1))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=3,
                           strides=2))
    model.add(Flatten())
    model.add(Dropout(0.5))
    model.add(Dense(4096,
              input_shape=(6 * 6 * 256, )))
    model.add(Activation('relu'))
    model.add(Dropout(0.5))
    model.add(Dense(4096))
    model.add(Activation('relu'))
    model.add(Dense(n_classes))
    model.add(Activation('softmax'))

    return model

if __name__ == '__main__':
    amodel = alexnet(10)
    amodel.summary()
```



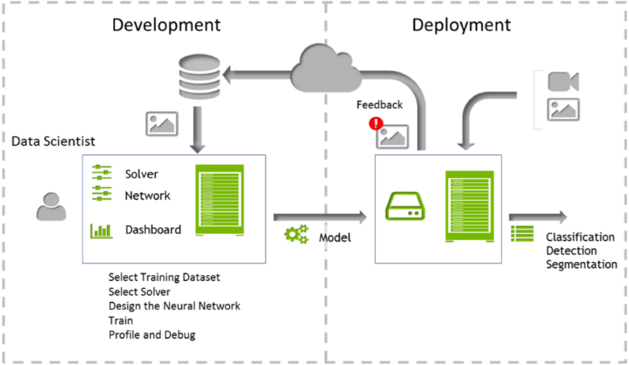
[2] A variant of the original AlexNet

Training vs. Inference


TRAINING
Learning a new capability from existing data
INFERENCE
Applying this capability to new data
Untrained Neural Network Model
Deep Learning Framework
TRAINING DATASET
"cat"
"dog"
"cat"
Trained Model New Capability
NEW DATA
"?"
App or Service Featuring Capability
Trained Model Optimized for Performance
"cat"

(Image: Nvidia)

Development & Deployment



(Image: Nvidia)

How to get Started?

Model Zoos make it easy to start:

- Use existing models
- Use pre-trained models for transfer learning

Model Zoo examples:

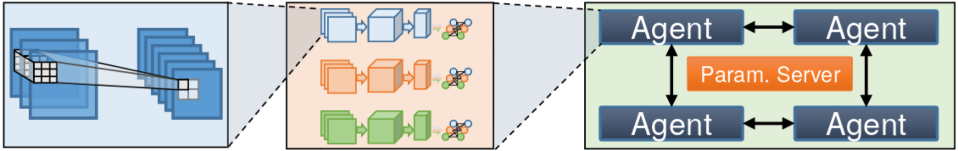
- Tensorflow: ▸ here
- PyTorch: ▸ here
- Caffe (BVLC): ▸ here
- ...

Pretrained models are also available (e.g. for ▸ object detection with Tensorflow)

Parallelism

Parallelism can be found in:

- Low level operations in the network
- Parallelized networks
- Distributed training

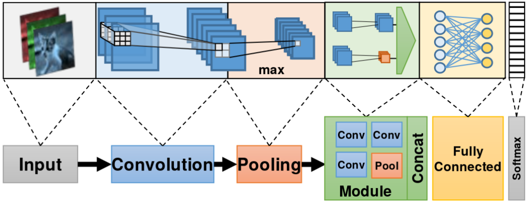


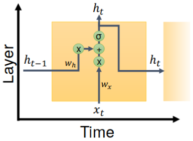
(Image: Ben-Nun, et al.)

Parallelism: Low Level Operations

Deep Networks consist of different building blocks:

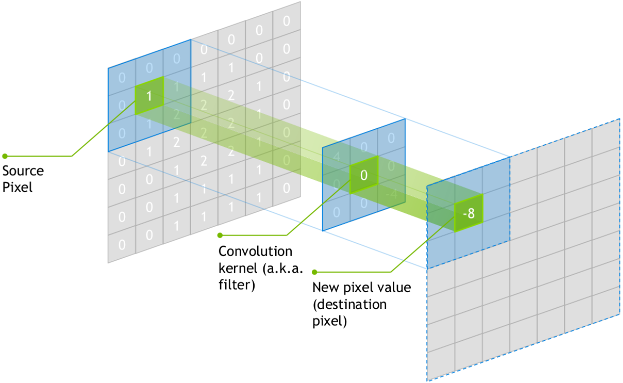
- Fully connected (dense) layer
- Convolution layer
- Pooling layer
- Recurrent Neuronal Network layer (RNN):
  Temporal information (e.g. Long-Short-Term Memory [LSTM])
- Batch normalization
- Activation functions





(Images: Ben-Nun, et al.)

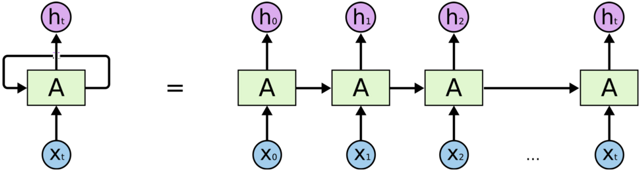
Excursion: Convolution


Source Pixel
1
Convolution kernel (a.k.a. filter)
0
New pixel value (destination pixel)
-8

(Image: Nvidia)

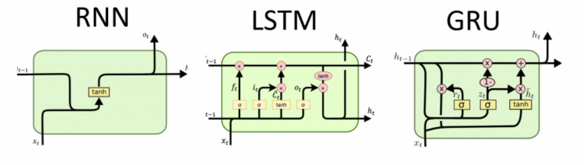
- Not fully connected (except if kernel is size of input)
- Convolutions can reduce the data
- Boundaries need handling (e.g. padding)

Excursion: RNNs



(Image: medium.com)

- Principle: Unroll cell over time
- Different types exist:
  - RNN (original)
  - GRU (Gated Recurrent Units)
  - LSTM (Long Short Term Memory)



(Image: medium.com/dprogrammer.org)

Math Involved

- Forward Propagation:
  - Inner product, vector- and matrix-matrix multiplications
  - Operations like activation functions, pooling, etc.
- Backward Propagation:
  - Differentiation
  - Solvers (SGD, ADAM . . .)

⇒ Linear Algebra (BLAS)

Implemented in[3]:

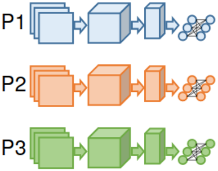
- Nvidia:
  NVIDIA CUDA® Deep Neural Network library (▸ cuDNN)
- Intel:
  Intel® Math Kernel Library for Deep Neural Networks (▸ MKL-DNN)

[3] cuBLAS, cuFFT, MKL and more are also used

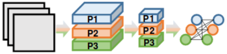
Parallelized Networks

Networks themselves can be parallelized:

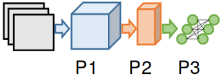
- Data parallelism



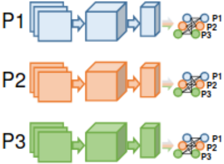
- Model parallelism



- Layer pipelining

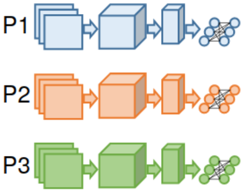


- Hybrid parallelism



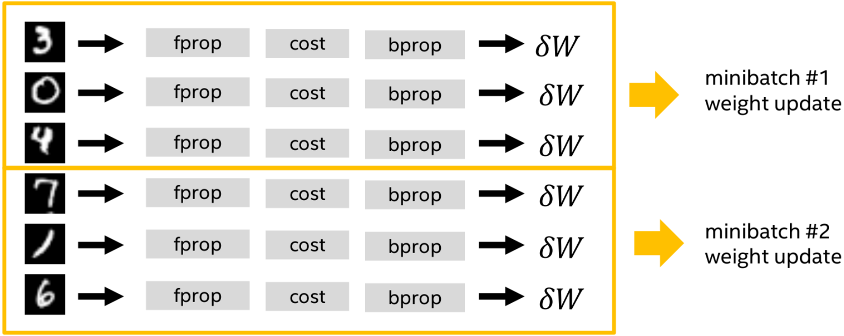
(Images: Ben-Nun, et al.)

Parallelized Networks: Data Parallelism



- Also called *pattern* parallelism and *bunch mode*
- Nowadays called: *minibatch*
- Across cores, sockets and nodes (multiple GPUs)

Excursion: Minibatch



3 → fprop cost bprop → δW

0 → fprop cost bprop → δW

4 → fprop cost bprop → δW

minibatch #1 weight update

7 → fprop cost bprop → δW

1 → fprop cost bprop → δW

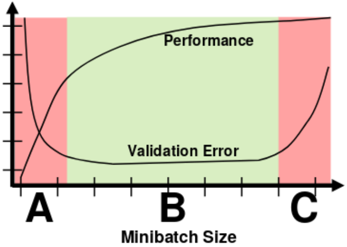
6 → fprop cost bprop → δW

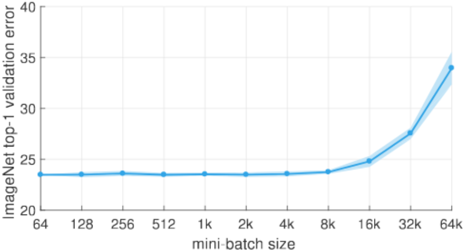
minibatch #2 weight update

(Image: Intel)

- Back-propagation is very expensive compared to forward-propagation
- Group training data in batches (so-called *minibatch*) of size N
- $N = \frac{training_{size}}{\#_{batches}}$
- A *minibatch* allows parallel forward-propagation

Minibatch Performance

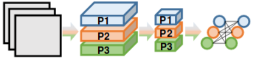




(Image: Ben-Nun, et al.)

- A higher mini-batch size increases performance
- **However:**
  - The larger the batch, the worse the training performance
  - The more memory is needed to store the parameters (problem for GPUs)
- Sweet spot needs to be found empirically

Parallelized Networks: Model Parallelism

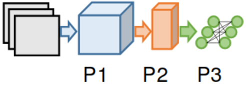


- Different network operations are executed on different cores
- Saves memory as model is distributed
- Could have significant communication needs
- Different approach: TreeNets with DNN ensembles

To mitigate communication needs:

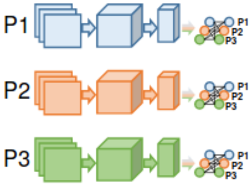
- Redundant computations
- Special optimizations:
  - Fully connected layers: Cannon's matrix
  - Convolutions: Locally Connected Networks (LCNs)

Parallelized Networks: Layer Pipelining



- Shares ideas from data and model parallelism
- Overlapping computations:
  - Widely used
  - Deep Stacking Network (DSNs)
- Partition by layers:
  - Only a subset of parameters per core (similar to model parallelism)
  - Layer boundaries define communication points
  - Caching can be exploited (parameters stay on same core)
  - Problem: Balancing of computational load is difficult

Parallelized Networks: Hybrid

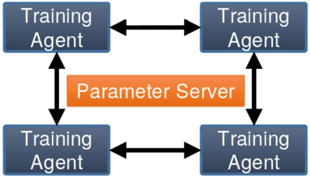


- Depends highly on the network topology
- E.g. AlexNet:
  - Convolutions are the most time critical part
  - Fully connected layer has most parameters (inbalance)
  - Solution:
    Data parallelism for convolutions and model parallelism for fully connected layers

Distributed Training

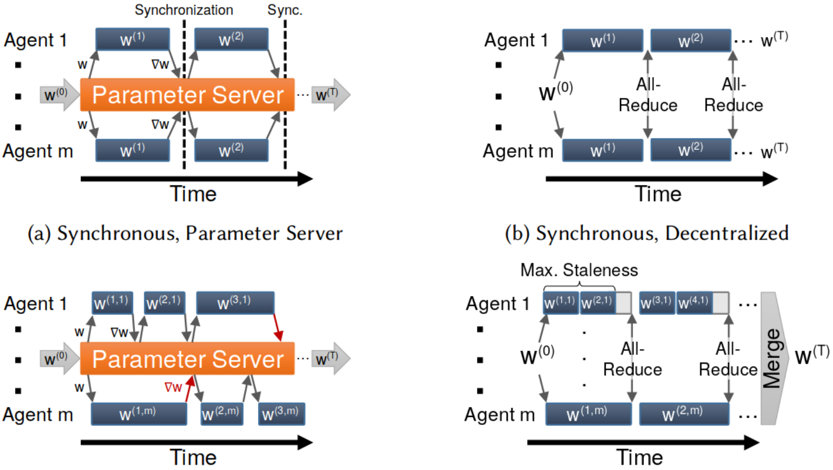
What if one node (GPU) is not enough?

- Model Consistency
- Parameter Distribution and Communication:
  Centralization and Compression (e.g. FP16)
- Training Distribution:
  Model Consolidation and Optimization Algorithms



(Image: Ben-Nun, et al.)

Distributed Training: Model Consistency



(a) Synchronous, Parameter Server

(b) Synchronous, Decentralized

(c) Asynchronous, Parameter Server

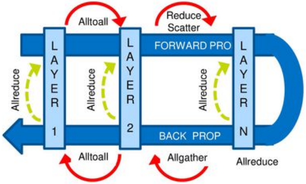
(d) Stale-Synchronous, Decentralized

(Image: Ben-Nun, et al.)

Distributed Training: Libraries

Different backends:

- Message Passing Interface (MPI)
- Nvidia Collective Communications Library (NCCL)
- Intel Machine Learning Scaling Library ▸ Intel MLSL (uses MPI)



(Image: Intel (MLSL))

Strategies vary among frameworks:

- Tensorflow ▸ Horovod (NCCL + MPI)
- PyTorch supports MPI, NCCL and Gloo (default)

Questions

Q&A

VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

IT4Innovations National Supercomputing Center

VŠB – Technical University of Ostrava
Studentská 6231/1B
708 00 Ostrava-Poruba, Czech Republic
www.it4i.cz

EUROPEAN UNION
European Structural and Investment Funds
Operational Programme Research,
Development and Education

MINISTRY OF EDUCATION, YOUTH AND SPORTS