XILINX
ALL PROGRAMMABLE™

Training Quantized Neural Networks

Nick Fraser, Giulio Gambardella, Michaela Blott, Thomas Preusser

Xilinx Research, Ireland

Xilinx Research - Ireland

- **Part of the CTO organization**
  - 9 (out of 35 worldwide) researchers
- **With a very active internship program**
  - 6-10 students & visiting scholars
- **Visiting professors on sabbatical**
- **Postdoc on Marie-Curie Fellowship**

IDA Ireland



XILINX ALL PROGRAMMABLE.

New York Times: "The Great A.I. Awakening"

(Dec 2016)

Elon Musk's Billion-Dollar AI Plan Is About Far More Than Saving the World

The Race For AI: Google, Twitter, Intel, Apple In A Rush To Grab Artificial Intelligence Startups

World's Largest Hedge Fund to Replace Managers with an AI System

Drones Can Defeat Humans Using Artificial Intelligence

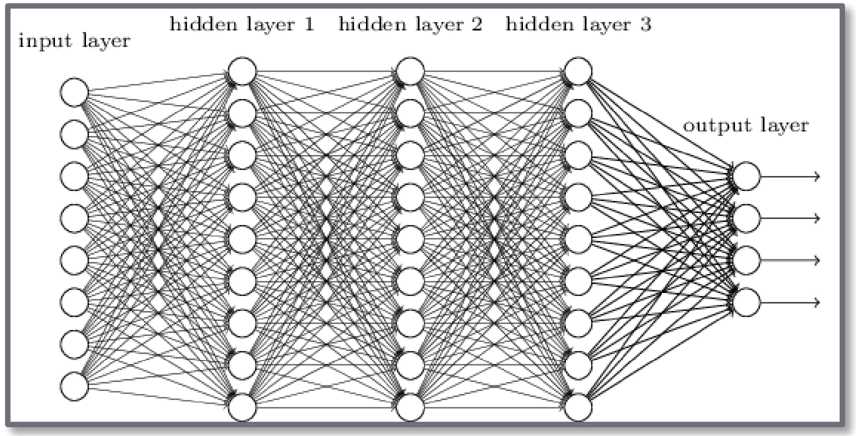
Elon Musk's leads 116 Experts on Open Letter to Ban Killer Robots

- **Demonstrated to work well for numerous use cases**

Neural Networks

- **NNs are the predominant AI algorithm**
  - Can outperform humans and traditional CV algorithms for image recognition

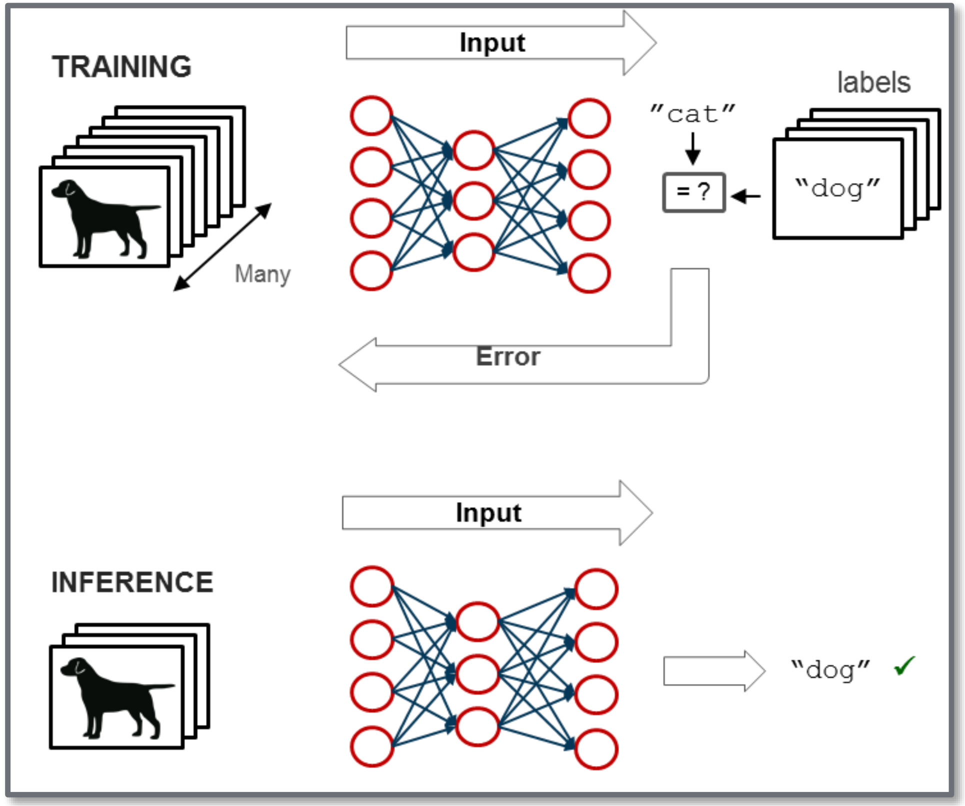
- **NNs have the theoretical property of being a “universal approximation function”**
  - Empirically outperforming other approximator functions



Increasing adoption: replacing other solutions and for previously unsolved problems

XILINX ALL PROGRAMMABLE™

Neural Networks: Training vs Inference



Training
Process for a machine to ***learn*** by optimizing models (weights) from data.

- Requires little expertise/specialization in the actual target domain.

Inference
Using trained models to predict or estimate outcomes from new observations.

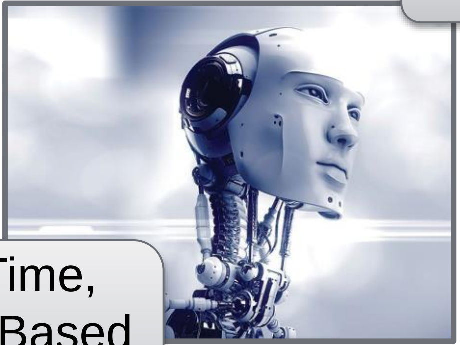
XILINX ALL PROGRAMMABLE.

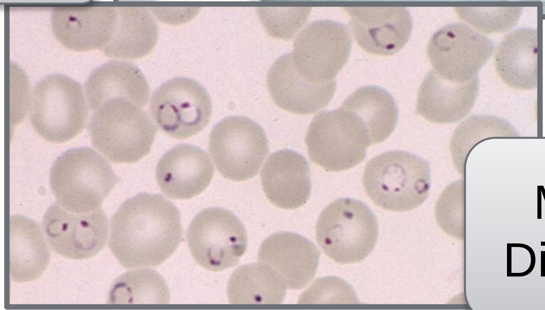
Challenges: Wide & Increasing Range of Applications

Translation Service

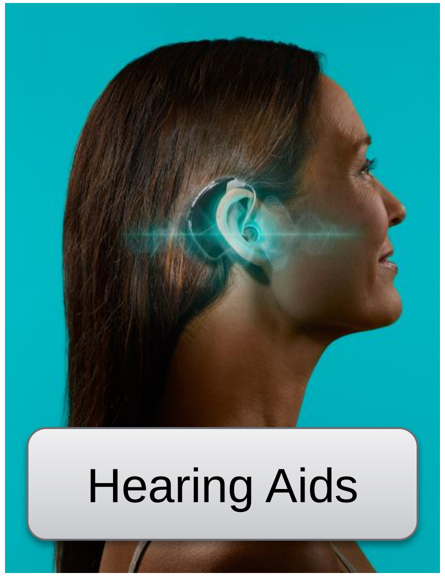
3D Reconstruction from Drone Images

ADAS



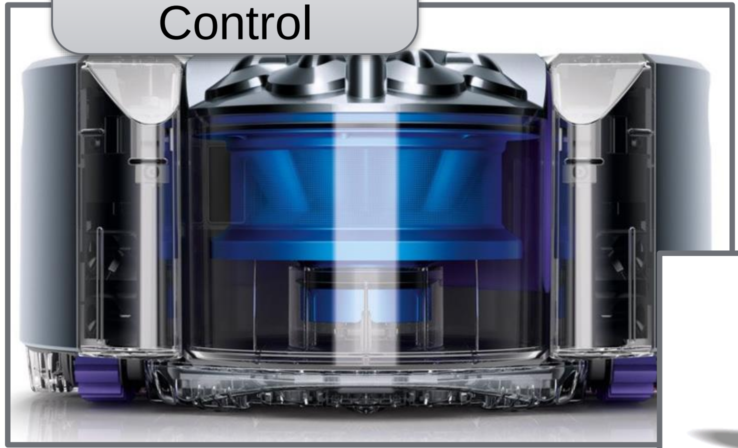


Medical Diagnoses

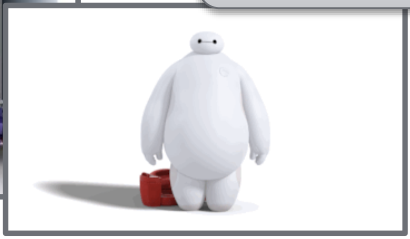


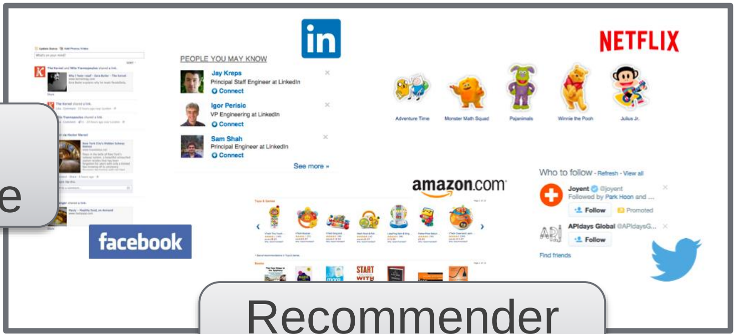
Hearing Aids

Real-Time, Sensor-Based Control



Health Assistance





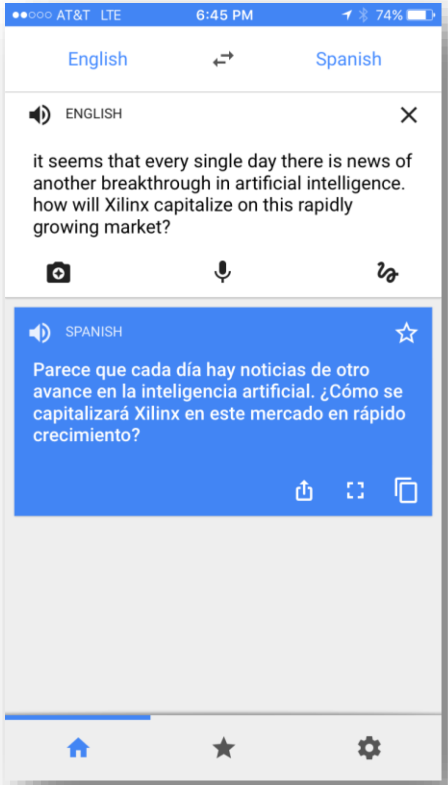
Recommender Systems

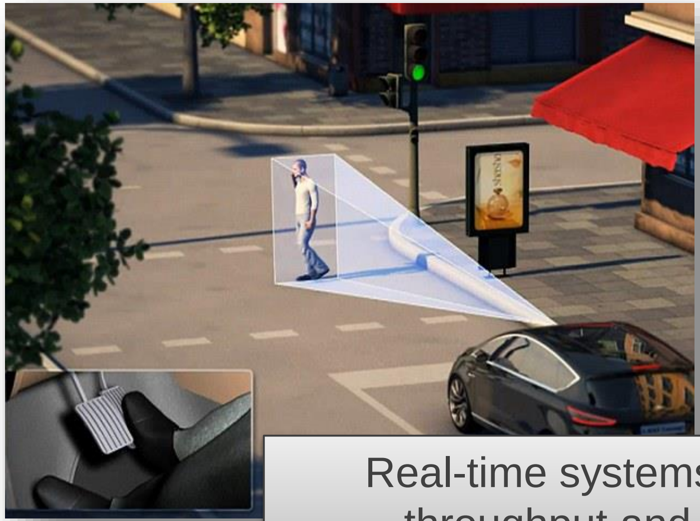
XILINX ALL PROGRAMMABLE.

Challenges: Different Figures of Merits

Accuracy requirements vary with applications:
Recommender systems, data analytics vs ADAS.

Reduced latency: Results in a better user experience in cloud-based systems (Google defines 7ms) and vital for robotics.





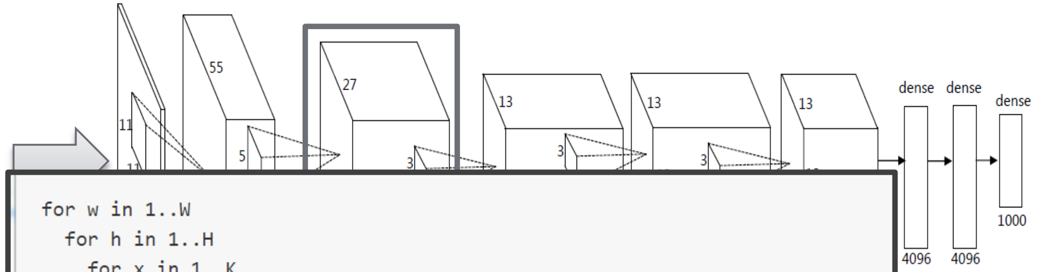
Real-time systems have clearly defined throughput and latency constraints.

Embedded Systems: heavily power constrained
Data Centers: OPEX = f(energy)

XILINX ALL PROGRAMMABLE.

Challenges: Highly Compute and Memory Intensive

- **The predominant CNN computation is linear algebra**
  - Demands lots of (simple) computation and lots of parameters (memory)
    - AlexNet: 244 MB & 1.5 GOPS, VGG16: 552 MB & 30.8 GOPS; GoogleNet: 41.9 MB & 3.0 GOPS for ImageNet



«cat»

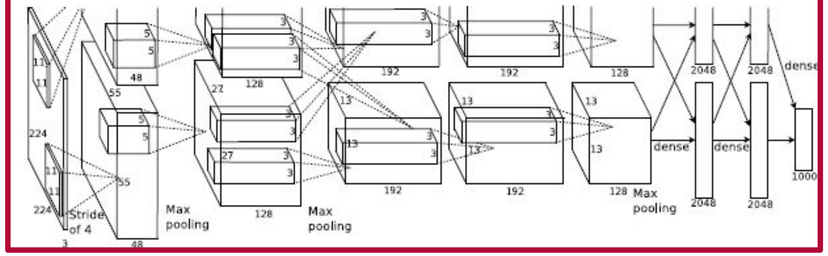
```
for w in 1..W
  for h in 1..H
    for x in 1..K
      for y in 1..K
        for m in 1..M
          for d in 1..D
            output(w, h, m) += input(w+x, h+y, d) *
          end
        end
      end
    end
  end
```

```
Output(w,h,m) +=
input(w+x,h+y,d)*filter(m,
x,y,d);
```

Challenge 2:
billions of multiply-accumulate ops & tens of megabytes of parameter data

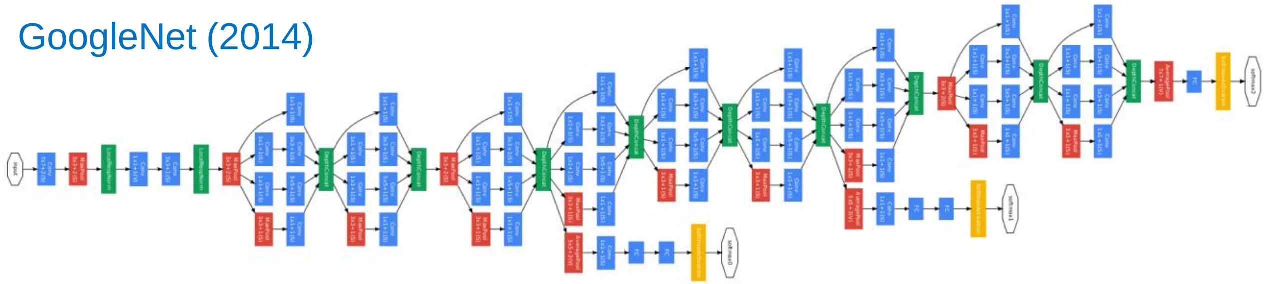
Challenges: Neural Networks Will Continue to Change

AlexNet (2012)

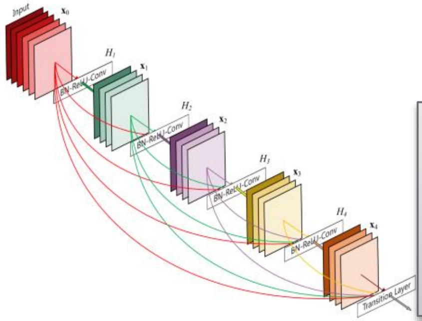


Number and **types** of layers are changing

GoogleNet (2014)



DenseNet (2016)



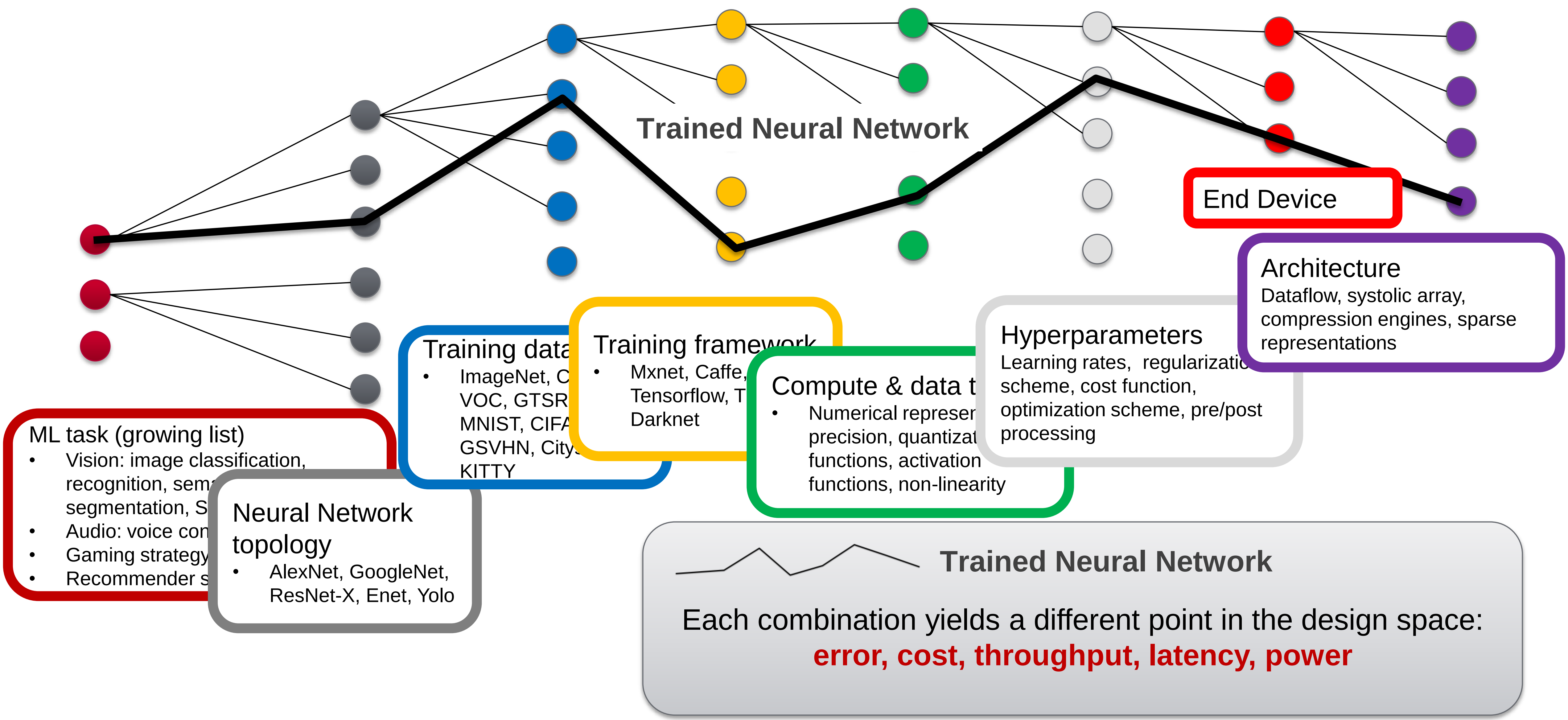
Data representations and **quantization methods** are changing

Graph Connectivity is changing

Challenge:
Continuous stream of new algorithms

XILINX ALL PROGRAMMABLE.

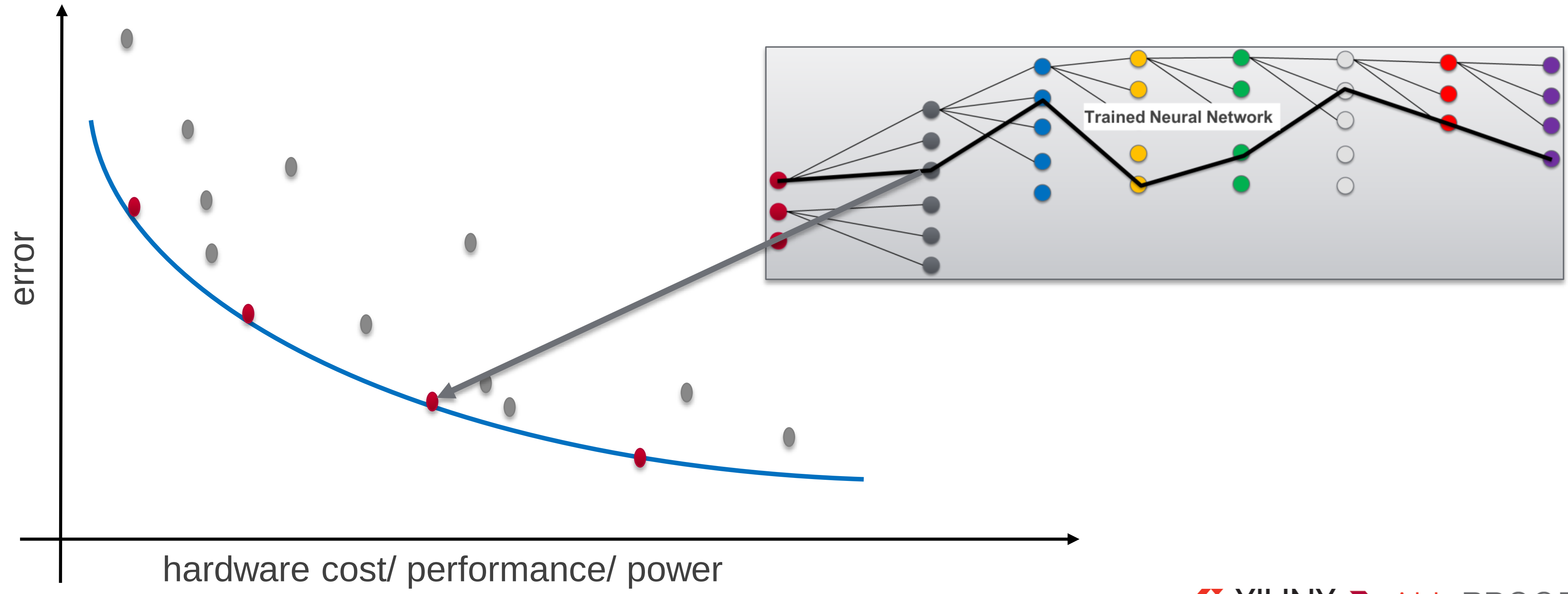
Challenge: Multidimensional Design Space



XILINX ALL PROGRAMMABLE.

Opportunity: Customized Neural Networks

- **Design and training of FPGA-friendly neural networks that provide end-solutions that are high-performance and more power-efficient than any other hardware**
  - Hardware cost, power, performance, latency

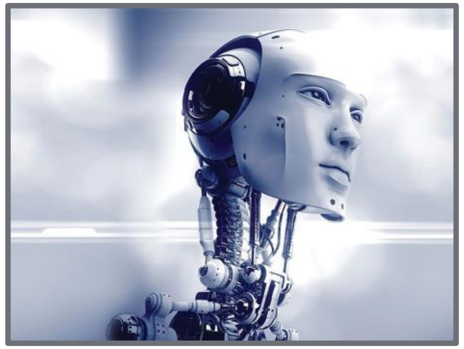


XILINX ALL PROGRAMMABLE.

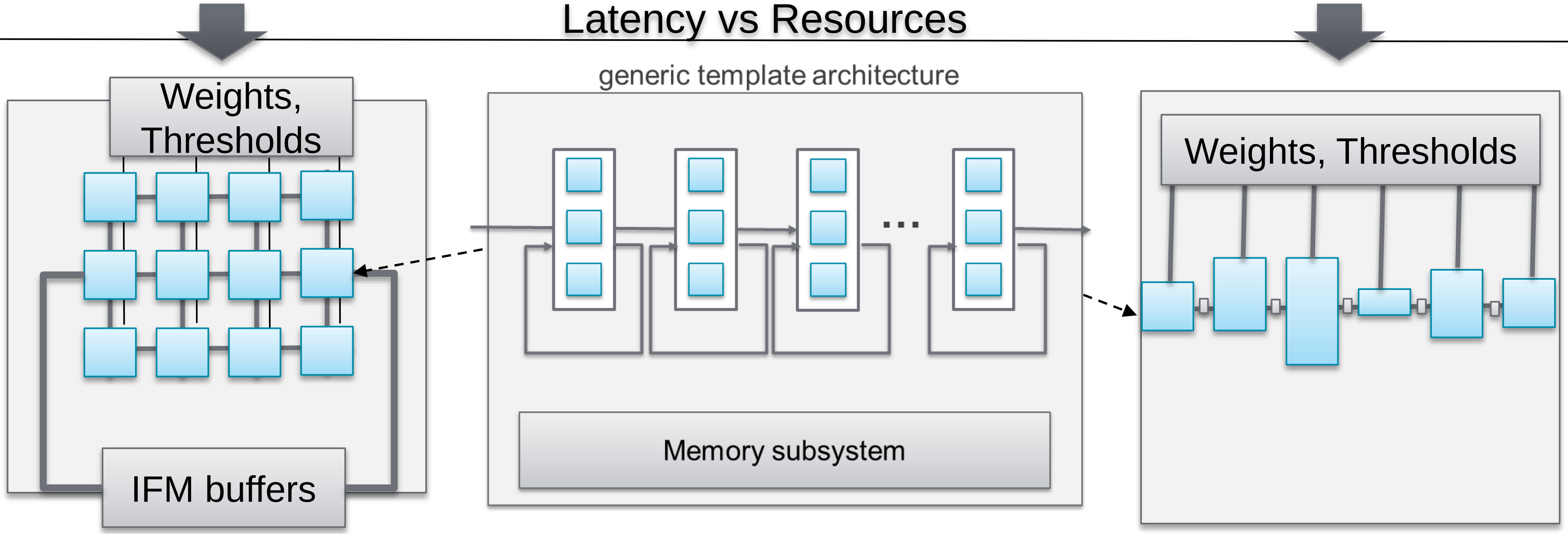
Opportunity: Customized ML Processor Datapath

AlphaGo

Latency vs generality



Generality vs Performance
Latency vs Resources



XILINX ALL PROGRAMMABLE™

Focus: Reduced Precision - Quantization

- **Cost per operation is greatly reduced**
- **Memory cost is greatly reduced**
  - Large networks can fit entirely into on-chip memory (OCM) (UltraRAM, BRAM)

- **Today's FPGAs have a much higher peak performance for reduced precision operations**

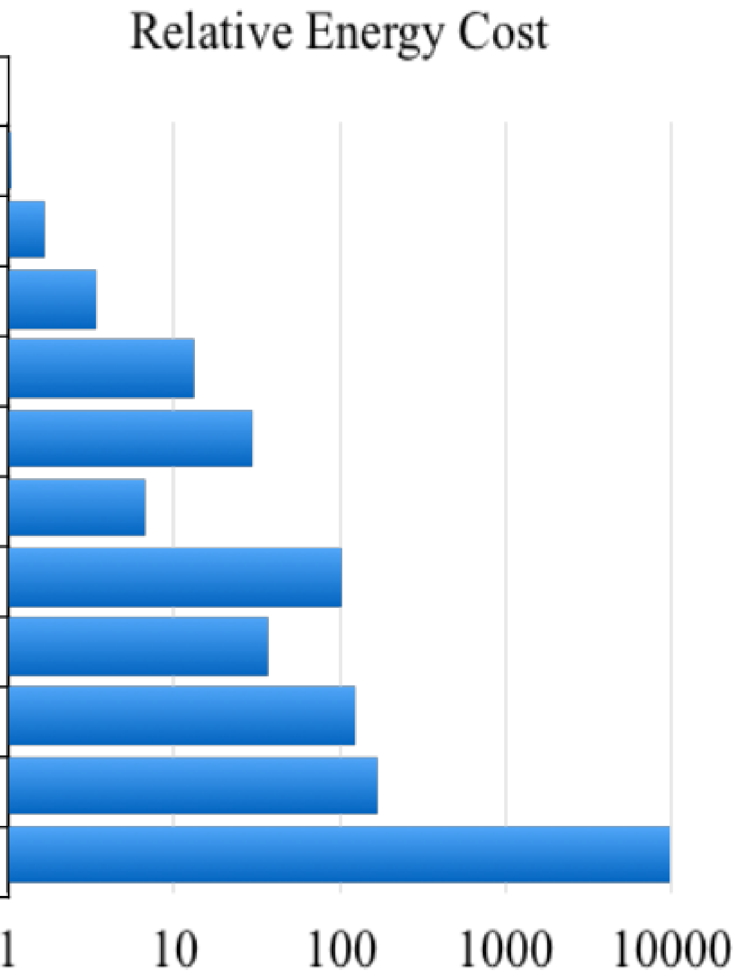
Precision	Cost per Op LUT	Cost per Op DSP	MB needed (AlexNet)	TOps/s (KU115)*	TOps/s (VU9P)**	TOps/s (ZU19EG)*
1b	2.5	0	7.6	~46	~100	~66
4b	16	0	30.5	~11	~15	~16
8b	45	0	61	~3	~6	~4
16b	15	0.5	122	~1	~4	~1
32b	178	2	244	~0.5	~1	~0.3

100x

*Assumptions: Application can fill device to 70% (fully parallelizable) 250MHZ
**Assumptions: Application can fill device to 70% (fully parallelizable) 300MHZ

Quantizing and Fixed Point saves Power

Operation:	Energy (pJ)
8b Add	0.03
16b Add	0.05
32b Add	0.1
16b FP Add	0.4
32b FP Add	0.9
8b Mult	0.2
32b Mult	3.1
16b FP Mult	1.1
32b FP Mult	3.7
32b SRAM Read (8KB)	5
32b DRAM Read	640

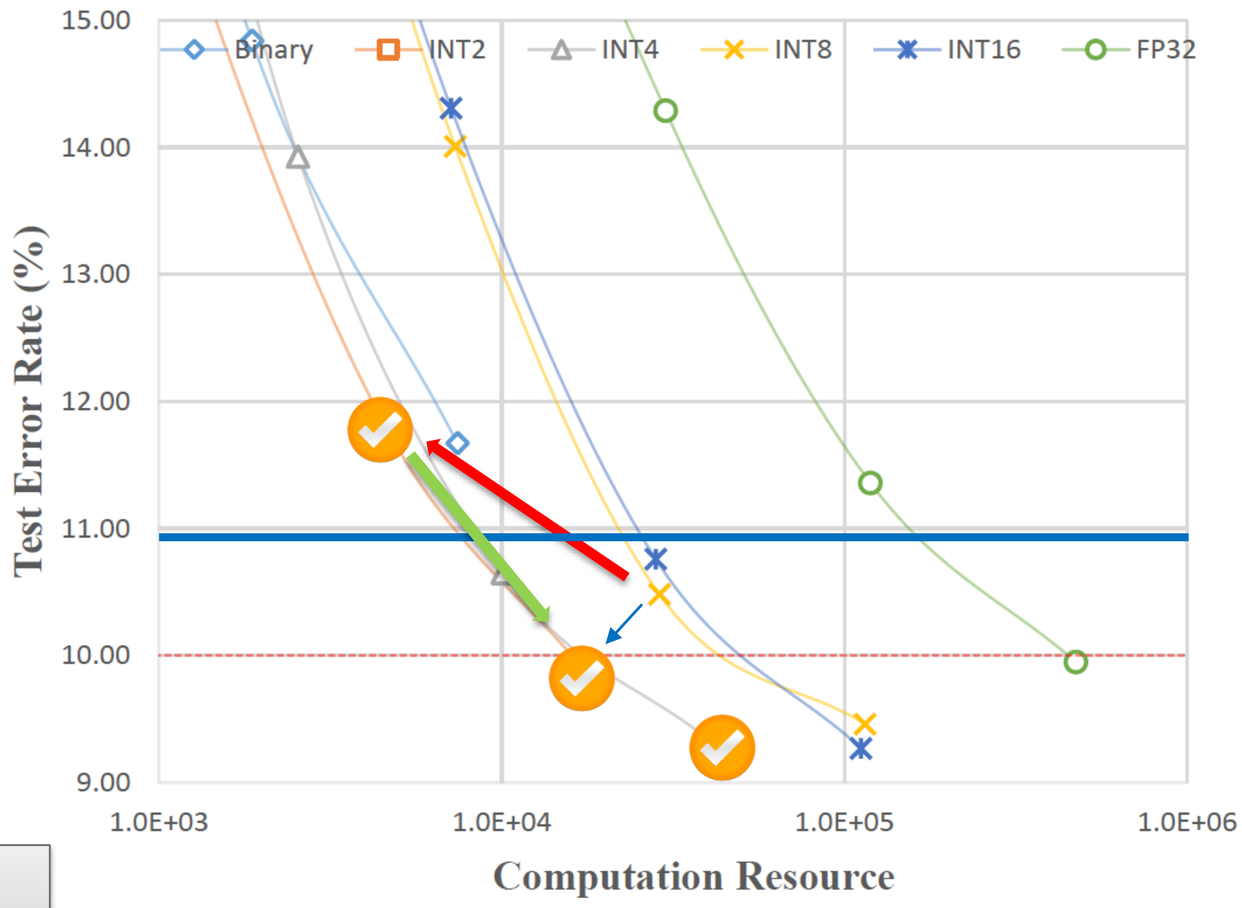


Source: Bill Dally (Stanford), Cadence Embedded Neural Network Summit, February 1, 2017

Do we loose Accuracy?

Compensating Quantization with Network Complexity

- **Just reducing precision, reduce hardware cost & increases error**
- **Recuperate accuracy by retraining & increasing network size**
- **1b, 2b and 4b provide pareto optimal solutions**

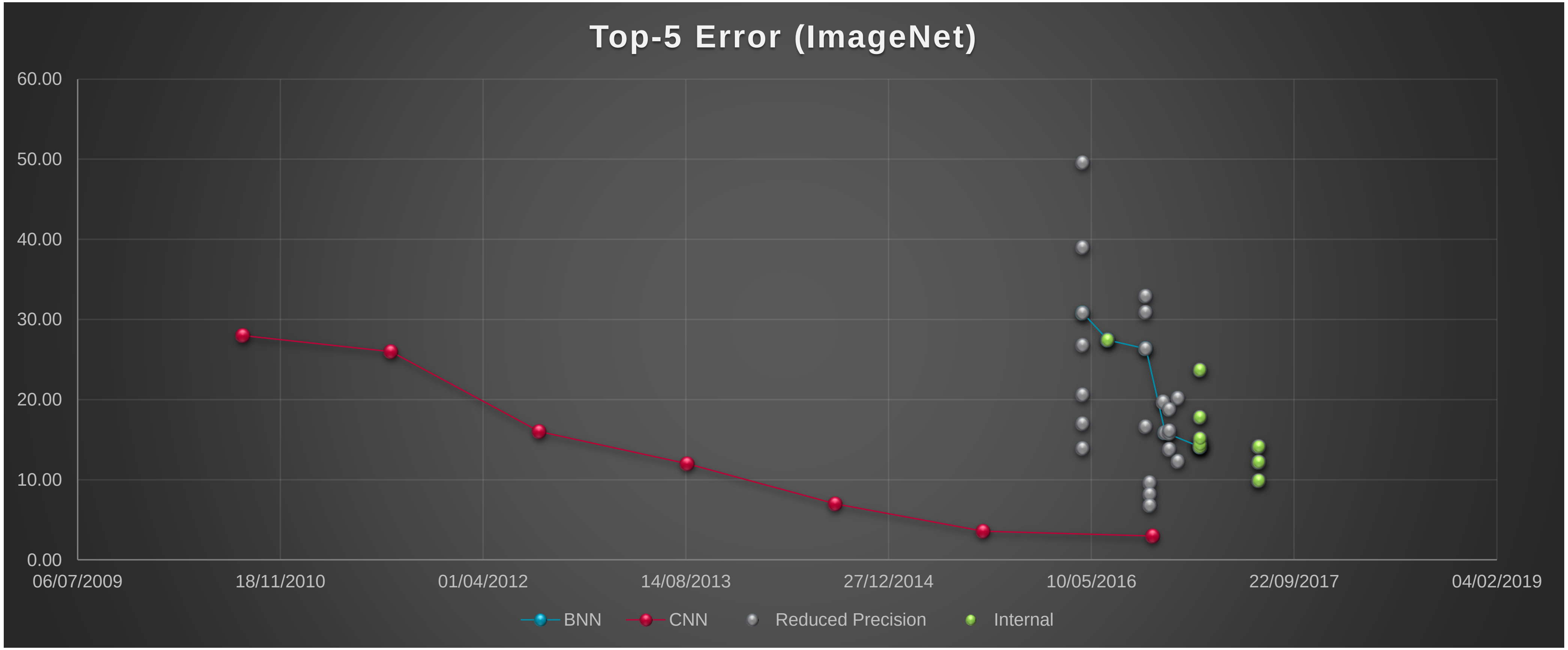


- Intel: Wide Reduced Precision Networks https://arxiv.org/pdf/1709.01134.pdf

XILINX ALL PROGRAMMABLE.

Accuracy of Quantized Neural Networks (QNNs) Improving

Published Results for FP CNNs, QNNs and binarized NNs (BNNs)

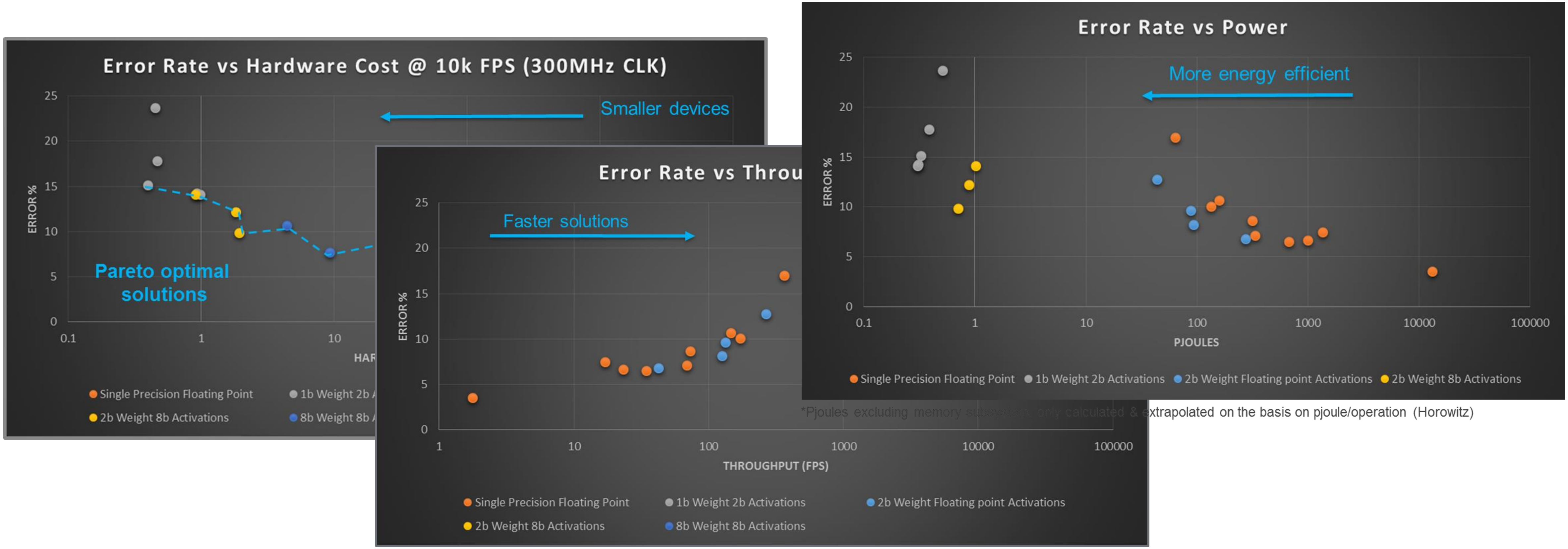


- Accuracy results are improving rapidly through for example new training techniques, topological changes and other methods

XILINX ALL PROGRAMMABLE.

Summary

- Quantized Neural Networks provide the opportunity to create hardware implementations that are faster, smaller, or more power-efficient.



*Pjoules excluding memory ... extrapolated on the basis on pjoule/operation (Horowitz)

XILINX ALL PROGRAMMABLE.

Agenda

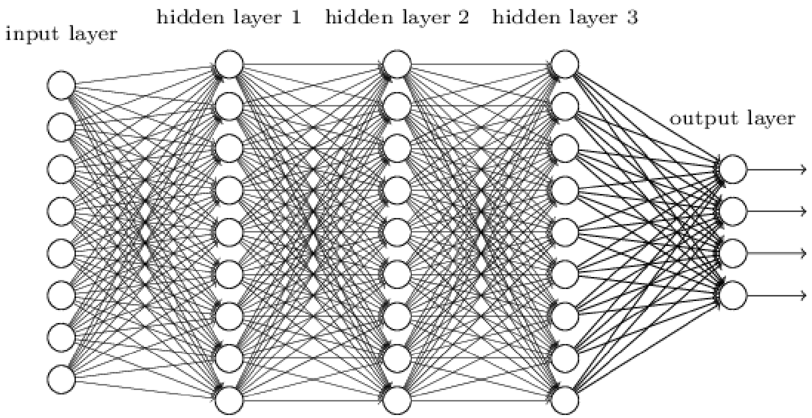
- **Introduction to Neural Networks:**
  - Neural network layers
  - The backpropagation algorithm
- **Quantized Neural Networks**
  - Data representations
  - Binarized Neural Networks
  - Quantization-aware backpropagation
- **Training Binary Neural Networks in Lasagne**

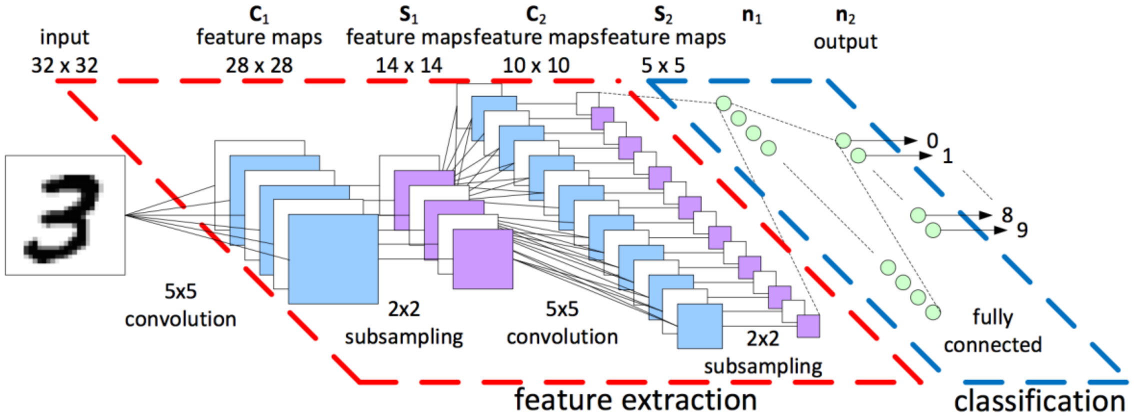
XILINX ALL PROGRAMMABLE™

Neural Networks: A Quick Introduction

Neural Networks - Layers

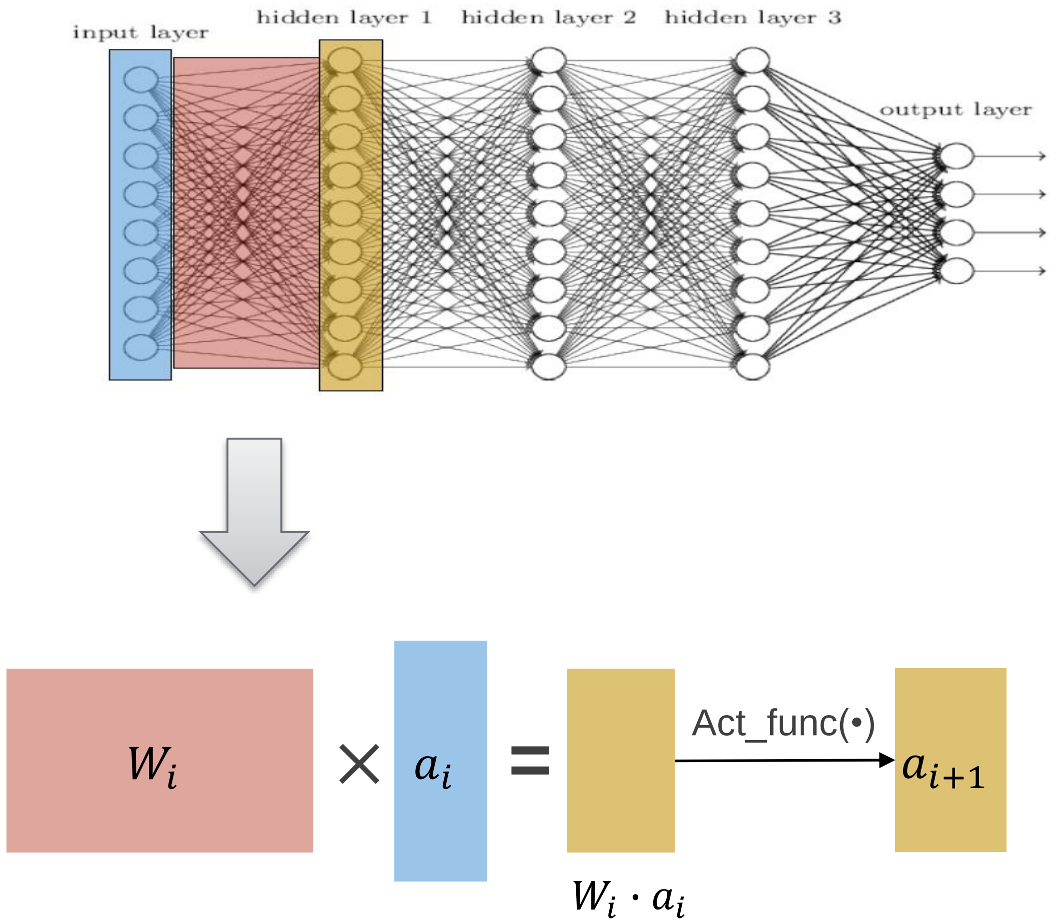
- Neural networks are computational graphs constructed from one or more layers.
- Layers: Usually linear operations followed by a non-linear activation function
  - Dot product = fully connected layer
  - 2D convolution = convolutional layer
- Other common layers:
  - Pooling layers (Max / Average)
  - Batch normalization





Neural Networks – Fully Connected Layer

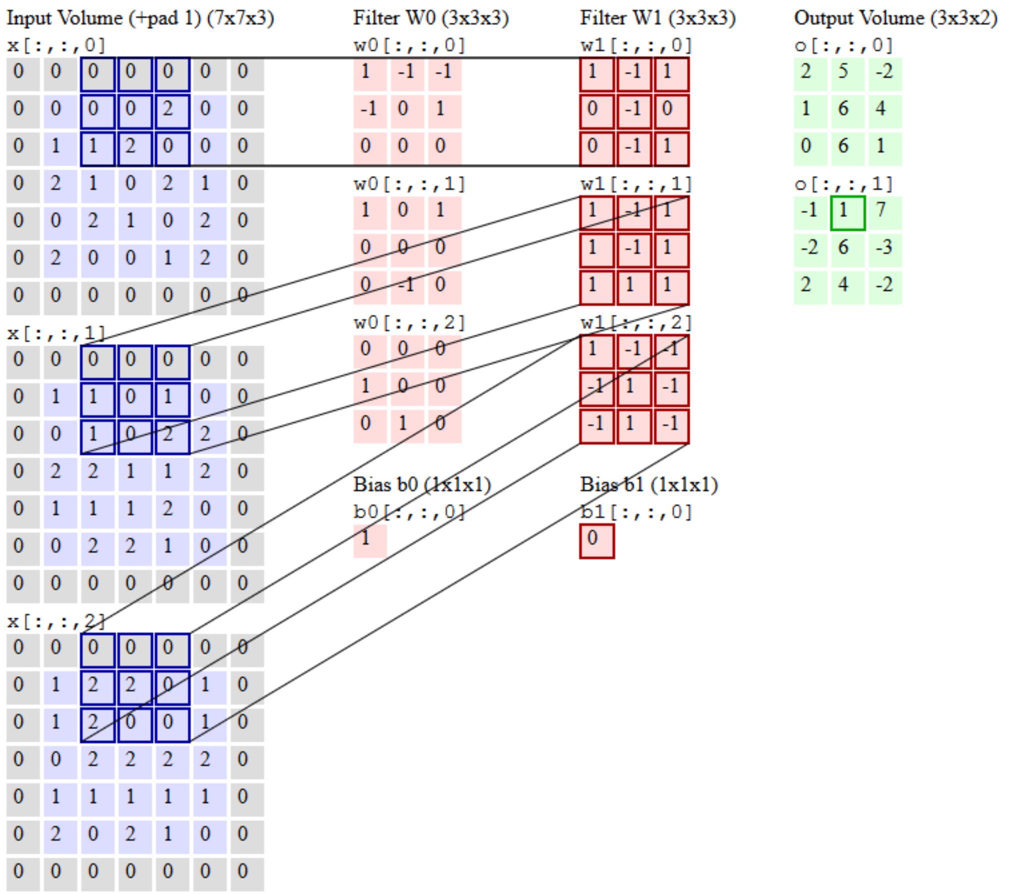
- Also known as: *inner product layer* or *dense layer*.
- Each neuron is connected to every neuron of the previous layer.
- A weight is associated with each “*synapse*”.
- Can be written as a matrix-vector product with an element-wise non-linearity applied afterwards.



XILINX ALL PROGRAMMABLE.

Neural Networks – Convolutional Layer

- Each neuron applies a convolution to all images in the previous layer.
- Weights represent the filters used for convolutions.
- Can be *lowered* to a matrix-matrix multiply.
- Non-linear activation applied to each output pixel.

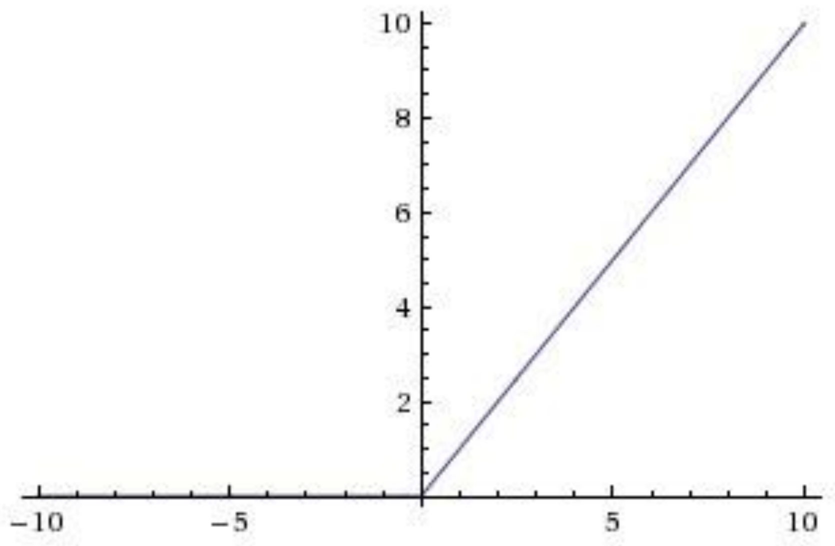


Source: http://cs231n.github.io/assets/conv-demo/index.html

XILINX ALL PROGRAMMABLE.

Neural Networks – Activation Functions

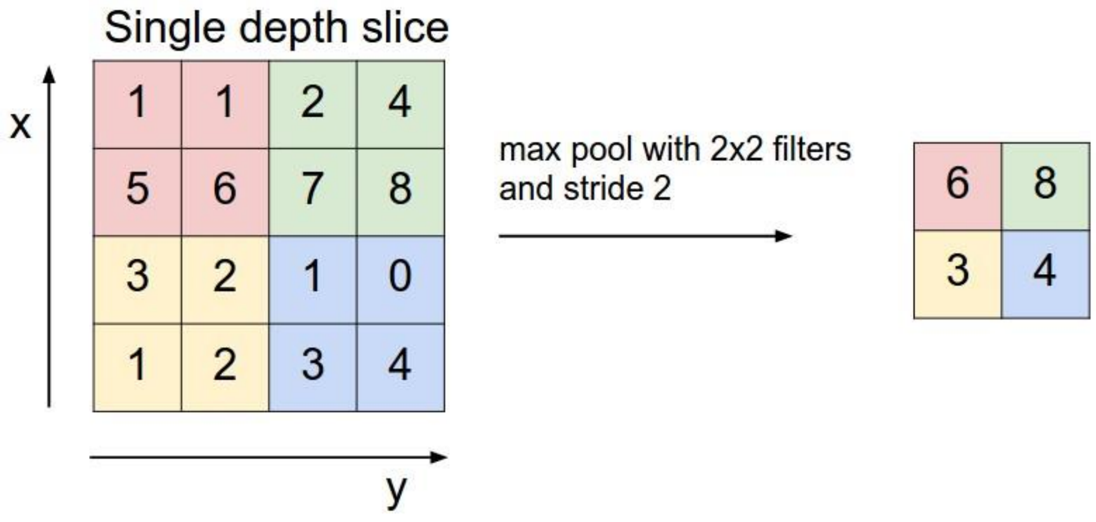
- Most popular: the rectified linear unit (ReLU)
- Other common ones include: tanh, leaky ReLU.
- For binarized neural networks, the step function is often used.



Source: http://cs231n.github.io/neural-networks-1/

Neural Networks – Pooling Layer

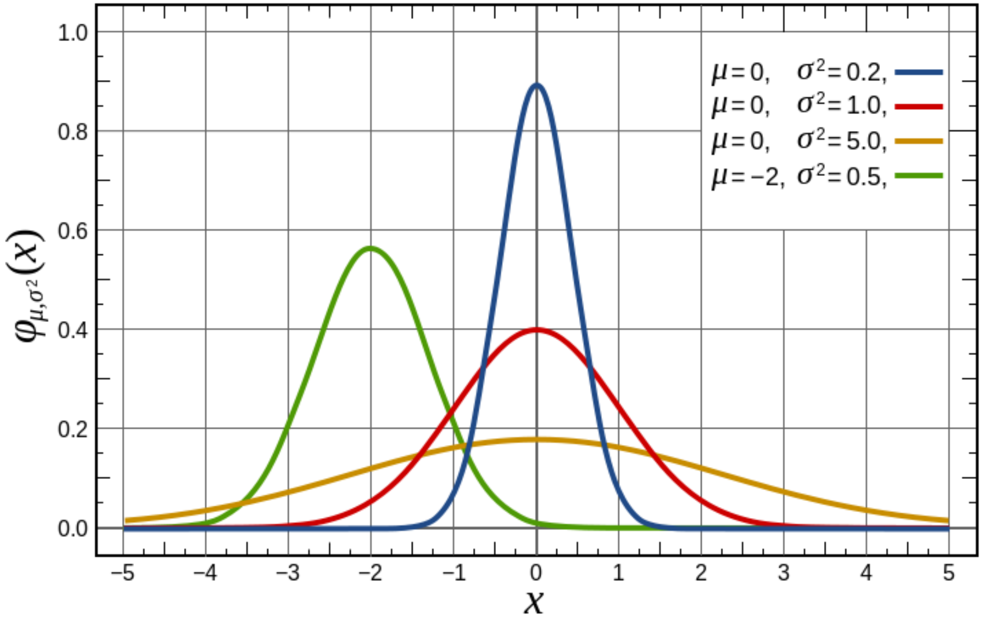
- Crude downsamplers of images.
- Reduces compute in subsequent layers.
- Max pooling takes the maximum value from a window of pixels.
- Average pooling is another common type.



Source: http://cs231n.github.io/convolutional-networks/

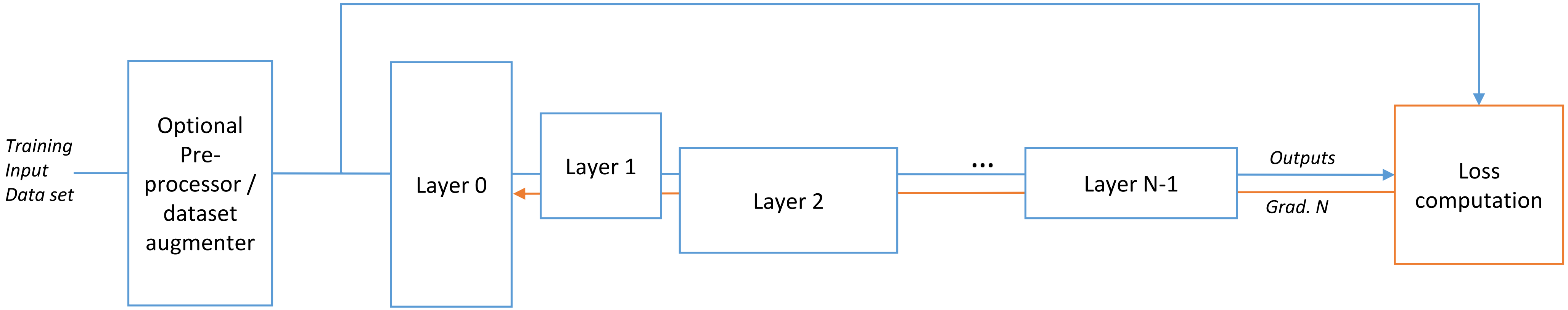
Batch Normalization Layer

- **Normalizes the statistics of activation values of particular neurons.**
- **Adds post-scaling to allow some neurons to be "more important" than others.**
- **Significantly reduces the training time of networks.**
- **Can improve the accuracy.**



XILINX ALL PROGRAMMABLE.

Training Neural Networks - Backpropagation

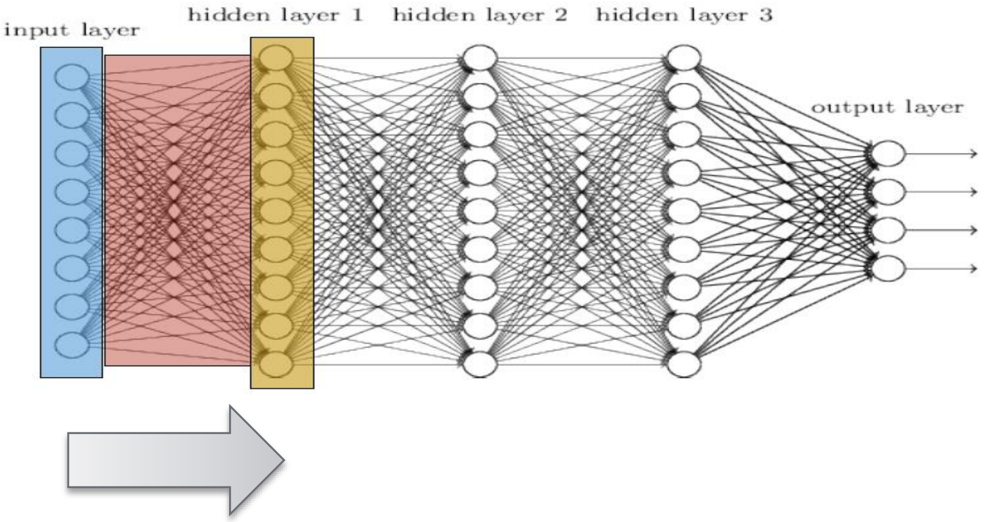


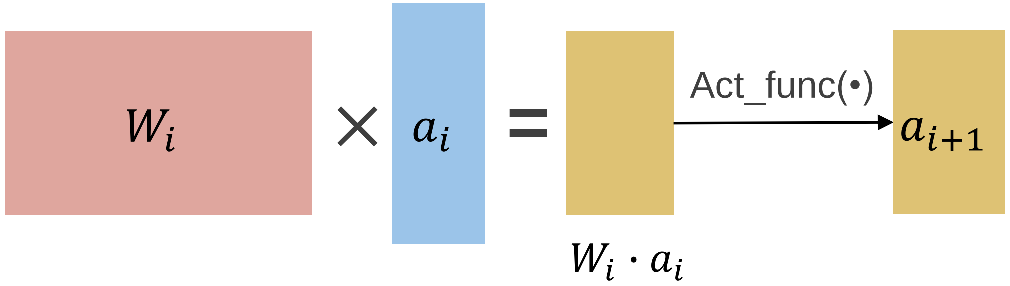
- **Purpose: calculate the gradients associated with each weight within a network.**

- **Forward path is the same as inference.**
- **Gradients calculated from a semi-differentiable loss function.**
- **Gradients passed back and transformed layer-by-layer.**
- **Weights updated from the provided gradients, input activations and an optimization algorithm.**

Backpropagation: Forward Path

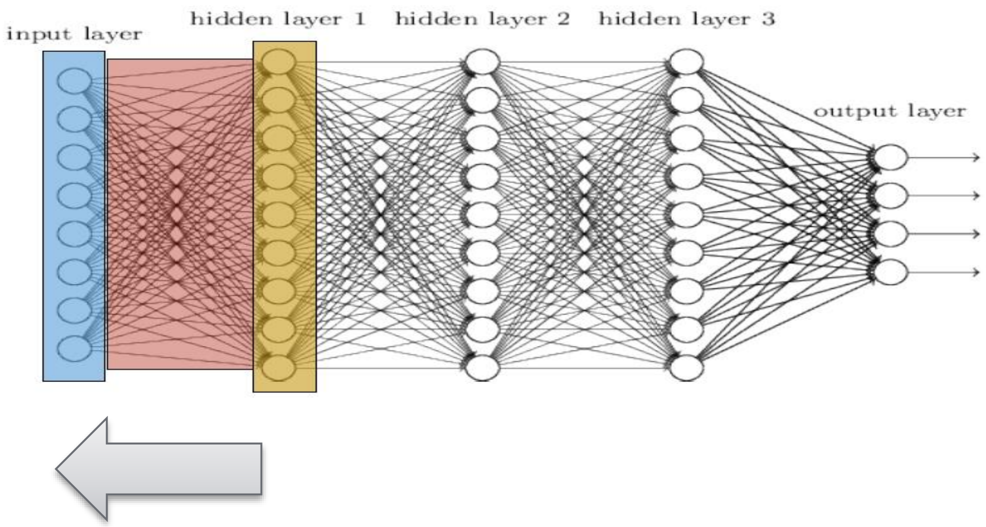
- Same as Inference:

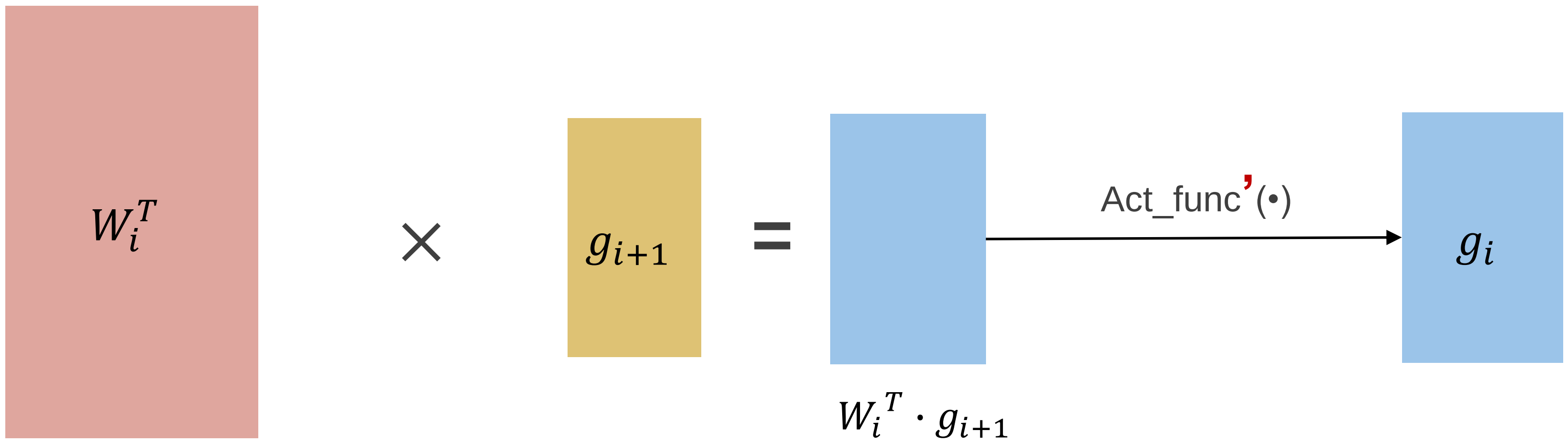




Backpropagation: Backward Path

- Pass gradients back through network:

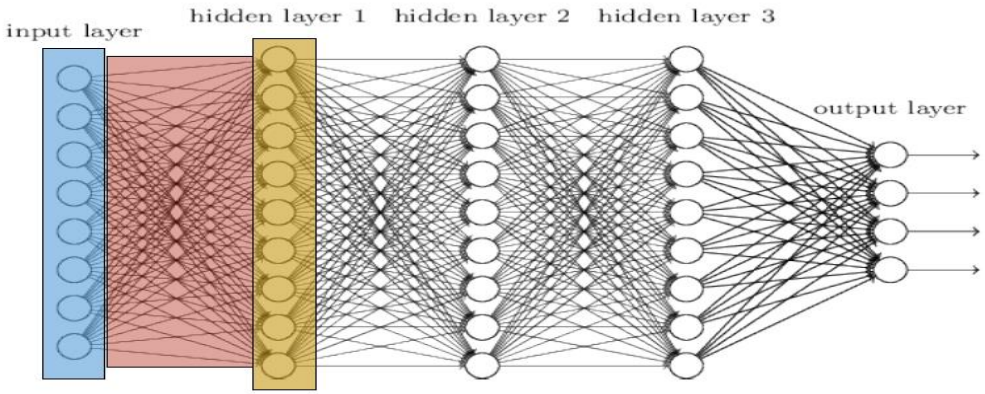




XILINX ALL PROGRAMMABLE™

Backpropagation: Weight Update

- Typically with an optimized weight update:
  - *Stochastic* gradient descent.
  - Adam.

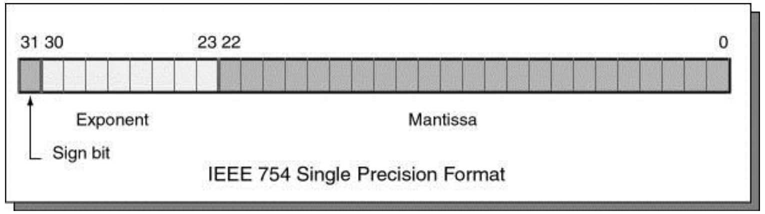


$$W_i^{+} := W_i + a_i \times {g_{i+1}}^{T}$$

Quantized Neural Networks

Data Representations & Reduced Precision

- **Floating Point**
  - Usually 32-bits
  - Large range, high precision
- **Fixed Point**
  - Fixed range
  - Simpler hardware
- **Binarized**
  - Multiply-accumulate becomes XNOR-popcount
  - 32x memory reduction
  - Extreme performance possible on FPGAs



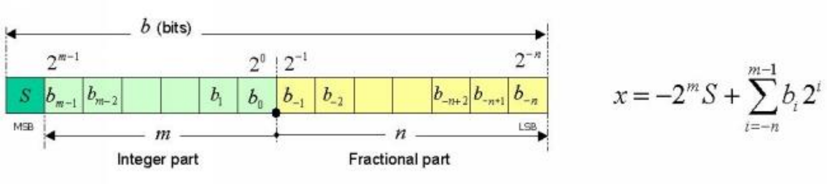
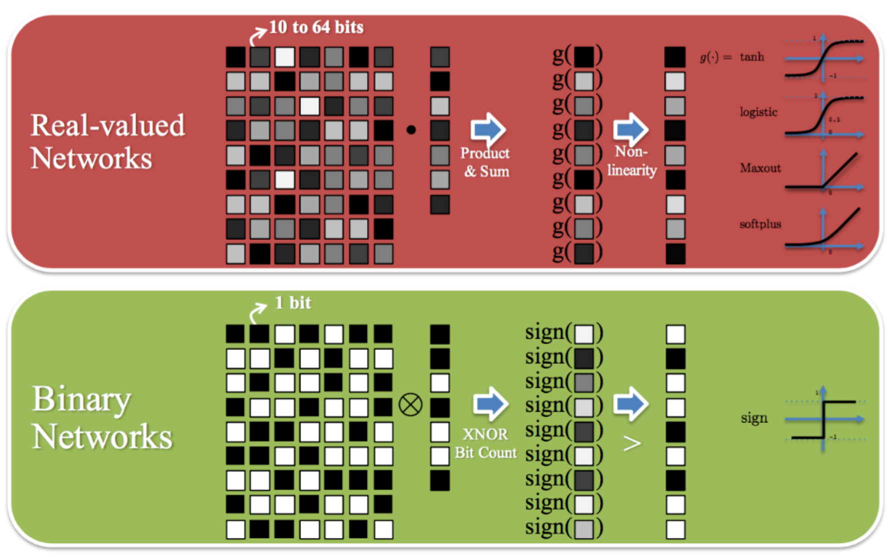


Fig 1 : Fixed-point data specification



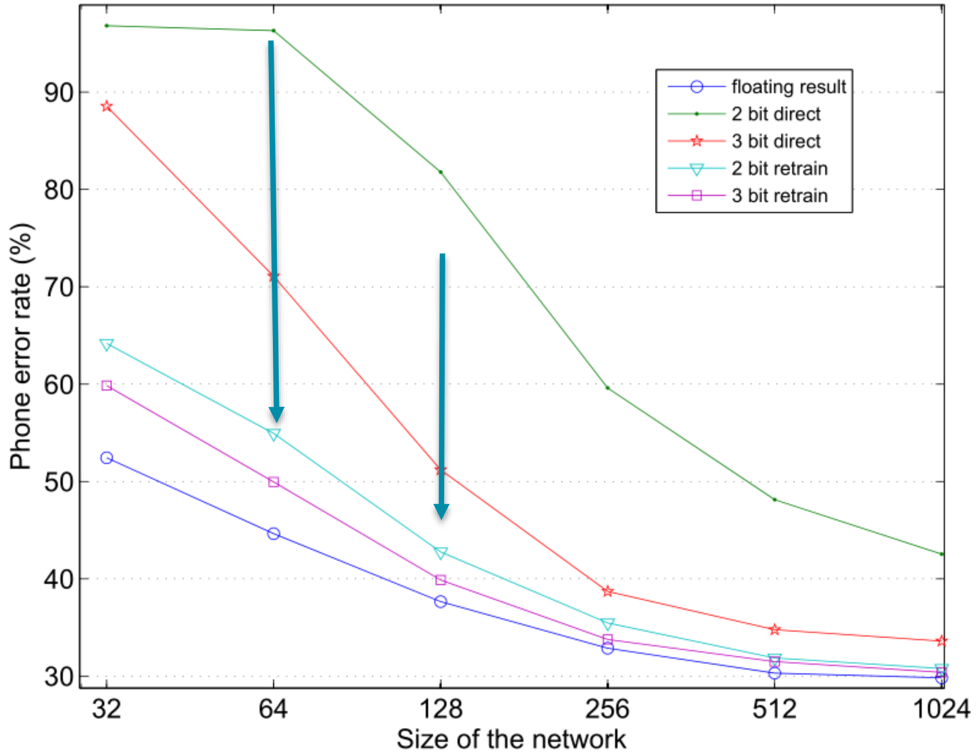
A forwardprop in a real-valued network versus in a BNN

XILINX ALL PROGRAMMABLE

Key Training Challenges When Reducing Precision

- **Training must be aware of quantization**
  - Direct quantization from FP -> RP tends to ruin accuracy when going below 8 bits.

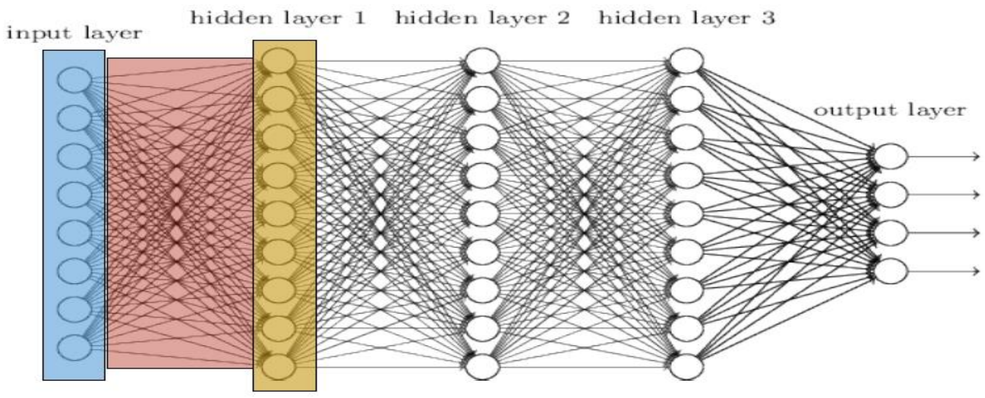
- **How to pass gradients through quantized activation functions?**

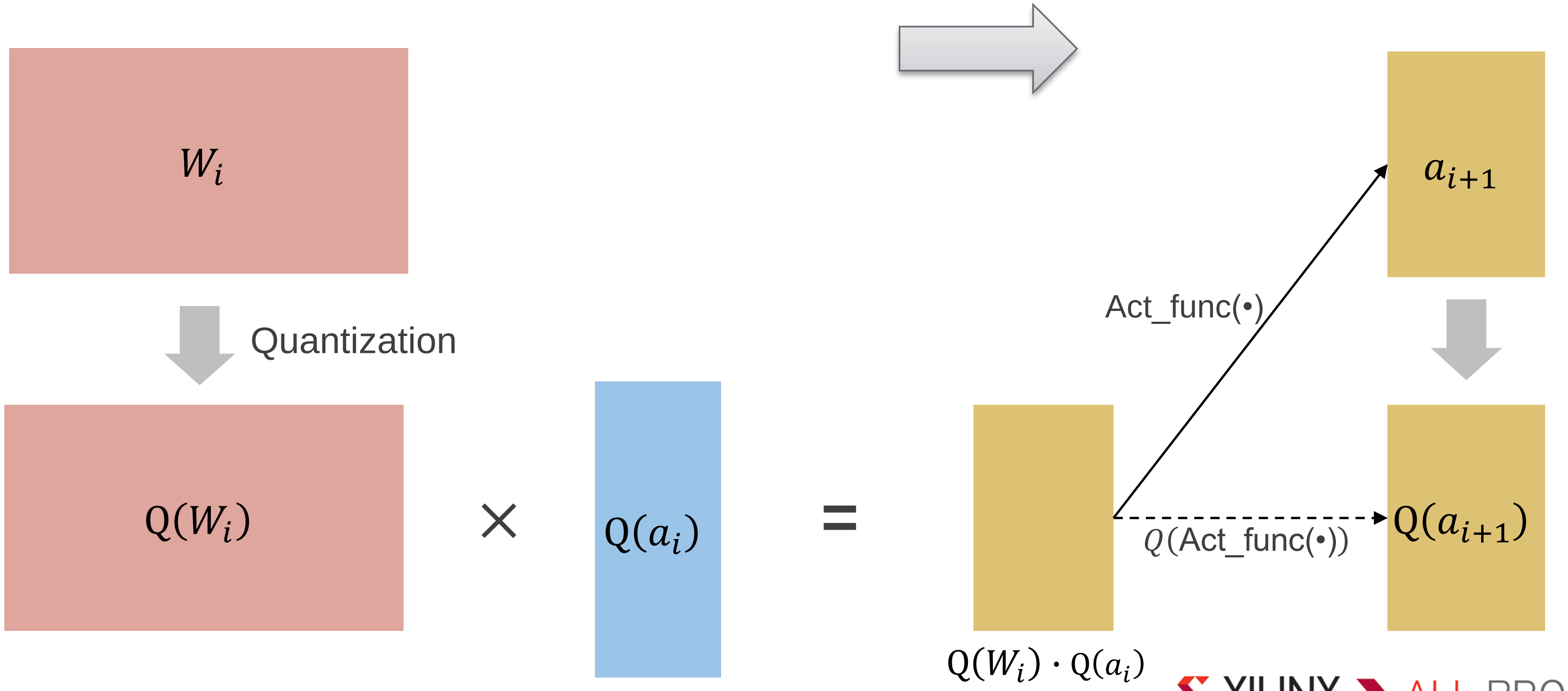


Source: https://arxiv.org/pdf/1511.06488.pdf

Quantization-Aware Forward Path

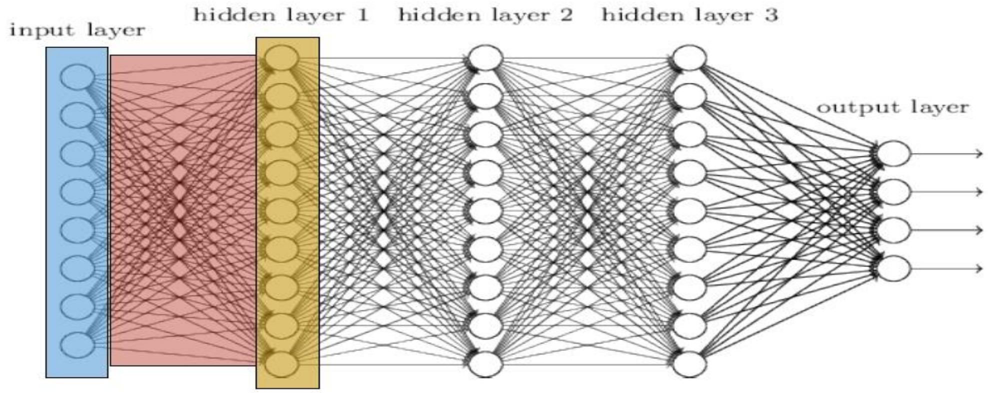
- On-the-fly quantization of weights
- Quantizing activation function





Quantization-Aware Backpropagation

- Non-quantized gradients
- Backpropagation based on quantized weights



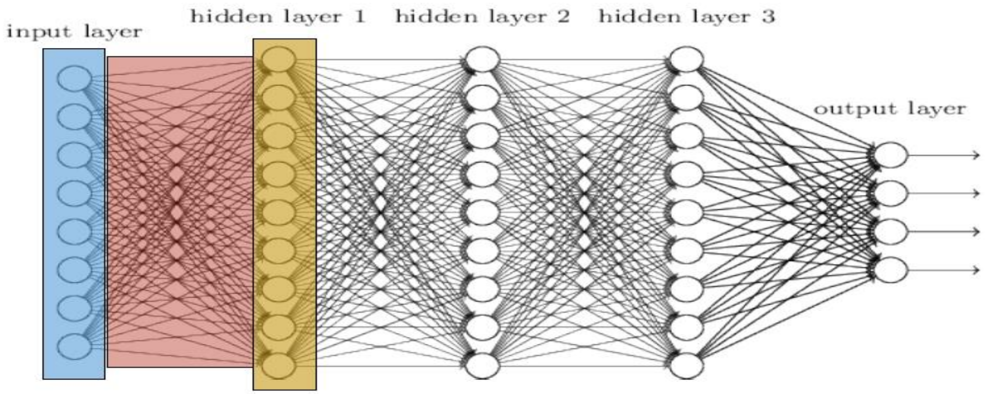
W_i^T

$$Q(W_i^T) \times g_{i+1} = Q(W_i^T) \cdot g_{i+1} \xrightarrow{\text{Act\_func}'(\bullet)} g_i$$

XILINX ALL PROGRAMMABLE.

Quantization-Aware Weight Update

- Update *real* weights



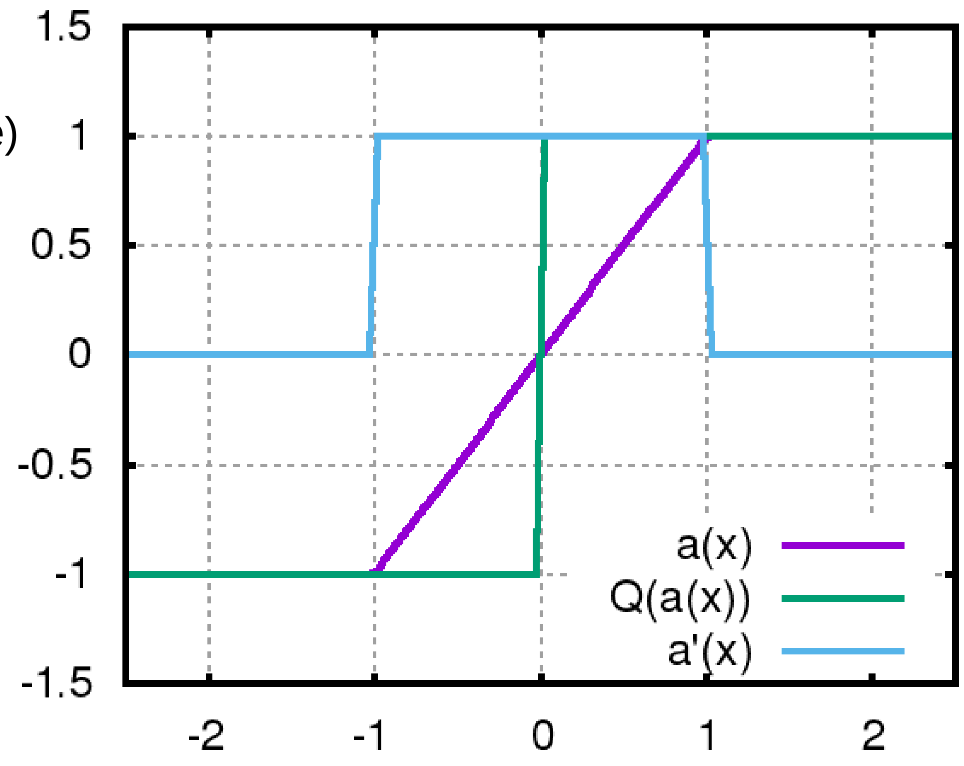
$$W_i{}^{+} := W_i + Q(a_i) \times g_{i+1}{}^{T}$$

XILINX ALL PROGRAMMABLE.

Backpropagation with Quantized Activations

- **Differentiating the sign function:**
  - Choose an activation function, a,
    which tends towards ± 1 as x tends towards $\pm\infty$.
    (The hard hyperbolic tangent function is a common, nice choice)
  - Create a quantized activation function as the composition $a ° Q : x \mapsto Q\big(a(x)\big)$.
  - For the purpose of differentiation, pretend that the quantization function Q had a gradient of 1 everywhere.

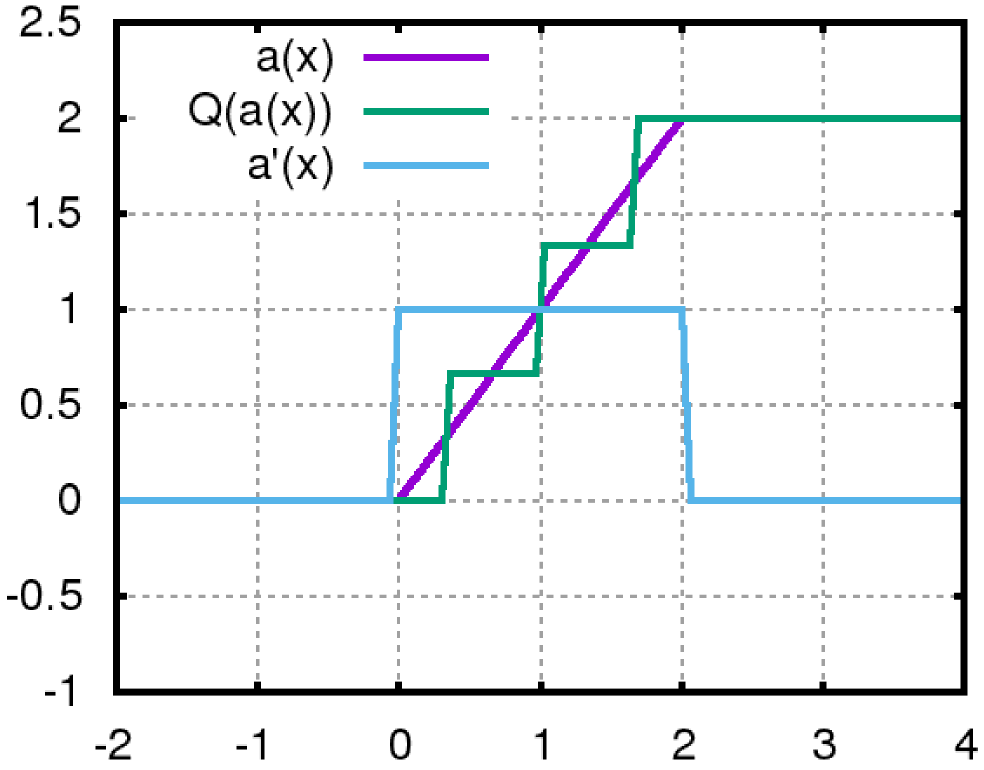
- **Clip gradients outside of range (optional, but recommended).**



XILINX ALL PROGRAMMABLE.

Backpropagation with Quantized Activations

- **Quantizing ReLU**
  - Clip ReLU at the maximum value you want to support.
  - Create a quantized activation function as the composition $a°Q: x \mapsto Q\big(a(x)\big)$.
    - Equal distance quantization over the specified range is a good choice and ensures a local average gradient of 1.
  - For the purpose of differentiation, pretend that the quantization function Q had a gradient of 1 everywhere.



- **Clip gradients outside of range (optional, but recommended).**

Batch Normalization

- **Improves convergence time, and accuracy of RPNNs.**
- **Fixed post-scaling gives full control over output distribution parameters, e.g.:**

$$\boldsymbol{\gamma = 1,\ \beta = 0}\ \text{ for }\ \boldsymbol{\mu = 0,\ \sigma_{\mathcal{B}}^2 = 1}$$

- **For extreme reduced precision, BN is free at inference time.**
- **For higher precisions, shift-based BN can be used.**

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Source: https://arxiv.org/pdf/1502.03167.pdf

QNNs In Lasagne

Frameworks with Reduced Precision Training Support

- **Lasagne (Theano)**
  - Supports binarized weights / activations
  - Extended to support fixed-point data types
- **Tensorpack (TensorFlow)**
  - Supports reduced-precision weights / activations
- **Caffe**
  - C++ framework
  - Supports binarized weights / activations
  - Supports uniform and non-uniform quantization
- **Darknet**
  - C-based NN library
  - Supports binarized weights / activations
- **Torch**
  - Lua based
  - Supports binarized weights / activations
  - Supports shift-based Adam / batch normalization
- **MXNet**
  - Supports binarized weights / activations

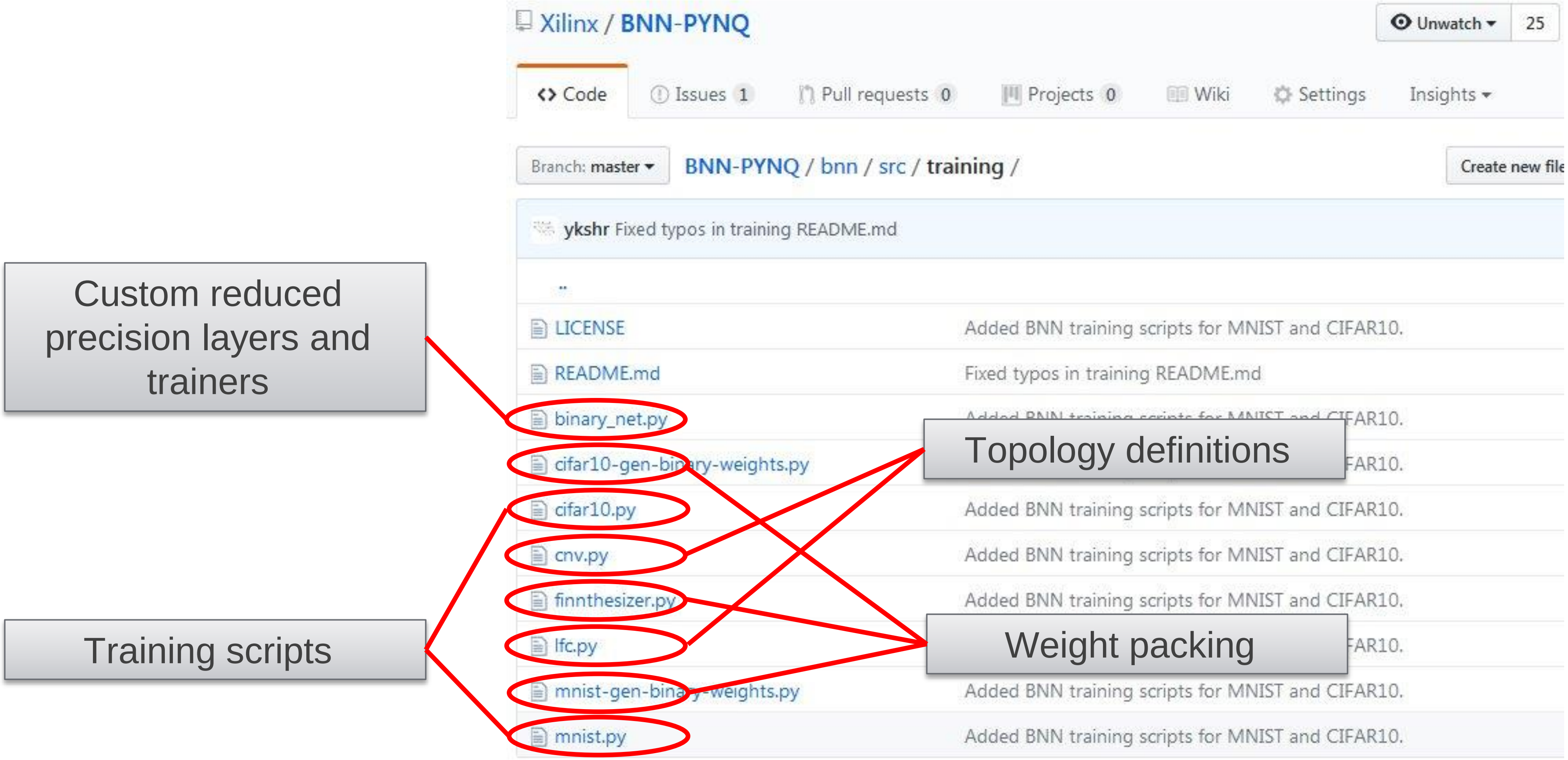
Popularity of reduced precision neural networks growing – support in other frameworks will probably arrive soon!

XILINX ALL PROGRAMMABLE.

Features of Lasagne

- **Python interface**
  - Easy integration with Numpy.
- **Automatic Differentiation**
  - Less code = fewer bugs!
- **CPU / GPU support**
  - Switch between CPU / GPU by simply setting an environment variable.
- **Extreme Flexibility**
  - Can implement any dataflow graph as a neural network.

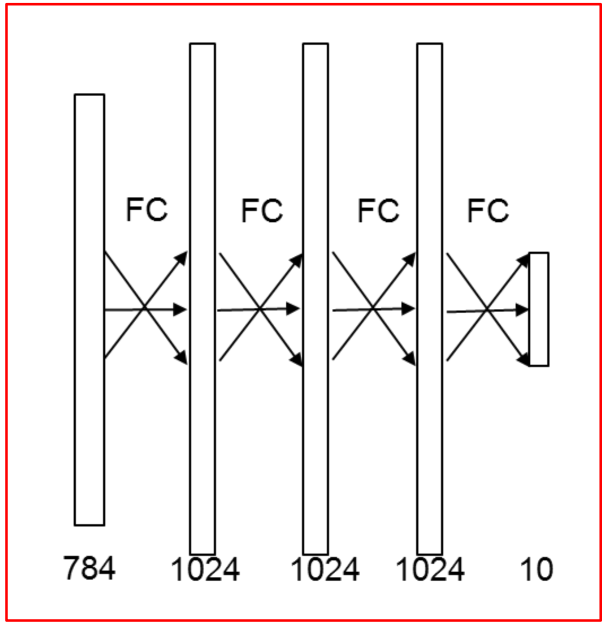
Full Installation Instructions Available on Github



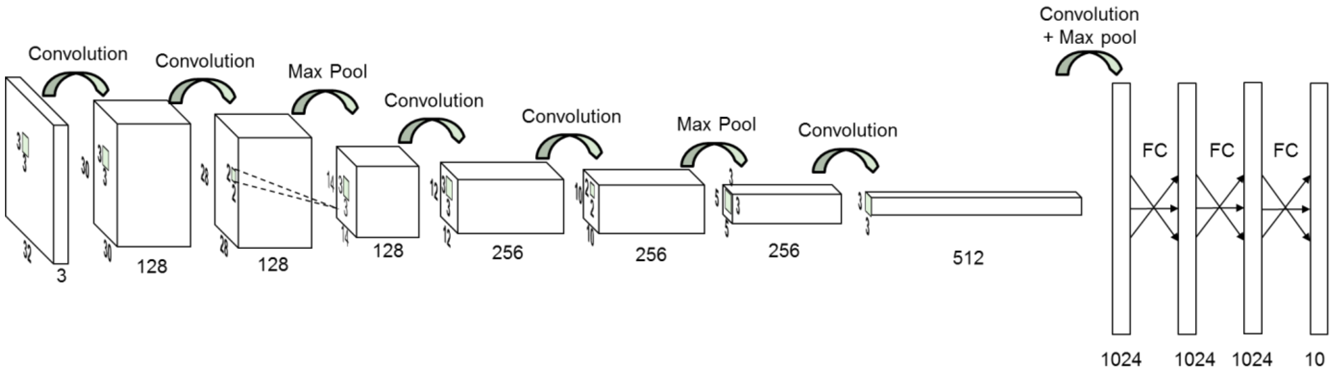
Source: https://github.com/Xilinx/BNN-PYNQ

Test Networks

- **LFC**
  - Input images: 28x28 pixels, binarized images
  - Number of layers: 3 FC layers, 1024 neurons each
  - Compute requirement: 5.8 MOps/Frame



- **CNV (VGG-16 derivative)**
  - Input images: 32x32 pixels, RGB image
  - Number of layers: 2 (3x3) Conv + Max Pool + 2 (3x3) Conv + Max Pool + 2 Convolutional + Max Pool + 3 FC
  - Compute requirement: 1.23 GOps/Frame



XILINX ALL PROGRAMMABLE.

BinaryNet in Lasagne – Training Script (mnist.py)

- ~150 lines of code
- Python library imports
- Setting hyperparameters
- Importing dataset
- Constructing the topology
  → Changes require bitstream update.
- Setting the loss function / network output
- Training the network

XILINX ALL PROGRAMMABLE.

BinaryNet in Lasagne – Importing the Dataset

- **Import sets and separate into *training*, *validation* and *test* sets – these are simply numpy arrays!**
  - Rule of thumb: 60% training, 20% validation, 20% test.
  - Beware of duplicates and data order.
- **Binarize input values (only required for LFC)**
- **Convert labels into a 1D array of class indices**
- **1-hot encode the class labels**
- **Modify result to match loss function**

```
print('Loading MNIST dataset...')

train_set = MNIST(which_set= 'train', start=0, stop = 50000, center = False)
valid_set = MNIST(which_set= 'train', start=50000, stop = 60000, center = False)
test_set = MNIST(which_set= 'test', center = False)

# bc01 format
# Inputs in the range [-1,+1]
# print("Inputs in the range [-1,+1]")
train_set.X = 2* train_set.X.reshape(-1, 1, 28, 28) - 1.
valid_set.X = 2* valid_set.X.reshape(-1, 1, 28, 28) - 1.
test_set.X = 2* test_set.X.reshape(-1, 1, 28, 28) - 1.

# Binarise the inputs.
train_set.X = np.where(train_set.X < 0, -1, 1).astype(theano.config.floatX)
valid_set.X = np.where(valid_set.X < 0, -1, 1).astype(theano.config.floatX)
test_set.X = np.where(test_set.X < 0, -1, 1).astype(theano.config.floatX)

# flatten targets
train_set.y = np.hstack(train_set.y)
valid_set.y = np.hstack(valid_set.y)
test_set.y = np.hstack(test_set.y)

# Onehot the targets
train_set.y = np.float32(np.eye(10)[train_set.y])
valid_set.y = np.float32(np.eye(10)[valid_set.y])
test_set.y = np.float32(np.eye(10)[test_set.y])

# for hinge loss
train_set.y = 2* train_set.y - 1.
valid_set.y = 2* valid_set.y - 1.
test_set.y = 2* test_set.y - 1.
```

BinaryNet in Lasagne – Constructing The Topology

- ~60 lines of code
- Configure global parameters
- Construct the topology
- **Modifying the code here will mean the weights may not work with the overlay!!**

```python
import lasagne
import binary_net

def genLfc(input, num_outputs, learning_parameters):
    # A function to generate the lfc network topology which matches the overlay for the Pynq board.
    # WARNING: If you change this file, it's likely the resultant weights will not fit on the Pynq overlay.
    if num_outputs < 1 or num_outputs > 64:
        error("num_outputs should be in the range of 1 to 64.")
    stochastic = False
    binary = True
    H = 1
    num_units = 1024
    n_hidden_layers = 3
    activation = binary_net.binary_tanh_unit
    W_LR_scale = learning_parameters.W_LR_scale
    epsilon = learning_parameters.epsilon
    alpha = learning_parameters.alpha
    dropout_in = learning_parameters.dropout_in
    dropout_hidden = learning_parameters.dropout_hidden

    mlp = lasagne.layers.InputLayer(
            shape=(None, 1, 28, 28),
            input_var=input)

    mlp = lasagne.layers.DropoutLayer(
            mlp,
            p=dropout_in)

    for k in range(n_hidden_layers):

        mlp = binary_net.DenseLayer(
                mlp,
                binary=binary,
                stochastic=stochastic,
                H=H,
                W_LR_scale=W_LR_scale,
                nonlinearity=lasagne.nonlinearities.identity,
                num_units=num_units)

        mlp = lasagne.layers.BatchNormLayer(
                mlp,
                epsilon=epsilon,
                alpha=alpha)

        mlp = lasagne.layers.NonlinearityLayer(
                mlp,
                nonlinearity=activation)

        mlp = lasagne.layers.DropoutLayer(
                mlp,
                p=dropout_hidden)

    mlp = binary_net.DenseLayer(
                mlp,
                binary=binary,
                stochastic=stochastic,
                H=H,
                W_LR_scale=W_LR_scale,
                nonlinearity=lasagne.nonlinearities.identity,
                num_units=num_outputs)

    mlp = lasagne.layers.BatchNormLayer(
            mlp,
            epsilon=epsilon,
            alpha=alpha)
    return mlp
```


XILINX ALL PROGRAMMABLE.

BinaryNet in Lasagne – Defining Layers

- **Basic layer pattern: Dense (or Conv2D) -> BatchNorm -> Activation -> Dropout (optional)**
- **Instantiate a layer with binary weights**
- **Binarize activations**
- **Modifying the code here will mean the weights may not work with the overlay!**

```
# k = 3, binary=true, stochastic=false, H=1, num_units=1024

for k in range(n_hidden_layers):

        mlp = binary_net.DenseLayer(
                mlp,
                binary=binary,
                stochastic=stochastic,
                H=H,
                W_LR_scale=W_LR_scale,
                nonlinearity=lasagne.nonlinearities.identity,
                num_units=num_units)

        mlp = lasagne.layers.BatchNormLayer(
                mlp,
                epsilon=epsilon,
                alpha=alpha)

        mlp = lasagne.layers.NonlinearityLayer(
                mlp,
                nonlinearity=binary_net.binary_tanh_unit)

        mlp = lasagne.layers.DropoutLayer(
                mlp,
                p=dropout_hidden)
```

Accuracy of Binary and Almost Binary Networks

Published Results

Dataset	FP32	BNN	Source
MNIST	99%	99%	[1]
SVHN	98%	97%	[1]
CIFAR-10	92%	90%	[1]
ImageNet (AlexNet arch)	80% top-5	69% top-5	[2]
ImageNet (ResNet-18 arch)	89% top-5	73% top-5	[2]
ImageNet (GoogleNet arch)	90% top-5	86% top-5	[2]
ImageNet (DoReFaNet)	56% top-1	50% top-1	[4] 2b activations

- Similar accuracy on small networks and promising results for larger networks

[1] Courbariaux, Matthieu, and Yoshua Bengio. "BinaryNet: Training deep neural networks with weights and activations constrained to+ 1 or-1." arXiv preprint arXiv:1602.02830 (2016).
[2] Rastegari, Mohammad, et al. "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks." arXiv preprint arXiv:1603.05279 (2016).
[3] Xundong Wu: High Performance Binarized Neural Networks trained on the ImageNet Classification Task" arXiv:1604.03058
[4] S. Zhou, z.Ni, X. Zhou, H.Wen, Y.Wu, Y. Zou: "DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients", http://arxiv.org/abs/1606.06160#

Binarized Neural Networks – Improving Accuracy

- **Quantizing networks from floating point to binary will introduce a drop in accuracy.**
- **Sometimes conversion of an existing network will "just work".**
- **Often, hyperparameters or even the network topology will have to change to get good accuracy results.**

- **Common methods to improve accuracy:**
  - Add batch normalization before activations.
  - Reduce learning rate.
  - Increase number of epochs.
  - Increase the size of the network:
    - Larger layers,
    - Deeper network (more layers).

Summary

- Combining quantized neural networks & FPGAs allows opportunities to create extreme high-throughput, low-power neural networks.

- There is some drop in accuracy compared to floating point accuracy.
  This is typically compensated by re-training and increasing the size of the network.

- Pynq + Lasagne – great platforms to get started training and implementing your own high-performance neural networks.

Hands-On Opportunities

- **GPU support for training helps a lot, AWS EC2 might help out.**
- **Checkout open-source QNN examples with trained models and Jupyter notebooks for Pynq-Z1 at http://www.pynq.io/community.html:**
  - Xilinx/BNN-PYNQ - LFC, CNV: CIFAR10, MNIST, Road Signs, …
  - Xilinx/QNN-MO-PYNQ - TinierYolo, DorefaNet: Object Detection, ImageNet Classification
  - tukl-msd/LSTM-PYNQ - LSTM: OCR for Fraktur text
- **Expect the QNN story to unfold for more platforms:**
  - Support for more boards.
  - AWS F1 solution.
- **See the XILINX booth!**

Thank You.