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

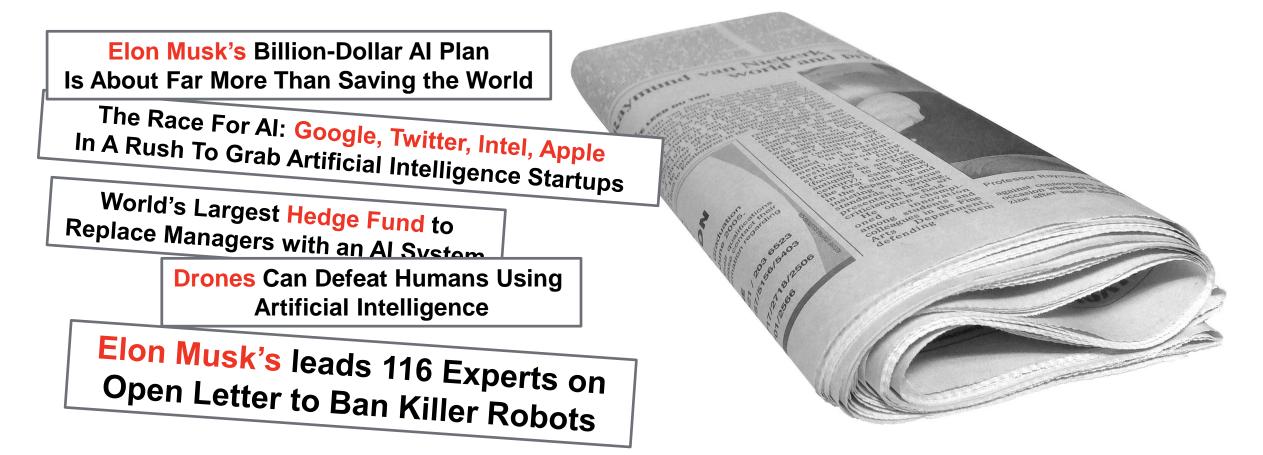




EXILINX > ALL PROGRAMMABLE.

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

(Dec 2016)

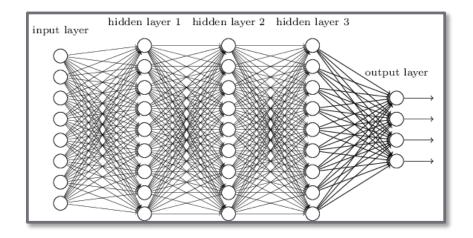


Demonstrated to work well for numerous use cases

Neural Networks

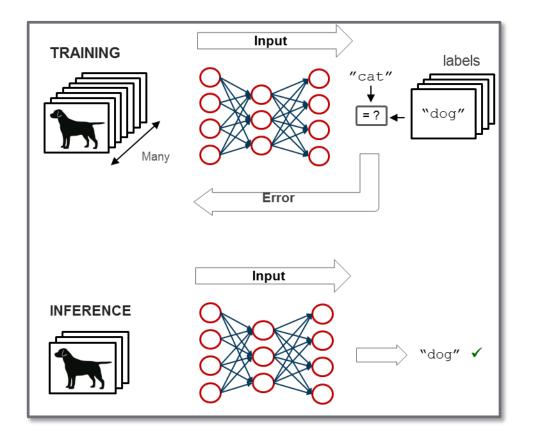
> NNs are the predominant AI algorithm

- 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

Neural Networks: Training vs Inference



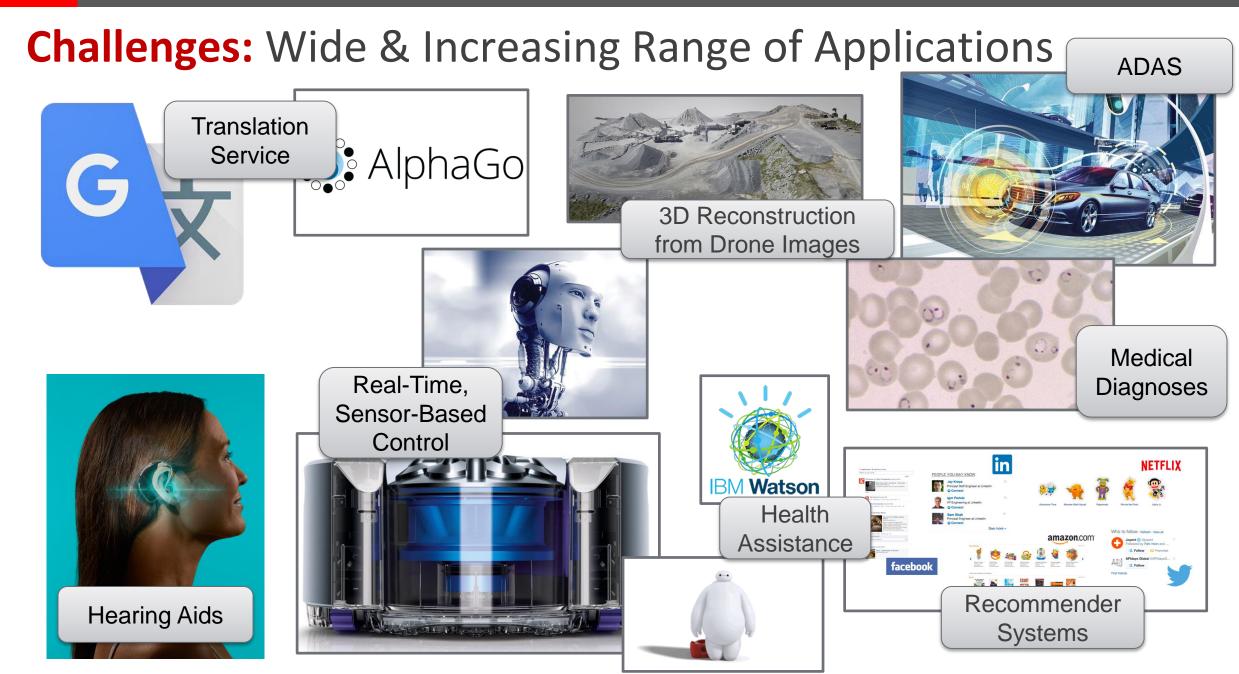
Training

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.

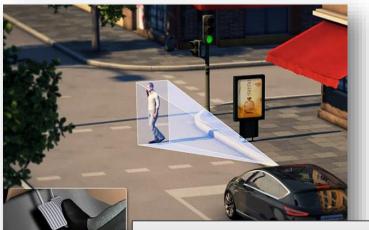
Challenges: Different Figures of Merits

1

STOP

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





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



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

••••• AT&T LTE	6:45 PM		◀ ∦ 7	4% 💷 🕨			
English	₊→	Sp	anish				
ENGLISH				×			
another breakt how will Xilinx of	it seems that every single day there is news of another breakthrough in artificial intelligence. how will Xilinx capitalize on this rapidly growing market?						
٥	Ŷ		2	6			
SPANISH				☆			
Parece que cada día hay noticias de otro avance en la inteligencia artificial. ¿Cómo se capitalizará Xilinx en este mercado en rápido crecimiento?							
capitalizará Xil							
capitalizará Xil							
capitalizará Xil		ado	en rápi	ido			
capitalizará Xil		ado	en rápi	ido			
capitalizará Xil		ado	en rápi	ido			
capitalizará Xil		ado	en rápi	ido			



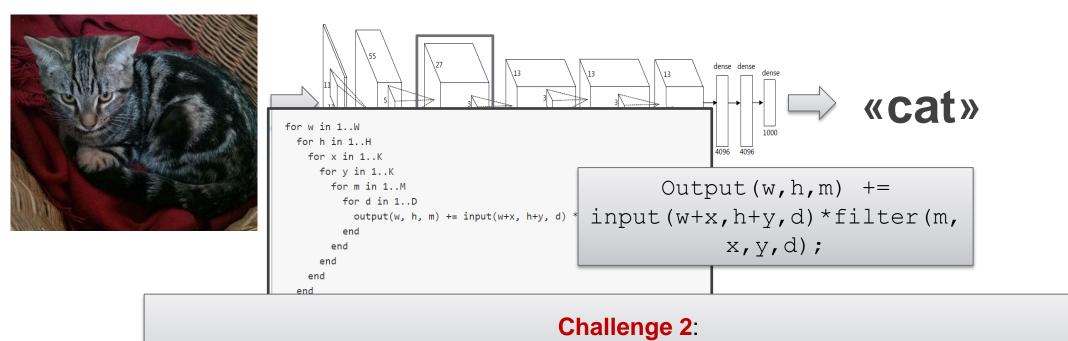
Embedded Systems: heavily power constrained Data Centers: OPEX = f(energy)

EXILINX > ALL PROGRAMMABLE.

Challenges: Highly Compute and Memory Intensive

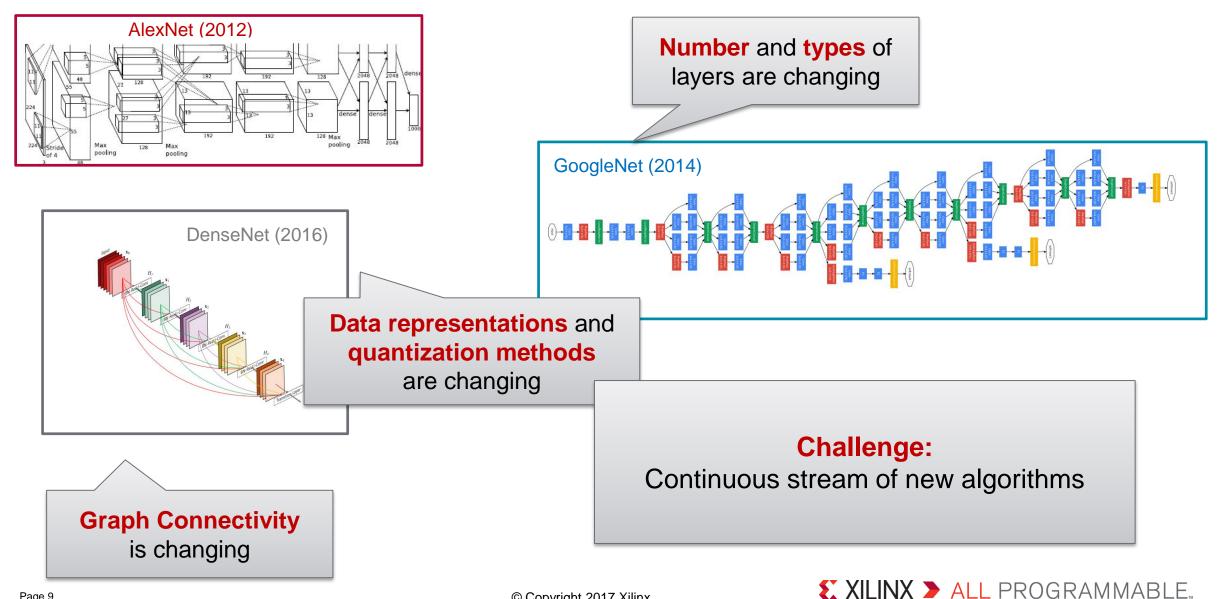
> The predominant CNN computation is linear algebra

- 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

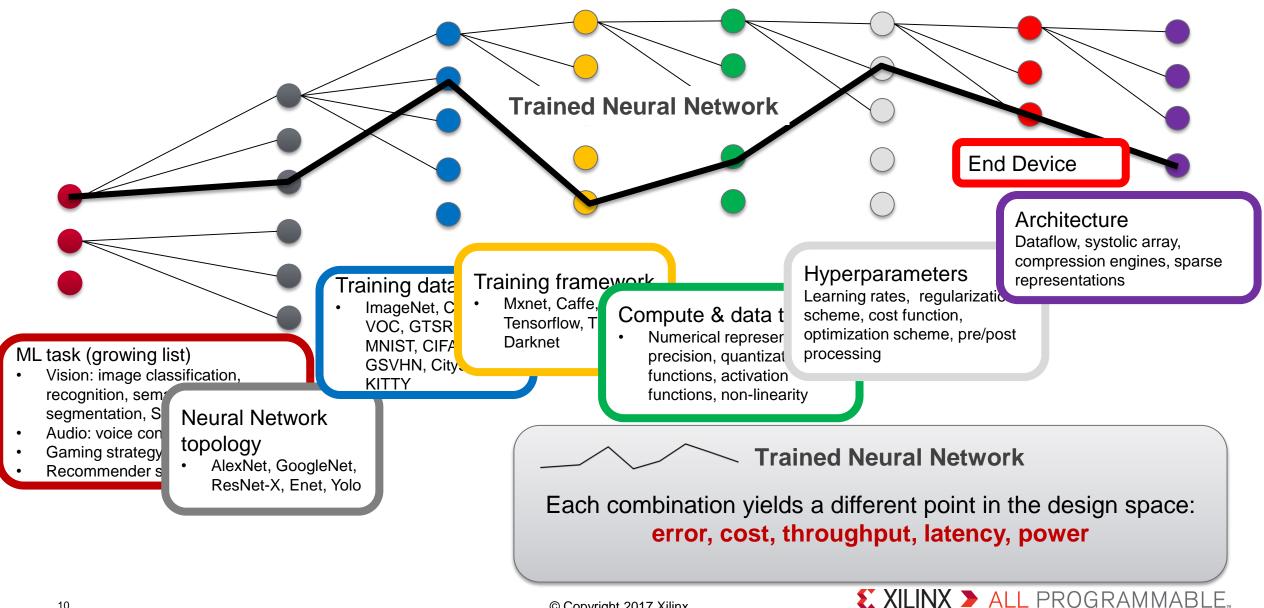


billions of multiply-accumulate ops & tens of megabytes of parameter data

Challenges: Neural Networks Will Continue to Change

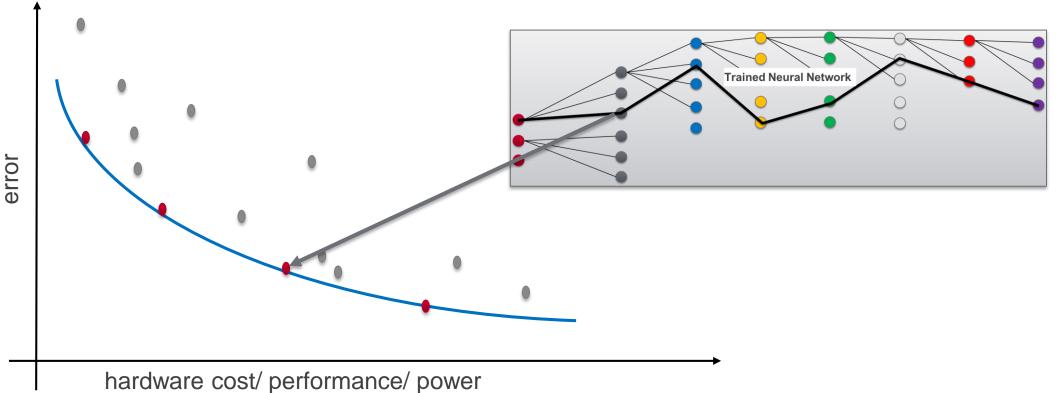


Challenge: Multidimensional Design Space

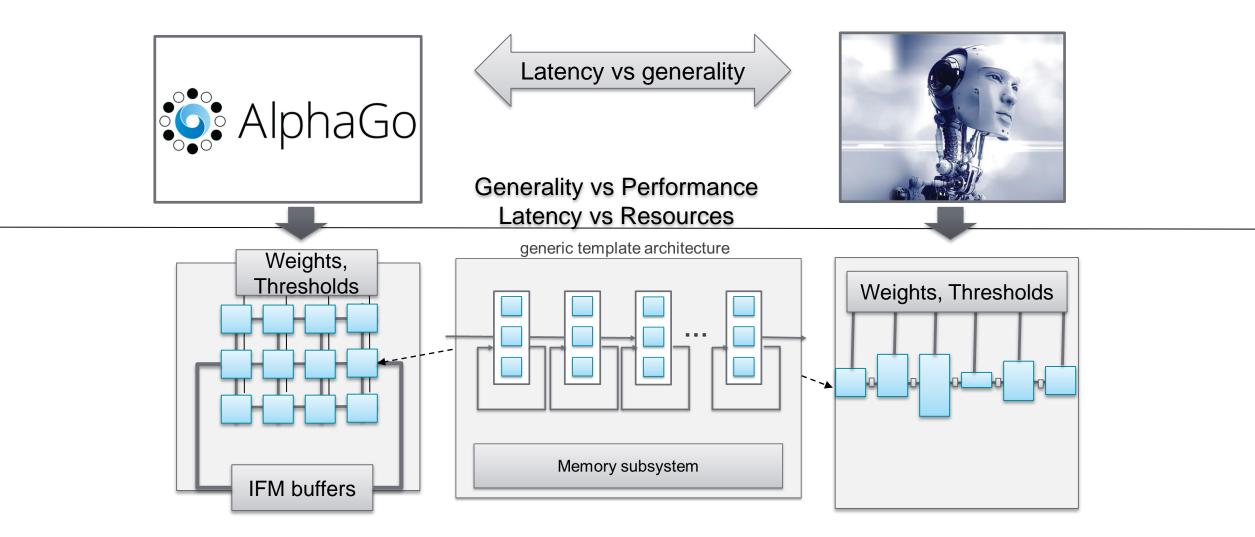


Opportunity: Customized Neural Networks

- 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



Opportunity: Customized ML Processor Datapath



EXILINX > 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
		% (fully parallelizable) 250 % (fully parallelizable) 30			amazon webservices	

Quantizing and Fixed Point saves Power

	_ Relative Energy Cost						
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						
		1 10	0	100	1000	10000	

Dalativa Energy Cost

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

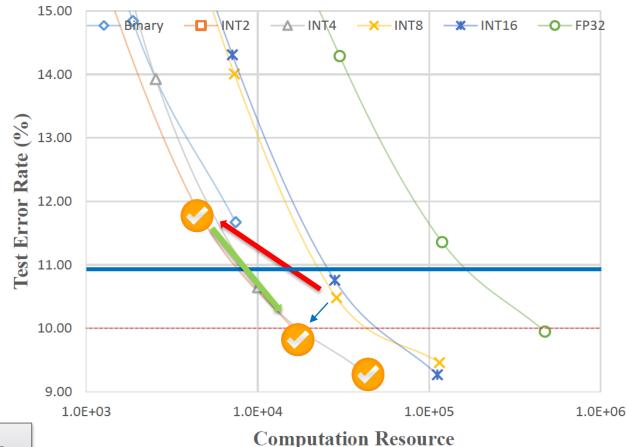
🗶 XILINX 🕨 ALL PROGRAMMABLE.

Do we loose Accuracy?

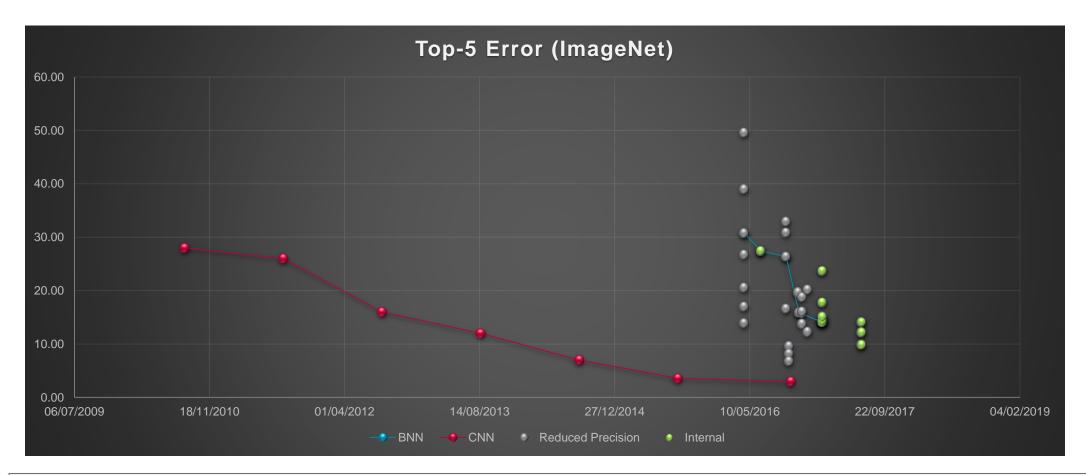
Compensating Quantization with Network Complexity

- 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 <u>https://arxiv.org/pdf/1709.01134.pdf</u>



Accuracy of Quantized Neural Networks (QNNs) Improving Published Results for FP CNNs, QNNs and binarized NNs (BNNs)

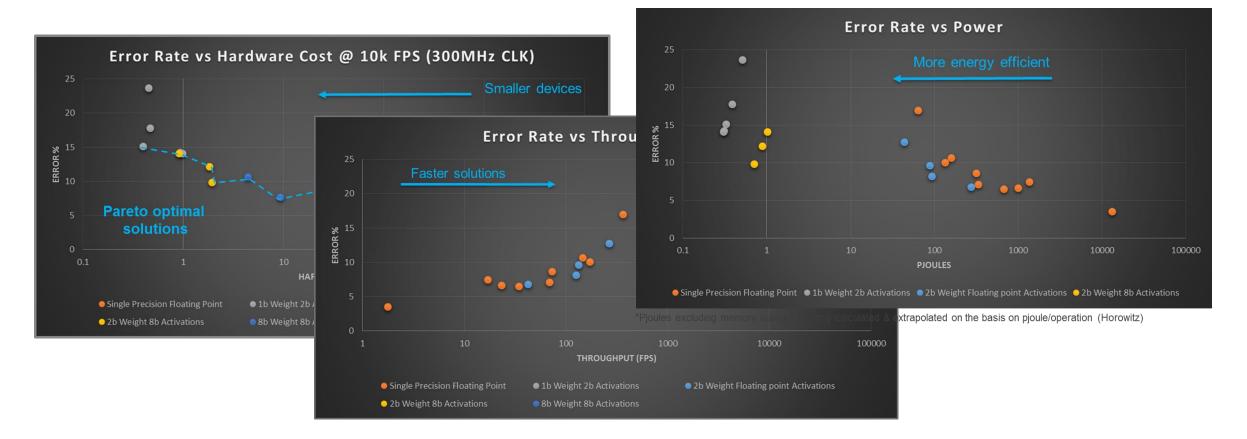


 Accuracy results are improving rapidly through for example new training techniques, topological changes and other methods

XILINX > ALL PROGRAMMABLE.

Summary

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



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

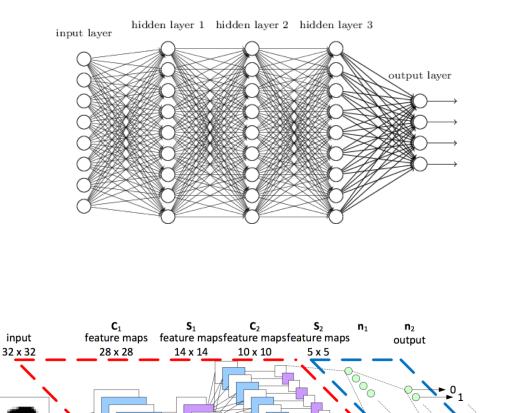
> Training Binary Neural Networks in Lasagne

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



5x5

convolution

feature extraction

EXILINX > ALL PROGRAMMABLE.

2x2

subsampling

fully

connected

classification

5x5

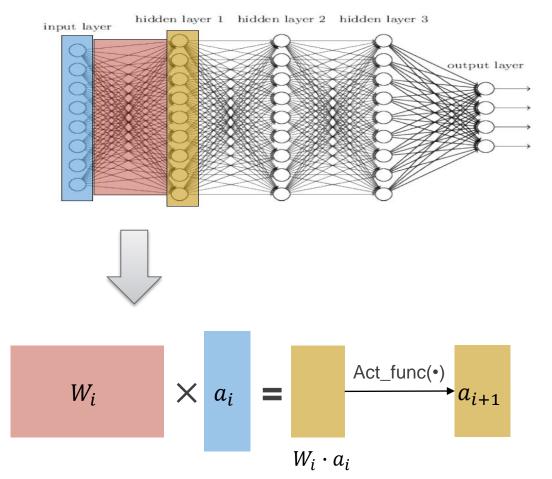
convolution

2x2

subsampling

Neural Networks – Fully Connected Layer

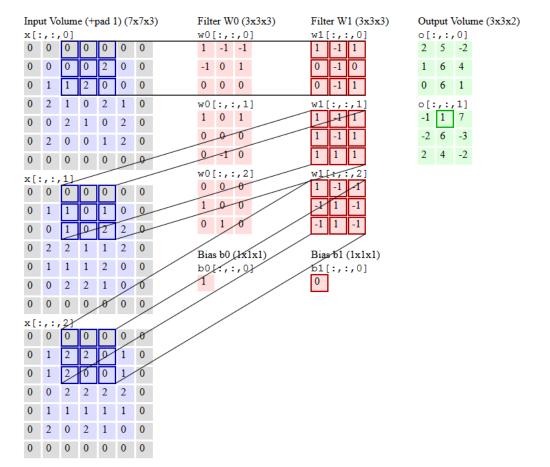
- Also known as: inner product layer or dense layer.
- Each neuron is connected to every neuron of the previous layer.
- A weight is associated with each "synapse".
- Can be written as a matrix-vector product with an element-wise nonlinearity applied afterwards.



XILINX ➤ ALL PROGRAMMABLE.

Neural Networks – Convolutional Layer

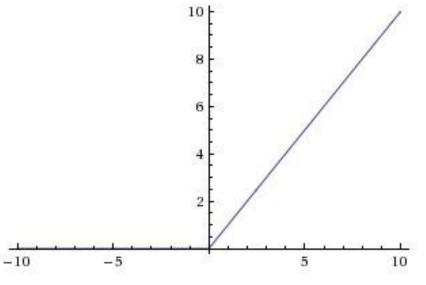
- Each neuron applies a convolution to all images in the previous layer.
- Weights represent the filters used for convolutions.
- Can be *lowered* to a matrix-matrix multiply.
- Non-linear activation applied to each output pixel.



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

Neural Networks – Activation Functions

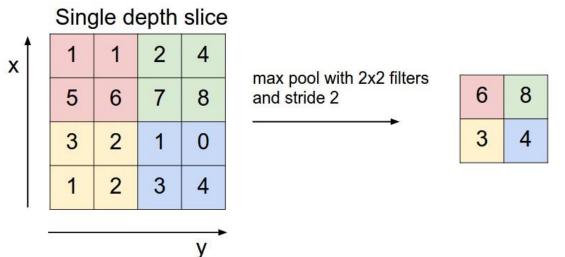
- Most popular: the rectified linear unit (ReLU)
- Other common ones include: tanh, leaky ReLU.
- For binarized neural networks, the step function is often used.



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.

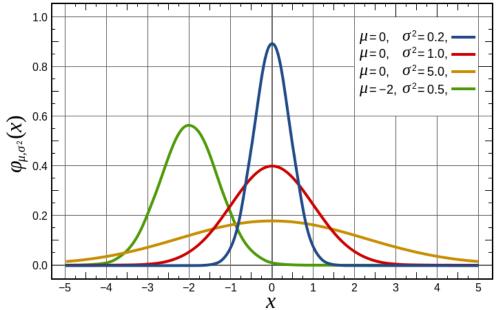


Source: http://cs231n.github.io/convolutional-networks/

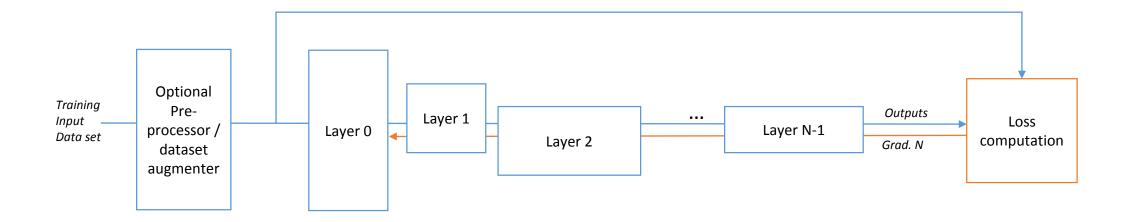
XILINX > ALL PROGRAMMABLE.

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.



Training Neural Networks - Backpropagation

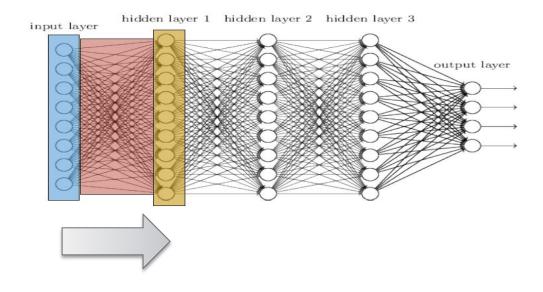


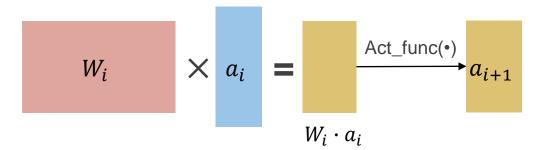
> Purpose: calculate the gradients associated with each weight within a network.

- > 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:

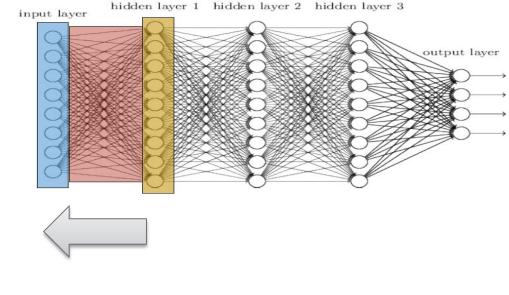


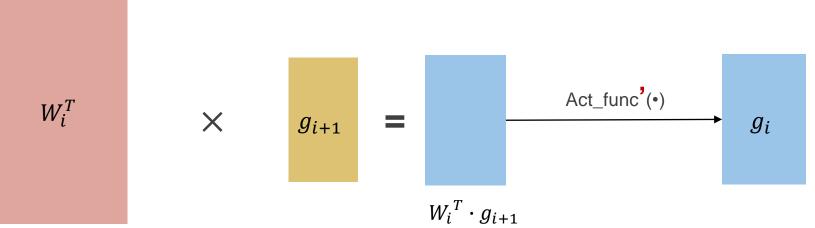


XILINX ➤ ALL PROGRAMMABLE.

Backpropagation: Backward Path

Pass gradients back through network:

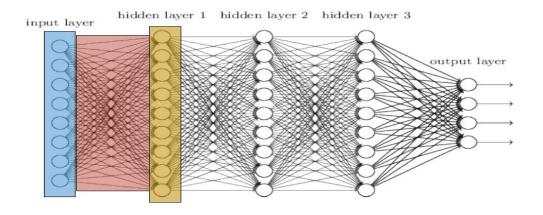


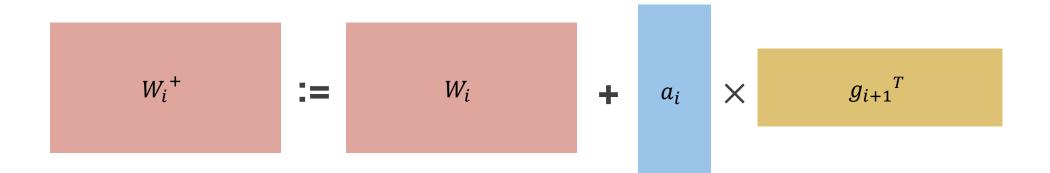




Backpropagation: Weight Update

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





Quantized Neural Networks



Data Representations & Reduced Precision

Floating Point

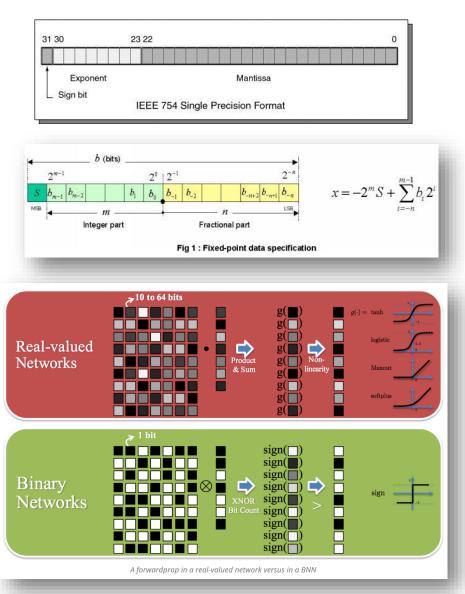
- Usually 32-bits
- Large range, high precision

Fixed Point

- Fixed range
- Simpler hardware

>Binarized

- Multiply-accumulate becomes XNOR-popcount
- 32x memory reduction
- Extreme performance possible on FPGAs



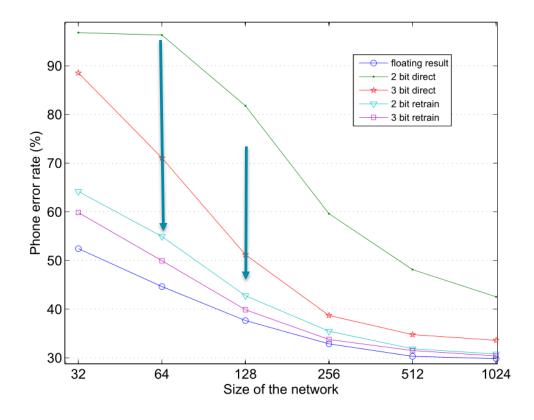
XILINX > ALL PROGRAMMABLE.

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?



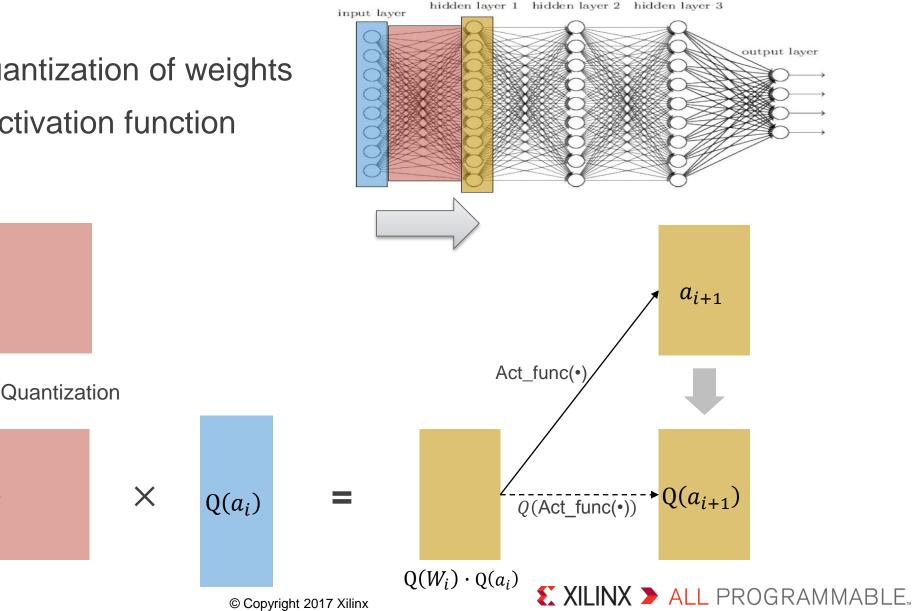
Source: https://arxiv.org/pdf/1511.06488.pdf

Quantization-Aware Forward Path

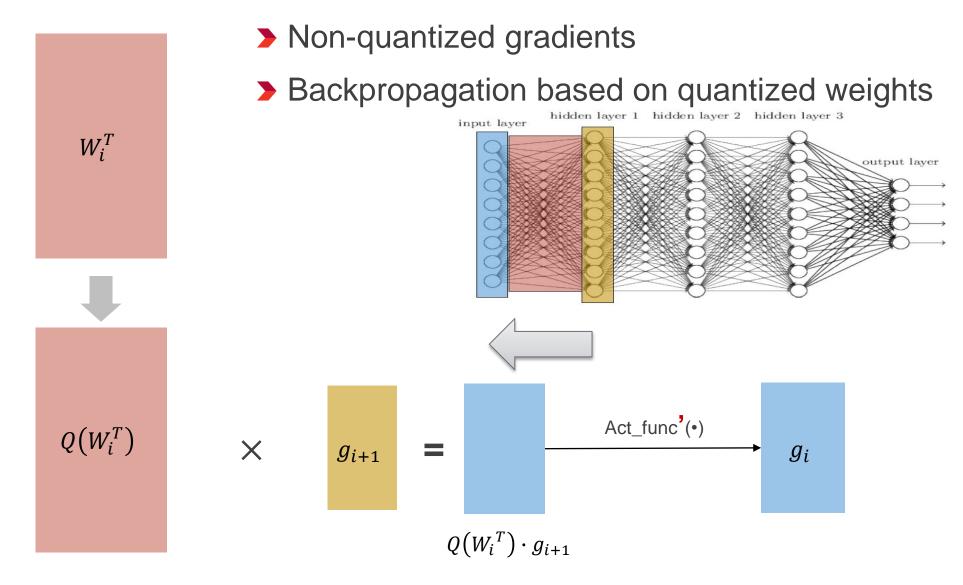
- > On-the-fly quantization of weights
- Quantizing activation function

 W_i

 $Q(W_i)$



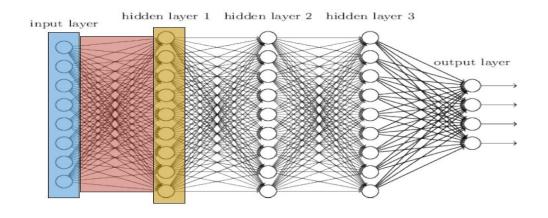
Quantization-Aware Backpropagation

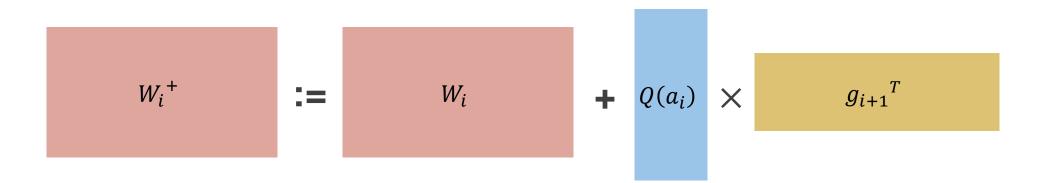


XILINX > ALL PROGRAMMABLE.

Quantization-Aware Weight Update

> Update *real* weights

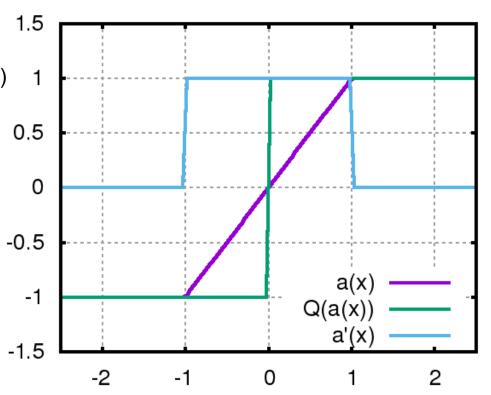




Backpropagation with Quantized Activations

> Differentiating the sign function:

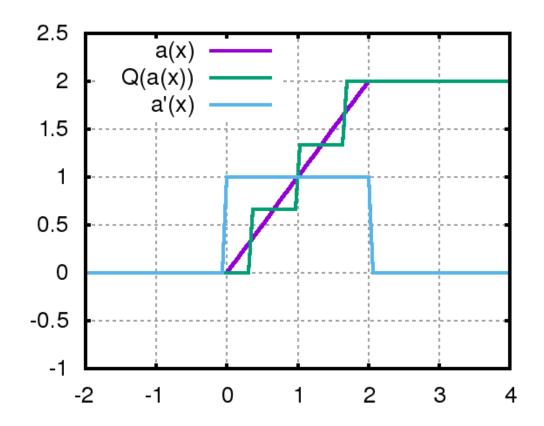
- Choose an activation function, *a*,
 which tends towards ±1 as x tends towards ±∞.
 (The hard hyperbolic tangent function is a common, nice choice)
- Create a quantized activation function as the composition $a^{\circ}Q: x \mapsto Q(a(x))$.
- For the purpose of differentiation, pretend that the quantization function Q had a gradient of 1 everywhere.
- Clip gradients outside of range (optional, but recommended).



Backpropagation with Quantized Activations

Quantizing ReLU

- Clip ReLU at the maximum value you want to support.
- Create a quantized activation function as the composition $a^{\circ}Q: x \mapsto Q(a(x))$.
 - 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.:

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

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

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!

Features of Lasagne

> Python interface

- Easy integration with Numpy.

> Automatic Differentiation

– Less code = fewer bugs!

> CPU / GPU support

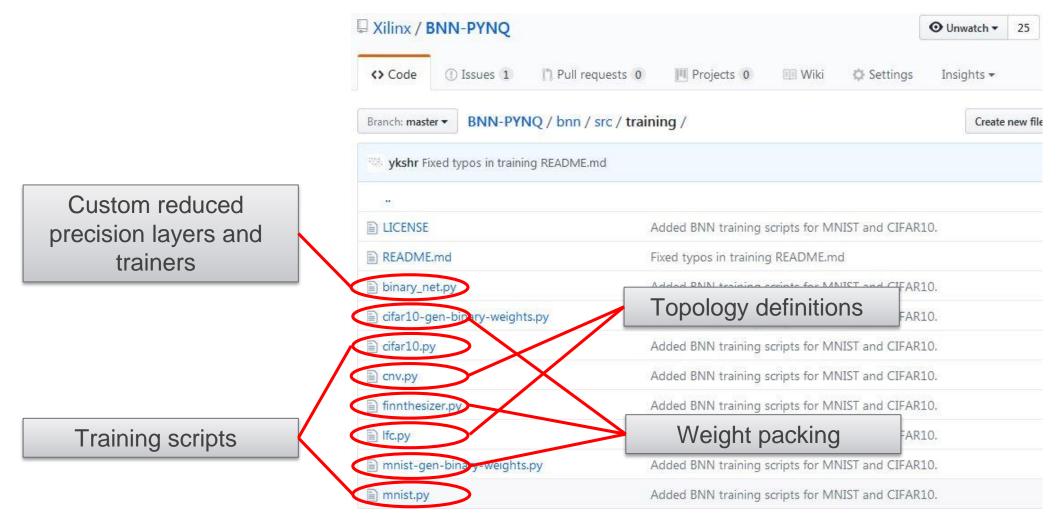
- Switch between CPU / GPU by simply setting an environment variable.

> Extreme Flexibility

- Can implement any dataflow graph as a neural network.



Full Installation Instructions Available on Github



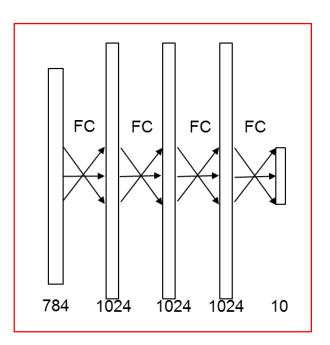
Source: https://github.com/Xilinx/BNN-PYNQ

🐮 XILINX ➤ ALL PROGRAMMABLE.

Test Networks

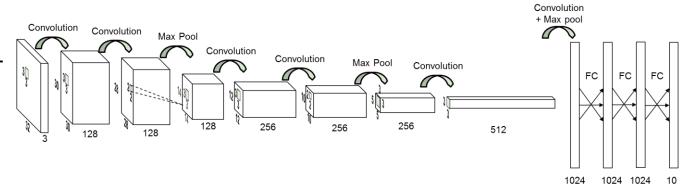
> LFC

- 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



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





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)</pre>
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!!

import lasagne import binary net

<pre>def genificLingst, neglogits, larring_meresters): # A method to generate the frametry togety with active the overlay for the Pysg based. # fram.output < 1 or neglogity is discussed to the control togethy is discussed to the two pysg based. # fram.output < 1 or neglogity is discussed togethy is discussed togethy is discussed togethy. # fram.output < 1 or neglogity is discussed togethy is discussed togethy is discussed togethy. # fram.output < 1 or neglogity is discussed togethy is discussed togethy. # fram.output < 1 or neglogity is discussed togethy is discussed togethy. # fram.output < 1 or neglogity is discussed togethy is discussed togethy is discussed togethy. # fram.output < 1 or neglogity is discussed togethy is discussed togethy is discussed togethy. # fram.output < 1 or neglogity is discussed togethy is discussed togethy is discussed togethy. # fram.output < 1 or neglogity is discussed togethy is discussed togethy is discussed togethy. # fram.output < 1 or neglogity is discussed togethy is discused togethy i</pre>		
<pre># uklnik: if you change this file, it's likely 'the resultant weights will not fit on the prog overlay. if adotted is a file resultant is file theory if resultant is file, it's likely 'the resultant weights will not fit on the prog overlay. if adotted is a file resultant is file theory if resultant is file, it's likely 'the resultant weights will not fit on the prog overlay. if adotted is a low in group of it to 64.7) theorem is a low in group of it to 64.7) the starting presenters.upped dropoot.if is a low in group of it to 64.7) the starting presenters.upped dropoot.if is a low in group of it's file of it's is a low in group of it's file of it's is a low in group of it's file of it's is a low in group of it's file of it's is a low in group of it's file of it's is a low in group of it's file of it's is a low in group of it's file of it's is a low in group of it's is a low</pre>	def genLfc	:(input, num_outputs, learning_parameters):
<pre>if num_outputs 1 in num_outputs > 44 record measures inter = inter = inte</pre>		
<pre>if num_outputs 1 in num_outputs > 44 record measures inter = inter = inte</pre>	# WARN	JING: If you change this file, it's likely the resultant weights will not fit on the Pyng overlay.
<pre>tothatic = Files binary = Tree h = 1 tothatic = Files binary = 1024 extivution = binary_tethbinary_tend unit kit = tearing_parameters.lpbs dopout_iniden = learning_parameters.depout_liden dopout_iniden = learning_parameters.depout_liden dopout_iniden_lists.depout_liden dopout_iniden_lists.depout_liden dopout_iniden_lists.depout_liden dopout_iniden = learning_parameters.depout_liden dopout_iniden = learning_parameters.depout_liden dopout_iniden = learning_parameters.depout_liden dopout_iniden dopout_iniden dopout_liden = learning_iniden = learning_parameters.depout_liden dopout_iniden dopout_liden = learning_iniden dopout_iniden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_liden dopout_lid</pre>		
<pre>kiary = Tree # = 1 mstit: _ yz4i stivit: = biary_stityz4i divit: = biary_stityz4i divit: = biary_stityz4i divit: = biary_stityz4i divit: = biary_stityz4i divit: = biary_stityz4i ndp = lissge.lipwr.lipuliyer(</pre>	er	ror("num_outputs should be in the range of 1 to 64.")
<pre>w = 1 ms_wis : 1024 n_lides[syst=s]; v_L[s_cscl= lasting_prametrs.u[s_cscle epsilon = lasting_prametrs.u[s_cscle epsi</pre>	stocha	istic = False
<pre>numunis = 1924 n_hides_lever.a3 ativetion = hisroy_stillservers.lb, texte upplion = learning_parameters.jb, texte upplion = learning_parameters.jb, texte upplion = learning_parameters.depoit_in depoit_textering_parameters.depoit_in depoit_textering_parameters.depoit_in depoit_textering_parameters.depoit_in depoit_textering_parameters.depoit_in depoit_textering_parameters.depoit_in depoit_textering_parameters.depoit_in depoit_textering_parameters.depoit_in depoit_textering_parameters.depoit_in depoit_textering_parameters.depoit_in depoit_textering_parameters.depoit_in depoit_textering_parameters.depoit_in depoit_textering_parameters.depoit_in depoint_textering_parameters.depoit_in depoint_textering_parameters.depoit_in depoint_textering_parameters.depoit_in depoint_textering_parameters.depoint_in depoint_textering_parameters.depoint_in depoint_textering_parameters.depoint_in depoint_textering_parameters.depoint_in depoint_textering_parameters.depoint_in depoint_textering_parameters.depoint_in depoint_textering_parameters.depoint_in depoint_textering_parameters.depoint_in depoint_textering_parameters.depoint_in depoint_textering_parameters.depoint_in depoint_textering_parameters.depoint_textering_parameters.depoint_textering_parameters.depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_depoint_dep</pre>	binary	
<pre>n_iden_layes *3 attivition * kinsy.tab.uit u[u]scale * Lenning_parameters u[u]scale other *1ming_parameters.apha other *1ming_p</pre>	H = 1	
<pre>stiution = timery_met.Binery_tabu_unit u_U_E_scale = tearning_parameters.u_U_E_scale epsilen = learning_parameters.dopout_lin dopout_lidden = learning_parameters.dopout_lin dopout_lidden = learning_parameters.dopout_lin dopout_lidden = learning_parameters.dopout_lin dopout_lidden = learning_parameters.dopout_lin dopout_lin) nut=_unseque.layers.Dopoutieger(mlp, totoAsticstochastic, mlp = linery_net.Bestayer(mlp, vitualization) nut=_unseque.layers.dopout_layer(mlp, stochasticstochastic, mlp = linery_net.Bestayer(mlp, vitualization) nut=_unseque.layers.dominerrites.identity, monlinerity_ustower(mlp, stochasticstochastic, mlp = linery_net.Bestayer(mlp, vitualization) nut=_unseque.layers.dominerrites.identity, monlinerity_ustower(mlp, stochasticstochastic, mlp = linery_net.Bestayer(mlp, vitualization) nut=_unseque.layers.dominerrites.identity, monlinerity_ustower(mlp, stochasticstochastic, mlp = linery_net.Bestayer(mlp, vitualization) nut=_unseque.layers.dominerrites.identity, monlinerity_ustower(mlp, stochasticstochastic, mlp = linery_net.Bestayer(mlp, upsice_bostayer(mlp, monlinerity_ustower(mlp, monlinerity_usto</pre>	num_un	bits = 1024
<pre>k_R_scale = terming_parameters.k_lis_scale gption = terming_parameters.eption alpha = terming_parameters.dopout_iniden dopout_iniden = terming_parameters.dopout_hiden alp. thepe=(tors, i, 38, 28) injot_vers.forg) rlp = tasgne.layers.parameters.dopout_iniden rlp = tasgne.layers.parameters.dopout_iniden noninearty-tasgnet noninearty-</pre>	n_hidd	len_layers = 3
<pre>epsilon = learning_parameters.siphs doport_in = learning_parameters.information doport_in = learning_parameters.information doport_in = learning_parameters.information doport_in = learning_parameters.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infort_vare.information infor</pre>	activa	ntion = binary_net.binary_tanh_unit
<pre>alpha = learning_parameters.shopout_ind dropout_hiddem = learning_parameters.dropout_liddem mapsot_hiddem = learning_parameters.dropout_liddem mapsot_hiddem = learning_parameters.dropout_liddem mapsot_tin for k in range(n_hiddem_layers): mlp = learning_toperstayer(</pre>	W_LR_s	cale = learning_parameters.W_LR_scale
<pre>dropout_in = learning_parameters.dropout_hidden dropout_inden = learning_parameters.dropout_hidden nip = leagne.layers.dropout_sidden nip = leagne.layers.dropout_sidden nip = leagne.layers.denties.identity, nem_unitson nip = leagne.layers.denties.identity, nonlinearity.activation) nip = leagne.layers.denties.identity, nonlinearity.activation nip =</pre>		
<pre>dropout_hidden = lasming_parameters.dropout_hidden #1 = lasagee.layers.inputiager(</pre>		
<pre>mlp = lasgne.layers.Inputayer(shaper(kome, 1, 28, 22), input_variupt) mlp = lasgne.layers.Dropoutlayer(mlp = lasgne.layers): mlp = binary.met.DensetLayer(abl, binary.met.DensetLayer(abl, binary.met.DensetLayer(abl, conlinearity.lasgne.nonlinearities.identity, mu_wints.mu_wints) mlp = lasgne.layers.BattWormLayer(alp, conlinearity.settivation) mlp = lasgne.layers.BattWormLayer(alp, conlinearity.settivation) mlp = lasgne.layers.BattWormLayer(alp, conlinearity.settivation) mlp = lasgne.layers.BattWormLayer(alp, conlinearity.settivation) mlp = lasgne.layers.BattWormLayer(alp, conlinearity.settivation) mlp = lasgne.layers.Depotitizer(alp, doftary.mu_mits.intertity.metr(alp, monlinearity.settivation) mlp = lasgne.layers.BattWormLayer(alp, monlinearity.settivation) mlp = lasgne.layers.BattWormLayer(alp, monlinearity.settivation) mlp = lasgne.layers.BattWormLayer(alp, monlinearity.setLayer(alp, monlinearity.setLayer(alp, monlinearity.setLayer(alp, monlinearity.setLayer(alp, monlinearity.setLayer(alp, monlinearity.setLayer(alp. alphaelpha) binary.mu_matters.layer(alphaelpha) binary.mu_matters.layer(alphaelpha) binary.mu_matters.layer(alphaelpha) binary.mu_matters.layer(alphaelpha) binary.mu_matters.layer(alphaelpha) binary.mu_matters.layer(alphaelpha) binary.mu_matters.layer(alphaelpha) binary.mu_matters.layer(alphaelpha) binary.mu_matters.layer(alphaelpha) binary.mu_matters.layer(alphaelpha) binary.mu_matters.layer(alphaelphaelphaelphaelphaelphaelphaelph</pre>		
<pre>shape(Nome, 1, 28, 28), iput_variup() nlp = lasgne.layers.Dropoutlayer(</pre>	dropou	tt_hidden = learning_parameters.dropout_hidden
<pre>shape(Nome, 1, 28, 28), iput_variup() nlp = lasgne.layers.Dropoutlayer(</pre>		
<pre>inpit_var=inpit inpit_var=inpit mlp,</pre>	mlp =	
<pre>mlp = lasgen.layers.Oropoutlayer(mlp, pdropout_in) for k in range(n_hidden_layers): alp = binary.net.DenseLayer(mlp, binary.binary, stochastics:stochastic, Het, W_LR_scale*Lk_scale, molinearity=lasgen.enninearities.identity, magnitionum_minits) alp = lasgen_layers.NethormLayer(mlp, molinearity=astronominearityLayer(mlp, molinearity=astronominearityLayer(mlp, binary.et.DensetLyer(mlp, binary.et.DensetLyer(mlp, binary.et.DensetLyer(mlp, binary.binary, stochastics:tochastic, Het, kit_scale*Lk_scale, molinearity=astronominearities.identity, ma_mlp, epsiloneepsilon, alph= lasgen_enapse.continearities.identity, ma_mlp, epsiloneepsilon, alph= lasgen_enapse.forminearities.identity, magnities.identity, map. epsiloneepsilon, alph=astronominearity=astronominearities.identity, map. epsiloneepsilon, alph=astronominearity=astronominearities.identity, map. epsiloneepsilon, alph=astronominearity=astronominearities.identity, map. epsiloneepsilon, alph=astronominearity=astronominearities.identity, map. epsiloneepsilon, alph=astronominearity=astronominearities.identity, epsiloneepsilon, alph=astronominearity=astronominearities.identity, epsiloneepsilon, alph=astronominearity=astronominearities.identity, epsiloneepsilon, alph=astronominearity=astronominearities.identity, epsiloneepsilon, alph=astronominearity=astronominearity=astronominearities.identity, epsiloneepsilon, alph=astronominearity=astronominearity=astronominearity=astronominearity=astronominearity=astronominearity=astronominearity=astronominearity=astronominearity=astronominearity=astronominearity=astronomine</pre>		
<pre>mlp, pdropout_in) for k in range(n_bidden_layers):</pre>		input_var=input)
<pre>mlp, pdropout_in) for k in range(n_hidden_layers):</pre>		
<pre>pdropout_in) for k in range(_nidem_layers): mlp, binary_net.Denselayer(mlp, binary_net.ivers.stathkomLayer(mlp, estimative.stoonstate, mlp, estimative.stoonstate(, mlp, stoonstate(.stoonstate(, stoonstate(.stoonstate(,</pre>	mib =	
<pre>for k in range(n,hiden_layers): mlp = binary_net.BenseLayer(</pre>		
<pre>mlp = binary_net.DenseLayer(mlp, binary-binary, stochastic.chastic, H+H, w_L&_scale=W_L&_scale, nonlinearity-lasage.nonlinearities.identity, num_units=num_units) mlp = lasage.layers.BatchkorsLayer(mlp, got_ensite() mlp = lasage.layers.KonlinearityLayer(mlp, nonlinearity-activation) mlp = lasage.layers.chopoutLayer(mlp, p=dropout_hidden) mlp = binary_net.DenseLayer(mlp, binary=binary, stochastic.stochastic, the stochastic.stochastic, mlp, mumits=num_outputs) mlp = lasage.layers.BatchkormLayer(mlp, epsilon-epsilon, alpha=alpha)</pre>		p=oropout_in)
<pre>mlp = binary.net.DenseLayer(mlp, binary-binary, stochastic.chastic, H+H, k_t_scale=w_LE_scale, nonlinearitylasage.nonlinearities.identity, mm_units=num_units) mlp = lasagne.layers.BatchkomLayer(alph=alpha) monlinearity=activation) mlp = lasagne.layers.NonDimearityLayer(nonlinearity=activation) mlp = lasagne.layers.PropoutLayer(nlp, p.dropout_hidden) mlp = binary.net.DenseLayer(mlp, binary-binary, stochestLestochastic, til, roulinearity=activation. mlp = lasagne.layers.BatchkonmLayer(alp, eptinonegosILon, alph=alpas=layers.BatchkonmLayer(alp, eptinonegosILon, alph=alpas=layers.BatchkonmLayer(alph=alpas=layers.BatchkonmLayer(alph=alpas=layers.BatchkonmLayer(alph=alpas=layers.BatchkonmLayer(alph=alpas=layers.BatchkonmLayer(alph=alpas=layers.BatchkonmLayer(alph=alpas=layers.BatchkonmLayer(alph=alpas=layers.BatchkonmLayer(alph=alpas=layers.BatchkonmLayer(alph=alpas=layers.BatchkonmLayer(alph=alpas=layers.BatchkonmLayer(alph=alpas=layers.BatchkonmLayer(alph=alpas=layers.BatchkonmLayer(alph=alpas=layers.BatchkonmLayer(alph=alpas=layers.BatchkonmLayer(alph=alpas=layers.BatchkonmLayer(alph=alpas=layers.BatchkonmLayer(alph=alpas=layers.BatchkonmLayer(alph=alpas=layers.BatchkonmLayer(blochonmLayers.BatchkonmLayer(blochonmLayers.BatchkonmLayer(blochonmLayers.BatchkonmLayer(blochonmLayers.BatchkonmLayer(blochonmLayers.BatchkonmLayer(blochonmLayers.BatchkonmLayer(blochonmLayers.BatchkonmLayer(blochonmLayers.BatchkonmLayer(blochonmLayers.BatchkonmLayer(blochonmLayers.BatchkonmLayer(blochonmLayers.BatchkonmLayer(blochonmLayers.BatchkonmLayer(blochonmLayers.BatchkonmLayer(</pre>	for k	in range(n hidden lavers).
<pre>mlp, timy-bimry, stochastic.stochastic, HH, vLt_scalewLE_scale, nonlinearity-lasgne.nonlinearitis.identity, mm_units-nu_units) mlp = lasgne.layers.BatchkormLayer(</pre>	TOT K	in range(n_iiiduen_iayers).
<pre>mlp, timy-bimry, stochastic.stochastic, HH, vLt_scalewLE_scale, nonlinearity-lasgne.nonlinearitis.identity, mm_units-nu_units) mlp = lasgne.layers.BatchkormLayer(</pre>	m]	n - hinary net Densel aver(
<pre>binary-binary, stochastic=stochastic, H+H, W_LE_scale=W_LE_scale, nonlinearity-lasgne.nonlinearities.identity, nm_units=nu_units) mlp = lasgne.layers.BatchMormLayer(mlp, esplion-epsilon, alpha=alpha) mlp = lasgne.layers.NonlinearityLayer(mlp, nonlinearity=activation) mlp = lasgne.layers.DropoutLayer(mlp, binary-binary, stochastics(chastic, H+H, W_LE_scale=W_LE_scale, nonlinearity=lasgne.nonlinearities.identity, num_units=num_outputs) mlp = lasgne.layers.BatchMormLayer(alp, epsilon-epsilon, alpha=alpha]</pre>		
<pre>stochastics_tochastic, H+H, V_LR_scalew_LE_scale, nonlinearity_lasgme_nonlinearities.identity, num_units=num_units) mlp = lasgme_layers.BothNormLayer(</pre>		
<pre>H+H; W_LR_sciew_LE_scale; nonlinearity-lasgene.nonlinearities.identity; nminsumu_units; mlp = lasgene.layers.BatchNormLayer(</pre>		
<pre>w L& scalew L& scale, nonlinearity-lasge.nonlinearities.identity, nmm_units-num_units) mlp = lasgene.layers.BatchNormLayer(</pre>		
<pre>nonlinearity-lasgne.nonlinearities.identity, nmm_unitsnum_units) mlp = lasgne.layers.BatchNomLayer(mlp, epsilon-epsilon, alpha=alpha) mlp = lasgne.layers.NonlinearityLayer(mlp, nonlinearity=activation) mlp = lasgne.layers.DropoutLayer(mlp, pedropout_hidden) mlp = binary.met.DecesLayer(slo, binary.met.DecesLayer(slo, binary.met.DecesLayer(nonlinearity-lasgne.nonlinearities.identity, num_unitsnum_outputs) mlp = lasgne.layers.BatchNomLayer(alp, epsilon-epsilon, alpha=alpha)</pre>		
<pre>num_units=num_units) mlp = lasagne.layers.BatchNormLayer(</pre>		
<pre>mlp = lasgne.layers.BatchNormLayer(mlp, epsilon.epsilon, alpha=alpha) mlp = lasgne.layers.NonlinearityLayer(mlp, nonlinearity=activation) mlp = lasgne.layers.DropoutLayer(mlp, pedropout_hidden) mlp = binary.net.DecomsLayer(slo, singry=hinary, stochastics(chastic, H+H, W_LR_scale=W_LE_scale, nonlinearity=asgne.nonlinearities.identity, num_units=num_outputs) mlp = lasgne.layers.BatchNormLayer(alp, epsilon-epsilon, alpha=alpha]</pre>		
<pre>mlp, epsilon-epsilon, alpha-alpha) mlp = lasgne.layers.NonlinearityLayer(mlp, nonlinearity-activation) mlp = lasgne.layers.Dropoutlayer(mlp, p=dropout_hidden) mlp = binary.net.DenseLayer(mlp, dstrophastic=stochastic, Het, KLR_scale=M_LE_scale, nonlinearity-lasgne.nonlinearities.identity, num_units=num_outputs) mlp = lasgne.layers.BatchNormLayer(alp, epsilon-epsilon, alpha=alpha)</pre>		
<pre>mlp, epsilon-epsilon, alpha-alpha) mlp = lasgne.layers.NonlinearityLayer(mlp, nonlinearity-activation) mlp = lasgne.layers.Dropoutlayer(mlp, p=dropout_hidden) mlp = binary.net.DenseLayer(mlp, disp, disp, disp, disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. disp. d</pre>	m]	n = lasagne.lavers.BatchNormlaver(
<pre>esilon-gsilon, alpha-alpha) mlp = lasgne_layers.NonlinearityLayer(mlp, nonlinearity-activation) mlp = lasgne_layers.DropoutLayer(mlp, p=dropout_hidden) mlp = binary_net.DenseLayer(mlp, binary_binary, stochastic<tothatic, H+H, M_LR_scale=M_LR_scale, nonlinearity-lasgne.nonlinearities.identity, num_units-num_outputs) mlp = lasgne.layers.BatchNormLayer(mlp, epsilon-epsilon, alpha=alpha)</tothatic, </pre>		
<pre>alpha=alpha) mlp = lasage.layers.BatchHormLayer(</pre>		
<pre>mlp = lasagne.layers.KonlinearityLayer(</pre>		
<pre>mlp, nonlinearity=activation) mlp = lasgme.layers.DropoutLayer(mlp, p=dropout_hidden) mlp = binary_net.DenseLayer(mlp, binary_eniary, stochastic=stochastic, H+H, M_LR_scale=M_LR_scale, nonlinearity_lasgme.nonlinearities.identity, num_units=num_outputs) mlp = lasgme.layers.BatchNormLayer(mlp, epsilon-epsilon, alpha=alpha)</pre>		
<pre>mlp, nonlinearity=activation) mlp = lasgme.layers.DropoutLayer(mlp, p=dropout_hidden) mlp = binary_net.DenseLayer(mlp, binary_eniary, stochastic=stochastic, H+H, M_LR_scale=M_LR_scale, nonlinearity_lasgme.nonlinearities.identity, num_units=num_outputs) mlp = lasgme.layers.BatchNormLayer(mlp, epsilon-epsilon, alpha=alpha)</pre>	ml	p = lasagne.layers.NonlinearityLayer(
<pre>mlp = lasagne.layers.DropoutLayer(mlp, p=dropout_hidden) mlp = binary_net.CenseLayer(mlp, binary=binary, stochastic=stochastic, H=H, M_LR_scale=M_LR_scale, nonlinearity_lasagne.nonlinearities.identity, num_units=num_outputs) mlp = lasagne.layers.BatchNormLayer(mlp, epsilon-repsilon, alpha=alpha)</pre>		
<pre>mlp, p=dropout_hidden) mlp = binary_net.DenseLayer(mlp, binary=binary, stochastic=stochastic, H=H, M_LR_scale=M_LR_scale, nonlinearity=lassgen.onolinearities.identity, num_units=num_outputs) mlp = lassgen.layers.BatchNormLayer(mlp, epsilon-epsilon, alpha=alpha)</pre>		nonlinearity=activation)
<pre>mlp, p=dropout_hidden) mlp = binary_net.DenseLayer(mlp, binary=binary, stochastic=stochastic, H=H, M_LR_scale=M_LR_scale, nonlinearity=lassgen.onolinearities.identity, num_units=num_outputs) mlp = lassgen.layers.BatchNormLayer(mlp, epsilon-epsilon, alpha=alpha)</pre>		
<pre>p=dropout_hiddm) mlp = binary_net.Denseiayer(mlp, binary=binary, stochastics=stochastic, H+H; scale=(L_Scale, non_inearity=lasage.nonlinearities.identity, non_units=num_outputs) mlp = lasagen_layers.BatchhormLayer(alph=alph=alph=3) </pre>	ml	p = lasagne.layers.DropoutLayer(
<pre>mlp = binary_net.Denselayer(mlp, binary_net.Oenselayer(mlp, stochastic.stochastic, H+H, M_LR_scale=M_LR_scale, nonlinearity-lassgen.nonlinearities.identity, num_units=num_outputs) mlp = lassgen.layers.BatchNormLayer(mlp, epsilonrepsilon, alpha=alpha)</pre>		mlp,
<pre>mlp, binary=binary, stochastic=stochastic, H+H, M_LR_scale=M_LR_scale, nonline=rity_lasage.nonlinearities.identity, num_units=num_outputs) mlp = lasage.layers.BatchNormLayer(mlp, epsilon=epsilon, alpha=lpha)</pre>		p=dropout_hidden)
<pre>mlp, binary=binary, stochastic=stochastic, H+H, M_LR_scale=M_LR_scale, nonlinearity=lasage.nonlinearities.identity, num_units=num_outputs) mlp = lasage.layers.BatchNormLayer(mlp, epsiloneepsilon, alpha=lpha)</pre>		
<pre>binary=binary, stochastic=stochastic, H+H; W LR_scale=M_LE_scale, nonlinearity=lasagne.nonlinearities.identity, nom_units=num_outputs) mlp = lasagne.layers.BatchNormLayer(</pre>	mlp =	
<pre>stochastic=stochastic, H+H, W_LR_scale=w_LR_scale, nonlinearity=lasgne.nonlinearities.identity, num_units=num_outputs) mlp = lasgne.layers.BatchNormLayer(mlp, epsiloneepsilon, alpha=lpha)</pre>		
<pre>H+H, W_LR_scale+M_LR_scale, nonlinearity-lasgne.nonlinearities.identity, num_units=num_outputs) mlp = lasgne.layers.BatchNormLayer(</pre>		
<pre>W_LR_scalew_ULR_scale, nonlinearity-lasgne.conlinearities.identity, num_units=num_outputs) mlp = lasgne.layers.BatchNormLayer(</pre>		
<pre>nonlinearity-lasgne.nonlinearities.identity, num_units=num_outputs) mlp = lasgne.layers.BatchNormLayer(mlp, epsilon=epsilon, alpha=alpha)</pre>		
<pre>num_units=num_outputs) mlp = lasagne.layers.BatchNormLayer(mlp, epsilonrepsilon, alpha=alpha)</pre>		
<pre>mlp = lasgne.layers.BatchNormLayer(mlp, epsilon=epsilon, alpha=alpha)</pre>		
alp, epsilon=epsilon, alpha=lpha)		num_units=num_outputs)
alp, epsilon=epsilon, alpha=lpha)		
epsilon=epsilon, alpha=alpha)	mlp =	
alpha=alpha)		
recurring the	not	
	return	• ••••

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):



XILINX > ALL PROGRAMMABLE.

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 <u>http://www.pynq.io/community.html</u>:
 - 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.



