Motivation

- Computational Efficiency (inkl. Horowitz) !! Why do we do this?
- Improved Generalization (train with low precision (lp) but keep hp weights and use them during test -¿ better results) Some power gains during training, no power gains during test.
- Energy efficiency during test time if you use lp weights, sacrifice accuracy for power saving
- Examples and related concepts: dropout, ReLu (sparse gradient), influence of feature discretization,

Binarization Schemes

- Deterministic vs. Stochastic

Gradient Estimation

- Path Derivative Gradient Estimators (reparametrization, ST, SlopeAnnealing)
- Score Function based Gradient Estimators (reinforce+many others)
- Expectation Backpropagation

See Gumble Softmax Paper for overview

Algorithms Adam + BN

- General Review
- Illustrate their importance for QNNs
- Implement algorithm to see how it works without Adam and BN?

Full Algorithm

- Try to visualize, illustrate procedure (highlight similarities and differences to conventional NN training and inference)

Experiments

- Describe different NN architectures (DNN, CNN, RNN)
- Describe data sets (MNIST, CIFAR, ImageNet, PennTree)
- binary vs low precision

Results

- Training results
- Performance results (look up older benchmarks to compare loss in in accuracy to)
- find comparisons from other references

Evolution of the QNN paper

Training NN with lp multiplications

- The inital motivation was to reduce the power consumption of multipliers, which are claimed to be the most power hungry.
- No Binarization of anything yet, only lp weights
- A common pattern already shows: lp multipliers + hp accumulators. Parameter updates are also hp.
- Cost of multiplier $O(precision^2)$ while cost of accumulator $O(precision)$
- Problem with fixed point: low dynamic range $\rightarrow$ dynamic fixed point.
- Evaluation of maxout on MNIST, CIFAR, SVHN with three formats (floating point, fixed point, dynamic fixed point)
- Techniques: Maxout, dropout, momentum, weight decay, dyn. fixed point, update prc vs prop prec.
- Comparison floating point vs. fixed point ?
- Explain demand for sufficient precison of update (compared to Propagations) due to SGD updates
- Half-precision has little to no impact (hp fine-tuning?)
- Couple of plots showing final test error as function of everything
- References to lp network training in the 90ies!
- No evaluation of computational gains, only show how robust NNs are to lp muls.

Binary Connect: Training DNNs with binary weights during propagations

- Emphasis shifts from efficiency to generalization properties (ala dropout)
- Only binarization of weights, activations not yet binary! Backprop still contains muls!
- mention discretization as form of noise which preserves expected value of weights.
- Again: need for hp accumulation (cite several studies incl haml limtied precision one), Interesting: reference which says brain synapse precision is around 6-12 bits
- Relations to dropout, dropconnect (!), variational weight noise?
- DropConnect: Only expected value of weights needs hp
- Another reference about hardware cost of add+mul [22]
- Binarization: Deterministic vs. stochastic, Hard sigmoid more efficient!
- Algorithm: Biases not binarized!
- Tricks: Weight clipping, batch norm, adam, normalized initialization, L2-SVM output layer
- Inference: Three ways: keep binary weights, use hp weights or ensemble of binary networks (sampled)
- Compute dropout+DNN Performance numbers for CIFAR + SVHN?
- Fig 1: feature coadaption comparison
- Ref. [39] about binary DNN: Fixed-Point feedforward deep Neural...retrains network!
- 2/3 of all muls are due to forward/backward prop
- No demonstration of power saving/efficieny gains

Neural Networks with few Multiplications

- Muls again in foucs ("Multiplier light networks")
- Binarization of weights + quantization (!) of activations
- No quantization of BN or ADAM
- Convert muls in backprop to bitshifts by pow2 quantization of activations
- Ref backprop without muls 1999
- First mention of random number generation cost
- Ternary connect -1,0,+1, sampling scheme similar to lp reference!
- how does hp error signal back prop? how about hp input?
- Most muls in weight updates (2MN+3M muls at least)
- Mention muls incurred by batch norm! Cost of divison!
- Table 2 analysis of total num of muls
- Compares only to ordinary SGD (but ok they use the same for the QNN)
- QNN converges slower but better in the end fig 1
- They dont explain the bit shift operations properly
- non-uniform distribution of activations
- Explanation of regularization by lp weights - VERY similar to entropy SDG ideas! large-basin solution, small description length, Additional reference (Neelakatan: Adding gradient noise)

Quantized Neural Networks

- Very lp activations + 6bit gradients (enabling bit-wise ops)
- first to actually consider gradient of det./stoch. binary units.
- first experiments on RNNs + ImageNet
- Extensive references (e.g. network compression for inference)
- MAC ops replaced by XNOR and popcount ops
- CNNs benefit mainly from activation quantization (large neuron to weight ratio)
- Binary kernel repetition (worth mentioning?)
- Abandon stochastic binarization (only in activations at train time)
- NEW: Gradient estimators (clipped straight through) due to binary activations.
- Shaky argument about independence of binarizations for the "derivation" of the clipped ST estimator. But intuitively ok (eq. 6)
- Address cost of BN, which is particularly large for CNNs + inverse sqrt
- How does the approximate shift work in case of negative argument?
- Make screenshpt of "very unoptimized" baseline gpu kernel (theano repo), shift based BN+ADAM are in Torch repo
- Algorithm 1: input binarization? activation functions missing - no, they use det./stoch. binary units!
- Need to disentangle back-prop equations...

- Input handling: 8bit fixed point , check algorithm 4...
- More than one bit: Quantization+XNORpopcount, Ref to DoReFa net and logarithmic data representation papers.