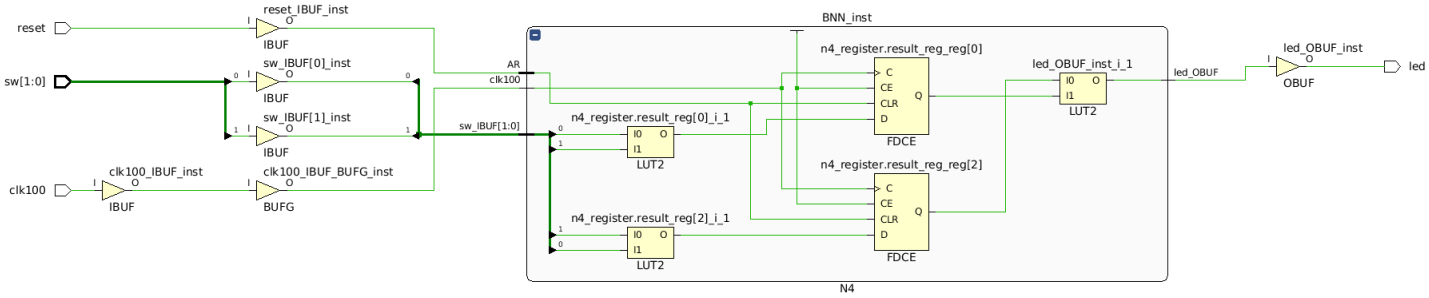

10 Days Of Grad: Deep Learning From The First Principles (/neural-networks)

Day 6: Saving Energy with Binarized Neural Networks

Bogdan Penkovsky

Jan 22, 2020 · 7 min read · 10 Days Of Grad (http://penkovsky.com/categories/10-days-of-grad/)

Last week Apple has acquired (https://techcrunch.com/2020/01/15/apple-buys-edge-based-ai-startup-xnor-ai-for-a-reported-200m/) XNOR.ai startup for amazing $200 million. The startup is known for promoting binarized neural network algorithms to save the energy and computational resources. That is definitely a way to go for mobile devices, and Apple just acknowledged that it is a great deal for them too. I feel now is a good time to explain what binarized neural networks are so that you can better appreciate their value for the industry.

Today's post is based on

- Day 1: Learning Neural Networks The Hard Way (/neural-networks/day1/)
- Day 2: What Do Hidden Layers Do? (/neural-networks/day2/)
- Day 4: The Importance Of Batch Normalization (/neural-networks/day4/)

The source code from this post is available on Github (https://github.com/masterdezign/10-days-of-grad/tree/master/day6).

The Basics

Knowing what a (real-weight) neural network is (/neural-networks/day1/) you already have the most essential information. In contrast, a binarized neural network (BNN) has its weights and activations limited to a single bit precision. Meaning that those have values of either $+1$ or -1. Below is a comparison table.

	Classical NN	BNN
Multiplication	Floating point, 32 bits	**XNOR gate**, 1 bit {-1, +1}
Linear	Sum	Popcount

Binarized neural network vs conventional neural network with real weights

Having single-bit values can drastically reduce hardware requirements to operate those networks. As you remember from Day 1 (/neural-networks/day1/), neural networks are built around activations of dot products. When every value is $+1$ or -1, multiplication can be replaced with XNOR binary operation, thus eliminating the need in hardware multipliers. The dot product sum is then replaced by a simple bit counting operation ($\mathrm{Popcount}$). And finally the activation is merely a thresholding (Sign) function.

Overall, the architecture can speed up (http://arxiv.org/abs/1602.02830) conventional hardware or even better, drastically reduce the area of dedicated ASICs (https://en.wikipedia.org/wiki/Application-specific_integrated_circuit). Reduced area means smaller manufacturing cost. Not less important is another implication, reduced energy consumption. That is crucial for edge (/project/edge-ai/) devices. To further save the energy one can even merge computation and memory giving rise to emerging in-memory computing (/project/edge-ai/) technology.

As we have discussed before (/neural-networks/day4/), batch normalization is helpful to prevent neurons from saturation and to faster train the network. It turns out that for binarized networks batchnorm is actually indispensable. Indeed, neurons with Sign activations tend to be useless ("dead") otherwise. Moreover, for an efficient training, to make the error landscape smooth, it is advisable to also include batch normalization immediately before the softmax layer. Yes, the most important lesson to learn here is is actually how to *train* a BNN. I attempt to answer the most frequently encountered questions in the following section.

Binarized Neural Networks FAQ

Here are some typical questions that people tend to ask.

Q: Do I need to provide binarized inputs to a BNN?

A: No, BNN inputs can remain non-binarized (real). That results in the first layer slightly different from the others: having binary weights (+1 and -1), yet producing real pre-activation values. After activation, those values are converted to binary ones. Here is a hint:

> It is relatively easy to handle continuous-valued inputs as fixed point numbers, with m bits of precision.
> — Courbariaux *et al.* Binarized Neural Networks: Training Deep Neural Networks... (2016)

You should also be aware that indeed there exist methods (Hirtzlin2019) to have binary inputs as well.

Q: Do I need more neurons?

A: Yes, typically one needs 3 to 4 times more *binary* neurons to compensate for information loss.

Q: Do I need a softmax layer?

A: Only for BNN training. For inference (and simpler hardware) it can be removed.

Q: Do binary neurons have learnable biases?

A: No, biases are redundant as normally you want to use batch normalization layers. Those layers already learn bias-equivalent parameters.

Q: Why BNNs are sometimes called "integer networks"?

A: Because bit count (before activation) results in an integer value.

Q: Sign **activation gradient is zero**

A: We approximate $\mathrm{Sign}(x)$ derivative with $\mathrm{Hardtanh}(x)$[1] derivative, i.e. $1_{|x|\leq 1}$.

Q: Are gradients binarized in backprop training?

A: No, during the training one typically deals with real gradients and thus, real weights. Therefore, it is only in the forward pass where the network applies its binarized weights.

Q: So what is the interest then?

A: BNNs are interesting to develop dedicated hardware hosting pretrained networks for inference, i.e. performing only the forward pass. Think about handwriting recognition-based automated systems as a use case. Think about energy-harvesting sensors. Solar-powered smart cameras. Drones. Smart wearable devices (/publication/medical-bnn/) and implants. Binarized networks facilitate smaller, faster, and more energy-efficient inference devices. Though, there is also an effort towards (on-chip) binarized networks training.

Implementing Binarized Neural Networks in Haskell

There is no better way to understand a binarized network than to create one ourselves. This easy demo is built on top of Day 4 (https://github.com/masterdezign/10-days-of-grad/commit/95b08aa50abe3a0051662f79134545ba7e99ffdd). Here are a few code highlights.

First of all, the layers types we will use

```
data Layer a = -- A linear layer (BNN training)
               | BinarizedLinear (Matrix a)
               -- Batch normalization with running mean, variance, and two
               -- learnable affine parameters
               | Batchnorm1d (Vector a) (Vector a) (Vector a) (Vector a)
               | Activation FActivation
```

Second, we provide the Sign activation

```
sign :: Matrix Float -> Matrix Float
sign = computeMap f
  where
    f x = if x <= 0
              then -1
              else 1
```

and its gradient approximation:

```
sign' :: Matrix Float
      -> Matrix Float
      -> Matrix Float
sign' x = compute. A.zipWith f x
  where
    f x0 dy0 = if (x0 > (-1)) && (x0 < 1)
                  then dy0
                  else 0
```

Finally, we accommodate the forward and backward passes of the network through the `BinarizedLinear` layer.

```
_pass inp (BinarizedLinear w : layers) = (dX, pred, BinarizedLinearGradients dW : t)
 where
  -- Binarize the weights (!)
  wB = sign w

  -- Forward
  lin = maybe (error "Incompatible shapes") compute (inp |*| wB)

  (dZ, pred, t) = _pass lin layers

  -- Backward
  dW            = linearW' inp dZ
  -- Gradient w.r.t. wB (!)
  dX            = linearX' wB dZ
```

In the `main` function we define the architecture, starting with the number of neurons in each layer. Good news: on the MNIST handwriting recognition benchmark we can achieve accuracy comparable to those of a 32 bit network. However, we may need more neurons (by `infl` factor) in the hidden layers.

```
let infl = 4

let [i, h1, h2, o] = [784, infl * 300, infl * 50, 10]
```

Here is the network specification. Note the `BinarizedLinear` layers defined above. The first `BinarizedLinear` has real inputs, however every subsequent one receives binary inputs coming from `Sign` activation.

```
let net =
      [ BinarizedLinear w1
      , Batchnorm1d (zeros h1) (ones h1) (ones h1) (zeros h1)
      , Activation Sign

      , BinarizedLinear w2
      , Batchnorm1d (zeros h2) (ones h2) (ones h2) (zeros h2)
      , Activation Sign

      , BinarizedLinear w3
      -- NB this batchnorm (!)
      , Batchnorm1d (zeros o) (ones o) (ones o) (zeros o)
      ]
```

A final technical note is that dividing by the batch variance (and rescaling by a learnable parameter `gamma`) in batchnorm (https://github.com/masterdezign/10-days-of-grad/blob/master/day6/src/NeuralNetwork.hs#L294-L311) actually makes little sense since division (multiplication) by a positive constant does not change the sign. If you would like to get your hands dirty with the code, that is a good place to optimize. Another suggestion is to transfer your newly acquired skills to a convolutional network (/neural-networks/day5/) architecture[2].

See the complete project on Github (https://github.com/masterdezign/10-days-of-grad/tree/master/day6). If you have any questions, remarks, or any typos found please send me an email. For suggestions about the code, feel free to open a new issue (https://github.com/masterdezign/10-days-of-grad/issues).

Summary

Binarized neural networks (BNNs) are valuable for low-power edge devices. Starting with energy-harvesting sensors and finishing with smart wearable medical devices and implants. Binarized networks facilitate smaller, faster, and more energy-efficient hardware.

We have illustrated how to train a BNN solving a handwritten digits recognition task. Our binarized network can achieve accuracy comparable to a full-precision 32 bit network provided that we moderately increase the number of binary neurons. To better grasp today's concept, train your own BNNs. Pay attention to small details. Do not forget the batch normalization before the softmax. And good luck!

Further reading

Binarized neural networks:

- Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1 (http://arxiv.org/abs/1602.02830)
- XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks (https://arxiv.org/abs/1603.05279)
- Emerging technology (low-power hardware) (/project/edge-ai/)

1. $\mathrm{Hardtanh}(x) = \max(-1, \min(1, x))$ ^
2. In convolutional layers you may also want to increase the number convolutional filters. ^

Deep Learning (http://penkovsky.com/tags/deep-learning/) Haskell (http://penkovsky.com/tags/haskell/)

Related

- Day 5: Convolutional Neural Networks Tutorial (/neural-networks/day5/)
- Day 4: The Importance Of Batch Normalization (/neural-networks/day4/)
- Day 3: Haskell Guide To Neural Networks (/neural-networks/day3/)
- Day 2: What Do Hidden Layers Do? (/neural-networks/day2/)
- Day 1: Learning Neural Networks The Hard Way (/neural-networks/day1/)