Binarized Neural Network

Presented by **Nancy Nayak (EE17D408)**
Guide: **Dr. Sheetal Kalyani**

Department of Electrical Engineering

Indian Institute of Technology Madras (IITM)

Importance of Deep Neural Network (DNN)

- Object recognition from images
- Speech recognition
- Atari and Go games in reinforcement learning
- Generating abstract art

Motivation behind compactifying DNN

Challenges in Deep Learning

- Billions of parameters for a practical Deep Network
- Deep learning requires a lot of memory and computing power
- This makes it difficult to run a pre-trained Neural Network on low-cost/low-power devices

Motivation behind compactifying DNN

Challenges in Deep Learning

- Billions of parameters for a practical Deep Network
- Deep learning requires a lot of memory and computing power
- This makes it difficult to run a pre-trained Neural Network on low-cost/low-power devices

Existing methods to reduce over parametrization

- Shallow network
- Pruning: Compress a pre-trained network
- Quantizing network parameters to several levels

Motivation behind compactifying DNN

Challenges in Deep Learning

- Billions of parameters for a practical Deep Network
- Deep learning requires a lot of memory and computing power
- This makes it difficult to run a pre-trained Neural Network on low-cost/low-power devices

Existing methods to reduce over parametrization

- Shallow network
- Pruning: Compress a pre-trained network
- Quantizing network parameters to several levels

Authors' solution: Binarized Neural Network (BNN) !!

Binarized Neural Network: Training Neural Networks with Weights and Activations Constrained to $+1$ or -1

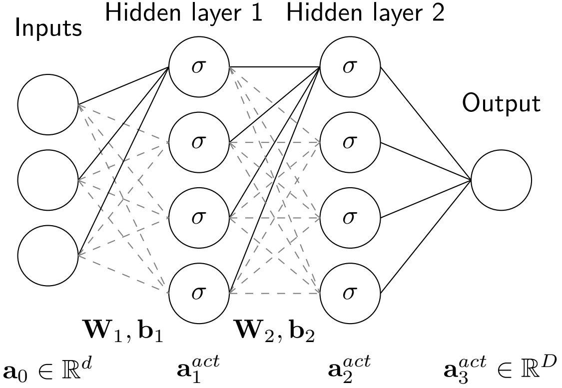
Published in NIPS 2016

Authors: Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran-El-Yaniv, Yoshua Bengio

Outline

Basic architecture of DNN

Basics of NN: Forward Propagation



$$\mathbf{a}_1^{act} = \sigma(\mathbf{s}_1) = \sigma(\mathbf{a}_0.\mathbf{W}_1 + \mathbf{b}_1)$$

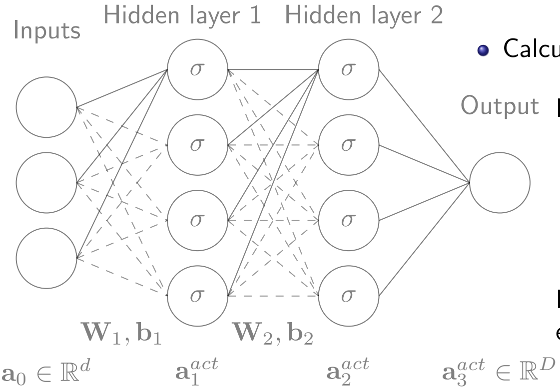
$$\mathbf{a}_2^{act} = \sigma(\mathbf{s}_2) = \sigma(\mathbf{a}_1^{act}.\mathbf{W}_2 + \mathbf{b}_2)$$

Output, $\mathbf{a}_3^{act} = S(\mathbf{a}_2^{act})$,

where S is Softmax and

$$S(\mathbf{a}_2^{act}(i)) = \frac{\mathbf{a}_2^{act}(i)}{\sum_i \mathbf{a}_2^{act}(i)}$$

Basics of NN: Loss functions



- Output at final layer L : $\mathbf{a}_3^{act}$
- Target output: $\mathbf{a}_3^{true}$
- Calculate Cost function C

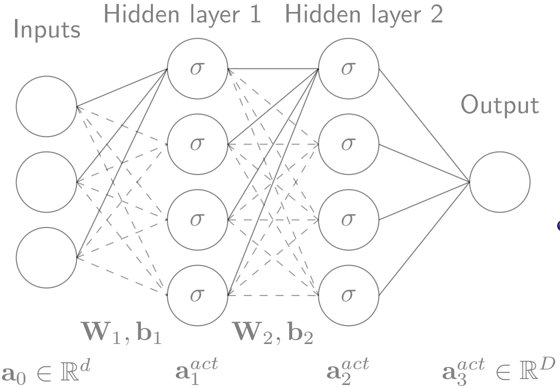
For MSE loss,

$$C = \frac{1}{D}\sum_{i=1}^{D}(a_3^{act}(i) - a_3^{true}(i))^2$$

For D class classification, Cross entropy loss,

$$C = \sum_{c=1}^{D} -a_3^{true}(c)log(a_3^{act}(c))$$

Basics of NN: Back Propagation



- If θ_i^t represents i^{th} parameter of a DNN at time instant t, then the gradient of cost function C wrt parameter θ_i^t is given by

$$g_i^t = \frac{\partial C}{\partial \theta_i^t}$$

- The general update equation for θ_i^t at time instant t is

$$\theta_i^{t+1} = \theta_i^t - \eta . g_i^t$$

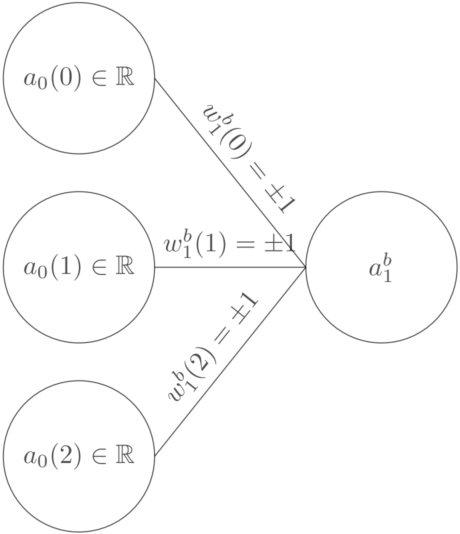
where η is step size

- eg. Stochastic Gradient Descent (SGD), Adam

Architecture of BNN

How is it different from DNN?

First layer in **train-time**

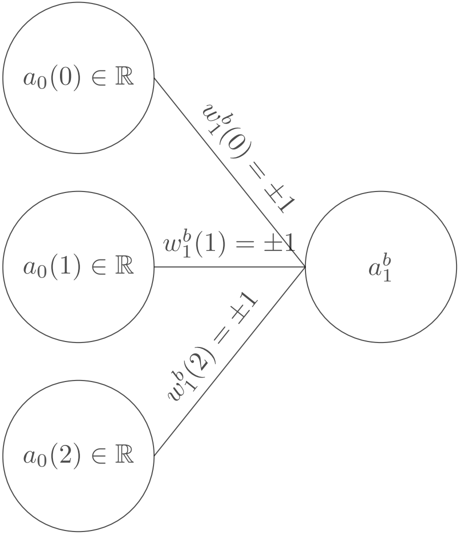


- Deterministic Binarization function to binarize weights and activations.

$$x^b = Sign(x) = \begin{cases} +1 & x \geq 0 \\ -1 & x < 0 \end{cases}$$

- Lets denote binarized weight $\mathbf{w}_1^b = Sign(\mathbf{w}_1)$
- Save binary weights $\mathbf{w}_0^b, \mathbf{w}_1^b, \mathbf{w}_2^b$ in binary variables along with real weights $\mathbf{w}_0, \mathbf{w}_1, \mathbf{w}_2$
- For SGD to work, the variables over which we optimize must be floats.

First layer in **train-time**

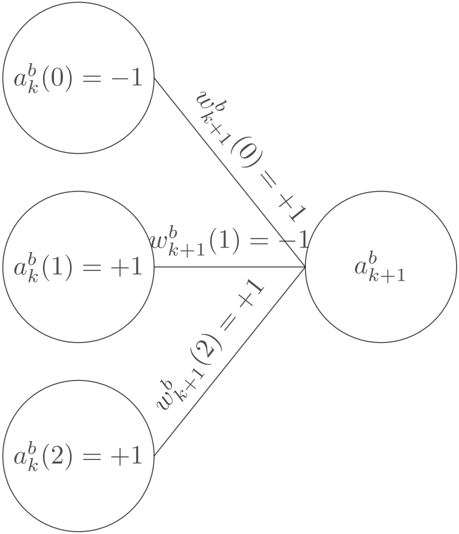


- *Sign* serves the role of nonlinearity σ

$$a_1^b = Sign(w_1^b(0)a_0(0) + w_1^b(1)a_0(1) + w_1^b(2)a_0(2)) = \pm 1$$

- Therefore excluding first layer, all other layer inputs are binarized

Other than first layer

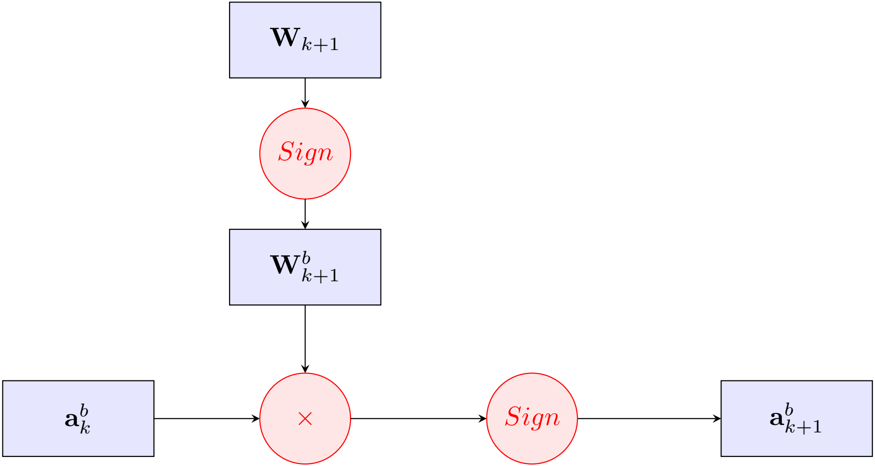


- For $k = 1, 2$

$$a_{k+1}^b = Sign(+1 * (-1) - 1 * (1) + 1 * (1)) = -1$$

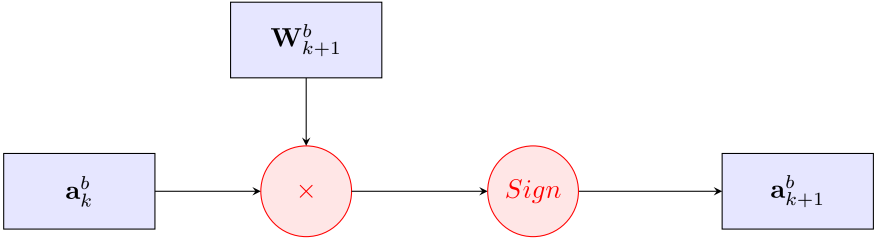
- Weights and activations are binary, So no arithmetic multiplication is required

BNN during train-time



The real weight matrix $\mathbf{W}_k$ and is converted to binarized weight matrix $\mathbf{W}^b_k$.

BNN during run-time



- Use the trained binarized weights during run-time
- It will not speed up training much, but after training we can discard real variables, keep binary weights resulting in less memory consumption and less computation of pre-trained BNN at run-time

Algorithm for training BNN

In comparison with algo. for training DNN

Algorithm1 : Computing parameter gradients

- Require:
  - A minibatch of inputs and targets $(\mathbf{a}_0, \mathbf{a}*)$
  - previous weights $\mathbf{W}^t$
  - previous Batch Normalization parameters θ^t
  - weight initialization coefficient γ_w
  - previous learning rate η^t

- Ensure:
  - Updated weights $\mathbf{W}^{t+1}$
  - Updated Batch Normalization parameters θ^{t+1}
  - Updated learning rate η^{t+1}

Forward propagation for DNN

- **for** $k = 1$ to L **do**

  $\mathbf{s}_k \leftarrow \mathbf{W}_k \mathbf{a}_{k-1}^{act}$

  $\mathbf{a}_k \leftarrow BatchNorm(\mathbf{s}_k, \theta_k)$

  if $k < L$ **then**

  $\mathbf{a}_k^{act} \leftarrow \sigma(\mathbf{a}_k)$

  end if

  end for

$$\mathbf{s}_k \leftarrow \mathbf{W}_1 \mathbf{a}_{k-1}$$

$$\hat{\mathbf{s}}_k \leftarrow \frac{\mathbf{s}_k - \mu_k}{\sqrt{\sigma_k^2 + \epsilon}} \quad (2)$$

$$\mathbf{a}_k \leftarrow \gamma_k \hat{\mathbf{s}}_k + \beta_k \quad (3)$$

$$\mathbf{a}_k^{act} \leftarrow \sigma(\mathbf{a}_k)$$

where

$$\mu_k = \mathbb{E}(\mathbf{s}_k)$$

$$\sigma_k^2 = Var(\mathbf{s}_k)$$

Forward propagation for DNN

- **for** $k = 1$ to L **do**

  $\mathbf{s}_k \leftarrow \mathbf{W}_k \mathbf{a}_{k-1}^{act}$
  $\mathbf{a}_k \leftarrow BatchNorm(\mathbf{s}_k, \gamma_k, \beta_k)$
  if $k < L$ **then**
  $\mathbf{a}_k^{act} \leftarrow \sigma(\mathbf{a}_k)$
  end if

  end for

- Inputs in minibatch of size m
- Inputs in one batch $\mathbf{a}_{0,0}, \mathbf{a}_{0,1}^{act}, \ldots \mathbf{a}_{0,m}^{act}$
- Parameters to be learned at k^{th} layer are γ_k, β_k
- Consider the i^{th} input of the batch $\mathbf{a}_{0,i}^{act}$ but for ease of representation let's remove i.

Forward propagation for BNN

$W_1^b \leftarrow Binarize(W_1)$
$\mathbf{s}_1 \leftarrow \mathbf{W}_1^b \mathbf{a}_0$
$\mathbf{a}_1 \leftarrow BatchNorm(\mathbf{s}_1, \gamma_1, \beta_1)$
$\mathbf{a}_1^b \leftarrow Sign(\mathbf{a}_1)$

for $k = 2$ to L **do**
  $W_k^b \leftarrow Binarize(W_k)$
  $\mathbf{s}_k \leftarrow \mathbf{W}_k^b \mathbf{a}_{k-1}^b$
  $\mathbf{a}_k \leftarrow BatchNorm(\mathbf{s}_k, \gamma_k, \beta_k)$
  if $k < L$ **then**
    $\mathbf{a}_k^b \leftarrow Sign(\mathbf{a}_k)$
  end if
end for

Computing parameter gradients at k^{th} layer of DNN

- **Aim for back prop:** To update parameter $\mathbf{W}_k$ with $\frac{\partial C}{\partial \mathbf{W}_k}$

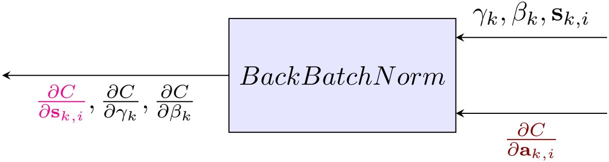
$$\frac{\partial C}{\partial \mathbf{W}_k} = \frac{\partial C}{\partial \mathbf{s}_k} . \frac{\partial \mathbf{s}_k}{\partial \mathbf{W}_k} = \frac{\partial C}{\partial \mathbf{s}_k} . \mathbf{a}_{k-1}^{act}$$

- Calculate $\frac{\partial C}{\partial \mathbf{a}_k^{act}}$ at k^{th} layer

- Other than final layer $\mathbf{a}_k^{act} = \sigma(\mathbf{a}_k)$. From this,

$$\frac{\partial C}{\partial \mathbf{a}_{k,i}} = \frac{\partial C}{\partial \mathbf{a}_{k,i}^{act}} . \frac{\partial \mathbf{a}_{k,i}^{act}}{\partial \mathbf{a}_{k,i}} = \frac{\partial C}{\partial \mathbf{a}_{k,i}^{act}} . \sigma'$$

Other than final layer $\frac{\partial \mathbf{a}_{k,i}^{act}}{\partial \mathbf{a}_{k,i}} = \sigma'$



Back Propagation of DNN

- Compute $\frac{\partial C}{\partial a_L}$

for $k = L$ to 1 **do**

  if $k < L$ **then**

$$\frac{\partial C}{\partial \mathbf{a}_{k,i}} \leftarrow \frac{\partial C}{\partial \mathbf{a}_{k,i}^{act}} \circ \sigma'$$

  end if

$$\left(\frac{\partial C}{\partial \mathbf{s}_{k,i}}, \frac{\partial C}{\partial \gamma_k}, \frac{\partial C}{\partial \beta_k}\right) \leftarrow BackBatchNorm\left(\frac{\partial C}{\partial \mathbf{a}_{k,i}}, \gamma_k, \beta_k, \mathbf{s}_{k,i}\right)$$

$$\frac{\partial C}{\partial \mathbf{a}_{(k-1),i}^{act}} \leftarrow \frac{\partial C}{\partial \mathbf{s}_{k,i}}.\mathbf{W}_k$$

$$\frac{\partial C}{\partial \mathbf{W}_k} \leftarrow \frac{\partial C}{\partial \mathbf{s}_{k,i}}^T.\mathbf{a}_{k-1}^{act}$$

end for

Accumulating the parameter gradients: DNN

- λ is the learning rate decay factor

for $k = 1$ to L **do**

$\gamma_k^{t+1} \leftarrow Update(\gamma_k^t, \eta, \frac{\partial C}{\partial \gamma_k})$

$\beta_k^{t+1} \leftarrow Update(\beta_k^t, \eta, \frac{\partial C}{\partial \beta_k})$

$\mathbf{W}_k^{t+1} \leftarrow Update(W_k^t, \gamma_k \eta, \frac{\partial C}{\partial \mathbf{W}_k})$

$\eta^{t+1} \leftarrow \lambda \eta^t$

end for

Accumulating the parameter gradients: BNN

- λ is the learning rate decay factor

for $k = 1$ to L **do**

$$\gamma_k^{t+1} \leftarrow Update(\gamma_k^t, \eta, \frac{\partial C}{\partial \gamma_k})$$

$$\beta_k^{t+1} \leftarrow Update(\beta_k^t, \eta, \frac{\partial C}{\partial \beta_k})$$

~~$\mathbf{W}_k^{t+1} \leftarrow Update(W_k^t, \gamma_k \eta, \frac{\partial C}{\partial \mathbf{W}_k})$~~

$$\mathbf{W}_k^{t+1} \leftarrow Clip(Update(W_k^t, \gamma_k \eta, \frac{\partial C}{\partial \mathbf{W}_k^b}), -1, 1)$$

$$\eta^{t+1} \leftarrow \lambda \eta^t$$

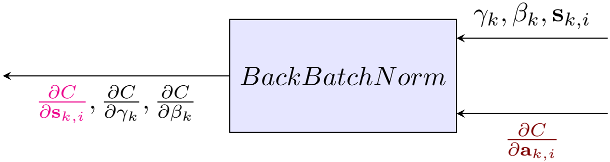
end for

Why is $Clip$ required?

- The real valued weights otherwise would grow very large without any impact on the binary weights

Computing parameter gradients: k^{th} layer of BNN

- To calculate $\frac{\partial C}{\partial \mathbf{W}_k^b}$, we need $\frac{\partial C}{\partial \mathbf{s}_{k,i}}$ and hence $\frac{\partial C}{\partial \mathbf{a}_{k,i}}$



- Calculate $\frac{\partial C}{\partial \mathbf{a}_k^b}$ at k^{th} layer
- Other than final layer $\mathbf{a}_k^b = Sign(\mathbf{a}_k)$. From this,

$$\frac{\partial C}{\partial \mathbf{a}_{k,i}} = \frac{\partial C}{\partial \mathbf{a}_{k,i}^b} . \frac{\partial \mathbf{a}_{k,i}^b}{\partial \mathbf{a}_{k,i}} = \frac{\partial C}{\partial \mathbf{a}_{k,i}^b} . Sign'$$

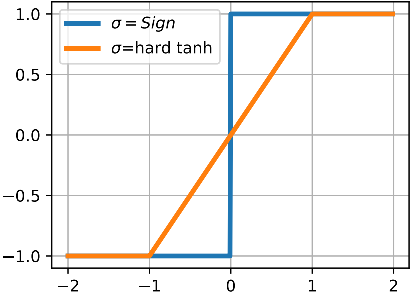
- Other than final layer $\frac{\partial \mathbf{a}_{k,i}^b}{\partial \mathbf{a}_{k,i}} = Sign'$

Difficulties of back prop through binarization

- Derivative of $Sign$, i.e. $Sign'$ is almost zero everywhere which makes $\frac{\partial C}{\partial \mathbf{a}^b_{k,i}}$ also almost zero everywhere.
- Incompatible for backpropagation
- Hinton introduced Straight Through Estimator (STE)

Straight Through Estimator (STE)

Idea of STE is to simply treat the binarization function $Sign$ as if it was a clipped identity function (called hard tanh) during back propagation.



$$\frac{\partial C}{\partial \mathbf{a}_{k,i}} = \frac{\partial C}{\partial \mathbf{a}^b_{k,i}} . Sign'$$
$$= \frac{\partial C}{\partial \mathbf{a}^b_{k,i}} \circ \mathbf{1}_{\left(\left|\frac{\partial C}{\partial \mathbf{a}^b_{k,i}}\right| \leq 1\right)}$$

Back propagation of BNN

- Compute $\frac{\partial C}{\partial a_L}$ **for** $k = L$ to 1 **do**

  if $k < L$ **then**

  $$\frac{\partial C}{\partial \mathbf{a}_{k,i}} = \frac{\partial C}{\partial \mathbf{a}^b_{k,i}} \circ \mathbf{1}_{\left(\left|\frac{\partial C}{\partial \mathbf{a}^b_{k,i}}\right| \leq 1\right)}$$

  end if

  $$\left(\frac{\partial C}{\partial \mathbf{s}_{k,i}}, \frac{\partial C}{\partial \gamma_k}, \frac{\partial C}{\partial \beta_k}\right) \leftarrow BackBatchNorm\left(\frac{\partial C}{\partial \mathbf{a}_{k,i}}, \gamma_k, \beta_k, \mathbf{s}_{k,i}\right)$$

  $$\frac{\partial C}{\partial \mathbf{a}^b_{(k-1),i}} \leftarrow \frac{\partial C}{\partial \mathbf{s}_{k,i}} . \mathbf{W}^b_k$$

  $$\frac{\partial C}{\partial \mathbf{W}^b_k} \leftarrow \frac{\partial C}{\partial \mathbf{s}_{k,i}}^T . \mathbf{a}^b_{k-1}$$

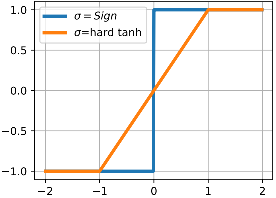
  end for

Back prop through STE

$$g_f = \frac{\partial C}{\partial \mathbf{s}_k}^T . \mathbf{a}_{k-1}^b$$

Identity

$$g_b = \frac{\partial C}{\partial \mathbf{s}_k}^T . \mathbf{a}_{k-1}^b$$



$$\frac{\partial C}{\partial \mathbf{a}_{k-1}^b} = \frac{\partial C}{\partial \mathbf{s}_k} . \mathbf{W}_k^b$$

Mult and batchnorm

Sign

$$\frac{\partial C}{\partial \mathbf{a}_k^b}$$

To previous layer

$$\frac{\partial C}{\partial \mathbf{a}_k} = \frac{\partial C}{\partial \mathbf{a}_{k,i}^b} \circ \mathbf{1}_{|\mathbf{a}_{k,i}| \leq 1}$$

g_f: gradient used to update real var

g_b: gradient used to see change in C for change in binary var

First layer in **Run-Time**

- At first layer, handle continuous valued inputs i.e. each element $a_0^{act}(i)$ of $\mathbf{a}_0^{act}$ as fixed point numbers with m bits of precision during run-time i.e.
-

$$a_0^{act}(i) = \sum_{n=1}^{8} 2^{n-1}[a_0^{act}(i)]^n \quad (1)$$

where $[a_0^{act}(i)]^n$ implies n^{th} bit of $a_0^{act}(i)$ when $a_0^{act}(i)$ is represented in 8-bit precision.

- So, at first layer,

$$s_1 = \sum_{i=1}^{3} \sum_{n=1}^{8} 2^{n-1}([a_0^{act}(i)]^n . w_1^b(i)) \quad (2)$$

Algorithm2: Running a BNN

- Require: A vector of 8 bit input $\mathbf{a}_0$, the binary weight $\mathbf{W}^b$, and the BatchNorm parameters γ, β
- Ensure: the MLP output a_L

$a_1 \leftarrow 0$ First layer

for $n = 1$ to 8 **do**

$\mathbf{a}_1 \leftarrow \mathbf{a}_1 + 2^{n-1} \times \overline{\mathbf{W}_1^b \oplus \mathbf{a}_0^n}$ Following (2)

end for

$\mathbf{a}_1^b \leftarrow Sign(BatchNorm(\mathbf{a}_1, \theta_1))$

for k=2 to L-1 **do** Remaining hidden layers

$\mathbf{a}_k \leftarrow \overline{\mathbf{W}_k^b \oplus \mathbf{a}_{k-1}^b}$

$\mathbf{a}_k^b \leftarrow Sign(BatchNorm(\mathbf{a}_k, \theta_k))$

end for

$\mathbf{a}_L \leftarrow \overline{\mathbf{W}_L^b \oplus \mathbf{a}_{L-1}^b}$ Output layer

$\mathbf{a}_L \leftarrow BatchNorm(\mathbf{a}_L, \theta_L)$

Simulation Results

Simulation Setup of BNN on MNIST

- MNIST consists of a training set of size 60K and test set of 10K 28×28 gray-scale images representing digits ranging from 0 to 9
- BNN consists of
  - 3 hidden layers
  - 4096 binary units
  - L2-SVM output layer instead of softmax
- Square Hinge loss is minimized with Adam
- Exponentially decaying global learning rate
- Batch-normalization with mini-batch size 100

Simulation for MNIST

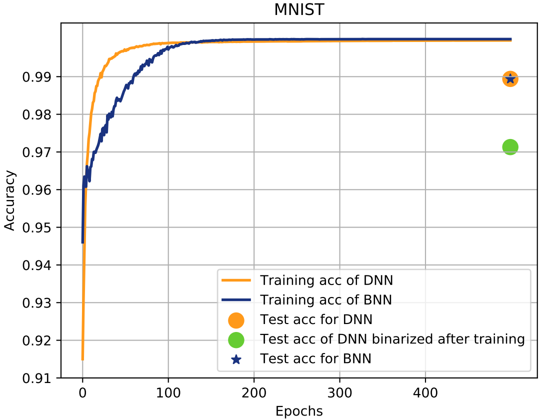


Figure: Comparison of Training and test accuracy of DNN and BNN on MNIST

Simulation Setup of ConvNet on CIFAR10

- CIFAR10 consists of a training set of size 50K and a test set of 10K 32×32 colour images
- ConvNet consists of the following architecture
  $(2 \times 128C_3) - MP_2 - (2 \times 256C_3) - MP_2 - (2 \times 512C_3) - MP_2 - (2 \times 1024FC) - 10SVM$
  where
  - C_3 : 3×3 Binary tanh convolution layer with batchnorm and nonlinearity
  - MP_2: 2×2 max pooling layer with batchnorm and nonlinearity
  - FC: Fully connected layer with batchnorm and nonlinearity
  - SM: Softmax output layer
  - SVM: L2-SVM output layer
- Square Hinge loss is minimized with Adam
- Exponentially decaying global learning rate
- Batch-normalization with mini-batch size 50

Simulation for CIFAR10

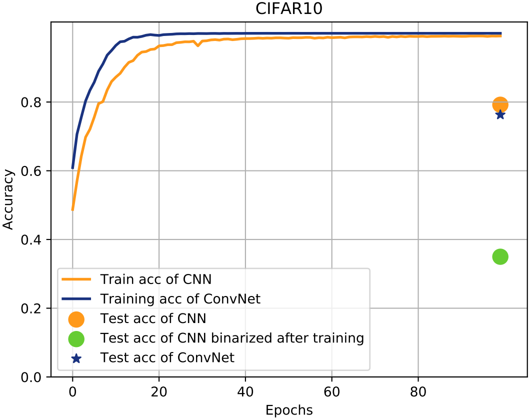


Figure: Comparison of Training and test accuracy of CNN and ConvNet on CIFAR10

Advantages of BNN

Advantages of BNN

- Power efficient in forward pass:
  - Binary weights/activations reduces memory size (32 times compared to single precision floating point (FP) DNN)
  - 1 bit XNOR count instead of 32 bit FP multiply-accumulation operation
  - Exploiting filter repetitions
- Faster on GPU at run time:
  - 32 binary variables can be concatenated into 32-bit registers, thus 32 times speed up on bit-wise operation

In CPU

- Number of trainable parameters N_{param}
- Number of bits used to save the parameters N_{bits}
- MNIST

Network	N_{param}	N_{bits}
DNN	36843530	$36843530 \times 32 \approx 1.17$ Bn
BNN	36843550	$36843550 \approx 36$ Mn

- CIFAR10

Network	N_{param}	N_{bits}
CNN	14033546	$14033546 \times 32 \approx 449$ Mn
ConvNet	14033566	$14033566 \approx 14$ Mn

RBNN to close the accuracy gap between DNN and BNN

- **Why RBNN?**: Reason for the accuracy gap is large quantization error

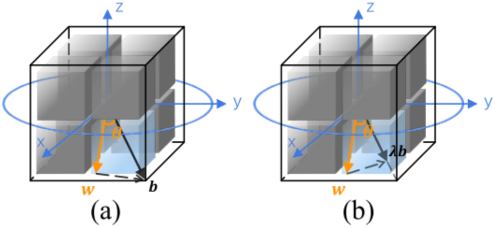


Figure: (a) Large quantization error is caused by: **Norm gap** and **Angular bias** (b) **Norm gap** is solved by $\min_{\lambda,\mathbf{b}} ||\lambda\mathbf{b} - \mathbf{w}||^2$. However, it cannot reduce the angular bias θ i.e. quantization error $||\mathbf{w}\sin\theta||^2$ is still high for a higher θ.

- RBNN to reduce the **angular bias** between the real weights $\mathbf{w}^i$ and its binarized value $\mathbf{b}_w^i$

Possible area of extension: Approximate Computing (AC)

- Strategies for AC:
  - Approximating circuits i.e. adders, multipliers and other logical circuits
  - Approximating storage eg. using precision scaling
  - Software level approximation: Using loop perforation, Skipping tasks and memory accesses Accelerating NN
  - Approximating neural networks
- Existing ways to Approximate NN and improve the accuracy further
  - Approximate deep network by shallow network, pruning, quantization, binarization
  - Designing compact layers (eg. replacing 3×3 convolution with 1×1 convolution)
  - FP weights as linear combination of binary weight bases and using multiple binary activations
  - Defining a new optimizer called BOP and use it instead of Adam
- Idea for extension: Exploring ways to approximate a NN efficiently

Computing parameter gradients: last layer of DNN

In forward-prop, at final layer we had,

$$\mathbf{s}_3 \leftarrow \mathbf{W}_3 \mathbf{a}_2^{act}$$
$$\hat{\mathbf{s}}_3 \leftarrow \frac{\mathbf{s}_3 - \mu_3}{\sqrt{\sigma_3^2 + \epsilon}}$$
$$\mathbf{a}_3 \leftarrow \gamma_3 \hat{\mathbf{s}}_3 + \beta_3$$
$$\mathbf{a}_3^{act} \leftarrow \mathbf{a}_3$$

Remembering inputs in a minibatch of size m

$$\mu_3 = \frac{1}{m} \sum_{i=1}^{m} \mathbf{s}_{3,i}$$
$$\sigma_3 = \frac{1}{m} \sum_{i=1}^{m} (\mathbf{s}_{3,i} - \mu_3)^2$$

- Calculate $\frac{\partial C}{\partial \mathbf{a}_3^{act}}$ at final layer

Computing parameter gradients: last layer of DNN

In forward-prop, at final layer we had,

$$\mathbf{s}_3 \leftarrow \mathbf{W}_3 \mathbf{a}_2^{act}$$
$$\hat{\mathbf{s}}_3 \leftarrow \frac{\mathbf{s}_3 - \mu_3}{\sqrt{\sigma_3^2 + \epsilon}}$$
$$\mathbf{a}_3 \leftarrow \gamma_3 \hat{\mathbf{s}}_3 + \beta_3$$
$$\mathbf{a}_3^{act} \leftarrow \mathbf{a}_3$$

Remembering inputs in a minibatch of size m

$$\mu_3 = \frac{1}{m} \sum_{i=1}^{m} \mathbf{s}_{3,i}$$
$$\sigma_3 = \frac{1}{m} \sum_{i=1}^{m} (\mathbf{s}_{3,i} - \mu_3)^2$$

- Calculate $\frac{\partial C}{\partial \mathbf{a}_3^{act}}$ at final layer
- **To update parameter $\mathbf{W}_3$ need to find $\frac{\partial C}{\partial \mathbf{W}_3}$**

$$\frac{\partial C}{\partial \mathbf{W}_3} = \frac{\partial C}{\partial \mathbf{s}_3} . \frac{\partial \mathbf{s}_3}{\partial \mathbf{W}_3} = \frac{\partial C}{\partial \mathbf{s}_3} . \mathbf{a}_2^{act}$$

Computing parameter gradients: last layer of DNN

In forward-prop, at final layer we had,

$$\mathbf{s}_3 \leftarrow \mathbf{W}_3 \mathbf{a}_2^{act}$$

$$\hat{\mathbf{s}}_3 \leftarrow \frac{\mathbf{s}_3 - \mu_3}{\sqrt{\sigma_3^2 + \epsilon}}$$

$$\mathbf{a}_3 \leftarrow \gamma_3 \hat{\mathbf{s}}_3 + \beta_3$$

$$\mathbf{a}_3^{act} \leftarrow \mathbf{a}_3$$

Remembering inputs in a minibatch of size m

$$\mu_3 = \frac{1}{m} \sum_{i=1}^{m} \mathbf{s}_{3,i}$$

$$\sigma_3 = \frac{1}{m} \sum_{i=1}^{m} (\mathbf{s}_{3,i} - \mu_3)^2$$

- Calculate $\frac{\partial C}{\partial \mathbf{a}_3^{act}}$ at final layer
- **To update parameter $\mathbf{W}_3$ need to find $\frac{\partial C}{\partial \mathbf{W}_3}$**

$$\frac{\partial C}{\partial \mathbf{W}_3} = \frac{\partial C}{\partial \mathbf{s}_3} . \frac{\partial \mathbf{s}_3}{\partial \mathbf{W}_3} = \frac{\partial C}{\partial \mathbf{s}_3} . \mathbf{a}_2^{act}$$

- Now

$$\frac{\partial C}{\partial \mathbf{s}_3} = \frac{\partial C}{\partial \hat{\mathbf{s}}_3} . \frac{\partial \hat{\mathbf{s}}_3}{\partial \mathbf{s}_3} + \frac{\partial C}{\partial \mu_3} . \frac{\partial \mu_3}{\partial \mathbf{s}_3} + \frac{\partial C}{\partial \sigma_3^2} . \frac{\partial \sigma_3^2}{\partial \mathbf{s}_3}$$

$$= \frac{\partial C}{\partial \hat{\mathbf{s}}_3} . \frac{1}{\sqrt{\sigma_3^2 + \epsilon}} + \frac{\partial C}{\partial \mu_3} . \frac{1}{m} + \frac{\partial C}{\partial \sigma_3^2} . \frac{1}{m} 2(\mathbf{s}_3 - \mu_3)$$

Computing parameter gradients: last layer of DNN

In forward-prop, at final layer we had,

$$\mathbf{s}_3 \leftarrow \mathbf{W}_3 \mathbf{a}_2^{act}$$

$$\hat{\mathbf{s}}_3 \leftarrow \frac{\mathbf{s}_3 - \mu_3}{\sqrt{\sigma_3^2 + \epsilon}}$$

$$\mathbf{a}_3 \leftarrow \gamma_3 \hat{\mathbf{s}}_3 + \beta_3$$

$$\mathbf{a}_3^{act} \leftarrow \mathbf{a}_3$$

Remembering inputs in a minibatch of size m

$$\mu_3 = \frac{1}{m} \sum_{i=1}^{m} \mathbf{s}_{3,i}$$

$$\sigma_3 = \frac{1}{m} \sum_{i=1}^{m} (\mathbf{s}_{3,i} - \mu_3)^2$$

- Calculate $\frac{\partial C}{\partial \mathbf{a}_3^{act}}$ at final layer
- **To update parameter $\mathbf{W}_3$ need to find $\frac{\partial C}{\partial \mathbf{W}_3}$**

$$\frac{\partial C}{\partial \mathbf{W}_3} = \frac{\partial C}{\partial \mathbf{s}_3} . \frac{\partial \mathbf{s}_3}{\partial \mathbf{W}_3} = \frac{\partial C}{\partial \mathbf{s}_3} . \mathbf{a}_2^{act}$$

- Now

$$\frac{\partial C}{\partial \mathbf{s}_3} = \frac{\partial C}{\partial \hat{\mathbf{s}}_3} . \frac{\partial \hat{\mathbf{s}}_3}{\partial \mathbf{s}_3} + \frac{\partial C}{\partial \mu_3} . \frac{\partial \mu_3}{\partial \mathbf{s}_3} + \frac{\partial C}{\partial \sigma_3^2} . \frac{\partial \sigma_3^2}{\partial \mathbf{s}_3}$$

$$= \frac{\partial C}{\partial \hat{\mathbf{s}}_3} . \frac{1}{\sqrt{\sigma_3^2 + \epsilon}} + \frac{\partial C}{\partial \mu_3} . \frac{1}{m} + \frac{\partial C}{\partial \sigma_3^2} . \frac{1}{m} 2(\mathbf{s}_3 - \mu_3)$$

- Next need to calculate $\frac{\partial C}{\partial \hat{\mathbf{s}}_3}$, $\frac{\partial C}{\partial \mu_3}$ and $\frac{\partial C}{\partial \sigma_3^2}$

Computing parameter gradients: last layer of DNN

In forward-prop, at final layer we had,

$$\mathbf{s}_3 \leftarrow \mathbf{W}_3\mathbf{a}_2^{act}$$
$$\hat{\mathbf{s}}_3 \leftarrow \frac{\mathbf{s}_3 - \mu_3}{\sqrt{\sigma_3^2 + \epsilon}}$$
$$\mathbf{a}_3 \leftarrow \gamma_3\hat{\mathbf{s}}_3 + \beta_3$$
$$\mathbf{a}_3^{act} \leftarrow \mathbf{a}_3$$

Remembering inputs in a minibatch of size m

$$\mu_3 = \frac{1}{m}\sum_{i=1}^{m}\mathbf{s}_{3,i}$$
$$\sigma_3 = \frac{1}{m}\sum_{i=1}^{m}(\mathbf{s}_{3,i} - \mu_3)^2$$

- Calculate $\frac{\partial C}{\partial \hat{\mathbf{s}}_3}$, $\frac{\partial C}{\partial \mu_3}$ and $\frac{\partial C}{\partial \sigma_3^2}$

Computing parameter gradients: last layer of DNN

In forward-prop, at final layer we had,

$$\mathbf{s}_3 \leftarrow \mathbf{W}_3 \mathbf{a}_2^{act}$$
$$\hat{\mathbf{s}}_3 \leftarrow \frac{\mathbf{s}_3 - \mu_3}{\sqrt{\sigma_3^2 + \epsilon}}$$
$$\mathbf{a}_3 \leftarrow \gamma_3 \hat{\mathbf{s}}_3 + \beta_3$$
$$\mathbf{a}_3^{act} \leftarrow \mathbf{a}_3$$

Remembering inputs in a minibatch of size m

$$\mu_3 = \frac{1}{m} \sum_{i=1}^{m} \mathbf{s}_{3,i}$$
$$\sigma_3 = \frac{1}{m} \sum_{i=1}^{m} (\mathbf{s}_{3,i} - \mu_3)^2$$

- Calculate $\frac{\partial C}{\partial \hat{\mathbf{s}}_3}$, $\frac{\partial C}{\partial \mu_3}$ and $\frac{\partial C}{\partial \sigma_3^2}$
-

$$\frac{\partial C}{\partial \hat{\mathbf{s}}_3} = \frac{\partial C}{\partial \mathbf{a}_3} \cdot \frac{\partial \mathbf{a}_3}{\partial \hat{\mathbf{s}}_3} = \frac{\partial C}{\partial \mathbf{a}_3} \cdot \gamma_3 = \frac{\partial C}{\partial \mathbf{a}_3^{act}} \cdot \gamma_3$$

Computing parameter gradients: last layer of DNN

In forward-prop, at final layer we had,

$$\mathbf{s}_3 \leftarrow \mathbf{W}_3 \mathbf{a}_2^{act}$$
$$\hat{\mathbf{s}}_3 \leftarrow \frac{\mathbf{s}_3 - \mu_3}{\sqrt{\sigma_3^2 + \epsilon}}$$
$$\mathbf{a}_3 \leftarrow \gamma_3 \hat{\mathbf{s}}_3 + \beta_3$$
$$\mathbf{a}_3^{act} \leftarrow \mathbf{a}_3$$

Remembering inputs in a minibatch of size m

$$\mu_3 = \frac{1}{m} \sum_{i=1}^{m} \mathbf{s}_{3,i}$$
$$\sigma_3 = \frac{1}{m} \sum_{i=1}^{m} (\mathbf{s}_{3,i} - \mu_3)^2$$

- Calculate $\frac{\partial C}{\partial \hat{\mathbf{s}}_3}$, $\frac{\partial C}{\partial \mu_3}$ and $\frac{\partial C}{\partial \sigma_3^2}$
- $$\frac{\partial C}{\partial \hat{\mathbf{s}}_3} = \frac{\partial C}{\partial \mathbf{a}_3} \cdot \frac{\partial \mathbf{a}_3}{\partial \hat{\mathbf{s}}_3} = \frac{\partial C}{\partial \mathbf{a}_3} \cdot \gamma_3 = \frac{\partial C}{\partial \mathbf{a}_3^{act}} \cdot \gamma_3$$
- $$\frac{\partial C}{\partial \sigma_3^2} = \sum_{i=1}^{m} \frac{\partial C}{\partial \hat{\mathbf{s}}_{3,i}} \frac{\partial \hat{\mathbf{s}}_{3,i}}{\partial \sigma_3^2} = \sum_{i=1}^{m} \frac{\partial C}{\partial \hat{\mathbf{s}}_{3,i}} (\mathbf{s}_{3,i} - \mu_3) \frac{-1}{2} (\sigma_3^2 + \epsilon)^{-3/2}$$

Computing parameter gradients: last layer of DNN

In forward-prop, at final layer we had,

$$\mathbf{s}_3 \leftarrow \mathbf{W}_3 \mathbf{a}_2^{act}$$

$$\hat{\mathbf{s}}_3 \leftarrow \frac{\mathbf{s}_3 - \mu_3}{\sqrt{\sigma_3^2 + \epsilon}}$$

$$\mathbf{a}_3 \leftarrow \gamma_3 \hat{\mathbf{s}}_3 + \beta_3$$

$$\mathbf{a}_3^{act} \leftarrow \mathbf{a}_3$$

Remembering inputs in a minibatch of size m

$$\mu_3 = \frac{1}{m} \sum_{i=1}^{m} \mathbf{s}_{3,i}$$

$$\sigma_3 = \frac{1}{m} \sum_{i=1}^{m} (\mathbf{s}_{3,i} - \mu_3)^2$$

$$\frac{\partial C}{\partial \mu_3} = \frac{\partial C}{\partial \sigma_3^2} \cdot \frac{\partial \sigma_3^2}{\partial \mu_3} + \sum_{i=1}^{m} \frac{\partial C}{\partial \hat{\mathbf{s}}_{3,i}} \cdot \frac{\partial \hat{\mathbf{s}}_{3,i}}{\partial \mu_3}$$

$$= \frac{\partial C}{\partial \sigma_3^2} \cdot \frac{1}{m} \sum_{i=1}^{m} -2(\mathbf{s}_{3,i} - \mu_3) + \sum_{i=1}^{m} \frac{\partial C}{\partial \hat{\mathbf{s}}_{3,i}} \cdot \frac{-1}{\sqrt{\sigma_3^2 + \epsilon}}$$

Computing parameter gradients: last layer of DNN

In forward-prop, at final layer we had,

$$\mathbf{s}_3 \leftarrow \mathbf{W}_3 \mathbf{a}_2^{act}$$
$$\hat{\mathbf{s}}_3 \leftarrow \frac{\mathbf{s}_3 - \mu_3}{\sqrt{\sigma_3^2 + \epsilon}}$$
$$\mathbf{a}_3 \leftarrow \gamma_3 \hat{\mathbf{s}}_3 + \beta_3$$
$$\mathbf{a}_3^{act} \leftarrow \mathbf{a}_3$$

Remembering inputs in a minibatch of size m

$$\mu_3 = \frac{1}{m} \sum_{i=1}^{m} \mathbf{s}_{3,i}$$
$$\sigma_3 = \frac{1}{m} \sum_{i=1}^{m} (\mathbf{s}_{3,i} - \mu_3)^2$$

- Gradients of parameters γ_3 and β_3 can also be found as,

$$\frac{\partial C}{\partial \gamma_3} = \sum_{i=1}^{m} \frac{\partial C}{\partial \mathbf{a}_{3,i}} . \frac{\partial \mathbf{a}_{3,i}}{\partial \gamma_3} = \sum_{i=1}^{m} \frac{\partial C}{\partial \mathbf{a}_{3,i}} . \hat{\mathbf{s}}_{3,i}$$
$$\frac{\partial C}{\partial \beta_3} = \sum_{i=1}^{m} \frac{\partial C}{\partial \mathbf{a}_{3,i}}$$

Idea behind Shift based Calculations

$$AP2(x) = sign(x) \times 2^{round(log_2|x|)}$$

- $7 * 5 = 7 * AP2(5) = 7 * 4 = 1.(2^2) + 1.(2^1) + 1.(2^0) * 2^2$
- Two left shift of 111_2 gives $11100_2 = 28_{10}$

Algorithm3 : Shift based $BatchNorm$

- Applicable to activation(x) over a minibatch
- $\ll$ represents left and $\gg$ represents right binary shift
- Require: values of x over a minibatch: $B = \{x_1, x_2, \ldots x_m\}$, parameters to be learned γ, β
- Ensure: $\{y_i = BN(x_i, \gamma, \beta)\}$

$$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \text{ \{mini batch mean\}}$$

$$C(x_i) \leftarrow (x_i - \mu_B) \text{ \{Centered input\}}$$

$$\sigma_b^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(C(x_i) \ll AP2(C(x_i))) \text{ \{apx variance\}}$$

$$\hat{x}_i \leftarrow C(x_i) \ll AP2((\sqrt{\sigma_b^2 + \epsilon})^{-1}) \text{ \{normalize\}}$$

$$y_i \leftarrow AP2(\gamma) \ll \hat{x}_i \text{ \{scale and shift\}}$$

Algorithm4 for *Update*: Shift based Adamax

- g_t^2 indicates element-wise square $g_t \circ g_t$
- Default settings $\alpha = 2^{-10}, 1 - \beta_1 = 2^{-3}, 1 - \beta_2 = 2^{-10}$
- β_1^t and β_2^t denotes β_1 and β_2 to the power t

- Require: previous parameters θ_{t-1} and their gradients g_t and learning rate α
- Ensure: Updated parameter θ_t

$$m_t \leftarrow \beta_1 . m_{t-1} + (1 - \beta_1) . g_t$$

Momentum based GD

$$v_t \leftarrow max(\beta_2 . v_{t-1}, |g_t|)$$

RMSprop with infinity norm

{Updated parameters}

$$\theta_t \leftarrow \theta_{t-1} - (\alpha \gg (1 - \beta_1)) . \hat{m} \ll v_t^{-1}$$

Shift based Update combining momentum based GD and RMSprop

Why to save both real and binary weights?

- Binary weights and activations (eg. $\mathbf{W}_2^b$, $\mathbf{a}_1^b$) are used to compute the parameter gradients
- Real valued gradients are accumulated in real valued variables
- Can BNN work by updating binary weights?

  No, Real valued weights are likely required to be updated for SGD to work because SGD explores the parameter space in small and noisy steps, and the noise is averaged out by the stochastic gradient contribution accumulated at each epoch

Can BNN be used for tasks other than classification?

- Yes, just the constraint functions change in SVM.
- Softmax and cross entropy is majorly used for classification task in current literature.
- To avoid arithmetic calculation involving exponential, the authors took L2SVM as final layer.
- L2SVM calculates Square Hinge Loss (SHL) at output layer. Idea behind minimizing SHL is maximizing the separation between the hyperplane and the closest data point
- To get an idea of SHL in binary classification (two classes $a_{3,i}^{true} \in \{+1, -1\}$) problem let's consider, training data and its corresponding label $(\mathbf{a}_{2,i}^{b}, a_{3,i}^{true})$, $i = 1, 2, \ldots k$, $\mathbf{a}_{2,i}^{b} \in \mathbb{R}^{d}$, L2SVM learning consists of

$$\min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + \sum_{i=1}^{k} max(1 - \mathbf{w}^T\mathbf{a}_{2,i}^{b}.a_{3,i}^{true}, 0)^2,$$

Talk about Binary-DetNet here

- Fully connect architecture cannot learn in variable channel condition