Week1 &2 lately report

Qian Jiang

1. Finished reading all papers about Xnor-Net & Summary of quantized network
2. Finished studying Neural Network related knowledge including perceptron, backpropagation, CNN, BNN...

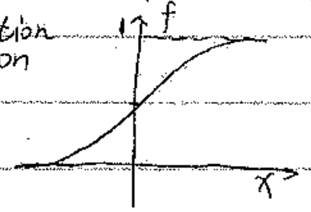
   (Notes are attched below)
3. Understood codes of XNOR-Net/util.py
4. Run codes (met lots of bugs, thanks to Weitian’s help)

Neural Network Basics

1. M-P Model

$x_1 \xrightarrow{w_1} \Sigma \rightarrow f \rightarrow$ output : $y = f\left(\sum_{i=1}^{n} w_i x_i - \theta\right)$

Activation function



$f = \frac{1}{1+e^{-x}}$ sigmoid function

2. Perceptron

$x_1 \xrightarrow{w_1} \Sigma \cdots + b \rightarrow f \rightarrow$ output : $y = f\left(\sum w_i x_i + b\right)$

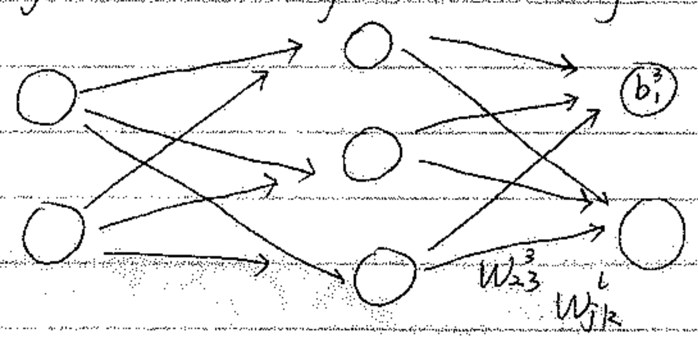
(for : linear seperation)

$\Downarrow$

Mulit-layer Perceptron.

3. Back Propagation

layer 1　　layer 2　　layer 3



$a_j^l = \sigma\left(\sum_k W_{jk}^l a_k^{l-1} + b_j^l\right)$

$a^l = \sigma(\underbrace{W^l a^{l-1} + b^l})$

Z^l: weighted input

Back-info:

1. $S \odot t = S_j t_j$.

eg. $\begin{pmatrix}1\\2\end{pmatrix} \odot \begin{pmatrix}3\\4\end{pmatrix} = \begin{pmatrix}3\\8\end{pmatrix}$

Hadamard Product.

2. $\nabla_a C$: matrix for $\frac{\partial C}{\partial a_j^l}$

Cost function (quadratic) : $C = \frac{1}{2n} \sum_x |y(x) - a^L(x)|^2$

$C_x = \frac{1}{2} |y - a|^2$

Equations of BP

① $\delta^L = \nabla_a C \odot \sigma'(Z^L)$　(L: Last)

$$\left(\begin{aligned} &\delta_j^L = \frac{\partial C}{\partial Z_j^L} = \frac{\partial C}{\partial a_j^L} \cdot \sigma'(Z_j^L) \\ &\delta^L = \nabla_a C \odot \sigma'(Z^L) \end{aligned} \right)$$

☆ $Z^{l+1} = f(Z^l)$ chain rule

② $\delta^l = \left((W^{l+1})^T \cdot \delta^{l+1}\right) \odot \sigma'(Z^l)$

$$\left\{ \begin{aligned} &\delta_j^l = \frac{\partial C}{\partial Z_j^l} = \sum_k \frac{\partial C}{\partial Z_k^{l+1}} \cdot \frac{\partial Z_k^{l+1}}{\partial Z_j^l} \\ &Z_k^{l+1} = \sum_i W_{ki}^{l+1} a_i^l + b_k^{l+1} = \sum_i W_{ki}^{l+1} \sigma(Z_i^l) + b_k^{l+1} \\ &\frac{\partial Z_k^{l+1}}{\partial Z_j^l} = W_{kj}^{l+1} \sigma'(Z_j^l) \end{aligned} \right.$$

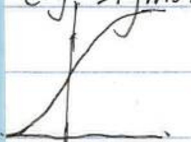
③ $\frac{\partial C}{\partial b_j^l} = \delta_j^l$

④ $\frac{\partial C}{\partial W_{jk}^l} = a_k^{l-1} \delta_j^l$

$$(z^l = w^l a^{l-1} + b^l)$$

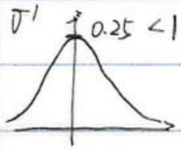
gradient vanishing:

e.g. sigmoid



⇓ derivation

σ' 0.25 < 1



$\frac{\partial C}{\partial b_i} = \frac{\partial C}{\partial a_L} \cdot \frac{\partial a_L}{\partial \cdots} \cdots$

⇓

smaller → vanish

gradient exploding:

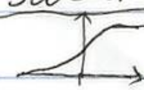
$\sigma' w > 1$

for ③ $\frac{\partial C}{\partial b_j^l} = \delta_j^l$ $\qquad \frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} \cdot \underbrace{\frac{\partial z_j^l}{\partial b_j^l}}_{1} = \frac{\partial C}{\partial z_j^l} = \delta_j^l$

for ④ $\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$ $\qquad \frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial w_{jk}^l}$

Also $\frac{\partial C}{\partial w} = a_{in} \delta_{out}$ $\qquad = \delta_j^l \cdot a_k^{l-1}$

☆ when $a_{in} \to 0$ or $\underbrace{\text{output is saturated}}_{\delta_{out} \to 0}$ (high or low)



Matrix Form:

$$\delta^L = \Sigma'(z^L) \nabla_a C \qquad \Sigma'(z^L) : \begin{pmatrix} \sigma'(z_1^L) & 0 & \cdots \\ 0 & \sigma'(z_2^L) & \cdots \\ \vdots & & \ddots \end{pmatrix}$$

$$\delta^l = \Sigma'(z^l)(w^{l+1})^T \delta^{l+1}$$

BP Algorithm:

1. Input x
2. Feedforward : $z^l = w^l a^{l-1} + b^l$, $a^l = \sigma(z^l)$
3. Output error : $\delta^L = \nabla_a C \cdot \sigma'(z^L)$
4. Backpropagate the error : $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$
5. Output : $\frac{\partial C}{\partial b_j^l} = \delta_j^l \qquad \frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$

Gradient descent:

update: $w^l \to w^l - \frac{\eta}{m} \delta^l (a^{l-1})^T$ $\qquad$ m: batchsize

$\qquad b^l \to b^l - \frac{\eta}{m} \delta^l$ $\qquad$ η: learning rate

$$\Delta C = \frac{\partial C}{\partial a_m^L} \cdot \frac{\partial a_m^L}{\partial a_n^{L-1}} \cdot \frac{\partial a_n^{L-1}}{\partial a_p^{L-2}} \cdots \frac{\partial a_j^l}{\partial w_{jk}^l} \cdot \Delta w_{jk}^l$$

↓

Σ all possible path

Convolutional Neural Networks (Stanford CS231n)

CONV → Relu → pool – Fully-Connected

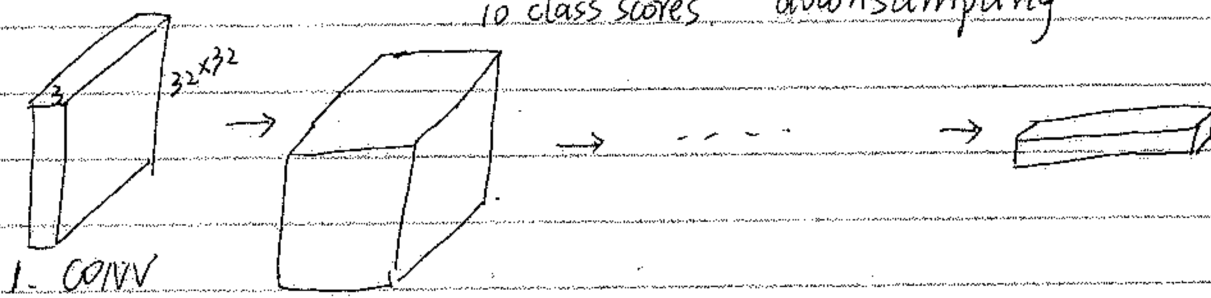
e.g. Input $[32\times32\times3]$ ⇒ CONV $[32\times32\times12]$ ⇒ RELU $[32\times32\times12]$

image size, 3 color channel; 12 filter; activation function applied

⇓

FC $[1\times1\times10]$ ⇐ POOL $[16\times16\times12]$

10 class scores; downsampling



Parameter Sharing: Filter remain same as input slide. $F\cdot F\cdot D_1$ weights per filter

1. CONV

depth: number of filters

stride: move the filter S pixel at a time

zero-padding: zeros around the input boarder

W: input volume size

F: filter size

P: amount of zero padding

S: stride

K: Number of filters

input $W_1 \times H_1 \times D_1$

output $W_2 = (W_1 - F + 2P)/S + 1$

$H_2 = (H_1 - F + 2P)/S + 1$

$D_2 = K$

e.g. $4\times4\times1$ (pad 1)

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 3 & 4 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} * \underbrace{\begin{matrix} 1 & 2 \\ 3 & 4 \end{matrix}}_{\text{Filter } 2\times2\times1} \xrightarrow{S=2} \begin{pmatrix} 4 & 6 \\ 6 & 4 \end{pmatrix}$$

$b_0 : 0$

$b_0 + 0\times1 + 3\times2 + 0\times3 + 0\times4 = 6$

2. Relu 

3. Pooling: $W_1 \times H_1 \times D_1$

F: spatial extent $\quad W_2 = (W_1 - F)/S + 1$

S: stride. $\quad H_2 = (H_1 - F)/S + 1$

$D_2 = D_1$

e.g.
$$\begin{matrix} 1 & 1 & 2 & 4 \\ 5 & 6 & 7 & 8 \\ 3 & 2 & 1 & 0 \\ 1 & 2 & 3 & 4 \end{matrix} \xrightarrow[\text{2x2 filter, stride 2}]{\text{max pool}} \begin{matrix} 6 & 8 \\ 3 & 4 \end{matrix}$$

Back propagation .....

4. (Normalization layer : optional)

5. Fully - Conneted Layer:

a layer have full connections to all activations in the previous layer.

(similar to normal Neulal Networks)

Some common CNN:

LeNet. AlexNet. ZF Net. GooLeNet. VGGNet. ResNet.

Paper Notes : CVPR 16 XNOR-Net : ImageNet Class--

XNOR-Net : ImageNet Classification ---- (CVPR 16)

Ab : BNN : weight binarized. 32 × memory saving

XNOR-Net : both input & weight binarized , FC binarize

58 × faster 32× memory saving

run networks on CPUs

classification accurancy : BWN of Alexnet is same as full-precision Al

compare with BinaryConnect , BinaryNets.

16% accurancy (top-1)

1. Intro

	~~speed~~	operation	Mem-saving	Computation saving	Accu (Alexn
standard	~~1×~~	+ . − . ×	1×	1×	56.7
BW	~~2×~~	+ . −	~32×	~2×	56.8
BW . BI		XNOR . bitcount	~32X	~58X	44.2

2. Related Work

1. shallow networks:

estimating a deep NN with a shallower model

pros : network size ↓

cons : difficult to train with large number of paramet

2. Compressing pre-trained deep networks :

pruning redundant non-informative weights

remove redundant weights . Huffman coding

HashNets . Matrix factorization

3. Quantizing parameters :

quantize the weights of FC layer by vector quantization

------

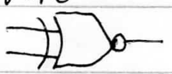
4. Network binarization :

binarize weights & activations

top-1 score : the top class (highest proba) is the same as the target label

top-5 score : the target label is one of the top 5 predictions.

XNOR :



a	b	f
0	0	1
0	1	0
1	0	0
1	1	1

loss function = cost function

Hash function : map data to fixed size

rgmin.
g max :
g : argument
gmin $F(x,y)$
$x \cdot y$ when
is min.

$* =$

optimal.

r $q = \mathrm{Sign}(r)$
$r = g_q 1_{|r| \le 1}$
an be seen
propagating
e gradient through
ard tanh
$\Downarrow$
tanh(x)
Clip(x, -1, 1)
max(-1, min(1, x))

3. Binary CNN ① BWN

I_l : input tensor for the lth layer

W_{lk} : the k-th weight filter in the l^{th} layer.

K^l : the number of weight filter in the l^{th} layer.

$I \in R^{c \times w_{in} \times h_{in}}$ (C : channel)

$W \in R^{c \times w \times h}$

Binary filter $B \in \{+1, -1\}^{c \times w \times h}$

scaling factor α : $W \approx \alpha B$

$$I * W = (I \oplus B)\alpha \qquad (1)$$

↳ convolution without multiplication

$$W_{lk} = \underbrace{A_{lk}}_{\text{scalar}} \underbrace{B_{lk}}_{\text{binary tensor}}$$

---

optimize $W \approx \alpha B$:

$$J(B, \alpha) = \| W - \alpha B \|^2 \qquad (2)$$

$$= \alpha^2 \underbrace{B^T B}_{\text{constant } n} - 2\alpha W^T B + \underbrace{W^T W}_{\text{constant } C} \qquad (3)$$

$$= \alpha^2 n - 2\alpha W^T B + C$$

$$B^* = \arg\max_B \{W^T B\} \Rightarrow B_i = \begin{cases} +1 & \text{if } W_i \ge 0 \\ -1 & \text{if } W_i < 0 \end{cases} \qquad (4)$$

$$\therefore B^* = \mathrm{Sign}(W)$$

$$\frac{\partial J}{\partial \alpha} = 2\alpha n - 2W^T B = 0 \Rightarrow \alpha^* = \frac{W^T B^*}{n} \qquad (5)$$

$$\therefore \alpha^* = \frac{W^T \mathrm{sign}(W)}{n} = \frac{\Sigma |W_i|}{n} = \frac{1}{n} \|W\|_l \qquad (6)$$

---

train BWN:

for sign(r) $\frac{\partial \mathrm{sign}}{\partial r} = r 1_{|r| \le 1}$

$$\frac{\partial C}{\partial W_i} = \frac{\partial C}{\partial \widetilde{W_i}} \left(\frac{1}{n} + \frac{\partial \mathrm{sign}}{\partial W_i} \alpha \right) \ ?$$

Key: ① only binarize the weights during forward & ~~bP~~ bP
② for update, use the high precision (real-value) weights

bitcount: count the number of (1) set bits in a string

② XNOR - Network

Key: 1. binarize both weights & inputs

2. convolution can be implemented by XNOR & bit

Binary dot product:

$$X^T W \approx \beta H^T \alpha B \ , \ H, B \in \{+1, -1\}^n \ , \ \beta, \alpha \in R^+$$

optimize: $\alpha^*, B^*, \beta^*, H^* = \underset{\alpha, B, \beta, H}{\operatorname{argmin}} \|X \odot W - \beta \alpha H \odot B\|$

$$Y \in R^n, \ Y_i = X_i W_i$$

$$C \in \{+1, -1\}^n \ , \ C_i = H_i B_i$$

$$\gamma \in R^+ \ , \ \gamma = \beta \alpha$$

$$\therefore \ \gamma^*, C^* = \underset{\gamma, C}{\operatorname{argmin}} \| Y - \gamma C \|$$

$$C^* = \operatorname{sign}(Y) = \operatorname{sign}(X) \odot \operatorname{sign}(W) = H^* \odot B^*$$

$$\because \ E[|Y_i|] = E[|X_i||W_i|] = E[|X_i|] E[|W_i|]$$

$$\therefore \ \gamma^* = \frac{\sum |Y_i|}{n} = \frac{\cancel{\sum |W|} \sum |X_i||W_i|}{n}$$

$$\approx \left(\frac{1}{n}\|X\|_{L_1}\right)\left(\frac{1}{n}\|W\|_{L_2}\right) = \beta^* \alpha^*$$

Binary convolution:

$$A = \frac{\sum |I_{:,:,i}|}{c} \qquad K = A * k$$

$\hookrightarrow$ 2D filter, $\forall k_{ij} = \frac{1}{w \times h}$

K_{ij} correspond to β for a sub-tensor centered at ij

$$\therefore \ I * W \approx (\operatorname{sign}(I) \circledast \operatorname{sign}(W)) \odot K \alpha$$

$\hookrightarrow$ bitcount & XNOR operation

Train XNOR-Net:

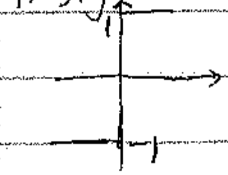
BNorm $\to$ BinActiv $\to$ BinConv $\to$ pool

Gradient: binarize g^{in} in backpass.

use $\max_i(|g_i^{in}|)$ as scaling factor

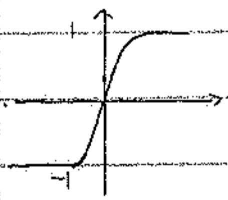
Common A - function.

1. Sign



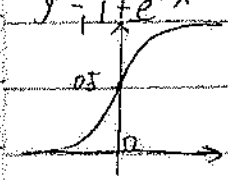
2. tanh

$y = \frac{e^x - e^{-x}}{e^x + e^{-x}}$



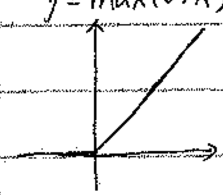
3. Sigmoid

$y = \frac{1}{1+e^{-x}}$



4. Relu

$y = \max(0, x)$



Batch Norm:

shifting inputs to: zero-mean

unit variance

Quantized Model (from paperweekly.blog)

1. Deep Compression

Pruning : — Quantization — Huffman Encoding

less number of weights   less bits per weight

2. Binary-Net

- Quantized Neural Networks:
  Training Neural Networks with low precision weights & activations
- Binarized Neural Networks:
  Training deep Neural Networks with W & A constrained to ±1
- XNOR-Net : ImageNet classification using binary convolutional Neural Networks

~~BNN~~ Binarization :

Sign

① $x^b = \mathrm{sign}(x) = \begin{cases} +1 & \text{if } x > 0 \\ -1 & \text{otherwise} \end{cases}$

② $x^b = \mathrm{sign}(x) = \begin{cases} +1 & \sigma(x) \\ -1 & 1-p \end{cases}$ probability

$\sigma(x) = \mathrm{clip}\left(\frac{x+1}{2}, 0, 1\right) = \max\left(0, \min \frac{x+1}{2}\right)$

$q = \mathrm{sign}(r)$

$g_r = g_q 1_{|r| \le 1}$??

Algorithm : 1. forward propagation:

for $k = 1$ to L : $W_k^b \leftarrow \mathrm{Binarize}(W_k)$; $S_k \leftarrow a_{k-1}^b W_k^b$;

$a_k \leftarrow \mathrm{BatchNorm}(S_k, \theta_k)$

if $k < L$, $a_k^b \leftarrow \mathrm{Binarize}(a_k)$

2. BP : for $k = L$-1 : if $k < L$, $g_{a_k} \leftarrow g_{a_k^b} \circ 1_{|a_k| \le 1}$

$g_{a_{k-1}^b} \leftarrow g_{s_k} W_k^b$

$g_{W_k^b} \leftarrow g_{s_k}^T a_{k-1}^b$

Time Complexity:

e.g.

$t = 5n^3 + 2n$

Time complexity $= O(n^3)$

3. Accumulating the parameters gradient

for $k = 1$ to L : $\theta_k^{t+1} \leftarrow update(\theta_k, \eta, g_{\theta_k})$

$W_k^{t+1} \leftarrow Clip(Update(W_k, \gamma_k \eta, g_{W_k}), -1, 1)$

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

BN → Shift-based Batch Normalization (SBN) (no matrix multiplication)

↓

shift-based AdaMax

Xnor-net : factor α.

$I * W = (I \oplus B)\alpha$