Обучение модели с бинарными весами для задачи классификации голосовых команд.

Задачи

- Подготовка датасета - конвертация команд с FFT, мы будем работать в этом домене, это нам позволит использовать признаки фиксированного размера.

- Обучение модели для классификации с бинарными весами.

- Написание кернелей на C++ под CPU для матричных умножений.

- Транспортировка полученных весов в C++ кернели

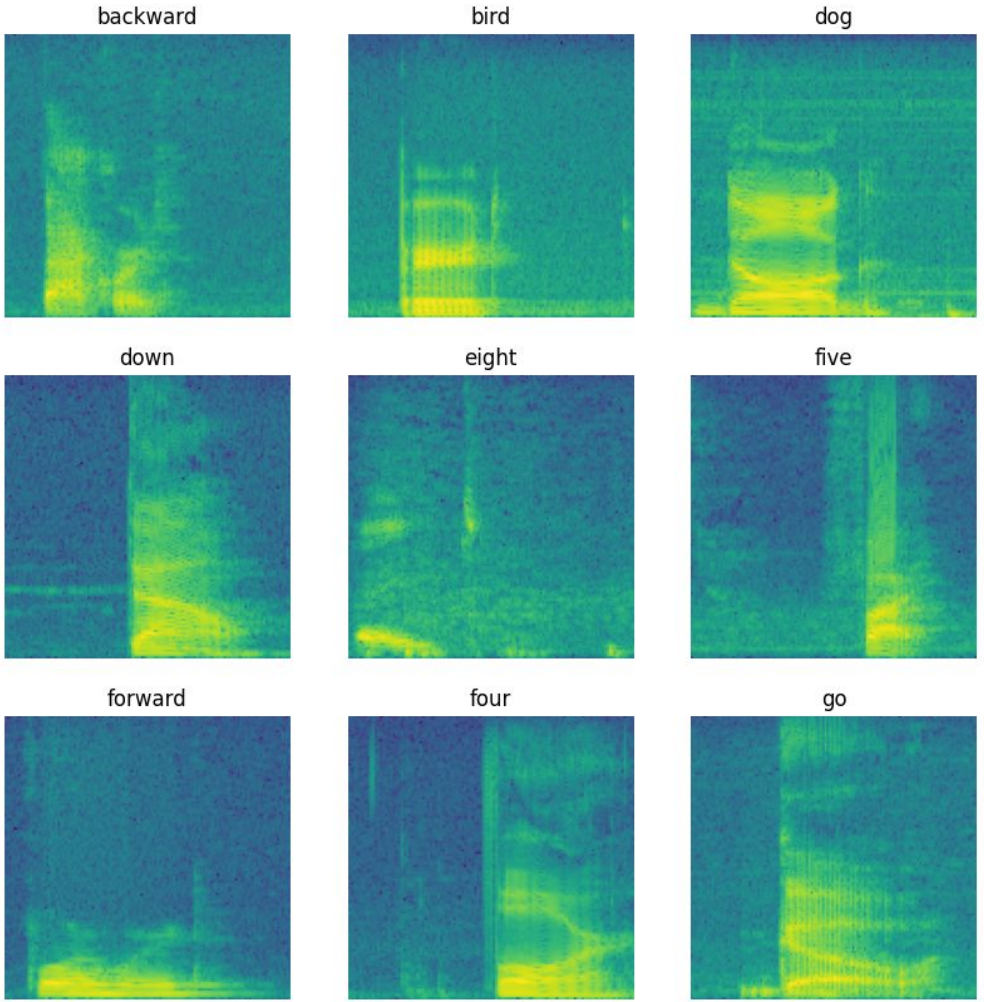
- Сравнение скорости и качества полученной модели с моделью на pytorch.

Mel spectrograms

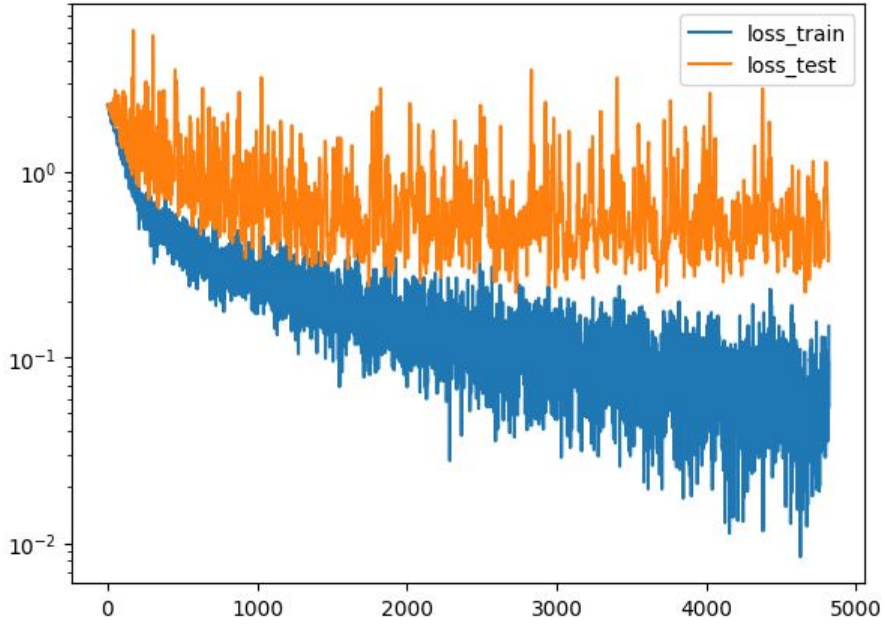
паддинг

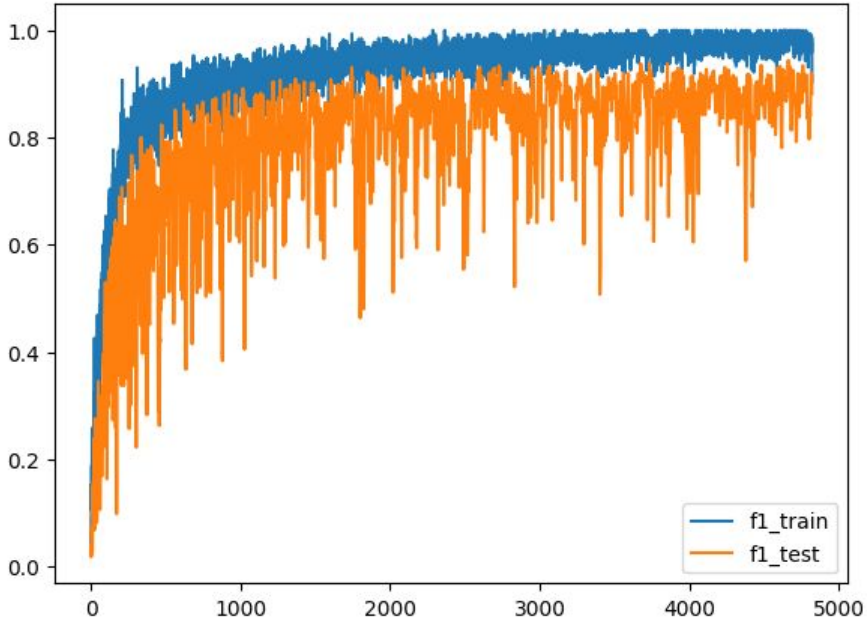
torchaudio.transforms.
MelSpectrogram

torchaudio.transforms.
AmplitudeToDB



Обучение маленькой сверточной сети без бинаризации





XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks

Estimating binary weights: Without loss of generality we assume $\mathbf{W}, \mathbf{B}$ are vectors in $\mathbb{R}^n$, where $n = c \times w \times h$. To find an optimal estimation for $\mathbf{W} \approx \alpha\mathbf{B}$, we solve the following optimization:

$$J(\mathbf{B}, \alpha) = \|\mathbf{W} - \alpha\mathbf{B}\|^2$$
$$\alpha^*, \mathbf{B}^* = \underset{\alpha,\mathbf{B}}{\operatorname{argmin}} J(\mathbf{B}, \alpha) \tag{2}$$

by expanding equation 2, we have

$$J(\mathbf{B}, \alpha) = \alpha^2\mathbf{B}^\mathsf{T}\mathbf{B} - 2\alpha\mathbf{W}^\mathsf{T}\mathbf{B} + \mathbf{W}^\mathsf{T}\mathbf{W} \tag{3}$$

since $\mathbf{B} \in \{+1, -1\}^n$, $\mathbf{B}^\mathsf{T}\mathbf{B} = n$ is a constant . $\mathbf{W}^\mathsf{T}\mathbf{W}$ is also a constant because $\mathbf{W}$ is a known variable. Lets define $\mathbf{c} = \mathbf{W}^\mathsf{T}\mathbf{W}$. Now, we can rewrite the equation 3 as follow: $J(\mathbf{B}, \alpha) = \alpha^2 n - 2\alpha\mathbf{W}^\mathsf{T}\mathbf{B} + \mathbf{c}$. The optimal solution for $\mathbf{B}$ can be achieved by maximizing the following constrained optimization: (note that α is a positive value in equation 2, therefore it can be ignored in the maximization)

$$\mathbf{B}^* = \underset{\mathbf{B}}{\operatorname{argmax}}\{\mathbf{W}^\mathsf{T}\mathbf{B}\} \quad s.t. \;\; \mathbf{B} \in \{+1, -1\}^n \tag{4}$$

This optimization can be solved by assigning $\mathbf{B}_i = +1$ if $\mathbf{W}_i \geq 0$ and $\mathbf{B}_i = -1$ if $\mathbf{W}_i < 0$, therefore the optimal solution is $\mathbf{B}^* = \operatorname{sign}(\mathbf{W})$. In order to find the optimal value for the scaling factor α^*, we take the derivative of J with respect to α and set it to zero:

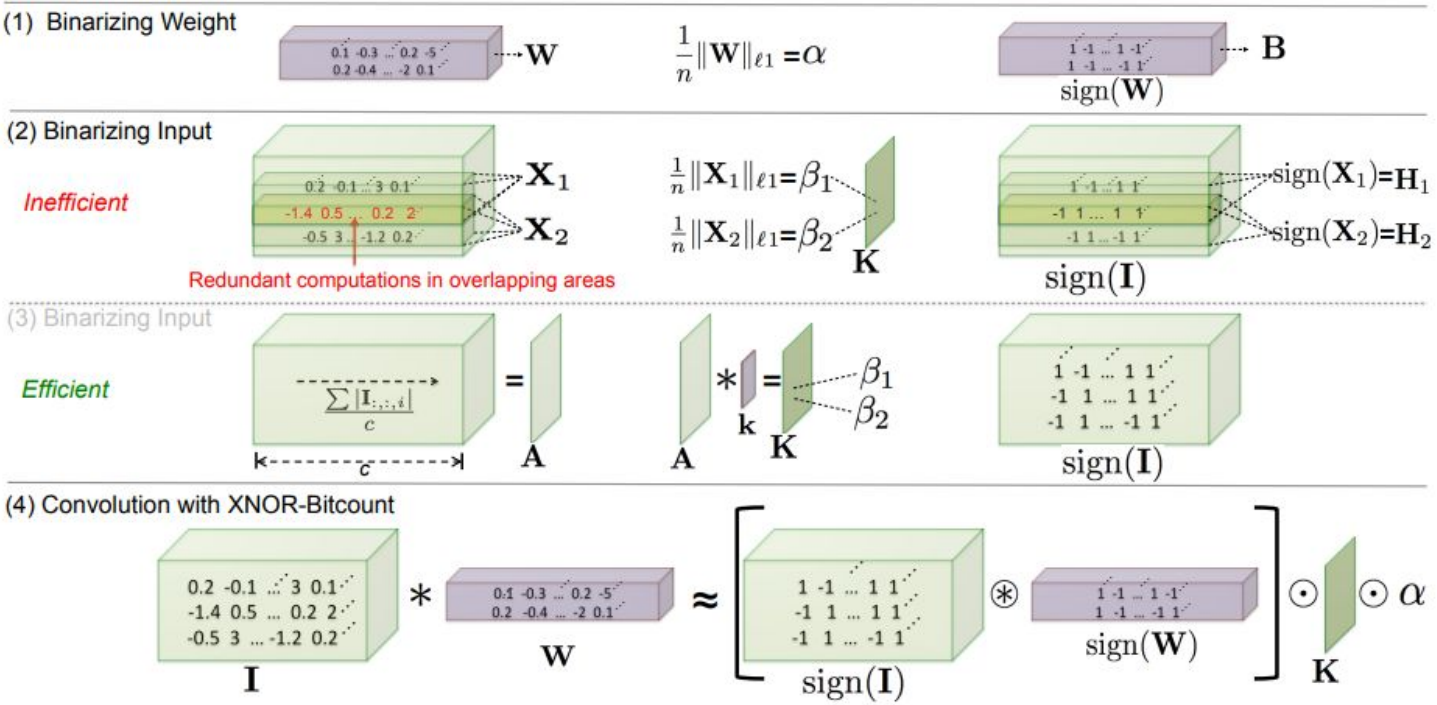
$$\alpha^* = \frac{\mathbf{W}^\mathsf{T}\mathbf{B}^*}{n} \tag{5}$$

By replacing $\mathbf{B}^*$ with $\operatorname{sign}(\mathbf{W})$

$$\alpha^* = \frac{\mathbf{W}^\mathsf{T}\operatorname{sign}(\mathbf{W})}{n} = \frac{\sum|\mathbf{W}_i|}{n} = \frac{1}{n}\|\mathbf{W}\|_{\ell 1} \tag{6}$$

https://arxiv.org/pdf/1603.05279

XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks



https://arxiv.org/pdf/1603.05279

XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks

- gradient for sign function
- порядок слоев

$$\frac{\partial \operatorname{sign}}{\partial r} = r1_{|r|\leq 1}$$

$$\frac{\partial C}{\partial W_i} = \frac{\partial C}{\partial \widetilde{W}_i}\left(\frac{1}{n} + \frac{\partial \operatorname{sign}}{\partial W_i}\alpha\right)$$

Fig. 3: This figure contrasts the block structure in our XNOR-Network (right) with a typical CNN (left).

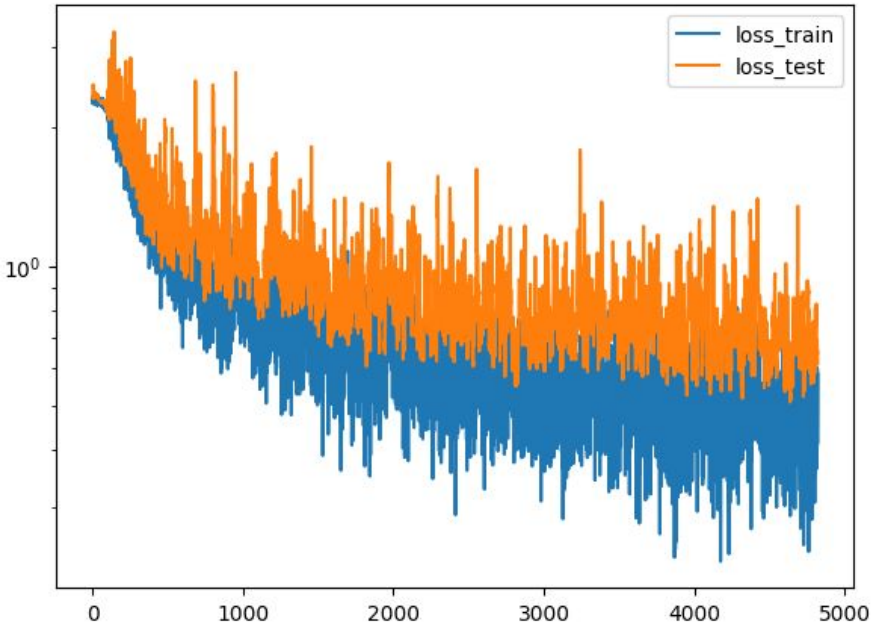
https://arxiv.org/pdf/1603.05279

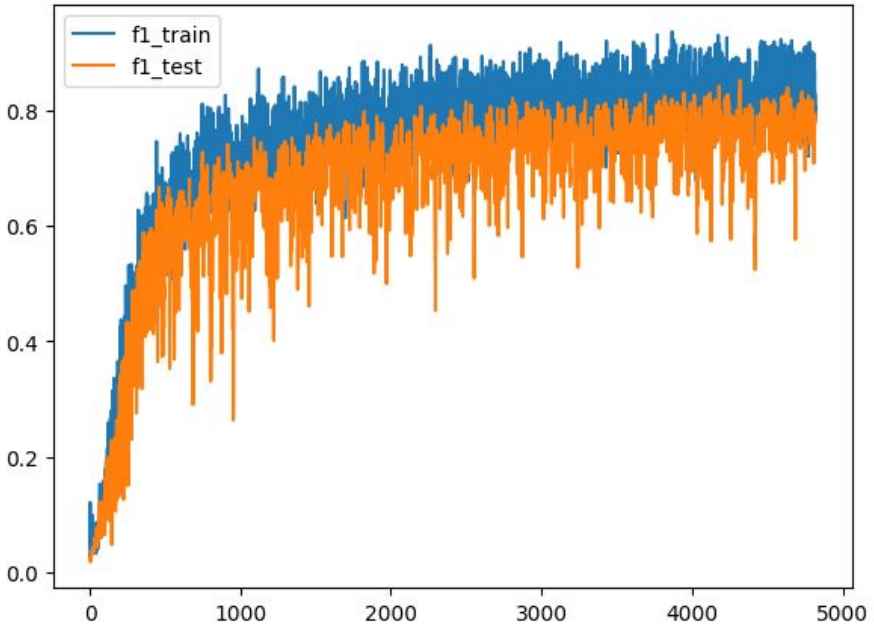
Наивная реализация XNOR-Net

```python
class SignSTE(torch.autograd.Function):
    @staticmethod
    def forward(ctx, x):
        ctx.save_for_backward(x)
        return x.sign()
    @staticmethod
    def backward(ctx, g):
        (x,) = ctx.saved_tensors
        mask = (x.abs() <= 1).to(g.dtype)
        return g * mask
```

```python
class NaiveXNORLinear(nn.Module):
    def __init__(self,ch_in,ch_out):
        super(NaiveXNORLinear, self).__init__()
        self.fc = nn.Linear(ch_in, ch_out, bias=True)

    def forward(self, input_x):
        quantized_weight = SignSTE.apply(self.fc.weight)
        quintized_input = SignSTE.apply(input_x)
        out = nn.functional.linear(quintized_input, quantized_weight)
        alpha = self.fc.weight.abs().mean()
        betta = input_x.abs().mean()
        return out*alpha*betta + self.fc.bias

class NaiveXNORConv2d(nn.Module):
    def __init__(self, ch_in,ch_out, kernel=3, padding=0, stride=1):
        super(NaiveXNORConv2d, self).__init__()
        self.padding = padding
        self.stride = stride
        self.conv = nn.Conv2d(ch_in, ch_out, kernel, stride, padding, bias=False)
        k = torch.ones((1, 1, kernel, kernel))/(kernel**2) #< Now consider only constants
        self.k = nn.parameter.Buffer(k, persistent=False)

    def forward(self, input_x):
        # A, k - BinActiv
        A = input_x.abs().mean(axis=1, keepdim=True)
        K = nn.functional.conv2d(A, self.k, None, self.stride, self.padding)
        alpha = self.conv.weight.abs().mean()

        quantized_weight = SignSTE.apply(self.conv.weight)
        quintized_input = SignSTE.apply(input_x)
        out = nn.functional.conv2d(quintized_input, quantized_weight, None, self.stride, self.padding)
        return out*K*alpha
```

Наивная реализация XNOR-Net





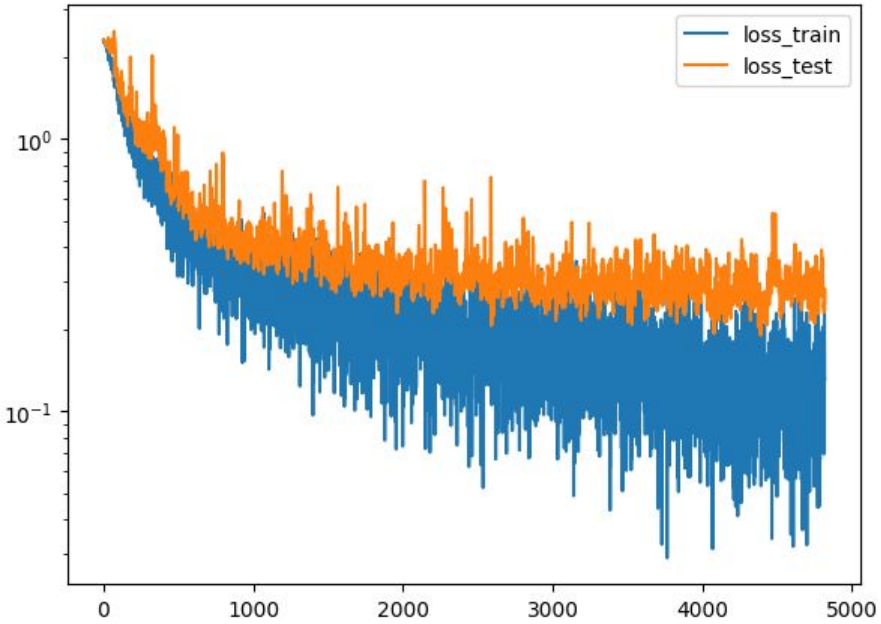
Смотрим готовую реализацию

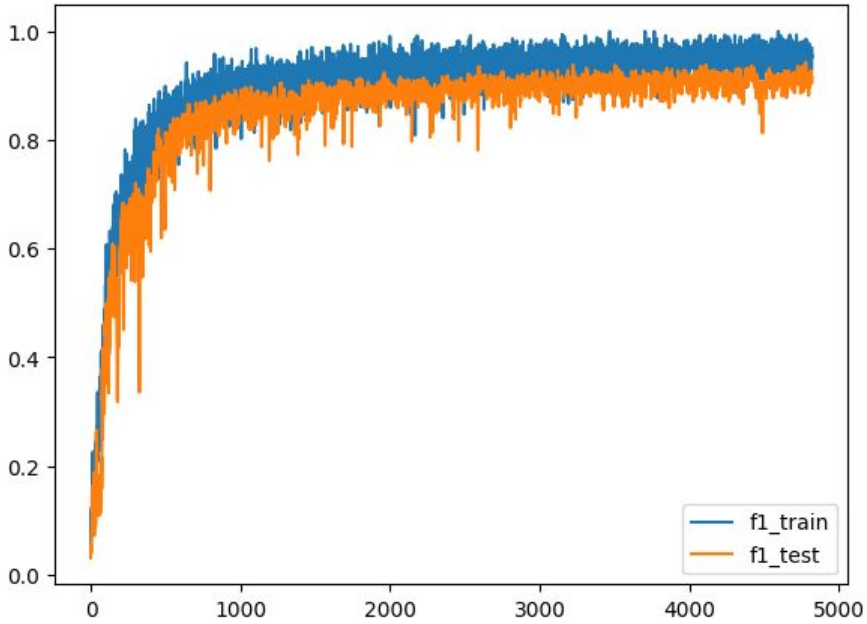
- https://github.com/allenai/XNOR-Net - оригинальный репозиторий на lua
- https://github.com/jiecaoyu/XNOR-Net-PyTorch

- class BinOp - бинаризация и градиент
- class BinActive - Sign forward, backward
- class BinConv2d - свертка и линейный слой (2 в одном) + batch norm + relu

```
bin_op = BinOp(model) #< use bin_op

for epoch in range(num_epochs):
    for i, (series, labels) in tqdm.tqdm(enumerate(train_loader), total=len(train_loader)):
        Net.train()
        bin_op.binarization() #< use bin_op
        outputs = Net(series.to(device))
        loss = loss_func(outputs, labels.to(device))
        self.loss_list.append(loss.item())

        optimizer.zero_grad()
        loss.backward()
        bin_op.restore() #< use bin_op
        bin_op.updateBinaryGradWeight() #< use bin_op

        optimizer.step()
```

Смотрим готовую реализацию





Небольшой рефакторинг

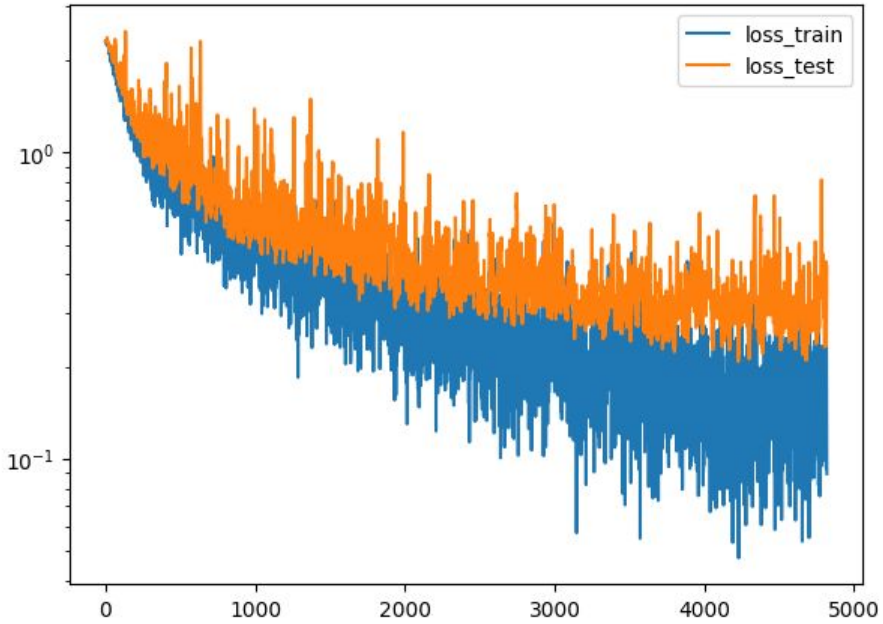
```python
@staticmethod
def forward(ctx, weight: torch.Tensor):
    if weight.dim() == 4:
        # [out, in, kh, kw]
        negMean = weight.mean(dim=1, keepdim=True).mul(-1).expand_as(weight)
    elif weight.dim() == 2:
        # [out, in]
        negMean = weight.mean(dim=1, keepdim=True).mul(-1).expand_as(weight)

    w_centered = weight + negMean
    w_clamped  = w_centered.clamp_(-1.0, 1.0)

    if weight.dim() == 4:
        n = weight[0].numel()
        m = (w_clamped.norm(p=1, dim=3, keepdim=True)
                      .sum(2, keepdim=True)
                      .sum(1, keepdim=True)
                      .div(n))
    else:
        n = weight.size(1)
        m = w_clamped.norm(p=1, dim=1, keepdim=True).div(n)

    w_bin = w_clamped.sign()

    ctx.save_for_backward(w_clamped)
    ctx.n = n
    return w_bin, m
```

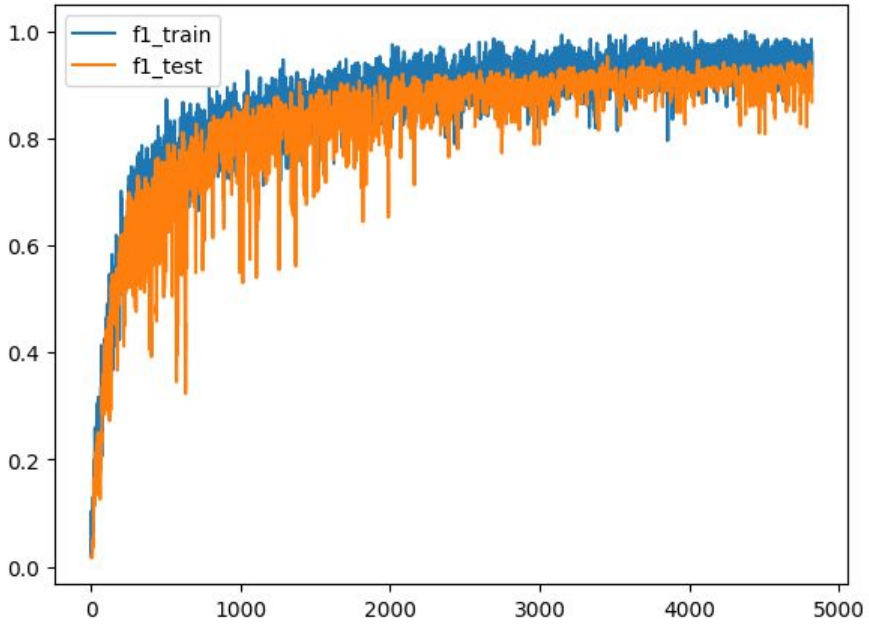
```python
@staticmethod
def backward(ctx, grad_out: torch.Tensor, scale_grad:torch.Tensor):
    (w_clamped,) = ctx.saved_tensors
    n = ctx.n
    s = w_clamped.size()

    g = grad_out
    if w_clamped.dim() == 4:
        m = (w_clamped.norm(p=1, dim=3, keepdim=True)
                      .sum(2, keepdim=True)
                      .sum(1, keepdim=True)
                      .div(n)).expand(s)
    else:
        m = w_clamped.norm(p=1, dim=1, keepdim=True).div(n).expand(s)

    m = m.clone()
    m[w_clamped.lt(-1.0)] = 0
    m[w_clamped.gt( 1.0)] = 0

    grad = m * g

    m_add = w_clamped.sign() * g
    if w_clamped.dim() == 4:
        m_add = (m_add.sum(3, keepdim=True)
                      .sum(2, keepdim=True)
                      .sum(1, keepdim=True)
                      .div(n)
                      .expand(s))
    else:
        m_add = m_add.sum(1, keepdim=True).div(n).expand(s)

    m_add = m_add * w_clamped.sign()
    grad = (grad + m_add) * (1.0 - 1.0 / s[1]) * n
    #grad = grad * 1e9
    return grad
```

https://github.com/allenai/XNOR-Net/issues/41

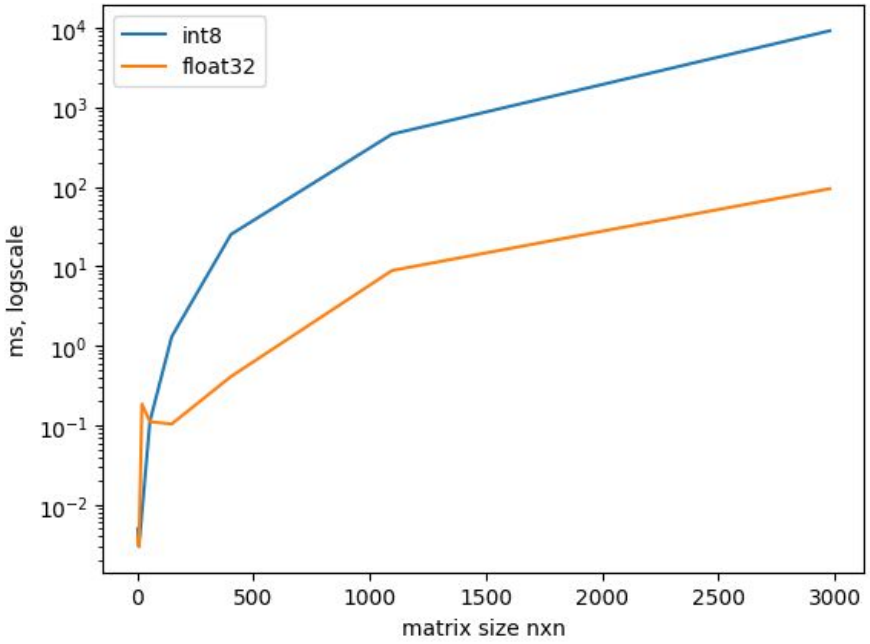
https://github.com/jiecaoyu/XNOR-Net-PyTorch/issues/126

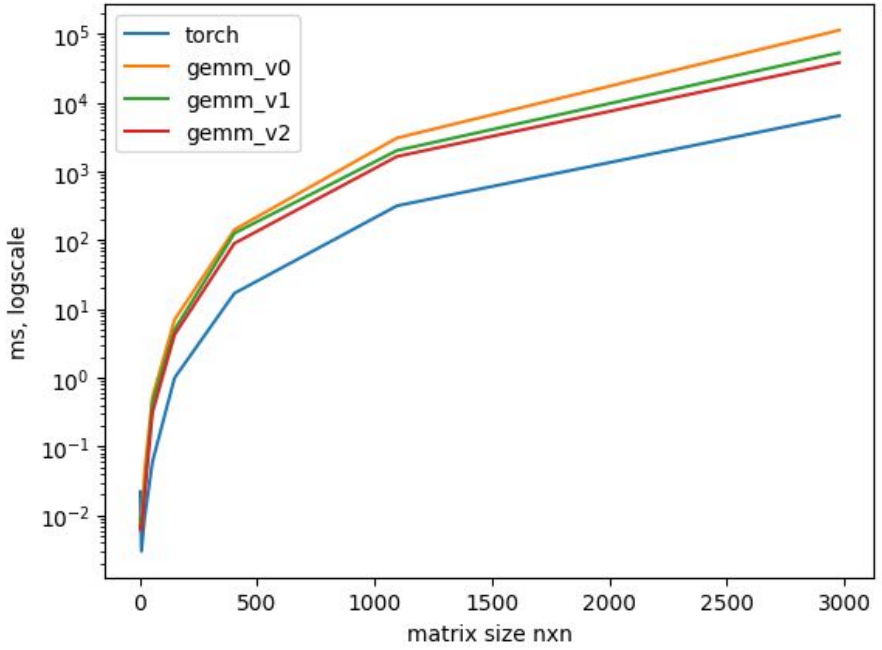
Использование квантизации из семинара





сравнение матричных умножений





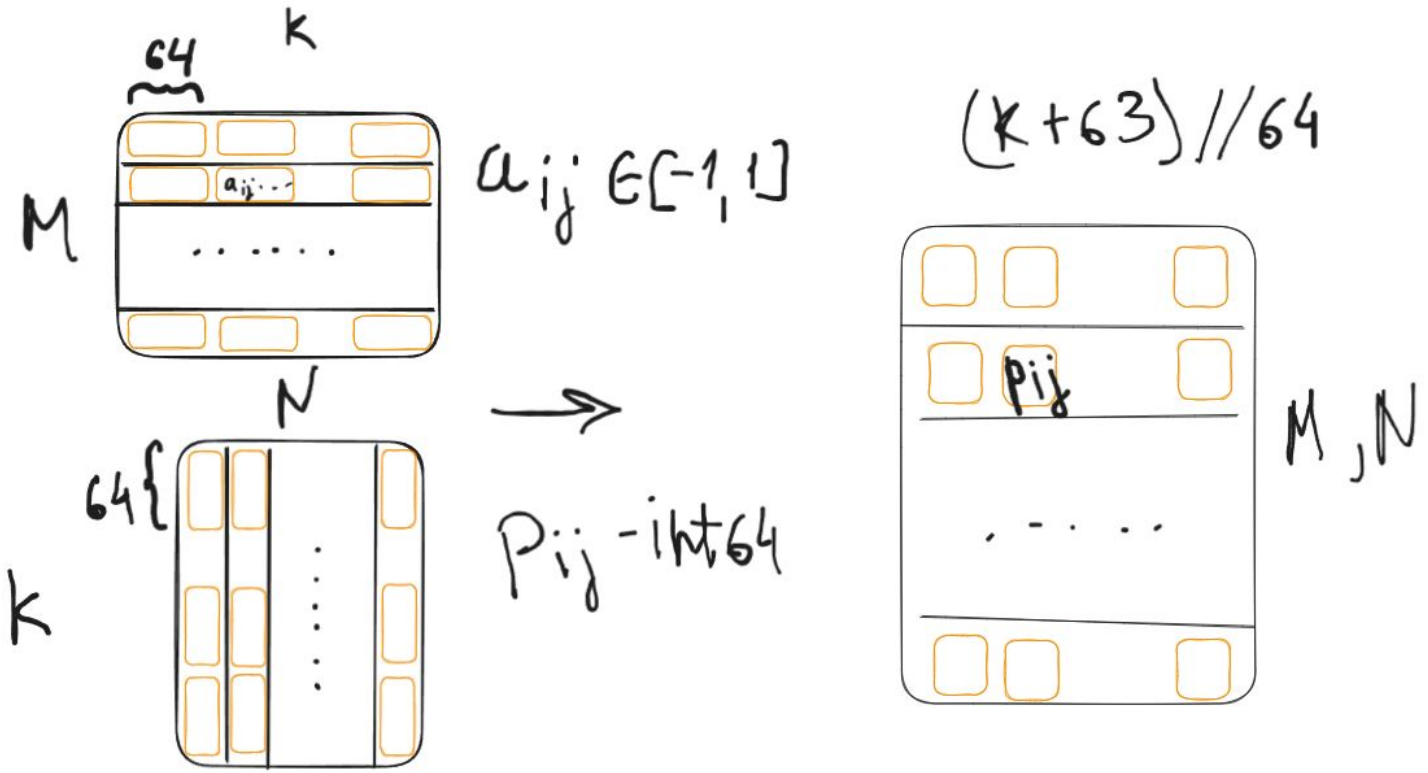
https://habr.com/ru/articles/359272/

https://stackoverflow.com/questions/79806482/why-is-eigen-c-int-matrix-multiplication-10x-slower-than-float-multiplication

torch.utils.cpp_extension

- несколько подходов: можно сделать через load_inline, можно через setup
- https://github.com/pytorch/extension-cpp
- https://github.com/AmirMardan/pytorch_extending_cpp_binding
- могут возникнуть проблемы, если setuptools>75.8.2 https://github.com/pypa/setuptools/issues/4874
- флаг "-march=native" - немного ускоряет кастомные реализации

XNOR + POPCOUNT

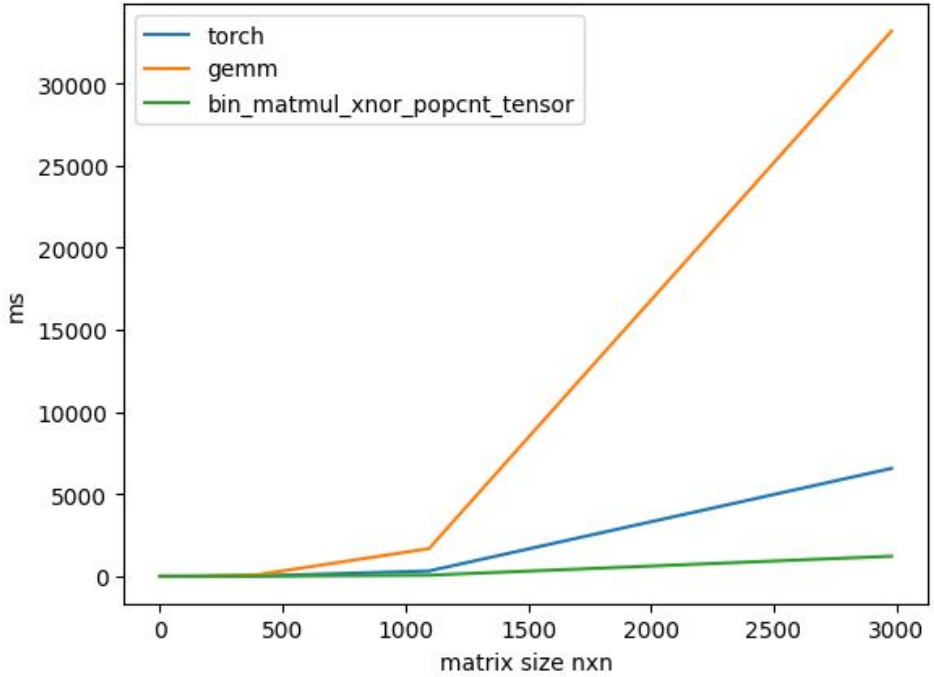


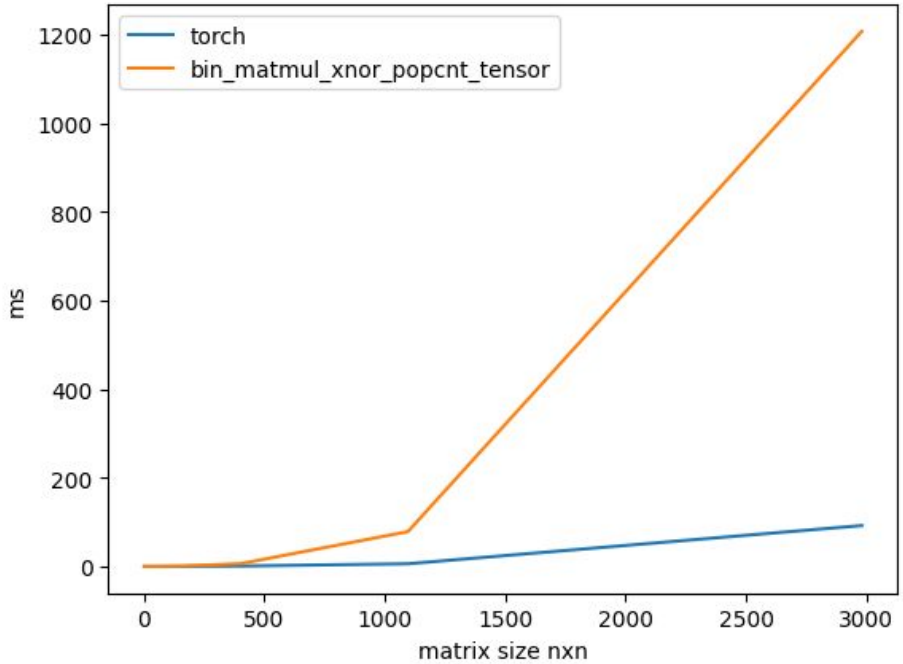
XNOR + POPCOUNT

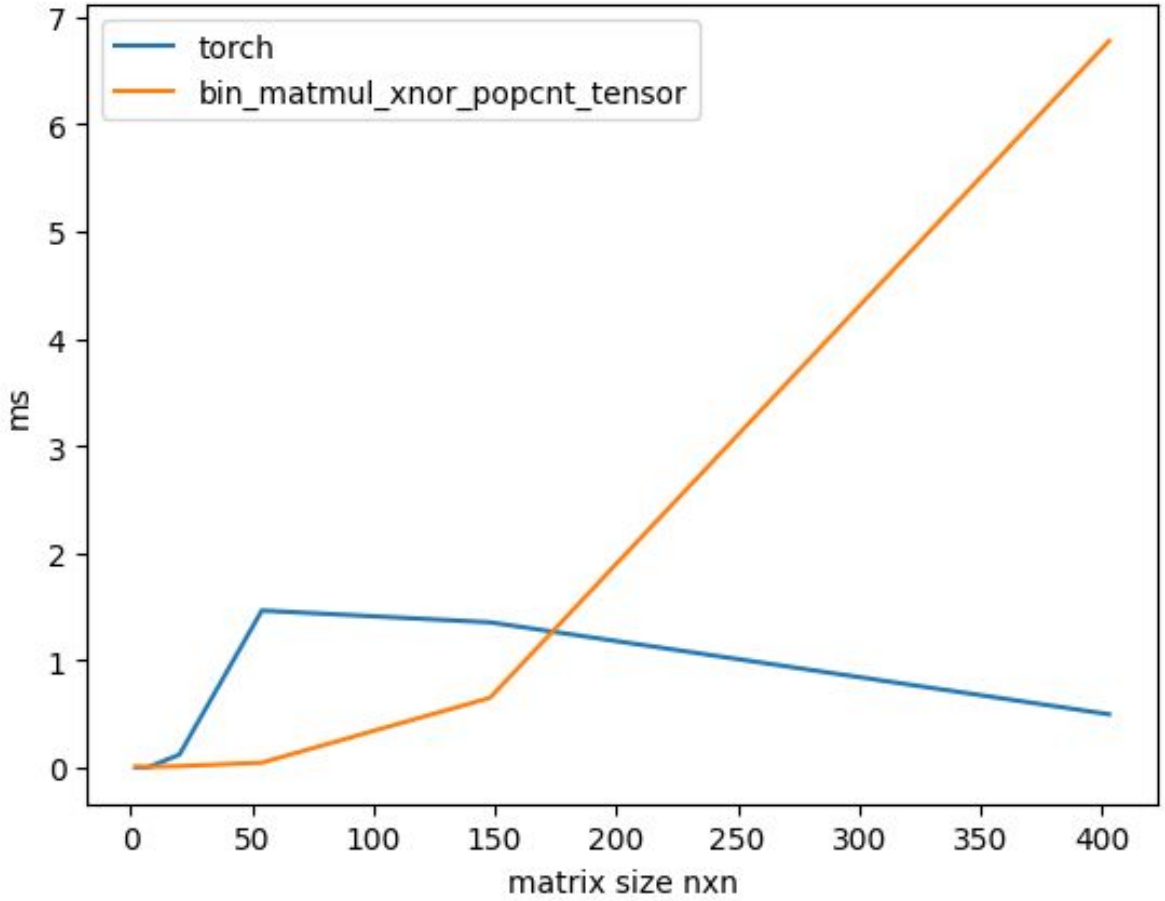
```
tensor([ 1., -1.,  1.,  1., -1.,  1.,  1., -1., -1.,  1., -1., -1., -1.,  1.,
        -1.,  1.,  1.,  1., -1.,  1.,  1., -1., -1.,  1., -1., -1., -1., -1.,
        -1.,  1., -1.,  1.,  1.,  1., -1., -1.,  1.,  1.,  1., -1.,  1.,  1.,
        -1., -1.,  1., -1.,  1., -1., -1., -1.,  1.,  1., -1., -1., -1., -1.,
         1., -1., -1.,  1., -1., -1., -1.,  1.])
tensor(-8571384234711997843)
1000100100001100010100110111001110100000100110111010001001101101
```

$$\sum_{i=1}^{K} a_i \cdot b_i = (+1) \cdot (equal) + (-1) \cdot (diff) = equal - (K - equal) = 2 \cdot equal - K$$

сравнение с pytorch






torch
bin_matmul_xnor_popcnt_tensor
ms
matrix size nxn
0
1
2
3
4
5
6
7
0
50
100
150
200
250
300
350
400

Другие реализации

- https://github.com/honcharov-danylo/ultraspeed
- https://github.com/cooooorn/Pytorch-XNOR-Net
- https://github.com/brycexu/BNN_Kernel
- https://github.com/tairenpiao/XNOR-popcount-GEMM-PyTorch-CPU-CUDA