Hw6 code review report

111062503 吳冠志

1. 程式

本次實作 prune.py 之 prune_by_percentile、prune_by_std、及 quantization.py 之 apply_weight_sharing

prune_by_percentile

透過不同層數的百分比設定 threshold

```python
def prune_by_percentile(self, q={'conv1': 16, 'conv2': 62, 'conv3': 65, 'conv4': 63, 'conv5': 63, 'fc1': 91, 'fc2': 91, 'fc3': 75}):
    ######################
    # TODO
    #   For each layer of weights W (including fc and conv layers) in the model, obtain the qth percentile of W as
    #   the threshold, and then set the nodes with weight W less than threshold to 0, and the rest remain unchanged.
    ######################
    for name, module in self.named_modules():
        if name in ['conv1','conv2','conv3','conv4','conv5']:
            t=abs(module.weight.data.cpu())
            t=t.reshape(-1).numpy()
            t=sorted(t)
            index=round(q[name]*0.01*len(t))
            threshold=t[index]
            print(f'Pruning with threshold : {threshold} for layer {name}')
            self.prune(module, threshold)
        if name in ['fc1', 'fc2', 'fc3']:
            t=abs(module.weight.data.cpu())
            t=t.reshape(-1).numpy()
            t=sorted(t)
            index=round(q[name]*0.01*len(t))
            # _k_sort=np.argpartition(t,-index)[-index:]
            threshold=t[index]
            print(f'Pruning with threshold : {threshold} for layer {name}')
            self.prune(module, threshold)
```

prune_by_std

透過 std 設定 threshold

```python
def prune_by_std(self, s=0.25):
    for name, module in self.named_modules():

        ################################
        # TODO:
        #    Only fully connected layers were considered, but convolution layers also needed
        ################################
        if name in ['conv1','conv2','conv3','conv4','conv5']:
            threshold = np.std(module.weight.data.cpu().numpy()) * s
            print(f'Pruning with threshold : {threshold} for layer {name}')
            self.prune(module, threshold)
        if name in ['fc1', 'fc2', 'fc3']:
            threshold = np.std(module.weight.data.cpu().numpy()) * s
            print(f'Pruning with threshold : {threshold} for layer {name}')
            self.prune(module, threshold)
```

apply_weight_sharing

以 Kmeans 聚類後更新 weight

```python
def apply_weight_sharing(model, bits=5):
    """
    Applies weight sharing to the given model
    """
    for name, module in model.named_children():
        dev = module.weight.device
        weight = module.weight.data.cpu().numpy()
        shape = weight.shape
        quan_range = 2 ** bits
        if len(shape) == 2:  # fully connected layers
            print(f'{name:20} | {str(module.weight.size()):35} | => quantize to {quan_range} indices')
            mat = csr_matrix(weight) if shape[0] < shape[1] else csc_matrix(weight)

            # weight sharing by kmeans
            space = np.linspace(min(mat.data), max(mat.data), num=quan_range)
            # kmeans = KMeans(n_clusters=len(space), init=space.reshape(-1, 1), n_init=1, precompute_distances=True, algorithm="full")
            kmeans = KMeans(n_clusters=len(space), init=space.reshape(-1, 1), n_init=1, algorithm="full")
            kmeans.fit(mat.data.reshape(-1, 1))
            new_weight = kmeans.cluster_centers_[kmeans.labels_].reshape(-1)
            mat.data = new_weight
            # Insert to model
            module.weight.data = torch.from_numpy(mat.toarray()).to(dev).float()
        elif len(shape) == 4:  # convolution layers
            print(f'{name:20} | {str(module.weight.size()):35} |  => quantize to {quan_range} indices')
            # print(f'{a2=}')
            a_np=np.zeros(weight.shape)
            a=weight.reshape(-1)
            a2= a[a != 0]
            space = np.linspace(min(a2), max(a2), num=quan_range)
            kmeans = KMeans(n_clusters=len(space), init=space.reshape(-1, 1), n_init=1, algorithm="full")
            kmeans.fit(a2.reshape(-1, 1))
            centers = kmeans.cluster_centers_
            centroid_dict={}
            for i,j in enumerate(centers):
                centroid_dict[i]=j
            for i in range(weight.shape[0]):…
            device=torch.device("cuda" if torch.cuda.is_available() else "cpu")
            module.weight.data = torch.from_numpy(a_np).to(device).float()
```