CARA PENGGUNAAN

1. Pruning

Mengurangi penggunaan memori untuk menyimpan model dengan meningkatkan sparsity (parameter nol) ke tingkat sparsity yang diinginkan. pangkas bobot dengan besaran terkecil. Ini akan menyetel parameter yang kurang penting ke nol dan kami tidak mendapatkan kecepatan komputasi apa pun tanpa perangkat keras khusus yang dapat bekerja dengan representasi tensor yang jarang.

Sebelum mulai memangkas model, disarankan untuk menganalisis sensitivitas setiap lapisan sehingga kita dapat memilih sparitas terbaik.

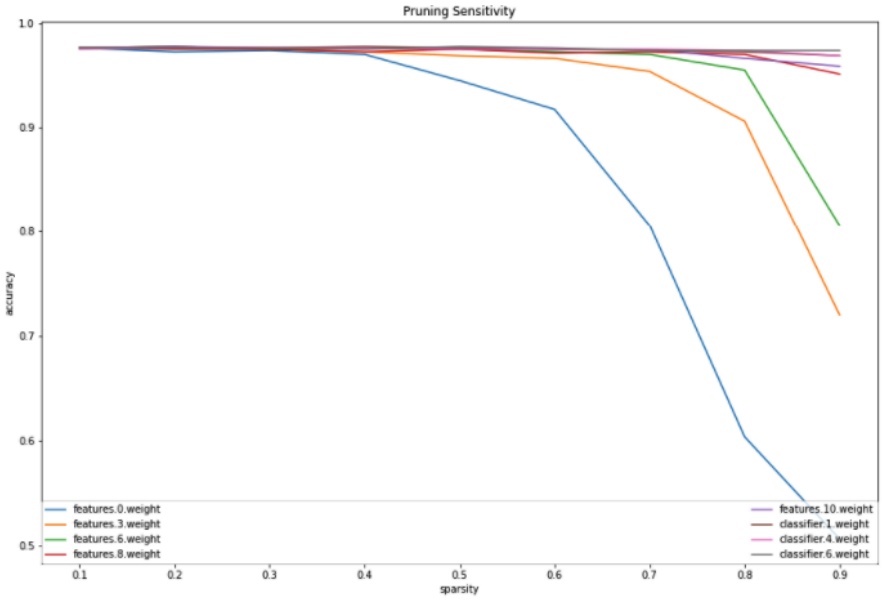
```
# sensitivity analysis usage
# the model should be fully trained before performing the sensitivity analysis
import pruner.sensitivity_scan as senscan

sparsities = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
criterion = nn.CrossEntropyLoss()

# model_ft, fully trained model
# val, validation data loader
# validate_func, validation function that return acc, loss
sensitivities = senscan.perform_sensitivity_analysis(
    model_ft, val, sparsities, validate_func, criterion
)

# plot the sensitivities
senscan.sensitivity_to_png(sensitivities, path)

# save sensitivities to csv
senscan.sensitivities_to_csv(sensitivities, path)
```



Contoh lebih lengkap bisa di lihat disini.

Berdasarkan hasil dari sparsity analisis selanjutnya penggunaan untuk kelas pruner seperti sebagai berikut:

```
from pruner.levelpruner import LevelPruner


# define the pruner
prune_options = [
       ['features.0.weight', 0.35],
       ['features.3.weight',  0.5],
       ['features.6.weight',  0.7],
       ['features.8.weight',  0.8],
       ['features.10.weight',  0.8],
       ['classifier.1.weight',  0.8],
       ['classifier.4.weight',  0.8],
       ['classifier.6.weight',  0.8]
]
pruner = LevelPruner(options=prune_options)

...
for inputs, labels in dataloaders:
  optimizer.zero_grad()
  outputs = model(inputs)
  loss = criterion(outputs, labels)
  loss.backward()
  optimizer.step()
  pruner.prune(model=model, update_masks=False)
...
```

Contoh lengkapnya bisa dilihat disini.

2. Quantization

Dalam repo ini K-Means clustering digunakan untuk menghitung centroids

Anda juga dapat mempertimbangkan untuk melakukan analisis untuk memilih bit yang tepat tetapi kode ini akan membutuhkan banyak waktu untuk dieksekusi terlebih lagi jika Anda memiliki jutaan parameter.

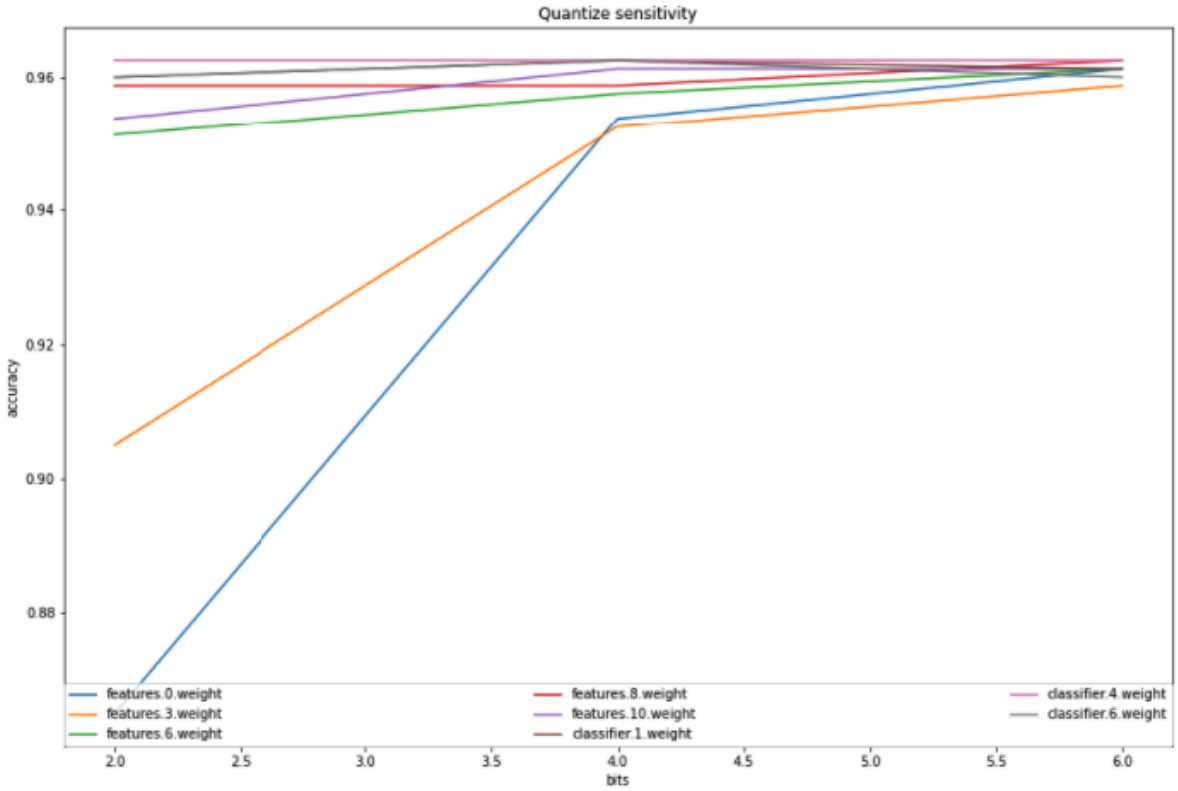
```
import quantizer.bits_analysis as bits_analysis

criterion = nn.CrossEntropyLoss()
bits = [6, 4, 2]

with torch.no_grad():
  sensitivities = bits_analysis.perform_bits_analysis(
      model_ft, val, bits, validate_func, criterion
  )

# plot the results
bits_analysis.sensitivities_to_png(sensitivities, path)

# save to csv
bits_analysis.sensitivities_to_csv(sensitivities, path)
```



Selanjutnya berdasarkan hasil dari analisis di atas kita bisa mengimplementasikanya ke kelas quantizer.

```
# Quantizer usage
from quantizer.quantizer import Quantizer

# initialize quantizer
quantizer_options = [
       ('features.0.weight', 6),
       ('features.3.weight', 6),
       ('features.6.weight', 6),
       ('features.8.weight', 6),
       ('features.10.weight', 4),
       ('classifier.1.weight', 2),
       ('classifier.4.weight', 2),
       ('classifier.6.weight', 2)
]
quantizer = Quantizer(options=quantizer_options)

#quantize model
with torch.no_grad():
  quantizer.quantize(model_ft)
```

Untuk contoh penerapanya bisa dilihat [disini.]