Reducción de redes

1 Punto de partida

Tomamos una red profunda grande, de las últimas que hemos probado. Si no la tienes a mano, prepárala. Para alguno de los apartados posteriores, convendrá que la crees con un `OrderedDict` Y también, por limitaciones de Pytorch, si usas `padding`, que no sea `replicate`.

Mide el tamaño total de sus matrices de pesos. Calcula las operaciones necesarias por cada elemento de muestra, teniendo en cuenta que:

- Pasar un filtro $n \times n$ sobre una imagen $m \times m$ supone $(m-n+1) \times (m-n+1)$ aplicaciones
- Cada aplicación son $n \times n$ operaciones
- Aparte hay que sumar las funciones de activación

Los elementos de comparación entre redes van a ser el tamaño, las operaciones y la precisión. Esto último, por ejemplo en toda la muestra. Al final tendremos una tabla con las distintas posibilidades. Pon ya la primera línea con este modelo base.

Vimos también algunas ideas para reducir la red. Recupera alguna de ellas y añade otra fila a la tabla.

En cualquier momento, si quieres ver los pesos, puedes hacer algo como:

```
for etiq, capa in red0.named_modules():
    if isinstance(capa, torch.nn.Conv2d) or isinstance(capa, torch.nn.
       Linear):
            print(etiq,capa,capa.weight)
```

2 Intentamos una red profunda "pequeña" de cara

2.1 Matrices de pesos simplificadas

Prueba a poner que las capas tengan matrices de pesos parametrizadas, por ejemplo, mediante una simplificación a ortogonales (`https://pytorch.org/docs/stable/generated/torch.nn.utils.parametrizations.orthogonal.html`) Sería sustituir las `Conv2d(...)` por `nn.utils.parametrizations.orthogonal(nn.Conv2d(...))`
Esto hace que el tamaño de una matriz $m \times n$ pase a $\frac{m(n+1)}{2}$ (podemos pensar en aproximadamente la mitad)

Prepara y ajusta. en principio a las de más impacto (más pesos) y vete ampliando hasta que veas que la precisión cae significativamente. Donde te quedes, anota otra línea en la tabla de comparación.

2.2 Aplicación de filtros separables

Sustituye las convoluciones normales por separables, al estilo de

Original	Con filtro separable
nn.Conv2d(4, 2, kernel_size=(5, 5), padding=2)	nn.Conv2d(4, 4, kernel_size=(5, 1), padding=(2, 0)), nn.Conv2d(4, 2, kernel_size=(1, 5), padding=(0, 2))

Fíjate que en ese ejemplo, hemos pasado de 25 pesos a 10 (aunque como hay 4,4 en la primera capa, el ahorro no es así de directo)

Prepara, ajusta primero las capas de más pesos, hasta que veas que la precisión se degrada. Hasta donde llegues, anota otra línea

3 Simplificación de la red original

En todos estos apartados partimos de la red original grande, la de la primera línea de la tabla. Recuerda que puedes cargar los pesos previamente guardados en un fichero en una red ya definida, con `red.load_state_dict(torch.load(nombrefichero))`

3.1 Recorta

3.1.1 Pesos

Utiliza

`torch.nn.utils.prune.L1Unstructured`

Juega con el ratio. Aparte tienes otras dos decisiones: ¿aplicas el ratio por capas o globalmente? ¿dejas los valores de pesos preservados o reajustas a los supervivientes? Haz pruebas:

- Vete subiendo el ratio mientras la precisión no se estropee mucho
- Aplicar el ratio por capa sería:

```
for _, capa in red.named_modules():
        if isinstance(capa, torch.nn.Conv2d) or isinstance(capa, torch.nn.
           Linear) or las que tengas:
            prune.l1_unstructured(capa,name='weight',amount=0.31416 o lo
               que sea)
            prune.remove(capa,'weight') # si quieres borrar
               definitivamente los originales
```

- Globalmente sería:

```
parametros=[]
for _, capa in red.named_modules():
        if isinstance(capa, torch.nn.Conv2d) or isinstance(capa, torch.nn.
           Linear) or las que vayas a considerar:
                parametros.append((capa,'weight'))
prune.global_unstructured(parametros,pruning_method=prune.L1Unstructured,
   amount=0.5 o lo que veas)
```

- Si reajustas, considera si tiene sentido usar inicialización aleatoria, o coger los valores que ya tenían los pesos. En este último caso, no procede probar varios puntos de partida, porque sólo hay uno.

Con lo mejor que consigas, añade otra línea a la tabla (o un par de ellas, en caso de duda entre opciones).

3.1.2 Procesadores

Igual, pero con

`torch.nn.utils.prune.LnStructured`

es decir, jugando con la cantidad de procesadores a quitar en cada capa, algo como:

```
for capa in red.children():
        try:
            if len(capa.weight)>1:# no vas a quitar  si sólo hay uno
                prune.ln_structured(capa,name='weight',amount=numprocaquitar,n
                    =2,dim=0)
        except:
            pass
```

3.2 Cuantiza

Vas a necesitar que el modelo base esté creado con un `OrderedDict`, para poder referirte a las capas por su nombre. Prepara otra red igual en que añadas un punto de cuantización al inicio y uno de decuantificación al final, algo como:

```
('inicuant1', torch.quantization.QuantStub()),
('convol1',nn.Conv2d(1,4,11)),
...
('ultima',nn.Tanh()),
('fincuant1', torch.quantization.DeQuantStub())
```

Sea esta `nuevared`

Copia en ella los pesos de la normal: `nuevared.load_state_dict(red0.state_dict())`

Prepárala para calibrar: `nuevared.eval()`

Prepara una configuración simple (el asunto se puede elaborar bastante):

`nuevared.qconfig = torch.quantization.get_default_qconfig('fbgemm')` Eso es optimizado para UCP Intel; si lo quieres para correr en un ARM cambia `fbgemm` por `qnnpack`

Puede convenir juntar las dos operaciones de dos capas para que se hagan en una sola pasada, por ejemplo la convolución y la no-lineal, pero eso sólo lo tiene Pytorch para algunos tipos de capas, así que quizás no puedas. De poder, sería como:

`nuevaredcuant = torch.quantization.fuse_modules(nuevared, [['convol', 'activ']])`

Inicializa la calibración: `nredinic = torch.quantization.prepare(nuevaredcuant)`

Calibra con un (sub)conjunto de datos: `nredinic(datos)`

Crea la red cuantizada: `redcuant = torch.quantization.convert(nredinic)`

Mide los errores que da la red cuantizada: `salidacuant = redcuant(datosmedir)`

A la hora de medir tamaños de matrices, ten en cuenta que la red cuantizada trabaja las convoluciones y activaciones con enteros de 8 bits

4 Conclusión

Puestos a elegir los menores error, tamaño y cantidad de cálculos, posiblemente contradictorios, ¿con qué opción te quedarías?