ESE 546, FALL 2020

HOMEWORK 2

SHEIL SARDA [SHEILS@SEAS],
COLLABORATORS: PRANAV P. [PILLAIP@SEAS], RAHUL M. [RMAG@SEAS]

Solution 1 (Time spent: 2 hours)**.** (1) Peculiarities in the code:

Batch normalization is applied before `ReLU` in the `resent-18` model. This normalizes the input to each layer for each batch, thus speeding up training.

(2) What does `model.train()` do

It specifies to all the layers that the model is in training mode, so they should behave accordingly; i.e. that gradients are computed correctly.

What does `model.eval()` do

It specifies to the layers that it is now in evaluation model, and training has completed; i.e. weights should not be updated further.

The above functionality applies to layers such as dropout layers which have dual functionality.

Why did we not have it in homework 1

In the previous homework, we did not have layers which behave differently under training and evaluation conditions.

(3) `resnet-18` architecture

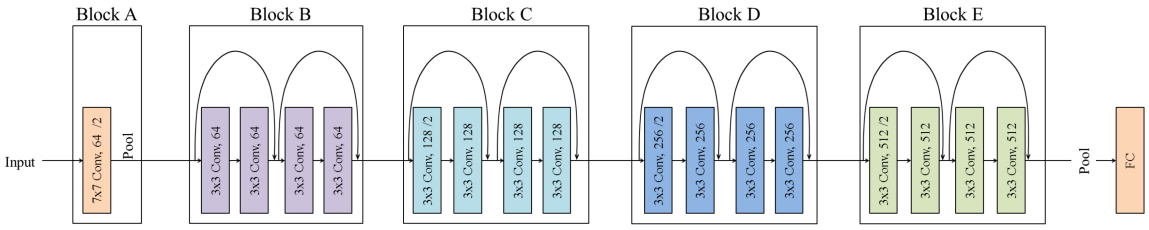

FIGURE 1. Sketch of the resnet-18 architecture

Number of parameters in each layer

torch.nn.modules.conv.Conv2d’	9408
torch.nn.modules.batchnorm.BatchNorm2d’	64
torch.nn.modules.conv.Conv2d’	36864
torch.nn.modules.batchnorm.BatchNorm2d’	64
torch.nn.modules.conv.Conv2d’	36864
torch.nn.modules.batchnorm.BatchNorm2d’	64
torch.nn.modules.conv.Conv2d’	36864
torch.nn.modules.batchnorm.BatchNorm2d’	64
torch.nn.modules.conv.Conv2d’	36864
torch.nn.modules.batchnorm.BatchNorm2d’	64
torch.nn.modules.conv.Conv2d’	73728
torch.nn.modules.batchnorm.BatchNorm2d’	128
torch.nn.modules.conv.Conv2d’	147456
torch.nn.modules.batchnorm.BatchNorm2d’	128
torch.nn.modules.conv.Conv2d’	8192
torch.nn.modules.batchnorm.BatchNorm2d’	128
torch.nn.modules.conv.Conv2d’	147456
torch.nn.modules.batchnorm.BatchNorm2d’	128
torch.nn.modules.conv.Conv2d’	147456
torch.nn.modules.batchnorm.BatchNorm2d’	128
torch.nn.modules.conv.Conv2d’	294912
torch.nn.modules.batchnorm.BatchNorm2d’	256
torch.nn.modules.conv.Conv2d’	589824
torch.nn.modules.batchnorm.BatchNorm2d’	256
torch.nn.modules.conv.Conv2d’	32768
torch.nn.modules.batchnorm.BatchNorm2d’	256
torch.nn.modules.conv.Conv2d’	589824
torch.nn.modules.batchnorm.BatchNorm2d’	256
torch.nn.modules.conv.Conv2d’	589824
torch.nn.modules.batchnorm.BatchNorm2d’	256
torch.nn.modules.conv.Conv2d’	1179648
torch.nn.modules.batchnorm.BatchNorm2d’	512
torch.nn.modules.conv.Conv2d’	2359296
torch.nn.modules.batchnorm.BatchNorm2d’	512
torch.nn.modules.conv.Conv2d’	131072
torch.nn.modules.batchnorm.BatchNorm2d’	512
torch.nn.modules.conv.Conv2d’	2359296
torch.nn.modules.batchnorm.BatchNorm2d’	512
torch.nn.modules.conv.Conv2d’	2359296
torch.nn.modules.batchnorm.BatchNorm2d’	512
torch.nn.modules.linear.Linear’	512000

(4) Weight decay should not be applied to the biases of the different layers in the network since its main function is to prevent overfitting by reducing the number of features which are significantly weighted. By restricting the value of the bias term, which is a simple offset applied to each sample, the weights now have to adjust for the bias implicitly which can lead to more features receiving higher weights.

(5)
```
import torchvision.models as models
import numpy as np
resnet18 = models.resnet18()

layers = list(resnet18.modules())
layer_type = {"BatchNorm": 0, "Bias": 0, "Misc": 0}
for layer in layers:
  if hasattr(layer, 'weight'):
    params = 1
    for weight in layer.weight.shape:
      params *= weight
    if "bias=True" in str(layer):
      layer_type["Bias"] += 1000
    if "BatchNorm" in str(layer):
      layer_type["BatchNorm"] += params
    else:
      layer_type["Misc"] += params
```

Solution 2 (Time spent: 4 hours)**.** (1) The above problem is not convex because the gradient and hessian of $||X - AB||_F$ is not positive semidefinite for all X, A, B.

Consider the scalar case:

$$\min(X - AB)^2 = \min X^2 - 2XAB + A^2B^2$$

The Hessian of $\phi_X(A, B) = X^2 - 2XAB - A^2B^2$ is

$$\nabla\phi_x(y, w) = \begin{bmatrix} 2y^2w - 2xy \end{bmatrix}$$
$$\nabla^2\phi_x(y, w) = \begin{bmatrix} 2B^2 & 4AB - 2X \\ 4AB - 2X & 2A^2 \end{bmatrix}$$

The Hessian is not positive semidefinite for all X, A, B.

(2) Find the global optimum for this loss function.

Using a truncated SVD, we find that the global optimum for this loss function as:

$$X = A \times \Sigma \times B$$

The global optima can be found by taking the derivative of

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\left((A\Sigma B)_{ij} - (AB)_{ij}\right)^2$$

Upon doing so, we find that

$$\Sigma = I \text{where } I \text{ is the } n \times n \text{ identity matrix}$$

Solution 3 (Time spent: 2 hours)**.** (1) Let $y = \lambda w$. Applying the chain rule to the given expression, we know:

$$\frac{dl(\lambda w)}{dw} = \frac{dl(y)}{dw} = \frac{dl(y)}{dy} \times \frac{dy}{dw} = \frac{dl(y)}{dy}\lambda = \frac{dl(\lambda w)}{d\lambda w}\lambda$$

However, since we know that

$$l(w) = l(\lambda w) \implies \frac{dl(\lambda w)}{d\lambda w} = \frac{1}{\lambda} \times \frac{dl(\lambda w)}{dw} = \frac{1}{\lambda} \times \frac{dl(w)}{dw}$$

□

(2) Given: $w^{t+1} = w^t - \eta \nabla \ell \left(w^t\right)$

$$\begin{aligned}
w^{t+1} &= w^t - \eta \nabla \ell \left(w^t\right) \\
\frac{w^{t+1}}{||w^t||_2} &= \frac{w^t}{||w^t||_2} - \frac{\eta}{||w^t||_2} \nabla \ell \left(w^t\right) \\
\frac{w^{t+1}}{||w^t||_2} &= v^t - \frac{\eta}{||w^t||_2^2} \times ||w^t||_2 \nabla \ell \left(w^t\right) \\
\frac{w^{t+1}}{||w^t||_2} &= v^t - \frac{\eta}{||w^t||_2^2} \times ||w^t||_2 \nabla \ell \left(v^t \times ||w^t||_2\right) \\
\frac{w^{t+1}}{||w^t||_2} &= v^t - \frac{\eta}{||w^t||_2^2} \nabla \ell \left(v^t\right) \\
\frac{w^{t+1}}{||w^{t+1}||_2} \times \frac{||w^{t+1}||_2}{||w^t||_2} &= v^t - \frac{\eta}{||w^t||_2^2} \nabla \ell \left(v^t\right) \\
v^{t+1} &= \left[v^t - \frac{\eta}{||w^t||_2^2} \nabla \ell \left(v^t\right)\right] \times \frac{||w^t||_2}{||w^{t+1}||_2}
\end{aligned}$$

□

Solution 4 (Time spent: 0 hours)**.** N / A

Solution 5 (Time spent: 7 hours)**.** (1) After training.

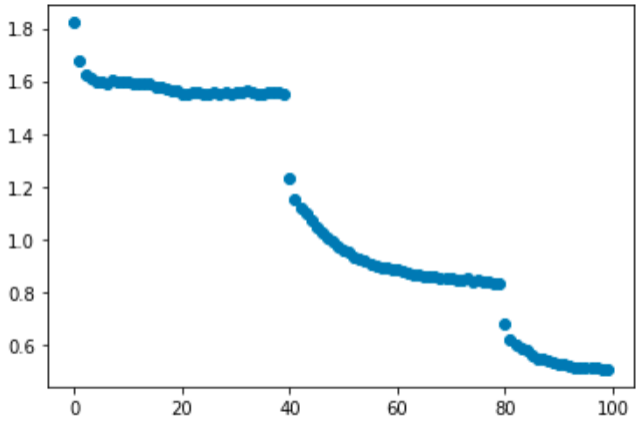

FIGURE 2. Training Loss

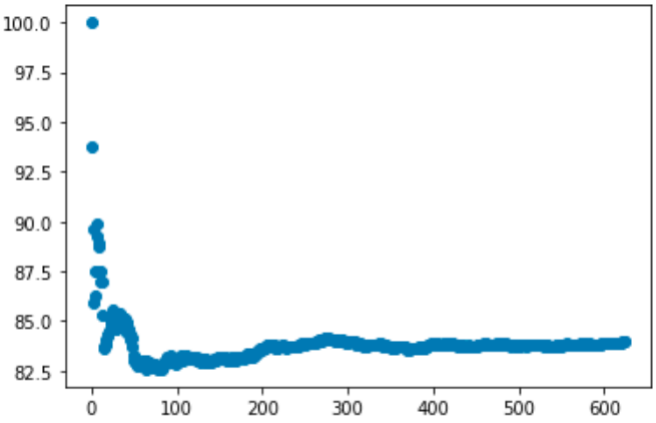

FIGURE 3. Validation Accuracy

(2) Gradient plots

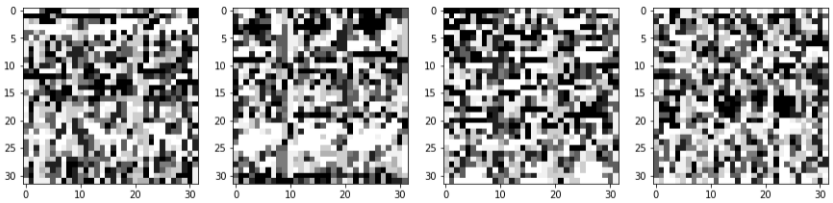

FIGURE 4. Visualization of gradients for correct predictions

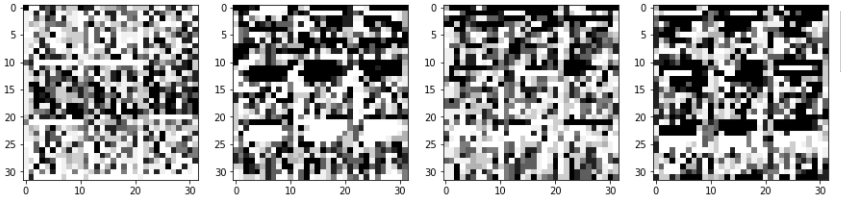

FIGURE 5. Visualization of gradients for incorrect predictions

Loss on perturbed images.

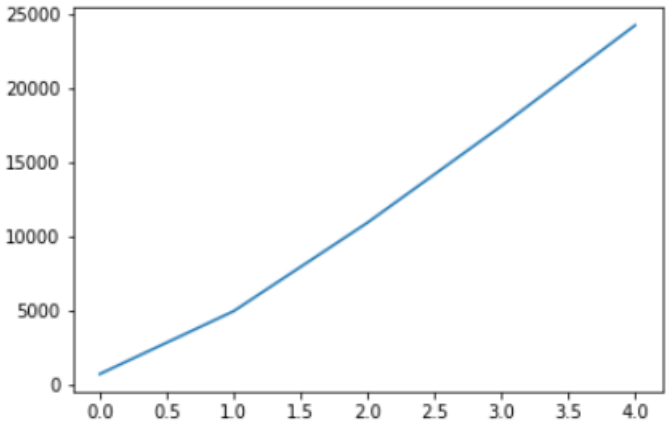

FIGURE 6. 5-step-attack loss

(3) 1-step perturbed images.

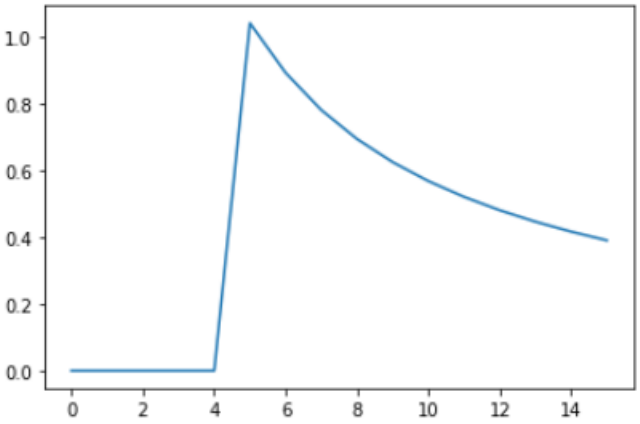

FIGURE 7. Accuracy of model on 1-step perturbed images

How does this compare against the accuracy on the clean validation set
The accuracy is much much lower and more noisy. Validation on clean set was in the high 80%.

Solution 6 (Time spent: 3 hours)**.** (1) The back-propagation gradient for w_z if the loss function is only a function of the hidden vector at time T.

$$\frac{dl(z^t)}{dw_z} = \frac{dl(z^t)}{dz^t} \times \frac{dz^t}{dw_z}$$
$$\frac{dl(z^t)}{dw_z} = \frac{dl(z^t)}{dz^t} \times \frac{dz^t}{dz^{t-1}} \times \frac{dz^{t-1}}{dz^{t-2}} \dots \times \frac{dz^1}{dw_z}$$
$$\frac{dl(z^t)}{dw_z} = \frac{dl(z^t)}{dz^t} \times w_z^t \times z^0$$

Compute the conditions on the weight matrix w_z^t under which the gradient explodes and vanishes.

In the above equation, w_z^t is exponentiated such that it explodes when $w_z^t > 1$ and vanishes when $w_z^t < 1$.

(2) How does the nonlinearity affect the exploding or vanishing gradients?

Nonlinearities or activation functions affect the exploding and vanishing gradients in the following way:

$$\frac{dz^t}{dz^{t-1}} = f(d\sigma)$$

That is, it becomes a function of the derivative of the activation function. For nonlinearities whose derivatives tend to evaluate to < 1, e.g. `sigmoid`, `tanh`, will lead to vanishing gradients.

Which nonlinearities are well-suited for training RNNs?

Activation functions / nonlinearities like `ReLU` whose gradient is fixed at 1 is much more well suited for the task of avoiding exploding and vanishing gradients.

(3) How to protect the weights $w_z^{\texttt{new}}$ from blowing up even if the gradient explodes

We can apply functions which preserve the sign of the gradient while ensuring the value does not blow up. An example of this transformation:

$$\sigma\left(\frac{dl}{dw_z} - \frac{1}{2}\right) \text{ where } \sigma \text{ is the sigmoid function}$$

Can you modify the updates to handle the vanishing gradient problem?

To handle vanishing gradients, we also need to lower bound the values of the gradient update.

(4) Explain how the LSTM solves the problem of vanishing gradients?

The long term dependencies and relations are encoded in the cell state vectors and it's the cell state derivative that can prevent the `LSTM` gradients from vanishing.

The presence of the forget gate, along with the additive property of the cell state gradients, enables the network to update parameters in such a way that the different sub gradients do not necessarily agree and behave in a similar manner, making it less likely that all of the gradients vanish, thus preventing the vanishing gradient problem.