Deep Learning & Applied AI

June 5, 2024

The following responses are taken from various students' answers on the past exam, all of which received the highest grades.

Question 1 (4 points) Suppose you just discovered a new library for training NNs. You were born curious, and as you peek into its source code, you are surprised to find that SGD implements the following equation:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha^{(t)} \nabla \ell(\theta^{(t)}) + \beta^{(t)}(\theta^{(t)} - \theta^{(t-1)})$$

1. What is the equation above doing?
2. Is it actually SGD?

Answer

1. It is a step of gradient descent plus a term that acts like momentum. This particular momentum term replaces the gradient of the previous iterations with finite differences.
2. Basically yes, although you would have to apply the iterations to minibatches to get a proper SGD.

Question 2 (4 points) You are participating in an AI hackathon and you are about to train a standard MLP with the classical MSE loss. Somebody from the other teams suggested that you initialize all the weights to one, and all the biases to zero. Is this a good idea? Why?

Answer

Not a good idea! When you initialize all weights to the same value, the gradients during backpropagation will be the same for each weight. This causes each weight to update in the same way, preventing the model from breaking symmetry and learning diverse features.

Question 3 (4 points) It's your first day of PhD, and you overheard a senior lab member saying: "*The number of parameters of a model does not necessarily correlate with its memory usage*". Since it's your first day, you are compelled to provide an example of this statement. What example(s) would you give to make a good impression? Why?

Answer

A few examples come to mind. 1) CNNs: here, the number of parameters scales with the size of the convolutional kernels, which are typically very few, but the intermediate feature maps can be very large. 2) Transformers: the attention matrices scale quadratically with the length of the sequence, and their size does not correlate with the number of learnable weights.

Question 4 (9 points) You are reading a highly cited paper on a new hot topic called *model merging*, and you are trying to understand what this is all about. The paper shows the following equation:

$$\mathbf{z}_{\ell+1} = \mathbf{P}^\top \mathbf{P} \mathbf{z}_{\ell+1} = \mathbf{P}^\top \mathbf{P} \sigma(\mathbf{W}_\ell \mathbf{z}_\ell + \mathbf{b}_\ell) = \mathbf{P}^\top \sigma(\mathbf{P}\mathbf{W}_\ell + \mathbf{P}\mathbf{b}_\ell)$$

Your experienced advisor looks at the equation and says, "*Hah, of course*". And, after a while, exclaims: "*Why didn't I think of this symmetry before? Now I see how they merge!*

It's your chance to show off to your advisor!

1. Motivate each of the '=' signs in the equation above.
2. What is this 'symmetry' your advisor is mentioning?
3. What did your advisor realize?

Hint: ***model merging*** *refers to the problem of averaging the weights of different neural networks together.*

Answer

Note: There was a typo in the last equality, which should read $\mathbf{P}^\top \sigma(\mathbf{P}\mathbf{W}_\ell \mathbf{z}_\ell + \mathbf{P}\mathbf{b}_\ell)$ (notice the missing $\mathbf{z}_\ell$ in the original text).

1. (i) For the equality to hold, it must be $\mathbf{P}^\top \mathbf{P} = \mathbf{I}$, i.e. matrix $\mathbf{P}$ must be orthogonal; (ii) action of a standard layer in a MLP; (iii) $\mathbf{P}$ commutes with the element-wise activation σ, which is only true if $\mathbf{P}$ is a permutation matrix.
2. The symmetry refers to the permutation-equivariance of the nonlinear activation function, that allows permuting with $\mathbf{P}$ the rows of a layer and then anti-permuting the result using $\mathbf{P}^\top$.
3. Since my advisor now understands how merging is performed, he/she probably realized that one can look for a permutation matrix such that $\mathbf{P}\mathbf{W}_\ell^A \approx \mathbf{W}_\ell^B$, where A and B are two different networks. Once the permutation has been found, one can simply compute an average such as $\frac{\mathbf{P}\mathbf{W}^A + \mathbf{W}^B}{2}$ to perform model merging.

Question 5 (4 points) Consider a deterministic AE. How would you modify it so that it behaves like PCA? You are free to modify the architecture, the loss, or both.

Answer

To modify an AE to behave like PCA, we must remove all the nonlinearities to get a purely linear encoder and a linear decoder. The loss may be left as it is, because PCA looks for a $\mathbf{W}$ such that $\mathbf{W}\mathbf{W}^\top\mathbf{X} \approx \mathbf{X}$, which looks like a standard reconstruction requirement $\mathbf{D}(\mathbf{E}(\mathbf{X})) \approx \mathbf{X}$ in an AE.

Question 6 (4 points) Consider the loss landscape of some neural network, trained on some task with some data.

1. How does a *fine-tuning* process explore the loss landscape compared to training a model from scratch?
2. Imagine fine-tuning only the last layer vs. adding new layers and fine-tuning them: what would change?

Answer

1. Fine-tuning typically starts from a good pre-trained model, hence one can expect that the minimum of the fine-tuning loss will be nearby the minimum obtained by the pre-trained model. Therefore, a short-lived and localized exploration in the parameter space is expected.
2. Fine-tuning only the last layer means that the rest of the parameters are frozen, or in other words, that the exploration in the loss landscape only happens along certain dimensions (the non-frozen ones). Adding new layers completely changes the loss landscape, since it affects the parameter domain.

Question 7 (4 points) Let X be a (m, n) tensor with $m = 5$ and $n = 3$. Which of these broadcasting operations works correctly?

1. `X[:, None] + X[None, :]`
2. `X[..., None] + X[None, ...]`
3. None of the above
4. What if $m = n = 2$?

Answer

1. Corresponds to adding a $(5, 1, 3)$ tensor to a $(1, 5, 3)$ tensor. Broadcasting will work as expected, resulting in a $(5, 5, 3)$ tensor.
2. Corresponds to a sum $(5, 3, 1) + (1, 5, 3)$; the intermediate dimensions are not compatible, and broadcasting will not work.
3. –
4. If $m = n = 2$, both options above will work, resulting in a tensor with shape $(2, 2, 2)$.

Question 8 (7 points) A friend of yours wants to try two neural networks, A and B, on dataset D (with labels $\hat{D}$) optimizing the following loss $\mathcal{L}$:

$$\begin{aligned}
\mathcal{L} &= \beta\mathcal{L}(A) + (1-\beta)\mathcal{L}(B) \\
\mathcal{L}(A) &= \text{Cross-Entropy}(A(D), \hat{D}) + \text{KL}(A(D), B(D)) \\
\mathcal{L}(B) &= \text{Cross-Entropy}(B(D), \hat{D}) + \text{KL}(B(D), A(D))
\end{aligned}$$

1. What is the loss $\mathcal{L}$ trying to enforce? Why?
2. Does this loss allow A and B to have different architectures (e.g. a CNN and a Transformer)? Why?
3. Your friend tried to optimize this loss using off-the-shelf optimizers from PyTorch (SGD, Adam, etc.), but couldn't get the models to converge. What do you think is happening?
4. Suggest a possible way to fix it.

Answer

1. The loss tries to encourage both A and B to classify correctly while behaving more like one another, in the sense that their output distributions should have low KL divergence. This is because the first term for A and B is a cross-entropy on the predictions, while the second is a KL divergence between the respective distributions.
2. There is no architecture constraint. A and B can have any architecture as long as their output size is correct and matches the number of classes.
3. PyTorch would simultaneously update both A and B, which might lead to loss instability. This is because it is possible that while A is updated to behave like B, the latter is updated to behave like A and they struggle to strike a balance and behave similarly.
4. A simple solution is to alternate the update between A and B. When updating A, fix B, and viceversa. This is not something PyTorch does by default.