Programming for ML: Quiz 1

14th September, 2025

Rules

- No use of AI agents.
- Open book/open notes/open internet.
- No help from other humans.

Questions

The first three problems concern simplified implementations of key parts of a Convolutional Neural Network (CNN) which you will study in detail the next semester but here you will write some of these by hand so you have some exposure.

1. Write a function called `convolution` which you implement directly (without using any existing functionality in any library like numpy/scipy/pytorch etc.). `convolution` takes the following arguments, `data` which is a list or 1D numpy array of floating point values, `kernel` which is another list or 1D numpy array of floating point values which always has an odd number of values, `stride` which is an integer which defaults to 1.

   The discrete convolution operation as implemented in CNNs basically goes like this, let `data` have `N` values, consider some index `0 <= i < N` takes each value of the `data` array. Let `kernel` have `k` values where `k` is odd. Let the result of the convolution be named `result` so we have `result[i] =` $\sum_{m=0}^{k-1}$ `data[i - k//2 + m]*k[m]`. The `stride` simply determines the values we are looking at. So if `stride` is 1, we iterate over all the values of the `data` array from index `0` to `N-1` if it is 2, we skip every alternate element. Note that the result array will be of size `(N+1)//stride`.

   Note that when `i<k//2` or `i >= N-k//2` we have a problem since the index is outside the valid range of the `data` array so you can assume that the value of `data` when `i-k//2 +m` is outside the acceptable range is zero. This is called zero-padding.

   Implement the function `convolution(data, kernel, stride)` which returns a numpy array of the result. Here are some examples. (6 marks)

```
>>> data = np.arange(5)
>>> kernel = [1/3, 1/3, 1/3]
>>> print(convolution(data, kernel)))
[0.33333333 1.         2.         3.         2.33333333]

>>> print(convolution(data, kernel, stride=2))
[0.33333333 2.         2.33333333]
```

2. Write a function called `max_pool(data, stride)` which is given a `data` array as before and a `stride` which defaults to 2. This function returns another array of size `(N+1)//stride` where in each sliding window (of size stride) we find the maximum value of the data elements and so, `result[i] =    max(data[i], data[i+1], ..., data[i+stride-1])`. Note that this is a simplified implementation. Once again, treat any missing values in the data array as zero. Return a numpy array with the result. Here are some examples. (3 marks)

```
>>> print(max_pool(range(5)))
[1 3 4]
>>> print(max_pool(range(6)))
[1 3 5]
>>> print(max_pool(range(6), stride=3))
[2 5]
```

3. We next implement layer normalization, write a function `layer_norm(data)` which returns normalized data by computing the mean `d_mean` and standard deviation, `d_sigma` of the `data` and finding `result[i] = (data[i] -    d_mean)/d_sigma`. (2 marks)

4. Consider the 5 batsmen, Rohit, Gambhir, Kohli, Sachin, and Sehwag. Given their batting score data (which you already have), create a box plot with their scores in one figure (all the box plots should show up together) appropriately labeled and pick the best of these. (2 mark)

5. For the above batsmen, compute the standard deviation of their respective scores along with their means. Use `plt.errorbar` to plot this information with the mean + the standard deviation to provide a visual representation of this information. Label the axes and ticks appropriately (hint: use `plt.xticks`). Comment on the result. (2 marks)

6. For the above 5 batsmen, compute their probability mass function as done in class using their scores data. Then use this to find the probability that they will score more than 50 if they have crossed 5 runs. Print this information for each batsman. Ideally do this in a loop rather than 5 cut-pasted code-blocks. (5 marks)

Submit a single jupyter notebook with your answers in sequence and upload this on the GL interface.