Week 2 Sample Oral Assessment Questions

CSE151B SS1 2026

Below are questions that will be similar to what you'll see on this week's oral assessment. As mentioned in class, you'll be asked two questions–first an "understand" question, and then an "experiment" question. For questions where we imagine there are a few possible phrasings, there are multiple sample acceptable answers listed. Overall, if you completed the implementations for A1 and understood how and why they worked (either by thinking through or experimenting with different components), you should be well prepared.

Note that all of these contain excerpts from your assignment solution code for your reference, but you don't need to do a detailed read of every line of code to answer them–it's just there in case it helps as you answer the question!

Remember that we will only count your answer after you say **"My answer is..."** and we will only give hints after you answer "yes" to "Would you like a hint?" or answer incorrectly.

Understand:

1. [Topic: A1 Part 1 util.py]

```python
def one_hot_encoding(labels, num_classes=10):
    """
    Encodes labels using one hot encoding.

    args:
        labels : N dimensional 1D array where N is the number of examples
        num_classes: Number of distinct labels that we have (10 for FashionMNIST)

    returns:
        oneHot : N X num_classes 2D array
    """
    labels = labels.flatten().astype(int)
    one_hot = np.zeros((len(labels), num_classes))
    one_hot[np.arange(len(labels)), labels] = 1
    return one_hot
```

Q: One reason we use one-hot encoding is to prevent models from learning incorrect relationships from label data. How does one-hot encoding do this?

Possible answers:

My answer is: "If we keep our label data in real values (i.e. 0, 1, 2, ...), then the model may infer that label 1 (pants) + label 4 (t-shirt) = label 5 (jacket). One-hot encoding makes it so

that each label does not have correlations with each other."

2. [Topic: A1 Part 1 Activation Functions]

Q: What will ReLU output when you input a large negative value into the ReLU activation function?

A: **My answer is:** "ReLU will output 0."

Experiment:

1. [Topic: A1 Part 1 Momentum]

```python
def backward(self, deltaCur, learning_rate, momentum_gamma, regularization, gradReqd=True):
    """
    Backward pass for this layer.

    deltaCur: delta signal from the next layer (N, out_units). For the output layer this
              is (y_hat - targets); for hidden layers it arrives already stripped of the
              bias row and needs to be element-wise multiplied by f'(a).

    Returns delta to propagate to the previous layer (before bias row is stripped).
    """
    # Multiply by the activation derivative (1 for output layer)
    delta = deltaCur * self.activation.backward(self.a)   # (N, out)

    # Gradient w.r.t weights (bias row included in self.x)
    # deltaCur already carries the 1/N factor from the loss
    self.dw = self.x.T @ delta   # (in+1, out)

    # L2 regularization on weights (skip bias row at index 0)
    if regularization > 0:
        reg = regularization * self.w
        reg[0, :] = 0   # don't regularize bias
        self.dw += reg

    if gradReqd:
        if momentum_gamma > 0:
            self.v = momentum_gamma * self.v + self.dw
            self.w -= learning_rate * self.v
        else:
            self.w -= learning_rate * self.dw

    # Delta to pass to the previous layer: strip the bias row (row 0) from w
    delta_prev = delta @ self.w[1:, :].T   # (N, in)
    return delta_prev
```

Q: A model currently has a momentum_gamma of 0.5 (input to backwards() in code snippet above). Let's say we increase the momentum factor to 1.0 instead. What changes about the way the model makes weight updates?

Possible answer: My answer is: "The model will rely more on past gradients more for weight updates."

2. [Topic: A1 Part 1 Util Functions]

```python
def normalize_data(inp):
    """
    Normalizes image pixels here to have 0 mean and unit variance.

    args:
        inp : N X d 2D array where N is the number of examples and d is the number of dimensions

    returns:
        normalized inp: N X d 2D array

    """
    #compute average pixel value per col
    mean_px = np.mean(inp, axis=0)

    #compute standard deviation per col
    std_dev = np.std(inp, axis=0)

    #toss out any 0 std devs, replace with 1
    std_dev[std_dev == 0] = 1

    #subtract each pixel by mean px value and divide by standard deviation
    normalized_inp = (inp - mean_px)/std_dev

    return normalized_inp
```

Q: What might happen if we trained a model without using z-score normalization in the code excerpt above?

Possible answers:

My answer is: "Features that are numerically much larger than other ones will be considered far more heavily than features that are smaller and dominate the model's decision process." (see https://www.geeksforgeeks.org/data-analysis/z-score-normalization-definition-and-examples/)

My answer is: "Particularly large features may lead to large gradients which can lead to the exploding gradient problem and lead to huge weight updates that lead to poor model performance."