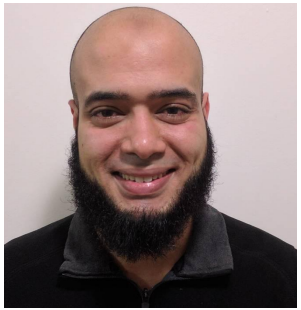
Machine Learning

Homework 1 - Multiclass Classifier

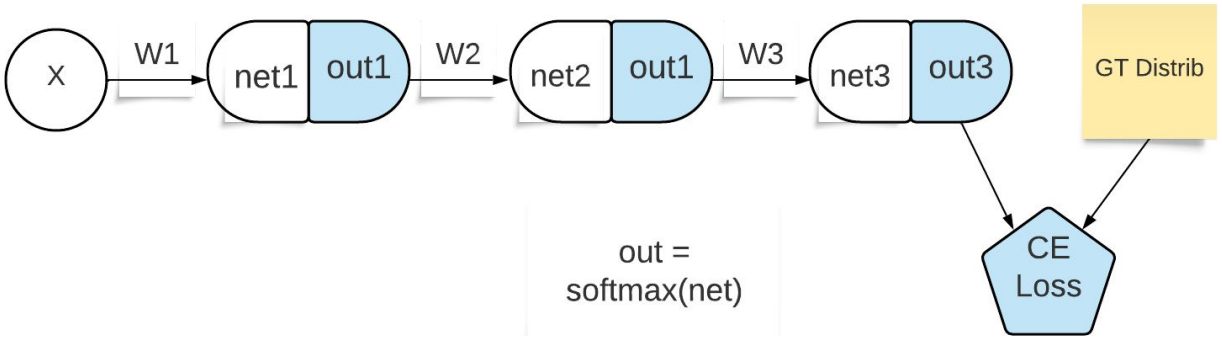
Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / MSc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



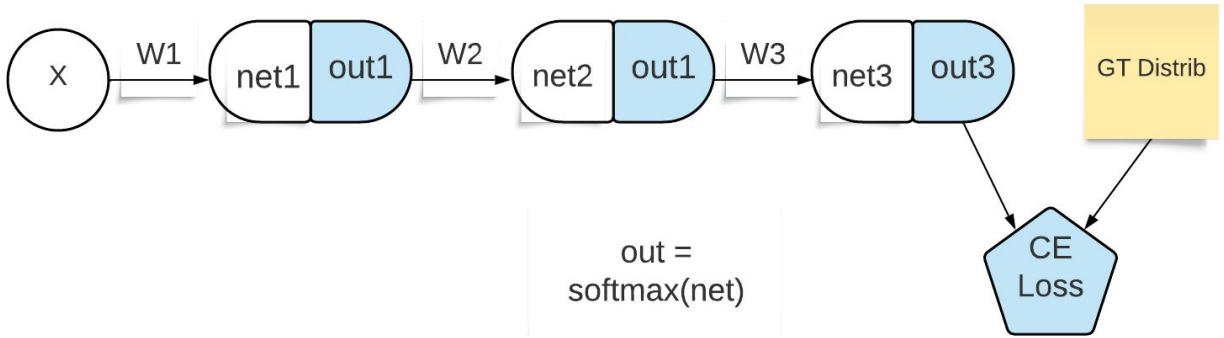
Big Picture

- Your goal is to provide a **vectorized batch** implementation for a **2-hidden** layer neural network to classify MNIST dataset (each image is 784 vector)
  - Assume batch size is 32
  - Input per feedforward: 32 x 784
  - Hidden layers sizes: 20 and 15
  - Output layer 32 x 10 (10 for digits 0, 1, 2, 3….9)
  - Use activation function tanh in ou1 and out2. Use softmax for out3
- Recall
  - ∂L / ∂net3 = out3 - gt3



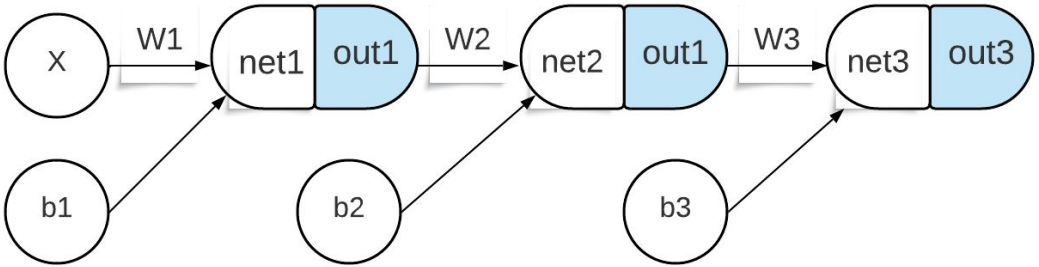
Vectorized Implementation

- In NN regression, we implemented an iterative version for NN
  - However this is slow
- Vectorized implementation depends on 1D and 2D numpy
  - It is way shorter code and runs faster on GPU
  - But needs matrix multiplication coding mentality!
- This exercise is your chance to explore that
  - In addition, you will attach bias also to the nodes



Vectorized Implementation

- You will have 3 2D arrays: W1, W2, W3
  - W1: input_dim x hidden_dim1 784 x 20
  - W2: hidden_dim1 x hidden_dim2 20 x 15
  - W3: hidden_dim2, output_dim 15 x 10
- In addition 3 bias vectors of length:
  - b1: 1 x 20 b2: 1 x 15 b3: 1 x 10
- net1 = x_batch x W1 + b1
  - (32 x 784) x (784x20) + (1x20) = (32 x 20) + (1x20) = (32 x 20) [broadcasting]



Batch Vectorized Implementation

- In batch implementation, as we saw we just make the input 2D (e.g. 32 x 784) instead of 784). This utilize better the GPU
- However, this also affects our softmax and cross entropy implementation

To make your life easier

- Before writing the whole code, we will write 4 separate programs
  - Then you merge together
- Programs
  - softmax_batch version
  - cross entropy batch version
  - Feedforward function (batch vectorized)
  - Backward function (batch vectorized)
- Download inputs and outputs from here
  - You will feed this data to your code
  - The compare the answer
  - In fact, use the template function provided

Numpy: axis

- Learn about using axis (e.g. axis=1, axis=0)

```
arr2d = np.array([[1, 2, 3, 4],
                  [1, 3, 7, 8],
                  [2, 6, 10, 5]], dtype=np.int32)


print(np.max(arr2d))            # 10
print(np.max(arr2d, axis=0))    # [ 2  6 10  8]
print(np.max(arr2d, axis=1))    # [ 4  8 10]
```

Numpy: keepdims

- Learn about keepdims

```
arr2d = np.array([[1, 2, 3, 4],
                  [1, 3, 7, 8],
                  [2, 6, 10, 5]], dtype=np.int32)

print(np.max(arr2d, axis=0, keepdims=True))
# [[ 2  6 10  8]]           (1, 4)

print(np.max(arr2d, axis=1, keepdims=True))
# [[ 4] [ 8]  [10]]         (3, 1)
```

Numpy: broadcasting

- Learn about broadcasting rules

```
arr2d = np.array([[1, 2, 3, 4],
                  [1, 3, 7, 8],
                  [2, 6, 10, 5]], dtype=np.int32)

print(arr2d + np.array([[10, 20, 30, 40]], dtype=np.int32))
'''
[[11 22 33 44]
 [11 23 37 48]
 [12 26 40 45]]
'''
print(arr2d + np.array([[100], [200], [300]], dtype=np.int32))
'''
[[101 102 103 104]
 [201 203 207 208]
 [302 306 310 305]]
'''
```

Task #1: Softmax Batch

- Output is (3, 4) where each example softmax is computed

```
def softmax_batch(x):
    ...     # TODO


if __name__ == '__main__':
    # batch of 3 examples, each is 4 features
    arr2d = np.array([[1, 2, 3, 4],
                      [1, 3, 7, 8],
                      [2, 6, 10, 5]])

    your_answer = softmax_batch(arr2d)
    right_answer = np.load(os.path.join(npy_root_dir, 'softmax.npy'))

    if np.allclose(your_answer, right_answer, atol=1e-6):
        print("Good job")
```

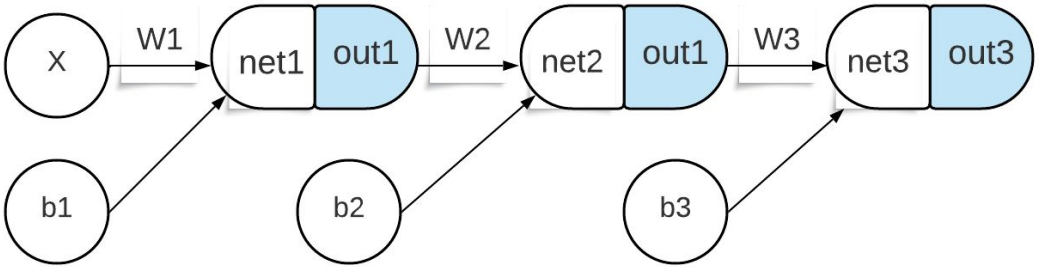
Task #2: Cross Entropy Batch

- Return a single value, the average of the cross entropy for each example

```
def cross_entropy_batch(y_true, y_pred):
    ...     # TODO


if __name__ == '__main__':
    # One-hot encoded true labels
    y_true = np.array([[1, 0],
                       [0, 1],
                       [0.8, 0.2]])     # soft labels for last example

    # Predicted probabilities
    y_pred = np.array([[0.9, 0.1],
                       [0.2, 0.8],
                       [0.7, 0.3]])

    your_answer = cross_entropy_batch(y_true, y_pred)
    right_answer = 0.284879527662735
```

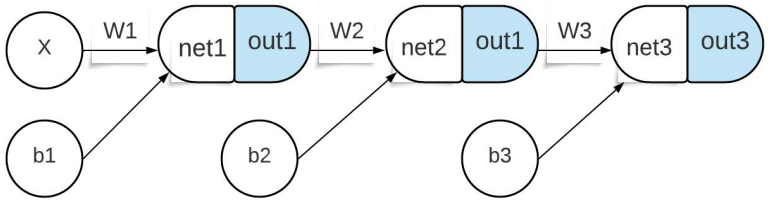
Task #3: Feedforward Vectorized Batch

```
def forward(X_batch, W1, b1, W2, b2, W3, b3):
    net1 = np.dot(X_batch, W1) + b1
    out1 = -1       # TODO
    net2 = -1       # TODO                  # use tanh
    out2 = -1       # TODO
    net3 = -1       # TODO
    out3 = softmax_batch(net3)

    return net1, out1, net2, out2, net3, out3
```



Task #4: backward Vectorized Batch

- One by one implement and verify in the given comparisons
- Think about the 4 rules we learned before
  - Think how to **simply** vectorize
  - Each dE is **previous** * something
- Now to think about
  - Derivatives for the 3 bases




```
def backward(X_batch, y_batch, W2, W3, out1, out2, out3):
    # remaining logic is same for even the regression cod
    dE_dnet3 = out3 - y_batch
    dE_dout2 = -1       # TODO
    dE_dnet2 = -1       # TODO
    dE_dout1 = -1       # TODO
    dE_dnet1 = -1       # TODO

    dW3 = -1        # TODO
    db3 = -1        # TODO
    dW2 = -1        # TODO
    db2 = -1        # TODO
    dW1 = -1        # TODO
    db1 = -1        # TODO

    return dW1, db1, dW2, db2, dW3, db3, \
           dE_dnet3, dE_dout2, dE_dnet2, dE_dnet1
```

Task 5: Full Classifier

- Use the given template to build the full code
- Using the 5k sample of data, you can get 75% and higher
    - This is enough in this task to indicate code correctness

About SKlearn

- The activation function of output layer is based on the task
  - **MLPRegressor**: The output layer's activation function is **identity** (unbounded)
  - **MLPClassifier**:
    - For multi-class classification, the output layer uses the **softmax activation** function
    - For binary classification, it uses the **logistic sigmoid** activation function.
- Intermediate activations
  - All applied with the same given activation
  - One from ['identity', 'logistic', 'relu', 'softmax', 'tanh']
  - Practically speaking: we either selects **relu or tanh**

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."