ML HW3

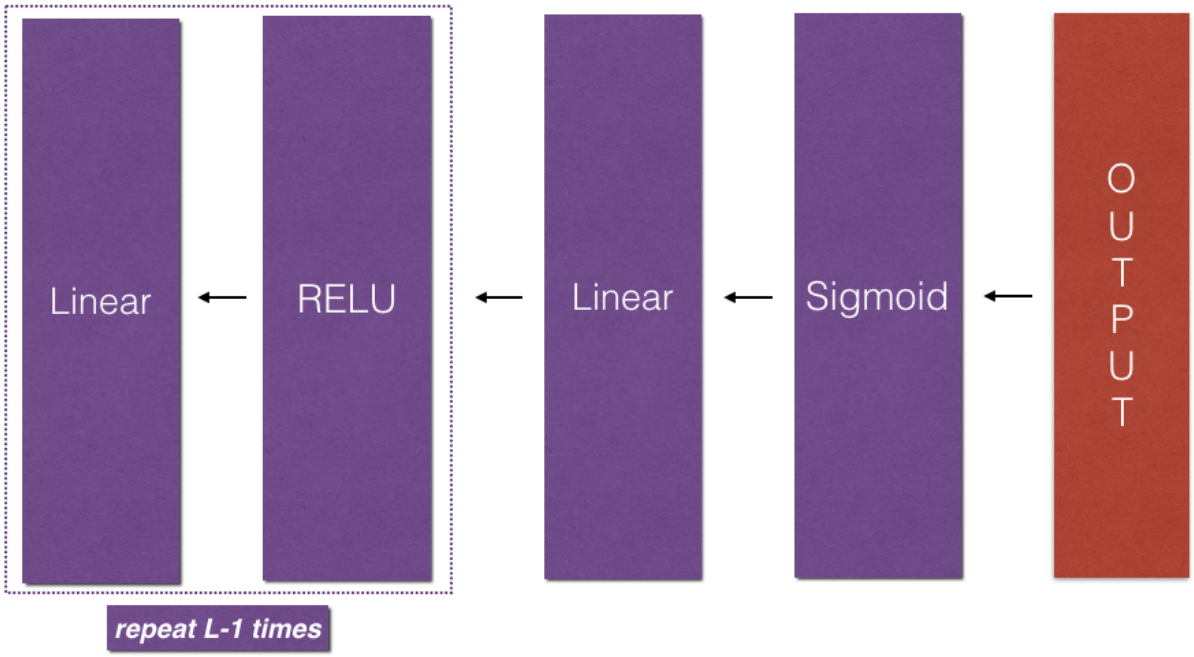
by 108062138 Po-Yu,Wu

hackmd: https://hackmd.io/@sBeNJ4fqRNqa67PhyWWV4A/BJbs9rJtt (https://hackmd.io/@sBeNJ4fqRNqa67PhyWWV4A/BJbs9rJtt)

source code: https://github.com/108062138/MLlab3.git (https://github.com/108062138/MLlab3.git)

BASIC PART

- problem's encounter and solve:mixing up caches:
  - There two types of cache: linear cache and activation cache. I first naively see them as the same A without knowing their meaning. But TA offers a graph, pointing out my misunderstanding:



  - According to the graph, it sudden dawned to me that the linear cache is the A that used to perform W@A+b. The result of W@A+b is then stored in the activation cache. Thus, we need to store the linear cache before the activation cache(forward part).

- how to build the binary classifier
  - The build process includes primary 5 steps: `init` / `froward` / `compute cross-entropy cost` / `backward` / `evaulate` .
    - `init` : Use He-initialization to init. each weight

- `forward` :
  - `linear forward` : It calculates Z=W@A+b and return both Z and cache(linear cache which is A)
  - `activation forward` : The last layer sigmoid/softmax apply sigmoid and the other layers apply relu. It returns both A(which is Z's acitvated result) and cache(activation cache which is Z(Z=W@A+b))
- `compute cross-entropy cost` J: Just implement TA's equation:

$$-\frac{1}{m}\sum_{i=1}^{m}(y^{(i)}\log\left(a^{[L](i)}\right))$$

- `backward update` : Compute dW,db,dA respectively(also from TA's equation) from dZ(later layer) for each layer and store them in grads. dA will be used to update former layer, dW and dA will update current layer

```
# Loop from l=L-2 to l=0
for l in reversed(range(L-1)):
    # lth layer: (RELU -> LINEAR) gradients.
    # Inputs: "grads["dA" + str(l + 1)], current_cache". Outputs: "grads["dA" + str(l)] , grads["dW" + str(l + 1)] , grads["db" + str(l
    current_cache = caches[l]
    dA_prev_temp, dW_temp, db_temp = linear_activation_backward(grads["dA" + str(l + 1)], current_cache, activation = "relu")
    grads["dA" + str(l)] = dA_prev_temp
    grads["dW" + str(l + 1)] = dW_temp
    grads["db" + str(l + 1)] = db_temp
```

- `evaulate` : TA has finished it for me.

> Noted that `froward` / `compute cross-entropy cost` / `backward` will be called #layers times in each iteration.

- parameter tuning:
  - Err, I just apply the parameter inside the comment and I get the fine accuracy!

BONUS PART

- problem encounter and solve
  - data type of numpy matrix is ambiguous
    - when implementing sigmoid/relu, I used to directly write `A=Z` , ignoring the fact that `A[i][j]` 's data type is int. This innocent ignorance directly truncate the calculated value that is less than 1(which takes me a while to figure it out). I hadn't met this bug until I implemented `linear_activation_backward` and get a bunch of 0. To tackle it, I add `A=A.astype(np.float64)` to make sure A's data type is float64.

    - data set’s size is too large
        - my compute don’t have GPU so I utilize google colab to accelerate the training speed.

- how to build the multiclass classifier

    - the approach is basicly same as building the binary classifier. The diffeerence lays in the last layer’s activation function is softmax instead of sigmoid. One thing need to be noticed is that the #class becomes 4 in this multiclass case.
    - With the aformention statement, the build process includes primary 5 steps:
      `init` / `froward` / `compute cross-entropy cost` / `backward` / `evaulate` .

- extra effort to improve the model: augument each layer’s size and increase #of layers. Besides, I enlarge the learning rate to reduce the training time and increase the #of iterations.

SPECIAL CREDIT:

- 李昕威:教我看懂 `L-Model Backward` /教我分辨 `cache`
- LIM-JIA-HE: 助教的.ipynb寫得很好，手把手教會我搭建neural network，讚啦