Deep Learning Specialization

1. Neural Networks & Deep Learning

- Linear Activation

  - It allows multiple outputs, not just binary like step functions.
  - It has 2 major problems:
    - not possible to use backpropagation to train models
    - all layers of NN collapse into one.

- Non-Linear Activations

  - They allow complex mappings between the network's inputs & outputs. They actually address the 2 major fallbacks of linear activations.

- Overfitting

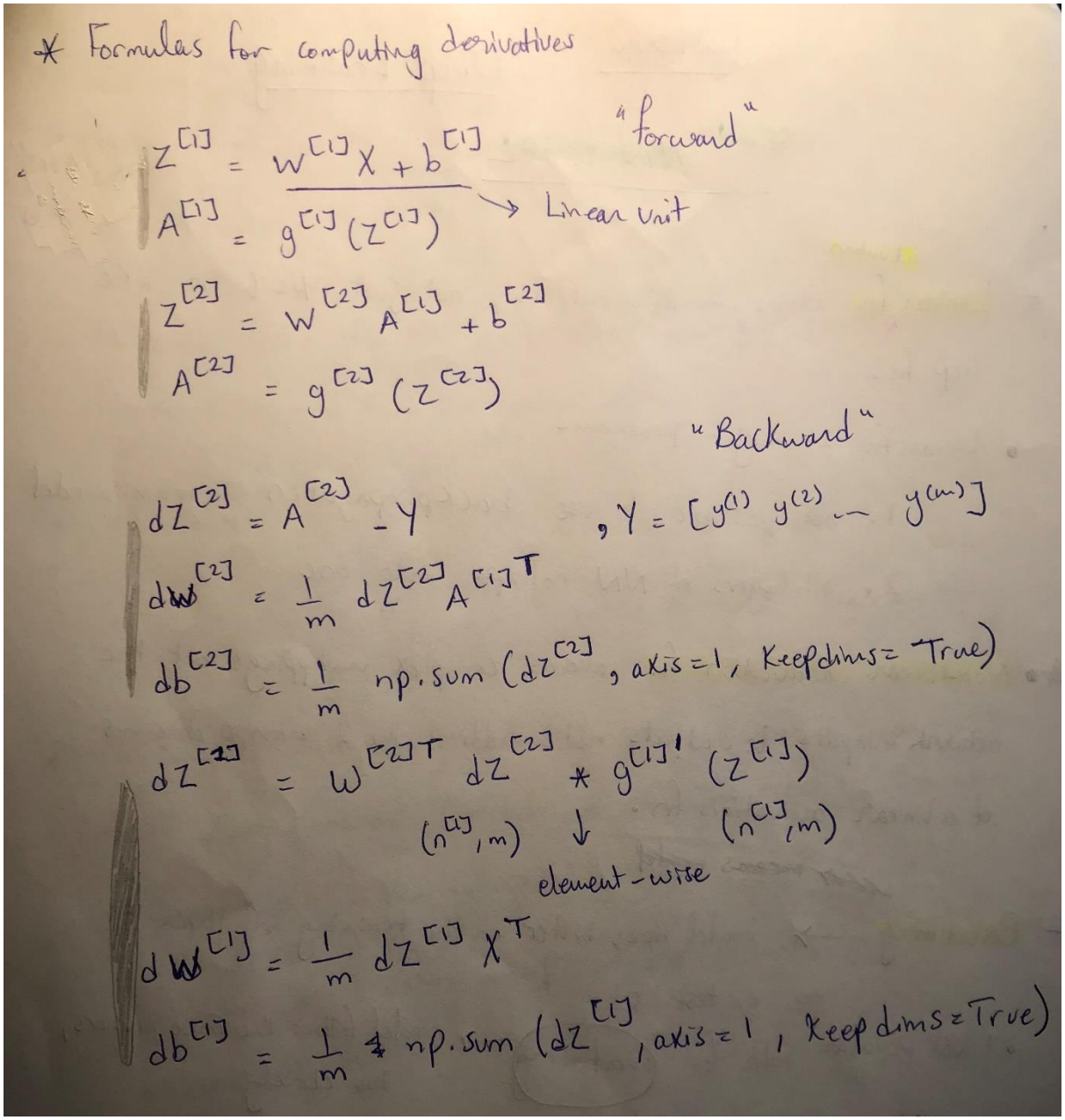
  - model does better on the training set than on test set.
    i.e. model follow the noise, errors too closely.
  - Most popular solutions for overfitting include:
    - cross-validation
    - train with more data
    - remove features
    - early stopping
    - regularization
    - ensembling (bagging - boosting)

- Formulas for computing derivatives

  - **tanh** activation is zero centred, it usually works better than sigmoid for hidden layers because the mean of its output is closer to zero; so it centres the data better for the next layer.
  - For **tanh** activation function, using np.random.randn(.,.) * 1000 will cause the inputs of tanh to be also very large, thus cause gradients to be close to zero. The optimization algorithm will thus become slow.

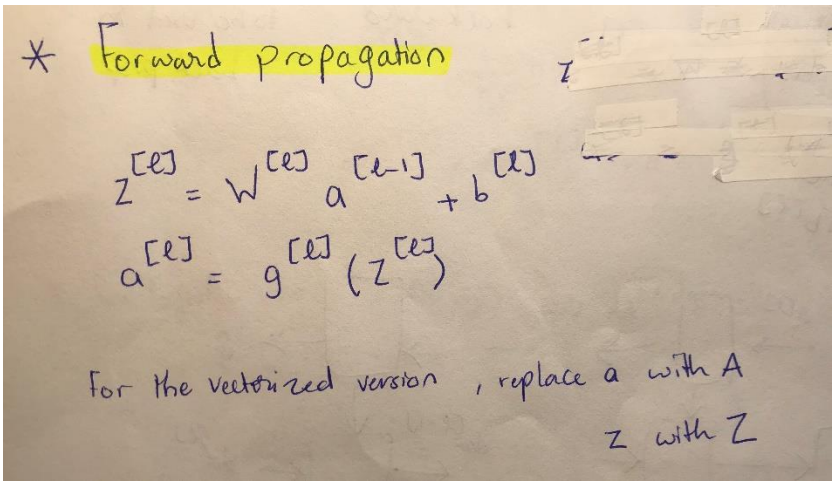
- Initializing **w** to zero will result a symmetric **w**, which is typically useless having all rows being the same. That's why we use np.random.rand(a,b) * 0.01
  yet with **b** = np.zeros(a,b)
- Initializing both **w** & **b** to be zero will cause each neuron in the first layer to perform the same computation. So even after multiple iterations of gradient descent each neuron in the layer will be computing the same thing as other neurons.

- Logistic regression does not have a hidden layer.

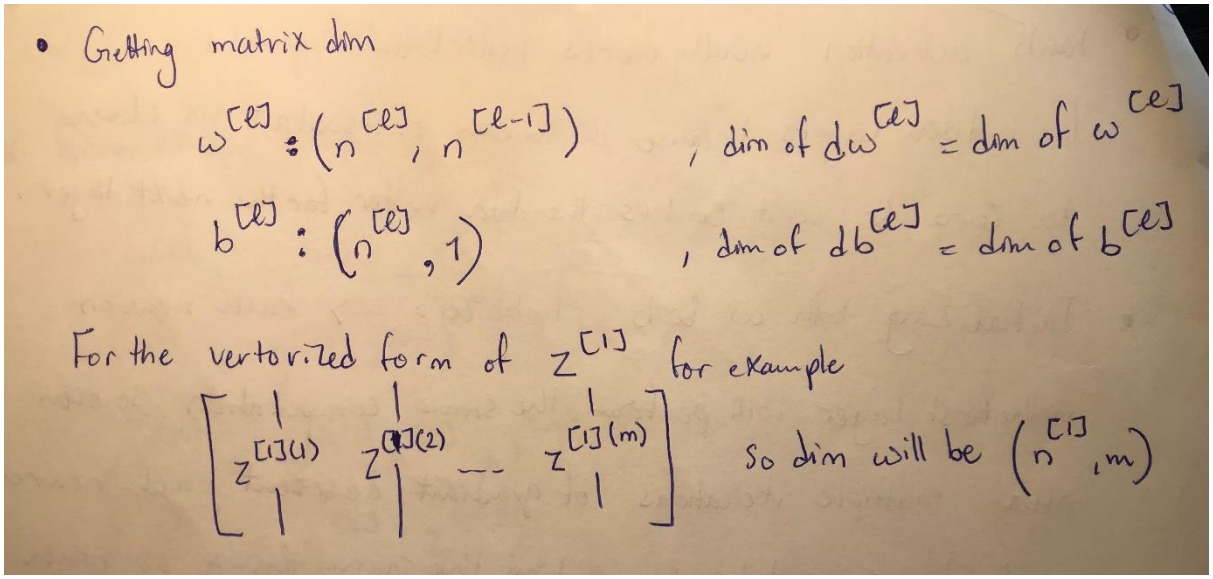


- Forward Propagation

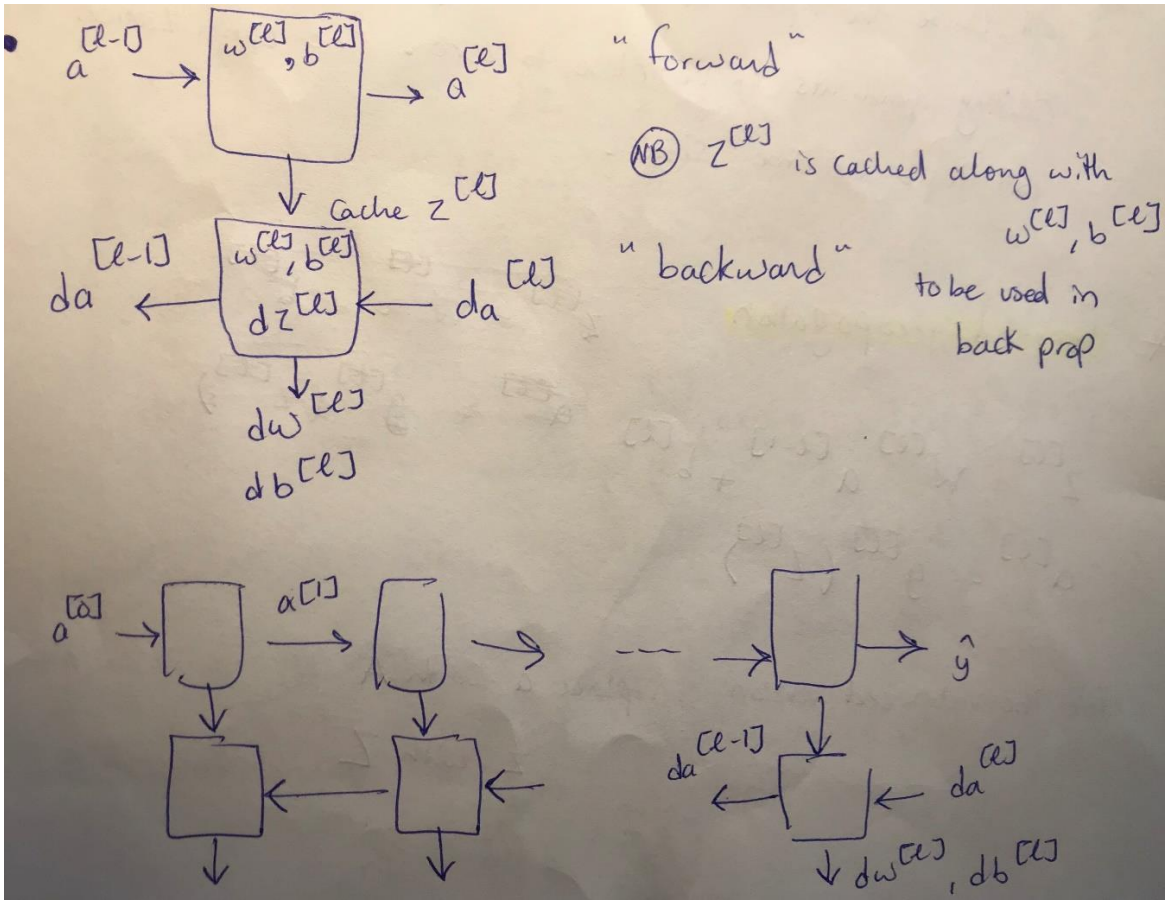
  - Forward propagation equations:



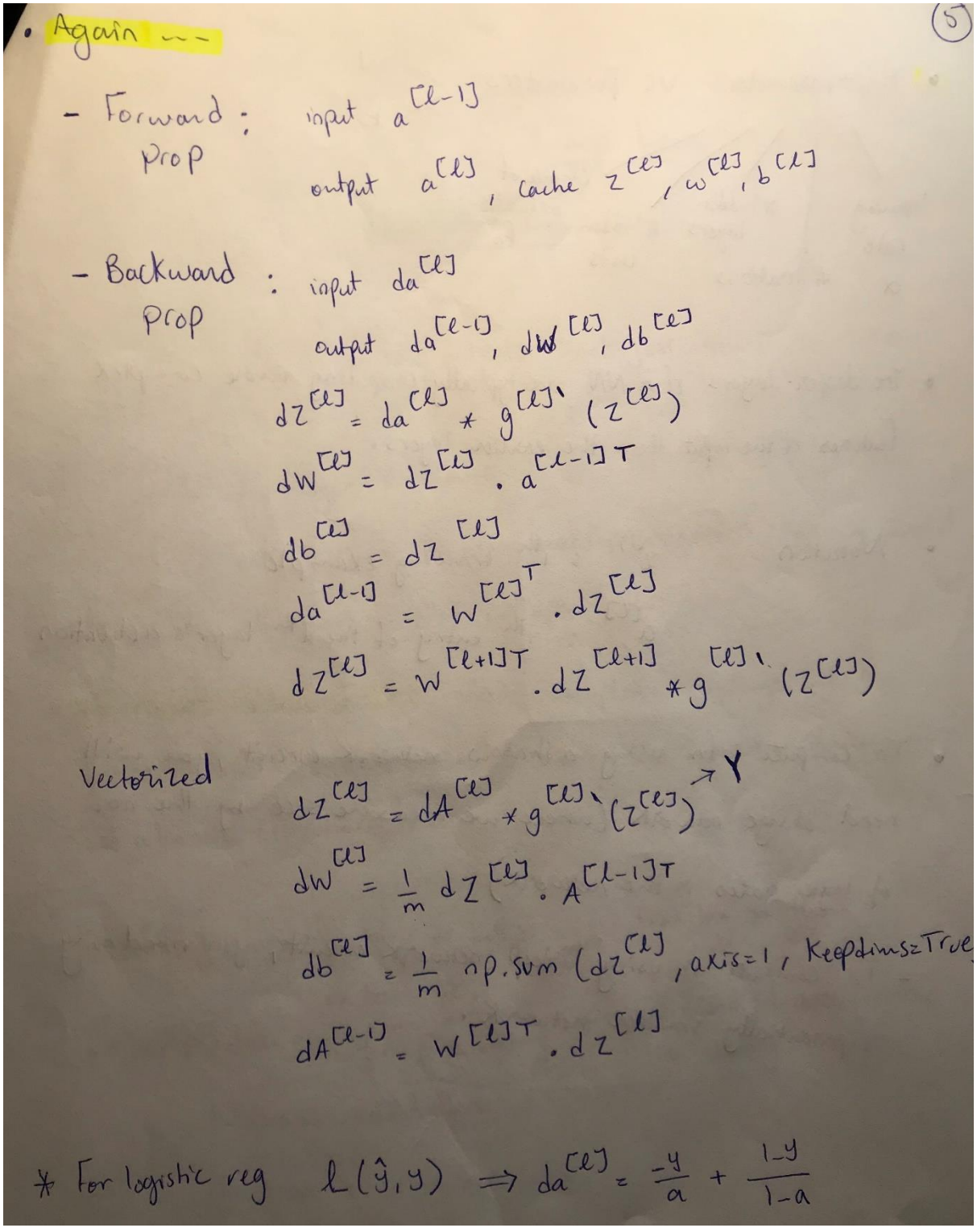
  - getting matrix dimensions for **w** & **b**:



- block diagram: how forward & backward props work



- summary of all equations:



• Again ---

(5)

- Forward Prop : input $a^{[l-1]}$

output $a^{[l]}$, cache $z^{[l]}, w^{[l]}, b^{[l]}$

- Backward Prop : input $da^{[l]}$

output $da^{[l-1]}, dw^{[l]}, db^{[l]}$

$$dz^{[l]} = da^{[l]} * g^{[l]'}(z^{[l]})$$

$$dw^{[l]} = dz^{[l]} \cdot a^{[l-1]T}$$

$$db^{[l]} = dz^{[l]}$$

$$da^{[l-1]} = w^{[l]T} \cdot dz^{[l]}$$

$$dz^{[l]} = w^{[l+1]T} \cdot dz^{[l+1]} * g^{[l]'}(z^{[l]})$$

Vectorized

$$dz^{[l]} = dA^{[l]} * g^{[l]'}(z^{[l]}) \rightarrow Y$$

$$dw^{[l]} = \frac{1}{m} dz^{[l]} \cdot A^{[l-1]T}$$

$$db^{[l]} = \frac{1}{m} np.sum(dz^{[l]}, axis=1, keepdims=True$$

$$dA^{[l-1]} = w^{[l]T} \cdot dz^{[l]}$$

* For logistic reg $\ell(\hat{y}, y) \Rightarrow da^{[l]} = \frac{-y}{a} + \frac{1-y}{1-a}$

- Hyperparameters vs Parameters

  - Hyperparameters: learning rate, # iterations, # hidden layers, choice of activation function.
  - Parameters: w & b.

  - The deeper layers of a NN are typically computing more complex features of the input than the earlier ones.

  - Notation: $x^{(i)}$: i^{th} training example.

    $a_i^{[l]}$: i^{th} entry of the l^{th} layer's activation.

  - To compute a function using a shallow network circuit, you will need a large network; where we measure size by # of logic gates in the network.
    To compute it using a deep network circuit, you need only exponentially smaller network.