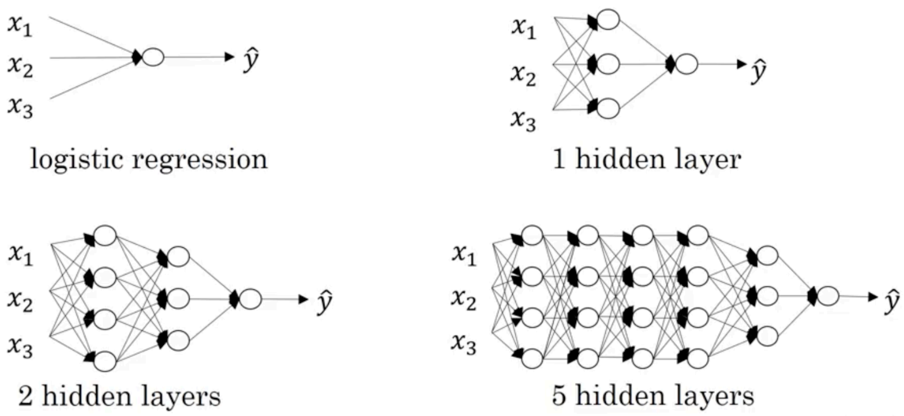
2025-07-17T09-49-15
Tags: DLS

DLS C1 week 4

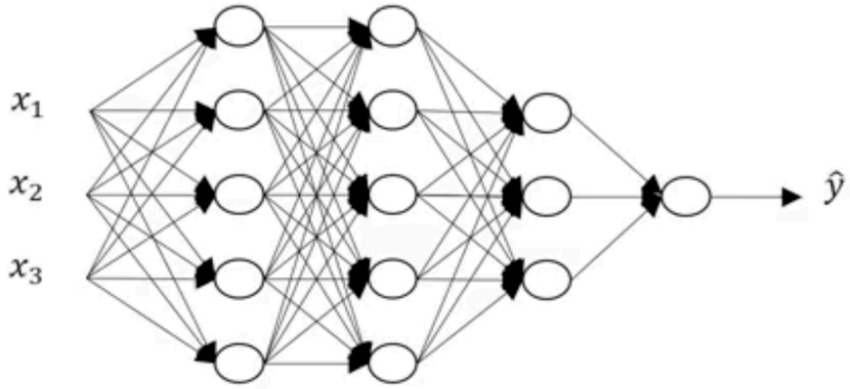
Deep L-layer neural network



Use L to denote the number of layers.
Use $n^{[l]}$ to denote the number of nodes or units in layer l.
Use $a^{[l]}$ to denote the activations in layer l s.t. $a^{[l]} = g^{[l]}(z^{[l]})$.
Use $w^{[l]}$ to denote the weights for $z^{[l]}$.

$n^{[0]} = n_x = 3$: the input layer has 3 units.

Forward propagation in a deep network



$x = a^{[0]}$
$z^{[1]} = w^{[1]}x + b^{[1]}, \quad a^{[1]} = g^{[1]}(z^{[1]})$
$z^{[2]} = w^{[2]}a^{[1]} + b^{[2]}, \quad a^{[2]} = g^{[2]}(z^{[2]})$
$z^{[3]} = w^{[3]}a^{[2]} + b^{[3]}, \quad a^{[3]} = g^{[3]}(z^{[3]})$
$z^{[4]} = w^{[4]}a^{[3]} + b^{[4]}, \quad a^{[4]} = g^{[4]}(z^{[4]})$

general form: $z^{[l]} = w^{[l]}a^{[l-1]} + b^{[l]}, \quad a^{[l]} = g^{[l]}(z^{[l]})$

vectorized form

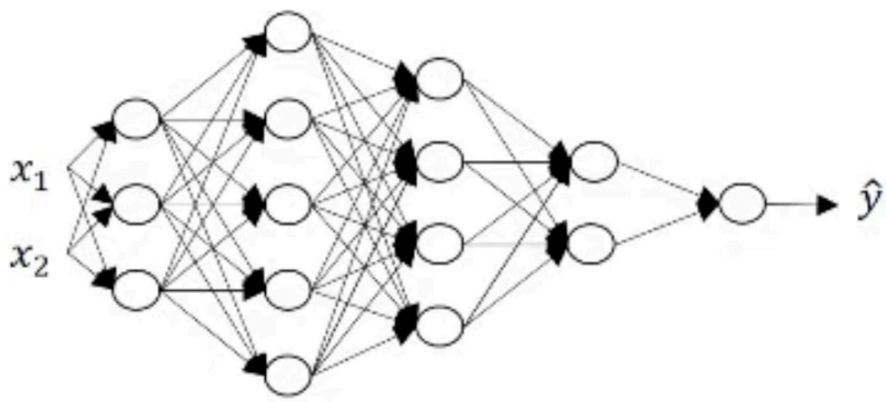
$X = A^{[0]}$
$Z^{[1]} = w^{[1]}X + b^{[1]}, \quad A^{[1]} = g^{[1]}(Z^{[1]})$
$Z^{[2]} = w^{[2]}A^{[1]} + b^{[2]}, \quad A^{[2]} = g^{[2]}(Z^{[2]})$
$Z^{[3]} = w^{[3]}A^{[2]} + b^{[3]}, \quad A^{[3]} = g^{[3]}(Z^{[3]})$
$Z^{[4]} = w^{[4]}A^{[3]} + b^{[4]}, \quad A^{[4]} = g^{[4]}(Z^{[4]})$
$\hat{Y} = g(Z^{[4]}) = A^{[4]}$

Remember that:

$$Z^{[l]} = \begin{bmatrix} | & | & & | \\ z^{[l](1)} & z^{[l](2)} & \dots & z^{[l](m)} \\ | & | & & | \end{bmatrix}$$

Know that it’s perfectly fine to use a for-loop to compute activations layer by layer (from 1 to L), as there’s no practical way to avoid it.

Matrix dimensions



$z^{[1]} = w^{[1]}x + b^{[1]}$
If $z^{[1]}$ is $(n^{[1]}, 1) = (3, 1)$ and $x = a^{[0]}$ is $(n^{[0]}, 1) = (2, 1)$ then $w^{[1]}$ must be $(3, 2) = (n^{[1]}, n^{[0]})$.

Here are the general forms:

- $z^{[l]}$ is $(n^{[l]}, 1)$
- $w^{[l]}$ is $(n^{[l]}, n^{[l-1]})$
- $b^{[l]}$ is $(n^{[l]}, 1)$
- $dw^{[l]}$ is $(n^{[l]}, n^{[l-1]})$
- $db^{[l]}$ is $(n^{[l]}, 1)$

Vectorized: $Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]}$

- $Z^{[l]}$ is $(n^{[l]}, m)$
- $W^{[l]}$ is $(n^{[l]}, n^{[l-1]})$
- $A^{[l-1]}$ is $(n^{[l-1]}, m)$
- $b^{[l]}$ is $(n^{[l]}, 1) \rightarrow (n^{[l]}, m)$ (broadcasted)
- $dZ^{[l]}$ is $(n^{[l]}, m)$
- $dA^{[l]}$ is $(n^{[l]}, m)$

Why deep representations

Deep neural networks containing multiple hidden layers enables hierachical feature learning.

Early layers captures simple features.

- Edges for images.
- Pitch and tones in audio.

Later layers combine preceeding features into more complex structures.

- Edges into face parts (e.g., eyes, nose).
- Pitch to phenomes to words to phrases to sentences.

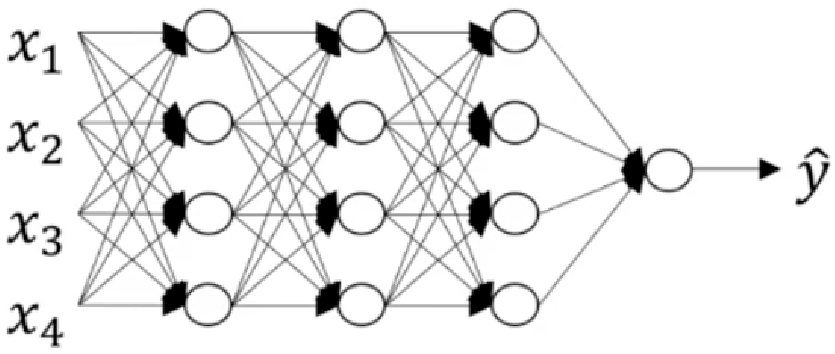
Given $y = x_1 \oplus x_2 \oplus x_3 \oplus \cdot \oplus x_n$

- Shallow networks requires $O(2^n)$.
- Deep networks only requires $O(\log n)$.

Remarks:

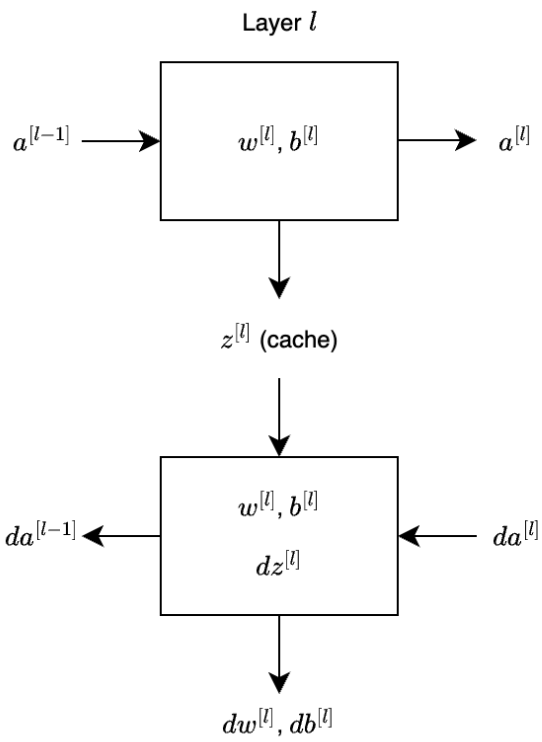
- Deep networks are effective but depth is not always required.
- When starting out on a new problem, start simple. Begin with logistic regression or shallow networks.
- Treat depth as a hyperparameter. Adjust it based on performance.
- There's a trend that extremely deep networks achieve SOTA results.

DNN building blocks



For a layer l:

- Parameters: $w^{[l]}$, $b^{[l]}$
- Forward prop: input $a^{[l-1]}$, output $a^{[l]}$, cache $z^{[l]}$
  - $z^{[l]} = w^{[l]}a^{[l-1]} + b^{[l]}$
  - $a^{[l]} = g^{[l]}(z^{[l]})$
- Backward prop: input $da^{[l]}$, $z^{[l]}$ (from cache), output $da^{[l-1]}$, $dw^{[l]}$, $db^{[l]}$



During implementation, include $w^{[l]}$, $b^{[l]}$ to the cache.

DNN forward and backward propagation

Forward propagation for layer l: input $a^{[l-1]}$, output $a^{[l]}$, cache $z^{[l]}$

- $Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$
- $A^{[l]} = g^{[l]}(Z^{[l]})$

Backward propagation for layer l: input $da^{[l]}$, output $da^{[l-1]}$, $dW^{[l]}$, $db^{[l]}$

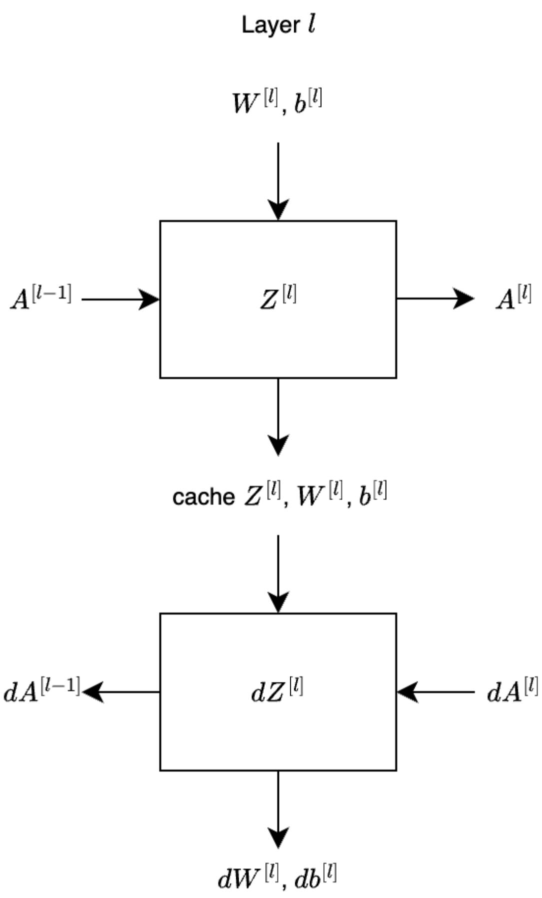
- $dZ^{[l]} = dA^{[l]} * g^{[l]\prime}(Z^{[l]})$
- $dW^{[l]} = \frac{1}{m} dZ^{[l]} \cdot A^{[l-1]\intercal}$
- $db^{[l]} = \frac{1}{m} \text{np.sum}(dZ^{[l]}, \text{axis=1, keepdims=True})$
- $dA^{[l-1]} = W^{[l]\intercal} \cdot dZ^{[l]}$

Summary of the whole training process:

1. Forward pass: start with $A^{[0]} = X$, propagate through layers to get $\hat{Y} = A^{[l]}$.
2. Compute loss based on $\hat{y}$ and true y.
3. Backward pass: start with $dA^{[l]}$, propagate backward to compute every gradient.
4. Parameter updates: $w^{[l]} := w^{[l]} - \alpha \cdot dw^{[l]}, \quad b^{[l]} := b^{[l]} - \alpha \cdot db^{[l]}$.

Know that $dA^{[l]} = -\frac{Y}{\hat{Y}} + \frac{1-Y}{1-\hat{Y}}$ (for logistic regression loss).

Look here for the derivations.



Parameters vs. Hyperparameters

Parameters	Hyperparameters
	- Set before training.

- Learned automatically during training.	- Controls training. - Indirectly affects parameters.

Examples of parameters:

- $W^{[1]}$, $b^{[1]}$, $W^{[2]}$, $b^{[2]}$, $\ldots$

Examples of hyperparameters:

- learning rate α
- no. of interations
- no. of hidden layers L
- no. of hidden units $n^{[1]}$, $n^{[2]}$, $\ldots$
- choice of activation functions (ReLU, sigmoid, etc.)
- momentum term, minibatch size, regularization parameters (covered later)

Hyperparameters are not optimized by gradient descent. They are set manually.

There is no universal best value for a hyperparameter. It depends on the following:

- data
- model architecture
- hardware

Tuning hyperparameters is an empirical process:

1. Start with a reasonable guess.
2. Refine iteratively.
3. Track hyperparameters and outcomes for comparison.
4. Validate using cross validation or a dedicated hold-out set.
5. Re-tune periodically as data, hardware, and model architecture changes.

For example:

1. If $\alpha = 0.01$, cost decreases slowly. α is too small.
2. If $\alpha = 0.05$, cost diverges. α is too large.