Week 1b - Regularizing your Neural network

Friday, August 14, 2020 11:25 PM

① Regularization

- For logistic regression lambd ← λ → regularization in Python parameter

$$\min_{w,b} J(w,b)$$

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\|w\|_2^2 + \cancel{\frac{\lambda}{2m}b^2}$$ (omit)

L_2 regularisation: $\|w\|_2^2 = \sum_{i=1}^{n_x} w_j^2 = w^T w$ → most used

L_1 regularisation: $\frac{\lambda}{m}\sum_{i=1}^{n_x}|w| = \frac{\lambda}{2m}\|w\|_1$ → w will be sparse (lot of zeros)

- For a neural network

$$\Rightarrow J(w^{[1]}, b^{[1]}, \ldots, w^{[L]}, b^{[L]}) = \frac{1}{m}\sum_{i=1}^{n}\mathcal{L}(\hat{y}^{[i]}, y^{[i]}) + \frac{\lambda}{2m}\sum_{l=1}^{L}\|w^{[l]}\|^2$$

where, $\|w^{[l]}\|^2 = \sum_{i=1}^{n^{[l]}}\sum_{j=1}^{n^{[l-1]}}(w_{ij}^{[l]})^2$ $\quad w: [n^{[l]}, n^{[l-1]}]$

↓ "Frobenius norm"

$$\Rightarrow \boxed{dw^{[l]} = (\text{from backprop}) + \frac{\lambda}{m}w^{[l]}} \quad \frac{\partial J}{\partial w} = dw^{[l]}$$

$$\Rightarrow w^{[l]} := w^{[l]} - \alpha\, dw^{[l]}$$

"weight decay"

$$w^{[l]} := w^{[l]} - \alpha\left[(\text{from backprop}) + \frac{\lambda}{m}w^{[l]}\right]$$

$$= w^{[l]} - \alpha\frac{\lambda}{m}w^{[l]} - \alpha(\text{from backprop})$$

$$w^{[l]}\left(1 - \frac{\alpha\lambda}{m}\right)$$ → slightly less than 1

② Why does regularization reduce overfitting

→ Intuitively as λ gets bigger, many $w^{[l]}$ get closer to zero and cancel their effect leading to simpler network

→ In tanh function,

$\lambda\uparrow \Rightarrow w^{[l]}\downarrow$

↓ After regularization

$\Rightarrow z^{[l]} = w^{[l]}a^{[l-1]} + b^{[l]}$

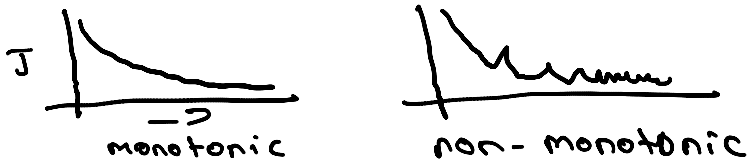
every layer ≈ linear

→ λ can be considered as a penalty

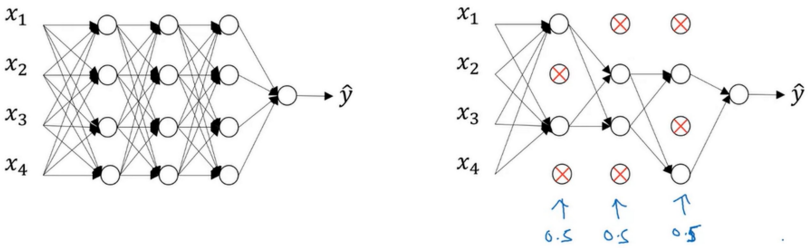
every layer is linear

→ λ can be considered as a penalty for the parameter being too large

→ It makes J reduce monotonically



③ Dropout regularization



eliminate nodes at random to reduce size of network & thereby reducing overfitting

→ Implementing dropout ("inverted dropout")

Consider a random layer (3 in this case)

dropout vector

```
d3 = np.random.rand(a3.shape[0], a3.shape[1]) < keep_prob
```

prob. of elimination

```
a3 = np.multiply(a3, d3)
a3 /= keep_prob
```

→ zeroes out nodes where d3 is 0

- d3 will be a boolean array
  eg: if keep_prob = 0.8,
  the prob. of a node not being eliminated is 80%.
  ⇒ 50 units ⇒ 40 units
  ⇓
  value of $z^{[4]}$ is lesser
  hence we divide by keep_prob to adjust for it

→ making predictions at test-time

$a^{[0]} = X$

- dropout isn't used in test as it has already influenced training
- If dropout is used in prediction, different outputs are obtained every time

④ Understanding dropout

④ Understanding dropout

→ Intuition:
- Can't rely on any one feature, so have to spread out weights
- Since any feature can disappear, the neuron doesn't decide too much based on one input
- Results in shrinking weights and spreaded out weights

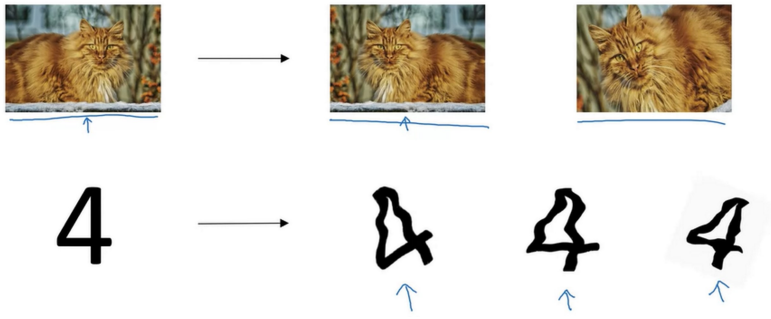
Note:
- keep_prob = 1 generally for input since we don't want to loose useful information
- used only if overfitting is a problem
  eg: In computer vision

→ Cost function is no longer well-defined since every time a different function is computed

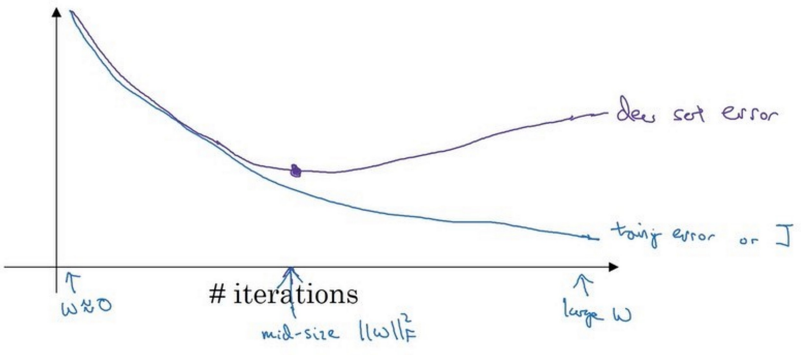
⑤ Other regularization methods

→ Data augmentation

- By flipping images, we can effectively double the no. of data
- Random distortions / zooms can also be applied.



→ Early stopping



- ML has 2 tasks primarily,

- ML has 2 tasks primarily, known as Orthogonalization

  i. Optimize cost fn. } - gradient ....

  ii. Not overfit } - Regularization ....

  Early stopping combines these two and limits our options and complicates the tuning process