Improving Deep Neural Networks

§1 Practical aspects of Learning Applications

§2 Optimization Algorithms

1. Mini-Batch Gradient Descent

Below is the vectorized implementation of one iteration of Gradient Descent using Mini-Batch:
Repeat for t = 1, …, 5,000 (5,000 is the number of Mini-Batches) {

Do Forward Propagation on $X^{\{t\}}$: $Z^{[1]} = w^{[1]}X^{\{t\}} + b^{[1]}$; $A^{[1]} = g^{[1]}\left(Z^{[1]}\right)$, …, $A^{[L]} = g^{[L]}\left(Z^{[L]}\right)$

Compute cost function: $J^{\{t\}} = \frac{1}{1000}\sum_{i=1}^{l} L(\hat{Y}^{(i)} - y^{(i)}) + \frac{\lambda}{2*1000}\sum_{l}\left\|w^{[l]}\right\|_F^2$

Backward Propagation to compute gradients cost by taking derivative of $J^{\{t\}}$ to each w and b.
Update w and b: $w^{[l]} = w^{[l]} - \alpha dw^{[l]}$; $b^{[l]} = b^{[l]} - \alpha db^{[l]}$

}

2. Understanding Mini-Batch Gradient Descent

- For each iteration, Gradient Descent can be run for **# of Mini-Batches** times.
- Choose Mini-Batch size:
  - If Mini-Batch size = m (m is the sample size): there is only one batch, and we call this **Batch Gradient Descent**. It will be too long to run each iteration;
  - If Mini-Batch size = 1: every sample is its own batch, and we call this **Stochastic Gradient Descent**.
  - If Mini-Batch size is between 1 and m (usually 1,000) and appropriate, we can achieve fast speeding from both vectorization and making progress without processing the entire data set.

$$V_t = \beta V_{t-1} + (1-\beta)\theta_{t-1}$$

V_t is an approximately averaging value over $1/(1-\beta)$ days.

For each V_t

$$V_t = \frac{V_t}{1-\beta^t}$$

As t grows, $(1-\beta^t) \rightarrow 1$.

§3 Hyperparameter Tuning and Batch Normalization

1. Hyperparameter Tuning

- Tuning Process: (1) Try random values, don't use a grid; (2) From coarse to fine;
- Using an appropriate scale to pick hyperparameters: (1) Pick hyperparameters at random; (2) Appropriate scale for hyperparameters; (3) Hyperparameters for exponentially weighted averages.
- Hyperparameter tuning in practice
  - Panda: Babysitting one model;
  - Caviar: Train many models in paralle

2. Batch Normalization

- Normalizing activations in a network, given some intermediate values in neural network layer l, $z^{[l](1)}, \dots, z^{[l](i)}, \dots, z^{[l](m)}$.

  $\mu = \frac{1}{m}\sum_i z^{[l](i)},\ \sigma^2 = \frac{1}{m}\sum_i\left(z^{[l](i)} - \mu\right) ** 2$

  $z_{norm}^{[l](i)} = \dfrac{z^{[l](i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$

  $\tilde{z}^{[l](i)} = \gamma z_{norm}^{[l](i)} + \beta$

  γ and β are learnable parameters of model.

- Apply Batch Normalization to a Neural Network: Implementation

  For t = 1, …, # of Mini-Batches

  Compute Forward Propagation on $x^{\{t\}}$: In each hidden layer, use Batch Normalization to replace $z^{[l]}$ with $\tilde{z}^{[l]}$.

  Use Backward Propagation to compute $dw^{[l]}$, $d\beta^{[l]}$ and $d\gamma^{[l]}$.

  Update parameter (we can also use momentum, RMSprop and Adam to update, too).

  $w^{[l]} = w^{[l]} - \alpha dw^{[l]};\ \beta^{[l]} = \beta^{[l]} - \alpha d\beta^{[l]};\ \gamma^{[l]} = \gamma^{[l]} - \alpha d\gamma^{[l]}$.

3. Multi-class Classification – Softmax Regression

Given the output has 4 classes, for the output layer

$$z^{[l]} = w^{[l]}a^{[l-1]} + b^{[l]}; a^{[l]} = g^{[l]}(z^{[l]})$$

Both $a^{[l]}$ and $z^{[l]}$ are (4, 1) vectors.

Loss function can be derived as

$$L(y, \hat{y}) = -\sum_{j=1}^{4} y_j log\hat{y}_j$$

If $y = [0, 1, 0, 0]^T$, $L(y, \hat{y}) = -log\hat{y}_2$.

And cost function is

$$J\left(w^{[1]}, b^{[1]}, \dots\right) = \frac{1}{m}\sum_{i=1}^{m} L(y, \hat{y})$$

So Gradient Descent with Sofmax is $dZ^{[L]} = \hat{y} - y$, which is a (4, 1) vector.