Improving Deep Neural Networks (week 2) - Optimization Methods

! Notation: $'da' = \frac{\partial J}{\partial a}$

Stochastic Gradient Descent (SGD)

Each mini-batch has 1 example.

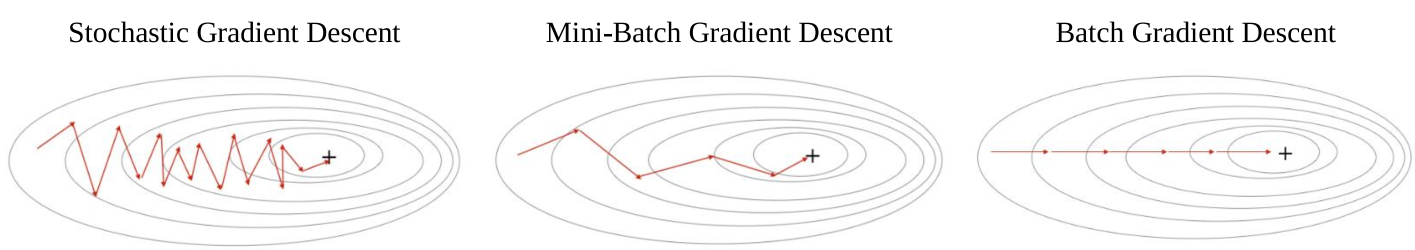
The difference between stochastic gradient descent and (batch) gradient descent:

- **(Batch) Gradient Descent**:

```
X = data_input
Y = labels
parameters = initialize_parameters(layers_dims)
for i in range(0, num_iterations):
    # Forward propagation
    a, caches = forward_propagation(X, parameters)
    # Compute cost.
    cost += compute_cost(a, Y)
    # Backward propagation.
    grads = backward_propagation(a, caches, parameters)
    # Update parameters.
    parameters = update_parameters(parameters, grads)
```

- **Stochastic Gradient Descent**:

```
X = data_input
Y = labels
parameters = initialize_parameters(layers_dims)
for i in range(0, num_iterations):
    for j in range(0, m):
        # Forward propagation
        a, caches = forward_propagation(X[:,j], parameters)
        # Compute cost
        cost += compute_cost(a, Y[:,j])
        # Backward propagation
        grads = backward_propagation(a, caches, parameters)
        # Update parameters.
        parameters = update_parameters(parameters, grads)
```

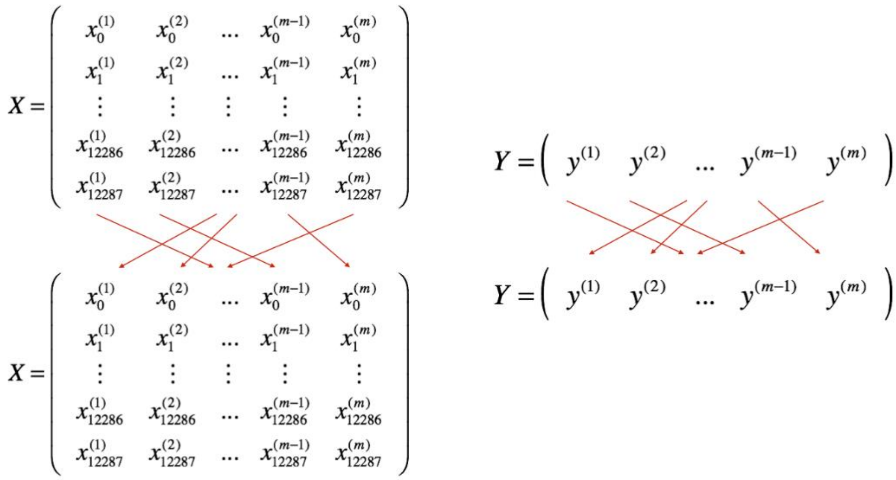


"+" denotes a minimum of the cost.

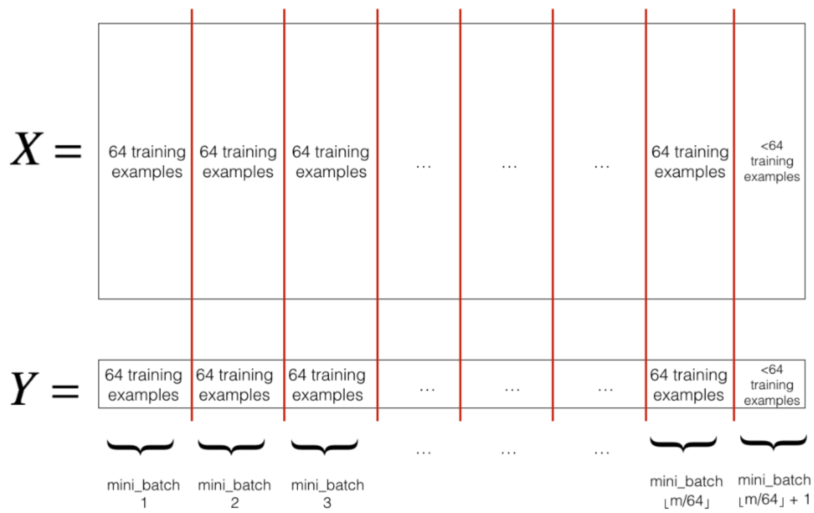
Mini-Batch Gradient descent

random_mini_batches function build mini-batches from the training set (X, Y) in 2 steps:

1) Shuffle – Each column of X and Y represents a training example. The random shuffling is done synchronously between X and Y, such that after the shuffling the i^{th} column of X is the example corresponding to the i^{th} label in Y. The shuffling step ensures that examples will be split randomly into different mini-batches:



2) Partition – Partition the shuffled (X, Y) into mini-batches of size *mini_batch_size* (default 64). The last mini-batch might be smaller:



Gradient Descent with Momentum

initialize_velocity function initializes the velocity v to a dictionary of zeros array.

update_parameters_with_momentum function updates the parameters according to the functions:

$$v_{dW^{[l]}} = \beta \cdot v_{dW^{[l]}} + (1 - \beta) \cdot dW^{[l]}$$

$$v_{db^{[l]}} = \beta \cdot v_{db^{[l]}} + (1 - \beta) \cdot db^{[l]}$$

$$W^{[l]} = W^{[l]} - \alpha \cdot v_{dW^{[l]}}$$

$$b^{[l]} = b^{[l]} - \alpha \cdot v_{db^{[l]}}$$

! α – learning rate

! The velocity is initialized with zeros. So the algorithm will take a few iterations to "build up" velocity and start to take bigger steps.

! If $\beta = 0$ then this just becomes standard gradient descent without momentum.

Choosing β:

The larger the momentum β is, the smoother the update because the more we take the past gradients into account. But if β is too big, it could also smooth out the updates too much.

Common values for β range from 0.8 to 0.999. $\beta = 0.9$ is often a reasonable default.

Adam optimization

initialize_adam function initializes the v, s to a dictionary of zeros array.

update_parameters_with_adam function updates the parameters according to the functions:

$$v_{dW^{[l]}} = \beta_1 \cdot v_{dW^{[l]}} + (1 - \beta_1) \cdot dW^{[l]}$$

$$v_{db^{[l]}} = \beta_1 \cdot v_{db^{[l]}} + (1 - \beta_1) \cdot db^{[l]}$$

$$v_{dW^{[l]}}^{corrected} = \frac{v_{dW^{[l]}}}{1 - \beta_1^t}$$

$$v_{db^{[l]}}^{corrected} = \frac{v_{db^{[l]}}}{1 - \beta_1^t}$$

$$s_{dW^{[l]}} = \beta_2 \cdot s_{dW^{[l]}} + (1 - \beta_2) \cdot {dW^{[l]}}^2$$

$$s_{db^{[l]}} = \beta_2 \cdot s_{db^{[l]}} + (1 - \beta_2) \cdot {db^{[l]}}^2$$

$$s_{dW^{[l]}}^{corrected} = \frac{s_{dW^{[l]}}}{1 - \beta_2^t}$$

$$s_{db^{[l]}}^{corrected} = \frac{s_{db^{[l]}}}{1 - \beta_2^t}$$

$$W^{[l]} = W^{[l]} - \alpha \cdot \frac{v_{dW^{[l]}}^{corrected}}{\sqrt{s_{dW^{[l]}}^{corrected}} + \varepsilon}$$

$$b^{[l]} = b^{[l]} - \alpha \cdot \frac{v_{db^{[l]}}^{corrected}}{\sqrt{s_{db^{[l]}}^{corrected}} + \varepsilon}$$

- t – counts the number of steps taken of Adam
- L – number of layers
- β_1 and β_2 – hyperparameters that control the two exponentially weighted averages
- α – learning rate
- ε – a very small number to avoid dividing by zero

Model with different optimization algorithms

The model implemented can be run with 3 different optimizations by changing *'optimizer'* argument in *model* function.

1) *optimizer = 'gd'*: Mini-batch Gradient descent
2) *optimizer = 'momentum'*: Mini-batch gradient descent with momentum
3) *optimizer = 'adam'*: Mini-batch with Adam mode