Learning and Loss

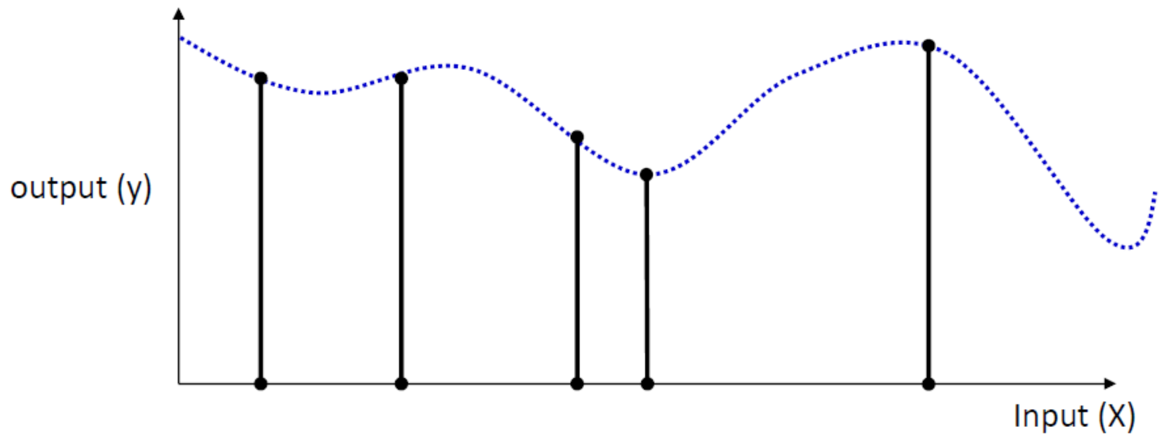
ENEE 4583/5583 Deep Learning

Dr. Alsamman

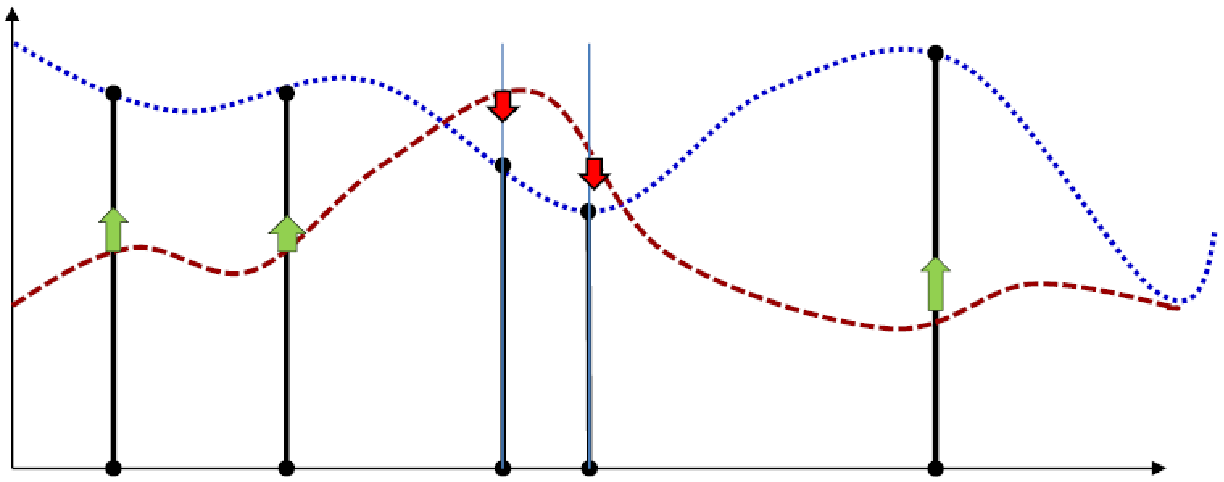
Slide Credits: Deep Learning Book, B. Raj, M. Nielsen, A. Radford, adventuresinmachinelearning.com

Batch Gradient Descent (BGD)

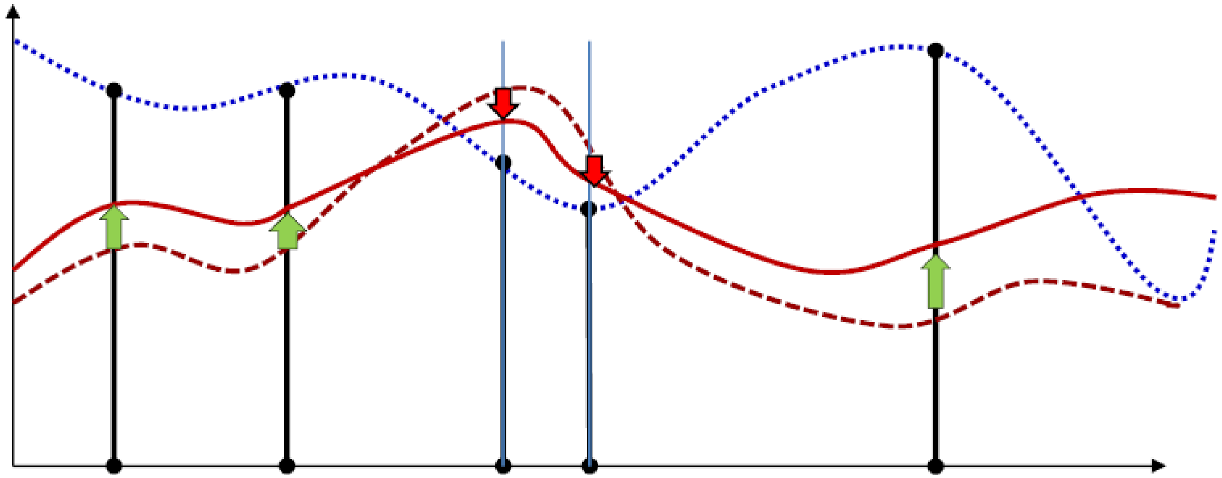
- Aka Gradient Descent
- Start with an initial function
- Adjust its value at all points to make the outputs closer to the required value
  - Gradient descent adjusts parameters
  - Goal is to adjust the function value at all points
  - Repeat this iteratively until we get arbitrarily close to the target function at the training points



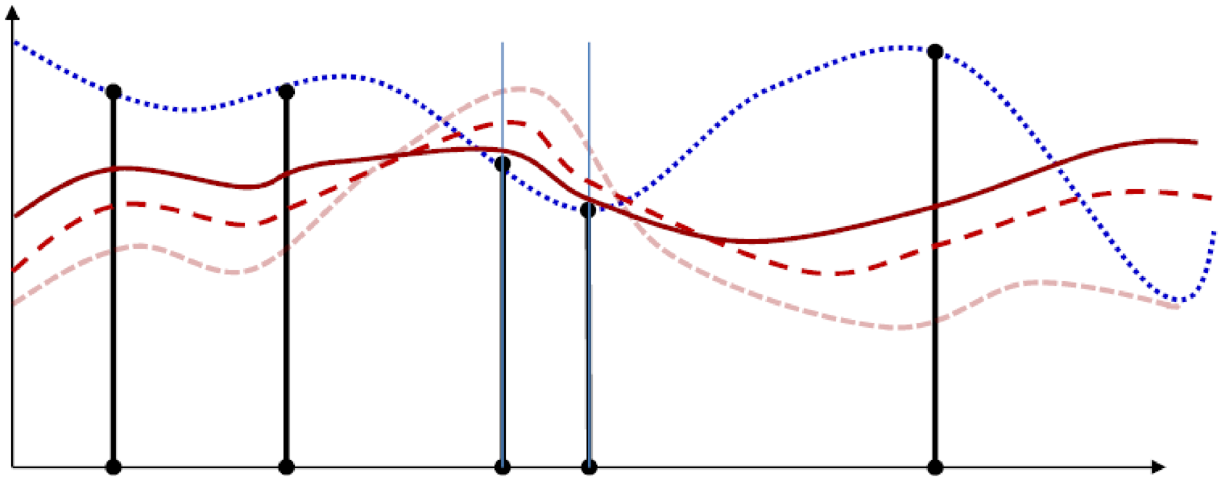
BGD



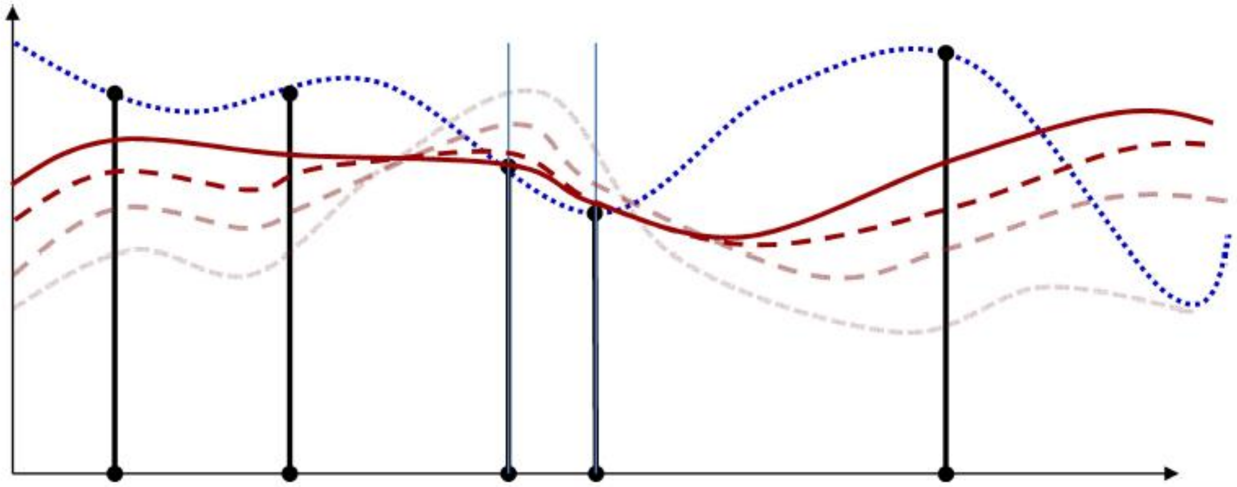
BGD



BGD



BGD

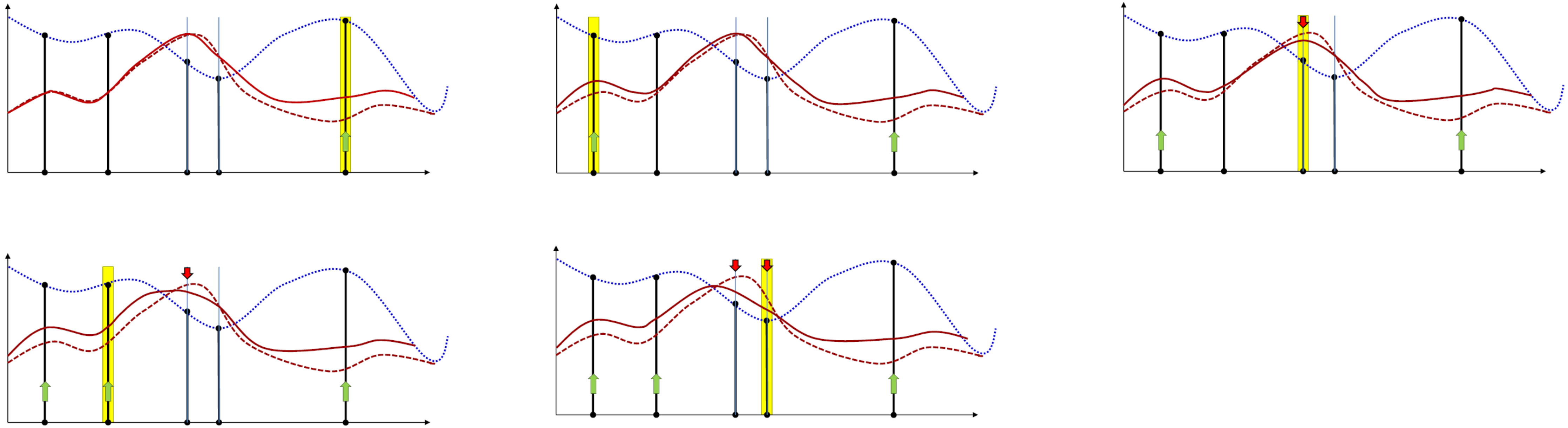


Batch GD

- Problem with conventional gradient descent: we try to simultaneously adjust the function at all training points
  - We must process all training points before making a single adjustment
  - “Batch” update

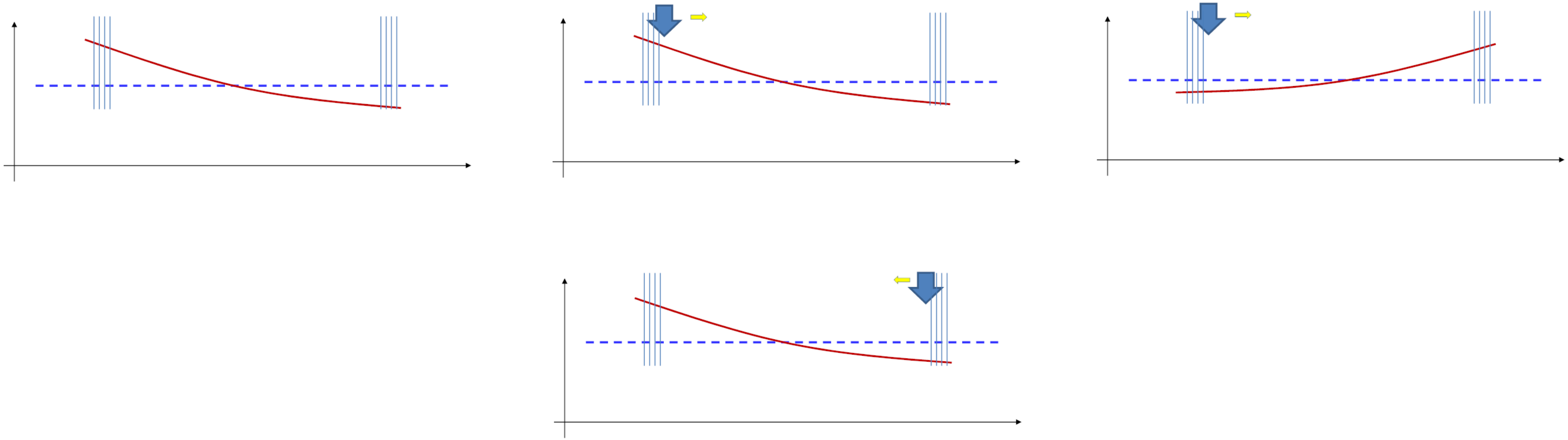
Point GD

- Adjust the function at one training point at a time
  - Keep adjustments small
  - Eventually, when we have processed all the training points, we will have adjusted the entire function
- Greater overall adjustment than we would if we made a single “Batch” update



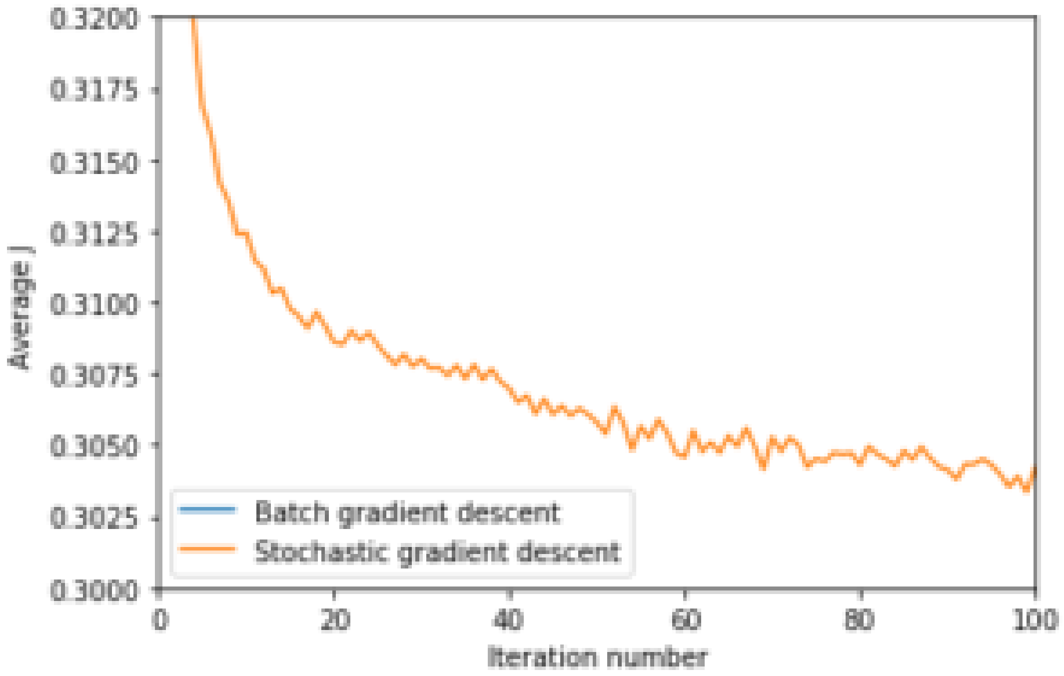
Stochastic GD (SGD)

- If we loop through the samples in the same order, we may get cyclic behavior
- We must go through them randomly to get more convergent behavior



SGD

- Problem: an optimal overall fit will look incorrect to individual instances
  - Correcting the function for individual instances will lead to never-ending, non-convergent updates
  - We must shrink the learning rate with iterations to prevent this

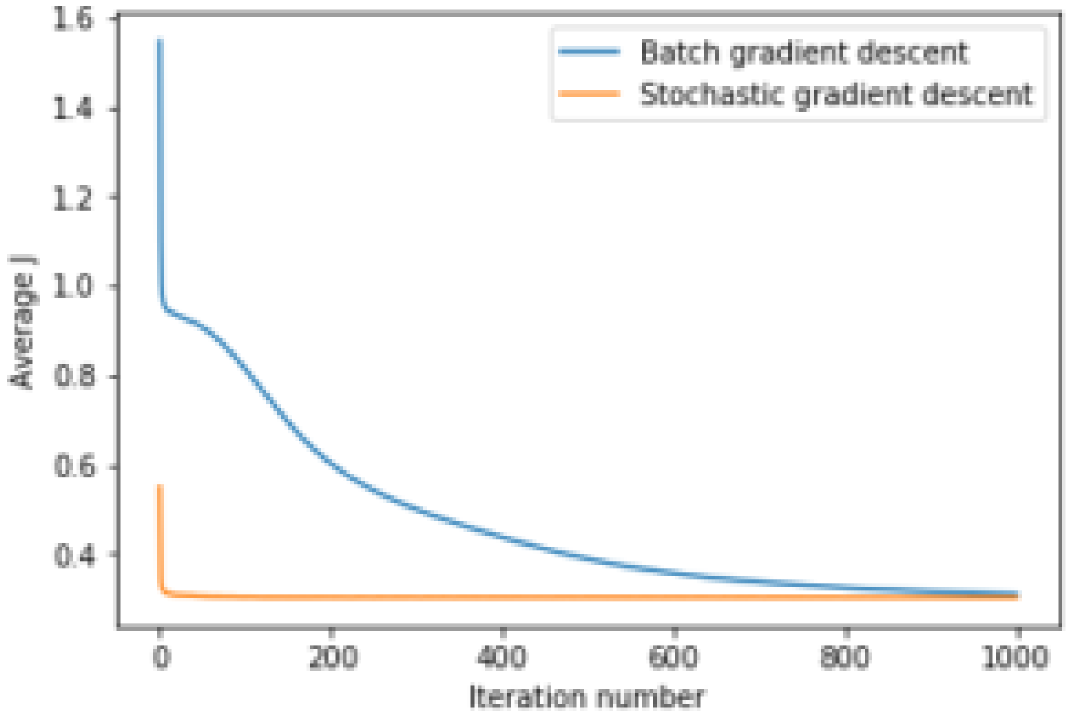


SGD

- Solution: Correcting the learning rate for individual instances
  - Learning rate reduces with each iteration
  - $\alpha \sim \frac{1}{m}$ (m=sample size)
  - will not modify the function
- Better solution: Make multiple passes over data
  - Each pass (aka iteration) is an “epoch”
- Convex => convergence global min
  - Non-Convex => converge local min
  - Convergence has been proven to be when $\alpha = 1/\sqrt{m}$ and stopping condition when $E = 1/\sqrt{m}$

BGD vs SGD

- SGD converges fast
- BGD outperforms SGD after many iterations

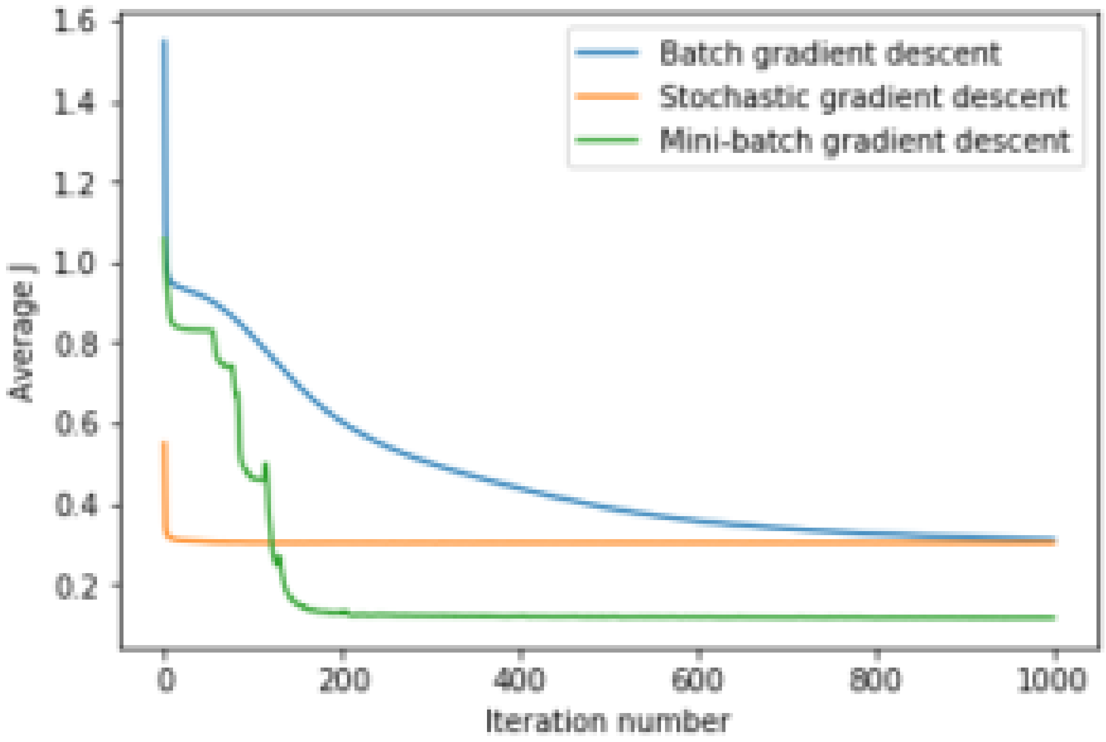


Mini-Batch GD

- SGD uses the gradient from only one sample at a time
  - Adv: Quicker updates than batch
  - Disadv: Noisy. One sample => high variance in error/cost
  - Disadv: BGD more optimal
- Mini batch:
  - adjust the function at a small, randomly chosen subset of points
  - Keep adjustments small
    - If the subsets cover the training set, we will have adjusted the entire function

BGD vs SGD vs Mini-BGD

- “Kicking” the cost function out of local mins



Mini-Batch Normalization

- Batches may not be similarly distributed
  - The smaller the batch size, the more likely the statistics will be influences by outliers
- Normalization: $\mu = 0, \sigma = 1$
  - Within each batch process: z

Mini-Batch Scheme

- Initialize mini batch size, b, learning rate, α
- Randomize permutations of $\{X, Y\}$
- While not (stop criterion)
  - Define batch runs: For $t = 1: b: m$
    - Intialize weights in each layer, $W^{(l)}$
    - Batch process: for $i = t: t + b - 1$
      - Forward propagate X_i, compute C_i
      - Calculate for each X_i: Calculate partial derivatives: $\nabla_{W^{(l)}} = \partial C_i / \partial W^{(l)}$
      - Calculate for each X_i: $\Delta W^{(l)} = \Delta W^{(l)} + \nabla_{W^{(l)}}$
    - Update weights after each mini-batch: $W^{(l)} = W^{(l)} - \alpha\, \Delta W^{(l)}$
    - Decrease α

Categorical Cross-Entropy Loss

- Supervised classification problem, Multi-label output: N-classes: $\boldsymbol{Y} \in \boldsymbol{\mathcal{R}}^{\boldsymbol{n}}$
- One-hot encoded
  - E.g. $n = 4$ classes: $\boldsymbol{Y}^{(\boldsymbol{i})} = \left[y_1^{(i)}, y_2^{(i)}, y_3^{(i)}, y_4^{(i)}\right]$
  - E.g.: $n = 4$, i=100 belongs to class 4: $\boldsymbol{Y}^{(\mathbf{100})} = [0,0,0,1]$
- Loss function:

$$\mathcal{L} = -\sum_{i}^{M}\sum_{n}^{N} y_n^{(i)} \cdot \log\left(\hat{y}_n^{(i)}\right)$$

  - M samples, N classes
  - $\hat{y}_n$ system n^th^ output, y_n GT
- BCE Binary version:

$$\mathcal{L} = -\sum_{i}^{M} y^{(i)} \cdot \log\left(\hat{y}^{(i)}\right) + \left(1 - y^{(i)}\right) \cdot \log\left(1 - \hat{y}^{(i)}\right)$$

Cross-Entropy Example

- Classify images 10 image as mouse, cat, dog
  - Mouse: Y = [1,0,0]
  - Cat: Y = [0,1,0]
  - Dog: Y = [0,0,1]
- Training:

	1	2	3	4	5	6	7	8	9	10
Mouse	0	0	0	0	0	0	1	1	1	0
Cat	0	0	0	1	1	1	0	0	0	0
Dog	1	1	1	0	0	0	0	0	0	1

- Outputs:

	1	2	3	4	5	6	7	8	9	10
Mouse	0.1	0.4	0.7	0.6	0.3	0.7	0.8	0.9	0.5	0.3
Cat	0.5	0.9	0.4	0.8	0.3	0.7	0.2	0.6	0.1	0.5
Dog	0.3	0.5	0.2	0.5	0.6	0.8	0.4	0.9	0.2	0.9

- Loss

loss	.52	.30	.70	.09	.52	.15	.09	.05	.30	.05

Gradient Descent Categorical CE

- Input/features: $\boldsymbol{X}$
- Binary output/prediction/hypothesis: $\hat{\boldsymbol{y}} = f(\boldsymbol{X})$
  - $f(\quad)$: neural network
- Ground truth/Label: $\boldsymbol{y}$
- GD Weight update: $$w_{ij}^{(l)} = w_{ij}^{(l)} - \frac{\alpha}{m}\frac{\partial \mathcal{L}}{\partial w_{ij}^{(l)}}$$
  - $w_{ij}^{(l)}$: connects neuron i in layer l to neuron j in layer $l+1$
  - α: learning rate
  - m: batch size
  - $\mathcal{L}$: BCE loss over entire batch
  - $\frac{\partial \mathcal{L}}{\partial w_{ij}^{(l)}}$: Partial derivative of loss, aka $\nabla\mathcal{L}$, aka ∇_w

Backprop BCE

$$\frac{\partial \mathcal{L}}{\partial w_{ij}^{(l)}} = \frac{\partial \mathcal{L}}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial Z_i} \cdots \frac{\partial Z_j^{(l+1)}}{\partial w_{ij}^{(l)}}$$

- Z_i: Logit used in softmax function
- $Z_p^{(q)}$: Input variable to neuron p in layer q
- $a_r^{(q-1)}$: activation function/output of neuron r in layer $q-1$
- … : $\frac{\partial Z_p^{(q)}}{\partial a_r^{(q-1)}} \frac{\partial a_r^{(q-1)}}{\partial Z_r^{(q-1)}}$ deriv of input to previous output times the deriv of prev out to its input
- $\frac{\partial \mathcal{L}}{\partial \hat{y}}$: Derivative of loss function with respect to the output
- $\frac{\partial \hat{y}}{\partial Z_p^{(q)}}$: Derivative of the activation function

$$\frac{\partial \mathcal{L}}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial Z_i}$$

- Derivative of Loss wrt Logit

$$\frac{\partial \mathcal{L}}{\partial \hat{y}_i} = \frac{\partial}{\partial \hat{y}_i} y_i \log \hat{y}_i = \frac{y_i}{\hat{y}_i}$$

$$\frac{\partial \hat{y}_i}{\partial Z_i} = \frac{\partial}{\partial Z_i} \frac{\exp(Z_i)}{\sum_j^N \exp(Z_j)} = \begin{cases} \hat{y}_i(1 - \hat{y}_i) & \text{for } i \neq j \\ -\hat{y}_i \hat{y}_j & \text{for } i \neq j \end{cases}$$

$$\frac{\partial \mathcal{L}}{\partial Z_i} = \hat{y}_i - y_i = \hat{y}_i - 1$$

http://www.adeveloperdiary.com/data-science/deep-learning/neural-network-with-softmax-in-python/

CE Forward Prop

- Given M samples of {X,Y} input, output pairs.
- Initialize W, b, alpha, batch -- W,b for each layer.
- for m = 0:batch:M -- process in batches
- in[0] = Xm -- in from layer 0 is the input to the system
- for l = 1:L -- layers from 1 to L-1
- Z[l] = W[l]*in[l-1] + b[l] -- inputs to each neuron in layer l
- in[l] = a(Z[l]) -- inputs to next layer
- out[L] = S(Z[L]) -- output of system

CE Back prop

- Loss = -sum(Y*out[L])
- dout[L] = -Y/out(L)
- da[L] = gradS(out[L]) -- gradient of softmax/sigmoid
- for l= L-1:0 -- propagate backwards in layers
- dW[l] = out[l+1]*da[l]*input[l]
- W[l] -= a/m*dW
- out[l] = W[l-1]

Special Cases: WCE

- Unbalanced data: more of one class than another
- Unbalance can be intentional
  - Example: face detection database has 100K face images, 900K background images
  - Typically, more background than faces in images
- Unbalance can be due to lack of data
  - Labeling/data can be expensive
- Class weights: $C_n = \frac{M}{m_i}$
  - M total samples, m_i samples of class i
- WCE Loss:

$$\mathcal{L} = -\sum_{i}^{M} \frac{M}{m_i} \sum_{n}^{N} y_n^{(i)} \cdot \log\left(\hat{y}_n^{(i)}\right)$$