MACHINE INTELLIGENCE

UNIT-5

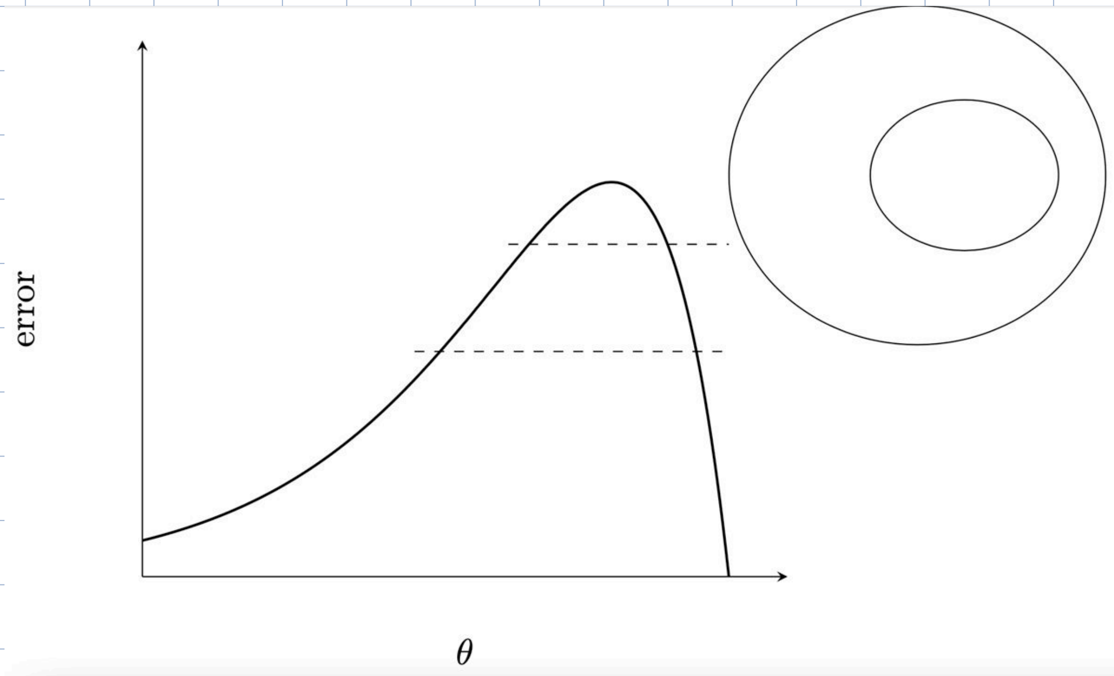
Optimizers

feedback/corrections: vibha@pesu.pes.edu

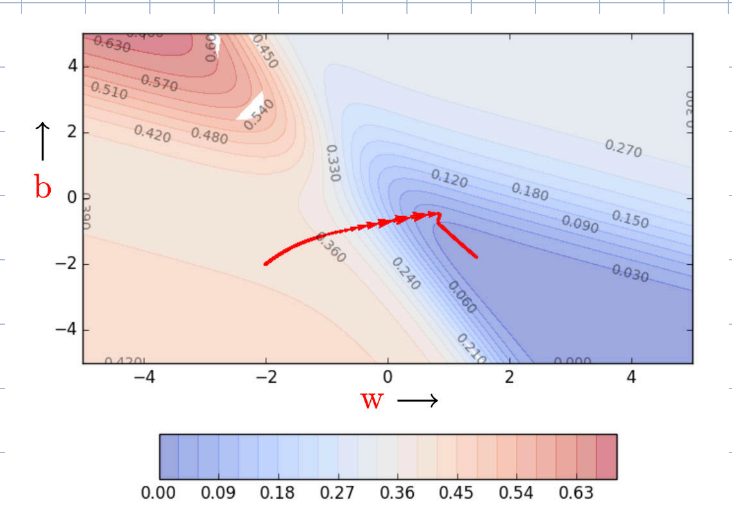
VIBHA MASTI

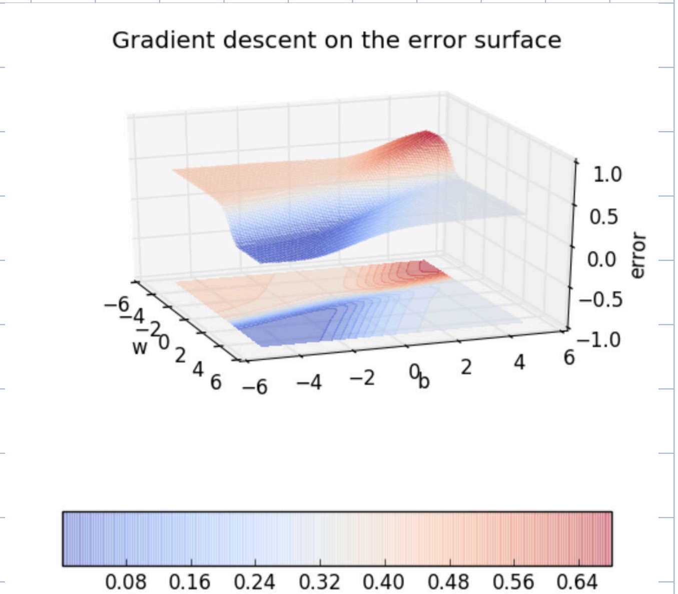
1. Contour Plots

- Visualise 3D in 2D
- Small distance between contours: steep slope



1.2 GD on Coutour Maps (Visualised)



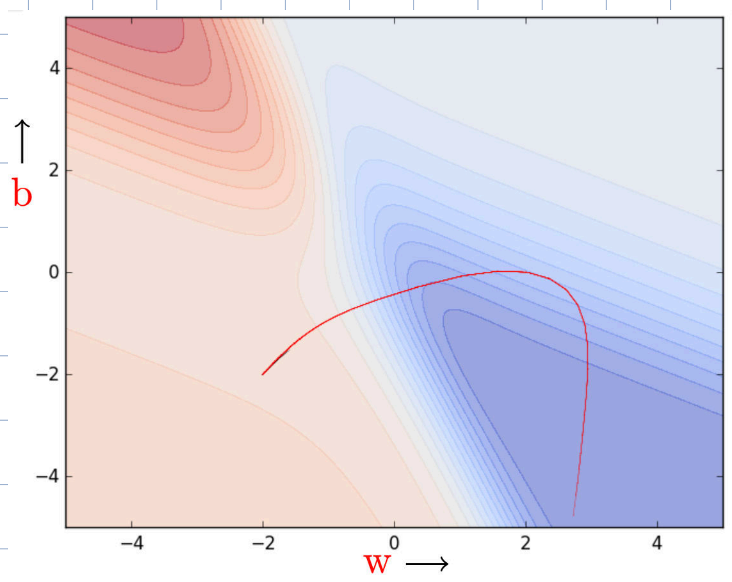


2. Momentum Based GD

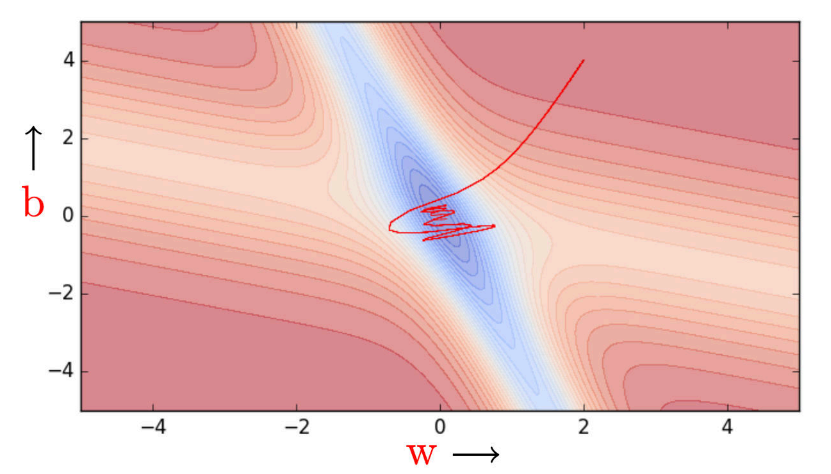
- If gradient is in same direction for multiple passes, take bigger steps
- Like a ball gaining momentum down a slope
- Adds fraction (γ) of previous update vector to present update vector

$$update_t = \gamma \cdot update_{t-1} + \eta \nabla \mathcal{L}(w)$$

$$w_{t+1} = w_t - update_t$$



- Speeds up convergence



oscillates

- Oscillates but is still faster than vanilla GD

- Can use momentum with SGD

3. Stochastic GD

- Vanilla GD : 1 pass over all data points to update w and b

$$\nabla \mathcal{L}(w) = \sum_{i=1}^{n} -2x_i(y_i - (wx_i + b)) = dw$$

$$\nabla \mathcal{L}(b) = \sum_{i=1}^{n} -2(y_i - (wx_i + b)) = db$$

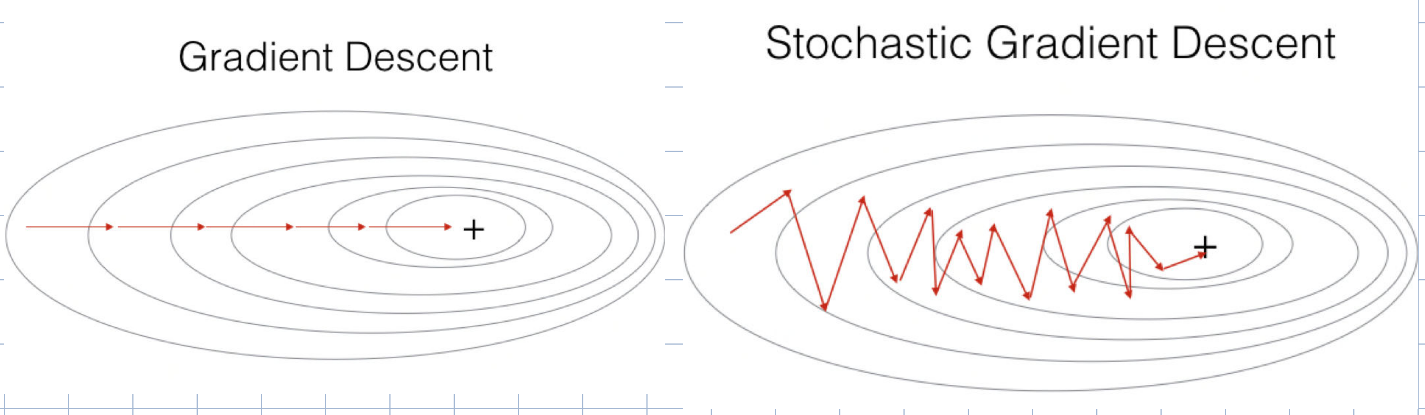
- Each epoch (pass over data) updates w and b only once

- Inefficient with large datasets
- SGD: pick one sample from entire dataset and calculate dw and db from it

$$dw = -2x(y-(wx+b))$$

$$db = -2(y-(wx+b))$$

- Each epoch now updates w and b n number of times where n = size of dataset



source: Andrew Ng's course

- Problem: too many oscillations (greedy solution wrt a single point)

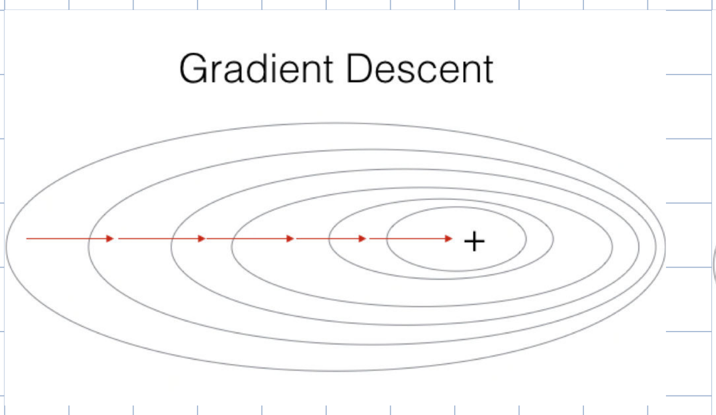
4. Mini Batch GD

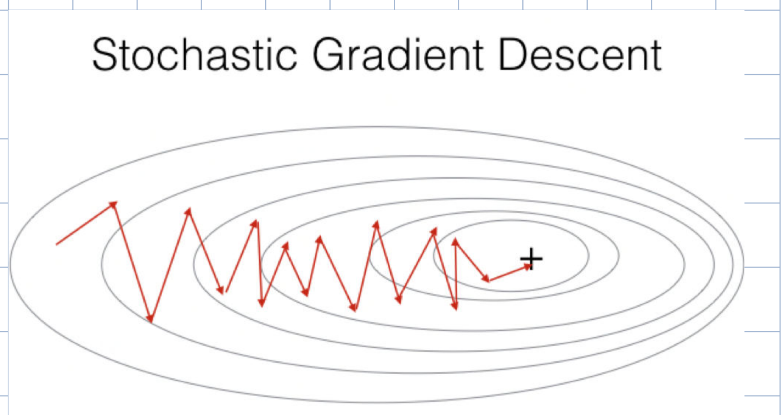
- Updates dw and db after seeing k number of data points

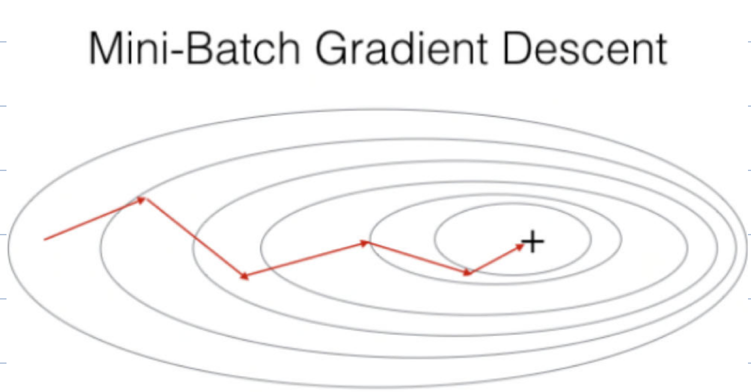
$$dw = \sum_{i=1}^{k} -2x_i(y_i - (wx_i + b))$$

$$db = \sum_{i=1}^{k} -2(y_i - (wx_i + b))$$

- k points out of n contribute to dw and db (not as greedy)







- 1 epoch = one pass over the entire data
- 1 step = one update of the parameters
- N = number of data points
- B = Mini batch size

Algorithm	# of steps in 1 epoch
Vanilla (Batch) Gradient Descent	1
Stochastic Gradient Descent	N
Mini-Batch Gradient Descent	$\frac{N}{B}$

- SGD code in keras

```
# creating a model
model = keras.Sequential([
        keras.layers.Flatten(input_shape=x_train[0].shape),
        keras.layers.Dense(250, activation='relu'),
        keras.layers.Dense(10, activation='softmax')])
# optimizer - sgd
opt = SGD(learning_rate=0.001)
# compile the model
model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])
# fit a model
history = model.fit(x_train,y_train, validation_data=(x_test, y_test), epochs=200, verbose=0)
```

Full code: https://drive.google.com/file/d/1zk5-sAsJIM-5rc2OnnEyFOjPVzBLWIJu/view

5. GD with Adaptive LR (Adagrad)

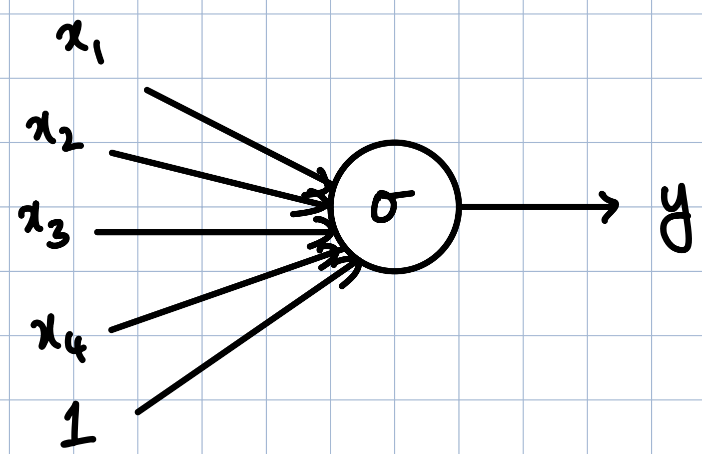
- Consider 4 features in x and their corresponding weights (x & w are vectors)

$$x = \{x_1, x_2, x_3, x_4\}$$

$$w = \{w_1, w_2, w_3, w_4\}$$

- Consider a simple network with a single sigmoid node

$$y = f(x) = \frac{1}{1+e^{-(w \cdot x + b)}}$$



$$\nabla w_1 = dw_1 = (f(x) - y) * f(x) * (1 - f(x)) * x_1$$

$$\nabla w_2 = dw_2 = (f(x) - y) * f(x) * (1 - f(x)) * x_2$$

and so on

- If x_2 is a sparse feature (usually 0), then dw_2 is usually 0 and w_2 will not update often

- Adagrad: decay LR for parameters (features) in proportion to their update history

- More updates → more decay

- Update rule for w_i

$$v_{i,t} = v_{i,t-1} + (\nabla w_{i,t})^2$$

squared sum of gradients

$$w_{i,t+1} = w_{i,t} - \frac{\eta}{\sqrt{v_{i,t} + \varepsilon}} * \nabla w_t$$

small value for division by 0 error

- Issue: LR decays very aggressively and gets stuck close to convergence as updates stop

6. RMS Prop

- Root mean square propagation

- Resolves issue with Adagrad by decaying the denominator term $v_{i,t}$

$$v_{i,t} = \beta * v_{i,t-1} + (1-\beta)(\nabla w_{i,t})^2$$

$$w_{i,t+1} = w_{i,t} - \frac{\eta}{\sqrt{v_{i,t} + \epsilon}} * \nabla w_t$$

- Solves convergence problem of Adagrad

7. Adam

- Adaptive moment estimation
- Combination of Adagrad and RMSprop
- Exponential moving average of grads to scale LR (like RMSprop)
- Also use cumulative history of grads
- Update rule for feature x_i

$$m_{i,t} = \beta_1 * m_{i,t-1} + (1-\beta_1) * \nabla w_{i,t}$$

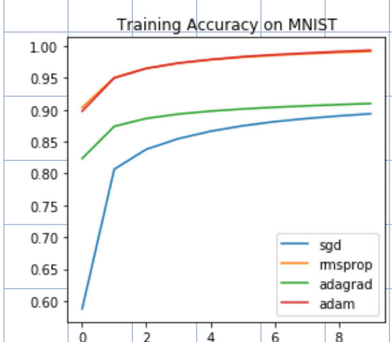
$$v_{i,t} = \beta_2 * v_{i,t-1} + (1-\beta_2) * (\nabla w_{i,t})^2$$

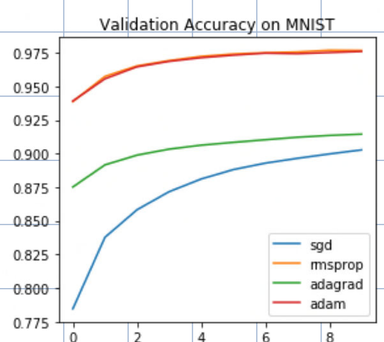
$$\hat{m}_{i,t} = \frac{m_{i,t}}{1-(\beta_1)^t} \qquad \hat{v}_{i,t} = \frac{v_{i,t}}{1-(\beta_2)^t}$$

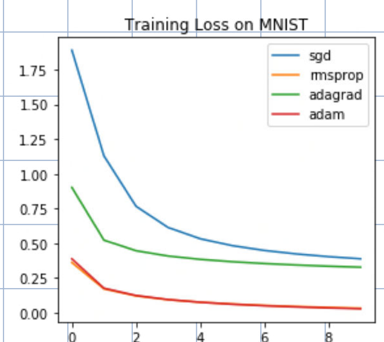
$$w_{i,t+1} = w_{i,t} - \frac{\eta}{\sqrt{\hat{v}_{i,t} + \varepsilon}} * \hat{m}_{i,t}$$

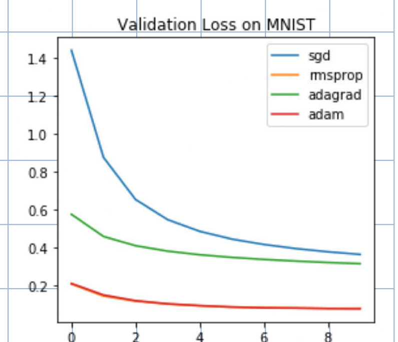
COMPARISON OF OPTIMIZERS

- Usually Adam works best
- Robust to initial LR (η) values but for sequence generation, $\eta = 0.001$ and $\eta = 0.0001$ best
- SGD with momentum also works well ($\eta = 0.001$ or 0.0001 for sequence generation problems)
- Accuracies & loss on MNIST with SGD, RMSprop, adagrad, adam

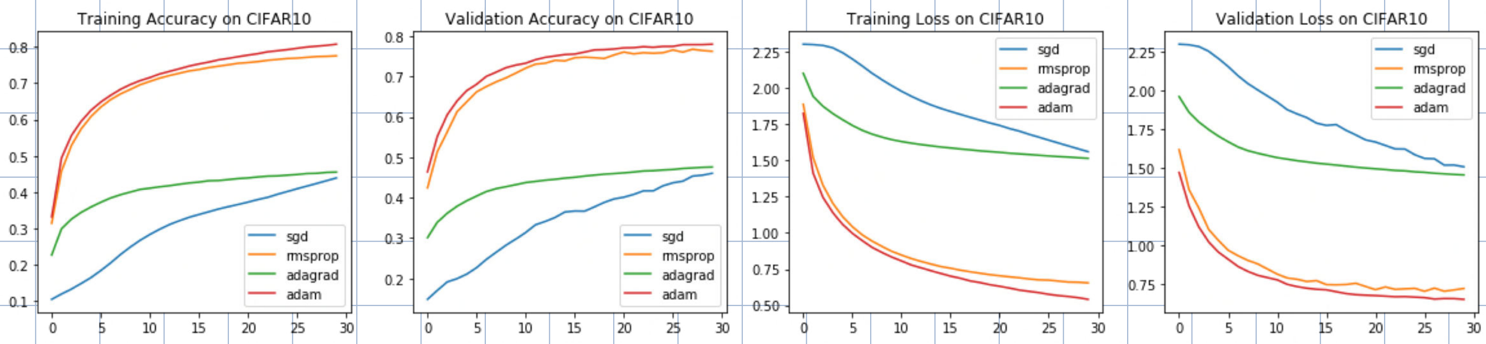




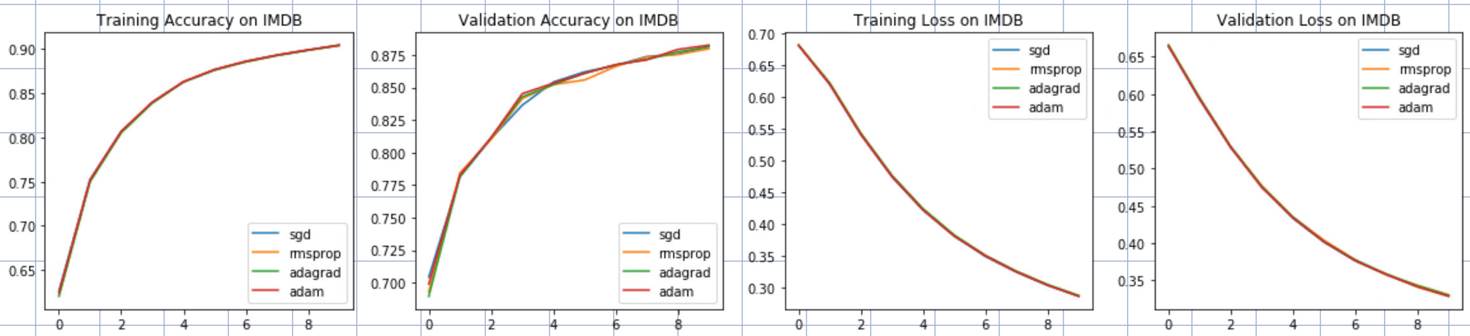




- Large dataset (CIFAR10)



- Small dataset (IMDB sentiment)



Source:

https://heartbeat.comet.ml/an-empirical-comparison-of-optimizers-for-machine-learning-models-b86f29957050

Code for multi layered NN in keras:

https://drive.google.com/file/d/1wEQ1R025PLL85QgMnlR76BJx8De9FKCD/view