Problem 1

solve the differential equation

0.0.1 Prob 1 Set up

$$J_i(\beta) = \begin{cases} y^{(i)} \log(f(x^{(i)} \cdot \beta)) & \text{if } y^{(i)} = 1 \\ (1 - y^{(i)}) \log(1 - f(x^{(i)} \cdot \beta)) & \text{if } y^{(i)} = 0 \end{cases}$$

$$\sigma(r) = (1 + exp(-r))^{-1}$$

$$J_i(\beta) = y^{(i)} \log(f(x^{(i)} \cdot \beta)) + (1 - y^{(i)}) log(1 - f(x^{(i)} \cdot \beta))$$

Thus, its gradient is:

$$\nabla J_{i,j}(\beta) = \frac{y^{(i)} f'(x^{(i)} \cdot \beta)}{f(x^{(i)} \cdot \beta)} - \frac{(1 - y^{(i)}) \cdot f'(x^{(i)} \cdot \beta)}{1 - f(x^{(i)} \cdot \beta)}$$

$$f'(x^{(i)} \cdot \beta) = f(x^{(i)} \cdot \beta) \cdot (1 - f(x^{(i)} \cdot \beta)) \cdot (x^{(i)} \cdot \beta)'$$

$$f'(x^{(i)} \cdot \beta) = f(x^{(i)} \cdot \beta) \cdot (1 - f(x^{(i)} \cdot \beta)) \cdot x^{(i,j)}$$

$$\nabla J_{i,j}(\beta) = \frac{y^{(i)} f'(x^{(i)} \cdot \beta)}{f(x^{(i)} \cdot \beta)} - \frac{(1 - y^{(i)}) \cdot f'(x^{(i)} \cdot \beta)}{1 - f(x^{(i)} \cdot \beta)}$$

$$\nabla J_{i,j}(\beta) = \frac{y^{(i)} \cdot x^{(i,j)} \cdot f(x^{(i)} \cdot \beta) \cdot (1 - f(x^{(i)} \cdot \beta))}{f(x^{(i)} \cdot \beta)} - \frac{(1 - y^{(i)}) \cdot x^{(i,j)} \cdot f(x^{(i)} \cdot \beta) \cdot (1 - f(x^{(i)} \cdot \beta)}{1 - f(x^{(i)} \cdot \beta)}$$

$$\nabla J_{i,j}(\beta) = y^{(i)} \cdot (1 - f(x^{(i)} \cdot \beta)) \cdot x^{(i,j)} - (1 - y^{(i)}) \cdot f(x^{(i)} \cdot \beta) \cdot x^{(i,j)}$$

1

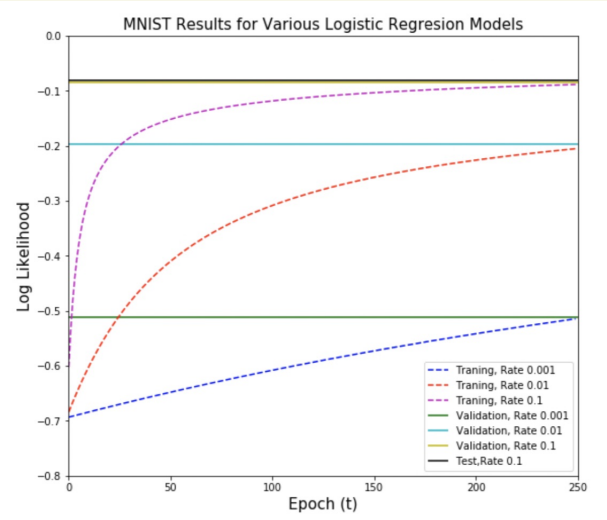
$$\nabla J_{i,j}(\beta) = y^{(i)} \cdot (1) \cdot x^{(i,j)} - (1) \cdot f(x^{(i)} \cdot \beta) \cdot x^{(i,j)}$$

$$\nabla J_{i,j}(\beta) = (y^{(i)} - f(x^{(i)} \cdot \beta)) \cdot x^{(i,j)}$$

$$\nabla l_j(\beta) = \frac{1}{N} \sum_{i=1}^{N} (y^{(i)} - f(x^{(i)} \cdot \beta)) \cdot x^{(i,j)}$$

log likelihood grad[j] = $np.dot((Y - f(X, beta)).T, X[:, j]).T/N$
log likelihood grad = $np.dot((Y - f(X, beta)).T, X).T/N$

Part 1.1



Part 1.2

Out[10]:

	Training Error	Validation Error	Test Error
Rate 0.001	0.055984	0.044782	0.043342
Rate 0.01	0.047322	0.038868	0.034987
Rate 0.1	0.027253	0.024081	0.022454

	Training likelihood	Validation likelihood	Test likelihood
Rate 0.001	-0.514456	-0.510959	-0.504993
Rate 0.01	-0.205288	-0.197685	-0.188261
Rate 0.1	-0.088661	-0.085487	-0.080554

Part 1.3

Briefly describe the results that you see in your plots and tables, e.g., convergence rates, error and log-likelihood values vary across training and validation sets, what would have happened if you had selected the best-performing model (e.g., using log-likelihood) on the validation data, etc.

from plot and table, we can see that, for these three rates, the larger, the better performance it is. when rate gets larger, it converges faster and get loglikelihood closer to zero at the end (both train, validation and test)
of the epoch time 250; rates gets larges error rate for train validation and test gets smaller.

if we choose best model Rate = 0.1

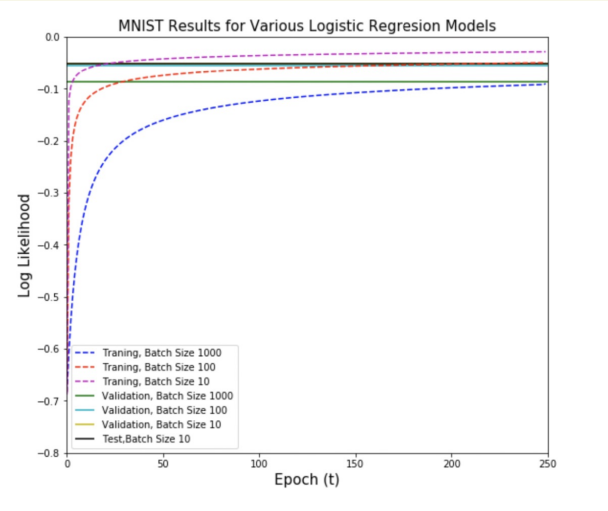
the error rate for traing validation and test

are 0.027253 0.024081 and 0.022454

loglikelihood fo them are

-0.088661 -0.085487 -0.080554

Part 2-1



Part 2-2

	Training Error	Validation Error	Test Error
Batch Size 1000	0.027358	0.023659	0.022454
Batch Size 100	0.015316	0.016054	0.015666
Batch Size 10	0.008662	0.015209	0.017232

]:

	Training Log Likelihood	Validation Log Likelihood	\
Batch Size 1000	-0.090410	-0.086988	
Batch Size 100	-0.049727	-0.056108	

Batch Size 10	-0.028701	-0.051667

	Test Log Likelihood
Batch Size 1000	-0.081827
Batch Size 100	-0.054370
Batch Size 10	-0.051401

Part 2.3

Briefly describe the results that you see in your plots and tables, e.g., convergence rates, error and log-likelihood values vary across training and validation sets, what would have happened if you had selected the best-performing model (e.g., using log-likelihood) on the validation data, etc.

$10^3, 10^2, 10$

from plot and table, we can see that, for these three batch size the smaller the better performance it is. when size gets smaller, it converges faster and get loglikelihood closer to zero at the end (both train, validation and test)
of the epoch time 250; batch sizes get smaller, error rate for train validation and test gets smaller at the end time

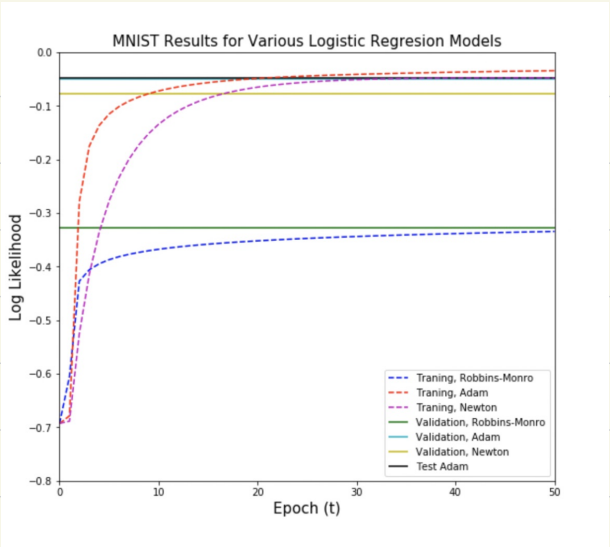
if we choose best model batch size = 10

the error rate for train validation and test
are 0.86% 1.52% 1.72%

loglikelihood fo them are
-0.049 -0.056 -0.0514

Problem 2

Part 1



	Training Error	Validation Error	Test Error
Robbins-Monro	0.052287	0.041825	0.038642
Adam	0.010246	0.013942	0.014099
Newton	0.019858	0.021969	0.018799

	Training Loglikelihood	Validation Loglikelihood	\
Robbins-Monro	-0.334296	-0.327885	
Adam	-0.034397	-0.049313	
Newton	-0.048226	-0.076712	

	Test Loglikelihood
Robbins-Monro	-0.317392
Adam	-0.047404
Newton	-0.083347

Robbins-Monro goes very good at the beggining, but does not continously because of the decreasing in learning rate. Newton is similar to Adam but converges slower. Adam perform best and adhieve the least error rate and largest likelihood in all training, validation and test. Adam with best test err 1.41% and best loglikelihood -0.047404

Part 2 (10 points): Answer the following three questions.

1. What's the intuition behind Newton's Method? In other words, what does the Hessian tell us about a given location on the function?
2. What's the intuition behind the AdaM update? Think about what happens to the step size when the variance of the gradients ($\hat{v}_t$) is large and when it is small. Is the underlying mechanism similar to Newton's method's use of the Hessian?
3. State the runtime and memory complexity (in big $\mathcal{O}$ notation) in terms of the feature dimensionality d and the batch size M for each update method.

Part 2

1.

Let $f(\theta,x)=\frac{1}{M}\sum_{m=1}^{M}(y-f(\theta,x))x_m$ be the loglikelihood of batch size M

then $f'(\theta,x)=\frac{-1}{M}\sum_{j=1}^{M}\sum_{m=1}^{M}f(\theta,x)(1-f(\theta,x))x_m x_j$

in vector form $f(\theta,x)=\frac{1}{M}(Y-f)X$

$f'(\theta,x)=\frac{1}{M}X^t[f(1-f)]X$

by newton's method

$$X^{new}=X^{old}-\frac{f(x)}{f'(x)}=X^{old}+\frac{\frac{1}{M}(Y-f)X}{\frac{1}{M}X^t f(1-f)X}$$

$$=X^{old}+\alpha\cdot\frac{1}{M}(Y-f)X$$

where learning α is $M\cdot(X^t(f(1-f))X)^{-1}$

thus the intution of Newton's method is to change each of θ_j in $\underline{\theta}$ with different rates which correspond to their derivative for the log likelihood function

Hessian matrix $H_{ij}=\frac{\partial^2 f}{\partial\theta_i\partial\theta_j}=f'(\theta,x)$ tells us about how the gridient $(\frac{\partial f}{\partial\theta_1},\dots\frac{\partial f}{\partial\theta_j})$ changes, in other word the change rate of gridient θ_i respect to the change of other gradient θ_j

by set learning rate $=H_{ij}$, it can make each θ_j goes in similar scaled rate for converging instead of leave some component goes too fast and thus ocilating or some goes too slow respect to the local curvature of each θ_j.

Part 2 (10 points): Answer the following three questions.

1. What's the intuition behind Newton's Method? In other words, what does the Hessian tell us about a given location on the function?
2. What's the intuition behind the AdaM update? Think about what happens to the step size when the variance of the gradients ($\hat{v}_t$) is large and when it is small. Is the underlying mechanism similar to Newton's method's use of the Hessian?
3. State the runtime and memory complexity (in big $\mathcal{O}$ notation) in terms of the feature dimensionality d and the batch size M for each update method.

(2.)

let $E[m_t] = E[f_t]$

$E[V_t] = E[f_t^2]$

set $m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$

$V_t = \beta_2 V_{t-1} + (1-\beta_2) g_t^2$

then $m_0 = 0$

$m_1 = \beta_1 m_0 + (1-\beta_1) g_1 = (1-\beta_1) g_1$

$m_2 = \beta_1 m_1 + (1-\beta_1) g_2 = \beta_1 (1-\beta_1) g_1 + (1-\beta_1) g_2$

$\vdots$

$m_t = (1-\beta_1) \sum_{i=0}^{t} \beta_1^{t-i} g_i$

$E[m_t] = E[(1-\beta_1) \sum_{i=1}^{t} \beta_1^{t-i} f_i]$

↓ approxiate f_i with f_t

$E[m_t] = (1-\beta_1) \sum_{i=1}^{t} \beta_1^{t-i} E[f_t] + \zeta$

$= (1-\beta_1^t) E[f_t] + \zeta$

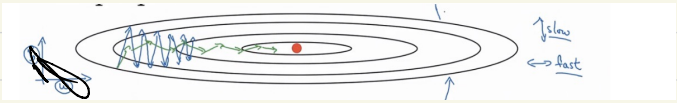
$\approx E[f_t]$

bias correction for the first momentum estimator

similar for $E[V_t] = E[f_t^2]$

get $\theta_{t+1} = \theta_t + \alpha_0 \frac{m_t}{\sqrt{V_t} + \varepsilon}$

AdaM means that if the variance of $\frac{\partial f}{\partial \theta_i}$ is large, than step size $\frac{\alpha_0}{\sqrt{V_{ti}}}$ is small if variance of $\frac{\partial f}{\partial \theta_i}$ is small, than stepsize $\frac{\alpha_0}{\sqrt{V_{ti}}}$ is large



blue m_t, green $\frac{m_t}{\sqrt{V_t}}$ for two dimension cases

Yes, it's underlying mechanism is similar to New's method of Hij slow down the fast gradient change and accelerate the low speed change

Part 2 (10 points): Answer the following three questions.

1. What's the intuition behind Newton's Method? In other words, what does the Hessian tell us about a given location on the function?
2. What's the intuition behind the AdaM update? Think about what happens to the step size when the variance of the gradients ($\hat{v}_t$) is large and when it is small. Is the underlying mechanism similar to Newton's method's use of the Hessian?
3. State the runtime and memory complexity (in big $\mathcal{O}$ notation) in terms of the feature dimensionality d and the batch size M for each update method.

(3)

$\exp(x) = \sum_{n=1}^{\infty} \frac{x^n n}{n!}$ $O(n)$ set $O(1)$ since bounded

calculate logistic : $O(Md)$

log like_grad_stoch :

choose M from N $O(N)$

calculate gradient $O(Md) + O(d^2)$

Robbins-Monro

run time : $O(\text{log like stoch}) = O(Md) + O(d^2) \approx O(Md)$

memory : $O(d)$

Adam

run time : $O(Md) + O(d^2) + O(d) + O(1) = O(Md) + O(d^2)$
$\approx O(Md)$

memory : $O(d)$

Newton :

run time : $O(Md) + O(d^2) + O(Md) + O(Md^2) + O(d^3) + O(d^2)$
$= O(d^3) + O(Md^2)$

memory : $O(A^{-1})$

$= O(d^2)$