CS57800: Statistical Machine Learning

Homework 3

Name: Mohammad Haseeb
Purdue ID: mhaseeb@purdue.edu

Due: November 13, 2018 on Tuesday

1 Logistic Regression Open-ended Questions

1. To calculate the gradient of the loss function, we will get the gradient of the sums inside the summation and then apply the summation to the gradient to get the final gradient.

   Let $h = y\log(g(z)) + (1-y)\log(1-g(z))$ be the function which consists of the terms inside the summation of the original loss function $Err(w)$. We want to compute the gradient with respect to the weight vector w. Then, we get the following expression for the gradient by using chain rule:

$$\frac{dh}{dw} = \frac{dH}{dg}\frac{dg}{dz}\frac{dz}{dw}$$

Simplifying this further, we get the following:

$$\begin{aligned}
\frac{dh}{dg} &= \frac{y}{g(z)} + \frac{1-y}{1-g(z)} \cdot (-1) \\
&= \frac{y}{g(z)} - \frac{1-y}{1-g(z)} \\
&= \frac{y-g(z)}{g(z)(1-g(z))} \\
\frac{dg}{dz} &= \frac{d}{dz}(1+e^{-z})^{-1} \\
&= \frac{e^{-z}}{(1+e^{-z})^2} \\
&= \frac{1+e^{-z}-1}{(1+e^{-z})^2} \\
&= \frac{1+e^{-z}}{(1+e^{-z})^2} - (\frac{1}{1+e^{-z}})^2 \\
&= \frac{1}{(1+e^{-z})} - (\frac{1}{1+e^{-z}})^2 \\
&= g(z)(1-g(z)) \\
\frac{dz}{dw} &= \frac{d}{dw} w^t x \\
&= x \\
\implies \frac{dh}{dw} &= \frac{y-g(z)}{g(z)(1-g(z))} \cdot g(z)(1-g(z)) \cdot x \\
&= (y-g(z))x
\end{aligned}$$

Now, putting in summation in the expression, we get the following gradient for the loss function:

$$\nabla Err(w) = -\sum_i (y^i - g(w, x^i) x_j^i)$$

2. We will use both the composition rules and the second derivative test to show that the logistic loss function is convex. To do so, we will rewrite the loss function as follows:

$$Err(w) = \sum_i y^i(-\log(g(z))) + (1+y^i)(-\log(1-g(z))) \tag{1}$$

We know that the sum/linear-combination of two or more convex functions is also convex. Therefore, we can show that the function $-\log(g(z))$ (and similarly $(-\log(1-g(z)))$) is convex, and therefore, the loss function as a whole is convex. To show that $-\log(g(z))$ is

convex, we use the second derivative test.

$$
\begin{aligned}
y &= -\log(g(z)) \\
y &= -\log(\frac{1}{1+e^{-z}}) \\
y &= \log(1+e^{-z}) \\
y' &= -1 + (1+e^{-z})^{-1} \\
y'' &= \frac{e^{-z}}{(1+e^{-z})^2} \geq 0
\end{aligned}
$$

The second derivative of $-\log(g(z))$ is always greater than or equal to zero, therefore, $-\log(g(z))$ is convex. We can similarly show that $-\log(1-g(z))$ is also convex. Therefore, the logistic loss function is convex.

3. Regularization is a form of inductive bias that allows us to prefer simpler models (Occam's Razor). We want our learner to be less prone to variance/noise in the data and the regulizer allows us to do this. It limits the complexity of the hypothesis by penalizing more complex hypotheses. So, a more complex hypothesis will only be learnt if it significantly performs better. In this way, we prevent overfitting since we prefer simpler models.

4. The gradient will be the same plus the gradient of the regularization term i.e. λw. So, we get the following:

$$\nabla Err(w) = -\sum_i (y^i - g(w, x^i)x_j^i) + \lambda w$$

5. We have several options when it comes to deciding the convergence criterion for gradient descent (both batch and stochastic). We can have a value of ϵ and define δ to be the difference in the training loss between every subsequent epoch. If $\delta < \epsilon$, we can say that our model has converged.

   The other option is to graph the accuracies for a large number of epochs. We can then say that our model has converged if the accuracy does not increase after a certain number of epochs. If we see the graph of the accuracy v.s. epochs in the later sections of the report, we see that the accuracy eventually levels off. So, we can set the threshold number of epochs to be the smallest value that gives the maximum accuracy. This is the convergence criterion that I have used for both batch and stochastic gradient descent, with different meanings of epoch for each algorithm.

   If we went with the former convergence criterion i.e. based on $\delta < \epsilon$, then this might not work for stochastic gradient descent version (SGD). This is because, unlike batch gradient descent (BGD), SGD updates weight on every example which might cause the loss to fluctuate instead of decrease steadily as is the case in BGD.

6. For both BGD and SGD, the bias term allows us to shift the graph/decision boundary. Without the bias term, the y-intercept (e.g. in a 2D case), would remain the same. When we introduce the bias, the y-intercept also changes depending on the data. The same is true in a high dimentional case. This allows us to learn generalize better, therefore, I use it.

2 Batch Gradient Descent with Logistic Function

2.1 Algorithm

Algorithm 1 Batch Gradient Descent(TrainData,NumEpochs,LearningRate,Digit)

$weights_d = random()$ for all d=0...numFeatures
*/*Number of epochs is the stopping criterion*/*
for *e = 0 ... numEpochs* **do**
  $\Delta w = 0$ for all d=0...numFeatures
  for *(img,imgLabel) in TrainData* **do**
    $y = 0$
    if *imgLabel == Digit* **then**
      $y = 1$
    end
    $activation = weights \cdot img$; `// compute activation for this image example by taking dot product`
    for *j = 0...numFeatures* **do**
      $\Delta w[j] = \Delta w[j] + learningRate \times (y - logistic(act)) \times img[j]$; `// logistic() is the logistic/sigmoid function`
    end
  end
  $weights = weights + \Delta w$
end
return *weights*

function LOGISTIC(z)
  return $1/(1+e^{-z})$
end function

2.2 Results

Each of the following graphs has different number of epochs. The value of the epochs corresponds to the convergence criterion for that particular combination of regularization and feature type. I found the best value of epochs by running the experiments for a large initial value and recording the accuracies. I then found the minimum epochs corresponding to the maximum accuracy.

From the graphs, we can see that feature type 1 gives a better accuracy than feature type 2. We can also note that when we use regularization, the graphs tend to be smoother with less fluctuations but with a slightly accuracy. This is as expected because the purpose of regularization is to diminish the effect of variance/noise in the data and prevent over fitting.

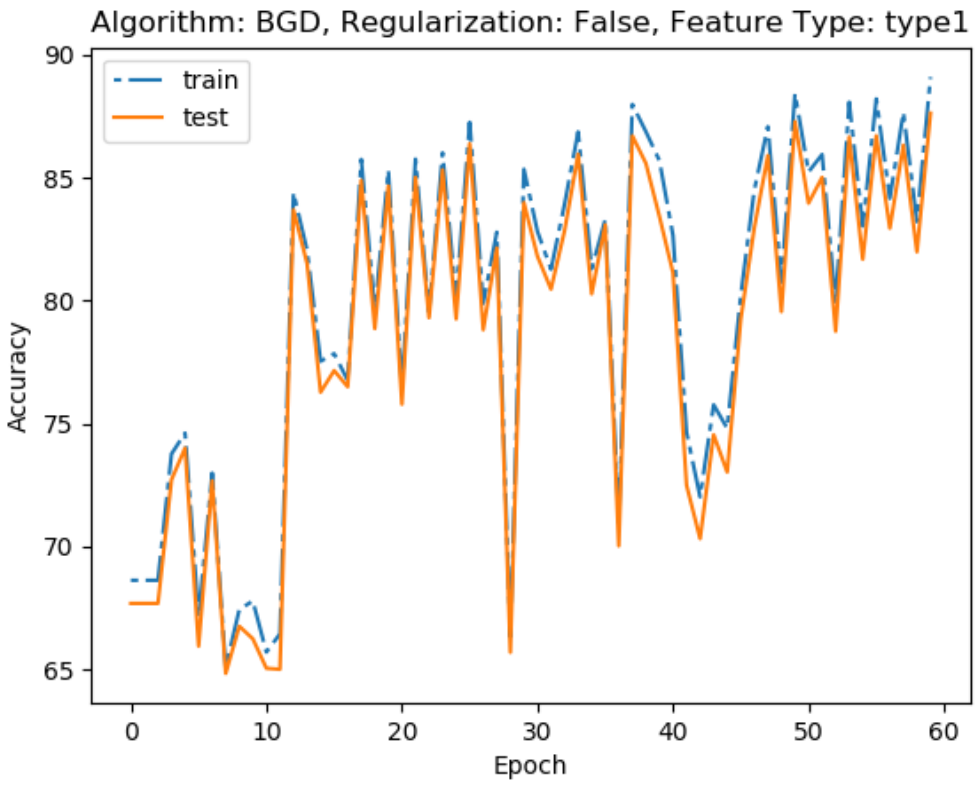

Figure 1: Accuracy v.s. Num Epochs for Batch Gradient Descent. Learning Rate = 0.1

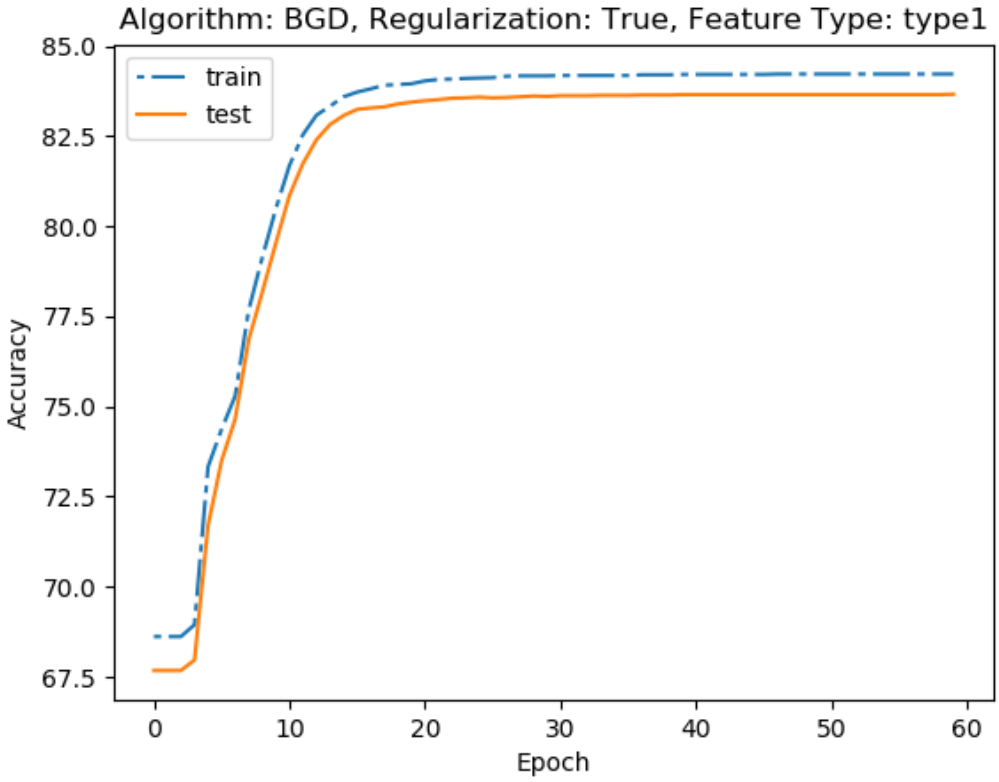

Figure 2: Accuracy v.s. Num Epochs for Batch Gradient Descent. $\lambda = 0.05$. Learning Rate $= 0.1$

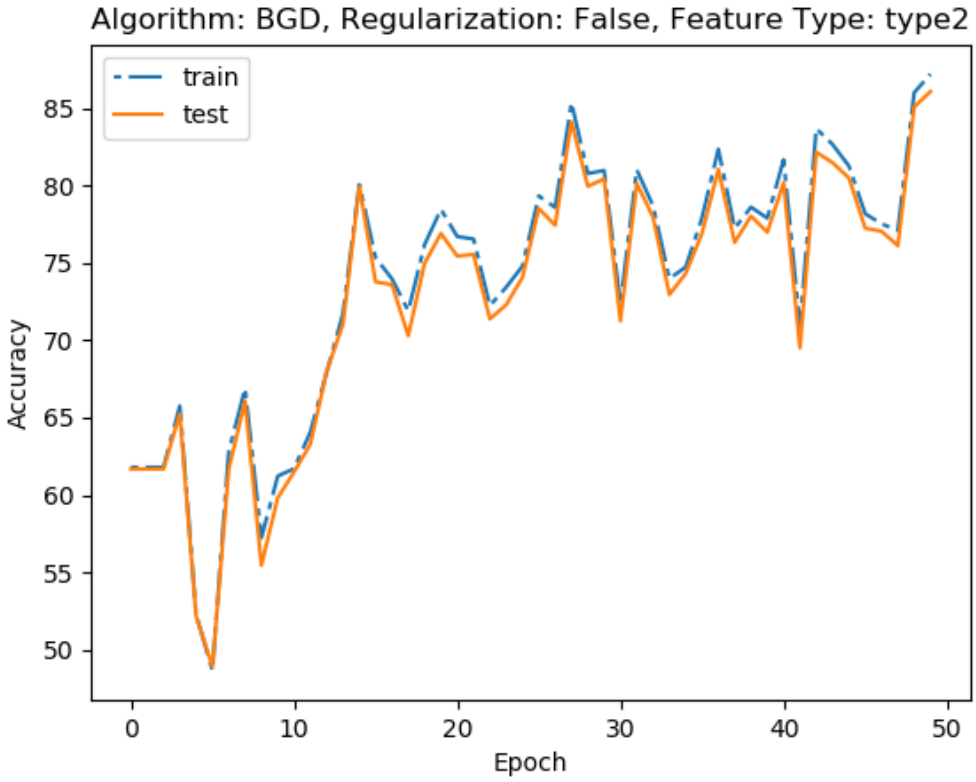

Figure 3: Accuracy v.s. Num Epochs for Batch Gradient Descent. Learning Rate = 0.1

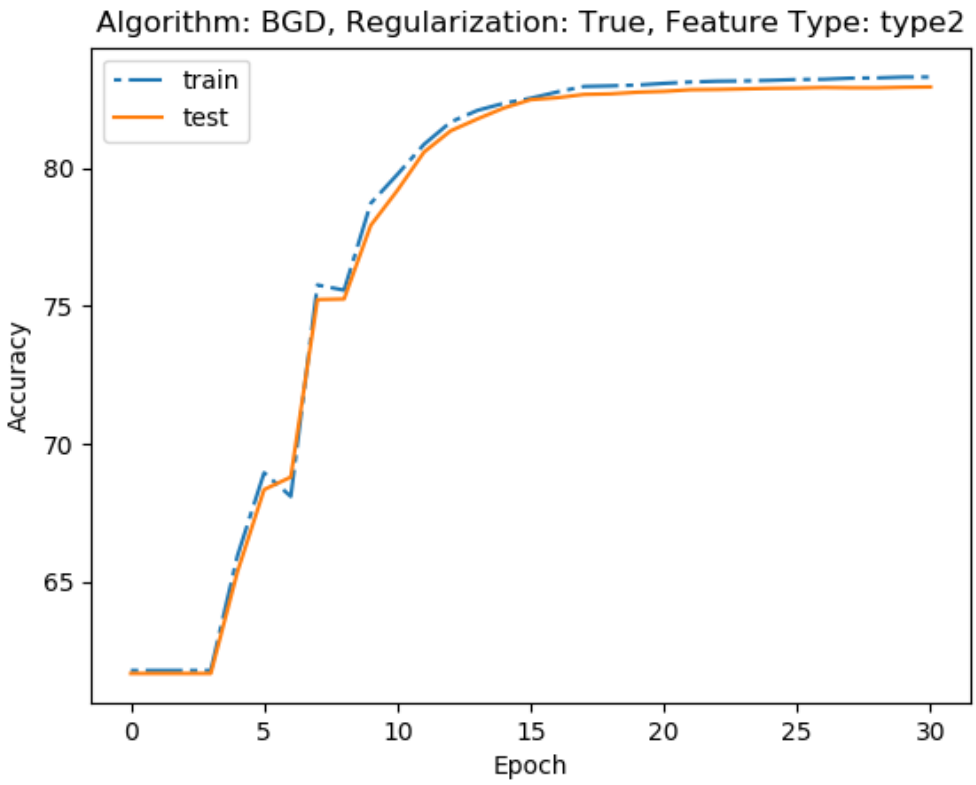

Figure 4: Accuracy v.s. Num Epochs for Batch Gradient Descent. $\lambda = 0.05$. Learning Rate = 0.1

2.2.1 Tuning λ

I tuned the value of λ by running the algorithm for different values of λ and each feature type. I set the number of epochs to be the value I got from the graphs above. We can see that for both the features, we get the best training accuracy when $\lambda = 0.05$. A smaller value (0.005) gives a behavior (fluctuations) similar to running without regularization at all. This is because such a small value has almost no effect on the weights and therefore, does not really provide any regularization. Larger values have a high effect on the weights (smoother lines) but result in lower accuracy as we highly discourage learning complicated hypothesis.

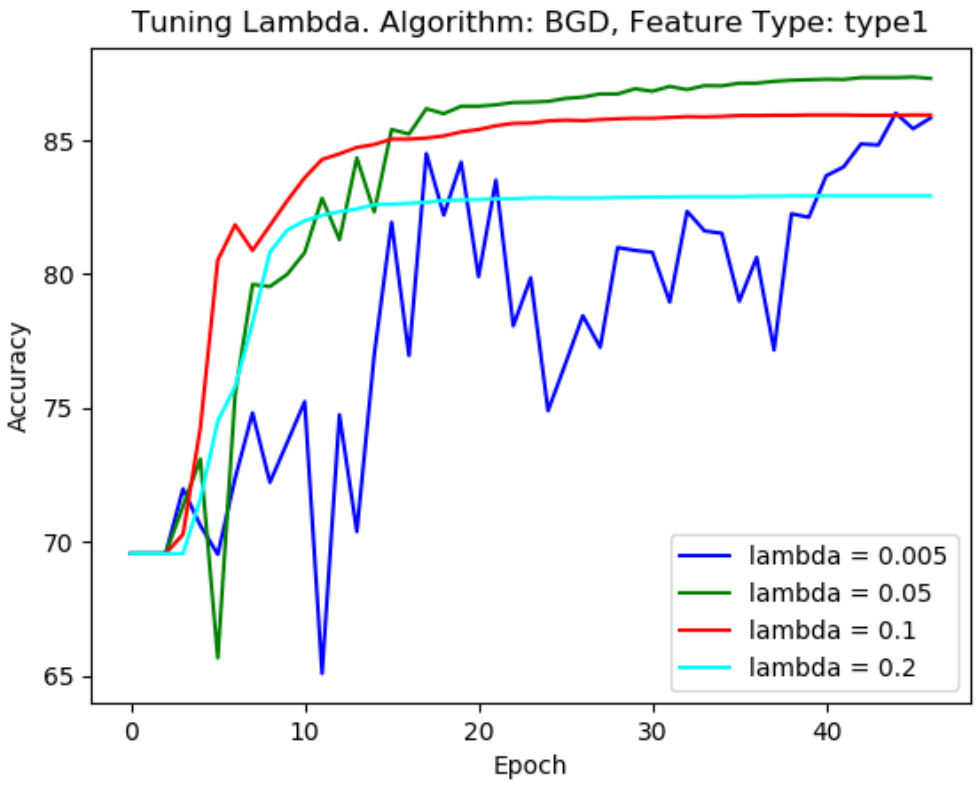

Figure 5: Accuracies for different values of lambda for feature type 1. Learning Rate = 0.1

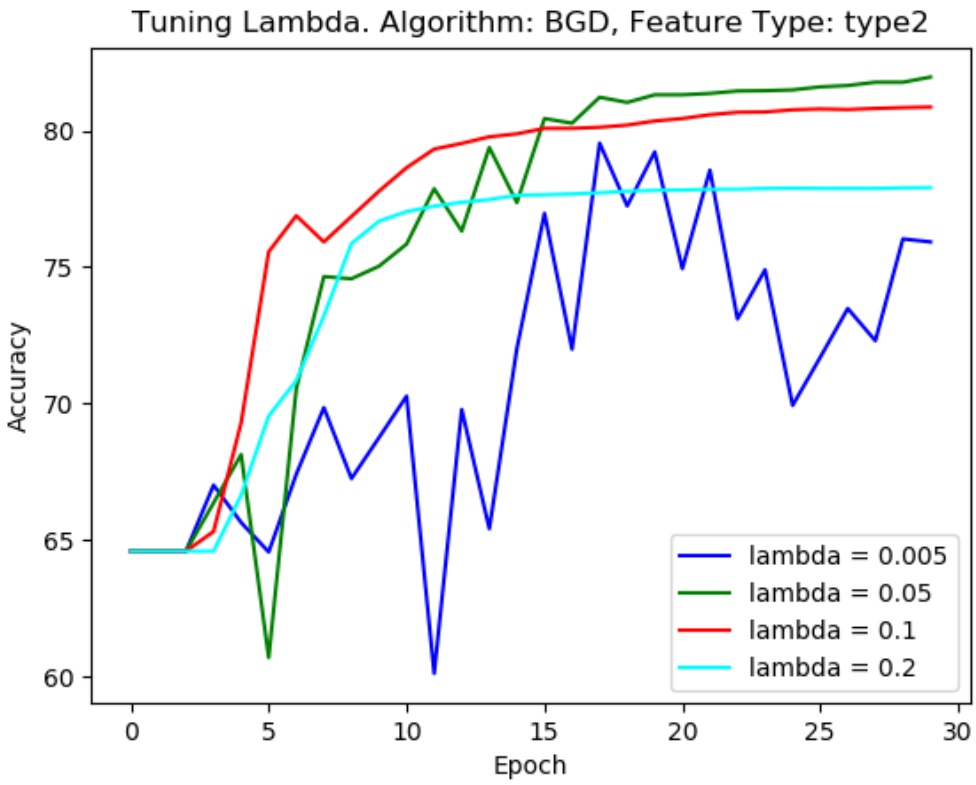

Figure 6: Accuracies for different values of lambda for feature type 2. Learning Rate = 0.1

2.3 Convergence Analysis

We can notice two interesting facts from the Accuracy v.s. Epoch graphs above.

First, notice that feature type 1 gives better results than feature type 2 - both with and without regularization. This is as expected because feature type 1 is much more representative of the data as compared to feature type 2. Feature type 2 results in an edge-type representation of the input example by taking the max value over a sliding window.

Second, the limited expressivity of feature type 2 does provide faster convergence times as can be seen from the graphs. This is a trade off that we can make, depending on how strict we are on the accuracy side.

3 Stochastic Gradient Descent with Logistic Function

3.1 Algorithm

Algorithm 2 Stochastic Gradient Descent(TrainData,NumEpochs,LearningRate,Digit)

$weights_d = random()$ for all d=0...numFeatures
*/*Number of epochs is the stopping criterion*/*
for *e = 0 ... numEpochs* **do**
  for *(img,imgLabel) in TrainData* **do**
    $y = 0$
    if *imgLabel == Digit* **then**
      $y = 1$
    end
    $activation = weights \cdot img$; `// compute activation for this image example by taking dot product`
    for *j = 0...numFeatures* **do**
      $\Delta w[j] = learningRate \times (y - logistic(act)) \times img[j]$; `// logistic() is the logistic/sigmoid function`
    end
    $weights = weights + \Delta w$
    $e = e + 1$; `// epoch definition is different for sgd`
  end
end
return *weights*

function LOGISTIC(z)
  return $1/(1+e^{-z})$
end function

3.2 Results

In stochastic gradient descent, we define an epoch to be whenever we update the weights. This can lead to a large number of epochs as we iterate over the data set multiple times if needed. Because of this, I recorded the accuracy readings after every 100 epochs.

Each of the following graphs has different number of epochs. The value of the epochs corresponds to the convergence criterion for that particular combination of regularization and feature type. I found the best value of epochs after running the experiments for a large initial value and recording the accuracies. I then found the minimum epochs corresponding to the maximum accuracy.

From the graphs, we can see that feature type 1 gives a better accuracy than feature type 2. We can also note that when we use regularization, the graphs tend to be smoother with less fluctuations but with a slightly less accuracy. This is as expected because the function of regularization is to diminish the effect of variance/noise in the data and prevent over fitting.

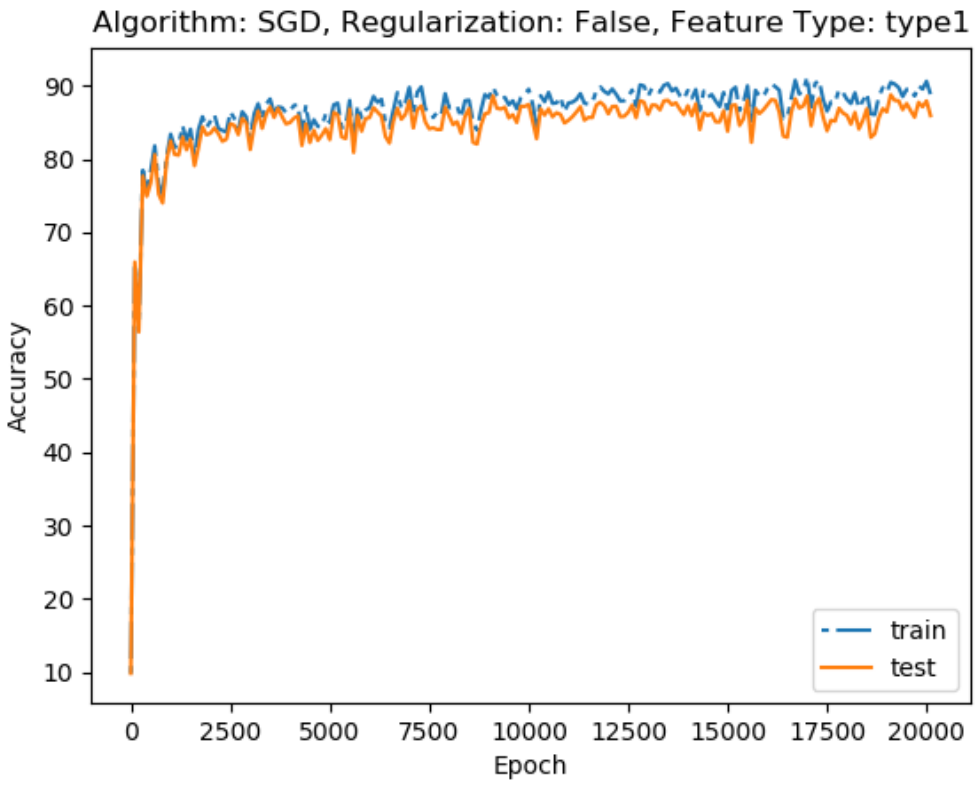

Figure 7: Accuracy v.s. Num Epochs for Stochastic Gradient Descent. Learning Rate = 0.1

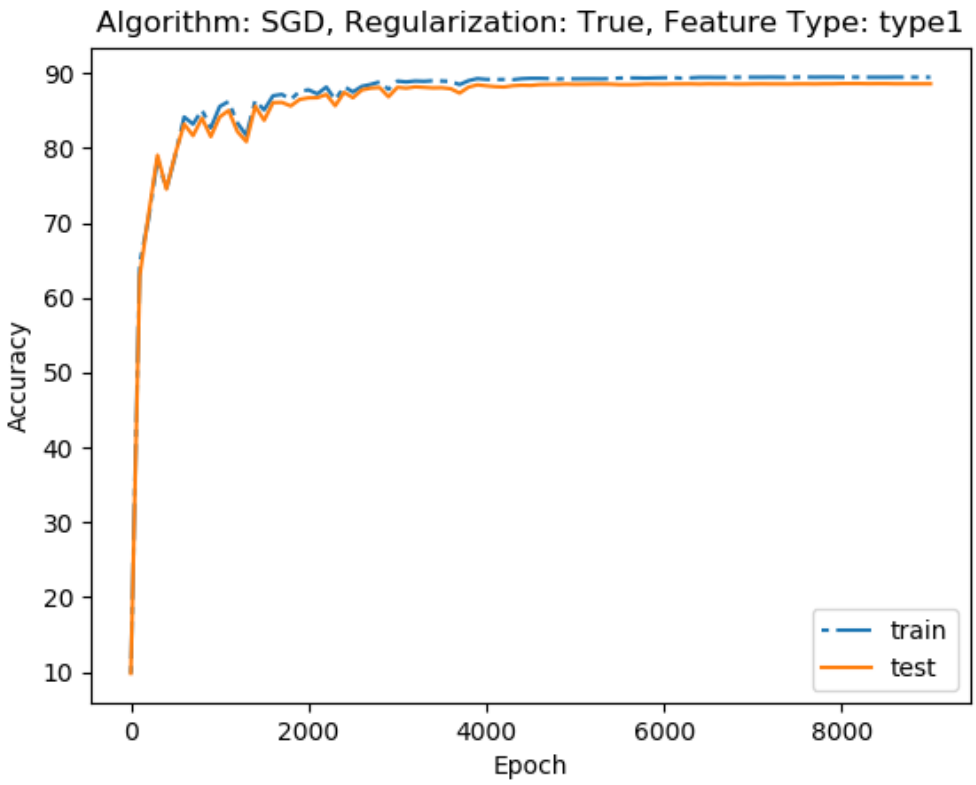

Figure 8: Accuracy v.s. Num Epochs for Stochastic Gradient Descent. $\lambda = 0.001$. Learning Rate = 0.1

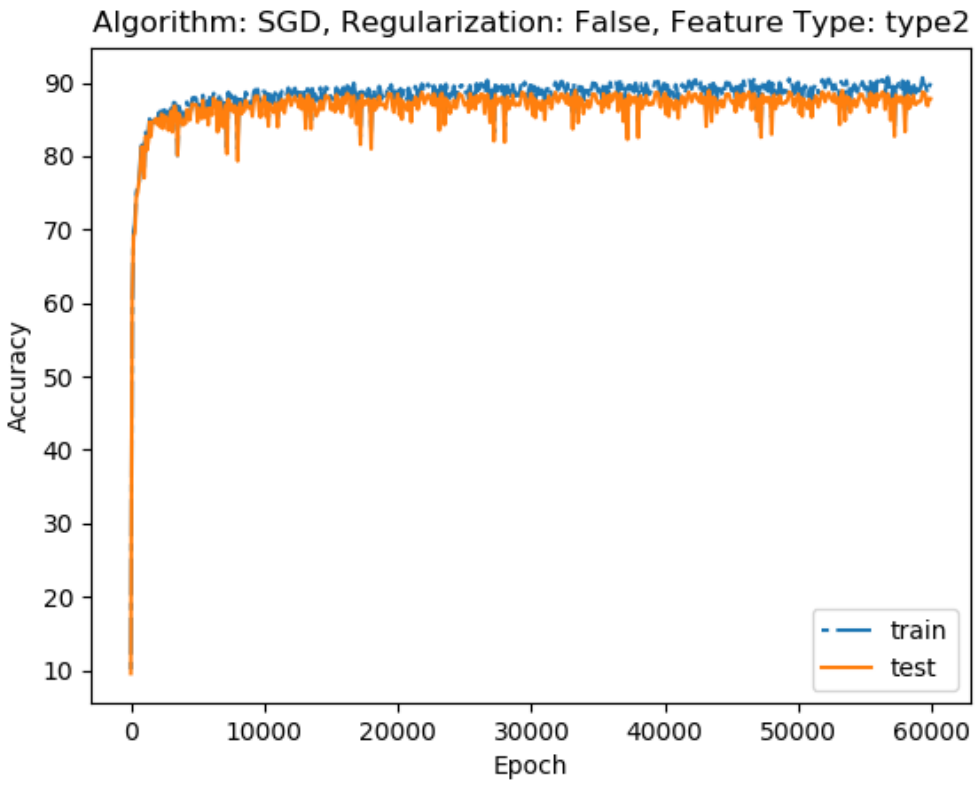

Figure 9: Accuracy v.s. Num Epochs for Stochastic Gradient Descent. Learning Rate = 0.1

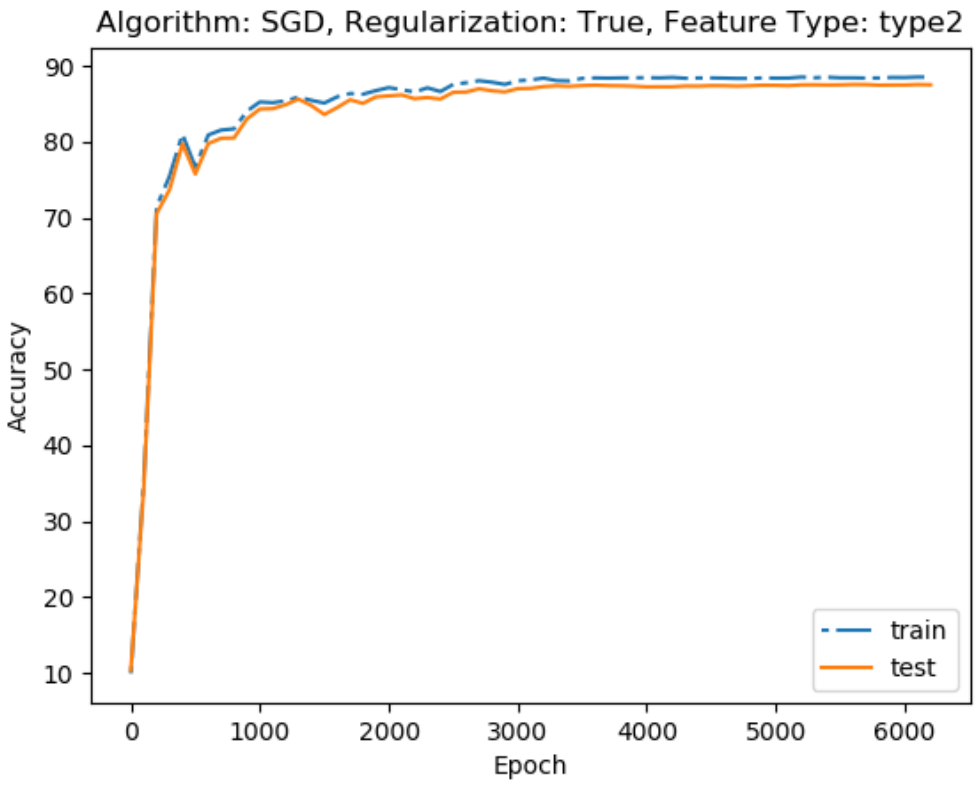

Figure 10: Accuracy v.s. Num Epochs for Stochastic Gradient Descent. $\lambda = 0.001$. Learning Rate = 0.1

3.2.1 Tuning λ

I tuned the value of λ by running the algorithm for different values of λ and each feature type. I set the number of epochs to be the value I got from the graphs above. We can see that for both the features, we get the best training accuracy when $\lambda = 0.001$. A smaller value (0.0001) gives a behavior (fluctuations) similar to running without regularization at all. This is because such a small value has almost no effect on the weights and therefore, does not really provide any regularization. Larger values have a high effect on the weights (smoother lines) but result in lower accuracy as we highly discourage learning complicated hypothesis.

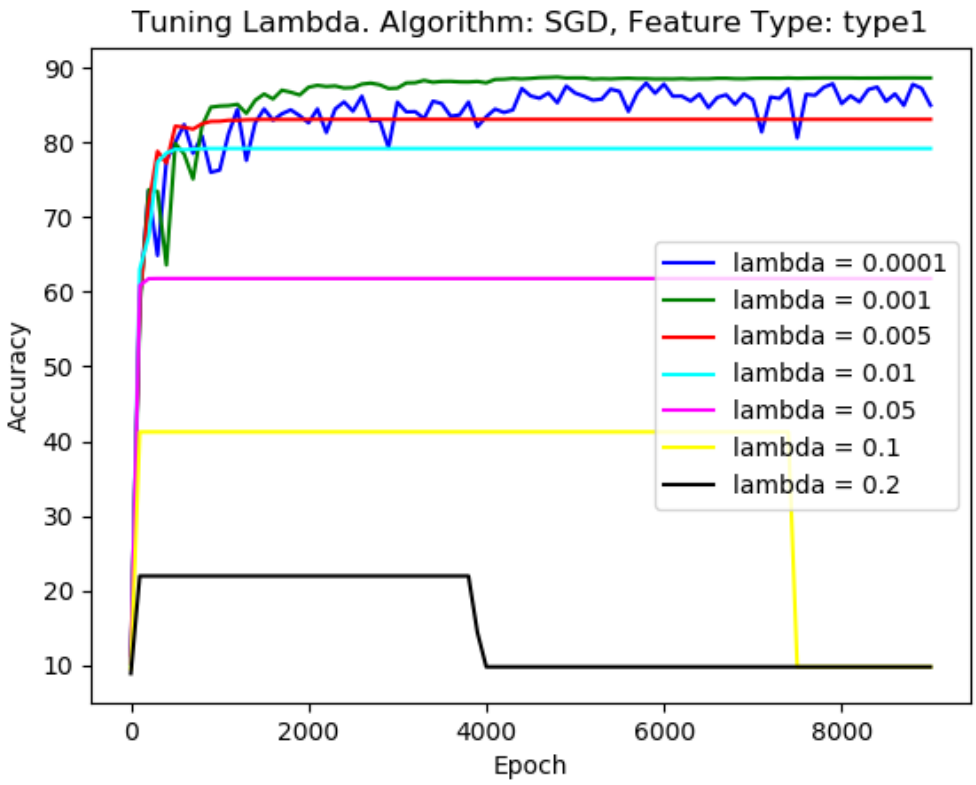

Figure 11: Accuracies for different values of lambda for feature type 1. Learning Rate = 0.1

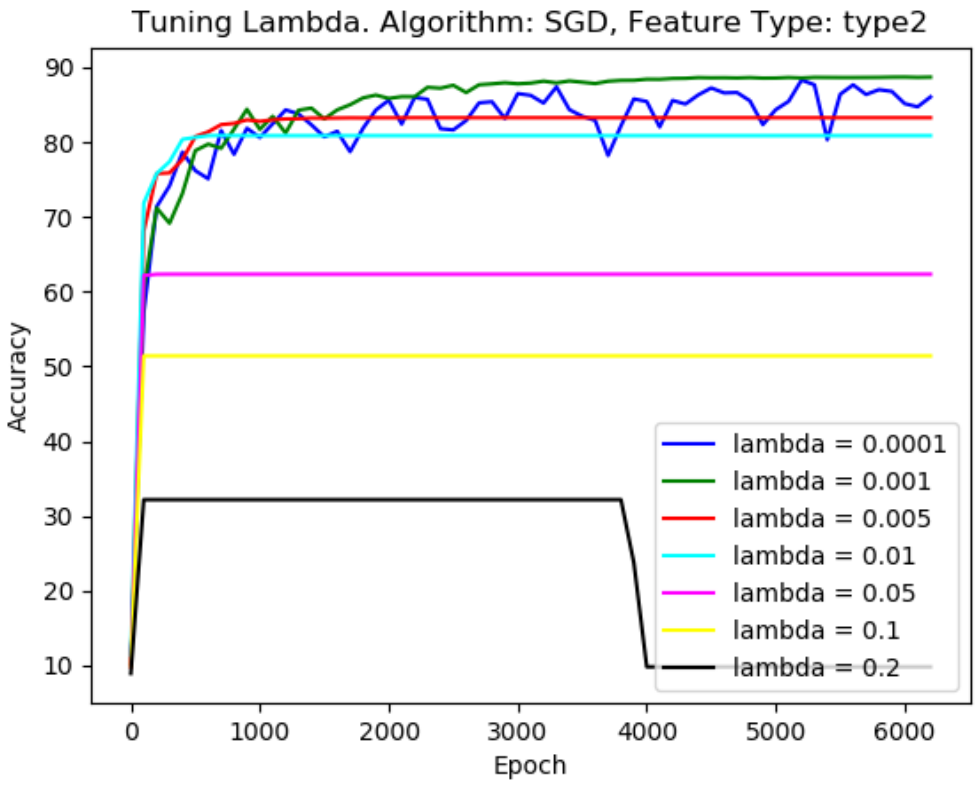

Figure 12: Accuracies for different values of lambda for feature type 2. Learning Rate = 0.1

3.3 Convergence Analysis

We can notice two interesting facts from the Accuracy v.s. Epoch graphs above.

First, notice that feature type 1 gives better results than feature type 2 - both with and without regularization. This is as expected because feature type 1 is much more representative of the data as compared to feature type 2. Feature type 2 results in an edge-type representation of the input example by taking the max value over a sliding window.

Second, the limited expressivity of feature type 2 does provide faster convergence times as can be seen from the graphs. This is a trade off that we can make, depending on how strict we are on the accuracy side.