Assignment 1 Theory Problem Set

DO NOT TAG

Name: Kevin McCarville
GT Email: kmccarville3@gatech.edu

Theory PS Q1. Feel free to add extra slides if needed.
Question:

In problem set 0, we derived the gradient of the log-sum-exp function (Q10). Now we will consider a similar function - the softmax function

$\mathbf{s}(\mathbf{z})$, which takes a vector input $\mathbf{z}$ and outputs a vector whose ith entry s_i is

$$s_i = \frac{e^{z_i}}{\sum_k e^{z_k}} \tag{1}$$

The input vector $\mathbf{z}$ to $\mathbf{s}(\cdot)$ is sometimes called the "logits", which just means the unscaled output of previous layers. Derive the gradient of $\mathbf{s}$ with respect to the logits, *i.e.* derive $\frac{\partial \mathbf{s}}{\partial \mathbf{z}}$. Consider re-using your work from PS0.

To derive the gradient of the softmax function with respect to the logits, let's start by writing down the softmax function more explicitly:

$$s_i(z) = \frac{e^{z_i}}{\sum_k e^{z_k}}$$

Here, si(z) is the i-th component of the softmax output vector s(z).

Now we'll express the softmax function in simpler terms:
S(z)=∑e^zk, for all k
Thus the softmax function can be rewritten as:

$$s_i(z) = \frac{e^{z_i}}{S(z)}$$

Theory PS Q1 (continued)

Now, we want to compute the partial derivative of si(z) with respect to the logits zj. There are two cases to consider:
Case 1: When i = j
Case 2: When i ≠ j

First Case 1:

$$\frac{\partial s_i}{\partial z_j} = \frac{\partial}{\partial z_j}\left(\frac{e^{z_i}}{S(z)}\right)$$

Now we apply the quotient rule:

$$\frac{\partial s_i}{\partial z_j} = \frac{S(z)\cdot \frac{\partial}{\partial z_j}\left(e^{z_i}\right) - e^{z_i}\cdot \frac{\partial}{\partial z_j}\left(S(z)\right)}{(S(z))^2}$$

Now, let's compute the derivatives:

$$\frac{\partial e^{z_i}}{\partial z_i} = e^{z_i} \text{ (since } i = j\text{)}.$$

$$\frac{\partial S(z)}{\partial z_j} = \frac{\partial}{\partial z_j}\left(\sum_k e^{z_k}\right) = e^{z_j}.$$

Substituting these into the expression:

$$\frac{\partial s_i}{\partial z_j} = \frac{S(z)\cdot e^{z_i} - e^{z_i}\cdot e^{z_j}}{(S(z))^2}$$

Since i = j, e^zj = e^zi:

$$\frac{\partial s_i}{\partial z_i} = \frac{e^{z_i}\cdot (S(z) - e^{z_i})}{(S(z))^2}$$

we can express this as:

$$\frac{\partial s_i}{\partial z_i} = \frac{e^{z_i}}{S(z)}\cdot\left(\frac{S(z) - e^{z_i}}{S(z)}\right) = s_i(z)\cdot(1 - s_i(z))$$

Now Case 2 where i ≠ j

$$\frac{\partial s_i}{\partial z_j} = \frac{\partial}{\partial z_j}\left(\frac{e^{z_i}}{S(z)}\right)$$

Apply the quotient rule:

$$\frac{\partial s_i}{\partial z_j} = \frac{S(z)\cdot \frac{\partial e^{z_i}}{\partial z_j} - e^{z_i}\cdot \frac{\partial S(z)}{\partial z_j}}{(S(z))^2}$$

Theory PS Q1 (continued)

Now $\frac{\partial e^{z_i}}{\partial z_j} = 0$, since i ≠ j, so we are left with:

$$\frac{\partial s_i}{\partial z_j} = -\frac{e^{z_i} \cdot e^{z_j}}{(S(z))^2}$$

Rewriting using the softmax definition:

$$\frac{\partial s_i}{\partial z_j} = -s_i(z) \cdot s_j(z)$$

Bringing it together we get:

$$\frac{\partial s_i}{\partial z_j} = \begin{cases} s_i(z) \cdot (1 - s_i(z)), & \text{if } i = j, \\ -s_i(z) \cdot s_j(z), & \text{if } i \neq j. \end{cases}$$

Theory PS Q2. Feel free to add extra slides if needed.
Question:

Implement AND and OR for pairs of binary inputs using a single linear threshold neuron with weights $\mathbf{w} \in \mathbb{R}^2$, bias $b \in \mathbb{R}$, and $\mathbf{x} \in \{0,1\}^2$:

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w}^T\mathbf{x} + b \geq 0 \\ 0 & \text{if } \mathbf{w}^T\mathbf{x} + b < 0 \end{cases} \tag{2}$$

That is, find $\mathbf{w}_{\text{AND}}$ and b_{AND} such that

x_1	x_2	$f_{\text{AND}}(\mathbf{x})$
0	0	0
0	1	0
1	0	0
1	1	1

Also find $\mathbf{w}_{\text{OR}}$ and b_{OR} such that

x_1	x_2	$f_{\text{OR}}(\mathbf{x})$
0	0	0
0	1	1
1	0	1
1	1	1

Theory PS Q2 (continued)

AND Operation:
We want the neuron to output 1 only when both inputs are 1.
Let's define the weights and bias:

$$\mathbf{w}_{AND} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}, \quad b_{AND}$$

We have the following conditions:
f(0,0)=0 ⟹ w1·0+w2·0+b<0⟹b<0
f(0,1)=0 ⟹ w1·0+w2·1+b<0⟹w2+b<0
f(1,0)=0 ⟹ w1·1+w2·0+b<0⟹w1+b<0
f(1,1)=1⟹ w1·1+w2·1+b≥0⟹w1+w2+b≥0

To satisfy these conditions, we can choose:

$$\mathbf{w}_{AND} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad b_{AND} = -1.5$$

Verification of these values:
f(0,0)=1·0+1·0−1.5=−1.5 <0 (Output: 0)
f(0,1)=1·0+1·1−1.5=−0.5 <0 (Output: 0)
f(1,0)=1·1+1·0−1.5=−0.5 <0 (Output: 0)
f(1,1)=1·1+1·1−1.5= 0.5 ≥ 0 (Output: 1)

OR Operation
Using the same logic, we define:

$$\mathbf{w}_{OR} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}, \quad b_{OR}$$

We have the following conditions:
f(0,0)=0 ⟹ w1·0+w2·0+b<0⟹b<0
f(0,1)=1 ⟹ w1·0+w2·1+b<0⟹w2+b<0
f(1,0)=1 ⟹ w1·1+w2·0+b<0⟹w1+b<0
f(1,1)=1⟹ w1·1+w2·1+b≥0⟹w1+w2+b≥0

To satisfy these conditions, we can choose:

$$\mathbf{w}_{OR} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad b_{OR} = -0.5$$

Verification of these values:
f(0,0)=1·0+1·0−0.5=−0.5 < 0 (Output: 0)
f(0,1)=1·0+1·1−0.5= 0.5 ≥ 0 (Output: 1)
f(1,0)=1·1+1·0−0.5= 0.5 ≥ 0 (Output: 1)
f(1,1)=1·1+1·1−0.5= 1.5 ≥ 0 (Output: 1)

Theory PS Q3. Feel free to add extra slides if needed.
Question:
Consider the XOR function
Prove that XOR can NOT be represented using a linear model with
the same form as (2).

A linear model (like a perceptron) aims to separate data using a linear boundary (a straight line in 2D). The equation of the boundary is typically of the form:
w1*x1 + w2*x2 +b = 0
Here, w1, w2, and b are the weights and bias that define the boundary.

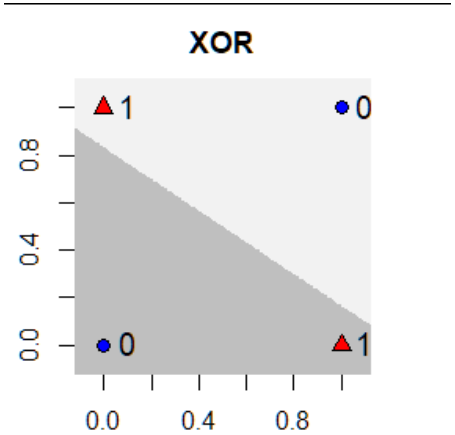
For XOR, we need a boundary that separates:
Points (0,1) and (1,0) (label 1) from
Points (0,0) and (1,1) (label 0)

However, this is not possible:
Any straight line that tries to separate (0,0) from (1,0) or (0,1) will incorrectly include/exclude (1,1) or (0,0)

The points (0,0) and (1,1) are diagonally opposite each other, as are (0,1) and (1,0). This requires a non-linear boundary to separate them correctly.

As can be seen in the figure to the right, there is no line that separates the values of 1 and 0.

Since no single straight line can separate the points of different labels for XOR, the XOR function cannot be represented using a linear model. This is a classic example demonstrating the need for non-linear models (such as multi-layer neural networks) to correctly model XOR.



Assignment 1 Paper Review

DO NOT TAG

Name: Kevin McCarville
GT Email: kmccarville3@gatech.edu

Provide a short preview of the paper of your choice.

For this portion of the assignment, I chose to review: "Understanding deep learning requires rethinking Generalization" by Zhang, et al.

Short review plus answers to the following questions (< 350 words):
- What is the main contribution of this paper? In other words, briefly summarize its key insights. What are some strengths and weaknesses of this paper?
- What is your personal takeaway from this paper? This could be expressed either in terms of your perceived novelty of this paper compared to others you've read in the field, potential future directions of research in the area that the authors haven't addressed, or anything else that struck you as being noteworthy.

The paper challenges traditional views of generalization in machine learning by showing that deep networks can fit random labels and noise, but still generalize well on structured data. The authors perform empirical experiments to demonstrate that deep neural networks have the capacity to memorize noise, and their generalization performance cannot be fully explained by classical generalization theories. They did this first by scrambling the labels, disconnecting them from the ground truth, and then injecting noise into the input images. They propose that the training process, particularly optimization methods like stochastic gradient descent, plays an implicit regularization role helping deep networks generalize, despite their over-parameterization.

The paper's key insight is that traditional metrics like VC dimension and uniform convergence cannot explain why deep learning models generalize well, even when they can fit noisy data. This raises the need for a new theoretical framework for understanding generalization in deep learning. The paper's strengths include its rigorous empirical investigation and thought-provoking challenge to long-held assumptions. However, one potential weakness is that the paper does not provide a clear theoretical framework for the observed phenomena, leaving questions unanswered about the mechanisms of implicit regularization. Its also impressive though not shocking, that with an input space as rich as images, there would be some random set of attributes the algorithm could learn to perfectly categorize the training data, even if it is just noise. This fact went mostly unaddressed.

Provide a short preview of the paper of your choice (continued)

What strikes me most about this paper is its disruptive nature - it forces a reevaluation of widely accepted principles of generalization in machine learning. The novel insight into deep networks' capacity to generalize despite fitting random data is a crucial starting point for new theories. It begs the question what neural networks are actually doing when the data isn’t scrambled. One would like to think they would be able to tell us when there are no patterns worth knowing, however, that does not seem to be the case. It also increases the importance of interpretability. How often are ostensibly “working” models keying off random unreliable patterns instead of the actual signals in order to produce accurate outputs?

Paper specific Q1. Feel free to add extra slides if needed.

Paper Specific Questions:
- If neural networks can “memorize” the data, which is the only thing they can do for random label assignments that don’t correlate with patterns in the data, why do you think neural networks learn more meaningful, generalizable representations when there are meaningful patterns in the data?

Neural networks "memorize" data when there are no meaningful patterns, but when such patterns exist, they learn generalizable representations due to the structure of the optimization process. Stochastic gradient descent tends to favor simpler, more generalizable solutions over memorization, even though the network has the capacity to memorize noise. When trained on meaningful patterns, the network aligns its parameters with the underlying data structure, capturing relationships that generalize well to new data. This is likely because SGD does not explore the entire hypothesis space uniformly; instead, it follows gradients that reinforce more meaningful features over noise. The gradient it follows will not be a small one in order to memorize one more sample. It will be the steepest gradient available, improving performance amongst several observations. These observations likely share characteristics the model can learn indicative of the underlying data rather than rote memorization.

In laymen’s terms, the model will try to learn to map labels to data as quickly and efficiently as possible. For example, if it is trying to learn whether an animal is a mammal or a reptile, it could just memorize every animal in each category one by one. However, it would be much quicker to learn that reptiles mostly have scales and mammals mostly have fur or other similar characteristics. Because these facts are more available than memorization (steeper gradient) it learns them first, but it will memorize if it has to (those gradients do not exist because the data has been scrambled)

Additionally, the inductive bias of neural networks plays a role. While deep networks have enough capacity to fit random labels, when presented with real-world data, they exploit the inherent structure in the data—like spatial correlations in images or temporal dependencies in sequences. As a result, even though the network can memorize, it still prioritizes learning useful representations when patterns exist. This implicit regularization, driven by the optimization process and the nature of the data, helps the network avoid overfitting noise.

Paper specific Q2. Feel free to add extra slides if needed.

Paper Specific Questions:
- How does this finding align or not align with your understanding of machine learning and generalization?

Honestly, this finding mostly aligns with my understanding. Traditionally, I believed that models generalize based on a balance between model complexity and regularization techniques. This paper does show that deep networks can generalize even without traditional regularization, suggesting that generalization is more strongly tied to the optimization dynamics of SGD and the structure of the data than previously thought. However the idea that networks can perfectly memorize data was not new to me. The fact that when all the underlying data was injected with randomness and the labels were jumbled to have no correlation and yet the model could still learn to memorize the data was not new knowledge to me. These models will go out and learn something, even if there is nothing useful to learn; that's what they do.

However, the fact that these models trained on noise still generalize well on real data was a surprise to me. This does indeed challenge some of my previous notions of how the training process was theoretically supposed to operate as well as conventional wisdom about overfitting and model capacity. I had previously assumed that if a model could memorize noise, it would be prone to overfitting and perform poorly on unseen data. However, the paper suggests that the optimization process has a built-in regularization effect that helps models favor simpler solutions, even if they have the capacity to memorize random data. This insight raises deeper questions about the role of implicit regularization in deep learning, and whether new theoretical frameworks are needed to fully explain why certain models generalize well despite simply attempting to memorize input data.

Additionally, this makes the interpretability of models even more important—since these models will "learn" something regardless of the data, it becomes crucial to ensure they are learning meaningful, reliable patterns instead of noise that happens to lead to correct predictions.

Assignment 1 Writeup

DO NOT TAG

Name: Kevin McCarville
GT Email: kmccarville3@gatech.edu

Two-Layer Neural Network

DO NOT TAG

1. Learning Rates

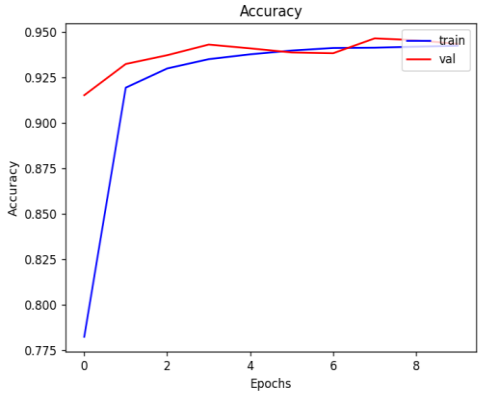
Tune the learning rate of the model with all other default hyper-parameters fixed. Fill in the table below:

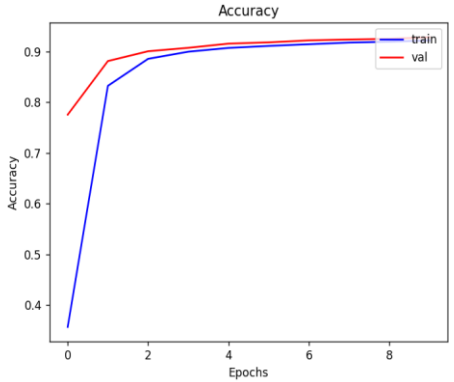
	lr=1	lr=1e-1	lr=5e-2	lr=1e-2
Training Accuracy	0.9423	0.9218	0.9134	0.7297
Test Accuracy	0.9461	0.9250	0.9089	0.7501

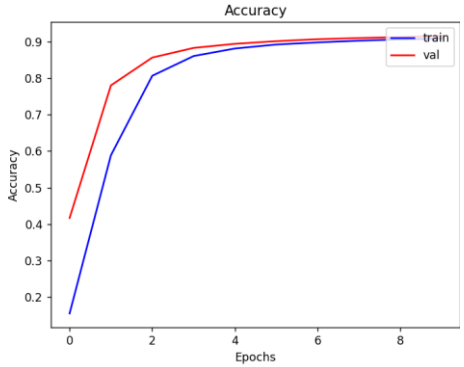
1. Learning Curve

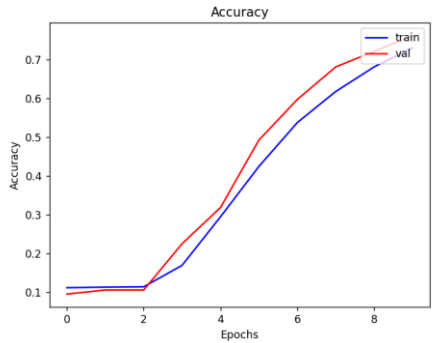
Plot the learning curves using the learning rates from the previous slide and put them below (you may add additional slides if needed).

From left to right, learning rates of 1, 0.1, 0.05, and 0.01









1. Learning Rates

Describe and Explain your findings: *Explanation should go into **WHY** things work the way they do in the context of Machine Learning theory/intuition, along with justification for your experimentation methodology. **DO NOT** just describe the results, for example, you should explain why the learning rate has the observed effect. Also, be cognizant of the best way to organize and show the results that best emphasizes your key observations. If you need more than one slide to answer the question, you are free to create new slides.*

The larger learning rate is producing better results for our model and as we decrease the learning rate, the performance degrades. The reason this is happening is that the model is underfit for the 3 lower learning rates. We see this in the learning curves as while the curves are flattening, they are still increasing indicating more improvement is left to be done (that could be achieved by increasing epochs). This is especially prominent in the chart of the 0.001 learning rate. The downside to have too large a learning rate is that it follows the gradient too far and overshoots the minima it was aiming for. This leads to unstable learning and can even be explosive in certain scenarios. The reason we don't see this here is that our data is mostly linearly separable / not too noisy. A 4 simply doesn't look much like an 8. Thus, our gradients will be more pure and less noisy and thus we can with more confidence, follow them down with a higher learning rate. One other noteworthy observation is the test value often outperforming the training batch average. This is a phenomenon mentioned in the slides. The test batch actually has the added benefit of the training done on the last training batch, while the training accuracy is an average of all the samples done during its batch. Thus the test batch has a slightly more developed model which explains the improvement. We can even see the training an test accuracy flipping back and forth towards the end of the lr=1 model as the model is more or less fully trained and thus there is minimal improvement going on and any difference in results will mostly be the result of random batch sampling. Also note that 100% accuracy isn't really possible do to human error in the data. If a human writes a 7 and says it's a 6, there really isn't anything the model can do to account for that. This is a problem with many datasets and a well known issue with the MNIST dataset. Garbage in, Garbage out.

2. Regularization

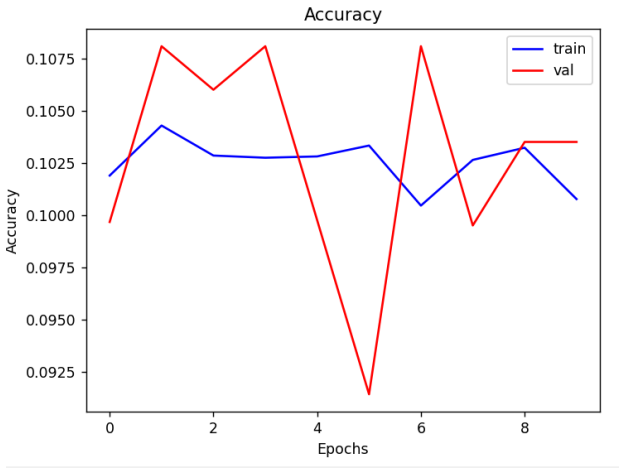
Tune the regularization coefficient of the model with all other default hyper-parameters fixed. Fill in the table below:

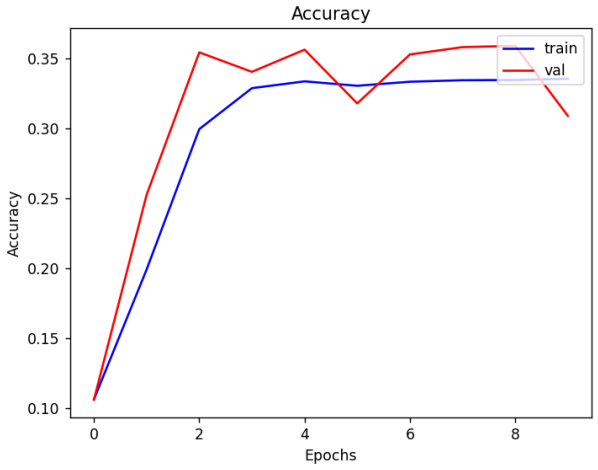
	alpha=1	alpha=1e-1	alpha=1e-2	alpha=1e-3	alpha=1e-4
Training Accuracy	0.1008	0.3356	0.8833	0.9213	0.9302
Validation Accuracy	0.1035	0.3090	0.8942	0.9277	0.9347
Test Accuracy	0.1028	0.3597	0.8921	0.9263	0.9318

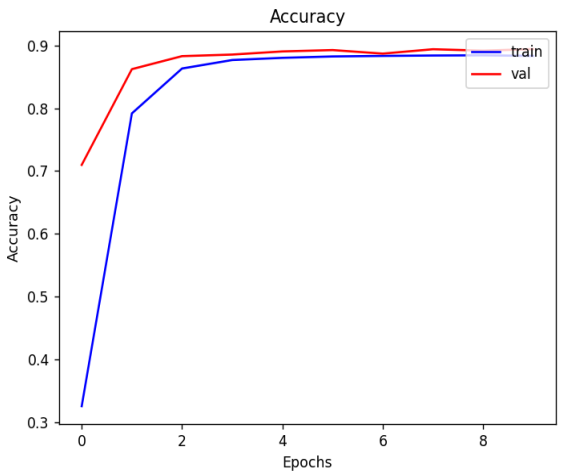
2. Regularization

Plot the learning curves using the regularization coefficients from the previous slide and put them below (you may add additional slides if needed).

From left to right, regularization rates of 1, 0.1, and 0.01



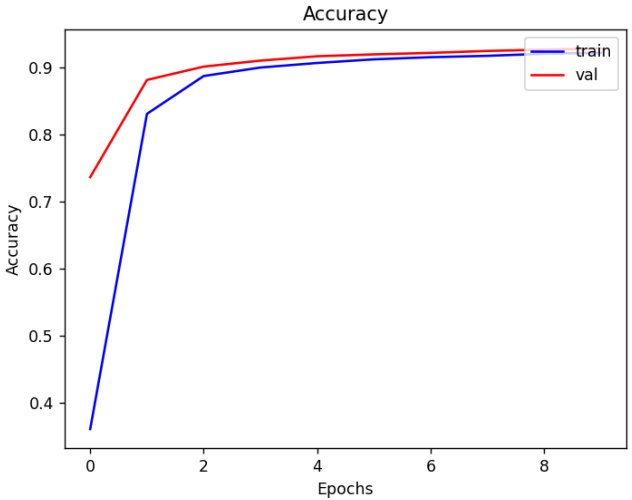


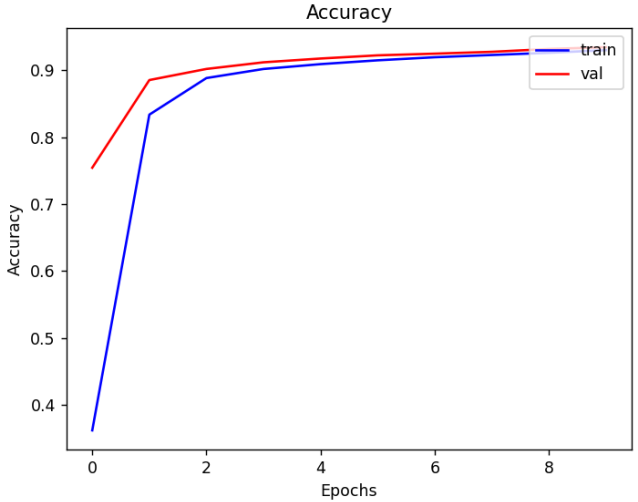


2. Regularization (continued)

Plot the learning curves using the regularization coefficients from the previous slide and put them below (you may add additional slides if needed).

From left to right, regularization rates of .001, 0.0001





2. Regularization

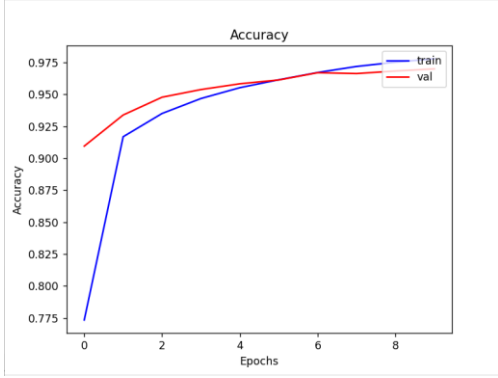
Describe and Explain your findings: *Explanation should go into **WHY** things work the way they do in the context of Machine Learning theory/intuition, along with justification for your experimentation methodology. **DO NOT** just describe the results, for example, you should explain why the regularization value affects performance as well as model weights. Also, be mindful of the best way to organize and show the results that best emphasizes your key observations. If you need more than one slide to answer the question, you are free to create new slides.*

The performance of the algorithm improved as we lowered the regularization. It did particularly well with our two smallest values for alpha and did extremely poorly with our two highest. This is logical given the nature of our experiment. The goal of regularization is to prevent overfitting by constraining the complexity of the model to help with generalization to new data. It does this by penalizing the model (adding to the loss function) any larger weights at different nodes. Our model however is showing no signs of overfitting. Our test accuracy is not considerably lower than our training accuracy (in fact its often higher) nor is our loss function increasing on test data as the model continues to train. Thus, regularization in this scenario will simply hold back performance of the model by not allowing it an improved complexity and lowering it will improve accuracy.

This makes sense with our MNIST dataset. Image detection involves complex non-linear relationships, in need of larger weight parameters to pattern match. The model needs the added complexity in order to fit the categorization of each digit. This is what the model with lower regularization is accomplishing as we allow it more freedom to develop complex relationships between its parameters.

We are seeing this level off at the lower levels of regularization. At some point the alpha is so close to 0 that it simply is not constraining the weights in any meaningful way, and we are starting to see that as the performance levels off around the very low values for alpha.

3. Hyper-parameter Tuning



You are now free to tune any hyper-parameters for better accuracy. Create a table below and put the configuration of your best model and accuracy into the table:

Momentum	Epochs	LR	**Alpha (reg)**	**Batch Size**	**Hidden Layer**	Training Acc.	Validation Acc.	Testing Acc.
0.9	10	0.1	**0.000001**	**12**	**256**	**0.9779**	**0.9699**	**0.9703**

Bold are the values that were changed from the default values

Explain why your choice works: *Explanation should go into **WHY** things work the way they do in the context of Machine Learning theory/intuition, along with justification for your experimentation methodology. **DO NOT** just describe the results, you should explain the reasoning behind your choices and what behavior you expected. Also, be cognizant of the best way to mindful and show the results that best emphasizes your key observations. If you need more than one slide to answer the question, you are free to create new slides.*

3. Hyper-parameter Tuning

You are now free to tune any hyper-parameters for better accuracy. Create a table below and put the configuration of your best model and accuracy into the table:

After much trial and error (and following our own gradient of learning), the variables with the most impact and improvement on performance were batch size, number of hidden layer nodes, and regularization. First, we will describe what we didn't change and why.

Momentum – tuning this hyperparameter had minimal effect on performance. We found as we increased it, the model learned quicker. This makes sense as our input space is not particularly noisy as mentioned previously so it will help the learning down the gradient in the early going. However, it did not materially improve the performance by the end of the epochs.

Learning Rate – Increasing learning rate did improve performance. This again makes sense and is in line with what we have seen from our earlier experiments. Our limitation is that we can only tune 3 hyperparameters so this one had to stay with the default so we could train others that had a bigger impact. The increased learning rate, similar to momentum, tended to increase the speed of training, but not necessarily the overall end state performance. This makes sense as it is after all the "learning rate". Our model was not complex enough, so the learning rate got the model to more or less its maximal performance, just quicker. So, it was kept as its default, and we improved elsewhere.

Alpha - The first parameter we changed was the regularization value. We saw in the previous experiments how important it was to let the model have more freedom to increase complexity so we lowered this value more or less to 0 (0.000001). This was a necessary first step to allow the model to improve on the ~92% achieved by the default values as now the parameters could adjust better to fit the data. This change had far and away the greatest impact on performance. We did risk some overfitting here but with just 10 epochs, it did not really hamper our results.

3. Hyper-parameter Tuning

Hidden Layer Nodes – We increased this to 256 nodes. This improved performance as now there are more parameters for the model to train to better fit the data. This did come at the expense of compute time but it was not a limiting factor by any means. We also tried increasing to 512 and 1024 nodes but the gains there were negligible and even worse on some runs so we stayed with 256. This was a little surprising but it appears to indicate that we are reaching the bounds on how much more nodes can improve performance in a theoretical sense. It would be interesting to see if adding more layers would further improve performance.

Batch Size – Finally we tuned the batch size. We settled on 12 as that appears to be the best performance although we did tune the value both higher and lower. The more frequent updates than the 64 batch size allowed for more precise and incremental updates which allowed the model to get further down its gradient by the end of the epochs. It was interesting that lowering it below 8 seemed to have a significant negative effect on performance. Our best guess is the smaller batches create nosy updates based on less data that may not be helpful and can create slower convergence. We were somewhat surprised that this was one of the parameters we ended up tuning as it wasn't an intuitive choice. Why would more frequent updates instead of less frequent ones base on more data create significant improvement? For the reasons above, it became clear why

Overall, we improved performance on the test set from ~92% to about 97%. A pretty significant improvement as 62.5% of the errors were corrected from the hyperparameter tuning. This is very impressive considering how simple our model is and the fact that the dataset has a maximum theoretical accuracy of ~99.6% given the inherent noisiness of the dataset. For future research, I believe creating a more complex setup, adding more layers and more nodes (and of course more epochs) would have the greatest impact on the remaining errors our model is making.