Nov 18th, 2022 Report

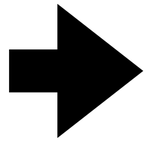
CSCI496

Cameron Kaminski

Code Changes

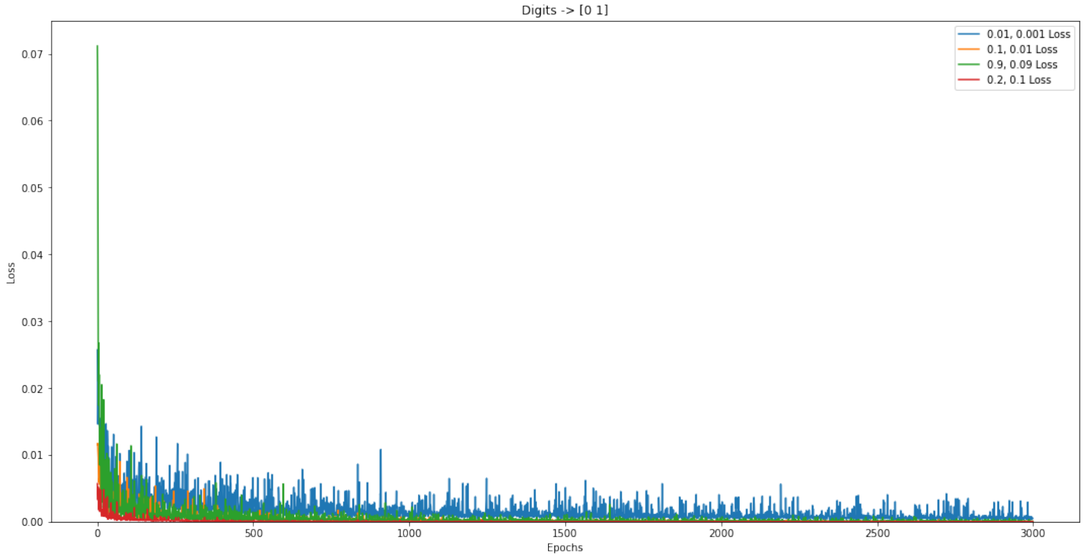
Minor Update to Fix Error

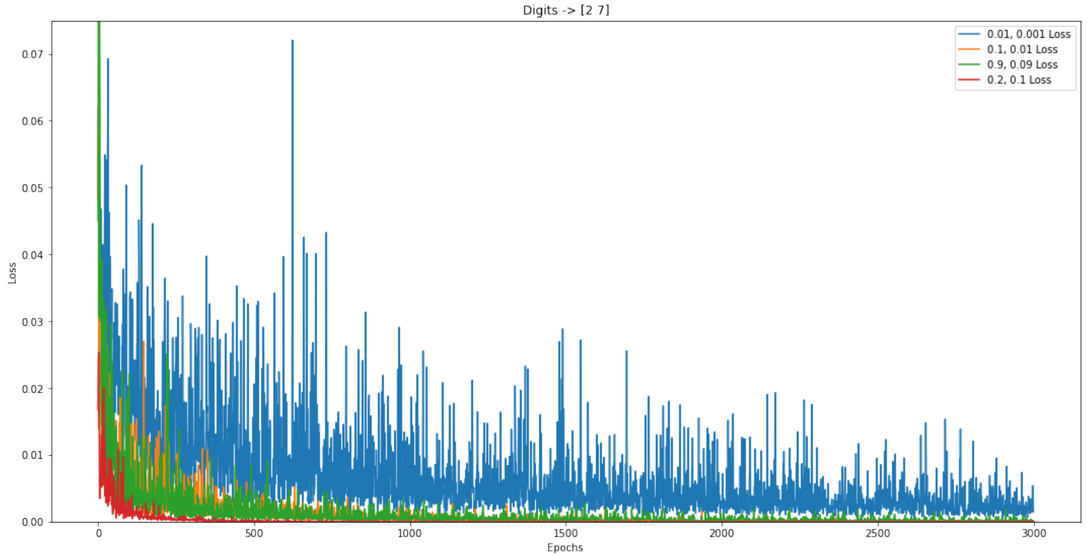
```python
def train_one_epoch(self, loader, loss_function, optimizer, record=True):

    # Array of centered kernal analysis.
    cka = torch.zeros(len(loader))

    for i, (data, targets) in enumerate(loader):
        data = data.reshape(data.shape[0], -1).to(device=self.device)
        targets = targets.to(torch.float32).to(device=self.device)
        classified_targets = classify_targets(targets, self.values)

        # Forwards pass.
        scores = self(data)

        labels = one_hot(classified_targets.long()% len(self.values)).to(torch.float32)
        output = loss_function(scores, labels)

        # Backwards Pass.
        optimizer.zero_grad()
        output.backward()

        # Step.
        optimizer.step()

        # Recording the C.K.A. for the batch index.
        if record:
            cka[i] = kernel_calc(self.device, targets, self.features(data).to(device=self.device))

    # Returning the C.K.A. and loss if the option to record was chosen.
    if record:
        return torch.mean(cka).item(), output
```

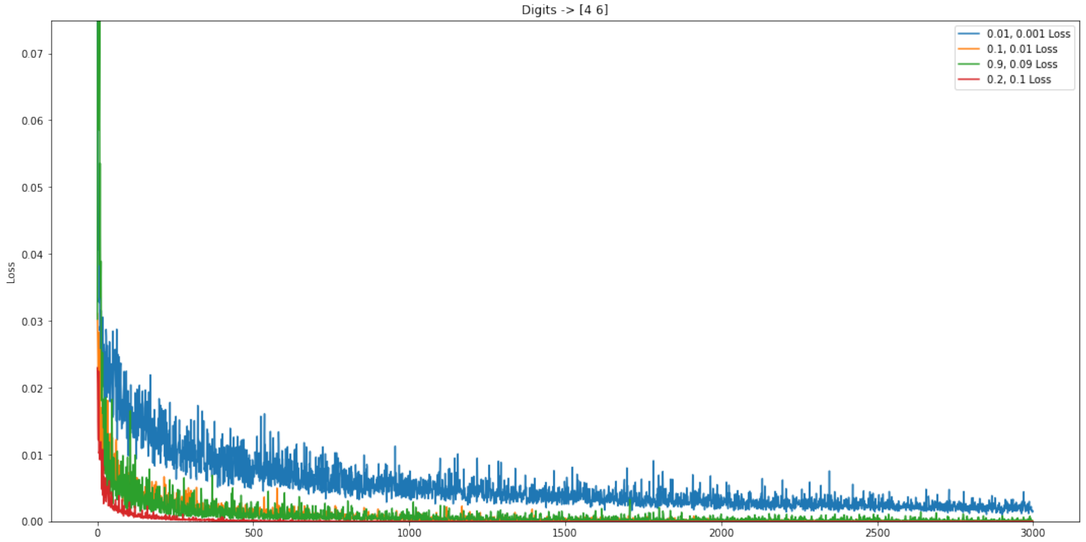


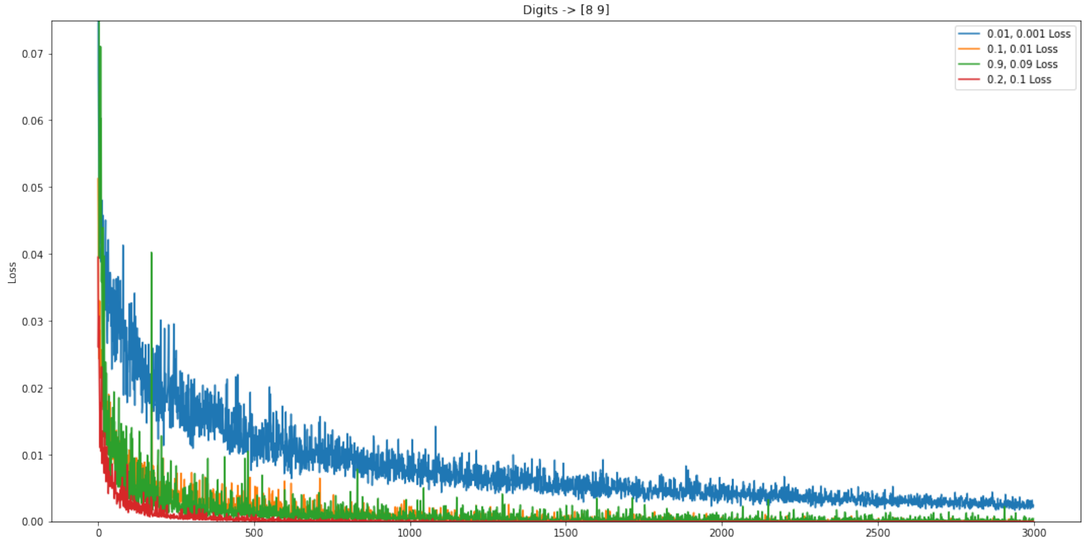
```python
def train_one_epoch(self, loader, loss_function, optimizer, record=True):

    # Array of centered kernal analysis.
    cka = torch.zeros(len(loader))

    for i, (data, targets) in enumerate(loader):
        data = data.reshape(data.shape[0], -1).to(device=self.device)
        targets = targets.to(torch.float32).to(device=self.device)
        classified_targets = classify_targets(targets, self.values)

        # Forwards pass.
        scores = self(data)

        labels = one_hot(classified_targets.long(), len(self.values)).to(torch.float32)
        output = loss_function(scores, labels)

        # Backwards Pass.
        optimizer.zero_grad()
        output.backward()

        # Step.
        optimizer.step()

        # Recording the C.K.A. for the batch index.
        if record:
            cka[i] = kernel_calc(self.device, targets, self.features(data).to(device=self.device))

    # Returning the C.K.A. and loss if the option to record was chosen.
    if record:
        return torch.mean(cka).item(), output
```

Loss For Different Learning Rates







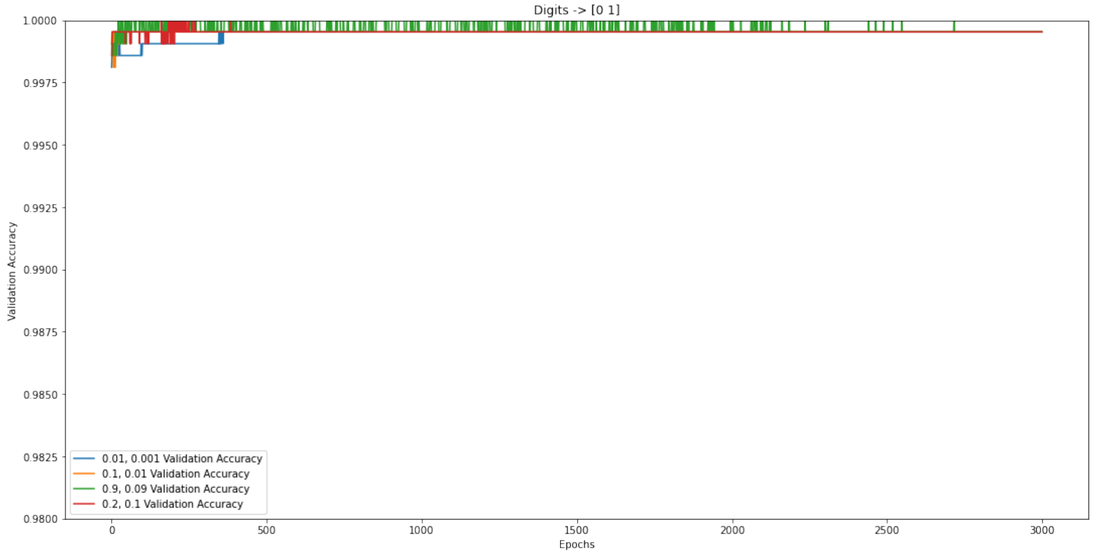


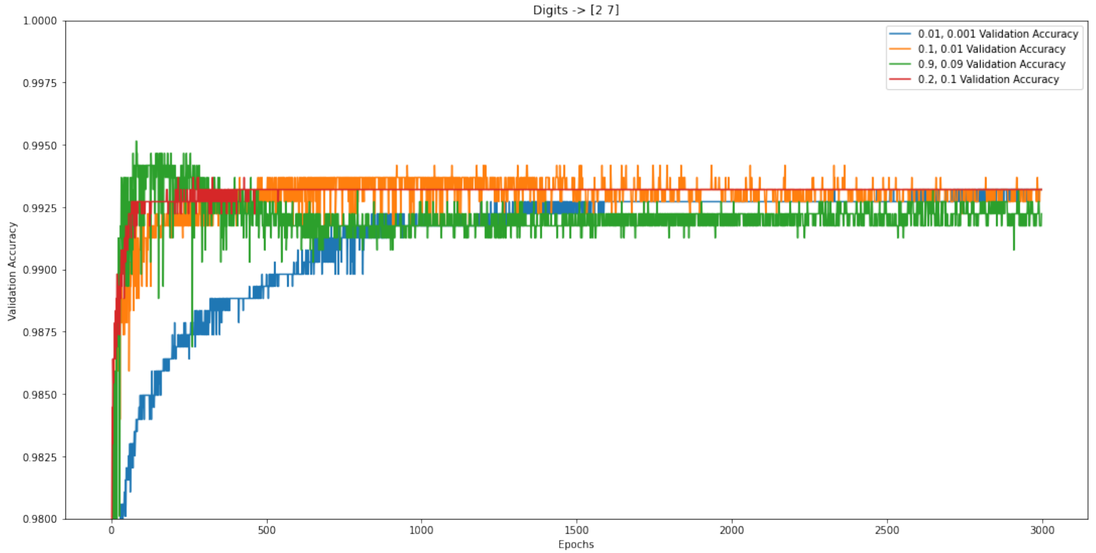
Loss for Different Learning Rates

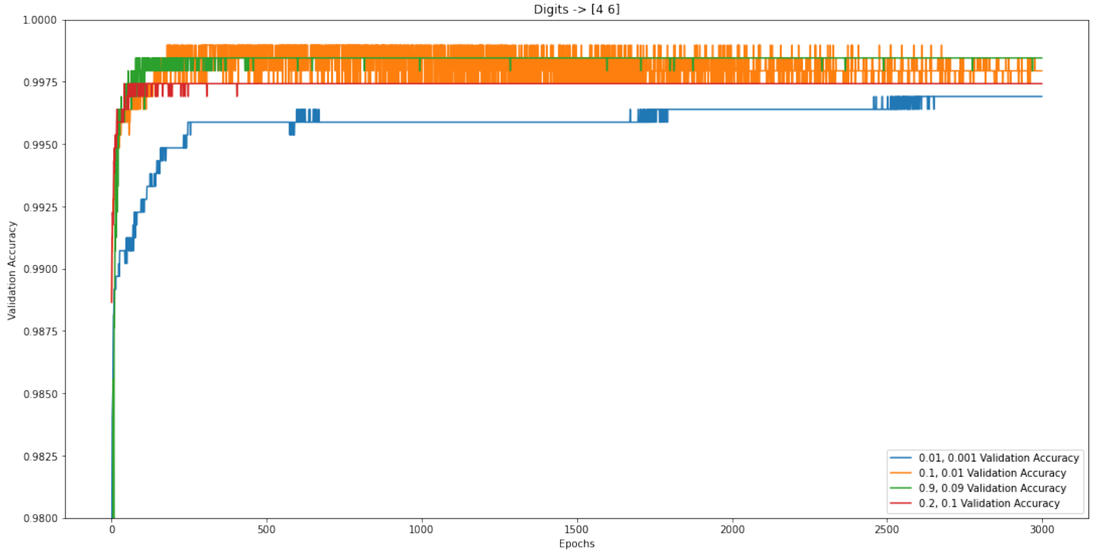
Side Notes:

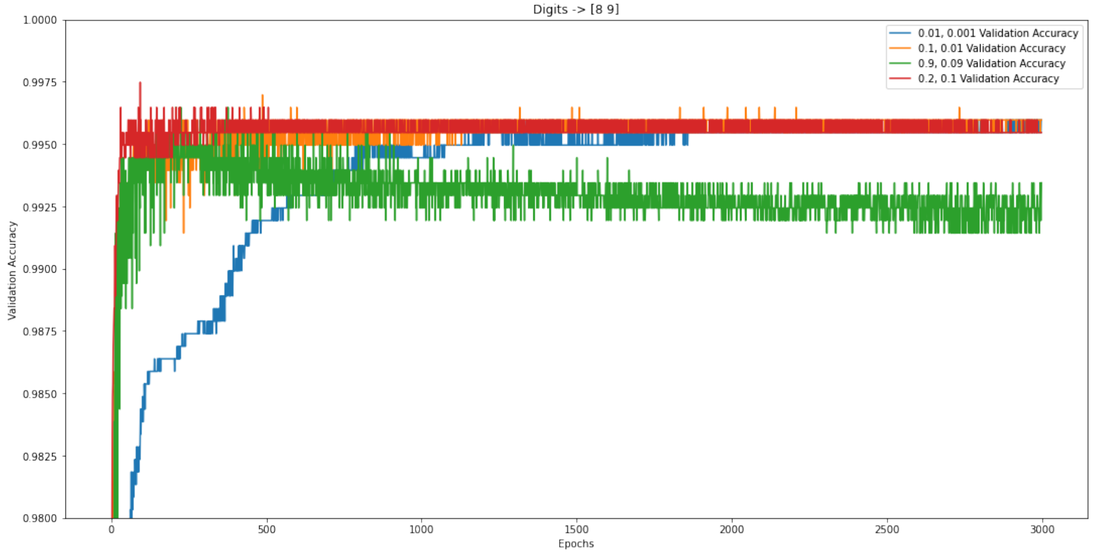
- The smallest tested learning rates (0.01, 0.001) achieve the highest values for loss and have the most noise.
- The learning rates with the largest readout loss (0.2, 0.1) achieved the lowest loss and least noise.
- For tests with smaller readout learning rates, the models don’t quite converge, while the largest learning rate does. Which is consistent w/ past findings.
- Perhaps run test (w/ better-chosen lr) to compare if the noise is at all relative to the size of the learning rate, or maybe if the difference between the readout learning rate and the rest of the model.

Accuracy For Different Learning Rate







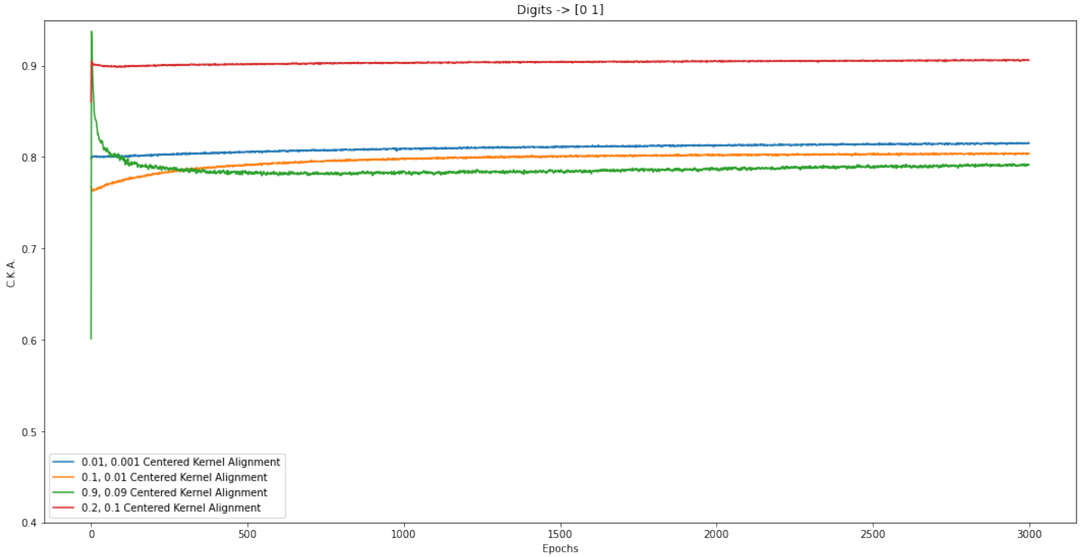


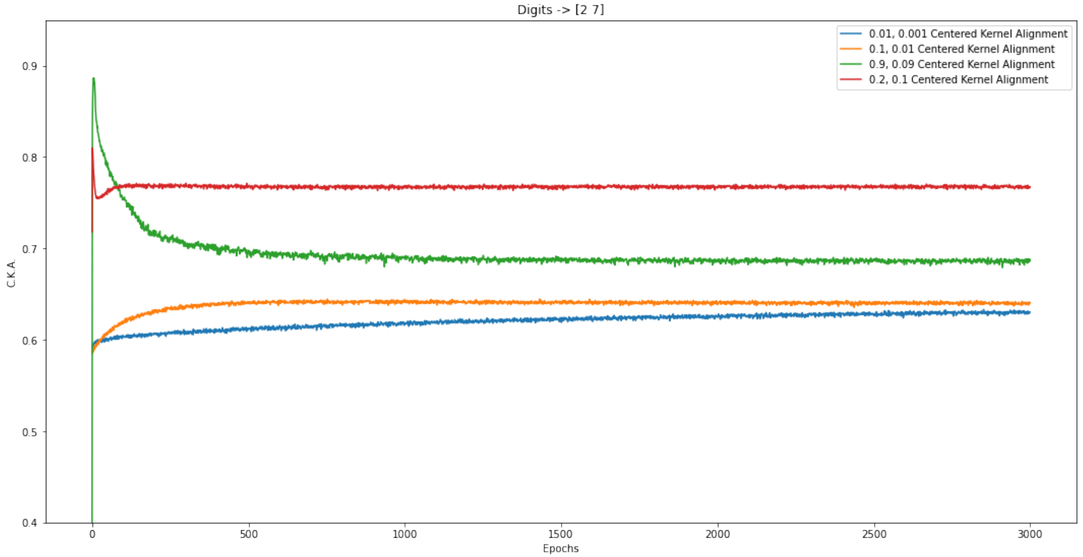
Accuracy For Different Learning Rates

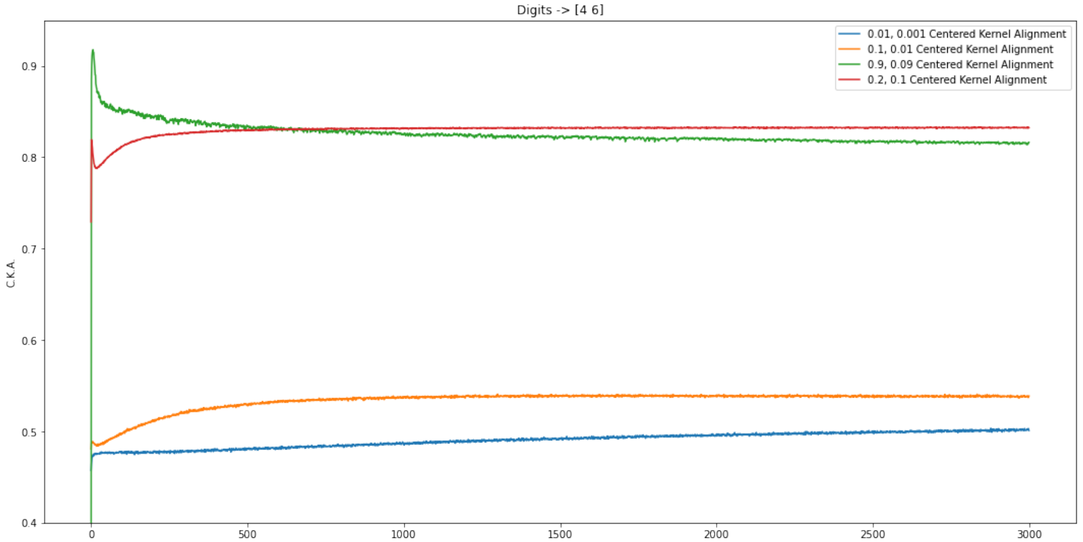
Side Notes:

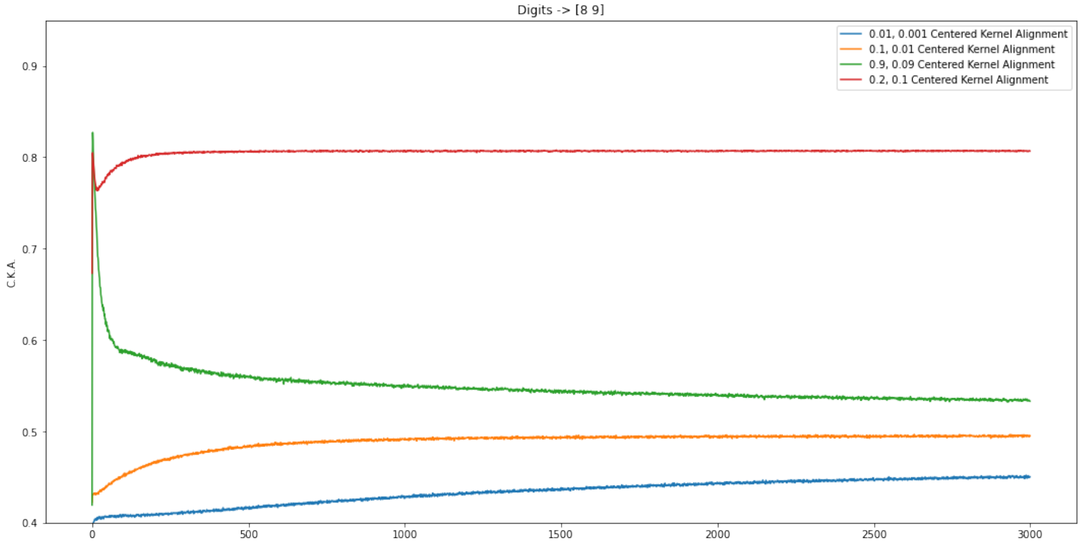
- As before [0 1] reach near-perfect accuracy very quickly.
- [4 6] also reach near-perfect accuracy.
- As before [2 7] has the worst accuracy.
- Notice how the smallest learning rates follow a significantly different path.
- Notice how the largest learning rate reaches its max and then worsens,

CKA For Different Learning Rates









CKA For Different Learning Rates

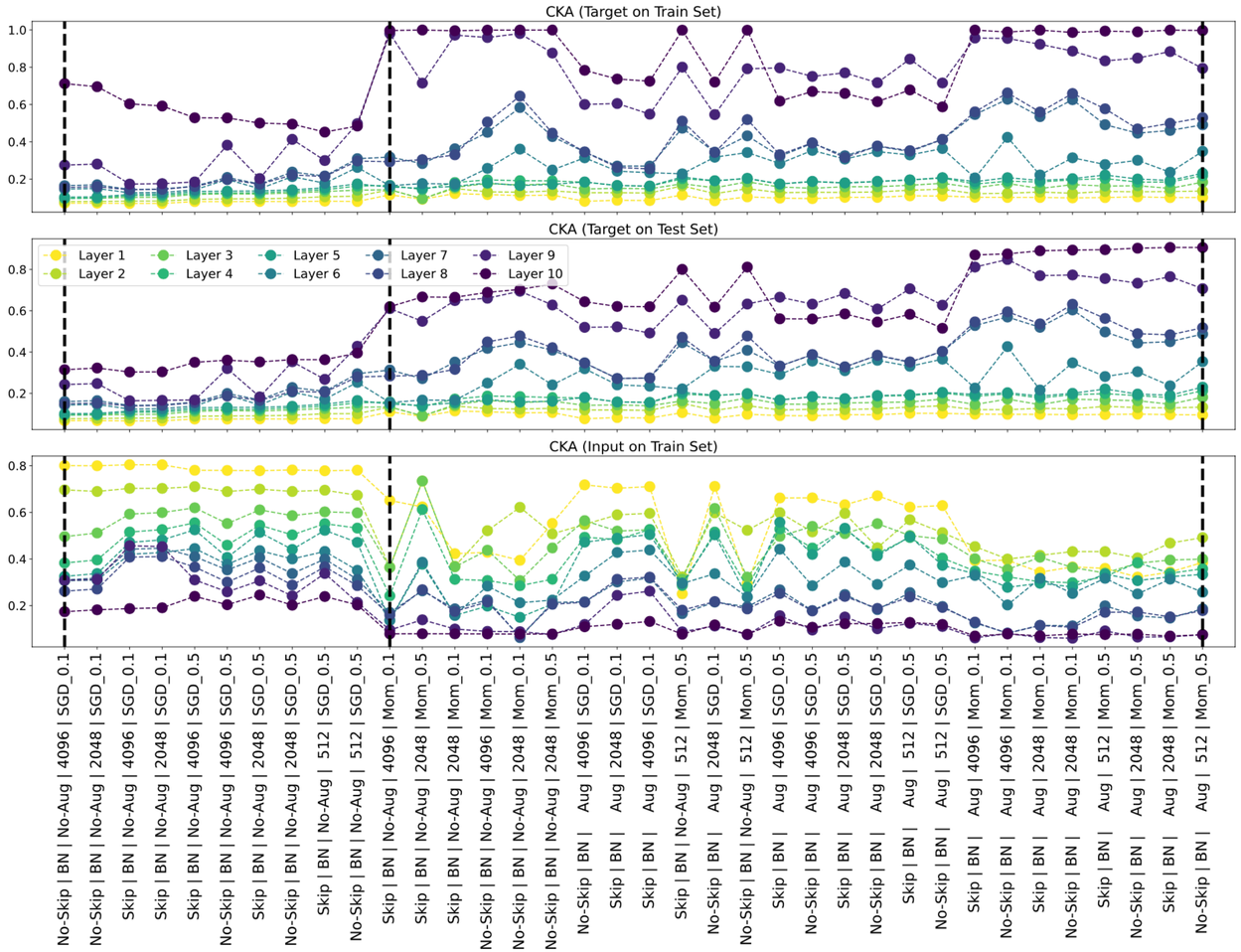
Side Notes:

- Learning rates (0.2 0.1) achieve the highest CKA.
- 8 and 9 achieve significantly lower CKA than [2 7] despite achieving higher accuracy.
- There seems to be a pattern where if the readout rate is small the CKA doesn’t jump to a high CKA before converging below that line.

Cantar and Pehlevan Article

CKA layer(s) Per Model

- CKA is higher for models w/ Momentum SGD.
- For predictive models, the later layers obtain a higher CKA.
- Observed that higher CKA between the layer kernels and the target doesn’t always imply better accuracy.
- Observed that CKA on test is better correlated with generalization.



Cantar and Pehlevan Article

Pearson Coefficients

- Loss has the highest (nontrivial) correlation with acc. then CKA.
- Test acc. has (as expected) a significant correlation with CKA.
- They also found LR has a negative correlation with CKA.
- Momentum is also very highly correlated with CKA.

TABLE I
PEARSON CORRELATION COEFFICIENTS BETWEEN DIFFERENT HYPERPARAMETERS AND MODEL PERFORMANCE METRICS

	Loss	Test Acc.	CKA (Test)	CKA (Train)
Skip	0.097	-0.024	-0.026	-0.024
Aug.	0.335	0.806	0.529	0.162
Batch Size	-0.039	-0.247	-0.181	-0.045
LR	-0.078	0.163	-0.035	-0.263
Momentum	0.099	0.448	0.823	0.936
Loss	1.000	0.384	0.366	0.253
Acc.	0.384	1.000	0.850	0.528
CKA (Test)	0.366	0.850	1.000	0.870
CKA (Train)	0.253	0.528	0.870	1.000
Avg. CKA (Test)	0.279	0.920	0.944	0.747
Avg. CKA (Train)	0.221	0.828	0.930	0.837

Cantar and Pehlevan Article

- Correlations between layer-averaged CKAs for trading and test sets correlate better with the test accuracy.
- Result: A well-generalizing model should not only have god alignments with the penultimate layer, but it should also learn important features of the target through the entire network.