Clustering-Oriented Representation Learning in Neural Networks

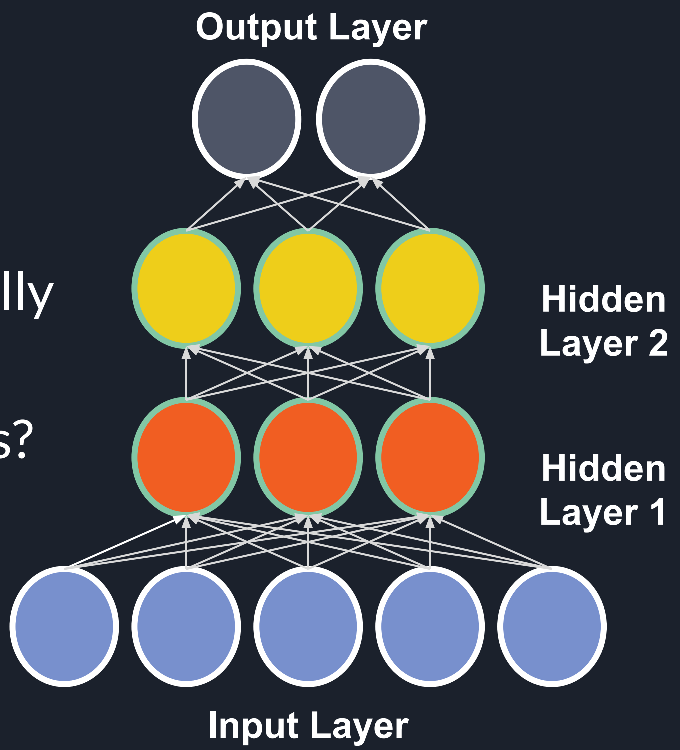
Kian Kenyon-Dean, M.Sc.
Supervised by Jackie Cheung & Doina Precup

RLLab
McGill University

“The purpose of abstraction is not to be vague, but to create a **new semantic level** in which one can be absolutely precise.”
Dijkstra

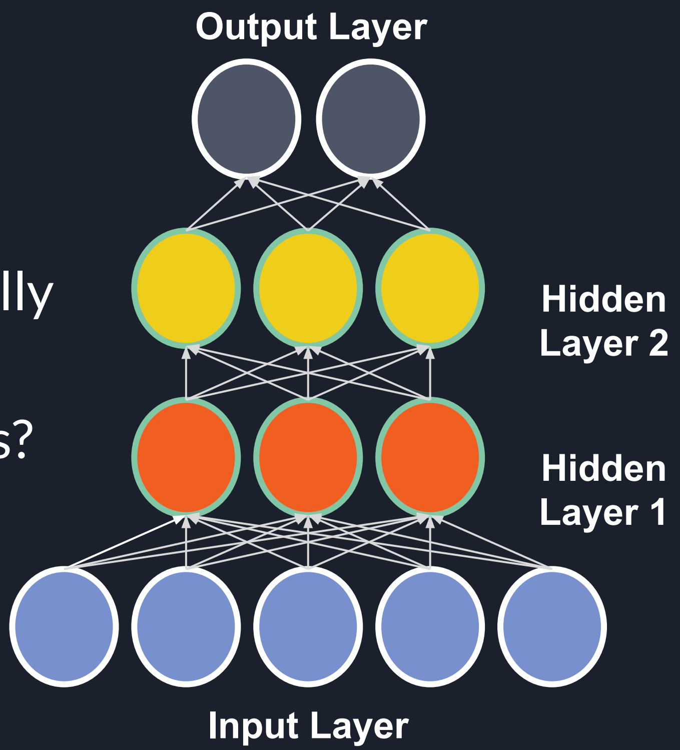
Neural Networks and Latent Representations

- What are the hidden layers in neural networks?
- Are they simply black boxes that magically solve problems, where we have no understanding of their internal workings?



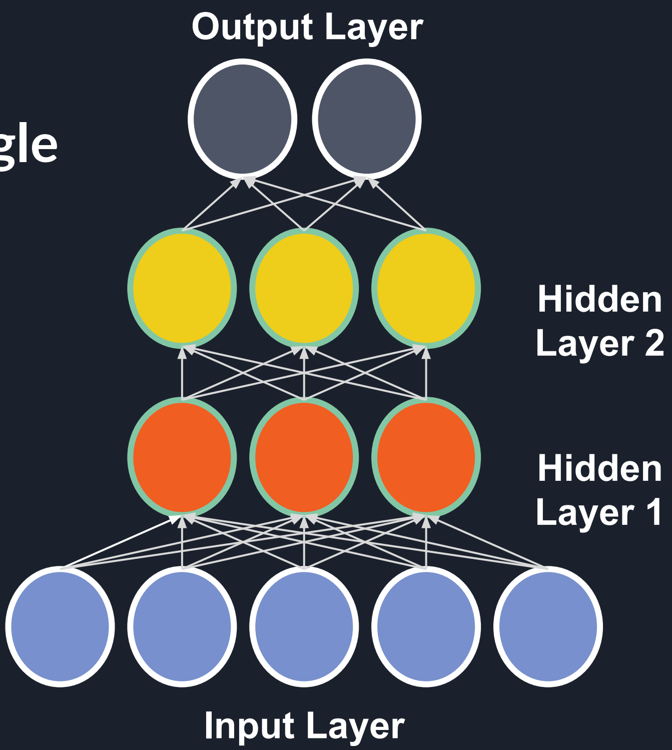
Neural Networks and Latent Representations

- What are the hidden layers in neural networks?
- Are they simply black boxes that magically solve problems, where we have no understanding of their internal workings?
- NO!

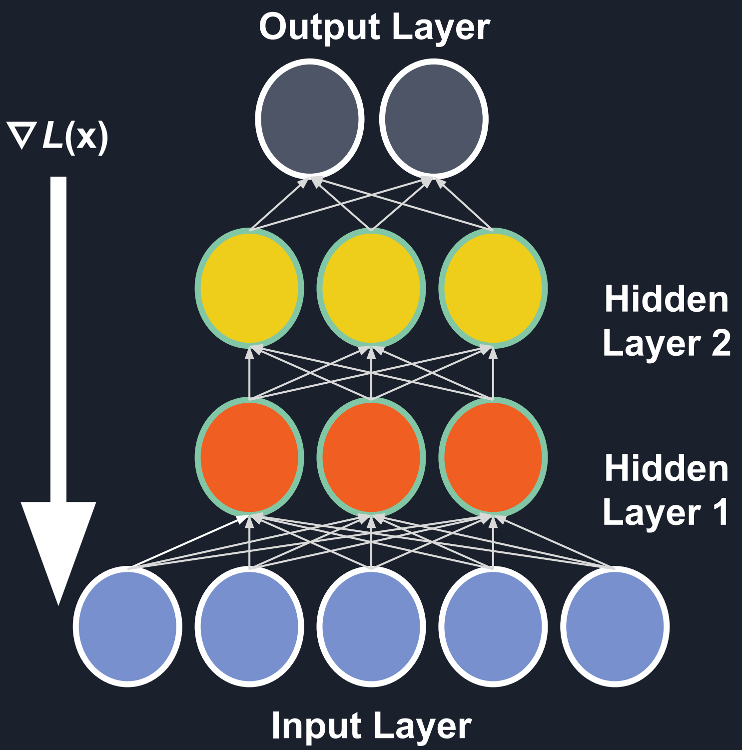


Neural Networks and Latent Representations

- Hidden layers are supposed to **disentangle the factors of variation in the data.**
- They use **nonlinear transformations** to project the data onto a new space with properties imposed by the loss function.
- The parameters of these nonlinear transformations are **learned with backpropagation**.



Neural Networks and Backpropagation



The usefulness of the hidden layers, and the properties they express, depends on the **loss function used to train the network**, the gradient of which is backpropagated through the network.

Categorical Cross Entropy Loss

- Categorical Cross Entropy is the standard loss function of a network designed for classification.
- Whereas logistic regression attempts to linearly separate the data in the **original feature space**, CCE in MLPs directly imposes the quality that the data should be linearly separable in a **new latent space**.

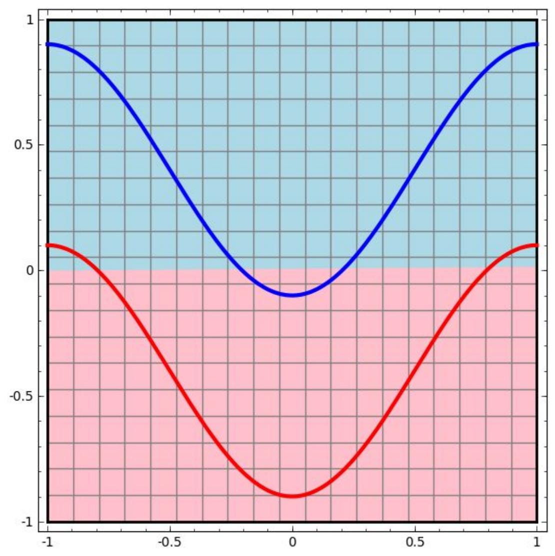
Linear Separability and Non-linear Transformations

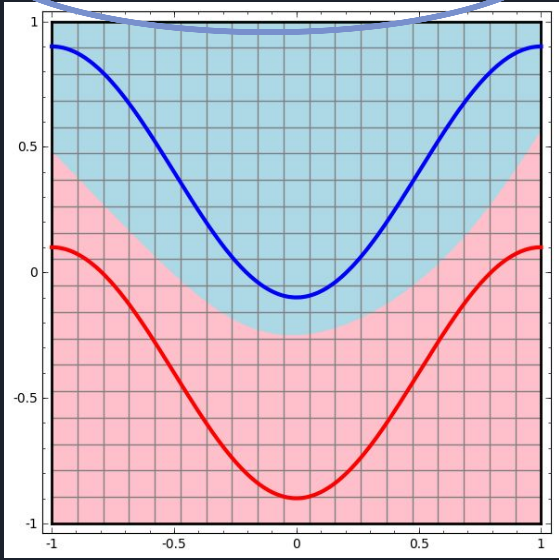
Same things, different views!

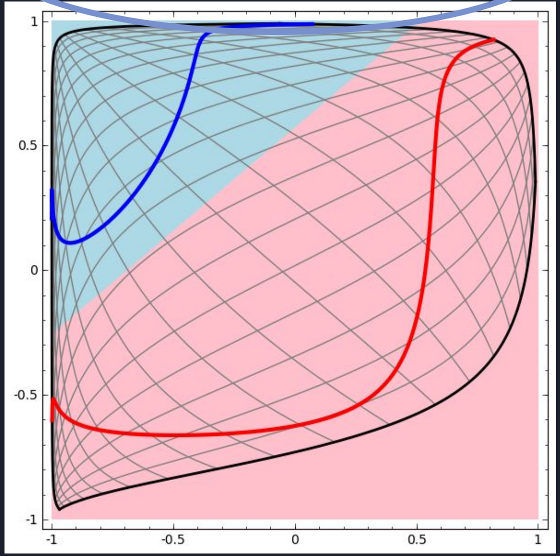
Linear Separation in Feature space, made by logistic regression. Not a perfect separation, and cannot be.

Non-Linear Separation in Feature space, made by neural net. But, this is actually a reflection of -->

The linear separation in the **learned latent space, the learned non-linear transformation of data.**







See: http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Categorical Cross Entropy Loss

- Linear separability, in the latent space, is the fundamental geometric property expressed by CCE.
- My research questions:
    - *Can we make the latent space **more "interpretable"** with a more geometrically motivated loss function?*
    - *Do we even **need an output layer**? Do we need linear separation to have a nice model?*

Clusterability for Latent Representations

According to (Bengio et al., 2013), a desirable quality for our latent representations would be if they were **naturally clusterable**, that is that "different values of categorical variables such as object classes are *associated with separate manifolds*."

Clustering like it's 1957 (the year K-Means was theorized...)

- We would like impose "clusterability" with a new loss function, since CCE doesn't do so (it only imposes linear separability).
- Let's be like K-Means and use **centroids**!
- Not just any centroids, but **latent categorical centroids**...

Clustering like it's 1957 (the year K-Means was theorized...)

Latent Categorical Centroids:

- Each class *c* gets a latent categorical centroid, E_c :
  - $E_c = (1/|C|) \sum h(x_i)$
  - *For each sample x_i in C; e.g., belonging to class C.*
- In other words, E_c is the mean latent representation, *h(x)*, over all samples, x_i, for class *c*.

Clustering like it's 2017... With Neural Networks!

- Now, if we have K classes in our dataset, we can construct K centroids, $E_1, \ldots E_K$, from our training set.
- Now, **how can we use these centroids in the loss function of a neural network, in order to impose the quality of "natural clustering"?**

Centroid-Clustering Loss in Neural Networks

If the following two criteria hold, then the data will be easily clusterable (in the latent space):

- *Centroid-Attraction*: samples belonging to class C should be **close to the centroid** of class C.
- *Centroid-Repulsion*: samples NOT belonging to class C should be **far away from the centroid** of class C.

Centroid-Clustering Loss in Neural Networks

We can express these criteria with explicit loss functions that work over the latent space:

- *Centroid-Attraction*: **minimize** the distance between samples and the centroids of their classes.
- *Centroid-Repulsion*: **maximize** the distance between samples and the centroids of other classes.

Centroid-Clustering Loss: Measuring Distance

- How should we measure how “close” something is to something else?

Centroid-Clustering Loss: Measuring Distance

- How should we measure how “close” something is to something else?
- If we are trying to maximize a distance, we probably need to use a distance function (measure of similarity) that does not diverge to infinity.

Centroid-Clustering Loss: Measuring Distance

- How should we measure how “close” something is to something else?
- If we are trying to maximize a distance, we probably need to use a distance function (measure of similarity) that does not diverge to infinity.
- So, no euclidean distance!

Centroid-Clustering Loss: Measuring Distance *with Cosine Distance*

- Let's use **cosine distance**!
    - Half of 1 minus the cosine similarity between the vectors (cosine similarity is in [-1, 1]).

$$cosd(u, v) = \frac{1}{2}\left(1 - \frac{u \cdot v}{||u||_2||v||_2}\right)$$

Centroid-Clustering Loss: Measuring Distance *with Cosine Distance*

- Let's use **cosine distance**!
  - If cosd(u,v)=0 then they are oriented in same direction.
  - If cosd(u,v)=1 then they are in opposite directions.

$$cosd(u, v) = \frac{1}{2}\left(1 - \frac{u \cdot v}{||u||_2||v||_2}\right)$$

Centroid-Clustering Loss: Measuring Distance *with Cosine Distance*

- Let's use **cosine distance**!
  - Computes the "angle between two vectors"
  - Magnitude invariant
  - Constrained between 0 and 1

Centroid-Clustering Loss: Measuring Distance *with Cosine Distance*

- Let's use **cosine distance**!
  - Computes the "angle between two vectors"
  - Magnitude invariant
  - Constrained between 0 and 1
  - Best understanding: **measures the squared euclidean distance between two vectors when projected onto the unit hypersphere.**

Centroid-Clustering Loss: Measuring Distance *with Cosine Distance*

- "But cosine-distance is less expressive since it is magnitude invariant!" *one might say...*

Centroid-Clustering Loss: Measuring Distance *with Cosine Distance*

- "But cosine-distance is less expressive since it is magnitude invariant!" *one might say…*
- It is well known that, in high dimensional spaces, euclidean distance is not meaningful and actually problematic due to hypersensitivity to small perturbations.
- So, this property of cosine-distance may actually be desirable, may make it *more expressive*! (Charu et al., 2001)
- But I'm open to suggestions for other distance metrics! Particularly ones that can be expressed in pure matrix form.

Centroid-Clustering Loss: Measuring Distance *with Cosine Distance*

- Note that, whatever distance function we choose, the model will learn non-linear transformations to manifest its properties as much as possible.
- So, perhaps, the distance function is not super decisive since the network will adapt to it regardless.

Centroid-Clustering Loss in Neural Networks

Centroid-Attraction: **minimize** the distance between samples and the centroids of their classes.

$$\mathbf{L}_{Att-SC} = \frac{\lambda_1}{n} \sum_{k=1}^{K} \sum_{i \in C_k} d(\boldsymbol{\mu}_k, \mathbf{h}_i)$$

Centroid-Repulsion: **maximize** the distance between samples and the centroids of other classes.

$$\mathbf{L}_{Rep-SC} = -\frac{\lambda_2}{n(K-1)} \sum_{k=1}^{K} \sum_{j \notin C_k} d(\boldsymbol{\mu}_k, \mathbf{h}_j)$$

Centroid-based Inference in Neural Networks

Note, if the *Centroid-Attraction* and *Centroid-Repulsion* criteria hold, and if the model has properly generalized, *then we **do not need an output layer** for our model.*

Centroid-based Inference in Neural Networks

Note, if the *Centroid-Attraction* and *Centroid-Repulsion* criteria hold, and if the model has properly generalized, *then we* ***do not need an output layer*** *for our model.*

Instead, we predict that the class of a new sample, *x*, is the class of the training-set centroid to which it is closest. E.g., `class(x) = argmin_c d(h(x), E_c)`

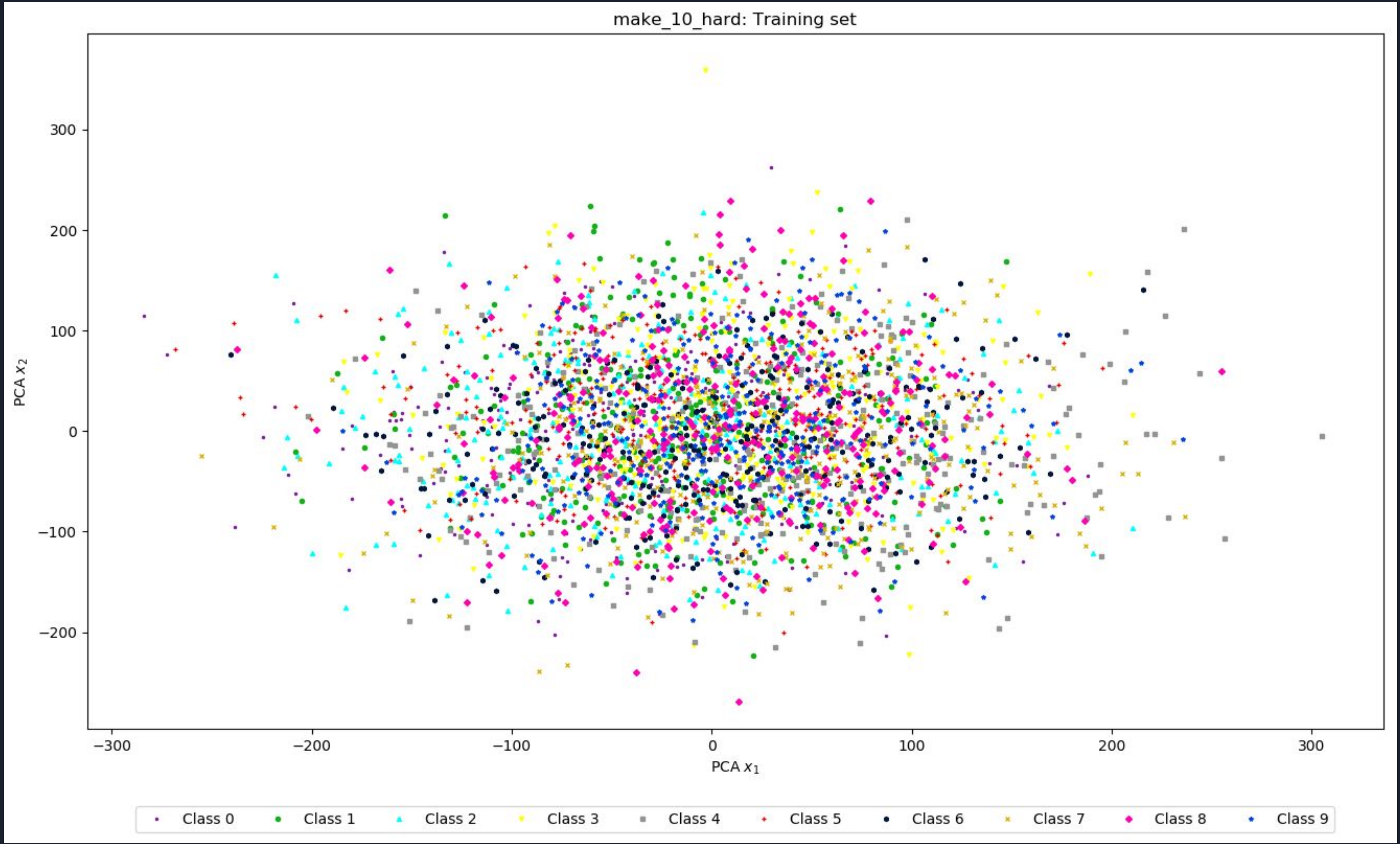
Summary of our Clustering-Oriented Representation Learning Network

- **No output layer**, works at the level of representation
- Dynamically maintains representations of the classes, the latent categorical centroids
- Uses clustering-oriented loss to optimize the network
    - **Attract** samples to their centroids
    - **Repulse** samples from other centroids
- Uses the centroids to perform inference

Experimental Design
(e.g., But does it work?)

We experiment with synthetic data to isolate model design from the specificities of working with real datasets.

- 3,400 training samples, 600 validation, 1000 test
- 1000 features per sample
- 10 classes


make_10_hard: Training set
PCA x_2
PCA x_1
300
200
100
0
−100
−200
−300
−200
−100
0
100
200
300
Class 0
Class 1
Class 2
Class 3
Class 4
Class 5
Class 6
Class 7
Class 8
Class 9

Dataset

Very hard! Very much *not* linearly separable!

Lots of noise!

Logistic regression only gets 20% accuracy!

SVM with RBF kernel (with highly tuned C) only gets 63% accuracy!

Experiments

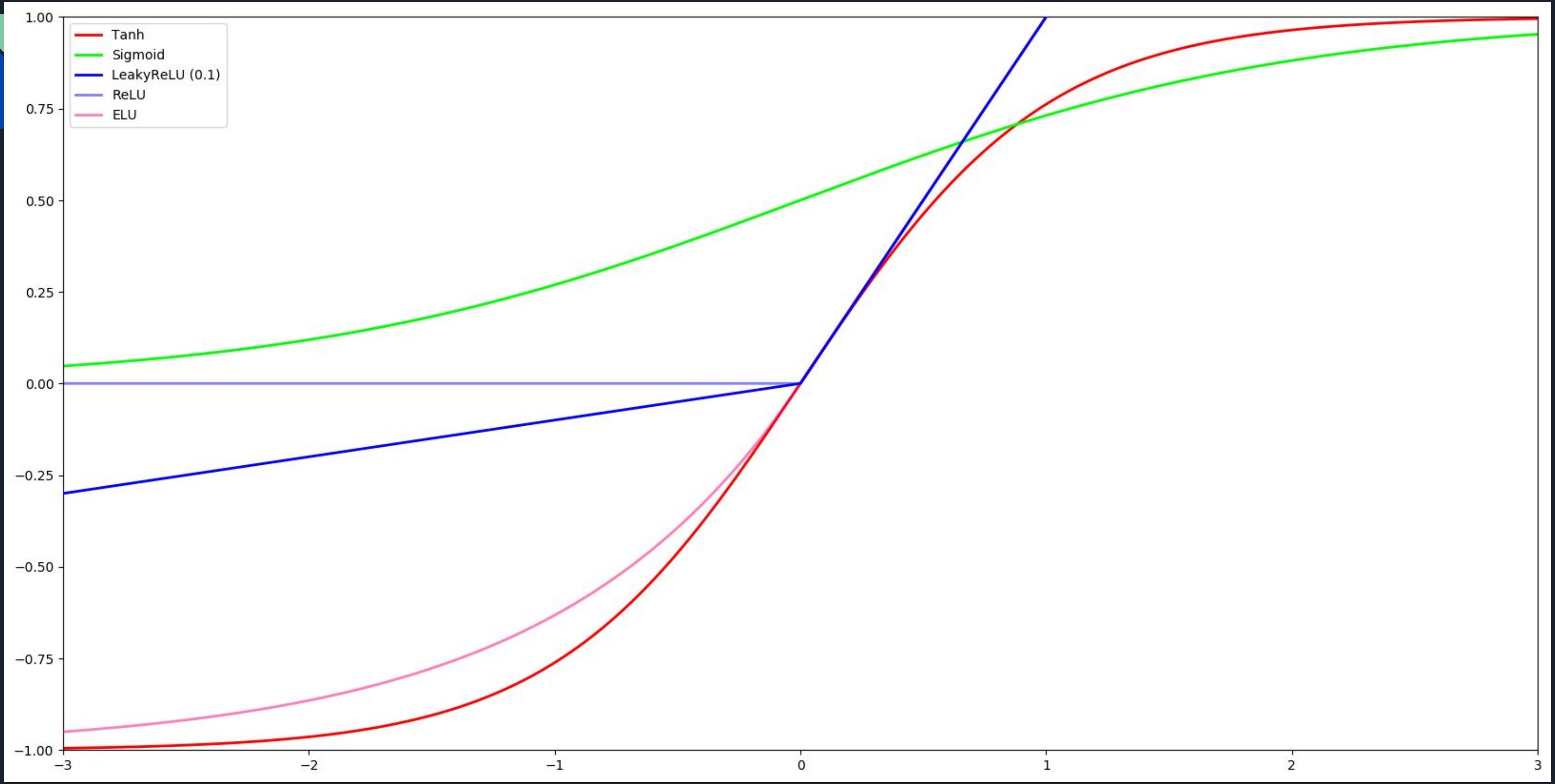
If our Centroid-Clustering loss is good, then the model trained with it should:

- Be better than other models
- Really should be better than a CCE feed-forward neural network

Parameter Tuning

Tested several thousand different neural network architecture variants for our model, including:

- Activation functions (Tanh, ReLU, LeakyReLU, PreLU…)
- Batch size (100, 340, 1700, 3400)
- Learning rate (many)
- Number and dimensionality of hidden layers (very many)



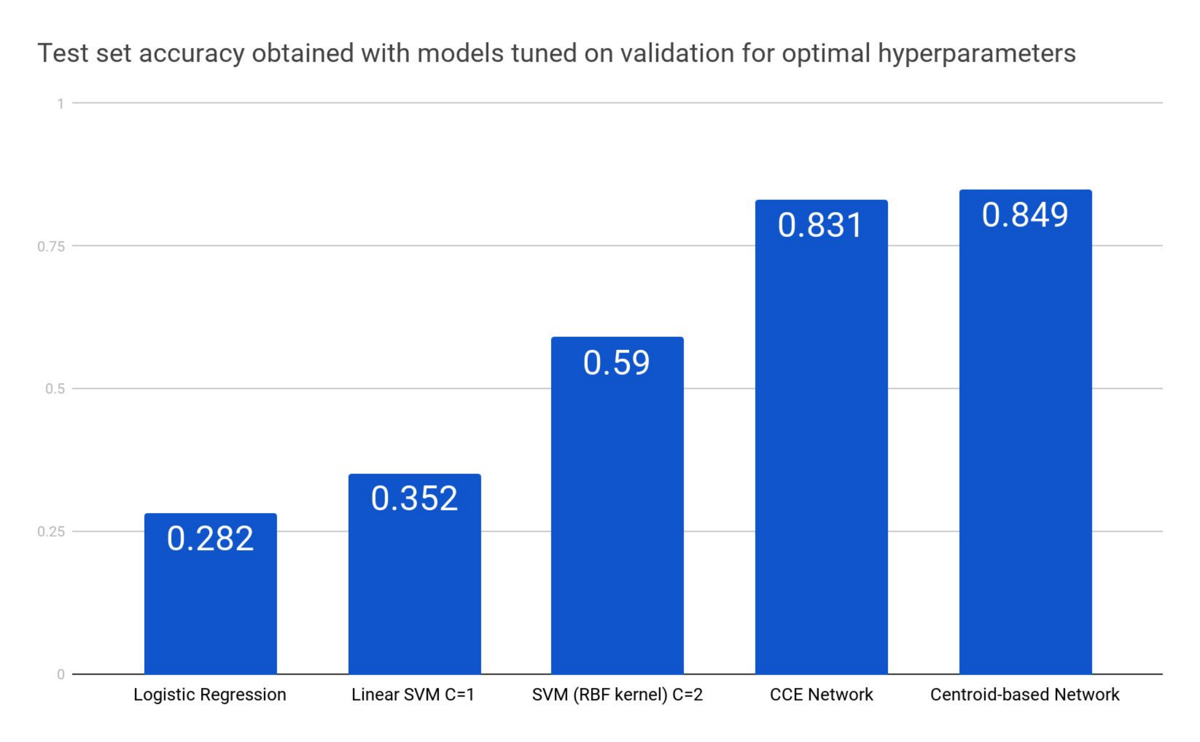
Tanh
Sigmoid
LeakyReLU (0.1)
ReLU
ELU
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−3
−2
−1
0
1
2
3

Parameter Tuning - Results

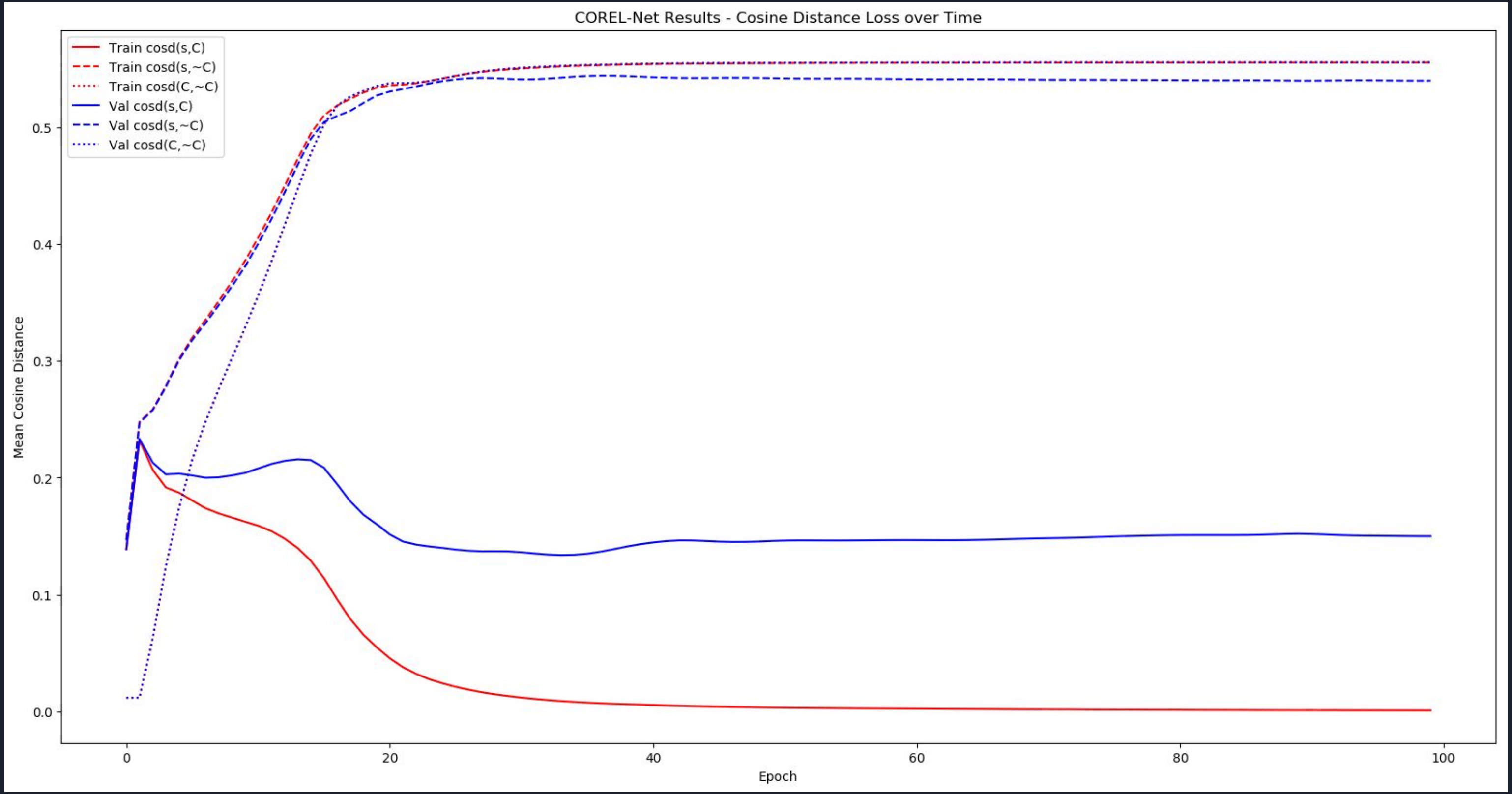
For our Centroid-based network, we found the following parameter settings were very important for obtaining optimal validation set accuracy:

- Using LeakyReLU, not ReLU (and definitely not Tanh!)
- Testing different variants of bottleneck networks, 3 layers worked quite well
  - E.g., Layer sizes [1000 -> 2048 -> 128 -> 4096]

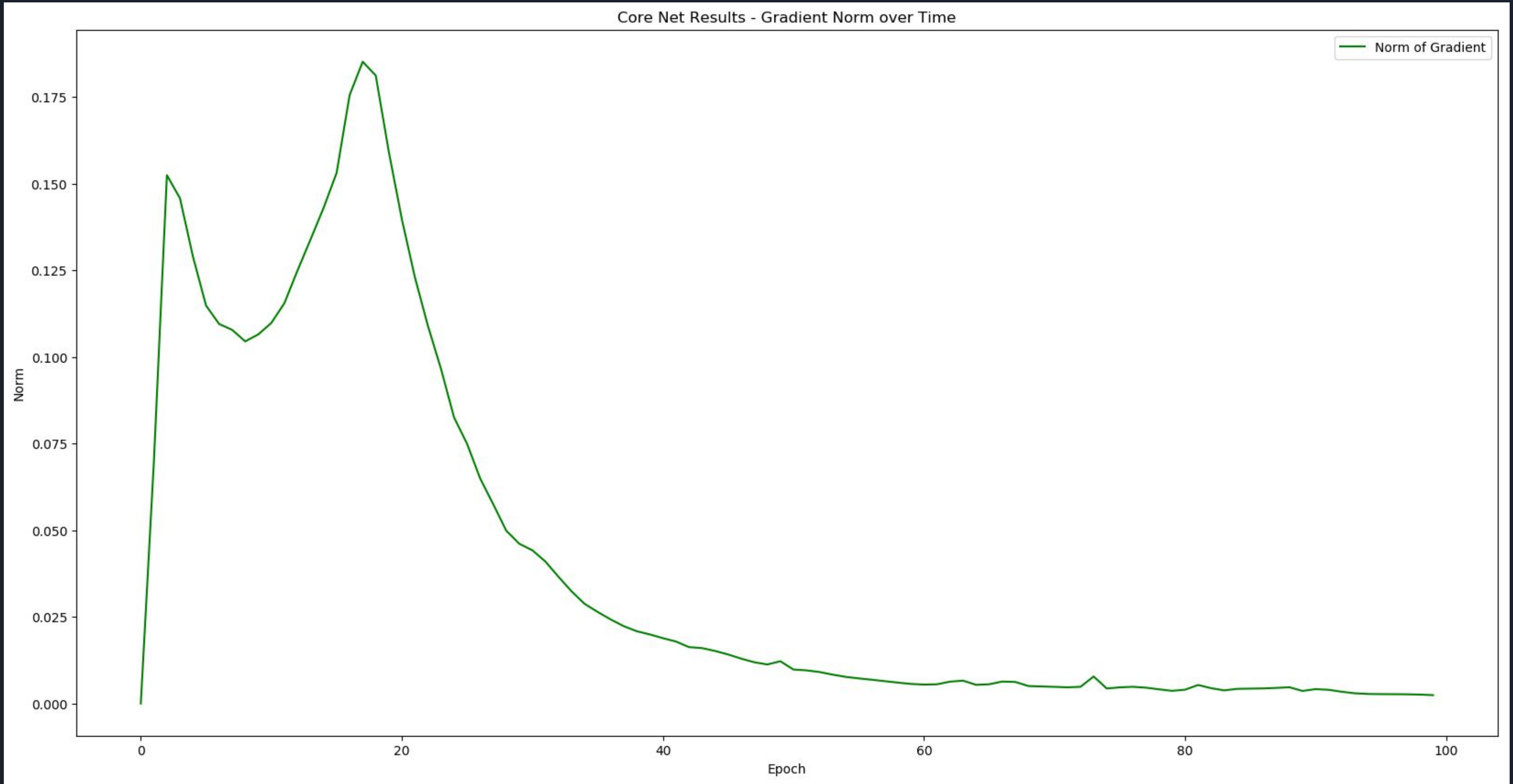
Final Test Set Results



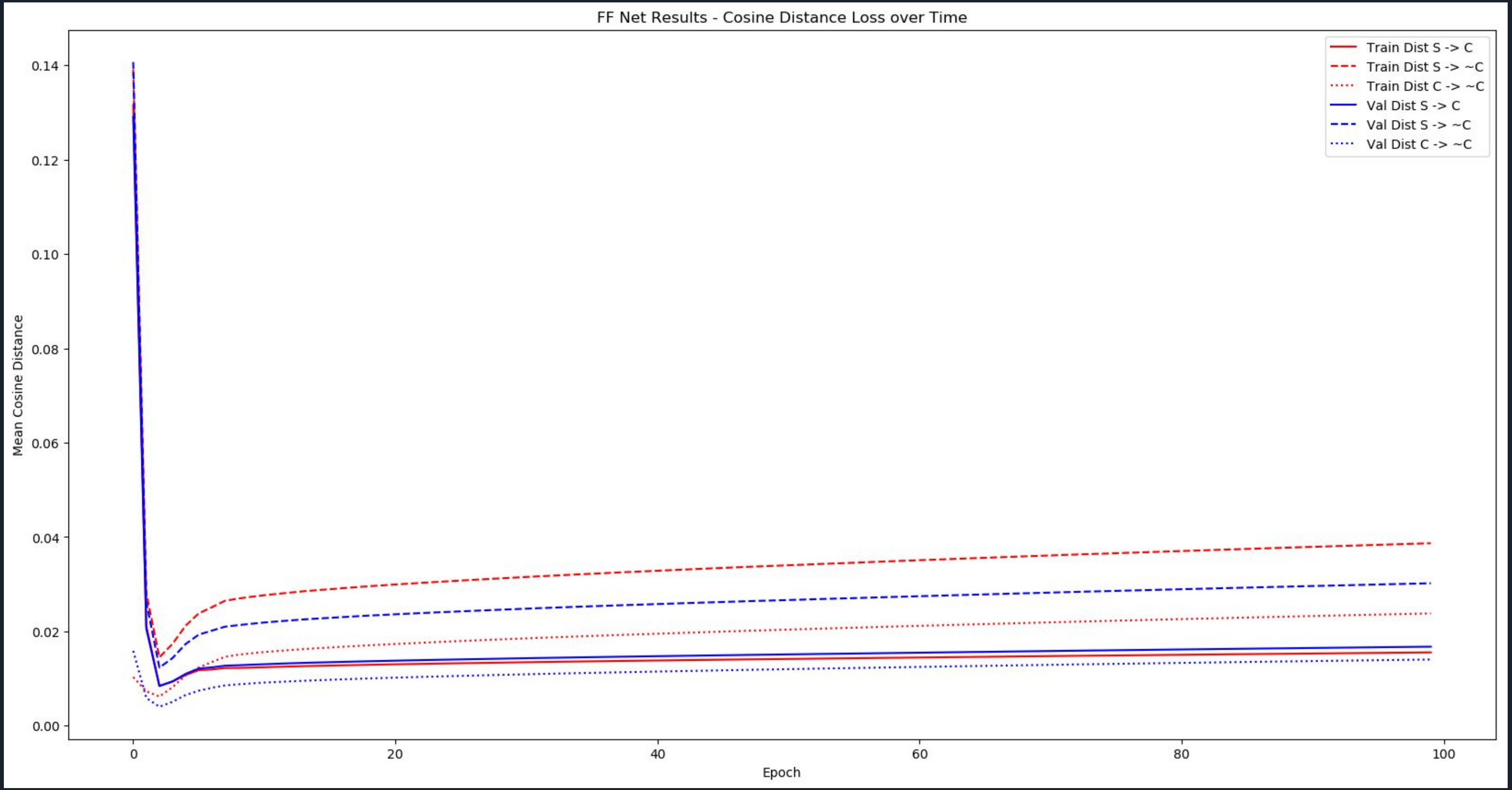
Learning to Cluster - Centroid Net



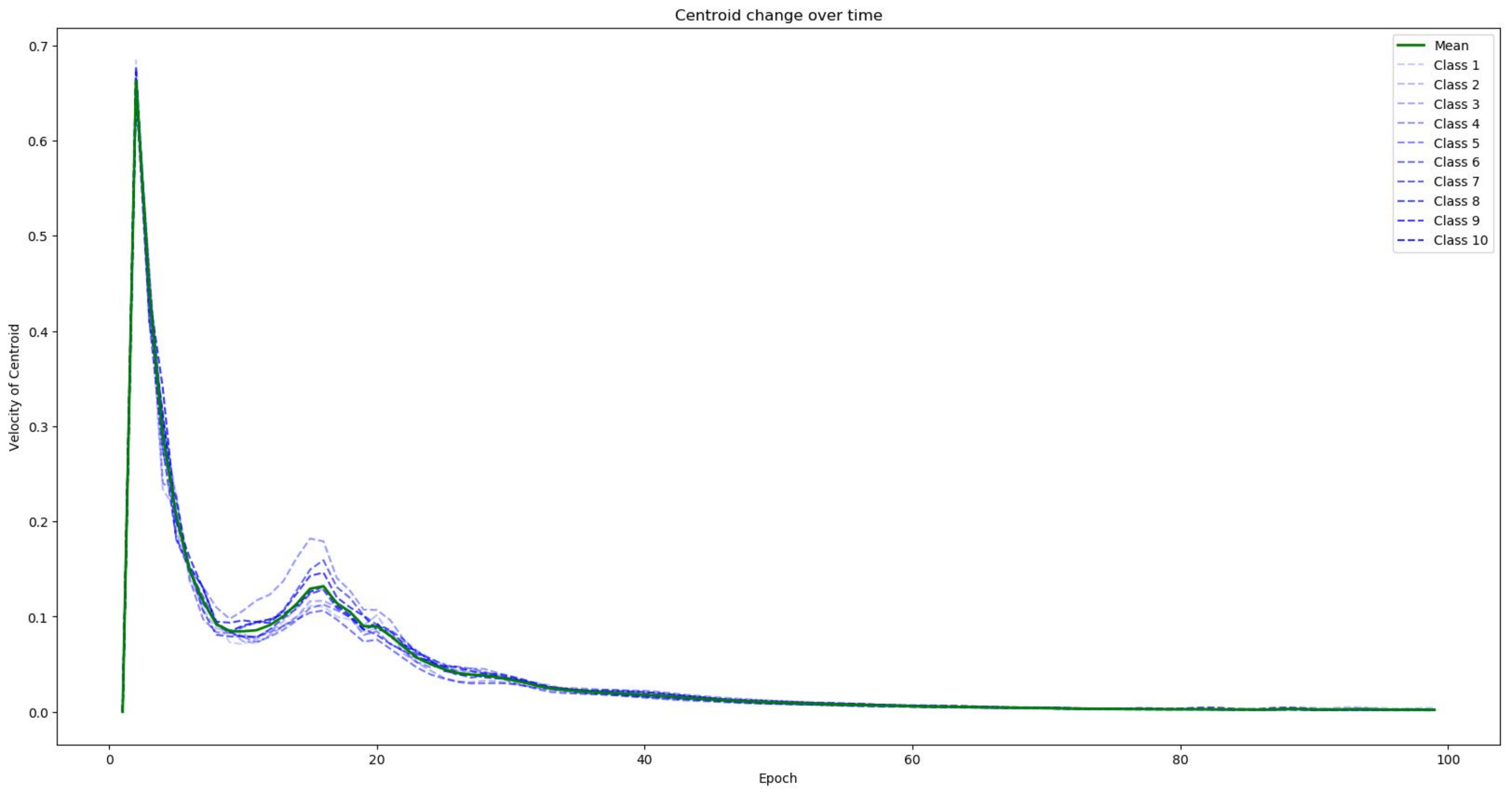
Norm of Gradient Over Time - Centroid Net



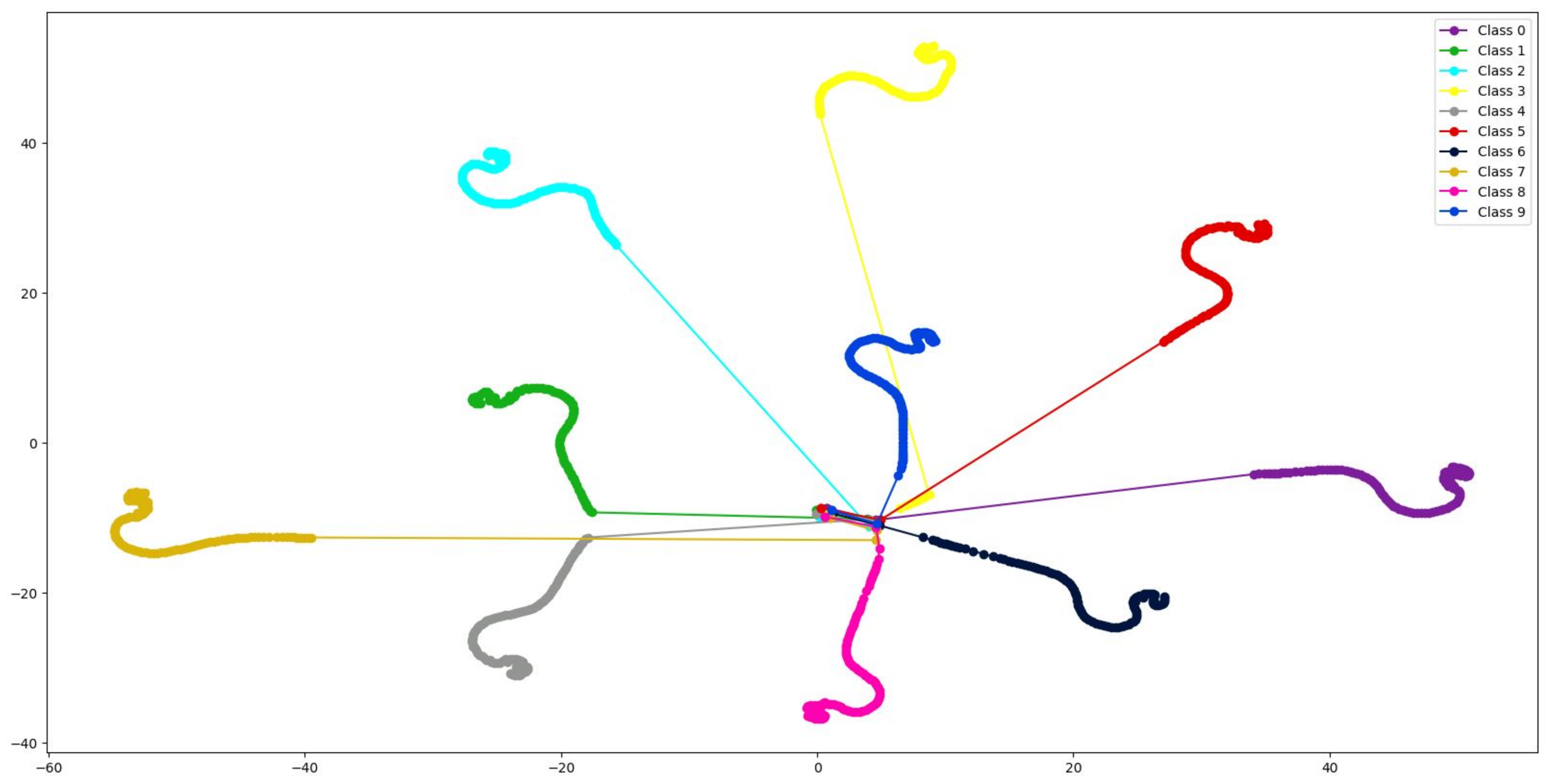
Learning to Cluster - CCE Net



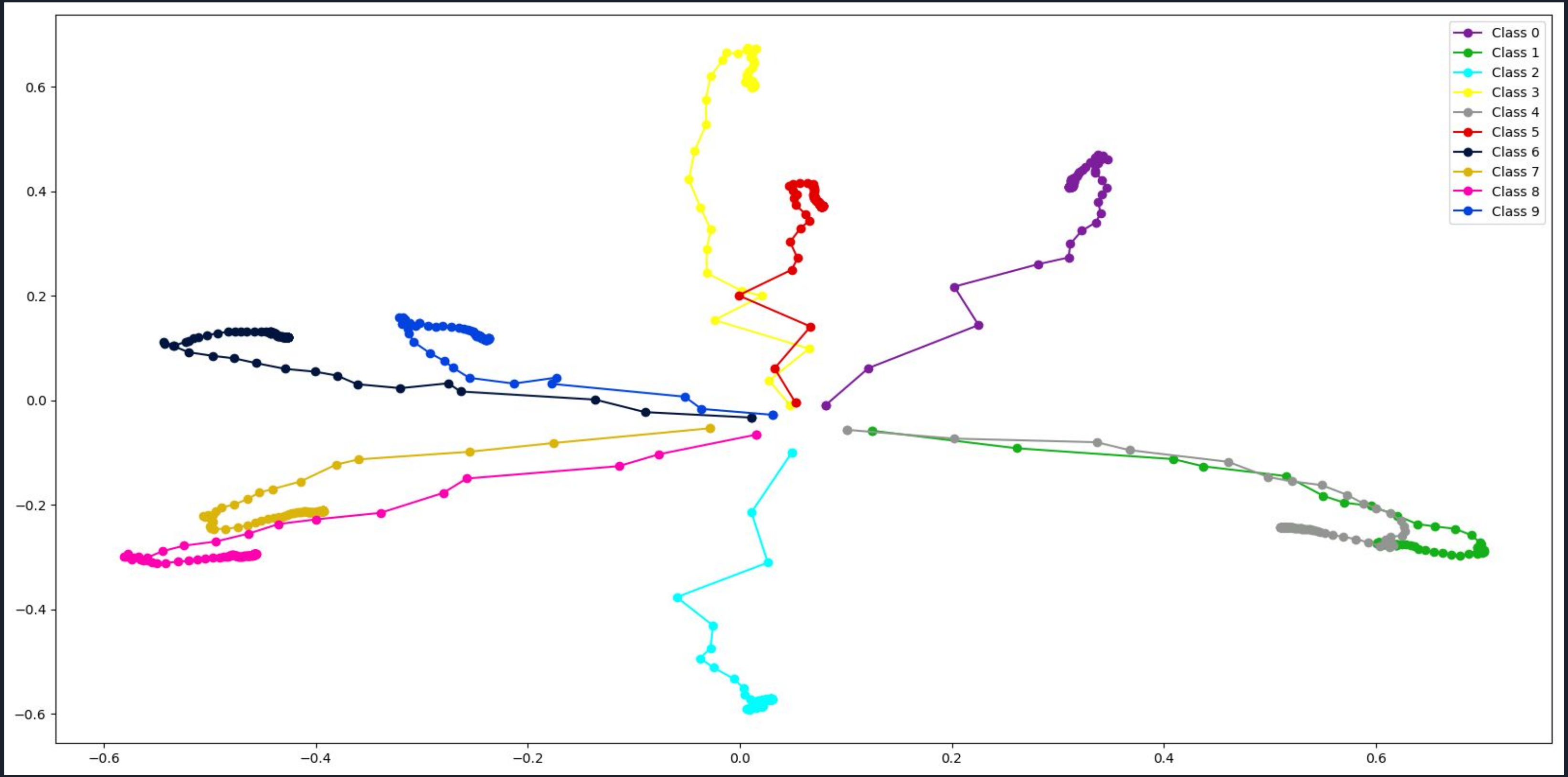
Centroid Changes - Centroid Net



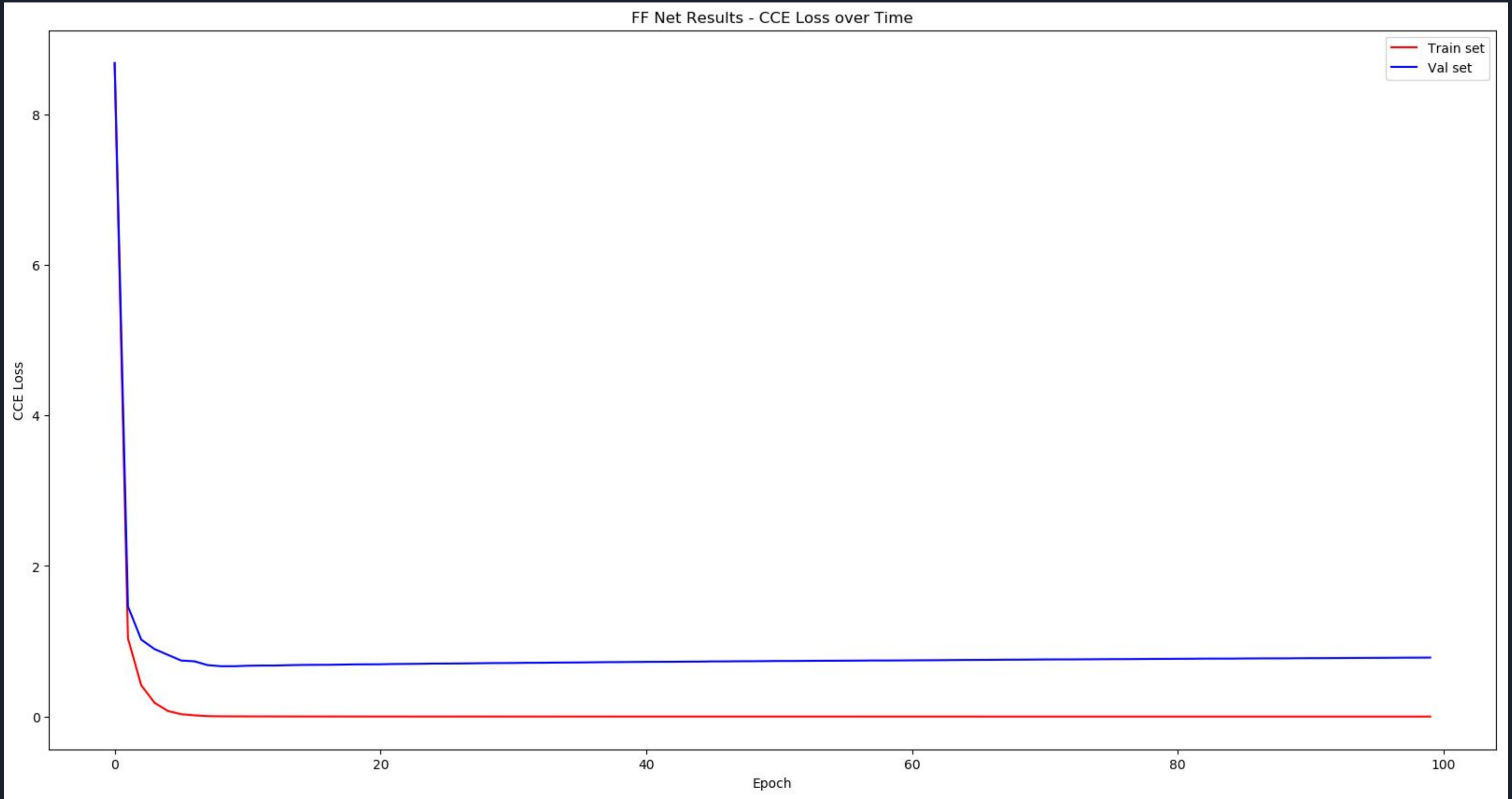
Centroid Changes during Training



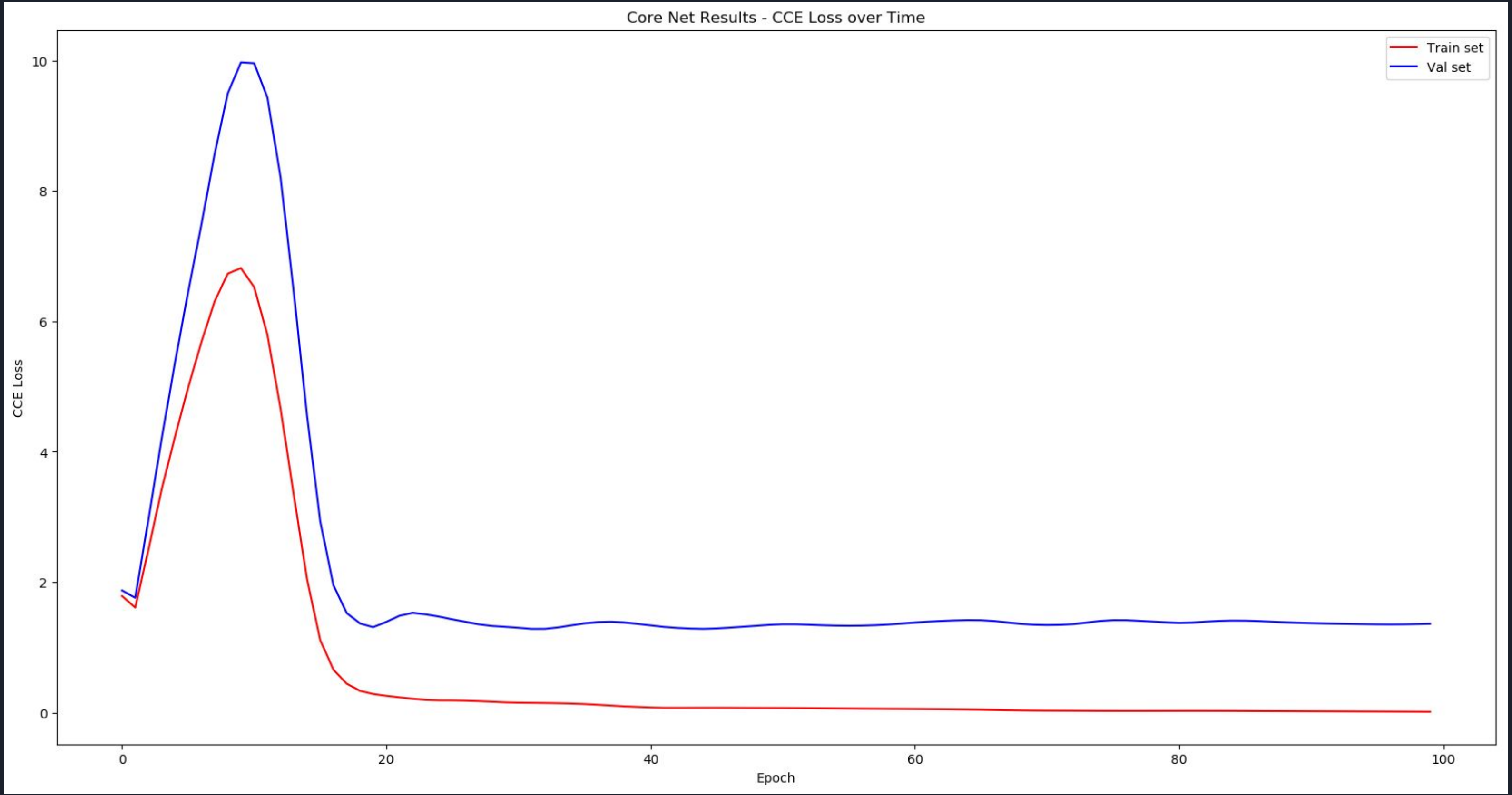
Centroid Changes - Centroid Net



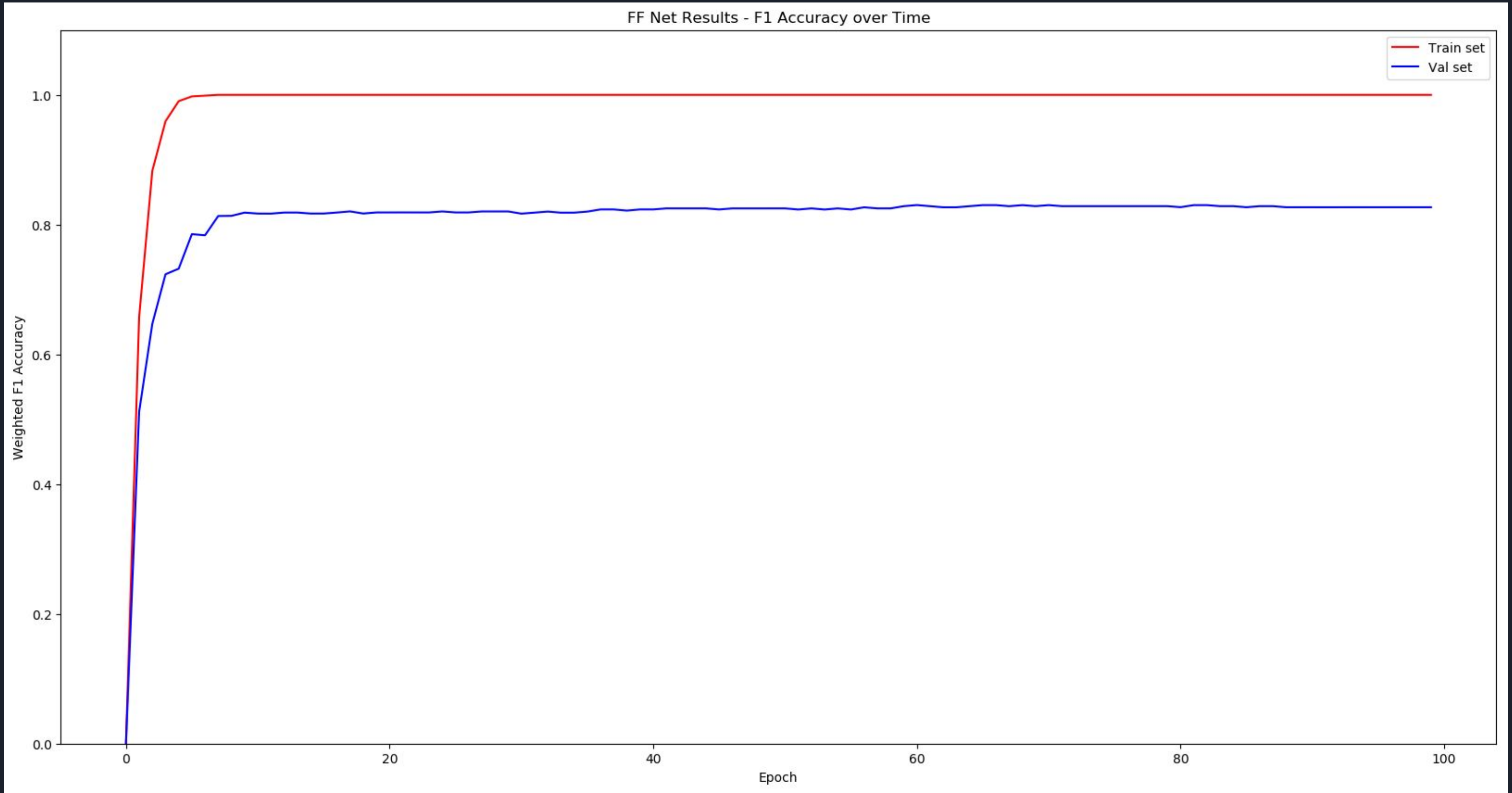
Learning to Predict - CCE Network



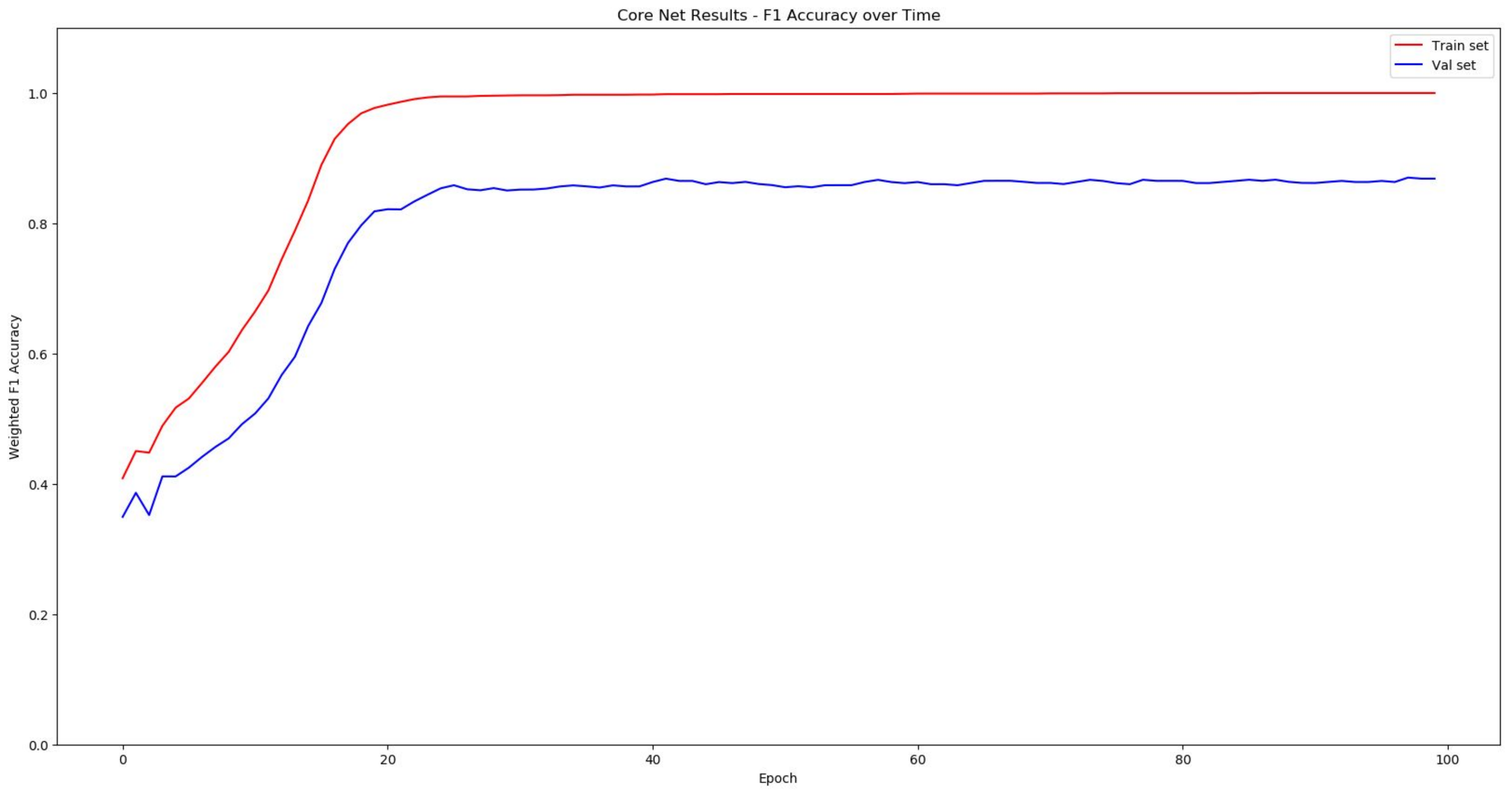
Learning to Predict - Centroid Network



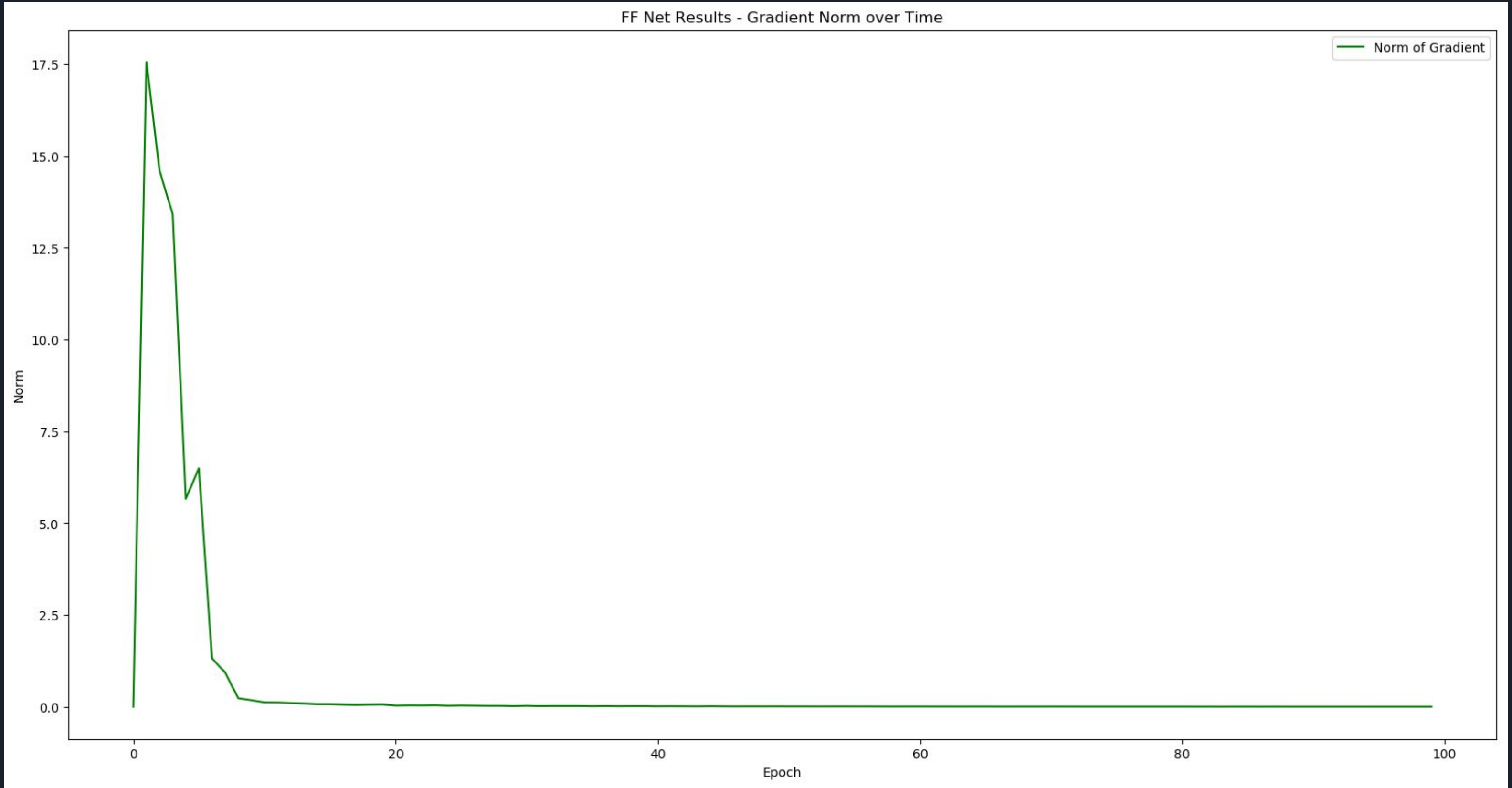
Accuracy Over Time - CCE Network



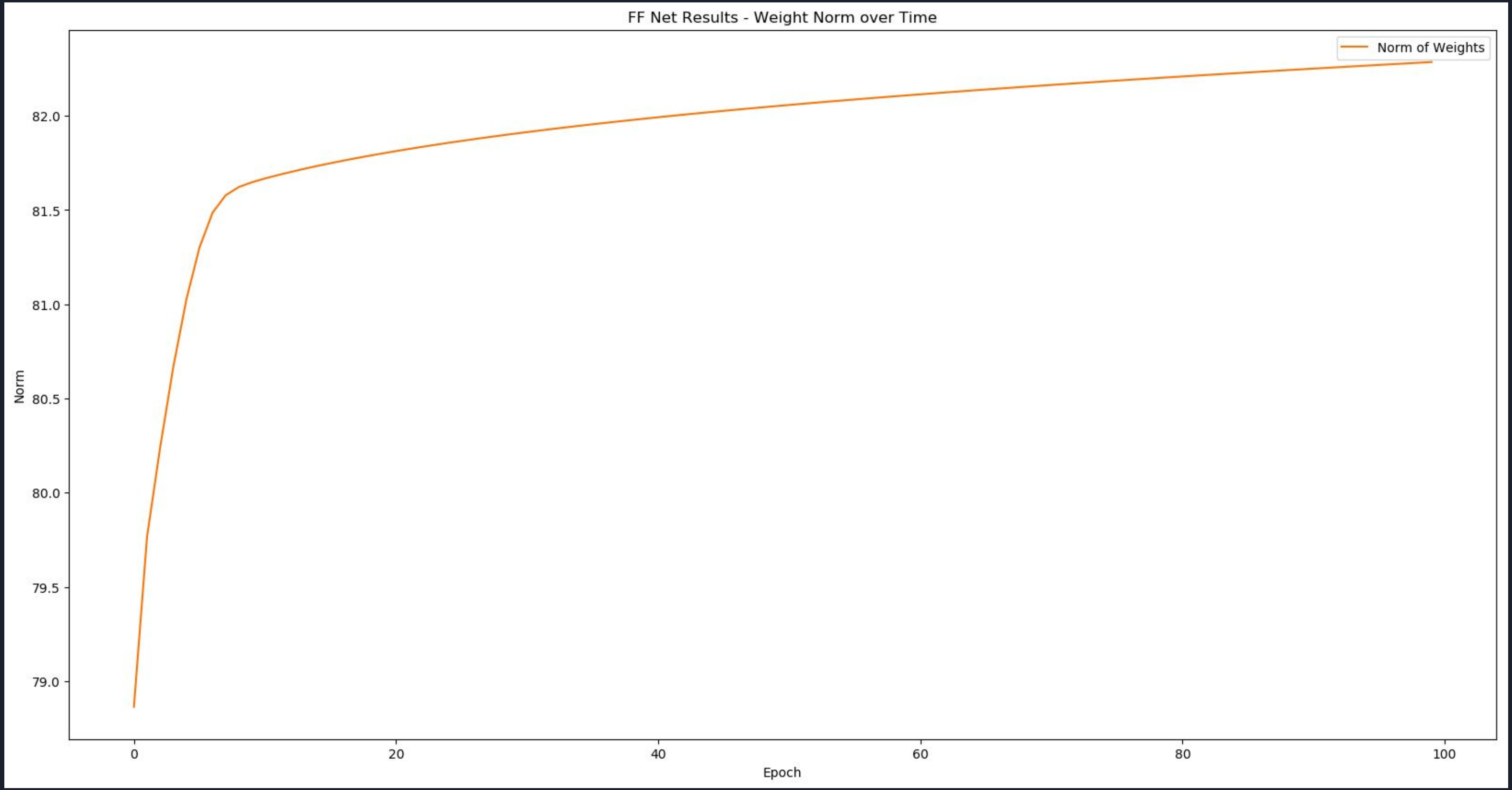
Accuracy Over Time - Centroid Network



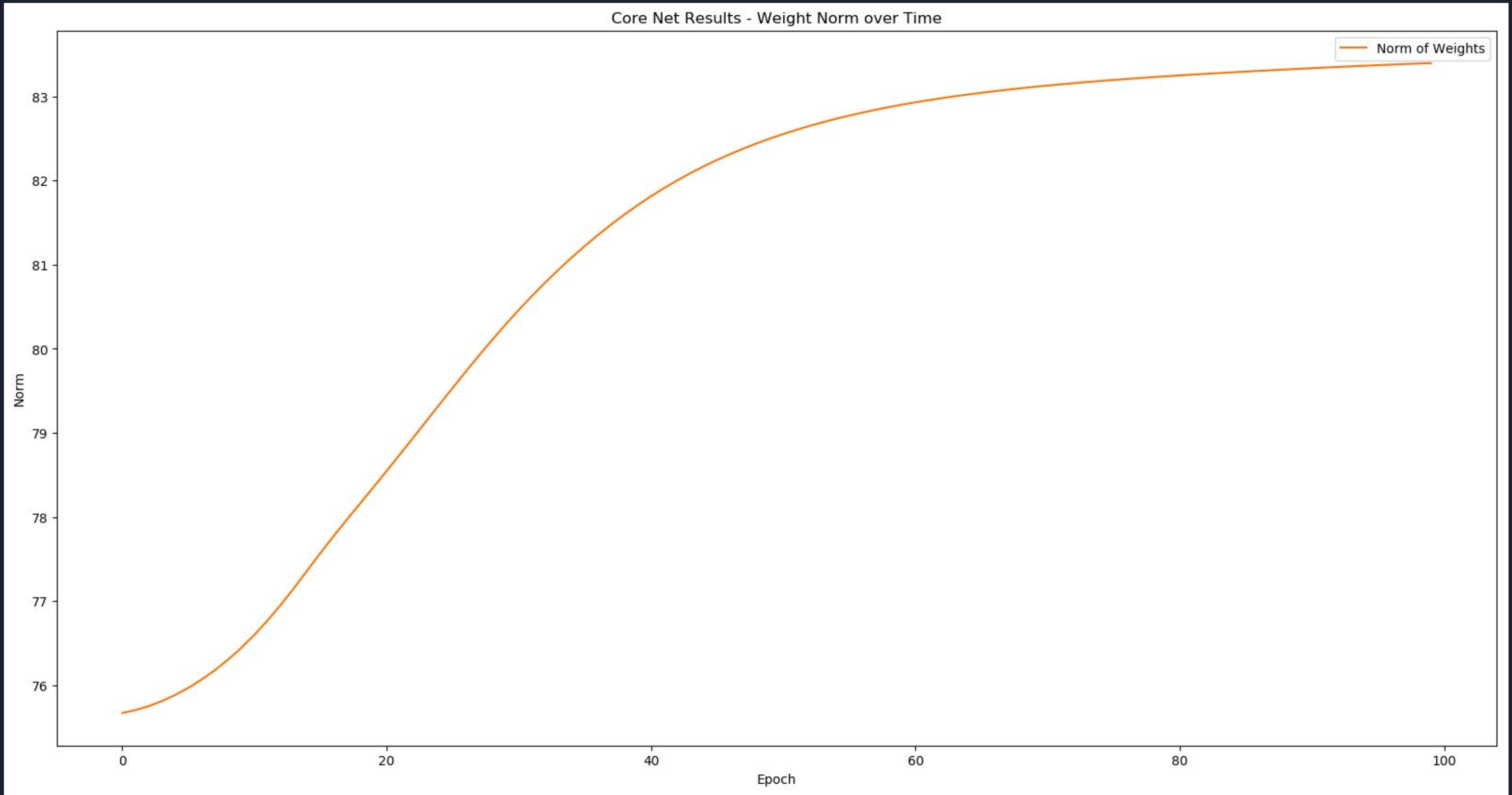
Norm of Gradient Over Time - CCE Net



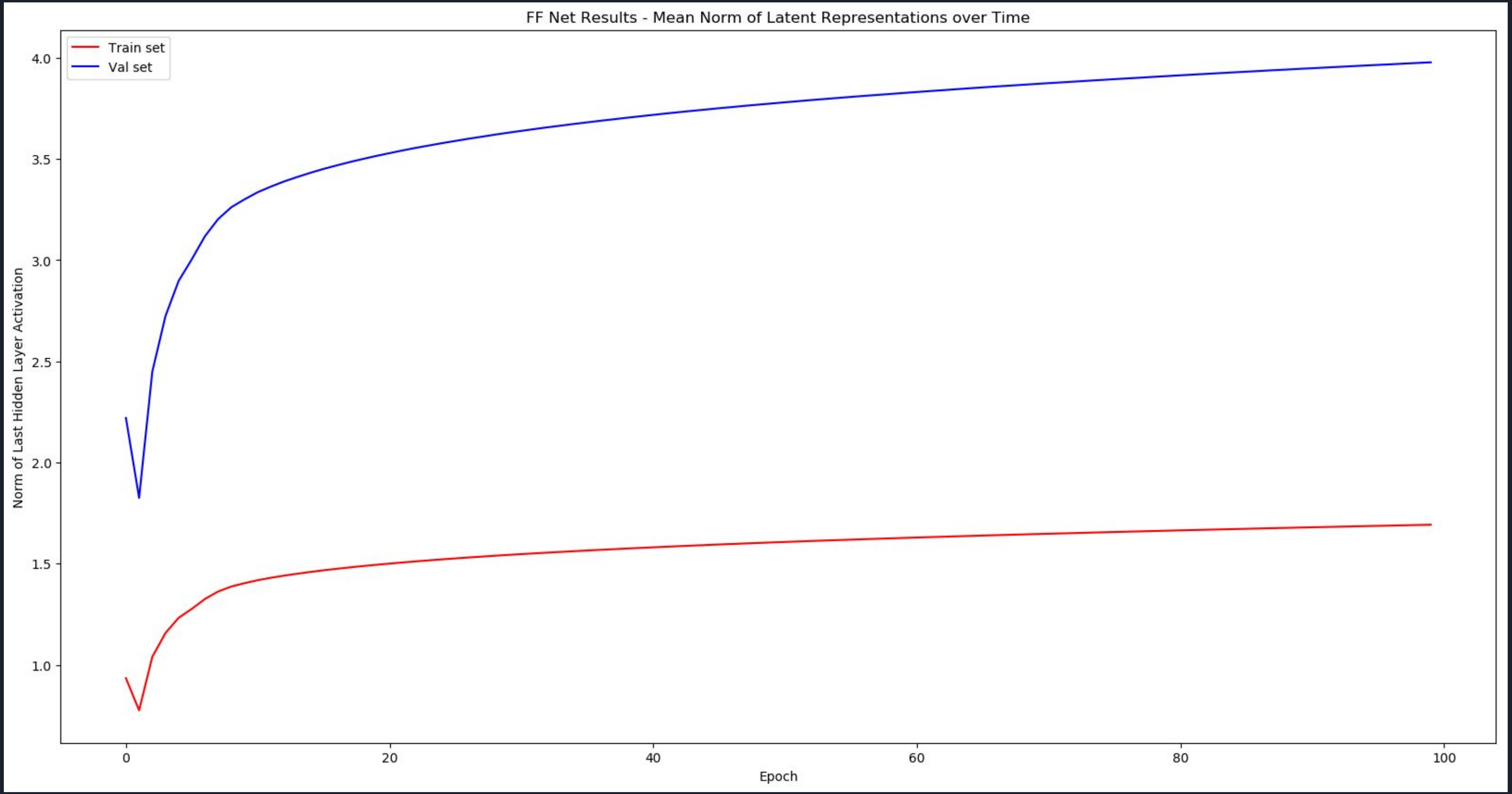
Norm of Weights Over Time - CCE Net



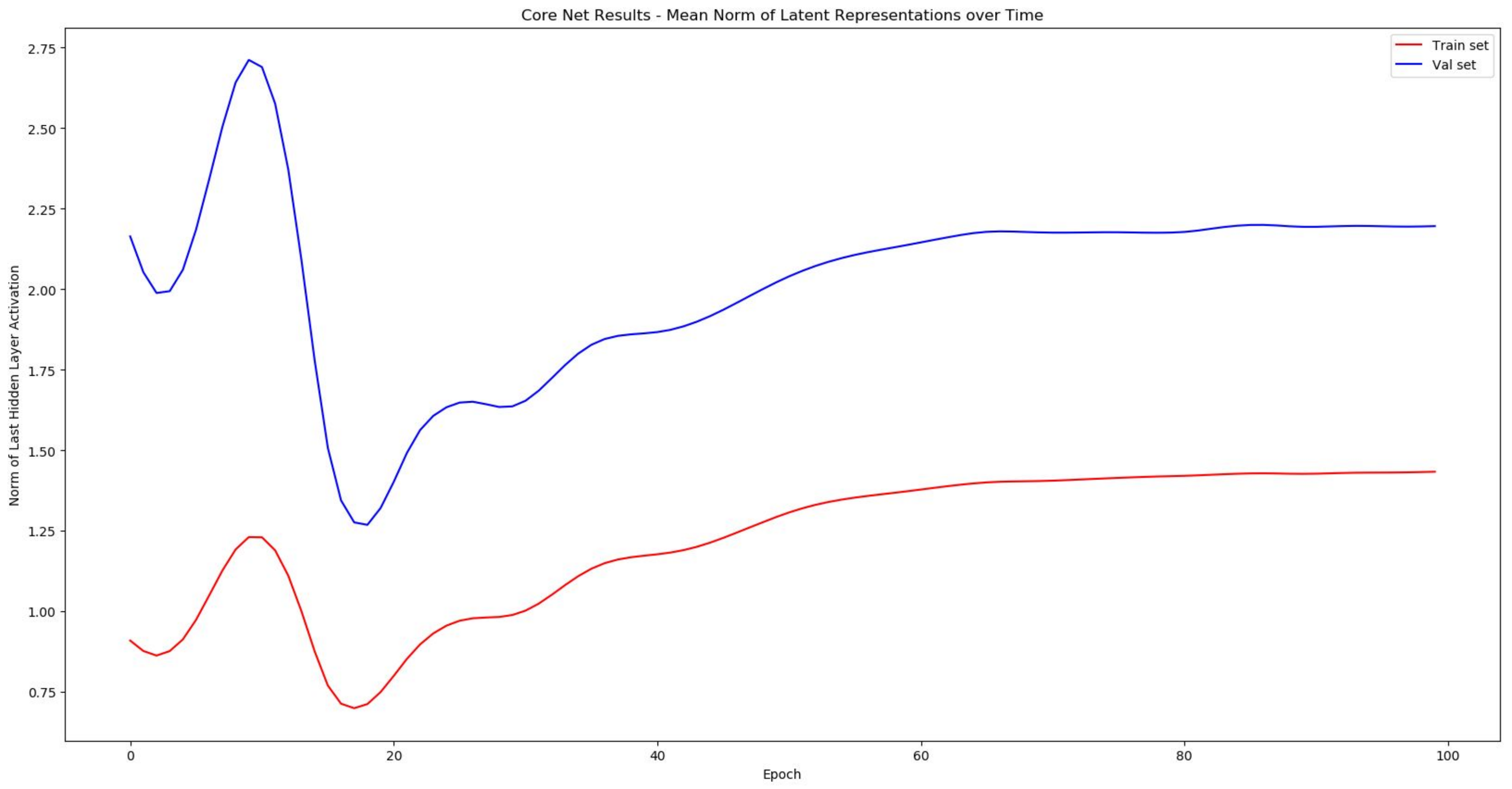
Norm of Weights Over Time - Centroid Net



Norm of Reps. over time - CCE Net



Norm of Reps. over time - Centroid Net



References

(Bengio *et al.*, 2013) - Bengio, Yoshua, Aaron Courville, and Pascal Vincent. "Representation learning: A review and new perspectives." IEEE transactions on pattern analysis and machine intelligence 35.8 (2013): 1798-1828.

(Charu et al., 2001) - Charu C Aggarwal, Alexander Hinneburg, and Daniel A Keim. “On the surprising behavior of distance metrics in high dimensional spaces”. In: ICDT. Vol. 1. Springer. 2001, pp. 420–434.

How did I do this project?

https://github.com/kiankd/nets

How did I do this project?
https://github.com/kiankd/nets

- Was it by:
    - Not commenting my code?

How did I do this project?
https://github.com/kiankd/nets

- Was it by:
  - Not commenting my code?
  - Not using “for loops”? Not declaring variables?

How did I do this project?
https://github.com/kiankd/nets

- Was it by:
    - Not commenting my code?
    - Not using “for loops”?
    - Not making separate files?

How did I do this project?
https://github.com/kiankd/nets

- Was it by:
  - Not commenting my code?
  - Not using “for loops”?
  - Not making separate files?
  - Not writing classes?

How did I do this project?

https://github.com/kiankd/nets

- Was it by:
  - Not commenting my code?
  - Not using “for loops”?
  - Not making separate files?
  - Not writing classes?
  - Handwriting the results of experiments on some scrap paper?

How did I do this project?

https://github.com/kiankd/nets

- **Surprise:** I did the exact opposite of those things!
- My philosophy when doing research is:
  - **Why not be more lazy?**

How did I do this project?

https://github.com/kiankd/nets

- **Surprise:** I did the exact opposite of those things!
- My philosophy when doing research is:
  - **Why not be more lazy?**
  - Programmers are lazy - if you want to do less work and have an easier life, **write generalized code**!

neural_core_tests4.tsv

File Edit View Insert Format Data Tools Add-ons Help Last edit was on December 7, 2017

kiank

Comments

100% $ % .0 .00 123 Arial 10

fx difficulty

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	difficulty	num_classes	make_blobs	MODEL_NAME	lr	activation	dense_layers	core	bsz	epochs	train_acc	val_acc	test_acc
2	hard	10	FALSE	0.00065_LeakyReLU (0.1)_2048-256-4096_lam1_1-lam2_1-lam3_0_3400_100	0.00065	<function <la	[2048, 256, 4096]	{'lam1': 1, 'lam2': 1, 'lam3': 0}	3400	100	0.9997058837	0.8788958533	0.8520300982
3	hard	10	FALSE	0.00075_LeakyReLU (0.1)_2048-128-2048_lam1_1-lam2_1-lam3_0_1700_100	0.00075	<function <la	[2048, 128, 2048]	{'lam1': 1, 'lam2': 1, 'lam3': 0}	1700	100	0.99911806	0.8785635043	0.8580507777
4	hard	10	FALSE	0.00065_LeakyReLU (0.1)_2048-256-1024_lam1_1-lam2_1-lam3_0_1700_100	0.00065	<function <la	[2048, 256, 1024]	{'lam1': 1, 'lam2': 1, 'lam3': 0}	1700	100	0.9988235295	0.8783391085	0.8402457849
5	hard	10	FALSE	0.0005_LeakyReLU (0.1)_2048-128-2048_lam1_1-lam2_1-lam3_0_3400_100	0.0005	<function <la	[2048, 128, 2048]	{'lam1': 1, 'lam2': 1, 'lam3': 0}	3400	100	0.9970578888	0.8768035331	0.842041324
6	hard	10	FALSE	0.00065_LeakyReLU (0.001)_2048-128-512_lam1_1-lam2_1-lam3_0_3400_100	0.00065	<function <la	[2048, 128, 512]	{'lam1': 1, 'lam2': 1, 'lam3': 0}	3400	100	1	0.8748730766	0.8422504078
7	hard	10	FALSE	0.0005_LeakyReLU (0.1)_2048-256-2048_lam1_1-lam2_1-lam3_0_3400_100	0.0005	<function <la	[2048, 256, 2048]	{'lam1': 1, 'lam2': 1, 'lam3': 0}	3400	100	1	0.8748213782	0.8489667423
8	hard	10	FALSE	0.00075_LeakyReLU (0.1)_2048-256-2048_lam1_1-lam2_1-lam3_0_3400_100	0.00075	<function <la	[2048, 256, 2048]	{'lam1': 1, 'lam2': 1, 'lam3': 0}	3400	100	0.9979411529	0.8747382289	0.8288306547
9	hard	10	FALSE	0.0005_LeakyReLU (0.05)_2048-128-512_lam1_1-lam2_1-lam3_0_1700_100	0.0005	<function <la	[2048, 128, 512]	{'lam1': 1, 'lam2': 1, 'lam3': 0}	1700	100	0.995881006	0.8746515365	0.8410526271
10	hard	10	FALSE	0.0005_LeakyReLU (0.1)_2048-128-1024_lam1_1-lam2_1-lam3_0_3400_100	0.0005	<function <la	[2048, 128, 1024]	{'lam1': 1, 'lam2': 1, 'lam3': 0}	3400	100	0.9994117635	0.8739633909	0.8500919647
11	hard	10	FALSE	0.0005_LeakyReLU (0.1)_2048-256-1024_lam1_1-lam2_1-lam3_0_1700_100	0.0005	<function <la	[2048, 256, 1024]	{'lam1': 1, 'lam2': 1, 'lam3': 0}	1700	100	0.9970592055	0.8733606278	0.8279267733
12	hard	10	FALSE	0.00075_LeakyReLU (0.001)_2048-256-1024_lam1_1-lam2_1-lam3_0_3400_100	0.00075	<function <la	[2048, 256, 1024]	{'lam1': 1, 'lam2': 1, 'lam3': 0}	3400	100	0.999705883	0.8712045756	0.8520640729
13	hard	10	FALSE	0.00065_LeakyReLU (0.05)_2048-256-4096_lam1_1-lam2_1-lam3_0_3400_100	0.00065	<function <la	[2048, 256, 4096]	{'lam1': 1, 'lam2': 1, 'lam3': 0}	3400	100	0.995588649	0.8711507056	0.8451626675
14	hard	10	FALSE	0.0005_LeakyReLU (0.005)_2048-256-1024_lam1_1-lam2_1-lam3_0_3400_100	0.0005	<function <la	[2048, 256, 1024]	{'lam1': 1, 'lam2': 1, 'lam3': 0}	3400	100	0.9991176681	0.868525862	0.8391724815
15	hard	10	FALSE	0.0005_LeakyReLU (0.1)_2048-256-4096_lam1_1-lam2_1-lam3_0_3400_100	0.0005	<function <la	[2048, 256, 4096]	{'lam1': 1, 'lam2': 1, 'lam3': 0}	3400	100	0.9982356956	0.8683563003	0.8431521768
16	hard	10	FALSE	0.00065_LeakyReLU (0.1)_2048-256-4096_lam1_1-lam2_1-lam3_0_1700_100	0.00065	<function <la	[2048, 256, 4096]	{'lam1': 1, 'lam2': 1, 'lam3': 0}	1700	100	0.9967651064	0.8680968406	0.8316953757
17	hard	10	FALSE	0.00065_LeakyReLU (0.01)_2048-256-1024_lam1_1-lam2_1-lam3_0_3400_100	0.00065	<function <la	[2048, 256, 1024]	{'lam1': 1, 'lam2': 1, 'lam3': 0}	3400	100	0.9976474567	0.867699727	0.8509550955
18	hard	10	FALSE	0.0005_LeakyReLU (0.05)_2048-128-4096_lam1_1-lam2_1-lam3_0_1700_100	0.0005	<function <la	[2048, 128, 4096]	{'lam1': 1, 'lam2': 1, 'lam3': 0}	1700	100	0.9944111081	0.8671277098	0.842241404
19	hard	10	FALSE	0.00075_LeakyReLU (0.05)_2048-128-2048_lam1_1-lam2_1-lam3_0_3400_100	0.00075	<function <la	[2048, 128, 2048]	{'lam1': 1, 'lam2': 1, 'lam3': 0}	3400	100	0.9958823187	0.8667582456	0.8301301759
20	hard	10	FALSE	0.00065_LeakyReLU (0.1)_2048-128-1024_lam1_1-lam2_1-lam3_0_1700_100	0.00065	<function <la	[2048, 128, 1024]	{'lam1': 1, 'lam2': 1, 'lam3': 0}	1700	100	0.9958818889	0.8665009634	0.8480816049
21	hard	10	FALSE	0.0005_LeakyReLU (0.1)_2048-128-2048_lam1_1-lam2_1-lam3_0_1700_100	0.0005	<function <la	[2048, 128, 2048]	{'lam1': 1, 'lam2': 1, 'lam3': 0}	1700	100	0.9973529137	0.8652520914	0.8428763353
22	hard	10	FALSE	0.00075_LeakyReLU (0.005)_2048-256-4096_lam1_1-lam2_1-lam3_0_3400_100	0.00075	<function <la	[2048, 256, 4096]	{'lam1': 1, 'lam2': 1, 'lam3': 0}	3400	100	0.9985294132	0.8651721758	0.8384392187
23	hard	10	FALSE	0.00065_LeakyReLU (0.1)_2048-256-2048_lam1_1-lam2_1-lam3_0_1700_100	0.00065	<function <la	[2048, 256, 2048]	{'lam1': 1, 'lam2': 1, 'lam3': 0}	1700	100	0.9976470369	0.8649061684	0.8392157036
24	hard	10	FALSE	0.0005_LeakyReLU (0.1)_2048-256-512_lam1_1-lam2_1-lam3_0_1700_100	0.0005	<function <la	[2048, 256, 512]	{'lam1': 1, 'lam2': 1, 'lam3': 0}	1700	100	0.9985298294	0.8648541341	0.8239931993
25	hard	10	FALSE	0.00075_LeakyReLU (0.05)_2048-256-1024_lam1_1-lam2_1-lam3_0_3400_100	0.00075	<function <la	[2048, 256, 1024]	{'lam1': 1, 'lam2': 1, 'lam3': 0}	3400	100	0.9976470369	0.8648487248	0.8240430764
26	hard	10	FALSE	0.0005_LeakyReLU (0.001)_2048-256-1024_lam1_1-lam2_1-lam3_0_3400_100	0.0005	<function <la	[2048, 256, 1024]	{'lam1': 1, 'lam2': 1, 'lam3': 0}	3400	100	0.9985298282	0.8648318789	0.846069314
27	hard	10	FALSE	0.00075_LeakyReLU (0.05)_2048-256-4096_lam1_1-lam2_1-lam3_0_3400_100	0.00075	<function <la	[2048, 256, 4096]	{'lam1': 1, 'lam2': 1, 'lam3': 0}	3400	100	0.9961759584	0.8648023673	0.8419349548
28	hard	10	FALSE	0.0005_LeakyReLU (0.005)_2048-256-2048_lam1_1-lam2_1-lam3_0_1700_100	0.0005	<function <la	[2048, 256, 2048]	{'lam1': 1, 'lam2': 1, 'lam3': 0}	1700	100	0.999705883	0.8647203675	0.836961246
29	hard	10	FALSE	0.00075_LeakyReLU (0.1)_2048-128-4096_lam1_1-lam2_1-lam3_0_3400_100	0.00075	<function <la	[2048, 128, 4096]	{'lam1': 1, 'lam2': 1, 'lam3': 0}	3400	100	0.9976474567	0.8636973242	0.8449513636
30	hard	10	FALSE	0.00065_LeakyReLU (0.1)_2048-128-4096_lam1_1-lam2_1-lam3_0_1700_100	0.00065	<function <la	[2048, 128, 4096]	{'lam1': 1, 'lam2': 1, 'lam3': 0}	1700	100	0.9997058932	0.8635035823	0.8358569104
31	hard	10	FALSE	0.0005_LeakyReLU (0.1)_2048-128-1024_lam1_1-lam2_1-lam3_0_1700_100	0.0005	<function <la	[2048, 128, 1024]	{'lam1': 1, 'lam2': 1, 'lam3': 0}	1700	100	0.9964701671	0.8634591373	0.8237343416
32	hard	10	FALSE	0.00065_LeakyReLU (0.05)_2048-128-1024_lam1_1-lam2_1-lam3_0_3400_100	0.00065	<function <la	[2048, 128, 1024]	{'lam1': 1, 'lam2': 1, 'lam3': 0}	3400	100	0.995295372	0.8633928703	0.8281714354
33	hard	10	FALSE	0.0005_LeakyReLU (0.1)_2048-128-4096_lam1_1-lam2_1-lam3_0_3400_100	0.0005	<function <la	[2048, 128, 4096]	{'lam1': 1, 'lam2': 1, 'lam3': 0}	3400	100	0.9997058799	0.8631679343	0.8473429094
34	hard	10	FALSE	0.00065_LeakyReLU (0.005)_2048-256-2048_lam1_1-lam2_1-lam3_0_1700_100	0.00065	<function <la	[2048, 256, 2048]	{'lam1': 1, 'lam2': 1, 'lam3': 0}	1700	100	0.9988239437	0.8631239026	0.8346588482
35	hard	10	FALSE	0.00065_LeakyReLU (0.1)_2048-256-1024_lam1_1-lam2_1-lam3_0_3400_100	0.00065	<function <la	[2048, 256, 1024]	{'lam1': 1, 'lam2': 1, 'lam3': 0}	3400	100	0.9976466119	0.8630939855	0.828962681
36	hard	10	FALSE	0.00075_LeakyReLU (0.001)_2048-256-1024_lam1_1-lam2_1-lam3_0_1700_100	0.00075	<function <la	[2048, 256, 1024]	{'lam1': 1, 'lam2': 1, 'lam3': 0}	1700	100	0.9973541573	0.8625670715	0.8391603007
37	hard	10	FALSE	0.00075_LeakyReLU (0.001)_2048-128-512_lam1_1-lam2_1-lam3_0_1700_100	0.00075	<function <la	[2048, 128, 512]	{'lam1': 1, 'lam2': 1, 'lam3': 0}	1700	100	0.9994117591	0.8618572736	0.8371776714

best_corenet_tracking_results.tsv

File Edit View Insert Format Data Tools Add-ons Help Last edit was on December 8, 2017

kiankd@gmail.com

Comments Share

fx difficulty

	difficulty	num_classes	make_blobs	lr	activation	dense_layers	core	bsz	epochs	MODEL_NAME	epoch	weight_norm	gradient_norm	activation_normt	activation_normv	cce_losstrain	cce_lossval	attractive_samp_	attracti
2	hard	10	FALSE	0.00065_LeakyR	0.00065	LeakyReLU (0.1)	[2048, 256, 4096	{'lam1': 1, 'lam2':	3400	100	0	75.6654892	No Grad...	0.9192941464	2.196246134	1.797406197	1.889685512	0.133137092	0.130
3	hard	10	FALSE	0.00065_LeakyR	0.00065	LeakyReLU (0.1)	[2048, 256, 4096	{'lam1': 1, 'lam2':	3400	100	1	75.70182037	0.06895793038	0.9057217946	2.13400533	1.606015921	1.78457582	0.1610992998	0.161
4	hard	10	FALSE	0.00065_LeakyR	0.00065	LeakyReLU (0.1)	[2048, 256, 4096	{'lam1': 1, 'lam2':	3400	100	2	75.74894714	0.1478565787	0.876474394	2.017456868	2.328990936	2.804929495	0.1774915755	0.182
5	hard	10	FALSE	0.00065_LeakyR	0.00065	LeakyReLU (0.1)	[2048, 256, 4096	{'lam1': 1, 'lam2':	3400	100	3	75.81006622	0.1481443878	0.8754388069	1.976783244	3.455342293	4.190791607	0.1814031303	0.19
6	hard	10	FALSE	0.00065_LeakyR	0.00065	LeakyReLU (0.1)	[2048, 256, 4096	{'lam1': 1, 'lam2':	3400	100	4	75.88304138	0.1326389134	0.9032941751	2.019108887	4.197408199	5.205317497	0.1803814918	0.192
7	hard	10	FALSE	0.00065_LeakyR	0.00065	LeakyReLU (0.1)	[2048, 256, 4096	{'lam1': 1, 'lam2':	3400	100	5	75.96821594	0.1167933739	0.9644761029	2.141779582	4.930743217	6.215703487	0.1762337089	0.192
8	hard	10	FALSE	0.00065_LeakyR	0.00065	LeakyReLU (0.1)	[2048, 256, 4096	{'lam1': 1, 'lam2':	3400	100	6	76.06547546	0.1122751904	1.048624339	2.316297607	5.682807922	7.342429161	0.1711863875	0.191
9	hard	10	FALSE	0.00065_LeakyR	0.00065	LeakyReLU (0.1)	[2048, 256, 4096	{'lam1': 1, 'lam2':	3400	100	7	76.17578888	0.1081420858	1.136433967	2.497807414	6.349761963	8.430276871	0.166725263	0.191
10	hard	10	FALSE	0.00065_LeakyR	0.00065	LeakyReLU (0.1)	[2048, 256, 4096	{'lam1': 1, 'lam2':	3400	100	8	76.2999115	0.1054417493	1.209095244	2.646006673	6.814012527	9.335530281	0.1629364341	0.192
11	hard	10	FALSE	0.00065_LeakyR	0.00065	LeakyReLU (0.1)	[2048, 256, 4096	{'lam1': 1, 'lam2':	3400	100	9	76.43852234	0.1056058588	1.254388787	2.732038371	6.977712154	9.931295395	0.1593448818	0.194
12	hard	10	FALSE	0.00065_LeakyR	0.00065	LeakyReLU (0.1)	[2048, 256, 4096	{'lam1': 1, 'lam2':	3400	100	10	76.5921936	0.1073336069	1.264251206	2.73514974	6.718180656	9.982587814	0.1551364809	0.19
13	hard	10	FALSE	0.00065_LeakyR	0.00065	LeakyReLU (0.1)	[2048, 256, 4096	{'lam1': 1, 'lam2':	3400	100	11	76.76095581	0.1142427909	1.234240005	2.643250122	5.993331909	9.355545044	0.1497836858	0.19
14	hard	10	FALSE	0.00065_LeakyR	0.00065	LeakyReLU (0.1)	[2048, 256, 4096	{'lam1': 1, 'lam2':	3400	100	12	76.94435883	0.1211752727	1.165448142	2.463157349	4.897107601	8.181108475	0.1428721696	0.20
15	hard	10	FALSE	0.00065_LeakyR	0.00065	LeakyReLU (0.1)	[2048, 256, 4096	{'lam1': 1, 'lam2':	3400	100	13	77.14120483	0.1312593466	1.06344188	2.205767212	3.601153374	6.609869003	0.1340707839	0.201
16	hard	10	FALSE	0.00065_LeakyR	0.00065	LeakyReLU (0.1)	[2048, 256, 4096	{'lam1': 1, 'lam2':	3400	100	14	77.34857178	0.142183813	0.9442019474	1.902767537	2.314720631	4.806187153	0.1227589473	0.199
17	hard	10	FALSE	0.00065_LeakyR	0.00065	LeakyReLU (0.1)	[2048, 256, 4096	{'lam1': 1, 'lam2':	3400	100	15	77.56015778	0.1517961276	0.8360095215	1.626215617	1.30820787	3.211443901	0.1090029255	0.192
18	hard	10	FALSE	0.00065_LeakyR	0.00065	LeakyReLU (0.1)	[2048, 256, 4096	{'lam1': 1, 'lam2':	3400	100	16	77.76766205	0.1664497091	0.7704398122	1.441729329	0.7463138103	2.194934368	0.09379908442	0.179
19	hard	10	FALSE	0.00065_LeakyR	0.00065	LeakyReLU (0.1)	[2048, 256, 4096	{'lam1': 1, 'lam2':	3400	100	17	77.96893311	0.1779068178	0.7469142061	1.34897227	0.4710393846	1.6461308	0.07840745151	0.167
20	hard	10	FALSE	0.00065_LeakyR	0.00065	LeakyReLU (0.1)	[2048, 256, 4096	{'lam1': 1, 'lam2':	3400	100	18	78.16651154	0.1789088372	0.7489531394	1.313486226	0.3263456523	1.424468517	0.06541463733	0.158
21	hard	10	FALSE	0.00065_LeakyR	0.00065	LeakyReLU (0.1)	[2048, 256, 4096	{'lam1': 1, 'lam2':	3400	100	19	78.36257172	0.167557869	0.7751585478	1.343866577	0.2678762078	1.41131556	0.05467592925	0.151
22	hard	10	FALSE	0.00065_LeakyR	0.00065	LeakyReLU (0.1)	[2048, 256, 4096	{'lam1': 1, 'lam2':	3400	100	20	78.55899048	0.1433134372	0.8225137868	1.423094381	0.2474112958	1.462405682	0.04549893737	0.145
23	hard	10	FALSE	0.00065_LeakyR	0.00065	LeakyReLU (0.1)	[2048, 256, 4096	{'lam1': 1, 'lam2':	3400	100	21	78.75717163	0.122943626	0.877918773	1.524176941	0.2333339751	1.517640352	0.03795795888	0.138
24	hard	10	FALSE	0.00065_LeakyR	0.00065	LeakyReLU (0.1)	[2048, 256, 4096	{'lam1': 1, 'lam2':	3400	100	22	78.95695496	0.1074515607	0.9281444594	1.613433431	0.2142132223	1.536291718	0.03212947771	0.134
25	hard	10	FALSE	0.00065_LeakyR	0.00065	LeakyReLU (0.1)	[2048, 256, 4096	{'lam1': 1, 'lam2':	3400	100	23	79.15736389	0.09579885358	0.9671373076	1.674329936	0.1934557706	1.564035535	0.02760543115	0.132
26	hard	10	FALSE	0.00065_LeakyR	0.00065	LeakyReLU (0.1)	[2048, 256, 4096	{'lam1': 1, 'lam2':	3400	100	24	79.35749817	0.08276455611	0.9945684455	1.709778646	0.1783224791	1.579733729	0.02403103746	0.131
27	hard	10	FALSE	0.00065_LeakyR	0.00065	LeakyReLU (0.1)	[2048, 256, 4096	{'lam1': 1, 'lam2':	3400	100	25	79.55630493	0.0734213973	1.012845028	1.728936157	0.1687776595	1.554910779	0.02108053677	0.129
28	hard	10	FALSE	0.00065_LeakyR	0.00065	LeakyReLU (0.1)	[2048, 256, 4096	{'lam1': 1, 'lam2':	3400	100	26	79.75263214	0.06538726532	1.024513442	1.735973918	0.1607030034	1.511618972	0.01856555231	0.127
29	hard	10	FALSE	0.00065_LeakyR	0.00065	LeakyReLU (0.1)	[2048, 256, 4096	{'lam1': 1, 'lam2':	3400	100	27	79.94540405	0.05989164874	1.030930678	1.732307943	0.1557636559	1.443056464	0.01641438156	0.125
30	hard	10	FALSE	0.00065_LeakyR	0.00065	LeakyReLU (0.1)	[2048, 256, 4096	{'lam1': 1, 'lam2':	3400	100	28	80.13365173	0.05288037017	1.034195485	1.723017375	0.1487817764	1.368454695	0.01465117	0.122
31	hard	10	FALSE	0.00065_LeakyR	0.00065	LeakyReLU (0.1)	[2048, 256, 4096	{'lam1': 1, 'lam2':	3400	100	29	80.31642151	0.04749021915	1.03849832	1.718240153	0.1398690641	1.303009033	0.01317549311	0.119
32	hard	10	FALSE	0.00065_LeakyR	0.00065	LeakyReLU (0.1)	[2048, 256, 4096	{'lam1': 1, 'lam2':	3400	100	30	80.49287415	0.04400815205	1.048484102	1.726365763	0.1310537308	1.255259871	0.01186014712	0.116
33	hard	10	FALSE	0.00065_LeakyR	0.00065	LeakyReLU (0.1)	[2048, 256, 4096	{'lam1': 1, 'lam2':	3400	100	31	80.6624527	0.04065045109	1.066666044	1.750401204	0.1236768737	1.237764239	0.0106917331	0.114
34	hard	10	FALSE	0.00065_LeakyR	0.00065	LeakyReLU (0.1)	[2048, 256, 4096	{'lam1': 1, 'lam2':	3400	100	32	80.82494354	0.03683506915	1.09165197	1.786445923	0.1173838153	1.248189807	0.009702078998	0.11
35	hard	10	FALSE	0.00065_LeakyR	0.00065	LeakyReLU (0.1)	[2048, 256, 4096	{'lam1': 1, 'lam2':	3400	100	33	80.98029327	0.03303685059	1.11951215	1.827189535	0.1167004332	1.276504517	0.008876625448	0.113
36	hard	10	FALSE	0.00065_LeakyR	0.00065	LeakyReLU (0.1)	[2048, 256, 4096	{'lam1': 1, 'lam2':	3400	100	34	81.12856293	0.03000264019	1.146502686	1.865378621	0.1198566556	1.306975126	0.008160671219	0.113
37	hard	10	FALSE	0.00065_LeakyR	0.00065	LeakyReLU (0.1)	[2048, 256, 4096	{'lam1': 1, 'lam2':	3400	100	35	81.26978302	0.02719655612	1.169972211	1.897185465	0.1242259294	1.334929347	0.007523773238	0.113

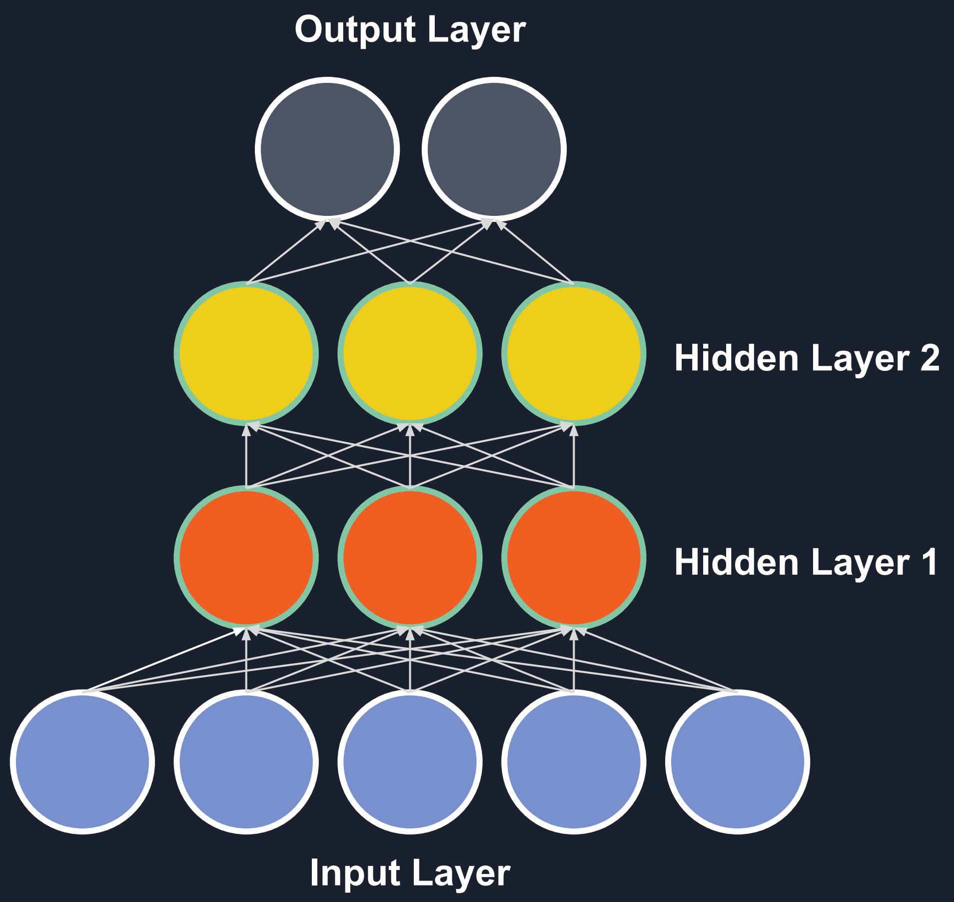
Java Update Available

Principles for Good Research Practices

- Write generalized code using OOP.
- Use multiple files and classes to separate tasks.
- Code it like you will use it in the future.
- Save your results every time you get them!

Principles for Good Research Practices

- Write generalized code using OOP.
- Use multiple files and classes to separate tasks.
- Code it like you will use it in the future.
- Save your results every time you get them!
- **USE VERSION CONTROL!** (e.g., Github)


Output Layer
Hidden Layer 2
Hidden Layer 1
Input Layer