Stone Mele
Structured Prediction: CS 6355
04/23/2019

Structured Multi-Label Prediction Exploration through Neural Networks

Introduction:

For this project I wanted to tackle some form of structured prediction making use of neural networks. It took a little bit of exploration, and searching but I stumbled upon a paper that tackled multi-label classification in a very interesting way. The paper in question is "Learning Deep Latent Spaces for Multi-Label Classification" by Chih-Kuan Yeh, Wei-Chieh Wu, Wei-Jen Ko, and Yu-Chiang Frank Wang. The approach outlined in this paper, and summarized in this report, combines various encoder decoder components that are influenced by unique and powerful loss functions. The authors named this model the Canonical Correlated AutoEncoder (C2AE) which is what I will refer to the architecture as throughout the report. Along with the following discussion my pytorch implementation for this model can be found here, as well as the jupyter notebooks used to generate the experiment results. Please refer to that for any implementation details not discussed in this report.

As stated above the problem I am tackling is that of multi-label classification. This refers to a classification task where a single example can belong to multiple class instances. This provides a unique and interesting challenge as the output space increases considerably when compared to a normal classification task. These multi-label classification tasks come up quite often in the real world, and can greatly benefit from models that take the structure of the problem into consideration. More specifically than just tackling the multi-label classification task is that I wanted to find a method that could fit into an end to end training scheme while still taking advantage of the structured task at hand. This is exactly what the C2AE paper accomplishes and why I wanted to reimplement it. In the next section I will discuss the details of the network, and challenges I faced while implementing it.

Architecture:

The C2AE model architecture can be broken down into 3 main components that are influenced by a 2 part loss function. Seen in **Figure 1** is a diagram of the model architecture with the main components outlined as follows:

- **X**: Input of the model. (N, D)
- F_x: Encoder model that takes input **X** and encodes it into latent space **Z**. This encoder has no restrictions on it and can be defined per problem. For cases explored in this report it's a simple feed forward network. If our input space were images this might be a convolutional network.
- **Y**: Labels for input **X**. Since this is multi-label these will be binary vectors. (N, k)
- F_e: Encoder model that takes input **Y** and encodes it into latent space **Z**. Like F_x no restrictions on this model and is defined per problem like F_x.
- **Z**: Latent space in which **X**, and **Y** are both encoded to by F_x, and F_e respectively. (N, l)
- F_d: Decoder model that takes as input vectors from the latent space **Z** and decodes them to a space the same dimension as **Y.** This model needs to use a sigmoid activation function at the end of it in order to transfer outputs from logits to probabilities.

Stone Mele
Structured Prediction: CS 6355
04/23/2019

- **O**: Output of the F_d model. This model will have a sigmoid activation function on it so the vector values are between $[0,\ 1]$. This then allows us to apply thresholding to gain our predictions.

I will be using function notation when referring to the forward pass of the various networks. For example if input x goes through F_x this can be represented as $z = F_x(x)$.

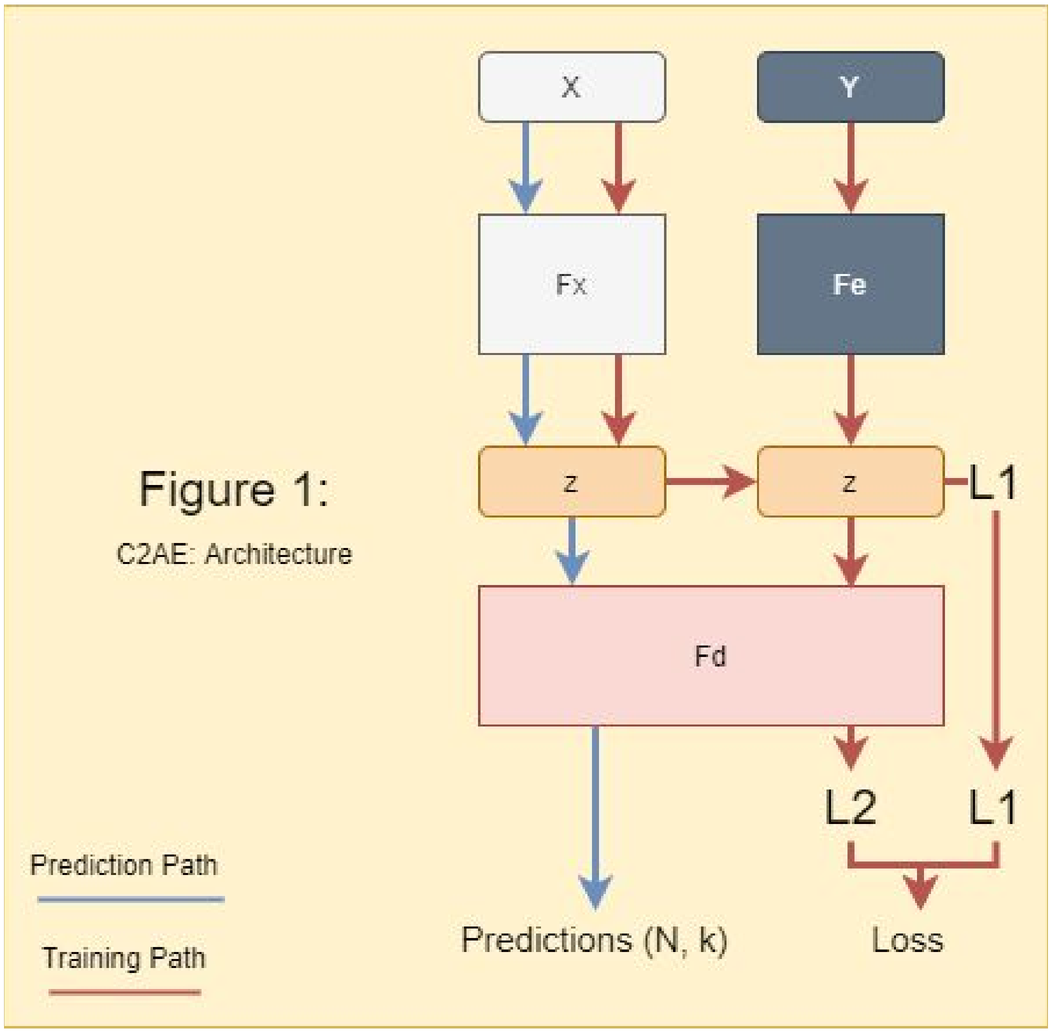

Figure 1:
C2AE: Architecture

Next I'll examine the components of the loss functions that influence these models to learn the correct embeddings, and weights. The first component of the loss deals with the encoder architectures F_x and F_e. The loss is defined as: $\|F_x(x) - F_e(y)\|^2_F$ such that $F_x(X)F_x(X)^T = F_e(Y)F_e(Y)^T = I$. The motivation for this loss is that we want to encode **X**, and **Y** both into a space that correlates the two. This loss was inspired by Kettering et al 1971, and reworked to allow us to achieve this goal in a deep learning context. I'll refer to this loss component as L_1. Once we have this trained it gives us the power to encode to the latent space **Z** from either **X**, or **Y** space. It's important to realize here that no actual label prediction has been done yet. In combination with the next loss the true power of the model is revealed. The next loss is defined as follows:

$$L_2 = \sum_i^N E_i$$

$g_i^j = F_d(F_e(y_i))^j$ corresponds to jth output probability of $F_d(F_e(y_i))$.

y_i^1 corresponds to set of one labels in y_i

y_i^0 corresponds to set of zero labels in y_i

$$E_i = \frac{1}{|y_i^1||y_i^0|} \sum_{(p,q \in y_i^1 \times y_i^0)} e^{(g_i^q - g_i^p)}$$

This loss is a little bit more involved but once you break it into pieces it becomes a lot more intuitive. Examining this loss reveals that it penalizes occurances of the incorrect pairs of positive, and negative predictions while not penalizing correct ones. From minimizing this loss we also thus in turn maximize the correct negative, positive label pair predictions that occur. This is exactly the type of property we want our loss function to possess if we are trying to decode back to the space of **Y**. Finally the losses are combined as follows $L_1 + \alpha L_2$. I added another tuning term β that I added to the final loss so that it is now given as $\beta L_1 + \alpha L_2$. With this we can now perform backpropagation through the network. These loss functions are accumulated once each minibatch, and are labeled at the locations they are collected in **Figure 1.**

<u>**TrainingInference:**</u>

Unlike a normal model this architecture uses two different paths when training, and predicting. During training time we are assumed to have access to samples from both **X**, and **Y**. To optimize L_1 we need access to both encodings $F_x(x)$ and $F_e(y)$. We also need to get access to the decoding of the y encoding $F_d(F_e(y))$ to calculate the L_2 loss. They are then combined and form the final loss. I tried various optimizers for the model, but ended up mostly using Adam, and RMSProp (RMSProp used in paper). The paths required for training are seen highlighted in red in **Figure 1**.

Another detail is that it seemed I used smaller models in comparison to that referenced in the Yeh et al. All their models had hidden layers of size 512 while mine varied depending on the dataset type. I made use of the leaky relu as that is what is used within the paper.

<u>***Inference:***</u>

During inference time we only have samples from **X** as we are trying to predict its corresponding value in **Y.** Since we no longer care about optimization we can simply compute $F_d(F_x(x))$. This will give us k probabilities per example representing the chance they represent that class. The use of the sigmoid activation function here is key if softmax or other alternatives were used the output space would be incorrect. The path required for training is seen highlighted in blue in **Figure 1**.

Stone Mele
Structured Prediction: CS 6355
04/23/2019

Experiments:

The experiments I conducted consisted of comparing three different model types: a (One vs All) Gradient Boosted Classifier (OvAGBC), a naive feed forward network, and the C2AE architecture. I wanted to see if there was a noticeable difference in the performance of the C2AE architecture, and the simpler approaches. For the OvAGBC I made use of the great suite of classifiers sklearn has, which provided a One vs All wrapper that could be used with any classifier! For the naive feed forward model I will be training a single linear layer with a sigmoid output in the shape of the labels. This model is not taking advantage of the correlation that C2AE is able to encode but it still can capture a lot of local information. Comparing these three models will allow me to see if encoding label information within your model is helpful, and if a more advanced encoding technique (C2AE) will pay off in comparison to a more naive approach.

For the experiments I will be taking the Accuracy, Precision, Recall, F1, and Hamming Loss for each model. I will be using the macro, and micro versions of Precision, Recall, and F1. Macro and Micro refer to the context in which the metric is calculated. For macro the metric is an average across the metric for each class label, while micro calculates metrics globally per positive, negative label. Accuracy is based on exact label matches, so an example will only be counted correctly classified if all the label dimensions match. This range of metrics should provide plenty of points for comparison between the models. The following datasets were the ones that I used to compare the three different models:

- Scene (Boutell et al. 2004): Dataset of images of scenes. I am using a dataset with preprocessed features.
- TMC2007 (Srivastavan et al. 2005): Dataset of featurized text reports detailing airplane failure reports.
- Mediamill(Snoek et al. 2006): Features extracted from video data containing various events.

I used the scene, tmc2007, and mediamill dataset to compare the different model types. These different datasets represent a good range of property types. The property that will most likely affect the results the most is "# of labels". In **Table 1** you can see a quick overview of some of the basic properties of the datasets. (More comprehensive breakdown on) As you can see there is a big difference in the scene, and mediamill dataset label dimension.

Table 1: Dataset Properties.

Name:	# examples	# labels	# features	cardinality	# in train split	# in test split
scene	2,407	6	294	1.0174	1,211	1,196
tmc2007	28,596	22	500	2.158	21,519	7,077
mediamill	43,907	101	120	4.376	30,993	12,914

Stone Mele
Structured Prediction: CS 6355
04/23/2019

After applying each of the different models to the data the results can be seen below in Table 2-4. I have bolded the best performing model per metric. I did not tune the threshold for applicable models and use 0.5 for any probability to prediction conversion.

Table 2: Scene results.

Model:	Accuracy	F1 [Macro]	F1 [Micro]	Precision [Macro]	Precision [Micro]	Recall [Macro]	Recall [Micro]	Hamming Loss
(OvA) Gradient Boosted	0.571906	**0.723272**	**0.722296**	**0.845217**	**0.846791**	0.635851	0.629715	**0.087653**
Naive Feed Forward	0.500836	0.684643	0.677105	0.765545	0.751879	0.620372	0.615858	0.106326
C2AE	**0.612876**	0.700145	0.697829	0.767502	0.762078	**0.653098**	**0.643571**	0.100891

The results from running the models on the scene dataset are somewhat surprising. I suspected C2AE to sweep the board, but it only topped out on accuracy, and both recall metrics. This could be due to how simple this dataset is with only 6 classes making it easy for the more naive models to perform well. The difference in accuracy is quite large between the naive network, and C2AE architecture though. It's also important to note we don't have a tremendous amount of data here, and building encoders often requires a fair amount to get a good latent representation. The subcomponents of the C2AE architecture used here are a two hidden layer network for the F_x encoder, and a single hidden layer network for both F_e and F_d. The space to explore here is quite large which is one of the downsides I found to this architecture. I am slowly gaining intuition on how these different parameters affect the model.

Table 3: TMC 2007 results.

Model:	Accuracy	F1 [Macro]	F1 [Micro]	Precision [Macro]	Precision [Micro]	Recall [Macro]	Recall [Micro]	Hamming Loss
(OvA) Gradient Boosted	0.301964	0.627317	0.690261	0.826195	0.754723	0.537204	0.635944	0.057112
Naive Feed Forward	0.3300833	0.654624	0.716949	0.788828	0.762776	0.579027	0.676315	0.053438

C2AE	**0.739437**	**0.830179**	**0.920582**	**0.848657**	**0.945956**	**0.827730**	**0.896534**	**0.015479**

The results for this experiment are clear. C2AE architecture performs much better than the more naive versions. My results for the two variants of F1 score also performed better than the experiments run in the original Yeh et al paper. C2AE has a lot more data to work with in this example which might be a reason for the performance increase. Training this model was also much easier to find convergence compared to previously. I did use the same hyper parameters as in the Scene dataset which most likely started it off in a good location. I also used the same subcomponents of the C2AE architecture as in scene with varying neuron sizes.These results are exciting, and it's quite cool to see how effective C2AE is at incorporating the structured information from the labels, and features.

Table 4: Mediamill results:

Model:	Accuracy	F1 [Macro]	F1 [Micro]	Precision [Macro]	Precision [Micro]	Recall [Macro]	Recall [Micro]	Hamming Loss
OvAGB	0.087812	0.102668	0.536478	0.203368	0.717113	0.085217	0.428534	0.032305
Naive Feed Forward	**0.089902**	0.065570	0.541632	**0.231999**	**0.767903**	0.050738	0.418358	**0.0308905**
C2AE	0.075731	**0.127881**	**0.582687**	0.213698	0.620816	**0.111957**	**0.548971**	0.034303

Unfortunately C2AE did not perform as well as I was hoping for the Mediamill dataset. This performance is most likely largely in part due to the increase in label dimension size. I had a really hard time getting this model to converge compared to the other datasets. The loss graphs also were much more erratic and imprecise, while the first two datasets were smooth and settled into their optima. This leads me to believe I didn't find the optimal set of parameters, or did not use the optimal model architecture for the job. If I had more time I would explore this but the training time to thoroughly explore the model space is overwhelming. Even in what I assume to be a non optimal version of the C2AE model it was able to compete and even perform better than the naive approaches in some metrics. I attempted this same training on another similar dataset, and also had weird convergence issues further leading to my suspicion that the large label dimension size is the culprit.

Further Exploration:

Even though the effort I have made has been significant there is still so much to explore related to this topic. First off I would spend more time experimenting with datasets like mediamill to see if I can increase the models performance. I would also like to attempt at training a model that starts with images and involves convolutions within the encoder layers. Depending on the models performance those results could really solidify this approach as a viable model architecture to throw into my play book. It would also be worth investigating if there are any network architectures designed to handle sparse inputs, as the F_e encoder of the network works solely on sparse input.

Another related area would be trying to apply the C2AE model architecture to different input, and output types. This would most likely involve reworking the loss functions within the network, but the power, and utility that the shared input, label encoding brings is quite useful, and would love to find a way to incorporate this in other structured tasks.

I would also like to explore the interpretability that this model provides with respect to the encoding space generated by the F_e and F_x encoders. Asking a trained model to encode various permutations of the y labels should reveal some interesting patterns. Since the latent dimension is quite large in some models dimensionality reduction will need to be applied. Doing this could allow us to see which labels were close to one another possibly hinting at issues that the models are getting tricked up in. Overall there are many areas to explore based on the concepts in this paper, and I hope I find time in the future to analyze them.

Conclusion:

If I run into a multi-label problem in the wild this approach will definitely come to mind. This approach seems to have a clear advantage over the naive feed forward network, and the naive structured approach of OvAGB in certain cases. I was actually quite impressed by how well the Ove Vs All wrapper that sklearn provided performed. It was by far the quickest and easiest model to get set up and running, which is why it will definitely be a model I will pull out when first approaching a multi label task such as this one to gain a quick and easy benchmark. With respect to C2AE I believe the architecture is quite useful. From the experiments above we can see that is also useful performance wise too. The only downside to this method is the amount of model tuning needed to be done considering the complex loss function, and network dynamics. Having to figure out a suitable value for all the hyperparameters was quite tasking, and challenging for some datasets. Even if I don’t get to use this exact model type in the future I have learned some valuable ways of setting up a model architecture to take advantage of the information at hand. I have also learned the benefit of trying to encode structure into a learning problem can bring. One does not only gain performance in the ideal cases, but I found myself thinking deeper about the problem at hand using the structure as a guide. I can't wait to try and tackle a similar task to the ones seen in this paper in the future.

Stone Mele
Structured Prediction: CS 6355
04/23/2019

Works Cited

M.R. Boutell, J. Luo, X. Shen, and C.M. Brown. Learning multi-labelscene classiffication. Pattern Recognition, 37(9):1757-1771, 2004.

Andrew, G.; Arora, R.; Bilmes, J. A.; and Livescu, K. 2013. Deep canonical correlation analysis. In ICML (3), 1247–1255.

Tsoumakas, Grigorios, et al. “Datasets.” Multilabel Datasets, mulan.sourceforge.net/datasets-mlc.html.

Kettenring, J. R. 1971. Canonical analysis of several sets of variables. Biometrika 58(3):433–451.

A. Srivastava, B. Zane-Ulman: Discovering recurring anomalies in text reports regarding complex space systems. In: 2005 IEEE Aerospace Conference. (2005)

C.G.M. Snoek, M. Worring, J.C. van Gemert, J.-M. Geusebroek, and A.W.M. Smeulders. The Challenge Problem for Automated Detection of 101 Semantic Concepts in Multimedia. In Proceedings of ACM Multimedia, pp. 421-430, Santa Barbara, USA, October 2006.

C.-K. Yeh, W.-C. Wu, W.-J. Ko, Y.-C.F. Wang, Learning deep latent space for multi-label classification, AAAI Conference on Artificial Intelligence, (2017)