AffinityNet: Semi-Supervised Few-Shot Learning for Disease Type Prediction

Tianle Ma, Aidong Zhang
Department of Computer Science and Engineering
State University of New York at Buffalo
Buffalo, New York 14260
{tianlema, azhang}@buffalo.edu

Abstract

While deep learning has achieved great success in computer vision and many other fields, currently it does not work very well on patient genomic data with the "big p, small N" problem (i.e., a relatively small number of samples with high-dimensional features). In order to make deep learning work with a small amount of training data, we have to design new models that facilitate few-shot learning. Here we present the Affinity Network Model (AffinityNet), a data efficient deep learning model that can learn from a limited number of training examples and generalize well. The backbone of the AffinityNet model consists of stacked k-Nearest-Neighbor (kNN) attention pooling layers. The kNN attention pooling layer is a generalization of the Graph Attention Model (GAM), and can be applied to not only graphs but also any set of objects regardless of whether a graph is given or not. As a new deep learning module, kNN attention pooling layers can be plugged into any neural network model just like convolutional layers. As a simple special case of kNN attention pooling layer, feature attention layer can directly select important features that are useful for classification tasks. Experiments on both synthetic data and cancer genomic data from TCGA projects show that our AffinityNet model has better generalization power than conventional neural network models with little training data.

Introduction

Patients, drugs, networks, etc., are all complex objects with heterogeneous features or attributes. Complex object clustering and classification are ubiquitous in real world applications. For instance, it is important to cluster cancer patients into subgroups and identify disease subtypes in cancer genomics (Shen et al. 2012; Wang et al. 2014; Ma and Zhang 2017). Compared with images, which have homogeneous structured features (i.e., pixels are arranged in a 3-D array as raw features), complex objects usually have heterogeneous features with unclear structures. Deep learning models such as Convolutional Neural Networks (CNNs) widely used in computer vision (LeCun, Bengio, and Hinton 2015; Krizhevsky, Sutskever, and Hinton 2012) and other fields (Bahdanau, Cho, and Bengio 2014; Sutskever, Vinyals, and Le 2014; Silver et al. 2016; Banino et al. 2018) cannot be directly applied to complex objects whose features are not ordered structurally.

One critical challenge in cancer patient clustering problem is the "big p, small N" problem: we only have a relatively small number of samples (i.e., patients) compared with high-dimensional features each sample has. In other words, we do not have an "ImageNet"(Russakovsky et al. 2015) to train deep learning models that can learn good representations from raw features. Moreover, unlike pixels in images, patient features such as gene expressions are much noisier and more heterogeneous. These features are not "naturally" ordered. Thus we cannot directly use convolutional neural networks with small filters to extract abstract local features.

For a clustering/classification task, nodes/objects belonging to the same cluster should have similar representations that are near the cluster centroid. Based on this intuition we developed the **k-nearest-neighbor (kNN) attention pooling** layer, which applies the attention mechanism to learning node representations. With the kNN attention pooling layer, each node's representation is decided by its k-nearest neighbors as well as itself, ensuring that similar nodes will have similar learned representations. Similar to Graph Attention Model (GAM) (Veličković et al. 2017), we propose the Affinity Network Model (AffinityNet) that consists of stacked kNN attention pooling layers to learn the deep representations of a set of objects. While GAM is designed to tackle representation learning on graphs (Veličković et al. 2017; Hamilton, Ying, and Leskovec 2017b) and it does not directly apply to data without a known graph, our AffinityNet model generalizes GAM to facilitate representation learning on any collections of objects with or without a known graph.

In addition to learning deep representations for classifying objects, feature selection is also important in biomedical research. Though the number of features (i.e., variables or covariates) in genomic data is usually very high, many features may be irrelevant to a specific task. For instance, a disease may only have a few risk factors involving a small number of features. In order to facilitate feature selection in a "deep learning" way, we propose a **feature attention** layer, a simple special case of the kNN attention pooling layer which can be incorporated into a neural network model and directly learn feature weights using backpropagation.

We performed experiments on both synthetic and real cancer genomics data. The results demonstrated that our AffinityNet model has better generalization power than conventional neural network models for few-shot learning.

Related work

kNN attention pooling layer is related to graph learning (Hamilton, Ying, and Leskovec 2017b; Kipf and Welling 2016; Veličković et al. 2017), attention model (Vaswani et al. 2017; Veličković et al. 2017), pooling and normalization layers (Ioffe and Szegedy 2015). In graph learning, a graph has a number of nodes and edges (both nodes and edges can have features). When available, combining node features with graph structure can do a better job than using node features alone. For example, Graph Convolutional Network (Kipf and Welling 2016) incorporates graph structure (i.e., edges) into the learning process to facilitate semi-supervised few-shot learning. Graph Attention Model (GAM) (Veličković et al. 2017) learns a representation for each node based on the weighted pooling (i.e., attention) of its neighborhood in the given graph, and then performs classification using the learned representations. However, all these graph learning algorithms require that a graph is known. Many algorithms also require the input to be the whole graph (Veličković et al. 2017), and thus do not scale well to large graphs. Our proposed AffinityNet model generalizes graph learning to a collection of objects with or without known graphs.

As the key component of AffinityNet, kNN attention pooling layer is also related to normalization layers in deep learning, such as batch normalization (Ioffe and Szegedy 2015), instance normalization (Jing et al. 2017), or layer normalization (Ba, Kiros, and Hinton 2016). All these normalization layers use batch statistics or feature statistics to normalize instance features, while kNN attention pooling layers apply the attention mechanism to the learned instance representations to ensure similar instances will have similar representations.

kNN attention pooling layer is different from the existing max or average pooling layers used in deep learning models, where features in a local neighborhood are pooled to extract the signal and reduce feature dimensions. Our proposed kNN attention pooling layer applies pooling on node representations instead of individual features. kNN attention pooling layer combines normalization, attention and pooling, making it more general and powerful. It can serve as an implicit regularizer to make the network generalize well for semi-supervised few-shot learning.

Affinity Network Model (AffinityNet)

One key ingredient for the success of deep learning is its ability to learn a good representation (Bengio, Courville, and Vincent 2013) through multiple complex nonlinear transformations. For classification tasks, the learned representation (usually the last hidden layer) is often linearly separable for different classes. If the output layer is a fully connected layer for classification, then the weight matrix for the last layer can be seen as the class centroids in the transformed feature space. While conventional deep learning models often perform well when lots of training data is available, our goal is to design new models that can learn a good feature transformation in a transparent and data efficient way. We developed the kNN attention pooling layer, and used it to construct the AffinityNet Model. In a typical AffinityNet model as shown in Fig. 1, the input layer is followed by a feature attention layer (a simple special case of kNN attention pooling layer used for raw feature selection), and then followed by multiple stacked kNN attention pooling layers (Fig. 1 only illustrates one kNN attention pooling layer). The output of the last kNN attention pooling layer will be the newly learned network representations, which can be used for classification or regression tasks. Though it is possible to train AffinityNet with only a few labeled examples, it is more advantageous to use it as a semi-supervised learning framework (i.e., using both labeled and unlabeled data during training).

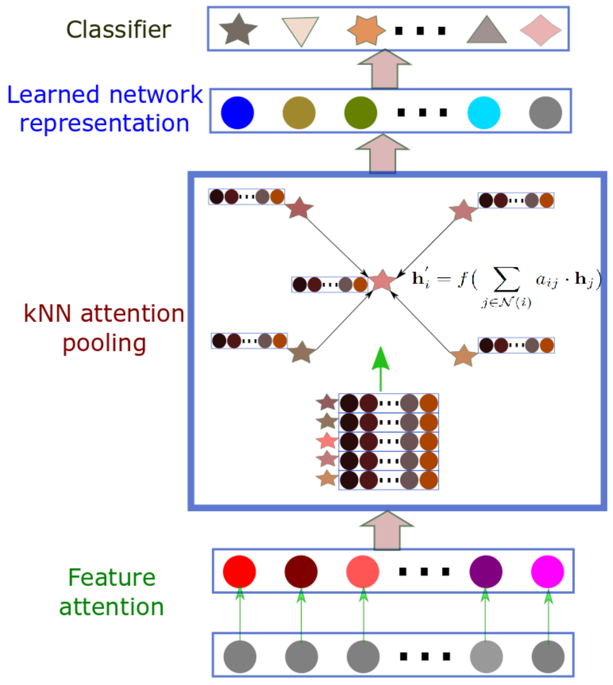

Classifier
Learned network representation
kNN attention pooling
$\mathbf{h}_i' = f(\sum_{j\in\mathcal{N}(i)} a_{ij}\cdot\mathbf{h}_j)$
Feature attention

Figure 1: AffinityNet model overview

As the main components of AffinityNet are stacked kNN attention pooling layers, we describe it in detail in the following section.

kNN attention pooling layer

A good classification model should have the ability to learn a feature transformation such that objects belonging to the same class have similar representations which are near the class centroid in the transformed feature space.

As an object's k-nearest neighbors should have similar feature representations, we propose the kNN attention pooling layer to incorporate neighborhood information using attention-based pooling (Eq. 1):

$$\mathbf{h}_i' = f\Big(\sum_{j\in\mathcal{N}(i)} a(\mathbf{h}_i,\mathbf{h}_j)\cdot\mathbf{h}_j\Big) \tag{1}$$

In Eq. 1, $\mathbf{h}_i$ and $\mathbf{h}_i^{'}$ are input feature representations and transformed feature representations for object i, respectively. $\mathcal{N}(i)$ represents the neighborhood of object i. If a graph is given, we can use the given graph to determine the neighborhood. If the given graph is very large with a high degree, in order to reduce the computational cost, we can randomly sample k (k is a fixed small number) neighbors for computing Eq. 1 (Hamilton, Ying, and Leskovec 2017a). In the kNN attention pooling layer, k is a hyperparameter that determines how many neighbors are used for calculating the representation of a node. $f(\cdot)$ is a nonlinear transformation, for example, an *affine* layer with weight $\mathbf{W}$ and bias $\mathbf{b}$ followed by *ReLU()* nonlinear activation:

$$f(\mathbf{h}) = max(\mathbf{W}\mathbf{h} + \mathbf{b}, \mathbf{0}) \tag{2}$$

$a_{ij} = a(\mathbf{h}_i, \mathbf{h}_j)$ in Eq. 1 is the normalized attention from object i to object j. $a(\cdot, \cdot)$ is the attention kernel that will be discussed in the next section.

Attention kernels Intuitively, if two objects are similar, their feature representations should be near each other. Objects belonging to the same class should be clustered together in the learned feature space. In order to achieve this, kNN attention pooling layer uses weighted pooling to "attract" similar objects together in the transformed feature space. Attention kernels essentially calculate the similarities among objects to facilitate weighted pooling.

There are many choices of attention kernels. For example:

- Cosine similarity:

$$\alpha_{ij} = \frac{\mathbf{h}_i \cdot \mathbf{h}_j}{||\mathbf{h}_i|| \cdot ||\mathbf{h}_j||} \tag{3}$$

- Inner product (Vaswani et al. 2017):

$$\alpha_{ij} = \mathbf{h}_i \cdot \mathbf{h}_j \tag{4}$$

- Perceptron affine kernel (Veličković et al. 2017):

$$\alpha_{ij} = \mathbf{w}^T \cdot (\mathbf{h}_i || \mathbf{h}_j) \tag{5}$$

- Inverse distance with weighted L_2 norm ($\mathbf{w}$ is the feature weight):

$$\alpha_{ij} = -||\mathbf{w} \odot \mathbf{h}_i - \mathbf{w} \odot \mathbf{h}_j||^2 \tag{6}$$

In order to calculate a weighted average of new representations, we can use the *Softmax* function to normalize the attention (other normalization is also feasible). Therefore the normalized attention kernel is:

$$a_{ij} = a(\mathbf{h}_i, \mathbf{h}_j) = \frac{e^{\alpha_{ij}}}{\sum_{j \in \mathcal{N}(i)} e^{\alpha_{ij}}} \tag{7}$$

If the graph is not given, in order to determine $\mathcal{N}(i)$, we can use attention kernel to calculate an affinity/similarity graph (i.e., the similarities among all the objects), and then use this affinity graph to decide the neighborhood $\mathcal{N}(i)$. As an additional regularizer, we can use one type of affinity kernels to calculate the affinity graph and another to compute the normalized attention.

Layer-specific dynamic affinity graph The kNN attention pooling layer can be applied to a collection of objects regardless of whether a graph (e.g., links among objects) is given or not. If a graph is given, we can directly use the graph to determine the neighborhood in Eq. 1 and Eq. 7, which is the same as in Graph Attention Model (Veličković et al. 2017). If the degree of the graph is too high, and some nodes have very large neighborhoods, then we can select only k nearest neighbors for calculating the attention when the computational cost is a big concern. Regardless of whether a graph is given or not, we can always calculate an affinity graph $\mathbf{G}_n$ based on node features using some similarity metric including the aforementioned attention kernels. As our AffinityNet model contains multiple kNN pooling layers stacked together, we can calculate a layer-specific dynamic affinity graph using the learned node feature representations from each layer during training.

Also, we can use the graph calculated using features from the previous layer to determine the k-nearest-neighborhood for the next layer. This can be seen as an implicit regularizer preventing the learned representation from drifting away from the previous layer too much in a single layer operation. Mathematically, for layer l, we can calculate a layer-specific dynamic affinity graph $\mathbf{G}^{(l)}$ using Eq. 8.

$$\mathbf{G}^{(l)} = \lambda \mathbf{G}_e + (1-\lambda)\big(\eta \mathbf{G}_n^{(l)} + (1-\eta)\mathbf{G}_n^{(l-1)}\big) \tag{8}$$

In Eq. 8, $\mathbf{G}_e$ is the given graph if available. When not available, we can simply set $\lambda = 0 (0 \leq \lambda \leq 1)$. $\mathbf{G}_n^{(l)}$ and $\mathbf{G}_n^{(l-1)}$ are the node-feature-derived affinity graphs for the current layer l and the previous layer $l-1$, respectively. We can combine $\mathbf{G}_n^{(l-1)}$ and $\mathbf{G}_n^{(l)}$ with a parameter $\eta, 0 \leq \eta \leq 1$.

If the input of the AffinityNet model consists of N objects, then we will learn dynamic affinity graphs for these N objects during training. After training, the final learned affinity graph from the last layer can also be used for spectral clustering (affinity graphs calculated using higher-level features may be more informative for separating different classes). In this sense, we also call our framework affinity network learning.

Semi-supervised few-shot learning Semi-supervised few-shot learning (Ravi and Larochelle 2017; Kingma et al. 2014; Kipf and Welling 2016; Rasmus et al. 2015) only allows using very few labeled instances to train a model and requires the model to generalize well. It is especially useful for cancer patient clustering problems, where we usually have only several hundred patients in a study. If we can obtain a few labeled training examples (for example, human experts can manually assign labels for some patients), we can use the AffinityNet model for semi-supervised learning. The input of the AffinityNet model is the patient-feature matrix consisting of all patients, and the output of the model is the newly learned patient representations as well as class labels. We only backpropagate the classification error for those labeled patients. Different from conventional neural network models where each instance is independently trained without batch normalization (Ioffe and Szegedy 2015), AffinityNet can utilize unlabeled instances for calculating kNN attention-based representations in the whole sample pool. In a sense, the kNN attention pooling layer performs both nonlinear transformation and "clustering" (attracting similar instances together in the learned feature space) during training. Even though the labels of most patients are unknown, their feature representations can be used for learning a global affinity graph, which is useful to cluster or classify all patients in the cohort.

Our AffinityNet model can also be used for data distillation (Radosavovic et al. 2017). We can train a few examples with true labels, and use our learned model to generate some noisy labels for unlabeled data. Then we can train our model with both clean and noisy labels and repeat this process iteratively. When dealing with very large graphs, we can feed a small batch of instances (i.e., a partial graph) at a time to the AffinityNet model to reduce the computational burden. Though each batch may contain different instances, the kNN pooling layer can still work well with the attention mechanism. Our PyTorch implementation of AffinityNet can even handle the extreme case where only one instance is fed into the model at a time, in which case the AffinityNet model operates as a conventional deep learning model to only learn a nonlinear transformation without kNN attention pooling operation.

Feature Attention Layer

Deep neural networks can learn good hierarchical local feature extractors (such as convolutional filters or inception modules (Szegedy et al. 2017)) automatically through gradient descent. Local feature operations such as convolutions require features to be ordered structurally. For images or videos, pixels near each other naturally form a neighborhood. However, in other applications, features are not ordered and the structural relations among features are unknown. Therefore we cannot directly learn a local feature extractor. Instead, we have to learn a feature selector that can select important individual features.

In addition, there can be many redundant, noisy, or irrelevant features, and the Euclidean distance between objects using all the features may be dominated by the irrelevant ones (Bellet, Habrard, and Sebban 2013). However, with proper feature weighting, we can separate objects from different classes well. This motivates us to develop a feature attention layer as a simple special case of kNN attention pooling layer.

Let $\mathbf{h}_i \in \mathbb{R}^p$ be the feature vector of object i, and $\mathbf{w} \in \mathbb{R}^p$ be the feature attention (i.e., weight), satisfying

$$\mathbf{w} = (w_1, w_2, \cdots, w_p), \quad \sum_{j=1}^{p} w_j = 1, w_j \geq 0 \tag{9}$$

Instead of the commonly used *affine* transformation followed by *ReLU()* nonlinearity as in Eq. 2, the feature attention layer performs element-wise multiplication (Eq. 10, $\odot$ is element-wise multiplication operator) with the weight constraint (Eq. 9). This is the only difference between the feature attention layer and the kNN attention pooling layer.

$$f(\mathbf{h}_i) = \mathbf{w} \odot \mathbf{h}_i \tag{10}$$

$$d_{ij} = ||\mathbf{h}_i - \mathbf{h}_j|| \tag{11}$$

$$d_{ij}^{'} = ||f(\mathbf{h}_i) - f(\mathbf{h}_j)|| = ||\mathbf{w} \odot \mathbf{h}_i - \mathbf{w} \odot \mathbf{h}_j||^2 \tag{12}$$

Before transformation, the learned distance between object i and j is d_{ij} (Eq. 11), which can be skewed by noisy and irrelevant features. After transformation, the distance $d_{ij}^{'}$ (Eq. 12) can be more informative for classification tasks. Note the kNN attention pooling (Eq. 1) is still used after the feature transformation (Eq. 10). The main difference between the feature attention layer and the kNN pooling layer is that the feature attention layer uses element-wise multiplication (Eq. 10) instead of *affine* layer followed by *ReLU()* (Eq. 2) as nonlinear transformation. Just like skip connections in ResNet (He et al. 2016) that can help gradient flow, the feature attention layer can help select important *individual* features much easier than the fully connected layer, and can increase the generalization power of a neural network model in certain cases with very few training examples.

In addition, for fully connected *affine* layer without weight constraints, the weight can be negative and unbounded. Even if we set non-negativity constraints to the weight, the transformed features are linear combinations of the input features. We cannot directly determine the importance of individual features. By contrast, the feature attention layer only has parameter w (Eq. 9), which directly corresponds to the learned feature weight. Because of the constraint on w (Eq. 9), the feature attention layer also learns a weighted Euclidean metric during training.

Experiments

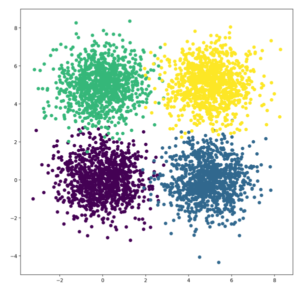
Simulations

We sampled 1000 points from each of the four 2-dimensional Gaussian distributions with the same covariance matrix $\Sigma = diag(1, 1)$ and four different mean ($\mu = (0, 0), (0, 5), (5, 0), (5, 5)$, respectively) as the true signal. We then appended the true signal with 40-dimensional Gaussian noise with mean $\mu = (2.5, 2.5, \cdots, 2.5)$ and covariance $\Sigma = diag(10, 10, \cdots, 10)$. Thus each point has 42 dimensions, with the first two containing the true signal, and the rest being random noise. With four different colors corresponding to the true cluster assignments (generated from four distributions), we plotted the true signal (i.e., the first two dimensions) in Fig. 2a and the "corrupted" signal (i.e., 42-dimensional vector) using PCA in Fig. 2b. While the true signal forms four "natural" clusters (Fig. 2a), the corrupted signal is dominated by the added irrelevant features and the clusters are no longer obvious.

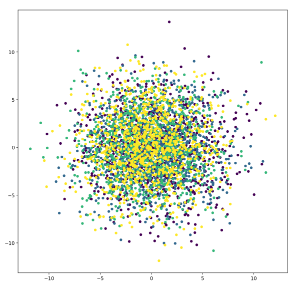
We constructed two models to predict class labels:

"NeuralNet": a neural network model with an input layer (42-dimensional), a hidden layer (100 hidden units) and an output layer (4 units corresponding to four classes);

"AffinityNet": same as "NeuralNet" model except adding one feature attention layer followed by kNN attention pooling after the input layer.



(a) 4000 points belonging to four "natural" clusters

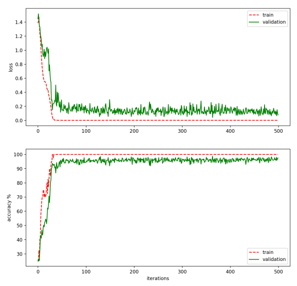


(b) After adding 40-dimensional Gaussian noise

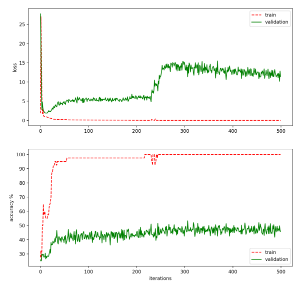
Figure 2: Plots of the true signal and the "corrupted" signal

We randomly selected 1% of data (40 out of 4000 points) for training two models and compared accuracies on the test set. Surprisingly, by only training 1% of the data, our model with feature attention layer can successfully select the true signal features and achieve 98.2% accuracy on the test set. By contrast, a plain neural network model only achieved 46.9% accuracy on the test set. Fig. 3a and Fig. 3b show the training loss and accuracy curves (the red curves are for training set and the green ones for test set) for "AffinityNet" and "NeuralNet", respectively. Even though both models achieve 100% training accuracy within a few iterations, the "AffinityNet" model generalizes better than the plain neural network model (there is a big gap between training and test accuracy curves for "NeuralNet" model when training data is small).

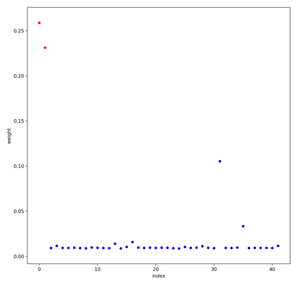
Strikingly, the good generalization of our model partly relies on the success of the feature attention layer picking up the true signals from the noise. Fig. 3c and Fig. 3d shows the learned weights by AffinityNet and NeuralNet for the 42-dimensional input features, with the red dots corresponding to the true signals and blue dots noise. The weights of the true signal are much higher than those noise in AffinityNet, while "NeuralNet" did not select the true signal very well.



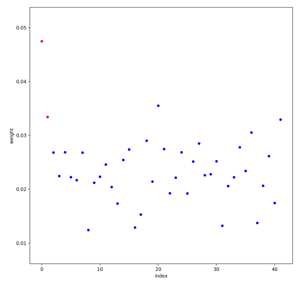
(a) AffinityNet training loss and accuracy



(b) NeuralNet training loss and accuracy



(c) feature weights learned by AffinityNet



(d) feature weights learned by NeuralNet

Figure 3: Training loss and accuracy and learned feature weights

Tumor disease type classification

Harmonized kidney and uterus cancer gene expression datasets were downloaded from Genomic Data Commons Data Portal (https://portal.gdc.cancer.gov) (Grossman et al. 2016). We preprocessed the data and selected top 1000 most variant gene expression features as the model input. Kidney cancer has 654 samples with three disease types, and uterus cancer has 475 samples with two disease types. Both kidney cancer and uterus cancer have highly unbalanced classes. We are trying to classify each tumor sample into its disease type for uterus and kidney cancer separately using the gene expression profiles.

We compared our model ("AffinityNet") with five other methods: "NeuralNet" (conventional deep learning model), "SVM", "Naive Bayes", "Random Forest", and "Nearest Neighbors" (kNN). Our model ("AffinityNet") consists of a feature attention layer, a kNN attention pooling layer (100 hidden units), and a fully connected layer. For kNN attention pooling layer, we use "cosine similarity" kernel and set the number of nearest neighbors $k = 2$ (kidney cancer) and $k = 3$ (uterus cancer). (We have tried other choices of k and the results are similar.) "NeuralNet" is a two-layer fully connected neural network with the hidden layer having 100 hidden units. For both "AffinityNet" and "NeuralNet", we use *ReLU()* nonlinear activation in the hidden layer. Since the input dimension is 1000 (i.e., top 1000 most variant gene expressions), the total number of the parameters of "NeuralNet" is 100,403 for kidney cancer with three classes (i.e., disease types), and 100,202 for uterus cancer with two classes. Our model "AffinityNet" has 101,403 parameters and 101,202 parameters for kidney and uterus cancer, respectively. Note our model only has 1000 more parameters than "NeuralNet" to facilitate fair comparisons. We do not use more layers in the neural network models because there are only several hundred samples to train, and larger models are more likely to overfit. We used the implementation from scikit-learn (http://scikit-learn.org) for "Naive Bayes", "SVM", "Nearest Neighbors", and "Random Forest" with default settings.

We progressively increased the training portion from 1% to 50% (i.e., 1%, 10%, 20%, 30%, 40%, and 50%), and reported the adjusted mutual information (AMI) on the test set (Table 1 and Table 2). AMI is an adjustment of the Mutual Information (MI) score to account for chance, which is suitable to measure the performance of clustering and classification with multiple unbalanced classes (AUC is a similar metric but is mainly suitable for binary classification).

We ran experiments 20 times with different random seeds to generate different training and test sets. For each run, the

Table 1: Adjusted Mutual Information on the test set for kidney cancer

	Train portion					
Method	0.01	0.1	0.2	0.3	0.4	0.5
AffinityNet	**0.84**	**0.87**	**0.86**	**0.85**	**0.85**	**0.85**
NeuralNet	0.70	0.76	0.77	0.78	0.78	0.80
SVM	0.70	0.77	0.78	0.79	0.80	0.81
Naive Bayes	0.25	0.59	0.71	0.76	0.78	0.80
Random Forest	0.36	0.61	0.67	0.68	0.71	0.72

Table 2: Adjusted Mutual Information on the test set for uterus cancer

	Train portion					
Method	0.01	0.1	0.2	0.3	0.4	0.5
AffinityNet	**0.62**	**0.66**	**0.76**	**0.84**	**0.85**	**0.84**
NeuralNet	0.41	0.52	0.59	0.61	0.63	0.68
SVM	0.43	0.54	0.60	0.62	0.65	0.70
Naive Bayes	0.00	0.28	0.62	0.58	0.55	0.58
Random Forest	0.03	0.06	0.10	0.12	0.18	0.14

training and test set for all six methods are identical. We reported the mean AMI scores for the top ten runs (results depending on the few selected training examples and other randomness) for all methods in Table 1 and Table. 2.

For both cancer types, our model clearly outperformed all other models, especially when the training portion is small. For example, when trained on only 1% of the data, our model achieved AMI=0.84 for kidney cancer and AMI=0.62 for uterus cancer (Table 1 and Table 2), while other methods performed badly with few training examples. This suggests our model is highly data efficient. One reason for this is that kNN attention pooling layer is in a sense performing "clustering" during training, and it is less likely to overfit a small number of training examples. The input of kNN attention pooling layers can contain not only labeled training examples but also unlabeled examples. It performs semi-supervised learning with a few labeled examples as a guide for finding "clusters" among all the data points. "NeuralNet" and other methods do not perform well with few labeled training examples because they tend to overfit the training set and cannot generalize well with a small training set. In these experiments, "NeuralNet" model does not outperform "SVM" because the dataset is quite small and the power of deep learning is manifested only when large amounts of data is available. For kidney cancer, unlike other methods, our model did not improve with more training data, partly because there are a few very hard cases in kidney cancer dataset, while all other cases are almost linearly separable. Our model can easily pick up the linearly separable clusters with only a few training examples, but it is hard to separate very hard cases even when more training data is available. Besides, we analyzed the top genes with the highest learned weights selected by the feature attention layer for kidney cancer. Surprisingly, we found literature evidence supporting that seven out of the top ten genes are relevant to kidney cancer.

Semi-supervised clustering

Cancer patient clustering and disease subtype discovery are very challenging because of the small sample size and lack of enough training examples with groundtruth labels. If we can obtain label information for a few samples, we can use "AffinityNet" for semi-supervised clustering (Weston et al. 2012). While other methods such as SVM do not produce explicit feature representations, both "AffinityNet" and "NeuralNet" can learn a new feature representation through multiple nonlinear transformations. For a classification model, the new feature representation is usually fed into a linear classifier. We can train our model with a few labeled examples, use the learned model to generate the transformed feature representations for all data points, and then perform clustering using the transformed features.

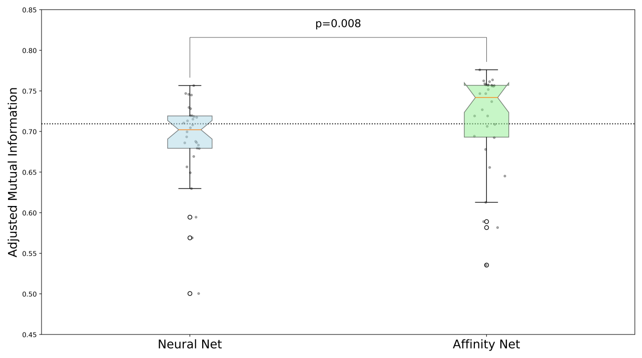


Figure 4: Adjusted Mutual Information achieved by semi-supervised clustering using "NeuralNet" and "AffinityNet" for kidney cancer

For "AffinityNet", we can use all the data points during training with kNN attention pooling, but only backpropagate on labeled training examples. We get the learned new representations for all the data points once the training process is finished. For conventional neural network models, since each data point is independently trained, we only use labeled examples during training. After training, we have to use the learned model to generate new feature representations for all the data points. In order to evaluate the quality of the learned feature representations with a few training examples, we performed clustering using these transformed features and using the original features, and compared them with groundtruth class labels.

We compared the performance using "AffinityNet" and "NeuralNet" on kidney data set as it has more samples. We randomly selected 1% of data for training, and ran experiments 30 times. After training, we performed spectral clustering on the learned patient-feature matrices. Fig. 4 shows the adjusted mutual information scores for all the 30 runs using "AffinityNet" and "NeuralNet". We also performed spectral clustering on the original patient-feature matrix as a baseline method (AMI = 0.71, blue dotted line in the figure). Our model outperformed the "NeuralNetwork" model

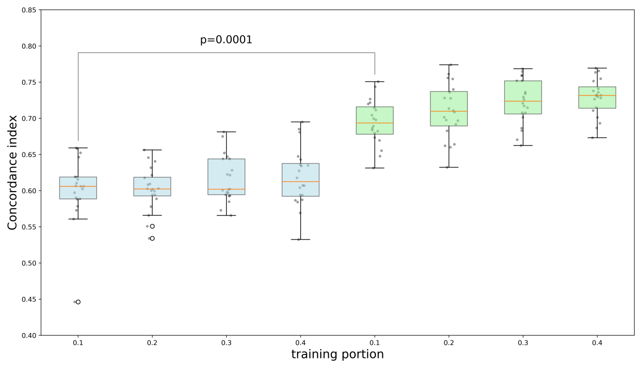


Figure 5: Concordance index achieved by "AffinityNet" and baseline Cox model for kidney cancer

($p = 0.008$, Wilcoxon signed rank test) and the baseline (the "Neural Network" model is slightly below the baseline because it probably had overfitted the training examples). While both "NeuralNet" and "AffinityNet" have approximately the same number of model parameters, only "AffinityNet" can learn a good feature transformation by facilitating semi-supervised few-shot learning with feature attention and kNN attention pooling layers.

Combine with Cox model for survival analysis

For many cancer genomics studies, cancer subtype information is not known, but patient survival information is available. We replaced the last layer (i.e., linear classifier) in the model (as shown in Fig. 1) with a regression layer following the Cox proportional hazards model (Mobadersany et al. 2018; Fox 2002). We used backpropagation to learn model parameters that maximize partial likelihood in the Cox model.

We performed experiments on kidney cancer dataset that has more than 600 samples. We progressively increased the training portion from 10% to 40%. We used 30% of data as validation and the remaining as the test set. As a baseline method, we used age, gender and known disease types as covariates to fit a Cox model. We ran experiments 20 times with random seeds, and reported the concordance index on the test set for both our model and the baseline Cox model (Fig. 5).

In Fig. 5, the light blue boxplots on the left side correspond to the results from the baseline method (i.e., the Cox model on age, gender and disease types), while the light green ones correspond to that from our model. The reported p-value between our model and the baseline method for training 10% data was calculated using the Wilcoxon signed rank test. Our model outperformed the baseline model by a significant margin (Table. 3 shows the mean concordance index in different settings).

Table 3: Mean concordance scores for kidney cancer

	Train portion			
Method	0.1	0.2	0.3	0.4
Baseline Cox Model	0.601	0.602	0.616	0.618
AffinityNet	0.694	0.710	0.723	0.729

Discussion and Conclusion

Deep learning has achieved great success in computer vision, natural language processing, and speech recognition, where features (e.g., pixels, words, and audio signals) are well structured and a large amount of training data is available. However, in biomedical research, the training sample size is usually small while the feature dimension is very high, where deep learning models tend to overfit the training data but fail to generalize. To alleviate this problem in the patient clustering/classification related tasks, we propose the AffinityNet model that contains stacked feature attention and kNN attention pooling layers to facilitate semi-supervised few-shot learning.

Regardless of whether a graph is given or not, kNN attention pooling layer can use attention kernels to calculate dynamic affinity graphs during training. The affinity graphs are used for selecting k-nearest neighbors for attention-based pooling. kNN attention pooling layers essentially add a "clustering" operation ("forcing" similar objects to have similar representations through attention-based pooling) after the nonlinear feature transformations, which can serve as an implicit regularizer for classification-related tasks. kNN attention pooling layers can be plugged into a deep learning model as a basic building block just like convolutional layers. With multi-view data, we can first use a few kNN attention pooling layers to process each view separately to learn a high-level representation for each view, and then combine all the views with their high-level feature representations (by concatenating them together or adding them up) and apply kNN attention pooling again to the combined view. Feature attention layer is a simple special case of kNN attention pooling layer. It is useful for selecting important individual input features automatically with a normalized non-negative weight learned for each feature.

Building upon stacked feature attention and kNN pooling layers, our AffinityNet model is more effective for semi-supervised few-shot learning than conventional deep learning models. We have conducted extensive experiments using AffinityNet on two cancer genomics datasets and achieved satisfactory results.

AffinityNet alleviates the problem of lack of a sufficient amount of labeled training data by utilizing unlabeled data with kNN attention pooling, and can be used to analyze a large bulk of cancer genomics data for patient clustering and disease subtype discovery. Future work may focus on designing deep learning modules that can incorporate biological knowledge for various tasks.

References

Ba, J. L.; Kiros, J. R.; and Hinton, G. E. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Bahdanau, D.; Cho, K.; and Bengio, Y. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Banino, A.; Barry, C.; Uria, B.; Blundell, C.; Lillicrap, T.; Mirowski, P.; Pritzel, A.; Chadwick, M. J.; Degris, T.; Modayil, J.; et al. 2018. Vector-based navigation using grid-like representations in artificial agents. *Nature*.

Bellet, A.; Habrard, A.; and Sebban, M. 2013. A survey on metric learning for feature vectors and structured data. *arXiv preprint arXiv:1306.6709*.

Bengio, Y.; Courville, A.; and Vincent, P. 2013. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence* 35(8):1798–1828.

Fox, J. 2002. Cox proportional-hazards regression for survival data. *An R and S-PLUS companion to applied regression* 2002.

Grossman, R. L.; Heath, A. P.; Ferretti, V.; Varmus, H. E.; Lowy, D. R.; Kibbe, W. A.; and Staudt, L. M. 2016. Toward a shared vision for cancer genomic data. *New England Journal of Medicine* 375(12):1109–1112.

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017a. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, 1025–1035.

Hamilton, W. L.; Ying, R.; and Leskovec, J. 2017b. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Ioffe, S., and Szegedy, C. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, 448–456.

Jing, Y.; Yang, Y.; Feng, Z.; Ye, J.; and Song, M. 2017. Neural style transfer: A review. *arXiv preprint arXiv:1705.04058*.

Kingma, D. P.; Mohamed, S.; Rezende, D. J.; and Welling, M. 2014. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*, 3581–3589.

Kipf, T. N., and Welling, M. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

LeCun, Y.; Bengio, Y.; and Hinton, G. 2015. Deep learning. *nature* 521(7553):436.

Ma, T., and Zhang, A. 2017. Integrate multi-omic data using affinity network fusion (anf) for cancer patient clustering. *arXiv preprint arXiv:1708.07136*.

Mobadersany, P.; Yousefi, S.; Amgad, M.; Gutman, D. A.; Barnholtz-Sloan, J. S.; Velázquez Vega, J. E.; Brat, D. J.; and Cooper, L. A. D. 2018. Predicting cancer outcomes from histology and genomics using convolutional networks. *Proceedings of the National Academy of Sciences*.

Radosavovic, I.; Dollár, P.; Girshick, R.; Gkioxari, G.; and He, K. 2017. Data distillation: Towards omni-supervised learning. *arXiv preprint arXiv:1712.04440*.

Rasmus, A.; Berglund, M.; Honkala, M.; Valpola, H.; and Raiko, T. 2015. Semi-supervised learning with ladder networks. In *Advances in Neural Information Processing Systems*, 3546–3554.

Ravi, S., and Larochelle, H. 2017. Optimization as a model for few-shot learning. *International Conference on Learning Representations (ICLR)*.

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; Berg, A. C.; and Fei-Fei, L. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115(3):211–252.

Shen, R.; Mo, Q.; Schultz, N.; Seshan, V. E.; Olshen, A. B.; Huse, J.; Ladanyi, M.; and Sander, C. 2012. Integrative subtype discovery in glioblastoma using icluster. *PloS one* 7(4):e35236.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of go with deep neural networks and tree search. *nature* 529(7587):484–489.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, 3104–3112.

Szegedy, C.; Ioffe, S.; Vanhoucke, V.; and Alemi, A. A. 2017. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, 12.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, 6000–6010.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; and Bengio, Y. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Wang, B.; Mezlini, A. M.; Demir, F.; Fiume, M.; Tu, Z.; Brudno, M.; Haibe-Kains, B.; and Goldenberg, A. 2014. Similarity network fusion for aggregating data types on a genomic scale. *Nature Methods* 11:333.

Weston, J.; Ratle, F.; Mobahi, H.; and Collobert, R. 2012. Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*. Springer. 639–655.