Towards Network Implementation of CBR: Case Study of a Neural Network K-NN Algorithm

Xiaomeng Ye[1], David Leake[0000−0002−8666−3416][2], Yu Wang[2], Ziwei Zhao[2], and David Crandall[2]

[1] Dept. of Mathematics and Computer Science
Berry College, Mount Berry, GA, 30149, USA
[2] Luddy School of Informatics, Computing, and Engineering
Indiana University, Bloomington IN 47408, USA
xye@berry.edu,{leake, yw173, zz47, djcran}@iu.edu

Abstract. Recent research brings the strengths of neural networks to bear on CBR tasks such as similarity assessment and case adaptation. This paper further advances this direction by implementing both retrieval and adaptation as a single neural network. Such an approach has multiple goals: From the perspective of CBR, it enables harmonizing the interaction between feature extraction, retrieval/similarity assessment, and case adaptation through end-to-end training. From the perspective of neural networks, a neural network implementing CBR processes ceases to be a black box and provides the natural interpretability of CBR. As a first step towards this goal, this paper presents *neural network based k-nearest neighbor* (NN-kNN), a network architecture that can be interpreted as a k-NN method. Unlike other network architectures, NN-kNN's decisions can be fully explained in terms of surface features, feature/case weights and nearest neighbors. It can be trained or fine-tuned using existing neural network methods. This study illustrates its feasibility and examines its strengths and limitations. The approach is evaluated for classification and regression tasks comparing NN-kNN, a standard neural network, and k-NN models using state-of-the-art distance metric learning algorithms. In these tests, NN-kNN achieves equal or less error when compared to the other models, while being fully interpretable as a k-NN method. The study also considered the limitations of NN-kNN and future directions to alleviate them.

1 Introduction

Neural network (NN) techniques are powerful but often uninterpretable black boxes. Tremendous effort has been devoted to opening the black box with post-hoc explanation [11]. As observed since the early days of CBR research, case-based reasoning provides interpretability [17], suggesting benefits of hybrid systems where a CBR system provides cases as explanations for network decisions [20]. However, Rudin [27] contends that post-hoc explanations have fundamental flaws and that intrinsically interpretable methods are needed for critical tasks.

An alternative to abandoning networks or relying on post-hoc explanation is to develop network architectures that are more interpretable. An interesting approach to this is to design network architectures so that their processes parallel CBR [20]. Li et al. [22] propose a neural network model in which predictions are based on learned prototypes, which that can subsequently be used to explain network decisions. Their model uses an autoencoder to encode features and reconstruct samples from features, and a special prototype layer which stores the features of prototypes found through training. The prototype layer calculates the distance between the input and each prototype, which then feeds to a fully connected layer for a final classification output. The model is trained with a loss function that encourages (1) high classification accuracy, (2) faithful reconstruction from the autoencoder, (3) every prototype being similar to at least one encoded input and every encoded input being similar to at least one prototype. In summary, the model learns prototypes in a low dimensional space, which are then used in distance-based classification.

Their approach enables explaining using learned prototypes and achieves strong accuracy. However, it has limitations as well. First, the system learns the prototypes through an autoencoder, which is a black box itself. Second, there is no guarantee that sufficiently similar prototypes exist to explain all or most of the possible situations, or that the prototypes will correspond to actual situations in the world, or that they will be comprehensible to humans (e.g., if a prototype were to blend several conflicting classes). Our goal is to develop a fully interpretable network model based on cases rather than prototypes. As a fully interpretable model, it is at one endpoint of the interpretability spectrum. Researchers can integrate it with black-box approaches to trade off interpretability for accuracy, ease of use, prepocessing, multi-modal learning, scalability or other benefits of black-box models.

Our model, called *Neural network based k-nearest neighbor*, is a neural network that compares a query with stored cases, activates neighboring cases, and makes a prediction based on the labels and activations of the neighboring cases, It can function like a classical k-NN system, with full interpretability. It can be chained with a feature extractor to learn features that best serve k-NN or that are harmonized with a network case adaptation module, or to be combined with other neural network modules for decisions based on the nearest neighbors found. Unlike CBR systems where feature extraction, similarity metrics, or adaptation methods are independent neural/symbolic components, our model does everything in a single neural network. Therefore, the different stages of CBR are trained from end to end to minimize a single loss function combining factors such as explainability, adaptation distance, and even case maintenance cost, achieving better overall system performance. Previous work has show the benefit of harmonizing similarity and adaptation knowledge [18, 29]; we hypothesize that analogous benefits can accrue to other CBR processes as well.

This paper is organized as follows. Section 2 discusses related research in learning for k-NN, ANN-CBR hybrid methods, and interpretable AI. Section 3 explains NN-kNN from the perspectives of neural networks and k-NN. Section 4 compares NN-kNN and other models in computational experiments. The paper concludes with discussion and future directions.

2 Background

2.1 K-Nearest Neighbor and Distance Metric Learning

K-NN can be viewed as a simple form of CBR with minimal adaptation (e.g., averaging of case values). In its most basic form, k-NN, for classification tasks, takes the majority vote of the top k nearest neighbors' class labels, and for regression tasks, averages the labels. [6]. K-NN models are appealing because they are easy to interpret and often provide good accuracy [2].

A simple k-NN approach assumes that all features are equally important in the calculation of distances between cases, and that all cases carry equal weight in the voting of the final prediction. Wettschereck et al. [34] relax both assumptions with a general distance calculation.

Following that, research on distance metric learning has yielded many variants of k-NN [1, 16, 24, 33]. Feature-weighted k-NN weights each feature differently (e.g. house age and number of rooms are more important in deciding a house's price that its longitude and latitude). Methods such as neighborhood components analysis (NCA) and large margin nearest neighbor (LMNN) learn a linear transformation of the feature space to enhance classification accuracy [10, 33]. Distance-weighted k-NN weights cases based on their distance from the query. Instance-weighted k-NN weights cases based on their own significance (e.g. a run-down house in a good neighborhood is an outlier and carries less weight) [3].

This paper proposes a general model wherein the calculations of feature distances, case distances, and target activations are implemented by layers in a neural network that can be trained together.

2.2 Hybrid ANN-CBR Models

ANN-CBR integrations have been proposed both to provide explanations for neural systems and to improve the performance of CBR. In twin systems, cases retrieved using features extracted from networks are used as post-hoc explanations of network decisions [14]. The goal of such systems is to illustrate the types of cases the network might consider similar, but the explanation is decoupled from the network processing. NN-kNN contrasts in providing an interpretable network: it retrieves cases and then adapts them towards the final prediction.

Hybrid ANN-CBR models have also used neural networks to implement different CBR stages [5, 28], with the goal of increasing the scope of CBR system applicability (e.g., to image-based data [31] or increasing the performance of CBR). For example, Mathisen et al. [23] used a siamese network to implement the similarity measurement in case retrieval, and Ye et al. [35] trained a neural network to carry CBR adaptation using case difference heuristics. Following this work, Leake and Ye built a CBR system that uses a siamese network for retrieval and another neural network for adaptation, and then trained the two stages using alternating optimization [21]. The goal of their approach was to harmonize retrieval and adaptation, enabling automated refinement of the coupling of retrieval and adaptation knowledge that had been shown important in the research on adaptation-guided retrieval [30] and case-based coupling similarity and adaptation learning [18]. NN-kNN combines both retrieval and adaptation in a single neural network model (instead of two) and trains both processes end to end to harmonize them. This is especially beneficial when the training needs to coordinate many independent components (e.g. feature extractor, retrieval, adaptation, and case base maintenance).

2.3 Interpretable Neural Networks

Rudin [27] argues that post-hoc explanation of neural networks is inadequate for critical applications. In response, Li et al. [22] propose a neural network model for image classification which can explain each prediction with a set of learned prototypes. Their model includes an autoencoder to encode features and reconstruct samples from features. A special prototype layer stores the features of prototypes found through training, and reconstructed images of prototypes are used for explanation. Their system is interpretable in that network classifications are based on prototypes, and explanations reveal the prototypes used by the system. However, using learned prototypes has potential limitations: the learned prototypes are not guaranteed to correspond to any real example and may blend several conflicting classes, making them less understandable or convincing to a user, and a case may not be covered by any prototype sufficiently similar to be a convincing explanation. Another approach, taken by Chen et al. [4] in ProtoPNet, focuses on explanation based on component extraction: the network "dissects the image by finding prototypical parts, and combines evidence from the prototypes to make a final classification".

Logical Neural Networks [26] address interpretability by endowing each node in a network with the meaning of a logical operator or proposition, and performing logical inference through the network process. They reflect the neuro-symbolic perspective that neural net and logic statements are two renderings of the same model. Our model has an analogous duality: the neural network and k-NN are two renderings of NN-kNN.

3 The NN-kNN Architecture

Neural network based k-nearest neighbor (NN-kNN) is a neural network with special structures and parameters to capture properties of k-NN. Similar to traditional k-NN algorithms, an NN-kNN stores a set of training cases. The basic process is: NN-kNN calculates the case distance between the query and each stored case; The closer cases gain higher activation values; Activated cases contribute to a weighted voting of the final prediction. A sample NN-kNN model

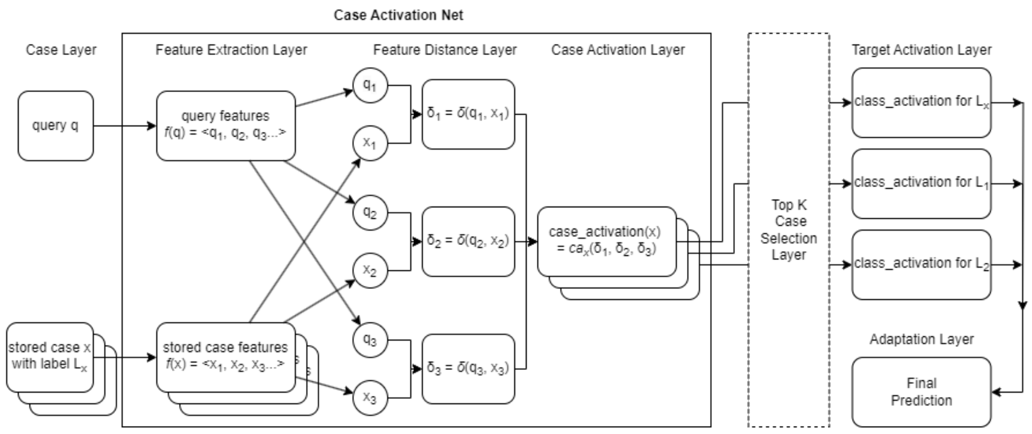


Fig. 1: The Model of NN-kNN.

is shown in Figure 1. Its workflow is as follows (this is an intuitive description of the general process, which we specify in the following section):

1. Given a query q, a copy of q is paired with each stored case x and passed to the feature extraction layer.
2. The feature extraction layer uses a feature extractor f to get the features of the query and cases. This corresponds to the CBR situation assessment process [15].
3. The feature distance layer δ measures distance between corresponding features as $\delta_i = \delta_i(q_i, x_i)$
4. The case activation layer sums up all feature distances δ_is between q and x and calculates a case activation value representing how much the case x activates for the query q.
5. The steps 1-4 are repeated for the query q and each stored cases. The NN-kNN produces one case activation value for each stored case.
6. An optional top k case selection layer then selects the top k activated cases.
7. The target activation layer activates the target output neuron by considering the case activation values. Then a final adaptation layer produces the output based on activated labels.

3.1 Default Implementation of NN-kNN

We consider NN-kNN for classification and regression tasks where each case x contains multiple features and a label L_x. Below we describe a default implementation of the model layers and explain why it is a k-NN method.

1. Feature extraction layer: As for traditional CBR situation assessment, when input cases are already represented as desired features, no processing is needed. In such case, this layer can be omitted. Let $f(x) =< x_1, x_2, x_3... >$ and $f(q) =< q_1, q_2, q_3... >$, where x_i and q_i are the i-th feature of x and q.

Potentially, the feature extraction layer can be a ResNet model for images, a doc2vec model for documents, or any neural feature extractor.

2. Feature distance layer: For $\delta_i()$ we use weighted mean square difference

$$\delta_i(q_i, x_i) = w_{xi}(q_i - x_i)^2$$

where w_{xi} is the feature weight for the case x's feature x_i. Here we assume non-nominal attributes, which requires alternative weightings. The function $\delta_i(q_i, x_i)$ outputs a non-negative number and bigger differences are weighted more after the square operation.

3. Case activation layer: This layer calculates

$$case\_activation(x) = ca_x(\delta_1, \delta_2, \delta_3 ...) = \sigma(w_{x\delta_1} * \delta_1 + w_{x\delta_2} * \delta_2 + ... + b_x)$$

where $w_{x\delta_i}$ is the weight for feature distance δ_i and b_x is the bias for the case x. The sigmoid function $\sigma(ca) = \frac{1}{1+e^{-ca}}$ limits *case_activation* within $[0, 1]$.

NN-kNN calculates the distance between the query and a stored case as a weighted combination of feature distances. Because higher feature distances decrease case activation, feature distance weights $w_{x\delta_i}$ are less than or equal to zero. Thus b_x can be considered as a baseline activation of the case x while $w_{x\delta_i} * \delta_i$ is an inhibitor reducing the activation. The case activation layer uses a sigmoid function to limit all *case_activations* in the range of $[0, 1]$. The lower limit prevents negative activations while the upper limit of (≤ 1) prevents any case from being overly active and overshadowing the influence of other cases.

In the two layers above, cases may share feature weighting, or each case may have its own feature weighting. To illustrate, consider two cases x and y:

1. Their feature distance functions may share a feature weight for the same feature, or $w_{xi} = w_{yi}$.
2. Their case activation functions may share the same feature distance weights, or $w_{x\delta_i} = w_{y\delta_i}$.

If all weight sharing is disabled, each case has its own set of weightings. It is as if each case has its own NN-kNN model with unique weights and biases. If all weight sharings are enabled, all cases share a single NN-kNN model. Case biases b_x and target activation weights $w_{(x,L)}$ (described later) are never shared.

Our choices of feature distance function δ and case activation function ca produce overlapping effects between the weight w_{xi} in the feature distance layer and the weight $w_{x\delta_i}$ in the case activation layer. Consequently, disabling weight sharing for either is effectively disabling both. We therefore regard the two weight sharing settings as a single "weight sharing" flag in Tables 1 and 2. This is not necessarily true for other choices of δ and ca functions.

4. Top k case selection layer: This layer keeps the top k activated cases' activation values and resets other activation values to 0. If this layer is disabled, all activation values are considered for the next step. It is recommended to disable this layer during training for classification, or altogether for regression, as explained later in the Section 5.

The optional top k case selection layer imitates a k-NN model selecting the top k nearest neighbors. When this layer is disabled, NN-kNN behaves more like a standard neural network classifier/regressor where all activations are considered.

5. Target Activation Layer and Final Adaptation Layer: In classification, the target activation layer has multiple neurons, each representing an unique class L. A neuron's output is a weighted sum of all *case_activation*s as

$$class\_activation_L = \Sigma_x(w_{(x,L)} * case\_activation(x)) + b_L$$

where $w_{(x,L)}$ is the weight of the case x for the class L, $case\_activation(x)$ is the *case_activation* of the case x and b_L is the bias of the class L. A rule is set that $w_{(x,L)} > 0$ if the case x is of class L and $w_{(x,L)} = 0$ otherwise, because a case is supposed to only contribute to its correct class label but not to other labels. The adaptation layer chooses the label L with the maximum *class_activation*.

In regression, the target activation layer and adaptation layer is a single neuron. Its output *num_prediction* is a numeric value representing the final prediction. Case activations are normalized based on the sum of all case activations and then they all contribute to the final prediction to the extent of their own labels.

$$normalized\_CA(x) = \frac{case\_activation(x)}{\Sigma_i case\_activation(i)}$$
$$num\_prediction = \Sigma_i(normalized\_CA(i) * L_i)$$

where $\Sigma_i case\_activation(i)$ is the total of all case activations and L_i is the correct label (numerical value) for case i.

During training, the loss is calculated as the error of the final output (cross entropy loss for classification and mean squared error for regression) and backpropagated to all layers in NN-kNN.

In distance-based k-NN, cases are weighted based on their distance to the query. Closer cases carry more weight in deciding the final label. In NN-kNN, a high *case_activation* indicates a closer case and contributes more to the downstream *class_activation* or *num_prediction*. In Regression, all case activations collectively contribute to one target node. The final prediction is a weighted average of the cases' labels, weighted by their *case_activation*s. In instance-weighted k-NN, regardless of their distances to the query, some cases are more important (e.g. a prototype) than others (e.g. an outlier). Instance-weighted k-NN assigns a weight for each case. NN-kNN imitates this behavior with the weight $w_{(x,L)}$ in classification, where a higher weight means the case contributes more to the final prediction.

3.2 NN-kNN is a k-NN method

In its simplest form, NN-kNN can be tuned to act as a vanilla k-NN classifier by choosing the following parameters for each case x: $w_{xi} = w_{x\delta_i} = 1$ for any feature i, so all features are weighted equally; $b_x = 1$ so all cases are weighted equally; The top k case selection layer is enabled; $w_{(x,L)} = 1, b_L = 0$ if the case x is of class L, so each case contributes equally to the final label.

The parameter settings above can also be tuned for NN-kNN to act as a k-NN regressor. A vanilla k-NN regressor selects the top k cases and then takes their average values as its final prediction. The required NN-kNN parameter settings are: $w_{xi} = w_{x\delta_i} = 1$ for any feature i, $b_x = 1$, the top k case selection layer is enabled, and the final adaptation layer sets non-zero case activations to $1/k$. Therefore the top k cases will be selected and their numeric labels averaged.

When weights and biases are preset as described above, NN-kNN replicates a vanilla k-NN. When certain weights are not fixed, NN-kNN can imitate the behavior of k-NN variants using feature weightings, distance weightings, instance weightings, or any combination of them.

We note that NN-kNN's default implementation contains no fully connected layers or other structures that would be considered black boxes. The whole model can be fully interpreted as a CBR system using k-NN for retrieval and a final neural layer for adaptation. However, the model is still a neural network and compatible with network training techniques and with black-box models. Therefore, this formulation of NN-kNN can serve as an endpoint in the spectrum of interpretability, and future variants might combine NN-kNN with black-box models to trade interpretability for accuracy, ease of use, adaptability, scalability or other benefits of black-box models.

3.3 The Benefits of A Network Architecture for k-NN

In addition to its interpretability and high accuracy (which are examined in Section 4), NN-kNN offers the following benefits:

Simple training: Because NN-kNN is trained by backpropagation regardless of its configuration, it is simple to train compared to traditional k-NN variants. Such variants can use combinations of several categories of weights but require several training schemes to train different weights separately. For example, Jie et al. [12] calculate the weighting of a feature based on accuracy of the data without a feature and then use class distributions to predict based on nearest neighbors. To our knowledge, no prior kNN variant trains multiple categories of weightings simultaneously using a single procedure, because different weightings interact with and influence each other in the model's final prediction. All the weights and biases in NN-kNN can be trained from end to end using back-propagation, as for any other neural network.

Harmonization across CBR knowledge containers: CBR systems are often built with retrieval and adaptation steps in independent but interacting modules. Harmonization between the two steps is not guaranteed. For example, the retrieval step might retrieve a case seemingly similar to the query, but whose solution is hard to adapt to solve the query [30]. NN-kNN encapsulates both retrieval and adaptation. NN-kNN achieves retrieval by all layers until the case activation layer, and adaptation by the target activation layer and a final adaptation layer. The two stages are trained together using backpropagation to minimize a total loss (e.g. prediction accuracy) for the whole system.

NN-kNN can easily chain with other network methods. Traditional k-NN methods struggle with high dimensional or complex data sets. Existing works may use a feature extractor such as a VGG net to extract facial features first and use those features as case features in k-NN [36]. There is no guarantee that the features extracted are useful for the k-NN method in predicting the target label. However, when the feature extractor and the NN-kNN model are trained together from end to end and if the performance of the combined model is good, we know that the feature extracted are actually useful features for NN-kNN to identify nearest neighbors and make a final prediction.

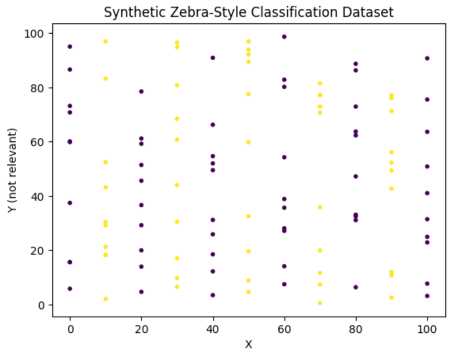
4 Evaluation

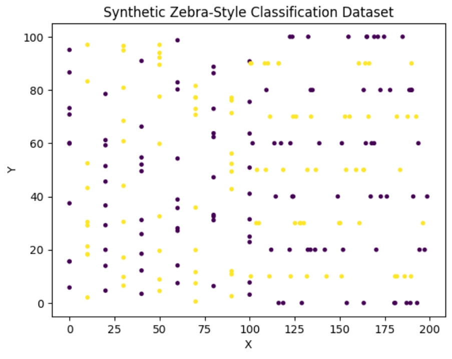
We carried out experiments to compare NN-kNN, a neural network of similar complexity and k-NN using state-of-the-art metric learning algorithms. We addressed two questions: 1) Does NN-kNN provide accuracy comparable to other models? 2) Are the cases activated by NN-kNN good explanations for the model's final decision? Potentially, NN-kNN as a neural network could make correct predictions based on combinations of evidence that do not align with the most similar cases, making their explanations unintuitive (human subjects studies show that the similarity of a case to a new situation affects user trust [9]). Consequently, here we examine if NN-kNN falls into this trap by checking if the top activated cases' labels are close to the real target.

4.1 Implementation Details

Our testbed implementation of NN-kNN followed Section 3.1. We omitted the feature extraction layer and used the features provided in the data sets.

Parameter Settings: We tested four different settings of NN-kNN, with feature weight sharing enabled/disabled and its top k case selection layer enabled/disabled. We compared NN-kNN with k-NN models using three distance metric learning algorithms (from sklearn [25] and metric-learn [32] implementation): Euclidean distance, Neighborhood Components Analysis (NCA) [10] and Large Margin Nearest Neighbor (LMNN) [33]. NCA and LMNN only work with classification data sets. For NN-kNN, NCA and LMNN we set $k = 5$ because it works well on all data sets. For LMNN we used a default learning rate of $1e-6$. Other parameters for NCA and LMNN are set to their default values. We also compare with standard neural network models. For classification tasks, the neural network comprises four hidden layers using the LeakyReLU activation function. For regression tasks, it includes three hidden layers with the ReLU activation function, and dropout layers with a 0.5 probability incorporated in the final two hidden layers. The number of parameters of the linear layers is slightly increased for data sets of higher complexity. Each network model (NN-kNN or a neural network) is trained until its testing performance does not improve for 40 epochs, and the best-so-far model is chosen. The training procedure uses the Adam optimizer (we used learning rate = 0.01 for NN-kNN and 0.0001 for a





(a) Only one feature is relevant. (b) Features relevant in different regions.

Fig. 2: Two Zebra-Style Data Sets

neural network because the settings produced the best average result) based on cross entropy loss for classification and mean square error loss for regression. [‡]

Data Sets and Procedure: Experiments used 10-fold cross validation for six classification data sets and four regression data sets. Classification data sets are Olivetti faces, digits, iris, wine, breast cancer, and balance. Regression data sets are diabetes, california housing, abalone, and body fat and their feature values are normalized befor experiments. They are all available from the UCI ML repository [7] or integrated in scikit-learn [25]. To investigate the performance of NN-kNN with instance-based feature weighting, we also created two data sets, Zebra (a) and Zebra (b) (shown in Fig. 2), which contain cases with (x, y) coodinates and one of two class labels. In Zebra (a), two classes (100 cases) alternate along the x axis, analogously to a zebra stripe pattern. In Zebra (b), two classes (100 cases) appear in a vertical zebra stripe pattern when $x < 100$ and (100 cases) in a horizontal zebra stripe pattern when $x \geq 100$. The data set Zebra (b) is more difficult than Zebra (a) as it requires the classifier to learn local patterns instead of a single global pattern.

4.2 Experimental Results

Tables 1 and 2 summarize the experimental results. Numbers for classification are classification accuracy (the higher the better). Numbers for regression are mean square error (the lower the better). For regression, the errors of the top k cases' average in NN-kNN are also shown. We first consider results on the zebra data sets, then classification, and finally regression.

Generated Zebra Stripe Data Sets: Many methods struggle in the generated zebra data sets as data are sparse and one of the two dimensions is irrelevant (row 1 and 2 in Table 1). Methods with feature weightings and instance weightings, such as NN-kNN, can potentially solve Zebra data sets well. For data set

[‡]Code is available at https://github.com/Heuzi/NN-kNN/tree/main

	NN-kNN				NNet	K-NN	NCA	LMNN
Weight Sharing	On	On	Off	Off				
Top K	On	Off	On	Off				
Zebra (a)	0.909	0.918	0.872	0.872	0.782	0.363	**0.981**	0.336
Zebra (b)	0.850	0.850	**0.968**	**0.968**	0.673	0.368	0.395	0.354
OFace	0.862	0.945	0.934	**0.987**	0.938	0.850	0.9375	0.975
Digits	0.988	0.987	0.979	0.988	0.983	0.987	0.989	0.988
Iris	0.926	0.966	0.953	0.960	**0.987**	0.966	0.966	0.953
Wine	0.961	**0.994**	0.966	0.983	0.836	0.680	0.747	0.954
Breast Cancer	0.940	0.943	0.929	**0.959**	0.951	0.933	0.927	0.947
Balance	0.894	0.929	0.894	0.944	**0.994**	0.8463	0.960	0.849

Table 1: Accuracy (the Higher the Better) on Classification Data Sets

	NN-kNN								NNet	K-NN
Wt Sharing	On		On		Off		Off			
Top K	On		Off		On		Off			
	Wtd	Top k	Wtd	Top k	Wtd	Top k	Wtd	Top k		
Diabetes	0.558	0.616	0.563	0.606	0.558	0.571	0.549	0.594	**0.513**	0.615
California	0.690	0.958	0.537	0.912	0.931	1.564	**0.358**	0.783	0.705	1.079
Abalone	0.500	0.530	0.442	0.918	0.455	0.944	0.436	1.229	**0.414**	0.483
Body Fat	0.671	0.531	0.493	0.663	0.741	0.616	**0.423**	1.013	0.585	0.609

Table 2: Mean Squared Error (the Lower the Better) for Regression Data Sets. NN-kNN provides both weighted average of activated cases ("Wtd"), or plain average of top k activated cases ("Top k").

2a, NN-kNN achieves highest accuracy when all training cases are used to train a shared set of feature weightings. For data set 2b, NN-kNN achieves highest accuracy when feature weighting is not shared. This is because the importance of a give feature varies in different regions of the data landscape. It is interesting to note that NN-kNN still functions well for data set 2b when feature weight sharing is on. We hypothesize that this is because it also learns a case weighting $w_{(x,L)}$, which can lower the importance of cases that are too close to class boundaries or are in ambiguous regions. Note that the neural network over-fits and performs poorly on these data sets.

Classification Data Sets: When top k case selection is enabled, NN-kNN achieves relatively good accuracy, indicating that the top activated cases generally share the same class label as the query and are indeed good explanations for its prediction. Turning off the top k case selection generally improves NN-kNN's classification accuracy as the model can utilize all cases' activations. Turning off the feature weight sharing also improves its accuracy as the model can generalize over local patterns.

The best performining models are either NN-kNN or the traditional neural network. NCA learns a metric well when certain features are obviously important (e.g., zebra (a)). Both NCA and LMNN perform relatively well on most data sets but struggle on some, most notably zebra (b). The standard k-NN suffers for harder data set while NN-kNN maintains performance.

Regression Data Sets: Enabling the top k case selection layer can lead to bad performance for regression tasks, as we explain in detail in Section 5. In addition to the NN-kNN model's error, which is based on a *weighted average* of the labels of the activated cases, we show the average error of the top k cases' *plain average*. We note: (1) The weighted average generally has lower error than the plain average, suggesting that NN-kNN learns to activate cases for a proper activation-based weighting. (2) Sometimes the error of the NN-kNN's plain average is worse than the error of k-NN. This is because NN-kNN may activate cases with labels far from the true label. However, NN-kNN can correct this by weighting them by their activation and also by considering all cases' activations, leading to a lower error than that of k-NN.

Discussion of Experimental Results

Accuracy Different methods excel at different data sets: The baseline neural network performs well when it captures a hidden rule (e.g. in the balance data set, a balance is achieved if the products of weight and distance are equal on two sides); NCA learns a metric that captures the underlying structure (certain features are obviously important such as in zebra (a)); LMNN performs well when local patterns matter by maximizing the margin between classes. NN-kNN performs the best or relatively well across all data sets. The best performing NN-kNN variant feature weight sharing disabled—for maximum degrees of freedom and has the top k case selection layer disabled, to use all cases.

NN-kNN is inherently a neural network with preset neuron meanings (in the case layer), connections (e.g. one case only activates a corresponding class) and preset-but-tunable parameters (weights and biases). It might appear that NN-kNN would sacrifice a fraction of performance because the preset connections and parameters constraints limit the degrees of freedom, in comparison to a neural network where all neurons and weights may vary freely. However, we observe that NN-kNN can perform equally good or even better than a neural network, as the preset neurons and connections function as case base knowledge infused into the network model.

Interpretability NN-kNN offers interpretability through its top activated cases. In classification, the accuracy of NN-kNN when the top k case selection layer is enabled is only slightly worse than or the same as when the layer is disabled, suggesting that the top k cases already provides a good estimation of the final label. In regression, this is not the case as the error of the plain average of the top k cases is potentially much worse than the weighted average. This is due to the nature of regression problems: Neighboring cases in classification likely share the same label, while neighboring cases in regression can have vastly different labels. Currently NN-kNN takes a weighted average based on case activations, correcting the error of the plain average. NN-kNN can offer explanations for regression by providing the top activated cases and their corresponding activations. A future direction is to lower the error of the plain average by including it as an additional term in the loss function, so that the training process may minimize it.

Moreover, the internal function of NN-kNN can be fully interpreted as a k-NN model. Given a query and a case in the NN-kNN system case base, (1) The feature distance layer's activations indicate which features are similar between the query and the case. (2) The feature weights indicate which feature is important in activating cases. (3) The case activation layer's activations indicate which case is most relevant for decision making for the particular query. (4) The case bias b_x and class bias b_L indicate the default significance of the case and a class respectively. (5) The weight $w_{(x,L)}$ indicates the weight of contribution from the case to the prediction of a class.

5 Discussion: Effect of Top K Case Selection

The effect of the top k case selection layer varies according to circumstances. We disabled it during training, because of potential detrimental effects during training. If the layer is enabled during training, each training query would only provide gradient descent information for the top k cases. Only a small set of cases are activated and a small portion of the network is trained. Moreover, the weights of one activated case are only trained to work with weights of $k-1$ other cases in that small set of top k cases. Given a new query during testing, NN-kNN is likely to activate a small set of nearest neighboring cases, that might have never activated together before during training. Therefore their weights are never trained to work in harmony. This last point is especially an issue for regression, where case activations are used to produce a weighted average of the final prediction. As supported by the experimental results in Section 4, we recommend disabling the top k case selection layer for regression.

Interestingly, we note that top k cases are always selected in k-NN and its variants, yet no other reports on k-NN to our knowledge have mentioned that selecting top k cases is an issue. This is due to the hidden assumption in most k-NN methods that all cases share the same feature weights (the feature x_i in case x is as important as the feature y_i in case y) and instance weights. Because cases are equally important, k-NN classifier can take a majority vote of the top k cases' class label in classification; k-NN regressor can take an average of the top k cases' numeric label in regression. This assumption does not hold in NN-kNN and enabling top k cases layer generally lowers the model's accuracy.

6 Challenges and Future Directions

NN-kNN can be considered as k-NN algorithm or as a neural network. It follows that NN-kNN unavoidably inherits some limitations from both sides, but we also see potential to circumvent these limitations with techniques from both sides.

Online Learning: A neural network does not have online learning ability: adding a training case means, in principle, that the whole model needs retraining. When weight sharing is disabled, adding a new case creates a separate NN-kNN and the whole model (other NN-kNNs along with the newly added NN-kNN) needs to be retrained. However, this drawback can be offset by turning on all weight sharing, so that the newly added cases reuse weights already trained. This can be seen as equivalent to simply adding cases to a kNN system with fixed weights. A next step is to build variants of NN-kNN with online learning ability.

Scaling Up and Case Base Maintenance: For extremely large data sets, the cost of NN-kNN becomes an issue. This may be partially alleviated by maintaining only the prototypical cases instead of all cases, as it was done by Li et al. [22]. Other solutions involve case base maintenance techniques [13, 19] or neural network pruning techniques [8].

Hyperparameters and Settings for Training NN-kNN: Like any neural network, NN-kNN depends on hyperparameters that may be tuned. Common settings for regular networks do not directly apply to NN-kNN. For example: 1) A learning rate of 1e-5 is common for neural networks, but it leads to underfitting in NN-kNN. Our NN-kNN used 1e-2 for learning rate in all experiments; 2) L2 regularization normally prevents overfitting, but it impedes the learning of NN-kNN, because for NN-kNN some cases/features should be learned to be (non-)important and L2 regularization defeats this learning purpose. An important future direction is finding a hyperparameter optimization strategy for NN-kNN's structure.

Case Adaptation: The model presented here uses simple case adaptation. The reasoning of CBR depends on rich adaptation capabilities. NN-kNN may be chained with an adaptation network [35], but if that network is uninterpretable it undermines the interpretability of the overall architecture. Thus making network adaptation architectures more interpretable is an interesting future task.

7 Conclusion

NN-kNN combines perspectives of connectionist and symbolic machine learning approaches. This method serves as an endpoint on a spectrum of interpretability, and new models can be built upon it by integrating additional interpretable, explainable, or black-box modules. NN-kNN has the potential to coordinate learning for multiple CBR processes, from feature extraction to retrieval and adaptation to case base maintenance, in a single coherent model.

References

1. Bellet, A., Habrard, A., Sebban, M.: Metric Learning. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers (2015)

2. Beyer, K., Goldstein, J., Ramakrishnan, R., Shaft, U.: When is nearest neighbor meaningful? In: Beeri, C., Buneman, P. (eds.) Database Theory ICDT99, LNCS, vol. 1540, pp. 217–235. Springer (1999)
3. Bicego, M., Loog, M.: Weighted k-nearest neighbor revisited. In: Twenty-Third International Conference on Pattern Recognition (ICPR). pp. 1642–1647. IEEE (2016)
4. Chen, C., Li, O., Tao, D., Barnett, A., Rudin, C., Su, J.K.: This looks like that: Deep learning for interpretable image recognition. In: Advances in Neural Information Processing Systems. vol. 32. Curran (2019), https://proceedings.neurips.cc/paper_files/paper/2019/file/adf7ee2dcf142b0e11888e72b43fcb75-Paper.pdf
5. Chen, J., Hsu, S.C.: Hybrid ANN-CBR model for disputed change orders in construction projects. Automation in Construction 17(1), 56–64 (Nov 2007)
6. Cover, T., Hart, P.: Nearest neighbor pattern classification. IEEE Transactions on Information Theory 13(1), 21–27 (1967)
7. Dua, D., Graff, C.: UCI machine learning repository (2017), http://archive.ics.uci.edu/ml
8. Frankle, J., Carbin, M.: The lottery ticket hypothesis: Finding sparse, trainable neural networks. In: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net (2019), https://openreview.net/forum?id=rJl-b3RcF7
9. Gates, L., Leake, D., Wilkerson, K.: Cases are king: A user study of case presentation to explain cbr decisions. In: Case-Based Reasoning Research and Development, ICCBR 2023. pp. 153–168. Springer (2023)
10. Goldberger, J., Hinton, G.E., Roweis, S., Salakhutdinov, R.R.: Neighbourhood components analysis. In: Advances in Neural Information Processing Systems. vol. 17. MIT Press (2004), https://proceedings.neurips.cc/paper_files/paper/2004/file/42fe880812925e520249e808937738d2-Paper.pdf
11. Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Giannotti, F., Pedreschi, D.: A survey of methods for explaining black box models. ACM Computing Surveys 51(5), 1–42 (2018)
12. Huang, J., Wei, Y., Yi, J., Liu, M.: An improved KNN based on class contribution and feature weighting. In: 2018 10th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA). pp. 313–316 (2018)
13. Juarez, J.M., Craw, S., Lopez-Delgado, J.R., Campos, M.: Maintenance of case bases: Current algorithms after fifty years. In: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18. pp. 5457–5463. IJCAI (2018)
14. Keane, M.T., Kenny, E.M.: How case-based reasoning explains neural networks: A theoretical analysis of XAI using post-hoc explanation-by-example from a survey of ann-cbr twin-systems. In: Case-Based Reasoning Research and Development. pp. 155–171. Springer, Cham (2019)
15. Kolodner, J., Leake, D.: A tutorial introduction to case-based reasoning. In: Leake, D. (ed.) Case-Based Reasoning: Experiences, Lessons, and Future Directions, pp. 31–65. AAAI Press, Menlo Park, CA (1996)
16. Kulis, B.: Metric learning: A survey. Found. Trends Mach. Learn. 5, 287–364 (2013), https://api.semanticscholar.org/CorpusID:55485900
17. Leake, D.: CBR in context: The present and future. In: Leake, D. (ed.) Case-Based Reasoning: Experiences, Lessons, and Future Directions, pp. 3–30. AAAI Press, Menlo Park, CA (1996), http://www.cs.indiana.edu/~leake/papers/a-96-01.html

18. Leake, D., Kinley, A., Wilson, D.: Learning to integrate multiple knowledge sources for case-based reasoning. In: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence. pp. 246–251. Morgan Kaufmann (1997)
19. Leake, D., Wilson, D.: Categorizing case-base maintenance: Dimensions and directions. In: Proceedings of the Fourth European Workshop on Case-Based Reasoning. pp. 196–207. Springer Verlag, Berlin (1998)
20. Leake, D., Crandall, D.: On bringing case-based reasoning methodology to deep learning. In: Case-Based Reasoning Research and Development, ICCBR-20. pp. 343–348. Springer, Cham (2022)
21. Leake, D., Ye, X.: Harmonizing case retrieval and adaptation with alternating optimization. In: Case-Based Reasoning Research and Development. pp. 125–139. Springer International Publishing, Cham (2021)
22. Li, O., Liu, H., Chen, C., Rudin, C.: Deep learning for case-based reasoning through prototypes: A neural network that explains its predictions. In: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence. pp. 3530–3537. AAAI Press (2018)
23. Mathisen, B.M., Aamodt, A., Bach, K., Langseth, H.: Learning similarity measures from data. Progress in Artificial Intelligence pp. 129–143 (10 2019)
24. Park, J., Im, K.H., Shin, C.K., Park, S.: Mbnr: Case-based reasoning with local feature weighting by neural network. Applied Intelligence 21 (11 2004)
25. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12, 2825–2830 (2011)
26. Riegel, R., Gray, A., Luus, F., Khan, N., Makondo, N., Akhalwaya, I.Y., Qian, H., Fagin, R., Barahona, F., Sharma, U., et al.: Logical neural networks. arXiv preprint arXiv:2006.13155 (2020)
27. Rudin, C.: Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. Nature Machine Intelligence 1, 206–215 (05 2019)
28. Sarabia, Y., Lorenzo, M., Perez, R., Martinez, R.: Extending CBR-ANN hybrid models using fuzzy sets. In: 2005 International Conference on Neural Networks and Brain. vol. 3, pp. 1755–1760 (2005)
29. Smyth, B., Keane, M.: Design à la Déjà Vu: Reducing the adaptation overhead. In: Leake, D. (ed.) Case-Based Reasoning: Experiences, Lessons, and Future Directions. AAAI Press, Menlo Park, CA (1996)
30. Smyth, B., Keane, M.: Adaptation-guided retrieval: Questioning the similarity assumption in reasoning. Artificial Intelligence 102(2), 249–293 (1998)
31. Turner, J.T., Floyd, M.W., Gupta, K., Oates, T.: Nod-cc: A hybrid cbr-cnn architecture for novel object discovery. In: Case-Based Reasoning Research and Development. pp. 373–387. Springer International Publishing, Cham (2019)
32. de Vazelhes, W., Carey, C., Tang, Y., Vauquier, N., Bellet, A.: metric-learn: Metric learning algorithms in python. Journal of Machine Learning Research 21(138), 1–6 (2020), http://jmlr.org/papers/v21/19-678.html
33. Weinberger, K.Q., Saul, L.K.: Distance metric learning for large margin nearest neighbor classification. J. Mach. Learn. Res. 10, 207–244 (jun 2009)
34. Wettschereck, D., Aha, D., Mohri, T.: A review and empirical evaluation of feature-weighting methods for a class of lazy learning algorithms. Artificial Intelligence Review 11(1-5), 273–314 (February 1997)

35. Ye, X., Leake, D., Crandall, D.: Case adaptation with neural networks: Capabilities and limitations. In: Case-Based Reasoning Research and Development ICCBR-22. pp. 143–158. Springer, Cham (2022)
36. Ye, X., Zhao, Z., Leake, D., Wang, X., Crandall, D.J.: Applying the case difference heuristic to learn adaptations from deep network features. CoRR abs/2107.07095 (2021), https://arxiv.org/abs/2107.07095