OS 4118 **Neural Network Example** Fall AY 2020

**Introduction:** In this example I want you to construct and run a neural network in Keras, just because I think it will be a beneficial experience.

**The Data:** In Sakai you can find a data set called “super.csv.” The data describes about 21,000 superconducting compounds. There are 81 predictors, which are attributes of the compounds, and a continuous response called critical\_temp which gives the temperature (in K) below which this compound acts as a superconductor. Reference: Hamidieh, K.: “A data-driven statistical model for predicting the critical temperature of a superconductor,” Computational Materials Science 154 (2018), pp. 346-354. I got this paper through the NPS library so I bet you can too.

It’s always a good idea to **scale** the data, and I have done that for you in a script called super.R, also on Sakai. Notice how we scale the training set first, and then use those scalings – the training set’s columns’ means and SDs – to scale the test set. We don’t first scale the whole thing, then divide into training and test sets, because the test set is supposed to start out as utterly independent of the training set. If we scaled the whole data set first, we’d be using some information from the test set in the training set.

Your mission: Your mission is to build a neural network to predict the (hint: numeric) value of critical\_temp, and then compare the predictions of the neural network made on the test set to the actual temperatures in the test set. Hamidieh uses RMSE *R*2 as his measure of quality, so let’s use those. Remember that if obs is the vector of actual critical temperatures, and pred is the vector of predictions made by the model, we can compute *R*2 in R as cor (obs, pred)^2, and RMSE is sqrt (mean ( (obs - pred)^2))).

Hamidieh says that a linear model produced an *R*2 of 0.74 with an RMSE of 17.6, but that the best of his 200 or so boosted tree models (varying different parameters) produced an *R*2 of 0.92 with an RMSE of 9.5. Since we don’t have the luxury of trying 200 neural networks let’s aim for an *R*2 that’s at least better than 0.74.

**Details:** Remember the steps in keras.

**Step 1:** We start by creating an empty model, like this: mod <- keras\_model\_sequential(). Of course you can call your model whatever you like.

**Step 2:** Now we add layers by making one or more calls to layer functions. For the fully-connected layers we’ve been talking about, the function is layer\_dense(). You will need to pass in the mod object, but you don’t have to assign the result back to mod. Select a nice number of units – a good starting point might be 50 nodes in the first hidden layer and 10 in the second. Remember that in the first hidden layer you will need to tell the input\_shape argument how many columns there are in the input data. You will also need to select an activation function. Even though the relu is widely used, I recommend activation = "sigmoid" in the hidden layers during this example. The final layer will need an activation, too: what will be best here?

Finally, as an optional step, you can add regularization with an argument like kernel\_regularizer = regularizer\_l1\_l2 (l1 = 0.01, l2 = 0.01). As you call tell, this particular regularization has both l1 (“lasso”) and l2 (“ridge”) components, but you could also have specified kernel\_regularizer = regularizer\_l1 (l1 = 0.01) if you only wanted lasso.

**Step 3:** Now we “compile” the model. For “loss” we want to use "mse" for mean-squared error, the usual choice in a regression problem. For “optimizer,” select either the character "sgd" or specify sgd with some more complicated options, with a line like

optimizer = optimizer\_sgd(lr = 0.01, momentum = 0.5, decay = 0.01),

and for “metrics,” we may as well use “mse” too. Interestingly the “loss” and the “metrics” don’t match exactly and I’m not yet sure why.

**Step 4:** Finally we “fit” the model. In addition to passing in the model, the training set as argument x, and the true response as argument y, you will want to pass in epochs (how many times to run through the data?) and batch\_size (how many observations in an epoch?). A starting point might be epochs = 400, batch\_size = 18000, which is the size of the full data set. **Optional:** You can add a validation set like this. First, create a vector of indices of the validation set. Perhaps we’ll set aside 2000 observations for validation, out of the 18,000 in the training set.

val.ind <- sample (18000, 2000, replace=F)

Then, when we call the fit() function, we pass in something like

x = super.train[-val,], y = super.train.y[-val,]

and also another argument

validation\_data = list (super.train[val.ind,], super.train.y[val.ind,]).

Adding a validation set is good practice.

There are lots and lots more things we can do in keras – I will touch on some of them. But this should be plenty to get you a model that might be able to compete with Hamidieh’s linear model.