Aleatoric and Epistemic uncertainty

We captured uncertainty in ways that increase trust in our model by including it in our new loss function that include both the prediction and the log variance. It also improves our results by decreasing the incorrect highly confident predication. The trust is embedded in the loss function where the model take into considertation in which interval we have more data and the results are reliable and which invertals are not.

We used MC-dropout where we compile n_passes forward passes and get the results. Then we get the mean and the variance to calculate the uncertainty. Including uncertainty in the minimization problem of the loss will lead to adjusting/improving the model by taking into consideration the error that arise from uncertainty since the topic includes many uncertainties about the factors included in the model. Our topic is looking into genes where the least measurement error can greatly affect our results. Aslo till now we dont have full grasp of known genes and unkown genes that may affect our results. We couldn’t compile the a sophisticated model nor we could use many(enough) epochss to improve the results because including uncertainity is heavy computationally.

We applied two models, with and without feature selection. The results show a slight improvement when using the feature selection. This shows that either almost all the features (genes) are important or including uncertainty in the loss function slightly cover the aspect of feature selection (just a hypothesis). Although feature selection is only improving the model slightly in our topic, we can notice that it decreases the computational cost thus we get a better result in less time.

Combined aleatoric and epistemic uncertainties without Feature selection

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import numpy.ma as ma
import pandas as pd
import sys
from matplotlib.backends.backend_pdf import PdfPages
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from tensorflow.python.framework import ops
from keras.utils import np_utils
import tensorflow.compat.v1 as tf
from sklearn.model_selection import train_test_split
tf.disable_v2_behavior()
```

```
X = pd.read_csv('/content/drive/MyDrive/train_features.csv')
X = np.asarray(X)[:,4:].astype(float)    #4

y = pd.read_csv('/content/drive/MyDrive/train_targets_scored.csv')
y = np.asarray(y)[:,1:]     #1


X_shape, y_shape = X.shape[1], y.shape[1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=123)
```

```
theme = {
    "truth": "blue",
    "prediction_mean": "red",
    "prediction_std": "red", # alpha higher
    "aleatoric": "green",
    "epistemic": "orange",
}
def plot_mean_vs_truth(x_truth, y_truth, x_prediction, y_prediction, std=None, ax=None):
    """
    :param x_truth: x training sample
    :param y_truth: y training sample
    :param x_prediction: x evaluation
    :param y_prediction: y evaluation (predicted)
    :param std: (optional) standard deviation for every prediction
    :param x: (optional) ax
    :return: fig, ax
    """
    if not ax:
        ax = plt.gca()
    ax.scatter(x_truth, y_truth, label="Truth", color=theme["truth"])
    ax.plot(x_prediction, y_prediction, label="Prediction", color=theme["prediction_mean"])

    if std is not None:
        ax.fill_between(x_prediction.flatten(), y_prediction - std, y_prediction + std,
                        color=theme["prediction_std"], alpha=0.3, label="Predictive Variance")
```

```
                    color=theme["prediction_std"], alpha=0.3, label="Predictive variance")
    ax.legend()


def plot_mean_vs_truth_with_uncertainties(x_truth, y_truth, x_prediction, y_prediction,
                                          aleatoric, epistemic, ax=None):
    """
    Same as plot_mean_vs_truth but with the uncertainties splitted into aleatoric and epistemic.
    :param x_truth:
    :param y_truth:
    :param x_prediction:
    :param y_prediction:
    :param std:
    :return: fig, ax
    """
    if not ax:
        ax = plt.gca()

    ax.scatter(x_truth, y_truth, label="Truth", color=theme["truth"])
    ax.plot(x_prediction, y_prediction, label="Prediction", color=theme["prediction_mean"])

    # inner tube
    ax.fill_between(x_prediction.flatten(), y_prediction - aleatoric , y_prediction + aleatoric,
                    color=theme["aleatoric"], alpha=0.2, label="aleatoric")

    # two outer tubes
    ax.fill_between(x_prediction.flatten(), y_prediction - aleatoric , y_prediction - aleatoric - epistemic / 2.0,
                    color=theme["epistemic"], alpha=0.3, label="epistemic")

    ax.fill_between(x_prediction.flatten(), y_prediction + aleatoric , y_prediction + aleatoric + epistemic / 2.0,
                    color=theme["epistemic"], alpha=0.3)

    # if std is not None:
    #     ax.fill_between(x_prediction.flatten(), y_prediction - std, y_prediction + std,
    #                     color=theme["prediction_std"], alpha=0.3, label="Prediction std")
    # ax.legend()
```

Model

We used MC-dropout so we defined the model with dropout. We kept the model simple because the computational cost is big. You can notice the dropout layers between each dense layer

```
def combined_model(x, dropout_rate):
    """
    Model that combines aleatoric and epistemic uncertainty.
    Based on the "What uncertainties do we need" paper by Kendall.
    Works for simple 2D data.
    :param x: Input feature x
    :param dropout_rate:
    :return: prediction, log(sigma^2)
    """

    # with tf.device("/gpu:0"):
    keep_prob = 1 - dropout_rate

    fc1 = tf.layers.dense(inputs=x, units=64, activation=tf.nn.relu)
    fc1 = tf.nn.dropout(fc1, keep_prob)

    fc2 = tf.layers.dense(inputs=fc1, units=128, activation=tf.nn.relu)
    fc2 = tf.nn.dropout(fc2, keep_prob)

    fc3 = tf.layers.dense(inputs=fc2, units=264, activation=tf.nn.relu)
    fc3 = tf.nn.dropout(fc2, keep_prob)

    fc4 = tf.layers.dense(inputs=fc3, units=128, activation=tf.nn.softmax)
    fc4 = tf.nn.dropout(fc2, keep_prob)

    fc5 = tf.layers.dense(inputs=fc3, units=64, activation=tf.nn.softmax)
    fc5 = tf.nn.dropout(fc2, keep_prob)
    # Output layers has predictive mean and variance sigma^2
    output_layer = tf.layers.dense(fc5, units=y_shape+1)

    predictions = tf.expand_dims(output_layer[:, 0], -1)
    log_variance = tf.expand_dims(output_layer[:, 1], -1)

    return predictions, log_variance
```

Model parameters:

we defined the loss function, learning rate and optimizer. We defined the session and initialize it. You can notice how we included both the error in the predicition and the uncertainty in the loss function

```
def combined_training(x_truth, y_truth, dropout, learning_rate, epochs, display_step=500):
    """
    Generic training of a Combined (uncertainty) network for 2D data.
    :param x_truth: training samples x
    :param y_truth: training samples y / label
    :param dropout:
    :param learning_rate:
    :param epochs:
    :param display_step:
    :return: session, x_placeholder, dropout_placeholder
    """
    tf.reset_default_graph()
    x_placeholder = tf.placeholder(tf.float32, [None, X_shape])
    y_placeholder = tf.placeholder(tf.float32, [None, y_shape])
    dropout_placeholder = tf.placeholder(tf.float32)

    prediction, log_variance = combined_model(x_placeholder, dropout_placeholder)

    tf.add_to_collection("prediction", prediction)
    tf.add_to_collection("log_variance", log_variance)
    loss = tf.reduce_sum(0.5 * tf.exp(-1 * log_variance) * tf.keras.backend.sum(tf.square(tf.abs(y_placeholder - prediction)))+ 0.5 *
    # loss = tf.losses.reduce_sum(0.5 * tf.exp(-1 * log_variance) * tf.square(tf.abs(y_placeholder - prediction))+ 0.5 * log_variance
    optimizer = tf.train.AdamOptimizer(learning_rate)
    train = optimizer.minimize(loss)

    init = tf.global_variables_initializer()
    sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))
    sess.run(init)

    for epoch in range(epochs):
        feed_dict = {x_placeholder: x_truth.reshape(-1,X_shape),
                     y_placeholder: y_truth.reshape(-1,y_shape),
                     dropout_placeholder: dropout}

        sess.run(train, feed_dict=feed_dict)

        if epoch % display_step == 0:
            print("Epoch {}".format(epoch))
            current_loss = sess.run(loss, feed_dict=feed_dict)
            print("Loss {}".format(current_loss))
            print("================")

    print("Training done")

    return sess, x_placeholder, dropout_placeholder
```

MC-dropout:

Since we used MC-dropout, we run n_passes forward passes. Then we get the mean and the variance of the results to calculate the uncertainty

```
def combined_evaluation(x_train, y_train,x_test,y_test, dropout, learning_rate, epochs, n_passes):
    """
    :param x:`
    :param y:
    :param dropout:
    :param learning_rate:
    :param epochs:
    :param n_passes:
    :return:
    """
    sess, x_placeholder, dropout_placeholder = \
        combined_training(x_train, y_train, 0.2, learning_rate, epochs)

    prediction_op = sess.graph.get_collection("prediction")
    log_variance = sess.graph.get_collection("log_variance")
    aleatoric_op = tf.exp(log_variance)

    feed_dict = {x_placeholder: x_train,
                 dropout_placeholder: dropout}

    predictions = []
    aleatorics = []
    for _ in range(n_passes):
        prediction, aleatoric = sess.run([prediction_op, aleatoric_op], feed_dict)
        predictions.append(prediction[0])
        aleatorics.append(aleatoric[0])

    y_eval = np.mean(predictions, axis=0).flatten()
    epistemic_eval = np.var(predictions, axis=0).flatten()
    aleatoric_eval = np.mean(aleatorics, axis=0).flatten()
    total_uncertainty_eval = epistemic_eval + aleatoric_eval

    # plot_mean_vs_truth_with_uncertainties(x_train[:,1], y_train, x_test[:,1], y_test, aleatoric_eval, epistemic_eval, ax)

    # fig.suptitle("Dropout - Learning Rate %f, Epochs %d, Dropout %.3f, Passes %d" %(learning_rate, epochs, dropout, n_passes))

    # ax.fill_between(x_eval.flatten(), 0, epistemic_eval, label="epistemic", color="green", alpha=0.4)
    # ax.fill_between(x_eval.flatten(), 0, aleatoric_eval, label="aleatoric", color="orange", alpha=0.4)
    # ax.legend()
```

```
# combined_evaluation(X_train, y_train, X_test, y_test, 0.1, 1e-2, 5000, 300)
```

```
combined_evaluation(X_train, y_train, X_test, y_test, 0.1, 1e-3, 20000, 50)
```

```
================
Epoch 10500
Loss 283354.1875
================
Epoch 11000
Loss 274919.21875
================
Epoch 11500
Loss 276921.375
================
Epoch 12000
Loss 257903.21875
================
Epoch 12500
Loss 248093.59375
================
Epoch 13000
Loss 238859.546875
================
Epoch 13500
Loss 228961.046875
================
Epoch 14000
Loss 221641.21875
================
Epoch 14500
Loss 221243.875
================
Epoch 15000
Loss 205094.34375
================
Epoch 15500
Loss 191521.09375
================
Epoch 16000
Loss 184630.859375
================
Epoch 16500
Loss 172132.40625
================
Epoch 17000
Loss 159510.375
================
Epoch 17500
Loss 147613.90625
================
Epoch 18000
Loss 133765.71875
================
Epoch 18500
Loss 124513.3046875
================
Epoch 19000
Loss 119744.2734375
================
Epoch 19500
Loss 116622.515625
================
Training done
```

Combined aleatoric and epistemic uncertainties with Feature selection

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import numpy.ma as ma
import pandas as pd
import sys
from matplotlib.backends.backend_pdf import PdfPages
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from tensorflow.python.framework import ops
from keras.utils import np_utils
import tensorflow.compat.v1 as tf
from sklearn.model_selection import train_test_split
tf.disable_v2_behavior()
from sklearn.feature_selection import SelectKBest, chi2 ,f_classif
```

```
X_pd = pd.read_csv('/content/drive/MyDrive/train_features.csv')
X = np.asarray(X_pd)[:,4:].astype(float)    #4

df_y = pd.read_csv('/content/drive/MyDrive/train_targets_scored.csv')
y_df = df_y.iloc[:, 1:]
y = np.array(y_df)

X_shape, y_shape = X.shape[1], y.shape[1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=123)
```

Featuer Selection

```
# Method 2: f_scores
selected_features = []
for label in y_df:
  selector = SelectKBest(f_classif, k='all')
  selector.fit(X, y_df[label])
  selected_features.append(list(selector.scores_))

final_scores = np.mean(selected_features, axis=0)
features_indexes = []; threshold = 50 # gave us 393 features
for i in range(len(final_scores)):
  if final_scores[i] > threshold:
    features_indexes.append(i)
print(len(features_indexes))
```

```
393
```

```
def get_selected_features_df(X_df, f_i):
  X_selected = X_pd.iloc[:, 4:].copy()
  i = 0
  for col in X_df.iloc[:, 4:]:
    if i not in f_i:
      X_selected.pop(col)
    i += 1
  return X_selected

train_data = get_selected_features_df(X_pd, features_indexes)
print(train_data.shape)
```

```
(23814, 393)
```

```
X_shape, y_shape = train_data.shape[1], y.shape[1]
X_train, X_test, y_train, y_test = train_test_split(np.array(train_data), np.array(y), test_size=0.2, random_state=123)
```

```
theme = {
    "truth": "blue",
    "prediction_mean": "red",
    "prediction_std": "red", # alpha higher
    "aleatoric": "green",
    "epistemic": "orange",
}
def plot_mean_vs_truth(x_truth, y_truth, x_prediction, y_prediction, std=None, ax=None):
    """
    :param x_truth: x training sample
    :param y_truth: y training sample
    :param x_prediction: x evaluation
    :param std: (optional) standard deviation for every prediction
    :param x: (optional) ax
    :return: fig, ax
    """
    if not ax:
        ax = plt.gca()
    ax.scatter(x_truth, y_truth, label="Truth", color=theme["truth"])
    ax.plot(x_prediction, y_prediction, label="Prediction", color=theme["prediction_mean"])

    if std is not None:
        ax.fill_between(x_prediction.flatten(), y_prediction - std, y_prediction + std,
                        color=theme["prediction_std"], alpha=0.3, label="Predictive Variance")
    ax.legend()


def plot_mean_vs_truth_with_uncertainties(x_truth, y_truth, x_prediction, y_prediction,
                                          aleatoric, epistemic, ax=None):
    """
    Same as plot_mean_vs_truth but with the uncertainties splitted into aleatoric and epistemic.
    :param x_truth:
    :param y_truth:
    :param x_prediction:
    :param y_prediction:
    :param std:
    :return: fig, ax
    """
    if not ax:
        ax = plt.gca()

    ax.scatter(x_truth, y_truth, label="Truth", color=theme["truth"])
    ax.plot(x_prediction, y_prediction, label="Prediction", color=theme["prediction_mean"])

    # inner tube
    ax.fill_between(x_prediction.flatten(), y_prediction - aleatoric , y_prediction + aleatoric,
                    color=theme["aleatoric"], alpha=0.2, label="aleatoric")

    # two outer tubes
    ax.fill_between(x_prediction.flatten(), y_prediction - aleatoric , y_prediction - aleatoric - epistemic / 2.0,
                    color=theme["epistemic"], alpha=0.3, label="epistemic")

    ax.fill_between(x_prediction.flatten(), y_prediction + aleatoric , y_prediction + aleatoric + epistemic / 2.0,
                    color=theme["epistemic"], alpha=0.3)

    # if std is not None:
    #     ax.fill_between(x_prediction.flatten(), y_prediction - std, y_prediction + std,
    #                     color=theme["prediction_std"], alpha=0.3, label="Prediction std")
    # ax.legend()
```

Model

We used MC-dropout so we defined the model with dropout. We kept the model simple because the computational cost is big. You can notice the dropout layers between each dense layer

```
def combined_model(x, dropout_rate):
    """
    Model that combines aleatoric and epistemic uncertainty.
    Based on the "What uncertainties do we need" paper by Kendall.
    Works for simple 2D data.
    :param x: Input feature x
    :param dropout_rate:
    :return: prediction, log(sigma^2)
    """

    # with tf.device("/gpu:0"):
    keep_prob = 1 - dropout_rate

    fc1 = tf.layers.dense(inputs=x, units=64, activation=tf.nn.relu)
    fc1 = tf.nn.dropout(fc1, keep_prob)

    fc2 = tf.layers.dense(inputs=fc1, units=128, activation=tf.nn.relu)
    fc2 = tf.nn.dropout(fc2, keep_prob)

    fc3 = tf.layers.dense(inputs=fc2, units=264, activation=tf.nn.relu)
    fc3 = tf.nn.dropout(fc2, keep_prob)

    fc4 = tf.layers.dense(inputs=fc3, units=128, activation=tf.nn.softmax)
    fc4 = tf.nn.dropout(fc2, keep_prob)

    fc5 = tf.layers.dense(inputs=fc3, units=64, activation=tf.nn.softmax)
    fc5 = tf.nn.dropout(fc2, keep_prob)
    # Output layers has predictive mean and variance sigma^2
    output_layer = tf.layers.dense(fc5, units=y_shape+1)

    predictions = tf.expand_dims(output_layer[:, 0], -1)
    log_variance = tf.expand_dims(output_layer[:, 1], -1)

    return predictions, log_variance
```

Model parameters:

we defined the loss function, learning rate and optimizer. We defined the session and initialize it. You can notice how we included both the error in the predicition and the uncertainty in the loss function

```
def combined_training(x_truth, y_truth, dropout, learning_rate, epochs, display_step=500):
    """
    Generic training of a Combined (uncertainty) network for 2D data.
    :param x_truth: training samples x
    :param y_truth: training samples y / label
    :param dropout:
    :param learning_rate:
    :param epochs:
    :param display_step:
    :return: session, x_placeholder, dropout_placeholder
    """
    tf.reset_default_graph()
    x_placeholder = tf.placeholder(tf.float32, [None, X_shape])
    y_placeholder = tf.placeholder(tf.float32, [None, y_shape])
    dropout_placeholder = tf.placeholder(tf.float32)

    prediction, log_variance = combined_model(x_placeholder, dropout_placeholder)

    tf.add_to_collection("prediction", prediction)
    tf.add_to_collection("log_variance", log_variance)
    loss = tf.reduce_sum(0.5 * tf.exp(-1 * log_variance) * tf.keras.backend.sum(tf.square(tf.abs(y_placeholder - prediction)))+ 0.5 *
    # loss = tf.losses.reduce_sum(0.5 * tf.exp(-1 * log_variance) * tf.square(tf.abs(y_placeholder - prediction))+ 0.5 * log_variance
    optimizer = tf.train.AdamOptimizer(learning_rate)
    train = optimizer.minimize(loss)

    init = tf.global_variables_initializer()
    sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))
    sess.run(init)

    for epoch in range(epochs):
        feed_dict = {x_placeholder: x_truth.reshape(-1,X_shape),
                     y_placeholder: y_truth.reshape(-1,y_shape),
                     dropout_placeholder: dropout}

        sess.run(train, feed_dict=feed_dict)

        if epoch % display_step == 0:
            print("Epoch {}".format(epoch))
            current_loss = sess.run(loss, feed_dict=feed_dict)
            print("Loss {}".format(current_loss))
            print("================")

    print("Training done")

    return sess, x_placeholder, dropout_placeholder
```

MC-dropout:

Since we used MC-dropout, we run n_passes forward passes. Then we get the mean and the variance of the results to calculate the uncertainty

```
def combined_evaluation(x_train, y_train,x_test,y_test, dropout, learning_rate, epochs, n_passes):
    """
    :param x:`
    :param y:
    :param dropout:
    :param learning_rate:
    :param epochs:
    :param n_passes:
    :return:
    """
    sess, x_placeholder, dropout_placeholder = \
        combined_training(x_train, y_train, 0.2, learning_rate, epochs)

    prediction_op = sess.graph.get_collection("prediction")
    log_variance = sess.graph.get_collection("log_variance")
    aleatoric_op = tf.exp(log_variance)

    feed_dict = {x_placeholder: x_train,
                 dropout_placeholder: dropout}

    predictions = []
    aleatorics = []
    for _ in range(n_passes):
        prediction, aleatoric = sess.run([prediction_op, aleatoric_op], feed_dict)
        predictions.append(prediction[0])
        aleatorics.append(aleatoric[0])

    y_eval = np.mean(predictions, axis=0).flatten()
    epistemic_eval = np.var(predictions, axis=0).flatten()
    aleatoric_eval = np.mean(aleatorics, axis=0).flatten()
    total_uncertainty_eval = epistemic_eval + aleatoric_eval

    # plot_mean_vs_truth_with_uncertainties(x_train[:,1], y_train, x_test[:,1], y_test, aleatoric_eval, epistemic_eval, ax)

    # fig.suptitle("Dropout - Learning Rate %f, Epochs %d, Dropout %.3f, Passes %d" %(learning_rate, epochs, dropout, n_passes))

    # ax.fill_between(x_eval.flatten(), 0, epistemic_eval, label="epistemic", color="green", alpha=0.4)
    # ax.fill_between(x_eval.flatten(), 0, aleatoric_eval, label="aleatoric", color="orange", alpha=0.4)
    # ax.legend()
```

```
# combined_evaluation(X_train, y_train, X_test, y_test, 0.1, 1e-2, 5000, 300)
```

```
combined_evaluation(X_train, y_train, X_test, y_test, 0.1, 1e-3, 20000, 50)
```

```
Loss 248341.703125
================
Epoch 10500
Loss 230072.09375
================
Epoch 11000
Loss 218517.046875
================
Epoch 11500
Loss 213383.109375
================
Epoch 12000
Loss 212432.375
================
Epoch 12500
Loss 202293.75
================
Epoch 13000
Loss 191490.71875
================
Epoch 13500
Loss 184319.5
================
Epoch 14000
Loss 177053.875
================
Epoch 14500
Loss 167606.546875
================
Epoch 15000
Loss 158563.0625
================
Epoch 15500
Loss 147350.328125
================
Epoch 16000
Loss 137863.78125
================
Epoch 16500
Loss 128851.1640625
================
Epoch 17000
Loss 122917.25
================
Epoch 17500
Loss 119734.984375
================
Epoch 18000
Loss 116789.921875
================
Epoch 18500
Loss 114262.5
================
Epoch 19000
Loss 112654.953125
================
Epoch 19500
Loss 111270.328125
================
Training done
```