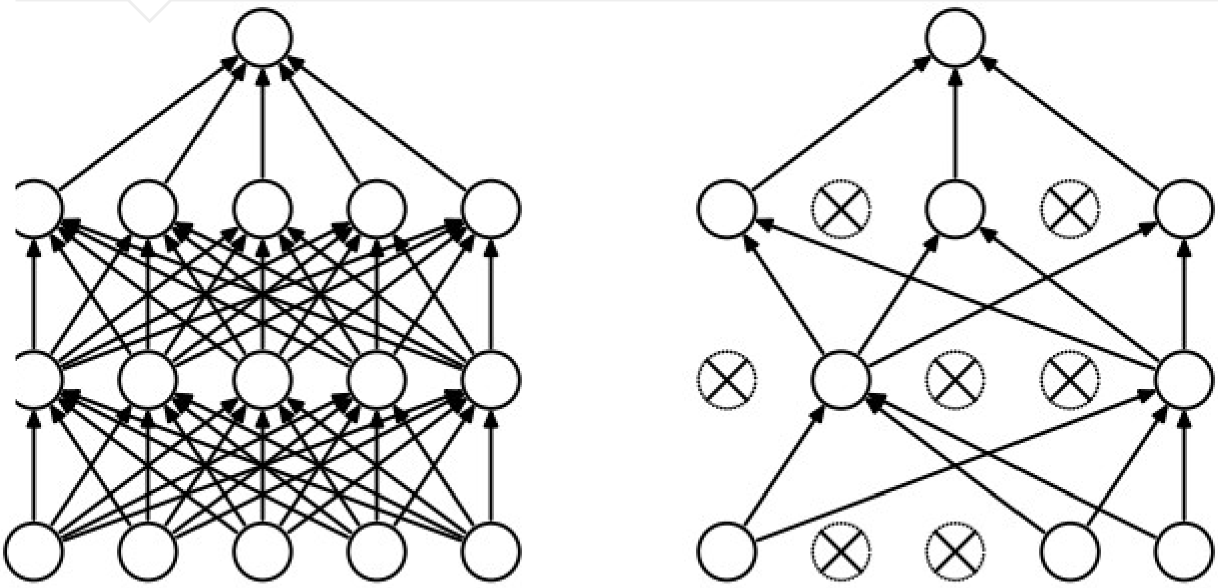
It's About Machine Learning, Data Science And More

Model Uncertainty In Deep Learning With Monte Carlo Dropout In Keras



(a) Standard Neural Net (b) After applying dropout.

Deep learning models have shown amazing performance in a lot of fields such as autonomous driving, manufacturing, and medicine, to name a few. However, these are fields in which representing model uncertainty is of crucial importance. The standard deep learning tools for regression and classification do not capture model uncertainty. In classification, predictive probabilities obtained at the end of the pipeline (the softmax output) are often erroneously interpreted as model confidence.

unjustified high confidence for points far from the training data.

Model uncertainty is indispensable for the deep learning practitioner as well. With model confidence at hand we can treat uncertain inputs and special cases explicitly. I recommend Vincent Warmerdam amazing talk "How to Constrain Artificial Stupidit" from PyData London 2019 for a general view on the topic. For example, in the case of classification, a model might return a result with high uncertainty. In this case we might decide to pass the input to a human for classification.

Gal et. al show that the use of dropout in neural networks can be interpreted as a Bayesian approximation of a Gaussian process, a well known probabilistic model. Dropout is used in many models in deep learning as a way to avoid over-fitting, and they show that dropout approximately integrates over the models weights. In this article we will see how to represent model uncertainty of existing dropout neural networks with keras. This approach, called Monte Carlo dropout, will mitigates the problem of representing model uncertainty in deep learning without sacrificing either computational complexity or test accuracy and can be used for all kind of models trained with dropout.

Load The MNIST Data

We will use the MNIST dataset to explore the proposed method of Monte Carlo dropout. We can conveniently load it with keras.

In [1]:

```
from __future__ import print_function
import keras
from keras.datasets import mnist
```

```
import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score

import matplotlib.pyplot as plt
plt.style.use("ggplot")
%matplotlib inline
```

```
Using TensorFlow backend.
```

In [3]:

```
batch_size = 128
num_classes = 10
epochs = 12

# input image dimensions
img_rows, img_cols = 28, 28
```

In [4]:

```
# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

In [5]:

```
if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)
```

In [6]:

```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

```
x_train shape: (60000, 28, 28, 1)
```

```
# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

Build Monte Carlo Model And Run Experiments

Modelling uncertainty with Monte Carlo dropout works by running multiple forward passes trough the model with a different dropout masks every time. Let's say we are given a trained neural network model with dropout . To derive the uncertainty for one sample we collect the predictions of inferences with different dropout masks. Here represents the model with dropout mask . So we obtain a sample of the possible model outputs for sample as

By computing the average and the variance of this sample we get an ensemble prediction, which is the mean of the models posterior distribution for this sample and an estimate of the uncertainty of the model regarding .

Note that the dropout NN model itself is not changed. To estimate the predictive mean and predictive uncertainty we simply collect the results of stochastic forward passes through the model. As a result, this information can be used with existing NN models trained with dropout.

without Monte Carlo dropout and compare their properties.

In [71]:

```
def get_dropout(input_tensor, p=0.5, mc=False):
    if mc:
        return Dropout(p)(input_tensor, training=True)
    else:
        return Dropout(p)(input_tensor)


def get_model(mc=False, act="relu"):
    inp = Input(input_shape)
    x = Conv2D(32, kernel_size=(3, 3), activation=act)(inp)
    x = Conv2D(64, kernel_size=(3, 3), activation=act)(x)
    x = MaxPooling2D(pool_size=(2, 2))(x)
    x = get_dropout(x, p=0.25, mc=mc)
    x = Flatten()(x)
    x = Dense(128, activation=act)(x)
    x = get_dropout(x, p=0.5, mc=mc)
    out = Dense(num_classes, activation='softmax')(x)

    model = Model(inputs=inp, outputs=out)

    model.compile(loss=keras.losses.categorical_crossentropy,
                  optimizer=keras.optimizers.Adadelta(),
                  metrics=['accuracy'])
    return model
```

In [72]:

```
model = get_model(mc=False, act="relu")
mc_model = get_model(mc=True, act="relu")
```

In [73]:

```
h = model.fit(x_train, y_train,
              batch_size=batch_size,
              epochs=10,
              verbose=1,
              validation_data=(x_test, y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 48s 802us/step - loss: 0.259
Epoch 2/10
```

```
60000/60000 [==============================] - 47s 780us/step - loss: 0.052
Epoch 5/10
60000/60000 [==============================] - 47s 780us/step - loss: 0.046
Epoch 6/10
60000/60000 [==============================] - 47s 781us/step - loss: 0.046
Epoch 7/10
60000/60000 [==============================] - 47s 780us/step - loss: 0.037
Epoch 8/10
60000/60000 [==============================] - 47s 781us/step - loss: 0.034
Epoch 9/10
60000/60000 [==============================] - 47s 783us/step - loss: 0.036
Epoch 10/10
60000/60000 [==============================] - 47s 781us/step - loss: 0.036
```

In [74]:

```
# score of the normal model
score = model.evaluate(x_test, y_test, verbose=0)

print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.029758870216909507
Test accuracy: 0.9914
```

In [76]:

```
h_mc = mc_model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=10,
                    verbose=1,
                    validation_data=(x_test, y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 48s 793us/step - loss: 0.262
Epoch 2/10
60000/60000 [==============================] - 46s 774us/step - loss: 0.087
Epoch 3/10
60000/60000 [==============================] - 46s 774us/step - loss: 0.066
Epoch 4/10
60000/60000 [==============================] - 47s 777us/step - loss: 0.055
Epoch 5/10
60000/60000 [==============================] - 47s 777us/step - loss: 0.046
Epoch 6/10
60000/60000 [==============================] - 46s 775us/step - loss: 0.046
```

```
Epoch 9/10
60000/60000 [==============================] - 47s 776us/step - loss: 0.031
Epoch 10/10
60000/60000 [==============================] - 47s 778us/step - loss: 0.029
```

In [93]:

```python
import tqdm

mc_predictions = []
for i in tqdm.tqdm(range(500)):
    y_p = mc_model.predict(x_test, batch_size=1000)
    mc_predictions.append(y_p)
```

```
100%|██████████| 500/500 [17:02<00:00,  2.04s/it]
```

In [97]:

```python
# score of the mc model
accs = []
for y_p in mc_predictions:
    acc = accuracy_score(y_test.argmax(axis=1), y_p.argmax(axis=1))
    accs.append(acc)
print("MC accuracy: {:.1%}".format(sum(accs)/len(accs)))
```

```
MC accuracy: 98.6%
```

In [98]:

```python
mc_ensemble_pred = np.array(mc_predictions).mean(axis=0).argmax(axis=1)
ensemble_acc = accuracy_score(y_test.argmax(axis=1), mc_ensemble_pred)
print("MC-ensemble accuracy: {:.1%}".format(ensemble_acc))
```

```
MC-ensemble accuracy: 99.2%
```

Look at the distributions of the monte carlo predictions and in blue you see the prediction of the ensemble.

In [99]:

```python
plt.hist(accs);
plt.axvline(x=ensemble_acc, color="b");
```

probability and uncertainty.

In [103]:

```
idx = 247
plt.imshow(x_test[idx][:,:,0])
```

Out[103]:

```
<matplotlib.image.AxesImage at 0x7fa9ad2d5b38>
```

In [104]:

```
p0 = np.array([p[idx] for p in mc_predictions])
print("posterior mean: {}".format(p0.mean(axis=0).argmax()))
print("true label: {}".format(y_test[idx].argmax()))
print()
# probability + variance
for i, (prob, var) in enumerate(zip(p0.mean(axis=0), p0.std(axis=0))):
    print("class: {}; proba: {:.1%}; var: {:.2%} ".format(i, prob, var))
```

```
posterior mean: 4
true label: 4

class: 0; proba: 0.0%; var: 0.20%
class: 1; proba: 9.8%; var: 19.46%
class: 2; proba: 12.5%; var: 22.34%
class: 3; proba: 0.0%; var: 0.01%
class: 4; proba: 56.0%; var: 34.90%
class: 5; proba: 0.1%; var: 1.17%
class: 6; proba: 21.0%; var: 28.19%
class: 7; proba: 0.1%; var: 1.67%
class: 8; proba: 0.5%; var: 4.71%
class: 9; proba: 0.0%; var: 0.02%
```

In [105]:

```
x, y = list(range(len(p0.mean(axis=0)))), p0.mean(axis=0)
plt.plot(x, y);
```

In [106]:

```
fig, axes = plt.subplots(5, 2, figsize=(12,12))

for i, ax in enumerate(fig.get_axes()):
```

We see, that the model is correct but fairly uncertain. This seems to be a hard example. We would probably let a human decide what to do with this example.

Find The Most Uncertain Examples

Next, we find the most uncertain examples. This can be useful to understand your dataset or where the model has problems.

1. Selection By Probability

First we select images by the predictive mean, our probability.

In [107]:

```
max_means = []
preds = []
for idx in range(len(mc_predictions)):
    px = np.array([p[idx] for p in mc_predictions])
    preds.append(px.mean(axis=0).argmax())
    max_means.append(px.mean(axis=0).max())
```

In [108]:

```
(np.array(max_means)).argsort()[:10]
```

Out[108]:

```
array([247, 175, 115, 445, 340, 320,  62, 321, 259, 449])
```

In [109]:

```
plt.imshow(x_test[247][:,:,0])
```

Out[109]:

```
<matplotlib.image.AxesImage at 0x7fa9ad1f7eb8>
```

2. Selection By Variance

Now we can select the images by the variance of the predictions.

In [110]:

```
max_vars = []
for idx in range(len(mc_predictions)):
    px = np.array([p[idx] for p in mc_predictions])
    max_vars.append(px.std(axis=0)[px.mean(axis=0).argmax()])
```

In [111]:

```
(-np.array(max_vars)).argsort()[:10]
```

Out[111]:

```
array([247, 259,  62, 449, 115, 320, 445, 492, 211, 326])
```

In [112]:

```
plt.imshow(x_test[259][:,:,0])
```

Out[112]:

```
<matplotlib.image.AxesImage at 0x7fa9af4fb3c8>
```

How Is The Uncertainty Measure Behaved In Random Regions Of The Feature Space?

One important test is how well can the uncertainty estimate identify out-of-scope samples. Here we just create random images and see what the model predicts.

In [118]:

```
random_img = np.random.random(input_shape)
```

In [119]:

```
plt.imshow(random_img[:,:,0]);
```

```
    y_p = mc_model.predict(np.array([random_img]))
    random_predictions.append(y_p)
```

```
100%|██████████| 500/500 [00:01<00:00, 258.86it/s]
```

In [122]:

```
p0 = np.array([p[0] for p in random_predictions])
print("posterior mean: {}".format(p0.mean(axis=0).argmax()))
print()
# probability + variance
for i, (prob, var) in enumerate(zip(p0.mean(axis=0), p0.std(axis=0))):
    print("class: {}; proba: {:.1%}; var: {:.2%} ".format(i, prob, var))
```

```
posterior mean: 8

class: 0; proba: 1.8%; var: 2.03%
class: 1; proba: 0.7%; var: 1.24%
class: 2; proba: 4.7%; var: 4.87%
class: 3; proba: 8.8%; var: 8.90%
class: 4; proba: 1.1%; var: 2.25%
class: 5; proba: 4.6%; var: 6.50%
class: 6; proba: 1.5%; var: 1.83%
class: 7; proba: 0.2%; var: 0.51%
class: 8; proba: 76.1%; var: 14.25%
class: 9; proba: 0.5%; var: 1.08%
```

In [123]:

```
x, y = list(range(len(p0.mean(axis=0)))), p0.mean(axis=0)
plt.plot(x, y);
```

Wow, this is bad! The model is pretty certain that this is an eight. But it is clearly just random noise. If you try different random images, you will find that the model always predicts them as eight. So there might be something wrong with the "understanding" of eight in our model. This is good to know and keep in mind when using the model.

In [124]:

```
fig, axes = plt.subplots(5, 2, figsize=(12,12))
```

```
    ax.label_outer()
```

Looking at the probability distributions of the predictions for different classes we can probably find a way to identify out-of-scope samples by using higher moments than just mean and variance. Try it and let me know what you find.

That's all for now. We saw a simple but effective way to derive uncertainty estimates for deep learning models, a useful tool in your machine learning toolbox. I hope you liked this tutorial and it will be helpful to you. Watch out for more to come on this topic!

You Might Also Be Interested In:

- Guide to sequence tagging with neural networks in python: Named entity recognition series: Introduction To Named Entity Recognition In Python Named Entity Re ...

- Image segmentation with test time augmentation with keras: In the last post, I introduced the U-Net model for segmenting salt depots in seismic images. This ti ...

- U-Net for segmenting seismic images with keras: Today I'm going to write about a kaggle competition I started working on recently. I will show you ...

Tobias Sterbak

08/05/2019

Uncategorized

Algorithms, computer vision, deep learning

Previous post

Next post

1 Comment

Kusha
08/30/2019 at 4:02 am

Hi!

This was super useful.

I've come across many papers such as "Dropout as a bayesian approximation", "Selective Classification for DNNs", "SelecitiveNet", etc., which employ MC dropout on image classification tasks.

Thanks! for a comprehensive explanation.

I have 2 question:

1. How do higher moments help with knowing more about the out-of-scope samples? Till now, I just had the idea that, variance is a pretty good estimator of uncertainty (it directly gives the spread of the distribution), hence I always used this as a measure of uncertainty (or negative variance as confidence).

the network for a number of times. However, the papers state running each sample one by one, a number of times through the network. In such case, model.predict won't work. Any reason I should not run samples one by one and rather stick with all at once like you have done?

Leave A Reply

Your email address will not be published.

Name

Email

Website

I agree with the storage and handling of my data by this website. *

SEARCH

Search form

nprint

rivacy Policy

SUBSCRIBE

Always get the latest posts:

Your email address

Sign up

 UP ↑