Neural Networks: CNN Generators, Interpretability

CS 360 Machine Learning
Week 12, Day 1

April 16, 2024

Contents

1 CNN Generators 1

2 Interpretability 2
- 2.1 LIME 2
- 2.2 Saliency Maps 3
- 2.3 Other methods 3

1 CNN Generators

Recall our discussion of GANs from last time, where generators are trying to create fake data and discriminators are trying to tell the difference between real and fake data.

The GAN discriminator is often a normal CNN. Recall that convolutional filters are generally smaller than the input and work to distill down information to smaller forms to eventually make a final prediction. If there is no padding and no strides you end up with a smaller output. You can also use pooling to distill down quickly. (Recall that sometimes you'll want padding so that you can match the output size to the input size.) In general, we think about normal convolutional filters as taking a larger matrix (often an image) and distilling it down to a smaller matrix that's passed to the next layer.

The GAN generator on the other hand is starting with something small and trying to expand it to an image; this often starts with a simple 100-entry vector input of noise. These might be, e.g., 100 random samples from a normal distribution. How can you go from something that's lower dimensional to higher dimensional – it seems like you're getting something for free?! But we can still use convolutions to do this essentially by upsampling and upscaling your image. You generally start with a lot of convolutional filters in the first layer and decrease the convolutions as you go through the layers. In these cases, the convolutions go over the sparse earlier layers using reshaping and padding so that there is a larger dimension in the output. Strides in this case can be used to spread out the input, filled with 0s between the input data. The result of each convolutional filter is a larger dimensional matrix than the input. The element-wise dot products encode some similarity between the image and the filter, with large numbers indicating high similarity. We can still think of each of these filters as for a feature in an image, e.g., an eye, and those features will be created by those filters. The convolutional filters will be learned and so each have special skills based on the target images or other data the generator is trained to create.

Why do we just use random noise to start the generator? While the discriminator takes real images, the generated images start from random noise with random start weights for the generator. This won't create good images! But recall that the discriminator will start out correctly identifying this random noise as fake, so the generator will be motivated by its loss function to move closer to real images. This will set off the joint learning process where the discriminator gets better at distinguishing real and fake images while the generator gets better at fooling the discriminator.

Suppose that the real-world images given to the discriminator are handwritten digits. Sometimes the generator could get very good during the learning process at generating the number 3, this could potentially be a well-written 3, but it wouldn't appropriately be sampled from the wider variety of possible numbers; this is called *mode collapse*. There are various changes that can be made to motivate the generator to create a wider variety of results.

2 Interpretability

In general, there are two broad types of interpretability and explainability; local interpretability and global interpretability. Local interpretability focuses on explaining a model's prediction on a specific example. We saw some of this in CS 260 where we considered what feature was the most important for a specific prediction. Global interpretability considers what the model learns overall; what features or other information about the model is important for the model's outputs. For example, when considering CNNs we might look at its filters. For today, we'll mostly talk about local interpretability.

Why do we want model explanations? Suppose that you're trying to determine if someone has the flu. A model might make a prediction, but having the additional explanation containing, e.g., symptoms the person has, can help a doctor make a decision about whether to trust the model's prediction. We'd like to be able to do this even in the case of complex models like neural networks.

2.1 LIME

We'll start by discussing LIME (introduced by this paper); this is a model-agnostic explanation approach, meaning that it can be used in a black-box way with any type of model. When thinking about model explanations, it's useful to think about predicted outcomes wholistically, i.e., not just thinking about the specific result of a maximum from softmax but also the whole probability distribution. Probabilities close to the max but not the maximum can help us understand what was important to the model. LIME highlights features as well as weights on specific features as a way of trying to understand the model's predictions at a specific point.

First, some notation. Let $x \in \mathbb{R}^d$ be one example and let $x' \in \{0,1\}^d$ be an interpretable version of that example. For text, this is the presence or absence of vocabulary words. For images, this is the presence or absence of pixel groups. For other data, this could be presence or absence of a feature. Encoding your data as 0 or 1 features helps the interpretability method to indicate importance. We'll let g be the explanation model. For our purposes g will be linear, but it can be generalized. Specifically:

$$g(z') = W_g \cdot z'$$

where W_g is the weights of the model g. Note that we've seen this before; whenever we found that there was a high weight (large absolute value) on a feature, we might say that it was important. Our goal is to find W_g.

Our original model (that we're trying to explain) is $f : \mathbb{R}^d \to \mathbb{R}$. We think about this as a model that's harder to understand inherenetly, e.g., a neural network, SVM, etc. Let $\pi_x(z)$ be the kernel function (a measure of similarity between z and x). We let k be the length of the explanation; this is the number of features we'll allow to be "important" for the purposes of explanation (in order to avoid an explanation so long as to be useless, e.g., simply outputting all the features of an image). Let x (defined above) be the example we want to explain the model's predictions on.

LIME Algorithm:

```
$Z = []$
for i = 1,2, ..., N (number of samples):
    $z_i \leftarrow sample\_around(x)$
    Z.append($z_i', f(z_i), \pi_x(z_i)$)      % features, targets, similarities
$W_g \leftarrow$ train_linear_regression(Z, k)
```

The above algorithm proceeds by sampling points from a neighborhood around x'. These points are run through the original model. We also get the similarities of these points. We'll create a weighted linear model based on these similarities. We train a LASSO method that penalizes too many non-zero weights (where the number of non-zero weights is k above, equal to the number of important features) based on the sampled points and similarities. The resulting model weights are the explanation of the model f on the specific example x. We have trained a linear model based on the local sampled points near to x and the original model predictions, with the goal that this interpretable model locally approximates the original model.

2.2 Saliency Maps

Another method we'll frequently use for interpretability is saliency maps. These have a lot of the same goals as LIME; identifying features that are important to a specific example's prediction. In general, saliency maps are often used on images and highlight the pixels from an image that would impact the classification scores the most if changed slightly. Saliency maps are often based on gradients. They can be useful for giving us an intuition for how a model is making predictions.

Consider some image I_0, class c (e.g., a prediction of "dog"), and model prediction $S_C(I)$. If this model were linear, we would be able to say $S_C(I) = W_c \cdot I + b_c$ where W_c are the weights for this class and I is a flattened version of the class and b_c is the bias for that class. But we're likely using a more complex model like a neural network! Again, we'll try to approximate a more complex model with a linear model at I_0 via a Taylor expansion:

$$S_c(I) \approx w_c \cdot I + b_c$$

$$\frac{\partial S_c(I)}{\partial I}\Big|_{I_0} = w_c$$

Changing any of the pixels of I_0 shows us how much the class prediction would change. This relies on the local gradient around a pixel in an image.

There are some important caveats when considering saliency maps! Some saliency methods don't actually appropriately capture model predictions, but are instead just based on the input image... but our goal was to explain the model. So it's important to be sure you're using a method that's been shown to genuinely capture information about the model's predictions.

2.3 Other methods

In general, both of these methods try to model a complex model with a more interpretable linear model. This technique can be generalized to non-linear but interpretable models such as decision trees (see, e.g., this paper which models an SVM with a decision tree). In general, the predictions of the original models are used as the target to train an interpretable model.

Another way to examine neural networks to make them more interpretable is to consider correlations between CNN filters and separate summary statistics that are interpretable in a specific domain, such as biology, to understand what those filters may be learning.