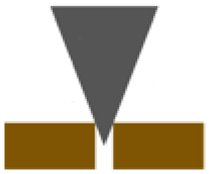
Wedge Dropout

Wedge Dropout drives apart CNN feature maps which are too close together.



Abstract

Wedge Dropout improves a CNN network by examining pairs of output feature maps and providing negative feedback if they are strongly correlated. Wedge Dropout is effective in almost all Convolutional Network pipelines. Preliminary testing has shown that it can give a very slight improvement to model accuracy, generally 0.05% to 0.1%, in many different 1D, 2D and 3D CNN-based models. Usually no hyperparameter tuning is required to achieve this improvement. Wedge Dropout has no run-time overhead. Like other Dropout algorithms, it is only active during training.

Introduction

Wedge Dropout is a Dropout algorithm tuned for convolutional networks. It is one of a recent wave of Dropout algorithms which are customized for Convolution Neural Networks (CNNs).

The Wedge Dropout algorithm is a variation of Spatial Dropout. Where Dropout zeros out individual values in a feature map, Spatial Dropout randomly zeros out entire feature maps. Since data within a feature map is strongly correlated, simple Dropout is not effective because the remaining data is still strongly correlated.

[Assumptions: understand neural networks, CNNs, and the role Dropout plays] A CNN works by generating overlapping pyramids based on an input image. These pyramids are called "feature maps". Given a picture of a cat, one feature map outlines the head, another the eyes, a third the ears. Collectively they describe aspects of the cat. Feature maps should be independent- if they are correlated, or "too similar", the CNN describes fewer unique aspects than it could.

The Wedge Dropout algorithm works by analyzing the final output of a convolutional neural network (CNN). Where Spatial Dropout zeroes out randomly chosen feature maps, Wedge Dropout analyzes randomly chosen pairs of feature maps and zeroes out both when they are "too similar".

Similar Work

Wedge Dropout is a combination of two recent trends in dropout research: CNN-specific dropout strategies and critiquing feature maps.

CNN-Specific Dropout Strategies

SpatialDropout and Dropblock are dropout strategies specifically designed around the nature of CNN feature maps. CNN feature maps are strongly correlated within the feature map, but different feature maps are not strongly correlated.

SpatialDropout

SpatialDropout zeroes out randomly chosen feature maps, to discourage them from becoming too correlated. It is best used right after the first convolutional layer.

DropBlock

DropBlock randomly chooses sub-rectangles in feature maps and randomly zeroes them out. Again, this discourages feature maps from becoming strongly correlated. It is not clear where it should be used in the network

Critiquing Feature Maps

It is possible to analyze feature maps and provide feedback when some of them become non-performant. This seems to be a recent invention.

Under this method, a layer scores all of the feature maps generated for an image. The score determines whether the feature map will be interfered with in some way. It could be zeroed out entirely, or the scoring function could indicate a specific region of the feature map that should be disrupted. We will examine two of these, CamDrop and InfoDrop.

CamDrop

In CamDrop, feature maps are analyzed and then a rectangular region is zeroed out in multiple, but not all, feature maps. This clearly can discourage correlation.

InfoDrop

CNNs are notorious for fixating on textural features to the detriment of spatial features. InfoDrop addresses this by finding regions of a feature map which correspond to a texture in the original image. It then zeroes out the corresponding area in the feature map.

Summary

In a sense, Wedge Dropout is to Spatial Dropout as CamDrop is to DropBlock: it replaces randomized dropout with a feedback mechanism based on analyzing feature maps. Wedge Dropout was directly inspired by CamDrop.

Algorithms

CNN feature maps generally have a low-valued "background" and high-valued "hills", where the hills are the feature(s) found in the original image. Two algorithms were investigated for Wedge Dropout's analysis phase. Both specifically ignore the background and only compare the hills, by using the mean value. It is not clear whether either algorithm is more robust across different CNN use cases.

A third algorithm, Rectification, is appropriate for fully populated vector spaces and was not further investigated.

Direct Comparison

This algorithm isolates the cells in each feature map which are above its mean value, and then compares individual values across both feature maps cell by cell in a simple Boolean AND operation. This counts the number of cells in both feature maps that tend to find the same feature. If this count is above a certain percentage of the total number of cells in the feature map, both feature maps are zeroed out.

Normalize and Multiply

This algorithm normalizes both feature maps to a range from 0.0 to 1.0, multiplies the two feature maps cellwise (Hadamard matrix multiplication), and counts the number of values above the mean of the resulting output. Again, if more than a certain percentage are above the mean, both feature maps are zeroed out.

Rectification

Under rectification, the result is the delta of the two feature maps. Since this would encourage the feature maps to be the inverse of the other, Rectification first normalizes the images around zero and rectifies them. Rectification nullifies the drive towards simply inverting half of the image.

Following are implementations of each algorithm in Python, using the following feature maps:

```
fmap1 = np.asarray([[1,2],[3,4]])
fmap2 = np.asarray([[3,1],[4, 9]])


print('Feature Map #1')
print(fmap1)
print('Feature Map #2')
print(fmap2)
```

```
Feature Map #1
[[1 2]
 [3 4]]
Feature Map #2
[[3 1]
 [4 9]]
```

```python
import numpy as np
```

```python
def similarity_multiply_norm(img1, img2):
    factor1 = np.linalg.norm(img1)
    factor2 = np.linalg.norm(img2)
    if factor1 == 0:
        factor1 = 0.0001
    if factor2 == 0:
        factor2 = 0.0001
    norm1 = img1 / factor1
    norm2 = img2 / factor2
    # normalized maps are in the same range
    mult = norm1 * norm2
    avg = np.mean(mult)
    correlated = mult > avg

    percentage = sum(correlated.flatten()) / len(img1.flatten())
    return percentage

def similarity_direct_comparison(img1, img2):
    mean1 = np.mean(img1)
    mean2 = np.mean(img2)

    visible1 = img1 > mean1
    visible2 = img2 > mean2

    correlated = visible1 == visible2

    percentage = sum(correlated.flatten()) / len(img1.flatten())

    return correlated, percentage

def similarity_multiply(img1, img2):
    # norm both to 0->1, multiply to produce 0->1
    min1 = np.min(img1)
    min2 = np.min(img2)
    base1 = np.max(img1) - min1
    base2 = np.max(img2) - min2
    if base1 == 0:
        base1 = 0.0001
    if base2 == 0:
        base2 = 0.0001
    norm1 = (img1 - min1) / base1
    norm2 = (img2 - min2) / base2
    #print('norm1:', norm1)
    #print('norm2:', norm2)
    # rectified maps range from 0 to 1
    mult = norm1 * norm2
    #print('mult:', mult)
    avg = np.mean(mult)
    #print('avg:', avg)
    correlated = mult > avg

    percentage = sum(correlated.flatten()) / len(img1.flatten())
    return correlated, percentage

def similarity_rectify(img1, img2):
    # normalize values in both images to a range from 0 to 1
    # somehow, there is no numpy function for this
    min1 = np.min(img1)
    min2 = np.min(img2)
    base1 = np.max(img1) - min1
    base2 = np.max(img2) - min2
    if base1 == 0:
        base1 = 0.0001
    if base2 == 0:
        base2 = 0.0001
    norm1 = (img1 - min1) / base1
    norm2 = (img2 - min2) / base2
    #print('norm1:', norm1)
    #print('norm2:', norm2)
    rect1 = np.abs(norm1 - 0.5)
    rect2 = np.abs(norm2 - 0.5)
    #print('rect1:', rect1)
    #print('rect2:', rect2)
    # rectified maps range from 0 to 1
    delta = (rect1 - rect2) / 2
    mean = np.abs(np.mean(delta))
    return 1 - mean

correlation1, percentage1 = similarity_direct_comparison(fmap1, fmap2)
print('Direct Comparison similarity score:', percentage1)
correlation2, percentage2 = similarity_multiply(fmap1, fmap2)
print('Normalize and Multiply similarity score:', percentage2)
percentage3 = similarity_rectify(fmap1, fmap2)
print('Rectification similarity score:', percentage3)


```

```
Direct Comparison similarity score: 0.75
Normalize and Multiply similarity score: 0.25
Rectification similarity score: 0.9947916666666666
```

We can see that the three algorithms provide different interpretations of 'correlated'.

Usage

Wedge Dropout operates by critiquing the values created by the CNN pipeline, and is best used at the end of a pipeline of CNN layers. Like Spatial Dropout and other CNN-specific Dropout algorithms, do not place a Batch Normalization layer after a Wedge Dropout layer. Wedge Dropout works well after a Batch Normalization layer. All successful tests have placed the Wedge Dropout layer right before the final summarization layer, usually a Dense or GlobalAveragePooling layer.

[Lift a standard CNN diagram and poke in a Wedge Dropout layer right before the final Dense/GlobalAveragePooling layer.]

Wedge Dropout only has one hyperparameter, the similarity coefficient. Preliminary testing on many different 1D, 2D and 3D CNN networks has shown that one value is optimal for almost all applications: 0.5 for the Direct Comparison algorithm, and 0.65 for the Normalize and Multiply algorithm.

Batch-wise Operation

It has proven very powerful to apply Wedge Dropout to all of the feature maps for a pair of random indexes, and then do a simple voting algorithm on the results. For example, if the batch size is 32, then if 16 pairs of feature maps are "too similar", then all 32 feature maps are zeroed out. If only 15 pairs are too similar, none of the feature maps are zeroed out.

When operating per sample, Wedge Dropout critiques a pair of feature maps. In batch-wise operation, Wedge Dropout critiques the engine, or causal chain, that created the feature maps.

Batch Normalization is a proven method of improving a CNN model, and is used in most reference architectures except image generators. Batch Normalization's performance improves as the batch size increases. Wedge Dropout's performance also increases as batch size increases, so it is a good match for existing CNN architectures. Wedge Dropout works well after a Batch Normalization layer.

Concluding Remarks

The Wedge Dropout algorithm can add a noticeable increase in tensor graph compilation time, and training time per epoch. For example, the final phase of EfficientNet "Zero" generates 1024 feature maps, and then applies GlobalAveragePooling to those. Wedge Dropout was most effective when inserted between the final convolution layer and the GlobalAveralPooling layer. Adding Wedge Dropout increased the tensor graph compilation phase by 20%, and added 10%-15% to the running time for each epoch. Wedge Dropout is not active during prediction, and does not contribute any values that need to be loaded for a model.

Utility

Wedge Dropout is not needed for any image architecture. However, given the lack of complex hyperparameters and simplicity of application, it will probably improve most production uses of CNNs. It has been tested with many example networks in the Keras documentation, and worked in all Conv1D, Conv2D and Conv3D-based applications. It did not improve a multi-layer fully trained networks (EfficientNet Level 0), but did improve an application which uses pre-trained ImageNet layers. It also did not help with LSTM2D.

Since Wedge Dropout works by critiquing the output of a CNN, it is possible that its feedback only works in a simple causal chain. It is possible that the causal graph of feature map creation is turbid in deep networks and 2D LSTMs.

Further Investigation

"Slice Dropout" [SliceDropout], a variant of Spatial Dropout, zeroes out only one half of a feature map instead of the entire feature map. A more complex version of Wedge Dropout's feature map comparison could decide that all of the offending cells are concentrated on one side of the feature map, or even just one quadrant. It would choose to zero out only that area.

It is possible that zeroing out values is not the only way to affect training. There may be ways to do a random fill which do not disrupt the operation of Batch Normalization.

Feature Maps and Attention Heads

The multi-head attention [cite] architecture creates several "answer" vectors of the same size, and combines them with addition. This set of vectors looks suspiciously like a set of feature maps: the information within the vector is strongly correlated, but information across vectors is not correlated. In fact, if the information is correlated across attention heads, the attention heads are learning the same answer set. It is possible that Wedge Dropout, or even Spatial Dropout, will improve the function of a multi-head attention model.

Citations

[CamDrop]

[DropBlock]

[SpatialDropout paper]

[SpatialDropout Keras man page]