SongRNN: Music Generation Through a Character Level LSTM

Brandon Szeto
Jacobs School of Engineering
University of California, San Diego
San Diego, CA 92122
bszeto@ucsd.edu

Darren Yu
Jacobs School of Engineering
University of California, San Diego
San Diego, CA 92122
dmyu@ucsd.edu

Nathaniel Thomas
Jacobs School of Engineering
University of California, San Diego
San Diego, CA 92122
nathomas@ucsd.edu

Abstract

This abstract provides a succinct overview of our work focused on semantic segmentation using the PASCAL VOC-2007 dataset. Our approach involves leveraging a novel deep learning architecture tailored for semantic segmentation tasks. We meticulously preprocess the dataset, addressing challenges such as class imbalance and data augmentation to enhance model generalization. Through experimentation, we explore hyperparameter tuning strategies to optimize performance. Our results showcase a notable improvement in both percent correct and Mean Intersection over Union (IoU) metrics, underscoring the efficacy of our methodology. Additionally, we uncover intriguing insights into the model's behavior, shedding light on its strengths and potential areas for refinement. This work showcases a framework for accurate and nuanced semantic segmentation.

Introduction

Semantic segmentation is the task of assigning each pixel in an image a specified label. A single image can be partitioned into multiple meaningful segments corresponding to a given object or region of interest. Labels are assigned to individual pixels, marking boundaries and shapes. Altogether, this task has many applications in computer vision including but not limited to autonomous driving, video surveillance, and object recognition.

In recent years, convolutional neural networks (CNNs) have proven to be an effective approach to semantic segmentation. Such models learn hierarchical features by capturing information using convolutional blocks. Convolutions and related techniques such as weight initialization, batch normalization, and pooling help avoid the flat, dense layers of a traditional fully connected neural network allowing for less computation and improved feature maps.

Xavier Weight Initialization

In our architecture, we use Xavier weight initialization. In Uniform Xavier Initialization, a layer's weights are chosen from a random uniform distribution bounded between

$$\pm\frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}} \tag{1}$$

where n_i is the number of incoming network connections and n_{i+1} is the number of outgoing network connections.

In Normal Xavier Initialization, a layer's weights are chosen from a normal distribution with

$$\sigma = \frac{\sqrt{2}}{\sqrt{n_i + n_{i+1}}} \tag{2}$$

where n_i is the number of incoming network connections and n_{i+1} is the number of outgoing network connections.

The Xavier initialization was created in response to the problem of vanishing and exploding gradients in deep neural networks in the context of symmetric nonlinearities (sigmoid, tanh). The intuition is that the weights should not be intialized randomly, but rather proportional to the size of two connected layers. As a result, the variance of activations and gradients would be maintained through the layers of a deep network.

Kaiming Weight Initialization

With the rise of ReLU, the nonlinearity can no longer be assumed to be symmetric. As such, the assumptions made by the Xavier Weight Initialization fall apart. In 2015, He et. al demonstrated that a ReLU layer typically has a standard deviation close to

$$\sqrt{\frac{n_i}{2}} \tag{3}$$

where n_i is the number of incoming network connections. As such, weights initially chosen from a normal distribution should be weighted by $\sqrt{\frac{n_i}{2}}$, with bias tensors initalized to zero.

Batch Normalization

During the training of a neural network, at each layer, inputs (activations) change as the parameters of the previous layers get updated. Batch normalization normalizes the activations of each layer by subtracting the mean and dividing by the standard deviation of the activations within a mini-batch. We implement batch normalization using PyTorch's `nn` module implementation. For example, we define a layer used for batch normalization as

```
self.bnd1 = nn.BatchNorm2d(32)
```

This ensures that the inputs to each layer have a consistent distribution, which helps in stabilizing the training process.

Pixel Accuracy

Pixel accuracy is a straightforward evaluation metric commonly used in image classification tasks to measure the percentage of correctly classified pixels in an image. It provides a simple measure of overall accuracy without considering class imbalance.

Pixel accuracy is calculated as:

$$\text{Pixel Accuracy} = \frac{\text{Number of Correctly Classified Pixels}}{\text{Total Number of Pixels}} \tag{4}$$

Where:

- Number of Correctly Classified Pixels: The count of pixels for which the predicted class matches the ground truth class.
- Total Number of Pixels: The total count of pixels in the image.

While pixel accuracy provides a quick measure of overall performance, it may not be the most informative metric for tasks with class imbalance or when individual classes are of interest.

Intersection over Union (IoU)

Intersection over Union (IoU), also known as the Jaccard index, is a popular evaluation metric in semantic segmentation tasks. It measures the overlap between the predicted segmentation mask and the ground truth mask for each class. IoU provides a more nuanced understanding of model performance by considering both true positives and false positives.

For a given class c, IoU is calculated as:

$$IoU_c = \frac{\text{Area of Intersection between predicted and ground truth mask for class } c}{\text{Area of Union between predicted and ground truth mask for class } c} \tag{5}$$

The overall IoU for all classes is often computed as the mean or weighted mean of individual class IoU scores:

$$IoU = \frac{1}{N} \sum_{c=1}^{N} IoU_c \tag{6}$$

Where:

- N is the total number of classes.
- IoU_c is the IoU score for class c.

IoU ranges from 0 to 1, with higher values indicating better segmentation accuracy. A value of 1 indicates perfect overlap between the predicted and ground truth masks, while a value of 0 indicates no overlap.

IoU is particularly useful for tasks where precise localization and segmentation accuracy are essential, such as medical image analysis, object detection, and scene understanding. It provides insights into how well the model captures the spatial extent of different classes within the image.

Related Works

U-Net

U-Net Briot et al. [2019] has proven to be highly effective for biomedical image segmentation tasks. Its architecture features a contracting path to capture context and an expansive path for precise localization. The innovative use of skip connections facilitates the flow of fine-grained details across different layers, enhancing the model's ability to accurately delineate object boundaries.

Methods

In the Methods section, you should describe the implementation and architectural details of your system - in particular, this addresses how you approached the problem and the design of your solution. For those who believe in reproducible science, this should be fine-grained enough such that somebody could implement your model/algorithm and reproduce the results you claim to achieve.

Training network using Teacher forcing

Describe the LSTM architecture, the training procedure, the loss criterion, the optimizer you used.

Song Generation

Describe the approach you took to generate a song by priming the network with an initial sequence given to it. You should also discuss the details of incorporating Temperature (T) in generating the output.

Hyper-parameter Tuning

Describe the RNN architecture you used in 5.a and briefly describe the approach you took in tuning your hyperparameters.

Feature Evaluation

Describe why it is important to do feature evaluation. Describe the approach you took in generating the Heatmap of the activation's of each neurons for each of the characters.

Results

In the Results section, you should demonstrate your baseline model's performance, performances for each of the parts of Q3, Q4, Q5 and Q6 (training, generation, hyperparameter tuning, feature evaluation). You are expected to report the following things in your submission:

a) Report results for your baseline model, a 1-hidden layer LSTM with 150 neurons. (2 pt)

b) Report the three generated music samples with the above-mentioned three different Temperature (T) settings. Submit their .txt and .midi files as a part of your gradescope submission. (3 pt)

c) Report loss plots for hyperparameter tuning experiments namely: RNNs vs LSTMs, changing number of neurons in the hidden layer and varying dropouts. (6 pt)

d) Report heatmaps with appropriate scales, for three different neurons from your best-trained model. (3 pt)

Discussion

Please discuss the following important points as well: How did the qualitative performance and loss of RNN vs LSTM models differ? Discuss the performance differences with respect to changes in the number of neurons and dropout. Also, draw insights from the heatmap of each of your neurons.

Contributions

Brandon Szeto: Data evaluation, loss graphs, diagrams, write up (methods, results, and discussion).

Darren Yu: RNN hyperparameter tuning, feature evaluation, and write up (abstract, introduction, and related work),

Nathaniel Thomas: SongRNN, model architecture, model training, and music generation.

References

Jean-Pierre Briot, Gaëtan Hadjeres, and François-David Pachet. Deep learning techniques for music generation – a survey, 2019.