Fabian Kowatsch, Marc Baumholz

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Recurrent and Generative ANNs 24/25

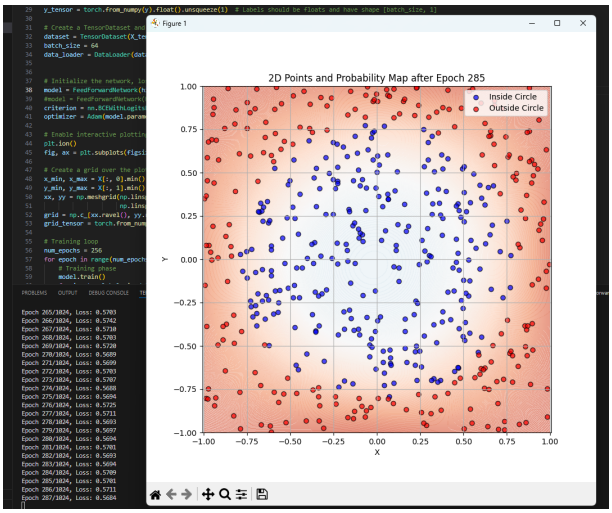
Assignment Solution 1

Date: November 1, 2024

1.1 Implementing Backpropagation

a)

The Code snipped of the backward pass can be found in our ZIP that we uploaded.



b)

The softmax activation function and Cross-Entropy Loss are both used in classification problems. Softmax is a generalization of sigmoid for multiple dimensions. Instead of converting logits into a binary probability distribution like sigmoid, it produces a probability distribution over multiple classes, with each class having a probability between 0 and 1 and all probabilities adding up to 1. Cross Entropy measures the error between a ground truth probability distribution, which is often a one-hot vector, and a predicted distribution which can be the output of a softmax activation. It penalizes false classifications and encourages correct ones. They both are well suited for classification problems and conveniently also have an easily computed gradient.

Cross-Entropy:

$L = -\sum_j y_j \log(\hat{y}_j)$

Softmax:

$softmax(z_j) = \frac{e^{z_j}}{\sum_k e^{z_k}}$

Combined:

$L = -\sum_j y_j \log\left(\frac{e^{z_j}}{\sum_k e^{z_k}}\right)$

Simplify:

$L = -\sum_j y_j z_j - \log\left(\sum_k e^{z_k}\right)$

Since only the correct class has $y_j = 1$ and all other classes have $y_j = 0$, this simplifies to:

$L = -z_{true} + \log\left(\sum_k e^{z_k}\right)$

Compute gradient of L w.r.t. z_j:

Term 1 $-z_{true}$:

$$\frac{\partial(-z_{true})}{\partial z_j} = \begin{cases} -1 \text{ if } j = true \\ 0 \text{ if } j \neq true \end{cases} = -y_j$$

Term 2 $\log\left(\sum_k e^{z_k}\right)$:

$\frac{\partial}{\partial z_j} \log\left(\sum_k e^{z_k}\right) = \frac{1}{\left(\sum_k e^{z_k}\right)} \cdot \frac{\partial}{\partial z_j} \sum_k e^{z_k}$
$\frac{\partial}{\partial z_j} \log\left(\sum_k e^{z_k}\right) = \frac{1}{\left(\sum_k e^{z_k}\right)} \cdot e^{z_k}$
$\frac{\partial}{\partial z_j} \log\left(\sum_k e^{z_k}\right) = \frac{e^{z_k}}{\left(\sum_k e^{z_k}\right)} = softmax(z_j) = \hat{y_j}$

Combining the 2 Terms:

$\frac{\partial L}{\partial z_j} = \hat{y_j} - y_j$

Cross-Entropy Loss with Alternative Normalization:

$L = -\sum_j y_j \log\left(\frac{z_j}{\sum_k z_k}\right)$

Simplify using the same methods above:

$L = -\log\left(\frac{z_{true}}{\sum_k z_k}\right)$
$L = -\log\left(z_{true}\right) + \log\left(\sum_k z_k\right)$

Compute gradient of L w.r.t. z_j:

Term 1 $-\log\left(z_{true}\right)$:

$$\frac{\partial}{\partial z_j}(-\log\left(z_{true}\right)) = \begin{cases} -\frac{1}{z_{true}} \text{ if } j = true \\ 0 \text{ if } j \neq true \end{cases} = -\frac{y_j}{z_j}$$

Term 2 $\log\left(\sum_k z_k\right)$:

$\frac{\partial}{\partial z_j} \log\left(\sum_k z_k\right) = \frac{1}{\sum_k z_k} \cdot \frac{\partial}{\partial z_j}\left(\sum_k z_k\right)$

Since $\sum_k z_k$ is just a sum of logits, the partial derivative w. r. t. z_j is just 1:

$\frac{\partial}{\partial z_j} \log\left(\sum_k z_k\right) = \frac{1}{\sum_k z_k}$

Comibined:

$\frac{\partial L}{\partial z_j} = \frac{1}{\sum_k z_k} - \frac{y_j}{z_j}$

The softmax activation causes the gradient to be in range $[-1, 1]$ By removing the exponentials, we introduce a division by z_j in the gradient, which can cause the gradient to converge towards 0 or infinity, if z_j is extremely small or large. Combining softmax and CE also has the benefit of only requiring terms that have already been calculated for the gradient computation, making it very efficient thanks to caching of results in the forward pass.

1.2 Getting familiar with Pytorch

Results are shown in the Code inside the ZIP file.

1.3 Exploring the decision boundary

We explored the number of the hidden size, which changes the decision boundary. Reducing the hidden size from 32 to 3, in our case forces the network to learn simpler, more distinct geometric patterns to separate classes, which limits the flexibility of the decision boundary and results in polygonal shapes like a hexagon. We learned that with fewer hidden units, the network cannot form highly complex boundaries and instead approximates a simpler, symmetrical structure that closely aligns with a hexagonal shape. The three neurons in the first hidden layer produce three piecewise linear boundaries with each boundary dividing the input space. The second layer is then able to add further complexity and can easily form a hexagonal decision boundary.

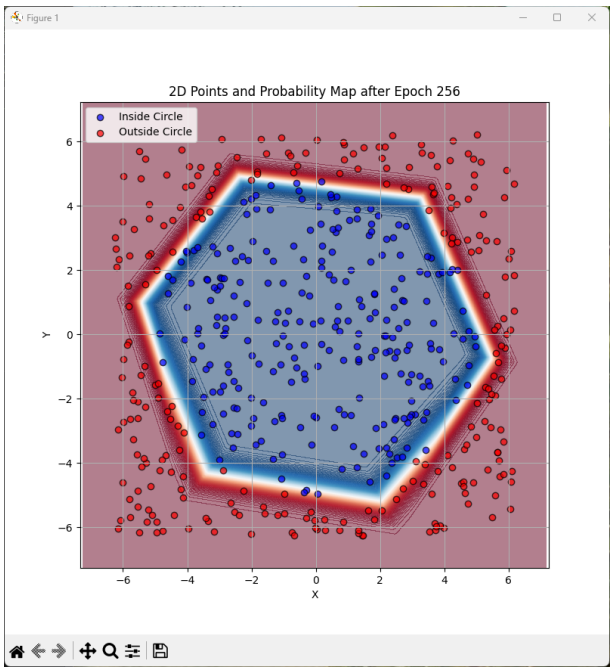


Figure 1: Result of our hexagonal decision boundary