The Tversky loss function and its modifications for medical image segmentation

Hidy Gábor

May 12, 2022

1 Introduction

The subject of this sememester's project was medical image segmentation, more precisely cancer detection. Here I will present the topics that I researched relating to this area, along with some experimental results.

1.1 Basic notions

Consider an image as an $\mathbf{x} \in [0,1]^{L \times H \times W}$ tensor.[1] The task of segmentation is to find a *segmentation mask* $\mathbf{y} \in \{0,1\}^{C \times H \times W}$. In multiclass segmentation, $\mathbf{y}$ is usually restricted such that for all (i,j), exactly one of $y_{1ij} \dots y_{Cij}$ is 1. In the binary case, $C = 1$, and one can talk about *positive* pixels (where the label is 1) and *negative* ones. As the focus of this document will be the binary case, C will often be omitted.

In practice, the output of a segmentation model will be a $\hat{\mathbf{y}} \in [0,1]^{H \times W}$, and the final prediction will be derived by thresholding at some value. In this case, $\hat{y}_{ij}$ corresponds to a probability distribution on $\{0,1\}$ where with expected value $\hat{y}_{ij}$.

For evaluating a segmentation model, one needs a metric that measures the similarity between $\mathbf{y}$ and $\hat{\mathbf{y}}$. Metrics such as accuracy or the area under the ROC curve can be used as an indicator. However, they are not a good measure for most medical segmentation tasks, where imbalanced datasets are frequent, and the classes do not have equal importance. Take cancer detection: usually cancerous regions take up a very small portion of tissue, but correctly identifying those regions is more important than correctly identifying healthy ones. For these tasks, the Tversky index is a commonly used metric.

Definition 1 (Tversky index)**.** *Let $\alpha, \beta > 0$. Let $\mathbf{y}, \hat{\mathbf{y}} \in \{0,1\}^M$. Then the* Tversky index *corresponding to $\mathbf{y}$ and $\hat{\mathbf{y}}$ is defined as*

$$T_{\alpha,\beta}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\mathrm{TP}}{\mathrm{TP} + \alpha \mathrm{FP} + \beta \mathrm{FN}} \tag{1}$$

where TP, FP and FN is the number of true positives, false positives and false negatives respectively.

$T_{\frac{1}{2},\frac{1}{2}}$ is called the *Dice index* and $T_{1,1}$ the *Jaccard index*.

1.2 Dataset

The data used in the experiment was acquired from the Országos Korányi Pulmonológiai Intézet (OKPI). The data consists of large resolution images of H&E stained lung tissue, and segmentation masks where a positive pixel indicates cancer in that region. The images were hand-annotated by a medical professional.

During preprocessing, the images were cut into 512×512 pixel pieces, with a stride of 256 (so the middle part of an image would appear in the left side of the next one), and then the images that did not contain any tissue were discarded. The derived dataset totalled around 2800, with only around 360 of them containing any cancerous regions.

[1] Here L denotes the channel size (eg. 3 for RGB images), following PyTorch's example and using the channel-first represenation.

2 Loss functions and regularisation techniques

For training neural networks, one needs a differentiable loss function that measures the similarity of the target and the prediction. Again, classification losses can be applied by taking the average over all pixels, but they will produce poor performance on highly imbalanced data. For that reason, the Tversky loss [1], a loss derived from the Tversky index, is commonly used.

2.1 Tversky loss

Definition 2 (smooth Tversky index)**.** *Let* $\alpha, \beta > 0$ *and* $\mathbf{y}, \hat{\mathbf{y}} \in [0,1]^M$*. Then the* smooth Tversky index *corresponding to* $\mathbf{y}$ *and* $\hat{\mathbf{y}}$ *is defined as*

$$\mathcal{T}_{\alpha,\beta}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\mathbf{y}\hat{\mathbf{y}}}{\mathbf{y}\hat{\mathbf{y}} + \alpha(\mathbb{1} - \mathbf{y})\hat{\mathbf{y}} + \beta\mathbf{y}(\mathbb{1} - \hat{\mathbf{y}})} \tag{2}$$

It is easy to see that if $\mathbf{y}, \hat{\mathbf{y}} \in \{0,1\}^M$, then $\mathcal{T}_{\alpha,\beta}(\mathbf{y}, \hat{\mathbf{y}})$ is the same as the discrete Tversky index described by Equation 1.

Remark. Unfortunately, $\mathcal{T}_{\alpha,\beta}$ is not defined if $\mathbf{y} = 0 = \hat{\mathbf{y}}$. For that reason, in practice, the formula

$$\mathcal{T}_{\alpha,\beta}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\mathbf{y}\hat{\mathbf{y}} + \delta}{\mathbf{y}\hat{\mathbf{y}} + \alpha(\mathbb{1} - \mathbf{y})\hat{\mathbf{y}} + \beta\mathbf{y}(\mathbb{1} - \hat{\mathbf{y}}) + \delta} \tag{3}$$

is used, where $\delta > 0$ is a small smoothing term.

The *smooth Tversky loss* is then defined as

$$\mathcal{L}_{\text{Tversky}}(\mathbf{y}, \hat{\mathbf{y}}) = 1 - \mathcal{T}_{\alpha,\beta}(\mathbf{y}, \hat{\mathbf{y}}) \tag{4}$$

It is easy to see that $\mathcal{L}_{\text{Tversky}}$ is differentiable, and if $\mathbf{y} \in \{0,1\}^M$, then it has its unique minimum at $\hat{\mathbf{y}} = \mathbf{y}$.

Batchwise and imagewise Tversky

So far, $\mathbf{y}$ (and $\hat{\mathbf{y}}$) was considered as a mask (and prediction) corresponding to a single image. However, during the training of a neural network, the loss is calculated, and backpropagation is performed, on multiple images forming a *batch*. This leads to two choices when calculating the Tversky loss: the imagewise and the batchwise approach.

The imagewise approach is the one most frequently used: calculate a loss for all input datapoints (in our case images), and then take its average as the batch loss. This, however, leads to a different loss function than the batchwise approach, which is just using the formula in Equation 4, but substituting a whole batch of images to $\mathbf{y}$ and $\hat{\mathbf{y}}$.

Batchwise Tversky is useful when the data is *two-way imbalanced*. Using Tversky loss combats *image-level imbalance* – when, in a typical image, the number of positive pixels is orders of magnitude smaller then the number of negative pixels. Using batchwise Tversky also combats *batch-level imbalance*, when there are few pixels that even have positive pixels at all. Taking the loss over the whole batch means that, if there are even a few positive pixels in the whole batch, the all-zero prediction – which can very easily become a local minimum in the case of imbalanced data – will be really far from the optimum.

2.2 Label smoothing

Classification (and segmenation) problems tend to suffer from so-called *ill-conditionedness*, when the output of their final layer has either really large, or really small coordinates. This results, after applying a softmax or sigmoid activation function, in predictions that are really close to (or may be equal to, up to computational errors) 0 or 1. This can contribute to overfitting, and it decreases the model's adaptability, since the gradient of the loss function tends to 0 in ∞ and $-\infty$, thus weight updates during training will have smaller magnitudes.

Label smoothing [2] aims to solve this by replacing $\mathbf{y}$ with

$$\tilde{\mathbf{y}} = (1-\varepsilon)\mathbf{y} + \varepsilon\mathbb{1} \tag{5}$$

for some $0 < \varepsilon < \frac{1}{2}$.

Remark. If there are C channels, Equation 5 is modified to $(1-\varepsilon)\mathbf{y} + \frac{\varepsilon}{C}\mathbb{1}$, so that $\mathbf{y}$ remains a distribution at every spacial coordinate.

When using the crossentropy loss function, label smoothing will have its desired effect, since the minimum will still be obtained at $\hat{\mathbf{y}} = \tilde{\mathbf{y}}$. (This is easy to see just by taking the gradient with respect to $\hat{\mathbf{y}}$.)

Since label smoothing has been widely used in various successful models, it would be good to adapt it to a segmentation task. One would hope that $\mathcal{T}_{\alpha,\beta}(\tilde{\mathbf{y}}, \hat{\mathbf{y}})$ also takes its optimum at $\hat{\mathbf{y}} = \tilde{\mathbf{y}}$, thus label smoothing can also be applied for the Tversky loss. Unfortunately, this is not the case: I will prove that when using label smoothing with a small ε, the optimum of $\hat{\mathbf{y}}$ is still a $\{0,1\}$ tensor.

Claim 1. *Let $\mathbf{y} \in \{\varepsilon, 1-\varepsilon\}^M$ and $\alpha + \beta = 1$. Let P be the number of indices where $y_i = 1-\varepsilon$, and $N = M - P$. If $B = \beta(P + \varepsilon(N-P)) > 0$, then there exists $\eta > 0$ such that if $\varepsilon < \eta$,* $\operatorname{argmax} \mathcal{T}_{\alpha,\beta}(\mathbf{y}, \cdot) \in \{0,1\}^M$.

Proof. Let $\hat{\mathbf{y}}$ be fix, and let

$$\hat{\mathbf{y}}_0 = \sum_{y_i=\varepsilon} \hat{y}_i, \quad \hat{\mathbf{y}}_1 = \sum_{y_i=1-\varepsilon} \hat{y}_i$$

Then

$$\mathcal{T}_{\alpha,\beta}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\varepsilon \sum\limits_{y_i=\varepsilon} \hat{y}_i + (1-\varepsilon) \sum\limits_{y_i=1-\varepsilon} \hat{y}_i}{\beta(P + \varepsilon(N-P)) + \alpha \sum\limits_{i=1}^{M} \hat{y}_i} = \frac{\varepsilon\hat{\mathbf{y}}_0 + (1-\varepsilon)\hat{\mathbf{y}}_1}{B + \alpha(\hat{\mathbf{y}}_0 + \hat{\mathbf{y}}_1)} \overset{\text{def}}{=} t(\hat{\mathbf{y}}_0, \hat{\mathbf{y}}_1)$$

It is obvious that $\hat{\mathbf{y}}_0 \in [0, N]$, $\hat{\mathbf{y}}_1 \in [0, P]$. I will show that in the optimum $\hat{\mathbf{y}}_0$ is minimal, $\hat{\mathbf{y}}_1$ is maximal, thus $\hat{\mathbf{y}} \in \{0,1\}^M$.

$$\partial_2 t(\hat{\mathbf{y}}_0, \hat{\mathbf{y}}_1) = \frac{(1-\varepsilon)B + (1-2\varepsilon)\alpha\hat{\mathbf{y}}_0}{\left(B + \alpha(\hat{\mathbf{y}}_0 + \hat{\mathbf{y}}_1)\right)^2}$$

Since $\varepsilon < \frac{1}{2}$, $\partial_2 t > 0$ for all $\hat{\mathbf{y}}_N$, so at the maximum of t, $\hat{\mathbf{y}}_P = P$. It can be shown then that for small ε, $\partial_1 t < 0$ for $\hat{\mathbf{y}}_P = P$, thus its optimum is indeed at $\hat{\mathbf{y}}_N = 0$. □

Remark. Claim 1 was stated for $\mathcal{T}_{\alpha,\beta}$ with no smoothing term as defined in Equation 2, but it obviously holds for the smoothed out version of Equation 3 using small δ.

Claim 1 means that small perturbation to the labels will not push the predictions away from the extrema, as in the case of the crossentropy loss, so it cannot be used for the same purpose. However, there is an equivalent formulation of label smoothing, presented in Claim 2, that can be applied to *any* loss function, as a regularisation term. I will state the equivalent form and provide a proof of equivalence below.

Claim 2. *Let $\mathcal{L}$ be the crossentropy loss. Then using label smoothing is equivalent to adding $D\left(\frac{1}{n}\mathbb{1} \| \hat{\mathbf{y}}\right)$, where D is the Kullback–Leibler divergence.*

Remark. Here it is assumed that $\mathbf{y}$ is a distribution at every spacial coordinate, that is, that $\mathcal{L}$ is the *categorical* crossentropy loss. An analogous claim obviously holds for binary $\mathbf{y}$ and the binary crossentropy.

Proof. Let $\tilde{\mathbf{y}} = (1-\varepsilon)\mathbf{y} + \frac{\varepsilon}{n}\mathbb{1}$. Then

$$\mathcal{L}(\tilde{\mathbf{y}}, \hat{\mathbf{y}}) = -\left((1-\varepsilon)\mathbf{y} + \frac{\varepsilon}{n}\mathbb{1}\right) \cdot \log \hat{\mathbf{y}} = -(1-\varepsilon)(\mathbf{y} \cdot \log \hat{\mathbf{y}}) - \frac{\varepsilon}{n}(\mathbb{1} \cdot \log \hat{\mathbf{y}}) = (1-\varepsilon)\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) - \frac{\varepsilon}{n}\sum_{i=1}^{n} \log \hat{y}_i$$

Meanwhile

$$D\left(\frac{1}{n}\mathbb{1}\middle\|\hat{\mathbf{y}}\right) = -\log n - \frac{1}{n}\sum_{i=1}^{n}\log\hat{y}_i$$

So

$$\mathcal{L}(\tilde{\mathbf{y}},\hat{\mathbf{y}}) = (1-\varepsilon)\mathcal{L}(\mathbf{y},\hat{\mathbf{y}}) + \varepsilon D\left(\frac{1}{n}\mathbb{1}\middle\|\hat{\mathbf{y}}\right) + \log n$$

Since $\frac{\varepsilon}{1-\varepsilon}$ can be set to an arbitrary positive value, they are indeed equivalent. □

2.3 Confidence penalty

The regularisation term $D\left(\frac{1}{n}\mathbb{1}\|\hat{\mathbf{y}}\right)$ – that I call *label noise penalty* – can be applied to any loss function, including the Tversky loss, in the same vain as other regularisation techniques, like the L^2 regularisation. It is useful for pushing the predictions away from 0 and 1 and toward the uniform distribution, since D is minimal if and only if its two arguments are equal.

Since the Kullback–Leibler divergence is not symmetric, taking $D\left(\hat{\mathbf{y}}\|\frac{1}{n}\mathbb{1}\right)$ instead would yield a different regularisation term – the confidence penalty. [3] Just as the label noise penalty is equivalent to taking the sum of the negative logarithm of the coordinates of $\hat{\mathbf{y}}$, the confidence penalty is equivalent to the negative entropy of $\hat{\mathbf{y}}$, since

$$D\left(\hat{\mathbf{y}}\middle\|\frac{1}{n}\mathbb{1}\right) = \log n - H(\hat{\mathbf{y}}) \tag{6}$$

As before, the constant $\log n$ does not matter, since only the gradient of the loss is used.

The confidence penalty also has the property that it pushes $\hat{\mathbf{y}}$ toward the uniform distribution – which has maximal entropy – but it has a different gradient from the label noise penalty. Either – or even both – can be applied as a regularisation term with any loss function, including, crucially, the Tversky loss.

3 Experiments

I ran some experiments on the dataset described in Section 1.2. All experiments used a U-Net [4] model, with optimiser Adam and learning rate 10^{-6}. The model was evaluated on 256×256 images. For training, these were obtained from the larger images by randomly sampling a rotated 256×256 subimage. For validation, the large images were partitioned into four parts, and all the model was evaluated on all four. I examined two tasks: one was segmentation of the whole dataset, the other was segmentation of only those images that contained a positive region.

Metrics

Several metrics were used to evaluate model performance: Dice index, Jaccard index, balanced accuracy, average precision score and modified Hausdorff distance. Here I will provide an overview for these metrics.

Balanced accuracy is the arithmetic mean of the true positive and true negative rates. Average precision score is the area under the precision–recall curve.

The *modified Hausdorff distance* of a point x and set Y (denoted $h(x,Y)$) is defined as the 95^{th} percentile of the set of distances between x and all $y \in Y$. The average modified Hausdorff distance is then calculated as

$$h(X,Y) = \frac{1}{2}\left(\frac{1}{|X|}\sum_{x\in X}h(x,Y) + \frac{1}{|Y|}\sum_{y\in Y}h(y,X)\right) \tag{7}$$

Average precision was calculated on the raw output. All other metrics were calculated on the output thresholded at 0.5.

3.1 Segmentation of all images

This proved a difficult task. Most configuration converged to an all-zero output, due to the highly imbalanced nature of the data. Two things were needed in order to force the model to start outputting positive predictions as well. The first one was using batchwise Tversky loss with a batch size of 64.[2] The second one was forcing each batch to have the same positive-negative image ration (around 1:7) as the whole dataset. Even with these, I could not get more than around 0.4 Dice index on the validation data, and it took 1000 epochs to reach this performance.

For this task, I also experimented with different parameters for the Tversky loss, but found that no configuration gave significantly better results than using the simple Dice loss.

3.2 Segmentation of the positively annotated images

For the positive images, using the Dice loss also proved to be as good as any other hyperparameter configuration. Here an imagewise loss also produces comparable results – that is probably due to the fact that (almost) all images have positive pixels. However, using a batchwise loss consistenlty gives a better performance regardless of batch size.

batch size	8		16		32		64	
loss domain	image	batch	image	batch	image	batch	image	batch
average precison ↑	0.74627	**0.80885**	0.69417	0.77071	0.70260	0.78253	0.71638	0.77838
balanced accuracy ↑	0.87294	0.87796	0.86982	0.87696	0.87675	0.88813	0.86041	**0.88449**
Dice index ↑	0.63520	0.69041	0.60610	0.66463	0.64115	**0.70522**	0.59301	0.66673
Jaccard index ↑	0.49333	0.54919	0.46456	0.52946	0.47573	**0.54796**	0.42374	0.50167
avg. Hausdorff dist. ↓	0.16421	0.15141	0.17085	**0.14911**	0.26964	0.18433	0.37718	0.31349

Table 1: Comparison of imagewise and batchwise Dice loss

Table 1 summarises the validation performance of models trained with different batch sizes and with either imagewise or batchwise Dice loss. The performances were measured after 100 epochs, but the shapes of the learning curves indicated further training would result in better performance. (↑ after a metric indicates that a larger number is better, ↓ means smaller is better.)

3.3 Future work

I intend to try and get an improved performance using confidence penalty or label noise penalty regularisation. Initial experiments run so far show no significant difference with or without regularisation, except for regularisation coefficients close to 1, when performance starts to deteriorate. This might change with a better calibrated coefficient, or even simply with longer experiments – the longest experiment with these regularisations run so far was 200 epochs, and even without regularisation, the model did not start to overfit. It is possible that using a regularisation term will improve the performance only later on the training, when overfitting becomes more likely.

References

[1] S. S. M. Salehi, D. Erdogmus, A. Gholipur. Tversky loss function for image segmentation using 3D fully convolutional deep networks. *Machine Learning in Medical Imaging*, September 2017, pp. 379–387.

[2] C. Szegedy *et al.* Rethinking the Inception architecture for computer vision. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 2818–2826.

[3] G. Pereyra *et al.* Regularizing neural networks by penalizing confident output distributions. arXiv preprint, Jan 2017. arXiv:1701.06548v1 [cs.NE]

[4] O. Ronneberger, P. Fischer, T. Brox. U-Net: Convolutional networks for biomedical image segmentation. *Medical Image Computing and Computer-Assisted Intervention*, November 2015, pp. 234–241.

[2] I did not go higher because 64 was the maximum batch size that the current GPU memory could handle.