You can use **dice score** to normalize the score of binary images such that if you have correct images where the binary mask is 20% on, 80% off, the network doesn't learn to just predict 100% off and get 80% accuracy

- **Binary / Categorical Cross Entropy** is the typical go-to loss function
  - This loss function is not ideal if you have a lot of background / class imbalance

Sensitivity -
Specificity -

$$\text{sensitivity} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

$$\text{specificity} = \frac{\text{true negatives}}{\text{true negatives} + \text{false positives}}$$

Dice Similarity Coefficient - measure of how well two contours overlap

- **0** means complete mismatch
- **1** means perfect contour match

$$\mathrm{DSC}(A, B) = \frac{2 \times |A \cap B|}{|A| + |B|}.$$

- Two sets **A** and **B** of voxels
- **A** is the prediction and **B** is the ground truth
- *For **Binary Dice Score**, turn the output into a binary map of **segmented vs unsegmented***
- *For **Categorical Dice Score**, compute BDS for each class, then **average** out the values*
- **You add ε to both sides to avoid division by 0**

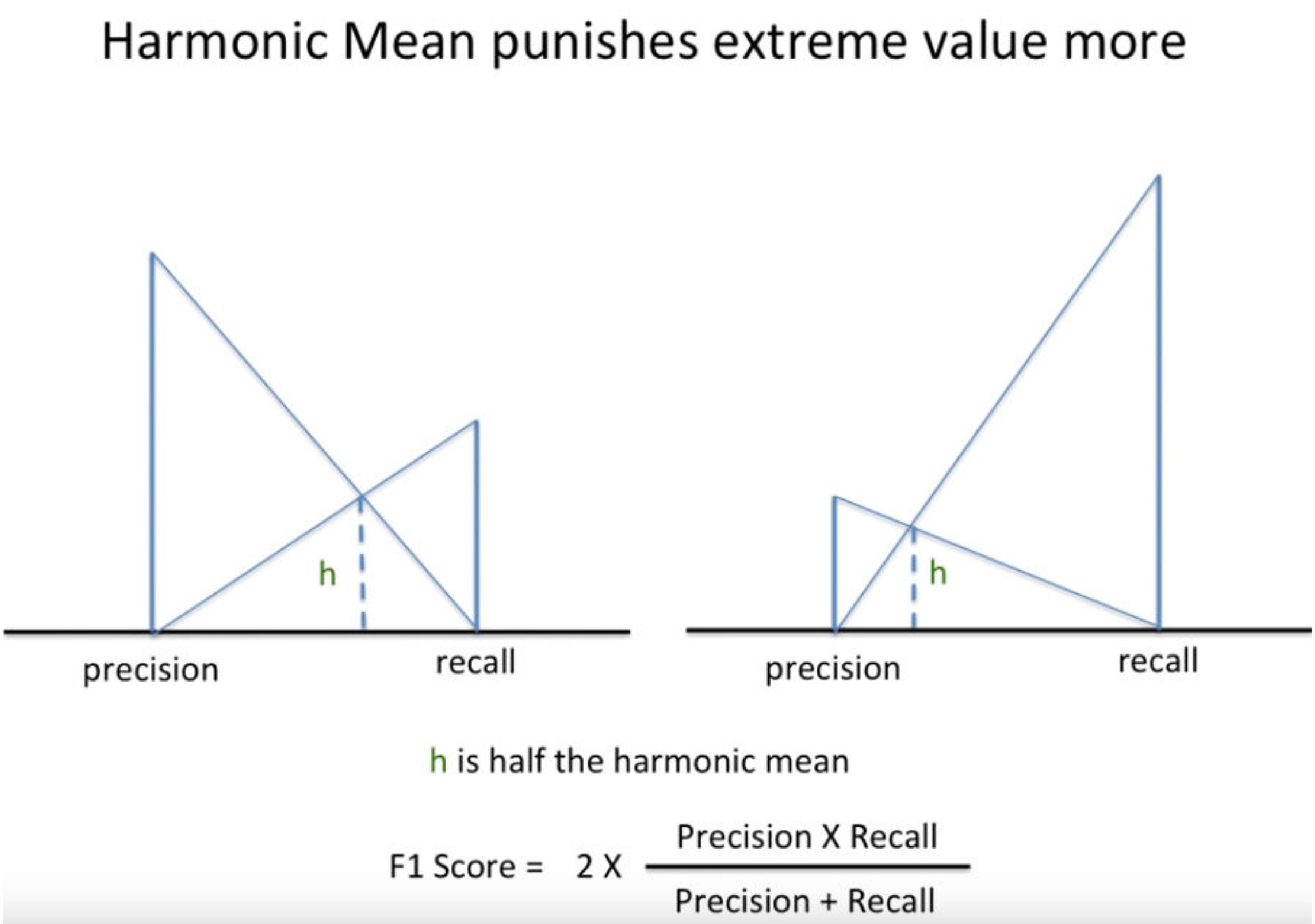
Soft Dice Similarity Coefficient - fixes some of the issues with normal dice score

- **Issues with normal Dice Score**
  - Takes in discrete values, i.e. 0s or 1s
  - The model predicts probabilities, through which we want to backpropagate through, so we would prefer an **analogous version of the dice score**
- **SDSC** output ranges between 0 with a perfectly matching truth distribution, all the way to 1

$$\mathcal{L}_{Dice}(p, q) = 1 - \frac{2 \times \sum_{i,j} p_{ij} q_{ij} + \epsilon}{\left(\sum_{i,j} p_{ij}^2\right) + \left(\sum_{i,j} q_{ij}^2\right) + \epsilon}$$

  - p is our predictions
    q is the ground truth
    In practice each q_i will either be 0 or 1.
    ϵ is a small number that is added to avoid division by zero

Multi-class Soft Dice Similarity Coefficient - As with binary DS, you just average out the soft dice scores



Dice Score is the same as F1 Score

Metric vs. loss function - Kind of the same thing, except metrics are used to only measure model performance, not carry out backpropagation. Therefore, loss functions often have simpler derivatives and can be computed over 1 batch

- Dice score is often a metric because we need to compute dice score over an epoch, whereas we want to upgrade gradients every batch
    - Batch-wise dice score is possible but not often used