CS 182/282A Designing, Visualizing and Understanding Deep Neural Networks
Spring 2022 Marvin Zhang

Discussion 10

This discussion focuses on confidence calibration, distribution shifts and adversarial robustness.

1 Robustness

1.1 Confidence Calibration

In real-world decision making systems, classification networks must not only be accurate, but also should indicate when they are likely to be incorrect. The probability associated with the predicted class label should reflect its ground truth correctness likelihood.

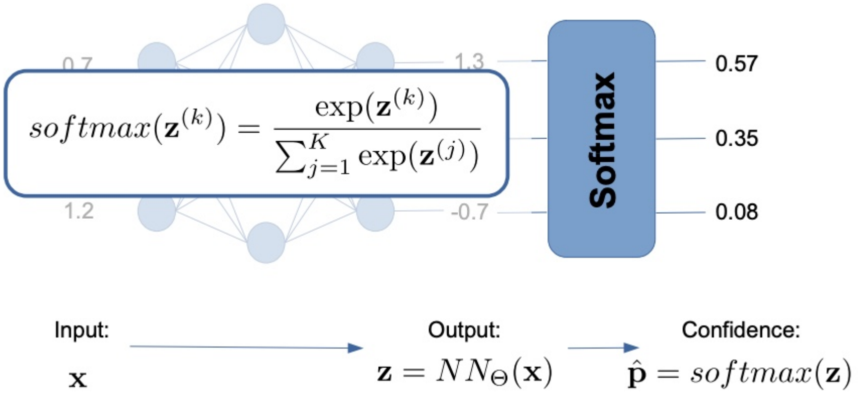

0.7
1.3
$softmax(\mathbf{z}^{(k)}) = \frac{\exp(\mathbf{z}^{(k)})}{\sum_{j=1}^{K} \exp(\mathbf{z}^{(j)})}$
1.2
-0.7
Softmax
0.57
0.35
0.08
Input:
$\mathbf{x}$
Output:
$\mathbf{z} = NN_{\Theta}(\mathbf{x})$
Confidence:
$\hat{\mathbf{p}} = softmax(\mathbf{z})$

Figure 1: Model prediction confidence.

Perfect calibration

$$\mathbb{P}(\hat{Y} = Y \mid \hat{P} = p) = p \quad \forall p \in [0, 1]$$

Model calibration

$$\mathbb{E}[|\mathbb{P}(\hat{Y} = Y \mid \hat{P} = p) - p|]$$

Expected Calibration Error (ECE) Computes weighted average of mis-calibration

1. Train neural network on training data

2. Create predictions and confidence estimates using the test data

3. Group the predictions into M bins, define bin B_M to be the set of all predictions $(\hat{y}_i, \hat{p}_i)$ for which it holds that $\hat{p}_i \in \left(\frac{m-1}{M}, \frac{m}{M}\right]$

4. Compute the accuracy and confidence of bin B_M as

$$\text{acc}\left(B_m\right) = \frac{1}{\left|B_m\right|} \sum_{i \in B_m} \mathbf{1}\left(\hat{y}_i = y_i\right) \quad \text{conf}\left(B_m\right) = \frac{1}{\left|B_m\right|} \sum_{i \in B_m} \hat{p}_i$$

5. Compute the expected calibration error as

$$ECE = \sum_{m=1}^{M} \frac{\left|B_m\right|}{n} \left|\text{acc}\left(B_m\right) - \text{conf}\left(B_m\right)\right|$$

Maximum Calibration Error Computes maximum mis-calibration. Useful for high risk applications

$$MCE = \max_{m \in \{1,\ldots,M\}} \left|\text{acc}\left(B_m\right) - \text{conf}\left(B_m\right)\right|$$

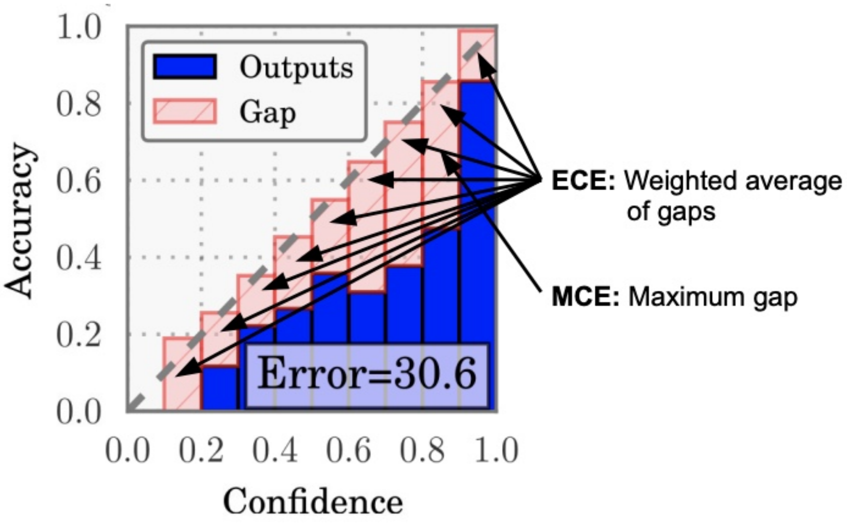


Figure 2: Reliability Diagram. Illustration of ECE and MCE errors.

Calibrating model by adjusting softmax temperature Models can be made more calibrated after training by adding a softmax temperature.

$$\hat{p}(y = i \mid x) = \frac{\exp\left(l_i / T\right)}{\sum_{j=1}^{k} \exp\left(l_j / T\right)}$$

Through experiments, [1] found that training using negative log-likelihood / cross-entropy loss increases confidence in correct classes, leading to overfitting in negative log-likelihood.

A simple solution to improve model calibration is minimizing negative log-likelihood by adjusting softmax temperature T after training model.

Problem: Model Calibration

Optimizing softmax temperature T to minimize negative log-likelihood can improve model calibration, does it hurt classification accuracy?

1.2 Distribution shift framework

1.2.1 Domain adaptation

Assuming have abundant training data from a source domain and only a small amount of training data from the target domain which is the actual domain of interest.

Importance weighting A classic approach to domain adaptation is to estimate importance weights for the source training data.

$$\mathbb{E}_{\text{target}}\left[\ell(\theta; X, Y)\right] = \mathbb{E}_{\text{source}}\left[\frac{p_{\text{target}}(X, Y)}{p_{\text{source}}(X, Y)}\ell(\theta; X, Y)\right] \approx \frac{1}{N}\sum_{i=1}^{N}\frac{p_{\text{target}}(\mathbf{x}_i, y_i)}{p_{\text{source}}(\mathbf{x}_i, y_i)}\ell\left(\theta; \mathbf{x}_i, y_i\right)$$

Estimating p_{source} and p_{target} from data are difficult in deep learning.

Invariant feature learning Invariance means that we wish for the feature distributions between the source and target data to look identical, if the model is outputting similar features for both source and target data and predicting well for source data, we may expect that it will also predict well on target data.

1.2.2 Subpopulation shift

In subpopulation shift, we assume several training domains rather than just two. The key challenge in subpopulation shift is that some domains are underrepresented in the training data.

1.2.3 Domain generalization

Similar to subpopulation shift, domain generalization assumes several domains are provided at training time. Different from it, we assume that we will be given new domains at test time, and our goal is to generalize to these new domains.

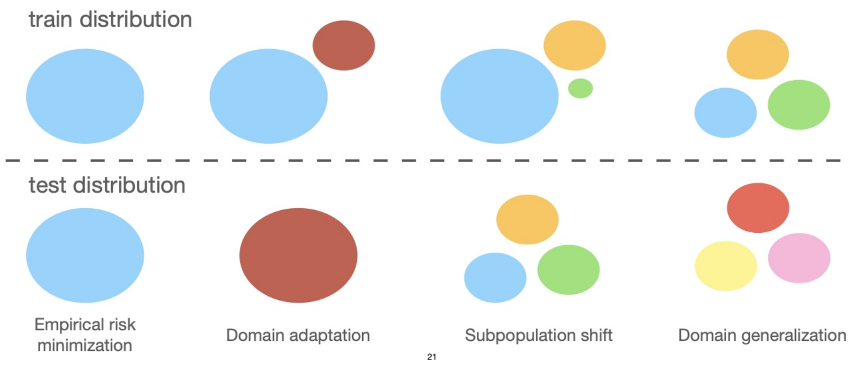


Figure 3: Summary of problem settings and frameworks for characterizing real-world distribution shifts.

2 Adversarial robustness

2.1 A simple adversarial attack

A simple threat model is to assume the adversary has an ℓ_p attack distortion ϵ i.e., for some assumed and ϵ,

$$\left\|\mathbf{x}_{\text{adv}} - \mathbf{x}\right\|_p \leq \epsilon$$

The adversary's goal is usually to find a distortion δ that maximizes the loss subject to its budget

$$\mathbf{x}_{\text{adv}} = \mathbf{x} + \arg \max_{\delta:\|\delta\|_p \leq \epsilon} \ell(\theta; \mathbf{x} + \delta, y)$$

2.2 Generate adversarial examples

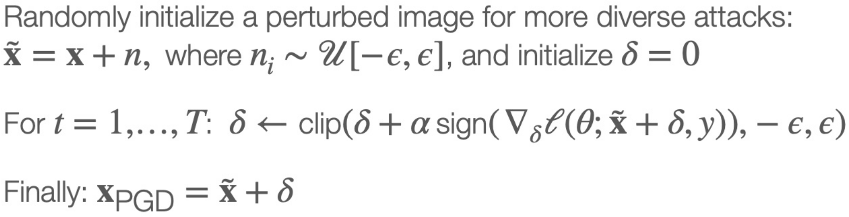
Fast gradient sign method (FGSM)

$$\mathbf{x}_{\text{FGSM}} = \mathbf{x} + \epsilon \, \text{sign}\left(\nabla_{\mathbf{x}} \ell(\theta; \mathbf{x}, y)\right)$$

This attack performs a single step of gradient ascent on the input to increase the model's loss, obeying an ℓ_∞ attack budget

$$\|\mathbf{x}_{\text{FGSM}} - \mathbf{x}\|_\infty = \epsilon$$

Projected gradient descent (PGD) The PGD attack uses multiple gradient ascent steps and thus is far more powerful than the FGSM attack which consists of a single step.



```
Randomly initialize a perturbed image for more diverse attacks:
$\tilde{\mathbf{x}} = \mathbf{x} + n$, where $n_i \sim \mathcal{U}[-\epsilon, \epsilon]$, and initialize $\delta = 0$

For $t = 1, \ldots, T$: $\delta \leftarrow \text{clip}(\delta + \alpha \, \text{sign}(\nabla_\delta \ell(\theta; \tilde{\mathbf{x}} + \delta, y)), -\epsilon, \epsilon)$

Finally: $\mathbf{x}_{\text{PGD}} = \tilde{\mathbf{x}} + \delta$
```

Figure 4: Pseudocode for a PGD attack with T steps and an ℓ_∞ attack budget ϵ.

Adversarial training (AT) Adversarial training works well on making models robustify to ℓ_p attacks but can reduce accuracy on non adversarial examples significantly.

- Sample minibatch $\left(\mathbf{x}^{(1)}, y^{(1)}\right), \ldots, \left(\mathbf{x}^{(B)}, y^{(B)}\right)$ from the training set
- Create $\mathbf{x}_{\text{adv}}^{(i)}$ $\left(\text{ e.g., } \mathbf{x}_{\text{PGD}}^{(i)}\right)$ from $\mathbf{x}^{(i)}$ for all i
- Optimize the average training loss on these adversarial training examples

Untargeted vs. targeted attacks Untargeted attacks tries to maximize the loss. Targeted attacks optimize examples to be misclassified as a predetermined target $\tilde{y}$

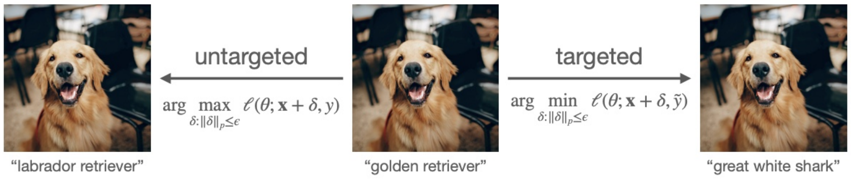

untargeted
$\arg \max_{\delta:\|\delta\|_p \leq \epsilon} \ell(\theta; \mathbf{x} + \delta, y)$
targeted
$\arg \min_{\delta:\|\delta\|_p \leq \epsilon} \ell(\theta; \mathbf{x} + \delta, \tilde{y})$
"labrador retriever"
"golden retriever"
"great white shark"

Figure 5: An example of untargeted vs. targeted attacks.

Methods for improving robustness

1. Using larger and more diverse data
2. Data augmentation
3. Using smooth activations such as GELUs instead of sharp activation such as ReLUs

References

[1] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning*, pages 1321–1330. PMLR, 2017.