CS 453X: Class 26

Jacob Whitehill

Practical suggestions

Start small

- Debug your code on a small subset of the data.
    - Less time for initialization.
    - Less time for training.

Start small

- Debug your code on a small subset of the data.
  - Less time for initialization.
  - Less time for training.
- But make sure the statistics of the sample match (approximately) those of the whole dataset.
  - All classes are represented!

Start simple

- Until you gain confidence & experience, train a simple model first:
  - They’re often faster to train and easier to debug than more powerful models.

Start simple

- Until you gain confidence & experience, train a simple model first:
  - They're often faster to train and easier to debug than more powerful models.
- Make sure your model's accuracy is above chance:
  - Take the prior class probabilities into account! (If the classes are 90/10, then the baseline rate for guessing the majority class is 90%.)
  - Make sure your model is not always predicting the dominant class.

Start small & simple

- Try to find a model (and hyper-parameters) whose training loss decreases *smoothly*.
- Afterwards, increase the size of the training set and model complexity.

Regularization

- If there is a large divergence between training accuracy and testing accuracy (i.e., overfitting), then try regularizing the model:

  - Increasing L_1, L_2 regularization strength.

  - Adding/increasing dropout (for NNs).

  - Reducing number of training epochs (for NNs).

  - Synthesizing more training examples with label-preserving transformations (geometric & noise-based).

Hyper-parameter optimization

- Try a variety of hyper-parameters:
  - Pick a reasonable range (e.g., for learning rate, 1e-5 to 1e0, spaced logarithmically)
  - Search systematically and automatically (ideally in parallel) on a validation set, not the test set!
  - As fun as it is (and believe me — this is addictive), try not to stare at the training trajectory too much — it rarely helps.

Normalization

- It can be helpful to put every feature onto the same scale.
- In particular, the scale can interact with the L_2 regularization strength.

Normalization: example

- Suppose you are predicting tomorrow's temperature based on (1) today's temperature and (2) wind speed.

- Suppose we measure temperature in Kelvin and wind speed in km/h.

- Suppose the optimal weights w_1, w_2 for these two features, for L_2-regularized linear regression, are 1 and 2, i.e.:

  - $\hat{y} = w_1 t + w_2 s$ (t = today's temp, s = today's wind speed)
    $\hat{y} = 1 * t + 2 * s$

Normalization: example

- Now, suppose we change the units for wind speed from km/h to m/s.
    - E.g., 18 km/h = 5 m/s **Numerical values reduced by 3.6x**
- If we don't adjust our model weights w_1, w_2, then our predictions will be wrong:
    - $\hat{y} = 1 * t + 2 * s$
      $\hat{y}(4, 18) = 4 + 36 = 40$ **km/h**
      $\hat{y}(4, 5) = 4 + 10 = 14$ **m/s**

Normalization: example

- Because the numerical values of the wind speed were reduced by factor of 3.6, the corresponding weight w_2 must compensate by increasing by 3.6x, i.e.:

  - $\hat{y} = w_1 t + \tilde{w}_2 s$ (t = today's temp, s = today's wind speed)
    $\hat{y} = 1{*}t + 3.6{*}2{*}s$

- Without regularization, the training procedure (e.g., minimize f_{MSE}) will account for the change-of-scale seamlessly, i.e.:

$$\arg\min_{\tilde{w}_2} f_{\mathrm{MSE}}^{\mathrm{m/s}}(\cdot) \quad \mathbf{= 3.6 *} \quad \arg\min_{w_2} f_{\mathrm{MSE}}^{\mathrm{km/h}}(\cdot)$$

Normalization: example

- But with L_2 regularization, the issue is more complicated:

$$\arg\min_{\tilde{w_2}} \left[f_{\text{MSE}}^{\text{km/h}}(\cdot) + \frac{1}{2} w_2^2 \right]$$

- The regularization term “discourages” w_2 from growing too big:
    - When we rescale from km/h to m/s, the L_2 term prevents the weight w_2 from compensating exactly.

Normalization: recommendations

- For features in a finite range, try rescaling to [0,1] or [-1,1].
- For features in infinite range, try subtracting the mean and dividing by standard deviation (so that the distribution has zero-mean and unit standard deviation).

Pre-training & fine-tuning

- For a ML problem with a relatively small (only a few thousand examples or less), try fine-tuning a pre-trained model for a related task.
    - E.g., VGG & Inception networks for image recognition.
    - This can be very effective at harnessing a powerful model for a new problem domain *without overfitting*.

Where to next?

Where to next

- Try your hand at a real ML problem.
  - Kaggle, DataDriven, KDDCup, etc.
- Keep your expectations modest at first, e.g.:
  - Try to reduce the baseline error rate by 1/2.
  - Try to keep improving your accuracy, even if the improvements are small.

Where to next

- Take a graduate course, e.g.:
  - CS 541 (deep learning)
  - CS 539 (machine learning)
  - CS 540 (artificial intelligence)
  - DS 501, 502, 503, 504 (data science courses)

Where to next

- Take an online course, e.g.:
  - Machine learning on edX: https://www.edx.org/course/machine-learning-columbiax-csmm-102x-2
  - Deep learning for natural language processing (NLP): https://cs224d.stanford.edu
  - Convolutional neural networks for visual recognition: http://cs231n.stanford.edu/

Where to next

- Read papers from ML conferences, e.g.:
  - NIPS, ICML, ICLR, AAAI, arxiv.
- But keep your expectations modest (again):
  - ML papers are often highly technical.
  - Read through a few abstracts; see which ones you understand the most; read those papers.

Where to next

- Ideally, try to find an internship or research assistantship related to machine learning.
    - Very helpful to talk to other people while tackling an ML problem.