Outline

- Set up the training, cont.
- Data pre-processing
- Model weight initialization
- Tips for model training
Model error and training setup

Given a data set, model error is:

\[ E_D \left[ (y - M(x; D))^2 \right] = E_D [M(x; D) - f(x)]^2 + E_D [E_D (M(x; D)) - M(x; D)]^2 + \sigma^2 \]

- Bias
- Variance
- Bayes error

Adjust model parameters

Adjust hyper-parameters

Unbiased estimation of model generalization performance
Ideally, Train, Dev, Test sets are from the same data distribution.

- Adjust model parameters
- Compare hyper-parameters
- Unbiased estimation of model performance

Unknown data distribution

What we care in the end!
Test and Train sets can be from different distribution

10K, MRI images

1K, CT images

https://www.clinicalradiologyonline.net/article/S0009-9260(17)30058-2/fulltext
Test and Train sets can be from different distribution

**Goal:** Detect whether heart is imaged in MR images

<table>
<thead>
<tr>
<th></th>
<th>Option 1</th>
<th>Option 2</th>
<th>Option 3</th>
<th>Option 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train</td>
<td>8K MRI</td>
<td>8K MRI + 1K CT</td>
<td>8K MRI</td>
<td>8K MRI</td>
</tr>
<tr>
<td>Dev</td>
<td>1K MRI</td>
<td>1K MRI</td>
<td>1K MRI + 1K CT</td>
<td>1K MRI</td>
</tr>
<tr>
<td>Test</td>
<td>1K MRI</td>
<td>1K MRI</td>
<td>1K MRI</td>
<td>1K MRI + 1K CT</td>
</tr>
</tbody>
</table>

10K, MRI images

1K, CT images
Test and Train sets can be from different distribution

Sample the Dev and Test sets from the **same data distribution** for the deployment time

Dev and Test sets define the target of the model

Test and Train sets can be from different distribution

Goal: Detect whether heart is imaged in CT images

<table>
<thead>
<tr>
<th></th>
<th>Option1</th>
<th>Option2</th>
<th>Option3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Train</strong></td>
<td>500 CT</td>
<td>10K MRI</td>
<td>10K MRI+500 CT</td>
</tr>
<tr>
<td><strong>Dev</strong></td>
<td>250 CT</td>
<td>500 CT</td>
<td>250 CT</td>
</tr>
<tr>
<td><strong>Test</strong></td>
<td>250 CT</td>
<td>500 CT</td>
<td>250 CT</td>
</tr>
</tbody>
</table>

10K, MRI images

1K, CT images
Dev and Train set are from the different distribution.

Goal: Detect whether heart is imaged in CT images.

Model variance can be overestimated, due to data mismatch.
Dev and Train set are from the different distribution

- Split the train set further to get a Tra-Dev set
- Train and Tra-Dev set have the same data distribution
- Used to estimate model variance
Train, Tra-Dev, Dev and Test sets

![Graph showing accuracy over epochs with labels for Bias, Training accuracy, TraDev accuracy, Data mismatch, Dev accuracy, and Dev set overfitting.]
Spot problems: Error analysis

Error analysis: exam the incorrect/less-perfect samples in Dev set to spot the problems

Typical sample in Train set

- Image is too dark: 5%
- New image view: 11%
- Image is flipped and with different contrast: 14%
Error analysis: exam the incorrect/less-perfect samples in Dev set to spot the problems

<table>
<thead>
<tr>
<th>Findings</th>
<th>Error rate</th>
<th>Remedies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image is too dark</td>
<td>5%</td>
<td>Add pre-processing</td>
</tr>
<tr>
<td>New image view</td>
<td>11%</td>
<td>Acquire more data</td>
</tr>
<tr>
<td>Image is flipped and with different contrast</td>
<td>14%</td>
<td>Add data augmentation</td>
</tr>
</tbody>
</table>
Iterate error analysis with training

- Human-level-performance: ~ 100% detection rate
- Iterative process
- The key for deep learning is to decide what to do at each step

Correct for first findings: add data preprocessing, data augmentation, add new data

Increase model depth + L2 Reg

Second error analysis

Until meet the performance requirement
Speed up the error analysis

**Eyeball set:** A portion of dev set to check

**Blackbox set:** A portion of dev set to apply model, but not exam for error
Active selection: use uncertainty

- Use the model output to pick samples to check
- Consider to use different metrics
- Combine domain knowledge, e.g. signal-noise-ratio in different imaging protocols, image appearances in different diseases (such as hypertrophic cardiomyopathy)
- Automate this process and optimize tools
Error analysis of a complex system

Model 1: R-R interval, location of S-T wave

Model 2: Segmentation of heart

Model 3: Disease classification
Error analysis of a complex system

Model 1

Manually labeled R-wave

Disease classification

Model 2

Model 3

Manual segmentation

https://ecg.utah.edu/lesson/3
Error analysis of a complex system

Model 1

Model 2

Model 3

Disease classification

Manual segmentation
Error analysis of a complex system

Model 1

Manual label

Disease classification

Model 2

Model 3

Manual label

Disease classification

Model 3

Model 2
Error analysis of a complex system

Model 1

Latent representation, Nx1

Model 2

Latent representation, Tx1

Model 3

Disease classification

- Debug as one big model
- End-to-end training
- Harder to isolate bugs
Outline

- Set up the training, cont.
- **Data pre-processing**
- Model weight initialization
- Tips for model training
Data pre-processing to reduce variation

Standardizing the input data and reducing the variation can **significantly** help training.

- **High spatial resolution**
- **Low spatial resolution**
- **Fixed high resolution of 1mm²**

- **Resample to fixed resolution**
- **Adjust image windowing**
Data pre-processing to understand

Understand the “edge” samples, exclude the corrupted samples and correct wrong labels

Noisy with artifacts

Signal loss due to devices

An average of 3.4% errors across the 10 benchmark ML datasets

Has impact on evaluating different models

https://labelerrors.com/
https://l7.curtisnorthcutt.com/confident-learning
Data pre-processing

Normalize different dimension of input data, help optimization

Features with varying magnitude
After scaling and centering
Normalize different dimension of input data, help optimization

\[ X \in \mathbb{R}^{N \times 2} \]

\[ X = X - \text{np.mean}(X, \text{axis} = 0) \]

Another approach often used: \[ X = \frac{X - \text{np.mean}(X, \text{axis} = 0)}{\text{np.std}(X, \text{axis} = 0)} \]
PCA: Principal Component Analysis

\[ X \in \mathbb{R}^{N \times 2} \]

\[ X = X - \text{np. mean}(X, \text{axis} = 0) \]

\[ [E, V] = \text{np.linalg.eig}(\text{np.dot}(X^T, X)/N) \]

\[ X = \text{np. dot}(X, V) \]

\[ X = X/\text{np. sqrt}(X \cdot \text{diagonal}()) \]
PCA: Principal Component Analysis

$X \in \mathbb{R}^{N \times D}$  If the input dimension $D$ is large, dimension reduction can be performed

$$[E, V] = \text{np.linalg.eig}(\text{np.dot}(X^T, X)/N)$$  
$$X = \text{np.dot}(X, V)$$  
$$X = X[:, 0:K]$$

Only keep the top $K$ dimension, $X \in \mathbb{R}^{N \times K}$, can filter out some noise

These steps are also called “Whitening”:

the covariance matrix of $X$ after applying the eigenvector and divided by $\sqrt{\text{eigenvalue}}$ becomes identity matrix
Tips for data processing

• For image, often only apply centering and scaling

\[ X = \frac{X - \text{np.mean}(X, \text{axis} = 0)}{\text{np.amax}(X, \text{axis} = 0) - \text{np.amin}(X, \text{axis} = 0)} \]

Or simply, \( X = X / \text{np.max}(X) \) ❯ scale image to be [0, 1]

• For time signal or multi-channel signals, PCA can be useful

• Start simple and plot results from every preprocessing steps

• Including the **identical** preprocessing in the regression test

• Make sure the inference data go through the **identical** pre-processing steps

• Computing the pre-processing statistics (e.g. dataset mean, eigen vectors, eigen values ...) on the training set **ONLY** and applying them in inference

• Complicated pre-processing can make model more vulnerable to data distribution shift
Outline

- Set up the training, cont.
- Data pre-processing
- Model weight initialization
- Tips for model training
Need a strategy to give weights some initial values

How do we initialize weights and biases?

\[ Z^{[1]} = W^{[1]}X + b^{[1]} \]
\[ a^{[1]} = f(Z^{[1]}) \]
\[ Z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} \]
\[ a^{[2]} = f(Z^{[2]}) \]
\[ Z^{[3]} = W^{[3]}a^{[2]} + b^{[3]} \]
\[ \hat{y} = a^{[3]} = f(Z^{[3]}) \]
Need a strategy to give weights some initial values

How about $W=0$ and $b=0$?

\[
Z^{[1]} = W^{[1]}X + b^{[1]}
\]
\[
a^{[1]} = f(Z^{[1]})
\]
\[
Z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}
\]
\[
a^{[2]} = f(Z^{[2]})
\]
\[
Z^{[3]} = W^{[3]}a^{[2]} + b^{[3]}
\]
\[
a^{[3]} = f(Z^{[3]}) = \hat{y}
\]

All $Z^{[l]}$ will be zero

All $a^{[l]}$ will be a constant, depending on the non-linear activation function used

When performing backprop, then gradient will be zero …

So SGD cannot update network parameters …
Need a strategy to give weights some initial values

How about \( W = \sigma \times \text{np.random.rand(input_D, output_D)} \), zero-mean, gaussian random values

\[
Z^{[1]} = W^{[1]}X + b^{[1]}
\]

\( W^{[1]} \) : 3xN matrix

\[
Z^{[1]}_k = \sum_{i=0}^{N-1} W_{k,i}^{[1]} x_i + b^{[1]}
\]

The k-th element of score Z

Let’s whitening the input \( X \), so it has same variance for all features.

\[
\text{Var}(Z^{[1]}_k) = \sum_{i=0}^{N-1} \text{Var}((W_{k,i}^{[1]} \text{Var}(x_i) + \text{Var}(b^{[1]}) = N \sigma^2 \text{Var}(x)
\]

Since all \( x_i \) and \( W_{k,i}^{[1]} \) are independent upon initialization, with zero mean

\[
\text{Var}(wx) = E(w^2)E(x^2) - E(w)^2 E(x)^2 = E(w^2)E(x^2) = \text{Var}(w)\text{Var}(x)
\]

If \( w \) and \( x \) are independent and have zero mean
For deep network, activation function can be saturated

\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]

\[ \tanh(x) \]

\[ \text{ReLU} \]

\[ \max(0, x) \]

tanh activation was used

Figure credit: Stanford CS231N notes
Xavier Initialization

\[ Var(Z_k^{[1]}) = N\sigma^2 \ Var(x) \]

For a deep network, the non-linear activation can be saturated for native initialization
Make the learning harder
Sensitive to hyperparameter configuration

To keep the output variance unchanged:

\[
W = \frac{\text{np.random.rand(output\_D, input\_D)}/\text{np.sqrt(input\_D)}}{\sigma = \frac{1}{\sqrt{N}}}
\]

\[
Var(Z_k^{[1]}) = \sum_{i=0}^{N-1} Var((\frac{1}{\sqrt{N}}W_{k,i}^{[1]}))Var(x_i) + Var(b^{[1]}) = Var(x)
\]

Xavier Glorot and Yoshua Bengio, “Understanding the difficulty of training deep feedforward neural networks”, AISTAT 2010
He initialization: Xavier Initialization for ReLU activation

\[ W = \text{np.random.rand(output\_D, input\_D)} \times \frac{2}{\sqrt{\text{input}\_D}} \]

\[ \sigma = \frac{2}{\sqrt{N}} \]

To compensate the ReLU function for the negative inputs

\[ \text{Var}(wx) = E(w^2)E(x^2) - E(w^2)E(x)^2 = E(w^2)E(x^2) = \text{Var}(w)E(x^2) = \frac{1}{2} \text{Var}(w)\text{Var}(x) \]

\( x \) is the response after ReLU, which has positive mean

- Use Xavier or He initialization
- Bias is initialized to zero; randomness in weights often is sufficient to drive the training
- With Batch Normalization, network training is much less sensitive to parameter initialization

He initialization: Xavier Initialization for ReLU activation

\[
\text{ReLU} \quad \max(0, x)
\]

\[x = \max(0, y)\]

\[y\] is the score before ReLU:

\[x\] is the activation after ReLU, which has positive mean

\[
E[x^2] = \int_{-\infty}^{+\infty} \max(0, y)^2 p(y) dy
\]

where the part \(y < 0\) does not contribute to the integral

\[
= \int_{0}^{+\infty} y^2 p(y) dy
\]

which we can write as half the integral over the entire real domain \((y^2)\) is symmetric around 0 and \(p(y)\) is assumed to be symmetric around 0:

\[
= \frac{1}{2} \int_{-\infty}^{+\infty} y^2 p(y) dy
\]

now subtracting zero in the square we get:

\[
= \frac{1}{2} \int_{-\infty}^{+\infty} (y - E[y])^2 p(y) dy
\]

which is

\[
= \frac{1}{2} E[(y - E[y])^2] = \frac{1}{2} \text{Var}[y]
\]

\[\text{Var}(wx) = E(w^2)E(x^2) - E(w)^2 E(x)^2 = E(w^2)E(x^2) = \text{Var}(w)E(x^2) = \frac{1}{2} \text{Var}(w)\text{Var}(x)\]

https://stats.stackexchange.com/questions/138035/variance-calculation-relu-function-deep-learning
Outline

- Set up the training, cont.
- Data pre-processing
- Model weight initialization
- Tips for model training
Debug deep neural network model can be difficult

Influenced by several factors

- Model architecture
- Implementation
- Datasets
- How the training is conducted

Initial decision
- Implementation
- Adjustment, model and data
- Iteration often slow
- Evaluation
- Bias-Variance analysis
- Hyperparameter searching
- Deployment
Before you started ...

• Have an estimation of Bayes accuracy
  o Human level performance as a surrogate
  o Expert performance – experienced operators vs. less experienced
  o Voting strategy
  o Baseline and results from publications
  o Whether it has related competition

Before you started …

• Have the Dev and Test set established

• Dev and test sets should be large enough to give a reliable estimation of model performance

• If the total amount of data is small, get more data and use cross-validation

Before you started …

- Have the evaluation metric
  Accuracy, Dice ratio, L2 distance, business specific metrics?
  Pick the key metric

 Accuracy = \frac{(TP+TN)}{(TP+TN+FP+FN)}

 Precision = \frac{TP}{(TP+FP)}

 Sensitivity = \frac{TP}{(TP+FN)}

 Specificity = \frac{TN}{(TN+FP)}

 F1 score = \frac{2 \times Pre \times Sen}{(Precision+Sensitivity)}

https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826
Before you started …

- Have the expectation
  Know when to stop
  Be the state-of-the-art?
  Meet the business needs?

Consider the resource limitation

Data and computing

90% and 90.2% may be the same for your deployment

https://paperswithcode.com/sota/image-classification-on-imagenet
Model architecture

- Start from standard choices and make changes if needed
- Look for available resources
- Look for related publications
- Read the code if possible and do not take it blindly

<table>
<thead>
<tr>
<th>Tasks</th>
<th>First try</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image classification</td>
<td>Standard ResNet, e.g. implementation in Pytorch repo</td>
</tr>
<tr>
<td>Segmentation</td>
<td>Unet architecture</td>
</tr>
<tr>
<td>Language, NLP</td>
<td>Pre-trained GPT family models</td>
</tr>
<tr>
<td>Time series signal</td>
<td>Bidirectional RNN with LSTM, transformer</td>
</tr>
<tr>
<td>Tabular data</td>
<td>Dense MLP, DropOut, BatchNorm</td>
</tr>
</tbody>
</table>
Implementation

• Pay attention to output layers

With the same base network, different functionalities can be achieved by changing the output layers:
   Pooling and dense linear for classification
   CONV layer for segmentation
   Dense linear for regression

• Pay attention to loss function – need logits or need probabilities?

different implementation may pack extra computation into loss function

```python
>>> m = nn.Sigmoid()
>>> loss = nn.BCELoss()
>>> input = torch.randn(3, requires_grad=True)
>>> target = torch.empty(3).random_(2)
>>> output = loss(m(input), target)
```

```python
>>> m = nn.LogSoftmax(dim=1)
>>> loss = nn.NLLLoss()
>>> # input is of size N x C = 3 x 5
>>> input = torch.randn(3, 5, requires_grad=True)
>>> # each element in target has to have 0 <= value < C
>>> target = torch.tensor([1, 0, 4])
>>> output = loss(m(input), target)
```
Overfit a small dataset, e.g. one mini-batch

- For classification, accuracy should quickly reach 100%
- For segmentation, Dice ratio should quickly increase to ~1.0

Very useful to find implementation bugs:

- Loss goes up
  - learning rate too high, error in output layer for class order…
- Loss is NaN
  - Overflow or underflow or divided-by-zero
- Accuracy plateaus
  - learning rate too low, bad initialization, regularization too high, data/label mismatch …
Check the initial loss function, if making sense

- Random initialized
- All output classes are treated equally

- These nodes will give roughly same scores for every class

- For the binary classification, initial loss will be close to $-\log(0.5)$
- For the multi-class classification, initial loss will be close to $-\log(1/C)$, $C$ is the number of total class
First turn off regularization/drop out/Batch Norm

$$L = \frac{1}{B} \sum_{i=0}^{B-1} L^{(i)} + \frac{\lambda}{2} \|W\|_2^2$$

- Set $\lambda = 0$ to begin with
- Make sure regularization loss not dominate total loss
- Turn off operations to introduce randomness to training, e.g. drop out
  - Add controls to allow turn them on/off easier
- Consider to turn off batch norm at first trials, since BN layer can hide problem such as NaN in computation, explosion/vanishing gradients, and make debug harder
- With BatchNorm, make sure Batch size large enough (>16)
Check **every** steps, starting from pre-processing

Errors can happen in pre-processing, such as
- forget to change label after processing images (e.g. image scaling for segmentation)

Errors can happen in datasets:
- forget the shuffle the samples, some samples are corrupted …

Errors can happen when loading data:
- image/label mismatch

Some images are blanket!

L2 norm of all samples

Scale the image and forget to scale mask
Check immediate inputs to your model training code

Print/plot/save a few mini-batch and check them before training

Batch A

Batch B
Check immediate outputs of your model before any post-processing

scores = model(images)

Model outputs are spatial probability maps

Goal: detect key landmarks

Postprocessing: extract landmark location
Add Debug mode and verbose mode, add `working_dir`

```python
if (self.working_dir is not None):
    self.save_batch(self.working_dir, images, batch_no)

scores = model(images)

if (self.working_dir is not None):
    self.save_output(self.working_dir, scores, batch_no)
```

- Easily enable or disable
- Fast, not slow down training if disabled
- Save intermediate results with its index/name/time stamp/data role …
- Keep these code lines in your repo
Monitor the learning curves

Loss is not decreasing quickly

- learning rate too low
- model underfitting

try to fix the problem by fitting to a tiny dataset

Monitor the learning curves

Loss goes up for dev set and goes down for training set

- model overfitting
- add regularization
- reduce mode capacity
- add more training data
Monitor the learning curves

Loss shows strong variance in dev set

- model overfitting
- training set too small
- Add more sample to training set
Monitor the learning curves

Loss is lower for dev set than train set

dev set is easier than train set
too much data augmentation
dev/tra set distribution mismatch
exam dev set
Monitor the learning curves

**Good training**

Accuracy is high, but still increasing
Consider to train longer

Note with one-cycle scheduler, at high LR, the accuracy can dip
Use Debugger

Tensor shape mismatch
Loss function exceptions
Nan or inf error
All other exceptions …

• set breakpoints
• check variable values
• interactive debugging

or, pdb or ipdb for CLI
experiences of debugging
Use Experimental management tools

- Record your training and parameters
- Integrated with python training code
- Support hyperparameter searching
- Easy to compare different experiments

Many iterations may be needed to reach required performance

My model is finally good, but I forget how it was trained!
Do not use hyperparameter searching as the first remedy

- Start hyperparameter searching after fixing obvious bias/variance problems
- Often large performance gain is from error analysis and collecting specific new data for problems found
- Fix bias/variance problems can change model and which hyperparameters to tune

Initial decision

- Implementation
- Bias-Variance analysis
- Adjustment, model and data

Iteration

- Evaluation
- Hyperparameter searching

Deployment

- Fix bias
- Fix variance
- Fix train/dev data mismatch

Settings

- Has new issues?

Yes

No