COMP 790-125: Goals for today

Practical tricks for training neural networks

- Data preprocessing
- Parameter initialization
- Regularization techniques
- Hyperparameter search

Parameters: network weights and bias terms
Hyperparameters: architecture, step-size, scaling constants for different parts of the objective etc.
Data preprocessing

- Centering
- Standardization and normalization
- Whitening, ZCA
Centering data

Centering or mean subtraction is a procedure that makes each feature have zero mean.

\[
\mu_i = \frac{1}{N} \sum_{i=1}^{N} x_{i,j}
\]

\[
x_{i,j}^{\text{centered}} = x_{i,j} - \mu_i
\]

Note:

- Only applicable to continuous features
- Removes large offset in features
- Features reflect deviation from the mean
- Some feature values become negative
- Linear combination of centered features has zero mean
Centering or mean subtraction is a procedure that makes each feature have zero mean.

\[
\sigma_i = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_{i,j}^{\text{centered}})^2}
\]

\[
x_{i,j}^{\text{standardized}} = \frac{x_{i,j}^{\text{centered}}}{\sigma_i}
\]

\[
\|x_i^{\text{centered}}\| = \sqrt{\sum_{i=1}^{N} (x_{i,j}^{\text{centered}})^2}
\]

\[
x_{i,j}^{\text{normalized}} = \frac{x_{i,j}^{\text{centered}}}{\|x_i^{\text{centered}}\|}
\]
Standardizing/normalizing data

- Only applicable to continuous features.
- Both remove scale in features
- Standardized feature vector has norm of $\sqrt{N}$
- Normalized feature vector has norm of 1
- Feature variance is lost
- Shrinks highly variable features; inflates less variable features
Whitening

Standardization and normalization scale each feature separately. However, these operations do not remove correlation between features.

One way to accomplish this is by using PCA. Recall

$$\text{Cov}(\mathbf{X}) = \frac{1}{N} (\mathbf{X}_{\text{centered}})^T \mathbf{X}_{\text{centered}}$$

Principal Component Analysis uses eigen decomposition of covariance

$$\text{Cov}(\mathbf{X}) = \mathbf{V} \text{diag}(\mathbf{d}) \mathbf{V}^T$$

where \text{diag} constructs a diagonal matrix from a vector, \( \mathbf{V} \) is an orthogonal matrix, and \( \mathbf{d} \) is a vector.
Whitening

\[ \text{Cov}(X) = V \text{diag}(d)V^T \]

- columns of \( V \) are eigenvectors corresponding to entries of \( d \), eigenvalues
- large eigenvalues \( d_{i,i} \) correspond to directions of large variability \( v_i \)
- first \( k \) columns of \( V \) are the top \( k \) principal components
Whitening

Let $\hat{V}$ denote a matrix composed of first $k$ columns of $V$ – top $k$ principal components. Let $\hat{d}$ denote a vector of first $k$ entries of $d$. Whitening is performed by

$$X^{\text{whitened}} = (X^{\text{centered}})^T \hat{V} \text{diag}(\hat{d} + \epsilon)^{-\frac{1}{2}}$$

where $\epsilon$ is a small number, for example $10^{-5}$. In contrast, standardization

$$X^{\text{standardized}} = (X^{\text{centered}})^T \text{I} \text{diag}(\sigma^2)^{-\frac{1}{2}}$$
Zero-phase component analysis – ZCA

An additional step, after whitening, can be used to get data back to the original feature space

\[ \mathbf{x}^{\text{ZCA whitened}} = \mathbf{x}^{\text{whitened}} \mathbf{v}^T \]

Or more completely

\[ \mathbf{x}^{\text{ZCA whitened}} = (\mathbf{x}^{\text{centered}})^T \begin{pmatrix} \hat{\mathbf{v}} & \text{diag}(\mathbf{d} + \epsilon)^{-\frac{1}{2}} & \hat{\mathbf{v}}^T \end{pmatrix} \]

rotate \hspace{1cm} \text{standardize PCA scores} \hspace{1cm} \text{rotate back}

And again for contrast

\[ \mathbf{x}^{\text{standardized}} = (\mathbf{x}^{\text{centered}})^T \begin{pmatrix} \mathbf{I} & \text{diag}(\sigma^2)^{-\frac{1}{2}} & \mathbf{I} \end{pmatrix} \]

\hspace{1cm} \text{no rotation} \hspace{1cm} \text{standardize features} \hspace{1cm} \text{no rotation}

On the board, we will illustrate the difference of these two techniques.
Data preprocessing

Note that all of our preprocessing techniques, such as centering, computed quantities used for data transformation, such as mean. It is important that these quantities are not computed on test data. Given training data mean, you can subtract that mean from the test data during the test time.

If your test data is any way touched by your code during training, you are making a mistake.
Parameter initialization

Typical deep network has several layers. Most layers have a large number of parameters associated with it:

- convolutional layer – filter weights
- fully connected layer – connection weights, biases

Initializing these parameters improperly can lead to bad local minima.
Q: Let’s consider a simple network with a single hidden layer. Why would setting all the weights to zero be a bad idea?

Another idea: initialize network weights with values drawn from a Gaussian with zero mean and small variance (0.1).

Q: What happens if a single unit has thousands of inputs?
Parameter initialization – considering variance of unit output

Given a linear unit which computes $z = \sum_{i=1}^{n_{in}} w_i h_i$ we can ask what is the variance of $z$

$$\text{Var}(z) = \sum_{i=1}^{n_{in}} w_i \text{Var}(h_i)$$

assuming all the units have the same variance $\text{Var}(h_1) = \text{Var}(h_2) = \ldots = \text{Var}(z)$ and zero mean

$$\text{Var}(z) = n_{in} \text{Var}(w_i) \text{Var}(h_i)$$

then weights sampled from distribution with $\text{Var}(w_i) = \frac{1}{n_{in}}$ lead to units with the same variance.
Parameter initialization – considering variance of unit output\(^1\)

Given an neuron with \(n_{\text{in}}\) input we can draw weights from

\[
\mathcal{N}(0, \frac{1}{n_{\text{in}}}).
\]

Another initialization takes into account the number of units that take input from the unit whose weights we are initializing (\(n_{\text{out}}\)).

\[
\mathcal{N}(0, \frac{2}{n_{\text{in}} + n_{\text{out}}}).
\]

or

\[
Unif \left( -\sqrt{\frac{6}{n_{\text{in}} + n_{\text{out}}}}, +\sqrt{\frac{6}{n_{\text{in}} + n_{\text{out}}}} \right)
\]

\(^1\text{http://jmlr.org/proceedings/papers/v9/glorot10a/glorot10a.pdf}\)
Regularization techniques

- Weight decay
- Early stopping
- Dropout
- Dataset augmentations
Weight decay

Weight decay is a very simple regularization technique:

\[
\text{loss}(w; X) + \frac{\lambda}{2} \sum_i w_i^2
\]

If we have already implemented gradient computation for the loss

\[
\nabla_w \text{loss}(w; X) + \lambda w
\]

In the context of linear regression, we know that this regularization technique

► shrinks regression weights
► make correlated predictors’ weights similar

In the context of deep nets, this penalty is applied equally to parameters in all layers.
Early stopping

A poor man’s regularization technique, which we end up applying regardless.

In a vanilla stochastic gradient descent

\[ w^{(k+1)} = w^{(k)} - \eta(k) \nabla_w \text{loss}(w^{(k)}; X_{\text{MiniBatch}_k}) \]

where \( \eta(k) \) is a learning rate (step size) and the direction of the step is gradient of loss on a minibatch.

For this algorithm to converge, gradient descent needs to be run for a large number of iterations.

Stopping before convergence, \textbf{early stopping}, can be seen as a form of regularization.

Q: Reason for me what happens in early stopping. Think small step size of 0.0001. What happens if we stop after \( k = 1 \)? Generalize from there.
Dropout – intuition

Imagine having many networks, they share parameters, but some units are removed, dropped out.

If there is \( n \) units in a network, there are \( 2^n \) possible dropout networks.

The complete network below has 15 nodes. A decimated network is shown on the right; 7 nodes have been dropped-out.

---

\(^2\)https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf
Dropout – intuition

All of the dropout networks have to share parameters. The only way that they can make a good prediction is if they do not rely on presence on any one node in the network too strongly.

(a) Standard Neural Net

(b) After applying dropout.
Dropout – practice

In practice, we cannot instantiate all of the $2^n$ possible networks and use them to train the weights.

Instead, we consider a random subset of these networks.

During training, we randomly dropout units according to probability $p$.

Forward propagation:

$$h_l = f(W(h_{l-1} \circ u_{l-1}) + b)$$

where $u_{l-1}$ denotes a mask, and $\circ$ is element-wise product

$$a \circ b = [a_1 b_1 \ a_2 b_2 \ .... \ a_n b_n]$$
How do we predict during test time?

In training time, the network is decimated. A fraction of units is present.

During test time, we use the whole network. Hence, average input into any one unit is going to be larger than what we saw during training.
Dropout – scaling in training time

In order to address the asymmetry during the test time, we can let

\[ r_{l,i} \sim \text{Bernoulli}(p) \]

\[ u_{l,i} = \begin{cases} 
0, & r_{l,i} = 0 \\
\frac{1}{p}, & r_{l,i} = 1
\end{cases} \]

Hence, in training, during forward propagation

\[ h_l = f(W(h_{l-1} \circ u_{l-1}) + b) \]

we mask and scale-up remaining units outputs.

In test time, during forward propagation

\[ h_l = f(Wh_{l-1} + b). \]
Dataset augmentation

Data augmentation is not strictly a regularization technique.
The idea is relatively simple: Take existing data and generate additional samples by applying simple transformations

For example:
- rotate, stretch images
- add independent noise
- domain specific perturbations: vocal tract length perturbation in speech

Note that naive augmentation can cause problems – creating a new example of digit ‘6’ by rotating by 180°.
Hyperparameter search

- Grid search
- Random search
- Bayesian optimization
Deep learning is plagued with hyperparameter setting

- learning rate
- weight decay penalty scale ($\lambda$)
- amount of noise in training denoising autoencoders
- number of layers and number units in different layers
- dropout probability $p$
- parameters of data augmentation techniques
Hyperparameters – grid search

For any hyperparameter $\lambda_i$ select a set of possible values $\Lambda_i$.

Try all vectors $\lambda$ in the set $\Lambda_1 \times \Lambda_2 \times \ldots \Lambda_s$ where $s$ is the number of hyperparameters.

The number of combinations is $\prod |\Lambda_i|$ where $|\cdot|$ denotes cardinality of the set.

For example

- possible weight decay constants $\{0.1, 0.01, 0.001, 0.0001\}$
- layer sizes $\{10, 20, 30\}$
- noise level for denoising autoencoder $\{0.01, 0.02, 0.03, 0.04, 0.05\}$

leads to $4 \times 3 \times 5 = 60$ different settings.
An alternative – random search\textsuperscript{3}

Two axis correspond to hyperparameter values. Curves on the top and side show the impact of the hyperparameter. Grid based search repeats testing the same hyperparameter value. Random approach explores both hyperparameters’ spaces more efficiently.

\textsuperscript{3}http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf
Even better – bayesian optimization\textsuperscript{4}

The idea: treat network’s performance on validation set as output, and hyperparameters as inputs. Apply Bayesian optimization.

As a reminder one algorithm for Bayesian Optimization using Gaussian Processes:

\textbf{Algorithm 1} The GP-UCB algorithm.

\begin{verbatim}
  Input: Input space \( D \), GP Prior \( \mu_0 = 0, \sigma_0, k \)
  for \( t = 1, 2, \ldots \) do
    Choose \( x_t = \operatorname{argmax}_{x \in D} \mu_{t-1}(x) + \sqrt{\beta_t \sigma_{t-1}(x)} \)
    Sample \( y_t = f(x_t) + \epsilon_t \)
    Perform Bayesian update to obtain \( \mu_t \) and \( \sigma_t \)
  end for
\end{verbatim}

\[\]
COMP 790-125: Goals for today

Practical tricks for training neural networks

- Data preprocessing
- Parameter initialization
- Regularization techniques
- Hyperparameter search