Multi-Layer Perceptron (MLP)
Training Issues I

M. R. Hasan
Readings

• Bishop: 5.1, 5.3, 5.5
• Murphy: 16.5, 16.5.4
• Alpaydin: 11
• Geron: 11
The Art & Science of Training Deep MLPs

• We will discuss some key issues regarding the training of MLPs.

• Some issues concern deep MLPs (with many hidden layers and neurons) or more generally deep neural networks (DNNs).
Issues Regarding Training Deep MLPs

- We will discuss the following issues.
  - Feature scaling
  - Intractability of computation
  - Computational complexity
  - Overfitting due to complex architecture
  - Overfitting due to overtraining
  - Weight initialization
  - Hidden Layer Activation Functions
  - Optimizing Stochastic Gradient Descent
  - Prediction invariance
  - Scalability with respect to the input size
Training Issues:
Feature Scaling
Training Issues: Feature Scaling

• For training MLPs, we need to **scale the features**.
• There are **two main reasons** for feature scaling.
  - Effective learning
  - Faster convergence
Training Issues: Feature Scaling

- Effective Learning:
- Consider the following loss gradient equation used in the SGD update rule.

\[
\nabla \mathcal{L}(W^{(k)}) = a^{(k)T} \delta^{(k+1)}
\]

\[
W^{(k)} = W^{(k)} - \eta \nabla \mathcal{L}(W^{(k)})
\]

Notice that **feature vector** \(a^k\) (extracted from the input \(x\)) is used in the weight updates (multiplied with the error term).
Training Issues: Feature Scaling

• Effective Learning:
• Thus, if features (components of $a^k$) have varying scales, then certain weights may update faster than others.

As a consequence, some features will have larger weights leading to an unstable learning.

$$\nabla \mathcal{L}(W^{(k)}) = a^{(k)T} \delta^{(k+1)}$$

$$W^{(k)} = W^{(k)} - \eta \nabla \mathcal{L}(W^{(k)})$$
Training Issues: Feature Scaling

• Effective Learning:
• By feature scaling (normalizing the components of $a^k$) we ensure that one feature doesn’t impact the model just because of its large magnitude.

\[
\nabla \mathcal{L}(W^{(k)}) = a^{(k)T} \delta^{(k+1)}
\]

\[
W^{(k)} = W^{(k)} - \eta \nabla \mathcal{L}(W^{(k)})
\]
Training Issues: Feature Scaling

• Fast Convergence:
• If some features have a **very large scale** then SGD takes **more numbers of iterations** to converge.
• Left: features have the same scale
• Right: feature 1 has much smaller values than feature 2
The most common **techniques of feature scaling**:

- Standardization: transforms the data to have 0 mean and a variance of 1.
- Normalization: transforms the data to bound the values between two numbers, typically, between [0, 1].
Training Issues: Feature Scaling

- **Standardization** \((z\text{-score normalization})\):
- First we make an assumption that the distribution of the features are **Gaussian**.

We determine the distribution **mean** and **standard deviation** for each feature.

Next we **subtract the mean** from each feature.

Then we **divide the values** (mean is already subtracted) of each feature by its standard deviation.

\[
x' = \frac{x - \bar{x}}{\sigma}
\]

This process standardizes features by **removing the mean** and scaling to unit variance.
Training Issues: Feature Scaling

- **Normalization** (min-max scaling):
  - The data is scaled to a fixed range, usually 0 to 1.
  - The cost of having this bounded range is that we will end up with **smaller standard deviations**, which can suppress the effect of outliers.
  - A Min-Max scaling is typically done via the following equation:

\[
X_{\text{norm}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}
\]

Example: grayscale image pixel value range 0 ~ 255

\[
X_{\text{norm}} = \frac{X - 0}{255 - 0} = \frac{X}{255}
\]
Issues Regarding Training Deep MLPs

- We will discuss the following issues.
  - Feature scaling
    - Intractability of computation
  - Computational complexity
  - Overfitting due to complex architecture
  - Overfitting due to overtraining
  - Weight initialization
  - Activation Functions
  - Optimizing Stochastic Gradient Descent
  - Prediction invariance
  - Scalability with respect to the input size
Training Issues: Intractable Computation
Training Issues: Intractable Computation

• The **training time** in Deep MLPs could be **intractable**!
Consider the error equations in backpropagation.

Observe that the errors at the hidden layers are defined recursively.

\[
\delta^{(k)} = \left( \delta^{(k+1)} W^{(k)T}_{\text{no bias}} \right) \ast g'(z^{(k)})
\]
Training Issues: Intractable Computation

- Observe that the errors at the hidden layers are defined recursively.

\[
\delta^{(k)} = \left( \delta^{(k+1)} \mathbf{W}_{no\ bias}^{(k)^T} \right) \ast g'(z^{(k)})
\]
Training Issues: Intractable Computation

Consider the calculation of error at the 2\textsuperscript{nd} layer.

\[
\delta_1^{(2)} = g'(z_1^{(2)}) \left[ W_{11}^{(2)} \delta_1^{(3)} + W_{12}^{(2)} \delta_2^{(3)} \right]
\]

\[
\delta_2^{(2)} = g'(z_2^{(2)}) \left[ W_{21}^{(2)} \delta_1^{(3)} + W_{22}^{(2)} \delta_2^{(3)} \right]
\]

Hidden layer error

\[
\delta_j^{(k)} = g'(z_j^{(k)}) \sum_{p} W_{jp}^{(k)} \delta_p^{(k+1)}
\]

layer \((k+1)\) except the bias

It will result into an exponential rise of complexity!

We are recomputing the error from each neuron of the previous layer.
Training Issues: Intractable Computation

- How do we mitigate this issue?
- Observe the nature of the problem.
- We have recursion and overlapping subproblems.
- Use dynamic programming!!!

\[
\delta^{(k)} = \left(\delta^{(k+1)} \mathbf{W}^{(k)\top}_{\text{no bias}}\right) * g'(\mathbf{z}^{(k)})
\]

Hidden layer error

Cache the error terms and reuse.

Don’t recompute!
Training Issues: Intractable Computation

- **Reflection**: Training an MLP was a big challenge because of the non-convexity and the sheer size of the MLP.
- How did we handle this issue?
- **Gradient descent** with some added tricks!

![Diagram showing MLP layers and chain rule for partial derivative]
Issues Regarding Training Deep MLPs

• We will discuss the following issues.
  - Feature scaling
  - Intractability of computation
  - Computational complexity
  - Overfitting due to complex architecture
  - Overfitting due to overtraining
  - Weight initialization
  - Hidden Layer Activation Functions
  - Optimizing Stochastic Gradient Descent
  - Prediction invariance
  - Scalability with respect to the input size
Training Issues: Computational Complexity
MLP: Complexity of Learning

• Let’s look at the **space and time complexity** of MLPs.
• An MLP with d inputs, M hidden units (neurons), and K outputs has the following no. of weights:

1\textsuperscript{st} Layer: 
\( M(d + 1) \) weights

2\textsuperscript{nd} Layer: 
\( K(M + 1) \) weights
MLP: Complexity of Learning

- Hence, both the **space and time complexity** of an MLP are:

  $$O(M \cdot (d + K)).$$
MLP: Complexity of Learning

- When $e$ denotes the number of training epochs, the training time complexity is:
  \[ O(e \cdot M \cdot (d + K)). \]
MLP: Complexity of Learning

- In a problem, \(d\) and \(K\) are predefined.
- The parameter that we play with to tune the complexity of the model is \(M\).

What happens if we create a complex MLP architecture with many neurons?
Issues Regarding Training Deep MLPs

• We will discuss the following issues.
  - Feature scaling
  - Intractability of computation
  - Computational complexity
  - **Overfitting due to complex architecture**
  - Overfitting due to overtraining
  - Weight initialization
  - Hidden Layer Activation Functions
  - Optimizing Stochastic Gradient Descent
  - Prediction invariance
  - Scalability with respect to the input size
Training Issues: Overfitting due to Complex Architecture
MLP: Overfitting

• Due to a large number of computational units or neurons, MLPs are **expressive**.
• However, this extreme flexibility causes **severe overfitting**.
• Let’s illustrate this problem.
Example: As complexity (number of hidden layer neurons) increases, training error remains fixed but the validation error starts to increase and the network starts to overfit.
MLP: Overfitting

• To combat overfitting caused by the network architecture, we need to **regularize** the model.
• The two popular regularization approaches are:
  - Weight Decay
  - Dropout
Combat Architectural Overfitting: Weight Decay Regularization
MLP: Overfitting

• Weight Decay:
• Regularize the MLP by **updating** its loss/cost function.
• It **penalizes** the model in proportion to the size of the model weights.

\[ L_{\text{regularized}}(\vec{w}_i) = L(\vec{w}_i) + \frac{\lambda}{2} \vec{w}_i^2 \]
MLP: Overfitting

• Weight Decay: The main idea is that some connections may not be necessary.
• We turn them off by setting their weights to 0.
• We give each connection a tendency to decay to 0 so that it disappears unless it is reinforced explicitly to decrease error.
MLP: Overfitting

- Weight Decay:
- For implementing weight decay we usually use the $l_p$ regularizer, where $p$ is any positive integer starting from 1.
  - $p = 1$: $l_1$ regularizer
  - $p = 2$: $l_2$ regularizer

$l_2$ Regularization:

$$L_{regularized}(\vec{w}_i) = L(\vec{w}_i) + \frac{\lambda}{2} \vec{w}_i^2$$
Combat Architectural Overfitting: Dropout Regularization
MLP: Overfitting

• Dropout:
• At each iteration, we drop every neuron (set its weight to zero) temporarily with a probability $p$.
• Here $p$ is a hyperparameter, which is known as the dropout rate.
• Its value is usually set between 10% to 50%.
MLP: Overfitting

- Dropout:
- It includes input layer “neurons” but excludes the output neurons.
- Dropout regularization technique was proposed by Hinton in 2012: [http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf](http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf)
MLP: Overfitting

• Dropout is an **effective** regularization technique.
• Let’s see why.
• During training, neurons tend to **co-adapt** with their **neighboring neurons** thereby overfit the training data.
• Due to random dropout, the **neurons become independent** as they cannot co-adapt.
MLP: Overfitting

- Thus, each individual neuron learns to perform effectively on its own.
- As a result, they become more **immune to the changes** in small variations in the input.
- When a Dropout trained MLP is fed with unseen test data, it **generalizes better**.
MLP: Overfitting

- Another useful aspect of Dropout is that at each iteration it creates a unique MLP.
- Because every neuron could be either on or off.
- Note that for $N$ neurons there are $2^N$ possible networks.
- Thus, it is unlikely that two networks at two iterations would be the same.
MLP: Overfitting

• As a result, at the end of training the network for $t$ iterations, there would be an **ensemble of $t$ unique networks**.

• The resulting network can be seen as an **averaging ensemble** of all these networks.

![Diagram showing network iterations](image)
MLP: Overfitting

- The inherent **diversity** of the ensemble improves the generalization performance.
- See the following notebook for an empirical understanding of dropout regularization:
Issues Regarding Training Deep MLPs

- Feature scaling
- Intractability of computation
- Computational complexity
- Overfitting due to complex architecture
- Overfitting due to overtraining
- Weight initialization
- Hidden Layer Activation Functions
- Optimizing Stochastic Gradient Descent
- Prediction invariance
- Scalability with respect to the input size
Training Issues: Overfitting due to Overtraining
MLP: Overtraining

- Another consequence of increased complexity (larger number of hidden units) is that as more training epochs are made, the error on the training set decreases.

However, the error on the validation set starts to increase beyond a certain point.

This problem is the direct result of overtraining!
MLP: Overtraining

• Why does longer training time reduce training error?
• Note that initially all the weights are close to 0 and thus have little effect.
• As training continues, the most important weights start moving away from 0 and are utilized.
MLP: Overtraining

• But if training is continued further on to get less and less error on the training set, almost all weights are updated away from 0.
• They effectively become parameters.

Thus as training continues, it is as if new parameters are added to the system, increasing the complexity and leading to poor generalization.
Combat Overtraining-caused Overfitting: Early Stopping Regularization
MLP: Overtraining

• To solve the overfitting problem of overtraining, **learning should be stopped early**.

• The optimal point to stop training, and the optimal number of hidden units, is determined through **cross-validation**.
MLP: Overtraining

- Since cross-validation is infeasible in deep MLPs, we set aside a **fixed validation set** for early stopping.
- The validation set is also used to create **comparative learning curves**.
Combat Overfitting: 3 Regularization Techniques
Summary: 3 Regularization Techniques to Combat Overfitting

• We have discussed three regularization techniques.
  - Weight Decay ($l_p$ regularization)
  - Dropout
  - Early Stopping
• In Deep Neural Networks, the dropout regularization has been found to be the most effective.