CPE504

Artificial Neural Networks (ANNs)

4.0. MULTI-LAYERED NEURAL NETWORKS: FUNDAMENTALS

What you will learn

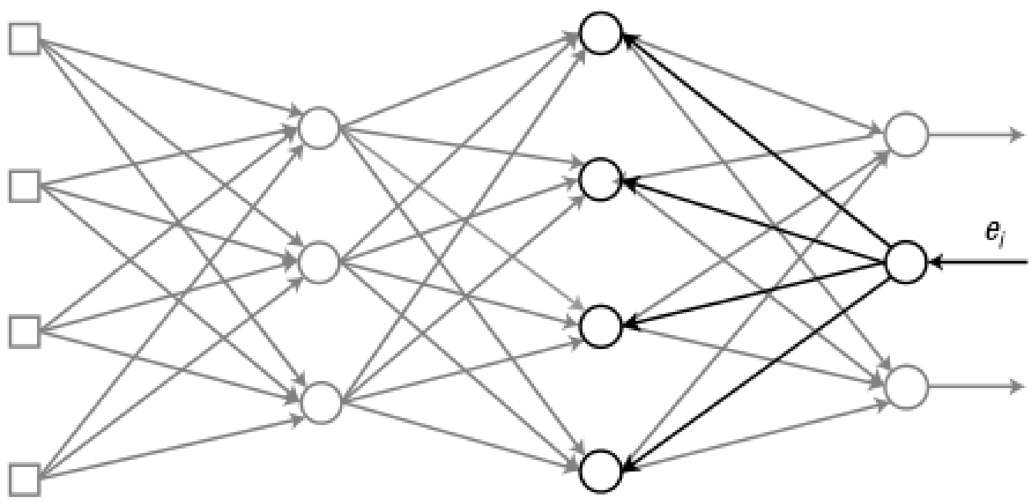
- The Great Stagnation
- Back-propagation Algorithm
- Filtering: GD with Momentum
- Learning with Cost Functions: Least-Squares and Cross-entropy
- Regularized Costs: Overcoming Overfitting
- Unanswered Problems
- Assignment: Algorithm Implementations
- Summary

The Great Stagnation

- We **wanted** to overcome the **practical limitations** of the **perceptron**, so a viable option was to evolve into a **multi-layer architecture**.
- However, it took about 3 decades to add just one single hidden layer of nodes to make the single-layer ANN become multi-layered.
- The **main issue** was essentially a **training (learning) problem**.
- To **learn**, **information** has to be **stored**, else such a network formulation would be useless.
- Unfortunately **feedback error**, the **essential element for learning**, was explicitly **undefined in the hidden layers**. The GD algorithm (delta rule) could only be **applied** to the **outermost output layer** because that was the only layer that benefited from the feedback error.
- This **error** of the output node is defined as the **difference** between the **correct** (**expected**) output and the **actual** output of the neural network.
- However, the **training data does not provide correct outputs for the hidden layer nodes**, and **so error cannot be fedback using the same approach** for the output nodes.
- The **technical problem** then was **how to define (or represent) the errors seen at the hidden nodes, so we can use the GD algorithm?**

The Great Stagnation

- The answer (use a **back-propagation** algorithm (**BP**)) was arguably found in 1986.
- See **Rumelhart**, D. E., **Hinton**, G. E., **Williams** R. J. (**1986**). "**Learning representations by back-propagating errors**". *Nature*
- The **significance** of BP was that it provided a **systematic** method to determine the **error of the hidden nodes**, so that the **SGD** can then be **applied** to **adjust the weights**.
- The **error BP** process is **illustrated** in the **image** on the **left**.
- Typically, **input signal/data** of the ANN is **fed forward** from the **input layer through the hidden layer(s) to the output layer**.
- In **contrast** in the error BP process, the **output error information** is **fed backwards** from the **output layer through the hidden layer(s) to the input layer**.



Backpropagation (BP)

✓ **Initialize** the **weights** with adequate values and enter the **input** from the **training data** { input, correct output } and obtain the neural network's output.

✓ **Compute** the **error** of the output to the correct output and the **delta**, δ, of the output nodes.

- $e = d - y$
- $\delta = y'\ e$

✓ **Propagate** the **output** node **delta**, δ, **backward**, and **compute** the **deltas** of the immediate next (left) nodes.

- $e^{k} = W^{T\,(k-1)} \delta^{k-1}$
- $\delta^{k} = y'^{(k)}\ e^{k}$

✓ **Repeat** the previous step **until it reaches the hidden layer** that is on the **immediate right of the input layer**.

✓ **Adjust** the **weights** according to the following **GD learning rule**.

- $\Delta W^{k} = \delta^{k}\, x^{T\,k}$
- $W^{k} = W^{k} + \alpha\, \Delta W^{k}$

✓ Repeat Steps 2-5 for every training data point.

✓ Repeat Steps 2-6 until the neural network is properly trained.

This two-step recursive back-propagation algorithm is applicable for training many hidden layers. **BP** is **a form** of **Automatic Differentiation**.

Filtering: GD with Momentum

- The **benefits** of using a more advanced weight adjustment formulas is essentially **two**: **better descent stability** and **faster convergence speed** in the **training** (learning) process of the ANN.
- These **characteristics** are especially favorable for **Deep Learning** as it is **harder to train**. Here we cover an advance to the **GD rule**, often called **SGD with momentum**, which have been used for a long time. There are various Momentum-based GD rules in the literature, of which the **ADAM algorithm** has been popular in contemporary times.
- However, such momentum-based operations are just **moving-average filtering transformations** of the **propagated gradient** (of the **cost-function**).
  - $v = \beta v + \alpha \Delta W$ (filter)
  - $W = W + v$ (update)
- The **performance comparisons** of advanced momentum-based GD rules with the standard SGD with momentum algorithm are **controversial**. They may or may not give better performance. Sometimes using only the GD rule may give the best performance.
- Another thing to note here is that **algorithm implementation is very important**.

Learning with Cost Functions

- **Cost functions** are used to **derive the learning rule** for tackling a particular **optimization problem**. It is an **integral part** of optimization.
- There is really at the moment, no known computational learning that can take place without a cost-function.

Standard Least-Squares Cost (LSQ)

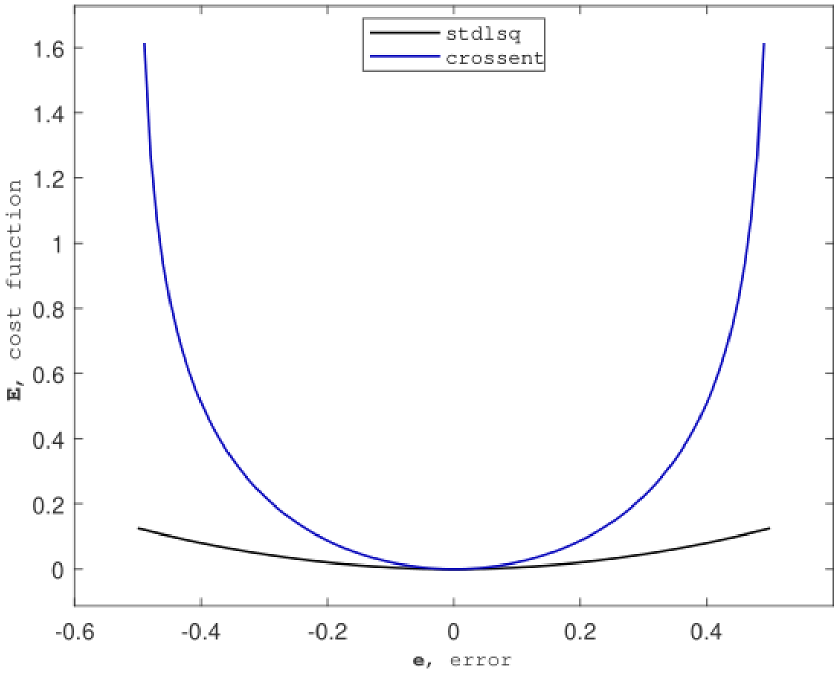
- $\mathbf{E} = 0.5 \times \sum_{i}^{m} \mathbf{e}^2$

Cross-Entropy Cost (CE)

- $\mathbf{E} = \sum_{i}^{m} - \mathbf{r} \ln \mathbf{y} - (\mathbf{1}-\mathbf{r}) \ln(\mathbf{1}-\mathbf{y})$

for i=1,…,m output nodes

- The **cost-function value** is usually made **proportional** to an **error value or variance measure.**
- **Cost Function implies a** *Error/Objective/Goal/Loss/Performance Index Function*

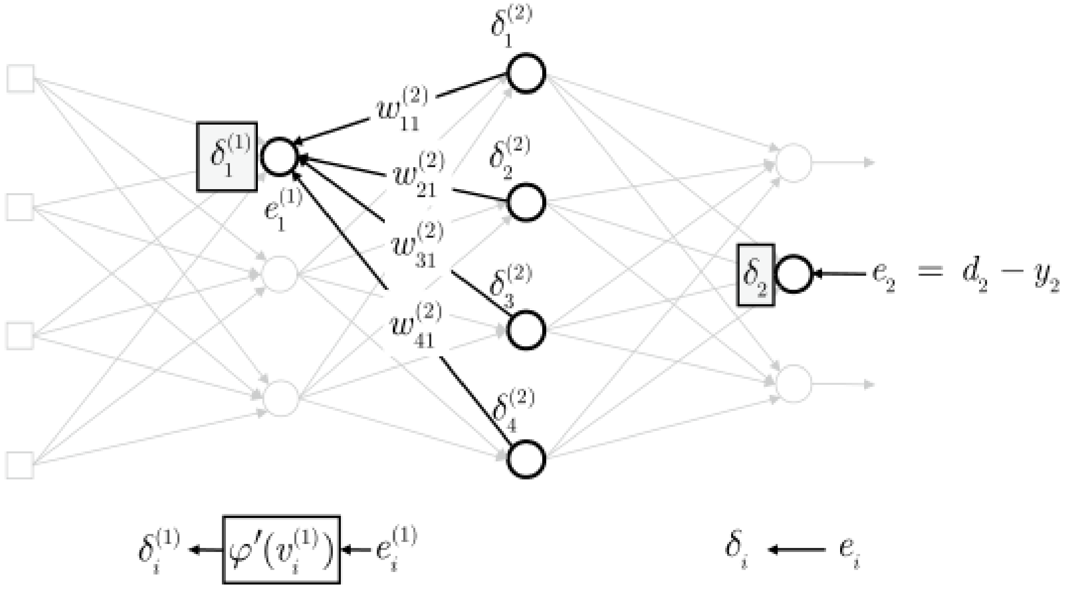


Learning with Cost Functions

- It turns out that the cross-entropy cost **penalizes errors in a geometric (logarithmic) manner**, hence it is **more sensitive to error**. For this reason, **learning rules derived from this cost often yield superior training performance**
- The cross-entropy cost was derived using **logistic-sigmoid output functions** such as the *simple-sigmoid* and *softmax activation functions* heavily used in the **output node architecture** of almost all ANN models.
- **Optimization** *is simply a zero-finding operation*. That is **minimizing** or **maximizing** a cost-function close or equal to zero. *Essentially, all* ***costs*** *follow the* **least-squares principle, also known as** *quadratic costs*
- It turns out that the **cross-entropy** itself is *a* ***weighted least-squares cost***.
- $\mathbf{E} = 0.5 \times \sum_{i}^{m} \lambda_e \, e^2$
- *where:*
- $\lambda_e = \text{inverse}(y')$ for the **cross-entropy cost,** and
- $\lambda_e = 1$ for the **standard least-squares cost.**
- ΔW is the **gradient of the cost-function** $\frac{\partial E}{\partial W}$

Cross-entropy vs Least-squares

- The *performance and implementation* of the **GD learning rule** *depends* on the **cost-function parametrization or design**.
- *Specifically*, for the case of **optimizing ANN models** with hidden layers, the cost-function *affects* the computation of the **delta** at the **output node**.
- The difference may seem insignificant. However, *cost function is a huge topic in optimization theory*. Most of the **neural network training approaches of Deep Learning** employ the cross entropy-driven GD learning rule. This is due to faster learning rate process and performance.
- Using cross-entropy, the BP procedure described then becomes the following.



Cross-entropy driven GD

✓ **Initialize** the **weights** with adequate values and enter the **input** from the **training data** { input, correct output } and obtain the neural network's output.

✓ **Compute** the **error** of the output to the correct output and the **delta**, δ, of the output nodes.

- $e = d - y$
- $\delta = e$

✓ **Propagate** the **output** node **delta**, δ, **backward**, and **compute** the **deltas** of the immediate next (left) nodes.

- $e^{k} = W^{T\,(k-1)} \delta^{k-1}$
- $\delta^{k} = y'^{(k)}\, e^{k}$

✓ **Repeat** the previous step **until it reaches the hidden layer** that is on the **immediate right of the input layer**.

✓ **Adjust** the **weights** according to the following **GD learning rule**.

- $\Delta W^{k} = \delta^{k} x^{T\,k}$
- $W^{k} = W^{k} + \alpha\, \Delta W^{k}, \quad \alpha = \left\|\Delta W^{k}\right\|^{-1}$

✓ Repeat Steps 2-5 for every training data point.

✓ Repeat Steps 2-6 until the neural network is properly trained.

This **recursive** back-propagation algorithm is applicable for training many hidden layers. Remember, **BP** is **a form** of **Automatic Differentiation**.

Regularized Costs: Overcoming Overfitting

- To **prevent** an **overfitted** model, a concept known as **regularization** is used.
- This is known to essentially mean making the learned weights of the ML model simpler or smaller. **What this means is**:
- Smaller control weights, can disconnect nodes in a complex ANN model of many hidden layers, making the model simpler.

- Mathematically, this can be achieved by adding a weighted least-square variance of the connection weights (control inputs) to the least square error component of the cost-function.
- $\mathbf{E} = 0.5 \times [\|\lambda_e\, \mathbf{e}^2\| + \|\lambda_w\, \Delta W^2\|]$
- The above is known sometimes as **L2-norm (or ridge) regularization**
- **Typically, λ_w is set to a small number to prevent underfitting.**

- The weight-adjustment then becomes of the following forms:

Coupled regularization

- $\Delta W^{k} = \delta^{k}\, x^{T\,k} + \lambda_w\, W^{k}$
- $W^{k} = W^{k} + \alpha\, \Delta W^{k}, \quad \alpha = \left\|\Delta W^{k}\right\|^{-1}$

Decoupled regularization

- $\Delta W^{k} = \delta^{k}\, x^{T\,k}$
- $W^{k} = (1 + \frac{\lambda_w}{\rho})\, W^{k} + \alpha\, \Delta W^{k}, \quad \rho = \left\|W^{k}\right\|^{-1}$

- *Another way* to do **regularization** is an approach known as **drop-out**, which **applies randomized permutation to disconnect nodes** (setting weights to zero) in the **hidden layers during training** by using a **certain ratio** of nodes to disconnect or not.

Unanswered Problems

- Great! Now we can train a **multi-layered neural network.**
- **We have seen that BP is an answer to the HOW question of training MLPs.**

Yet, an **MLP** introduces some more problems, relating to specifying its architecture

- How **wide** do we set the **hidden layers**?

Number of hidden layers

- How **deep** do we set the **hidden layers**?

Number of output nodes in each of the hidden layers

- To attempt answering these questions, a part of DL research is concerned about **Neural Architecture search (NAS).**
- Most recent methods, use an **evolutionary metaheuristic optimization** method as a major approach to do this by searching over some grid of randomized parameters.
- We will continue later with some **discussion of some problems associated with Learning in Deep Neural Network models.**

Assignments: Algorithm Implementation

TASKS

- **Library/Functions** (**API**) : **Upgrade the slp() to mlp()**.

1. Use it to train a model that identifies or learns the **XOR function**
2. Add backprop functionality
3. Add filtering/momentum functionality
4. Modify the GD rule to be cross-entropy driven.

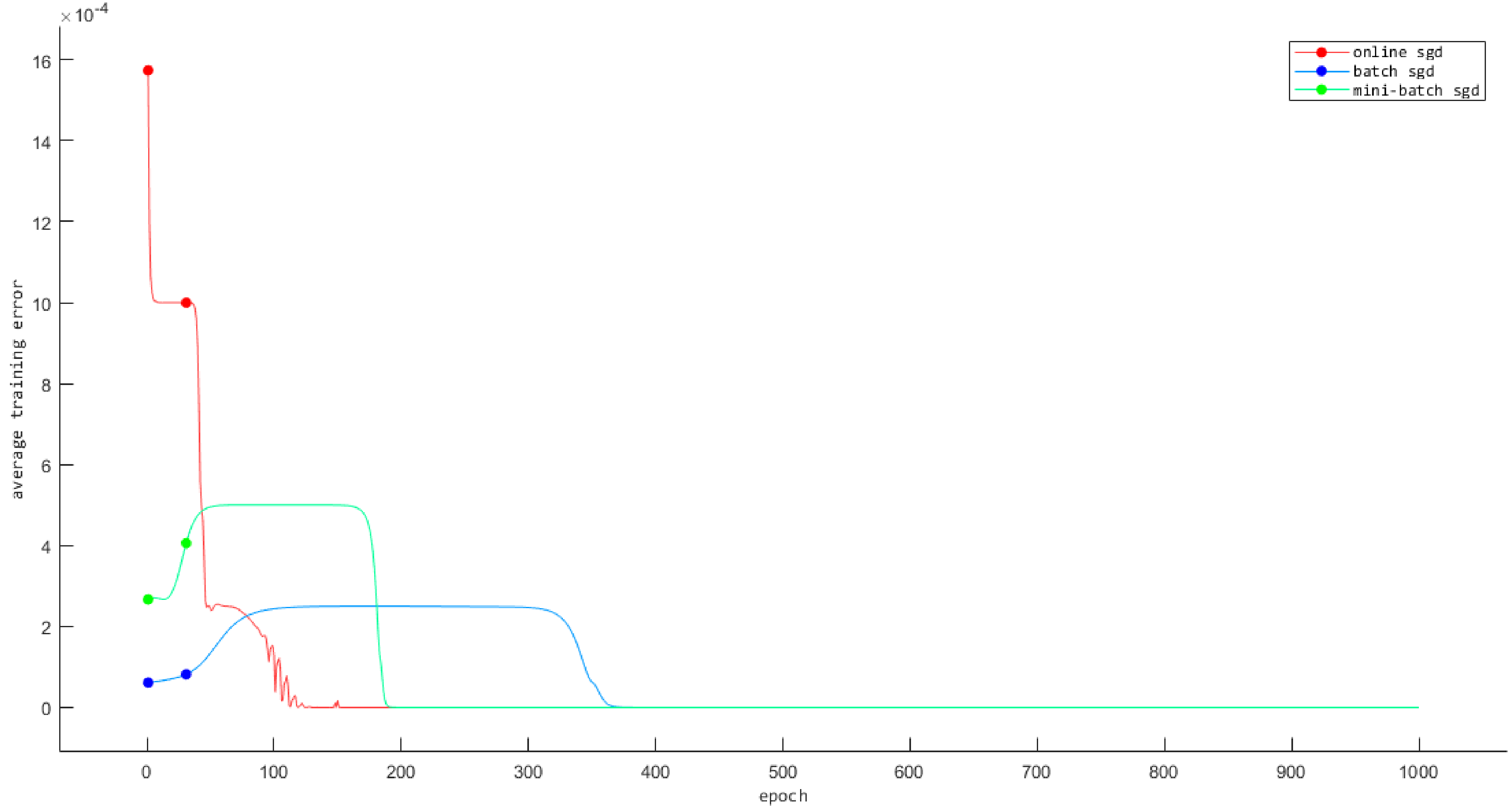
RESULTS:

Training Performance

1. Compare the Cost-Functions. **Comment.**
2. Compare Cost-Functions with or without regularization. **Comment.**
3. Compare the filtered and unfiltered SGD algorithm. **Comment.**

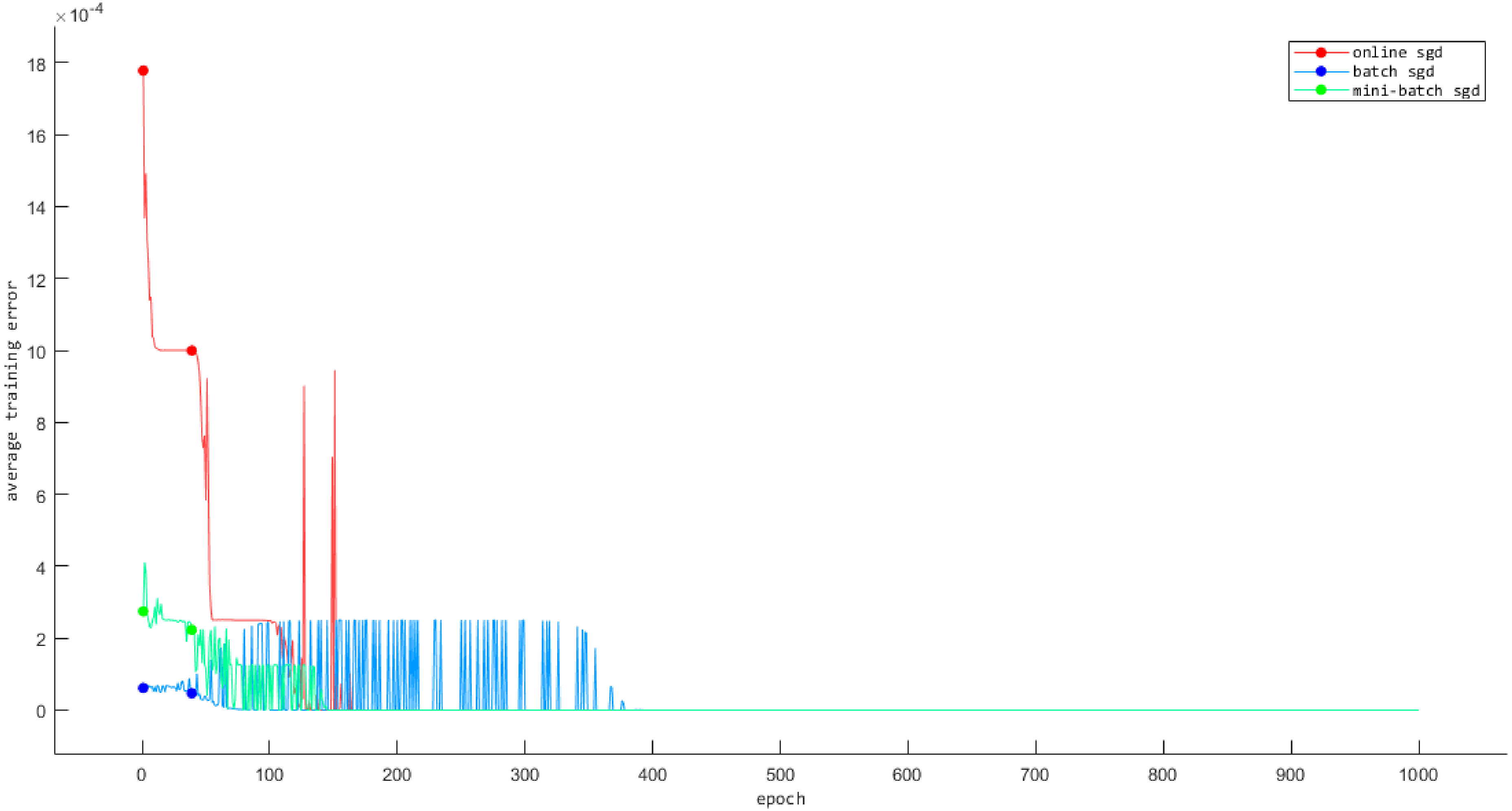
Assignments: Algorithm Implementation

VISUALIZER: TYPICAL GD



Assignments: Algorithm Implementation

VISUALIZER: STOCHASTIC GD



Summary

Recap:

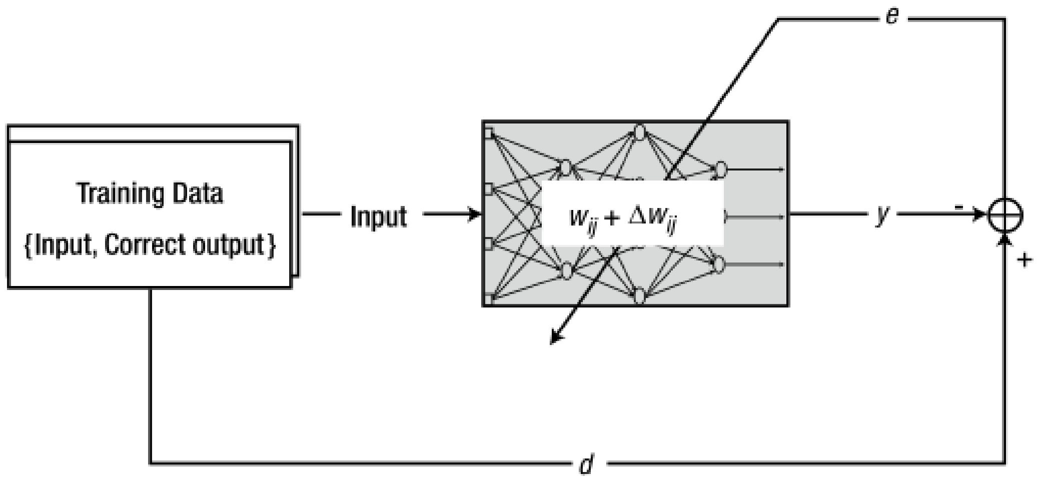
- **Most practical problems are linearly inseparable. The multi-layer neural network is capable of modeling the linearly inseparable problems.**
- **The hidden layers of the multi-layer neural network can be trained using the back-propagation algorithm. The representative learning rule of Deep Learning is Gradient Descent by Back Propagation.**
- **The back-propagation algorithm is important as it systematically defines the hidden layer error as it propagates the output error backward from the output layer.**
- **Many types of momentum-based GD rules for weight adjustment are available. The search for such methods is due to the pursuit of a more stable and faster learning of the network. These characteristics are particularly beneficial for hard-to-learn Deep Learning.**
- **The cost function addresses the output error of the neural network and is proportional to the error.**
- **The form and algorithmic implementation of the learning rule of the neural network varies depending on the cost function and activation function.**
- **Regularization is used to overcome overfitting**

Summary

Recap:

$$\Delta\theta = \left(J^T D\, J\right)^{-1} D J^T e$$

- **The above is the representative learning rule (algorithm) for iterative least-squares optimization.** *It can be found in many scientific and engineering areas where numerical optimization is carried out.* **Gradient Descent is a form of the above**.



Tools

Recommended Languages

- MATLAB (Fast Matrix Prototyping)
- JavaScript (Language of the Web)

Instructions to Student

- **All ANN functions will be written from scratch in form of custom libraries**
- **Learn to transfer maths to software.**
- **Copying of another person's code (or work) will be heavily penalized.**

Recommended Texts

Main Texts

- **MATLAB Deep Learning: With Machine Learning, Neural Networks and Artificial Intelligence** by Phil Kim
- **Pattern Recognition and Machine Learning** by Christopher M. Bishop
- **Understanding Machine Learning: From Theory to Algorithms** by Shai Shalev-Shwartz and Shai Ben-David
- **PATTERNS, PREDICTIONS, AND ACTIONS: A story about machine learning** by Moritz Hardt and Benjamin Recht
- **Mathematics for Machine Learning** by Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong

Good luck!