**IEOR LAB**

**Course Lab Report**

Semester 2 - 2025

**Instructors:**

Prof. K. S. Mallikarjuna Rao Prof. A. Mahajan

Department of Industrial Engineering & Operations Research

**Submitted by:**

Riya Aggarwal

Roll No: 24N0452

August 16, 2025

**Contents**

1. **Newton’s Method 1**
   1. What is Newton’s Method? . . . . . . . . . . . . . . . . . . . . . . . . . 1
   2. Mathematical Formulation / Approach . . . . . . . . . . . . . . . . . . . 1
   3. Use Cases . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
   4. Benefits . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
   5. Limitations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
   6. Python Code Snippet . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
2. **BFGS 3**
   1. What is BFGS? . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
   2. Mathematical Formulation / Approach . . . . . . . . . . . . . . . . . . . 3
   3. Use Cases . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
   4. Benefits . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
   5. Limitations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
   6. Python Code Snippet . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
3. **Regression 5**
   1. What is Regression? . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
   2. Mathematical Formulation / Approach . . . . . . . . . . . . . . . . . . . 5
   3. Use Cases . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
   4. Benefits . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
   5. Limitations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
   6. Python Code Snippet . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
4. **Classification 7**
   1. What is Classification? . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
   2. Mathematical Formulation / Approach . . . . . . . . . . . . . . . . . . . 7
   3. Use Cases . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
   4. Benefits . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
   5. Limitations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
   6. Python Code Snippet . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
5. **Dimensionality Reduction 9**
   1. What is Dimensionality Reduction? . . . . . . . . . . . . . . . . . . . . . 9
   2. Mathematical Formulation / Approach (PCA) . . . . . . . . . . . . . . . 9
   3. Use Cases . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
   4. Benefits . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
   5. Limitations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
   6. Python Code Snippet . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

i

1. **Feature Engineering 11**
   1. What is Feature Engineering? . . . . . . . . . . . . . . . . . . . . . . . . 11
   2. Methodology / Approach . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
   3. Use Cases . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
   4. Benefits . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
   5. Limitations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
   6. Python Code Snippet . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
2. **Outlier Detection 13**
   1. What is Outlier Detection? . . . . . . . . . . . . . . . . . . . . . . . . . . 13
   2. Mathematical Formulation / Approach . . . . . . . . . . . . . . . . . . . 13
   3. Use Cases . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
   4. Benefits . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
   5. Limitations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
   6. Python Code Snippet . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14
3. **Momentum Gradient Descent 15**
   1. What is Momentum? . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
   2. Mathematical Formulation / Approach . . . . . . . . . . . . . . . . . . . 15
   3. Use Cases . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
   4. Benefits . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
   5. Limitations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
   6. Python Code Snippet . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
4. **Nesterov Accelerated Gradient (NAG) 17**
   1. What is NAG? . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
   2. Mathematical Formulation / Approach . . . . . . . . . . . . . . . . . . . 17
   3. Use Cases . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
   4. Benefits . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
   5. Limitations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
   6. Python Code Snippet . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
5. **Adam Optimizer 19**
   1. What is Adam? . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
   2. Mathematical Formulation / Approach . . . . . . . . . . . . . . . . . . . 19
   3. Use Cases . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
   4. Benefits . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
   5. Limitations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
   6. Python Code Snippet . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20

ii

**Chapter 1**

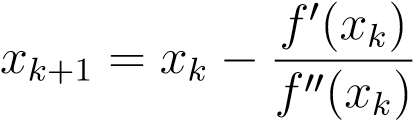
# Newton’s Method

## 1.1 What is Newton’s Method?

Newton’s Method (Newton–Raphson) is a classical second-order iterative algorithm used both for root-finding and unconstrained optimization. In optimization, it finds stationary points by using the gradient (first derivative) and Hessian (second derivative matrix) to form a local quadratic approximation of the objective and then stepping to the minimizer of that quadratic. When conditions are favorable, Newton’s method attains quadratic convergence near the optimum — meaning error squares each iteration — which makes it extremely powerful for high-accuracy solutions.

## 1.2 Mathematical Formulation / Approach

For a scalar function *f*(*x*):

*.*

For multivariate *f* : R*n* → R:

*xk*+1 = *xk* − *H*(*xk*)−1∇*f*(*xk*)*,*

where ∇*f*(*xk*) is the gradient and *H*(*xk*) = ∇2*f*(*xk*) is the Hessian. Practically, one often solves the linear system *H*(*xk*)∆*xk* = −∇*f*(*xk*) for ∆*xk* and uses a damped update *xk*+1 = *xk* + *αk*∆*xk* with a line search to guarantee decrease.

## 1.3 Use Cases

Newton’s Method is widely used in:

* Maximum likelihood estimation (solving score equations).
* Calibration and inverse problems in engineering.
* As an inner solver for constrained methods (e.g., barrier or trust-region methods).
* High-accuracy local optimization in scientific computing.

## 1.4 Benefits

* Quadratic convergence near a non-degenerate minimizer.
* Makes use of curvature — typically fewer iterations than first-order methods.
* Very accurate local solutions when Hessian is available or can be approximated.

## 1.5 Limitations

* Requires Hessian (or Hessian-vector products); costly or infeasible for very large *n*.
* Can diverge if started far from optimum or if Hessian is indefinite.
* Each iteration may be expensive (solving linear systems).

## 1.6 Python Code Snippet

|  |
| --- |
| import numpy as np from scipy.optimize import minimize  # Quadratic test: f(x) = (x0-2)^2 + (x1-3)^2 f = lambda x: (x[0]-2)\*\*2 + (x[1]-3)\*\*2 jac = lambda x: np.array([2\*(x[0]-2), 2\*(x[1]-3)]) hes = lambda x: np.array([[2.0, 0.0],[0.0, 2.0]])  res = minimize(f, x0=np.array([0.0,0.0]), method=’Newton-CG’, jac=jac, hess=hes) print("Newton solution:", res.x) |

**Chapter 2**

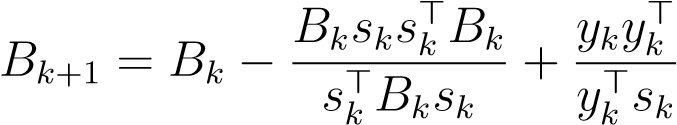
# BFGS

## 2.1 What is BFGS?

BFGS (Broyden–Fletcher–Goldfarb–Shanno) is a quasi-Newton optimization algorithm that builds an approximation of the Hessian (or its inverse) using gradient differences, avoiding direct Hessian computation. It maintains a symmetric positive-definite matrix that approximates curvature and updates it using low-rank corrections each iteration.

## 2.2 Mathematical Formulation / Approach

Given *sk* = *xk*+1 − *xk* and *yk* = ∇*f*(*xk*+1) − ∇*f*(*xk*), the BFGS update for the Hessian approximation *Bk* is:

*.*

Search direction) and step length *αk* chosen (Wolfe/Armijo). L-BFGS stores only recent (*s,y*) pairs for memory efficiency.

## 2.3 Use Cases

* Training generalized linear models with many features.
* Medium-scale smooth optimization (e.g., calibration, parameter estimation).
* Situations where Hessian is expensive but gradient evaluations are feasible.

## 2.4 Benefits

* Superlinear convergence under standard assumptions.
* Avoids explicit Hessian — lower per-iteration cost.
* Robust and widely used in practice.

## 2.5 Limitations

* Still needs line search; sensitive to noisy gradients.
* Not ideal for non-smooth objectives.
* For very large problems, use L-BFGS for memory reasons.

## 2.6 Python Code Snippet

|  |
| --- |
| from scipy.optimize import minimize import numpy as np  f = lambda x: (x[0]-2)\*\*2 + (x[1]+3)\*\*2 + 0.1\*np.sin(3\*x[0]) res = minimize(f, x0=[0.0, 0.0], method=’BFGS’) print("BFGS solution:", res.x) |

**Chapter 3**

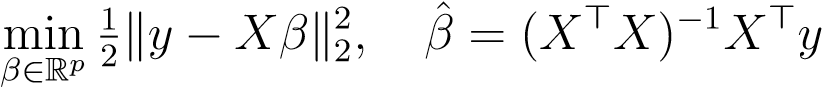
# Regression

## 3.1 What is Regression?

Regression is a family of supervised learning techniques to model and predict a continuous response variable *y* from predictors *X*. Linear regression is the simplest form, assuming *y* depends linearly on features. Regularized variants (Ridge, Lasso) add penalties to control complexity and avoid overfitting. Generalized linear models extend regression to non-Gaussian responses.

## 3.2 Mathematical Formulation / Approach

Ordinary Least Squares (OLS):

 (if invertible)*.*

Ridge (Tikhonov) adds; Lasso adds *λ*∥*β*∥1 to encourage sparsity. Model selection and validation rely on cross-validation, RMSE, and residual diagnostics.

## 3.3 Use Cases

* Sales and demand forecasting.
* House price prediction.
* Estimating effects in econometric studies.

## 3.4 Benefits

* Interpretable coefficients.
* Fast to train and conceptually simple.
* Strong statistical foundation and inference tools.

## 3.5 Limitations

* Captures only linear effects unless features are engineered.
* Sensitive to outliers and multicollinearity.

## 3.6 Python Code Snippet

|  |
| --- |
| from sklearn.model\_selection import train\_test\_split from sklearn.linear\_model import LinearRegression from sklearn.metrics import mean\_squared\_error, r2\_score import numpy as np  # Synthetic linear data with noise rng = np.random.default\_rng(0) X = rng.uniform(0, 10, size=(200, 1))  y = 3.0 \* X[:,0] + 5.0 + rng.normal(0, 1.0, size=200)  X\_tr, X\_te, y\_tr, y\_te = train\_test\_split(X, y, test\_size=0.25, random\_state=42)  model = LinearRegression().fit(X\_tr, y\_tr) y\_pred = model.predict(X\_te) print("coef, intercept:", model.coef\_, model.intercept\_) print("RMSE, R2:", mean\_squared\_error(y\_te, y\_pred, squared=False), r2\_score(y\_te, y\_pred)) |

**Chapter 4**

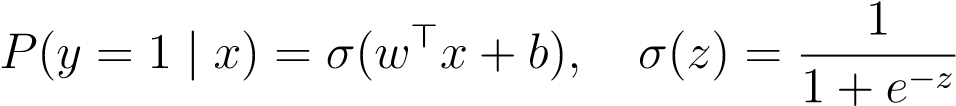
# Classification

## 4.1 What is Classification?

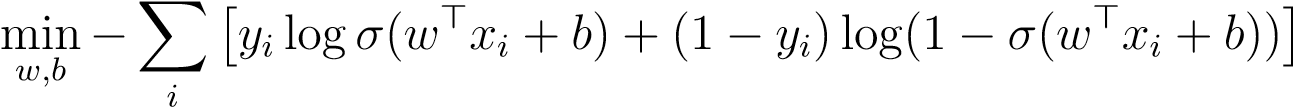
Classification assigns discrete labels to inputs. Models can be probabilistic (logistic regression), margin-based (SVM), tree-based, or deep neural networks. Evaluation uses accuracy, precision, recall, F1-score, ROC-AUC, and confusion matrices.

## 4.2 Mathematical Formulation / Approach

Binary logistic regression:

*.*

Parameters are fit by minimizing cross-entropy:

 *.*

## 4.3 Use Cases

• Spam detection, medical diagnosis, churn prediction, image classification.

## 4.4 Benefits

* Probabilistic outputs helpful for decision thresholds.
* Many scalable and well-understood algorithms.

## 4.5 Limitations

* Class imbalance needs careful treatment (resampling, class weights).
* Linear models may be insufficient for complex boundaries.

## 4.6 Python Code Snippet

|  |
| --- |
| from sklearn.datasets import load\_iris from sklearn.linear\_model import LogisticRegression from sklearn.preprocessing import StandardScaler from sklearn.pipeline import Pipeline from sklearn.model\_selection import train\_test\_split from sklearn.metrics import classification\_report  X, y = load\_iris(return\_X\_y=True)  X\_tr, X\_te, y\_tr, y\_te = train\_test\_split(X, y, test\_size=0.25, random\_state=0)  pipe = Pipeline([  ("scaler", StandardScaler()),  ("clf", LogisticRegression(max\_iter=1000, multi\_class=’auto’))  ]) pipe.fit(X\_tr, y\_tr) print(classification\_report(y\_te, pipe.predict(X\_te))) |

**Chapter 5**

# Dimensionality Reduction

## 5.1 What is Dimensionality Reduction?

Dimensionality reduction maps high-dimensional data to a lower-dimensional representation while trying to preserve important structure. It reduces storage, speeds up training, aids visualization, and can reduce overfitting.

## 5.2 Mathematical Formulation / Approach (PCA)

PCA centers data *Xc* and computes covariance. PCA solves:

*Swi* = *λiwi,*

and projects data as *Z* = *XcWk* using the top *k* eigenvectors *Wk*. SVD is an efficient alternative: *Xc* = *U*Σ*V* ⊤, principal directions are columns of *V* .

## 5.3 Use Cases

* Preprocessing for ML pipelines (reduce features).
* Visualization (2D/3D embedding).
* Noise reduction and compression.

## 5.4 Benefits

* Orthogonal components ordered by explained variance.
* Reduces correlation among features.

## 5.5 Limitations

* Linear; may not capture complex manifolds (use t-SNE/UMAP).
* Components can lack interpretability.

## 5.6 Python Code Snippet

|  |
| --- |
| from sklearn.decomposition import PCA from sklearn.datasets import load\_digits from sklearn.model\_selection import train\_test\_split from sklearn.linear\_model import LogisticRegression from sklearn.pipeline import Pipeline  X, y = load\_digits(return\_X\_y=True)  X\_tr, X\_te, y\_tr, y\_te = train\_test\_split(X, y, test\_size=0.25, random\_state=0)  pipe = Pipeline([  ("pca", PCA(n\_components=25, whiten=True)),  ("clf", LogisticRegression(max\_iter=2000))  ]) pipe.fit(X\_tr, y\_tr) print("Accuracy with PCA:", pipe.score(X\_te, y\_te)) |

**Chapter 6**

# Feature Engineering

## 6.1 What is Feature Engineering?

Feature engineering is the process of transforming raw data into features that improve the predictive performance of models. It includes scaling, encoding, constructing interaction terms, extracting domain-specific features, and feature selection.

## 6.2 Methodology / Approach

Common transforms:

* Scaling: standardization (*x* − *µ*)*/σ* or min-max.
* Categorical encoding: one-hot, ordinal, target-encoding, WOE.
* Polynomial and interaction features to capture nonlinearities.
* Time-based features: lags, rolling statistics.

Always fit transformers on training data and apply to test/validation to avoid leakage.

## 6.3 Use Cases

* Credit scoring (WOE/IV).
* Text (TF–IDF, embeddings).
* Time-series (lags, windows).

## 6.4 Benefits

* Can dramatically improve model performance.
* Enables simpler models to capture complex relationships.

## 6.5 Limitations

* Extra engineering effort; risk of overfitting with many features.
* Danger of data leakage if applied incorrectly.

## 6.6 Python Code Snippet

|  |
| --- |
| from sklearn.preprocessing import StandardScaler, OneHotEncoder,  PolynomialFeatures from sklearn.compose import ColumnTransformer from sklearn.pipeline import Pipeline from sklearn.linear\_model import Ridge import numpy as np  # Mixed numeric + categorical toy data  X\_num = np.array([[1.0],[2.0],[3.0],[4.0]])  X\_cat = np.array([[’A’],[’B’],[’A’],[’B’]]) X = np.hstack([X\_num, X\_cat]) y = np.array([3.0, 5.0, 7.0, 9.0])  pre = ColumnTransformer([ ("num", Pipeline([  ("scaler", StandardScaler()),  ("poly", PolynomialFeatures(degree=2, include\_bias=False)) ]), [0]),  ("cat", OneHotEncoder(handle\_unknown=’ignore’), [1])  ])  pipe = Pipeline([("pre", pre), ("model", Ridge(alpha=1.0))]) pipe.fit(X, y) print("Pred:", pipe.predict(X)) |

**Chapter 7**

# Outlier Detection

## 7.1 What is Outlier Detection?

Outlier (anomaly) detection identifies observations that deviate significantly from the bulk of the data. Outliers may indicate fraud, sensor faults, or rare events. Techniques range from simple statistical rules to sophisticated machine learning models.

## 7.2 Mathematical Formulation / Approach

* **Statistical**: Z-score *zi* = (*xi* − *µ*)*/σ*, IQR rules.
* **Robust estimators**: median, MAD.
* **Model-based**: One-Class SVM, Isolation Forest, Local Outlier Factor (LOF).

## 7.3 Use Cases

* Fraud detection in finance.
* Equipment failure monitoring.
* Data cleaning before modelling.

## 7.4 Benefits

* Improves model robustness by removing or flagging anomalies.
* Can surface rare but important events.

## 7.5 Limitations

* Hard to evaluate without labels.
* Choice of hyperparameters (contamination) matters.

### 7.6

|  |
| --- |
| from sklearn.ensemble import IsolationForest import numpy as np  rng = np.random.default\_rng(0)  X\_normal = rng.normal(0, 1, size=(300, 2)) X\_anom = rng.uniform(6, 8, size=(5, 2)) X = np.vstack([X\_normal, X\_anom])  iso = IsolationForest(random\_state=0, contamination=0.02).fit(X) labels = iso.predict(X) # -1 = anomaly, 1 = normal print("Anomaly count:", (labels==-1).sum()) |

**Chapter 8**

# Momentum Gradient Descent

## 8.1 What is Momentum?

Momentum is an optimization technique that accumulates an exponentially decaying moving average of past gradients to accelerate convergence, especially in directions with small but consistent gradients, and to damp oscillations across steep directions.

## 8.2 Mathematical Formulation / Approach

With learning rate *η* and momentum *β* ∈ [0*,*1):

*vt* = *βvt*−1 + (1 − *β*)∇*f*(*xt*)*, xt*+1 = *xt* − *ηvt.*

In many implementations the update is written *vt* = *βvt*−1 + ∇*f*(*xt*) and then *xt*+1 = *xt* − *ηvt*.

## 8.3 Use Cases

Common in deep learning to speed up SGD on ill-conditioned loss surfaces (CNNs, RNNs).

## 8.4 Benefits

* Faster convergence than plain SGD.
* Reduces oscillations in narrow valleys.

## 8.5 Limitations

* Introduces an extra hyperparameter *β*.
* Can overshoot if learning rate is too large.

### 8.6

|  |
| --- |
| import torch  x = torch.tensor([5.0], requires\_grad=True) opt = torch.optim.SGD([x], lr=0.1, momentum=0.9) for \_ in range(100):  opt.zero\_grad() loss = (x-3)\*\*2 # convex bowl loss.backward() opt.step()  print("Momentum result:", x.item()) |

**Chapter 9**

# Nesterov Accelerated Gradient (NAG)

## 9.1 What is NAG?

Nesterov Accelerated Gradient improves classical momentum by computing the gradient at a look-ahead point: it anticipates the next position and uses that to form a corrective term, leading to improved responsiveness and reduced overshoot.

## 9.2 Mathematical Formulation / Approach

The typical NAG update:

*vt* = *βvt*−1 + ∇*f*(*xt* − *βvt*−1)*, xt*+1 = *xt* − *ηvt.*

This look-ahead gradient often yields faster practical convergence than classical momentum.

## 9.3 Use Cases

Training large-scale deep neural networks where anticipatory updates help avoid wasted steps.

## 9.4 Benefits

* Better corrective behavior compared to classical momentum.
* Often improves convergence speed and stability.

## 9.5 Limitations

• Sensitive to hyperparameters and batch settings.

### 9.6

import torch

x = torch.tensor([2.0], requires\_grad=True) opt = torch.optim.SGD([x], lr=0.1, momentum=0.9, nesterov=True) for \_ in range(100): opt.zero\_grad() loss = (x-1.0)\*\*2 loss.backward() opt.step() print("NAG result:", x.item())

**Chapter 10**

# Adam Optimizer

## 10.1 What is Adam?

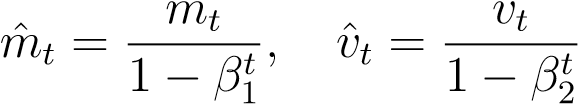
Adam (Adaptive Moment Estimation) is an optimizer combining ideas from momentum (first moment) and adaptive learning rates (second moment, like RMSProp). It maintains exponentially decaying averages of past gradients and squared gradients, applies bias correction, and adapts per-parameter learning rates. Adam is widely used in deep learning due to its robustness and low tuning requirements.

## 10.2 Mathematical Formulation / Approach

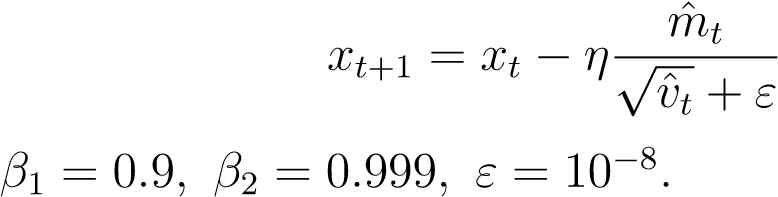
Given gradient *gt* = ∇*f*(*xt*):

*.*

Bias-corrected:

*.*

Update:

*.*

Typical defaults:

## 10.3 Use Cases

Training transformers, CNNs, RNNs, and models with sparse gradients (embeddings).

## 10.4 Benefits

* Fast convergence, works well with default hyperparameters.
* Handles sparse and noisy gradients effectively.
* Per-parameter adaptive learning rates.

## 10.5 Limitations

* Sometimes leads to worse generalization than SGD with momentum.
* Requires careful use of weight decay (AdamW recommended).
* Learning rate schedules / warmup often needed for large models.

## 10.6 Python Code Snippet

import torch

x = torch.tensor([0.0], requires\_grad=True) opt = torch.optim.Adam([x], lr=0.1) for \_ in range(200): opt.zero\_grad() loss = (x-7.0)\*\*2 loss.backward() opt.step() print("Adam result:", x.item())