Week 1 Notes - ML

Start Date	@27 October 2025

These are notes that I've taken whilst reading the notes from Chapter 1 and 2 of the Machine Learning with PyTorch & Scikit-Learn book by Sebastian Raschka and others. The notes are roughly the same as the hand-written notes, however sometimes there are minor changes.

Types of Machine Learning

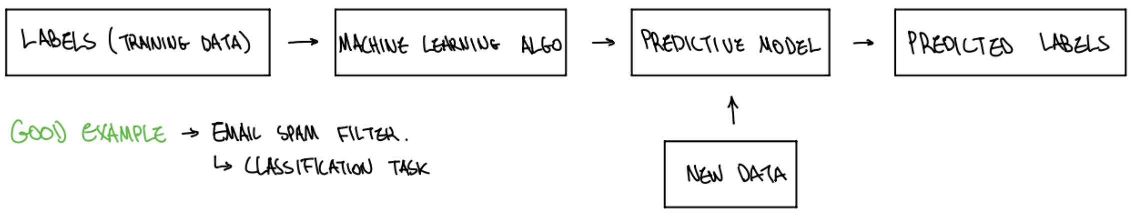
- **Supervised Learning** → Labelled data, direct feedback, predict outcome/future
- **Unsupervised Learning** → Unlabelled data, no feedback, find hidden structure in data.
- **Reinforcement Learning** → Decision process, reward system, learn series of actions

Supervised Learning

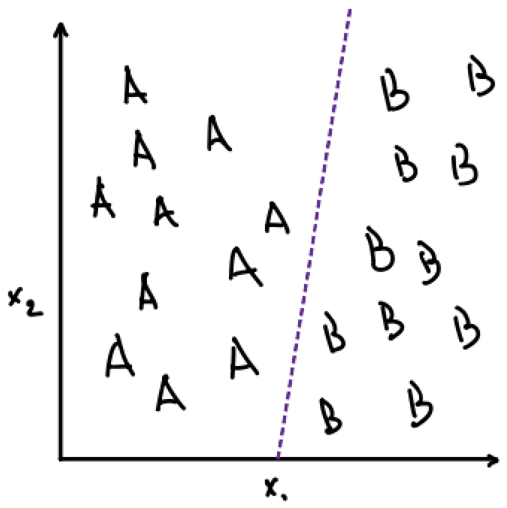
📌 Model learns from labelled training data → Makes future predictions → aka. "Labelled Learning"

Good example → Email spam filter (Classification task)

Classification



- Sub-category of supervised learning
- **Aim** → Predict categorial class labels of new instances, based on past examples



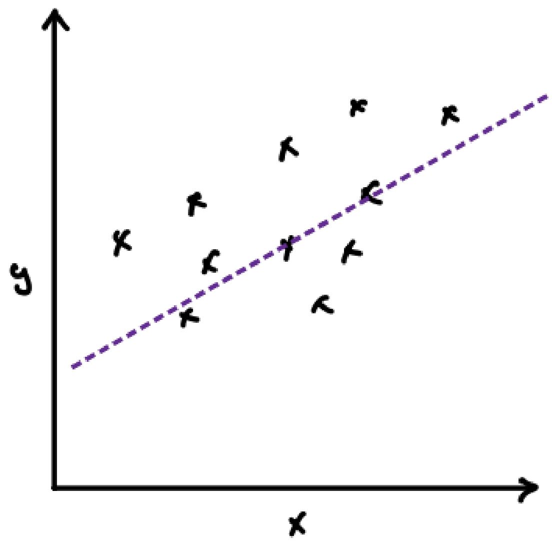
Classification (Class A or Class B)

- Email spam filter → **Binary Classification** task, is the email a spam or not a spam email
- Supervised learning trained model → assign any class label to new data
  - **Multiclass Classification**
    - Handwritten character recognition → “A”, “B”, “C”
    - However, this model wouldn’t recognise numbers for example, because it was only trained on letters

Regression

⭐ Variables are called **features, target variables**

- Finding the relationship between variables to predict an outcome.

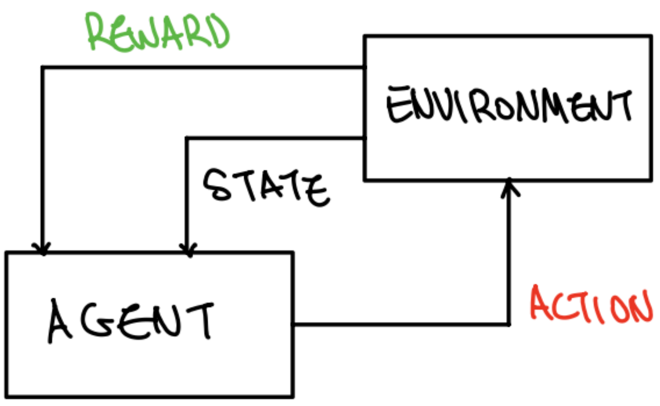


Regression model

- E.g. → Finding correlation between time spent studying vs. exam results (training dataset) → predict future scores of students depending on the time they spent studying

Reinforcement Learning

- Develop agent → Improves performance based on interaction with environment
- Trial-and-error → machine learns with **rewards** system → chooses a series of actions to maximise rewards
  - Feedback is immediate or delayed feedback
- E.g. → Chess program → Win or lose with different moves → understands whilst playing the player, improves system is it goes on.



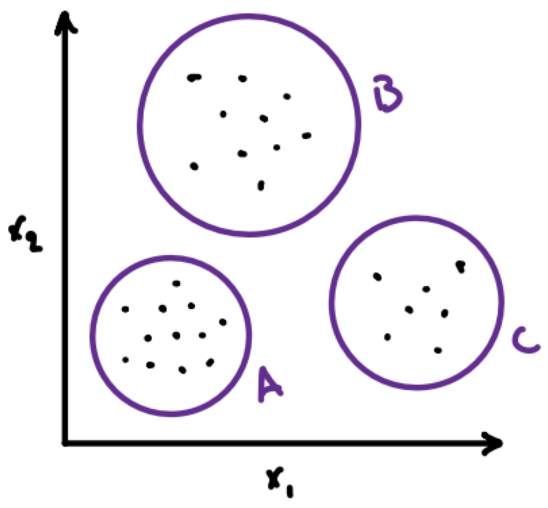
Reinforcement Learning

Unsupervised Learning

- Model trains with unlabelled data
- Explore structure of data to extract data without guidance from known outcome variables

Clustering

- Organise information into subgroups (clusters) → without prior information about group
- Sometimes called **Unsupervised Classification**
- Good technique for structuring information + finding relationships from data



Clustering

Dimensional Reduction

- Often we work with high dimensionality (larger number of features)
- Compresses data onto smaller dimensional subspace while retaining most of the relevant information
- **Commonly used** → feature reprocessing to remove noise from data
- Can be useful for data visualisation (sometimes)

Terminology

- **Training example** → Row in a table representing dataset used to teach a model how to make predictions

- **Training** → Model fitting
- **Feature (x)** → Input/variable
- **Target (y)** → Output/outcome
- **Loss Function**
    - Cost function/error function
    - **"Loss"** → Loss measured for single data point
    - **"Cost"** → Compute loss over the entire dataset (average or summed)

Machine Learning Systems

Preprocessing

⭐ **One of the most crucial steps** → shape data for learning algorithms

- Select features → should be the same scale
- Dimensional Reduction → Highly correlated features, therefore redundant to certain degree
    - **Advantage** → Less storage space required + algorithm can run faster
- Divide dataset to test new data against trained data

⭐ 80:20 → 80 test, 20 validation (testing)

Training, Selecting and Evaluating Models

- One model isn't necessarily good for all applications
- Essential to train multiple models and train them to evaluate which works best
- Common metric to measure performance → **Classification accuracy**
- "How do we know which model performs well on final data?" → **Cross-validation**
- Cross-validation → Further divide dataset into training + validation subset
- Frequent use → Hyperparameter fine-tuning
- **Generalization Error** → How accurately will the training model perform with test dataset (predict outcome)

- Same procedures applied to training dataset applied to test dataset → otherwise model will **overfit**

💡 **Overfitting** → Machine learning model learns from the training data too well (includes noise), resulting in poor performance on new, unseen data.

Classification models

- **Perceptron**
- **Adaptive Linear Neurons (Adaline)**

Artificial Neurons

- First computational model of a neuron → **McCulloh-Pitts** (1943)
  - Concept made by Warren McCulloh + Walter Pitts
- First concept of perceptron learning rule based on MCP → Frank Rosenblatt (1957)
  - Proposed an algorithm that learns optimal weight → multiplied with input features → order to make decision if neuron fires or doesn't
  - **Current Day** → In Regression + Classification
    - This algorithm could be used to predict whether a new data point belongs to one class or other
- **Binary Classification** → whether data is 0 or 1

$$w = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix} (1) \quad x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} (2) \quad \sigma(z) = \begin{cases} 1 & \text{if } z \geq \theta \\ 0 & \text{otherwise} \end{cases}$$

Linear Algebra

- Abbreviate the sum of products of value `x` and `w` → vector dot product

$$a = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} (3) \quad \text{Transpose } a \text{ to } a^T \quad a^T = \begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix}$$

$$\mathbf{a}^T\mathbf{b} = \sum_{i=1} a_i b_i = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

- Example:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

Perceptron Learning Rule

- Reductionist approach to mimic how single neutron works in the brain.
- Algorithm summarised:
  1. Init weight + bias → until 0 or small number is reached
  2. For each training example $x^{(i)}$:
     a. Compute output $\hat{y}^{(i)}$
     b. Update weight + bias unit
- Output is class label predicted by unit step function

$$w_j := w_j + \Delta w_j \quad b := b + \Delta b$$

$$\Delta w_j = \eta \left(y^{(i)} - \hat{y}^{(i)} \right) x_j^{(i)} \quad \Delta b = \eta \left(y^{(i)} - \hat{y}^{(i)} \right)$$

$$w_j = x_j \text{ in dataset}$$
$$\eta = \text{Learning rate (Typically constant between 0.0 and 1.0}$$
$$y^{(i)} = \text{True class label of } i\text{th traning example}$$
$$\hat{y}^{(i)} = \text{Predict class label}$$

- Bias unit + weight → constantly updated simultaneously

$$\Delta w_1 = \eta \left(y^{(i)} - \text{Output}^{(i)} \right) x_1^{(i)}$$

$$\Delta w_2 = \eta \left(y^{(i)} - \text{Output}^{(i)} \right) x_2^{(i)}$$

$$\Delta b = \eta \left(y^{(i)} - \text{Output}^{(i)} \right)$$

- **Perceptron predicts correctly** → Bias unit + weight remain unchanged → value are 0:

$$y^{(i)} = 0, \quad \hat{y}^{(i)} = 0, \quad \Delta w_j = \eta(0-0)x_j^{(i)} = 0, \quad \Delta b = \eta(0-0) = 0$$

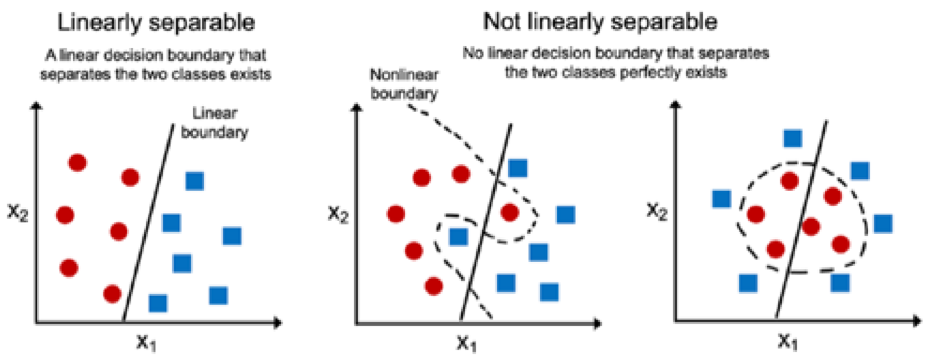
$$y^{(i)} = 1, \quad \hat{y}^{(i)} = 0, \quad \Delta w_j = \eta(1-1)x_j^{(i)} = 0, \quad \Delta b = \eta(1-1) = 0$$

- **Wrong prediction** → Leans positive or negative target class:

$$y^{(i)} = 0, \quad \hat{y}^{(i)} = 0, \quad \Delta w_j = \eta(1-0)x_j^{(i)} = \eta x_j^{(i)}, \quad \Delta b = \eta(0-0) = \eta$$

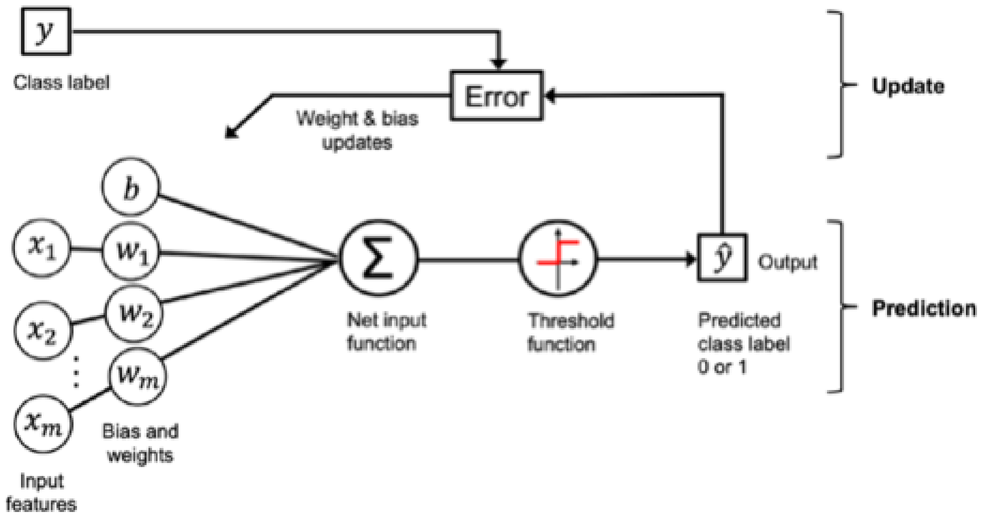
$$y^{(i)} = 1, \quad \hat{y}^{(i)} = 0, \quad \Delta w_j = \eta(1-1)x_j^{(i)} = -\eta x_j^{(i)}, \quad \Delta b = \eta(1-1) = -\eta$$

- Convergence of perceptron only guaranteed → two classes are linearly seperable


Linearly separable
A linear decision boundary that separates the two classes exists
Linear boundary
X₂
X₁
Not linearly separable
No linear decision boundary that separates the two classes perfectly exists
Nonlinear boundary
X₂
X₁
X₂
X₁

- If two classes can't be separated by linear boundary → set maximum number of passes over training dataset (epoch)

General Concept of Perceptron



- **Input examples** (x) **+ bias** (b) **+ weights** (w) ⇒ compute net input
  - Helps shift decision boundary
- **Threshold Function** → Checks if sum x crosses certain threshold

$$\text{Decides Otuput} = \begin{cases} 1 & \text{if } z \geq \theta \\ 0 & \text{otherwise} \end{cases}$$

- **Learning Phase**
  - When prediction is wrong → adjusts weights to get closer to correct output
  - Continues until perceptron classifies all training examples correctly

Perceptron on Iris Dataset

- Only use two flower classes → Perceptron is a binary classifier
- Perceptron algorithm can be extended → One-versus-All (OvA) technique
- **One-versus-All (OvA)** → Mutli-class Classifier
  - Sometimes called One-versus-Rest (OvR)
  - Extend Binary Classifier → multi-class problem
  - Train one **classifier per class** → treated as positive + examples from all other classes are negative
  - New unlabelled data → use n classifiers, n = Number of class labels → assign to highest confidence to particular instance

    - In case of perceptron → OvA → Choose class label that is associated with largest absolute net input value

Perceptron Convergence

💡 Convergence → Biggest problem for the perceptron

- If classes cannot be separated → weights will never stop updating unless we set maximum number of epochs

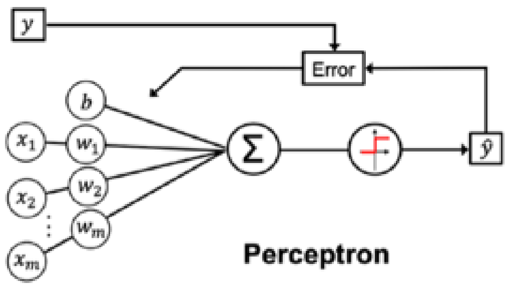
Adaptive Linear Neurons (Adaline)

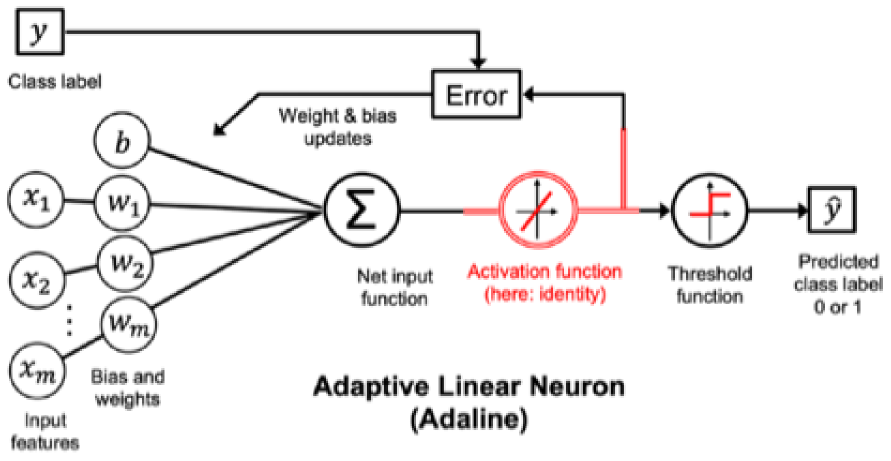
- Single-layer Neural Network → Adaptive Linear Neurons (Adaline) 1960
    - Bernard Wifrow + Tedd Hoff

    💡 Key concepts of defining and minimising continuous loss function

    - Lay the ground works for udnerstanding other machine learning algorithms for classification. Examples:
        - Logistic Regression
        - Support Vector Machines
        - Multi-layer Neural Networks
        - Also → Linear Regression Model
    - **Key difference with Rosenblatt's perceptron** → Weights updated based on linear activation function vs. unit step function in Rosenblatt's perceptron
    - Linear activation function = $\sigma(z)$ → Identity function of net input $\sigma(z) = z$

Main Concept of Adaline





- **Input examples (xx) + bias (bb) + weights (ww) →→ compute net input**
  - Sums up weighted inputs and bias for every training example
- **Linear Output →→ Direct continuous value (no threshold yet)**
  - Output is not just "yes/no," but any real number
- **Learning Phase**
  - Calculates error: how far the output is from the correct value
  - Uses gradient descent/delta rule to adjust weights and bias to minimise this error
- **Final Decision (Threshold)**
  - After training, applies a threshold function to convert output to class:

$$\text{Decides Otuput} = \begin{cases} 1 & \text{if } z \geq \theta \\ 0 & \text{otherwise} \end{cases}$$

Minimising Loss Function with Gradient Descent

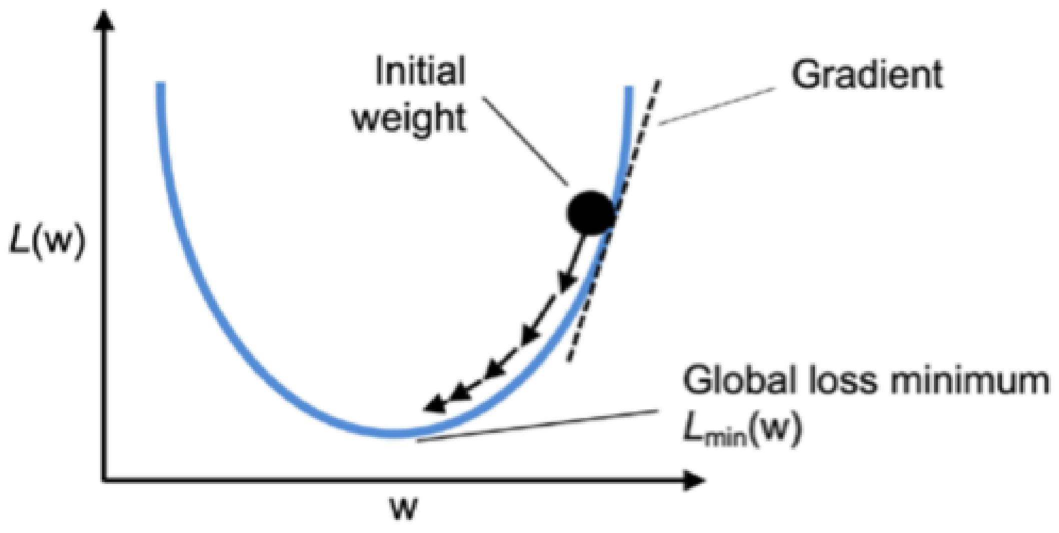
- Key ingredient for supervised learning → Objective Function

- **Objective Function** → Loss or cost function that we want to minimise
- Adaline → Loss function (L) → Learn model parameters as Mean Square Error (MSE) → between calculated + true class label

$$L(w,b) = \frac{1}{2n}\sum_{i=1}^{n}\left(y^{(i)} - \sigma\left(z^{(i)}\right)\right)^2$$

- $\frac{1}{2}$ for our own convenience → easier to derive the gradient of loss function
- Advantage:
    - Loss function becomes differentiable
    - It's convex → Can use Gradient Descent → Find weights that maximise loss function to classify examples in a dataset

Gradient Descent



- "Climbing down hill" → Until loss or global loss minimum is reached
- Each iteration → step opposite direction of gradient
    - Step determined by learning rate + slope of gradient
- Opposite direction of $\nabla L(w,b)$: $\quad w : w + \Delta w, \quad b : b + \Delta b$
- $\Delta w, \Delta b$ defined as negative gradient multiplied by learning rate η

$$\Delta w = -\eta \nabla_w L(w,b), \quad \Delta b = -\eta \nabla_b L(w,b)$$

- Gradient of loss function → partial derivative of loss function → respect to each weight, w_j

$$\frac{\partial L}{\partial w_j} = -\frac{2}{n}\sum_i \left(y^{(i)} - \sigma\left(z^{(i)}\right)\right) x_j^{(i)}$$

- Partial derivative of loss respected to bias

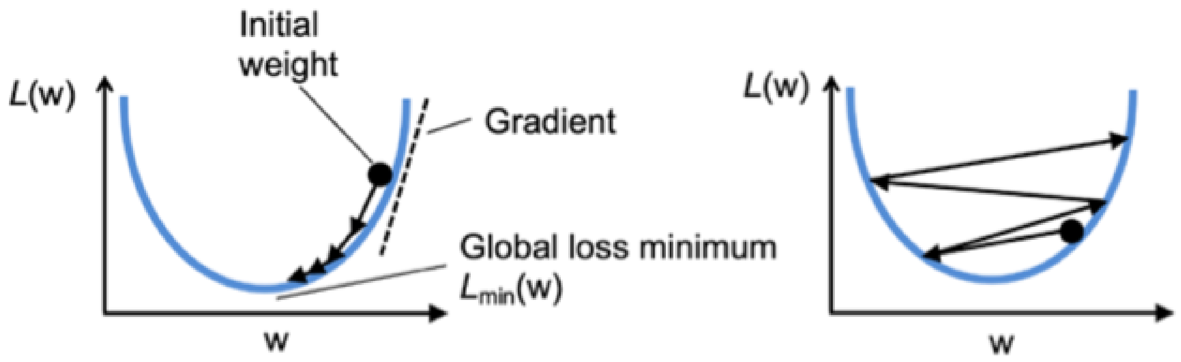
$$\frac{\partial L}{\partial w_j} = \frac{\partial}{\partial w_j}\frac{1}{n}\sum_i \left(y^{(i)} - \sigma(z^{(i)})\right)^2 = \frac{1}{n}\frac{\partial}{\partial w_j}\sum_i \left(y^{(i)} - \sigma(z^{(i)})\right)^2$$

$$= \frac{2}{n}\sum_i \left(y^{(i)} - \sigma(z^{(i)})\right)\frac{\partial}{\partial w_j}\left(y^{(i)} - \sigma(z^{(i)})\right)$$

$$= \frac{2}{n}\sum_i \left(y^{(i)} - \sigma(z^{(i)})\right)\frac{\partial}{\partial w_j}\left(y^{(i)} - \sum_j (w_j x_j^{(i)}) + b\right)$$

$$= \frac{2}{n}\sum_i \left(y^{(i)} - \sigma(z^{(i)})\right)\frac{\partial}{\partial w_j}\left(y^{(i)} - \sum_j (w_j x_j^{(i)}) + b\right)$$

$$= \frac{2}{n}\sum_i \left(y^{(i)} - \sigma(z^{(i)})\right)(-x_j^{(i)}) = -\frac{2}{n}\sum_i \left(y^{(i)} - \sigma(z^{(i)})\right) x_j^{(i)}$$



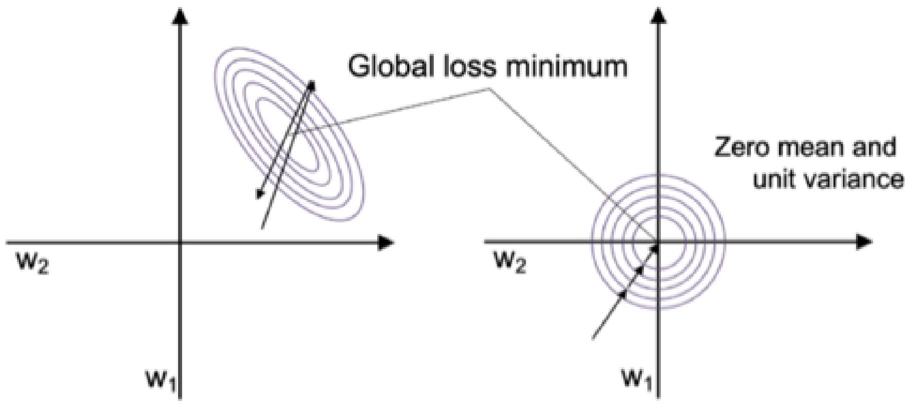
- Left Image → well-chosen learning rate (hyperparameters) where loss decreases gradually
- Right Image → Overshot global minimum → Learning rate is too large

Standardisation

- Normalisation that helps gradient descent learning to converge more quickly. Dataset not normally distributed
  - Shifts the mean of each feature → centred at zero + each features standard deviation = 1

$$x'_j = \frac{x_j - \mu_j}{\sigma_j}$$

- Helps gradient descent learning → Easier to find learning rate that works well for weights + bias
- Great for features with different scales → stabilises training → optimiser can go through fewer steps for optimal solution



- Left unscaled features
- Right standardised features

Stochastic Gradient Descent (SGD)

- Aka. Iterative or online gradient descent
- Parameters are updated incrementally for each training example over sum of accumulated errors over all training examples

$$\Delta w_j = \eta(y^{(i)} - \sigma(z^{(i)}))x_j^{(i)}, \quad \Delta b = \eta(y^{(i)} - \sigma(z^{(i)}))$$

- SGD considered an approximation to gradient descent → typically reaches convergence faster → more frequent weight updates
- Error surface in noisier than gradient descent → advantage for SGD → escape shallow minima more readily → nonlinear loss function
- SGD → present training data in random order → shuffle data every epoch → prevents cycles
- SGD - adaptive learning rate → decreases over time

$$\frac{c_1}{\text{number of iterations} + c_2}$$

- Advantage of SGD → Can do Online Learning
    - Model trained with new data on the fly
    - Useful for large amounts of data
    - System can immediately adapt to changes

Mini-batch Gradient Descent

- Full batch gradient descent to smaller subsets of training data
- Advantage over full batch → Convergence reached faster (finds optimal weights and biases faster)