PRACTICAL MACHINE LEARNING: OPTIMISATION & NON-LINEAR REGRESSION

Dr. Alexander Booth (he / him)
Universidad Católica del Norte, Chile
October 8th, 2024

Queen Mary
University of London

Timely News!

BBC

Home News US Election Sport Business Innovation Culture Arts Travel Earth Video Live

'Godfather of AI' shares Nobel Physics Prize

3 hours ago

Share Save

Georgina Rannard
Science reporter

Graham Fraser
Technology reporter

Getty Images

Recap

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{y_i - (mx_i + c)}{\sigma_i} \right)^2$$

- Obtaining the best values of m (gradient) and c (offset) to define a line describing the data.

- Defined a metric, χ^2, to quantify "how good" a particular choice of m and c was.

- Used a gradient descent algorithm to systematically vary m and c to minimise χ^2.

Recap

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{y_i - (mx_i + c)}{\sigma_i} \right)^2$$

- Obtaining the best values of m (gradient) and c (~~offset~~) ~~to define a~~ line describing the data.

Model parameters / weights

- Defined a metric, χ^2, to quantify "how good" a particular choice of m and c was.

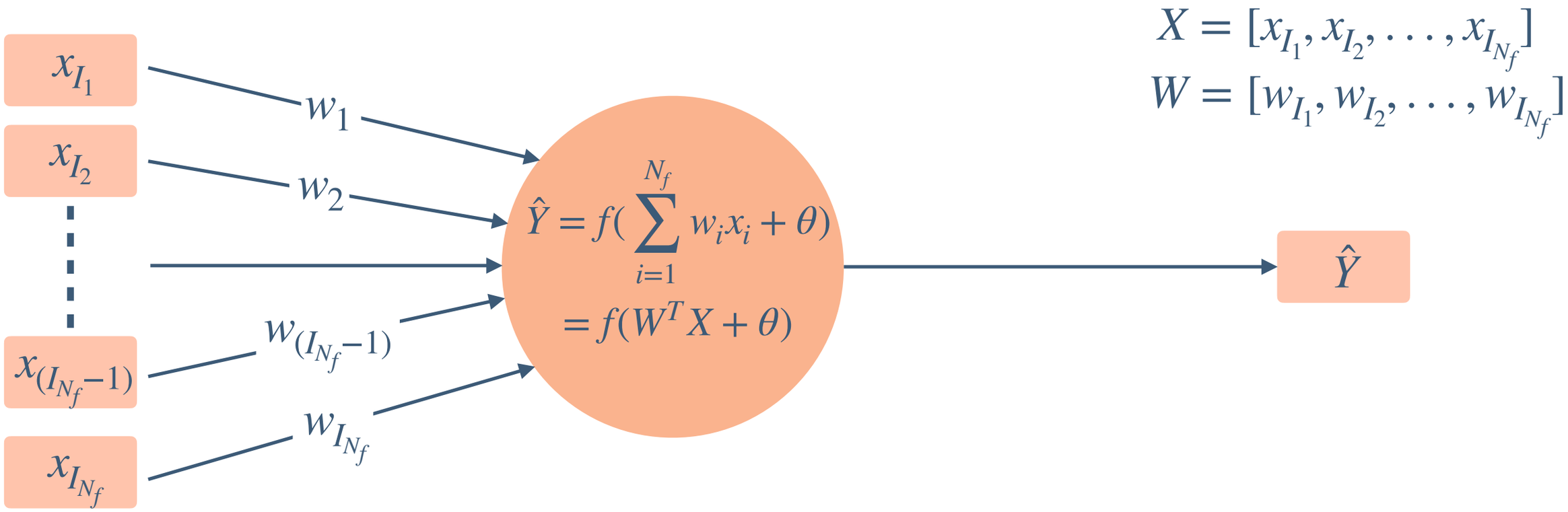
Loss function / figure of merit (FOM)

- Used a gradient decent algorithm to systematically vary m and c to minimise χ^2.

Optimisation / training

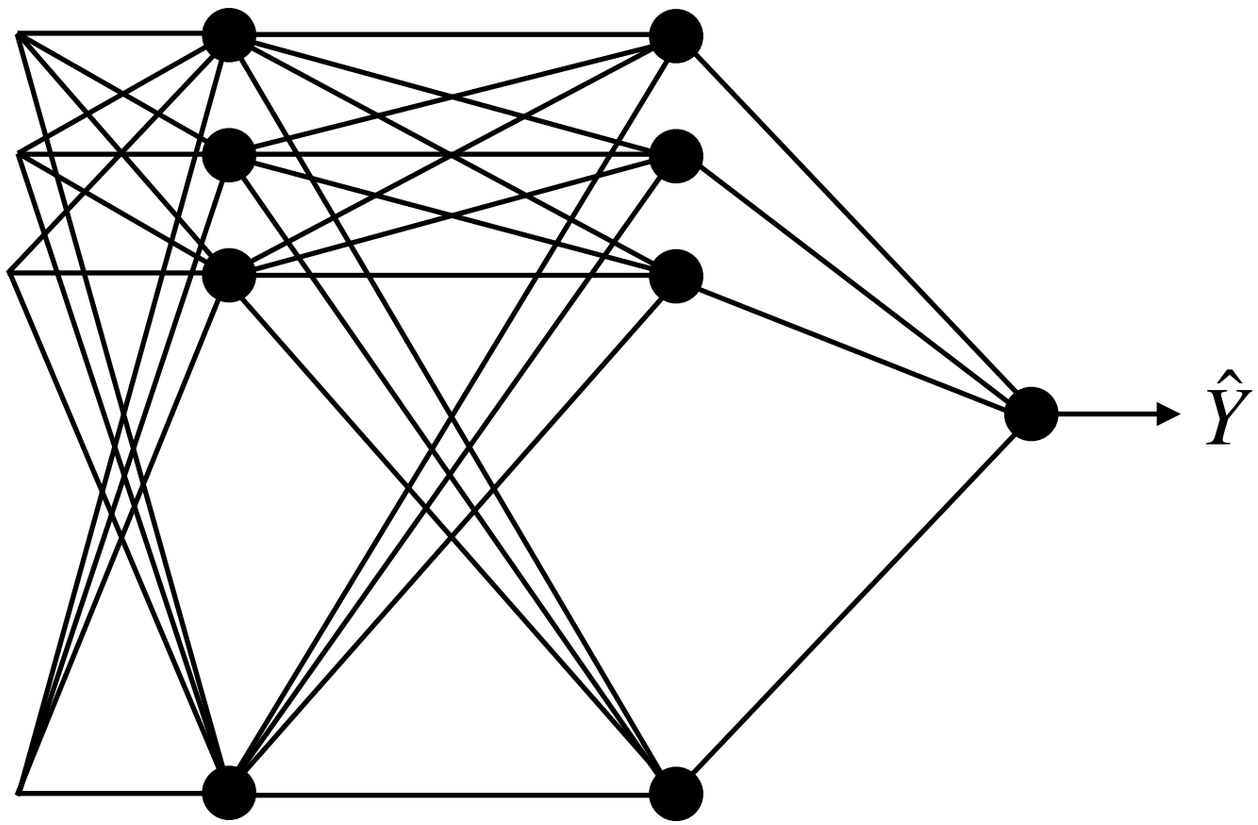
Algorithm moving toward an optimal solution -> **learning.**

How Many Parameters?



- Consider a single perceptron taking N_f **input features**.
- This model has $N_f + 1$ parameters (N_f weights and 1 bias).

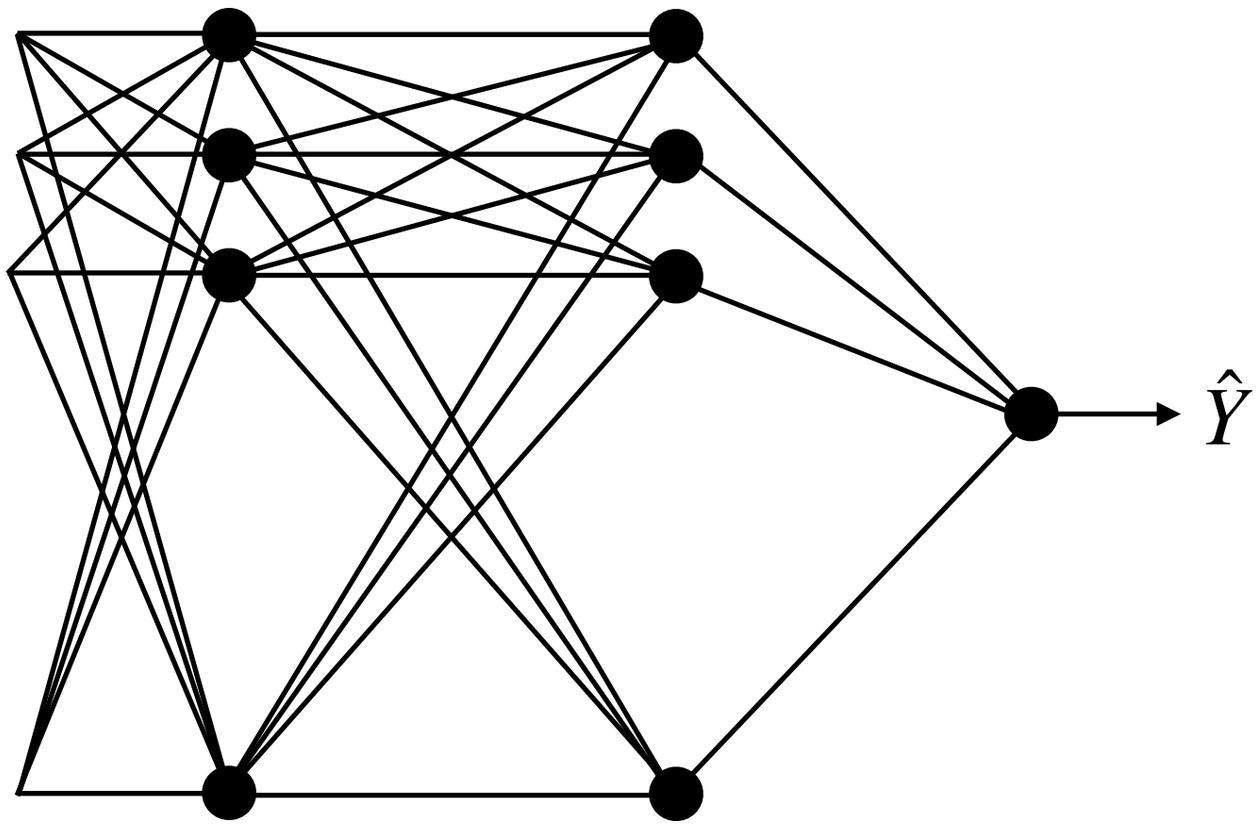
How Many Parameters?



- Neural networks have a **lot** of weights. Deep networks have **millions** of weights to optimise.

- Training a neural network therefore requires:
  - Appropriate computing resources (GPUs).
  - Efficient methods for parameter optimisation.
    - What's acceptable for an optimisation of 10 to 100 weights will **not** generally scale well to a network with 10^3 to 10^6 weights.

How Many Parameters?



- Neural networks have a **lot** of weights. Deep networks have **millions** of weights to optimise.

- Training a neural network therefore requires:
  - Appropriate amount of data requited to obtain a **generalisable** solution for the weights (more weights require more data).

How Do We Train?

- We will use **supervised learning** techniques to train our models.
  - Present the optimisation algorithm with instances of a dataset that are **labelled** (truth information), optimising weights in order to minimise a **loss function**.

	Feature 1: Fur Colour	Feature 2: Weight	Label
Instance 1	Ginger	11 kg	Cat
Instance 2	Brown	8 kg	Dog
Instance 3	Grey	55 kg	Dog
Instance 4	Black	5 kg	Cat

How Do We Train?

- We will use **supervised learning** techniques to train our models.
  - Present the optimisation algorithm with instances of a dataset that are **labelled** (truth information), optimising weights in order to minimise a **loss function**.

	Feature 1: Fur Colour	Feature 2: Weight
Instance 1	Ginger	11 kg
Instance 2	Brown	8 kg
Instance 3	Grey	55 kg
Instance 4	Black	5 kg

In **unsupervised learning** there are no labels.

Typical use case in clustering problems - leave the algorithm to figure out how to group the data.

Loss Function Examples

- There are a number of different loss functions that are commonly used.

- In the linear regression exercise we used the **mean squared error (MSE)** or the sum of training dataset instances of:

$$\epsilon_i = (y_i - \hat{y}_i)^2 \quad , \quad i \in [1, N_I]$$

- Where:
  - ϵ_i is the loss function contribution for the i^{th} instance given the model output y_i and the true target label value $\hat{y}_i$.

Loss Function Examples

- There are a number of different loss functions that are commonly used.

- Also commonly used is **mean absolute error (MAE)** or the sum of training dataset instances of:

$$\epsilon_i = |y_i - \hat{y}_i| \quad , \quad i \in [1, N_I]$$

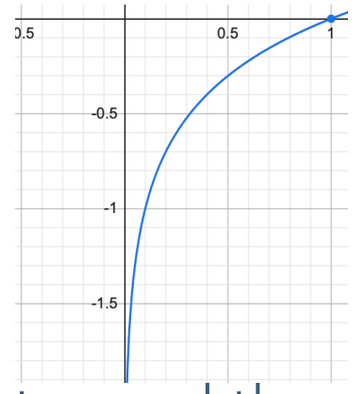
- Where:
  - ϵ_i is the loss function contribution for the i^{th} instance given the model output y_i and the true target label value $\hat{y}_i$.

Loss Function Examples

- There are a number of different loss functions that are commonly used.

- In classification problems the most commonly used loss function is the **cross-entropy loss (log loss).**

$$\epsilon_i = -\, y_i \log(\hat{y}_i) \quad , \quad i \in [1, N_I]$$



- Where:
  - ϵ_i is the loss function contribution for the i^{th} instance given the model output y_i and the true target label value $\hat{y}_i$.

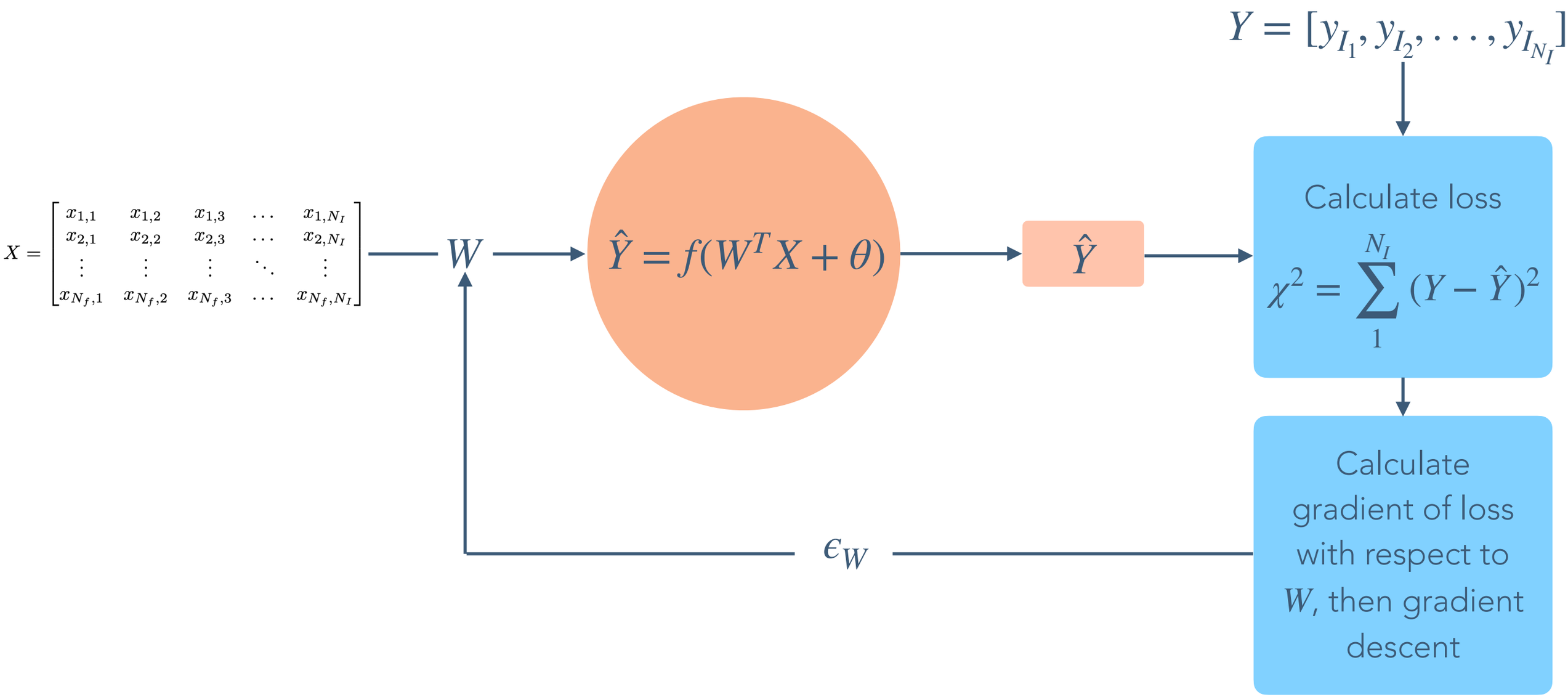
How Do We Train?

- Within **supervised learning** is the concept of **batch learning** -> working with a “complete dataset”.

- If working with large dataset, can’t feed all instances through network in one go - computationally difficult / inefficient-> **mini-batch learning.**

How Do We Train? Mini-batch Learning

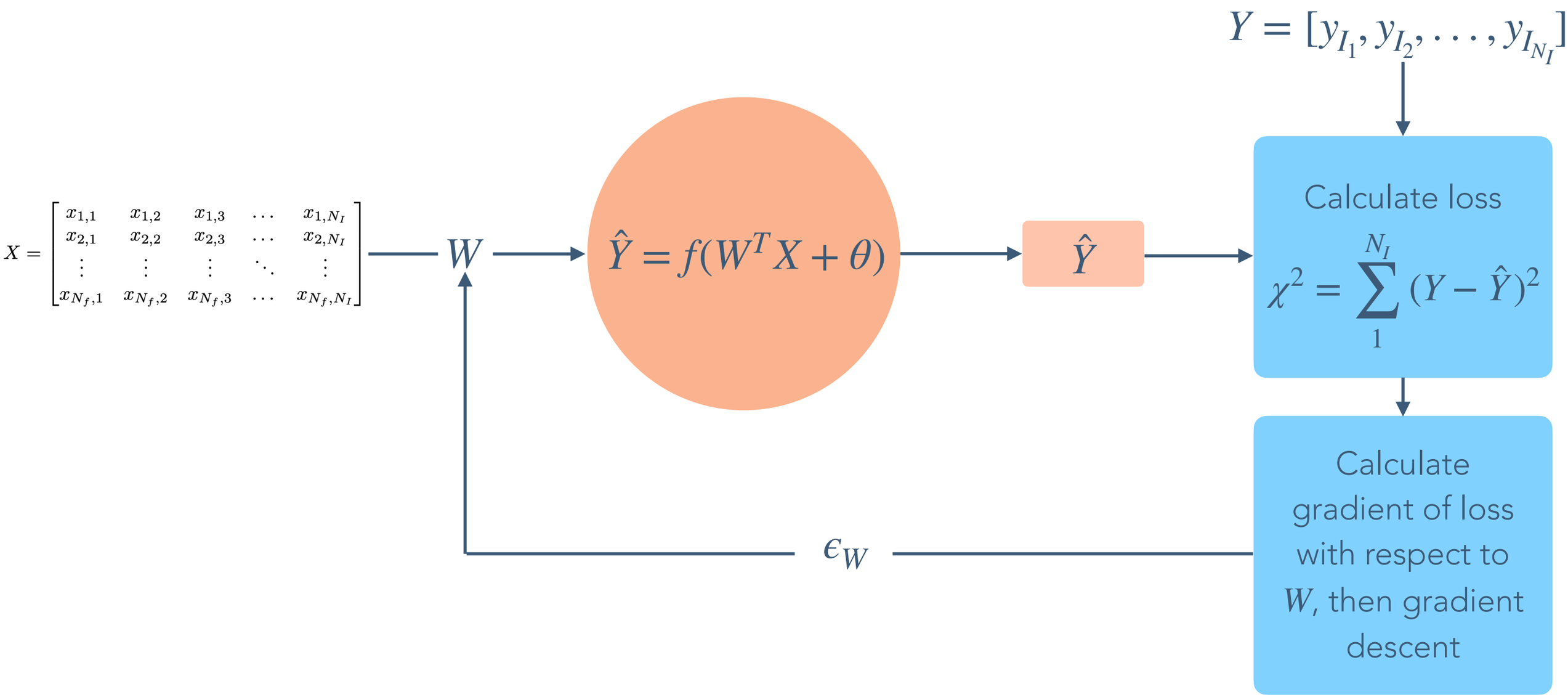
- An **epoch** -> "a single run of the optimisation algorithm".



How Do We Train? Mini-batch Learning

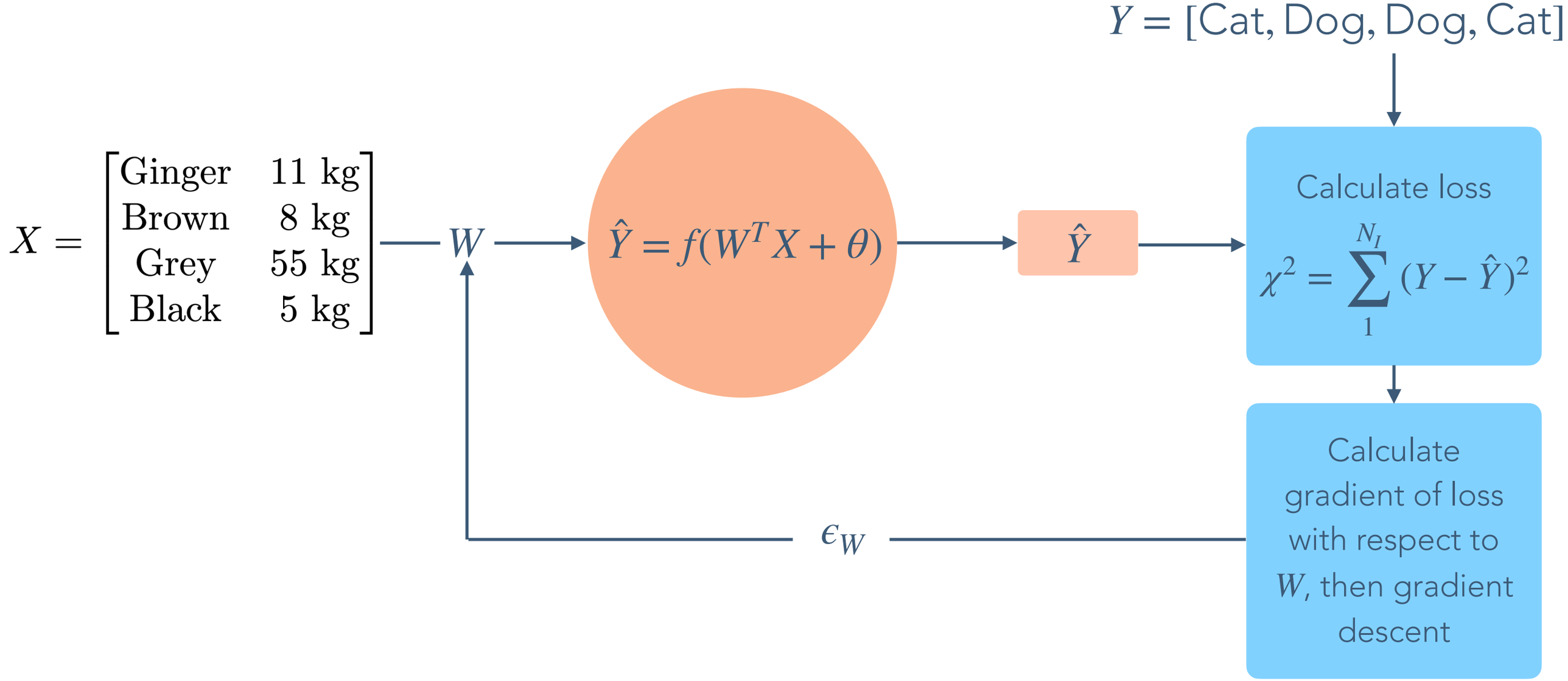
- An **epoch** -> ~~"a single run of the optimisation algorithm"~~.

An **epoch**: one **complete pass** of the training dataset (all instances) through the optimisation algorithm.



How Do We Train? Batch Learning

	Feature 1: Fur Colour	Feature 2: Weight	Label
Instance 1	Ginger	11 kg	Cat
Instance 2	Brown	8 kg	Dog
Instance 3	Grey	55 kg	Dog
Instance 4	Black	5 kg	Cat



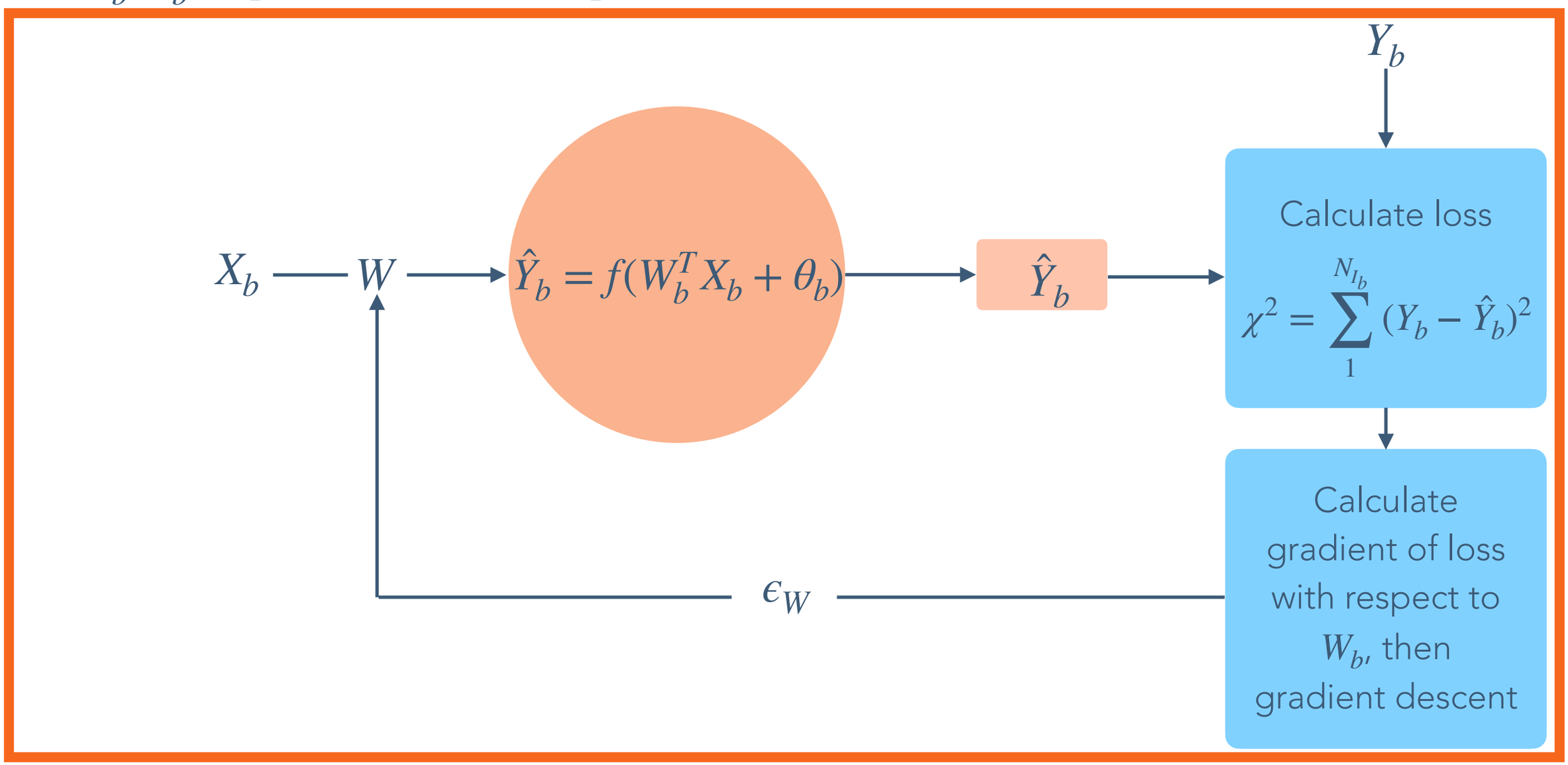
How Do We Train? Mini-batch Learning

Batch 1

	Feature 1: Fur Colour	Feature 2: Weight	Label
Instance 1	Ginger	11 kg	Cat
Instance 2	Brown	8 kg	Dog
Instance 3	Grey	55 kg	Dog
Instance 4	Black	5 kg	Cat

Batch 2

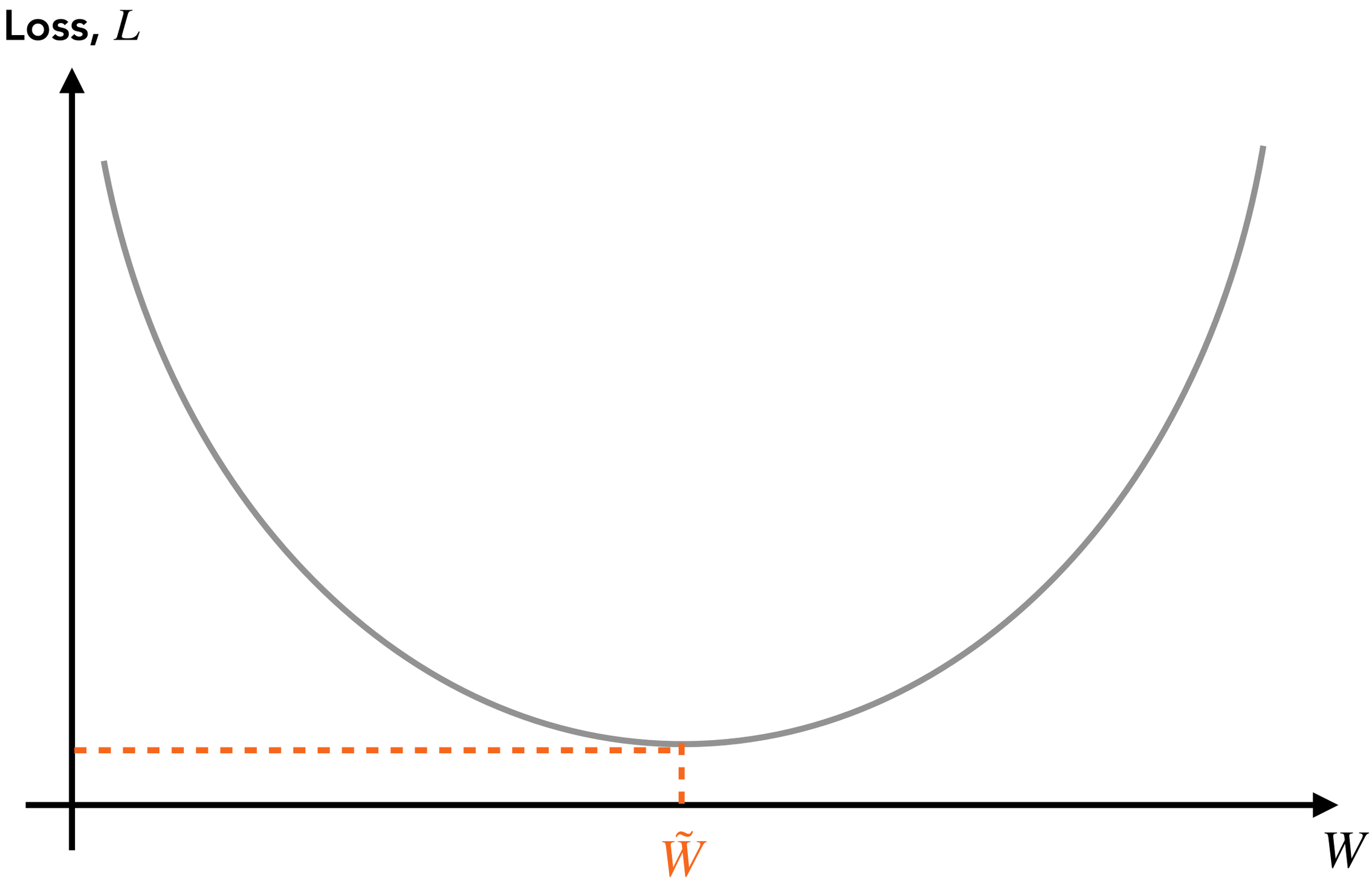
for $X_b, Y_b \in$ [Batch 1, Batch 2]



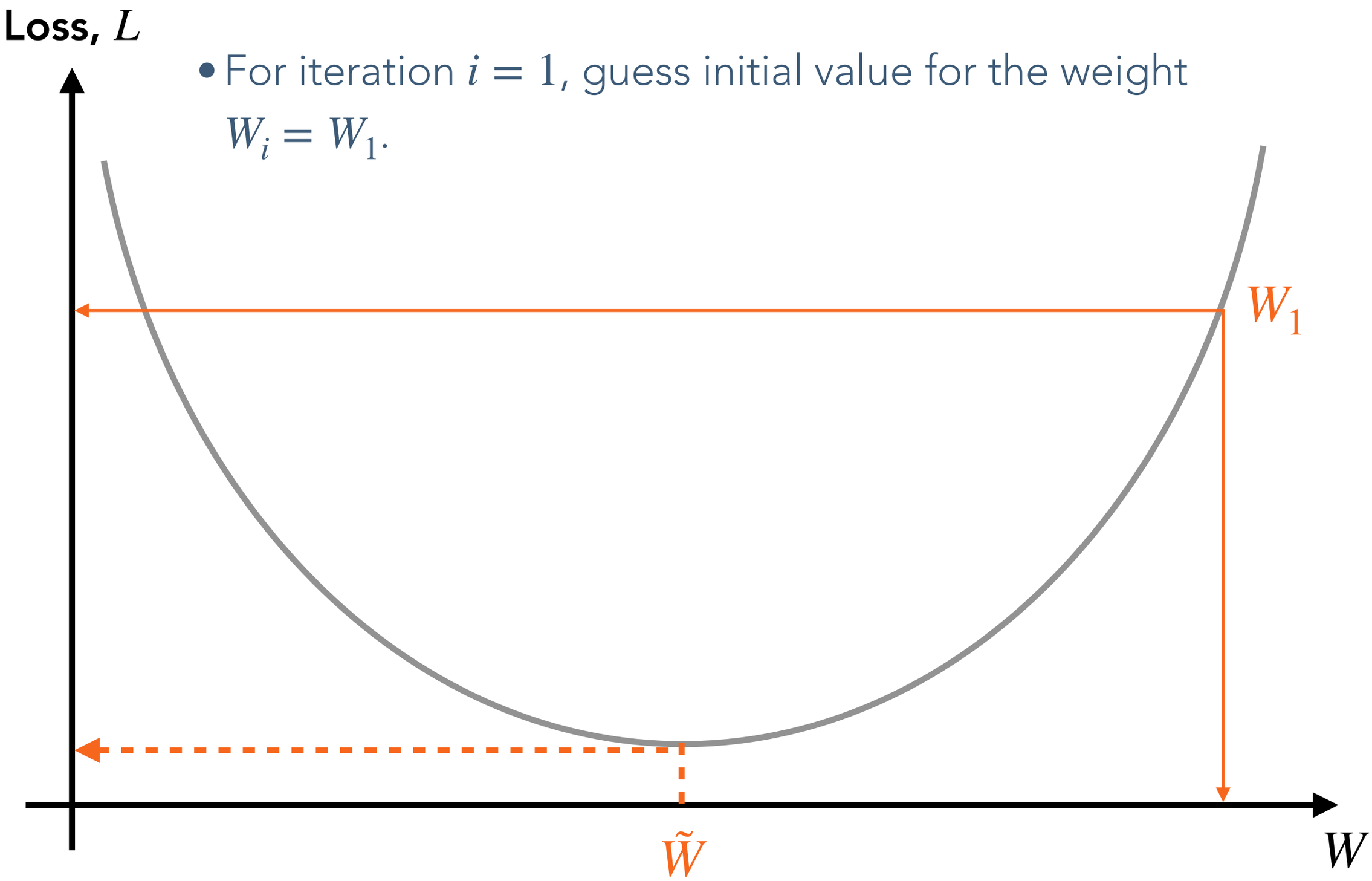
How Do We Train? Gradient Descent

- Gradient descent is a very generic optimisation algorithm capable of finding optimal solution to a wide range of problems.
- General idea is to **tweak parameters iteratively** in order to minimise a loss function.
- Imagine you're lost in the mountains in a dense fog and want to get to the bottom of the valley - good strategy is to go downhill in the direction of steepest slope. This is what gradient descent does!
  - Measure local gradient with respect to W.
  - Shift W in the direction of descending gradient until hit a minimum.

Gradient Descent



Gradient Descent

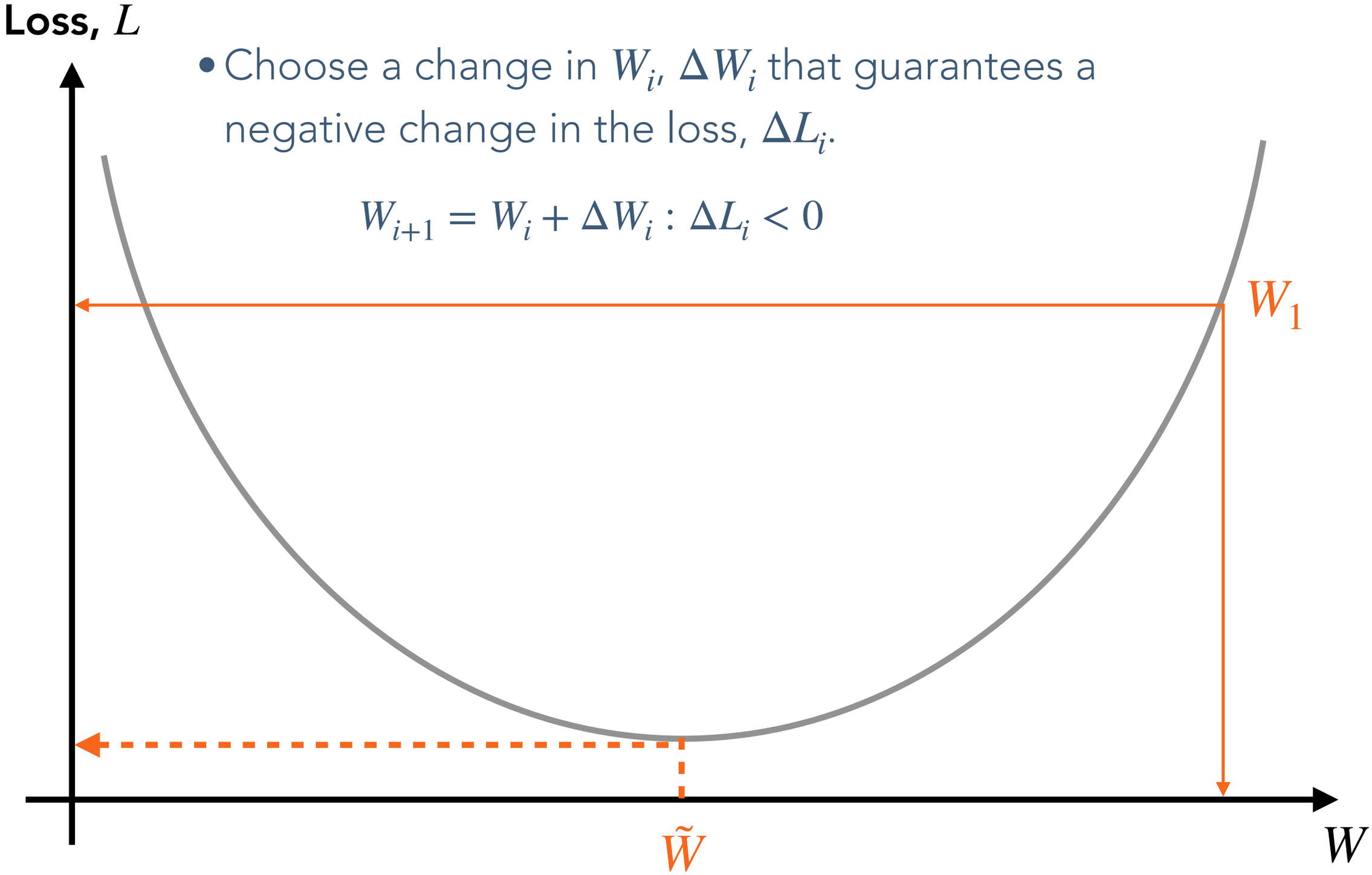


- For iteration $i = 1$, guess initial value for the weight $W_i = W_1$.

Gradient Descent

- Choose a change in W_i, ΔW_i that guarantees a negative change in the loss, ΔL_i.

$$W_{i+1} = W_i + \Delta W_i : \Delta L_i < 0$$

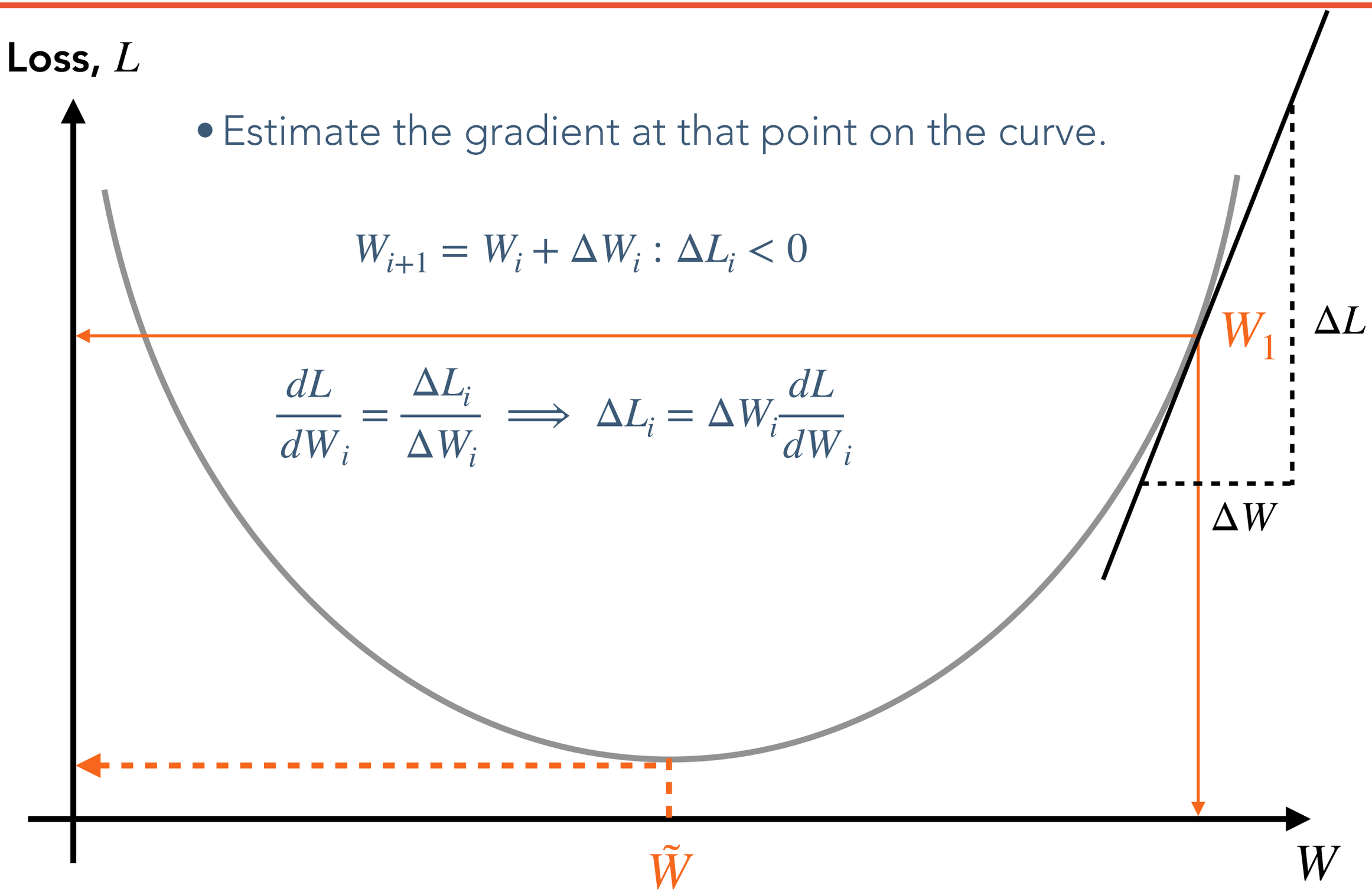


Gradient Descent

- Estimate the gradient at that point on the curve.

$$W_{i+1} = W_i + \Delta W_i : \Delta L_i < 0$$

$$\frac{dL}{dW_i} = \frac{\Delta L_i}{\Delta W_i} \implies \Delta L_i = \Delta W_i \frac{dL}{dW_i}$$



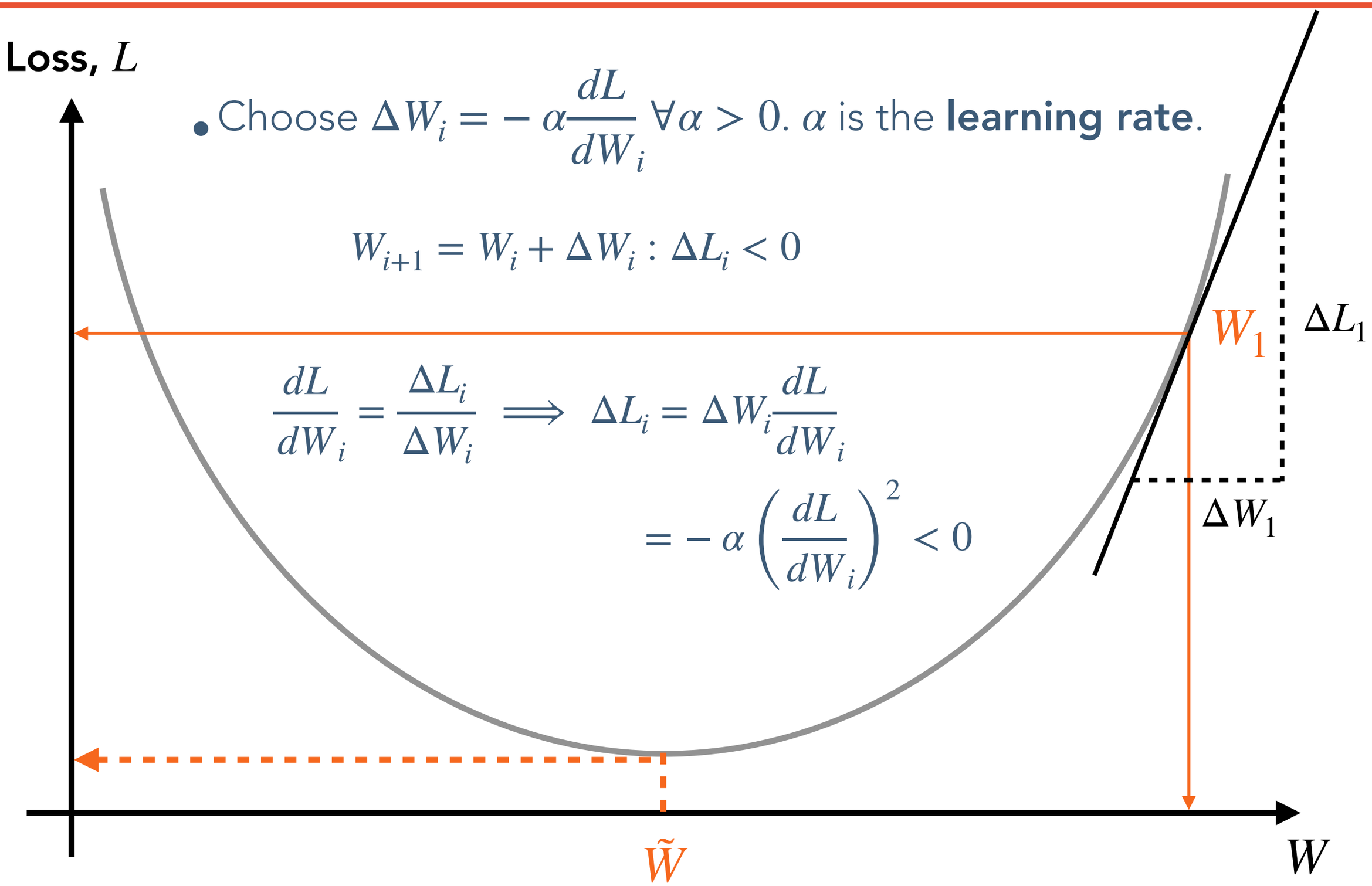
Gradient Descent

- Choose $\Delta W_i = -\alpha \frac{dL}{dW_i} \; \forall \alpha > 0$. α is the **learning rate**.

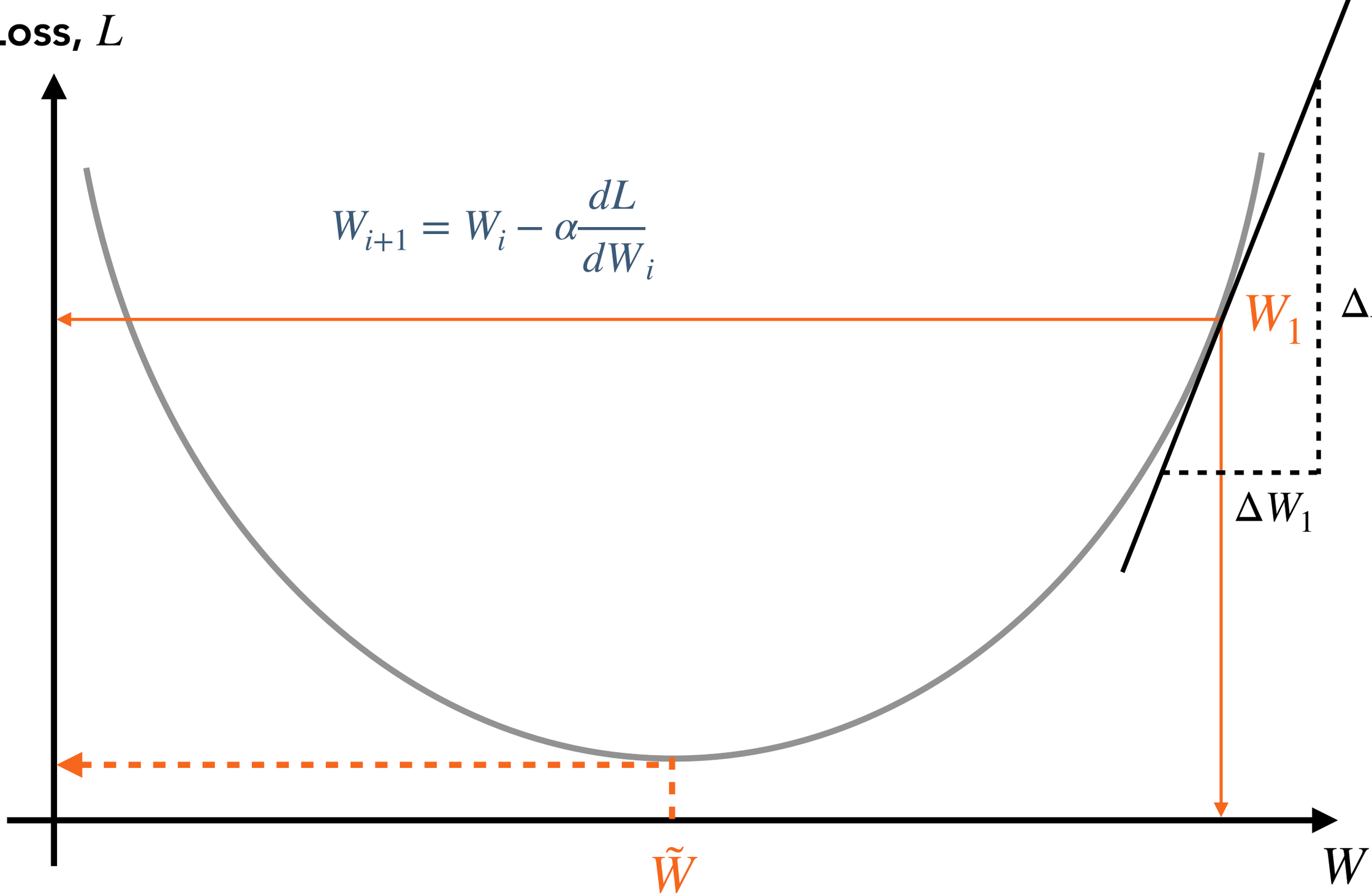
$$W_{i+1} = W_i + \Delta W_i : \Delta L_i < 0$$

$$\frac{dL}{dW_i} = \frac{\Delta L_i}{\Delta W_i} \implies \Delta L_i = \Delta W_i \frac{dL}{dW_i}$$

$$= -\alpha \left(\frac{dL}{dW_i} \right)^2 < 0$$



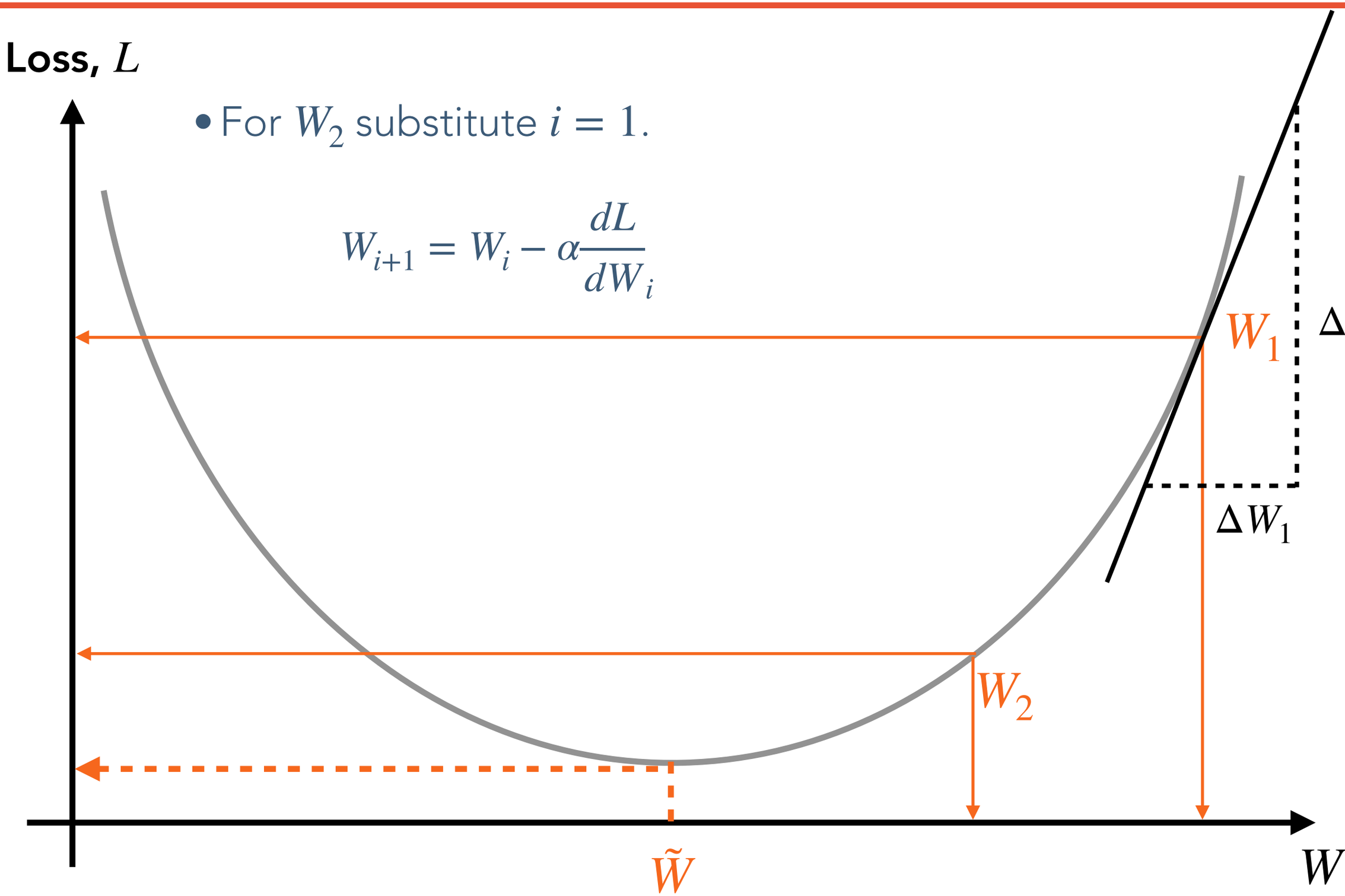
Gradient Descent



Gradient Descent

- For W_2 substitute $i = 1$.

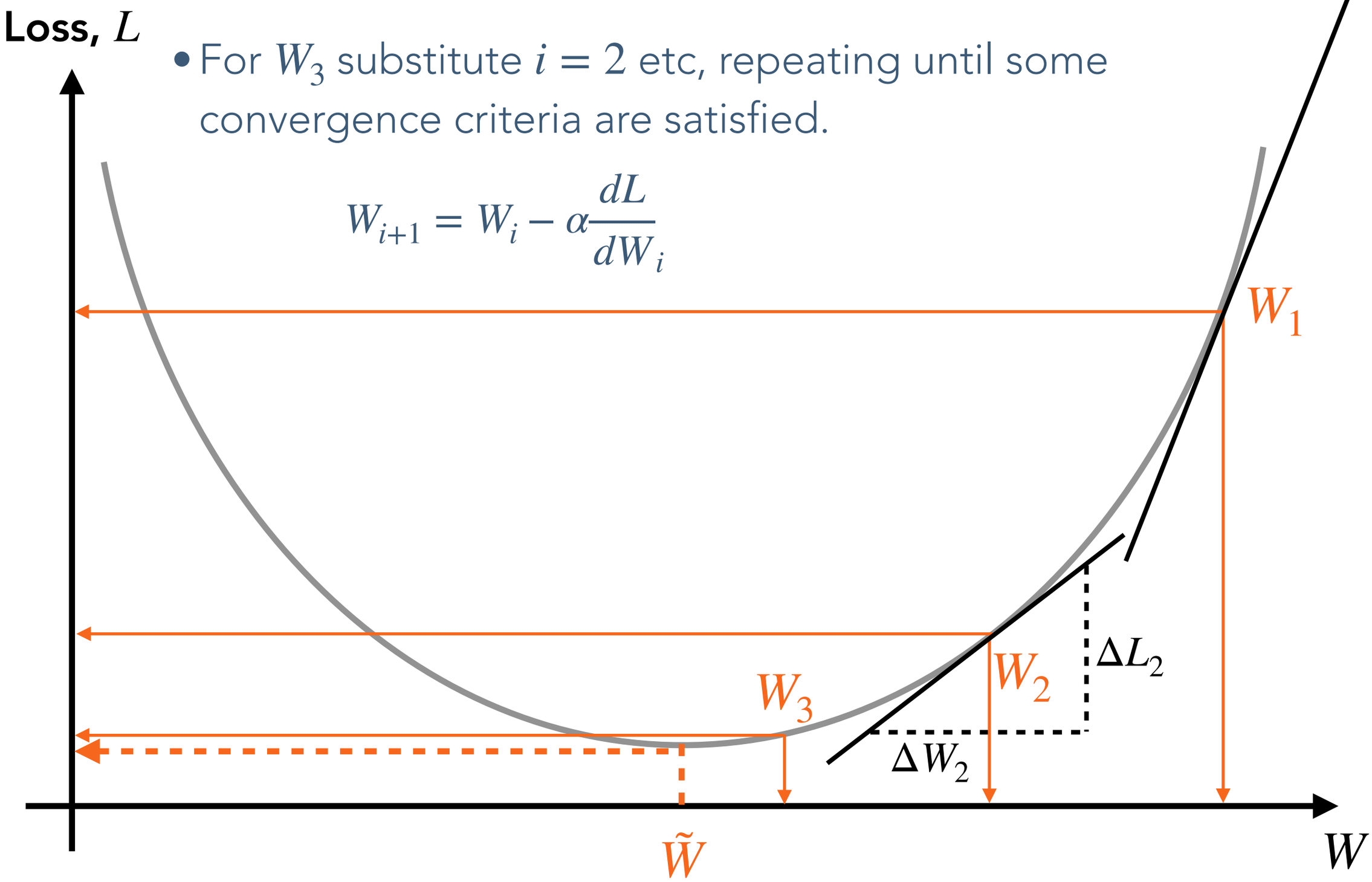
$$W_{i+1} = W_i - \alpha \frac{dL}{dW_i}$$



Gradient Descent

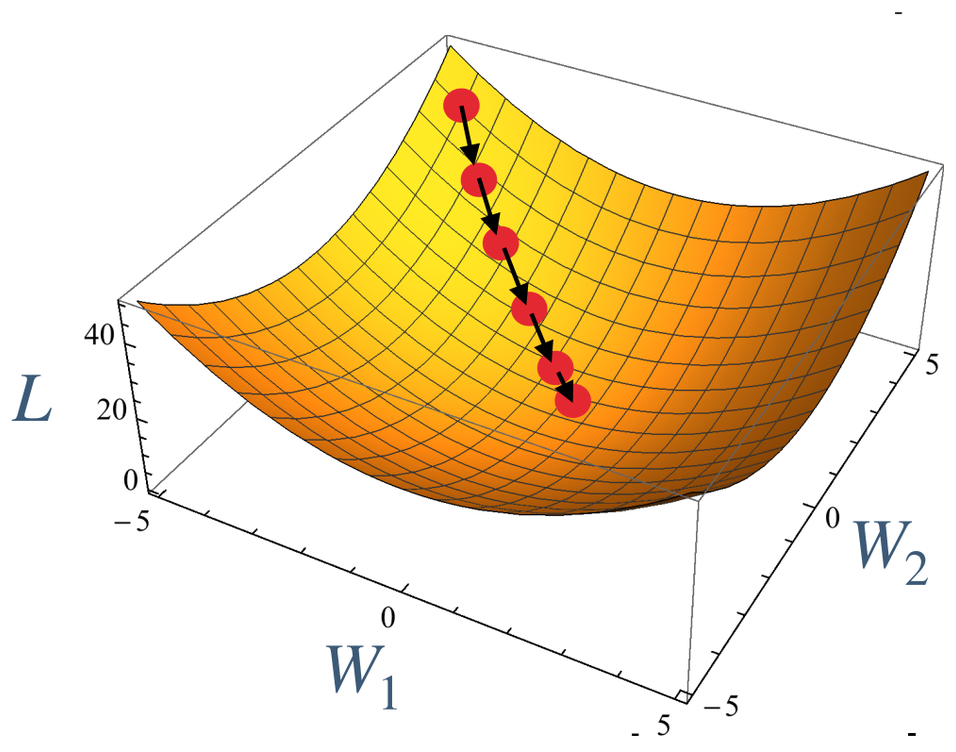
- For W_3 substitute $i = 2$ etc, repeating until some convergence criteria are satisfied.

$$W_{i+1} = W_i - \alpha \frac{dL}{dW_i}$$



Gradient Descent

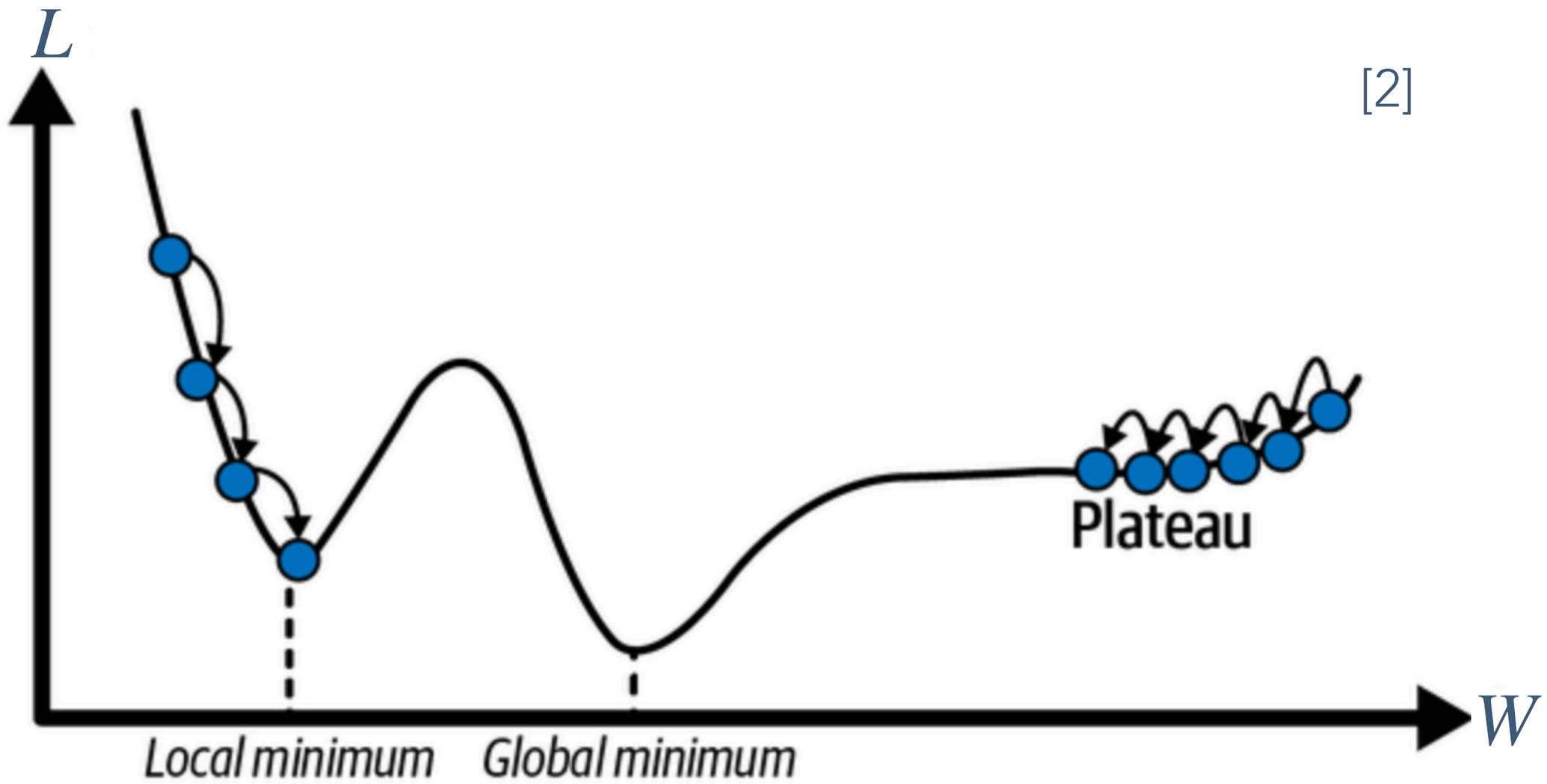
- Can easily be extended many dimensions.



$$W_{i+1} = W_i - \alpha(\nabla_W L)_i$$

$$= W_i - \alpha\left(\frac{\partial}{\partial W_{1\,i}} + \ldots + \frac{\partial}{\partial W_{N\,i}}\right) L$$

Gradient Descent: Limitations



[2]

- Obtaining the gradients in this way requires calculations over the full (or mini-batch) training set -> **slow** on large datasets.
- Gradient descent algorithms are built on the assumption that there is one minimum in the loss function -> rarely the case.
- How do we know that we have converged on the **global** minimum?

Gradient Descent: ADAM

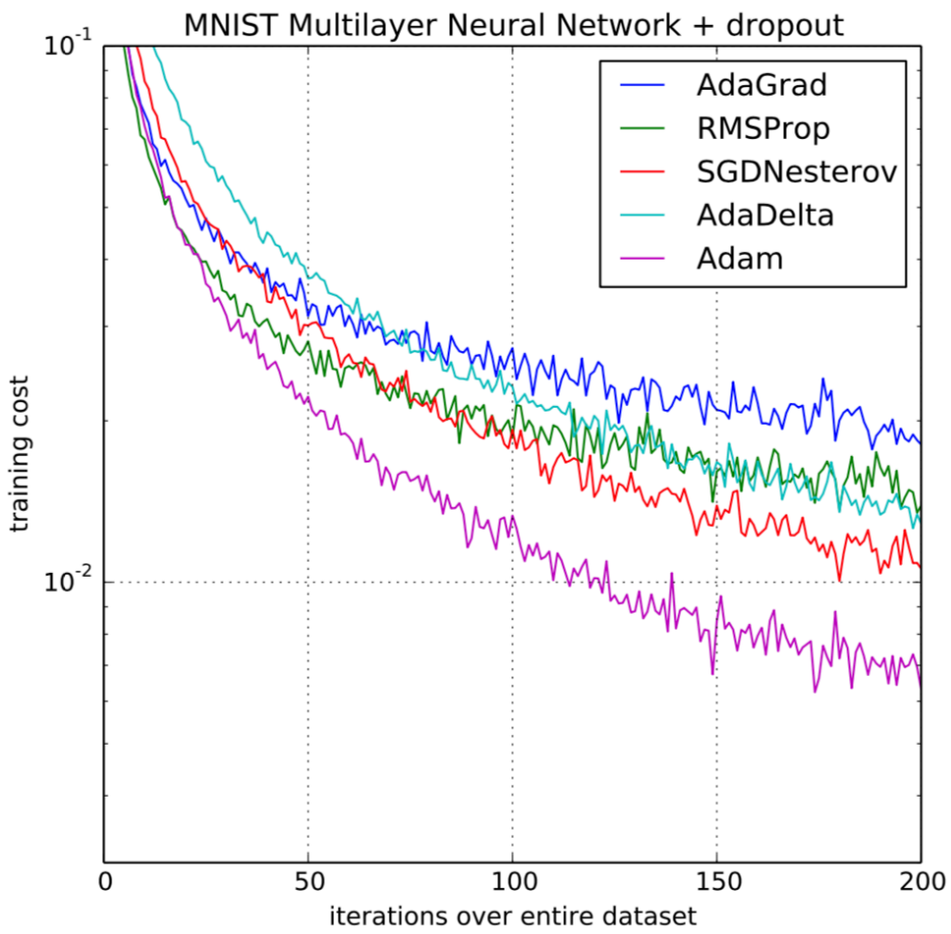
- **ADA**ptive **M**oment (**ADAM**) [1] optimiser is a type of gradient descent algorithm.
  - Is **stochastic** -> gradients are calculated for a **single instance chosen at random** from the training set.
  - **Uses momentum** -> uses knowledge of past gradients to inform current step **to not get stuck in plateaus**.
  - **Uses RMSProp** -> uses knowledge of past gradients to inform current step to **work through vanishing and exploding gradients**.

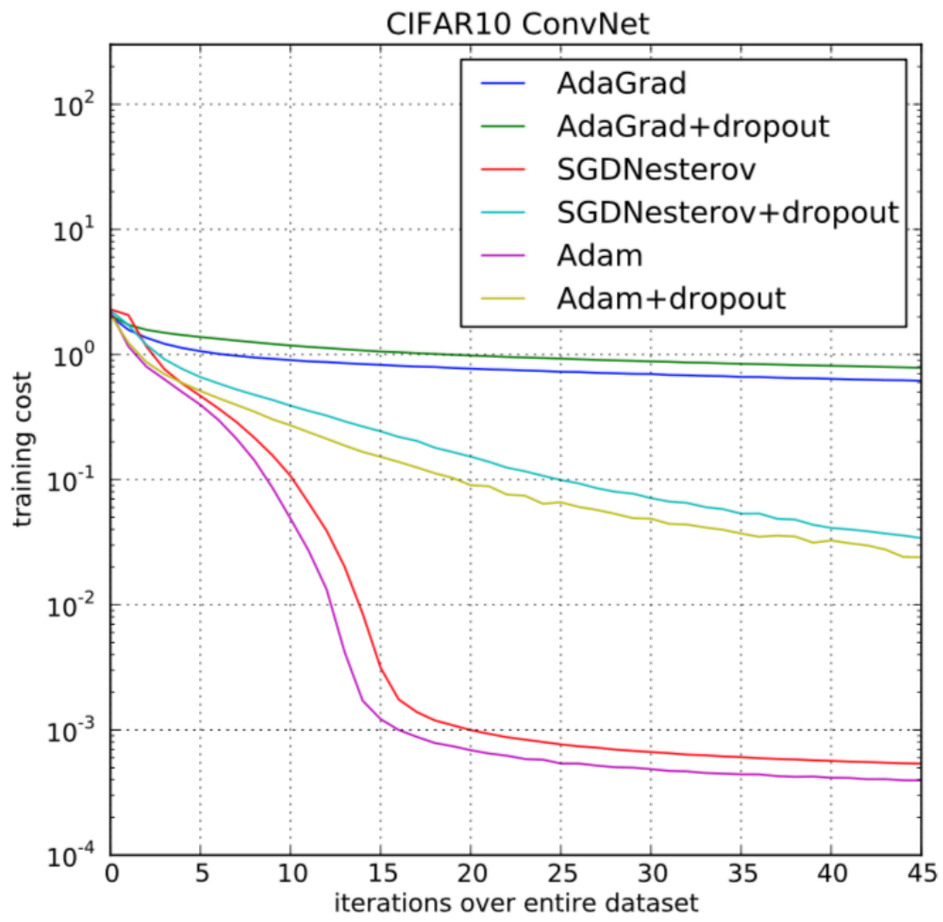
Gradient Descent: ADAM

- Benchmarking of ADAM performance using MNIST and CFAR10 datasets efficacy of the technique.

- Faster drop-off in loss and lower overall loss obtained.

[1]





Over Fitting (Over Training)

Queen Mary
University of London

What is it?

How to spot it.

Over Fitting (Over Training)

How to mitigate it.

Queen Mary
University of London

What is a (Good) Model?

- A model is a mathematical function which is a **simplified representation** of some phenomenon or process that's captured in our data [3].

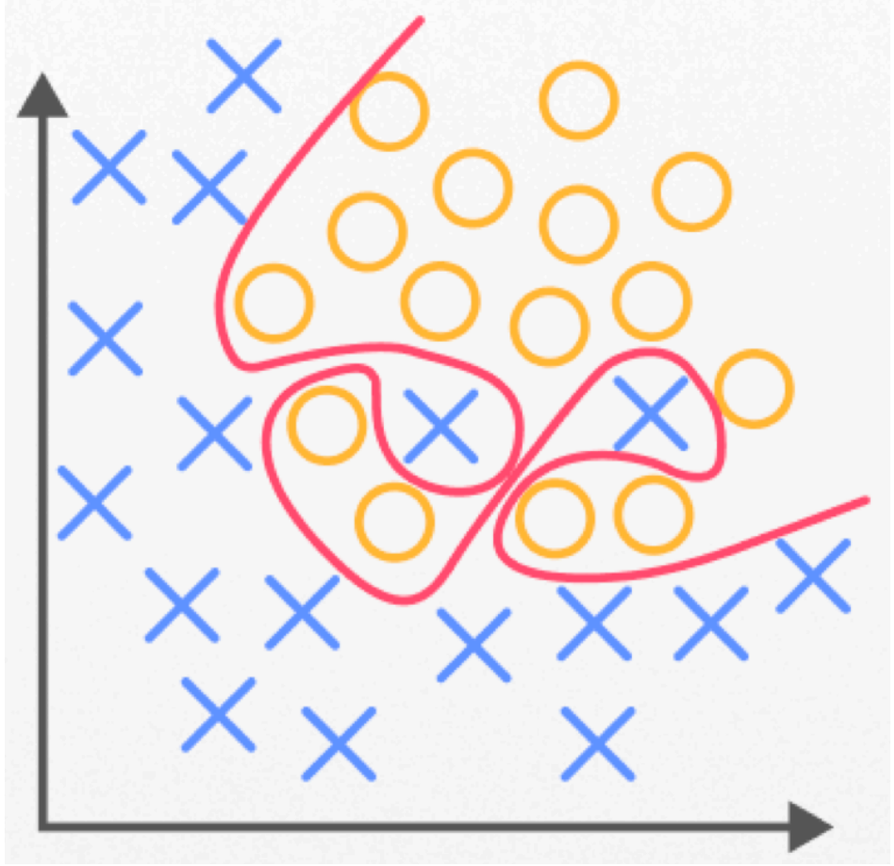
- A good model is one that has just the right amount of complexity. It is **simple enough** that it excludes all unnecessary and irrelevant details but it's **complex enough** to accurately represent the thing we are trying to model [3].

Over Fitting: What is it?

- A model is over fitted if the parameters that have been determined are **tuned to the statistical fluctuations** in the dataset - **too complex**.

- Neural networks can detect subtle patterns in the data, but if training set is noisy (or small, introducing sampling noise) the model is likely to detect patterns in the noise itself -> **patterns will not generalise to new instances**.

- Model is so closely aligned to the **training data** that it does not know how to respond to new unseen data.

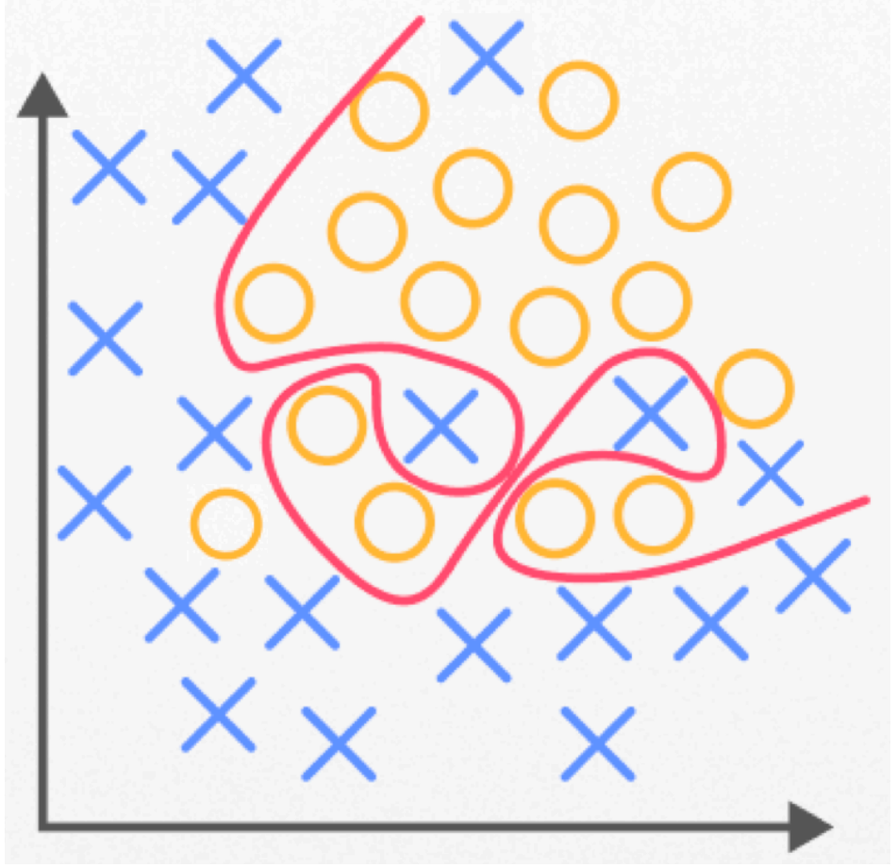


Over Fitting: What is it?

- A model is over fitted if the parameters that have been determined are **tuned to the statistical fluctuations** in the dataset - **too complex**.

- Neural networks can detect subtle patterns in the data, but if training set is noisy (or small, introducing sampling noise) the model is likely to detect patterns in the noise itself -> **patterns will not generalise to new instances**.

- Model is so closely aligned to the training data that it does not know how to respond to **new unseen data**.

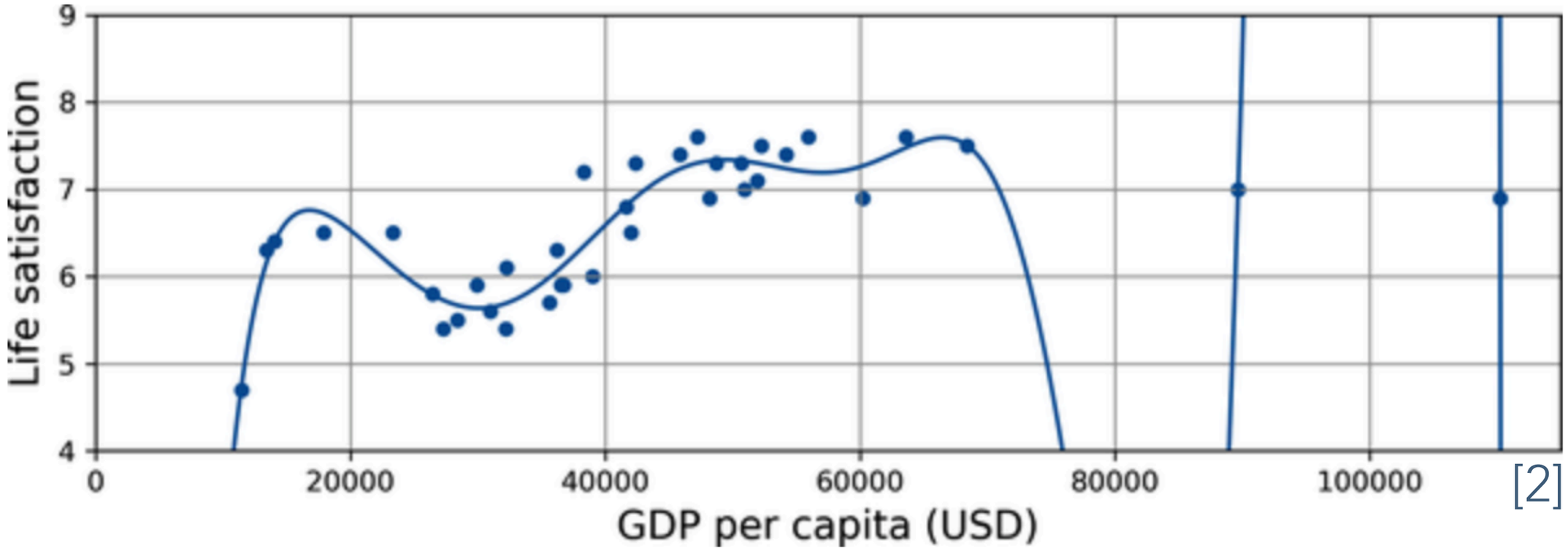


Over Fitting: What is it?

- A model is over fitted if the parameters that have been determined are **tuned to the statistical fluctuations** in the dataset - **too complex**.

- Neural networks can detect subtle patterns in the data, but if training set is noisy (or small, introducing sampling noise) the model is likely to detect patterns in the noise itself -> **patterns will not generalise to new instances**.

- Model is so closely aligned to the **training data** that it does not know how to respond to new unseen data.



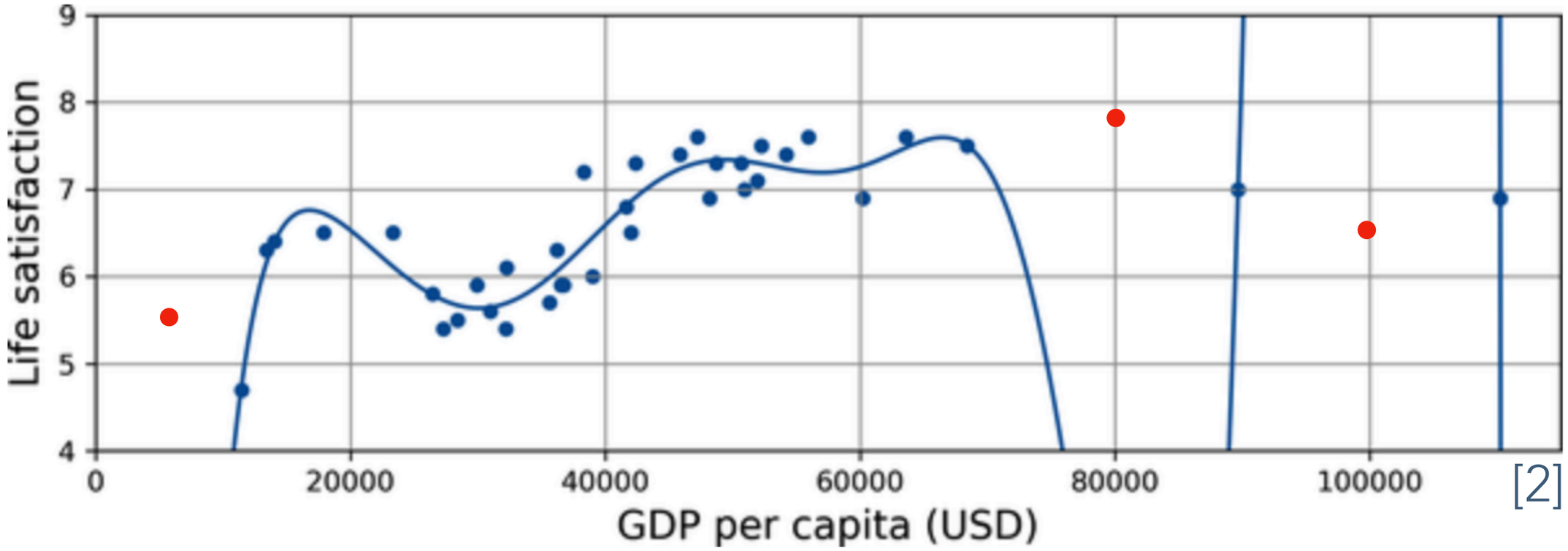
[2]

Over Fitting: What is it?

- A model is over fitted if the parameters that have been determined are **tuned to the statistical fluctuations** in the dataset - **too complex**.

- Neural networks can detect subtle patterns in the data, but if training set is noisy (or small, introducing sampling noise) the model is likely to detect patterns in the noise itself -> **patterns will not generalise to new instances**.

- Model is so closely aligned to the training data that it does not know how to respond to **new unseen data**.



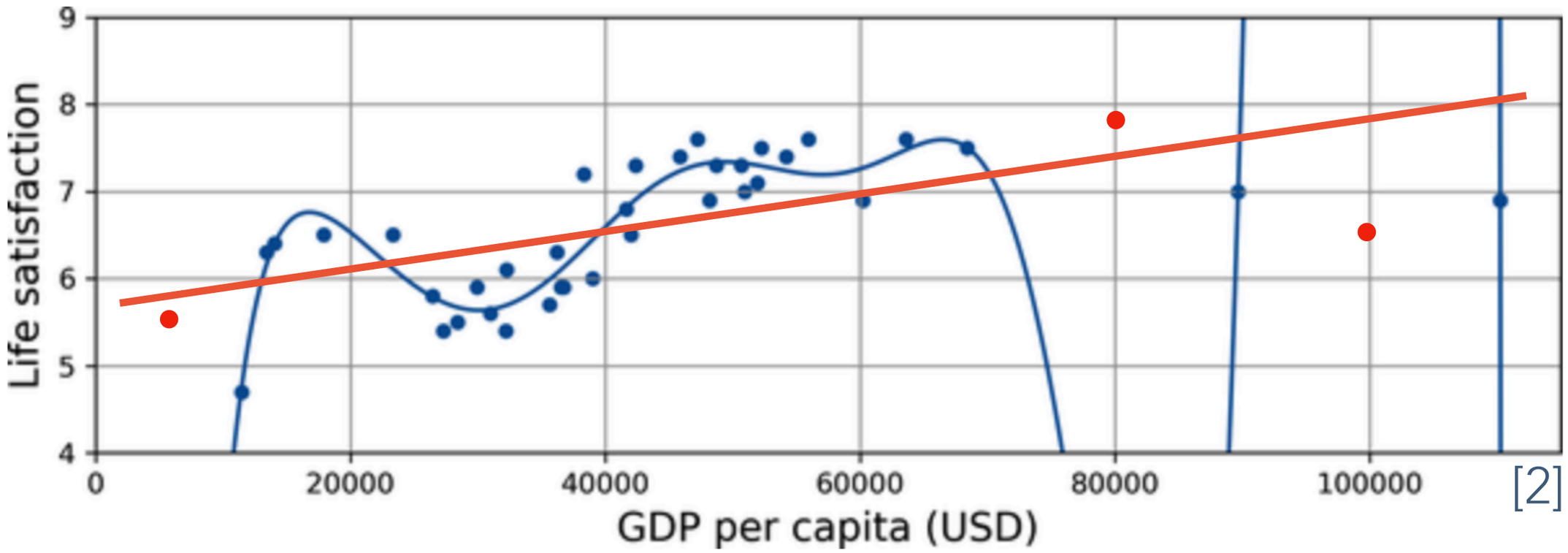
[2]

Over Fitting: What is it?

- A model is over fitted if the parameters that have been determined are **tuned to the statistical fluctuations** in the dataset - **too complex**.

- Neural networks can detect subtle patterns in the data, but if training set is noisy (or small, introducing sampling noise) the model is likely to detect patterns in the noise itself -> **patterns will not generalise to new instances**.

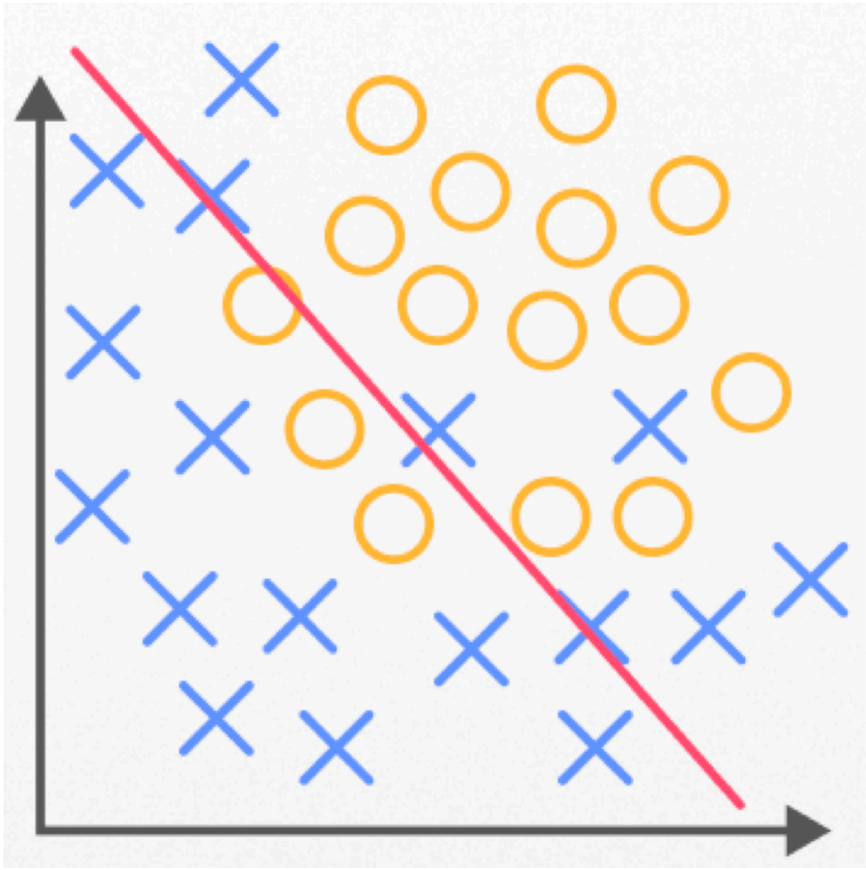
- Model is so closely aligned to the training data that it does not know how to respond to **new unseen data**.



[2]

Over Fitting: What is it?

- Can also **under fit** - model is **too simple** to capture behaviours in the data we wish to reproduce.

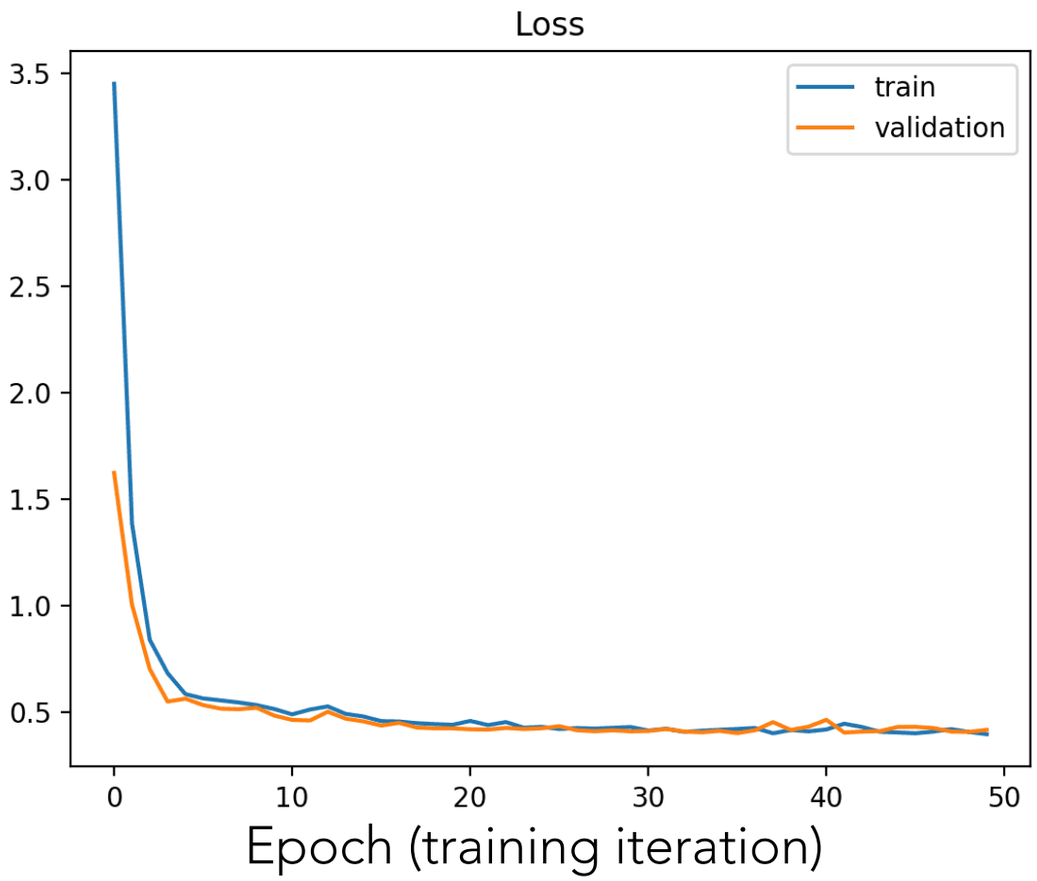


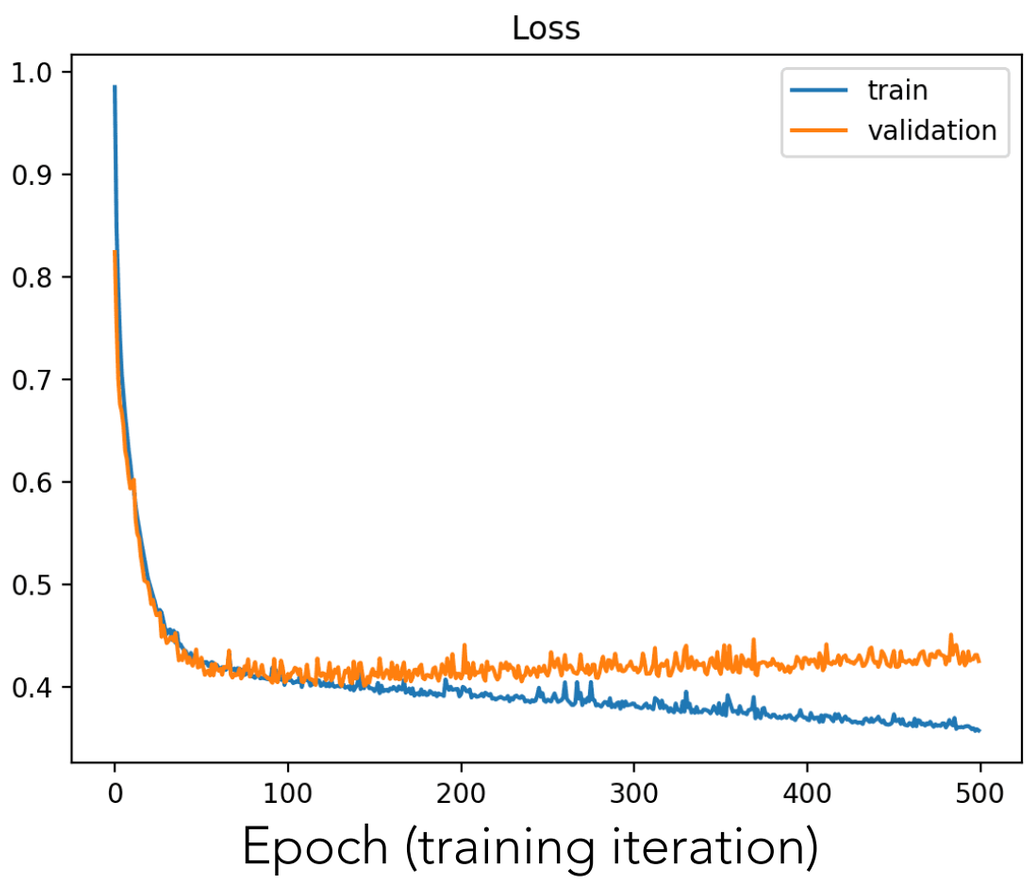
Over Fitting: How to spot it?

- Monitor the model's generalisability during training by introducing **a validation step** using a **validation sample**.
  - Before training, set aside 5% to 20% of your dataset to be **a validation sample**.

Over Fitting: How to spot it?

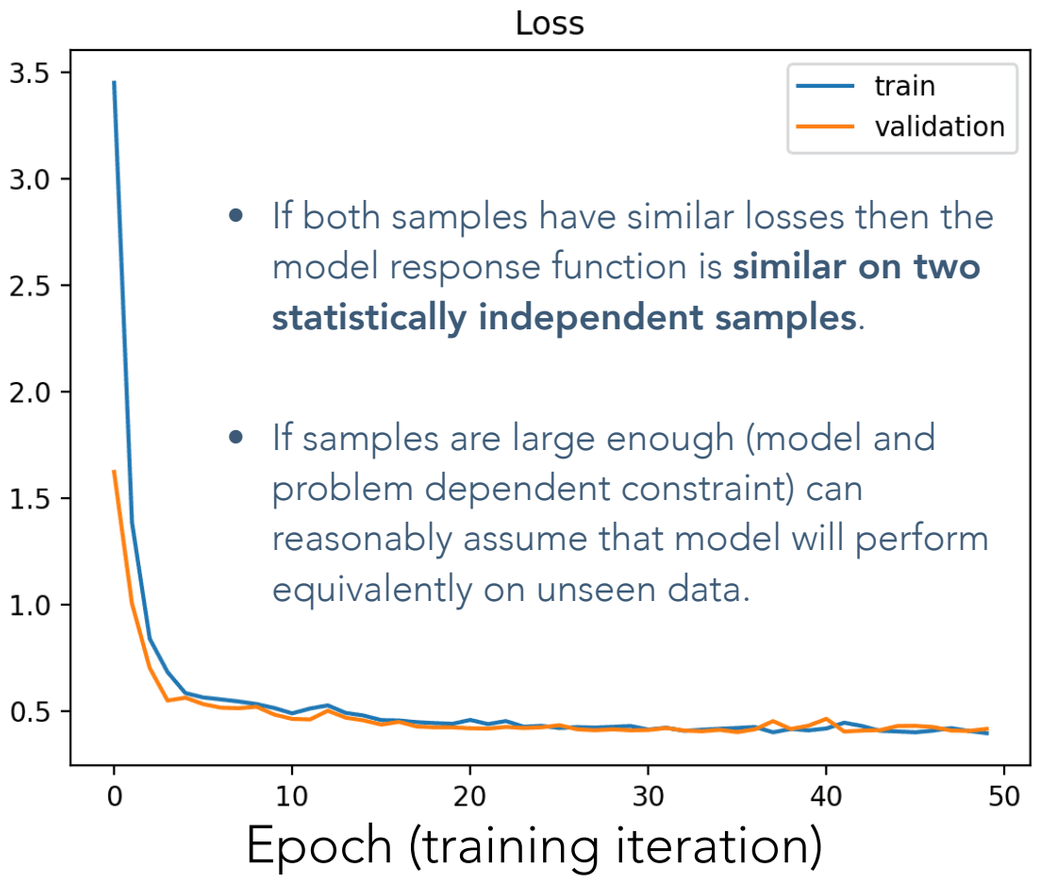
- Monitor the model's generalisability during training by introducing **a validation step** using a **validation sample**.
  - Before training, set aside 5% to 20% of your dataset to be **a validation sample**.
  - At regular intervals in the training (at the end of each epoch), evaluate the **current model** (evaluate the loss function) **on the validation sample**.
  - Monitor using the **learning curve**.

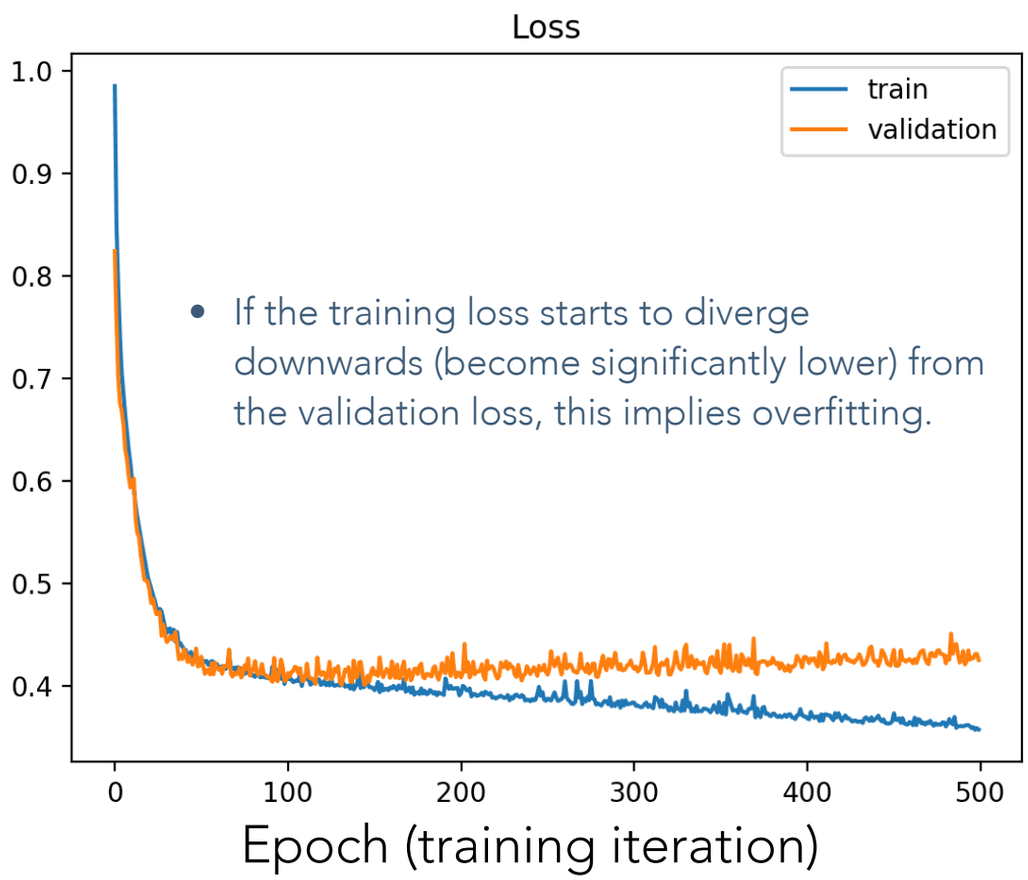




Over Fitting: How to spot it?

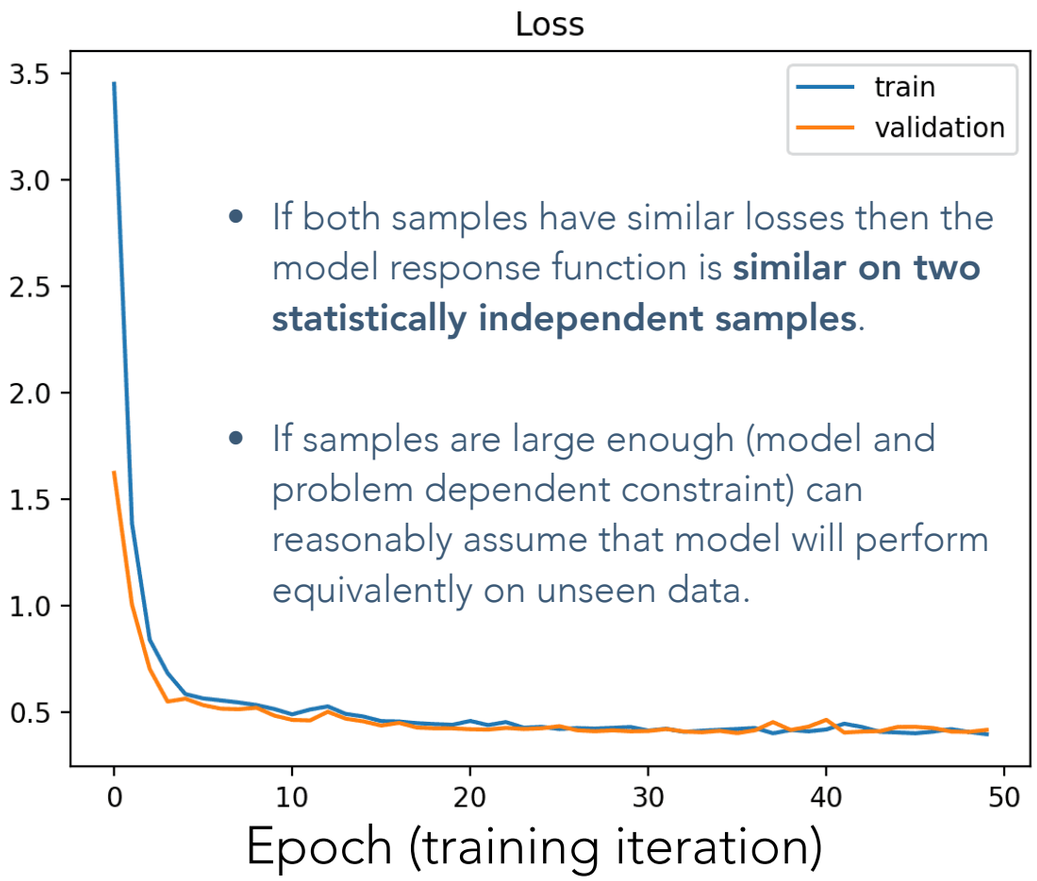
- Monitor the model's generalisability during training by introducing **a validation step** using a **validation sample**.
  - Before training, set aside 5% to 20% of your dataset to be **a validation sample**.
  - At regular intervals in the training (at the end of each epoch), evaluate the **current model** (evaluate the loss function) **on the validation sample**.
  - Monitor using the **learning curve**.

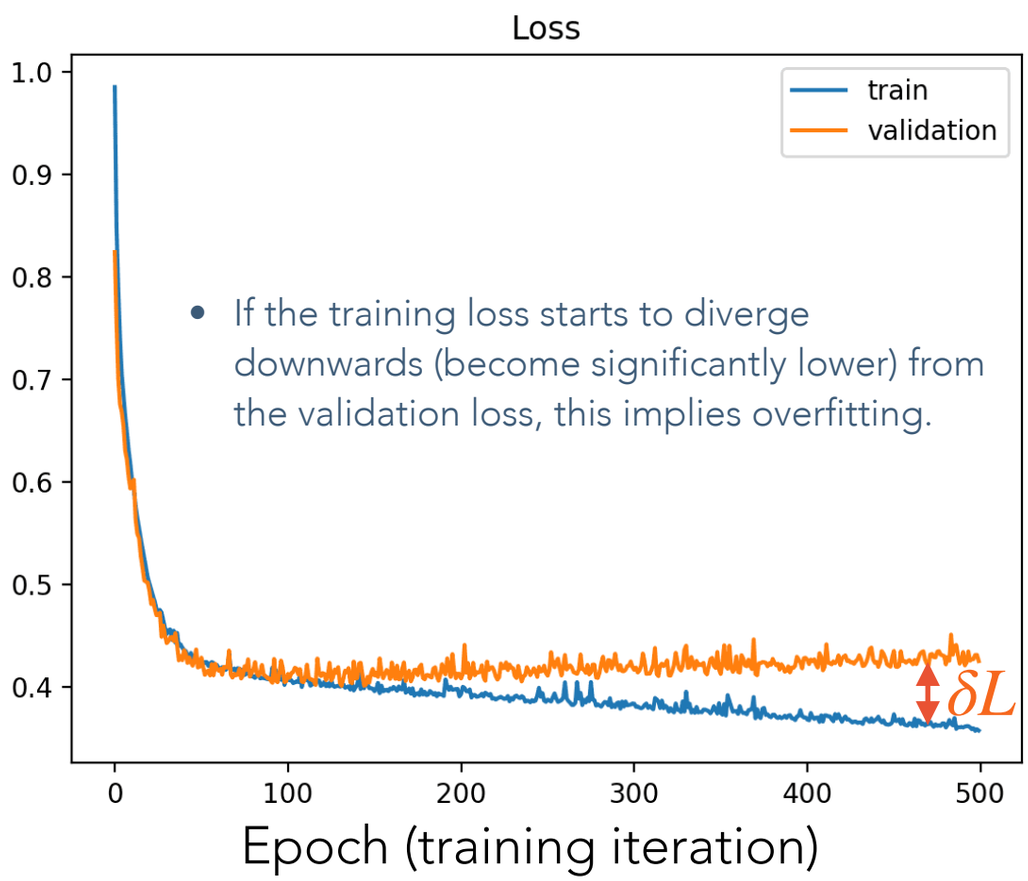




Over Fitting: How mitigate it?

- Monitor the model's generalisability during training by introducing **a validation step** using a **validation sample**.
  - Before training, set aside 5% to 20% of your dataset to be **a validation sample**.
  - At regular intervals in the training (at the end of each epoch), evaluate the **current model** (evaluate the loss function) **on the validation sample**.
  - Monitor using the **learning curve**.

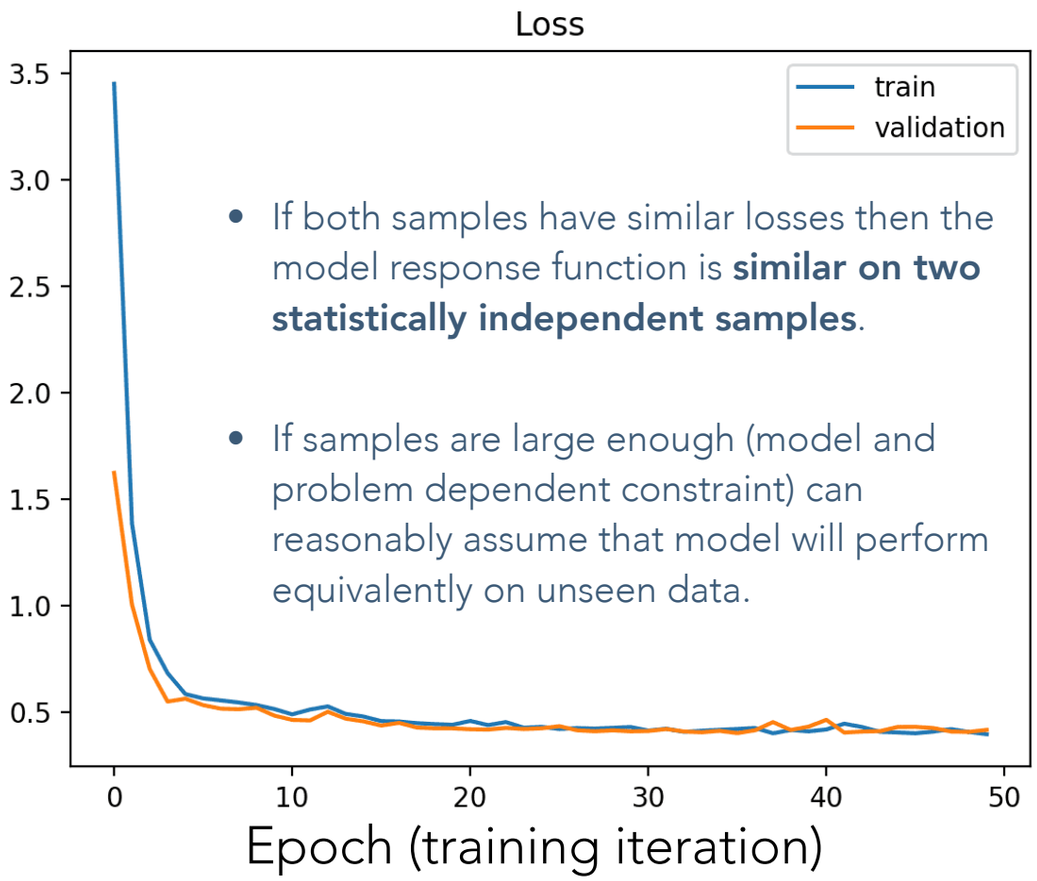


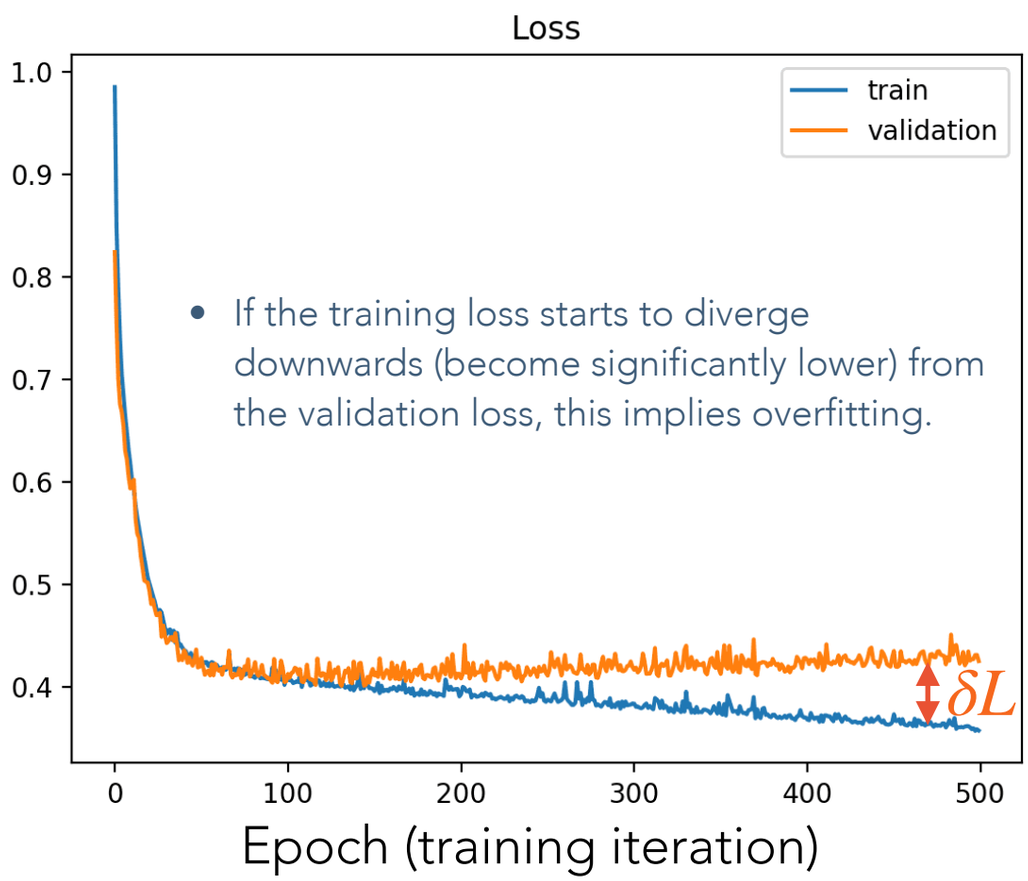


Terminate training if the loss difference δL becomes larger than some value.

Over Fitting: How mitigate it?

- Monitor the model's generalisability during training by introducing **a validation step** using a **validation sample**.
  - Before training, set aside 5% to 20% of your dataset to be **a validation sample**.
  - At regular intervals in the training (at the end of each epoch), evaluate the **current model** (evaluate the loss function) **on the validation sample**.
  - Monitor using the **learning curve**.





Adapt the model to be more robust against overfitting.

Over Fitting: How mitigate it?

- Can use **weight regularisation**.
  - Involves adding a penalty term to the loss function which **gets larger as model complexity grows**.

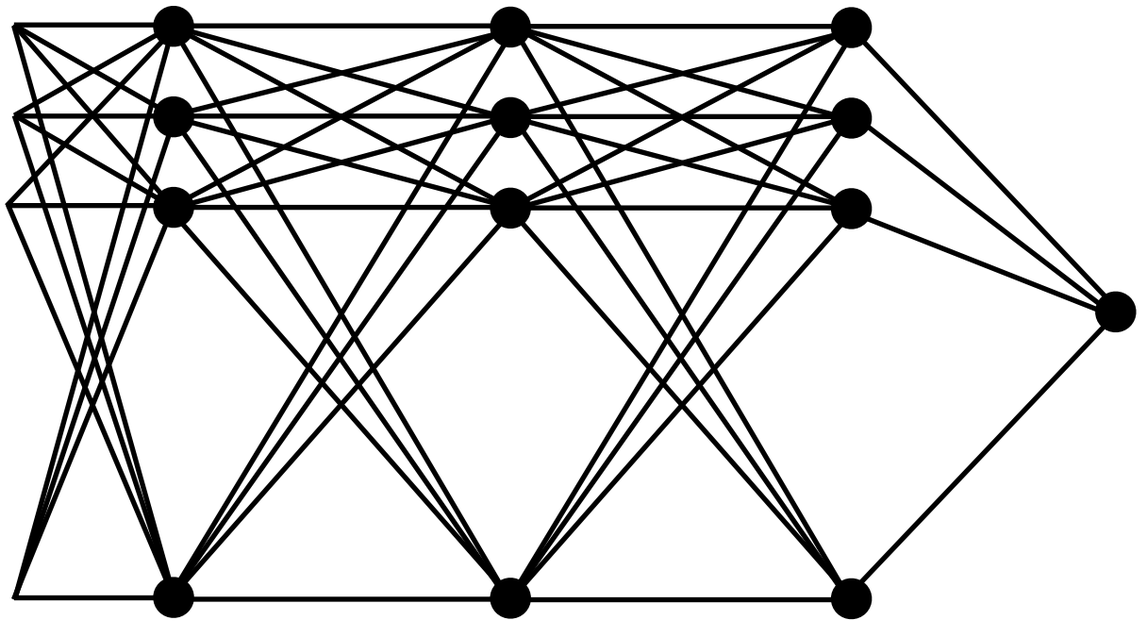
- Consider a quartic polynomial model $y = a + bx + cx^2 + dx^4$.
  - Without any regularisation, the model has **four** parameters (degrees of freedom).

- If c and d regularised -> loss function is penalised if c and d get large.

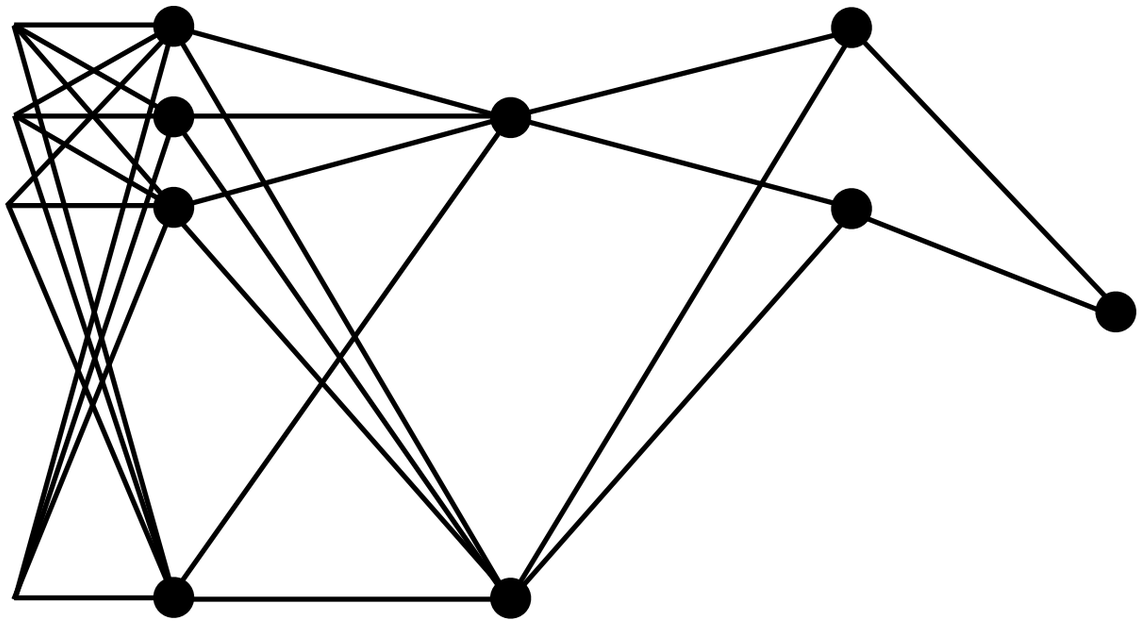
- Model now has somewhere between two and four degrees of freedom -> reduced complexity, reduced chances of overfitting.

Over Fitting: How mitigate it?

- Can use **dropout**.
  - Involves compromising the model randomly in different epochs of the training by **removing nodes** (setting corresponding weights to zero).
  - Controlled by specifying a keep probability for nodes - e.g. 0.4 -> drop 60% of nodes.



Standard neural net



With dropout on 2 hidden layers, keep prob = 0.5

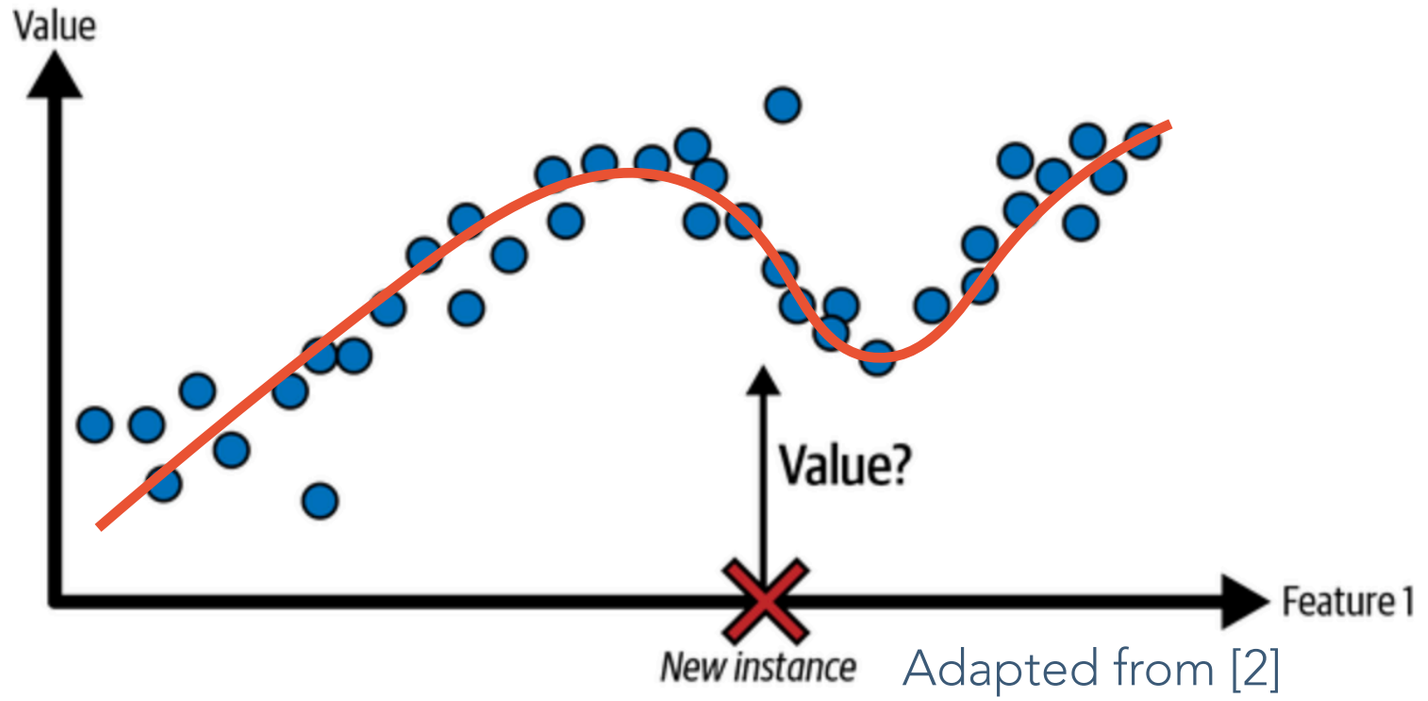
- Why does it work?
  - Neurons cannot coadapt with their neighbouring neurons -> they have to be as useful as possible on their own (company workforce analogy).
  - A unique neural network is being generated at each training step, only a low probability that the same network will be sampled twice -> resulting networking can be seen as an averaging ensemble of these smaller neural networks!

Non-linear Regression

Recap: What is Regression?

- Predicting a target **numeric value** (**response**) given a set of features.
  - House prices given number of bedrooms.
  - Time to run a marathon given resting heart rate and blood pressure.

- Obtain a model to **approximate a function** to go from a set of input features to a target feature.



Adapted from [2]

Approximating a Function

- Let's assume we know that the relationship between an input feature x and target feature y is given by the function $y = f(x) = \sin(x)$.

- What model can we use to approximate this function?

Approximating a Function

- Let's assume we know that the relationship between an input feature x and target feature y is given by the function $y = f(x) = \sin(x)$.

- What model can we use to approximate this function?

Can use a polynomial - but which one?

$$y = a_1 x$$

$$y = a_1 x + a_2 x^2$$

$$y = a_1 x + a_2 x^2 + a_3 x^3$$

$$y = a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4$$

- Each model is an approximation of the function we are interested in.

- For some region of x the approximation will be good (accurate prediction) for others it will be bad (inaccurate prediction).

- As with all approximations, there are conditions associated with the **validity of a given model**.

Approximating a Function

- Let's assume we know that the relationship between an input feature x and target feature y is given by the function $y = f(x) = \sin(x)$.
- What model can we use to approximate this function?

Can use a polynomial - but which one?

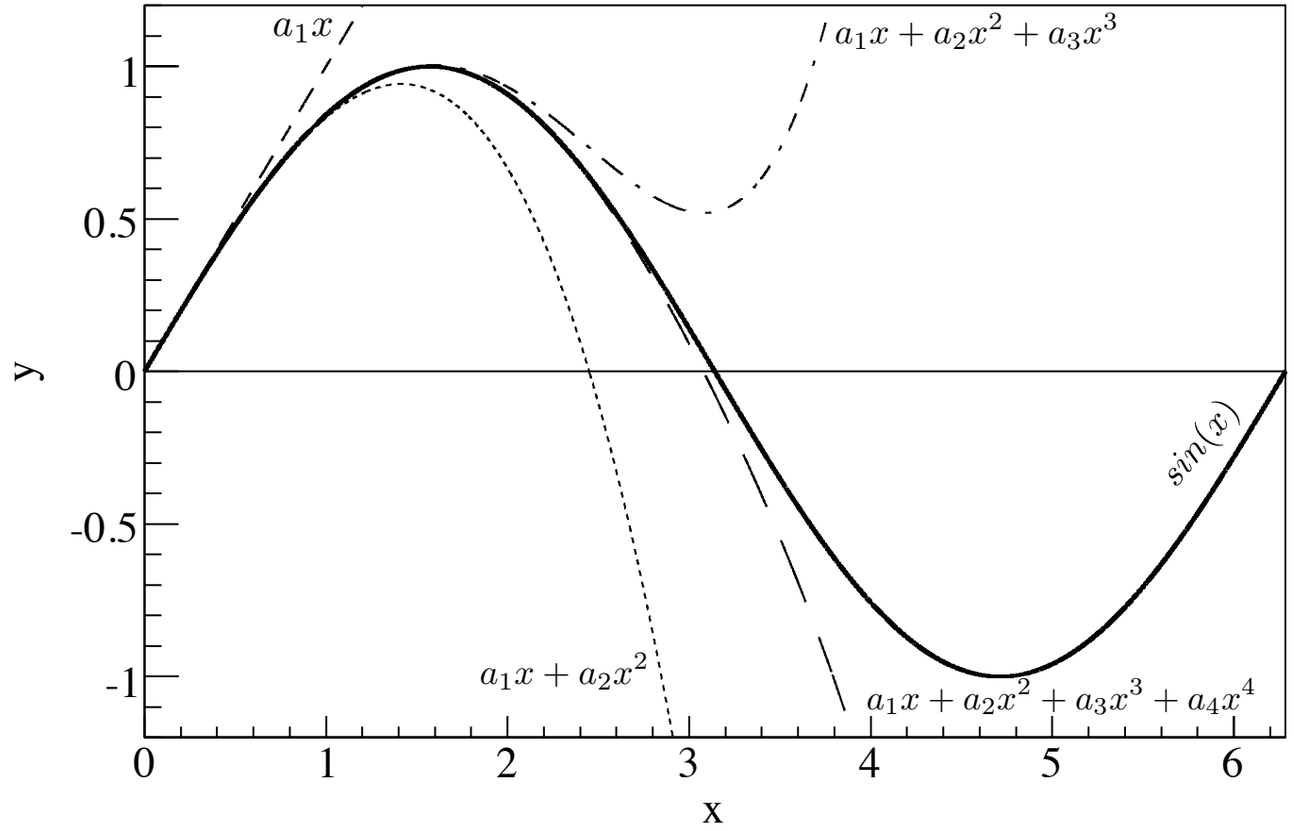
Range of validity increases with the complexity of the model.

$$y = a_1 x$$

$$y = a_1 x + a_2 x^2$$

$$y = a_1 x + a_2 x^2 + a_3 x^3$$

$$y = a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4$$



Approximating a Function

- Let's assume we know that the relationship between an input feature x and target feature y is given by the function $y = f(x) = \sin(x)$.
- What model can we use to approximate this function?

Can use a polynomial - but which one?

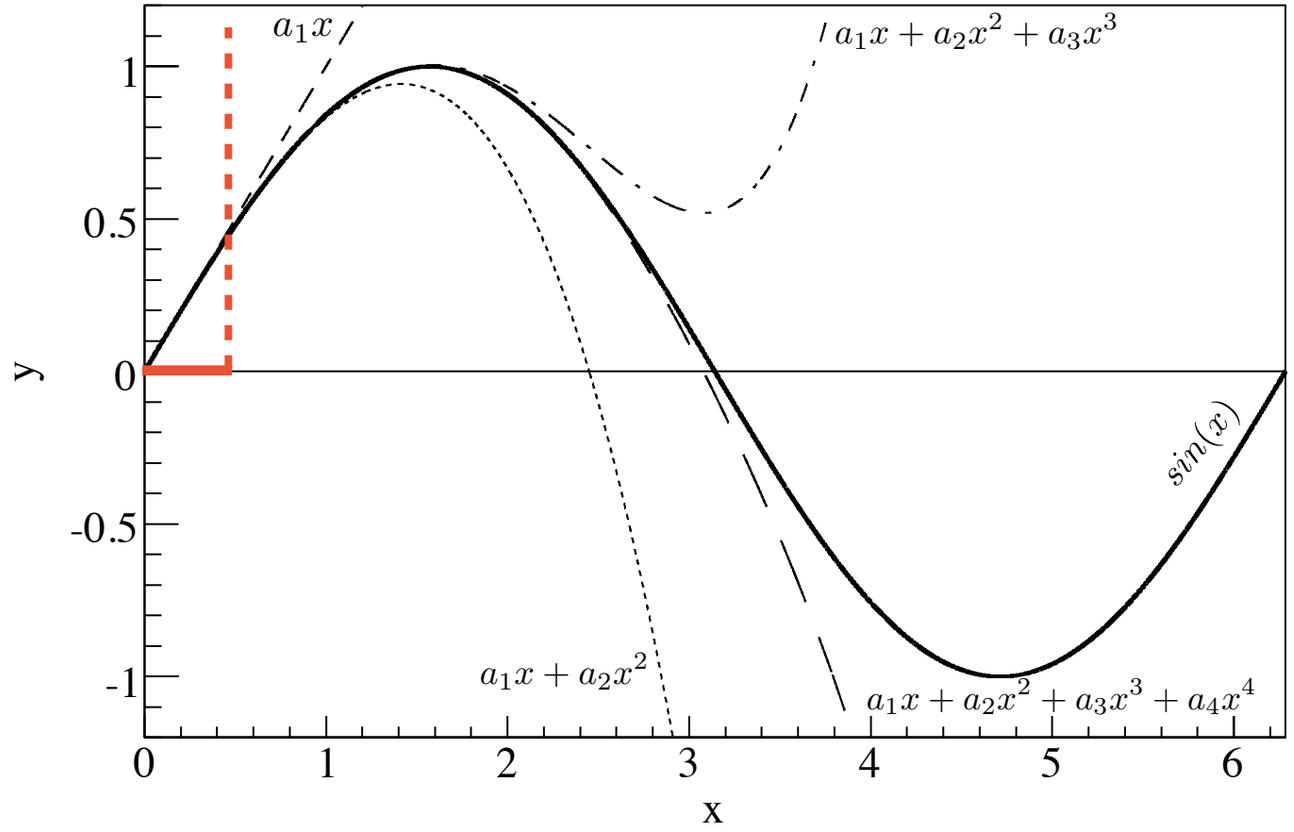
Range of validity increases with the complexity of the model.

$y = a_1 x$

$y = a_1 x + a_2 x^2$

$y = a_1 x + a_2 x^2 + a_3 x^3$

$y = a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4$



Approximating a Function

- Let's assume we know that the relationship between an input feature x and target feature y is given by the function $y = f(x) = \sin(x)$.
- What model can we use to approximate this function?

Can use a polynomial - but which one?

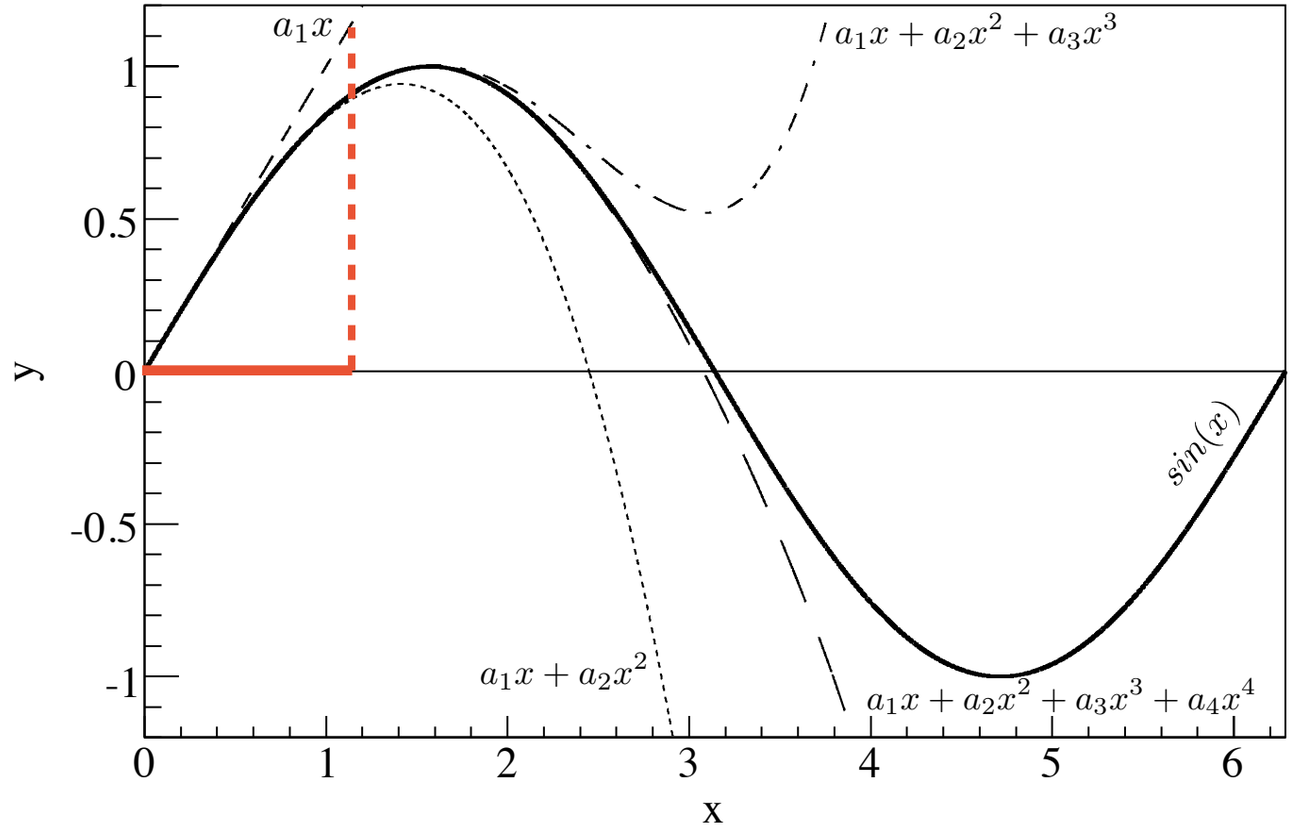
$$y = a_1 x$$

$$y = a_1 x + a_2 x^2$$

$$y = a_1 x + a_2 x^2 + a_3 x^3$$

$$y = a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4$$

Range of validity increases with the complexity of the model.



Approximating a Function

- Let's assume we know that the relationship between an input feature x and target feature y is given by the function $y = f(x) = \sin(x)$.
- What model can we use to approximate this function?

Can use a polynomial - but which one?

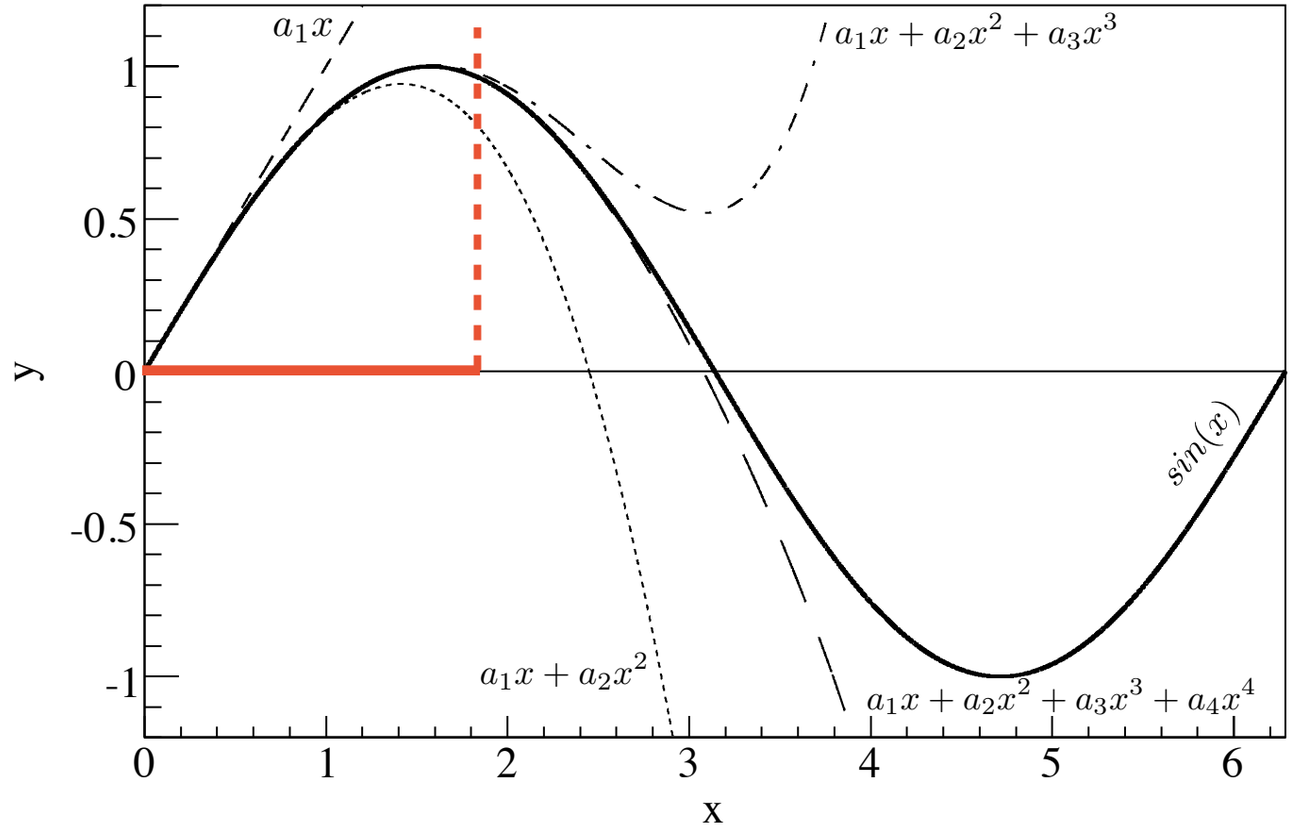
Range of validity increases with the complexity of the model.

$$y = a_1 x$$

$$y = a_1 x + a_2 x^2$$

$$y = a_1 x + a_2 x^2 + a_3 x^3$$

$$y = a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4$$



Approximating a Function

- Let's assume we know that the relationship between an input feature x and target feature y is given by the function $y = f(x) = \sin(x)$.
- What model can we use to approximate this function?

Can use a polynomial - but which one?

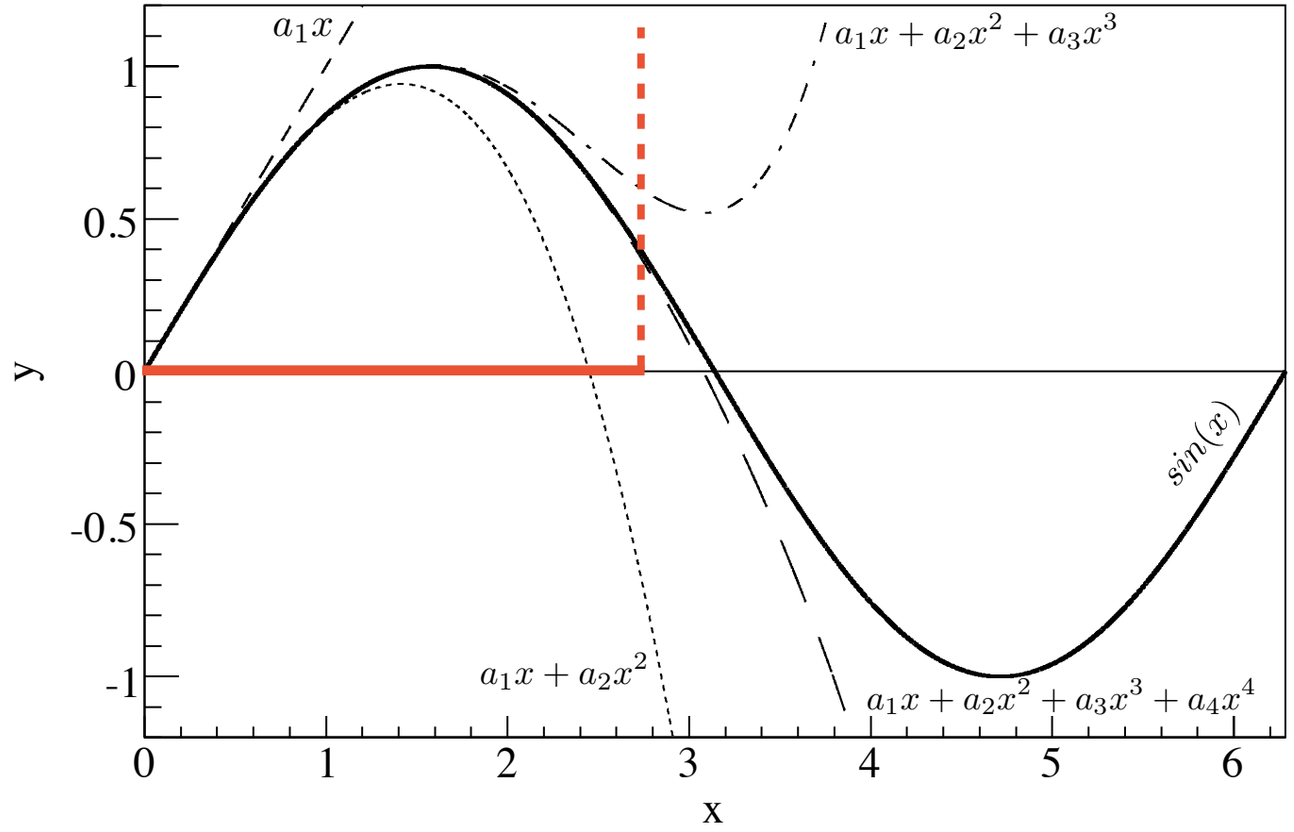
$$y = a_1 x$$

$$y = a_1 x + a_2 x^2$$

$$y = a_1 x + a_2 x^2 + a_3 x^3$$

$$y = a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4$$

Range of validity increases with the complexity of the model.



Approximating a Function

- Let's assume we know that the relationship between an input feature x and target feature y is given by the function $y = f(x) = \sin(x)$.

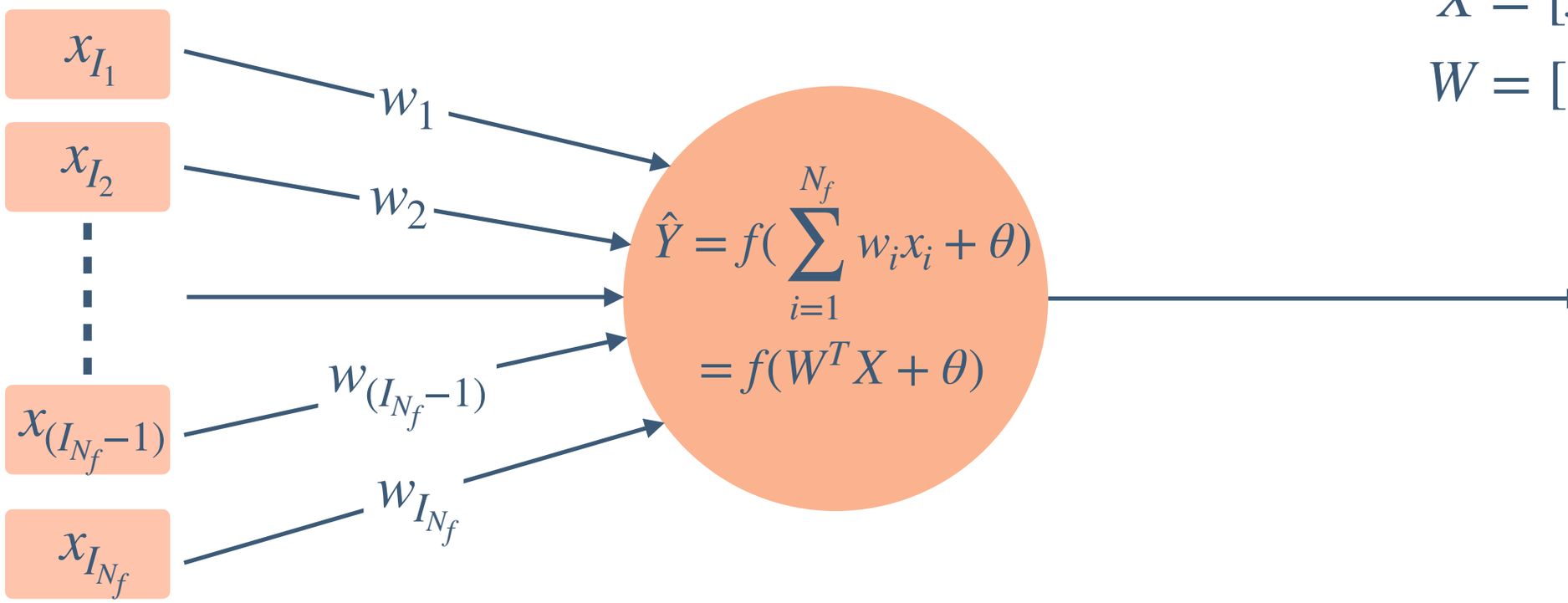
- What model can we use to approximate this function?

Can use a polynomial - but which one?

A more complex model can provide a better approximation more complex (non-linear) function.

Neural Networks & Regression

- Yesterday used a single neuron with a linear activation function to do linear regression.



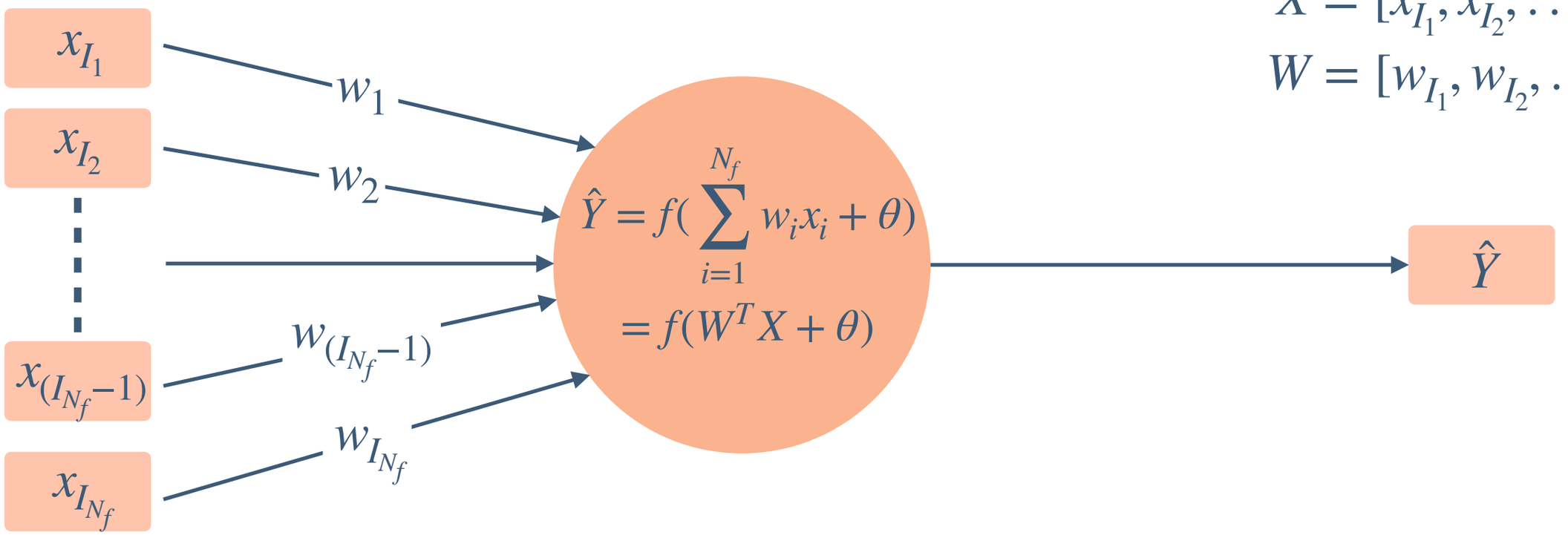
$$X = [x_{I_1}, x_{I_2}, \ldots, x_{I_{N_f}}]$$
$$W = [w_{I_1}, w_{I_2}, \ldots, w_{I_{N_f}}]$$

$$\hat{Y} = \hat{Y}(f(W^T X + \theta))$$
$$= \hat{Y}(w_1 x_{I_1} + \ldots + w_1 x_{I_{N_f}} + \theta)$$

Analogous to $y = a_1 x$

Neural Networks & Regression

- Can extend to generalised linear model where activation function is not linear, e.g Leaky ReLU.



$$X = [x_{I_1}, x_{I_2}, \ldots, x_{I_{N_f}}]$$
$$W = [w_{I_1}, w_{I_2}, \ldots, w_{I_{N_f}}]$$

$$\hat{Y} = \hat{Y}(f(W^T X + \theta))$$
$$= \hat{Y}(f(w_1 x_{I_1} + \ldots + w_1 x_{I_{N_f}} + \theta))$$

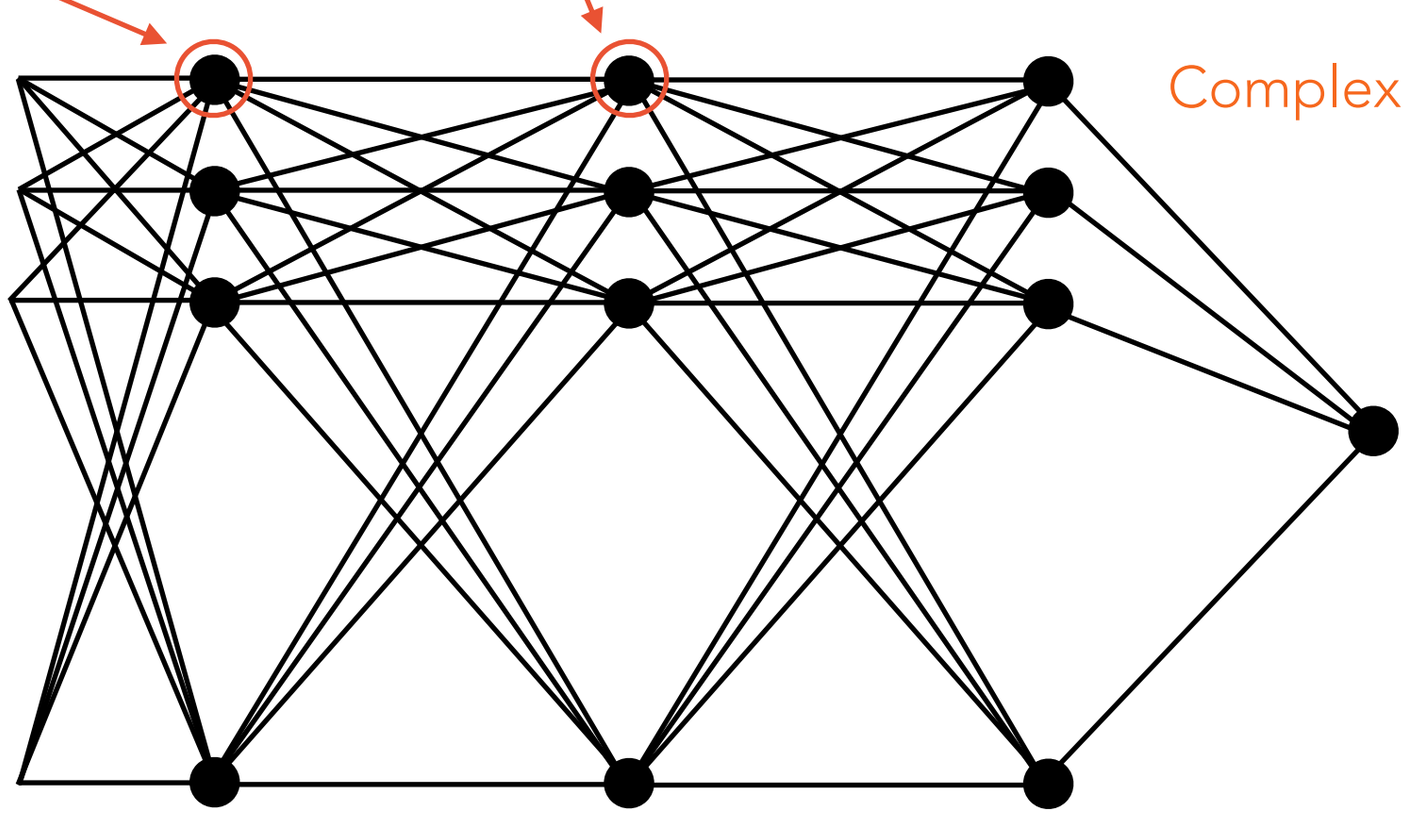
Adds some non-linearity complexity.

Neural Networks & Regression

- In neural networks, each hidden layer adds more non-linearity and more complexity.

$$\hat{Y}_{N_1 L_1} = f_{L_1}(W^T_{N_1 L_1} X_{N_1 L_1} + \theta_{N_1 L_1})$$

$$\hat{Y}_{N_1 L_2} = f_{L_2}(W^T_{N_1 L_2} X_{N_1 L_2} + \theta_{N_1 L_2})$$



Complex function of non-linear functions

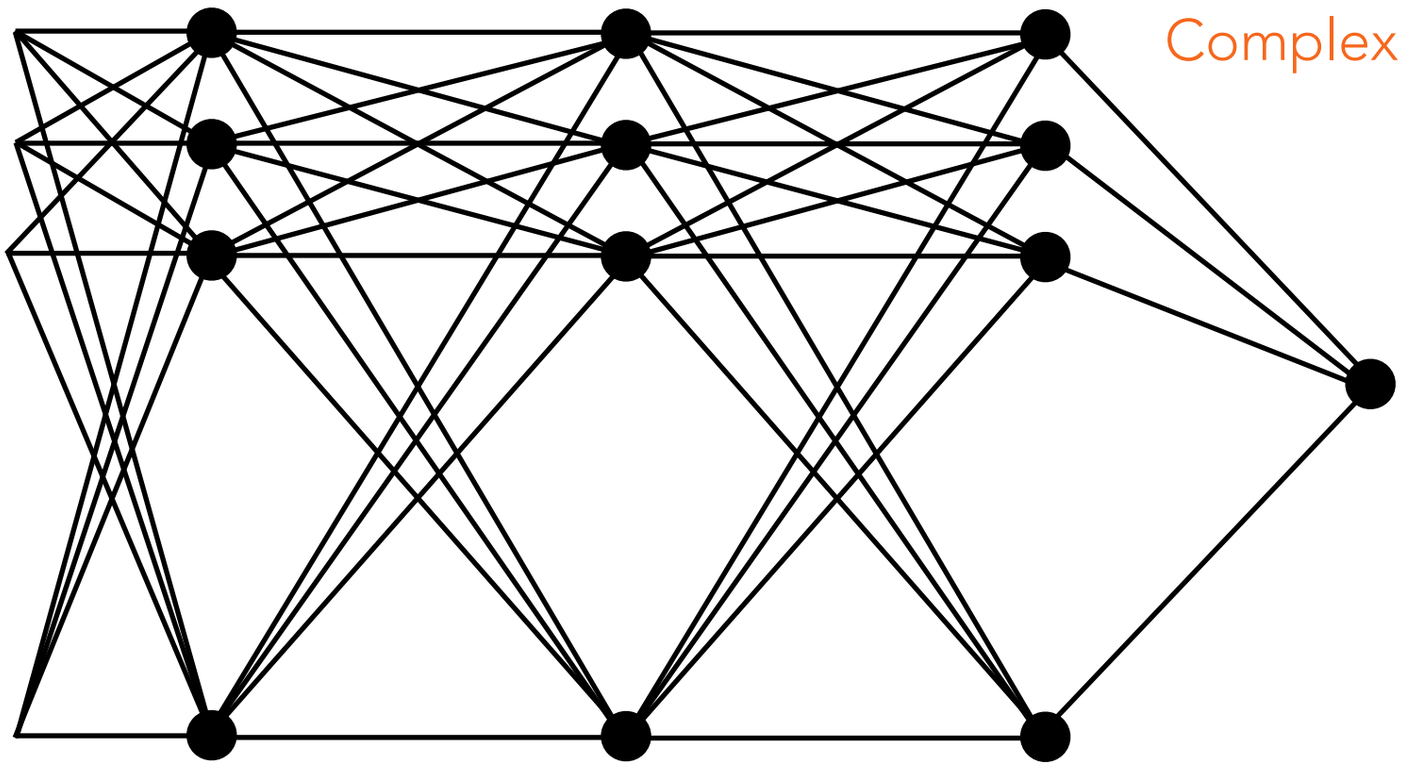
$$\hat{Y} = \hat{Y}(f_{L_1}(f_{L_2} \dots))$$

Neural Networks & Regression

- Neural networks can be thought of as non-linear regression functions -> can model complex non-linear relationships!

- If there are not enough training instances to determine model parameters, even a complicated neural network can provide a bad approximation for a function.

- Larger the model -> larger training sample needed!



Complex function of non-linear functions

$$\hat{Y} = \hat{Y}(f_{L_1}(f_{L_2} \dots))$$

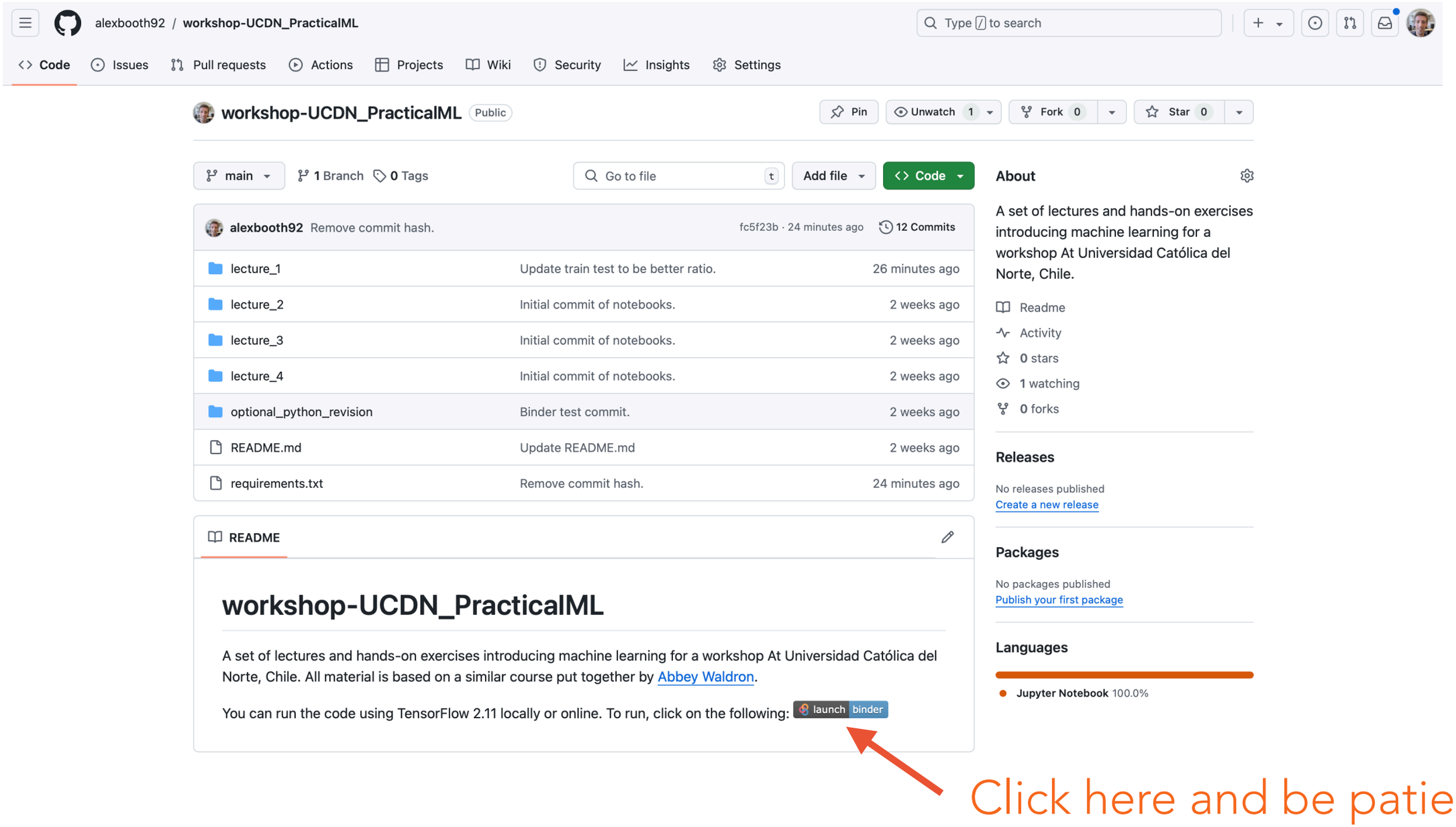
Non-Linear Regression: Today's Notebook

- Learned how a network with a single neuron can be extended to a deep neural network to solve a non-linear regression problem.
- Discussed how to train such a network, limiting over fitting by using dropout.
- Going to train a neural network to approximate a parabola. Will investigate:
  - Impacts of changing training dataset (validation split, batch size).
  - Effects of using dropout.
  - Changing the architecture of the network.

Accessing the Notebooks

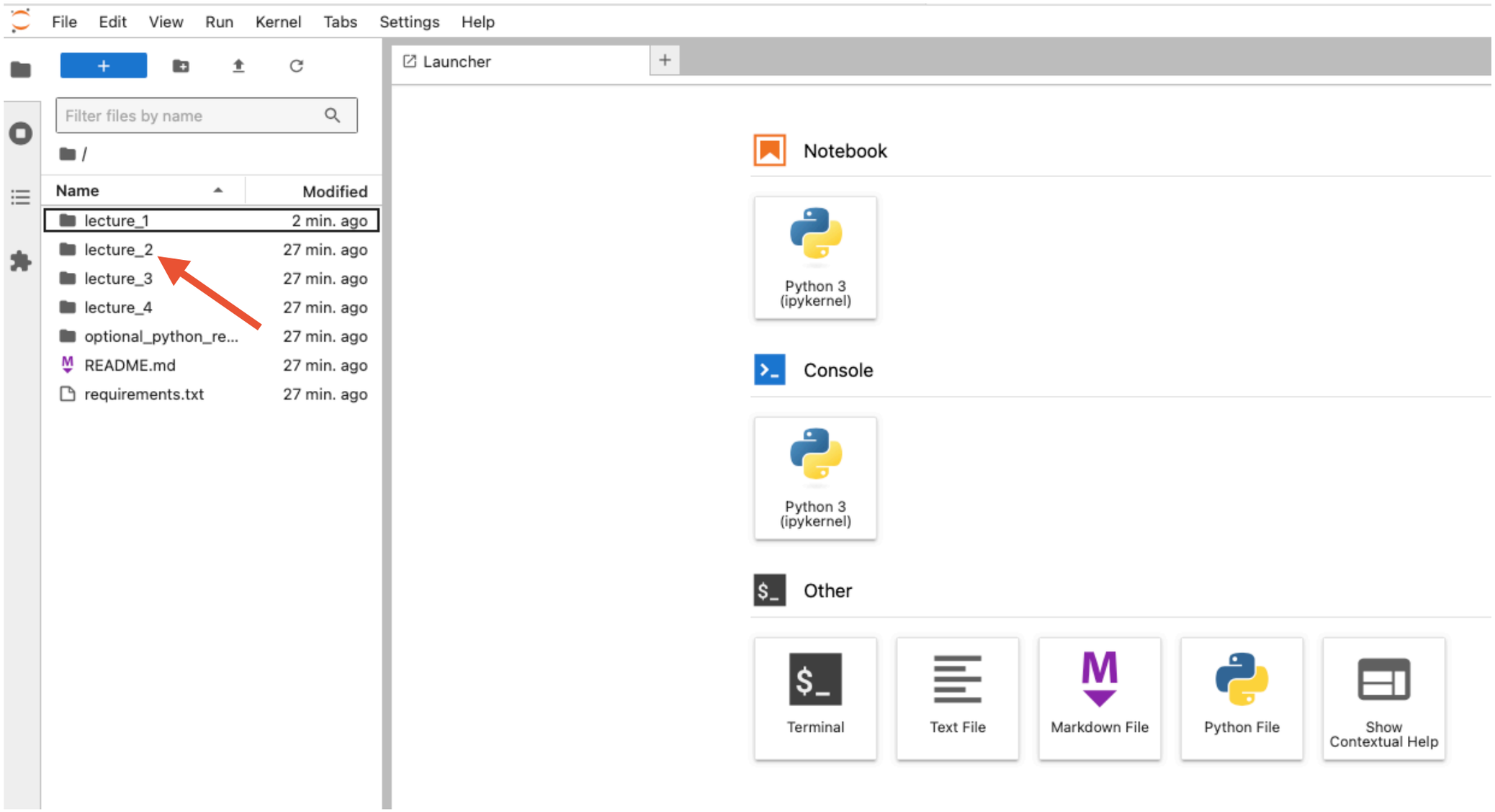
Follow this link:

https://github.com/alexbooth92/workshop-UCDN_PracticalML.git



Click here and be patient!

Accessing the Notebooks



Have Fun!

Queen Mary
University of London

References

[1] Kingma and Ba, Proc. 3rd Int Conf. Learning Representations, San Diego, 2015

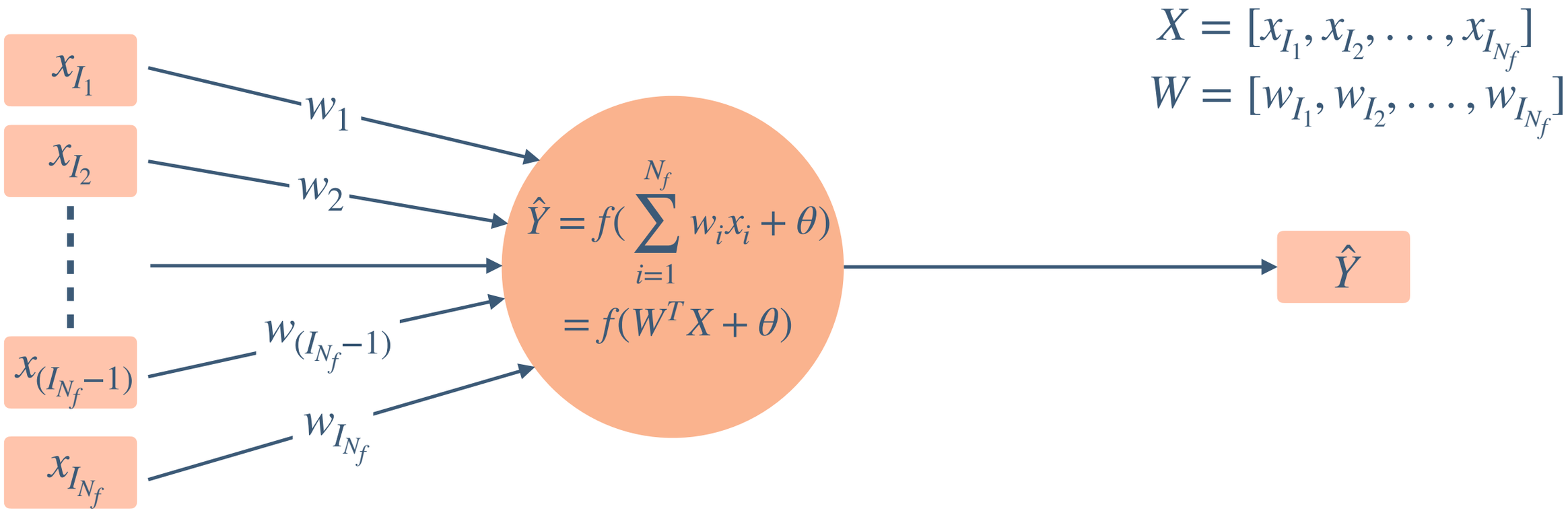
[2] A. Géron. Hands on Machine Learning with Scikit-Learn & TensorFlow, 2017.

[3] https://medium.com/@karenovna.ak/part-ii-evaluating-a-predictive-model-cross-validation-and-bias-and-variance-tradeoff-9874b836cd2e

Back-up

Queen Mary
University of London

How Many Parameters?



- Consider a single perceptron taking N_f **input features**.

- This model has $N_f + 1$ parameters (N_f weights and 1 bias).

Important language:

- If activation function f is a **linear** (nothing done to weighted sum of inputs, like in yesterday's notebook) it's a **linear model**.
- If activation function f is **non-linear** (e.g a step function) it's a **generalised linear model**.