

## Introduction to Machine Learning

placeholder

# 1 Regression

## 1.1 Supervised Learning

**Supervised Learning** is the task of 'learning' a function relationship, based on a given set of inputs/outputs.

Some terminology:

$x \in \mathbb{R}^d$	Inputs (Attributes/Covariates)
$\phi(x) \in \mathbb{R}^p$	Features
$y \in \mathbb{R}$	Outputs (Targets/Labels)
$D = \{(x_i, y_i)\}_{i=1}^n$	Training Set
$D'$	Test Set
$f : \mathbb{R}^p \rightarrow \mathbb{R}$	Predictor (Model)
$l(f(x), y)$	Loss

**Machine Learning Pipelines** can often be classified using:

$F$	Function Class
$L(f)$	Training Loss
	Optimization Method

The function class  $F$  is a set of parametrized functions. We are looking for the  $f \in F$  that minimizes  $L(f)$ .

### D. Training Loss

$$L(f) := \frac{1}{n} \sum_{i=1}^n l(f(x_i), y)$$

## 1.2 Multiple Linear Regression

**Multiple Linear Regression** directly uses the  $x \in \mathbb{R}^d$ .

Here,  $F_{\text{affine}} = \{f(x) = w^\top x + w_0 \mid w \in \mathbb{R}^d, w_0 \in \mathbb{R}\}$ .

**Rmk.** Why are we using linear functions instead?

Any estimator  $f \in F_{\text{affine}}$  can be rewritten as  $f((x, 1)) = (w, w_0)^\top \cdot (x, 1)$ , thus we can augment the inputs  $x \mapsto (x, 1)$  and instead search in  $F_{\text{linear}} = \{f(x) = \hat{w}^\top x \mid \hat{w} \in \mathbb{R}^{d+1}\}$

## Loss Functions

**D. Squared Loss**  $l(f(x), y) := (f(x) - y)^2$

Most common Loss Function, but sensitive to outliers.

**D. Absolute Loss**  $l_{\text{abs}}(f(x), y) := |f(x) - y|$

Less sensitive to outliers, but not differentiable.

**D. Huber Loss**

$$l_{\text{huber}}(f(x), y) := \begin{cases} \frac{1}{2}(f(x) - y)^2 & |f(x) - y| \leq \delta \\ \delta(|f(x) - y| - \frac{1}{2}\delta) & |f(x) - y| > \delta \end{cases}$$

Using parameter  $\delta$ , the penalization of outliers can be controlled

**Assymmetric Loss:** In some cases it is desirable to penalize overestimation harder than underestimation, or vice versa.

**D. Quantile Loss**

$$l_\tau(f(x), y) := \tau \max\{y - f(x), 0\} + (1 - \tau) \max\{f(x) - y, 0\}$$

Using parameter  $\tau$ , over/underestimation can be penalized

## Linear Regression

To find  $\hat{f} := \arg \min_{f \in F_{\text{linear}}} L(f)$  we just look for  $w \in \mathbb{R}^d$ .

$$\hat{w} := \arg \min_{w \in \mathbb{R}^d} L(f_w) = \frac{1}{n} \sum_{i=1}^n \underbrace{(y_i - w^\top x_i)^2}_{l(f(x_i), y_i)}$$

A natural abuse of notation here is  $L(w) := L(f_w)$ .

This can be rewritten in matrix notation:

$$\sum_{i=1}^n (y_i - w^\top x_i)^2 = \|y - Xw\|^2$$

The factor  $\frac{1}{n}$  is irrelevant for Optimization, it doesn't depend on  $w$

So we find the usual problem:

$$\hat{w} = \arg \min_{w \in \mathbb{R}^d} \|y - Xw\|^2$$

The solution is a stationary point, so:

$$\nabla_w \|y - Xw\|^2 = 2X^\top(Xw - y) = 0$$

Which yields the **Normal Equation** from linear algebra.

$$X^\top X \hat{w} = X^\top y$$

## 2 Classification

In regression, we search an  $\hat{f} : \mathbb{R}^d \rightarrow \mathbb{R}$ , i.e.  $y, \hat{y} \in \mathbb{R}$ .  
In classification, we want  $\hat{y} \in \mathcal{Y} \subset \mathbb{R}$ , s.t.  $\mathcal{Y}$  is discrete.

### 2.1 Binary Classification

We generally use  $\mathcal{Y} = \{+1, -1\}$  and set  $\hat{y} = \text{sgn}(\hat{f}(x))$ .  
So, a linear classifier where  $\hat{f}(x) = w^\top x$  takes the form:

$$x \mapsto \begin{cases} 1 & w^\top x > 0 \\ -1 & w^\top x < 0 \end{cases}$$

**D. Decision Boundary**  $\{x \in \mathbb{R}^d \mid \hat{f}(x) = 0\}$

Like in regression, using features is again possible.

### 2.2 Surrogate Loss

We'd like to reuse the loss minimization from regression.  
A natural metric for accuracy is simply checking if  $\hat{y} = y$ .

**D. Zero-One Loss**

$$l_{0-1}(\hat{y}, y) := \mathbb{I}_{\hat{y} \neq y} = \begin{cases} 1 & \hat{y} \neq y \\ 0 & \hat{y} = y \end{cases}$$

We could try minimizing this:

$$\sum_{(x,y) \in \mathcal{D}} l_{0-1}(\hat{y}, y) = \sum_{(x,y) \in \mathcal{D}} \mathbb{I}_{f_w(x) \cdot y < 0}$$

Unfortunately,  $l_{0-1}$  is non-continuous and non-convex.  
We introduce *surrogate loss* to still apply GD.

Note how  $\mathbb{I}_{\hat{y} \neq y} = \mathbb{I}_{\hat{y} \cdot y < 0}$ , so  $l_{0-1}$  only depends on  $z := \hat{y} \cdot y$ .  
We thus define losses over  $z$ , that are cont. and convex.

**D. Surrogate Loss**

$$l_{\text{exp}} = e^{-z} \quad l_{\log} = \log(1 + e^{-z})$$

A notable difference is that  $l'_{\text{exp}}$  is unbounded,  
while  $l'_{\log} = \frac{1}{1+e^z} \in (-\frac{1}{2}, -1)$  for  $z < 0$ .  
This is better for outliers, thus  $l_{\log}$  is usually preferred.

## 2.3 Logistic Regression

We assume  $w_0 = 0$

We try to minimize  $l_{\log} = \log(1 + e^{-z})$ , so:

$$L(w) = \frac{1}{n} \sum_{i=1}^n l_{\log}(z_i) = \frac{1}{n} \sum_{i=1}^n \log\left(1 + e^{-\overbrace{y_i \cdot w^\top x_i}^{z_i}}\right)$$

Assume  $\{x_i, y_i\}_{i=1}^n$  is linearly separable, i.e.

$$\exists w \in \mathbb{R}^d : \underbrace{y_i \cdot w^\top x_i}_{z_i} > 0 \quad \forall i \leq n$$

Then there are multiple valid decision boundaries. Additionally,  $L(w)$  is then convex.

the distance  $x_0$  to the decision boundary is:  $\|x_0\|_2 \cdot |\cos(\theta)|$ .  
 $\theta$  between  $w, x_0 \in \mathbb{R}^d$

$$\|x_0\|_2 \cdot |\cos(\theta)| = \|x_0\|_2 \cdot \frac{|w^\top x_0|}{\|w\|_2 \cdot \|x_0\|_2} = \frac{|w^\top x_0|}{\|w\|_2}$$

Note if  $w$  is a unit-vector, this is just  $|w^\top x_0|$

**D. Margin**  $\text{margin}(w) := \min_{1 \leq i \leq n} y_i \cdot w^\top x_i$

## 2.4 Solutions

**D. Maximum Margin Solution**

$$w_{\text{MM}} := \max_{\|w\|_2=1} \min_{1 \leq i \leq n} (y_i \cdot w^\top x_i)$$

If  $\mathcal{D}$  is linearly separable, this is convex.

**Rmk.** Also called *hard-margin SVM*, assuming lin.-sep. data

**D. Support Vector Machine**

$$w_{\text{SVM}} := \min_{w \in \mathbb{R}^d} \|w\|_2 \quad \text{s.t.} \quad y_i \cdot w^\top x_i \geq 1 \quad \forall i \leq n$$

Solving these problems is actually equivalent, up to scaling:

**L.**  $\frac{w_{\text{SVM}}}{\|w_{\text{SVM}}\|_2} = w_{\text{MM}}$  (This also holds for the case  $w_0 \neq 0$ )

By relaxing the SVM constraints, we can use the SVM problem on lin.-insep. data too:

$$y_i \cdot w^\top x_i \leq 1 \quad \rightarrow \quad y_i \cdot w^\top x_i \leq 1 - \zeta$$

$\zeta = (\zeta_1, \dots, \zeta_n)$  s.t.  $\zeta_i \geq 0$

**D. Soft-margin SVM**

$$w_{\text{SM}} = \min_{w \in \mathbb{R}^d, \zeta \in \mathbb{R}^n} \left( \|w\|^2 + \lambda \sum_{i=1}^n \zeta_i \right) \quad \text{s.t.} \quad \begin{cases} y_i \cdot w^\top x_i \geq 1 - \zeta_i \\ \zeta_i \geq 0 \quad \forall i \leq n \end{cases}$$

$\lambda$  (hyperparam.) intuitively controls how much a violation is penalized

Another perspective: The optimal  $\zeta_i$  are:

$$\zeta_i = \begin{cases} 1 - y_i \cdot w^\top x_i & \text{if } y_i \cdot w^\top x_i \leq 1 \\ 0 & \text{else} \end{cases}$$

So the problem can be formulated without  $\zeta$  too:

**D.  $l_2$ -penalized Hinge Loss Optimization**

$$\min_{w \in \mathbb{R}^d} \left( \|w\|^2 + \lambda \underbrace{\sum_{i=1}^n \max(0, 1 - y_i \cdot w^\top x_i)}_{\text{Hinge Loss}} \right)$$

## 2.5 Gradient Descent for Classification

In practice, instead of explicitly solving  $w_{\text{SVM}}$  or  $w_{\text{MM}}$ , GD is usually applied on a diff.-able convex surrogate loss.

### 2.5.1 On linearly separable data

Assuming  $\{x_i, y_i\}_{i=1}^n$  is lin. separable,  $L(w) = \frac{1}{n} \sum_{i=1}^n l_{\log}(z_i)$  is convex, but no global optimum exists. Using GD,  $L(w)$  will approach 0, but the iterates  $\{w^t \mid t \in \mathbb{N}\}$  diverge. However,  $w^t$  may converge *in direction*, and interestingly:

**Th.** GD converges to  $w_{\text{MM}}$  for lin.-sep. data (log. loss)

$$\lim_{t \rightarrow \infty} \frac{w^t}{\|w^t\|} = w_{\text{MM}}$$

On  $L(w)$  (logistic regression),  $\mu = 1$

### 2.5.2 On linearly inseparable data

Assuming  $\{x_i, y_i\}_{i=1}^n$  is strictly lin. inseparable, i.e.

$$\forall w \neq 0: \exists i \leq n: y_i \cdot w^\top x_i < 0$$

**Th.** GD converges on lin.-insep. data (log. loss)

$$\exists \hat{w} \in \mathbb{R}^d: \lim_{t \rightarrow \infty} w^t = \hat{w}$$

On  $L(w)$  (logistic regression),  $\mu = \frac{4}{\sigma_{\max}^2(X)}$

**Rmk.** Only holds for  $l_{\log}$ . In general, this is a hard problem.

## 2.6 Multiclass Classification

What if  $|\mathcal{Y}| > 2$ ? E.g.  $\mathcal{Y} = \{\text{cat}, \text{dog}, \text{fish}\}$

Idea: Train  $K := |\mathcal{Y}|$  classifiers  $\hat{f}_1, \dots, \hat{f}_K \in F$ .

Why not one  $\hat{f}$ ? E.g. discretize further:  $1 \mapsto \text{cat}, 2 \mapsto \text{dog}, 3 \mapsto \text{fish}$ .

Problem: this assignment suggests cats are closer to dogs than fish.

We can then predict the class using these  $\hat{f}_k$ :

$$\hat{y}(x) = \arg \max_{1 \leq k \leq K} \hat{f}_k(x)$$

This leads to one decision boundary per class.

### D. One-vs-Rest Training

Train each model separately by relabeling for each  $\hat{f}_k$ :

1. Define  $\mathcal{D}_k = \{x_i, \tilde{y}_i\}$  where  $\tilde{y}_i := \begin{cases} -1 & y_i = k \\ 1 & y_i \neq k \end{cases}$
2. Run binary classification on  $\mathcal{D}_k$  to get  $\hat{f}_k$

This leads to  $K$  classification problems, which might be slow

Another way to reuse the existing methodology is to use a new loss:

### D. Cross-Entropy Loss

$$l_{\text{ce}}(\hat{f}_1(x), \dots, \hat{f}_K(x), y) = -\log\left(\frac{\exp(\hat{f}_y(x))}{\sum_{k=1}^K \exp(\hat{f}_k(x))}\right)$$

$$y \in \{1, \dots, K\}, \hat{f}_i(x) \in \mathbb{R}$$

$l_{\text{ce}}$  encourages the *true class*  $\hat{f}_{y_i}(x_i)$  to be the largest  $\hat{f}_k(x_i)$ .

**Rmk.** For  $K = 2$ , if we use  $\mathcal{Y} = \{+1, -1\}$  then  $l_{\text{ce}} = l_{\log}$ .

The parametrized optimization problem then is:

$$\min_{w_1, \dots, w_K \in \mathbb{R}^d} \left( \sum_{i=1}^n l_{\text{ce}}(f_w(x_i), y_i) \right)$$

Here,  $w \in \mathbb{R}^{d \times K}$  is a matrix:  $w = (w_1, \dots, w_K)$

This then yields  $\hat{f}_k = f_{\hat{w}_k}$

**Rmk.** These methods may lead to very different decision boundaries!

## 2.7 Generalization

First, a lot of assumptions:

1. Inputs  $X \in \mathcal{X}$  come from a prob. distribution  $\mathbb{P}_X$
2. Training & Test set sampled i.i.d. from same distribution  
Note that in general, this is rarely true.
3. There exists a ground-truth  $y^*$
4. The observed labels are noisy:  $(y \mid x) = \epsilon \cdot y^*(x)$   
A *multiplicative* noise model, unlike lin.-reg. :  $\mathcal{Y} = f^*(X) + \epsilon$
5.  $\epsilon$  is also from a prob. distribution  $\mathbb{P}_\epsilon$   
Not necessarily indep. from  $x$ !

Focusing on  $y^*(x) \in \{+1, -1\}$ , we set  $\epsilon \in \{+1, -1\}$ .

Intuitively:  $\epsilon$  just flips the label

This allows defining a Joint-Distribution

$$\mathbb{P}[x, y] = \mathbb{P}_x[x] \cdot \mathbb{P}[y \mid x]$$

### 2.7.1 Evaluation

An intuitive metric to check is proximity to  $y^*$ :

Which can be done using the 0-1-loss

$$l(\hat{y}(x), y^*(x)) = \mathbb{I}_{\hat{y}(x) \neq y^*(x)}$$

Now, we can define the expected classification error:

$$\mathbb{E}_X[l(\hat{y}(x), y^*(x))] = \mathbb{E}_X[\mathbb{I}_{\hat{y}(x) \neq y^*(x)}] = \mathbb{P}[\hat{y}(x) \neq y^*(x)]$$

We can't compute or estimate this, since we don't have  $y^*$ . However, we can find an estimate of  $\mathbb{E}_{X, Y}[l(\hat{y}(X), Y)]$  using the observed  $X, Y$ .

Why is this useful? It approximates the generalisation error:

### D. Generalisation Error (0-1-Loss)

$$L(\hat{f}; \mathbb{P}_{X, Y}) = \mathbb{E}_{X, Y}[l(\hat{f}(X), Y)] = \mathbb{P}_{X, Y}(\hat{y} \neq y)$$

We can empirically evaluate this on a test set:

$$\frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{(x, y) \in \mathcal{D}_{\text{test}}} \mathbb{I}_{\hat{y}(x) \neq y^*(x)}$$

## 2.8 Hypothesis Testing

Classical statistical methods can also be used.

### D. Asymmetric Errors

Misclassifications may be weighted differently.

Mistakenly classifying  $x$  with  $y^*(x) = 1$  as 2, may be worse than 3.

For binary classification, we may label:

$$+1 \mapsto \text{"Positive"} \quad -1 \mapsto \text{"Negative"}$$

This leads to the notion of confusion matrices.

	$y = -1$	$y = +1$
$\hat{y} = -1$	TN	FN (Type II)
$\hat{y} = +1$	FP (Type I)	TP

### D. Empirical Measure

For an event  $A \subset \mathcal{X} \times \{+1, -1\}$

$$\mathbb{P}_n[A] := \frac{1}{n} \sum_{i=1}^n \mathbb{I}_{(x_i, y_i) \in A}$$

$$\mathcal{D} = \{(x_i, y_i) \mid i \leq n\} \subset \mathcal{X} \times \{+1, -1\}$$

$\mathbb{P}_n[A]$  is the percentage of  $(x, y) \in \mathcal{D}_{\text{train}}$  that belong to  $A$ .

L.  $\mathbb{P}_n[A]$  is an estimate of  $\mathbb{P}_{X,Y}[A]$ :

$$\lim_{n \rightarrow \infty} \mathbb{P}_n[A] = \mathbb{P}_{X,Y}[A] \quad (\text{Law of large numbers})$$

### Rmk. Asymmetric Loss in binary classification

We can now weigh FP, FN differently in the 0-1-error:

$$\frac{c_{FN}}{|\{x \mid y = +1\}|} \underbrace{\sum_{(x,y), y=1} \mathbb{I}_{\hat{y} \neq -1}}_{\#FN} + \frac{c_{FP}}{|\{x \mid y = -1\}|} \underbrace{\sum_{(x,y), y=-1} \mathbb{I}_{\hat{y} \neq +1}}_{\#FP}$$

Here  $c_{FP}, c_{FN}$  are the weights for penalization

Generally, reducing FP errors increases FN errors, and vice versa.

## 2.9 ROC Curves

Rmk. A side-effect of using  $\hat{y}(x) = \text{sign} \hat{f}(x)$  is that the magnitude  $|\hat{f}(x)|$  can be interpreted as *confidence*.

We can set:

$$\hat{y}_\tau(x) = \text{sign}(\hat{f}(x) - \tau) = \begin{cases} +1 & \text{if } \hat{f}(x) > \tau \\ -1 & \text{if } \hat{f}(x) < \tau \end{cases}$$

Now  $\tau$  can be used to penalize FP ( $\tau > 0$ ) or FN ( $\tau < 0$ ).

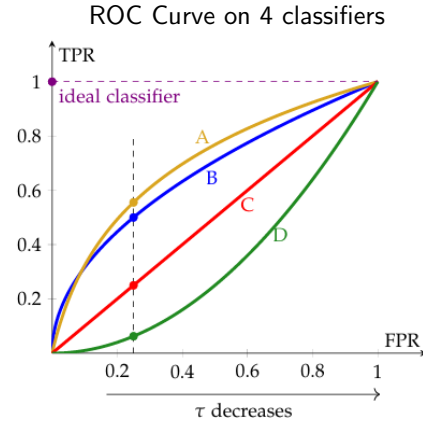
Note how this way, we don't modify the Optimization problem.

What if we don't know which FP/TP rate is desired?

Formally: which  $\tau$  should be used?

### D. ROC Curve (Receiver Operating Characteristic)

Plots TP Rate against FP Rate for different  $\tau$ .



Introduction to Machine Learning (2026), p. 160

Rmk. **How to read this?** A straight line is equivalent to random guessing, anything above is better.  $\tau$  isn't directly included in the curve, but it follows from the definition that  $\tau$  decreases as the FP rate increases.

How can we measure performance independent of  $\tau$ ?

### D. AUROC (Area under ROC)

AUROC is 1 for the ideal classifier, and always in  $[0, 1]$ .

## 3 Kernels

**Motivation:** Regression using feature maps  $\phi: \mathbb{R}^d \rightarrow \mathbb{R}^p$ :

$$\min_{w \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^n l(y_i, w^\top \cdot \phi(x_i))$$

What if computing/storing  $\phi(x)$  is expensive/infeasible?

e.g. if  $p$  is large, or infinite

Rmk. To store a poly.  $p(x): \mathbb{R}^d \rightarrow \mathbb{R}$  with  $\deg(p) = m$  we require  $p = \mathcal{O}(d^m)$  features. Storing  $n$  data points requires  $\mathcal{O}(nd^m)$  memory. Not good.

### 3.1 Kernelization

By constraining  $w$  to  $\text{span}(\Phi^\top) \subset \mathbb{R}^p$  we can drastically improve memory usage. Since we know a minimizer exists here, we don't "lose anything".

### D. Kernelization

1. **Reparametrization:** We assume  $w = \Phi^\top \alpha$  (i)

2. **Loss via Inner Products:** Observe:

$$f(x) = w^\top \phi(x) \stackrel{(i)}{=} (\Phi^\top \alpha)^\top \phi(x) = \sum_{i=1}^n \alpha_i (\phi(x_i)^\top \phi(x))$$

Note:  $x_i$  only appears in *inner products*  $\phi(x_i)^\top \phi(x_j)$

3. **Replace Inner Products:** We define:

$$k: \begin{cases} \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R} \\ k(x, x') = \phi(x)^\top \phi(x') \end{cases} \quad K: \begin{cases} K \in \mathbb{R}^{n \times n} \\ K_{ij} = k(x_i, x_j) \end{cases}$$

Now, we can reformulate the optimization problem:

$$\min_{\alpha \in \mathbb{R}^n} \frac{1}{n} \sum_{i=1}^n l\left(y_i, \sum_{j=1}^n \alpha_j k(x_i, x_j)\right) = \min_{\alpha \in \mathbb{R}^n} \frac{1}{n} \sum_{i=1}^n l(y_i, (K\alpha)_i)$$

By storing  $K \in \mathbb{R}^{n \times n}$  instead of  $\phi(x) \in \mathbb{R}^p$  for  $i = 1, \dots, n$ , the memory usage is reduced:  $\mathcal{O}(np) \rightarrow \mathcal{O}(n^2)$ .

### 3.2 The Kernel Trick

Using  $k$ , the computation time is still  $\mathcal{O}(n^2p)$  if

$$k(x_i, x_j) = \phi(x_i)^\top \phi(x_j)$$

So let's replace  $k$  with a simple function, which guarantees the existence of some  $\phi$  (which we never calculate).

**Rmk.** Since we only *implicitly* specify  $\phi$  via  $k$ , we can use  $\phi$  s.t.  $p = \infty$  now.

**D. Kernel Function**  $k : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$

- $k$  is symmetric:  $\forall x, x' : k(x, x') = k(x', x)$
- $k$  is PSD:  $\forall n \in \mathbb{N}, \forall (x_1, \dots, x_n) \in \mathbb{R}^d$ :

$$K = \begin{bmatrix} k(x_1, x_1) & \dots & k(x_1, x_n) \\ \vdots & \ddots & \vdots \\ k(x_n, x_1) & \dots & k(x_n, x_n) \end{bmatrix} \text{ is PSD}$$

**Th. Kernels guarantee existence of  $\phi$**

If  $k$  is a kernel, there exists a Hilbert Space  $(\mathcal{H}, \langle \cdot, \cdot \rangle_{\mathcal{H}})$  s.t.

$$\exists \phi : \mathbb{R}^d \rightarrow \mathcal{H} \text{ s.t. } k(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{H}} \quad \forall x, x' \in \mathbb{R}^d$$

$\mathcal{H}$  may be, for example,  $\mathbb{R}^p$  with  $\|\cdot\|_2$ .

**L. Properties of Kernels**

- Composed feature maps are Kernels

$$\left. \begin{array}{l} \phi : \mathbb{R}^d \rightarrow \mathbb{R}^p \\ \psi : \mathbb{R}^d \rightarrow \mathbb{R}^p \end{array} \right\} k(x, x') = \langle \psi(\phi(x)), \psi(\phi(x')) \rangle$$

- Kernels can be added in 2 ways, yielding a kernel

$$\begin{aligned} \text{(i)} \quad k((x, y), (x', y')) &= k_1(x, x') + k_2(y, y') \\ \text{(ii)} \quad k(x, x') &= k_1(x, x') + k_2(x, x') \end{aligned}$$

- Kernels can be multiplied in 2 ways, yielding a kernel

## 4 Neural Networks

**Motivation:** So far, when looking for  $\hat{f}(x)$  the form was  $\hat{f}(x) = w^\top x$ , or  $\hat{f}(x) = w^\top \phi(x)$ . Note how the features  $x, \phi(x)$  are predetermined. Why not learn them?

**New Optimization Problem:**

The new joint-optimization problem, for  $w$  and  $\phi$ :

$\Theta$  is a set of parameters for  $\phi$

$$\hat{w} = \arg \min_{w \in \mathbb{R}^m, \Theta \in \mathbb{R}^{m \times d}} \left( \frac{1}{n} \sum_{i=1}^n l(w^\top \phi(x_i; \Theta), y_i) \right)$$

Where  $\phi(x, \Theta) = (\phi_1(x; \theta_1), \dots, \phi_m(x; \theta_m))$ .

$\theta_i$  is the  $i$ th row of  $\Theta$ , i.e.  $\theta_i := (\Theta)_{i,:}$ .

More compact, in terms of  $\Theta$ , which combines  $w, \phi$ :

$$\Theta^* := \arg \min_{\Theta} (L(\Theta; \mathcal{D})) = \arg \min_{\Theta} \left( \frac{1}{n} \sum_{i=1}^n l(\Theta; x_i, y_i) \right)$$

$\Theta$  may also encapsulate  $w, \phi$  for multiple layers, depending on definition

### 4.1 Definitions

**D. Activation Function**

We set  $\phi_i(x; \theta_i) = \psi(\theta_i^\top x)$ ,  $\psi$  is the activation function.

$\theta_i \in \mathbb{R}^d, \psi : \mathbb{R} \rightarrow \mathbb{R}$

**Notation** More concisely,  $\phi(x; \Theta) = \psi(\Theta x)$

Activation Function	Definition
Identity	$\psi(z) = z$
Sigmoid	$\psi(z) = \frac{1}{1+e^{-z}}$
Hyperbolic tangent	$\psi(z) = \tanh(z)$
Rectified Linear Unit (ReLU)	$\psi(z) = \max(0, z)$

**D. Artificial Neural Network**

The output functions of the above problem take the form:

$$f(x; w, \theta) = \sum_{j=1}^m w_j \psi(\theta_j^\top x)$$

**Rmk.** Also called Multi-Layer Perceptron (MLP)

### What is happening here?

Explaining the calculation steps for such an  $f$  naturally leads to the common pictorial depiction of neural networks.

- $x = (x_1, \dots, x_n) \in \mathbb{R}^d$  (Input Vector)
- $z = \Theta x$  (Linear transformation)
- $h_i = \psi(z_i)$  (Activation function)
- $f(x) = \sum_{j=1}^m w_j h_j$  (Output)

**D. Hidden Layer**  $h = \psi(z)$

**D. Bias Term**  $b \in \mathbb{R}^m$

Needed, as  $f$  might not pass through origin. Similar to using  $F_{lin}$  in regression, these can also be added by augmenting the input & hidden layers.

**Does this work at all?**

Yes, for most functions this does work.

**D. Sigmoidal Function**

$$\sigma(t) \text{ s.t. } \begin{cases} \sigma : \mathbb{R} \rightarrow \mathbb{R} \\ \lim_{t \rightarrow \infty} = 1 \text{ and } \lim_{t \rightarrow -\infty} = 0 \end{cases}$$

**Th. Universal Approximation Theorem**

$\hat{f}$ , that uniformly approximates  $f$ , exists and takes this form:

$$\hat{f}(x) = \mathbf{W}^{(2)} \psi(\mathbf{W}^{(1)} x + b)$$

$f : [0, 1]^d \rightarrow \mathbb{R}$  continuous,  $\psi$  sigmoidal  
 $\mathbf{W}^{(1)} \in \mathbb{R}^{m \times d}, \mathbf{W}^{(2)} \in \mathbb{R}^{1 \times m}, m \in \mathbb{N}$

Note how  $m$  could be very large.

$m$  can intuitively be understood as the "width" of the ANN

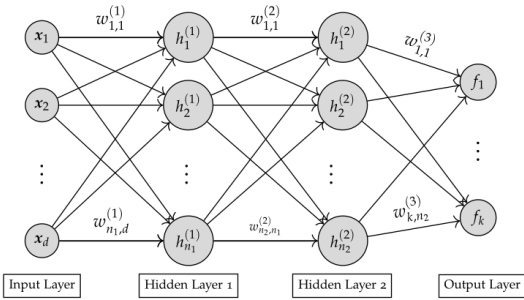
## D. Fully Connected Neural Network

More complex ANNs might have:

1. More hidden layers
2. Multiple outputs
3. Different activation functions across layers

These are called *fully connected*, since every node in a layer is connected to every node in the adjacent layers.

There are also more complex architectures.



*Introduction to Machine Learning (2026), p. 183*

**Notation** Weights:  $\mathbf{W}^{(i)} := [w_{k,l}^{(i)}]$ , Biases:  $b_k^{(i)}$   
and  $\Theta = (\mathbf{W}^{(1)}, \dots, \mathbf{W}^{(L)}, b^{(1)}, \dots, b^{(L)})$  (All parameters)

$w_{k,l}^{(i)}$ : "Weight at layer  $i$  to node  $k$  from node  $l$ "

## 4.2 Forward Propagation

How can we make predictions, i.e. how can  $\hat{f}$  be evaluated?

### D. Forward Propagation

This is just the computation for 1-layer ANN generalized for  $L$  layers

#### Algorithm 1: Forward Propagation

```

 $h^{(0)} \leftarrow x;$ 
for  $l = 1, \dots, L$  do
     $z^{(l)} = \mathbf{W}^{(l)} h^{(l-1)} + b^{(l)}$ 
     $h^{(l)} = \psi(z^{(l)})$ 
end
 $f \leftarrow \mathbf{W}^{(L)} h^{(L-1)} + b^{(L)}$ 
return  $f$ 
    
```

## 4.3 Backwards Propagation

How can we get all gradients needed for model training?

### D. Backwards Propagation

**Intuition:** An efficient way to get the gradients is to reuse results from forward prop. and previous steps. This works best when starting at the back, at  $\nabla_{\mathbf{W}^{(L)}} l$ .

**Goal:**  $\nabla_{\mathbf{W}^{(1)}} l, \dots, \nabla_{\mathbf{W}^{(L)}} l, \nabla_{b^{(1)}} l, \dots, \nabla_{b^{(L)}} l$

**Step 1:** Calculate  $\nabla_{\mathbf{W}^{(L)}} l$ , i.e. start from the back.

$$\begin{aligned}
 \nabla_{\mathbf{W}^{(L)}} l &= \frac{\partial l}{\partial \mathbf{W}^{(L)}} \\
 &= \frac{\partial l}{\partial f} \cdot \frac{\partial f}{\partial \mathbf{W}^{(L)}} \quad (\text{Chain Rule}) \\
 &= \frac{\partial l}{\partial f} \cdot \begin{bmatrix} (h^{(L-1)})^\top \\ \vdots \\ (h^{(L-1)})^\top \end{bmatrix} \quad (f = \mathbf{W}^{(L)} h^{(L-1)} + b^{(L)}) \\
 &= \nabla_f l \cdot \begin{bmatrix} (h^{(L-1)})^\top \\ \vdots \\ (h^{(L-1)})^\top \end{bmatrix} \quad \left( \frac{\partial l}{\partial f} = \nabla_f l \right)
 \end{aligned}$$

Notice how  $h^{(L-1)}$  was computed during forward prop.

**Step 2:** Calculate  $\nabla_{\mathbf{W}^{(L-1)}} l$ .

$$\nabla_{\mathbf{W}^{(L-1)}} l = \underbrace{\frac{\partial l}{\partial f}}_{(1)} \cdot \underbrace{\frac{\partial f}{\partial h^{(L-1)}}}_{(2)} \cdot \underbrace{\frac{\partial h^{(L-1)}}{\partial z^{(L-1)}}}_{(3)} \cdot \underbrace{\frac{z^{(L-1)}}{\partial \mathbf{W}^{(L-1)}}}_{(4)} \quad (\text{Chain Rule})$$

1. Already done in Step 1.
2. Already done in forward propagation, equal to  $\mathbf{W}^{(L)}$ :

$$f \stackrel{\text{def}}{=} \mathbf{W}^{(L)} h^{(L-1)} + b^{(L)} \implies \frac{\partial f}{\partial h^{(L-1)}} = \mathbf{W}^{(L)}$$

3. **Not done.** Needs to be calculated:

$$\begin{aligned}
 \frac{\partial h^{(L-1)}}{\partial z^{(L-1)}} &= \frac{\partial \psi(z^{(L-1)})}{\partial z^{(L-1)}} \\
 &= \text{diag}(\psi'(z^{(L-1)})) \\
 &= \begin{bmatrix} \psi'(z_1^{(L-1)}) & 0 & \dots & 0 \\ 0 & \psi'(z_2^{(L-1)}) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \psi'(z_n^{(L-1)}) \end{bmatrix}
 \end{aligned}$$

4. Already done in forward propagation, analogous to step 1.

$$\frac{\partial z^{(L-1)}}{\partial \mathbf{W}^{(L-1)}} = \begin{bmatrix} (h^{(L-2)})^\top \\ \vdots \\ (h^{(L-2)})^\top \end{bmatrix}$$

**Step  $i \leq L$ :** Calculate  $\nabla_{\mathbf{W}^{(L-i)}} l$  Analogous to step 2.

The biases  $\nabla_{b^{(i)}} l$  are analogous.

## 4.4 Optimization

**Problem:** How can we train the model, i.e find  $\Theta^*$ ?

$$\Theta^* := \arg \min_{\Theta} (L(\Theta; \mathcal{D})) = \arg \min_{\Theta} \left( \frac{1}{n} \sum_{i=1}^n l(\Theta; x_i, y_i) \right)$$

**Rmk.**  $L(\Theta; \mathcal{D})$  is generally not convex.  
i.e. local minima, saddle points may exist

**Rmk.**  $\dim(\Theta)$  is the total param. count of NN, may be very large

**Solution:** Gradient Descent (with optimizations)

- Stochastic Gradient Descent  
(Why?  $\dim(\Theta)$  is very large,  $\nabla_{\Theta} l(\Theta; x_i, y_i)$  are expensive)
- Minibatch Gradient Descent  
(Why?  $\mathcal{D}$  may be very large, so there are *many* gradients)

The standard GD update for  $\Theta$  is:

$$\Theta^{t+1} = \Theta^t - \eta_t \cdot \nabla_{\Theta} L(\Theta; \mathcal{D})$$

In Minibatch GD, this becomes:

Where  $\mathcal{S} \subset \{1, \dots, n\}$

$$\Theta^{t+1} = \Theta^t - \eta_t \cdot \nabla_{\Theta^t} \left( \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} l(\Theta^t; x_i, y_i) \right)$$

**Rmk.** An advantage: If  $\Theta^t$  approaches a stat. point (which isn't the global minimum), GD will converge, but MB-GD may not converge.

### 4.4.1 Vanishing & Exploding Gradients

$$\nabla_{\Theta^t} \left( \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} l(\Theta^t; x_i, y_i) \right) = \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} (\nabla_{\Theta^t} l(\Theta^t; x_i, y_i))$$

The terms  $\nabla_{\Theta^t} l(\Theta^t; x_i, y_i)$  are composed of  $\nabla_{\mathbf{W}^{(l)}} l(\mathbf{W}^{(l)}; x_i, y_i)$ .

**Problem:** Optimization might fail if:

$$\left\| \nabla_{\mathbf{W}^{(l)}} l \right\| \rightarrow \infty \quad \text{or} \quad \left\| \nabla_{\mathbf{W}^{(l)}} l \right\| \rightarrow 0$$

**Solution:**  $\left\| \nabla_{\mathbf{W}^{(l)}} l \right\|$  depends linearly on  $\text{diag}(\psi'(z^{(l)}))$ , so the choice of  $\psi$  ( $\psi'$ ) can be used to constrain  $\left\| \nabla_{\mathbf{W}^{(l)}} l \right\|$   
Generally, the gradient follows the behaviour of  $\psi'$

Which features do we want  $\psi$  ( $\psi'$ ) to fulfill?

- $\psi'$  should be fast to calculate
- $\psi'$  should be non-zero (and not get too close)

**Rmk.** The  $\left\| \nabla_{\mathbf{W}^{(l)}} l \right\|$  may still vanish for any  $\psi$ .

### 4.4.2 Random Weight Initialization

$$\begin{aligned} \nabla_{\mathbf{W}^{(l)}} l &= \frac{\partial l}{\partial f} \cdot \frac{\partial f}{\partial h^{(l)}} \cdot \frac{\partial h^{(l)}}{\partial z^{(l)}} \cdot \frac{\partial z^{(l)}}{\partial \mathbf{W}^{(l)}} \\ &= \frac{\partial l}{\partial f} \cdot \frac{\partial f}{\partial h^{(l)}} \cdot \text{diag}(\psi'(z^{(l)})) \cdot \begin{bmatrix} (h^{(l-1)})^{\top} \\ \vdots \\ (h^{(l-1)})^{\top} \end{bmatrix} \end{aligned}$$

So, the gradient  $\left\| \nabla_{\mathbf{W}^{(l)}} l \right\|$  also depends on  $\|h^{(l-1)}\|$ .

For  $l \in \{1, \dots, L\}$

**Problem:** It might be that  $\|h^{(l-1)}\| \rightarrow \infty$  or  $\|h^{(l-1)}\| \rightarrow 0$ .

**Solution:** Set  $h^{(l-1)}$  randomly, bound mean  $\mu$  and var.  $\sigma^2$ .  
There is no generally optimal bound for  $\sigma^2$ , it depends on the NN.

Some useful distributions for common  $\psi$ :