

December 4, 2025

Vincent Van Schependom

## Contents

<b>1</b>	<b>Basics &amp; Feed Forward Networks</b>	<b>3</b>
1.1	Multilayer Perceptron (MLP)	3
1.2	Full network	3
1.3	Probabilistic Inference	3
1.4	Parameter Count	4
1.5	Activation Functions	4
1.6	Universal Approximation Theorem	4
1.7	Other	4
<b>2</b>	<b>Training &amp; Optimization</b>	<b>5</b>
2.1	Loss Functions	5
2.2	Maximum Likelihood Estimation (MLE)	5
2.3	Gradient Descent	6
2.3.1	Stochastic Gradient Descent (SGD)	6
2.3.2	Adam (Adaptive Moment Estimation)	6
2.4	Backpropagation	7
<b>3</b>	<b>Initialization &amp; Regularization</b>	<b>8</b>
3.1	Weight Initialization	8
3.1.1	He-Kaiming Initialization (ReLU)	8
3.1.2	Xavier-Glorot Initialization	8
3.2	Bias-Variance Tradeoff	8
3.3	Regularization Techniques	8
3.3.1	Weight Decay	8
3.3.2	Other Regularization Techniques	9
<b>4</b>	<b>Residual Neural Networks</b>	<b>10</b>
<b>5</b>	<b>Convolutional Neural Networks (CNNs)</b>	<b>11</b>
5.1	Invariance	11
5.2	Convolution Operation	11
5.3	Pooling	11
5.4	Output dimensionality	12
<b>6</b>	<b>Recurrent Neural Networks (RNNs)</b>	<b>13</b>
6.1	MLE for RNNs	13
6.2	Long Short-Term Memory (LSTM)	14
<b>7</b>	<b>Transformers &amp; Attention</b>	<b>15</b>
7.1	Notation and Dimensions	15
7.2	Scaled Dot-Product Attention	15
<b>8</b>	<b>Unsupervised Deep Learning</b>	<b>16</b>
8.1	Autoencoders (AE)	16
8.1.1	Limitation	16

8.2	Variational Autoencoders (VAE)	16
8.2.1	Intractability	16
8.2.2	Amortized Variational Inference	16
8.2.3	Objective: Evidence Lower Bound (ELBO)	16
8.2.4	Reparameterization Trick	17
8.3	Semi-supervised Learning	17

# 1 Basics & Feed Forward Networks

## 1.1 Multilayer Perceptron (MLP)

A feed-forward neural network (FNN) with multiple layers. The output of layer  $l$  is the input to layer  $l + 1$ . Notation follows the slides: superscripts denote layers, subscripts denote components.

Name	Symbol	Dimension	
Input Vector	$\mathbf{x} = \mathbf{h}^{(0)}$	$D^{(0)} \times 1$	Layer computation: $\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}$ $\mathbf{h}^{(l)} = \sigma(\mathbf{z}^{(l)})$
Weights (Layer $l$ )	$\mathbf{W}^{(l)}$	$D^{(l)} \times D^{(l-1)}$	
Bias (Layer $l$ )	$\mathbf{b}^{(l)}$	$D^{(l)} \times 1$	
Pre-activation (Layer $l$ )	$\mathbf{z}^{(l)}$	$D^{(l)} \times 1$	Weight matrix: $\mathbf{W}^{(l)} = \begin{bmatrix} w_{1 \leftarrow 1}^{(l)} & w_{1 \leftarrow 2}^{(l)} & \cdots & w_{1 \leftarrow D^{(l-1)}}^{(l)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{D^{(l)} \leftarrow 1}^{(l)} & w_{D^{(l)} \leftarrow 2}^{(l)} & \cdots & w_{D^{(l)} \leftarrow D^{(l-1)}}^{(l)} \end{bmatrix}$
Activation Function	$\sigma(\cdot)$		
Hidden State (Layer $l$ )	$\mathbf{h}^{(l)}$	$D^{(l)} \times 1$	
Output	$\mathbf{y} = \mathbf{h}^{(L)}$	$D^{(L)} \times 1$	Here, $w_{j \leftarrow i}^{(l)}$ is the weight from input neuron $i$ in layer $l - 1$ to output neuron $j$ in layer $l$ .
Number of Layers	$L$		
Neurons per Layer	$D$		
Distribution params	$\boldsymbol{\theta}$		
Model params	$\phi$		

## 1.2 Full network

We predict the **distribution** (parameters  $\boldsymbol{\theta}$ ) of the labels  $\mathbf{y}$  given the inputs  $\mathbf{x}$  using multi-output model  $\mathbf{f}_\phi(\mathbf{x})$ :

$$\boldsymbol{\theta} = \mathbf{f}_\phi(\mathbf{x}) \rightsquigarrow p(\mathbf{y}|\mathbf{f}_\phi(\mathbf{x})), \quad \phi = \{\mathbf{W}^{(l)}, \mathbf{b}^{(l)}\}_{l=1}^L$$

We usually assume that all  $D$  outputs of  $\mathbf{f}_\phi(\mathbf{x}) = [\mathbf{f}_{\phi,1}(\mathbf{x}), \dots, \mathbf{f}_{\phi,D}(\mathbf{x})]$  are independent:

$$p(\mathbf{y}|\mathbf{f}_\phi(\mathbf{x})) = \prod_{d=1}^D p(y_d|\mathbf{f}_{\phi,d}(\mathbf{x}))$$

Using these distribution parameters, we calculate the distributions:

- Regression (homo-/heteroscedastic):

$$p(\mathbf{y}|\mathbf{f}_\phi(\mathbf{x})) = p(\mathbf{y}|\mu_1, \dots, \mu_k, \sigma_1, \dots, \sigma_k) = \mathcal{N}(\mathbf{y}|\mu_1, \dots, \mu_k, \sigma_1, \dots, \sigma_k)$$

- Classification ( $\mathbf{y} \in \{0, 1\}^K$  one-hot-encoded):

$$p(\mathbf{y}|\mathbf{f}_\phi(\mathbf{x})) = p(\mathbf{y}|\pi_1, \dots, \pi_K) \stackrel{\text{indep.}}{=} \prod_{d=1}^K p(y_d|\pi_d) = \prod_{d=1}^K \pi_d^{y_d}$$

## 1.3 Probabilistic Inference

For learned model parameters  $\hat{\phi}$ , make predictions  $\hat{\mathbf{y}}$  using  $p(\mathbf{y}|\mathbf{f}_{\hat{\phi}}(\mathbf{x}))$ :

- Most probable value:

$$\hat{\mathbf{y}} = \arg \max_{\mathbf{y}} p(\mathbf{y}|\mathbf{f}_{\hat{\phi}}(\mathbf{x}))$$

- Expected value:

$$\hat{\mathbf{y}} = \mathbb{E}_{\mathbf{y} \sim p(\mathbf{y} | f_{\hat{\phi}}(\mathbf{x}))}[\mathbf{y}]$$

- Sample:

$$\hat{\mathbf{y}} \sim p(\mathbf{y} | f_{\hat{\phi}}(\mathbf{x}))$$

## 1.4 Parameter Count

For a network with input dimension  $D^{(0)}$ ,  $K$  hidden layers each with  $D$  neurons, and output dimension  $D^{(L)}$ :

$$D^{(0)} \cdot D + D + K \cdot (D \cdot D + D) + D \cdot D^{(L)} + D^{(L)}$$

Simplified for 1 input and 1 output:

$$3D + 1 + (K - 1)D(D + 1)$$

## 1.5 Activation Functions

Name	Formula	Layer Type
Sigmoid	$\sigma(z) = \frac{1}{1+e^{-z}}$	Hidden or output (binary classification)
Arc-tangent	$\sigma(z) = \arctan(z)$	Hidden
Hyperbolic tangent	$\sigma(z) = \tanh(z)$	Hidden
ReLU	$\sigma(z) = \max(0, z)$	Hidden
Leaky ReLU	$\sigma(z) = \max(\alpha z, z), \alpha \ll 1$	Hidden
Linear	$\sigma(z) = z$	Output
Softmax (Output)	$\sigma(z_d) = \pi_d = \frac{e^{z_d}}{\sum_d e^{z_d}}$	Output (multiclass classification)

## 1.6 Universal Approximation Theorem

A two-layer network with linear outputs can uniformly approximate any continuous function on a compact input domain (compact subset of  $\mathbb{R}^N$ ) to arbitrary accuracy provided the network has sufficiently large number of hidden units.

This is because:

- Pre-activation = piecewise linear
- Number of **linear regions** for 1 input and  $D$  neurons =  $D + 1$
- (only  $D$  of them are independent and 1 is either zero or the sum of all other regions)

## 1.7 Other

Multiple inputs:

- Multiple *outputs*: Joints are in the same place for each neuron
- Multiple *inputs*: Linear regions are convex polytopes in the multidimensional input space
- Shallow networks almost always have  $D > D_{\text{in}}$  and create between  $2^{D_{\text{in}}}$  and  $2^D$  linear regions
- Deep networks with 1 input, 1 output and  $K$  layers of  $D > 2$  hidden units can create a function with up to  $(D + 1)^K$  linear regions

## 2 Training & Optimization

Given dataset  $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$ , calculate mismatch using loss function:

$$L(\phi) = \frac{1}{N} \sum_{i=1}^N \ell(f_{\phi}(\mathbf{x}_i), \mathbf{y}_i)$$

We learn/fit the model by minimizing this loss:

$$\hat{\phi} = \arg \min_{\phi} L(\phi)$$

### 2.1 Loss Functions

Name	Formula	Type
Mean Squared Error (MSE)	$\frac{1}{N} \sum_{n=1}^N \ f_{\phi}(\mathbf{x}_n) - \mathbf{y}_n\ ^2$	Regression
Binary Cross-Entropy	$-\frac{1}{N} \sum_{n=1}^N [y_n \log \pi_n + (1 - y_n) \log(1 - \pi_n)]$	Binary Class.
Categorical Cross-Entropy	$-\frac{1}{N} \sum_{n=1}^N \sum_{d=1}^D y_{nd} \log \pi_{nd}$	Multi-Class
Negative Log-Likelihood (NLL)	$-\frac{1}{N} \sum_{n=1}^N \log p(\mathbf{y}_n   f_{\phi}(\mathbf{x}_n))$	General

### 2.2 Maximum Likelihood Estimation (MLE)

Unless working with time series data, we assume that each data point is i.i.d:

$$p(\mathbf{y}_1, \dots, \mathbf{y}_N | \mathbf{x}_1, \dots, \mathbf{x}_N, \phi) = \prod_{i=1}^N p(\mathbf{y}_i | f_{\phi}(\mathbf{x}_i))$$

Maximising Likelihood is equivalent to minimising NLL, since log is monotonically increasing.  
:

$$\hat{\phi} = \arg \max_{\phi} \prod_{i=1}^N p(\mathbf{y}_i | f_{\phi}(\mathbf{x}_i)) = -\arg \min_{\phi} \sum_{i=1}^N \log p(\mathbf{y}_i | f_{\phi}(\mathbf{x}_i))$$

Find parameters that maximise the probability of the data  $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$  using loss:

$$L(\phi) = - \sum_{i=1}^N \log p(\mathbf{y}_i | f_{\phi}(\mathbf{x}_i))$$

Assuming that dimensions of each  $\mathbf{y}_i$  are independent the parameters:

$$p(\mathbf{y}_i | f_{\phi}(\mathbf{x}_i)) = \prod_{d=1}^D p(y_{id} | f_{\phi}(\mathbf{x}_i))$$

which yields the loss function

$$L(\phi) = - \sum_{i=1}^N \sum_{d=1}^D \log p(y_{id} | f_{\phi}(\mathbf{x}_i))$$

For multiclass classification, we had  $p(\mathbf{y}_i | f_{\phi}(\mathbf{x}_i)) = \prod_{d=1}^D \pi_{id}^{y_{id}}$ , so the **cross-entropy loss** is

$$L(\phi) = - \sum_{i=1}^N \sum_{d=1}^D y_{id} \log \pi_{id}$$

with class probabilities  $\pi \in [0, 1]$ , which sum to 1 ( $\sum_d \pi_{id} = 1$ ):

$$\pi = \begin{bmatrix} \pi_1 \\ \vdots \\ \pi_K \end{bmatrix} = \begin{bmatrix} \text{softmax}(z_1) \\ \vdots \\ \text{softmax}(z_K) \end{bmatrix} = \begin{bmatrix} \frac{e^{z_1}}{\sum_d e^{z_d}} \\ \vdots \\ \frac{e^{z_K}}{\sum_d e^{z_d}} \end{bmatrix}$$

## 2.3 Gradient Descent

Minimize  $L(\phi)$ : initialise  $\phi^{(0)}$  and update iteratively with **learning rate**  $\eta$ :

$$\phi^{(t+1)} = \phi^{(t)} - \eta \nabla_{\phi} \mathcal{L}(\phi^{(t)}), \quad \nabla_{\phi} \mathcal{L}(\phi) = \begin{bmatrix} \frac{\partial \mathcal{L}(\phi)}{\partial \phi^{(1)}} \\ \vdots \\ \frac{\partial \mathcal{L}(\phi)}{\partial \phi^{(D)}} \end{bmatrix} = \begin{bmatrix} \frac{\partial \mathcal{L}(\phi)}{\partial \mathbf{W}^{(1)}} \\ \frac{\partial \mathcal{L}(\phi)}{\partial \mathbf{b}^{(1)}} \\ \vdots \\ \frac{\partial \mathcal{L}(\phi)}{\partial \mathbf{W}^{(L)}} \\ \frac{\partial \mathcal{L}(\phi)}{\partial \mathbf{b}^{(L)}} \end{bmatrix}$$

### 2.3.1 Stochastic Gradient Descent (SGD)

Draw minibatches  $\mathcal{B}_t \subseteq \{1, \dots, N\}$  **without replacement**:

$$\phi^{(t+1)} = \phi^{(t)} - \sum_{i \in \mathcal{B}_t} \frac{\partial \ell_i(\phi^{(t)})}{\partial \phi}$$

where  $\ell_i(\phi)$  is the loss of the  $i$ -th sample  $(\mathbf{x}_i, \mathbf{y}_i)$  and  $L(\phi) = \sum_{i=1}^N \ell_i(\phi)$ .  
A full pass through the dataset is called an **epoch**.

### 2.3.2 Adam (Adaptive Moment Estimation)

Compute first and second moment of gradients:

$$\begin{aligned} \mathbf{m}^{(t+1)} &= \beta \mathbf{m}^{(t)} + (1 - \beta) \nabla_{\phi} \ell_i(\phi^{(t)}) \\ \mathbf{v}^{(t+1)} &= \gamma \mathbf{v}^{(t)} + (1 - \gamma) \nabla_{\phi} \ell_i(\phi^{(t)})^2 \end{aligned}$$

Compensate for initial values close to zero:

$$\tilde{\mathbf{m}}^{(t+1)} = \frac{\mathbf{m}^{(t+1)}}{1 - \beta^{t+1}}, \quad \tilde{\mathbf{v}}^{(t+1)} = \frac{\mathbf{v}^{(t+1)}}{1 - \gamma^{t+1}}$$

Update parameters after normalization by the second moment.  
This way, we take the same step size in each direction (stable).

$$\phi^{(t+1)} = \phi^{(t)} - \eta \frac{\tilde{\mathbf{m}}^{(t+1)}}{\sqrt{\tilde{\mathbf{v}}^{(t+1)} + \epsilon}}$$

where  $\eta$  is the learning rate, and  $\epsilon$  is a small constant to prevent division by zero.

## 2.4 Backpropagation

Used to calculate gradients  $\nabla_{\phi} L(\phi)$ .

First, we define the forward pass for a single layer  $l$ . Let  $\mathbf{h}^{(l-1)}$  be the input to layer  $l$  (where  $\mathbf{h}^{(0)} = \mathbf{x}$ ).

$$\begin{aligned}\mathbf{z}^{(l)} &= \mathbf{W}^{(l)} \mathbf{h}^{(l-1)} + \mathbf{b}^{(l)} \\ \mathbf{h}^{(l)} &= \sigma(\mathbf{z}^{(l)})\end{aligned}$$

The objective is to compute the gradient of the loss  $L(\phi)$  with respect to the parameters  $\mathbf{W}^{(l)}$  and  $\mathbf{b}^{(l)}$ . We apply the chain rule starting from the pre-activation  $\mathbf{z}^{(l)}$ .

$$\boldsymbol{\delta}^{(l)} \equiv \frac{\partial L}{\partial \mathbf{z}^{(l)}}$$

Using the chain rule, we express the gradients for the weights and biases at layer  $l$  in terms of  $\boldsymbol{\delta}^{(l)}$ .

$$\begin{aligned}\frac{\partial L}{\partial \mathbf{W}^{(l)}} &= \frac{\partial L}{\partial \mathbf{z}^{(l)}} \cdot \frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{W}^{(l)}} \\ &= \boldsymbol{\delta}^{(l)} (\mathbf{h}^{(l-1)})^T\end{aligned}$$

$$\begin{aligned}\frac{\partial L}{\partial \mathbf{b}^{(l)}} &= \frac{\partial L}{\partial \mathbf{z}^{(l)}} \cdot \frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}} \\ &= \boldsymbol{\delta}^{(l)}\end{aligned}$$

To compute  $\boldsymbol{\delta}^{(l)}$ , we propagate the error backwards from the next layer  $(l+1)$ . We use the chain rule to expand  $\frac{\partial L}{\partial \mathbf{z}^{(l)}}$ :

$$\boldsymbol{\delta}^{(l)} = \frac{\partial L}{\partial \mathbf{z}^{(l+1)}} \cdot \frac{\partial \mathbf{z}^{(l+1)}}{\partial \mathbf{h}^{(l)}} \cdot \frac{\partial \mathbf{h}^{(l)}}{\partial \mathbf{z}^{(l)}}$$

Substituting the known terms:

$$\begin{aligned}\frac{\partial L}{\partial \mathbf{z}^{(l+1)}} &= \boldsymbol{\delta}^{(l+1)} \\ \frac{\partial \mathbf{z}^{(l+1)}}{\partial \mathbf{h}^{(l)}} &= \mathbf{W}^{(l+1)} \\ \frac{\partial \mathbf{h}^{(l)}}{\partial \mathbf{z}^{(l)}} &= \sigma'(\mathbf{z}^{(l)})\end{aligned}$$

This gives us the recursive formula for backpropagation:

$$\boldsymbol{\delta}^{(l)} = \left( (\mathbf{W}^{(l+1)})^T \boldsymbol{\delta}^{(l+1)} \right) \odot \sigma'(\mathbf{z}^{(l)})$$

## 3 Initialization & Regularization

### 3.1 Weight Initialization

Avoid vanishing/exploding gradients during backprop. Initialize  $\phi_i \sim \mathcal{N}(0, \sigma^2)$ . Below,  $\alpha = 1$  for tanh and  $\alpha = 2$  for ReLU.

#### 3.1.1 He-Kaiming Initialization (ReLU)

$$\sigma^2 = \frac{2\alpha}{D_{\text{in}}} \quad \Leftarrow \text{Var}[h_i^{(l)}] = \text{Var}[h_i^{(l-1)}]$$

#### 3.1.2 Xavier-Glorot Initialization

$$\sigma^2 = \frac{2\alpha}{D_{\text{in}} + D_{\text{out}}}$$

### 3.2 Bias-Variance Tradeoff

Estimate the generalization error

$$E^{\text{gen}} = \mathbb{E}_{\mathbf{x}, \mathbf{y} \sim p_D(\mathbf{x}, \mathbf{y})} [L(f_\phi(\mathbf{x}), \mathbf{y})] = \int L(f_\phi(\mathbf{x}), \mathbf{y}) p_D(\mathbf{x}, \mathbf{y}) d\mathbf{x} d\mathbf{y}$$

with a Monte-Carlo estimate:

$$E^{\text{gen}} \approx \frac{1}{N} \sum_{i=1}^N L(f_\phi(\mathbf{x}_i), \mathbf{y}_i)$$

where  $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$  are sampled from  $p_D(\mathbf{x}, \mathbf{y})$ .

The expected generalization error if we train  $f_{\phi(\mathcal{D})}$  on datasets  $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$  assuming a squared loss:

$$\begin{aligned} \mathbb{E}_{\mathcal{D}}[E^{\text{gen}}] &= \mathbb{E}_{\mathbf{x} \sim p_D(\mathbf{x})} \left[ [\bar{\mathbf{y}}(\mathbf{x}) - \bar{f}_{\phi(\mathcal{D})}(\mathbf{x})]^2 + \text{Var}_{\mathcal{D}}[f_{\phi(\mathcal{D})}(\mathbf{x})] + \text{Var}[\mathbf{y}(\mathbf{x})] \right] \\ &= \text{Bias}^2 + \text{Variance} + \text{Irreducible Error} \end{aligned}$$

### 3.3 Regularization Techniques

#### 3.3.1 Weight Decay

$$L'(\phi) = L(\phi) + g(\phi)$$

- $\ell^2$ -regularization

- $g(\phi) = \frac{\lambda}{2} \|\phi\|_2^2 = \frac{\lambda}{2} \sum_{i=1}^D \phi_i^2$

- Gradient:  $\frac{\partial g(\phi)}{\partial \phi_j} = \lambda \phi_j$

- $\ell^1$ -regularization

- $g(\phi) = \lambda \|\phi\|_1 = \lambda \sum_{i=1}^D |\phi_i|$

- Gradient:  $\frac{\partial g(\phi)}{\partial \phi_j} = \lambda \text{sign}(\phi_j)$

### 3.3.2 Other Regularization Techniques

- **Early stopping**
- **Data augmentation**
- **Injecting noise** to input data, activations, or weights
- **Ensemble methods**: bagging = bootstrap aggregating = resampling with replacement
- **Dropout**:
  - Randomly delete nodes with probability  $\rho = 0.5$
  - At test time, multiply weights by  $\rho$
  - Use as an ensemble of  $2^{(\# \text{ of hidden nodes})}$  networks
- **Transfer learning**
- **Multi-task learning**
- **Self-supervised learning**: generative (with masks) or contrastive (with pairs)

## 4 Residual Neural Networks

Add an identity connection to prevent shattered (uncorrelated) gradients:

$$\mathbf{h}^{(l)} = \mathbf{h}^{(l-1)} + f_{\phi^{(l)}}(\mathbf{h}^{(l-1)})$$

Allows gradients to flow through:

$$\frac{\partial \mathbf{h}^{(l)}}{\partial \mathbf{h}^{(l-1)}} = I + \frac{\partial f_{\phi^{(l)}}(\mathbf{h}^{(l-1)})}{\partial \mathbf{h}^{(l-1)}}$$

## 5 Convolutional Neural Networks (CNNs)

Allow for **local connectivity** and **parameter sharing**.

Name	Symbol	Dimension	
Input Image	$\mathbf{X}$	$H \times W \times c_{\text{in}}$	
Kernel/Filter	$\mathbf{W}$	$w \times h \times c_{\text{in}} \times c_{\text{out}}$	
Bias	$\mathbf{b}$	$c_{\text{out}} \times 1$	TODO
Output Feature Map	$\mathbf{Z}$	$H' \times W' \times c_{\text{out}}$	
Stride	$s$	Scalar (or per dim)	
Padding	$p$	Scalar (or per dim)	

### Important terms:

Kernel size, stride, padding, dilation rate (number of interspersed zero-values in kernel)

### 5.1 Invariance

FCN's have no notion of locality. We want layers to be **equivariant** to translations.

- **Equivariant:**  $f(t(x)) = t(f(x))$
- **Invariant:**  $f(t(x)) = f(x)$

The convolution operation is **equivariant** to translations.

### 5.2 Convolution Operation

Replace vectors with **tensors** indexed by  $(x, y, c)$ :

- Width:  $x$
- Height:  $y$
- **Channel:**  $c$

Convolution weights are tensors  $\mathbf{W} \in \mathbb{R}^{w \times h \times c_{\text{in}} \times c_{\text{out}}}$ .

$$h_{x,y,c}^{(l)} = \sum_{c',m,n} h_{x+m,y+n,c'}^{(l-1)} W_{m,n,c',c} + b_c$$

Each convolution produces a new set of hidden variables = **feature map** or **channel**.

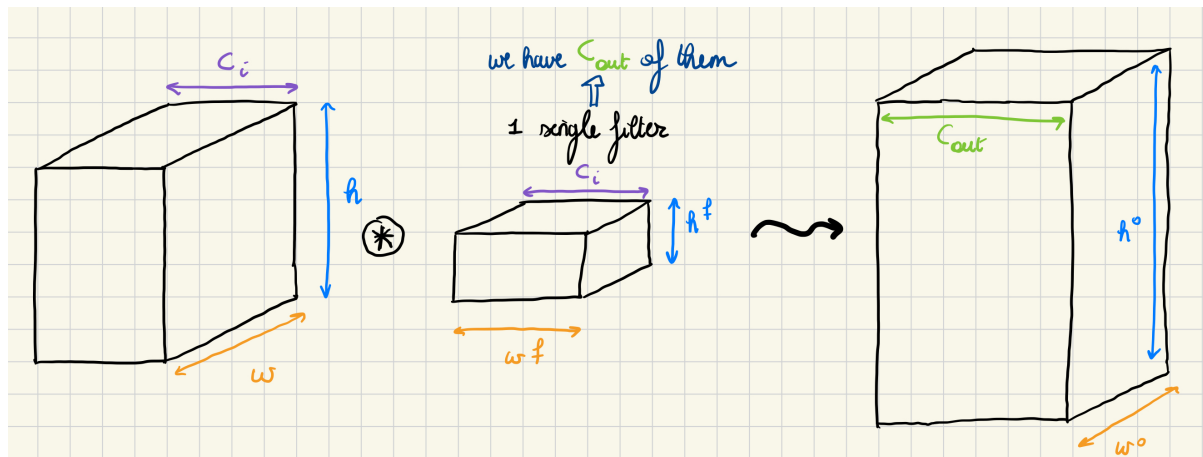
The **receptive field** of units in successive layers increases s.t. information from across the input is gradually aggregated.

### 5.3 Pooling

- **Increase channels** in **convolution** layers
- **Decrease resolution** in **pooling** layers

Variants of pooling:

- **Max Pooling**
- **Average Pooling**
- **Inverse Pooling:** Upsampling



## 5.4 Output dimensionality

Given:

- Input:  $C_i \times w \times h$
- Filters:  $C_i \times w_f \times h_f$
- Number of filters:  $C_o$
- Stride:  $s$
- Padding:  $p$

Output:

- $C_o$  channels
- Output width:  $\left\lfloor \frac{w+2p-w_f}{s} + 1 \right\rfloor$
- Output height:  $\left\lfloor \frac{h+2p-h_f}{s} + 1 \right\rfloor$

Each channel is a weighted sum of  $C_i$  input channels.

If we consider the kernel as a 4D tensor, the weights are shared across all output channels.

If we consider the kernel as a 3D tensor, each of the  $C_o$  kernels are different filters.

Each convolutional layer has  $C_i \cdot C_o \cdot w_f \cdot h_f$  weights and  $C_o$  biases.

MaxPool halves the spatial dimensions: e.g.  $13 \times 13 \times 256 \rightarrow 6 \times 6 \times 256$ .

SoftMax layer has no parameters.

## 6 Recurrent Neural Networks (RNNs)

- Input of length  $T$ :

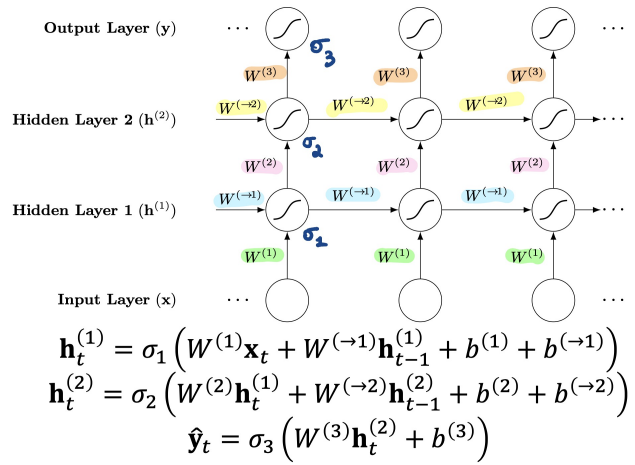
$$\mathbf{x} = \{\mathbf{x}_t\}_{t=1}^T, \quad \mathbf{x}_t \in \mathbb{R}^{D_x}$$

- Output of length  $S$ :

$$\mathbf{y} = \{\hat{\mathbf{y}}_t\}_{t=1}^S, \quad \hat{\mathbf{y}}_t \in \mathbb{R}^{D_y}$$

- The length  $T$  and  $S$  may vary between datapoints
- The length of the two sequences may differ:  $T \neq S$

Name	Symbol	Dimension
Sequence length	$T$	Scalar
Input at Time $t$	$\mathbf{x}_t$	$D_x \times 1$
Hidden State at $t$	$\mathbf{h}_t$	$D_h \times 1$
Output at $t$	$\hat{\mathbf{y}}_t$	$D_y \times 1$
Input Weights	$\mathbf{W}^{(i)}$	$D_h \times D_x$
Recurrent Weights	$\mathbf{W}^{(\rightarrow i)}$	$D_h \times D_h$
Output Weights	$\mathbf{W}^{(L)}$	$D_y \times D_h$
Biases	$\mathbf{b}^{(h)}, \mathbf{b}^{(y)}$	



### 6.1 MLE for RNNs

For an input-output pair

$$\begin{cases} \mathbf{x} = \mathbf{x}_1, \dots, \mathbf{x}_T \\ \mathbf{y} = \mathbf{y}_1, \dots, \mathbf{y}_S \end{cases}$$

we usually assume

$$p(\mathbf{y} | f_\phi(\mathbf{x})) = \prod_{t=1}^S p(\mathbf{y}_t | f_\phi(\mathbf{x}_t)) \quad (1)$$

$$= \prod_{t=1}^S p(\mathbf{y}_t | f_\phi(\mathbf{x}_{\leq t})) \quad (2)$$

## 6.2 Long Short-Term Memory (LSTM)

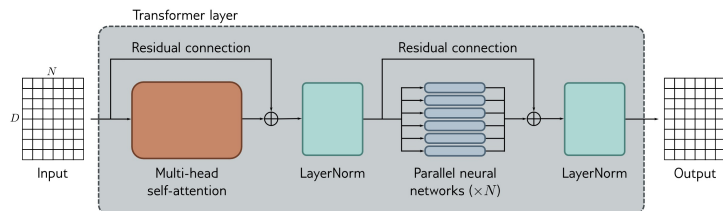
Gate/Component	Formula
Forget Gate	$\mathbf{f}_t = \sigma(\mathbf{W}_f[\mathbf{h}_{t-1}; \mathbf{x}_t] + \mathbf{b}_f)$
Input Gate	$\mathbf{i}_t = \sigma(\mathbf{W}_i[\mathbf{h}_{t-1}; \mathbf{x}_t] + \mathbf{b}_i)$
Cell Candidate	$\tilde{\mathbf{C}}_t = \tanh(\mathbf{W}_c[\mathbf{h}_{t-1}; \mathbf{x}_t] + \mathbf{b}_c)$
Cell State	$\mathbf{C}_t = \mathbf{f}_t \odot \mathbf{C}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{C}}_t$
Output Gate	$\mathbf{o}_t = \sigma(\mathbf{W}_o[\mathbf{h}_{t-1}; \mathbf{x}_t] + \mathbf{b}_o)$
Hidden State	$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{C}_t)$

## 7 Transformers & Attention

### 7.1 Notation and Dimensions

For sequences of length  $T'$ , embedding dim  $D'$ .

Name	Symbol	Dimension
Input Embeddings	$\mathbf{X}$	$T' \times D$
Queries	$\mathbf{Q}$	$T \times D$
Keys	$\mathbf{K}$	$T' \times D$
Values	$\mathbf{V}$	$T' \times D'$
Attention Weights	$\mathbf{A}$	$T \times T'$
Outputs	$\mathbf{Y}$	$T \times D'$
Number of Heads	$h$	-



### 7.2 Scaled Dot-Product Attention

$$\mathbf{A} = \text{softmax} \left( \frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{D'}} \right)$$

where

$$\underbrace{\mathbf{Q}}_{T \times D} \cdot \underbrace{\mathbf{K}^T}_{D \times T'} \in \mathbb{R}^{T \times T'}$$

Compute attention (runtime  $\mathcal{O}(n^2)$ !) as weighted sum of values:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathbf{A}\mathbf{V}$$

In multiheaded attention ( $h$  heads):

- Reduce dimensions of  $\mathbf{Q}$ ,  $\mathbf{K}$ ,  $\mathbf{V}$ :

$$\mathbf{Q}_i = \mathbf{Q}\mathbf{W}_i^Q$$

$$\mathbf{K}_i = \mathbf{K}\mathbf{W}_i^K$$

$$\mathbf{V}_i = \mathbf{V}\mathbf{W}_i^V$$

- Compute attention for each head:

$$\mathbf{A}_i = \text{softmax} \left( \frac{\mathbf{Q}_i\mathbf{K}_i^T}{\sqrt{D'}} \right)$$

$$\text{head}_i = \mathbf{A}_i\mathbf{V}_i$$

- Concatenate:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)$$

- Project up to original dimension:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V})\mathbf{W}_O$$

## 8 Unsupervised Deep Learning

### 8.1 Autoencoders (AE)

Encoder  $f_\phi : \mathbf{x} \rightarrow \mathbf{z}$ , Decoder  $g_\theta : \mathbf{z} \rightarrow \hat{\mathbf{x}}$ .

Name	Symbol	Dimension
Input	$\mathbf{x}$	$D_x \times 1$
Latent Code	$\mathbf{z}$	$D_z \times 1$ ( $D_z < D_x$ )
Reconstructed	$\hat{\mathbf{x}}$	$D_x \times 1$

- **Goal:** Learn compressed representation  $\mathbf{z}$  (bottleneck).
- **Loss:** Reconstruction loss (e.g., MSE).

$$L(\mathbf{x}, \hat{\mathbf{x}}) = \|\mathbf{x} - \hat{\mathbf{x}}\|^2 = \|\mathbf{x} - g_\theta(f_\phi(\mathbf{x}))\|^2$$

#### 8.1.1 Limitation

We want to sample the latent space  $\mathbf{z}$  to generate new data. However, in standard AE, the latent space is not regularized, so sampling from it (e.g.,  $\mathbf{z} \sim \mathcal{N}(0, I)$ ) does not guarantee meaningful generations.

### 8.2 Variational Autoencoders (VAE)

Probabilistic generative model. We assume a generative process:

$$\mathbf{z} \sim p_\theta(\mathbf{z}), \quad \mathbf{x} \sim p_\theta(\mathbf{x}|\mathbf{z})$$

where  $p_\theta(\mathbf{z})$  is the prior (usually  $\mathcal{N}(0, I)$ ) and  $p_\theta(\mathbf{x}|\mathbf{z})$  is the observation model.

#### 8.2.1 Intractability

We want to maximize the marginal likelihood (evidence)  $p_\theta(\mathbf{x})$ :

$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}, \mathbf{z}) d\mathbf{z} = \int p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z}) d\mathbf{z}$$

This integral is **intractable** because it requires integrating over all possible latent variables  $\mathbf{z}$ . Consequently, the true posterior  $p_\theta(\mathbf{z}|\mathbf{x})$  is also intractable:

$$p_\theta(\mathbf{z}|\mathbf{x}) = \frac{p_\theta(\mathbf{x}, \mathbf{z})}{p_\theta(\mathbf{x})}$$

#### 8.2.2 Amortized Variational Inference

To overcome this, we use **Variational Inference** with an approximate posterior  $q_\phi(\mathbf{z}|\mathbf{x}) \approx p_\theta(\mathbf{z}|\mathbf{x})$ . We use **Amortized Inference**, meaning the inference parameters  $\phi$  (encoder weights) are shared across all data points, mapping  $\mathbf{x}$  to the parameters of  $q_\phi(\mathbf{z}|\mathbf{x})$  (e.g.,  $\boldsymbol{\mu}_\phi(\mathbf{x}), \boldsymbol{\sigma}_\phi(\mathbf{x})$ ).

#### 8.2.3 Objective: Evidence Lower Bound (ELBO)

We maximize the ELBO, which is a lower bound on the log-likelihood:

$$\begin{aligned} \log p_\theta(\mathbf{x}) &\geq \text{ELBO} = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})) \\ &= \text{Reconstruction Term} - \text{Regularization Term} \end{aligned}$$

#### 8.2.4 Reparameterization Trick

To backpropagate through the sampling  $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}_\phi(\mathbf{x}), \boldsymbol{\sigma}_\phi^2(\mathbf{x})\mathbf{I})$ , we use:

$$\mathbf{z} = \boldsymbol{\mu}_\phi(\mathbf{x}) + \boldsymbol{\sigma}_\phi(\mathbf{x}) \odot \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

This makes the sampling operation differentiable w.r.t.  $\phi$ .

### 8.3 Semi-supervised Learning

Semi-supervised learning

- $\neq$  transfer learning!
- Little labeled data  $\mathbf{y}$
- Lots of unlabeled data  $\mathbf{x}$
- Auto-encode (unsupervised) to  $\mathbf{z}$
- Train classifier on  $\mathbf{z}$  (supervised)