02456 Deep Learning

Cheat Sheet

December 3, 2025

Vincent Van Schependom

Contents

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	$\boldsymbol{x} = \boldsymbol{h}^{(0)}$	$D^{(0)} \times 1$
Weights (Layer l)	$\boldsymbol{W}^{(l)}$	$D^{(l)} \times D^{(l-1)}$
Bias (Layer l)	$\boldsymbol{b}^{(l)}$	$D^{(l)} \times 1$
Pre-activation (Layer l)	$\boldsymbol{z}^{(l)}$	$D^{(l)} \times 1$
Activation Function	$\sigma(\cdot)$	
Hidden State (Layer l)	$\boldsymbol{h}^{(l)}$	$D^{(l)} \times 1$
Output	$\boldsymbol{y} = \boldsymbol{h}^{(L)}$	$D^{(L)} \times 1$
Number of Layers	L	
Neurons per Layer	D	
Distribution params	$\boldsymbol{\theta}$	
Model params	$\boldsymbol{\phi}$	

Layer computation:

$$\boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)}\boldsymbol{h}^{(l-1)} + \boldsymbol{b}^{(l)}$$

$$\boldsymbol{h}^{(l)} = \sigma(\boldsymbol{z}^{(l)})$$

Weight matrix:

$$\boldsymbol{W}^{(l)} = \begin{bmatrix} w^{(l)}_{1\leftarrow 1} & w^{(l)}_{1\leftarrow 2} & \cdots & w^{(l)}_{1\leftarrow D^{(l-1)}} \\ \vdots & \vdots & \ddots & \vdots \\ w^{(l)}_{D^{(l)}\leftarrow 1} & w^{(l)}_{D^{(l)}\leftarrow 2} & \cdots & w^{(l)}_{D^{(l)}\leftarrow D^{(l-1)}} \end{bmatrix}$$

Here, $w^{(l)}_{j\leftarrow i}$ is the weight from input neuron i in layer $l-1$ to output neuron j in layer l.

1.2 Full network

We predict the **distribution** (parameters $\boldsymbol{\theta}$) of the labels $\boldsymbol{y}$ given the inputs $\boldsymbol{x}$ using the model $f_{\boldsymbol{\phi}}(\boldsymbol{x})$:

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

Using these distribution parameters, we calculate the distributions:

- Regression:

$$p(\boldsymbol{y}|f_{\boldsymbol{\phi}}(\boldsymbol{x})) = p(\boldsymbol{y}|\mu_1, \ldots, \mu_k, \sigma_1, \ldots, \sigma_k) = \boxed{\mathcal{N}(\boldsymbol{y}|\mu_1, \ldots, \mu_k, \sigma_1, \ldots, \sigma_k)}$$

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

$$p(\boldsymbol{y}|f_{\boldsymbol{\phi}}(\boldsymbol{x})) = p(\boldsymbol{y}|\pi_1, \ldots, \pi_K) = \boxed{\Pi_{d=1}^{K} \pi_d^{y_d}}$$

1.3 Probabilistic Inference

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

- Most probable value:

$$\hat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y}} p(\boldsymbol{y}|f_{\hat{\boldsymbol{\phi}}}(\boldsymbol{x}))$$

- Expected value:

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

- Sample:

$$\hat{\boldsymbol{y}} \sim p\left(\boldsymbol{y}|f_{\hat{\boldsymbol{\phi}}}(\boldsymbol{x})\right)$$

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
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

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.

2 Training & Optimization

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

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

We learn/fit the model by minimizing this loss:

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

2.1 Loss Functions

Name	Formula	Type
Mean Squared Error (MSE)	$\frac{1}{N}\sum_{n=1}^{N} \|\|f_{\boldsymbol{\phi}}(\boldsymbol{x}_n) - \boldsymbol{y}_n\|\|^2$	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)	$-\frac{1}{N}\sum_{n=1}^{N} \log p(\boldsymbol{y}_n \| f_{\boldsymbol{\phi}}(\boldsymbol{x}_n))$	General

2.2 Maximum Likelihood Estimation (MLE)

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

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

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

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

Assuming that dimensions of $\boldsymbol{y}$ are independent given $\boldsymbol{x}$:

$$p(\boldsymbol{y} | f_{\boldsymbol{\phi}}(\boldsymbol{x})) = \prod_{d=1}^{D} p(y_d | f_{\boldsymbol{\phi}}(\boldsymbol{x}))$$

which yields the loss function

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

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

$$\boxed{L(\boldsymbol{\phi}) = -\sum_{i=1}^{N}\sum_{d=1}^{K} y_{id} \log \pi_d}$$

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

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

.

2.3 Gradient Descent

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

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

2.3.1 Stochastic Gradient Descent (SGD)

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

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

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

2.3.2 Adam (Adaptive Moment Estimation)

Compute first and second moment of gradients:

$$\boldsymbol{m}^{(t+1)} = \beta \boldsymbol{m}^{(t)} + (1-\beta) \nabla_{\boldsymbol{\phi}} \ell_i(\boldsymbol{\phi}^{(t)})$$
$$\boldsymbol{v}^{(t+1)} = \gamma \boldsymbol{v}^{(t)} + (1-\gamma) \nabla_{\boldsymbol{\phi}} \ell_i(\boldsymbol{\phi}^{(t)})^2$$

Compensate for initial values close to zero:

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

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

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

where η is the learning rate, and ϵ is a small constant to prevent division by zero.

2.4 Backpropagation

Used to calculate gradients $\nabla_{\boldsymbol{\phi}} L(\boldsymbol{\phi})$. TODO: chain rule derivatives in backward pass.

3 Initialization & Regularization

3.1 Weight Initialization

Avoid vanishing/exploding gradients during backprop. Initialize $\boldsymbol{\phi}_i \sim \mathcal{N}(0, \sigma^2)$.

3.1.1 He-Kaiming Initialization (ReLU)

$$\sigma^2 = \frac{2}{D_{\text{in}}} \qquad \Longleftarrow \text{Var}\big[h_i^{(l)}\big] = \text{Var}\big[h_i^{(l-1)}\big]$$

3.1.2 Xavier-Glorot Initialization

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

3.2 Bias-Variance Tradeoff

Estimate the generalization error

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

with a Monte-Carlo estimate:

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

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

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

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

3.3 Regularization Techniques

3.3.1 Weight Decay

$$L'(\boldsymbol{\phi}) = L(\boldsymbol{\phi}) + g(\boldsymbol{\phi})$$

- ℓ^2-regularization
  - $g(\boldsymbol{\phi}) = \frac{\lambda}{2}||\boldsymbol{\phi}||_2^2 = \frac{\lambda}{2}\sum_{i=1}^{D}\phi_i^2$
  - Gradient: $\frac{\partial g(\boldsymbol{\phi})}{\partial \boldsymbol{\phi}_j} = \lambda \boldsymbol{\phi}_j$
- ℓ^1-regularization
  - $g(\boldsymbol{\phi}) = \lambda||\boldsymbol{\phi}||_1 = \lambda\sum_{i=1}^{D}|\phi_i|$
  - Gradient: $\frac{\partial g(\boldsymbol{\phi})}{\partial \boldsymbol{\phi}_j} = \lambda \text{sign}(\boldsymbol{\phi}_j)$

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 ρ
  - Use as an ensemble of $2^{(\#\text{ of hidden nodes})}$ networks

4 Residual Neural Networks

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

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

Allows gradients to flow through:

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

5 Convolutional Neural Networks (CNNs)

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

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

TODO

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**: $f(t(x)) = f(x)$

5.2 Convolution Operation

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

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

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

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

5.3 Pooling

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

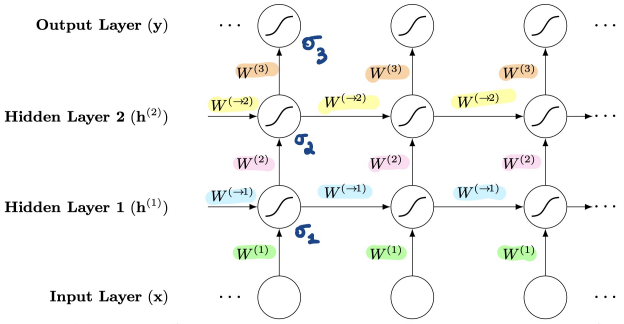
Variants of pooling:

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

6 Recurrent Neural Networks (RNNs)

- Input of length T:
$$\mathbf{x} = \{\boldsymbol{x}_t\}_{t=1}^{T}, \qquad \boldsymbol{x}_t \in \mathbb{R}^{D_x}$$
- Output of length S:
$$\mathbf{y} = \{\hat{\boldsymbol{y}}_t\}_{t=1}^{S}, \qquad \hat{\boldsymbol{y}}_t \in \mathbb{R}^{D_y}$$
- 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	$\boldsymbol{x}_t$	$D_x \times 1$
Hidden State at t	$\boldsymbol{h}_t$	$D_h \times 1$
Output at t	$\hat{\boldsymbol{y}}_t$	$D_y \times 1$
Input Weights	$\boldsymbol{W}^{(i)}$	$D_h \times D_x$
Recurrent Weights	$\boldsymbol{W}^{(\rightarrow i)}$	$D_h \times D_h$
Output Weights	$\boldsymbol{W}^{(L)}$	$D_y \times D_h$
Biases	$\boldsymbol{b}^{(h)}, \boldsymbol{b}^{(y)}$	



$$\mathbf{h}_t^{(1)} = \sigma_1\left(W^{(1)}\mathbf{x}_t + W^{(\rightarrow 1)}\mathbf{h}_{t-1}^{(1)} + b^{(1)} + b^{(\rightarrow 1)}\right)$$
$$\mathbf{h}_t^{(2)} = \sigma_2\left(W^{(2)}\mathbf{h}_t^{(1)} + W^{(\rightarrow 2)}\mathbf{h}_{t-1}^{(2)} + b^{(2)} + b^{(\rightarrow 2)}\right)$$
$$\hat{\mathbf{y}}_t = \sigma_3\left(W^{(3)}\mathbf{h}_t^{(2)} + b^{(3)}\right)$$

6.1 MLE for RNNs

For an input-output pair
$$\begin{cases} \boldsymbol{x} = \boldsymbol{x}_1, \ldots, \boldsymbol{x}_T \\ \boldsymbol{y} = \boldsymbol{y}_1, \ldots, \boldsymbol{y}_S \end{cases}$$

we usually assume

$$p(\boldsymbol{y} \mid f_{\boldsymbol{\phi}}(\boldsymbol{x})) = \prod_{t=1}^{S} p(\boldsymbol{y}_t \mid f_{\boldsymbol{\phi}}(\boldsymbol{x}_t)) \tag{1}$$
$$= \prod_{t=1}^{S} p(\boldsymbol{y}_t \mid f_{\boldsymbol{\phi}}(\boldsymbol{x}_{\leq t})) \tag{2}$$

6.2 Long Short-Term Memory (LSTM)

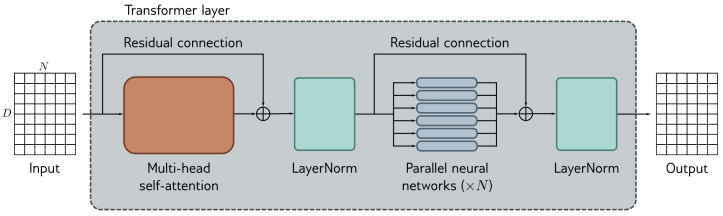
Gate/Component	Formula
Forget Gate	$\boldsymbol{f}_t = \sigma\big(\boldsymbol{W}_f[\boldsymbol{h}_{t-1}; \boldsymbol{x}_t] + \boldsymbol{b}_f\big)$
Input Gate	$\boldsymbol{i}_t = \sigma\big(\boldsymbol{W}_i[\boldsymbol{h}_{t-1}; \boldsymbol{x}_t] + \boldsymbol{b}_i\big)$
Cell Candidate	$\tilde{\boldsymbol{C}}_t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

7 Transformers & Attention

7.1 Notation and Dimensions

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

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



7.2 Scaled Dot-Product Attention

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

where

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

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

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

In multiheaded attention (h heads):

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

$$\begin{aligned}\boldsymbol{Q}_i &= \boldsymbol{Q}\boldsymbol{W}_i^Q \\ \boldsymbol{K}_i &= \boldsymbol{K}\boldsymbol{W}_i^K \\ \boldsymbol{V}_i &= \boldsymbol{V}\boldsymbol{W}_i^V\end{aligned}$$

- Compute attention for each head:

$$\begin{aligned}\boldsymbol{A}_i &= \text{softmax}\left(\frac{\boldsymbol{Q}_i\boldsymbol{K}_i^T}{\sqrt{D'}}\right) \\ \text{head}_i &= \boldsymbol{A}_i\boldsymbol{V}_i\end{aligned}$$

- Concatenate:

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

- Project up to original dimension:

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

8 Unsupervised Deep Learning

8.1 Autoencoders (AE)

Encoder $f_{\phi_e} : \boldsymbol{x} \to \boldsymbol{z}$, Decoder $g_{\phi_d} : \boldsymbol{z} \to \hat{\boldsymbol{x}}$.

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

- **Encoder**: $f : \boldsymbol{x} \mapsto \boldsymbol{z}$
- **Decoder**: $g : \boldsymbol{z} \mapsto \hat{\boldsymbol{x}}$
- **Reconstruction**: $\hat{\boldsymbol{x}} = g(f(\boldsymbol{x}))$
- We want $\hat{\boldsymbol{x}} \approx \boldsymbol{x}$

8.2 Autoencoders (AE)

Focus on **bottleneck** autoencoders, i.e. $D_z < D_x$.

8.2.1 Limitation

We want to sample the latent space $\boldsymbol{z}$ using the decoder to generate new data $\hat{\boldsymbol{x}}$.
But we didn't make any assumptions about the distribution of $\boldsymbol{z}$, so we can't sample from it.

8.3 Variational Autoencoders (VAE)

8.4 Semi-supervised Learning

Semi-supervised learning

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