Deep Metric Learning

Deep Learning (ENGG*6600*07)

School of Engineering,
University of Guelph, ON, Canada

Course Instructor: Benyamin Ghojogh
Fall 2023

Introduction

Introduction

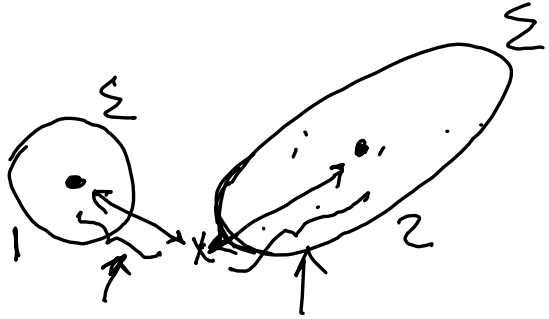
- Both **spectral** and **probabilistic** metric learning methods use the generalized Mahalanobis distance, i.e. Eq. (1):

$$\begin{aligned} \|x_i - x_j\|_W &:= \sqrt{(x_i - x_j)^\top W (x_i - x_j)}. \\ \therefore \quad \|x_i - x_j\|_W^2 &:= (x_i - x_j)^\top W (x_i - x_j), \end{aligned} \tag{1}$$

  and learn the weight matrix in the metric.
- **Deep metric learning**, however, has a different approach. The methods in deep metric learning usually do not use a generalized Mahalanobis distance but they learn an **embedding space** using a neural network.
- The network learns a p-dimensional embedding space for **discriminating classes or the dissimilar points** and **making the similar points close** to each other. The network embeds data in the embedding space (or subspace) of metric.
- Then, any distance metric $d(.,.) : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}$ can be used in this embedding space. In the loss functions of network, we can use the distance function $d(.,.)$ in the embedding space. For example, an option for the distance function is the squared ℓ_2 norm or squared Euclidean distance:

$$d\big(\mathbf{f}(x_i^1), \mathbf{f}(x_i^2)\big) := \|\mathbf{f}(x_i^1) - \mathbf{f}(x_i^2)\|_2^2, \tag{2}$$

  where $\mathbf{f}(x_i) \in \mathbb{R}^p$ denotes the output of network for the input x_i as its p-dimensional embedding.
- We train the network using mini-batch methods such as the mini-batch stochastic gradient descent and denote the mini-batch size by b. The weights of network are denoted by the learnable parameter θ.



$$\sqrt{(x_i-\mu)^T(x_i-\mu)} = \|x_i-\mu\|_2$$

$$\sqrt{(x_i-\mu)^T \Sigma^{-1} (x_i-\mu)}$$

$$\sqrt{(x_i-\mu)^T W (x_i-\mu)}$$

$W \succcurlyeq 0$

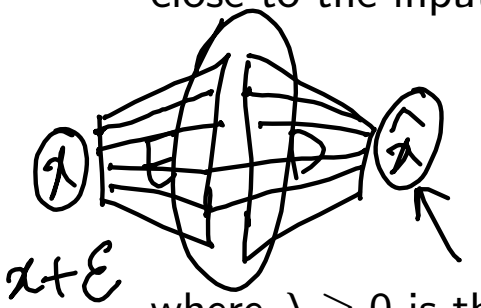
Reconstruction Autoencoders

Types of Autoencoders

- An **autoencoder** is a model consisting of an **encoder** $E(.)$ and a **decoder** $D(.)$.
- There are several types of autoencoders. All types of autoencoders learn a **code layer** in the middle of encoder and decoder.
- **Inferential autoencoders** learn a stochastic latent space in the code layer between the encoder and decoder.
- **Variational autoencoder** [1] and **adversarial autoencoder** [2] are two important types of inferential autoencoders.
- Another type of autoencoder is the **reconstruction autoencoder** consisting of an encoder, transforming data to a code, and a decoder, transforming the code back to the data. Hence, the decoder reconstructs the input data to the encoder.
- The code is a representation for data. Each of the encoder and decoder can be multiple layers of neural network with activation functions.

Reconstruction Loss

- We denote the input data point to the encoder by $\boldsymbol{x} \in \mathbb{R}^d$ where d is the dimensionality of data.
- The reconstructed data point is the output of decoder and is denoted by $\widehat{\boldsymbol{x}} \in \mathbb{R}^d$.
- The representation code, which is the output of encoder and the input of decoder, is denoted by $\mathbf{f}(\boldsymbol{x}) := E(\boldsymbol{x}) \in \mathbb{R}^p$. We have $\widehat{\boldsymbol{x}} = D(E(\boldsymbol{x})) = D(\mathbf{f}(\boldsymbol{x}))$.
- If the dimensionality of code is greater than the dimensionality of input data, i.e. $p > d$, the autoencoder is called an **over-complete autoencoder** [3]. Otherwise, if $p < d$, the autoencoder is an **under-complete autoencoder** [3].
- The loss function of reconstruction autoencoder tries to make the reconstructed data close to the input data:



$$\underset{\theta}{\text{minimize}} \quad \overbrace{\sum_{i=1}^{b} \Big(d\big(\boldsymbol{x}_i, \widehat{\boldsymbol{x}}_i\big) + \lambda \Omega(\theta) \Big)}, \tag{3}$$

where $\lambda \geq 0$ is the regularization parameter and $\Omega(\theta)$ is some penalty or regularization on the weights. Here, the distance function $d(.,.)$ is defined on $\mathbb{R}^d \times \mathbb{R}^d$. Note that the penalty term can be regularization on the code $\mathbf{f}(\boldsymbol{x}_i)$.

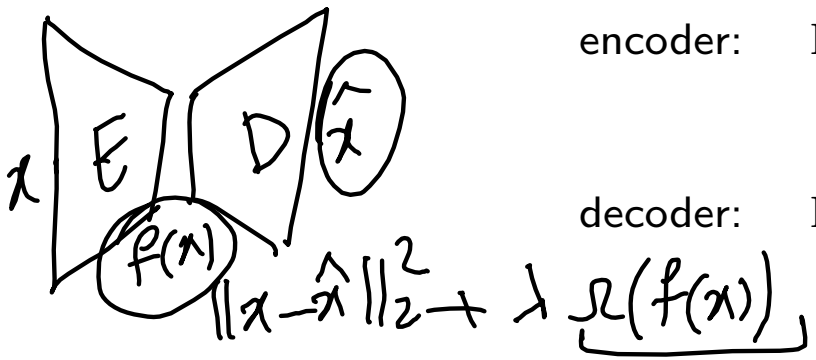
- If the used distance metric is the squared Euclidean distance, this loss is named the regularized Mean Squared Error (MSE) loss.

Denoising Autoencoder

- A problem with **over-complete autoencoder** is that its training only **copies** each feature of data input to one of the neurons in the code layer and then **copies** it back to the corresponding feature of output layer.
- This is because the number of neurons in the code layer is greater than the number of neurons in the input and output layers.
- In other words, the networks just **memorizes** or gets **overfit**.
- This copying happens by making some of the weights equal to one (or a scale of one depending on the activation functions) and the rest of weights equal to zero.
- To avoid this problem in over-complete autoencoders, one can **add some noise to the input data** and try to **reconstruct the data without noise**.
- This forces the over-complete autoencoder to **not just copy data to the code layer**. This autoencoder can be used for denoising as it reconstructs the data without noise for a noisy input. This network is called the **Denoising Autoencoder (DAE)** [3].

Metric Learning by Reconstruction Autoencoder

- The **under-complete reconstruction autoencoder** can be used for **metric learning** and **dimensionality reduction**, especially when $p \ll d$.
- The loss function for learning a low-dimensional representation code and reconstructing data by the autoencoder is Eq. (3). The code layer between the encoder and decoder is the **embedding space** of metric.
- Note that if the **activation functions of all layers are linear**, the under-complete autoencoder is reduced to **Principal Component Analysis** [4]. Let $\boldsymbol{U}_l$ denote the weight matrix of the l-th layer of network, ℓ_e be the number of layers of encoder, and ℓ_d be the number of layers of decoder. With linear activation function, the encoder and decoder are:



$$\text{encoder:} \quad \mathbb{R}^p \ni \mathbf{f}(\boldsymbol{x}_i) = \underbrace{\boldsymbol{U}_{\ell_e}^\top \boldsymbol{U}_{\ell_e-1}^\top \dots \boldsymbol{U}_1^\top}_{\boldsymbol{U}_e^\top} \boldsymbol{x}_i,$$

$$\text{decoder:} \quad \mathbb{R}^d \ni \widehat{\boldsymbol{x}}_i = \underbrace{\boldsymbol{U}_1 \dots \boldsymbol{U}_{\ell_d-1} \boldsymbol{U}_{\ell_d}}_{\boldsymbol{U}_d} \mathbf{f}(\boldsymbol{x}_i),$$

where the ℓ concatenated linear projections can be replaced by a linear projection with projection matrices $\boldsymbol{U}_e$ and $\boldsymbol{U}_d$ in the encoder and decoder, respectively.

- For learning complicated data patterns, we can use **nonlinear activation functions** between layers of the encoder and decoder to have nonlinear metric learning and dimensionality reduction.

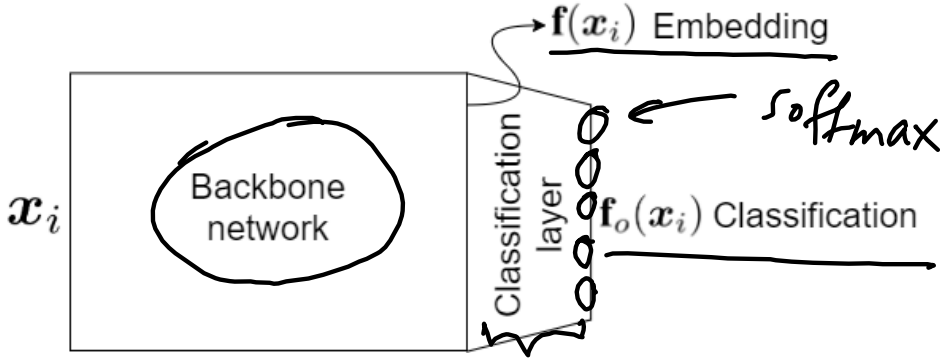
Metric Learning by Reconstruction Autoencoder

- It is noteworthy that nonlinear neural network can be seen as an **ensemble or concatenation of dimensionality reduction (or feature extraction) and kernel methods**. The justification of this claim is as follows.
- Let the dimensionality for a layer of network be $\boldsymbol{U} \in \mathbb{R}^{d_1 \times d_2}$ so it connects d_1 neurons to d_2 neurons. Two cases can happen:
  - If $d_1 \geq d_2$, this layer acts as **dimensionality reduction** or **feature extraction** because it has reduced the dimensionality of its input data. If this layer has a nonlinear activation function, the dimensionality reduction is nonlinear; otherwise, it is linear.
  - If $d_1 < d_2$, this layer acts as a **kernel method** which maps its input data to the high-dimensional feature space in some Reproducing Kernel Hilbert Space (RKHS). This kernelization can help nonlinear separation of some classes which are not separable linearly [5]. An example use of kernelization in machine learning is kernel support vector machine [6].
- Therefore, a neural network is a complicated feature extraction method as a concatenation of dimensionality reduction and kernel methods, where **each layer of network learns its own features from data**.

Supervised Metric Learning by Supervised Loss Functions

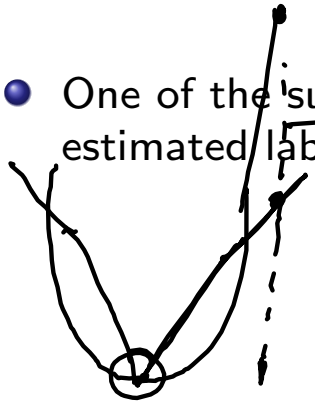
Supervised Metric Learning by Supervised Loss Functions

- Various loss functions exist for supervised metric learning by neural networks.
- **Supervised** loss functions can teach the network to **separate classes in the embedding space** [7].
- For this, we use a network whose **last layer is for classification** of data points. The features of the **one-to-last layer** can be used for feature **embedding**. The last layer after the embedding features is named the **classification layer**.



- Let the i-th point in the mini-batch be denoted by $\boldsymbol{x}_i \in \mathbb{R}^d$ and its label be denoted by $y_i \in \mathbb{R}$. Suppose the network has one output neuron and its output for the input $\boldsymbol{x}_i$ is denoted by $\mathbf{f}_o(\boldsymbol{x}_i) \in \mathbb{R}$. This output is the **estimated class label** by the network. We denote output of the **one-to-last layer** by $\mathbf{f}(\boldsymbol{x}_i) \in \mathbb{R}^p$ where p is the number of neurons in that layer which is equivalent to the **dimensionality of the embedding space**.
- The last layer of network, connecting the p neurons to the output neuron is a **fully-connected layer**. The network until the one-to-last layer can be any **feed-forward or convolutional network** depending on the type of data. If the network is convolutional, it should be flattened at the one-to-last layer.
- The network learns to classify the classes, by the supervised loss functions, so the features of the one-to-last layer will be **discriminating features** and suitable for embedding.

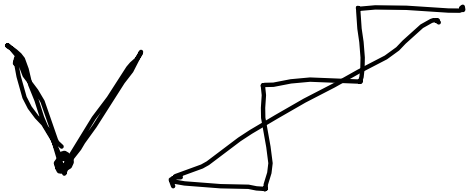
Mean Squared Error and Mean Absolute Value Losses



- One of the supervised losses is the **Mean Squared Error (MSE)** which makes the estimated labels close to the true labels using squared ℓ_2 norm:

$$\underset{\theta}{\text{minimize}} \quad \sum_{i=1}^{b} (\mathbf{f}_o(\boldsymbol{x}_i) - y_i)^2. \tag{4}$$

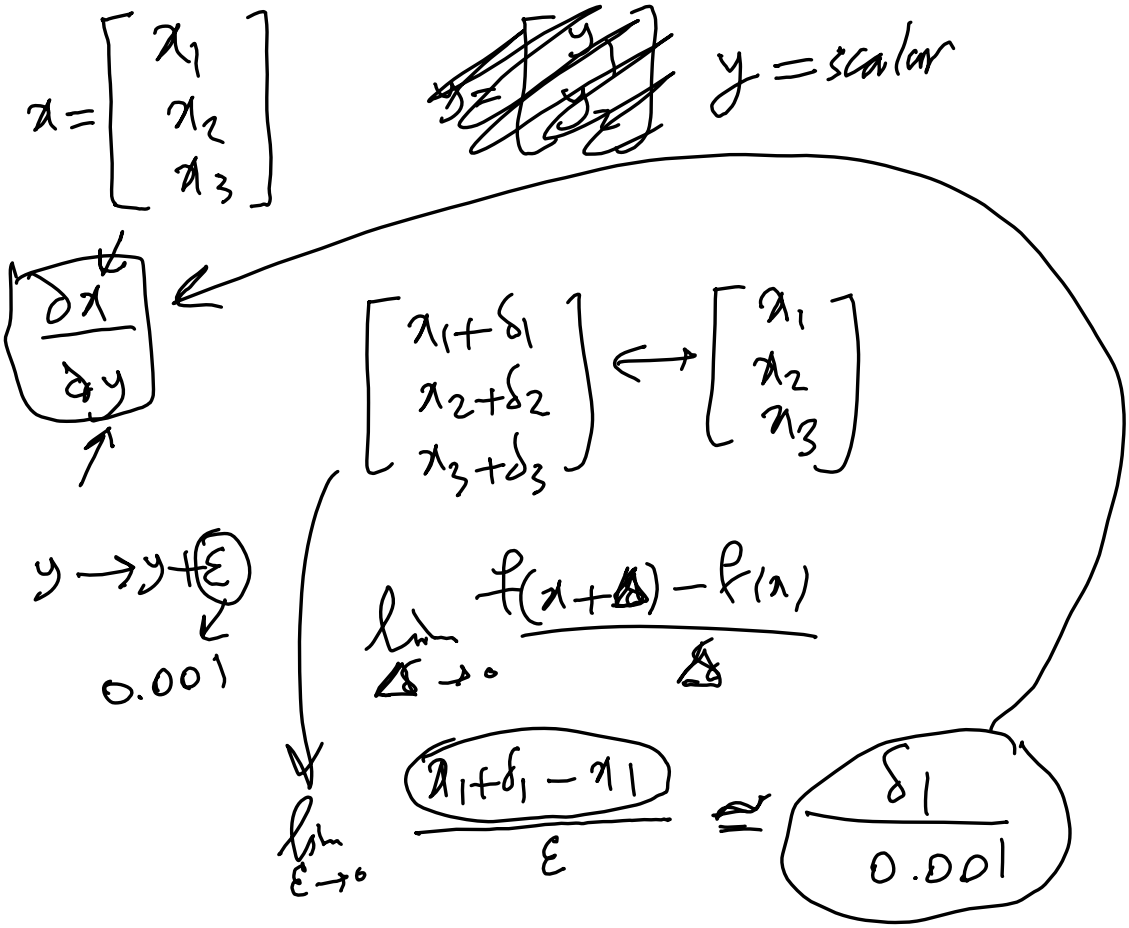
- One problem with this loss function is **exaggerating outliers** because of being squared but its advantage is its **differentiability**.
- Another loss function is the **Mean Absolute Error (MAE)** which makes the estimated labels close to the true labels using ℓ_1 norm or the absolute value:



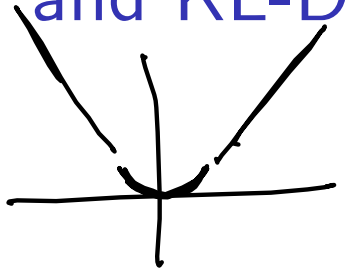
$$\underset{\theta}{\text{minimize}} \quad \sum_{i=1}^{b} |\mathbf{f}_o(\boldsymbol{x}_i) - y_i|. \tag{5}$$

The distance used in this loss is also named the **Manhattan distance**.

- This loss function does not have the problem of MSE and it can be used for imposing **sparsity** in the embedding. It is **not differentiable** at the point $\mathbf{f}(\boldsymbol{x}_i) = y_i$ but as the **derivatives are calculated numerically** by the neural network, this is not a big issue nowadays.


$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$
y
$y = \text{scalar}$
$\frac{\partial x}{\partial y}$
$\begin{bmatrix} x_1+\delta_1 \\ x_2+\delta_2 \\ x_3+\delta_3 \end{bmatrix} \longleftrightarrow \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$
$y \to y+\varepsilon$
0.001
$\lim_{\Delta \to 0} \frac{f(x+\Delta)-f(x)}{\Delta}$
$\lim_{\varepsilon \to 0} \frac{x_1+\delta_1-x_1}{\varepsilon} \simeq \frac{\delta_1}{0.001}$

Huber and KL-Divergence Losses

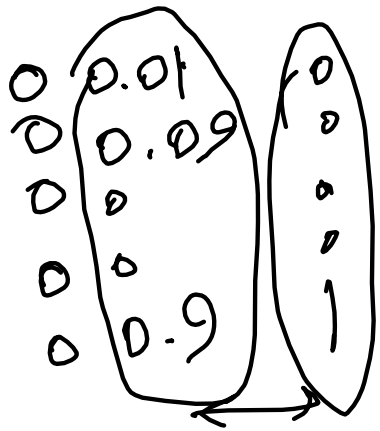


- Another loss function is the **Huber loss** which is a **combination of the MSE and MAE** to have the advantages of both of them:

$$\underset{\theta}{\text{minimize}} \quad \sum_{i=1}^{b} \begin{cases} 0.5(\mathbf{f}_o(\boldsymbol{x}_i) - y_i)^2 & \text{if } |\mathbf{f}_o(\boldsymbol{x}_i) - y_i| \leq \delta \\ \delta(|\mathbf{f}_o(\boldsymbol{x}_i) - y_i| - 0.5\delta) & \text{otherwise.} \end{cases} \tag{6}$$

- **KL-divergence loss function** makes the distribution of the estimated labels close to the distribution of the true labels:

$$\underset{\theta}{\text{minimize}} \quad \text{KL}(\mathbb{P}(\mathbf{f}(\boldsymbol{x}))\|\mathbb{P}(y)) = \sum_{i=1}^{b} \mathbf{f}(\boldsymbol{x}_i) \log(\frac{\mathbf{f}(\boldsymbol{x}_i)}{y_i}). \tag{7}$$



Hinge Loss

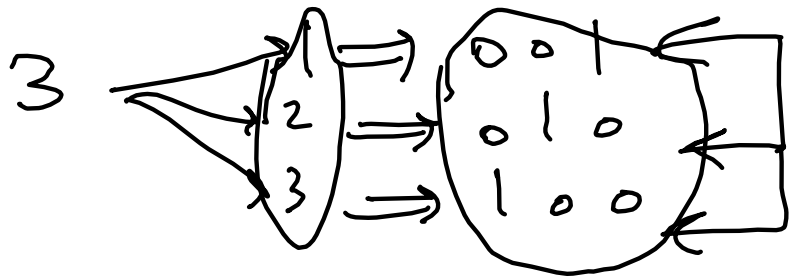
- If there are **two classes**, i.e. $c = 2$, we can have true labels as $y_i \in \{-1, 1\}$. In this case, a possible loss function is the Hinge loss:

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^{b} \left[m - y_i \, \mathbf{f}_o(\boldsymbol{x}_i)\right]_+, \tag{8}$$

where $[\cdot]_+ := \max(\cdot, 0)$ and $m > 0$ is the margin.

- If the signs of the estimated and true labels are different, the loss is positive (i.e., $\left[m - y_i \, \mathbf{f}_o(\boldsymbol{x}_i)\right]_+ = m - y_i \, \mathbf{f}_o(\boldsymbol{x}_i) > 0$) which should be minimized.
- If the signs are the same and $|\mathbf{f}_o(\boldsymbol{x}_i)| \geq m$, then the loss function is zero (because $m - y_i \, \mathbf{f}_o(\boldsymbol{x}_i) < 0$).
- If the signs are the same but $|\mathbf{f}_o(\boldsymbol{x}_i)| < m$, the loss is positive and should be minimized because the estimation is correct but not with enough margin from the incorrect estimation.

Cross-entropy Loss



- For any number of classes, denoted by c, we can have a **cross-entropy loss**. For this loss, we have c **neurons, rather than one neuron, at the last layer**.
- In contrast to the MSE, MAE, Huber, and KL-divergence losses which use linear activation function at the last layer, cross-entropy requires **softmax** or **sigmoid** activation function at the last layer so the output values are between zero and one.
- For this loss, we have c outputs, i.e. $\mathbf{f}_o(\boldsymbol{x}_i) \in \mathbb{R}^c$ (continuous values between zero and one), and the true labels are one-hot encoded, i.e., $\boldsymbol{y}_i \in \{0, 1\}^c$. This loss is defined as:

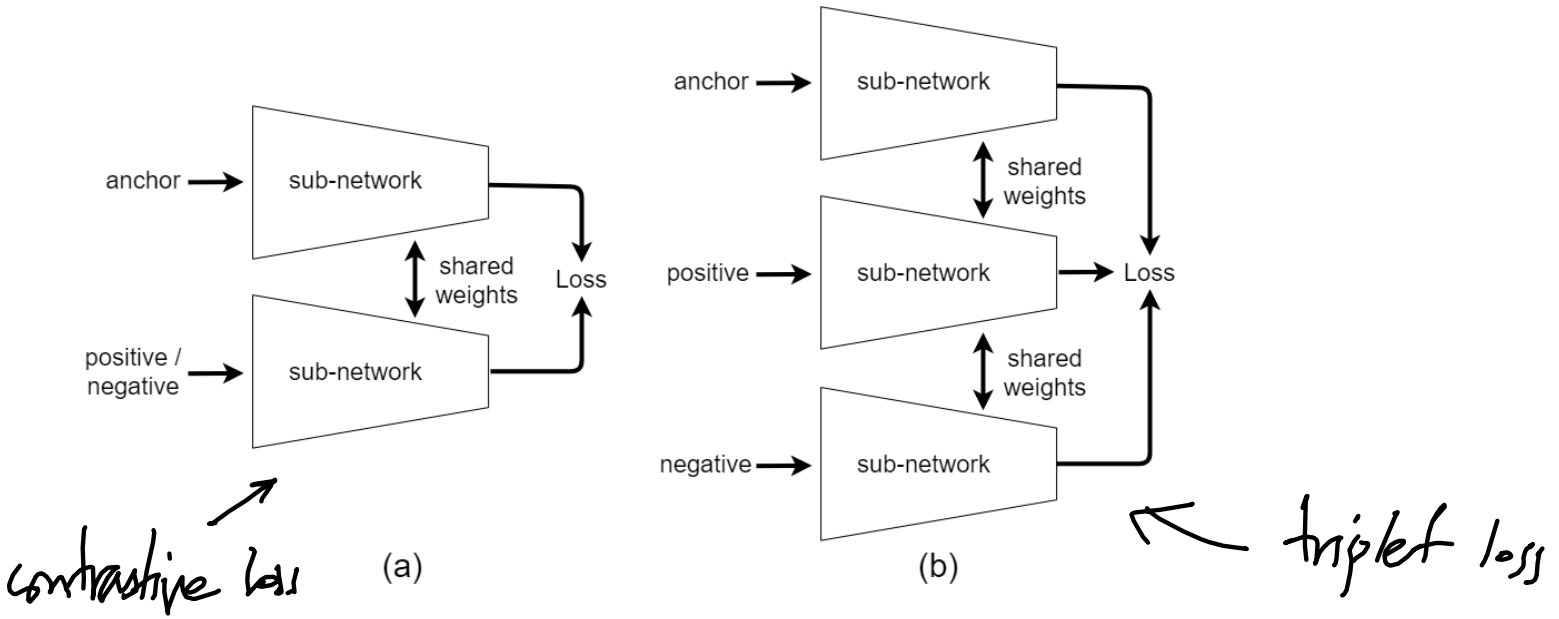
$$\underset{\theta}{\text{minimize}} \quad -\sum_{i=1}^{b}\sum_{l=1}^{c}(\boldsymbol{y}_i)_l \log\big(\mathbf{f}_o(\boldsymbol{x}_i)_l\big), \tag{9}$$

  where $(\boldsymbol{y}_i)_l$ and $\mathbf{f}_o(\boldsymbol{x}_i)_l$ denote the l-th element of $\boldsymbol{y}_i$ and $\mathbf{f}_o(\boldsymbol{x}_i)$, respectively.
- Minimizing this loss **separates classes** for classification; this separation of classes also gives us a **discriminating embedding** (an embedding which discriminates the classes) in the one-to-last layer [7, 8].

Metric Learning by Siamese Networks

Siamese and Triplet Networks

- One of the important deep metric learning methods is **Siamese network** which is widely used for feature extraction.
- Siamese network, originally proposed in (1993) [9], is a network consisting of several equivalent sub-networks sharing their weights.
- The number of sub-networks in a Siamese network can be any number but it usually is **two or three**.
- A Siamese network with three sub-networks is also called a **triplet network** [10].
- The weights of sub-networks in a Siamese network are trained in a way that the **intra- and inter-class variances are decreased and increased, respectively**. In other words, the similar points are pushed toward each other while the dissimilar points are pulled away from one another.
- Siamese networks have been used in various applications such as computer vision [11] and natural language processing [12].



Pairs and Triplets of Data Points

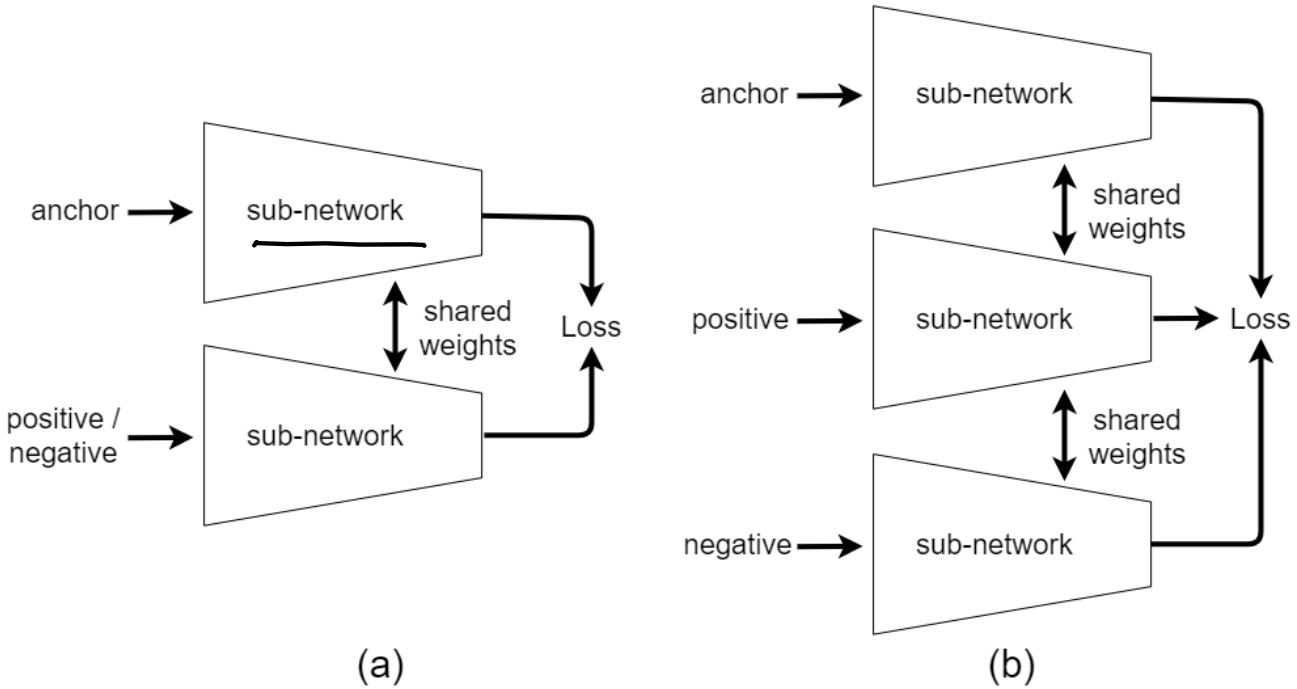
- Depending on the number of sub-networks in the Siamese network, we have loss functions for training.
- The loss functions of Siamese networks usually require **pairs or triplets of data points**. Siamese networks do not use the data points one by one but we need to **make pairs or triplets of points out of dataset** for training a Siamese network.
- For making the pairs or triplets, we consider every data point as the **anchor point**, denoted by $\boldsymbol{x}_i^a$. Then, we take one of the **similar** points to the anchor point as the **positive (or neighbor) point**, denoted by $\boldsymbol{x}_i^p$. We also take one of the **dissimilar** points to the anchor point as the **negative (or distant) point**, denoted by $\boldsymbol{x}_i^n$.
- If class labels are available, we can use them to find the **positive point as one of the points in the same class as the anchor point**, and to find the the **negative point as one of the points in a different class from the anchor point's class**.
- Another approach is to **augment the anchor point**, using one of the augmentation methods, to obtain **positive** points for the anchor point [13, 14]. processing [12].

Pairs and Triplets of Data Points

- For Siamese networks with **two sub-networks**, we make pairs of **anchor-positive points** $\{(\boldsymbol{x}_i^a, \boldsymbol{x}_i^p)\}_{i=1}^{n_t}$ and **anchor-negative points** $\{(\boldsymbol{x}_i^a, \boldsymbol{x}_i^n)\}_{i=1}^{n_t}$, where n_t is the number of pairs.
- For Siamese networks with **three sub-networks**, we make **triplets of anchor-positive-negative points** $\{(\boldsymbol{x}_i^a, \boldsymbol{x}_i^p, \boldsymbol{x}_i^n)\}_{i=1}^{n_t}$, where n_t is the number of triplets. If we consider **every point of dataset as an anchor**, the number of pairs/triplets is the same as the number of data points, i.e., $n_t = n$.
- Various **loss functions** of Siamese networks use **pairs or triplets of data points** to **push the positive point towards the anchor point** and **pull the negative point away from it**. Doing this iteratively for all pairs or triplets will make the **intra-class variances smaller** and the **inter-class variances larger** for better discrimination of classes or clusters.

Implementation of Siamese Networks

- A Siamese network with two and three sub-networks is depicted below.



- We denote the output of Siamese network for input $\boldsymbol{x} \in \mathbb{R}^d$ by $\mathbf{f}(\boldsymbol{x}) \in \mathbb{R}^p$ where p is the dimensionality of embedding (or the number of neurons at the last layer of the network) which is usually much less than the dimensionality of data, i.e., $p \ll d$.
- Note that the sub-networks of a Siamese network can be any **fully-connected or convolutional network** depending on the type of data. The used network structure for the sub-networks is usually called the **backbone network**.

Implementation of Siamese Networks

- The **weights** of sub-networks are **shared** in the sense that the values of their weights are equal.
- Implementation of a Siamese network can be done in two ways:
  1. We can implement **several sub-networks in the memory**. In the training phase, we **feed every data point in the pairs or triplets to one of the sub-networks** and take the outputs of sub-networks to have $\mathbf{f}(\boldsymbol{x}_i^a)$, $\mathbf{f}(\boldsymbol{x}_i^p)$, and $\mathbf{f}(\boldsymbol{x}_i^n)$. We use these in the loss function and **update the weights of only one of the sub-networks by backpropagation** [15]. Then, we **copy the updated weights to the other sub-networks**. We repeat this for all mini-batches and epochs until convergence. In the test phase, we feed the test point $\boldsymbol{x}$ to only one of the sub-networks and get the output $\mathbf{f}(\boldsymbol{x})$ as its embedding.
  2. We can implement **only one sub-network in the memory**. In the training phase, we feed the data points in the pairs or triplets to the sub-network one by one and take the outputs of sub-network to have $\mathbf{f}(\boldsymbol{x}_i^a)$, $\mathbf{f}(\boldsymbol{x}_i^p)$, and $\mathbf{f}(\boldsymbol{x}_i^n)$. We use these in the loss function and **update the weights of the sub-network by backpropagation** [15]. We repeat this for all mini-batches and epochs until convergence. In the test phase, we feed the test point $\boldsymbol{x}$ to the sub-network and get the output $\mathbf{f}(\boldsymbol{x})$ as its embedding.
- The advantage of the first approach is to have all the sub-networks ready and we do not need to feed the points of pairs or triplets one by one. Its disadvantage is using more memory. As the number of points in the pairs or triplets is small (i.e., only two or three), the **second approach is more recommended** as it is **memory-efficient**.

Loss Functions for Siamese Networks

Contrastive Loss

- One loss function for Siamese networks is the **contrastive loss** which uses the **anchor-positive and anchor-negative pairs** of points.
- Suppose, in each mini-batch, we have b pairs of points $\{(\boldsymbol{x}_i^1, \boldsymbol{x}_i^2)\}_{i=1}^b$ some of which are anchor-positive and some are anchor-negative pairs.
- The points in an anchor-positive pair are similar, i.e. $(\boldsymbol{x}_i^1, \boldsymbol{x}_i^2) \in \mathcal{S}$, and the points in an anchor-negative pair are dissimilar, i.e. $(\boldsymbol{x}_i^1, \boldsymbol{x}_i^2) \in \mathcal{D}$, where $\mathcal{S}$ and $\mathcal{D}$ denote the similar and dissimilar sets.
- We define:

$$y_i := \begin{cases} 0 & \text{if } (\boldsymbol{x}_i^1, \boldsymbol{x}_i^2) \in \mathcal{S} \\ 1 & \text{if } (\boldsymbol{x}_i^1, \boldsymbol{x}_i^2) \in \mathcal{D}. \end{cases} \qquad \forall i \in \{1, \ldots, n_t\}. \tag{10}$$

- The **main contrastive loss** was proposed in (2006) [16] and is:

$$\underset{\theta}{\text{minimize}} \quad \sum_{i=1}^{b} \Big((1 - y_i) d\big(\mathbf{f}(\boldsymbol{x}_i^1), \mathbf{f}(\boldsymbol{x}_i^2)\big) + y_i \big[- d\big(\mathbf{f}(\boldsymbol{x}_i^1), \mathbf{f}(\boldsymbol{x}_i^2)\big) + m \big]_+ \Big), \tag{11}$$

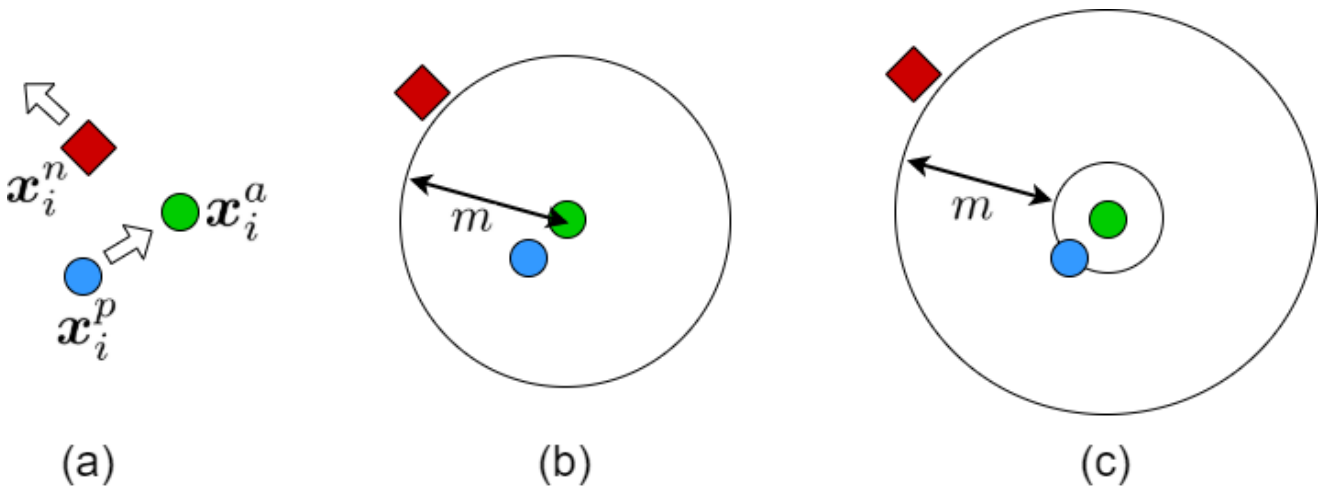
where $m > 0$ is the margin and $[.]_+ := \max(., 0)$ is the standard Hinge loss.

Contrastive Loss

- We had:

$$\underset{\theta}{\text{minimize}} \quad \sum_{i=1}^{b} \Big((1 - y_i) d\big(\mathbf{f}(\boldsymbol{x}_i^1), \mathbf{f}(\boldsymbol{x}_i^2)\big) + y_i \big[- d\big(\mathbf{f}(\boldsymbol{x}_i^1), \mathbf{f}(\boldsymbol{x}_i^2)\big) + m \big]_+ \Big).$$

- The first term of loss **minimizes the embedding distances of similar points** and the second term **maximizes the embedding distances of dissimilar points**.
- As shown in this figure, it tries to make the distances of similar points as small as possible and the distances of dissimilar points at least greater than a margin m (because the term inside the Hinge loss should become close to zero).



Triplet Loss

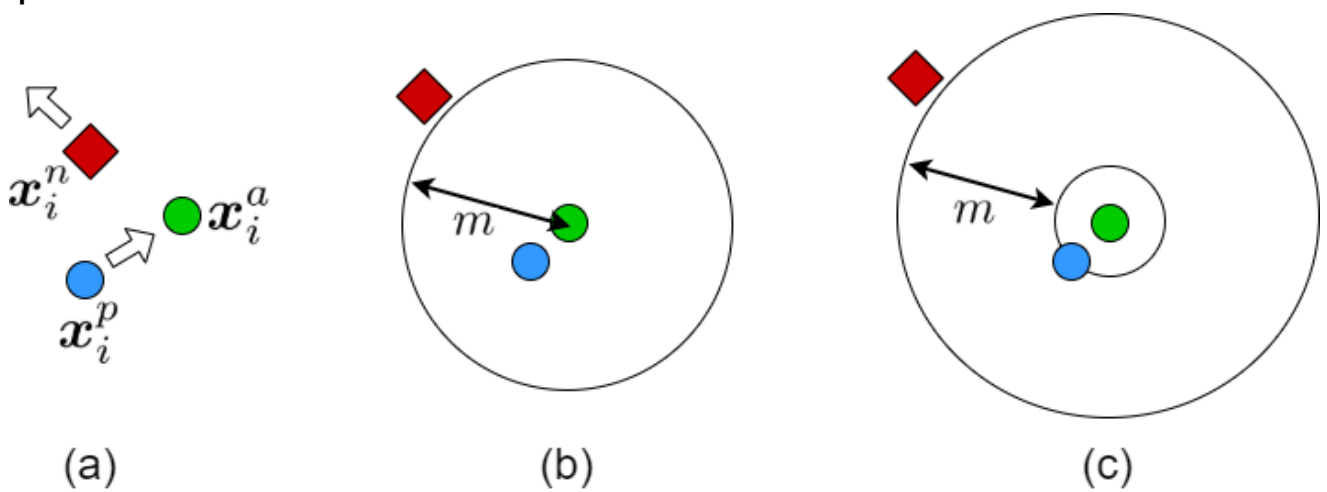
- One of the losses for Siamese networks with three sub-networks is the **triplet loss** (2015) [11] which uses the triplets in mini-batches, denoted by $\{(\boldsymbol{x}_i^a, \boldsymbol{x}_i^p, \boldsymbol{x}_i^n)\}_{i=1}^b$. It is defined as:

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^{b} \Big[d\big(\mathbf{f}(\boldsymbol{x}_i^a), \mathbf{f}(\boldsymbol{x}_i^p)\big) - d\big(\mathbf{f}(\boldsymbol{x}_i^a), \mathbf{f}(\boldsymbol{x}_i^n)\big) + m \Big]_+, \tag{12}$$

  where $m > 0$ is the margin and $[.]_+ := \max(., 0)$ is the standard Hinge loss.
- As shown in this figure, because of the used Hinge loss, this loss makes the **distances of dissimilar points greater than the distances of similar points by at least a margin** m; in other words, there will be a **distance of at least margin** m **between the positive and negative points**.
- This loss desires to eventually have:

$$d\big(\mathbf{f}(\boldsymbol{x}_i^a), \mathbf{f}(\boldsymbol{x}_i^p)\big) + m \leq d\big(\mathbf{f}(\boldsymbol{x}_i^a), \mathbf{f}(\boldsymbol{x}_i^n)\big), \tag{13}$$

  for all triplets.



Neighborhood Component Analysis Loss

- **Neighborhood Component Analysis (NCA)** (2005) [17] was originally proposed as a spectral metric learning method.
- After the success of deep learning, it was used as the **loss function of Siamese networks** where we **minimize the negative log-likelihood using Gaussian distribution** or the **softmax form** within the mini-batch.
- Assume we have c classes in every mini-batch. We denote the class index of $\boldsymbol{x}_i$ by $c(\boldsymbol{x}_i)$ and the data points of the j-th class in the mini-batch by $\mathcal{X}_j$. The NCA loss is:

$$\underset{\theta}{\text{minimize}} \quad -\sum_{i=1}^{b} \log\Bigg(\frac{\exp\Big(-d\big(\mathbf{f}(\boldsymbol{x}_i^a), \mathbf{f}(\boldsymbol{x}_i^p)\big)\Big)}{\sum_{j=1, j\neq c(\boldsymbol{x}_i)}^{c} \sum_{\boldsymbol{x}_j^n \in \mathcal{X}_j} \exp\Big(-d\big(\mathbf{f}(\boldsymbol{x}_i^a) - \mathbf{f}(\boldsymbol{x}_j^n)\big)\Big)}\Bigg). \tag{14}$$

- The numerator **minimizes the distances of similar points** and the denominator **maximizes the distances of dissimilar points**.
- For more information on other losses for Siamese networks, see our tutorial paper "Spectral, probabilistic, and deep metric learning: Tutorial and survey" [18].

Triplet Mining

Triplet Mining

- In every mini-batch containing data points from c classes, we can **select and use triplets** of data points in different ways.
- For example, we can use **all similar and dissimilar points** for every anchor point as positive and negative points, respectively.
- Another approach is to only use **some of the similar and dissimilar points** within the mini-batch. These approaches for selecting and using triplets are called **triplet mining** [19].
- Suppose b is the mini-batch size, $c(\boldsymbol{x}_i)$ is the class index of $\boldsymbol{x}_i$, $\mathcal{X}_j$ denotes the points of the j-th class in the mini-batch, and $\mathcal{X}$ denotes the data points in the mini-batch.
- **Batch-all:** Batch-all triplet mining (2015) [20] considers every point in the mini-batch as an anchor point. **All points** in the mini-batch which are in the **same class** as the anchor point are used as **positive** points. **All points** in the mini-batch which are in a **different class** from the class of anchor point are used as **negative** points:

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^{b} \sum_{\boldsymbol{x}_j \in \mathcal{X}_{c(\boldsymbol{x}_i)}} \sum_{\boldsymbol{x}_k \in \mathcal{X} \setminus \mathcal{X}_{c(\boldsymbol{x}_i)}} \Big[d\big(\mathbf{f}(\boldsymbol{x}_i), \mathbf{f}(\boldsymbol{x}_j)\big) - d\big(\mathbf{f}(\boldsymbol{x}_i), \mathbf{f}(\boldsymbol{x}_k)\big) + m \Big]_+ . \tag{15}$$

- Batch-all mining makes use of all data points in the mini-batch to utilize **all available information**.

Triplet Mining

- **Batch-hard:** Batch-hard triplet mining (2017) [21] considers every point in the mini-batch as an anchor point. The **hardest positive**, which is the **farthest point from the anchor point in the same class**, is used as the positive point. The **hardest negative**, which is the **closest point to the anchor point from another class**, is used as the negative point:

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^{b} \Big[\max_{\boldsymbol{x}_j \in \mathcal{X}_{c(\boldsymbol{x}_i)}} d\big(\mathbf{f}(\boldsymbol{x}_i), \mathbf{f}(\boldsymbol{x}_j)\big) - \min_{\boldsymbol{x}_k \in \mathcal{X} \setminus \mathcal{X}_{c(\boldsymbol{x}_i)}} d\big(\mathbf{f}(\boldsymbol{x}_i), \mathbf{f}(\boldsymbol{x}_k)\big) + m \Big]_+ . \tag{16}$$

- Bath-hard mining uses hardest points so that the network learns the hardest cases. By learning the **hardest cases**, **other cases are expected to be learned properly**. Learning the hardest cases can also be justified by the **opposition-based learning** [22].
- **Batch-semi-hard:** Batch-semi-hard triplet mining (2015) [11] considers **every point** in the mini-batch as an **anchor** point. **All points** in the mini-batch which are in the **same class** as the anchor point are used as **positive** points. The **hardest negative** (closest to the anchor point from another class), which is **farther than the positive point**, is used as the **negative** point:

$$\begin{aligned} \underset{\theta}{\text{minimize}} \sum_{i=1}^{b} \sum_{\boldsymbol{x}_j \in \mathcal{X}_{c(\boldsymbol{x}_i)}} \Big[d\big(\mathbf{f}(\boldsymbol{x}_i), \mathbf{f}(\boldsymbol{x}_j)\big) - \min_{\boldsymbol{x}_k \in \mathcal{X} \setminus \mathcal{X}_{c(\boldsymbol{x}_i)}} \big\{ d\big(\mathbf{f}(\boldsymbol{x}_i), \mathbf{f}(\boldsymbol{x}_k)\big) \mid \\ d\big(\mathbf{f}(\boldsymbol{x}_i), \mathbf{f}(\boldsymbol{x}_k)\big) > d\big(\mathbf{f}(\boldsymbol{x}_i), \mathbf{f}(\boldsymbol{x}_j)\big) \big\} + m \Big]_+ . \end{aligned} \tag{17}$$

Triplet Mining

- **Easy-positive:** Easy-positive triplet mining (2020) [23] considers **every point** in the mini-batch as an **anchor** point. The **easiest positive** (closest to the anchor point from the same class) is used as the **positive** point. **All points** in the mini-batch which are in a **different class** from the class of anchor point are used as **negative** points:

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^{b} \sum_{\boldsymbol{x}_k \in \mathcal{X} \setminus \mathcal{X}_{c(\boldsymbol{x}_i)}} \Big[\min_{\boldsymbol{x}_j \in \mathcal{X}_{c(\boldsymbol{x}_i)}} d\big(\mathbf{f}(\boldsymbol{x}_i), \mathbf{f}(\boldsymbol{x}_j)\big) - d\big(\mathbf{f}(\boldsymbol{x}_i), \mathbf{f}(\boldsymbol{x}_k)\big) + m \Big]_+ . \tag{18}$$

- We can use this triplet mining approach in NCA loss function. For example, we can have [23]:

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^{b} \frac{\min_{\boldsymbol{x}_j \in \mathcal{X}_{c(\boldsymbol{x}_i)}} \exp\left(\mathbf{f}(\boldsymbol{x}_i)^\top \mathbf{f}(\boldsymbol{x}_j)\right)}{\min_{\boldsymbol{x}_j \in \mathcal{X}_{c(\boldsymbol{x}_i)}} \exp\left(\mathbf{f}(\boldsymbol{x}_i)^\top \mathbf{f}(\boldsymbol{x}_j)\right) + \sum_{\boldsymbol{x}_k \in \mathcal{X} \setminus \mathcal{X}_{c(\boldsymbol{x}_i)}} \exp\left(\mathbf{f}(\boldsymbol{x}_i)^\top \mathbf{f}(\boldsymbol{x}_k)\right)}, \tag{19}$$

  where the embeddings for all points of the mini-batch are normalized to have length one.

Triplet Sampling

- Rather than using the **extreme (hardest or easiest)** positive and negative points [19], we can **sample positive and negative points from the points in the mini-batch or from some distributions**.
- There are several approaches for the positive and negative points to be sampled [24]:
  - Sampled by **extreme distances** of points,
  - Sampled **randomly from classes**,
  - Sampled by **distribution** but **from existing points**,
  - Sampled stochastically from **distributions of classes**.
- The **first, second, and third** approaches sample the positive and negative points **from the set of points in the mini-batch**. This type of sampling is called **survey sampling** [25].
- The **third and fourth** approaches sample points from **distributions** stochastically.
- An example for the third approach, i.e., sampling by distribution but from existing points, is **distance weighted sampling** (2017) [26]. An example for the forth approach is **Bayesian updating theorem** (2021) [27].

Acknowledgment

- This slide deck is based on our tutorial paper "Spectral, probabilistic, and deep metric learning: Tutorial and survey" [18].
- For more information on deep metric learning, refer to our tutorial paper [18].

References

[1] B. Ghojogh, A. Ghodsi, F. Karray, and M. Crowley, "Factor analysis, probabilistic principal component analysis, variational inference, and variational autoencoder: Tutorial and survey," *arXiv preprint arXiv:2101.00734*, 2021.

[2] B. Ghojogh, A. Ghodsi, F. Karray, and M. Crowley, "Generative adversarial networks and adversarial autoencoders: Tutorial and survey," *arXiv preprint arXiv:2111.13282*, 2021.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[4] B. Ghojogh and M. Crowley, "Unsupervised and supervised principal component analysis: Tutorial," *arXiv preprint arXiv:1906.03148*, 2019.

[5] B. Ghojogh, A. Ghodsi, F. Karray, and M. Crowley, "Reproducing kernel Hilbert space, Mercer's theorem, eigenfunctions, Nyström method, and use of kernels in machine learning: Tutorial and survey," *arXiv preprint arXiv:2106.08443*, 2021.

[6] V. Vapnik, *The nature of statistical learning theory*. Springer science & business media, 1995.

[7] M. Sikaroudi, A. Safarpoor, B. Ghojogh, S. Shafiei, M. Crowley, and H. R. Tizhoosh, "Supervision and source domain impact on representation learning: A histopathology case study," in *2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, pp. 1400–1403, IEEE, 2020.

References (cont.)

[8] M. Boudiaf, J. Rony, I. M. Ziko, E. Granger, M. Pedersoli, P. Piantanida, and I. B. Ayed, "A unifying mutual information view of metric learning: cross-entropy vs. pairwise losses," in *European Conference on Computer Vision*, pp. 548–564, Springer, 2020.

[9] J. Bromley, J. W. Bentz, L. Bottou, I. Guyon, Y. LeCun, C. Moore, E. Säckinger, and R. Shah, "Signature verification using a âSiameseâ time delay neural network," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 7, no. 04, pp. 669–688, 1993.

[10] E. Hoffer and N. Ailon, "Deep metric learning using triplet network," in *International workshop on similarity-based pattern recognition*, pp. 84–92, Springer, 2015.

[11] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 815–823, 2015.

[12] L. Yang, M. Zhang, C. Li, M. Bendersky, and M. Najork, "Beyond 512 tokens: Siamese multi-depth transformer-based hierarchical encoder for long-form document matching," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pp. 1725–1734, 2020.

[13] S. Khodadadeh, L. Bölöni, and M. Shah, "Unsupervised meta-learning for few-shot image classification," in *Advances in neural information processing systems*, 2019.

References (cont.)

[14] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International conference on machine learning*, pp. 1597–1607, 2020.

[15] B. Ghojogh, A. Ghodsi, F. Karray, and M. Crowley, "KKT conditions, first-order and second-order optimization, and distributed optimization: Tutorial and survey," *arXiv preprint arXiv:2110.01858*, 2021.

[16] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 2, pp. 1735–1742, IEEE, 2006.

[17] J. Goldberger, G. E. Hinton, S. T. Roweis, and R. R. Salakhutdinov, "Neighbourhood components analysis," in *Advances in neural information processing systems*, pp. 513–520, 2005.

[18] B. Ghojogh, A. Ghodsi, F. Karray, and M. Crowley, "Spectral, probabilistic, and deep metric learning: Tutorial and survey," *arXiv preprint arXiv:2201.09267*, 2022.

[19] M. Sikaroudi, B. Ghojogh, A. Safarpoor, F. Karray, M. Crowley, and H. R. Tizhoosh, "Offline versus online triplet mining based on extreme distances of histopathology patches," in *International Symposium on Visual Computing*, pp. 333–345, Springer, 2020.

References (cont.)

[20] S. Ding, L. Lin, G. Wang, and H. Chao, "Deep feature learning with relative distance comparison for person re-identification," *Pattern Recognition*, vol. 48, no. 10, pp. 2993–3003, 2015.

[21] A. Hermans, L. Beyer, and B. Leibe, "In defense of the triplet loss for person re-identification," *arXiv preprint arXiv:1703.07737*, 2017.

[22] H. R. Tizhoosh, "Opposition-based learning: a new scheme for machine intelligence," in *International conference on computational intelligence for modelling, control and automation and international conference on intelligent agents, web technologies and internet commerce (CIMCA-IAWTIC'06)*, vol. 1, pp. 695–701, IEEE, 2005.

[23] H. Xuan, A. Stylianou, and R. Pless, "Improved embeddings with easy positive triplet mining," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 2474–2482, 2020.

[24] B. Ghojogh, *Data Reduction Algorithms in Machine Learning and Data Science*. PhD thesis, University of Waterloo, 2021.

[25] B. Ghojogh, H. Nekoei, A. Ghojogh, F. Karray, and M. Crowley, "Sampling algorithms, from survey sampling to Monte Carlo methods: Tutorial and literature review," *arXiv preprint arXiv:2011.00901*, 2020.

References (cont.)

[26] C.-Y. Wu, R. Manmatha, A. J. Smola, and P. Krahenbuhl, "Sampling matters in deep embedding learning," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2840–2848, 2017.

[27] M. Sikaroudi, B. Ghojogh, F. Karray, M. Crowley, and H. R. Tizhoosh, "Batch-incremental triplet sampling for training triplet networks using Bayesian updating theorem," in *2020 25th International Conference on Pattern Recognition (ICPR)*, pp. 7080–7086, IEEE, 2021.