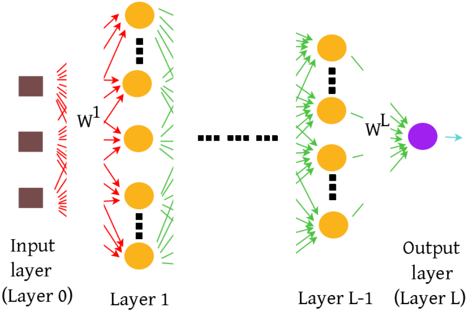
Deep Learning - Theory and Practice

IE 643
Lectures 9, 10

August 30 & Sep 2, 2022.

1. Recap
   - MLP for prediction tasks
2. MLP for multi-class classification
   - Cross-entropy
   - Training MLP for multi-class classification
3. MLP for multi-label classification

Multi Layer Perceptron for Prediction Tasks

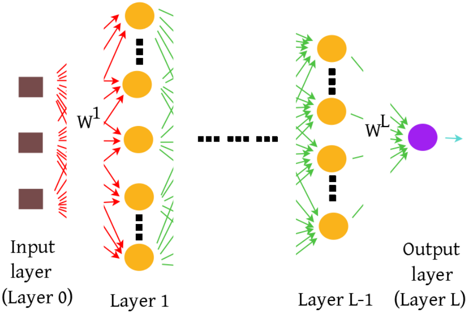


- **Input:** Training Data $D = \{(x^i, y^i)\}_{i=1}^{S}$, $x^i \in \mathcal{X} \subseteq \mathbb{R}^d$, $y \in \mathcal{Y}$, $\forall i \in \{1, \ldots, S\}$ and MLP architecture parametrized by weights w.

- **Aim of training MLP:** To learn a parametrized map $h_w : \mathcal{X} \to \mathcal{Y}$ such that for the training data D, we have $y^i = h_w(x^i)$, $\forall i \in \{1, \ldots, S\}$.

- **Aim of using the trained MLP model:** For an unseen sample $\hat{x} \in \mathcal{X}$, predict $\hat{y} = h_w(\hat{x}) = MLP(\hat{x}; w)$.

Multi Layer Perceptron for Prediction Tasks



Methodology for training MLP

- Design a suitable loss (or error) function $e : \mathcal{Y} \times \mathcal{Y} \to [0, +\infty)$ to compare the actual label y^i and the prediction $\hat{y}^i$ made by MLP using $e(y^i, \hat{y}^i)$, $\forall i \{1, \dots, S\}$.
- Usually the error is parametrized by the weights w of the MLP and is denoted by $e(\hat{y}^i, y^i; w)$.
- Use Gradient descent/SGD/mini-batch SGD to minimize the total error:

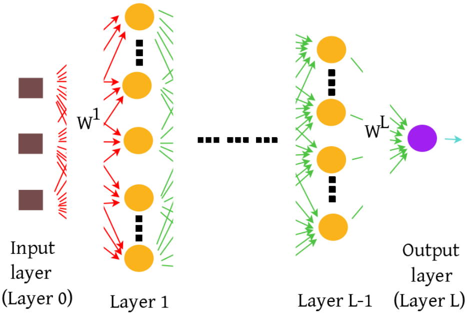
$$E = \sum_{i=1}^{S} e(\hat{y}^i, y^i; w) =: \sum_{i=1}^{S} e^i(w).$$

Stochastic Gradient Descent for training MLP

SGD Algorithm to train MLP

- **Input:** Training Data $D = \{(x^i, y^i)\}_{i=1}^S$, $x^i \in \mathcal{X} \subseteq \mathbb{R}^d$, $y^i \in \mathcal{Y}$, $\forall i$; MLP architecture, max epochs K, learning rates γ_k, $\forall k \in \{1, \dots, K\}$.
- Start with $w^0 \in \mathbb{R}^d$.
- For $k = 0, 1, 2, \dots, K$
  - Choose a sample $j_k \in \{1, \dots, S\}$.
  - Find $\hat{y}^{j_k} = \text{MLP}(x^{j_k}; w^k)$. (forward pass)
  - Compute error $e^{j_k}(w^k)$.
  - Compute error gradient $\nabla_w e^{j_k}(w^k)$ using backpropagation.
  - Update: $w^{k+1} \leftarrow w^k - \gamma_k \nabla_w e^{j_k}(w^k)$.
- **Output:** $w^* = w^{K+1}$.

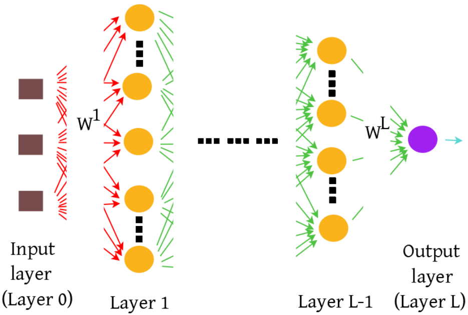
Multi Layer Perceptron for Prediction Tasks



Recall forward pass: For an arbitrary sample (x, y) from training data D, and the MLP with weights $w = (W^1, W^2 \ldots, W^L)$, the prediction $\hat{y}$ is computed using forward pass as:

$$\hat{y} = \text{MLP}(x; w) = \phi(W^L \phi(W^{L-1} \ldots \phi(W^1 x) \ldots)).$$

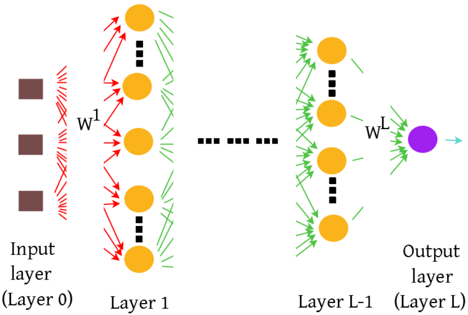
Multi Layer Perceptron for Prediction Tasks



Recall backpropagation: For an arbitrary sample (x, y) from training data D, and the MLP with weights $w = (W^1, W^2 \dots, W^L)$, the error gradient with respect to weights at ℓ-th layer is computed as:

$$\nabla_{W^\ell} e = \text{Diag}(\phi^{\ell'})\delta^\ell(a^{\ell-1})^\top$$

Multi Layer Perceptron for Prediction Tasks

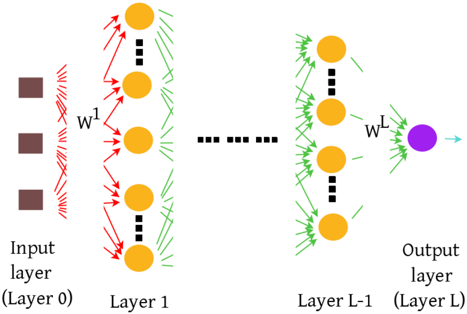


Recall backpropagation: For an arbitrary sample (x, y) from training data D, and the MLP with weights $w = (W^1, W^2 \dots, W^L)$, the error gradient with respect to weights at ℓ-th layer is computed as:

$$\nabla_{W^\ell} e = \text{Diag}(\phi^{\ell\prime})\delta^\ell (a^{\ell-1})^\top$$

where $\text{Diag}(\phi^{\ell\prime}) = \begin{bmatrix} \phi'(z_1^\ell) & & \\ & \ddots & \\ & & \phi'(z_{N_\ell}^\ell) \end{bmatrix}$, $\delta^\ell = \begin{bmatrix} \frac{\partial e}{\partial a_1^\ell} \\ \vdots \\ \frac{\partial e}{\partial a_{N_\ell}^\ell} \end{bmatrix}$ and $a^{\ell-1} = \begin{bmatrix} a_1^{\ell-1} \\ \vdots \\ a_{N_{\ell-1}}^{\ell-1} \end{bmatrix}$.

Multi Layer Perceptron for Prediction Tasks

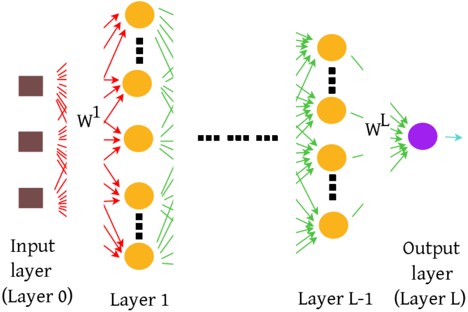


Recall backpropagation: For an arbitrary sample (x, y) from training data D, and the MLP with weights $w = (W^1, W^2 \ldots, W^L)$, the error gradient with respect to weights at ℓ-th layer is computed as:

$$\nabla_{W^\ell} e = \mathrm{Diag}(\phi^{\ell\prime})\delta^\ell(a^{\ell-1})^\top = \mathrm{Diag}(\phi^{\ell\prime})V^{\ell+1}V^{\ell+2}\ldots V^L\delta^L(a^{\ell-1})^\top$$

where $V^{\ell+1} = (W^{\ell+1})^\top \mathrm{Diag}(\phi^{\ell+1\prime})$.

Multi Layer Perceptron for Prediction Tasks



- **Task considered so far:** $\mathcal{Y} = \{+1, -1\}$.
- Corresponds to two-class (or binary) classification.
- Usually a single neuron at the last (L-th) layer of MLP, with logistic sigmoid function $\sigma : \mathbb{R} \to (0, 1)$ with $\sigma(z) = \frac{1}{1+e^{-z}}$, for some $z \in \mathbb{R}$.
- **Prediction:** $\text{MLP}(\hat{x}; w) = \sigma(W^L a^{L-1})$, followed by a thresholding function.

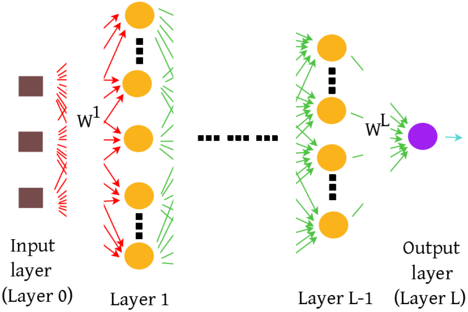
MLP for multi-class classification

- **Input:** Training Data $D = \{(x^i, y^i)\}_{i=1}^{S}$, $x^i \in \mathcal{X} \subseteq \mathbb{R}^d$, $y^i \in \mathcal{Y}$, $\forall i \in \{1, \ldots, S\}$ and MLP architecture parametrized by weights w.
- **New Task:** $\mathcal{Y} = \{1, \ldots, C\}, C \geq 2$.
- Corresponds to multi-class classification.

Question 1: What is a suitable architecture for the MLP's last (or output) layer?

Question 2: What is a suitable loss (or error) function?

MLP for multi-class classification



Question 1: Can the same MLP architecture with single output neuron used in binary classification be used for multi-class classification?

Question 2: Can the same logistic sigmoidal activation function for the output neuron used in binary classification be used for multi-class classification?

MLP for multi-class classification

We will use the following approach for multi-class classification:

- **Input:** Training Data $D = \{(x^i, y^i)\}_{i=1}^S$, $x^i \in \mathcal{X} \subseteq \mathbb{R}^d$, $y^i \in \mathcal{Y}$, $\forall i \in \{1, \dots, S\}$ and MLP architecture parametrized by weights w.

- **New Task:** $\mathcal{Y} = \{1, \dots, C\}, C \geq 2$ corresponds to multi-class classification.

- Transform $y = c$ to $y^{onehotenc} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$.

- **Note:** $y^{onehotenc} \in \{0, 1\}^C$ corresponding to $y = c \in \mathcal{Y}$ has a 1 at c-th coordinate, and other entries as zeros.

- $y^{onehotenc}$ is called the one-hot encoding of y.

MLP for multi-class classification

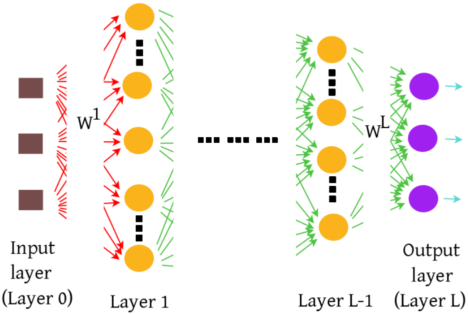
We will use the following approach for multi-class classification:

- **Input:** Training Data $D = \{(x^i, y^i)\}_{i=1}^{S}$, $x^i \in \mathcal{X} \subseteq \mathbb{R}^d$, $y^i \in \mathcal{Y}$, $\forall i \in \{1, \ldots, S\}$ and MLP architecture parametrized by weights w.
- **New Task:** $\mathcal{Y} = \{1, \ldots, C\}, C \geq 2$ corresponds to multi-class classification.
- Transform $y = c$ to $y^{onehotenc} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$.
- **Note:** $y^{onehotenc} \in \{0, 1\}^C$ corresponding to $y = c \in \mathcal{Y}$ has a 1 at c-th coordinate, and other entires as zeros.
- $y^{onehotenc}$ is called the one-hot encoding of y.
- $y^{onehotenc}$ for $y = c$ corresponds to a discrete probability distribution with its entire mass concentrated at the c-th coordinate.

MLP for multi-class classification

What change can be made to the network architecture so that the MLP outputs a discrete probability distribution?

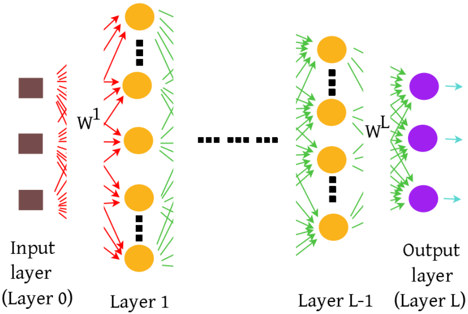
Step 1: Since $\mathcal{Y} = \{1, \ldots, C\}$, output layer of MLP to contain C neurons.



MLP for multi-class classification

What change can be made to the network architecture so that the MLP outputs a discrete probability distribution?

Step 1: Since $\mathcal{Y} = \{1, \ldots, C\}$, output layer of MLP to contain C neurons.

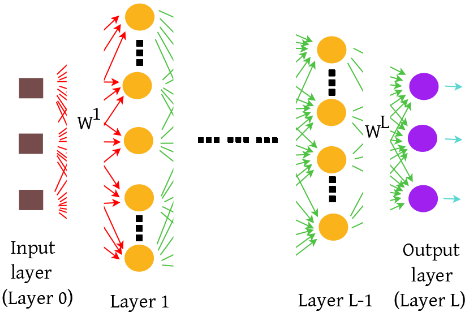


However activation functions of output neurons might be arbitrary!

MLP for multi-class classification

What change can be made to the network architecture so that the MLP outputs a discrete probability distribution?

Step 1: Since $\mathcal{Y} = \{1, \ldots, C\}$, output layer of MLP to contain C neurons.



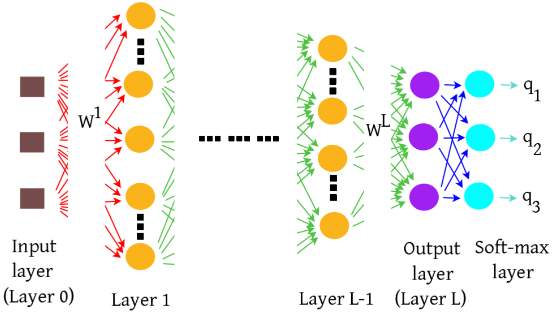
However activation functions of output neurons might be arbitrary!

How do we get probabilities as outputs?

MLP for multi-class classification

How do we get probabilities as outputs?

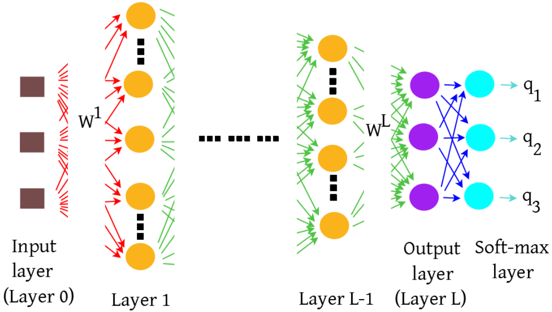
Step 2: Perform a soft-max function over the outputs from output layer so that the outputs are transformed into probabilities.



MLP for multi-class classification

How do we get probabilities as outputs?

Step 2: Perform a soft-max function over the outputs from output layer so that the outputs are transformed into probabilities.



What is a soft-max function?

MLP for multi-class classification

What is a soft-max function?

- Given arbitrary activations $a_1^L, a_2^L, \ldots, a_C^L$ from an output layer (L-th layer), how do we get probabilities?
- Perform the following transformation:

$$q_j = \frac{\exp(a_j^L)}{\sum_{r=1}^{C} \exp\left(a_r^L\right)}, \ \forall j = 1, \ldots, C.$$

- $q_1, \ldots, q_C$ form a discrete probability distribution. **(Verify this claim!)**

The transformation used to obtain the probabilities q_j is called the soft-max function.

MLP for multi-class classification

Now that the MLP outputs a discrete probability distribution, how do we compare the one-hot encoding and the output distribution?

- We will use the popular divergence measure called Kullback-Leibler divergence (or KL-divergence).
- Given two discrete probability distributions $p = (p_1, \ldots, p_C)$ and $q = (q_1, \ldots, q_C)$, where $q_j > 0 \ \forall j = 1, \ldots, C$, KL-divergence between p and q is defined as:

$$KL(p||q) = \sum_{j=1}^{C} p_j \log \frac{p_j}{q_j}.$$

- **Note:** The distribution p is usually called the true distribution and the distribution q is called the predicted distribution.
- Does the soft-max function give predictions $q_j > 0$, $j = 1, \ldots, C$?

MLP for multi-class classification: KL Divergence

Some useful properties of KL-divergence:

- KL-divergence is **not** a distance function.

MLP for multi-class classification: KL Divergence

Some useful properties of KL-divergence:

- KL-divergence is **not** a distance function.
- **Recall:** A distance function $d : X \times X \rightarrow [0, \infty)$ has the following properties:
  - $d(x, x) = 0, \ \forall x \in X$ (identity of indistinguishables)
  - $d(x, y) = d(y, x), \forall x, y \in X$ (Symmetry)
  - $d(x, z) \leq d(x, y) + d(y, z), \forall x, y, z \in X$ (triangle inequality)

MLP for multi-class classification: KL Divergence

Some useful properties of KL-divergence:

- KL-divergence is **not** a distance function.
- **Recall:** A distance function $d : X \times X \to [0, \infty)$ has the following properties:
  - $d(x, x) = 0, \ \forall x \in X$ (identity of indistinguishables)
  - $d(x, y) = d(y, x), \forall x, y \in X$ (Symmetry)
  - $d(x, z) \leq d(x, y) + d(y, z), \forall x, y, z \in X$ (triangle inequality)
- KL-divergence does not obey symmetry property.
  - Simple example: compute $KL(p||q)$ and $KL(q||p)$ for $p = (1/4, 3/4)$ and $q = (1/2, 1/2)$.

MLP for multi-class classification: KL Divergence

Some useful properties of KL-divergence:

- KL-divergence is **not** a distance function.
- **Recall:** A distance function $d : X \times X \to [0, \infty]$ has the following properties:
  - $d(x, x) = 0, \ \forall x \in X$ (identity of indistinguishables)
  - $d(x, y) = d(y, x), \forall x, y \in X$ (Symmetry)
  - $d(x, z) \leq d(x, y) + d(y, z), \forall x, y, z \in X$ (triangle inequality)
- Does KL-divergence obey triangle inequality?

MLP for multi-class classification: KL Divergence

Some useful properties of KL-divergence:

- For two discrete probability distributions $p = (p_1, p_2, \ldots, p_C)$ and $q = (q_1, q_2, \ldots, q_C)$, $q_j > 0,\ \forall j = 1, \ldots, C$, $KL(p||q) \geq 0$.

MLP for multi-class classification: KL Divergence

KL-Divergence: Equivalent Representation

- Given two discrete probability distributions $p = (p_1, \ldots, p_C)$ and $q = (q_1, \ldots, q_C)$, where $q_j > 0 \ \forall j = 1, \ldots, C$, KL-divergence between p and q is defined as:

$$KL(p||q) = \sum_{j=1}^{C} p_j \log \frac{p_j}{q_j} = \sum_{j=1}^{C} p_j \log p_j - \sum_{j=1}^{C} p_j \log q_j.$$

MLP for multi-class classification: KL Divergence

KL-Divergence: Equivalent Representation

- Given two discrete probability distributions $p = (p_1, \ldots, p_C)$ and $q = (q_1, \ldots, q_C)$, where $q_j > 0 \; \forall j = 1, \ldots, C$, KL-divergence between p and q is defined as:

$$KL(p||q) = \sum_{j=1}^{C} p_j \log \frac{p_j}{q_j} = \sum_{j=1}^{C} p_j \log p_j - \sum_{j=1}^{C} p_j \log q_j.$$

  Note: $\sum_{j=1}^{C} p_j \log p_j$ is called negative entropy associated with distribution p (denoted by $NE(p)$) and $-\sum_{j=1}^{C} p_j \log q_j$ is called cross-entropy between p and q (denoted by $CE(p, q)$).

- Hence $KL(p||q) = NE(p) + CE(p, q)$.

MLP for multi-class classification

Question:

- Why should we transform $y \in \mathcal{Y}$ taking an integer value, to a $y^{onehotenc}$ which represents a discrete probability distribution?

One possible answer:

- If the prediction $\hat{y}$ made by MLP is also a discrete probability distribution, then the comparison between $\hat{y}$ and $y^{onehotenc}$ becomes a comparison between two discrete probability distributions.
- Multiple ways to compare two probability distributions.
- Loss function design becomes possibly simpler?

MLP for multi-class classification

Loss function using KL-Divergence:

- Given two discrete probability distributions $p = (p_1, \ldots, p_C)$ and $q = (q_1, \ldots, q_C)$, where $q_j > 0 \ \forall j = 1, \ldots, C$, KL-divergence between p and q is defined as:

$$\begin{aligned} KL(p||q) = \sum_{j=1}^{C} p_j \log \frac{p_j}{q_j} &= \sum_{j=1}^{C} p_j \log p_j - \sum_{j=1}^{C} p_j \log q_j. \\ &= NE(p) + CE(p, q). \end{aligned}$$

- Our aim is to minimize the error measured using KL-divergence between the true distribution p and the predicted distribution q.

MLP for multi-class classification

Loss function using KL-Divergence:

- Given two discrete probability distributions $p = (p_1, \ldots, p_C)$ and $q = (q_1, \ldots, q_C)$, where $q_j > 0 \; \forall j = 1, \ldots, C$, KL-divergence between p and q is defined as:

$$
\begin{aligned}
KL(p||q) = \sum_{j=1}^{C} p_j \log \frac{p_j}{q_j} &= \sum_{j=1}^{C} p_j \log p_j - \sum_{j=1}^{C} p_j \log q_j. \\
&= NE(p) + CE(p, q).
\end{aligned}
$$

- Our aim is to minimize the error measured using KL-divergence between the true distribution p and the predicted distribution q.

- However, minimizing KL-divergence between p and q is equivalent to minimizing cross-entropy between p and q. (why?)

MLP for multi-class classification

Loss function using KL-Divergence:

- Given two discrete probability distributions $p = (p_1, \dots, p_C)$ and $q = (q_1, \dots, q_C)$, where $q_j > 0 \ \forall j = 1, \dots, C$, KL-divergence between p and q is defined as:

$$
\begin{aligned}
KL(p||q) = \sum_{j=1}^{C} p_j \log \frac{p_j}{q_j} &= \sum_{j=1}^{C} p_j \log p_j - \sum_{j=1}^{C} p_j \log q_j. \\
&= NE(p) + CE(p, q).
\end{aligned}
$$

- Our aim is to minimize the error measured using KL-divergence between the true distribution p and the predicted distribution q.
- However, minimizing KL-divergence between p and q is equivalent to minimizing cross-entropy between p and q. (why?)
- Hence we will consider the cross-entropy between p and q as our loss function.

MLP for multi-class classification

Cross-entropy loss function:

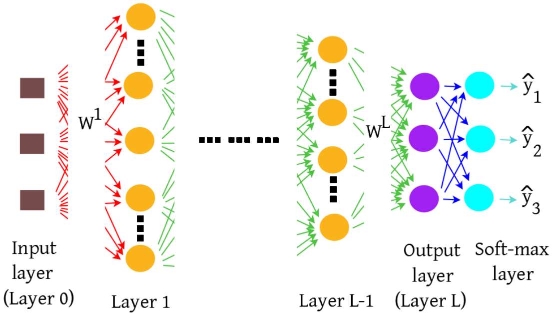
- Given two discrete probability distributions $p = (p_1, \ldots, p_C)$ and $q = (q_1, \ldots, q_C)$, where $q_j > 0 \; \forall j = 1, \ldots, C$, cross-entropy between p and q is defined as:

$$CE(p, q) = -\sum_{j=1}^{C} p_j \log q_j.$$

MLP for multi-class classification

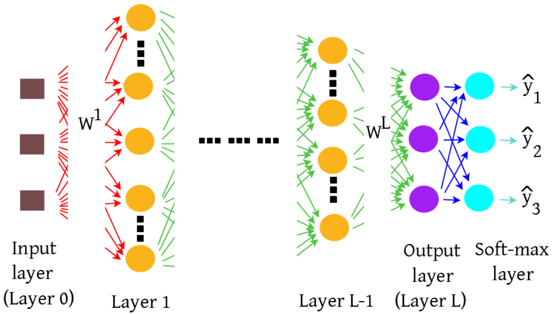
- **Input:** Training Data $D = \{(x^i, y^i)\}_{i=1}^{S}$, $x^i \in \mathcal{X} \subseteq \mathbb{R}^d$, $y^i \in \tilde{\mathcal{Y}}$, $\forall i \in \{1, \dots, S\}$ and MLP architecture parametrized by weights w.
- Without loss of generality, assume $\tilde{\mathcal{Y}} = \{0, 1\}^C$ corresponding to the output space $\mathcal{Y} = \{1, \dots, C\}, C \geq 2$ and y^i are one-hot encoded vectors.

MLP for multi-class classification



- **Input:** Training Data $D = \{(x^i, y^i)\}_{i=1}^S$, $x^i \in \mathcal{X} \subseteq \mathbb{R}^d$, $y^i \in \tilde{\mathcal{Y}}$, $\forall i \in \{1, \ldots, S\}$ and MLP architecture parametrized by weights w.
- **Aim of training MLP:** To learn a parametrized map $h_w : \mathcal{X} \to \tilde{\mathcal{Y}}$ such that for the training data D, we have $y^i = h_w(x^i)$, $\forall i \in \{1, \ldots, S\}$.
- **Aim of using the trained MLP model:** For an unseen sample $\hat{x} \in \mathcal{X}$, predict $\hat{y} = h_w(\hat{x}) = MLP(\hat{x}; w)$; to find integer label use $\text{argmax}_j \hat{y}_j$.

MLP for multi-class classification



Methodology for training MLP

- Design a suitable loss (or error) function $e : \tilde{\mathcal{Y}} \times \tilde{\mathcal{Y}} \to [0, +\infty)$ to compare the actual label y^i and the prediction $\hat{y}^i$ made by MLP using $e(y^i, \hat{y}^i)$, $\forall i\{1, \ldots, S\}$.
- **recall:** $e(y^i, \hat{y}^i) = \text{CE}(y^i, \hat{y}^i)$, the cross-entropy loss function.
- Usually the error is parametrized by the weights w of the MLP and is denoted by $e(\hat{y}^i, y^i; w)$.
- Use Gradient descent/SGD/mini-batch SGD to minimize the total error:

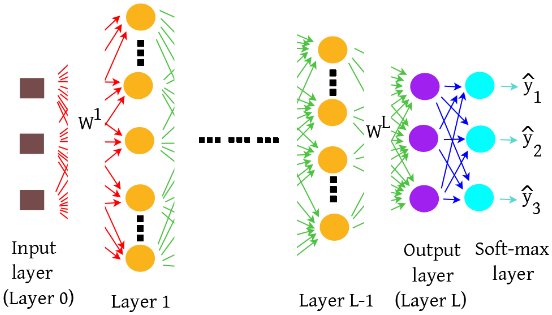
$$E = \sum_{i=1}^{S} e(\hat{y}^i, y^i; w) =: \sum_{i=1}^{S} e^i(w).$$

SGD for training MLP for multi-class classification

SGD Algorithm to train MLP

- **Input:** Training Data $D = \{(x^i, y^i)\}_{i=1}^S$, $x^i \in \mathcal{X} \subseteq \mathbb{R}^d$, $y^i \in \tilde{\mathcal{Y}}$, $\forall i$; MLP architecture, max epochs K, learning rates γ_k, $\forall k \in \{1, \ldots, K\}$.
- Start with $w^0 \in \mathbb{R}^d$.
- For $k = 0, 1, 2, \ldots, K$
  - Choose a sample $j_k \in \{1, \ldots, S\}$.
  - Find $\hat{y}^{j_k} = \text{MLP}(x^{j_k}; w^k)$. (forward pass)
  - Compute error $e^{j_k}(w^k)$ using cross-entropy loss function.
  - Compute error gradient $\nabla_w e^{j_k}(w^k)$ using backpropagation.
  - Update: $w^{k+1} \leftarrow w^k - \gamma_k \nabla_w e^{j_k}(w^k)$.
- **Output:** $w^* = w^{K+1}$.

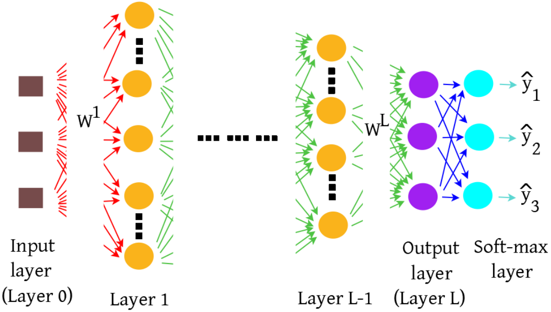
Training MLP for multi-class classification



Recall forward pass: For an arbitrary sample (x, y) from training data D, and the MLP with weights $w = (W^1, W^2 \ldots, W^L)$, the prediction $\hat{y}$ is computed using forward pass as:

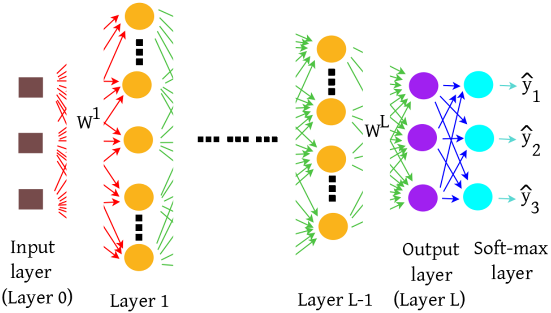
$$\hat{y} = \text{MLP}(x; w) = \text{softmax}(\phi(W^L \phi(W^{L-1} \ldots \phi(W^1 x) \ldots))).$$

Training MLP for multi-class classification



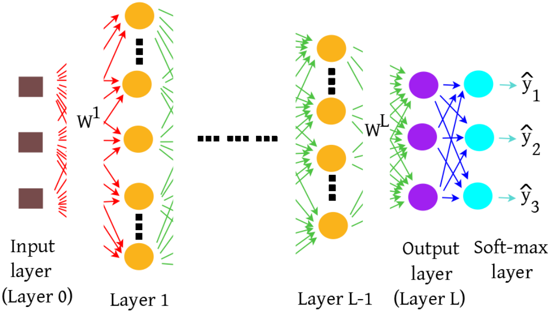
Backpropagation: For an arbitrary sample (x, y) from training data D, and the MLP with weights $w = (W^1, W^2 \ldots, W^L)$, how do we compute the error gradient?

Training MLP for multi-class classification



Backpropagation: **Indeed, the error gradients are very similar to the binary classification setup**.

Training MLP for multi-class classification



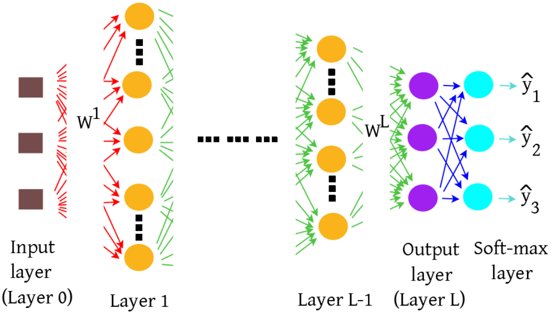
Backpropagation: **Indeed, the error gradients are very similar to the binary classification setup**.

For an arbitrary sample (x, y) from training data D, and the MLP with weights $w = (W^1, W^2 \dots, W^L)$, the error gradient with respect to weights W^ℓ in ℓ-th layer is:

$$\nabla_{W^\ell} e = \text{Diag}(\phi^{\ell\prime})\delta^\ell (a^{\ell-1})^\top$$

where $\text{Diag}(\phi^{\ell\prime}) = \begin{bmatrix} \phi'(z_1^\ell) & & \\ & \ddots & \\ & & \phi'(z_{N_\ell}^\ell) \end{bmatrix}$, $\delta^\ell = \begin{bmatrix} \frac{\partial e}{\partial a_1^\ell} \\ \vdots \\ \frac{\partial e}{\partial a_{N_\ell}^\ell} \end{bmatrix}$ and $a^{\ell-1} = \begin{bmatrix} a_1^{\ell-1} \\ \vdots \\ a_{N_{\ell-1}}^{\ell-1} \end{bmatrix}$.

Training MLP for multi-class classification



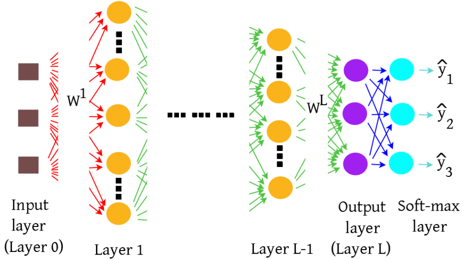
Backpropagation: **Indeed, the error gradients are very similar to the binary classification setup**.

For an arbitrary sample (x, y) from training data D, and the MLP with weights $w = (W^1, W^2 \ldots, W^L)$, the error gradient with respect to weights W^ℓ in ℓ-th layer is:

$$\nabla_{W^\ell} e = \text{Diag}(\phi^{\ell'})\delta^\ell(a^{\ell-1})^\top = \text{Diag}(\phi^{\ell'})V^{\ell+1}V^{\ell+2}\ldots V^L\delta^L(a^{\ell-1})^\top$$

where $V^\ell = (W^{\ell+1})^\top \text{Diag}(\phi^{\ell+1'})$.

Training MLP for multi-class classification



Backpropagation: **Indeed, the error gradients are very similar to the binary classification setup**.

For an arbitrary sample (x, y) from training data D, and the MLP with weights $w = (W^1, W^2 \dots, W^L)$, the error gradient with respect to weights W^ℓ in ℓ-th layer is:

$$\nabla_{W^\ell} e = \mathrm{Diag}(\phi^{\ell\prime})\delta^\ell(a^{\ell-1})^\top = \mathrm{Diag}(\phi^{\ell\prime})V^{\ell+1}V^{\ell+2}\dots V^L\delta^L(a^{\ell-1})^\top$$

The main difference arises in the procedure to find $\delta^L = \begin{bmatrix} \frac{\partial e}{\partial a_1^L} \\ \vdots \\ \frac{\partial e}{\partial a_C^L} \end{bmatrix}$.

Training MLP for multi-class classification: Finding error gradients at the output layer

Recall: For an arbitrary sample (x, y) from training data D, and the prediction $\hat{y}$ from the MLP, we know the following:

- y is in one-hot encoded form: $y = \begin{bmatrix} y_1 \\ \vdots \\ y_C \end{bmatrix}$ with some particular y_c taking value 1 (corresponding to the label c) and $y_j = 0, j \neq c$.

Training MLP for multi-class classification: Finding error gradients at the output layer

Recall: For an arbitrary sample (x, y) from training data D, and the prediction $\hat{y}$ from the MLP, we know the following:

- y is in one-hot encoded form: $y = \begin{bmatrix} y_1 \\ \vdots \\ y_C \end{bmatrix}$ with some particular y_c taking value 1 (corresponding to the label c) and $y_j = 0, j \neq c$.
- y and $\hat{y}$ are discrete probability distributions.

Training MLP for multi-class classification: Finding error gradients at the output layer

Recall: For an arbitrary sample (x, y) from training data D, and the prediction $\hat{y}$ from the MLP, we know the following:

- y is in one-hot encoded form: $y = \begin{bmatrix} y_1 \\ \vdots \\ y_C \end{bmatrix}$ with some particular y_c taking value 1 (corresponding to the label c) and $y_j = 0, j \neq c$.
- y and $\hat{y}$ are discrete probability distributions.
- $\hat{y} = \begin{bmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_C \end{bmatrix}$.

Training MLP for multi-class classification: Finding error gradients at the output layer

Recall: For an arbitrary sample (x, y) from training data D, and the prediction $\hat{y}$ from the MLP, we know the following:

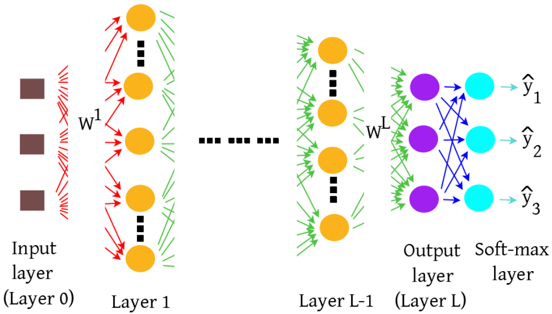
- y is in one-hot encoded form: $y = \begin{bmatrix} y_1 \\ \vdots \\ y_C \end{bmatrix}$ with some particular y_c taking value 1 (corresponding to the label c) and $y_j = 0, j \neq c$.
- y and $\hat{y}$ are discrete probability distributions.
- $\hat{y} = \begin{bmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_C \end{bmatrix}$.
- $e(\hat{y}, y)$ is the cross-entropy error function: $e(\hat{y}, y) = \sum_{j=1}^{C} e^j = -\sum_{j=1}^{C} y_j \log \hat{y}_j$.

Training MLP for multi-class classification: Finding error gradients at the output layer

Recall: For an arbitrary sample (x, y) from training data D, and the prediction $\hat{y}$ from the MLP, we know the following:

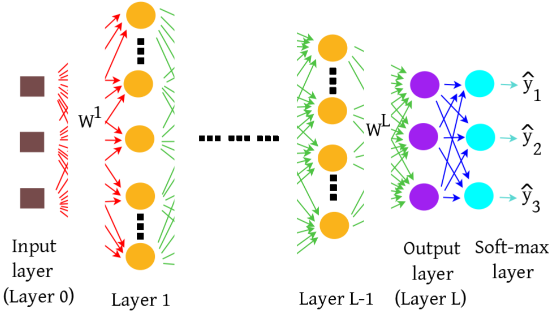
- y is in one-hot encoded form: $y = \begin{bmatrix} y_1 \\ \vdots \\ y_C \end{bmatrix}$ with some particular y_c taking value 1 (corresponding to the label c) and $y_j = 0, j \neq c$.
- y and $\hat{y}$ are discrete probability distributions.
- $\hat{y} = \begin{bmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_C \end{bmatrix}$.
- $e(\hat{y}, y)$ is the cross-entropy error function: $e(\hat{y}, y) = \sum_{j=1}^{C} e^j = -\sum_{j=1}^{C} y_j \log \hat{y}_j$.
- **Note:** $\hat{y}_j = \frac{\exp(a_j^L)}{\sum_{r=1}^{C} \exp(a_r^L)}, \forall j = 1, \ldots, C$.

Training MLP for multi-class classification: Finding error gradients at the output layer



- $e(\hat{y}, y)$ is the cross-entropy error function: $e(\hat{y}, y) = \sum_{j=1}^{C} e^j = -\sum_{j=1}^{C} y_j \log \hat{y}_j$, with $\hat{y}_j = \frac{\exp(a_j^L)}{\sum_{r=1}^{C} \exp(a_r^L)}$, $\forall j = 1, \ldots, C$.

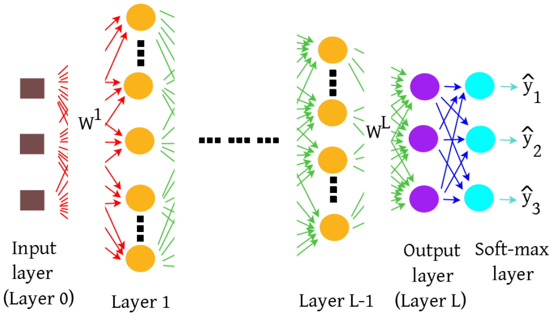
Training MLP for multi-class classification: Finding error gradients at the output layer



- $e(\hat{y}, y)$ is the cross-entropy error function: $e(\hat{y}, y) = \sum_{j=1}^{C} e^j = -\sum_{j=1}^{C} y_j \log \hat{y}_j$, with $\hat{y}_j = \frac{\exp(a_j^L)}{\sum_{r=1}^{C} \exp(a_r^L)}$, $\forall j = 1, \ldots, C$.

$$\text{Aim: To find } \delta^L = \begin{bmatrix} \frac{\partial e}{\partial a_1^L} & \cdots & \frac{\partial e}{\partial a_C^L} \end{bmatrix}^\top.$$

Training MLP for multi-class classification: Finding error gradients at the output layer



- $e(\hat{y}, y)$ is the cross-entropy error function: $e(\hat{y}, y) = \sum_{j=1}^{C} e^j = -\sum_{j=1}^{C} y_j \log \hat{y}_j$, with $\hat{y}_j = \frac{\exp(a_j^L)}{\sum_{r=1}^{C} \exp(a_r^L)}, \forall j = 1, \ldots, C.$

$$\text{Aim: To find } \delta^L = \begin{bmatrix} \frac{\partial e}{\partial a_1^L} & \cdots & \frac{\partial e}{\partial a_C^L} \end{bmatrix}^\top.$$

- We have $\frac{\partial e}{\partial a_j^L} = \sum_{m=1}^{C} \frac{\partial e^m}{\partial \hat{y}_m} \frac{\partial \hat{y}_m}{\partial a_j^L}, \ \forall j.$

Training MLP for multi-class classification: Finding error gradients at the output layer

- $e(\hat{y}, y)$ is the cross-entropy error function: $e(\hat{y}, y) = \sum_{j=1}^{C} e^j = -\sum_{j=1}^{C} y_j \log \hat{y}_j$, with $\hat{y}_j = \frac{\exp(a_j^L)}{\sum_{r=1}^{C} \exp(a_r^L)}, \forall j$.
- We have $\frac{\partial e}{\partial a_j^L} = \sum_{m=1}^{C} \frac{\partial e^m}{\partial \hat{y}_m} \frac{\partial \hat{y}_m}{\partial a_j^L}$.

Training MLP for multi-class classification: Finding error gradients at the output layer

- $e(\hat{y}, y)$ is the cross-entropy error function: $e(\hat{y}, y) = \sum_{j=1}^{C} e^j = -\sum_{j=1}^{C} y_j \log \hat{y}_j$, with $\hat{y}_j = \frac{\exp(a_j^L)}{\sum_{r=1}^{C} \exp(a_r^L)}, \forall j$.
- We have $\frac{\partial e}{\partial a_j^L} = \sum_{m=1}^{C} \frac{\partial e^m}{\partial \hat{y}_m} \frac{\partial \hat{y}_m}{\partial a_j^L}$.
- $\frac{\partial e^m}{\partial \hat{y}_m} = -\frac{y_m}{\hat{y}_m}$.
- $\frac{\partial \hat{y}_m}{\partial a_j^L} = \frac{\partial}{\partial a_j^L} \left(\frac{\exp(a_m^L)}{\sum_{r=1}^{C} \exp(a_r^L)} \right)$.

Training MLP for multi-class classification: Finding error gradients at the output layer

- $e(\hat{y}, y)$ is the cross-entropy error function: $e(\hat{y}, y) = \sum_{j=1}^{C} e^j = -\sum_{j=1}^{C} y_j \log \hat{y}_j$, with $\hat{y}_j = \frac{\exp(a_j^L)}{\sum_{r=1}^{C} \exp(a_r^L)}, \forall j$.
- We have $\frac{\partial e}{\partial a_j^L} = \sum_{m=1}^{C} \frac{\partial e^m}{\partial \hat{y}_m} \frac{\partial \hat{y}_m}{\partial a_j^L}$.
- $\frac{\partial e^m}{\partial \hat{y}_m} = -\frac{y_m}{\hat{y}_m}$.
- $\frac{\partial \hat{y}_m}{\partial a_j^L} = \frac{\partial}{\partial a_j^L} \left(\frac{\exp(a_m^L)}{\sum_{r=1}^{C} \exp(a_r^L)} \right) = \begin{cases} \hat{y}_j(1-\hat{y}_j) & \text{if } j = m \\ -\hat{y}_m \hat{y}_j & \text{otherwise.} \end{cases}$ (Homework!)

Training MLP for multi-class classification: Finding error gradients at the output layer

- $e(\hat{y}, y)$ is the cross-entropy error function: $e(\hat{y}, y) = \sum_{j=1}^{C} e^j = -\sum_{j=1}^{C} y_j \log \hat{y}_j$, with $\hat{y}_j = \frac{\exp(a_j^L)}{\sum_{r=1}^{C} \exp(a_r^L)}, \forall j$.
- We have $\frac{\partial e}{\partial a_j^L} = \sum_{m=1}^{C} \frac{\partial e^m}{\partial \hat{y}_m} \frac{\partial \hat{y}_m}{\partial a_j^L}$.
- $\frac{\partial e^m}{\partial \hat{y}_m} = -\frac{y_m}{\hat{y}_m}$.
- $\frac{\partial \hat{y}_m}{\partial a_j^L} = \frac{\partial}{\partial a_j^L}\left(\frac{\exp(a_m^L)}{\sum_{r=1}^{C} \exp(a_r^L)}\right) = \begin{cases} \hat{y}_j(1-\hat{y}_j) & \text{if } j = m \\ -\hat{y}_m \hat{y}_j & \text{otherwise.} \end{cases}$ (Homework!)
- Hence by substitution, $\frac{\partial e}{\partial a_j^L} = \sum_{\substack{m=1 \\ m \neq j}}^{C} y_m \hat{y}_j - y_j(1-\hat{y}_j) = \hat{y}_j - y_j$.

Training MLP for multi-class classification: Finding error gradients at the output layer

- **Recall:** We wanted to find $\delta^L = \begin{bmatrix} \frac{\partial e}{\partial a_1^L} \\ \vdots \\ \frac{\partial e}{\partial a_C^L} \end{bmatrix}$.
- We have $\frac{\partial e^m}{\partial a_j^L} = \hat{y}_j - y_j$.
- Hence $\delta^L = \begin{bmatrix} \hat{y}_1 - y_1 \\ \vdots \\ \hat{y}_C - y_C \end{bmatrix}$.

Training MLP for multi-class classification

Since the backpropagation for multi-class classification is essentially similar to that of binary classification, it suffers from

- Exploding gradient problem and
- Vanishing gradient problem.

MLP for multi-label classification

- **Input:** Training Data $D = \{(x^i, y^i)\}_{i=1}^{S}$, $x^i \in \mathcal{X} \subseteq \mathbb{R}^d$, $\forall i \in \{1, \ldots, S\}$ and MLP architecture parametrized by weights w.
- **New Task:** $y^i \subseteq \mathcal{Y} = \{1, \ldots, C\}, C \geq 2$.
- Corresponds to multi-label classification.

Question 1: What is a suitable architecture for the MLP's last (or output) layer?

Question 2: What is a suitable loss (or error) function?

MLP for multi-label classification

- **Input:** Training Data $D = \{(x^i, y^i)\}_{i=1}^{S}$, $x^i \in \mathcal{X} \subseteq \mathbb{R}^d$, $\forall i \in \{1, \ldots, S\}$ and MLP architecture parametrized by weights w.
- **New Task:** $y^i \subseteq \mathcal{Y} = \{1, \ldots, C\}, C \geq 2$.
- Corresponds to multi-label classification.

Question 1: What is a suitable architecture for the MLP's last (or output) layer?

Question 2: What is a suitable loss (or error) function? (Homework!)