Foundations of Deep Learning, Winter Term 2021/22

Week 2: From Logistic Regression to MLPs

From Logistic Regression to MLPs

Frank Hutter Abhinav Valada

University of Freiburg

Overview of Week 2

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

Week 2: From Logistic Regression to MLPs

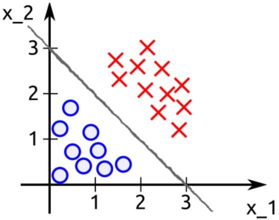
Recap of Logistic Regression

Frank Hutter   Abhinav Valada

University of Freiburg

Logistic Regression: Decision Boundary

- Logistic regression is a classification method
- The decision boundary is a linear function



$$w_0 + w_1 x_1 + w_2 x_2 = 0$$

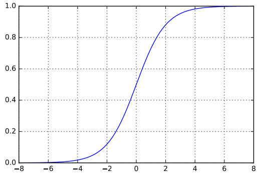
$$\mathbf{w} = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$$

Predict "$y = 1$" if $\mathbf{w}^\intercal \mathbf{x} \geq 0$ (with $x_0 = 1$ for the bias)

- Remark: with non-linear basis functions the decision boundary can also be non-linear

Logistic Regression: Probabilistic Prediction

- Logistic regression yields a probabilistic estimate
  - How likely is it that data point $\mathbf{x}$ belongs to class 1?
- Logistic regression computes this probability as $h_{\mathbf{w}}(\mathbf{x}) = g(\mathbf{w}^\top \mathbf{x})$
  - Here, g is the logistic function $g(z) = \frac{1}{1+e^{-z}}$



- We're maximally uncertain about points on the decision boundary

$$\text{E.g., } \mathbf{w}^\top \mathbf{x} = 0 \quad \Leftrightarrow \quad h_{\mathbf{w}}(\mathbf{x}) = 0.5$$
$$\text{E.g., } \mathbf{w}^\top \mathbf{x} = 5 \quad \Leftrightarrow \quad h_{\mathbf{w}}(\mathbf{x}) = 0.993$$

Logistic Regression: Derivation of Cross-Entropy Loss

- The true value of y is unknown; we model it as a random variable Y
  - Y has a Bernoulli distribution
- Logistic regression predicts the value of Y: $h_{\mathbf{w}}(\mathbf{x}) = p_{\text{model}}(Y = 1 \mid \mathbf{x}; \mathbf{w})$
  - Model's estimated probability that $y = 1$,
    given that the input is $\mathbf{x}$ and the model is parameterized by $\mathbf{w}$
- The actual true labels are still discrete ($y = 0$ or $y = 1$)
  - The estimated probabilities need to add to one, so:
    $p_{\text{model}}(Y = 0 \mid \mathbf{x}; \mathbf{w}) = 1 - p_{\text{model}}(Y = 1 \mid \mathbf{x}; \mathbf{w}) = 1 - h_{\mathbf{w}}(\mathbf{x})$
- Likelihood of the true data under the model:

$$p_{\text{model}}(Y = y \mid \mathbf{x}; \mathbf{w}) = \begin{cases} h_{\mathbf{w}}(\mathbf{x}) & \text{for} \quad y = 1 \\ 1 - h_{\mathbf{w}}(\mathbf{x}) & \text{for} \quad y = 0 \end{cases}$$

- Cross-entropy loss: the negative log[1] likelihood of the true data under the model:

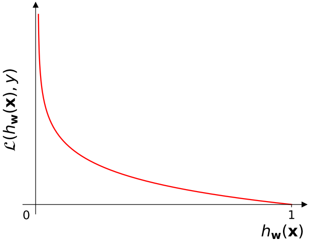
$$\mathcal{L}(h_{\mathbf{w}}(\mathbf{x}), y) = \begin{cases} -\log(h_{\mathbf{w}}(\mathbf{x})) & \text{for} \quad y = 1 \\ -\log(1 - h_{\mathbf{w}}(\mathbf{x})) & \text{for} \quad y = 0 \end{cases}$$

[1] Unless mentioned otherwise, the natural logarithm is used in all formulas we will see in this lecture.

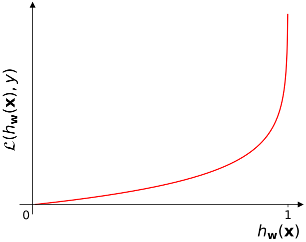
Visualization of the Cross-Entropy Loss Function

$$\mathcal{L}(h_{\mathbf{w}}(\mathbf{x}), y) = \begin{cases} -\log(h_{\mathbf{w}}(\mathbf{x})) & \text{for} \quad y = 1 \\ -\log(1 - h_{\mathbf{w}}(\mathbf{x})) & \text{for} \quad y = 0 \end{cases}$$

Case: $y = 1$



Case: $y = 0$



Different Way of Writing the Cross-Entropy Loss function

- Previously, we wrote the cross-entropy loss function for a single data point as:

$$\mathcal{L}(h_{\mathbf{w}}(\mathbf{x}), y) = \begin{cases} -\log(h_{\mathbf{w}}(\mathbf{x})) & \text{for} \quad y = 1 \\ -\log(1 - h_{\mathbf{w}}(\mathbf{x})) & \text{for} \quad y = 0 \end{cases}$$

- We can exploit that y is 0 or 1 to rewrite this in a single line:

$$\mathcal{L}(h_{\mathbf{w}}(\mathbf{x}), y) = -y \log(h_{\mathbf{w}}(\mathbf{x})) - (1 - y) \log(1 - h_{\mathbf{w}}(\mathbf{x})).$$

- Using shorthand $\hat{y} = h_{\mathbf{w}}(\mathbf{x})$ yields:

$$\mathcal{L}(\hat{y}, y) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

- It doesn't look like that anymore, but this is still the negative log likelihood of the true label under the model.

The Loss Function for the Entire Data Set

$$\mathcal{L}(\hat{y}_n, y_n) = -y_n \log \hat{y}_n - (1 - y_n) \log(1 - \hat{y}_n)$$

$\rightsquigarrow$ Loss for a single data point $\langle \mathbf{x}_n, y_n \rangle$

- For the entire training data set $\mathcal{D}_{\text{train}} = \{\langle \mathbf{x}_1, y_1 \rangle, \ldots, \langle \mathbf{x}_N, y_N \rangle\}$, the loss is:

$$L(\mathbf{w}) = -\frac{1}{N} \sum_{n=1}^{N} y_n \log \hat{y}_n + (1 - y_n) \log(1 - \hat{y}_n)$$

- This is a convex function of $\mathbf{w}$
- We minimize it via an iterative solver (SGD)

Question to Answer for Yourself / Discuss with Friends

- Repeating a derivation: Derive the cross-entropy loss function for logistic regression:

$$\mathcal{L}(\hat{y}, y) = -y \log \hat{y} - (1-y) \log(1-\hat{y})$$

- Application of what you just learned: What is the computational complexity (in big-O notation) of computing the cross-entropy loss

$$L(\mathbf{w}) = -\frac{1}{N} \sum_{n=1}^{N} y_n \log \hat{y}_n + (1-y_n) \log(1-\hat{y}_n)$$

for logistic regression on a data set of N data points with d dimensions?

- Application of what you just learned: Which of these two predictions does cross-entropy loss prefer for a dataset $\{\langle \mathbf{x}_1, 0\rangle, \langle \mathbf{x}_2, 1\rangle\}$?
  1. $\hat{y}(\mathbf{x}_1) = 1$, $\hat{y}(\mathbf{x}_2) = 1$, or
  2. $\hat{y}(\mathbf{x}_1) = 0.5$, $\hat{y}(\mathbf{x}_2) = 0.5$?

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

Week 2: From Logistic Regression to MLPs

Cross Entropy, KL Divergence, and Maximum Likelihood

Frank Hutter Abhinav Valada

University of Freiburg

Entropy and Cross-Entropy

- **Information Entropy:** $H(P) = -\mathbb{E}_{\mathrm{x}\sim P}\left[\log P(\mathrm{x})\right]$
  - quantifies uncertainty about random variable X
  - for random variable with K possible outcomes:

$$H(P) = -\sum_{k=1}^{K} P(X = x_k) \log P(X = x_k)$$

  - average number of bits[2] needed to code an event drawn from $P(\mathrm{x})$

- **Cross-Entropy:** $H(P, Q) = -\mathbb{E}_{\mathrm{x}\sim P}\left[\log Q(\mathrm{x})\right]$
  - for random variable with K possible outcomes:

$$H(P, Q) = -\sum_{k=1}^{K} P(X = x_k) \log Q(X = x_k)$$

  - average number of bits[1] needed to code an event drawn from $P(\mathrm{x})$ using a code that is optimized for the "wrong" distribution $Q(x)$

[2] bits when using $\log_2$; nats when using $\ln$

Kullback-Leibler (KL) Divergence

- Kullback-Leibler (KL) Divergence:

$$D_{KL}(P||Q) = -\mathbb{E}_{\mathrm{x}\sim P}\left[\log\left(\frac{Q(x)}{P(x)}\right)\right]$$
$$= H(P, Q) - H(P)$$

  - widely used way to assess similarity of two distributions P and Q
  - cross entropy minus entropy of P
  - zero if $P = Q$; but not symmetric

Maximum Likelihood (ML) Estimation

- We would like to estimate a model parameter $\boldsymbol{\theta}$
- Intuitive goal: maximize the probability of some data under the model
- Consider a set of m examples $\mathbb{X} = \{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ drawn from the (true but unknown) data-generating distribution $p_{\text{data}}(\boldsymbol{x})$
- Let $p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta})$ be a parametric family of distributions, indexed by $\boldsymbol{\theta}$
  - We want to set $\boldsymbol{\theta}$ to make this as similar to $p_{\text{data}}(\boldsymbol{x})$ as possible
- The maximum likelihood estimator $\boldsymbol{\theta}_{ML}$ for $\boldsymbol{\theta}$ is then defined as:

$$\boldsymbol{\theta}_{ML} = \text{argmax}_{\boldsymbol{\theta}} p_{\text{model}}(\mathbb{X}; \boldsymbol{\theta}) \tag{5.56}$$

Equation numbers from the very nice book "Deep Learning"
by Ian Goodfellow, Yoshua Bengio and Aaron Courville: `http://www.deeplearningbook.org/`.

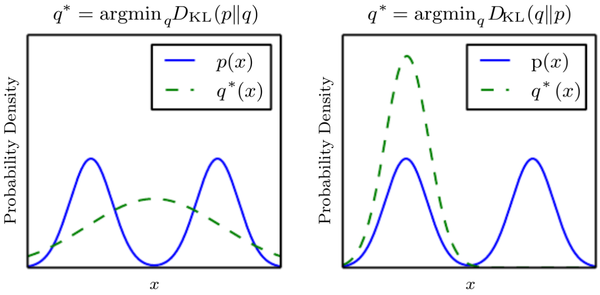
Maximum Likelihood (ML) Estimation

- This maximum likelihood estimator $\boldsymbol{\theta}_{ML}$ for $\boldsymbol{\theta}$ also minimizes the cross entropy and the KL divergence between p_{model} and $\hat{p}_{\text{data}}$:

$$\begin{aligned}
\boldsymbol{\theta}_{ML} &= \text{argmax}_{\boldsymbol{\theta}} p_{\text{model}}(\mathbb{X}; \boldsymbol{\theta}) && (5.56)\\
&= \text{argmax}_{\boldsymbol{\theta}} \prod_{i=1}^{m} p_{\text{model}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) && (5.57)\\
&= \text{argmax}_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{\text{model}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) && (5.58)\\
&= \text{argmax}_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x} \sim \hat{p}_{\text{data}}} \log p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta}) && (5.59)\\
&= \text{argmin}_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x} \sim \hat{p}_{\text{data}}} [-\log p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta})] && (5.61)\\
&= \text{argmin}_{\boldsymbol{\theta}} H(\hat{p}_{\text{data}}, p_{\text{model}}(\cdot; \boldsymbol{\theta})) && \text{(cross entropy)}\\
&= \text{argmin}_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x} \sim \hat{p}_{\text{data}}} [\log \hat{p}_{\text{data}}(\boldsymbol{x}) - \log p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta})] && (5.60)\\
&= \text{argmin}_{\boldsymbol{\theta}} \; D_{KL}(\hat{p}_{\text{data}}(\cdot) || p_{\text{model}}(\cdot; \boldsymbol{\theta})) && \text{(KL divergence)}
\end{aligned}$$

Maximum Likelihood (ML) Estimation

- Interpreting the ML estimator $\boldsymbol{\theta}_{ML} \in \text{argmin}_{\boldsymbol{\theta}}\, D_{KL}(\hat{p}_{\text{data}}(\cdot)||p_{\text{model}}(\cdot;\boldsymbol{\theta}))$



[Goodfellow et al., 2016, p. 74, fig 3.6]

  - The model parameter $\boldsymbol{\theta}_{ML}$ that makes the observed data most likely under the model $q^* = p_{\text{model}}(\cdot;\boldsymbol{\theta}_{ML}))$. The left case in the figure.
  - We do *not* aim for samples from q^* to be likely under p (the right case)

Question to Answer for Yourself / Discuss with Friends

- Transfer: Why would it be interesting to track the KL divergence between p_{model} and p_{data}, rather than the typically-used cross-entropy loss?

  (Hint: think about how close to optimal the prediction already is.)

- Transfer: Show that the formula

$$L(\mathbf{w}) = -\frac{1}{N}\sum_{n=1}^{N}\{y_n \log \hat{y}_n + (1-y_n)\log(1-\hat{y}_n)\}$$

  we derived for the negative log likelihood of the Bernoulli predictions of logistic regression is just a special case of the general form

$$H(P,Q) = -\mathbb{E}_{\mathrm{x}\sim P}\left[\log Q(\mathrm{x})\right]$$

  of cross-entropy.

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

Week 2: From Logistic Regression to MLPs

Logistic Regression as a Neural Network: The Perceptron

Frank Hutter   Abhinav Valada

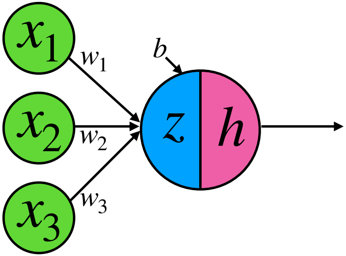
University of Freiburg

From Logistic Regression to the Perceptron

$$h_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + exp(-(\mathbf{w}^\top \mathbf{x} + w_0))} = \frac{1}{1 + exp(-z)}$$

Input $\mathbf{x}$ → **Linear Transformation** $z = \mathbf{w}^\top \mathbf{x} + w_0$ → **Logistic Activation** $h(z) = \dfrac{1}{1 + exp(-z)}$

Logistic Regression



$$z = \mathbf{w}^\top \mathbf{x} + b$$

$$h = \frac{1}{1 + exp(-z)}$$

Perceptron

Question to Answer for Yourself / Discuss with Friends

- Repetition:
  Write down the forward pass of a perceptron as a succession of two formulas.
- Repetition of previous material / transfer:
  What is the computational complexity (in big-O notation) of computing the cross-entropy loss for a perceptron on a data set of N data points with d dimensions?

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

Week 2: From Logistic Regression to MLPs

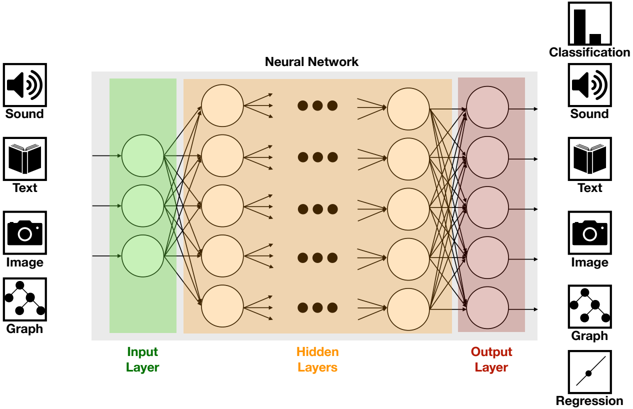
Multilayer Perceptrons

Frank Hutter   Abhinav Valada

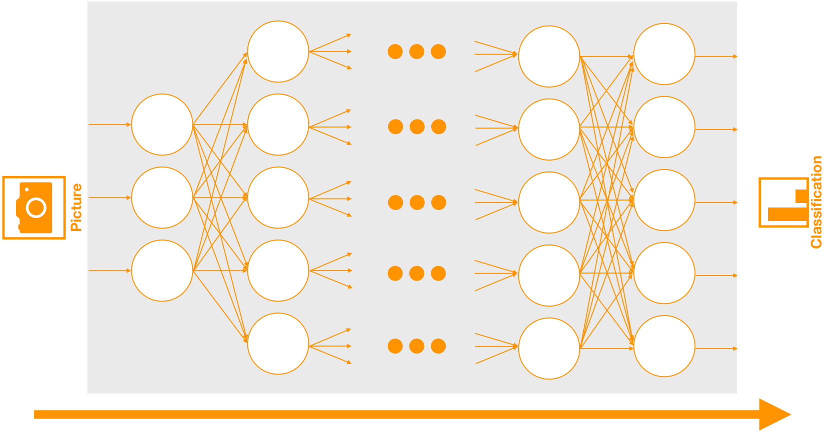
University of Freiburg

Multilayer Perceptrons (MLPs)

- We now add hidden layers
  - These layers learn nonlinear features for the final logistic regression
- Successive layers are fully-connected



Computation is Performed Layer-by-Layer



Computation in the First Hidden Layer

- For input vector $\mathbf{x}$, compute pre-activations $\mathbf{z}^{(1)}$ in layer 1 as

$$\mathbf{z}^{(1)} = \mathbf{W}^{(1)^\intercal}\mathbf{x} + \mathbf{b}^{(1)}$$

- Pre-activations are transformed through a differentiable, nonlinear activation function $g^{(1)}(\cdot)$, resulting in activation vector $\mathbf{h}^{(1)}$ of the first hidden layer:

$$\mathbf{h}^{(1)} = g^{(1)}(\mathbf{z}^{(1)})$$

- The units in this layer implement the adaptable basis functions.

Computation in the Second Hidden Layer etc.

- Outputs $\mathbf{h}^{(1)}$ from layer 1 are combined linearly in the next layer 2:

$$\mathbf{z}^{(2)} = \mathbf{W}^{(2)^{\mathsf{T}}} \mathbf{h}^{(1)} + \mathbf{b}^{(2)}$$

- Pre-activations $\mathbf{z}^{(2)}$ are again transformed through a nonlinear activation function $g^{(2)}$ to compute the activations $\mathbf{h}^{(2)}$:

$$\mathbf{h}^{(2)} = g^{(2)}(\mathbf{z}^{(2)})$$

- This repeats from each layer k to $k+1$, all the way to output layer K
  - The network then outputs the output layer's activations: $\hat{\mathbf{y}} := \mathbf{h}^{(K)}$.

- E.g., for a network with one hidden layer, the overall network output $\hat{\mathbf{y}}$ for input $\mathbf{x}$ is:

$$\hat{\mathbf{y}} = g^{(2)}(\mathbf{W}^{(2)^{\mathsf{T}}} g^{(1)}(\mathbf{W}^{(1)^{\mathsf{T}}} \mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)})$$

Question to Answer for Yourself / Discuss with Friends

- Application of what you just learned:
  What is the *time* complexity (in big-O notation) of a forward pass of a single data point of input dimensionality d in an MLP with two hidden layers (of size k_1 and k_2, respectively)?
  Hint: The *time* complexity of multiplying 2 matrices with dimensions $n \times m$ and $m \times k$ is $O(nmk)$.

- Application of what you just learned:
  What is the *memory* complexity (in big-O notation) of a forward pass of a single data point of input dimensionality d in an MLP with two hidden layers (of size k_1 and k_2, respectively)?
  Hint: The *memory* complexity of multiplying 2 matrices with dimensions $n \times m$ and $m \times k$ is $O(nm + mk)$.

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

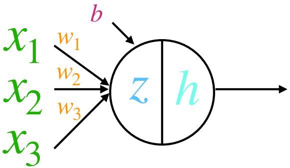
Week 2: From Logistic Regression to MLPs

Matrix Dimensions

Frank Hutter Abhinav Valada

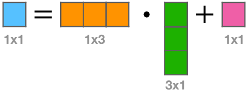
University of Freiburg

One Neuron, One Input Vector

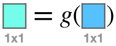


$$z = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

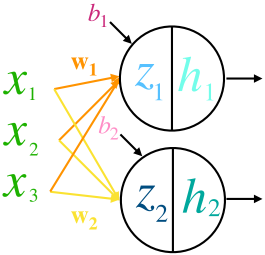
$$z = \mathbf{w}^\top \mathbf{x} + b$$



$$h = g(z)$$



Two Neurons, One Input Vector



$$z_1 = \mathbf{w_1}^\top \mathbf{x} + b_1$$
$$h_1 = g(z_1)$$
$$z_2 = \mathbf{w_2}^\top \mathbf{x} + b_2$$
$$h_2 = g(z_2)$$

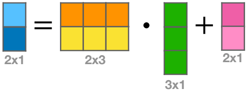
$\Rrightarrow$

Put together into one:

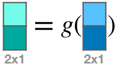
$$\mathbf{z} = \mathbf{W}^\top \mathbf{x} + \mathbf{b}$$
$$\mathbf{h} = g(\mathbf{z})$$

Visual illustration:

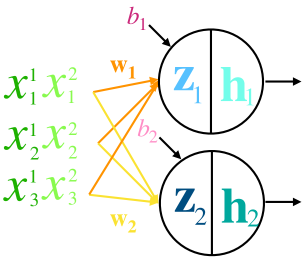
$$\mathbf{z} = \mathbf{W}^\top \mathbf{x} + \mathbf{b}$$



$$\mathbf{h} = g(\mathbf{z})$$



Two Neurons, Batch of Two Input Vectors



$$\mathbf{z_1} = \mathbf{w_1}^\top \mathbf{X} + b_1$$
$$\mathbf{h_1} = g(z_1)$$
$$\mathbf{z_2} = \mathbf{w_2}^\top \mathbf{X} + b_2$$
$$\mathbf{h_2} = g(z_2)$$

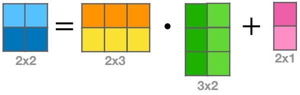
$\Rrightarrow$

Put together into one:

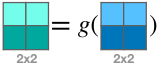
$$\mathbf{Z} = \mathbf{W}^\top \mathbf{X} + \mathbf{b}$$
$$\mathbf{h} = \mathbf{g}(\mathbf{Z})$$

Visual illustration:

$$\mathbf{Z} = \mathbf{W}^\top \mathbf{X} + \mathbf{b}$$

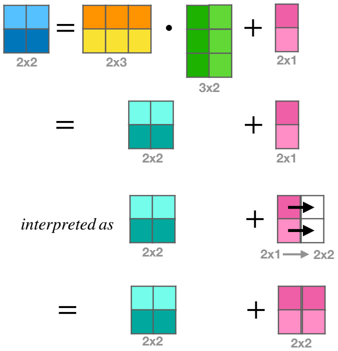


$$\mathbf{H} = \mathbf{g}(\mathbf{Z})$$



Two Neurons, Batch of Two Input Vectors

$$\mathbf{Z} = \mathbf{W}^{\top}\mathbf{X} + \mathbf{b}$$



Warning: Different Common Notations in Math and in Code

- Python frameworks for Deep Learning (like PyTorch) use a different notation
  - In the slides, we follow the standard notation (e.g., in DL book) of $\mathbf{x}$ being a column vector
  - In PyTorch, data points $\mathbf{x}$ are row vectors
  - We will use the Pytorch notation for our coding exercises

- Summary of PyTorch notation
  - The inputs $\mathbf{X} \in \mathbb{R}^{N \times D}$ have N datapoints in the rows and D features in the columns
  - A single linear layer has weight $\mathbf{W} \in \mathbb{R}^{D \times M}$ and bias $\mathbf{b} \in \mathbb{R}^{M}$
  - The bias is expanded to $\mathbf{B} \in \mathbb{R}^{N \times M}$ by repeating it for each datapoint.

  - The formula for output $\mathbf{Z} \in \mathbb{R}^{N \times M}$ is then:

$$\mathbf{Z} = \mathbf{X}\mathbf{W} + \mathbf{B}$$

Questions to Answer for Yourself / Discuss with Friends

- Repetition:
  Write down the forward pass of a perceptron as a succession of two formulas, for a batch of B data points with d dimensions; for each term in the formulas, include the vector/matrix dimensions.

- Application of what you just learned:
  What is the time complexity (in big-O notation) of a forward pass in an MLP with M layers of k units each, depending on the batch size B and input dimensionality d?

- Application of what you just learned:
  What is the memory complexity (in big-O notation) of a forward pass in an MLP with M layers of k units each, depending on the batch size B and input dimensionality d?

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

Week 2: From Logistic Regression to MLPs

Other Activation Functions and Loss Functions

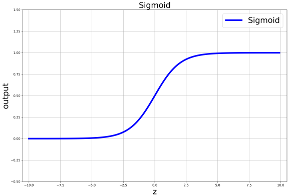
Frank Hutter Abhinav Valada

University of Freiburg

Activation Functions - Examples

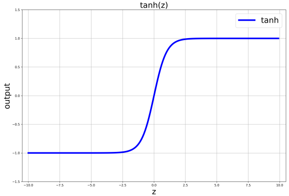
Logistic sigmoid activation function:

$$g_{logistic}(z) = \frac{1}{1 + \exp(-z)}$$



Logistic hyperbolic tangent activation function:

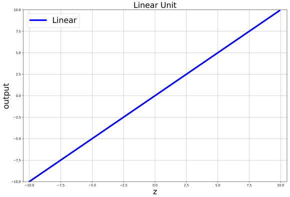
$$\begin{aligned} g_{tanh}(z) &= \tanh(z) \\ &= \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)} \end{aligned}$$



Activation Functions - Examples (cont.)

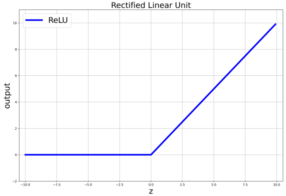
Linear activation function:

$$g_{linear}(z) = z$$



Rectified Linear (ReLU) activation function:

$$g_{relu}(z) = \max(0, z)$$

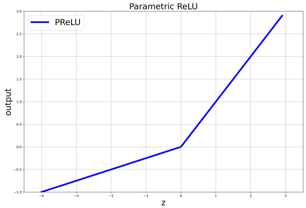


Activation Functions - Examples (cont.)

Parametric ReLU (PReLU) activation function
[He et al., 2015]:

$$PReLU(z) = \begin{Bmatrix} z, & z > 0 \\ az, & z \leq 0 \end{Bmatrix},$$

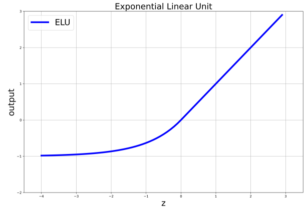
where $a > 0$ is a learnable parameter controlling the slope of the negative part



Exponential Linear Unit (ELU) activation function
[Clevert et al., 2015]:

$$ELU(z) = \begin{Bmatrix} z, & z > 0 \\ \alpha(\exp(z) - 1), & z \leq 0 \end{Bmatrix},$$

where $\alpha > 0$ controls the saturation for negative z

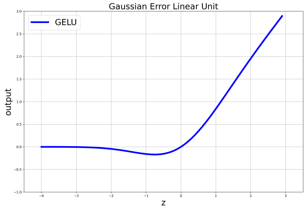


Activation Functions - Examples (cont.)

Gaussian Error Linear Unit (GELU) activation function [Hendrycks, Gimpel, 2016]:

$$GELU(z) = z\Phi(z),$$

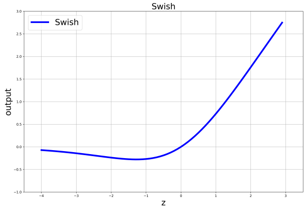
where Φ is the Cumulative Distribution Function



Swish activation function [Ramachandran et al., 2017]:

$$Swish(z) = z\sigma(\beta z),$$

where $\sigma(z)$ is the sigmoid function, and $\beta \geq 0$ is a constant or trainable parameter



Output Unit Activation Functions

Depending on the task, typically:

- for regression: output neurons with linear activation
- for binary classification: output neurons with logistic/tanh activation
- for multiclass classification with K classes: use K output neurons and softmax activation

$$(\hat{\mathbf{y}}(\mathbf{x}, \mathbf{w}))_k = p(y_k = 1) = g_{softmax}(\mathbf{z})_k = \frac{\exp((\mathbf{z})_k)}{\sum_j \exp((\mathbf{z})_j)}$$

$\rightarrow$ so for the complete output layer:

$$\hat{\mathbf{y}}(\mathbf{x}, \mathbf{w}) = \begin{bmatrix} p(y_1 = 1|\mathbf{x}) \\ p(y_2 = 1|\mathbf{x}) \\ \vdots \\ p(y_K = 1|\mathbf{x}) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} \exp((\mathbf{z})_j)} \exp(\mathbf{z})$$

Natural Pairing of Output Activation and Error Function

- For binary classification, use cross-entropy error:

$$L(\mathbf{w}) = -\frac{1}{N}\sum_{n=1}^{N}\{y_n \log \hat{y}_n + (1 - y_n)\log(1 - \hat{y}_n)\}$$

- For linear outputs, use mean squared error function:

$$L(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}^{N}\{\hat{y}(\mathbf{x}_n, \mathbf{w}) - y_n\}^2$$

- For multiclass classification, use generalization of cross-entropy error:

$$L(\mathbf{w}) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K} y_{kn} \log \hat{y}_k(\mathbf{x}_n, \mathbf{w})$$

Questions to Answer for Yourself / Discuss with Friends

- Transfer: Why is a softmax function used for multiclass classification instead of simply taking the (hard)-max?
- Transfer: What would happen if you used a ReLU output activation function for regression?

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

Week 2: From Logistic Regression to MLPs

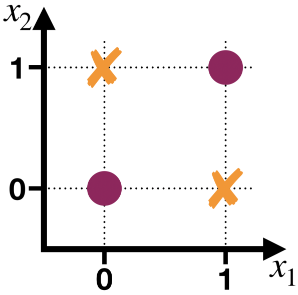
Representational Power of MLPs

Frank Hutter Abhinav Valada

University of Freiburg

Perceptrons Can Only Model Linearly Separable Functions

- Their decision boundary is a linear function
- Famously, they cannot learn, e.g., the XOR function [Minsky, Papert, 1969]



Multilayer Perceptrons are Universal Function Approximators

Theoretical result concerning the representational power of MLPs:

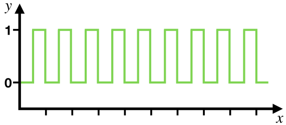
Universal Function Approximation Theorem [Cybenko, 1989]

1. Any Boolean function can be realized by an MLP with one hidden layer.
2. Any bounded continuous function can be approximated with arbitrary precision by a MLP with one hidden layer.

- The main idea of the proof:
  - Sums of (arbitrarily many) sigmoids can approximate any function
  - Similar to a Taylor expansion
- The hidden layer may have to have extremely many units
  - In the limit: infinitely many
- The theorem does not show that we can learn any function
  - It only shows that an MLP exists that approximates the function
  - It does not show that this MLP can be learned from data

The Power of Depth

- With a single hidden layer all computation has to happen in parallel
- Multiple layers allow us to re-use computation many times
  - Compositional structure of deep networks allows them to re-use pieces of computation exponentially often in terms of the network's depth.



Theorem: Depth Increases Representational Capacity Exponentially [Montufar, 2014]

A neural network with n_0 inputs and L layers of n units each, with ReLU activations can represent functions that have $\Omega((n/n_0)^{(L-1)n_0} n^{n_0})$ linear regions.

- Note: depth L is in the exponent, while width n is only in the base.

Questions to Answer for Yourself / Discuss with Friends

- Repetition: What does the universal function approximation theorem state? What does it *not* state?
- Repetition: What is the representational capacity of MLPs without a hidden layer, with one hidden layer, and with many hidden layers?

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

Week 2: From Logistic Regression to MLPs

Further Reading, Summary of the Week, References

Frank Hutter   Abhinav Valada

University of Freiburg

Summary by Learning Goals

- Cross-entropy loss is the negative $\log$ of the probability of predicting the correct label
- Logistic regression can be expressed by a perceptron with a logistic activation function
- Cross-entropy has a close connection to KL divergence and the maximum-likelihood estimator
- In multilayer perceptrons (MLPs) computations are performed layer-by-layer
- There are different activation functions (logistic, tanh, ReLU, linear, ELU, PReLU, etc.)
- Depending on the task, specific output unit activation functions are typically used
- Similarly, there is a natural pairing of output activations and error functions
- MLPs can represent any Boolean function, but this does not mean that they can learn any function from data
- Multiple layers (depth) increase the representational capacity of the model exponentially

Further Reading

Read chapter 6 of the Deep Learning Book for a detailed discussion of MLPs.

- For the latest developments on activation functions, you might find these blog posts interesting:
  - Swish vs Mish
  - FTSwishPlus and others

References

Goodfellow, I., Bengio, Y., Courville, A. (2016)
Deep Learning
M.I.T. Press.
https://www.deeplearningbook.org/

Minsky, M., Papert, S. (1969)
Perceptrons
M.I.T. Press
https://www.researchgate.net/publication/3081582_Review_of_'Perceptrons_An_Introduction_to_Computational_Geometry'_Minsky_M_and_Papert_S_1969

Cybenko, G. (1989)
Approximation by Superpositions of a Sigmoidal Function
Journal Mathematics of Control, Signals and Systems, 303-314
https://web.eecs.umich.edu/~cscott/smlrg/approx_by_superposition.pdf

Montúfar, G., Pascanu, R., Cho, K., Bengio, Y. (2014)
On the Number of Linear Regions of Deep Neural Networks
Conference NeurIPS 2014
https://arxiv.org/pdf/1402.1869.pdf

Clevert, D. A., Unterthiner, T., Hochreiter, S. (2015)
Fast and accurate deep network learning by exponential linear units (elus)
Conference ICLR 2015
https://arxiv.org/pdf/1511.07289.pdf

He, K., Zhang, X., Ren, S., Sun, J. (2015).
Delving deep into rectifiers: Surpassing human-level performance on imagenet classification
Conference IEEE international conference on computer vision 2015
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.725.4861&rep=rep1&type=pdf

Hendrycks, D., Gimpel, K. (2016)
Gaussian error linear units (gelus)
https://arxiv.org/pdf/1606.08415.pdf

Ramachandran, P., Zoph, B., Le, Q. V. (2017)
Searching for activation functions.
https://arxiv.org/pdf/1710.05941.pdf