Deep Learning

03 - Model Training. Loss function

Fabrizio Silvestri

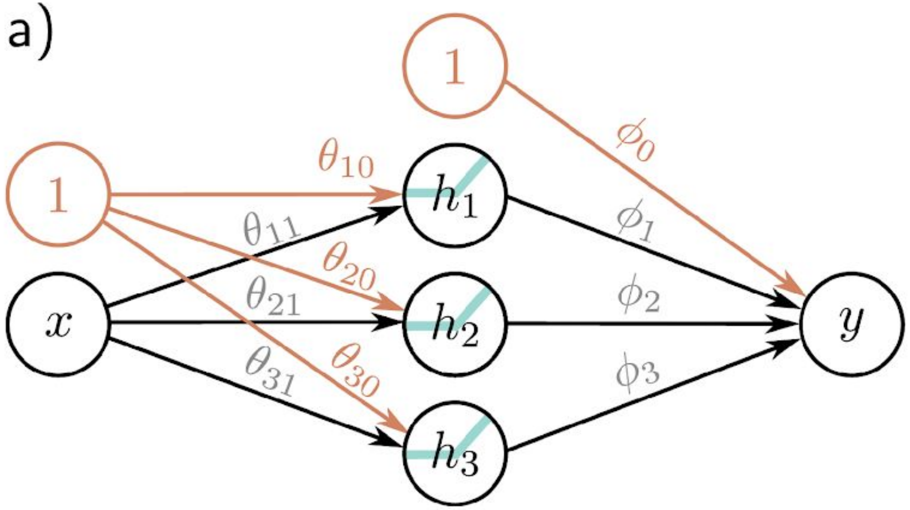
Recap

Recapping

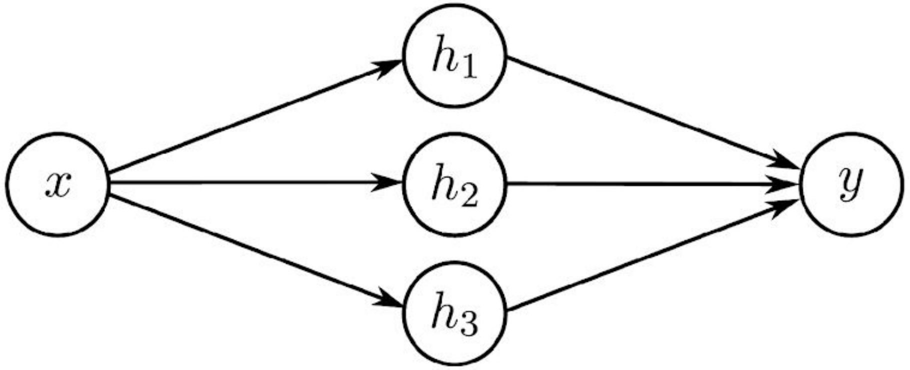
- We have described linear regression, shallow neural networks, and deep neural networks.
- Each represents a family of functions that map input to output, where the particular member of the family is determined by the model parameters ϕ.
- When we train these models, we seek the parameters that produce the best possible mapping from input to output for the task we are considering.
- What does “best possible” mapping mean?

Neural Networks

a)



b)



Universal Approximation Theorem

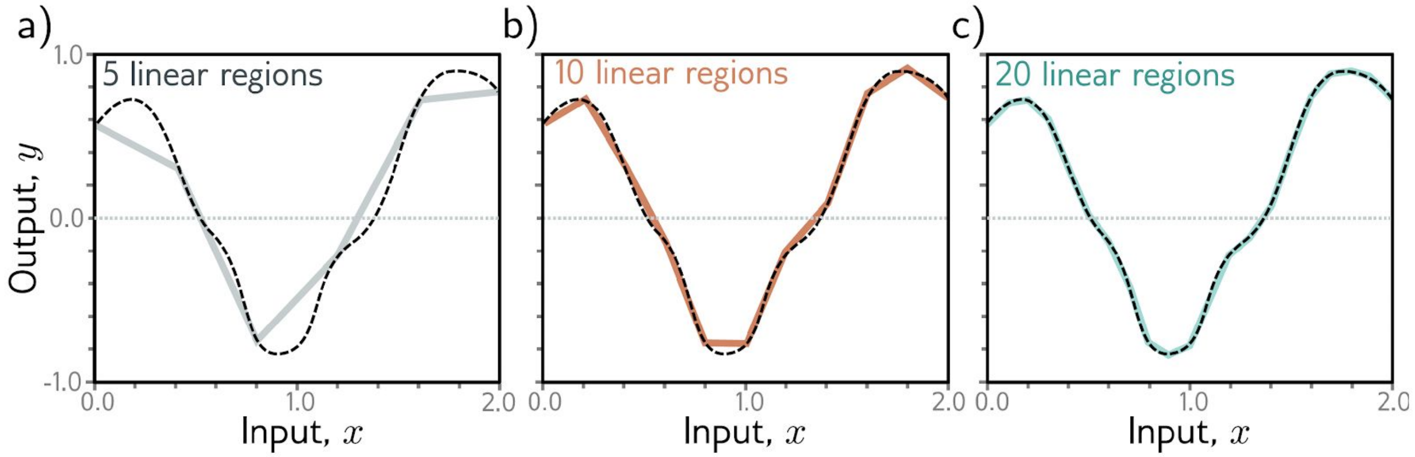
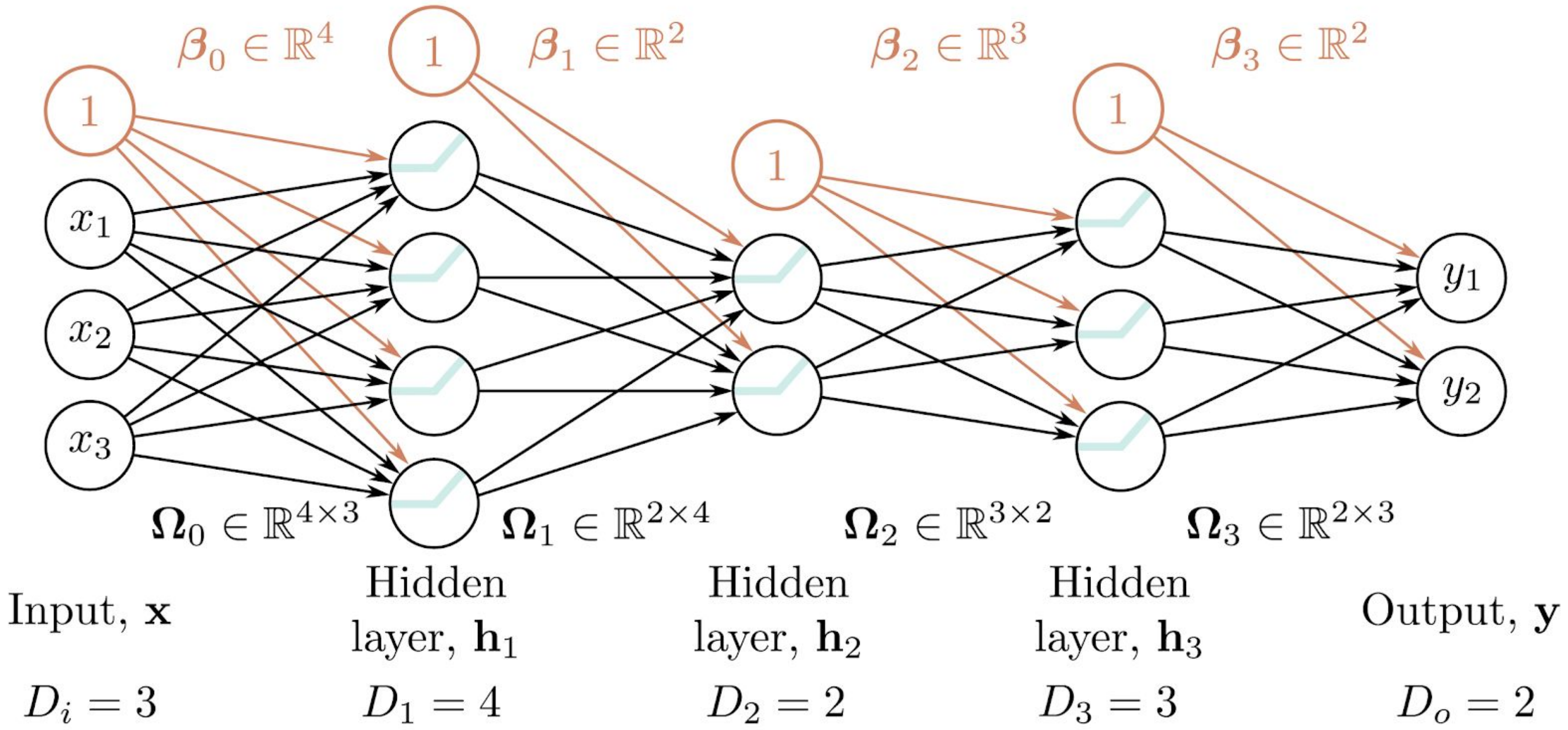


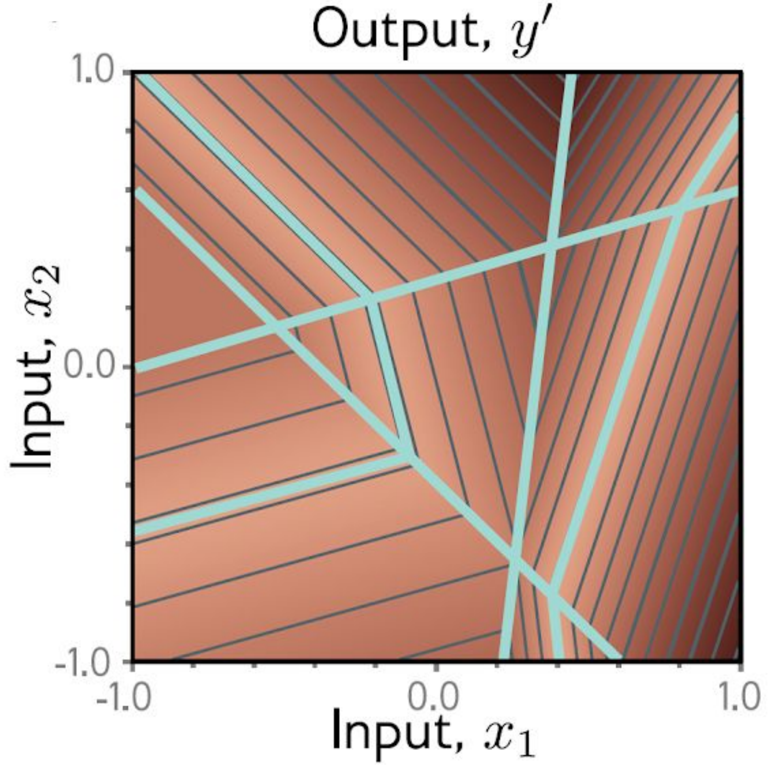
Figure 3.5 Approximation of a 1D function (dashed line) by a piecewise linear model. a–c) As the number of regions increases, the model becomes closer and closer to the continuous function. A neural network with a scalar input creates one extra linear region per hidden unit. The universal approximation theorem proves that, with enough hidden units, there exists a shallow neural network can describe any given continuous function defined on a compact subset of $\mathbb{R}^D$ to arbitrary precision.

SAPIENZA
UNIVERSITÀ DI ROMA

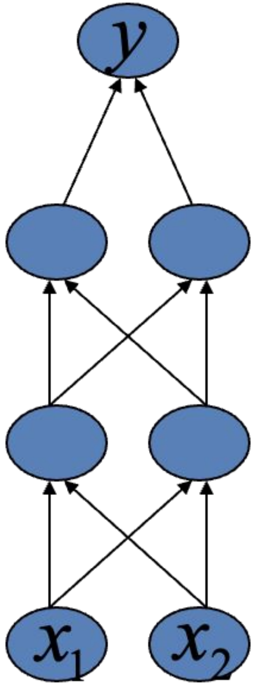
More Hidden Layers


$\boldsymbol{\beta}_0 \in \mathbb{R}^4$
$\boldsymbol{\beta}_1 \in \mathbb{R}^2$
$\boldsymbol{\beta}_2 \in \mathbb{R}^3$
$\boldsymbol{\beta}_3 \in \mathbb{R}^2$
1
1
1
1
x_1
x_2
x_3
y_1
y_2
$\boldsymbol{\Omega}_0 \in \mathbb{R}^{4\times3}$
$\boldsymbol{\Omega}_1 \in \mathbb{R}^{2\times4}$
$\boldsymbol{\Omega}_2 \in \mathbb{R}^{3\times2}$
$\boldsymbol{\Omega}_3 \in \mathbb{R}^{2\times3}$
Input, $\mathbf{x}$
Hidden layer, $\mathbf{h}_1$
Hidden layer, $\mathbf{h}_2$
Hidden layer, $\mathbf{h}_3$
Output, $\mathbf{y}$
$D_i = 3$
$D_1 = 4$
$D_2 = 2$
$D_3 = 3$
$D_o = 2$

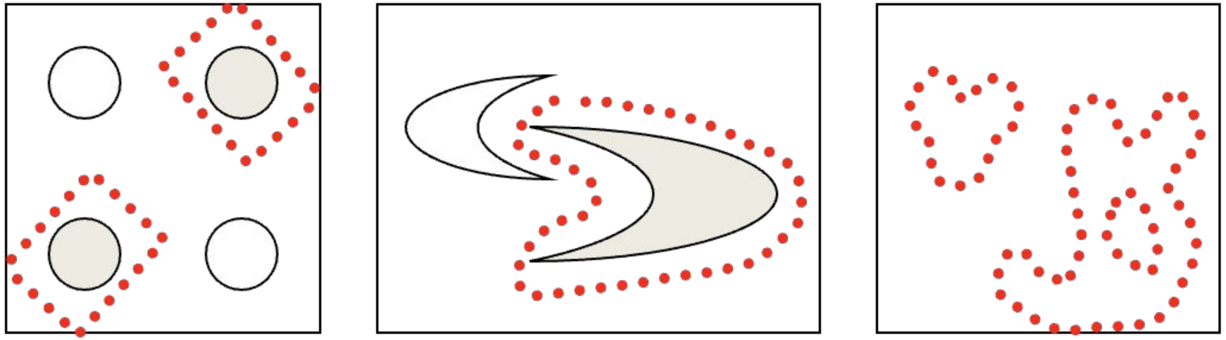
More Hidden Layers



Decision Boundary



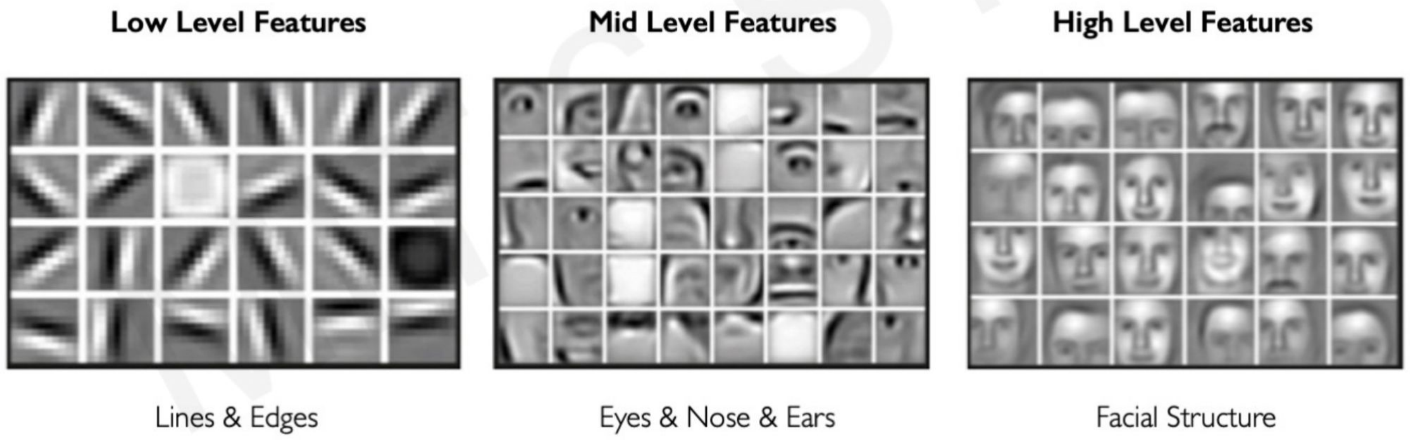
- 2 hidden layer
  - Combinations of convex regions



Example from to Eric Postma via Jason Eisner

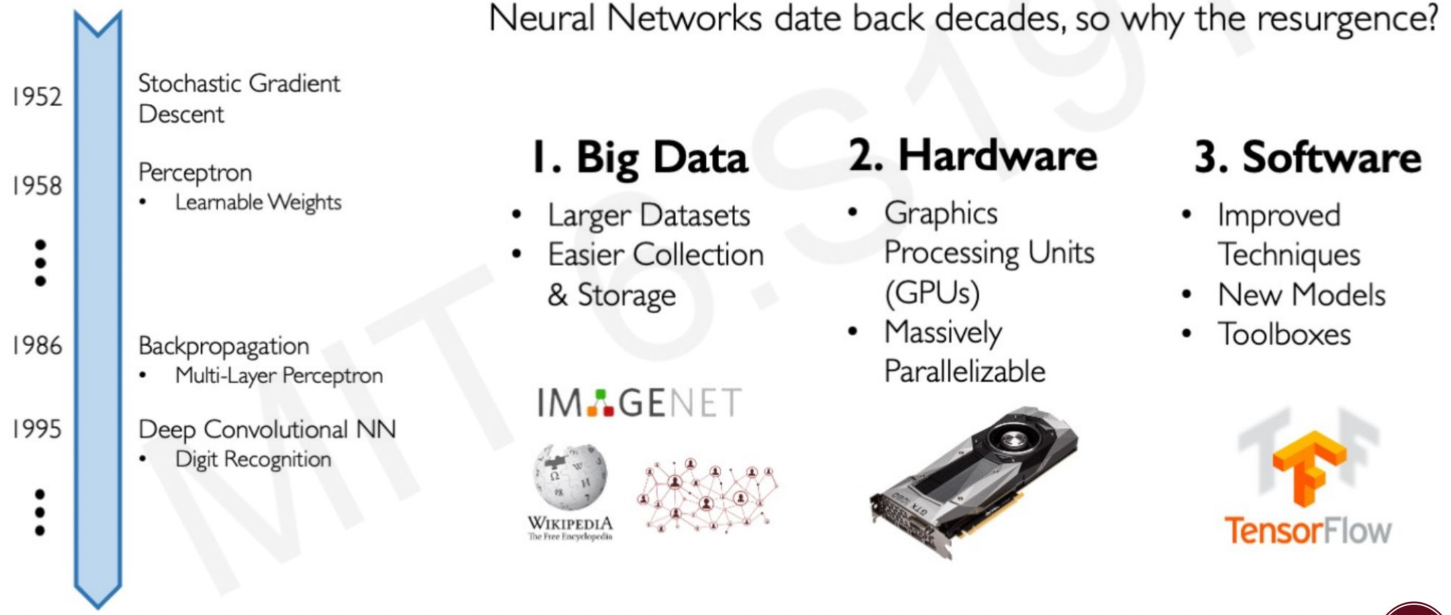
Why Deep Learning

- Features that are engineered by hand are difficult to maintain, error prone, and not necessarily the best for the task
- Can we learn feature directly from data?



Why Now?

- ANNs are not new (The Perceptron is almost 65 y.o.)



1952	Stochastic Gradient Descent
1958	Perceptron • Learnable Weights
⋮	
1986	Backpropagation • Multi-Layer Perceptron
1995	Deep Convolutional NN • Digit Recognition
⋮	

Neural Networks date back decades, so why the resurgence?

1. Big Data
- Larger Datasets
- Easier Collection & Storage

2. Hardware
- Graphics Processing Units (GPUs)
- Massively Parallelizable

3. Software
- Improved Techniques
- New Models
- Toolboxes

Loss Functions

(Chapter 5)

Requirements

- That definition requires a training dataset $\{x_i, y_i\}$ of input/output pairs.
- A loss function, or cost function, $L[\boldsymbol{\phi}]$ returns a single number that describes the mismatch between the model predictions $f[x_i, \boldsymbol{\phi}]$ and their corresponding ground-truth outputs y_i.
- During training, we seek parameter values $\tilde{\boldsymbol{\phi}}$ that minimize the loss and hence map the training inputs to the outputs as closely as possible.

$$\tilde{\phi} = \arg\min_{\phi} L[\phi]$$

Maximum Likelihood

Models as Conditional Probability Distributions

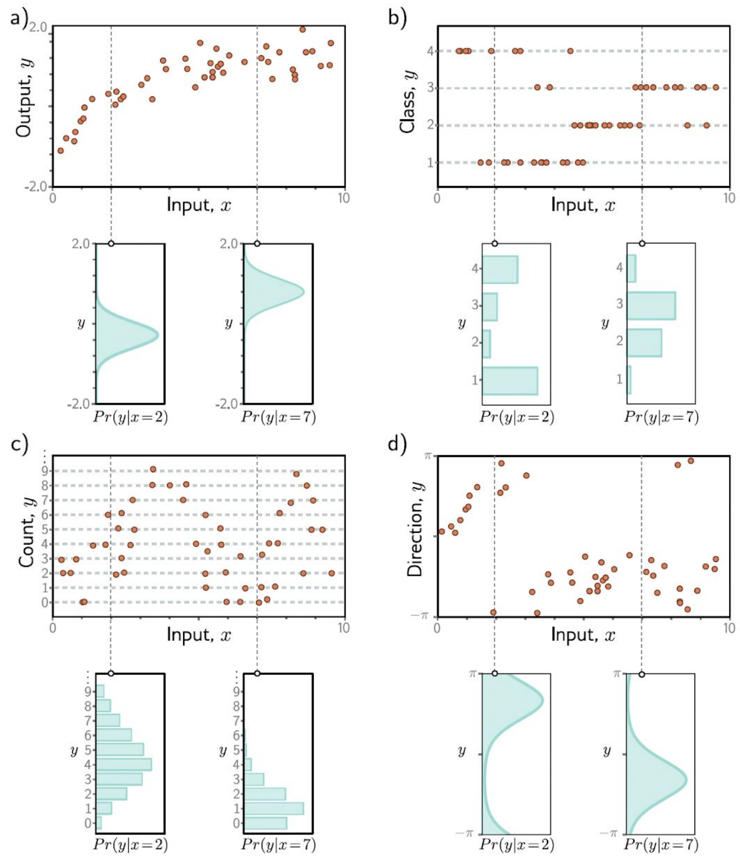


Figure 5.1 Predicting distributions over outputs. a) Regression task, where the goal is to predict a real-valued output y from the input x based on training data $\{x_i, y_i\}$ (orange points). For each input value x, the machine learning model predicts a distribution $Pr(y|x)$ over the output $y \in \mathbb{R}$ (cyan curves show distributions for $x = 2.0$ and $x = 7.0$). The loss function aims to maximize the probability of the observed training outputs y_i under the distribution predicted from the corresponding inputs x_i. b) To predict discrete classes $y \in \{1, 2, 3, 4\}$ in a classification task, we use a discrete probability distribution, so the model predicts a different histogram over the four possible values of y_i for each value of x_i. c) To predict counts $y \in \{0, 1, 2, \ldots\}$ and d) direction $y \in (-\pi, \pi]$, we use distributions defined over positive integers and circular domains, respectively.

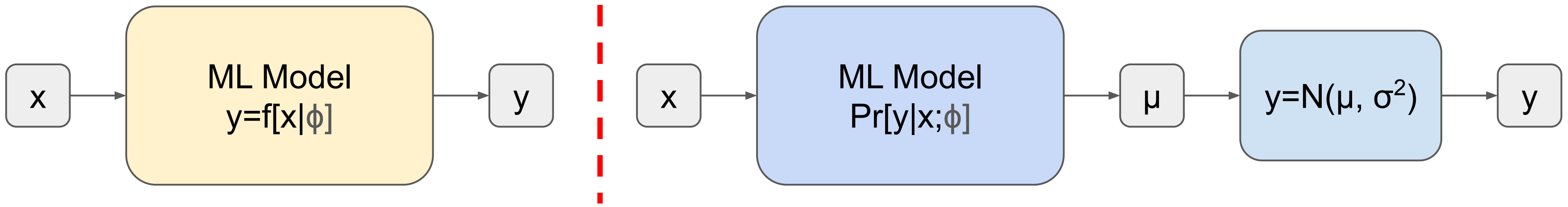
Computing a Distribution over Outputs

- How do we do that?
- Simple
    - $\mathbf{y}=\mathrm{f}[\mathbf{x}, \boldsymbol{\phi}] \Rightarrow \Pr\{\mathbf{y}|\mathbf{x}; \boldsymbol{\phi}\}$

Image courtesy of: https://www.polygon.com/22947332/spider-man-no-way-home-pointing-meme

Example

- Suppose the prediction domain is the set of real numbers, i.e., $y \in \mathbb{R}$.
- Here, we might choose the univariate normal distribution, which is defined on $\mathbb{R}$.
  - This distribution is defined by the mean μ and variance σ^2, so $\boldsymbol{\theta} = \{\mu, \sigma^2\}$.
- The machine learning model might predict the mean μ, and the variance σ^2 could be treated as an unknown constant.



MLE Criterion

- The model now computes different distribution parameters $\theta_i = f[x_i, \phi]$ for each training input x_i.
    - Each observed training output y_i should have high probability under its corresponding distribution $Pr\{y_i|\theta_i\}$.
- Hence, we choose the model parameters ϕ so that they maximize the combined probability across all *i* training examples

$$
\begin{aligned}
\hat{\phi} &= \underset{\phi}{\operatorname{argmax}} \left[\overset{\text{Likelihood}}{\boxed{\prod_{i=1}^{I} Pr(\mathbf{y}_i|\mathbf{x}_i)}} \right] \\
&= \underset{\phi}{\operatorname{argmax}} \left[\prod_{i=1}^{I} Pr(\mathbf{y}_i|\boldsymbol{\theta}_i) \right] \\
&= \underset{\phi}{\operatorname{argmax}} \left[\prod_{i=1}^{I} Pr(\mathbf{y}_i|\mathbf{f}[\mathbf{x}_i, \boldsymbol{\phi}]) \right]
\end{aligned}
$$

MLE Criterion: Assumptions

- Observations (x_i, y_i) are:
    - Independent
    - Identically distributed
- Observations i.i.d.

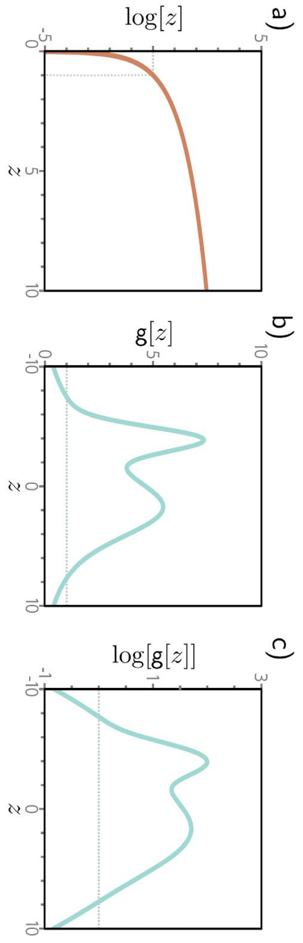
$$\begin{aligned}\hat{\phi} &= \underset{\phi}{\operatorname{argmax}}\left[\prod_{i=1}^{I} Pr(\mathbf{y}_i|\mathbf{x}_i)\right] \\ &= \underset{\phi}{\operatorname{argmax}}\left[\prod_{i=1}^{I} Pr(\mathbf{y}_i|\boldsymbol{\theta}_i)\right] \\ &= \underset{\phi}{\operatorname{argmax}}\left[\prod_{i=1}^{I} Pr(\mathbf{y}_i|\mathbf{f}[\mathbf{x}_i,\phi])\right]\end{aligned}$$

$$\boxed{Pr(\mathbf{y}_1,\mathbf{y}_2,\ldots,\mathbf{y}_I|\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_I) = \prod_{i=1}^{I} Pr(\mathbf{y}_i|\mathbf{x}_i)}$$

SAPIENZA
UNIVERSITÀ DI ROMA

Log Likelihood

$$
\begin{aligned}
\hat{\boldsymbol{\phi}} &= \underset{\boldsymbol{\phi}}{\operatorname{argmax}} \left[\prod_{i=1}^{I} Pr(\mathbf{y}_i | \mathbf{f}[\mathbf{x}_i, \boldsymbol{\phi}]) \right] \\
&= \underset{\boldsymbol{\phi}}{\operatorname{argmax}} \left[\boxed{\log} \left[\prod_{i=1}^{I} Pr(\mathbf{y}_i | \mathbf{f}[\mathbf{x}_i, \boldsymbol{\phi}]) \right] \right] \\
&= \underset{\boldsymbol{\phi}}{\operatorname{argmax}} \left[\sum_{i=1}^{I} \log \left[Pr(\mathbf{y}_i | \mathbf{f}[\mathbf{x}_i, \boldsymbol{\phi}]) \right] \right]
\end{aligned}
$$



Minimizing Negative Log Likelihood (NLL)

- We note that, by convention, model fitting problems are framed in terms of minimizing a loss.
- To convert the maximum log-likelihood criterion to a minimization problem, we multiply by minus one, which gives us the negative log-likelihood criterion:

$$\begin{aligned}\hat{\phi} &= \underset{\phi}{\operatorname{argmin}}\left[\boxed{-}\sum_{i=1}^{I}\log\left[Pr(\mathbf{y}_i|\mathbf{f}[\mathbf{x}_i,\phi])\right]\right] \\ &= \underset{\phi}{\operatorname{argmin}}\left[\mathrm{L}[\phi]\right],\end{aligned}$$

Inference

- The network no longer directly predicts the outputs y but instead determines a probability distribution over y.
- When we perform inference, we often want a point estimate rather than a distribution, so we return the maximum of the distribution:

$$\hat{\mathbf{y}} = \underset{\mathbf{y}}{\operatorname{argmax}}\left[Pr(\mathbf{y}|\mathbf{f}[\mathbf{x}, \hat{\phi}])\right]$$

- It is usually possible to find an expression for this in terms of the distribution parameters θ predicted by the model.
- For example, in the univariate normal distribution, the maximum occurs at the mean μ

Recipe for a Loss

1. Choose a suitable probability distribution Pr(y|θ) defined over the domain of the predictions y with distribution parameters θ.
2. Set the machine learning model f[x, ϕ] to predict one or more of these parameters, so θ = f[x, ϕ] and Pr(y|θ) = Pr(y|f[x, ϕ]).
3. To train the model, find the network parameters $\hat{\phi}$ that minimize the negative log-likelihood loss function over the training dataset pairs {x_i, y_i}:

$$\hat{\phi} = \underset{\phi}{\operatorname{argmin}}\Big[L[\phi]\Big] = \underset{\phi}{\operatorname{argmin}}\left[-\sum_{i=1}^{I}\log\Big[Pr(\mathbf{y}_i|\mathbf{f}[\mathbf{x}_i,\phi])\Big]\right]$$

4. To perform inference for a new test example x, return either the full distribution Pr(y|f[x,$\hat{\phi}$|) or the maximum of this distribution.

Example 1: Univariate Regression

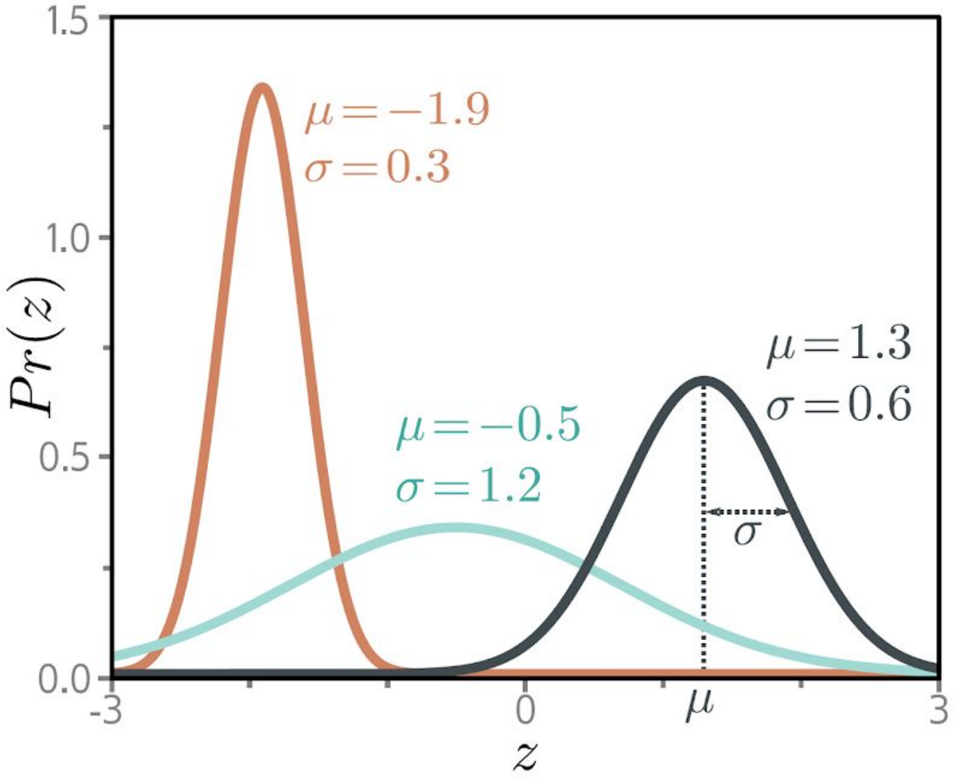


Figure 5.3 The univariate normal distribution (also known as the Gaussian distribution) is defined on the real line $z \in \mathbb{R}$ and has parameters μ and σ^2. The mean μ determines the position of the peak. The positive root of the variance σ^2 (the standard deviation) determines the width of the distribution. Since the total probability density sums to one, the peak becomes higher as the variance decreases and the distribution becomes narrower.

Example 1: Univariate Regression

- Goal: predict a single scalar output y ∈ ℝ from input x using a model f[x, ϕ] with parameters ϕ.
- Following the recipe, we choose a probability distribution over the output domain y

$$Pr(y|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(y-\mu)^2}{2\sigma^2}\right]$$

pdf
Probability Density Function

- Second, we set the machine learning model f[x,ϕ] to compute one or more of the parameters of this distribution. Here, we just compute the mean so μ = f[x, ϕ]:

$$Pr(y|\mathrm{f}[\mathbf{x},\phi],\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(y-\mathrm{f}[\mathbf{x},\phi])^2}{2\sigma^2}\right]$$

SAPIENZA
UNIVERSITÀ DI ROMA

Example 1: Univariate Regression

- We aim to find the parameters ϕ that make the training data {x_i, y_i} most probable under this distribution.
- To accomplish this, we choose a loss function L[ϕ] based on the negative log-likelihood:

$$\begin{aligned} L[\phi] &= -\sum_{i=1}^{I} \log\left[Pr(y_i|\mathrm{f}[\mathbf{x}_i,\phi],\sigma^2)\right] \\ &= -\sum_{i=1}^{I} \log\left[\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left[-\frac{(y_i-\mathrm{f}[\mathbf{x}_i,\phi])^2}{2\sigma^2}\right]\right] \end{aligned}$$

NLLLoss

- When we train the model, we seek parameters $\hat{\phi}$ that minimize this loss

Example 1: Univariate Regression

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[-\sum_{i=1}^{I} \log \left[\frac{1}{\sqrt{2\pi\sigma^2}} \exp \left[-\frac{(y_i - \mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}])^2}{2\sigma^2} \right] \right] \right]$$

Example 1: Univariate Regression

$$
\begin{aligned}
\hat{\phi} &= \underset{\phi}{\operatorname{argmin}} \left[-\sum_{i=1}^{I} \log \left[\frac{1}{\sqrt{2\pi\sigma^2}} \exp \left[-\frac{(y_i - \mathrm{f}[\mathbf{x}_i, \phi])^2}{2\sigma^2} \right] \right] \right] \\
&= \underset{\phi}{\operatorname{argmin}} \left[-\sum_{i=1}^{I} \log \left[\frac{1}{\sqrt{2\pi\sigma^2}} \right] - \frac{(y_i - \mathrm{f}[\mathbf{x}_i, \phi])^2}{2\sigma^2} \right]
\end{aligned}
$$

SAPIENZA UNIVERSITÀ DI ROMA

Example 1: Univariate Regression

$$
\begin{aligned}
\hat{\boldsymbol{\phi}} &= \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[-\sum_{i=1}^{I} \log \left[\frac{1}{\sqrt{2\pi\sigma^2}} \exp \left[-\frac{(y_i - \mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}])^2}{2\sigma^2} \right] \right] \right] \\
&= \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[-\sum_{i=1}^{I} \log \left[\frac{1}{\sqrt{2\pi\sigma^2}} \right] - \frac{(y_i - \mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}])^2}{2\sigma^2} \right] \\
&= \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[-\sum_{i=1}^{I} - \frac{(y_i - \mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}])^2}{2\sigma^2} \right]
\end{aligned}
$$

SAPIENZA
UNIVERSITÀ DI ROMA

Example 1: Univariate Regression

$$
\begin{aligned}
\hat{\phi} &= \underset{\phi}{\operatorname{argmin}} \left[-\sum_{i=1}^{I} \log \left[\frac{1}{\sqrt{2\pi\sigma^2}} \exp \left[-\frac{(y_i - \mathrm{f}[\mathbf{x}_i, \phi])^2}{2\sigma^2} \right] \right] \right] \\
&= \underset{\phi}{\operatorname{argmin}} \left[-\sum_{i=1}^{I} \log \left[\frac{1}{\sqrt{2\pi\sigma^2}} \right] - \frac{(y_i - \mathrm{f}[\mathbf{x}_i, \phi])^2}{2\sigma^2} \right] \\
&= \underset{\phi}{\operatorname{argmin}} \left[-\sum_{i=1}^{I} -\frac{(y_i - \mathrm{f}[\mathbf{x}_i, \phi])^2}{2\sigma^2} \right] \\
&= \boxed{\underset{\phi}{\operatorname{argmin}} \left[\sum_{i=1}^{I} (y_i - \mathrm{f}[\mathbf{x}_i, \phi])^2 \right]}
\end{aligned}
$$

MSELoss

Colab time: Least Squares Loss

https://colab.research.google.com/drive/1gy3KiCiKl4ZoI2eKgH9XIzZif-S0hUPB

What about the Variance?

Homoscedastic Regression

- Treating σ^2 as a parameter of the model and minimizing the loss with respect to both the model parameters ϕ and the distribution variance σ^2:

$$\hat{\phi} = \underset{\phi,\sigma^2}{\operatorname{argmin}}\left[-\sum_{i=1}^{I}\log\left[\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left[-\frac{(y_i-\mathrm{f}[\mathbf{x}_i,\phi])^2}{2\sigma^2}\right]\right]\right]$$

- At inference time, the model predicts the mean μ = f[x, $\hat{\phi}$] from the input, and we learned the variance σ^2 during the training process.
- The former is the best prediction. The latter tells us about the *uncertainty of the prediction*.

SAPIENZA UNIVERSITÀ DI ROMA

Heteroscedastic Regression

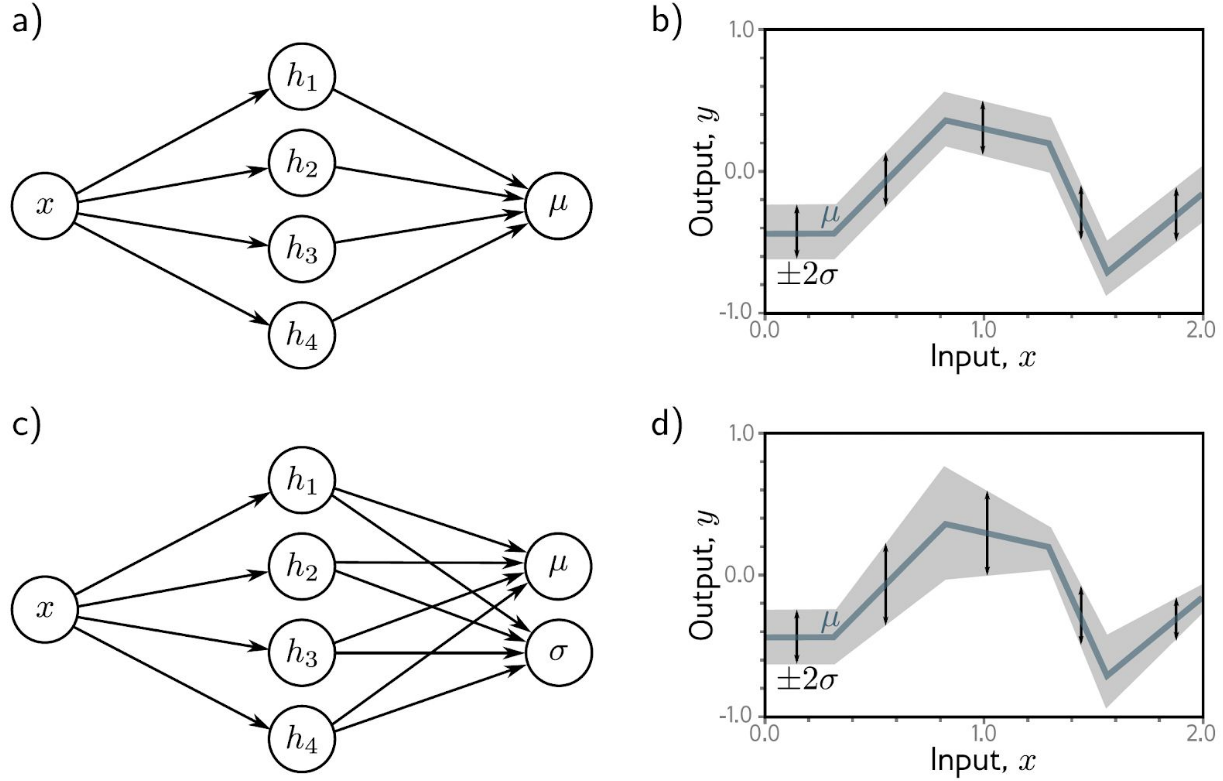
- When the uncertainty of the model varies as a function of the input data, we refer to this as heteroscedastic.
- Train $\mathbf{f}[\mathbf{x},\boldsymbol{\phi}] = [f_1[\mathbf{x},\boldsymbol{\phi}],f_2[\mathbf{x},\boldsymbol{\phi}]]$
    - $f_2[\mathbf{x},\boldsymbol{\phi}]$ can be negative → square it

$$
\begin{aligned}
\boxed{\mu \;=\; \mathrm{f}_1[\mathbf{x},\boldsymbol{\phi}]} \\
\boxed{\sigma^2 \;=\; \mathrm{f}_2[\mathbf{x},\boldsymbol{\phi}]^2}
\end{aligned}
$$

- which results in the loss function:

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}}\left[-\sum_{i=1}^{I}\log\left[\frac{1}{\sqrt{2\pi\boxed{\mathrm{f}_2[\mathbf{x}_i,\boldsymbol{\phi}]^2}}}\right]-\frac{(y_i-\boxed{\mathrm{f}_1[\mathbf{x}_i,\boldsymbol{\phi}])}^2}{2\boxed{\mathrm{f}_2[\mathbf{x}_i,\boldsymbol{\phi}]^2}}\right]$$

Heteroscedastic vs. Homoscedastic Regression



Example 2: Binary Classification

- In binary classification, the goal is to assign the data x to one of two discrete classes $y \in \{0,1\}$. In this context, we refer to y as a label.
- Examples of binary classification include:
    - predicting whether a restaurant review is positive ($y = 1$) or negative ($y = 0$) from text data x
    - predicting whether a tumor is present ($y = 1$) or absent ($y = 0$) from an MRI scan x

SAPIENZA UNIVERSITÀ DI ROMA

Reminder: Recipe for a Loss

1. Choose a suitable probability distribution Pr(y|θ) defined over the domain of the predictions y with distribution parameters θ.
2. Set the machine learning model f[x, ϕ] to predict one or more of these parameters, so θ = f[x, ϕ] and Pr(y|θ) = Pr(y|f[x, ϕ]).
3. To train the model, find the network parameters $\hat{\phi}$ that minimize the negative log-likelihood loss function over the training dataset pairs $\{x_i, y_i\}$:

$$\hat{\phi} = \underset{\phi}{\operatorname{argmin}} \Big[L[\phi] \Big] = \underset{\phi}{\operatorname{argmin}} \left[-\sum_{i=1}^{I} \log\Big[Pr(\mathbf{y}_i | \mathbf{f}[\mathbf{x}_i, \phi]) \Big] \right]$$

4. To perform inference for a new test example x, return either the full distribution Pr(y|f[x,$\hat{\phi}$|) or the maximum of this distribution.

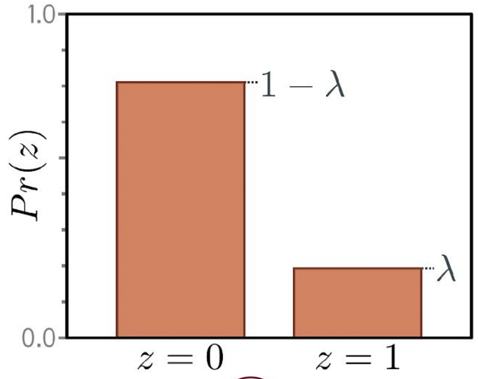
Example 2: Binary Classification

- Let's build the loss function, then…
- First, we choose a probability distribution over the output space y ∈ {0, 1}. A suitable choice is the Bernoulli distribution, which is defined on the domain {0, 1}. This has a single parameter λ ∈ [0, 1] that represents the probability that y takes the value one

$$Pr(y|\lambda) = \begin{cases} 1-\lambda & y=0 \\ \lambda & y=1 \end{cases}$$

- which can equivalently be written as:

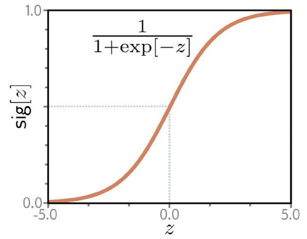
$$Pr(y|\lambda) = (1-\lambda)^{1-y} \cdot \lambda^{y}$$



Example 2: Binary Classification

- Second, we set the machine learning model f[**x**,**ϕ**] to predict the single distribution parameter λ.
- However, λ can only take values in the range [0, 1], and we cannot guarantee that the network output will lie in this range.
- Consequently, we pass the network output through a function that maps the real numbers R to [0, 1]. A suitable function is the logistic sigmoid

$$\mathrm{sig}[z] = \frac{1}{1+\exp[-z]}$$



- Hence, we predict the distribution parameter as λ = sig[f[**x**,**ϕ**]]

Example 2: Binary Classification

- The likelihood is, therefore:

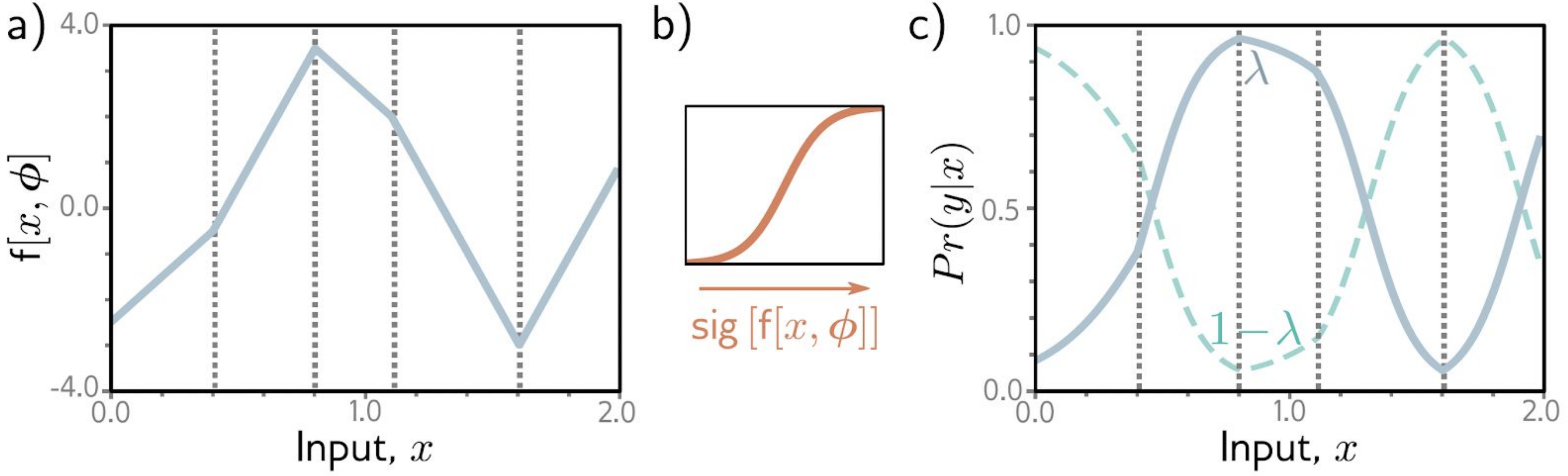
$$Pr(y|\mathbf{x}) = (1 - \text{sig}[\text{f}[\mathbf{x}, \boldsymbol{\phi}]])^{1-y} \cdot \text{sig}[\text{f}[\mathbf{x}, \boldsymbol{\phi}]]^{y}$$

- And the Negative Log Likelihood is:

$$L[\boldsymbol{\phi}] = \sum_{i=1}^{I} -(1 - y_i) \log\Big[1 - \text{sig}[\text{f}[\mathbf{x}_i, \boldsymbol{\phi}]]\Big] - y_i \log\Big[\text{sig}[\text{f}[\mathbf{x}_i, \boldsymbol{\phi}]]\Big]$$

- A.k.a. Binary Cross Entropy Loss

Example 2: Binary Classification



Example 3: Multiclass Classification

- Goal is to assign an input data example x to one of $K > 2$ classes, so $y \in \{1, 2, \ldots, K\}$.
- Real-world examples include: (i) predicting which of $K = 10$ digits y is present in an image x of a handwritten number and (ii) predicting which of K possible words y follows an incomplete sentence x.

Reminder: Recipe for a Loss

1. Choose a suitable probability distribution Pr(y|θ) defined over the domain of the predictions y with distribution parameters θ.
2. Set the machine learning model f[x, ϕ] to predict one or more of these parameters, so θ = f[x, ϕ] and Pr(y|θ) = Pr(y|f[x, ϕ]).
3. To train the model, find the network parameters $\hat{\phi}$ that minimize the negative log-likelihood loss function over the training dataset pairs $\{x_i, y_i\}$:

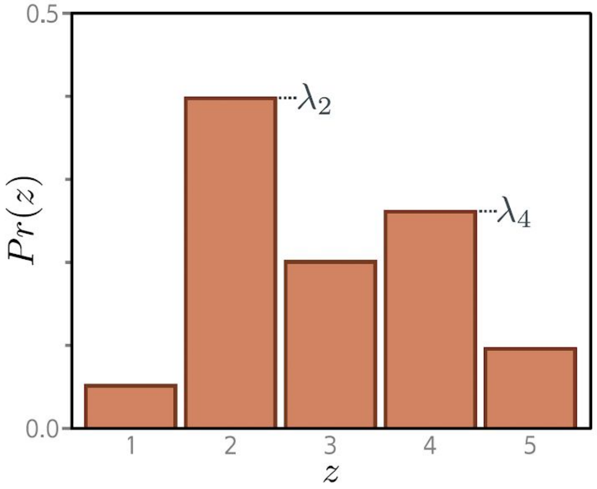
$$\hat{\phi} = \underset{\phi}{\operatorname{argmin}}\Big[L[\phi]\Big] = \underset{\phi}{\operatorname{argmin}}\left[-\sum_{i=1}^{I}\log\Big[Pr(\mathbf{y}_i|\mathbf{f}[\mathbf{x}_i,\phi])\Big]\right]$$

4. To perform inference for a new test example x, return either the full distribution Pr(y|f[x,$\hat{\phi}$|) or the maximum of this distribution.

Example 3: Multiclass Classification

- We first choose a distribution over the prediction space y.
- In this case, we have $y \in \{1, 2, \dots, K\}$, so we choose the categorical distribution, which is defined on this domain. This has K parameters $\lambda_1, \lambda_2, \dots, \lambda_K$, which determine the probability of each category: $Pr(y = k) = \lambda_k$

- The parameters are constrained to take values between zero and one, and they must collectively sum to one to ensure a valid probability distribution.



Example 3: Multiclass Classification

- Then we use a network **f**[**x**,**ϕ**] with K outputs to compute these K parameters from the input x.
- Unfortunately, the network outputs will not necessarily obey the sum-to-1 constraints.
  - Consequently, we pass the K outputs of the network through a function that ensures these constraints are respected.
- A suitable choice is the softmax function. This takes an arbitrary vector of length K and returns a vector of the same length but where the elements are now in the range [0, 1] and sum to one. The kth output of the softmax function is:

$$\mathrm{softmax}_k[\mathbf{z}] = \frac{\exp[z_k]}{\sum_{k'=1}^{K} \exp[z_{k'}]}$$

$$\text{softmax}_k[\mathbf{z}] = \frac{\exp[z_k]}{\sum_{k'=1}^{K} \exp[z_{k'}]}$$

Example 3: Multiclass Classification

- Therefore the likelihood is given by:

$$Pr(y = k|\mathbf{x}) = \text{softmax}_k\Big[\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]\Big]$$

- And the NLL is given by:

$$\begin{aligned} L[\boldsymbol{\phi}] &= -\sum_{i=1}^{I} \log\Big[\text{softmax}_{y_i}\Big[\mathbf{f}[\mathbf{x}_i, \boldsymbol{\phi}]\Big]\Big] \\ &= -\sum_{i=1}^{I}\left(\text{f}_{y_i}[\mathbf{x}_i, \boldsymbol{\phi}] - \log\left[\sum_{k'=1}^{K} \exp[\,\text{f}_{k'}[\mathbf{x}_i, \boldsymbol{\phi}]]\right]\right) \end{aligned}$$

SAPIENZA
UNIVERSITÀ DI ROMA

Colab time: Multiclass cross-entropy loss

https://colab.research.google.com/drive/1eP81zYuaSWQitBYrNUjnvXc9Yqf6NAjb

Multiple Output

- Often, we wish to make more than one prediction with the same model, so the target output y is a vector.
  - For example, we might want to predict a molecule’s melting and boiling point (a multivariate regression problem, or the object class at every point in an image (a multivariate classification problem.
- While it is possible to define multivariate probability distributions and use a neural network to model their parameters as a function of the input, it is more usual to treat each prediction as independent.

$$Pr(\mathbf{y}|\mathbf{f}[\mathbf{x}_i, \phi]) = \prod_d Pr(y_d|\mathbf{f}_d[\mathbf{x}_i, \phi])$$

Multiple Output

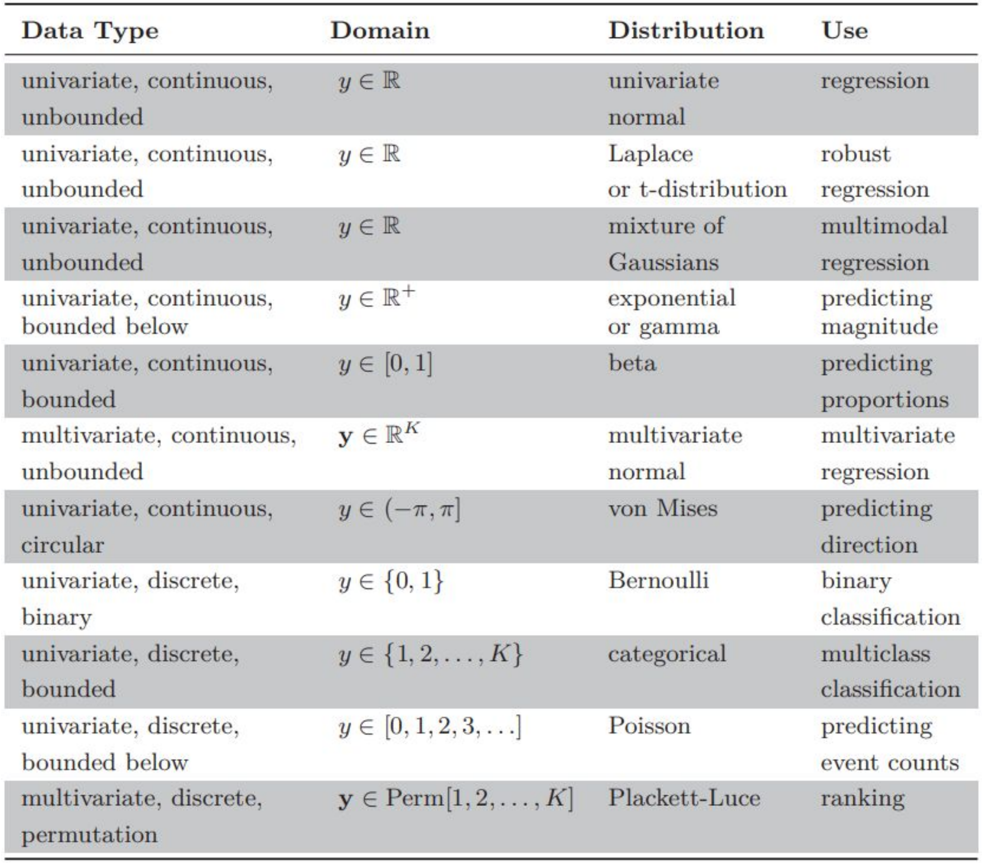
- The minimization is then done in the usual way by minimizing the Negative Log Likelihood

$$L[\phi] = -\sum_{i=1}^{I} \log\Big[Pr(\mathbf{y}|\mathbf{f}[\mathbf{x}_i, \phi])\Big] = -\sum_{i=1}^{I}\sum_{d} \log\Big[Pr(y_{id}|\mathbf{f}_d[\mathbf{x}_i, \phi])\Big]$$

where y_{id} is the dth output from the ith training example.

SAPIENZA UNIVERSITÀ DI ROMA

Distributions for Loss Functions for Different Prediction Types



Data Type	Domain	Distribution	Use
univariate, continuous, unbounded	$y \in \mathbb{R}$	univariate normal	regression
univariate, continuous, unbounded	$y \in \mathbb{R}$	Laplace or t-distribution	robust regression
univariate, continuous, unbounded	$y \in \mathbb{R}$	mixture of Gaussians	multimodal regression
univariate, continuous, bounded below	$y \in \mathbb{R}^+$	exponential or gamma	predicting magnitude
univariate, continuous, bounded	$y \in [0, 1]$	beta	predicting proportions
multivariate, continuous, unbounded	$\mathbf{y} \in \mathbb{R}^K$	multivariate normal	multivariate regression
univariate, continuous, circular	$y \in (-\pi, \pi]$	von Mises	predicting direction
univariate, discrete, binary	$y \in \{0, 1\}$	Bernoulli	binary classification
univariate, discrete, bounded	$y \in \{1, 2, \ldots, K\}$	categorical	multiclass classification
univariate, discrete, bounded below	$y \in [0, 1, 2, 3, \ldots]$	Poisson	predicting event counts
multivariate, discrete, permutation	$\mathbf{y} \in \text{Perm}[1, 2, \ldots, K]$	Plackett-Luce	ranking

SAPIENZA
UNIVERSITÀ DI ROMA

Why Cross Entropy?

- We have often use the term Cross Entropy to describe the loss commonly used in classification problems, but… Why is it a form of “Entropy”?
- Consider the Kullback-Leibler divergence between two distributions p, and q. It measures the “distance” between the two distributions:

$$D_{KL}[q||p] = \int_{-\infty}^{\infty} q(z)\log[q(z)]dz - \int_{-\infty}^{\infty} q(z)\log[p(z)]dz$$

- Now consider that we observe an empirical data distribution at points $\{y_i\}$.
    - We can describe this as a weighted sum of point masses:

Dirac’s Delta.

$$\delta(x-\alpha) = \frac{1}{2\pi}\int_{-\infty}^{\infty} dp\ \cos(px - p\alpha)$$

$$q(y) = \frac{1}{I}\sum_{i=1}^{I} \boxed{\delta}[y - y_i]$$

$$q(y) = \frac{1}{I}\sum_{i=1}^{I}\delta[y - y_i]$$

Why Cross Entropy?

- We want to minimize the KL divergence between q(y) and the real distribution Pr(y|θ):

$$\begin{aligned}\hat{\boldsymbol{\theta}} &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\left[\int_{-\infty}^{\infty} q(y)\log\big[q(y)\big]dy - \int_{-\infty}^{\infty} q(y)\log\big[Pr(y|\boldsymbol{\theta})\big]dy\right] \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\left[-\int_{-\infty}^{\infty} q(y)\log\big[Pr(y|\boldsymbol{\theta})\big]dy\right]\end{aligned}$$

SAPIENZA
UNIVERSITÀ DI ROMA

Why Cross Entropy?

$$q(y) = \frac{1}{I}\sum_{i=1}^{I}\delta[y-y_i]$$

- We want to minimize the KL divergence between q(y) and the real distribution Pr(y|θ):

$$\begin{aligned}
\hat{\boldsymbol{\theta}} &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\left[\int_{-\infty}^{\infty} q(y)\log\bigl[q(y)\bigr]dy - \int_{-\infty}^{\infty} q(y)\log\bigl[Pr(y|\boldsymbol{\theta})\bigr]dy\right] \\
&= \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\left[-\int_{-\infty}^{\infty} q(y)\log\bigl[Pr(y|\boldsymbol{\theta})\bigr]dy\right] \\
&= \underset{\theta}{\operatorname{argmin}}\left[-\int_{-\infty}^{\infty}\left(\frac{1}{I}\sum_{i=1}^{I}\delta[y-y_i]\right)\log\bigl[Pr(y|\boldsymbol{\theta})\bigr]dy\right]
\end{aligned}$$

$$q(y) = \frac{1}{I}\sum_{i=1}^{I}\delta[y - y_i]$$

Why Cross Entropy?

- We want to minimize the KL divergence between q(y) and the real distribution Pr(y|θ):

$$\begin{aligned}
\hat{\boldsymbol{\theta}} &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\left[\int_{-\infty}^{\infty} q(y)\log\big[q(y)\big]dy - \int_{-\infty}^{\infty} q(y)\log\big[Pr(y|\boldsymbol{\theta})\big]dy\right] \\
&= \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\left[-\int_{-\infty}^{\infty} q(y)\log\big[Pr(y|\boldsymbol{\theta})\big]dy\right] \\
&= \underset{\theta}{\operatorname{argmin}}\left[-\int_{-\infty}^{\infty}\left(\frac{1}{I}\sum_{i=1}^{I}\delta[y-y_i]\right)\log\big[Pr(y|\boldsymbol{\theta})\big]dy\right] \\
&= \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\left[-\frac{1}{I}\sum_{i=1}^{I}\log\big[Pr(y_i|\boldsymbol{\theta})\big]\right]
\end{aligned}$$

SAPIENZA
UNIVERSITÀ DI ROMA

$$q(y) = \frac{1}{I}\sum_{i=1}^{I}\delta[y - y_i]$$

Why Cross Entropy?

- We want to minimize the KL divergence between q(y) and the real distribution Pr(y|θ):

$$\begin{aligned}
\hat{\boldsymbol{\theta}} &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\left[\int_{-\infty}^{\infty} q(y)\log\big[q(y)\big]dy - \int_{-\infty}^{\infty} q(y)\log\big[Pr(y|\boldsymbol{\theta})\big]dy\right] \\
&= \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\left[-\int_{-\infty}^{\infty} q(y)\log\big[Pr(y|\boldsymbol{\theta})\big]dy\right] \\
&= \underset{\theta}{\operatorname{argmin}}\left[-\int_{-\infty}^{\infty}\left(\frac{1}{I}\sum_{i=1}^{I}\delta[y-y_i]\right)\log\big[Pr(y|\boldsymbol{\theta})\big]dy\right] \\
&= \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\left[-\frac{1}{I}\sum_{i=1}^{I}\log\big[Pr(y_i|\boldsymbol{\theta})\big]\right] \\
&= \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\left[-\sum_{i=1}^{I}\log\big[Pr(y_i|\boldsymbol{\theta})\big]\right]
\end{aligned}$$

SAPIENZA
UNIVERSITÀ DI ROMA

Why Cross Entropy?

$$q(y) = \frac{1}{I}\sum_{i=1}^{I}\delta[y - y_i]$$

- We want to minimize the KL divergence between q(y) and the real distribution Pr(y|θ):

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\left[-\sum_{i=1}^{I}\log\left[Pr(y_i|\boldsymbol{\theta})\right]\right]$$

- By substituting θ with f[$\mathbf{x}_i$,**ϕ**] computed by the ML model we get

$$= \underset{\boldsymbol{\phi}}{\operatorname{argmin}}\left[-\sum_{i=1}^{I}\log\left[Pr(y_i|\mathbf{f}[\mathbf{x}_i,\boldsymbol{\phi}])\right]\right]$$

- The **negative log-likelihood criterion** (from maximizing the data likelihood) **and the cross-entropy criterion** (from minimizing the distance between the model and empirical data distributions) **are equivalent**.

SAPIENZA
UNIVERSITÀ DI ROMA

Exercise

Problem 5.6 Consider building a model to predict the number of pedestrians $y \in \{0, 1, 2, \ldots\}$ that will pass a given point in the city in the next minute, based on data $\mathbf{x}$ that contains information about the time of day, the longitude and latitude, and the type of neighborhood. A suitable distribution for modeling counts is the Poisson distribution (figure 5.15). This has a single parameter $\lambda > 0$ called the *rate* that represents the mean of the distribution. The distribution has probability density function:

$$Pr(y = k) = \frac{\lambda^k e^{-\lambda}}{k!}. \tag{5.36}$$

Design a loss function for this model assuming we have access to I training pairs $\{\mathbf{x}_i, y_i\}$.

Solution

- Given the probability density function of the Poisson distribution: $P_r(y = k) = \frac{\lambda^k e^{-\lambda}}{k!}$

Solution

- Given the probability density function of the Poisson distribution: $P_r(y = k) = \frac{\lambda^k e^{-\lambda}}{k!}$
- The log-likelihood for a single data point (x, y) is: $\log P_r(y = k) = k \log \lambda - \lambda - \log(k!)$

Solution

- Given the probability density function of the Poisson distribution: $P_r(y=k)=\frac{\lambda^k e^{-\lambda}}{k!}$
- The log-likelihood for a single data point (x,y) is: $\log P_r(y=k)=k\log\lambda-\lambda-\log(k!)$
- For I training pairs $\{x_i, y_i\}$, the total log-likelihood is the sum of the log-likelihoods for each data point:

$$L(\lambda)=\sum_{i=1}^{I}\left(y_i\log\lambda-\lambda-\log(y_i!)\right)$$

Solution

- Given the probability density function of the Poisson distribution: $P_r(y = k) = \frac{\lambda^k e^{-\lambda}}{k!}$
- The log-likelihood for a single data point (x, y) is: $\log P_r(y = k) = k \log \lambda - \lambda - \log(k!)$
- For I training pairs $\{x_i, y_i\}$, the total log-likelihood is the sum of the log-likelihoods for each data point:

$$L(\lambda) = \sum_{i=1}^{I} (y_i \log \lambda - \lambda - \log(y_i!))$$

- To turn this into a loss function, we take the negative of the log-likelihood:
  - $\text{Loss}(\lambda) = -L(\lambda)$
  - $\text{Loss}(\lambda) = -\sum_{i=1}^{I} (y_i \log \lambda_i - \lambda_i - \log(y_i!))$

Solution

- Given the probability density function of the Poisson distribution: $P_r(y = k) = \frac{\lambda^k e^{-\lambda}}{k!}$
- The log-likelihood for a single data point (x, y) is: $\log P_r(y = k) = k \log \lambda - \lambda - \log(k!)$
- For I training pairs $\{x_i, y_i\}$, the total log-likelihood is the sum of the log-likelihoods for each data point:

$$L(\lambda) = \sum_{i=1}^{I} \left(y_i \log \lambda - \lambda - \log(y_i!)\right)$$

- To turn this into a loss function, we take the negative of the log-likelihood:
  - $\text{Loss}(\lambda) = -L(\lambda)$
  - $\text{Loss}(\lambda) = -\sum_{i=1}^{I} \left(y_i \log \lambda_i - \lambda_i - \log(y_i!)\right)$
- In practice, the model will predict a different rate λ_i for each data point x_i based on the input features (time of day, longitude, latitude, type of neighborhood). So, the loss function becomes:

$$\text{Loss}(\lambda) = -\sum_{i=1}^{I} \left(y_i \log \lambda - \lambda - \log(y_i!)\right)$$

- The term log(y_i!) can be ignored during optimization since it doesn't depend on the model's parameters. It's a constant with respect to λ

SAPIENZA
UNIVERSITÀ DI ROMA

Deep Learning

End of Lecture

03 - Model Training. Loss function

SAPIENZA
UNIVERSITÀ DI ROMA

Fabrizio Silvestri