Munich Center for Machine Learning

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

From Statistical Modeling to Deep Learning

David Rügamer, LMU Munich and MCML

Workshop @ TU Dortmund, Oct 12, 2023

Data Science Group @ LMU Munich

https://datascience-lmu.github.io/website/

My Research:

38% Statistics

- Post-Selection Inference
- Model Selection
- Functional Data Analysis
- Neural Statistical Models

30% Deep Learning

- Semi-Structured Models
- Generative Models
- Deep Probabilistic Models
- Representation Learning

18% Applications

- Biomechanics
- Medical Research
- Psychology

14% Machine Learning

- Boosting
- AutoML

More than 50% at the intersection

t1p.de/dl_handout | t1p.de/dl_notebook

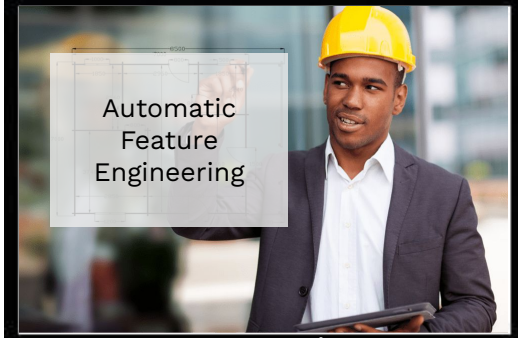
LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

mcml

This Workshop

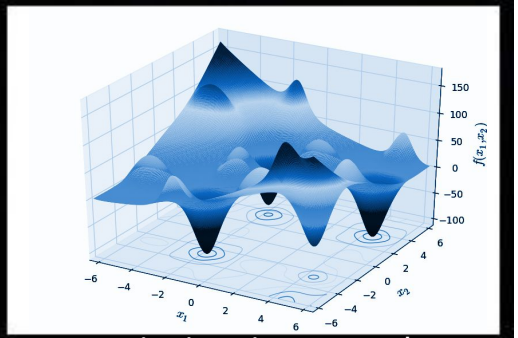
What’s Deep Learning (good for)?

What's Deep Learning (good for)?

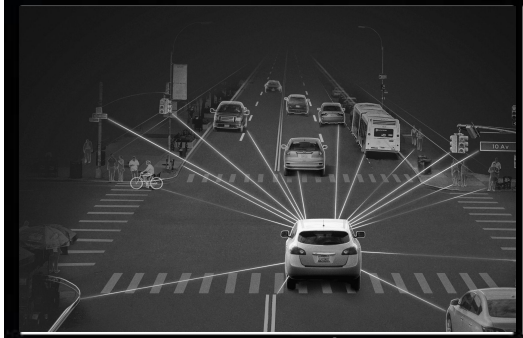


Pragmatist

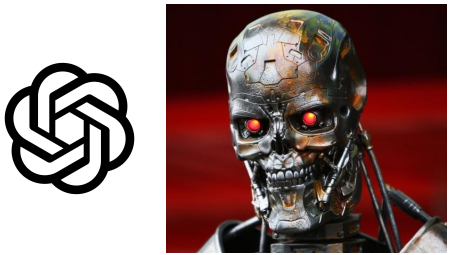
Enthusiast



Optimization Oracle



AI Researcher



General Public

?

Statistical Modeller

Prerequisites

- Linear Models
- Statistical Modeling (basics)
- Machine Learning (basics)
- R

Learning Goals

- When to use Deep Learning / Neural Networks
- Foundations of Neural Networks
- Neural Networks from a Statistical Point of View
- Semi-Structured Neural Networks
- Practical Implementation of Neural Networks

Learning Goals

- When to use Deep Learning / Neural Networks
- Foundations of Neural Networks
- Neural Networks from a Statistical Point of View
- Semi-Structured Neural Networks
- Practical Implementation of Neural Networks

Learning Goals

- When to use Deep Learning / Neural Networks
- Foundations of Neural Networks
- Neural Networks from a Statistical Point of View
- Semi-Structured Neural Networks
- Practical Implementation of Neural Networks

Google Colab Link

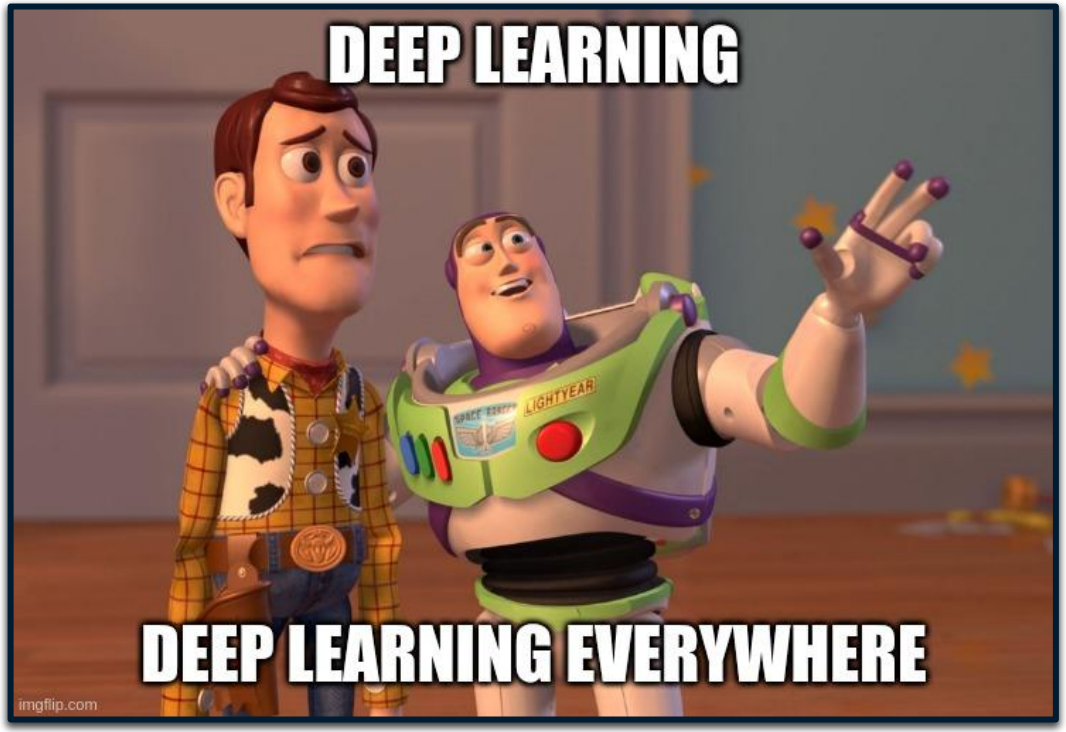
Learning Goals

- When to use Deep Learning / Neural Networks
- Foundations of Neural Networks
- Neural Networks from a Statistical Point of View
- Semi-Structured Neural Networks
- Practical Implementation of Neural Networks

Google Colab Link

copy notebook and run
the first two chunks

Why (not) Deep Learning


DEEP LEARNING
DEEP LEARNING EVERYWHERE
LIGHTYEAR
imgflip.com

Deep Learning “Taxonomy”

Deep Learning

= Learning with neural networks that are deep

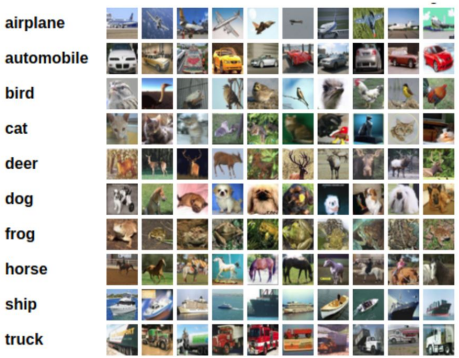
Deep Learning “Taxonomy”

Deep Learning

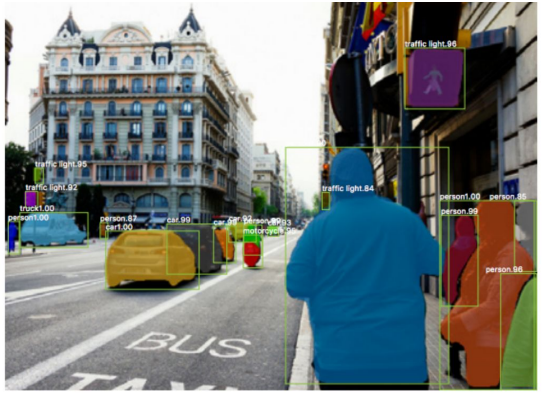
= Learning with neural networks that are deep

Deep ⇒ different definitions, most common:
… at least one hidden layer
… at least two hidden layers

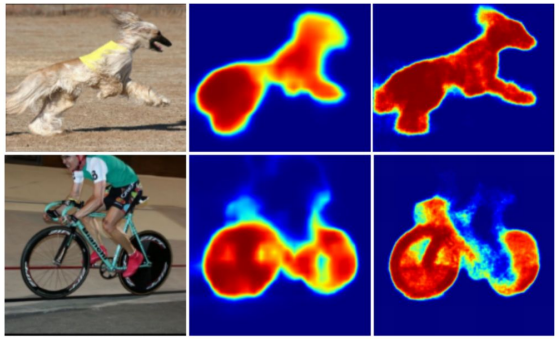
Possible Use-Cases



Credit: Alex Krizhevsky (2009)



Credit: Kaiming He (2017)



Credit: Hyeonwoo Noh (2015)

Positive

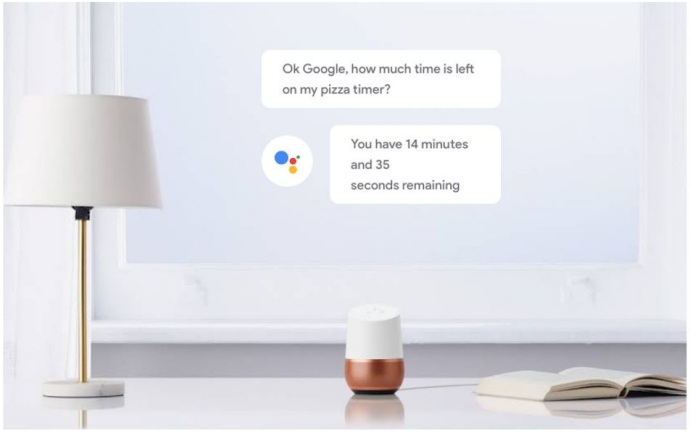
Great job! Your customer support is fantastic.

Neutral

Not bad, but it should be improved in the future.

Negative

The worst customer service I have ever seen.



t1p.de/ai_handout | t1p.de/ai_notebook

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

mcml

Why **not** Deep Learning

Why **not** Deep Learning

- While Neural Networks (NNs) can match performance of tree-based methods [1,2]

Why **not** Deep Learning

- While Neural Networks (NNs) can match performance of tree-based methods [1,2]
  - NNs require careful training
  - Tree-based methods provide the better inductive bias + regularization off-the-shelf [3]

Why **not** Deep Learning

- While Neural Networks (NNs) can match performance of tree-based methods [1,2]
  - NNs require careful training
  - Tree-based methods provide the better inductive bias + regularization off-the-shelf [3]
  - ➢ not (necessarily) the best choice for predictive performance on tabular data

Why **not** Deep Learning

- While Neural Networks (NNs) can match performance of tree-based methods [1,2]
  - NNs require careful training
  - Tree-based methods provide the better inductive bias + regularization off-the-shelf [3]
  - ➢ not (necessarily) the best choice for predictive performance on tabular data

Why **not** Deep Learning

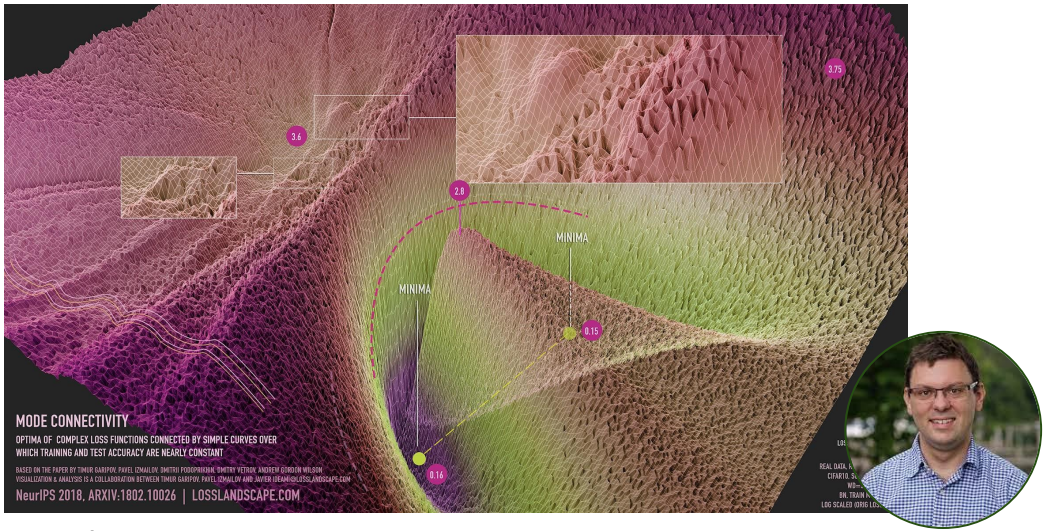
- While Neural Networks (NNs) can match performance of tree-based methods [1,2]
  - NNs require careful training
  - Tree-based methods provide the better inductive bias + regularization off-the-shelf [3]
  - ➢ not (necessarily) the best choice for predictive performance on tabular data

Why **not** Deep Learning

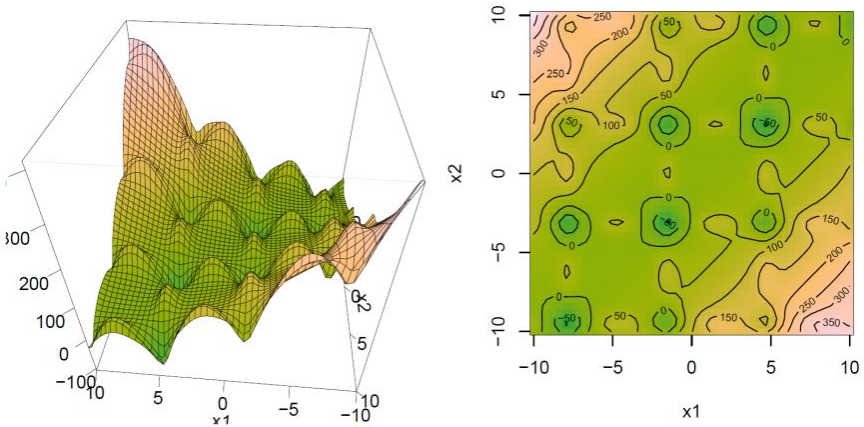
- Even for simpler neural networks,
  - optimization is still not perfectly understood
  - no well-understood uncertainty quantification

Why **not** Deep Learning

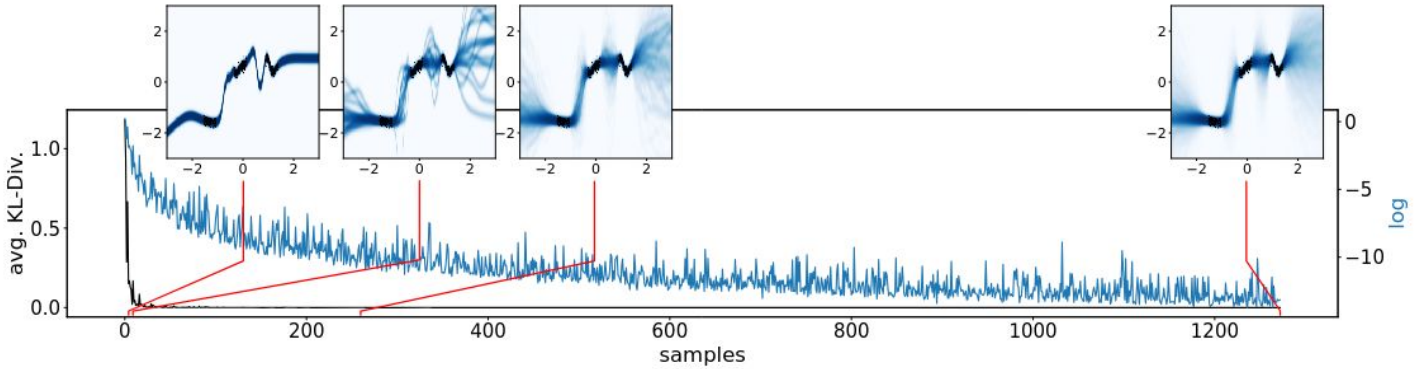
- Even for simpler neural networks,
  - optimization is still not perfectly understood
  - no well-understood uncertainty quantification



Why **not** Deep Learning



[7] **Best Paper @ ECML-PKDD 2023**



Why ~~not~~ Deep Learning

Flexibility & Automation

Why ~~not~~ Deep Learning

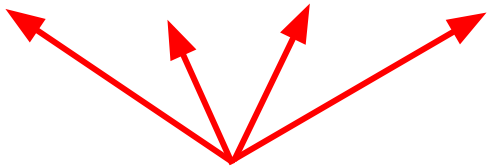
Flexibility & Automation

- Data modalities (Text, Images, Audio, Video, ...)
- Modularity
- Scalability
- Software

Why ~~not~~ Deep Learning

Flexibility & Automation

- Data modalities (Text, Images, Audio, Video, ...)
- Modularity
- Scalability
- Software



Often no way around Automation

Why ~~not~~ Deep Learning

Flexibility & Automation

- Data modalities (Text, Images, Audio, Video, ...)
- Modularity
- Scalability
- Software

Generative Models?

Why ~~not~~ Deep Learning

Flexibility & Automation

- Data modalities (Text, Images, Audio, Video, ...)
- Modularity
- Scalability
- Software

Generative Models?

Why ~~not~~ Deep Learning

Flexibility & Automation

- Data modalities (Text, Images, Audio, Video, ...)
- Modularity
- Scalability
- Software

Generative Models?

(a) *"Large pleural effusion is in the right lower lung. A pacemaker is in the left upper chest."*

(b) *"Acute cardiomegaly."*

(c) *"Prominent left-sided atelectasis."*

(d) *"Mass filling is in the upper zone of the right lung."*

(e) *"Pneumonia is in the right lung."*

t1p.de/dl_handout | t1p.de/dl_notebook

[5]

Why ~~not~~ Deep Learning

Flexibility & Automation

- Data modalities (Text, Images, Audio, Video, ...)
- Modularity
- Scalability
- Software

Generative Models?
Yes, but

Why ~~not~~ Deep Learning

Flexibility & Automation

- Data modalities (Text, Images, Audio, Video, …)
- Modularity
- Scalability
- Software

Generative Models?
Yes, but

- not the focus of today's session
- requires enormous amounts of data

Why ~~not~~ Deep Learning

Flexibility & Automation

- Data modalities (Text, Images, Audio, Video, ...)
- Modularity
- Scalability
- Software

- Optimization

Why ~~not~~ Deep Learning

Flexibility & Automation

- Data modalities (Text, Images, Audio, Video, ...)
- Modularity
- Scalability
- Software

- Optimization
  - ➢ Implicit Regularization ... leads to
  - ➢ ... Benign Overfitting [7]
  - ➢ ... Better test error in linear models than Lasso/Ridge

Why ~~not~~ Deep Learning

Flexibility & Automation

- Data modalities
- Modularity
- Scalability
- Software

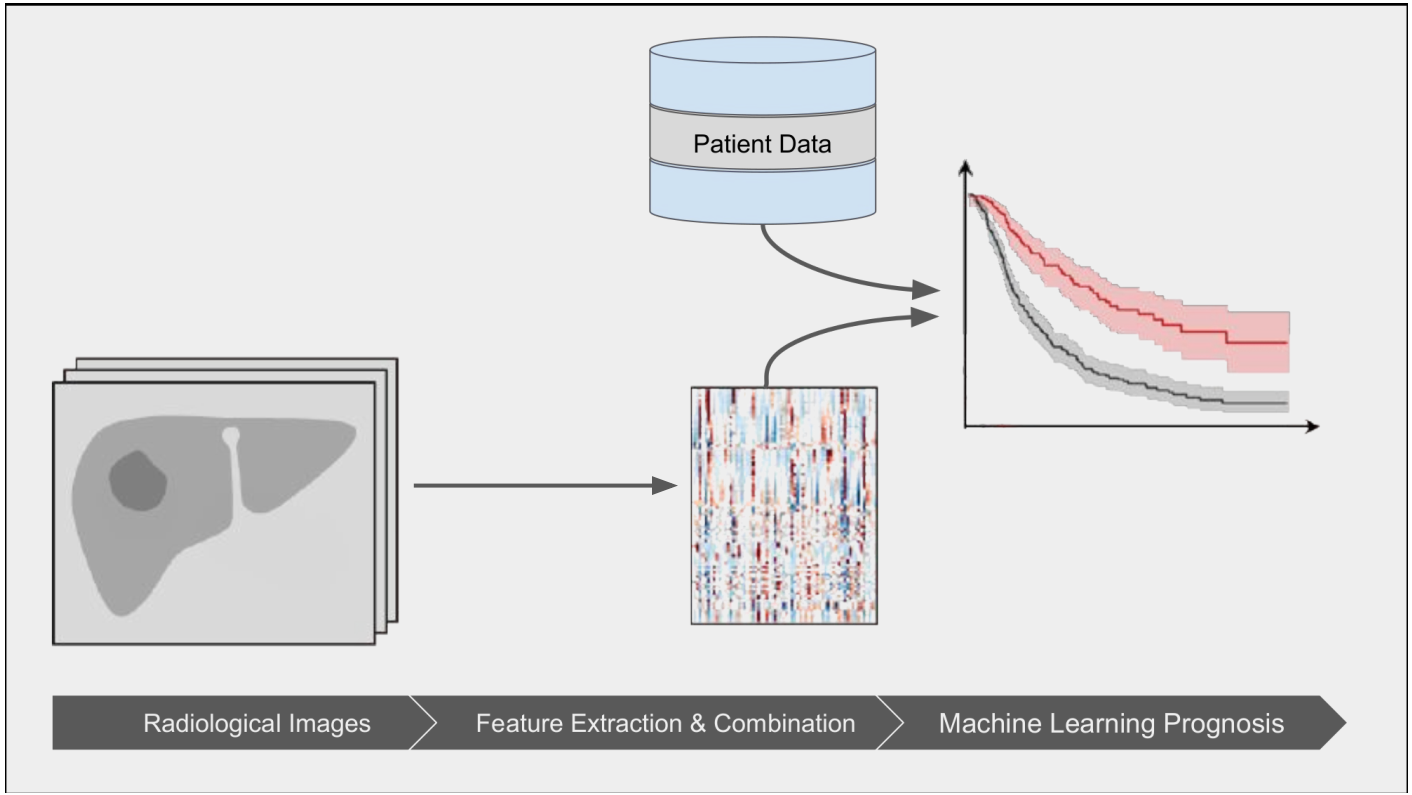
Why ~~not~~ Deep Learning

Flexibility & Automation

- Data modalities
- Modularity
- Scalability
- Software

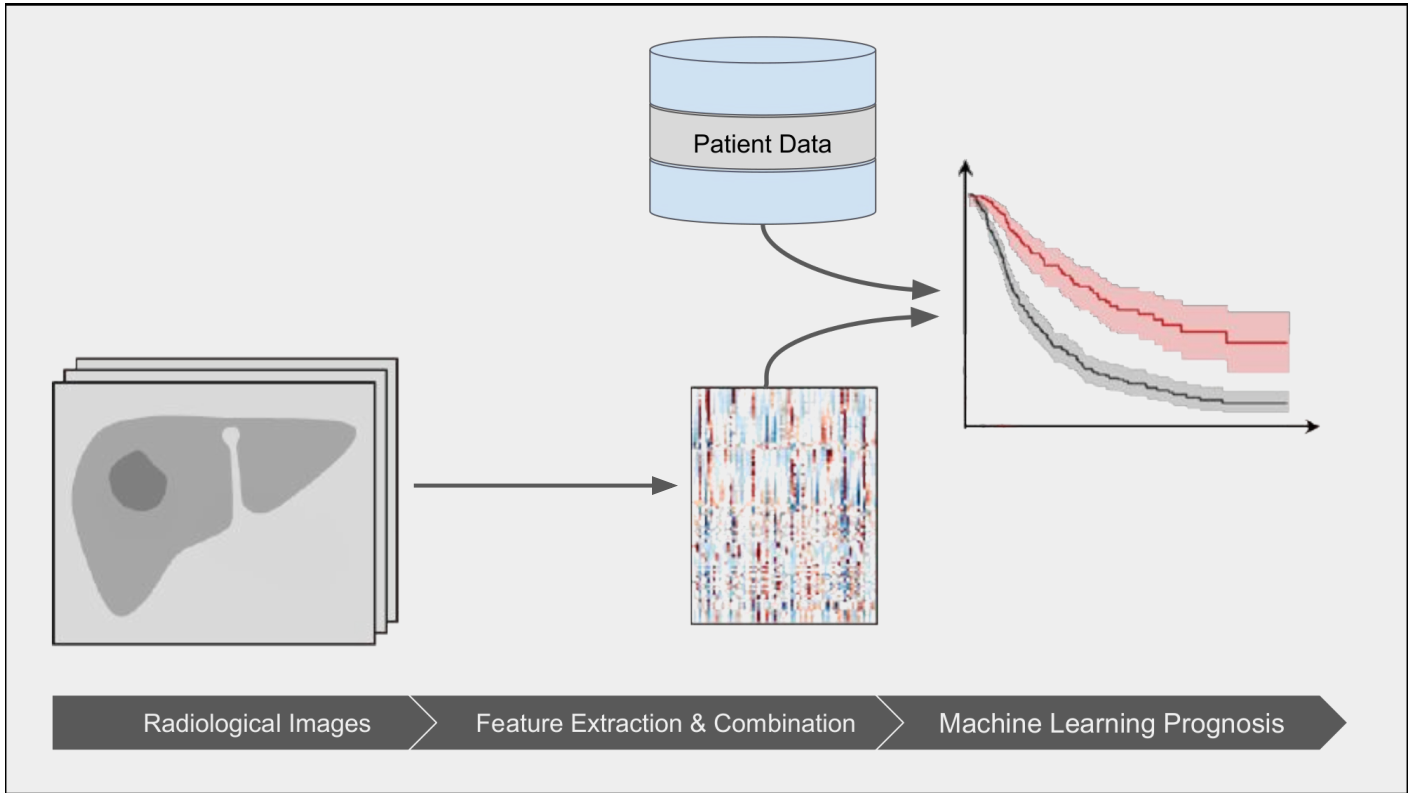
⇒ Deep Learning and Neural Networks — for Statistical Modeling

Motivating Example: Multimodal Data Sets



Prof. Ingrisch (LMU)
Clinical Data Science

Motivating Example: Multimodal Data Sets

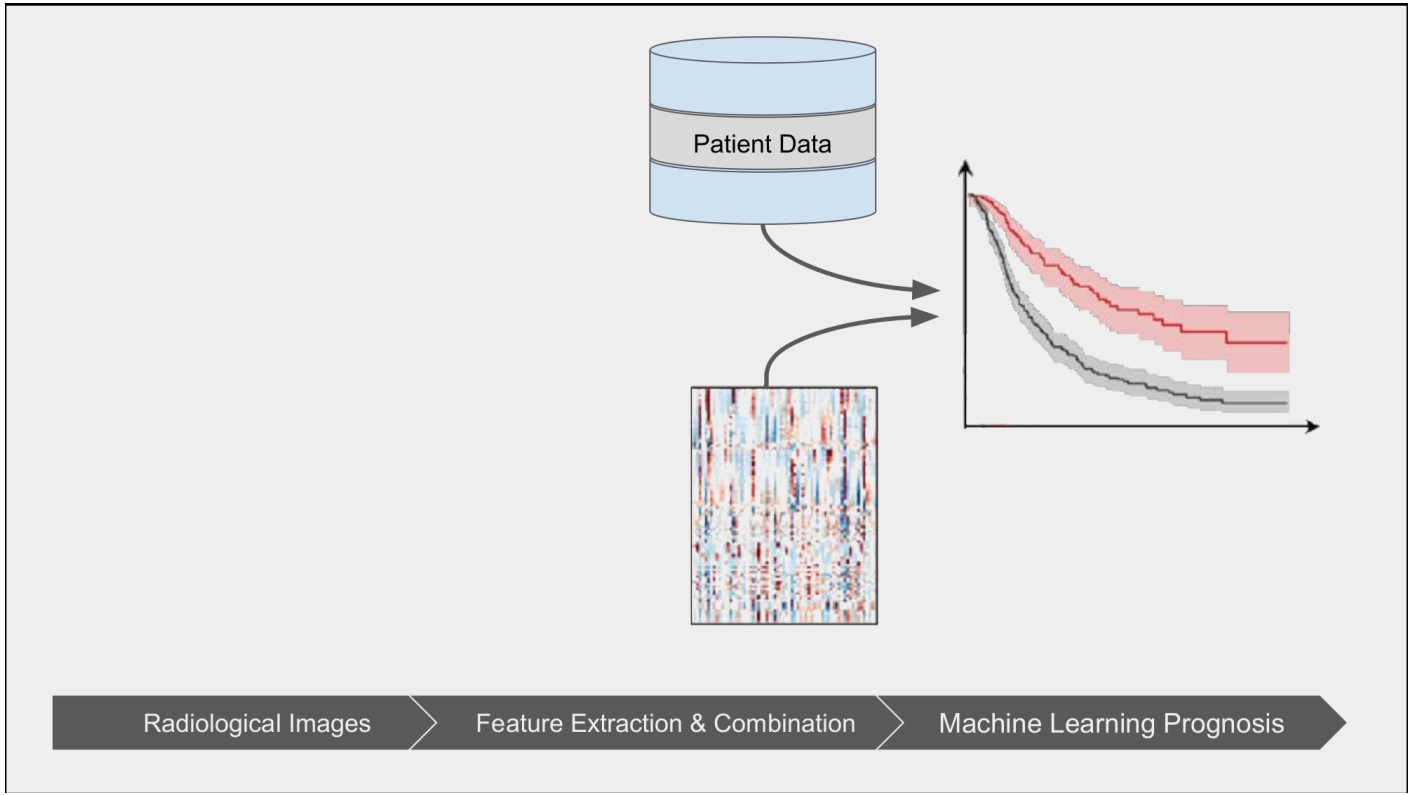


Motivating Example: Multimodal Data Sets

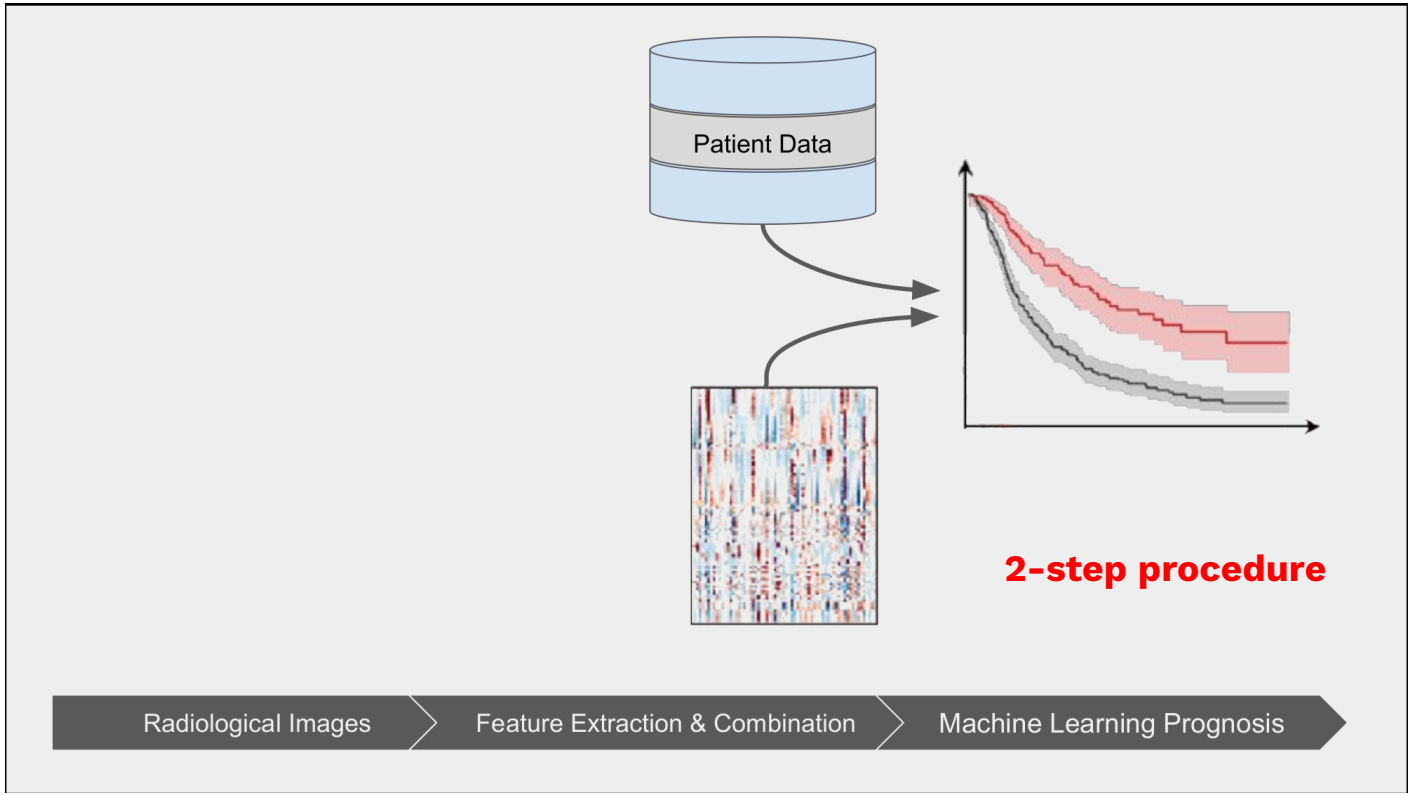
Radiomics

Radiological Images → Feature Extraction & Combination → Machine Learning Prognosis

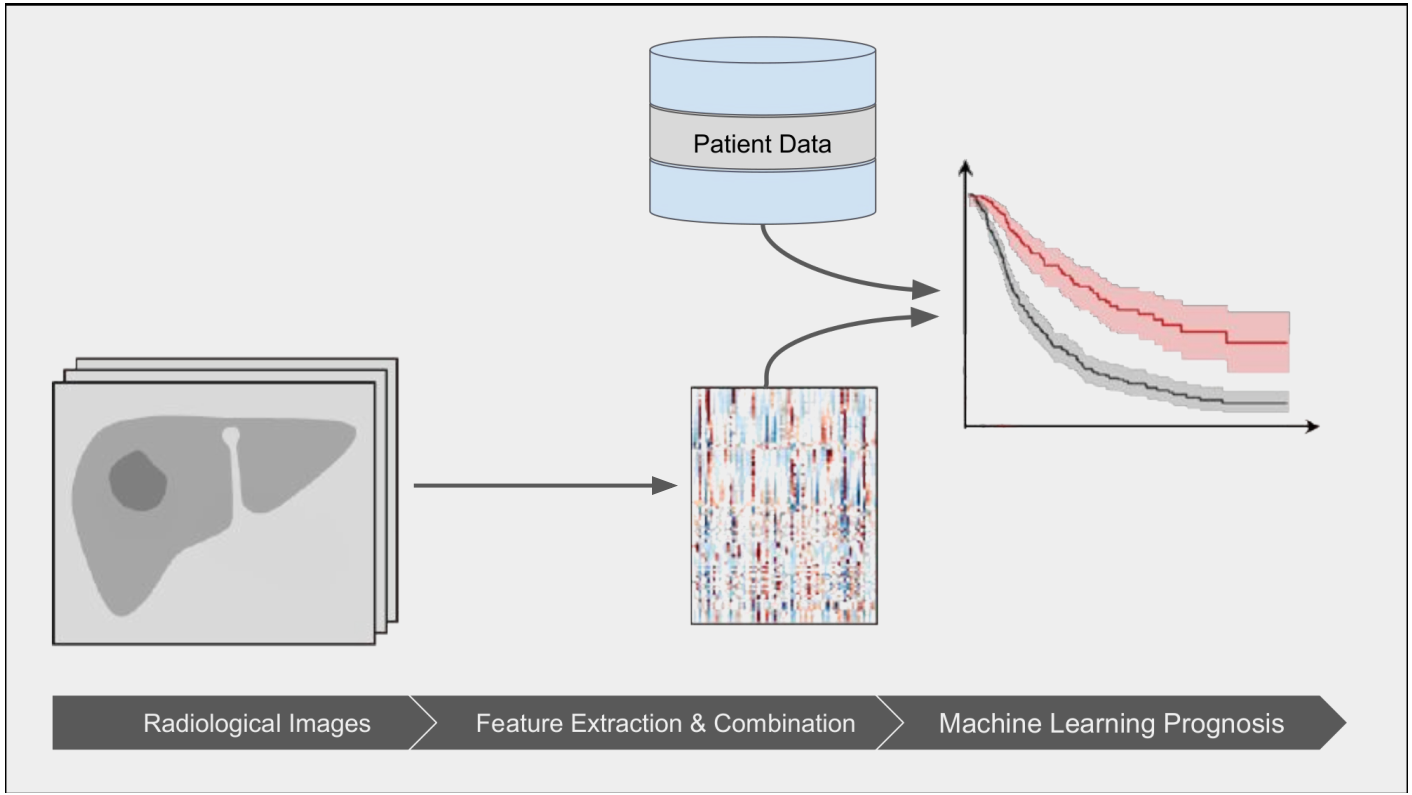
Motivating Example: Multimodal Data Sets



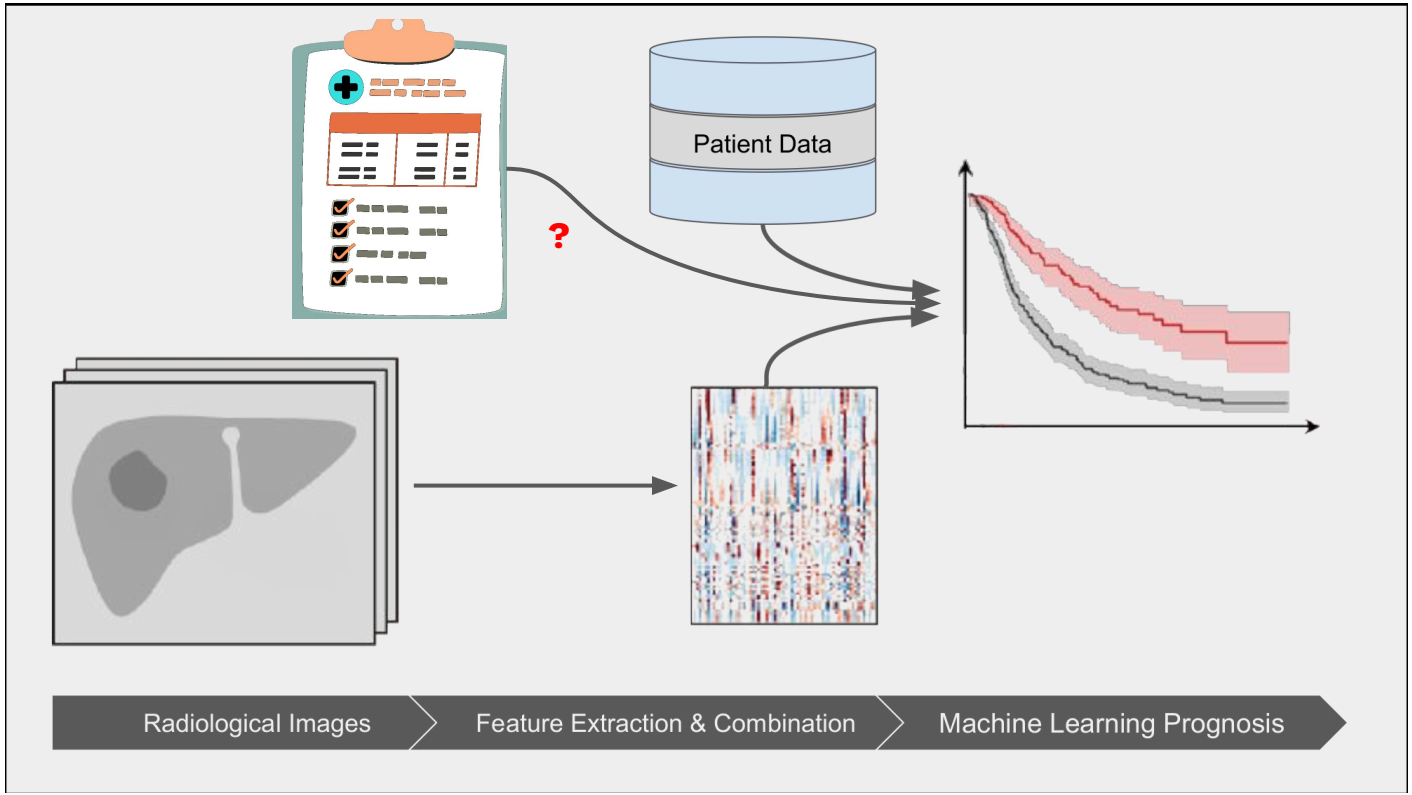
Motivating Example: Multimodal Data Sets



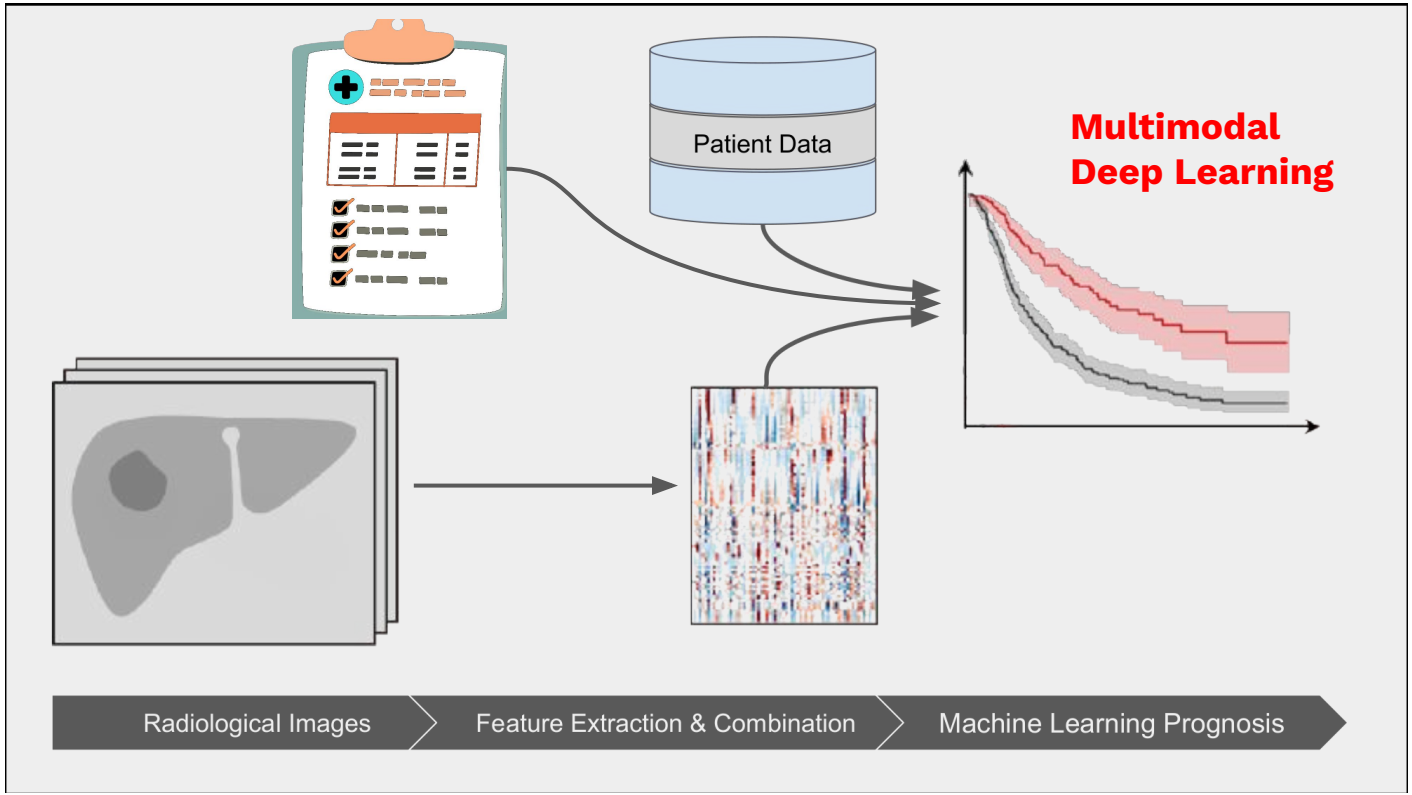
Motivating Example: Multimodal Data Sets



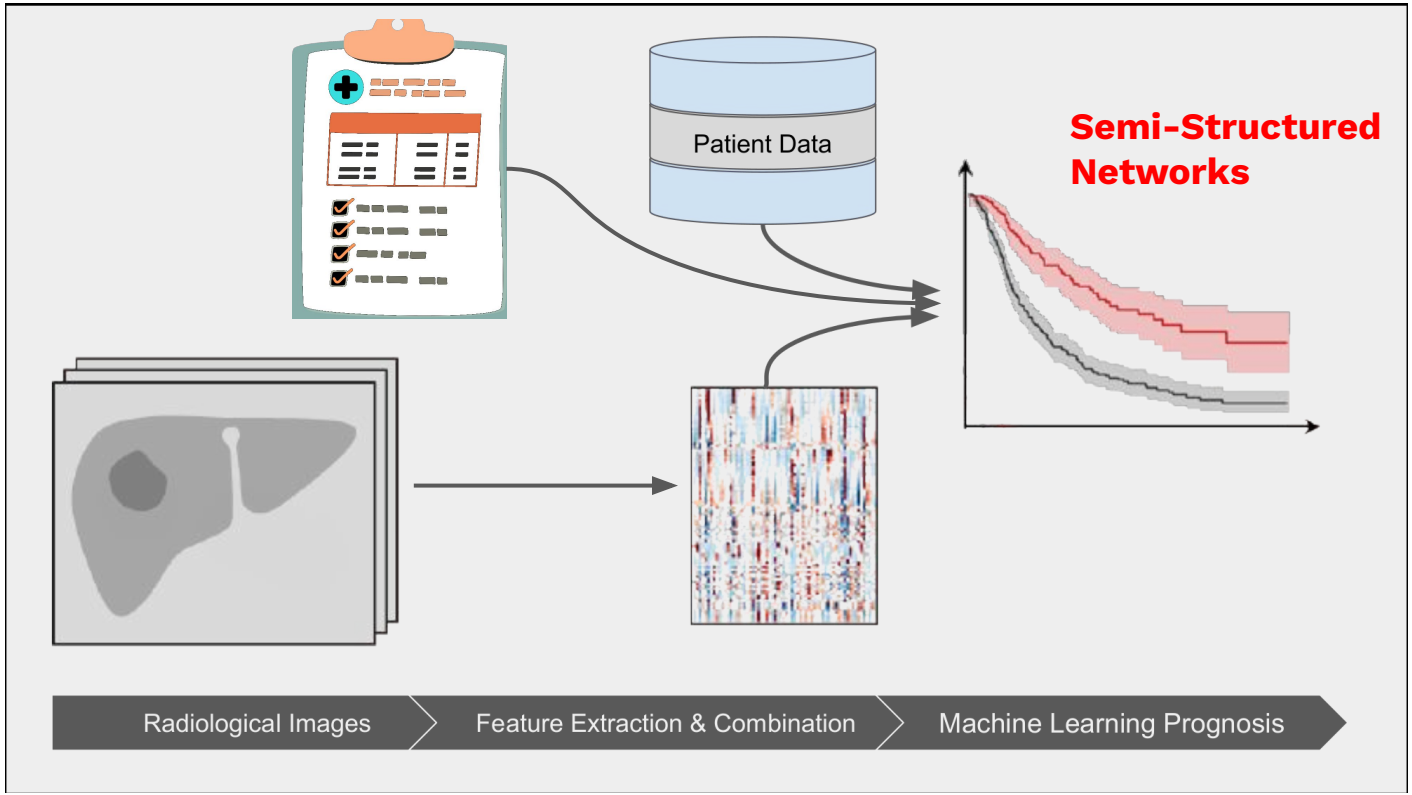
Motivating Example: Multimodal Data Sets



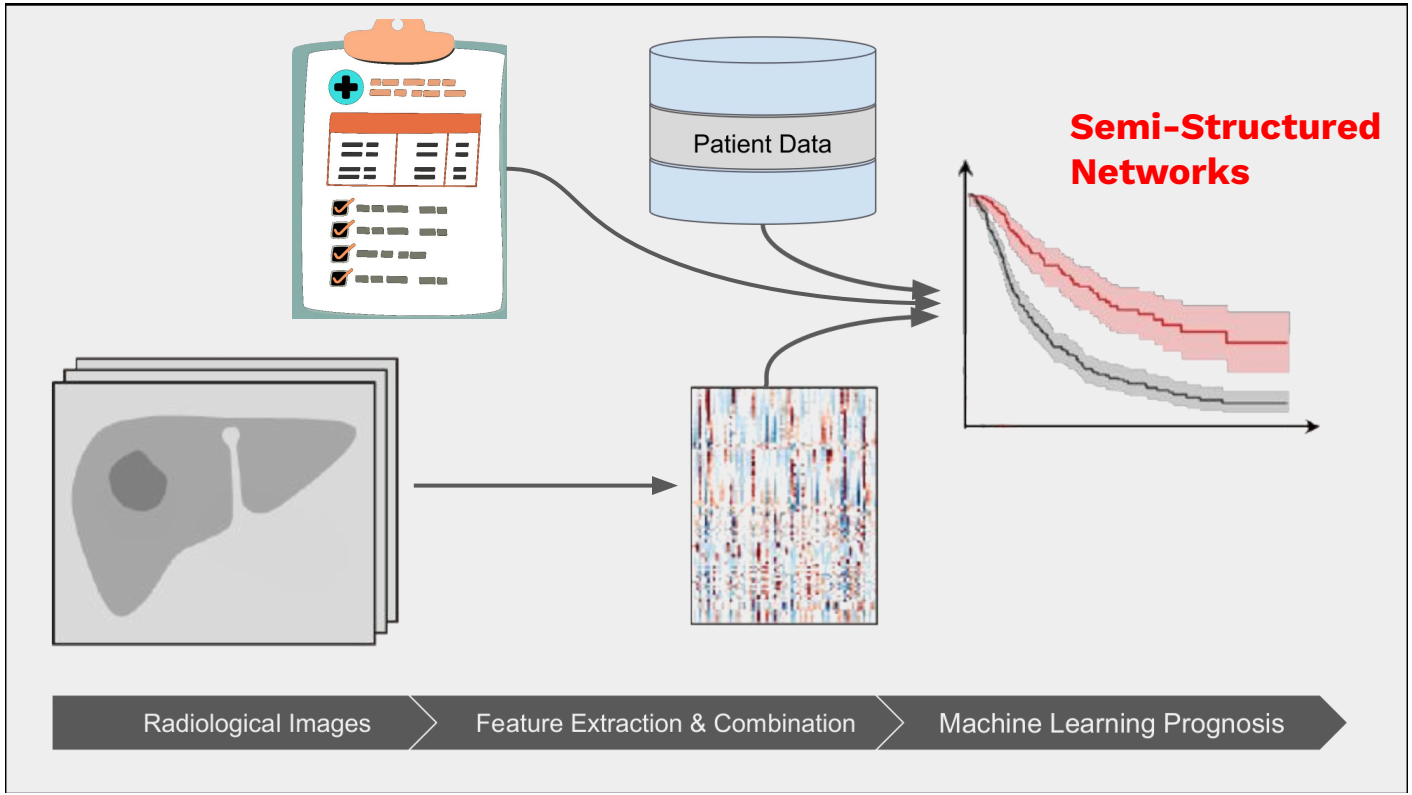
Motivating Example: Multimodal Data Sets



Motivating Example: Multimodal Data Sets



Motivating Example: Multimodal Data Sets



Inner Workings and Optimization of Neural Networks

Components of Neural Networks

- Architecture
- Loss
- Optimizer / Training Schedul(er)s

Components of Neural Networks

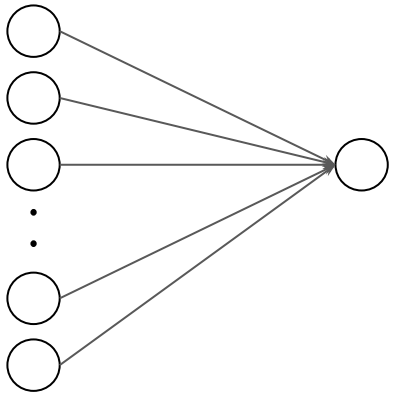
- Architecture
- Loss
- Optimizer / Training Routine

Components of Neural Networks

- Architecture
  - Often stacked in layers
  - Different layers for different purposes

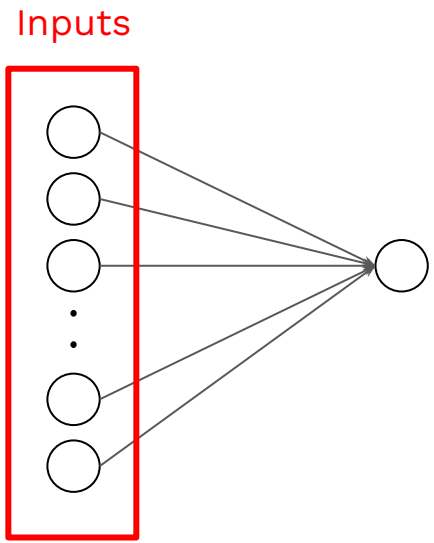
Components of Neural Networks

- Architecture
  - Often stacked in layers
  - Different layers for different purposes



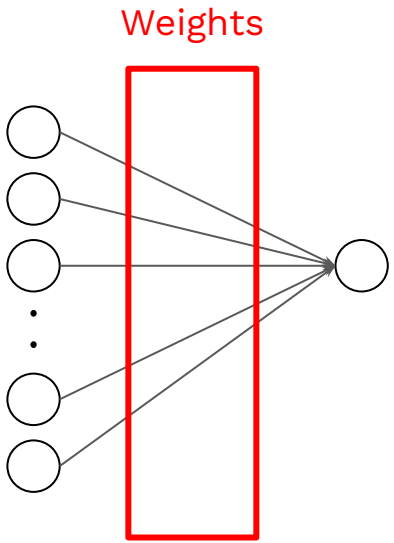
Components of Neural Networks

- Architecture
  - Often stacked in layers
  - Different layers for different purposes



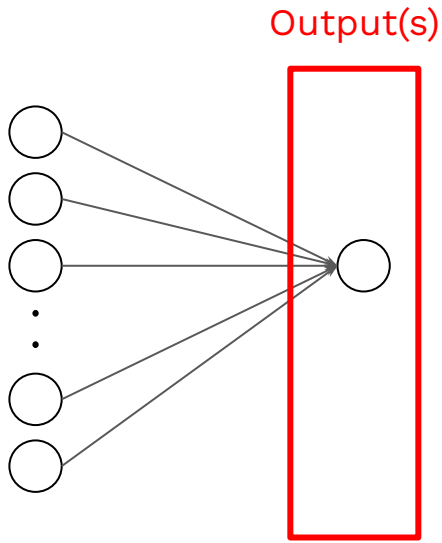
Components of Neural Networks

- Architecture
  - Often stacked in layers
  - Different layers for different purposes



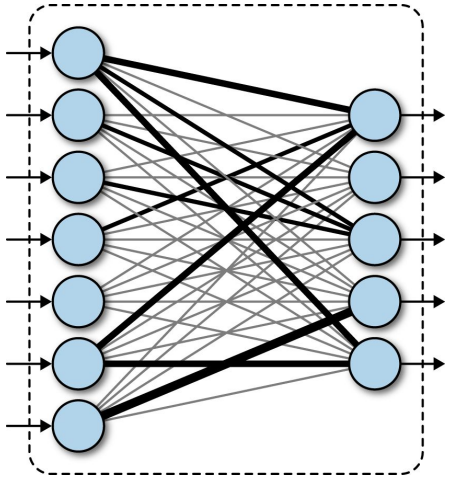
Components of Neural Networks

- Architecture
  - Often stacked in layers
  - Different layers for different purposes



Components of Neural Networks

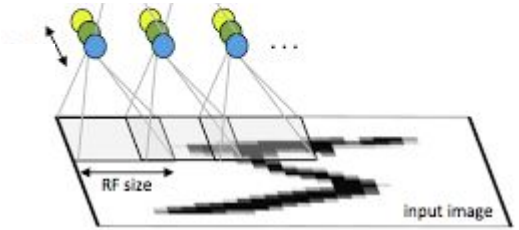
- Architecture
  - Often stacked in layers
  - Different layers for different purposes
    - → Fully-connected layers to learn interactions



Source: oreilly.com

Components of Neural Networks

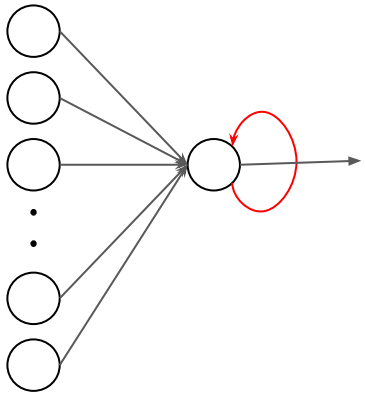
- Architecture
  - Often stacked in layers
  - Different layers for different purposes
    - → Fully-connected layers to learn interactions
    - → Convolutional layers for image processing



Source: stanford.edu

Components of Neural Networks

- Architecture
  - Often stacked in layers
  - Different layers for different purposes
    - → Fully-connected layers to learn interactions
    - → Convolutional layers for image processing
    - → Recurrent layers for sequential data



Components of Neural Networks

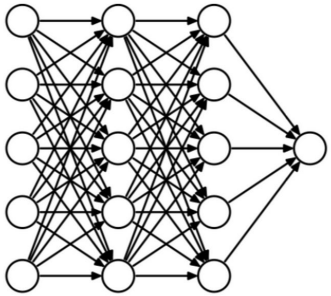
- Architecture
  - Often stacked in layers
  - Different layers for different purposes
    - → Fully-connected layers to learn interactions
    - → Convolutional layers for image processing
    - → Recurrent layers for sequential data
    - → Attention layers for everything :)

Components of Neural Networks

- Architecture
  - Often stacked in layers
  - Different layers for different purposes
    - → Fully-connected layers to learn interactions
    - → Convolutional layers for image processing
    - → Recurrent layers for sequential data
    - → Attention layers for everything :)
    - → Further layers for
      - regularization (e.g. dropout)
      - normalization
      - reshaping

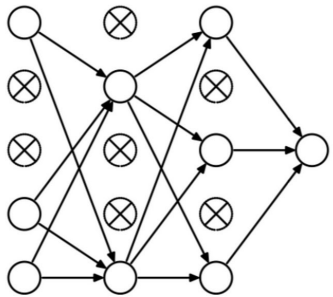
Components of Neural Networks

- Architecture
  - Often stacked in layers
  - Different layers for different purposes
    - → Fully-connected layers to learn interactions
    - → Convolutional layers for image processing
    - → Recurrent layers for sequential data
    - → Attention layers for everything :)
    - → Further layers for
      - regularization (e.g. dropout)
      - normalization
      - reshaping



Components of Neural Networks

- Architecture
  - Often stacked in layers
  - Different layers for different purposes
    - → Fully-connected layers to learn interactions
    - → Convolutional layers for image processing
    - → Recurrent layers for sequential data
    - → Attention layers for everything :)
    - → Further layers for
      - regularization (e.g. dropout)
      - normalization
      - reshaping



Components of Neural Networks

- Architecture
- Loss
- Optimizer / Training Schedul(er)s

Components of Neural Networks

- Architecture
- Loss
- Optimizer / Training Schedul(er)s

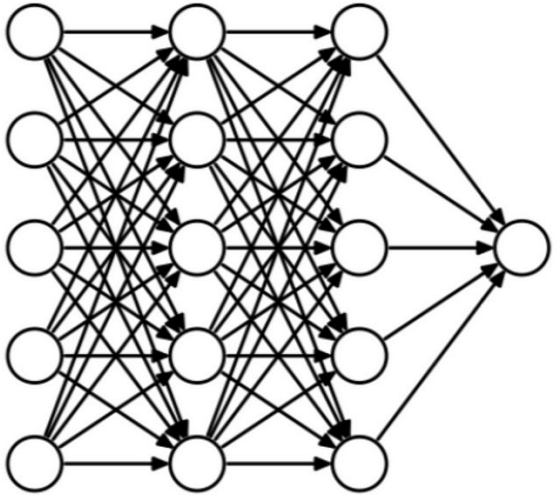
Components of Neural Networks

- Architecture
- Loss
  - The typical decision process:
    - ➢ “Regression” ⇒ L2-loss
    - ➢ “Classification” ⇒ Binary Cross Entropy / Log-loss
    - ➢ “Multiclass” ⇒ Cross Entropy

Components of Neural Networks

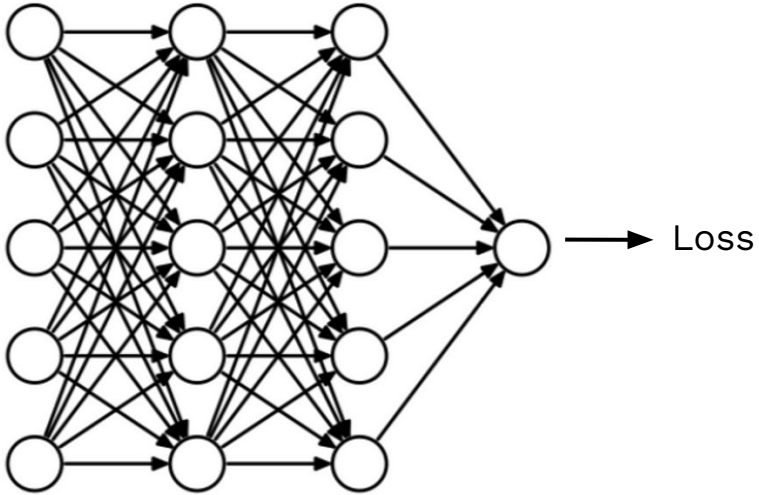
- Architecture
- Loss
  - The typical decision process:
    - ➢ “Regression” ⇒ L2-loss
    - ➢ “Classification” ⇒ Binary Cross Entropy / Log-loss
    - ➢ “Multiclass” ⇒ Cross Entropy
  - Any differentiable loss function is possible
    - ➢ DL Software works with **Auto Differentiation**
      ⇒ No need to derive anything by hand
      ⇒ Extreme flexibility

Components of Neural Networks



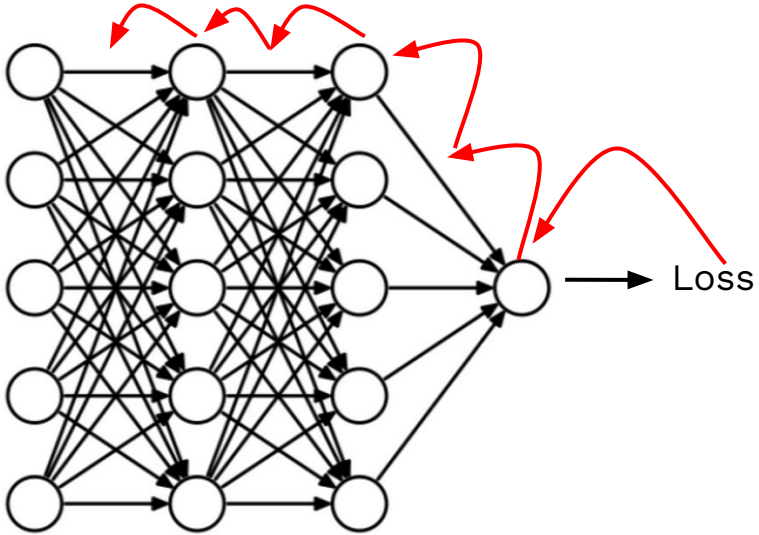
- Architecture
- Loss
  - The typical decision process:
    - ➢ “Regression” ⇒ L2-loss
    - ➢ “Classification” ⇒ Binary Cross
    - ➢ “Multiclass” ⇒ Cross Entropy
  - Any differentiable loss function is possible
    - ➢ DL Software works with **Auto Differentiation**
      ⇒ No need to derive anything by hand
      ⇒ Extreme flexibility

Components of Neural Networks



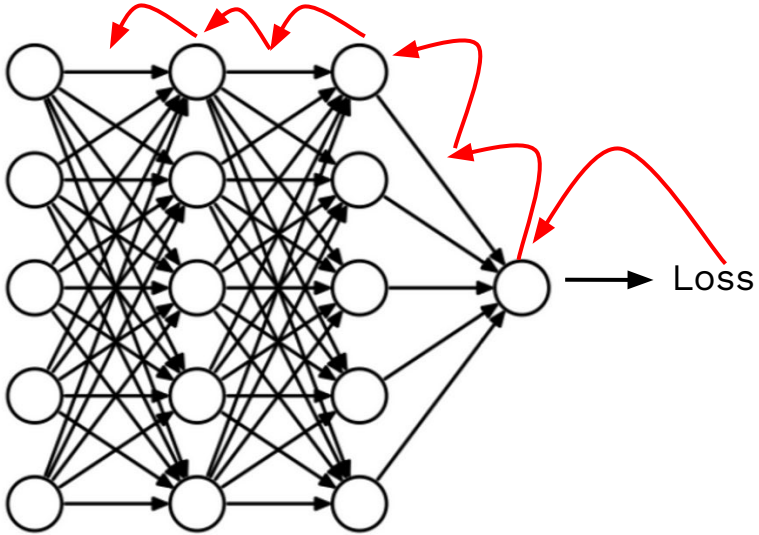
- Architecture
- Loss
  - The typical decision process:
    - ➢ “Regression” ⇒ L2-loss
    - ➢ “Classification” ⇒ Binary Cross
    - ➢ “Multiclass” ⇒ Cross Entropy
  - Any differentiable loss function is possible
    - ➢ DL Software works with **Auto Differentiation**
      ⇒ No need to derive anything by hand
      ⇒ Extreme flexibility

Components of Neural Networks



- Architecture
- Loss
  - The typical decision process:
    - ➢ “Regression” ⇒ L2-loss
    - ➢ “Classification” ⇒ Binary Cross
    - ➢ “Multiclass” ⇒ Cross Entropy
  - Any differentiable loss function is possible
    - ➢ DL Software works with **Auto Differentiation**
      ⇒ No need to derive anything by hand
      ⇒ Extreme flexibility

Components of Neural Networks



- Architecture
- Loss
  - The typical decision process:
    - ➢ “Regression” ⇒ L2-loss
    - ➢ “Classification” ⇒ Binary Cross
    - ➢ “Multiclass” ⇒ Cross Entropy
  - Any differentiable loss function is possible
    - ➢ DL Software works with **Auto Differentiation**
      ⇒ No need to derive anything by hand
      ⇒ Extreme flexibility

TensorFlow, PyTorch, JAX, MXNET

Components of Neural Networks

- Architecture
- Loss
  - The typical decision process:
    - ➢ “Regression” ⇒ L2-loss
    - ➢ “Classification” ⇒ Binary Cross Entropy / Log-loss
    - ➢ “Multiclass” ⇒ Cross Entropy
  - Any differentiable loss function is possible
    - ➢ DL Software works with **Auto Differentiation**
      ⇒ No need to derive anything by hand
      ⇒ Extreme flexibility

Components of Neural Networks

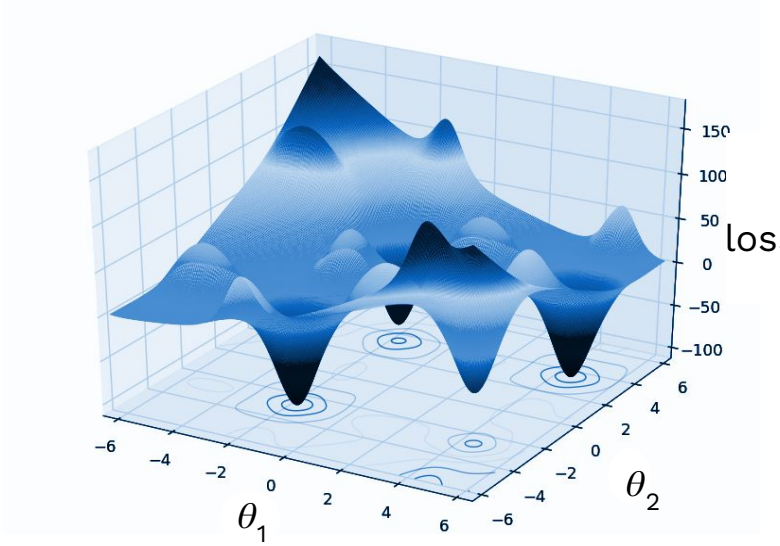
- Architecture
- Loss
- Optimizer / Training Schedul(er)s

Components of Neural Networks

- Architecture
- Loss
- Optimizer / Training Schedul(er)s

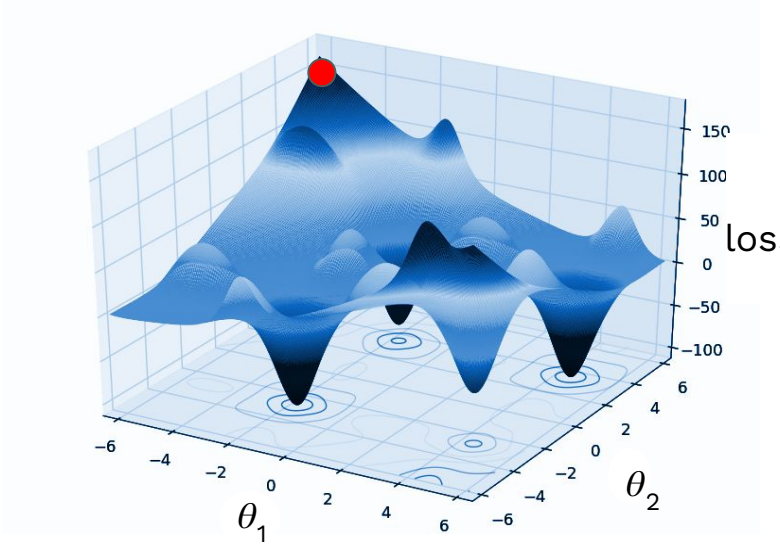
Components of Neural Networks

- Architecture
- Loss
- Optimizer / Training Schedul(er)s



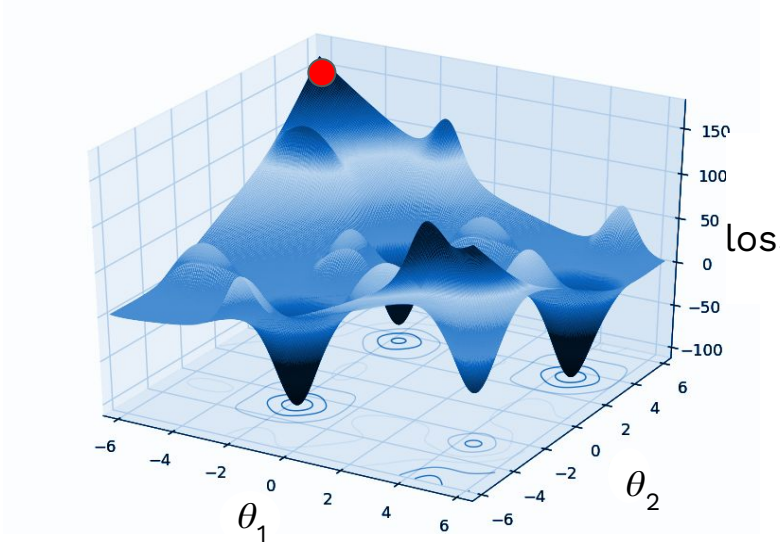
Components of Neural Networks

- Architecture
- Loss
- Optimizer / Training Schedul(er)s



Components of Neural Networks

- Architecture
- Loss
- Optimizer / Training Schedul(er)s

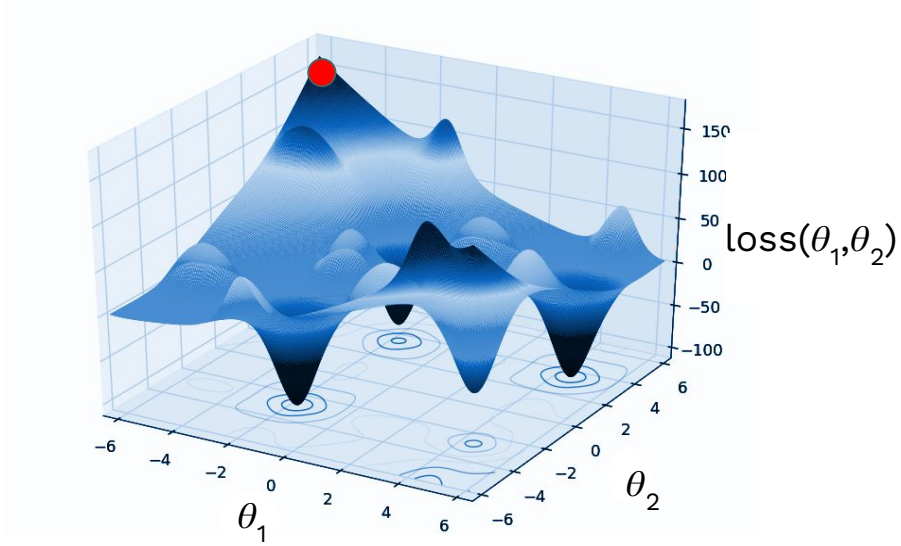


Most common variants are some form of gradient descent:

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^{t} + \boldsymbol{\nu} \cdot \nabla_{\boldsymbol{\theta}} \text{loss}$$

Components of Neural Networks

- Architecture
- Loss
- Optimizer / Training Schedul(er)s



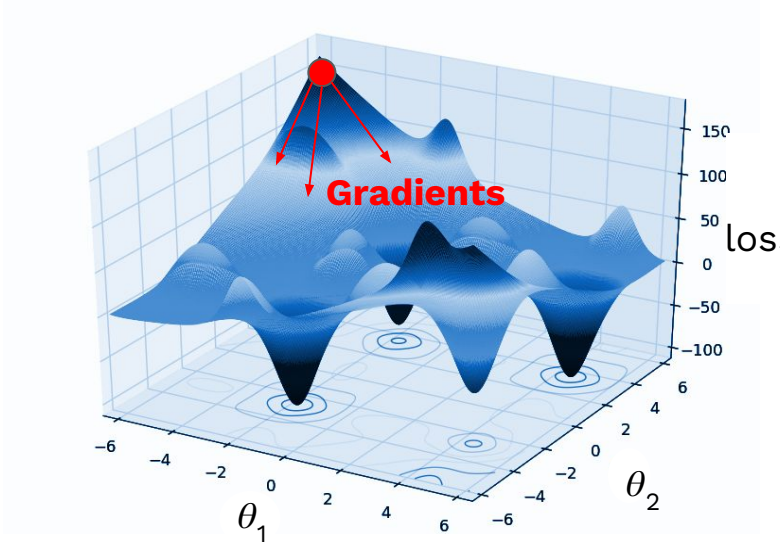
Most common variants are some form of gradient descent:

$$\theta^{t+1} = \theta^{t} + \nu \cdot \nabla_{\theta} \text{loss}$$

new weights — old weights — learning rate — gradients

Components of Neural Networks

- Architecture
- Loss
- Optimizer / Training Schedul(er)s



Most common variants are some form of gradient descent:

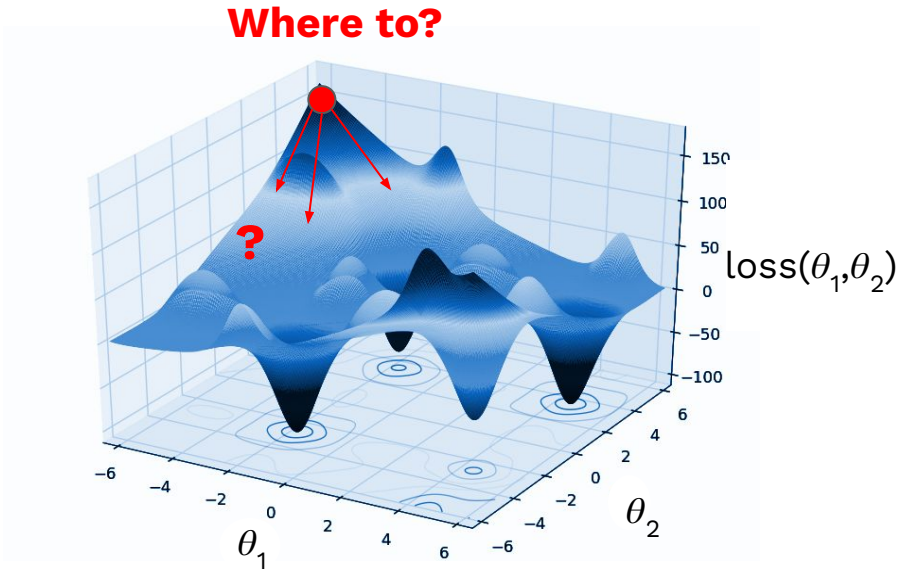
$$\theta^{t+1} = \theta^{t} + \nu \cdot \nabla_{\theta} \text{loss}$$

new weights — old weights — learning rate — gradients

t1p.de/dl_handout | t1p.de/dl_notebook

Components of Neural Networks

- Architecture
- Loss
- Optimizer / Training Schedul(er)s



Most common variants are some form of gradient descent:

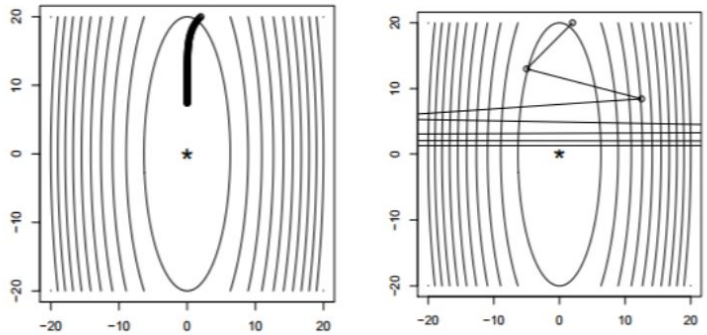
$$\theta^{t+1} = \theta^{t} + \nu \cdot \nabla_{\theta} \text{loss}$$

new weights | old weights | learning rate | gradients

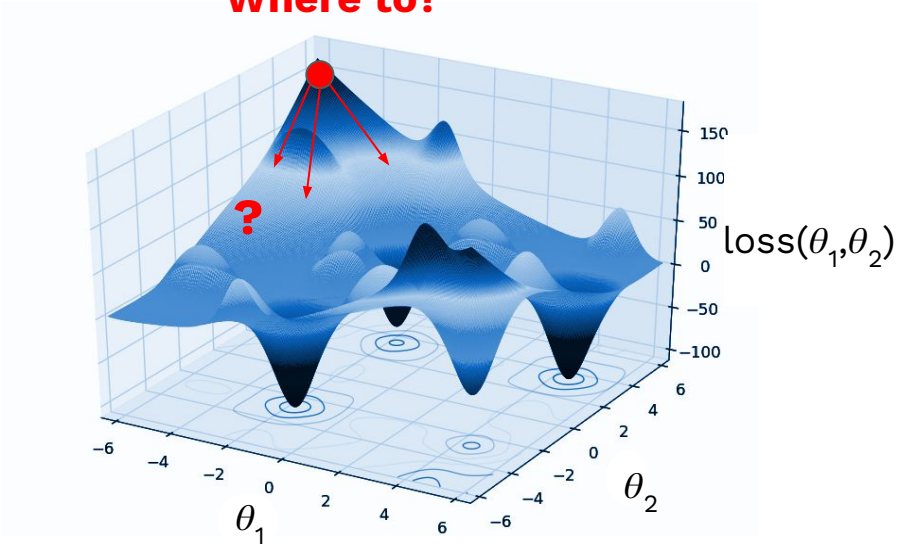
Components of Neural Networks

- Architecture
- Loss
- Optimizer / Training Schedul(er)s

What speed?



Where to?



Components of Neural Networks

- Architecture
- Loss
- Optimizer / Training Schedul(er)s

Components of Neural Networks

- Architecture
- Loss
- Optimizer / Training Schedul(er)s
  - One of the major driving factors for the success of DL

Components of Neural Networks

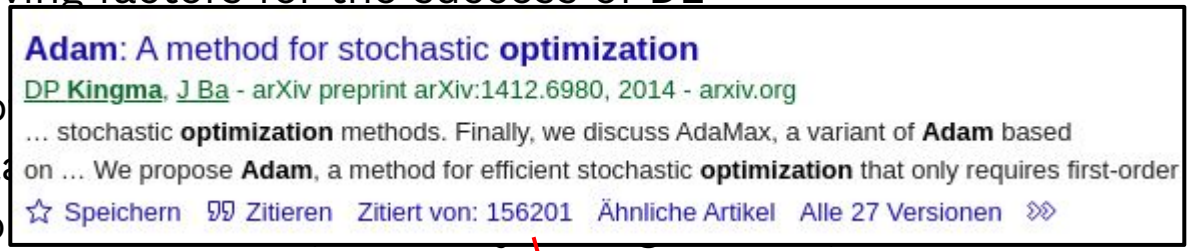
- Architecture
- Loss
- Optimizer / Training Schedul(er)s
  - One of the major driving factors for the success of DL
  - While there is
    - → a great variety of optimizers is available
    - → and software platforms allow custom routines
    - → most are first-order methods (work only with gradients)

Components of Neural Networks

- Architecture
- Loss
- Optimizer / Training Schedul(er)s
  - One of the major driving factors for the success of DL
  - While there is
    - → a great variety of optimizers is available
    - → and software platforms allow custom routines
    - → most are first-order methods (work only with gradients)
    - → The most successful ones (Adam, RMSprop, Adagrad, ...) use
      - Momentum
      - Weight-specific learning rates
      - Curvature approximations

Components of Neural Networks

- Architecture
- Loss
- Optimizer / Training Schedul(er)s
  - One of the major driving factors for the success of DL
  - While there is
    - → a great variety o
    - → and software pl
    - → most are first-o
    - → The most successful ones (Adam, RMSprop, Adagrad, ...) use
      - Momentum
      - Weight-specific learning rates
      - Curvature approximations



on average 48 citations per day for the last 9 years

Components of Neural Networks

- Architecture
- Loss
- Optimizer / Training Routine
  - One of the major driving factors for the success of DL
  - While there is
    - → a great variety of optimizers is available
    - → and software platforms allow custom routines
    - → most are first-order methods (work only with gradients)
    - → The most successful ones (Adam, RMSprop, Adagrad, ...) use
      - Momentum
      - Weight-specific learning rates
      - Curvature approximations

Source: Towards Data Science

Components of Neural Networks

- Architecture
- Loss
- Optimizer / Training Schedul(er)s
  - One of the major driving factors for the success of DL
  - The most successful ones (Adam, RMSprop, Adagrad, ...) use
    - → Momentum
    - → Weight-specific learning rates
    - → Curvature approximations

Components of Neural Networks

- Architecture
- Loss
- Optimizer / Training Schedul(er)s
  - One of the major driving factors for the success of DL
  - The most successful ones (Adam, RMSprop, Adagrad, ...) use
    - → Momentum
    - → Weight-specific learning rates
    - → Curvature approximations ⇒ WHY NO 2nd-order methods?

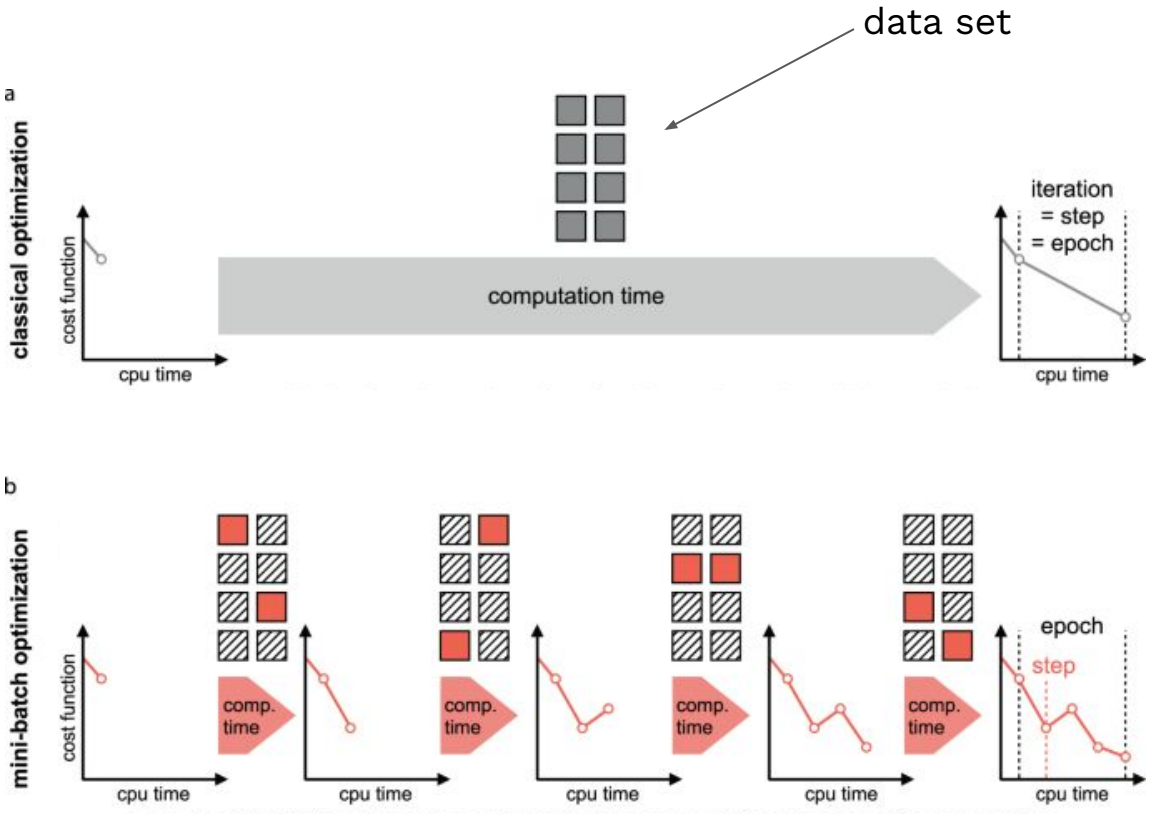
Components of Neural Networks

- Architecture
- Loss
- Optimizer / Training Schedul(er)s
  - One of the major driving factors for the success of DL
  - The most successful ones (Adam, RMSprop, Adagrad, ...) use
    - → Momentum
    - → Weight-specific learning rates
    - → Curvature approximations
  - Optimization is usually done in Mini-batches

Components of Neural Networks

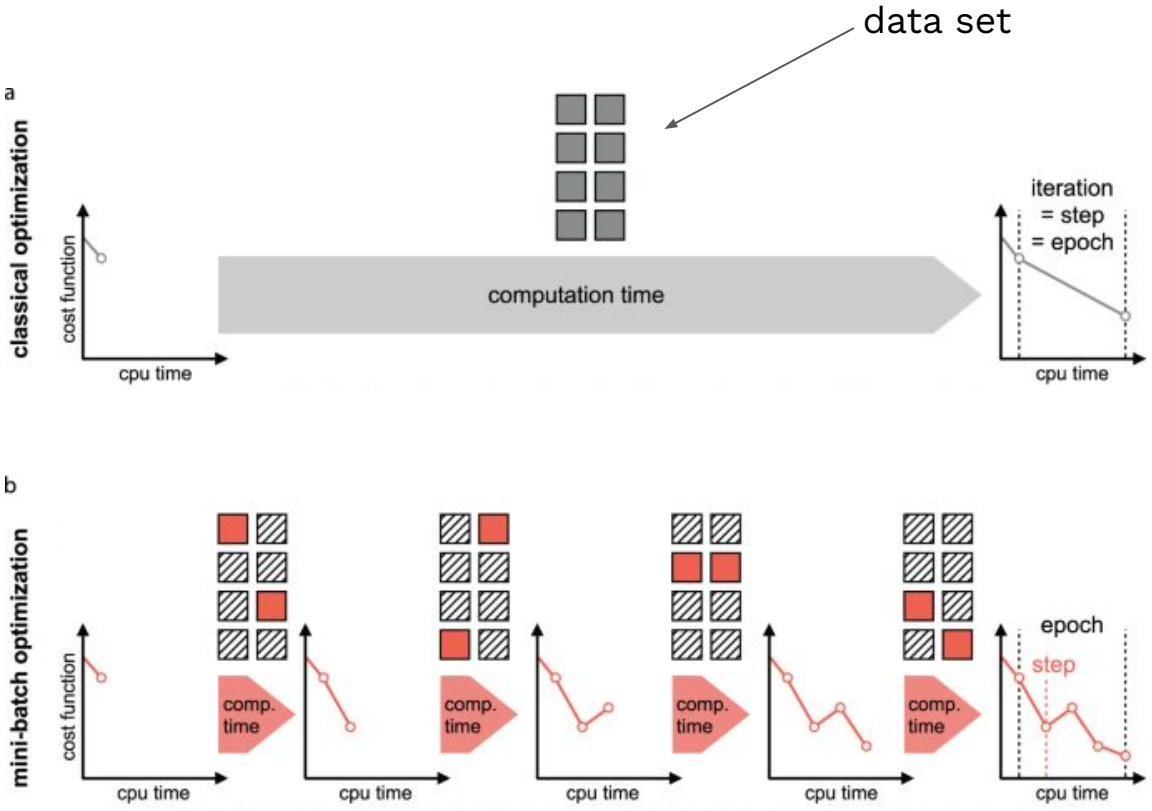
- Architecture
- Loss
- Optimizer / Training Schedul(er)s
  - One of the major driving factors for the success of DL
  - The most successful ones (Adam, RMSprop, Adagrad, ...) use
    - → Momentum
    - → Weight-specific learning rates
    - → Curvature approximations
  - Optimization is usually done in Mini-batches
    - → for each iteration (called *epoch*), data is divided in *B* batches
    - → i.e. *B* optimization steps for the whole data set
    - → “Stochastic gradient descent (SGD)-optimization”

Components of Neural Networks



Source: [9]

Components of Neural Networks



Typical batch sizes:
32, 64, 128

Source: [9]

Components of Neural Networks

- Architecture
- Loss
- Optimizer / Training Schedul(er)s

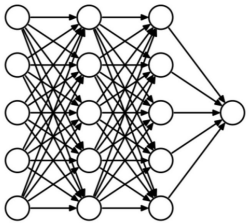
Components of Neural Networks

- Architecture
- Loss
- Optimizer / Training Schedul(er)s

Putting all together

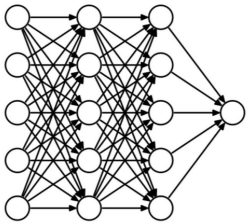
Components of Neural Networks

1. Define Architecture



Components of Neural Networks

1. Define Architecture

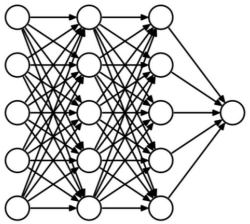


2. Define Loss

- MSE
- Cross-Entropy
- ...

Components of Neural Networks

1. Define Architecture



2. Define Loss

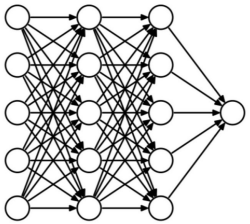
- MSE
- Cross-Entropy
- ...

3. Define Optimizer

- Adam
- SGD
- ...

Components of Neural Networks

1. Define Architecture



2. Define Loss

- MSE
- Cross-Entropy
- ...

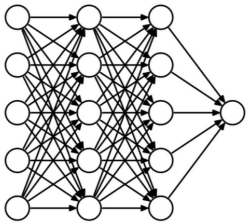
3. Define Optimizer

- Adam
- SGD
- ...

4. Initializes network (graph / gradients etc.)

Components of Neural Networks

1. Define Architecture



2. Define Loss

- MSE
- Cross-Entropy
- ...

3. Define Optimizer

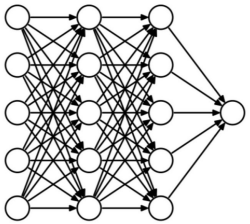
- Adam
- SGD
- ...

4. Initializes network (graph / gradients etc.)

5. Until some stopping criterion is met, do
for(i in 1:nr_iterations)
    for(b in 1:nr_batches)
        - compute predictions
        - calculate loss
        - update weights

Components of Neural Networks

1. Define Architecture



2. Define Loss

- MSE
- Cross-Entropy
- ...

3. Define Optimizer

- Adam
- SGD
- ...

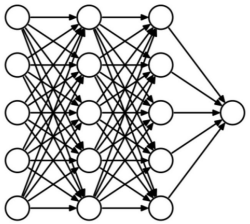
4. Initializes network (graph / gradients etc.)

DL software will do this for you

5. Until some stopping criterion is met, do
for(i in 1:nr_iterations)
    for(b in 1:nr_batches)
        - compute predictions
        - calculate loss
        - update weights

Components of Neural Networks

1. Define Architecture



2. Define Loss

- MSE
- Cross-Entropy
- ...

3. Define Optimizer

- Adam
- SGD
- ...

4. Initializes network (graph / gradients etc.)

DL software will do this for you

5. Until some stopping criterion is met, do
for(i in 1:nr_iterations)
    for(b in 1:nr_batches)
        - compute predictions
        - calculate loss
        - update weights

higher-level APIs (keras, luz, lightning) will do this for you

Components of Neural Networks

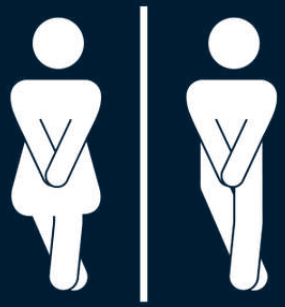
- Architecture
- Loss
- Optimizer / Training Schedul(er)s

Putting all together

Munich Center for Machine Learning

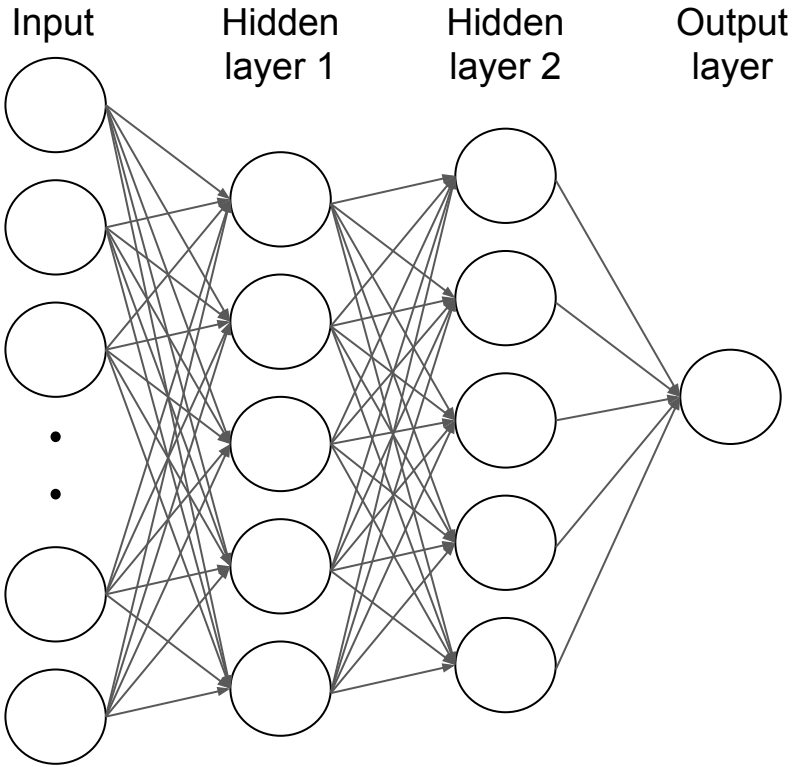
LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Break?

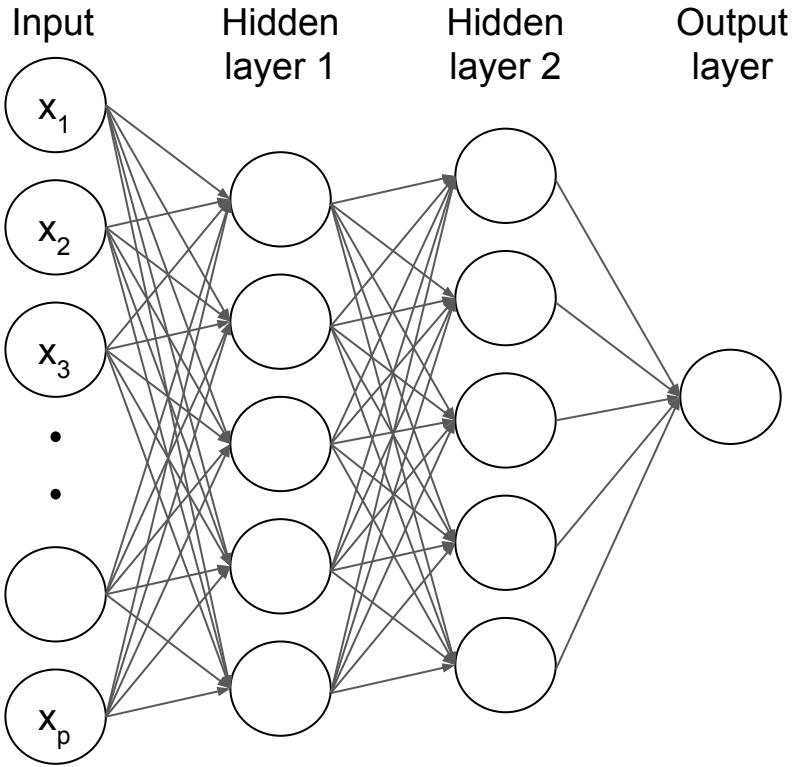


Neural Networks from a Statistical Point of View

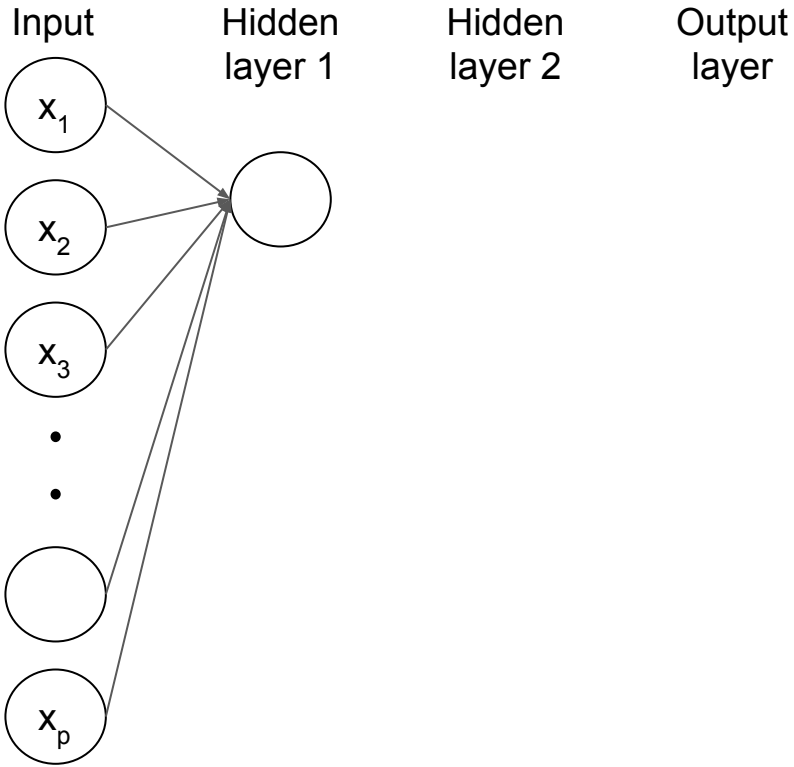
A Multi-layer Perceptron



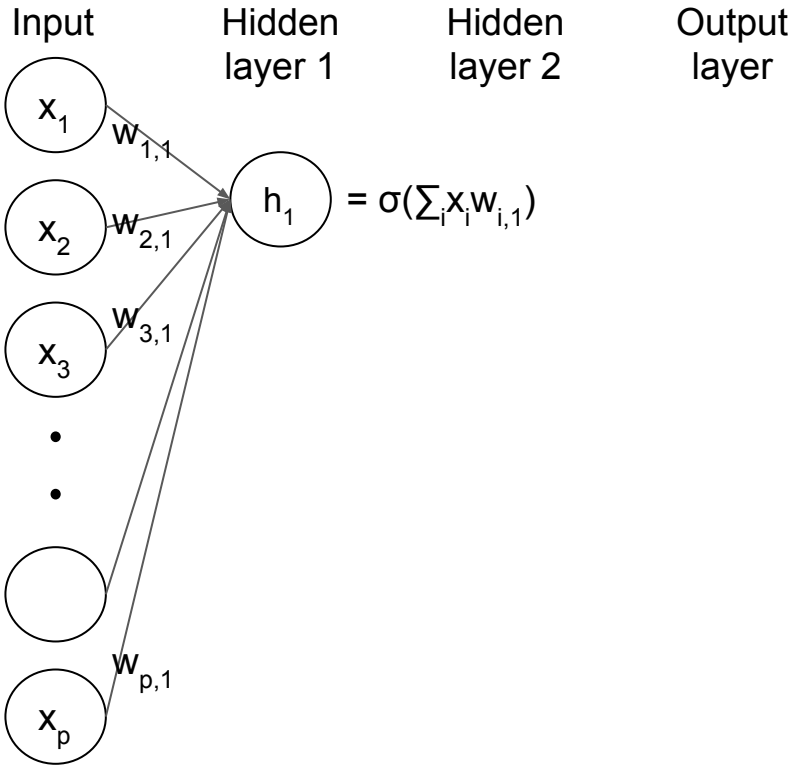
A Multi-layer Perceptron



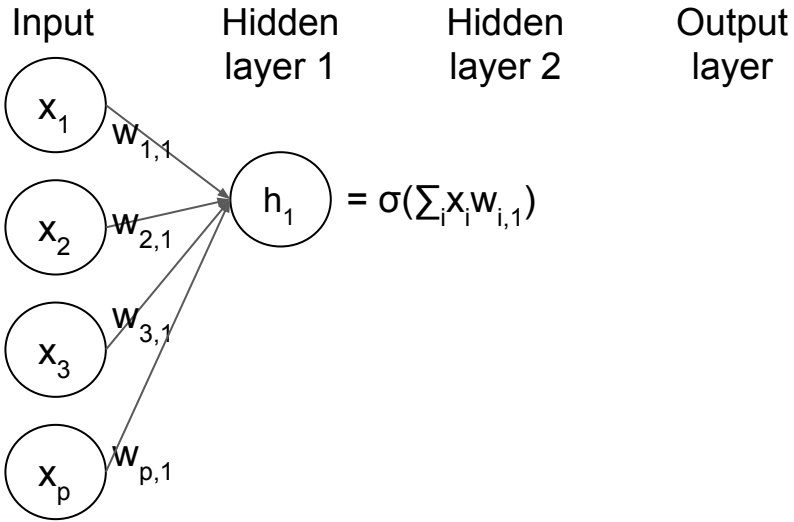
A Multi-layer Perceptron



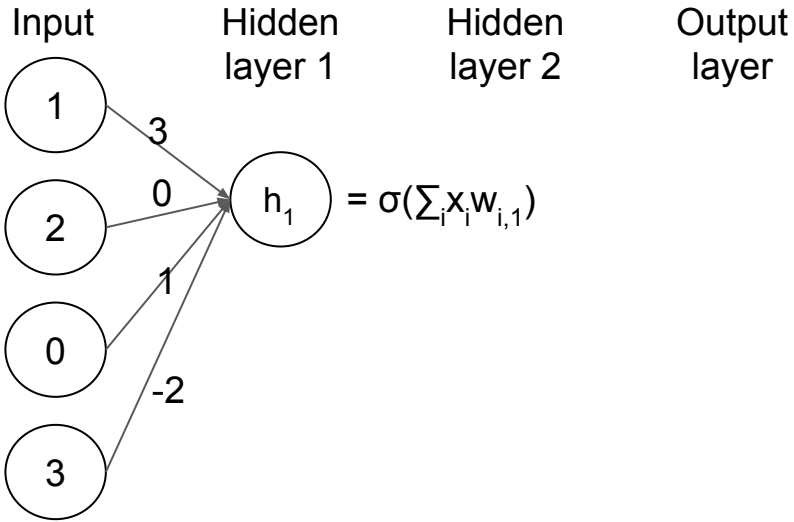
A Multi-layer Perceptron



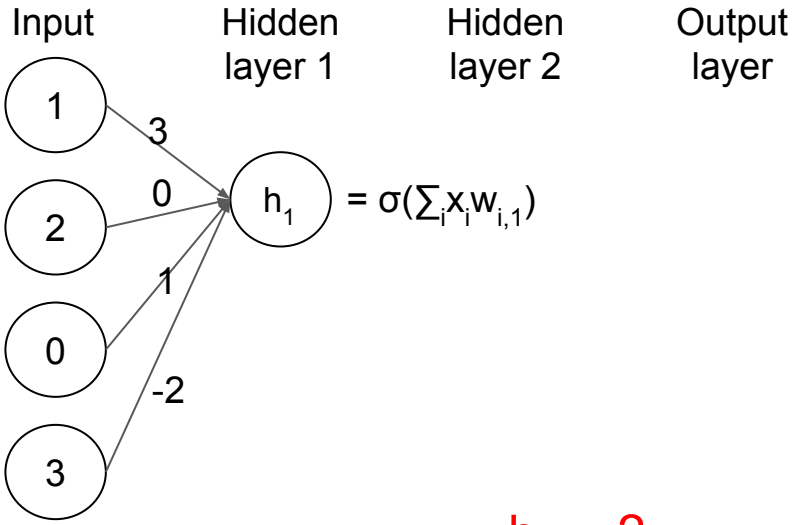
A Multi-layer Perceptron



A Multi-layer Perceptron

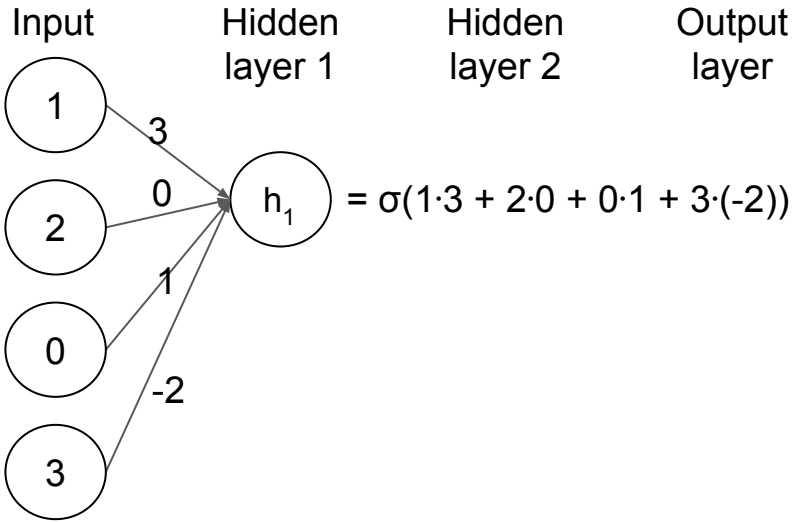


A Multi-layer Perceptron

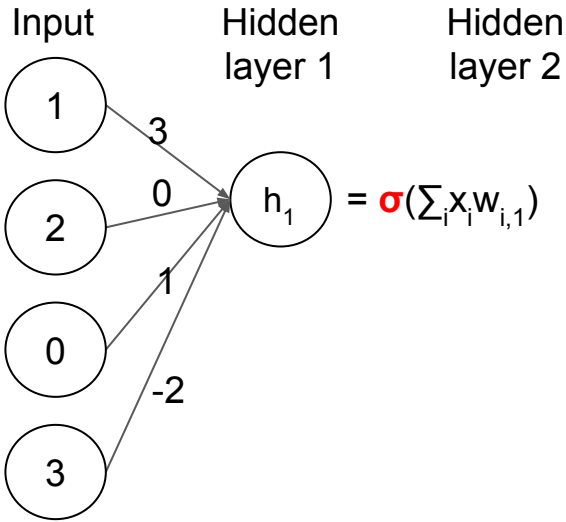


$h_1 = ?$

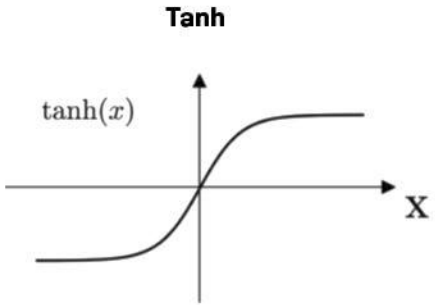
A Multi-layer Perceptron

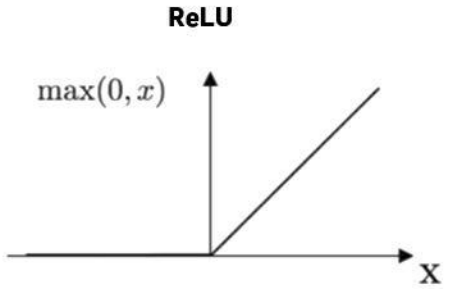


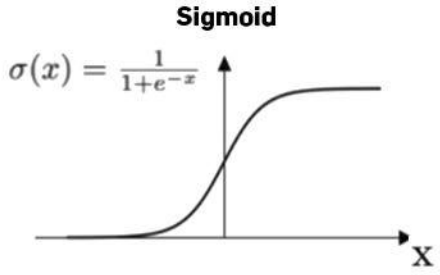
A Multi-layer Perceptron

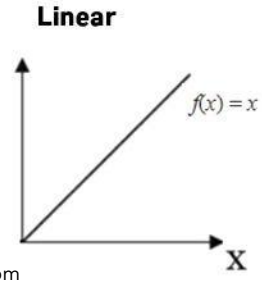


σ: Activation Function



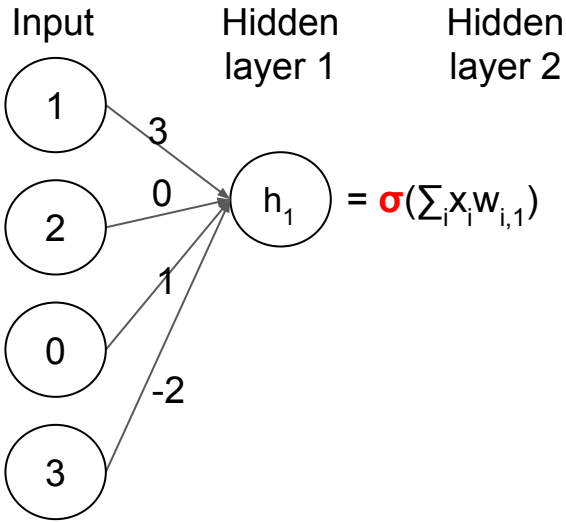




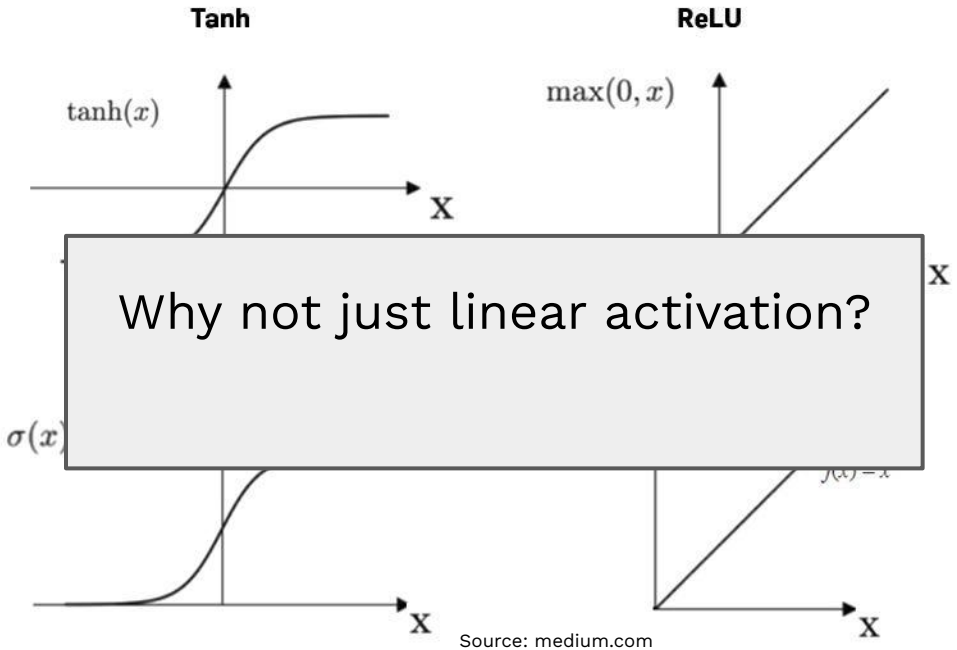


Source: medium.com

A Multi-layer Perceptron



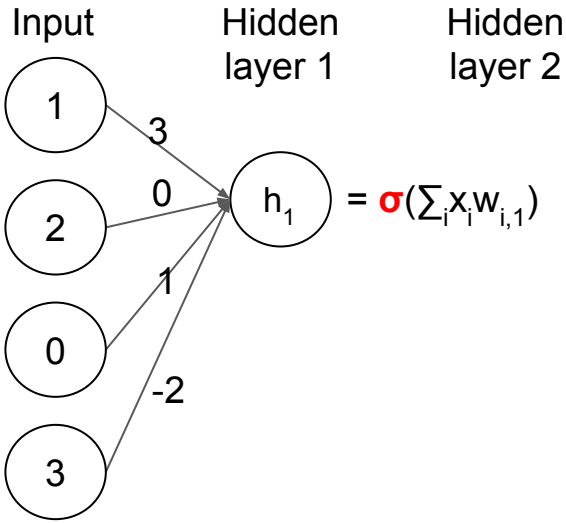
σ: Activation Function



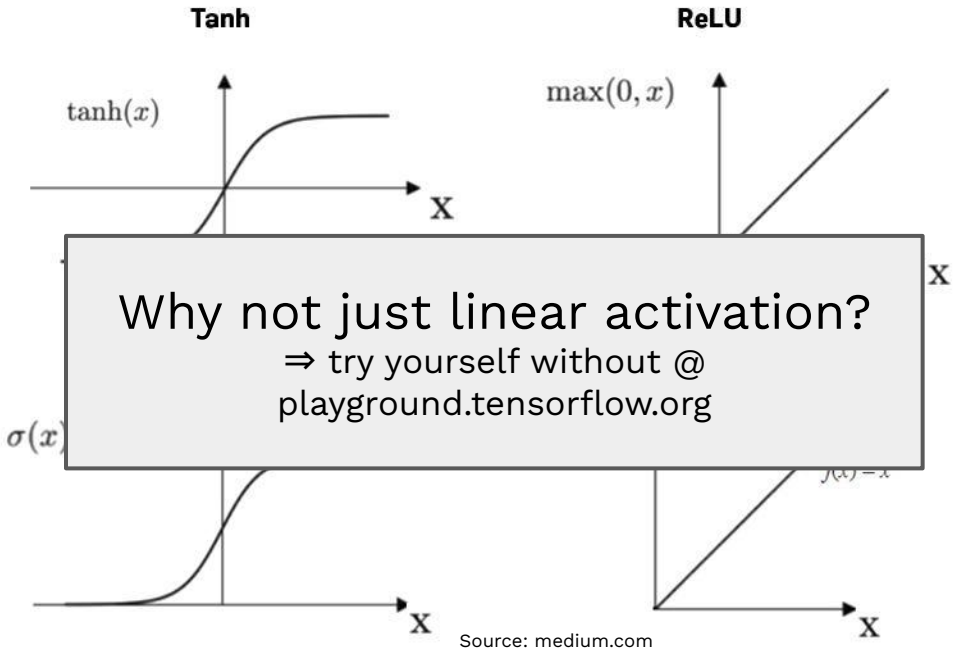
Why not just linear activation?

Source: medium.com

A Multi-layer Perceptron

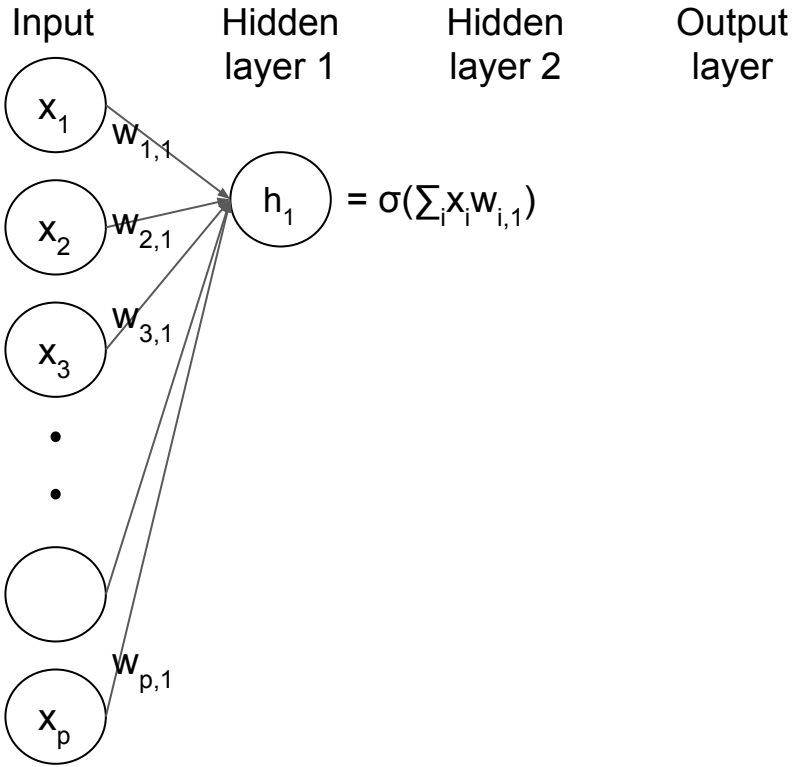


σ: Activation Function

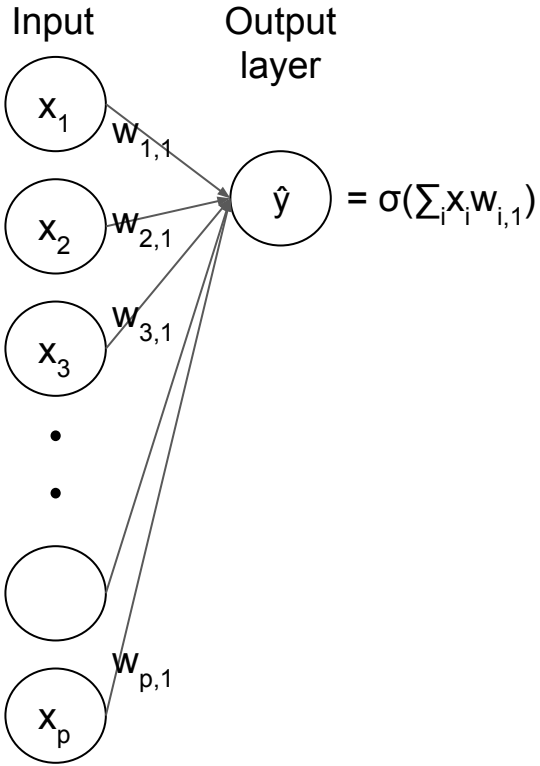


Source: medium.com

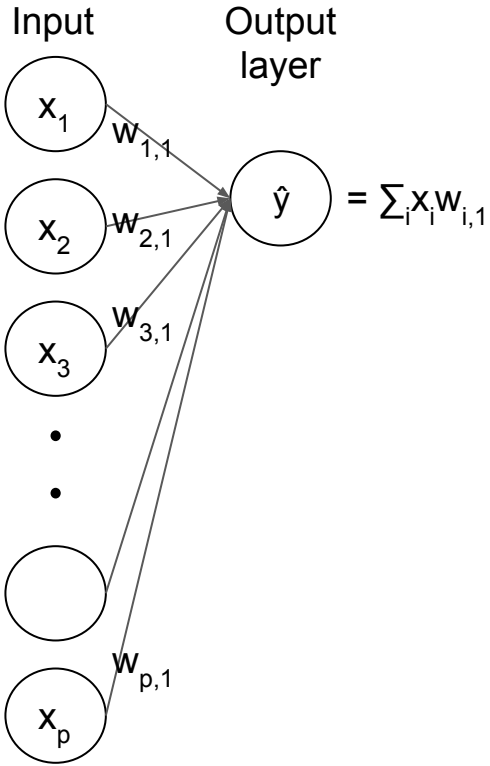
A Multi-layer Perceptron



A Single-layer Perceptron

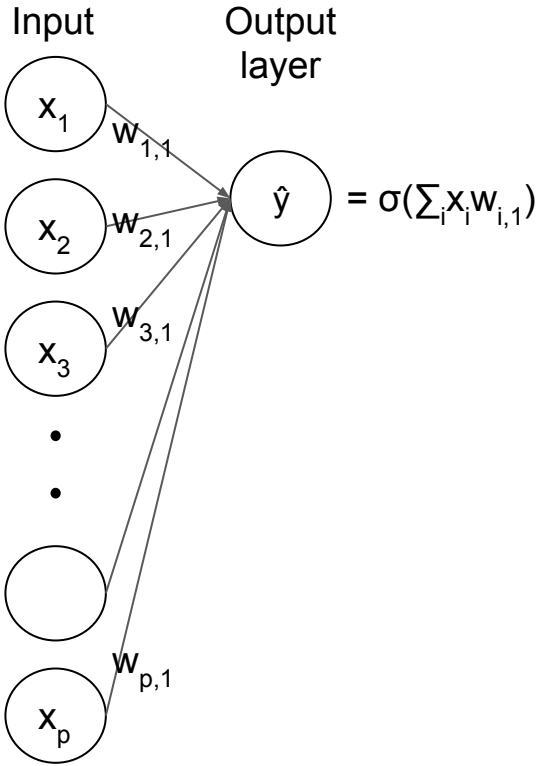


A Linear Model



with loss $(\hat{y}-y)^2$

A Generalized Linear Model



with loss = neg. log-Likelihood

and σ = …

… sigmoid for Bernoulli
… exp for Poisson

Components of Neural Networks

- Architecture
- Loss
  - The typical decision process:
    - ➢ “Regression” ⇒ L2-loss
    - ➢ “Classification” ⇒ Binary Cross Entropy / Log-loss
    - ➢ “Multiclass” ⇒ Cross Entropy

Components of Neural Networks

- Architecture
- Loss
  - The typical decision process:
    - ➢ “Regression” ⇒ L2-loss ⇔ equivalent to neg. Gaussian Log-likelihood
    - ➢ “Classification” ⇒ Binary Cross Entropy / Log-loss
      ⇔ equivalent to neg. Bernoulli Log-likelihood
    - ➢ “Multiclass” ⇒ Cross Entropy
      ⇔ equivalent to neg. Multinomial Log-likelihood

Components of Neural Networks

- Architecture
- Loss
  - The typical decision process:
    - ➢ “Regression” ⇒ L2-loss ⇔ equivalent to neg. Gaussian Log-likelihood
    - ➢ “Classification” ⇒ Binary Cross Entropy / Log-loss
      ⇔ equivalent to neg. Bernoulli Log-likelihood
    - ➢ “Multiclass” ⇒ Cross Entropy
      ⇔ equivalent to neg. Multinomial Log-likelihood

In many cases: **Empirical Risk Minimization** = **Maximum Likelihood Estimation**

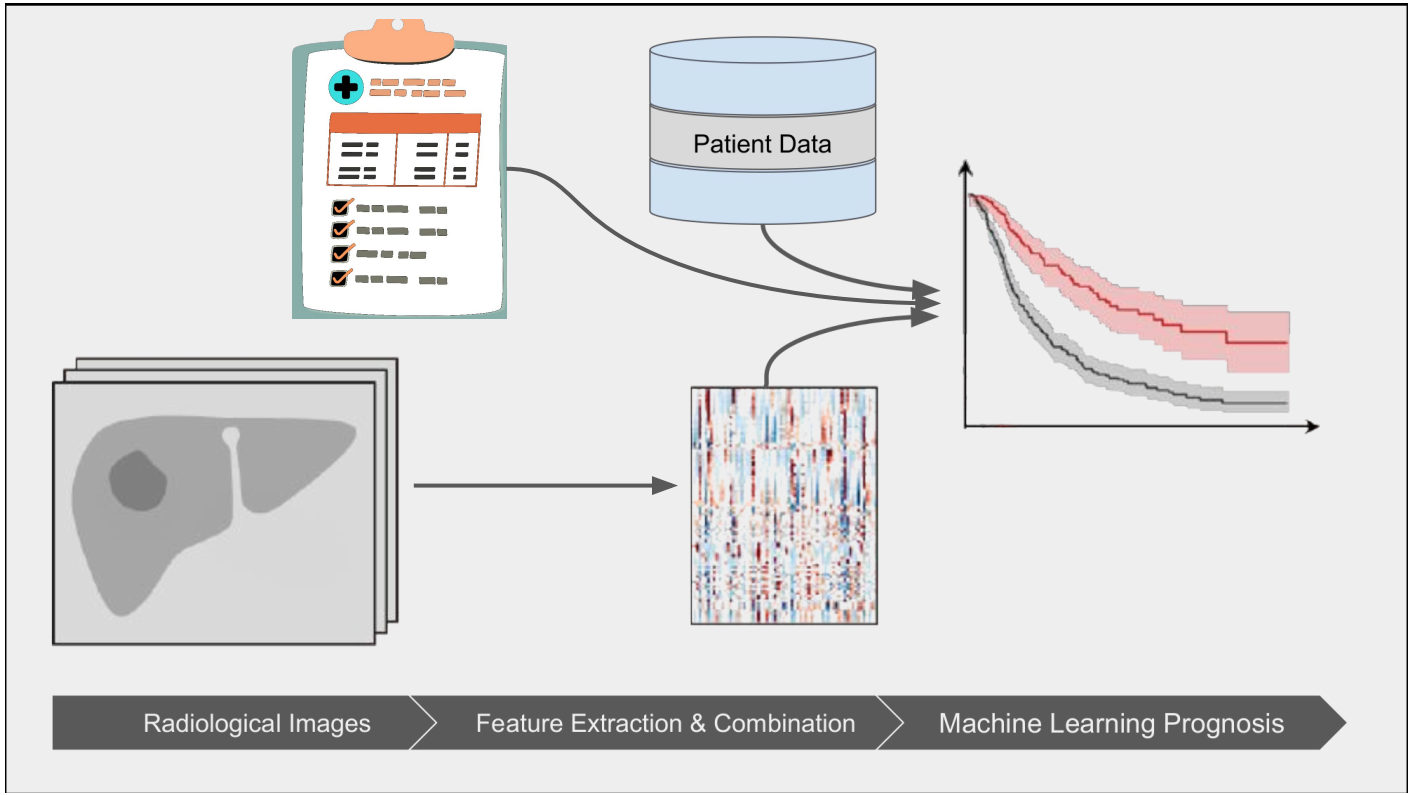
Components of Neural Networks

- Architecture
- Loss
  - The typical decision process:
    - ➢ “Regression” ⇒ L2-loss ⇔ equivalent to neg. Gaussian Log-likelihood
    - ➢ “Classification” ⇒ Binary Cross Entropy / Log-loss
      ⇔ equivalent to neg. Bernoulli Log-likelihood
    - ➢ “Multiclass” ⇒ Cross Entropy
      ⇔ equivalent to neg. Multinomial Log-likelihood

In many cases: **Empirical Risk Minimization** = **Maximum Likelihood Estimation**

Semi-Structured Neural Networks

Motivating Example: Multimodal Data Sets

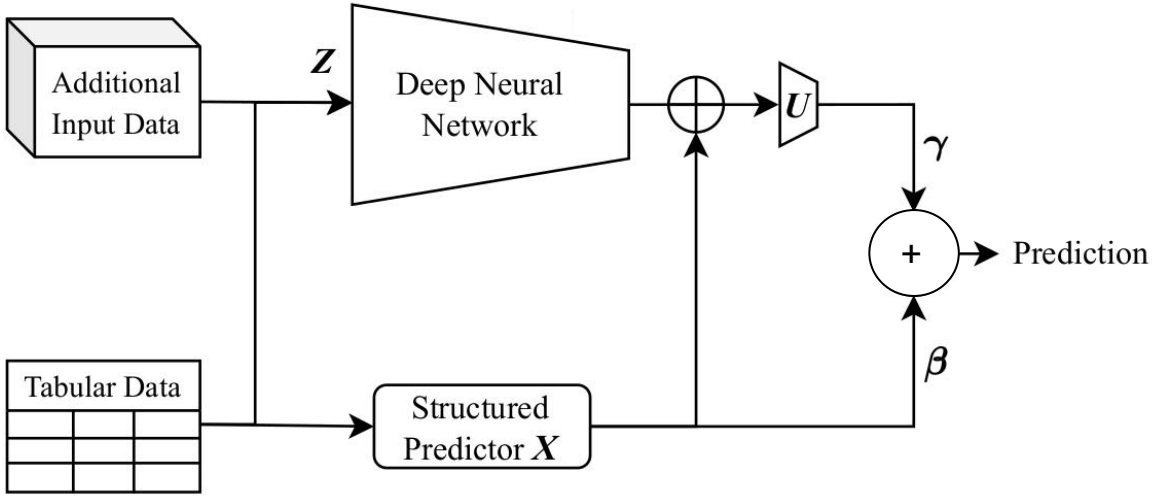


Semi-Structured Neural Networks

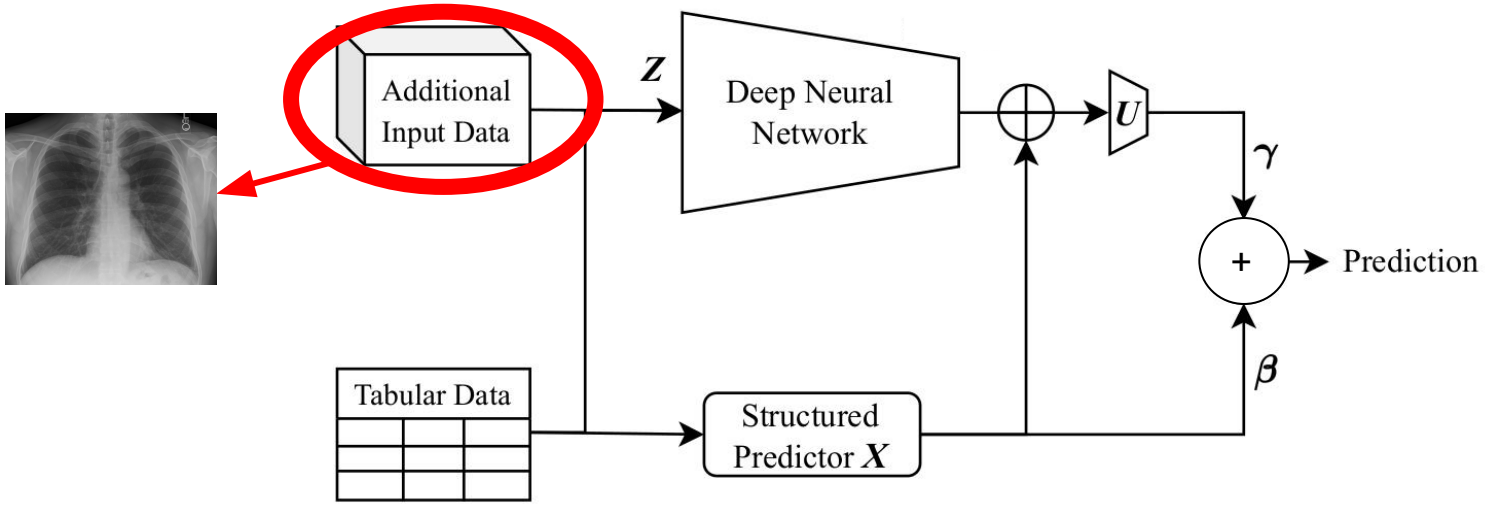
Idea:

- Jointly train statistical model
- and deep neural network(s)
- in one unifying large network

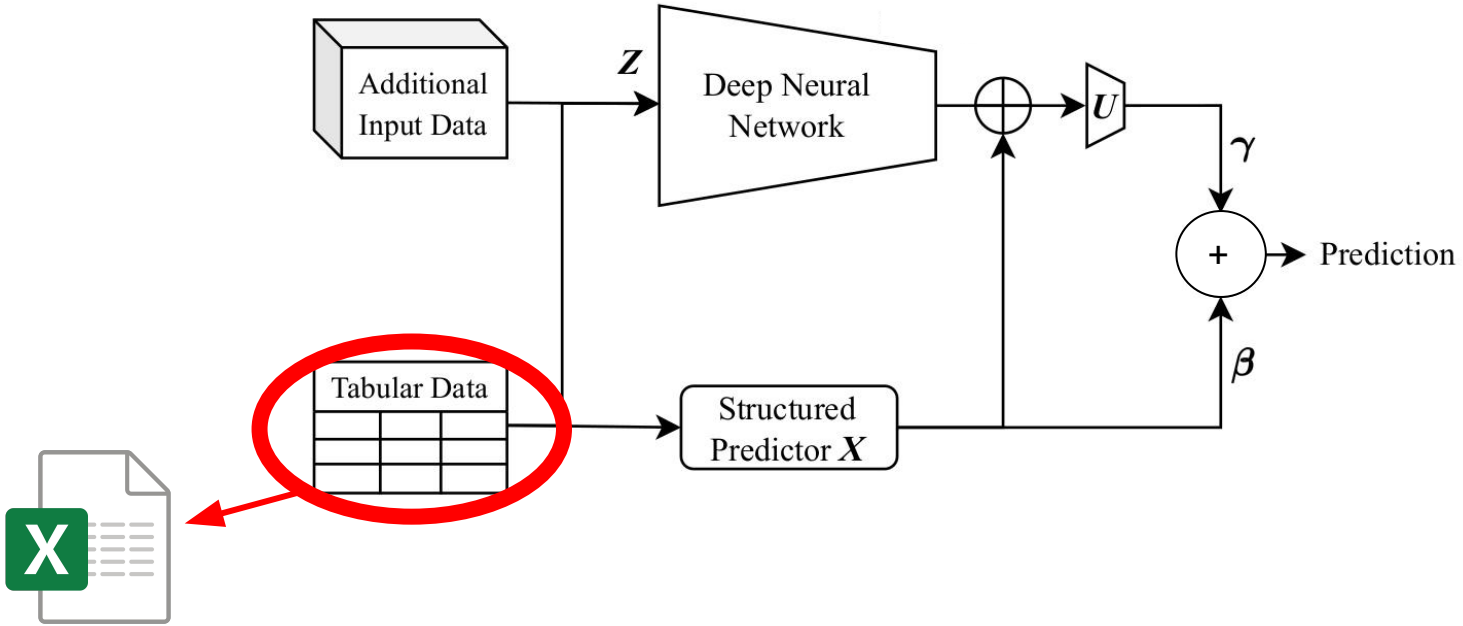
Semi-Structured Neural Networks



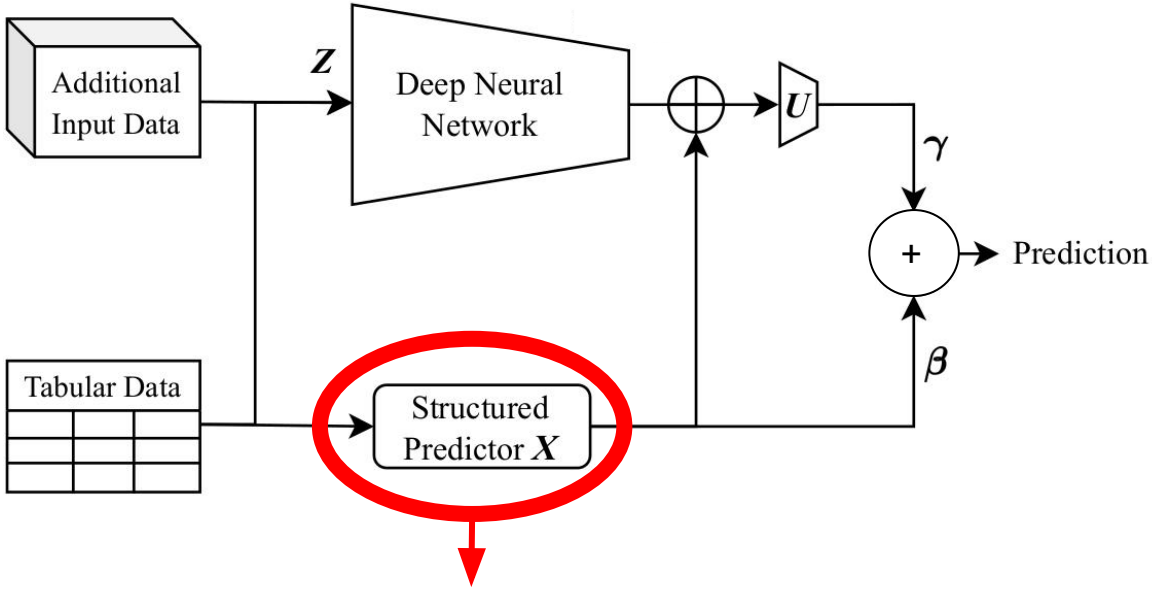
Semi-Structured Neural Networks



Semi-Structured Neural Networks

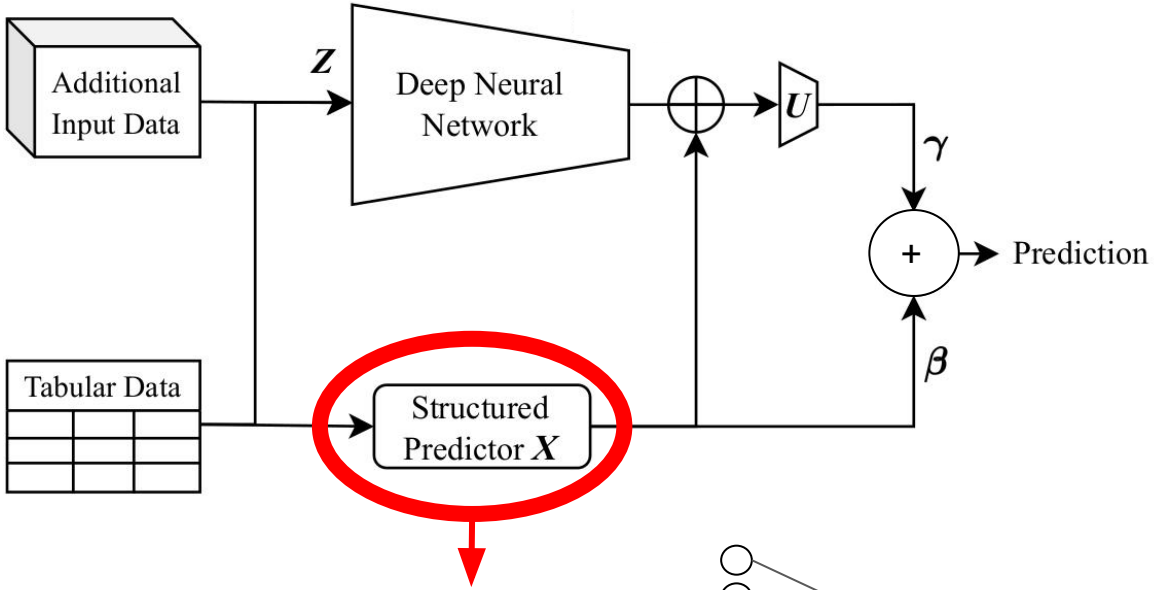


Semi-Structured Neural Networks

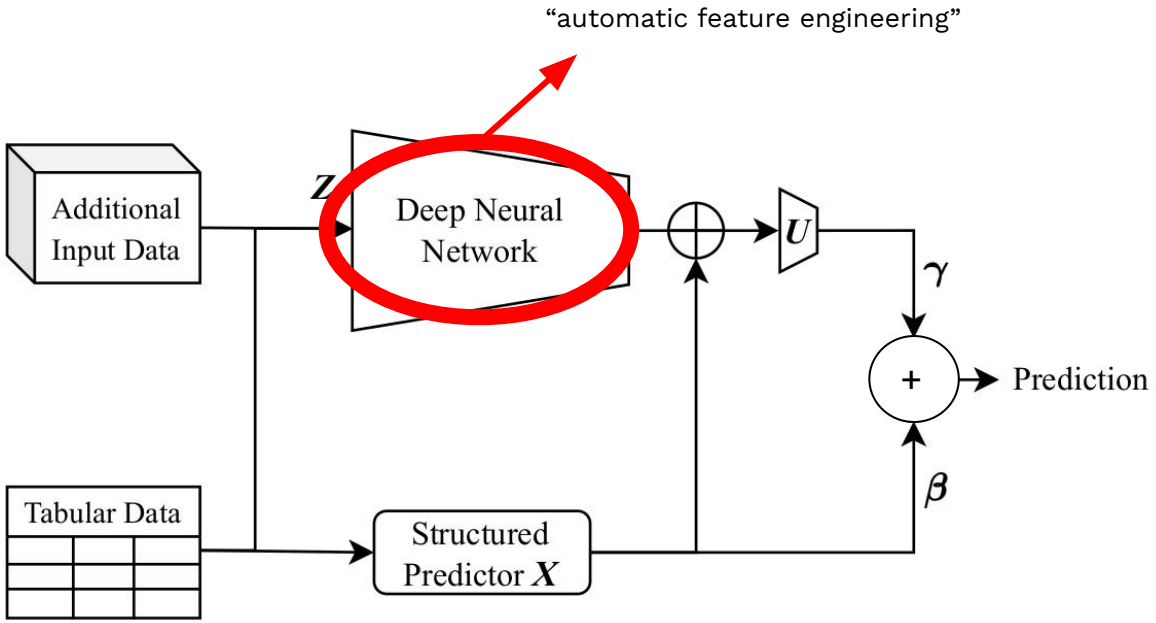


- linear effects
- splines
- tensor-product splines
- ...

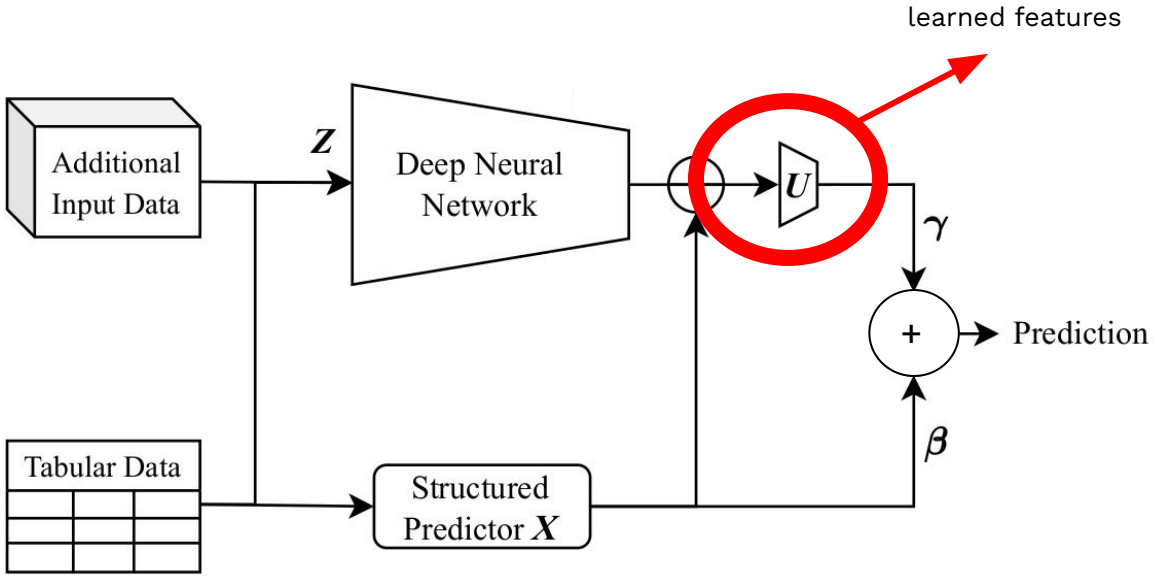
Semi-Structured Neural Networks



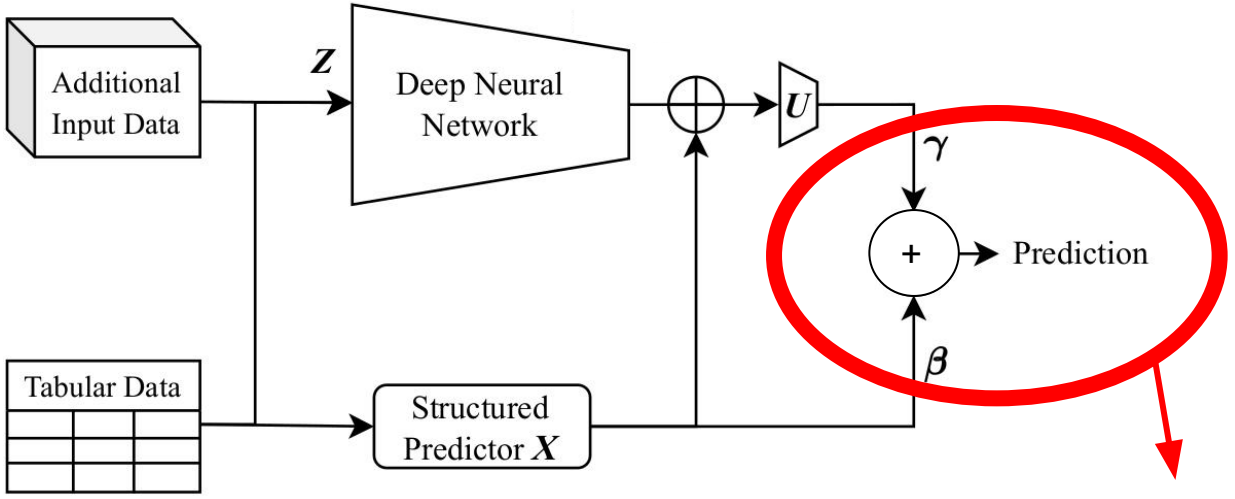
Semi-Structured Neural Networks



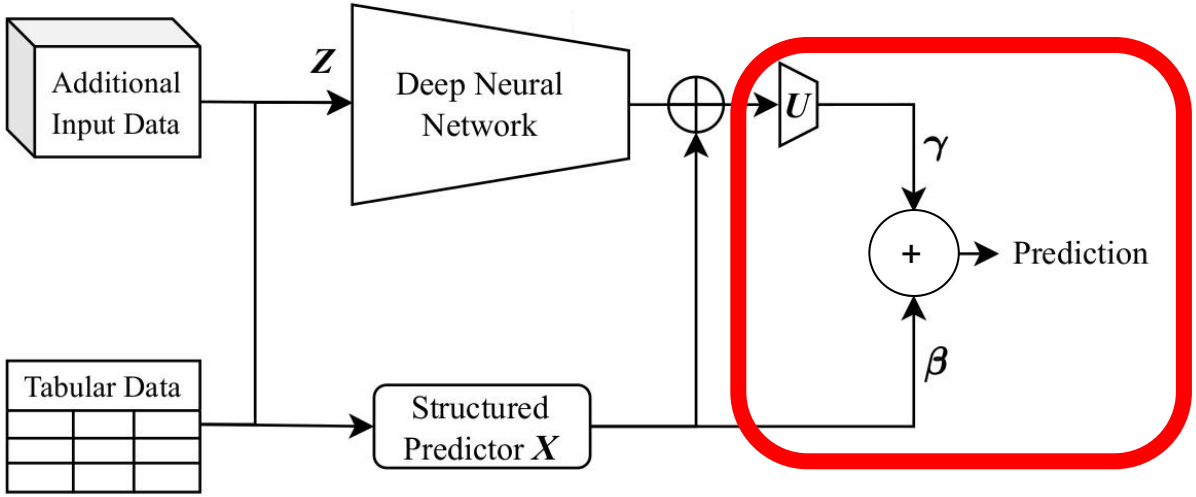
Semi-Structured Neural Networks



Semi-Structured Neural Networks

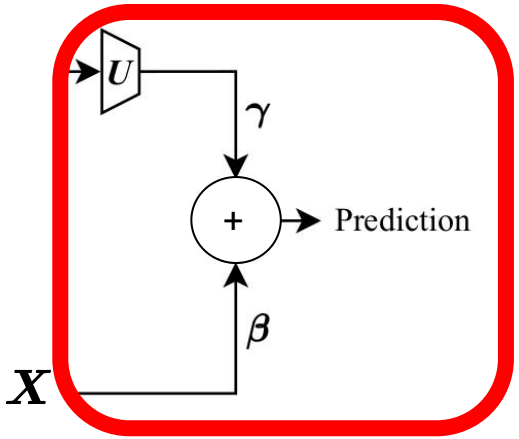


Semi-Structured Neural Networks



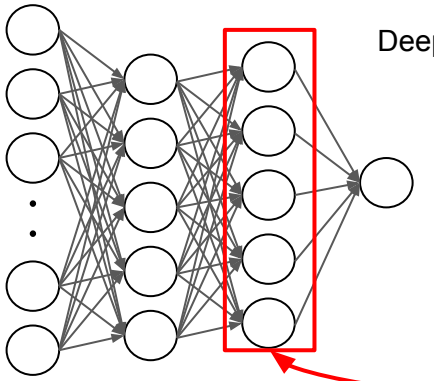
An additive model

Semi-Structured Neural Networks



$$\mathbb{E}(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{Z}) = \boldsymbol{\eta} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{U}\boldsymbol{\gamma}$$

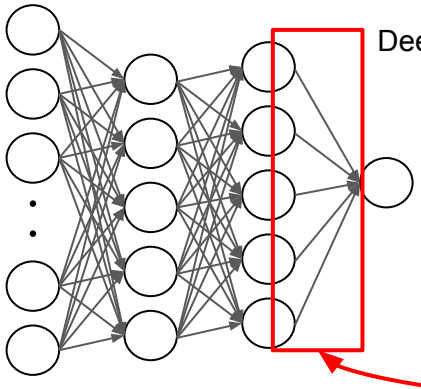
Semi-Structured Neural Networks



Deep Neural Network

$$\mathbb{E}(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{Z}) = \boldsymbol{\eta} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{U}\boldsymbol{\gamma}$$

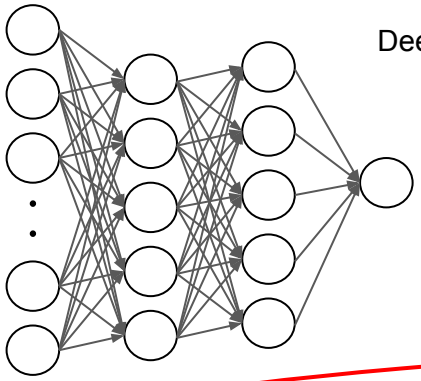
Semi-Structured Neural Networks



Deep Neural Network

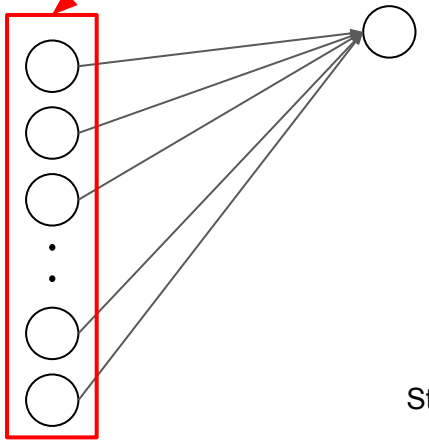
$$\mathbb{E}(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{Z}) = \boldsymbol{\eta} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{U}\boldsymbol{\gamma}$$

Semi-Structured Neural Networks



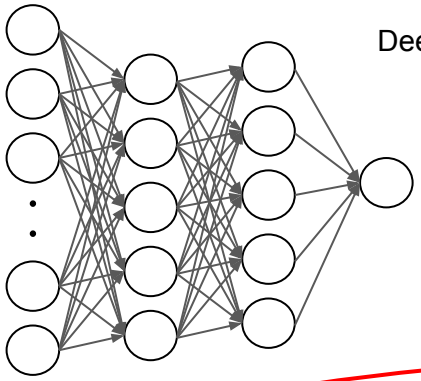
Deep Neural Network

$$\mathbb{E}(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{Z}) = \boldsymbol{\eta} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{U}\boldsymbol{\gamma}$$



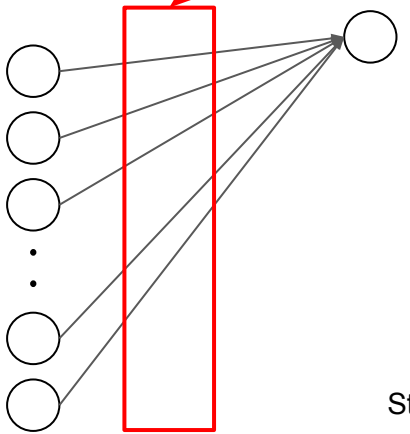
Structured Predictor

Semi-Structured Neural Networks



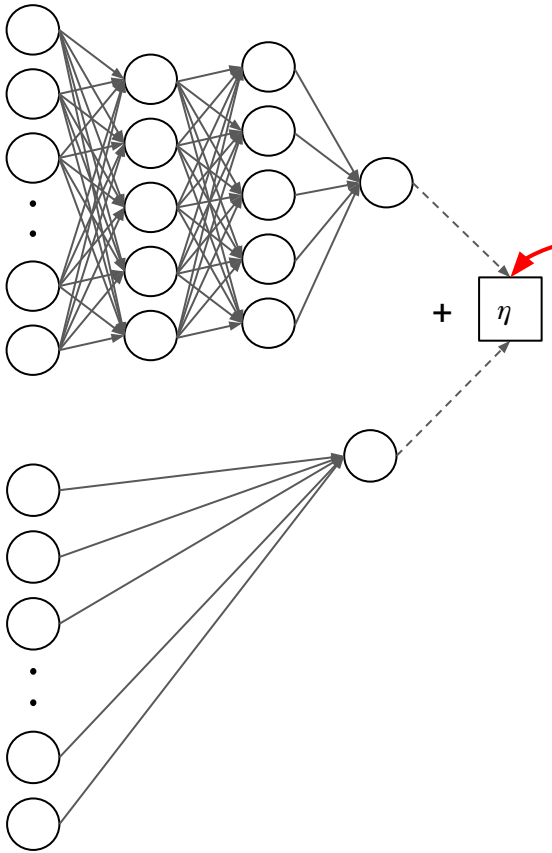
Deep Neural Network

$$\mathbb{E}(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{Z}) = \boldsymbol{\eta} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{U}\boldsymbol{\gamma}$$



Structured Predictor

Semi-Structured Neural Networks



$$\mathbb{E}(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{Z}) = \boldsymbol{\eta} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{U}\boldsymbol{\gamma}$$

Practical Implementation of Neural Networks

Software implementation

`deepregression` [10] (CRAN / Github)
R package combining TensorFlow/torch and mgcv

- Many different models (GLMs, GAMs, GAMLSS, etc.)
- Many different feature effects (splines, lasso, ridge, etc.)
- Convenience functions such as `coef`, `predict`, `plot`

Software implementation

`deepregression` [10] (CRAN / Github)
R package combining TensorFlow/torch and mgcv

- Many different models (GLMs, GAMs, GAMLSS, etc.)
- Many different feature effects (splines, lasso, ridge, etc.)
- Convenience functions such as `coef`, `predict`, `plot`

- Combine it with arbitrary neural networks

Example: Predicting Airbnb Prices in Munich

- Tabular information:
  - bathrooms, bedrooms, room type,
  - latitude, longitude,
  - ...
- Text description
- Image

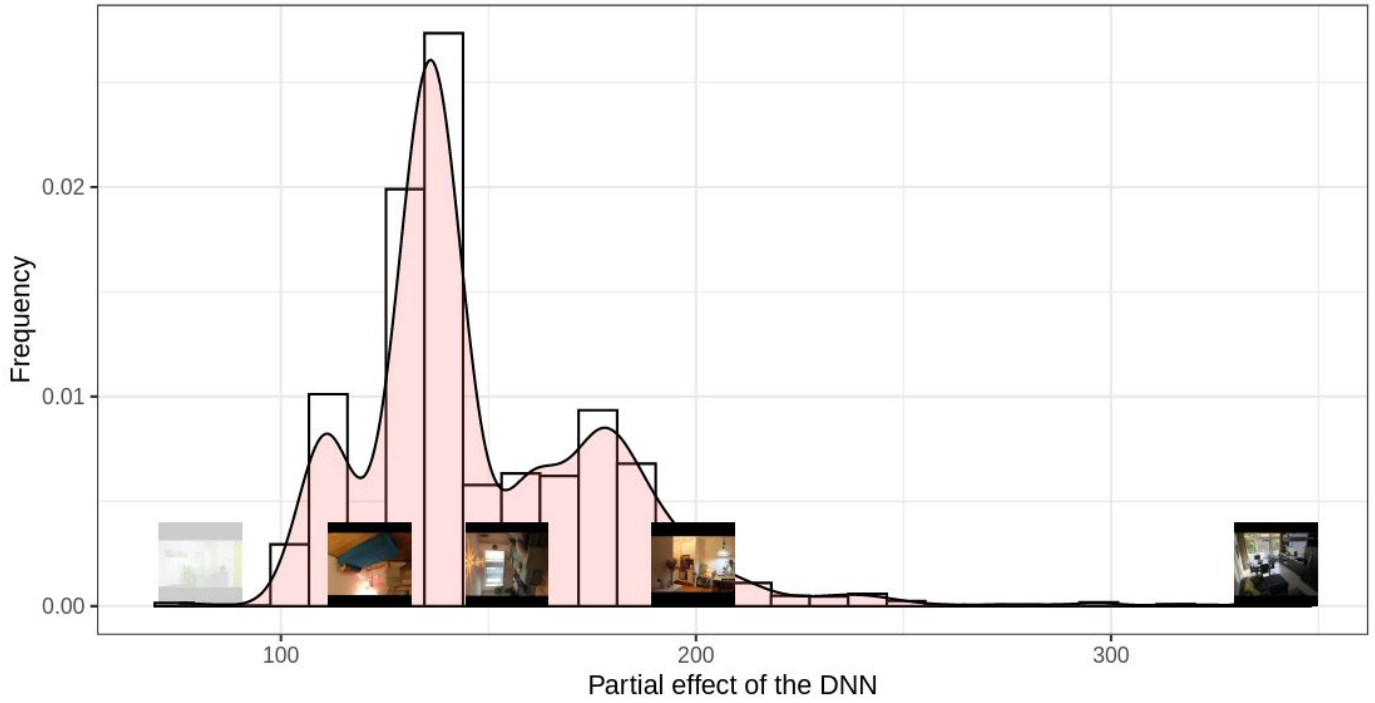
Example: Predicting Airbnb Prices in Munich

[...] Station for Metro, Bus, Tram are ony 1 minute walking.
The room is 16 square meters, Free WIFI.
We use together Bathroom and Kichen.
I prepare your Bath Towel.
you get Breakfast, cafe oder Tee, Bread..

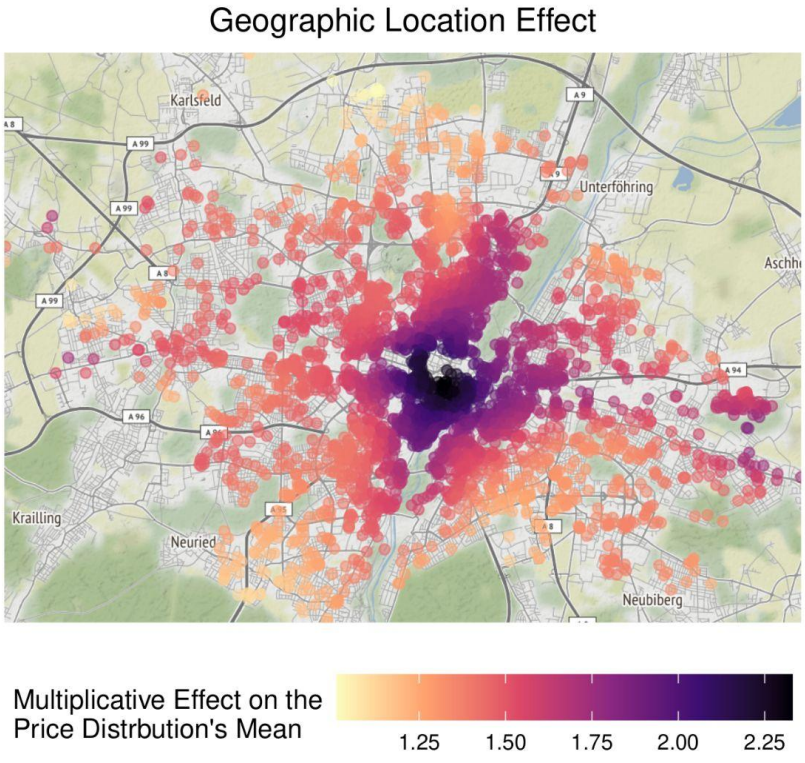
Example: Predicting Airbnb Prices in Munich

```
mod_airbnb <- deepregression(
    y = price,
    family = "log_normal",
    list_of_formulas = list(
      location = ~1 + te(latitude, longitude) + room_type + bedrooms +
                  cnn(image) + lstm(desc),
      scale = ~1
    ),
    data = d_train
)
```

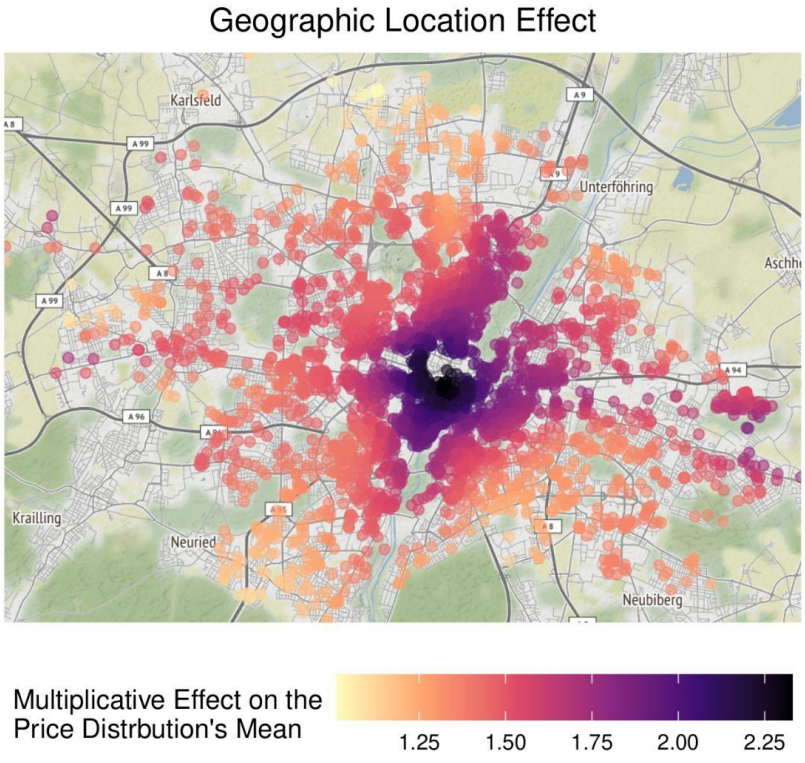
Example: Predicting Airbnb Prices in Munich



Example: Predicting Airbnb Prices in Munich



Example: Predicting Airbnb Prices in Munich



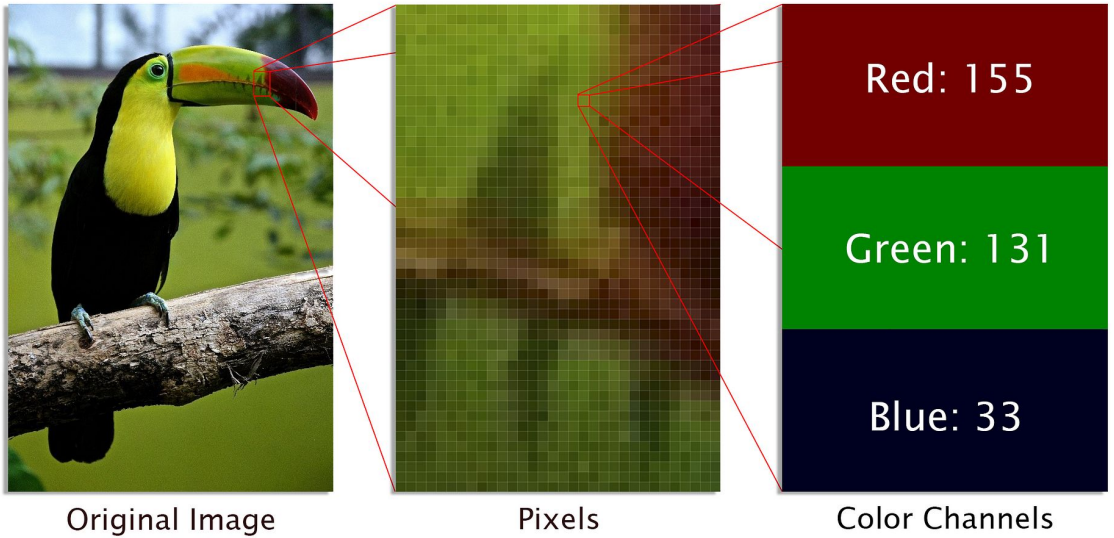
Outlook: How do different layers work?

Outlook: How do different layers work?
Convolutional Layers

... for processing Images

Outlook: How do different layers work?
Convolutional Layers

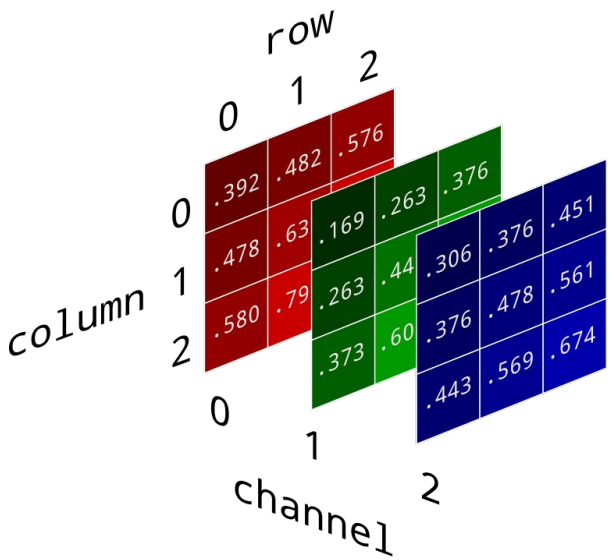
... for processing Images



Source: medium.com

Outlook: How do different layers work?
Convolutional Layers

... for processing Images



Source: e2eml.school

Outlook: How do different layers work?
Convolutional Layers

Why not fully-connected layers?

Hint: Think about the scaling for a 512x512 pixel image in RGB

Source: e2eml.school

Outlook: How do different layers work?
Convolutional Layers

Why not fully-connected layers?

Hint: Think about the scaling for a 512x512 pixel image in RGB

With only one hidden layer and a moderate amount of units (say 20) ...

... we would have 512*512*3*20 ≈ 15.7 mio parameters already

Source: e2eml.school

Outlook: How do different layers work?
Convolutional Layers

Why not fully-connected layers?

Hint: Think about the scaling for a 512x512 pixel image in RGB

With only one hidden layer and a moderate amount of units (say 20) …

… we would have 512*512*3*20 ≈ 15.7 mio parameters already
… and we probably wouldn’t even learn something meaningful

Source: e2eml.school

Outlook: How do different layers work?
Convolutional Layers

Why not fully-connected layers?

Hint: Think about the scaling for a 512x512 pixel image in RGB

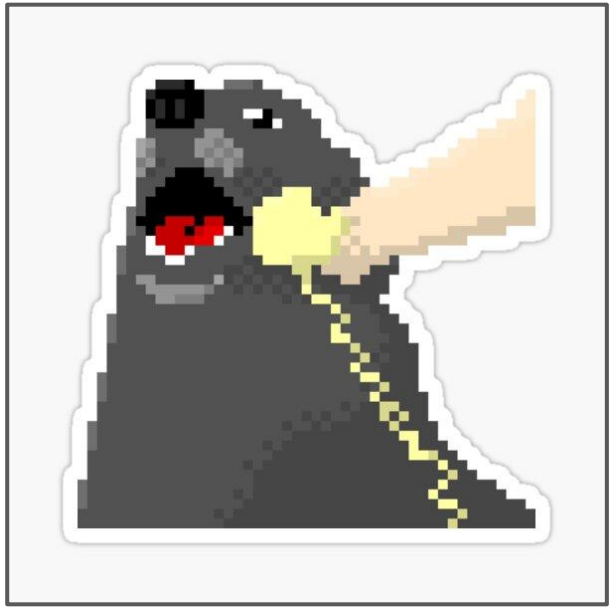
With only one hidden layer and a moderate amount of units (say 20) …

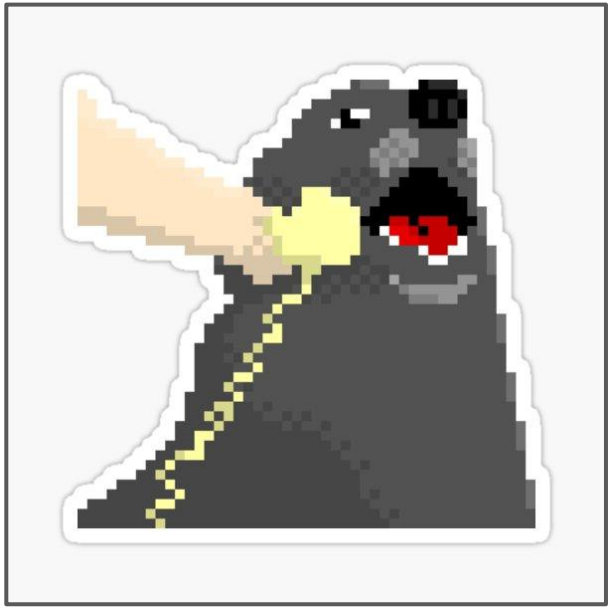
… we would have 512*512*3*20 ≈ 15.7 mio parameters already
… and we probably wouldn't even learn something meaningful **WHY?**

Source: e2eml.school

Outlook: How do different layers work?

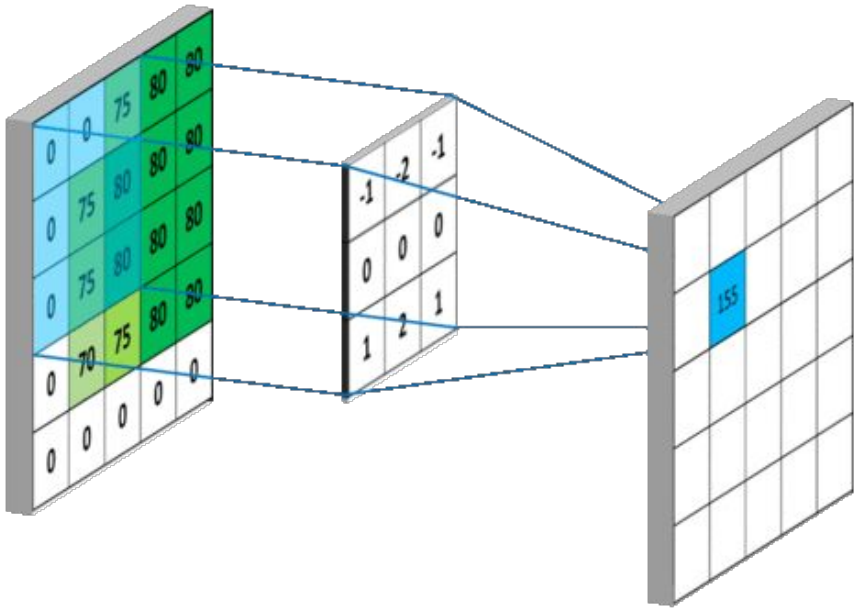
Convolutional Layers





Source: redbubble.com

Outlook: How do different layers work?
Convolutional Layers



Keyword: Weight sharing

Source: medium.com

Outlook: How do different layers work?
Convolutional Layers

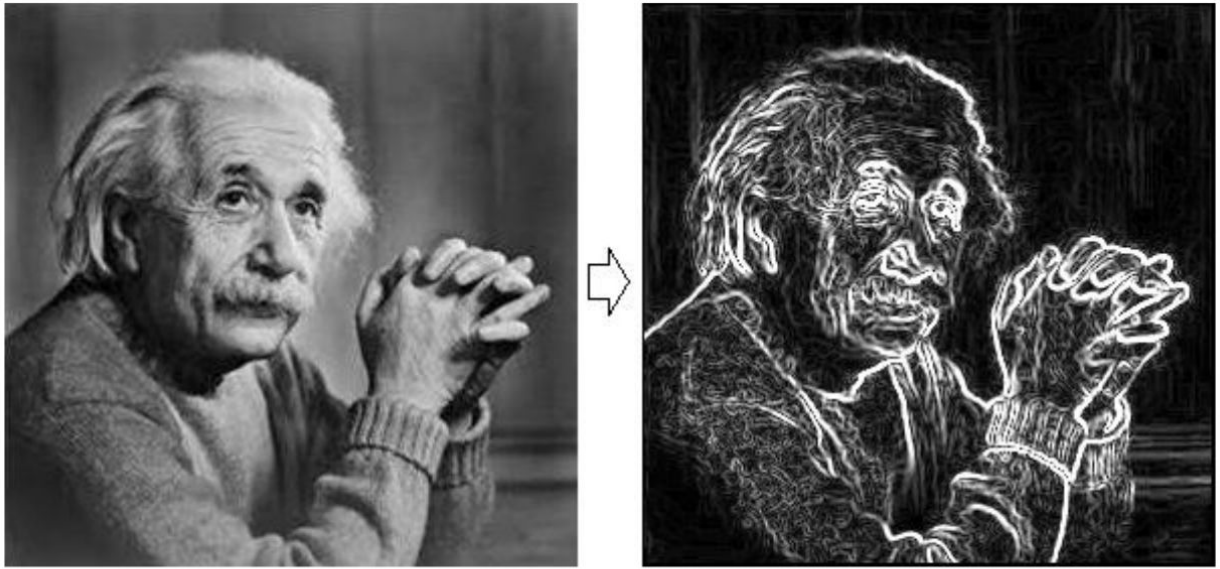
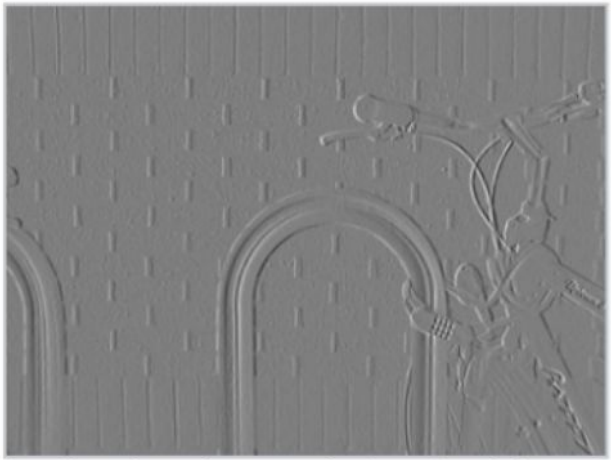


Figure: Sobel-filtered image.

Outlook: How do different layers work?
Convolutional Layers

Input



Vertical edges detected by S_x

Outlook: How do different layers work?
Convolutional Layers

Figure: Sobel-filtered image.

Sobel showed that the gradient image $\mathbf{G}_x$ of original image $\mathbf{A}$ in x-dimension can be approximated by

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A} = \mathbf{S}_x * \mathbf{A}$$

where $*$ indicates a mathematical operation known as a convolution, not a traditional matrix multiplication

Outlook: How do different layers work?
Convolutional Layers

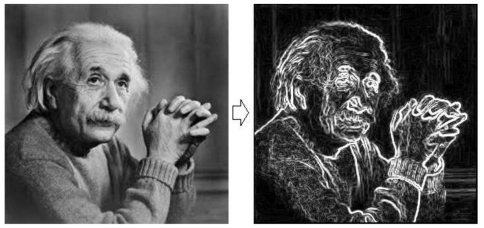


Figure: Sobel-filtered image.

Sobel showed that the gradient image $\mathbf{G}_x$ of original image $\mathbf{A}$ in x-dimension can be approximated by

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A} = \mathbf{S}_x * \mathbf{A}$$

the filter

where $*$ indicates a mathematical operation known as a convolution, not a traditional matrix multiplication

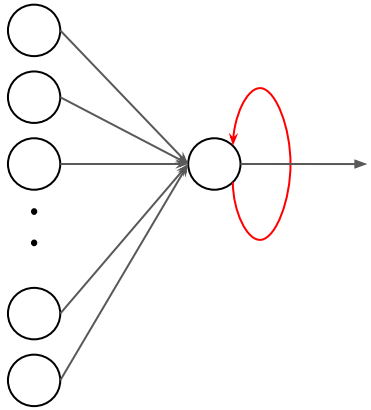
Outlook: How do different layers work?

Convolutional Layers

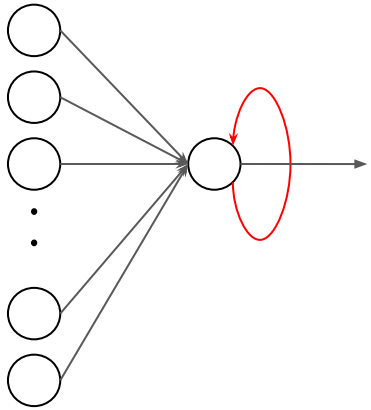
Source: medium.com

Outlook: How do different layers work?

Recurrent Neural Network Layers

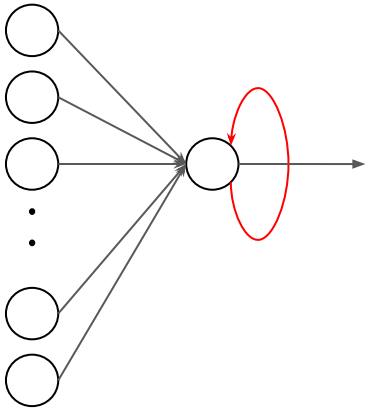


Outlook: How do different layers work?
Recurrent Neural Network Layers



Usually for sequential data

Outlook: How do different layers work?
Recurrent Neural Network Layers

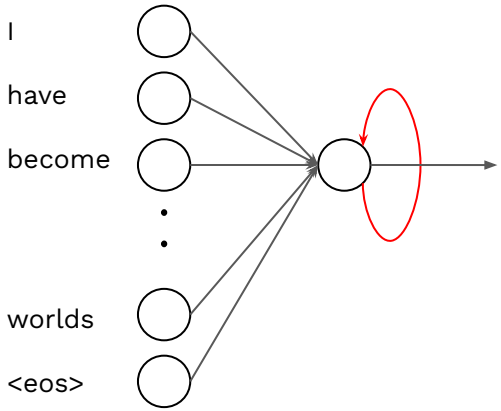


Usually for sequential data

Here: Text sentence

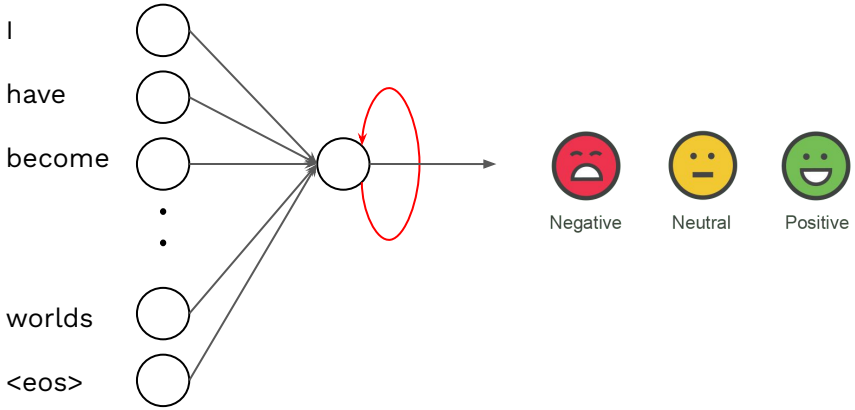
Outlook: How do different layers work?

Recurrent Neural Network Layers



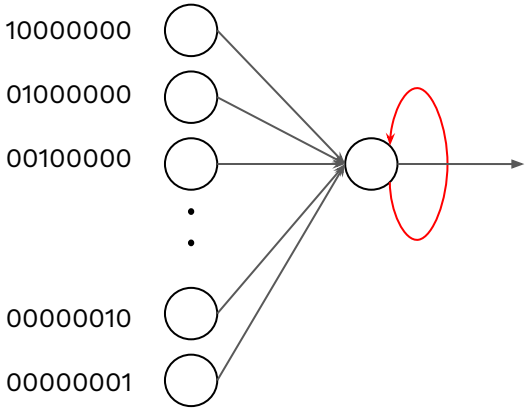
Outlook: How do different layers work?

Recurrent Neural Network Layers



Outlook: How do different layers work?

Recurrent Neural Network Layers

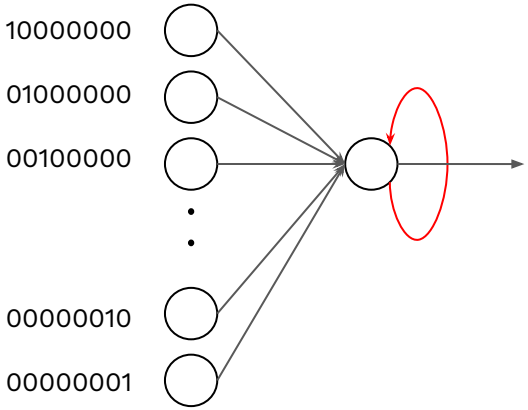


After binary text encoding*,
we can feed the text into a network

*there are better ways to encode or tokenize text

Outlook: How do different layers work?

Recurrent Neural Network Layers

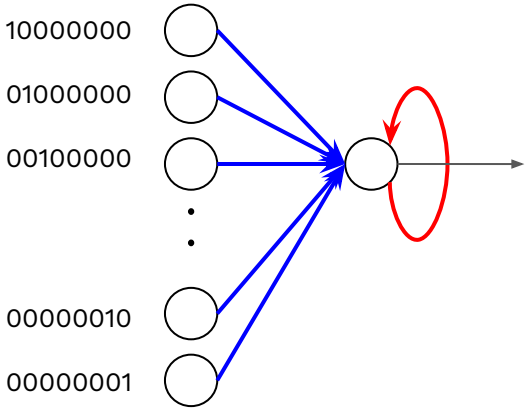


After binary text encoding*,
we can feed the text into a network

Keyword: Weight sharing

Outlook: How do different layers work?

Recurrent Neural Network Layers

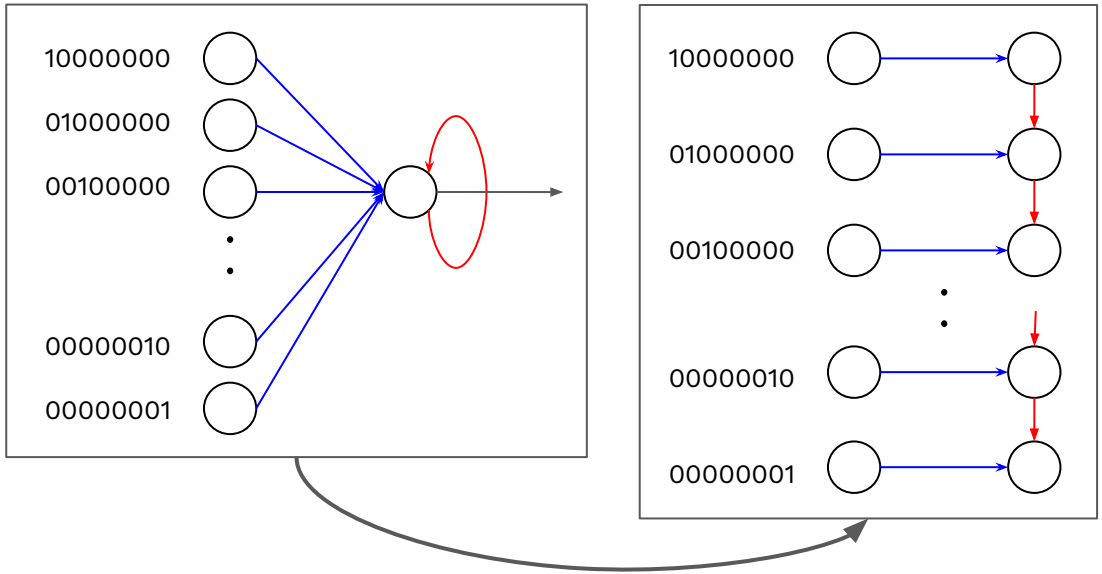


Blue connections (weights) are shared, same for red connection

Keyword: Weight sharing

Outlook: How do different layers work?

Recurrent Neural Network Layers

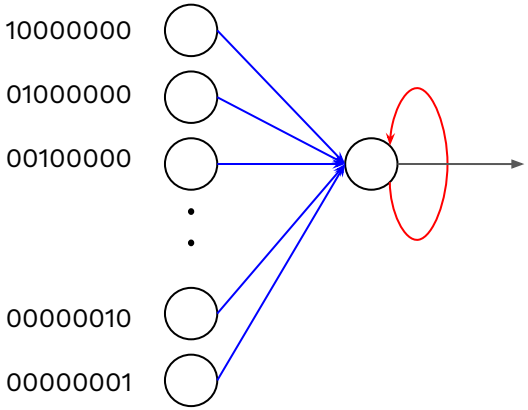


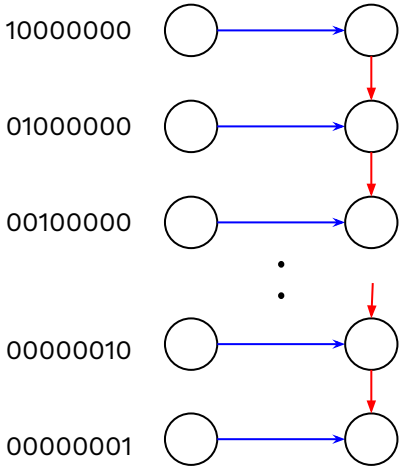
unroll network

Keyword: Weight sharing

Outlook: How do different layers work?

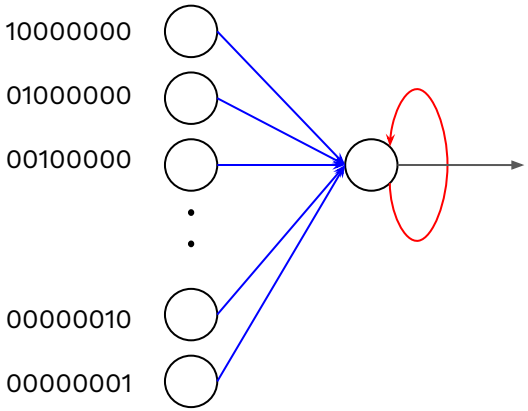
Recurrent Neural Network Layers

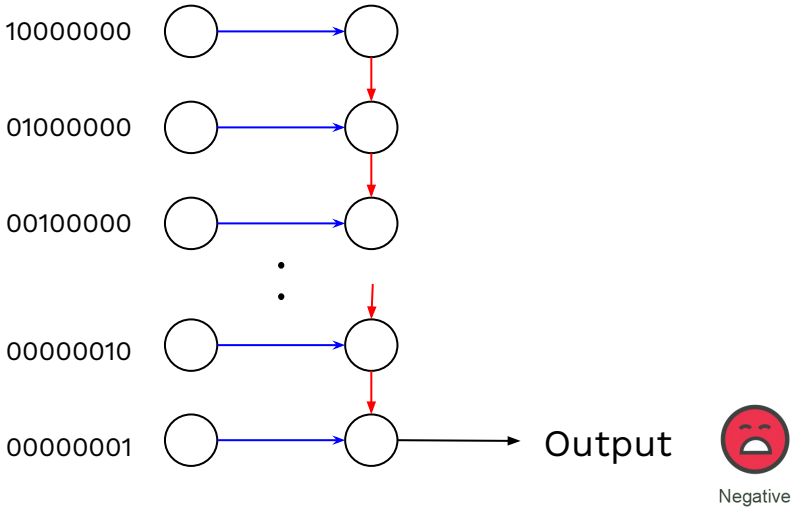




Outlook: How do different layers work?

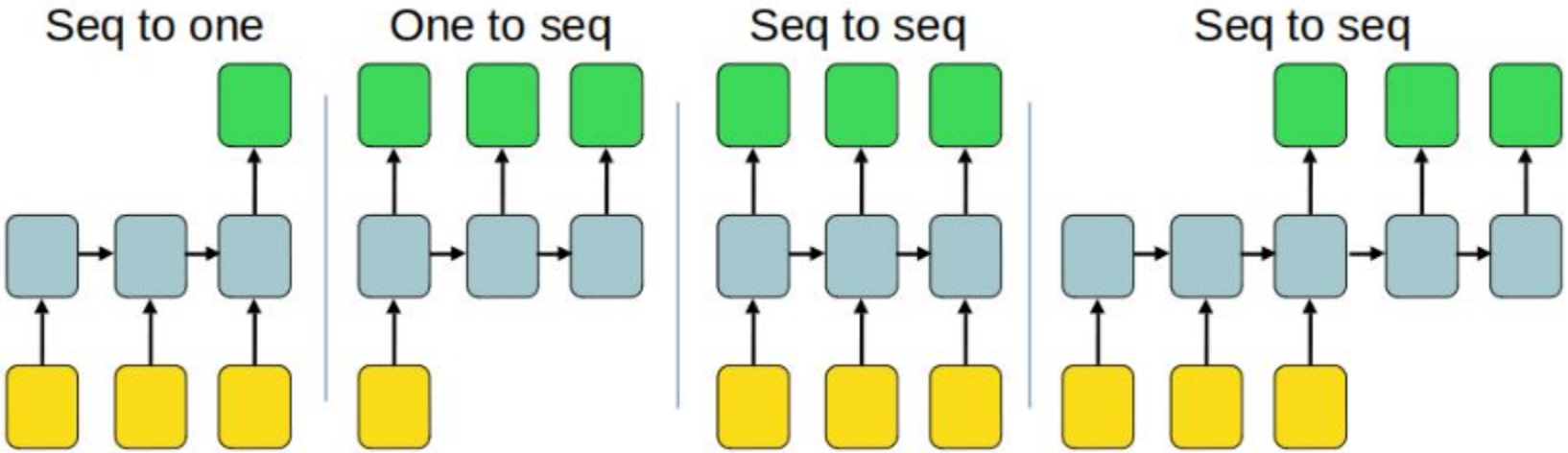
Recurrent Neural Network Layers





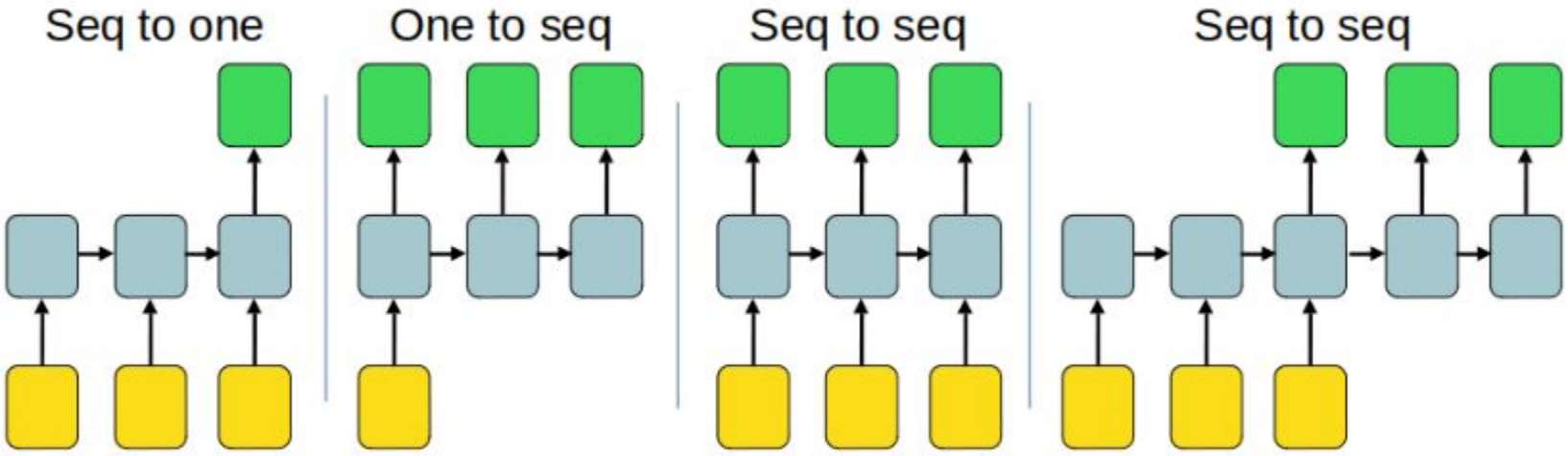
Outlook: How do different layers work?

Recurrent Neural Network Layers



Outlook: How do different layers work?

Recurrent Neural Network Layers



Munich Center for Machine Learning

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Thank you for your attention.

mcml.ai

Munich Center for Machine Learning

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Next steps:

- Learn about theory ⇒ Deep Learning Book
- Learn software ⇒ Dive into Deep Learning (Python only)
- deepregression ⇒ Tutorial Paper

References

[1] Well-tuned Simple Nets Excel on Tabular Datasets https://proceedings.neurips.cc/paper_files/paper/2021/hash/c902b497eb972281fb5b4e206db38ee6-Abstract.html

[2] TabNet: Attentive Interpretable Tabular Learning https://ojs.aaai.org/index.php/AAAI/article/view/16826

[3] Why do tree-based models still outperform deep learning on typical tabular data? https://openreview.net/forum?id=Fp7__phQszn

[4] Loss Surfaces, Mode Connectivity, and Fast Ensembling of DNNs https://proceedings.neurips.cc/paper_files/paper/2018/hash/be3087e74e9100d4bc4c6268cdbe8456-Abstract.html

[5] Cascaded Latent Diffusion Models for High-Resolution Chest X-ray Synthesis https://link.springer.com/chapter/10.1007/978-3-031-33380-4_14

[6] Adversarial Random Forests for Density Estimation and Generative Modeling https://proceedings.mlr.press/v206/watson23a.html

[7] Towards Efficient MCMC Sampling in Bayesian Neural Networks by Exploiting Symmetry https://www.springerprofessional.de/towards-efficient-mcmc-sampling-in-bayesian-neural-networks-by-e/26051704

[8] Implicit Regularization Leads to Benign Overfitting for Sparse Linear Regression https://proceedings.mlr.press/v202/zhou23d

[9] Mini-batch optimization enables training of ODE models on large-scale datasets https://www.nature.com/articles/s41467-021-27374-6

[10] deepregression: A Flexible Neural Network Framework for Semi-Structured Deep Distributional Regression https://www.jstatsoft.org/article/view/v105i02

[11] Smoothing the Edges: A General Framework for Smooth Optimization in Sparse Regularization using Hadamard Overparametrization https://arxiv.org/abs/2307.03571

[12] Semi-Structured Distributional Regression https://www.tandfonline.com/doi/abs/10.1080/00031305.2022.2164054

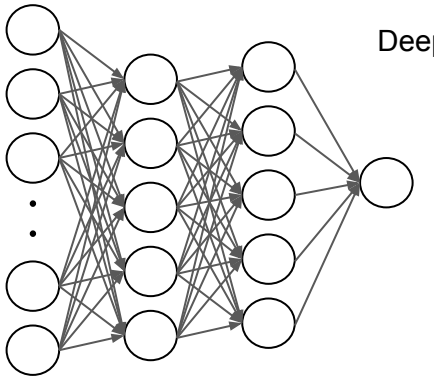
[13] A New PHO-rmula for Improved Performance of Semi-Structured Networks https://proceedings.mlr.press/v202/rugamer23a.html

Backup

Distributional Regression in Neural Networks

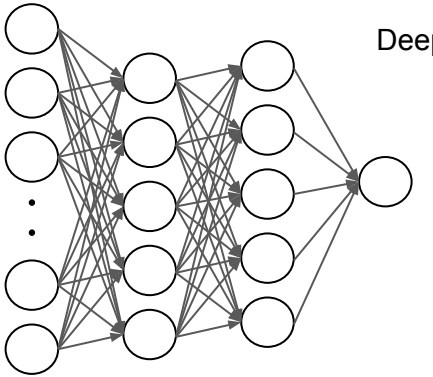
LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
MCML

Semi-Structured Neural Networks

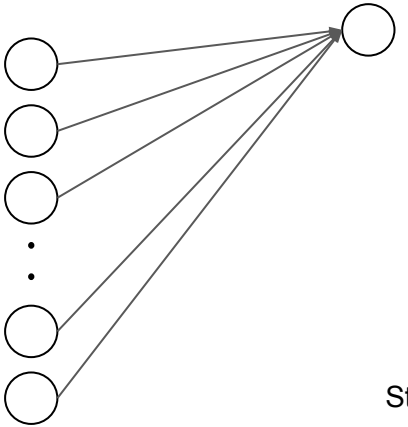


Deep Neural Network

Semi-Structured Neural Networks

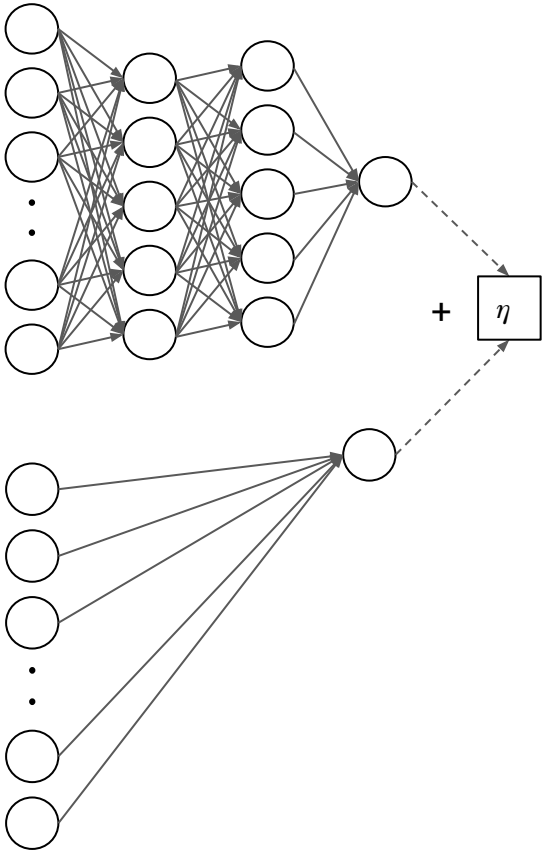


Deep Neural Network

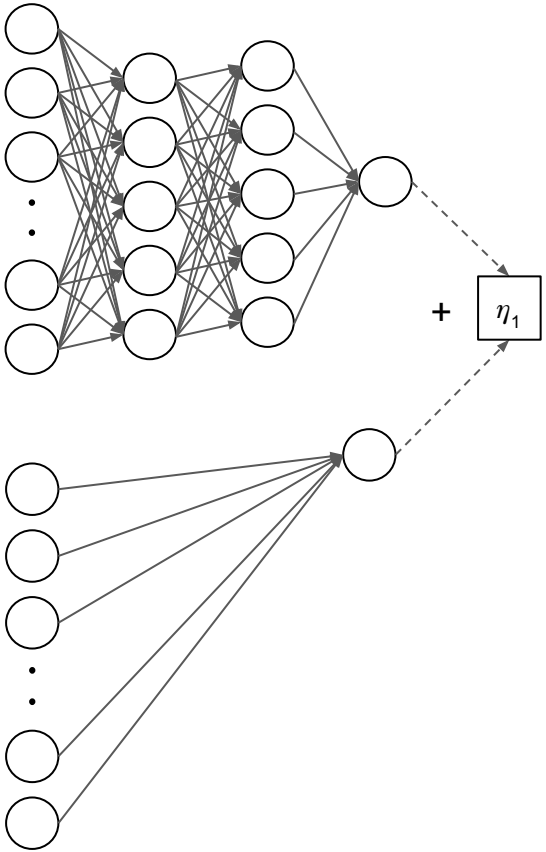


Structured Predictor

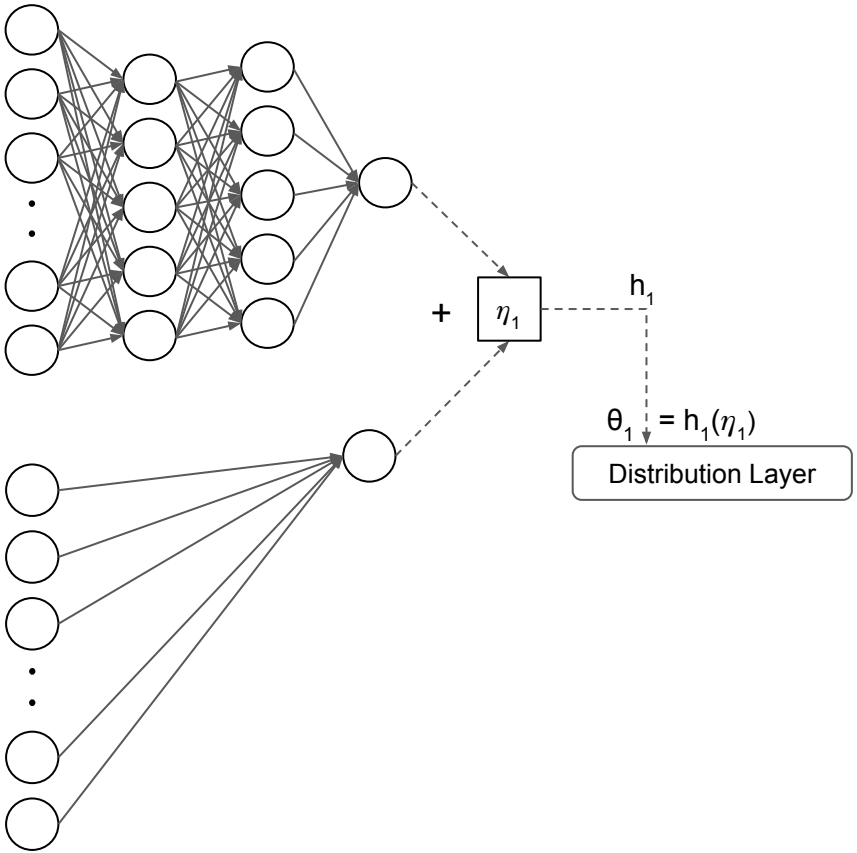
Semi-Structured Neural Networks



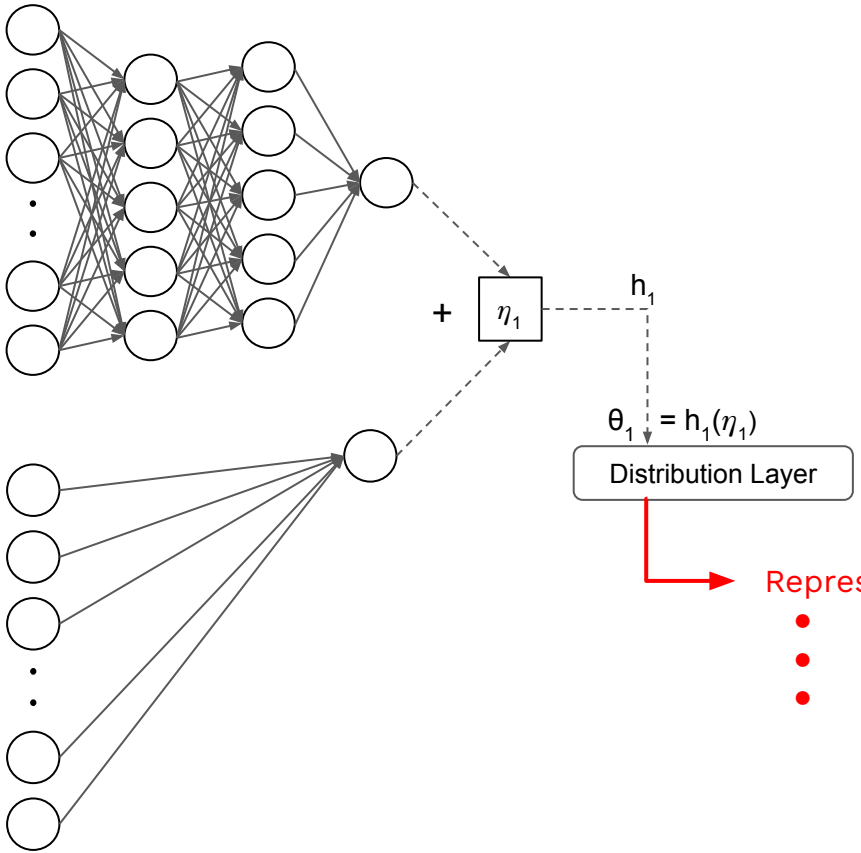
Semi-Structured Neural Networks



Semi-Structured Neural Networks



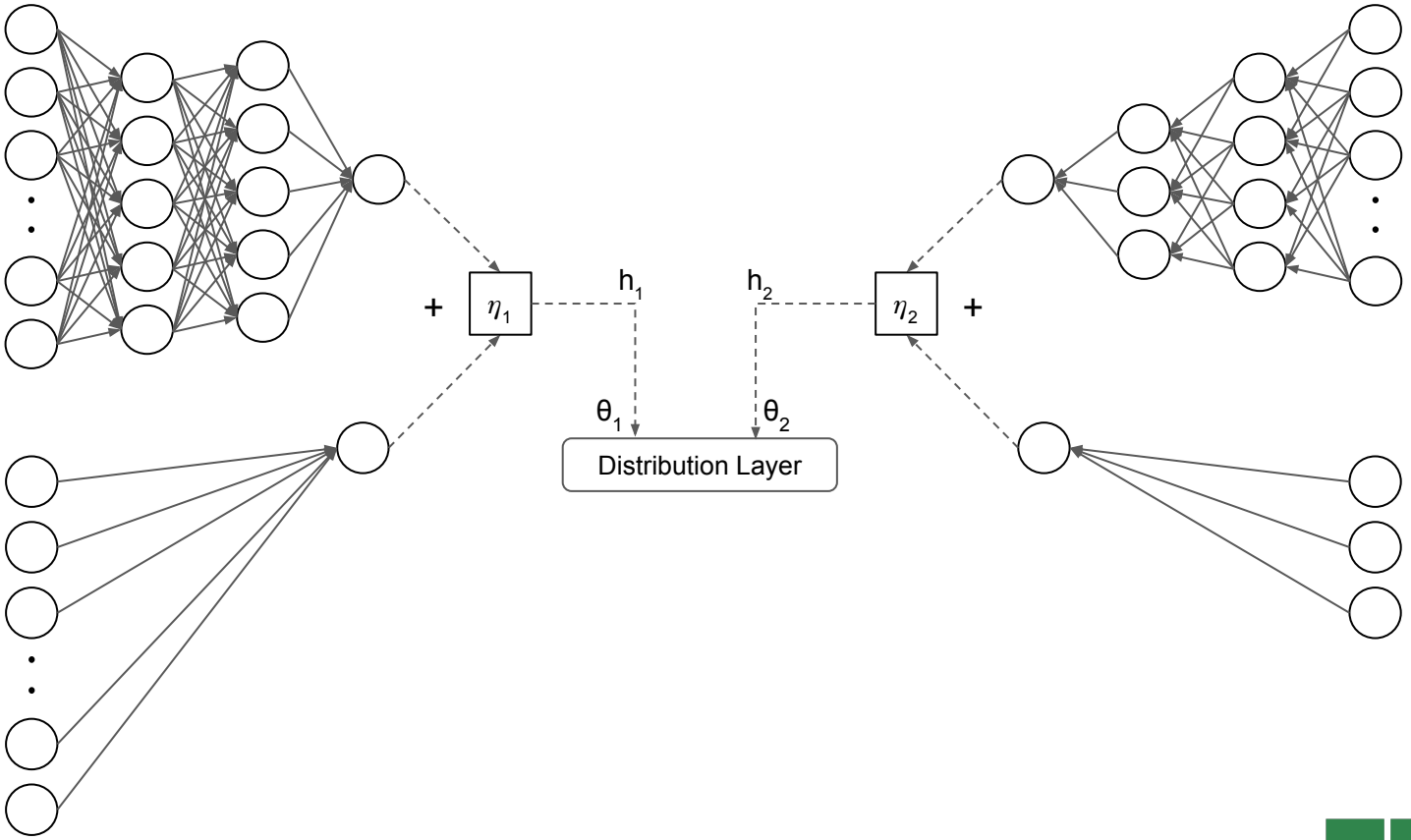
Semi-Structured Neural Networks



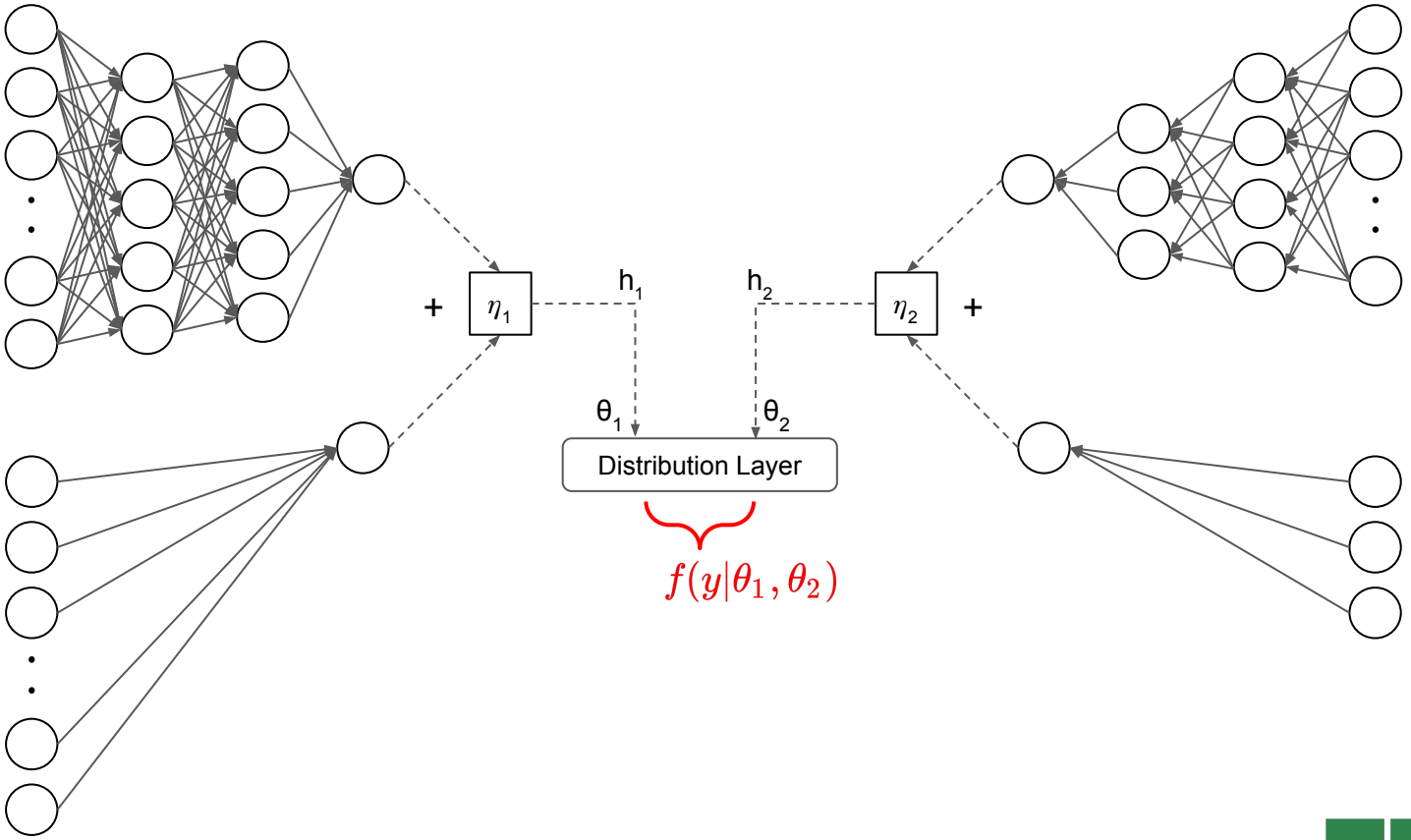
Representation of a pre-defined distribution

- Compute quantiles
- Sample
- Compute density

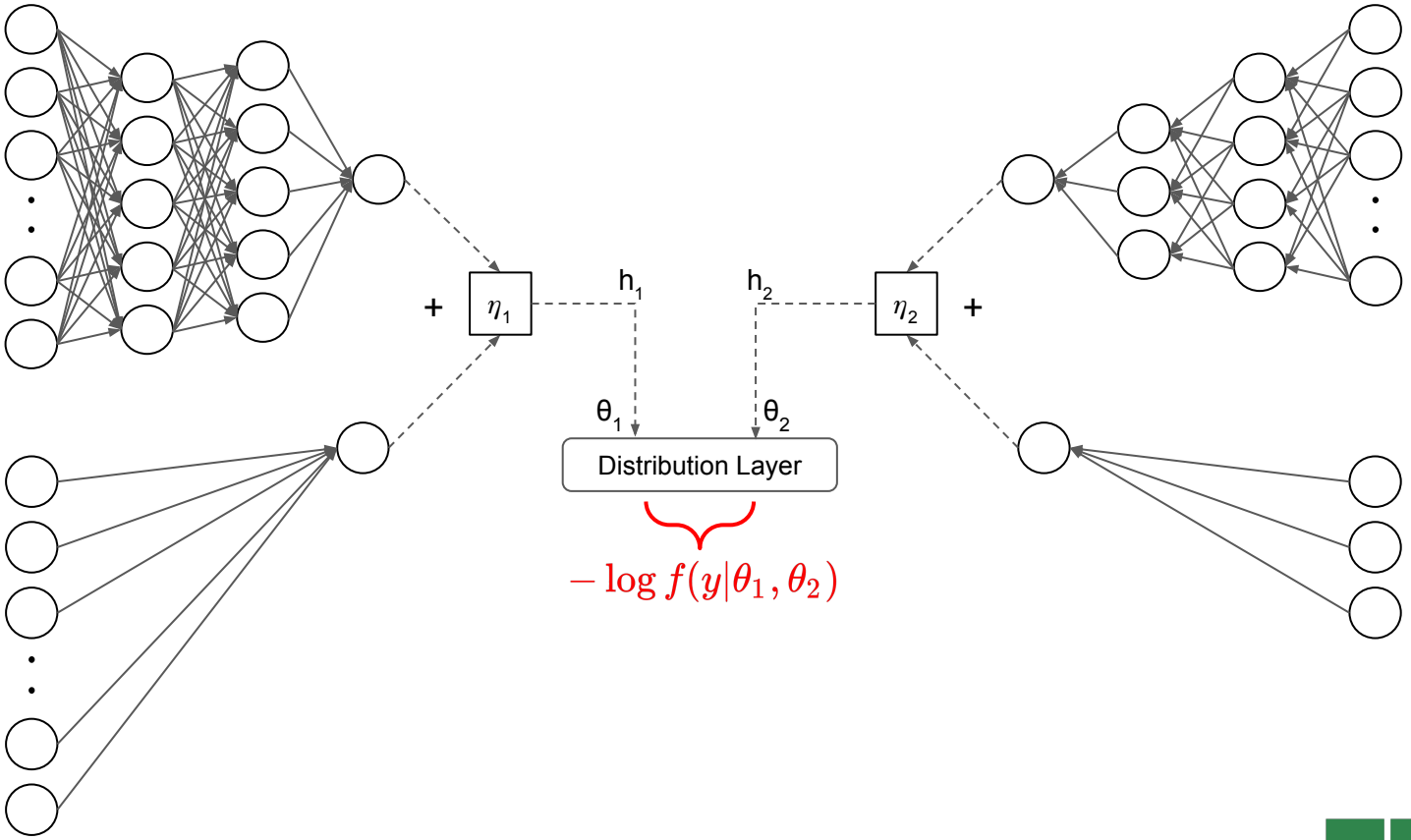
Semi-Structured Neural Networks



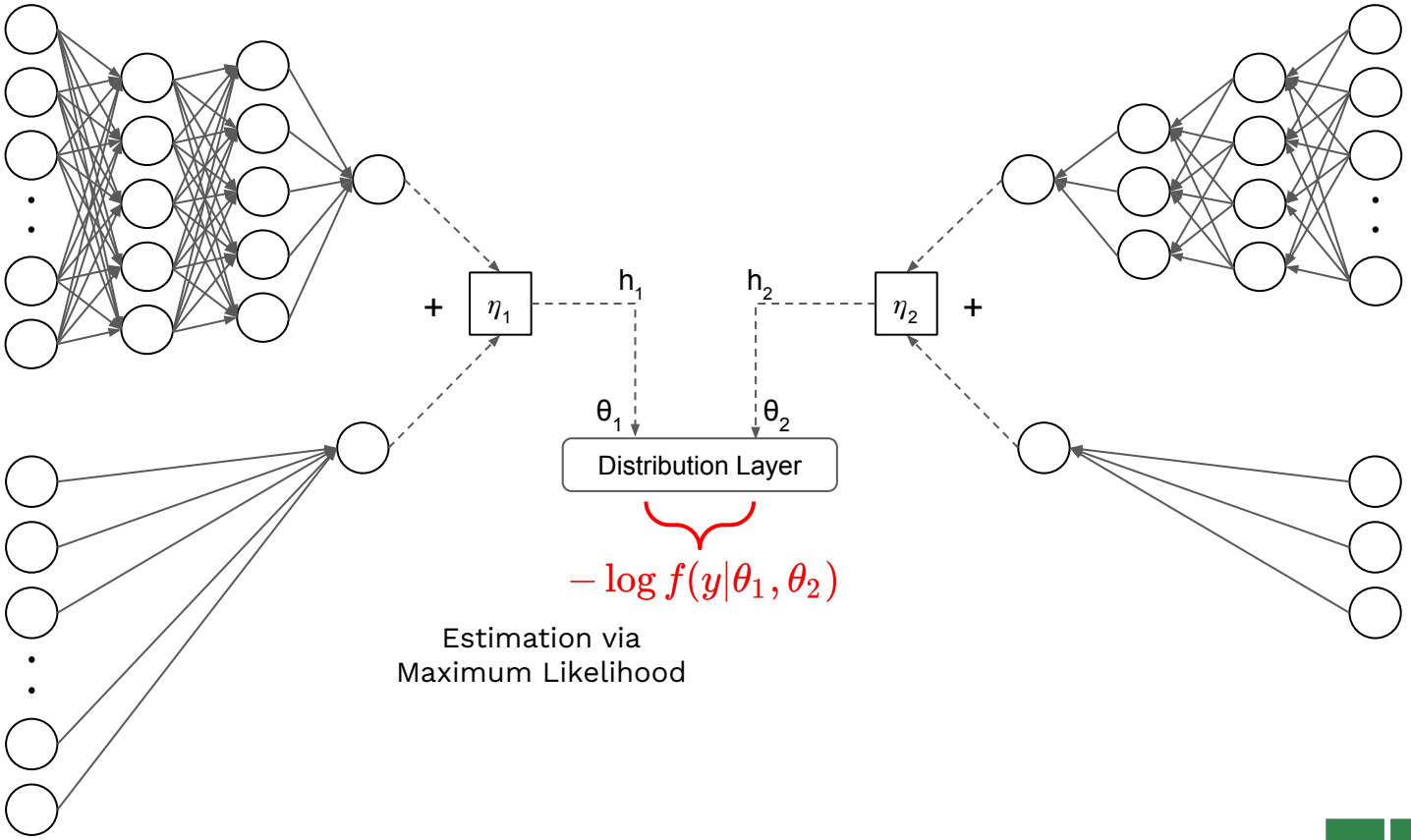
Semi-Structured Neural Networks



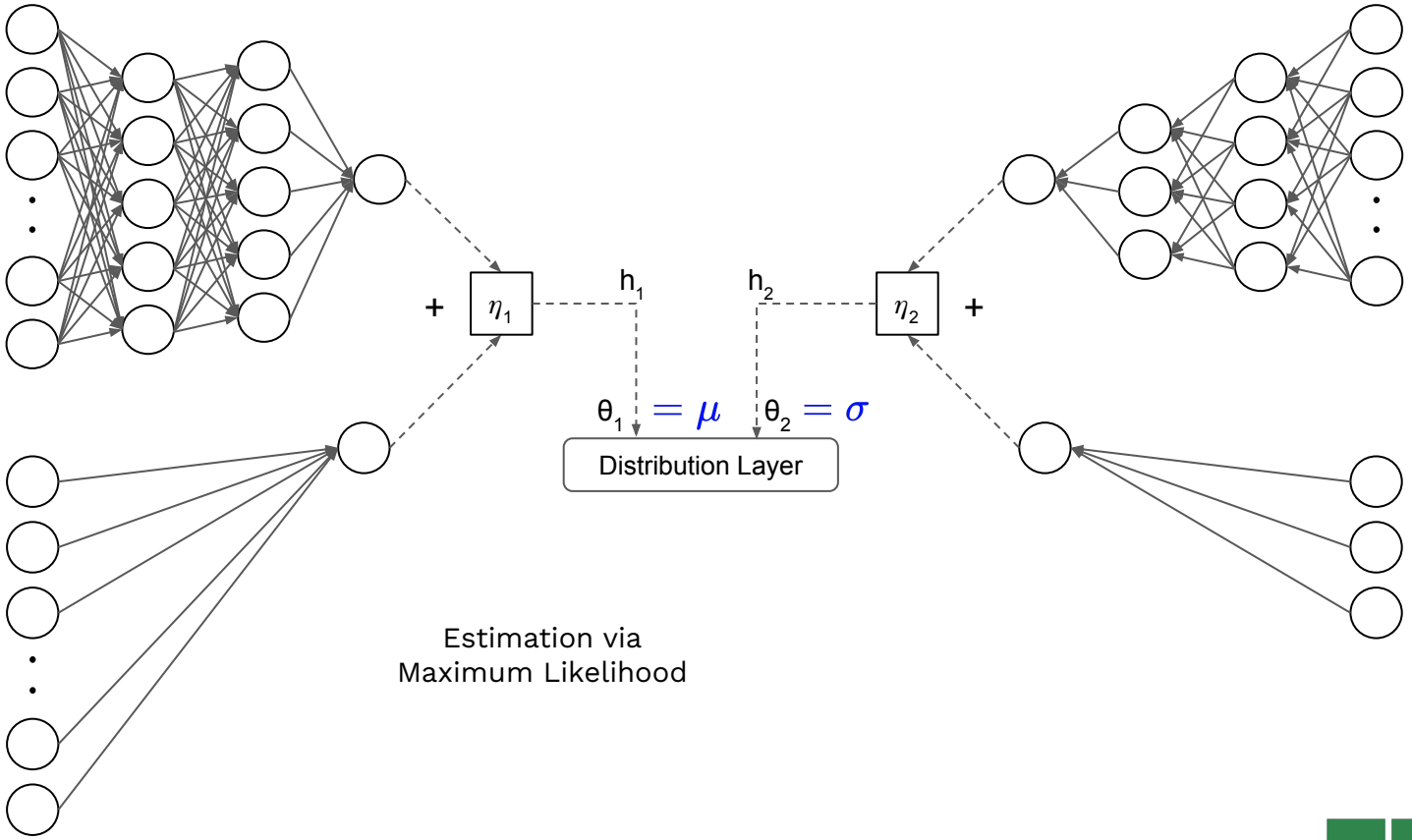
Semi-Structured Neural Networks



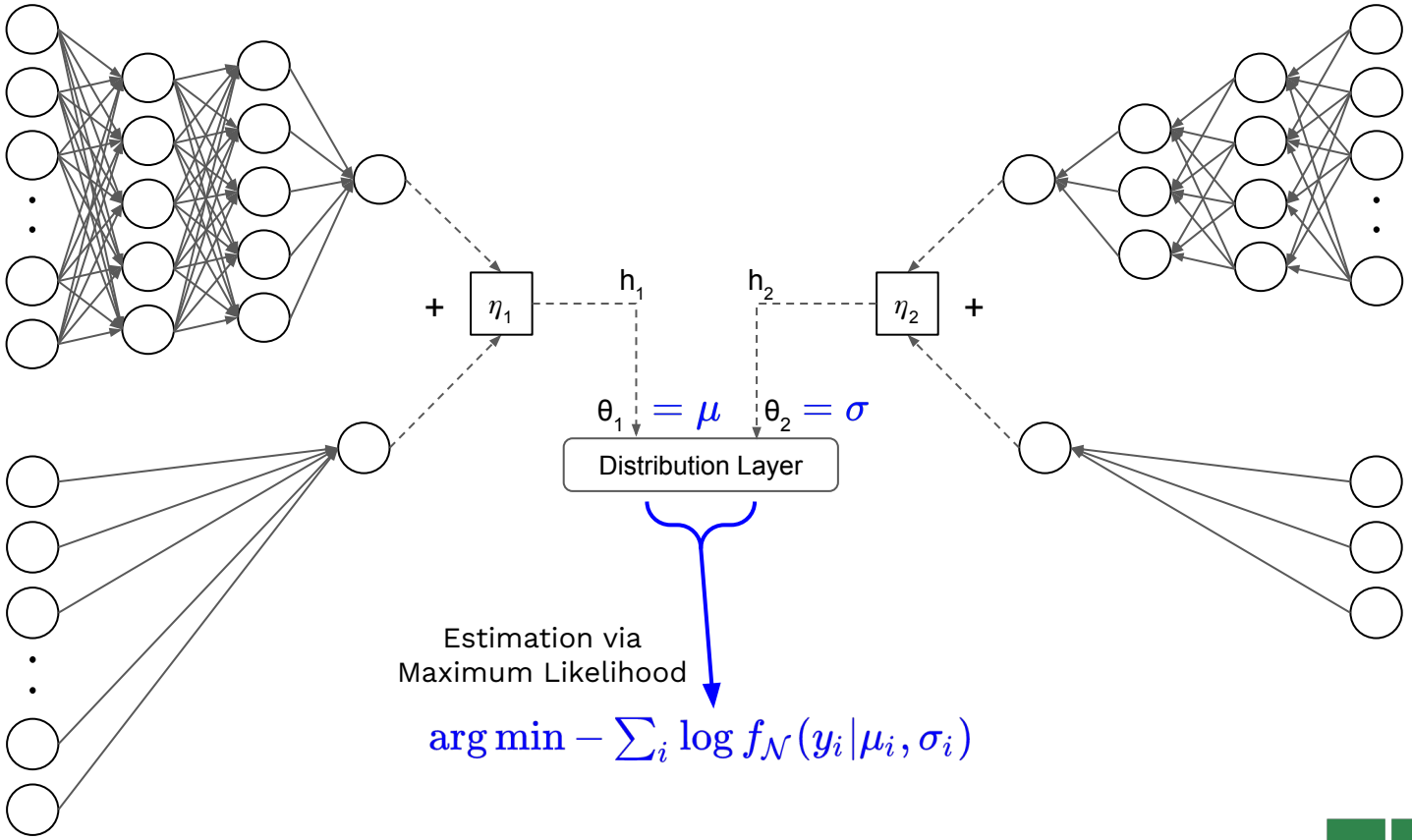
Semi-Structured Neural Networks



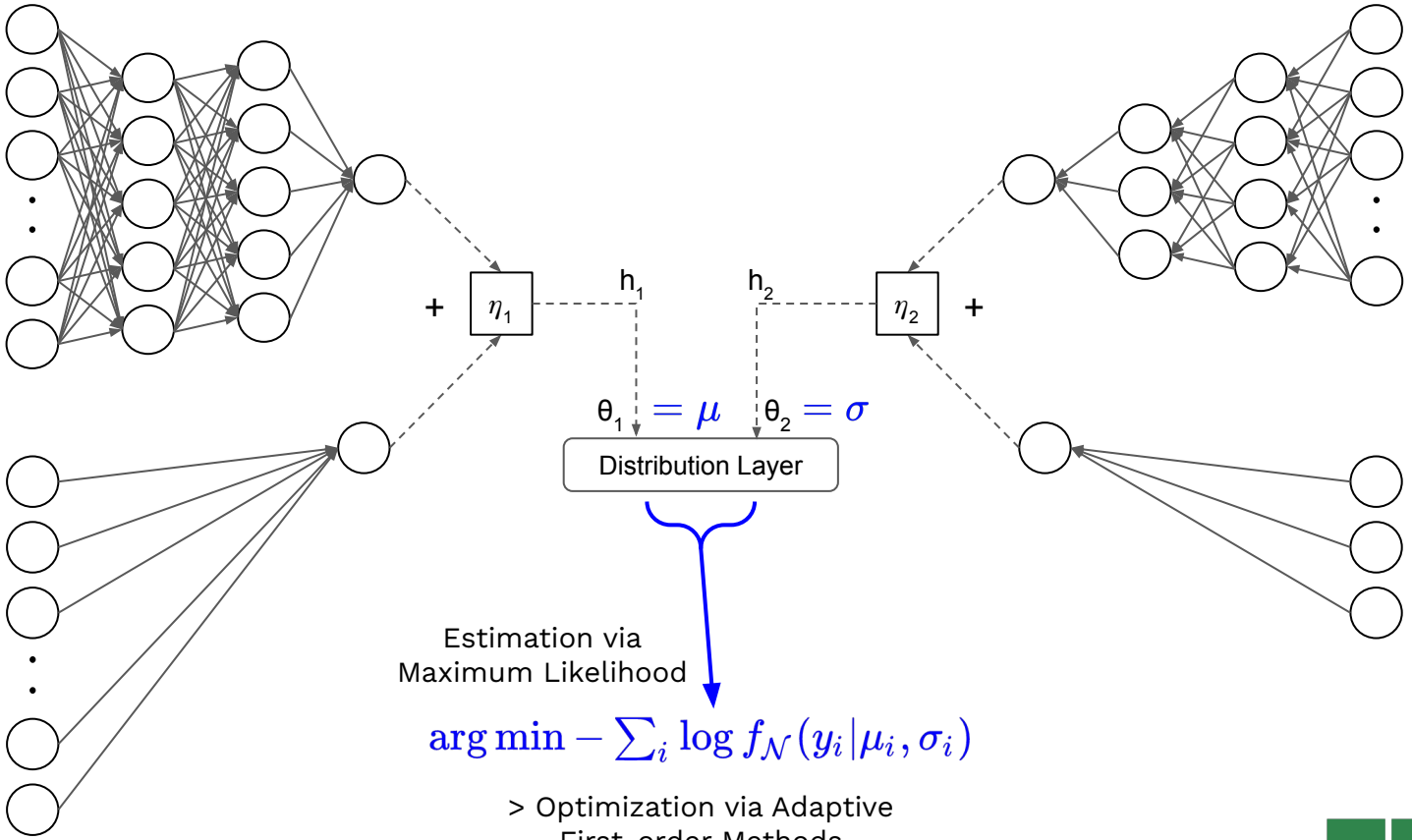
Semi-Structured Neural Networks



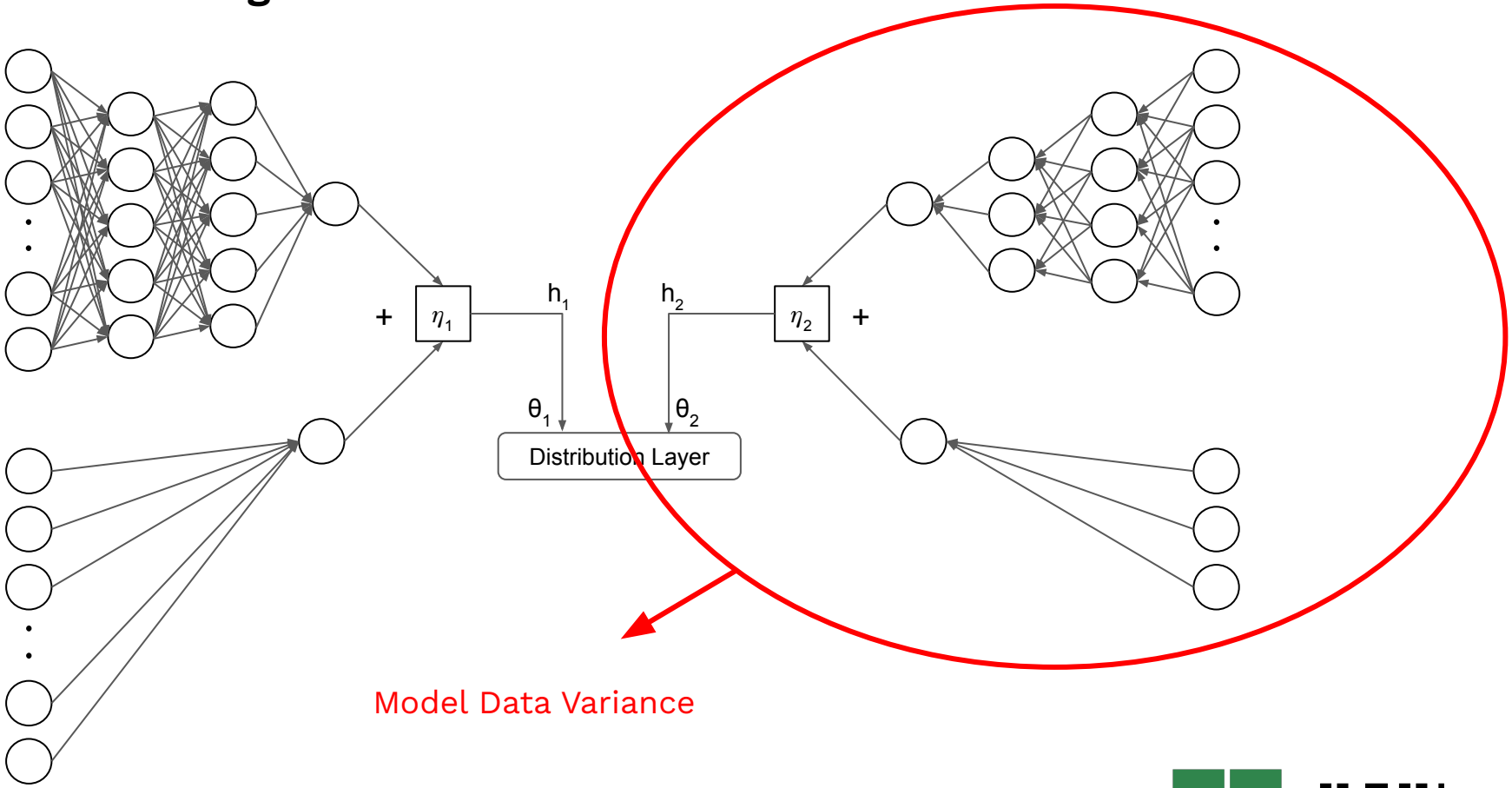
Semi-Structured Neural Networks



Semi-Structured Neural Networks



Distributional Regression – Model the Variance



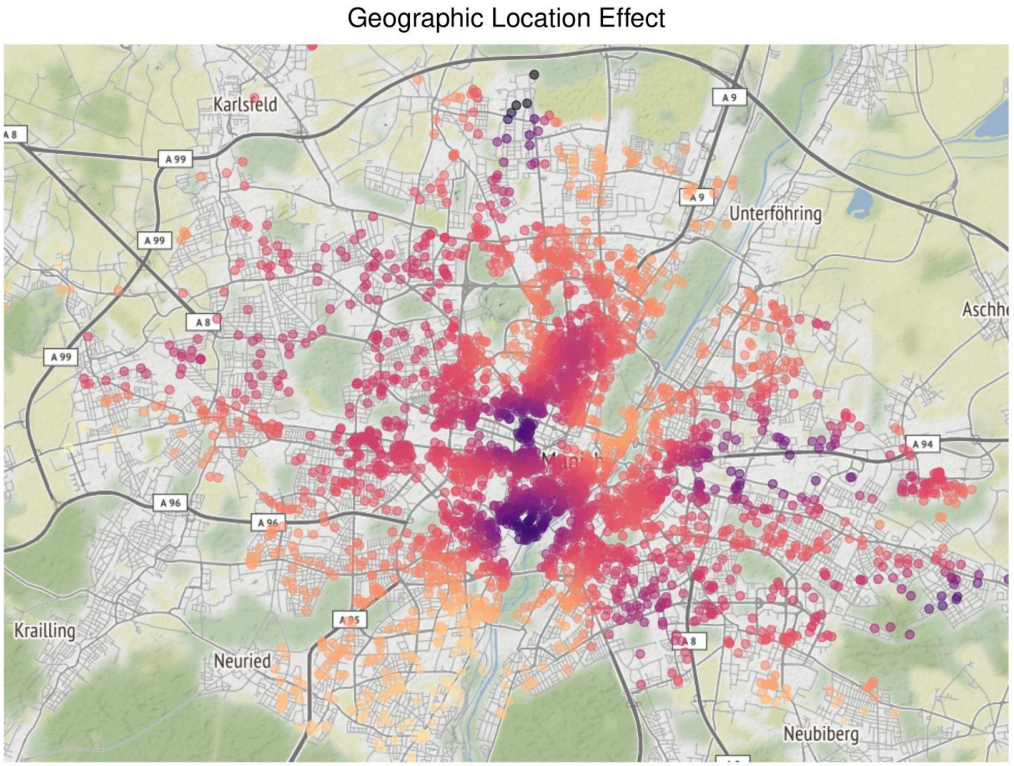
Example: Predicting Airbnb Prices in Munich

```
mod_airbnb <- deepregression(
    y = price,
    family = "log_normal",
    list_of_formulas = list(
      location = ~1 + te(latitude, longitude) + room_type + bedrooms +
                  cnn(image) + lstm(desc),
      scale = ~1
    ),
    data = d_train
)
```

Example: Predicting Airbnb Prices in Munich

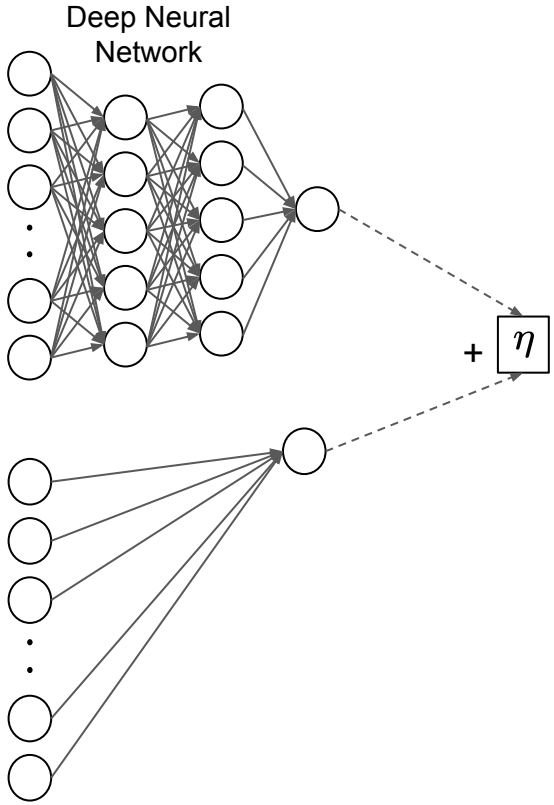
```
mod_airbnb <- deepregression(
    y = price,
    family = "log_normal",
    list_of_formulas = list(
      location = ~1 + te(latitude, longitude) + room_type + bedrooms +
                  cnn(image) + lstm(desc),
      scale = ~1 + te(latitude, longitude)
    ),
    data = d_train
)
```

Example: Predicting Airbnb Prices in Munich

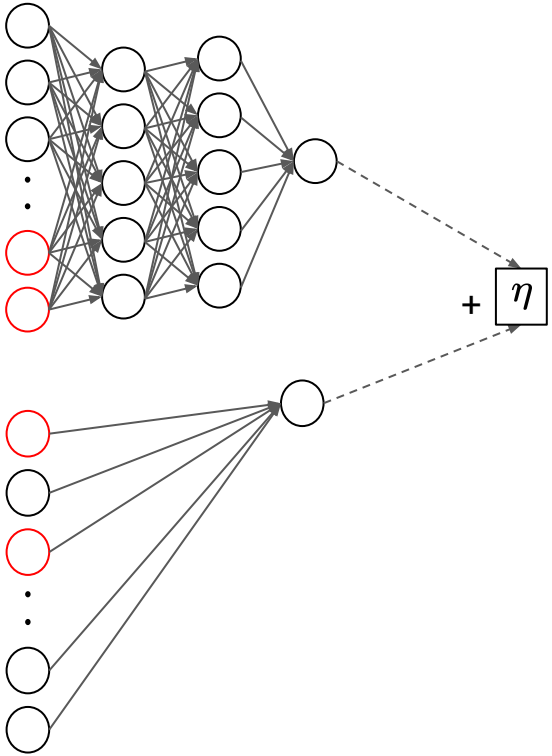


Identifiability

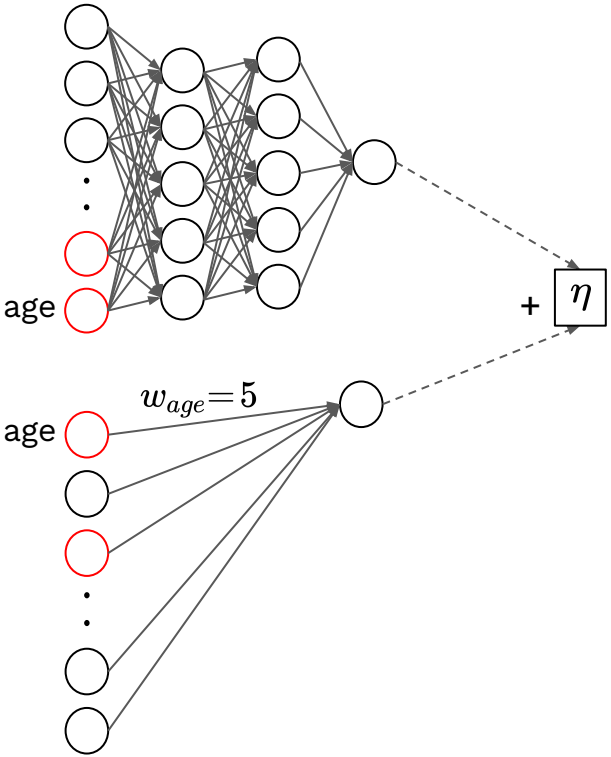
Non-Identifiability <-> Interpretability



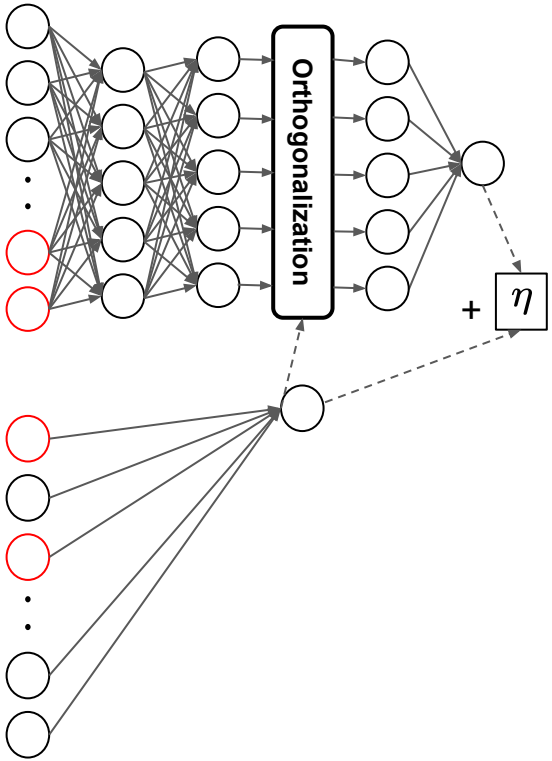
Non-Identifiability <-> Interpretability



Non-Identifiability <-> Interpretability

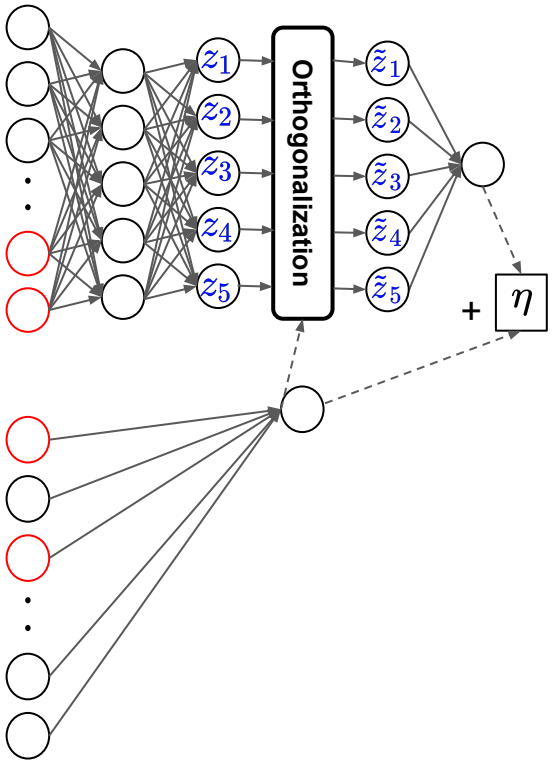


Non-Identifiability <-> Interpretability



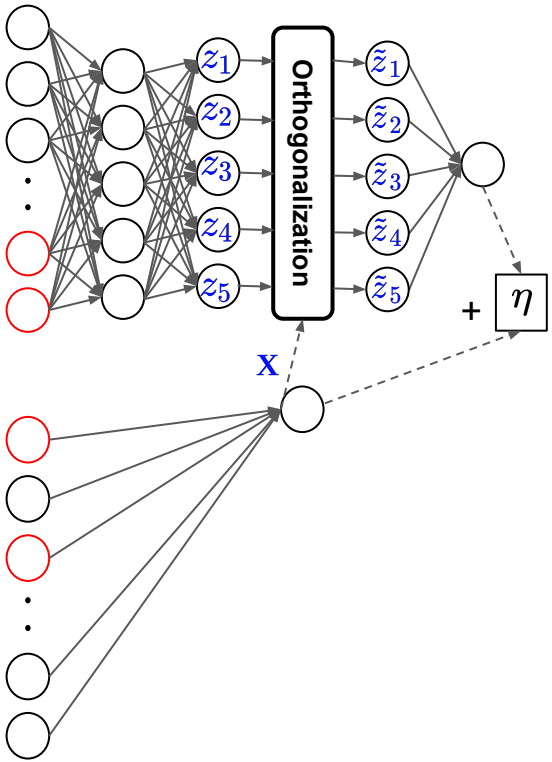
Rügamer et al., 2023,
The American Statistician

Non-Identifiability <-> Interpretability



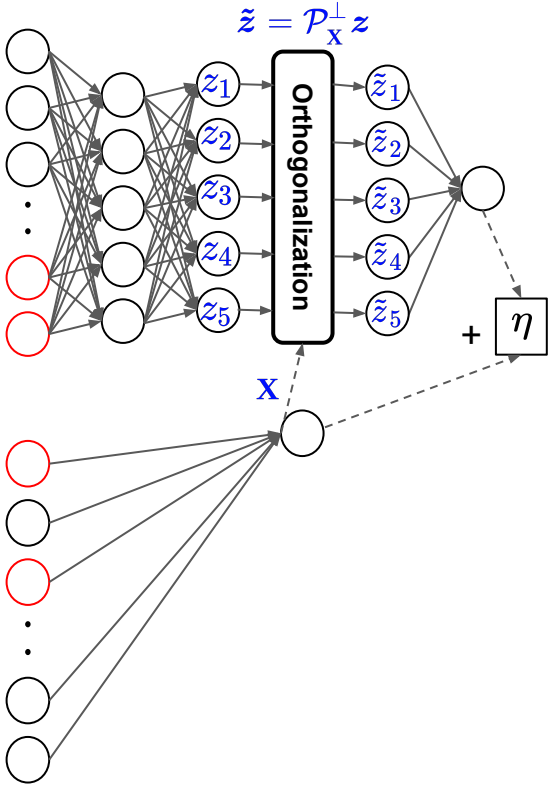
[12] Rügamer et al., 2023,
The American Statistician

Non-Identifiability <-> Interpretability



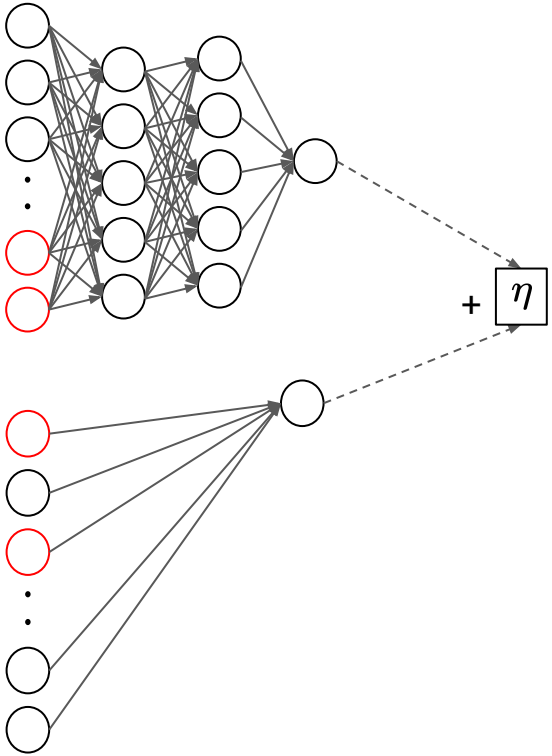
[12] Rügamer et al., 2023,
The American Statistician

Non-Identifiability <-> Interpretability



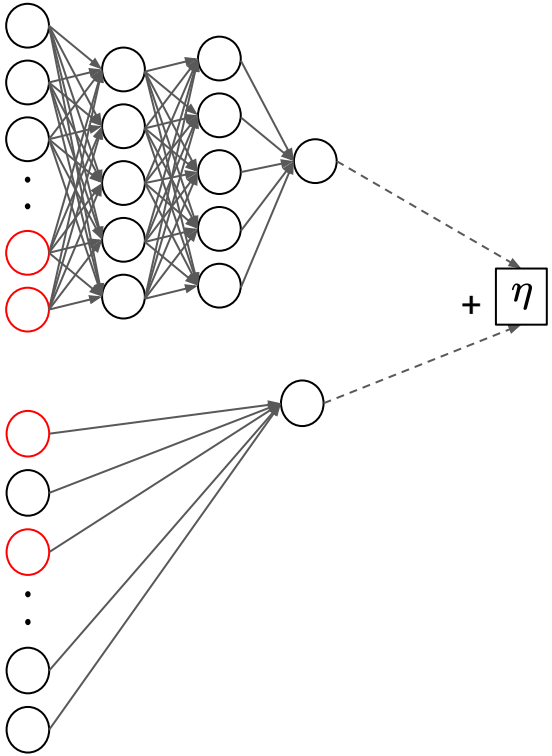
[12] Rügamer et al., 2023, *The American Statistician*

Non-Identifiability <-> Interpretability



Post-hoc Orthogonalization
[13] Rügamer 2023, *ICML*

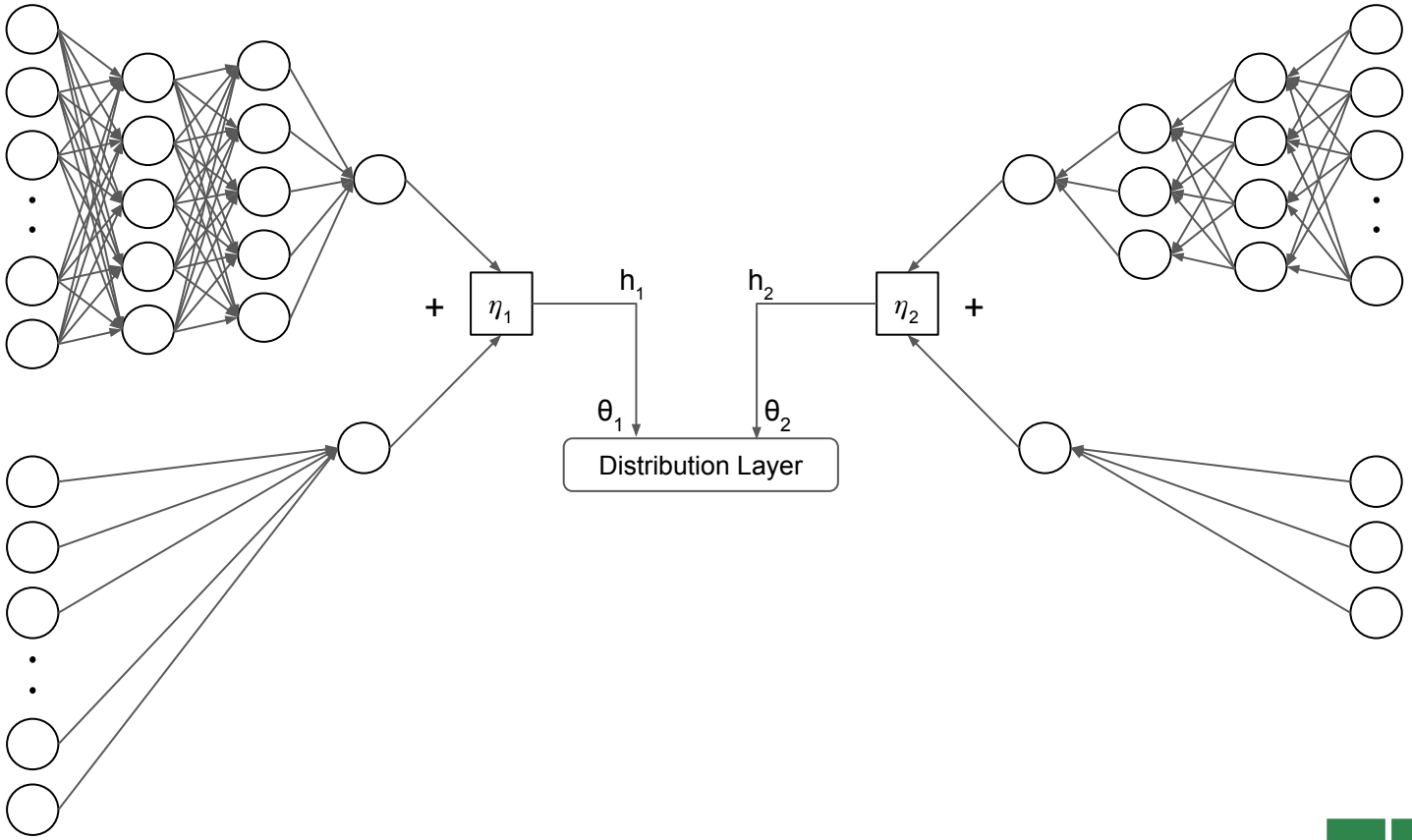
Non-Identifiability <-> Interpretability



Post-hoc Orthogonalization
[13] Rügamer 2023, *ICML*

- Fit network w/o constraints
- Orthogonalize after
- At costs of 1 forward pass
- More flexible
- Similar to functional ANOVA (Hooker)

Distributional Regression – Model the Variance



Expert Part:
Sparsity in Neural Networks

Expert Part: How can I obtain sparsity (in multimodal settings)?

Expert Part: How can I obtain sparsity (in multimodal settings)?

⇒ [11] **Sparse Modality Regression (Best Paper IWSM 2023)**

Why is sparse regularization not more common in DL?

- DL models are almost exclusively optimized "heuristically" using stoch. gradient descent (SGD) and variations (e.g., Adam)

Why is sparse regularization not more common in DL?

- DL models are almost exclusively optimized "heuristically" using stoch. gradient descent (SGD) and variations (e.g., Adam)
- Works well for non-convex objectives, but not for sparsity-inducing penalties

Why is sparse regularization not more common in DL?

- DL models are almost exclusively optimized "heuristically" using stoch. gradient descent (SGD) and variations (e.g., Adam)
- Works well for non-convex objectives, but not for sparsity-inducing penalties

  ⇒ non-smoothness causes slow convergence and oscillating parameter updates

Sparsity in Neural Networks – Starting point

Lasso:

Sparsity in Neural Networks – Starting point

Lasso:

$$\frac{1}{n}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^2 + \lambda||\boldsymbol{\beta}||_1$$

Sparsity in Neural Networks – Starting point

Lasso:

$$\underbrace{\frac{1}{n}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^2}_{\text{cont. + convex}} + \underbrace{\lambda||\boldsymbol{\beta}||_1}_{\text{cont. + convex}}$$

Sparsity in Neural Networks – Starting point

Lasso:

$$\underbrace{\frac{1}{n}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^2}_{\text{cont. + convex}} + \underbrace{\lambda||\boldsymbol{\beta}||_1}_{\substack{\text{cont. + convex} \\ \textbf{but non-smooth}}}$$

Sparsity in Neural Networks – Starting point

Lasso:

$$\frac{1}{n}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^2 + \lambda||\boldsymbol{\beta}||_1$$

⇒ Convex and continuous, but non-smooth

Sparsity in Neural Networks – Starting point

Lasso:

$$\frac{1}{n}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^2 + \lambda||\boldsymbol{\beta}||_1$$

⇒ Convex and continuous, but non-smooth

problem for Deep Learning (DL)

Why is sparse regularization not more common in DL?

Why is sparse regularization not more common in DL?

- DL models are almost exclusively optimized "heuristically" using stoch. gradient descent (SGD) and variations (e.g., Adam)

Why is sparse regularization not more common in DL?

- DL models are almost exclusively optimized "heuristically" using stoch. gradient descent (SGD) and variations (e.g., Adam)
- Works well for non-convex objectives, but not for sparsity-inducing penalties

Why is sparse regularization not more common in DL?

- DL models are almost exclusively optimized "heuristically" using stoch. gradient descent (SGD) and variations (e.g., Adam)
- Works well for non-convex objectives, but not for sparsity-inducing penalties

  ⇒ non-smoothness causes slow convergence and oscillating parameter updates

Our proposal: Hadamard product parameterization

- = Continuously differentiable surrogate penalty
- Smooth ⇒ off-the-shelf SGD

Hadamard Product Parameterization

for Lasso

$$\frac{1}{n}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^2 + \lambda||\boldsymbol{\beta}||_1$$

Hadamard Product Parameterization

for Lasso

$$\frac{1}{n}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^2 + \lambda||\boldsymbol{\beta}||_1$$

- Parametrize $\boldsymbol{\beta} = \boldsymbol{u} \odot \boldsymbol{v}$

Hadamard Product Parameterization

for Lasso

$$\frac{1}{n}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^2 + \lambda||\boldsymbol{\beta}||_1$$

- Parametrize $\boldsymbol{\beta} = \boldsymbol{u} \odot \boldsymbol{v}$ → $\frac{1}{n}(\boldsymbol{y} - \boldsymbol{X}(\boldsymbol{u} \odot \boldsymbol{v}))^2 + \lambda||\boldsymbol{\beta}||_1$

Hadamard Product Parameterization

for Lasso

$$\frac{1}{n}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^2 + \lambda||\boldsymbol{\beta}||_1$$

- Parametrize $\boldsymbol{\beta} = \boldsymbol{u} \odot \boldsymbol{v}$
- Replace non-smooth $||\boldsymbol{\beta}||_1$

  by smooth $||\boldsymbol{u}||_2^2 + ||\boldsymbol{v}||_2^2$

$$\frac{1}{n}(\boldsymbol{y} - \boldsymbol{X}(\boldsymbol{u} \odot \boldsymbol{v}))^2 + \lambda||\boldsymbol{\beta}||_1$$

Hadamard Product Parameterization

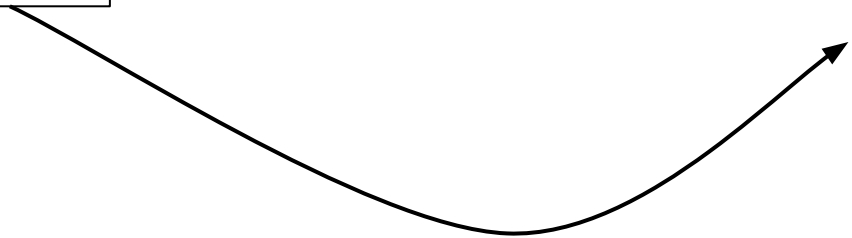
for Lasso

- Parametrize $\boldsymbol{\beta} = \boldsymbol{u} \odot \boldsymbol{v}$
- Replace non-smooth $||\boldsymbol{\beta}||_1$ by smooth $||\boldsymbol{u}||_2^2 + ||\boldsymbol{v}||_2^2$

$$\frac{1}{n}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^2 + \lambda||\boldsymbol{\beta}||_1$$

$$\frac{1}{n}(\boldsymbol{y} - \boldsymbol{X}(\boldsymbol{u} \odot \boldsymbol{v}))^2 + \lambda||\boldsymbol{\beta}||_1$$

$$\frac{1}{n}(\boldsymbol{y} - \boldsymbol{X}(\boldsymbol{u} \odot \boldsymbol{v}))^2 + \lambda(||\boldsymbol{u}||_2^2 + ||\boldsymbol{v}||_2^2)$$



Hadamard Product Parameterization

for Lasso

$$\frac{1}{n}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^2 + \lambda||\boldsymbol{\beta}||_1$$

- Parametrize $\boldsymbol{\beta} = \boldsymbol{u} \odot \boldsymbol{v}$
- Replace non-smooth $||\boldsymbol{\beta}||_1$
  by smooth $||\boldsymbol{u}||_2^2 + ||\boldsymbol{v}||_2^2$

$$\frac{1}{n}(\boldsymbol{y} - \boldsymbol{X}(\boldsymbol{u} \odot \boldsymbol{v}))^2 + \lambda||\boldsymbol{\beta}||_1$$

$$\frac{1}{n}(\boldsymbol{y} - \boldsymbol{X}(\boldsymbol{u} \odot \boldsymbol{v}))^2 + \lambda(||\boldsymbol{u}||_2^2 + ||\boldsymbol{v}||_2^2)$$

Hadamard Product Parameterization

for Lasso

$$\frac{1}{n}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^2 + \lambda||\boldsymbol{\beta}||_1$$

- Parametrize $\boldsymbol{\beta} = \boldsymbol{u} \odot \boldsymbol{v}$
- Replace non-smooth $||\boldsymbol{\beta}||_1$
  by smooth $||\boldsymbol{u}||_2^2 + ||\boldsymbol{v}||_2^2$

$$\frac{1}{n}(\boldsymbol{y} - \boldsymbol{X}(\boldsymbol{u} \odot \boldsymbol{v}))^2 + \lambda||\boldsymbol{\beta}||_1$$

$$\frac{1}{n}(\boldsymbol{y} - \boldsymbol{X}(\boldsymbol{u} \odot \boldsymbol{v}))^2 + \lambda(||\boldsymbol{u}||_2^2 + ||\boldsymbol{v}||_2^2)$$

⇒ Optimal solution is the same,
& introduces no additional local minima

Practical implications

- “Truly” sparse solutions in neural networks using SGD
- Can be used for **anywhere** in the network

Practical implications

- “Truly” sparse solutions in neural networks using SGD
- Can be used for **anywhere** in the network
- Idea works for a large class of penalizations

$$\boldsymbol{\beta} \mapsto \sum_{j=1}^{L} \omega_j ||\boldsymbol{\beta}_j||_p^q \qquad 0 < q \leq p \leq 2,\ \omega_j > 0\ \forall\, j \in [L]$$

Empirical results: Linear model

$\beta = 0$

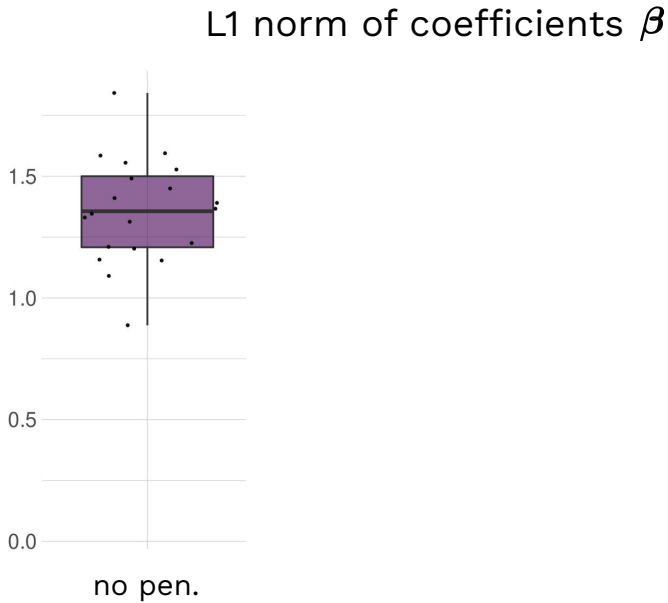
Empirical results: Linear model

$\beta = 0$

L1 norm of coefficients β

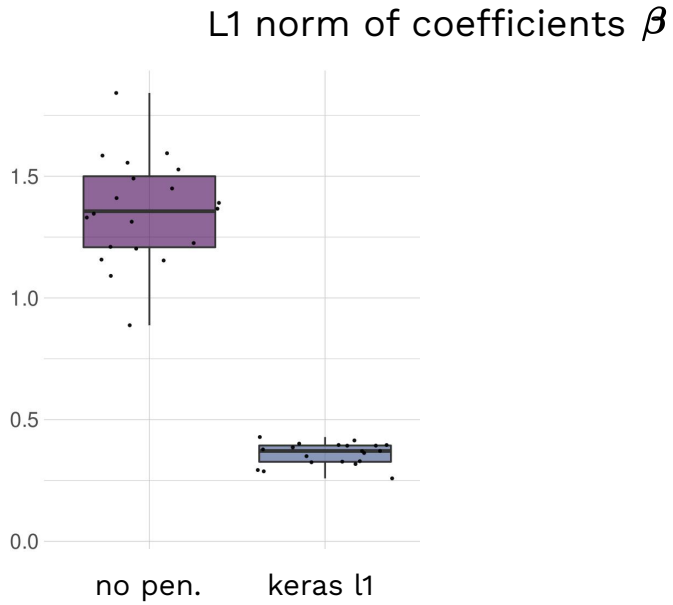
Empirical results: Linear model

$\beta = 0$



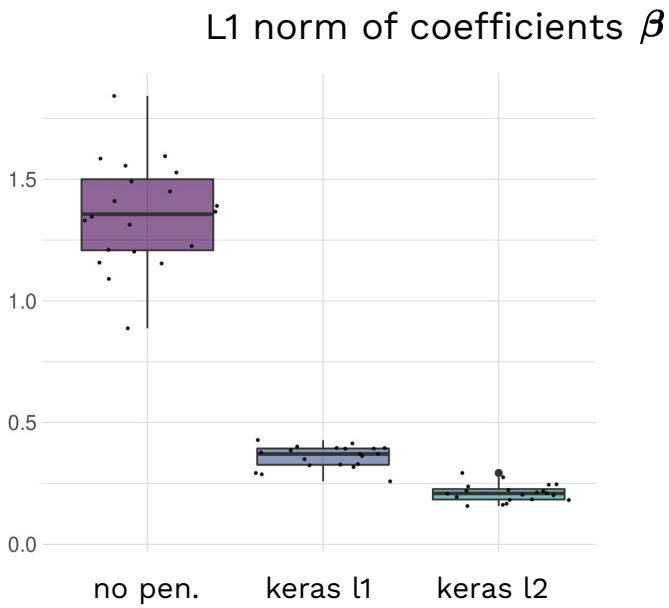
Empirical results: Linear model

$\beta = 0$



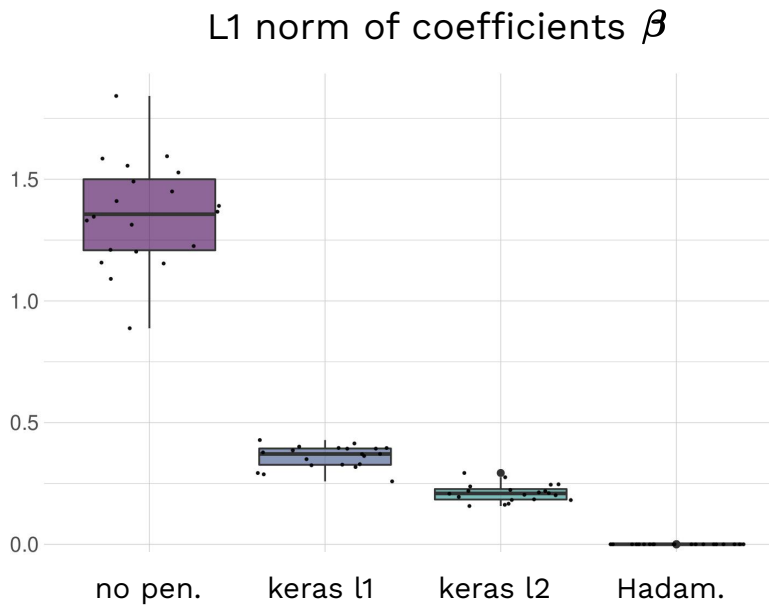
Empirical results: Linear model

$\beta = 0$

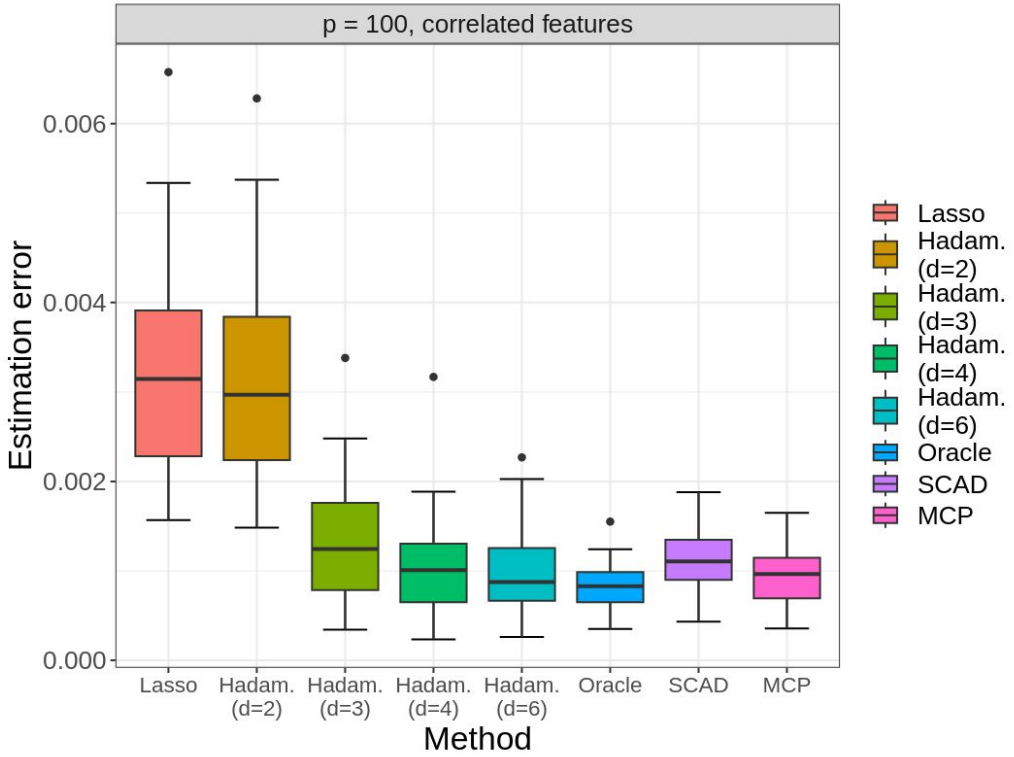


Empirical results: Linear model

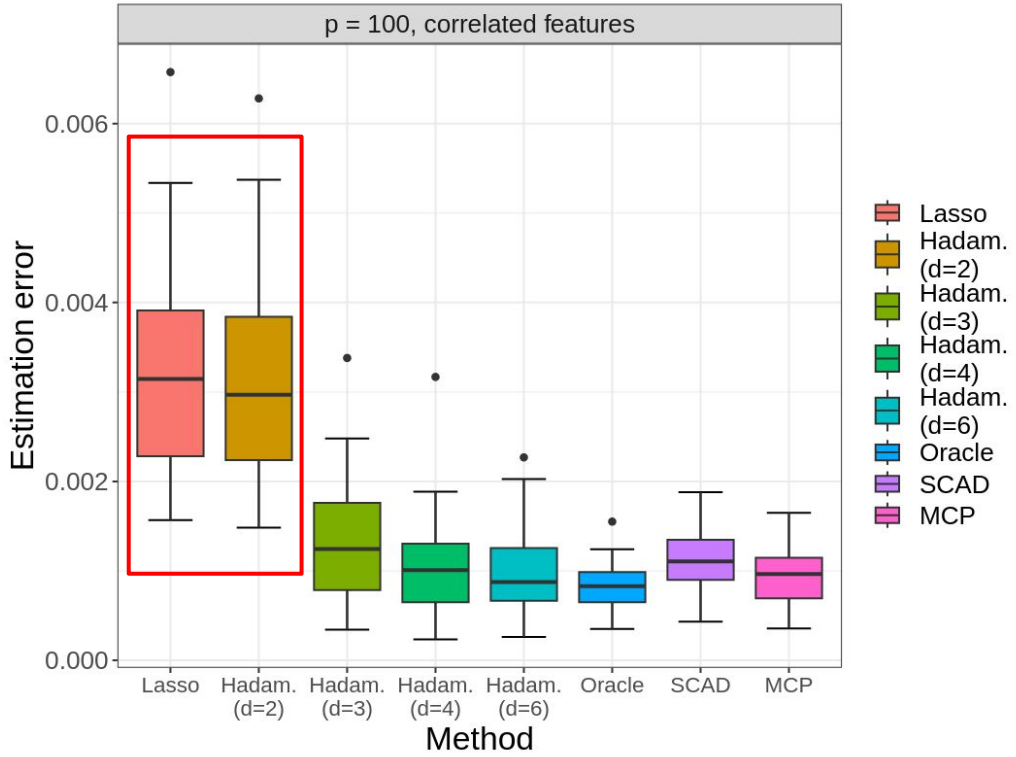
$\beta = 0$



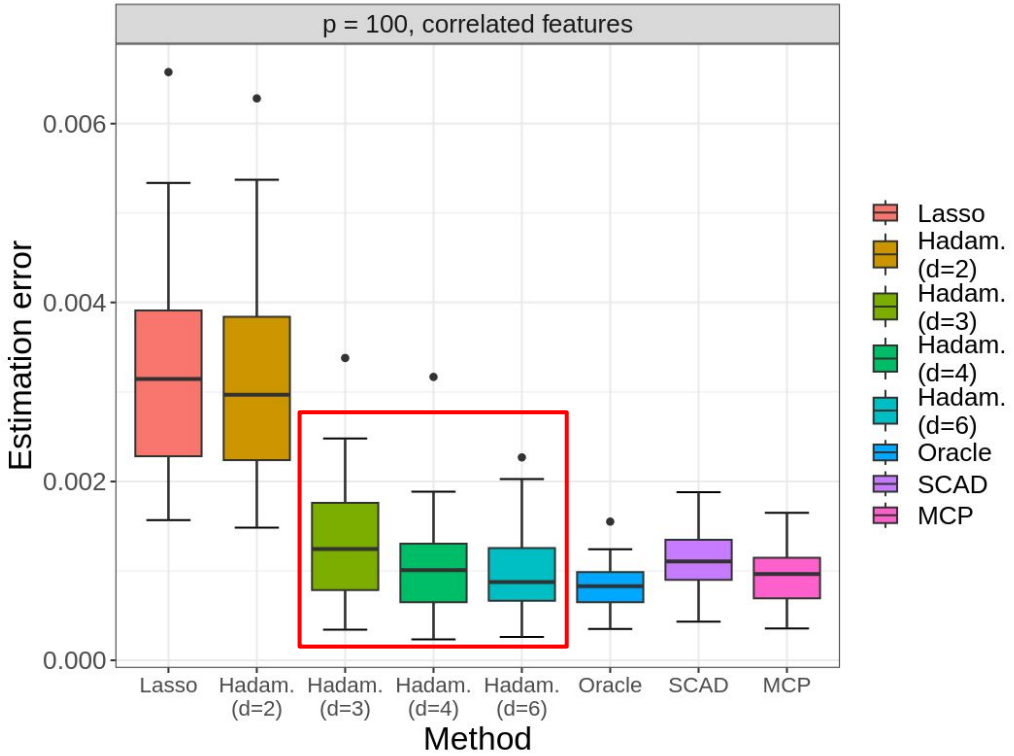
Empirical results: Linear model



Empirical results: Linear model

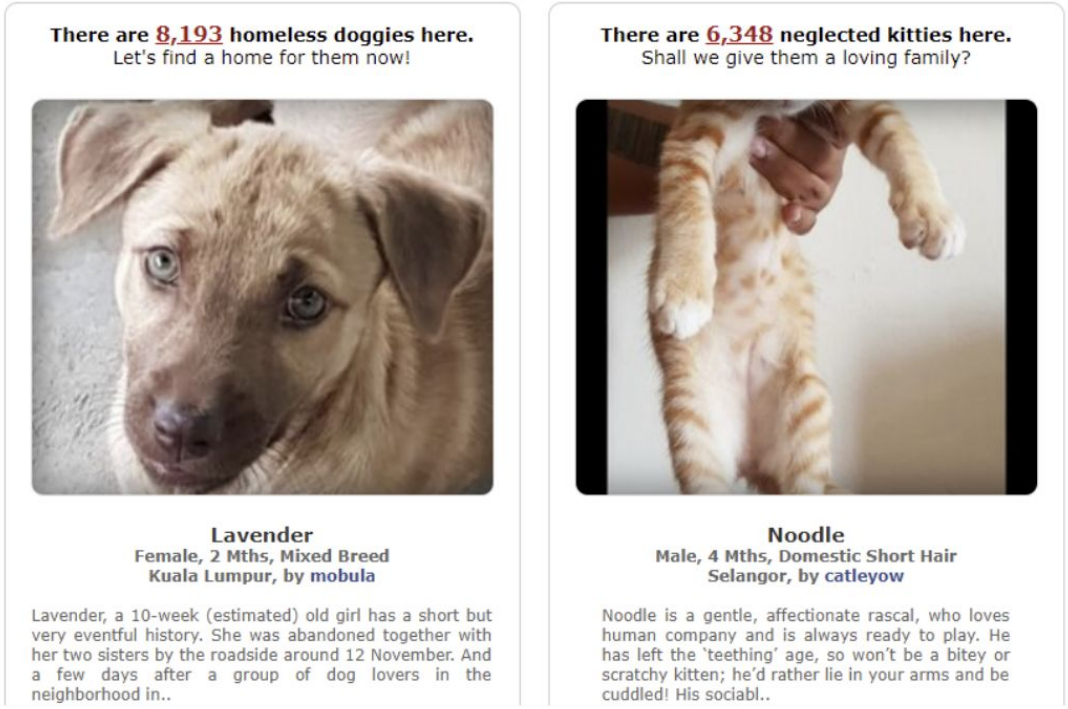


Empirical results: Linear model



$$\boldsymbol{\beta} = \boldsymbol{u}_1 \odot \boldsymbol{u}_2 \odot \cdots \odot \boldsymbol{u}_d \longrightarrow \ell_{2/d}\text{-norm}$$

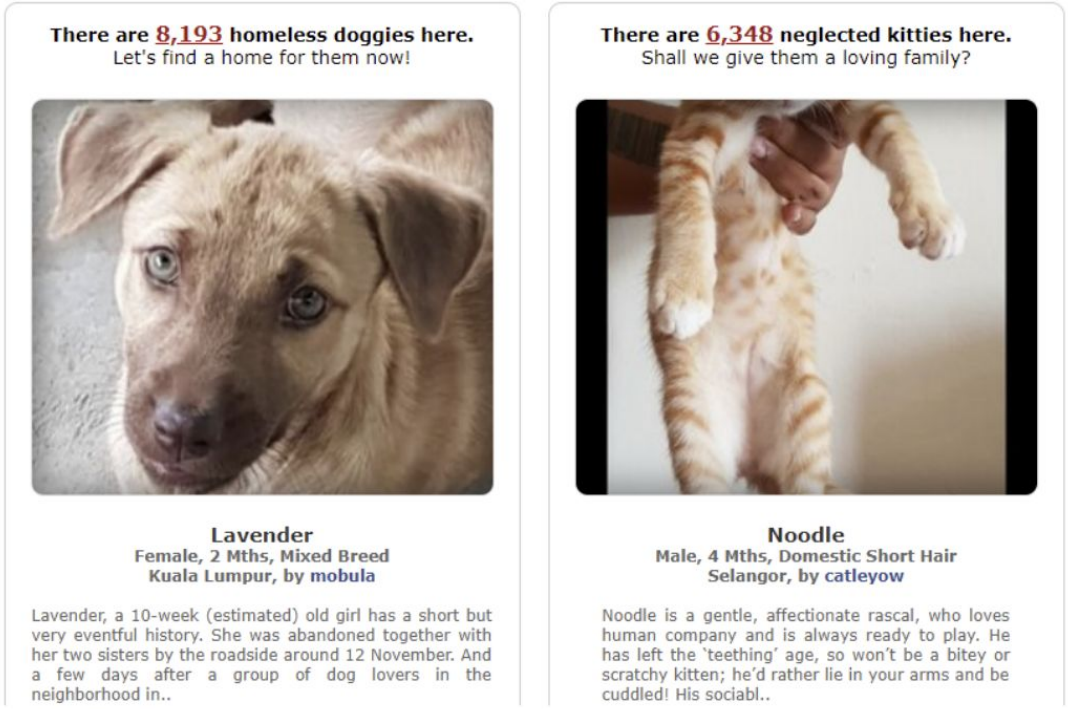
Empirical results: Sparse modality regression



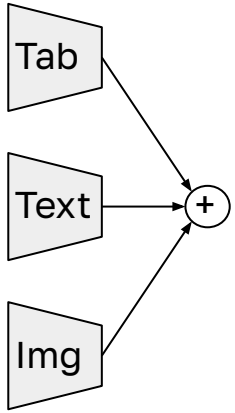
Tabular + Image + Text Data

Source: Petfinder.my

Empirical results: Sparse modality regression



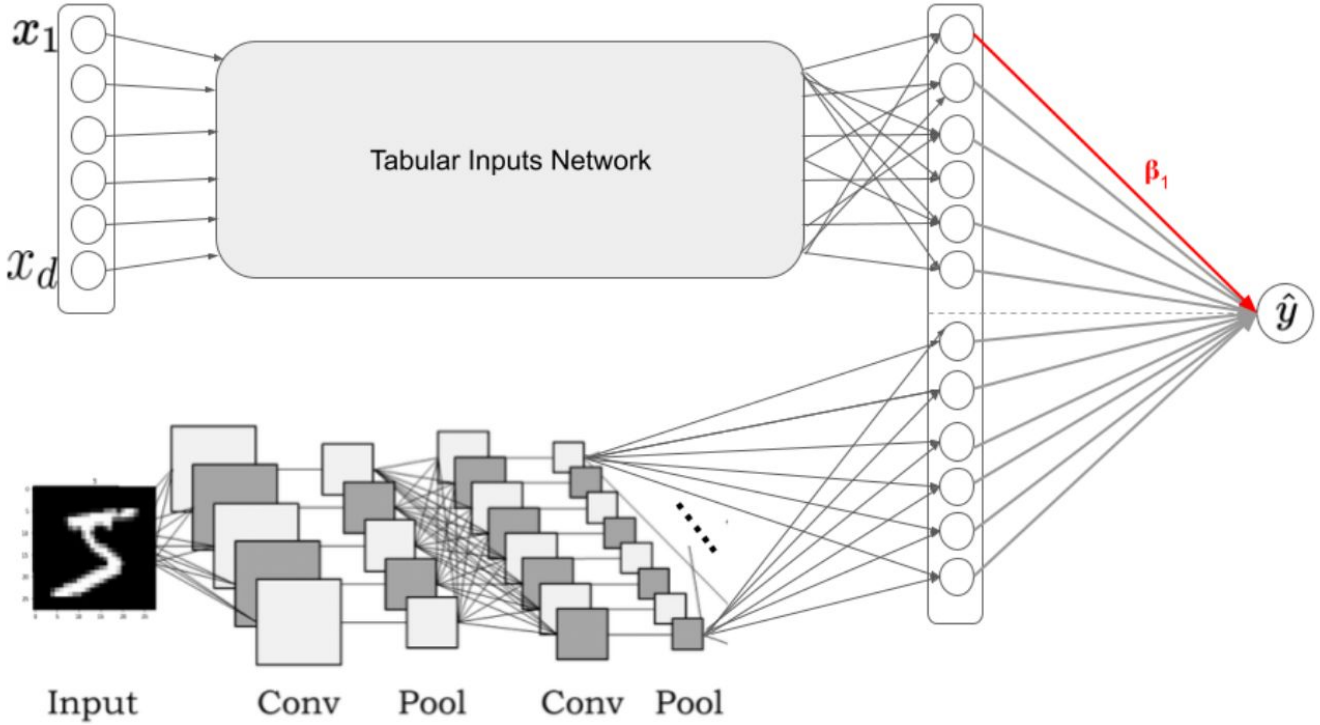
Semi-structured Neural Network



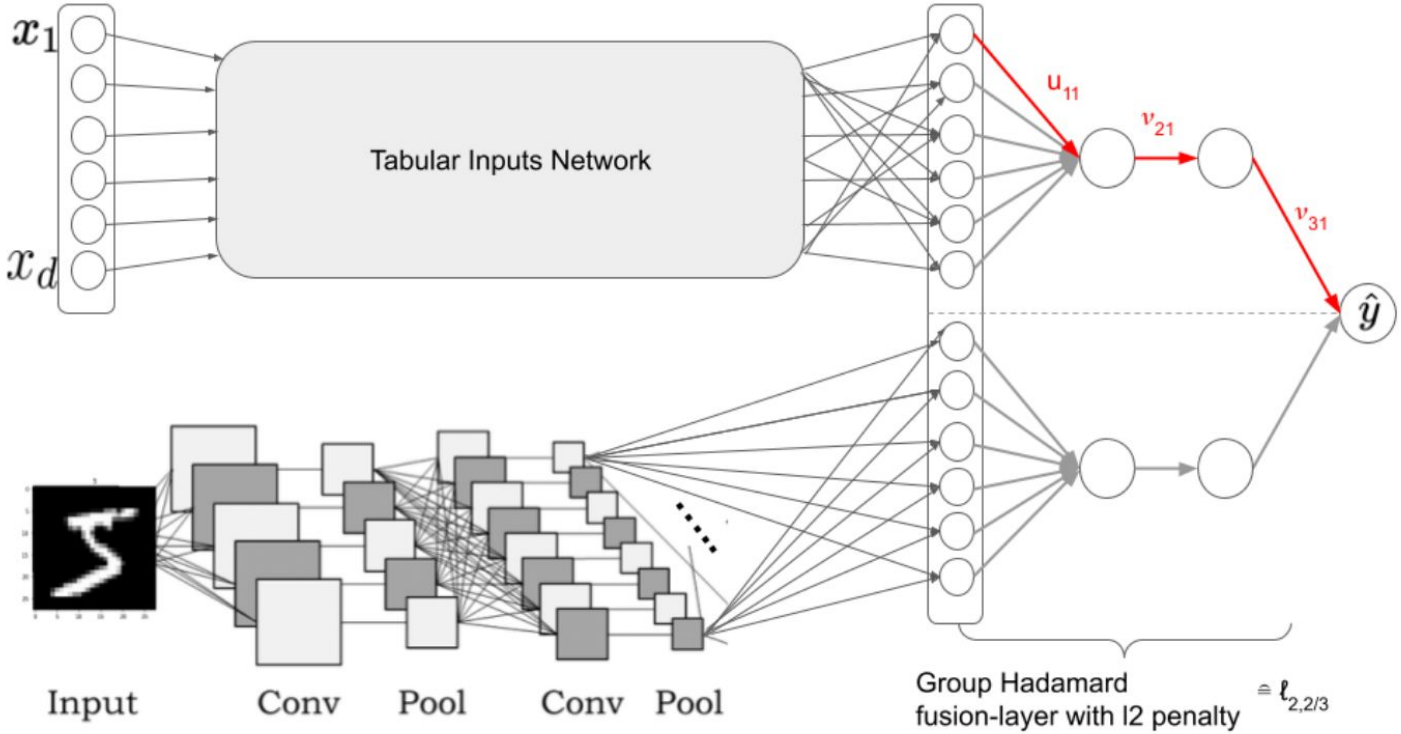
Tabular + Image + Text Data

Source: Petfinder.my

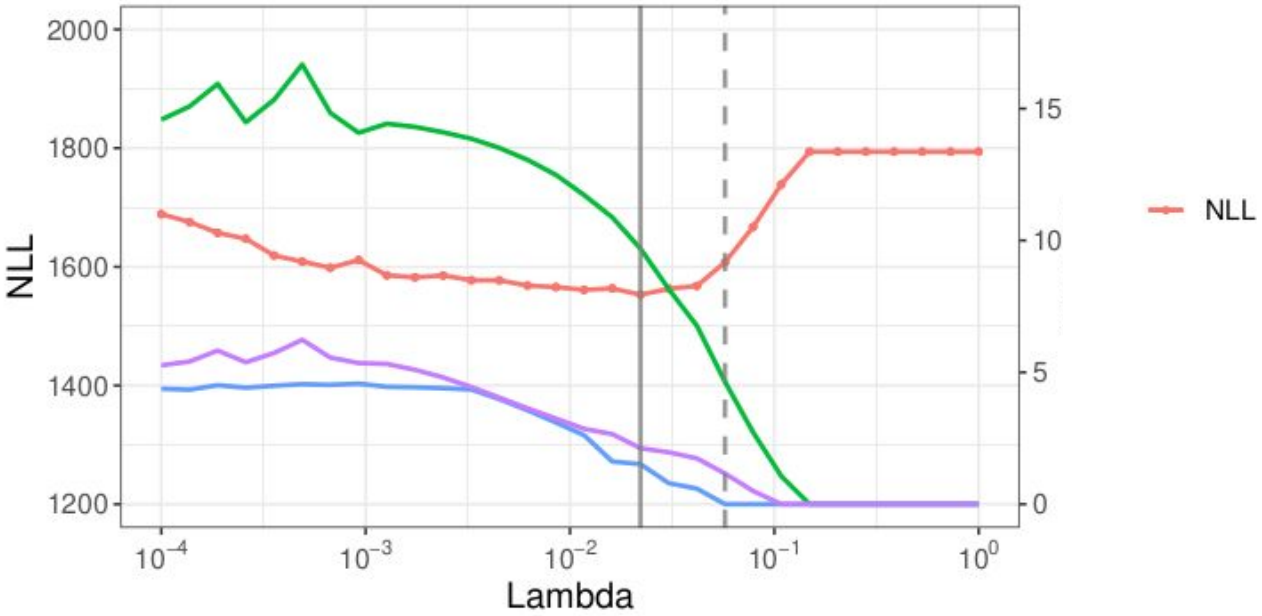
Empirical results: Sparse modality regression



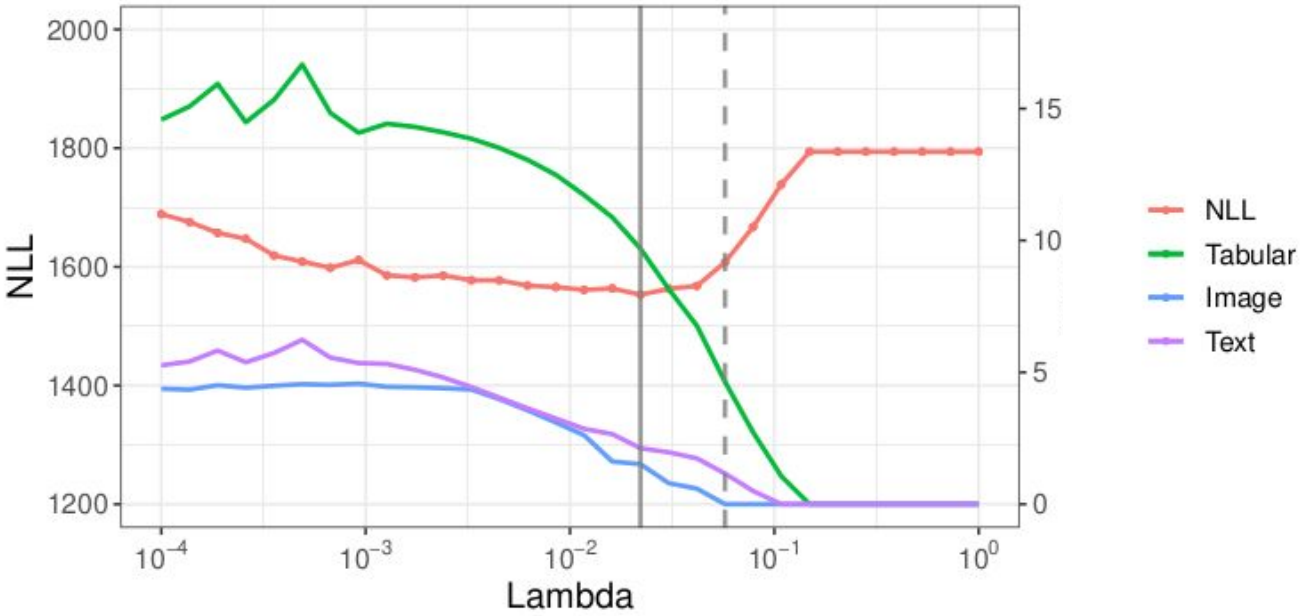
Empirical results: Sparse modality regression



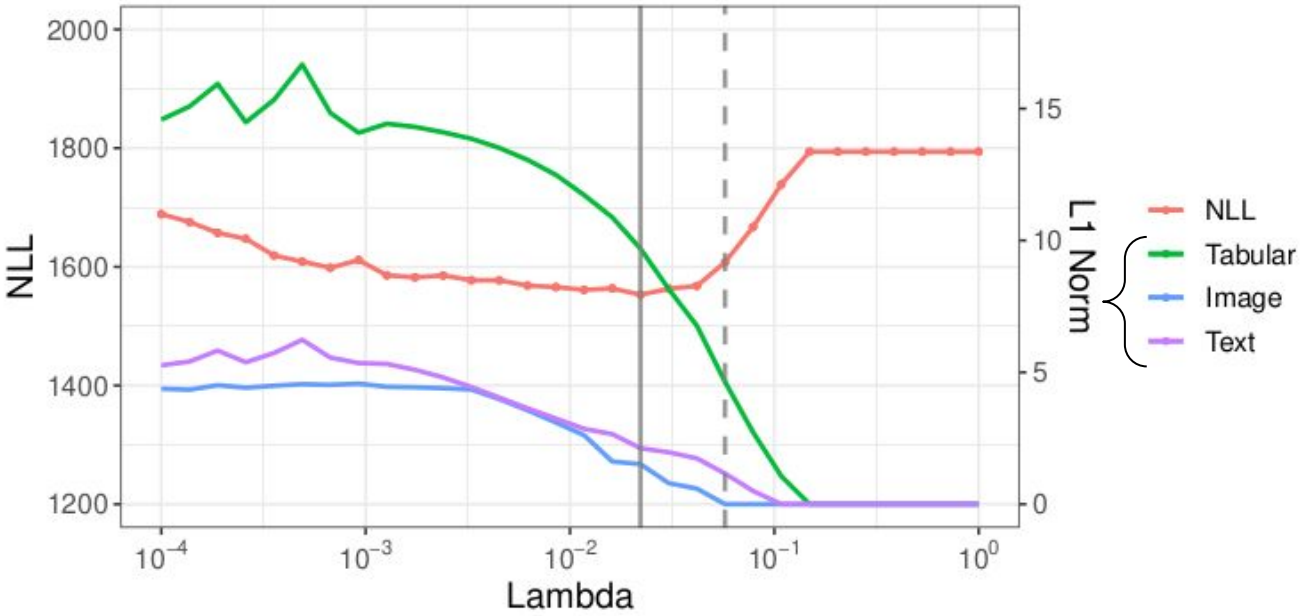
Empirical results: Sparse modality regression



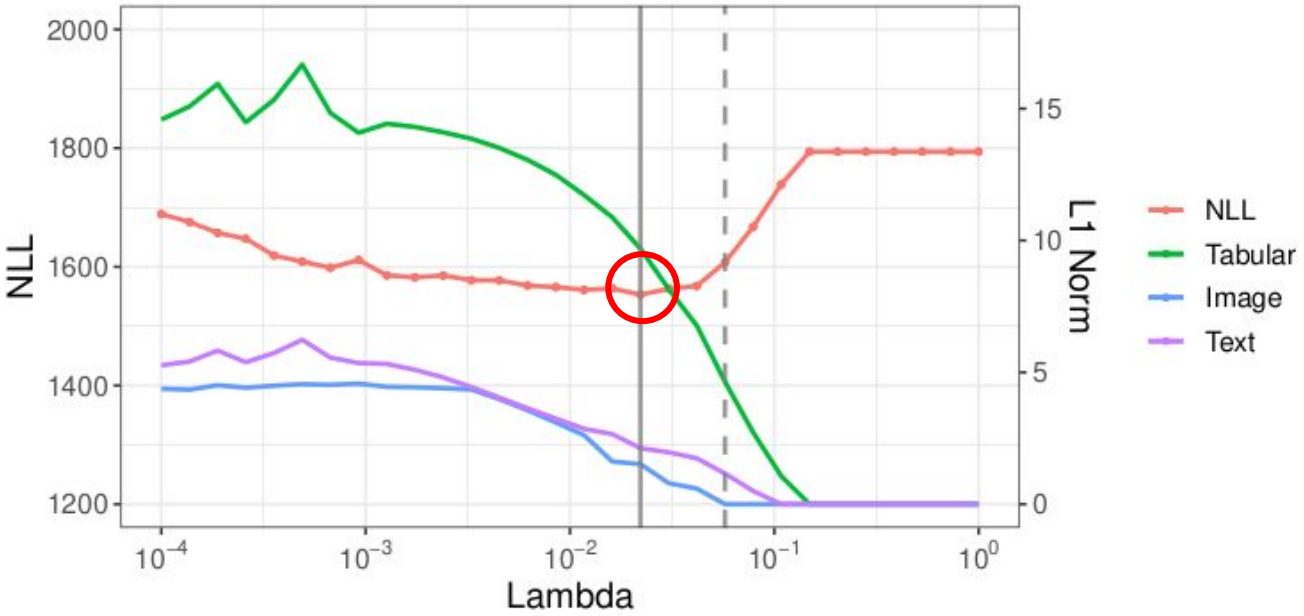
Empirical results: Sparse modality regression



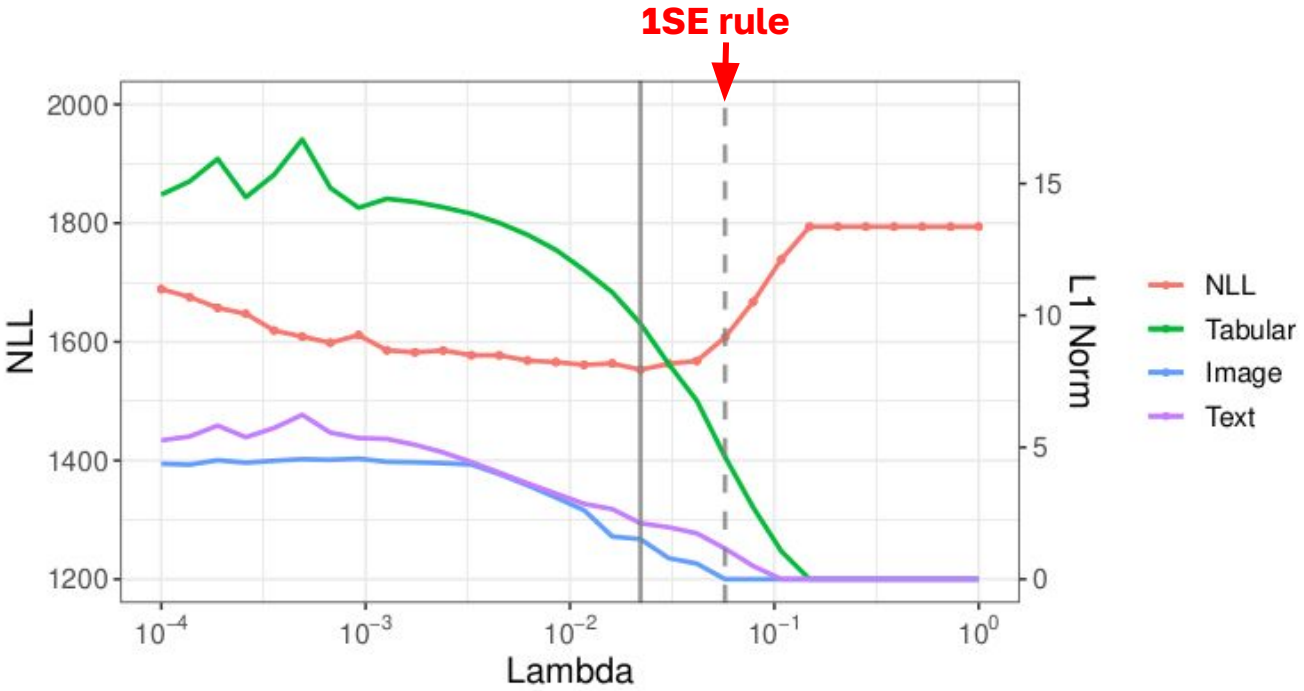
Empirical results: Sparse modality regression



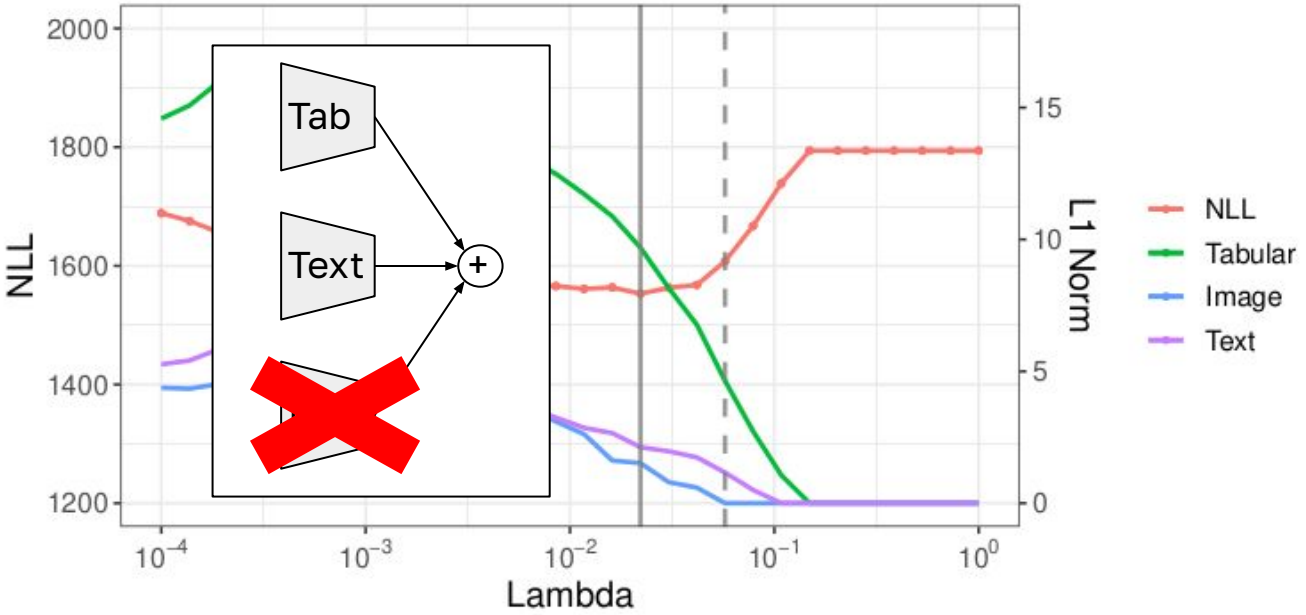
Empirical results: Sparse modality regression



Empirical results: Sparse modality regression



Empirical results: Sparse modality regression



Empirical results: Sparse modality regression