Neural Network Building Blocks


[Thomas Eakins, Public Domain]

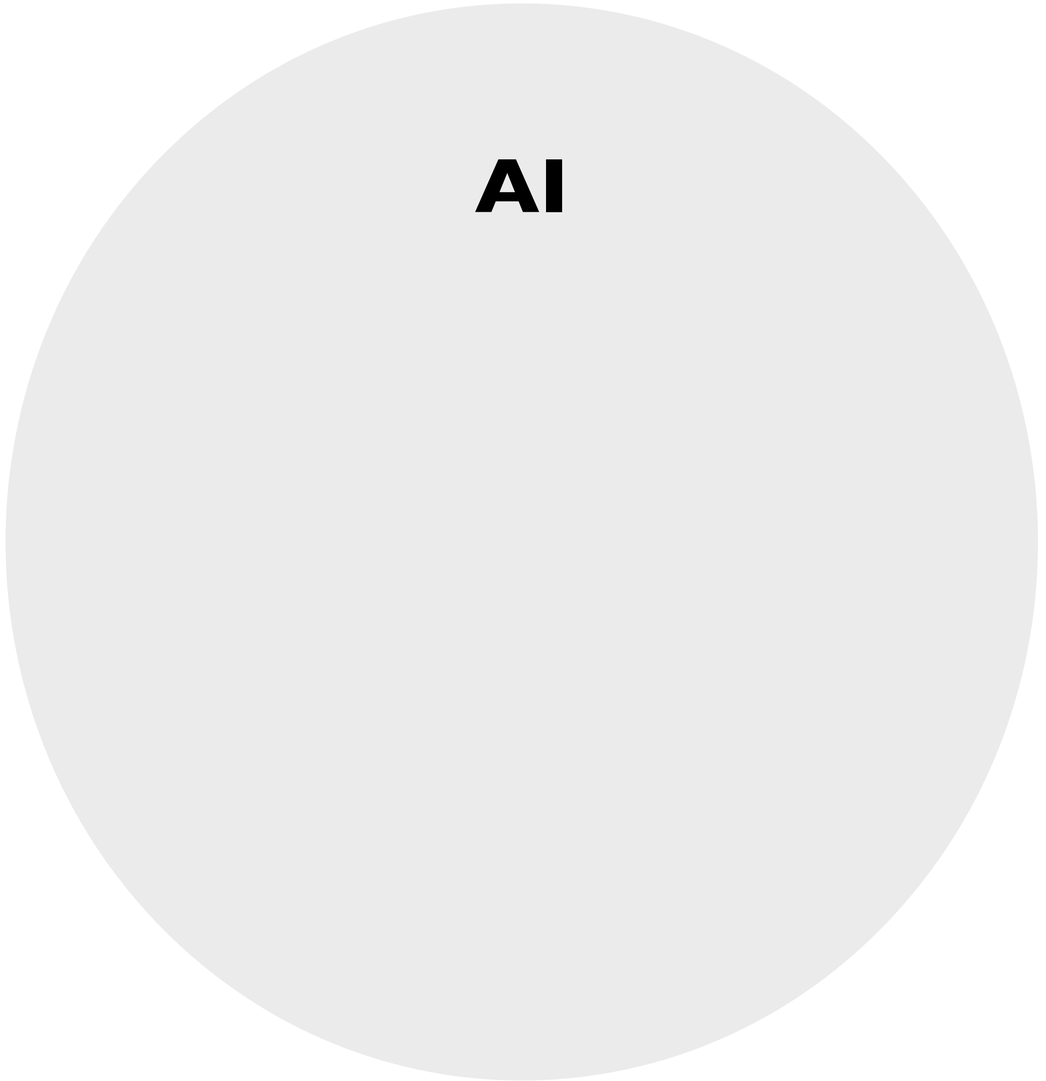
Johannes Erdmann
RWTH Aachen University

Deep Learning Basics Train-the-Trainer Workshop
February 7-8, 2023
Garching

RWTH AACHEN UNIVERSITY

Artificial Intelligence

- Numerous AI applications in our every-day lives

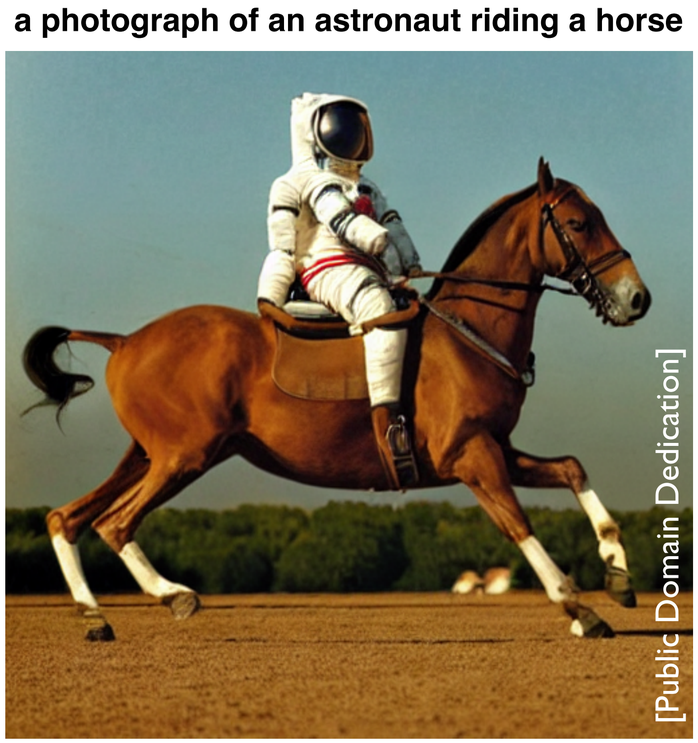


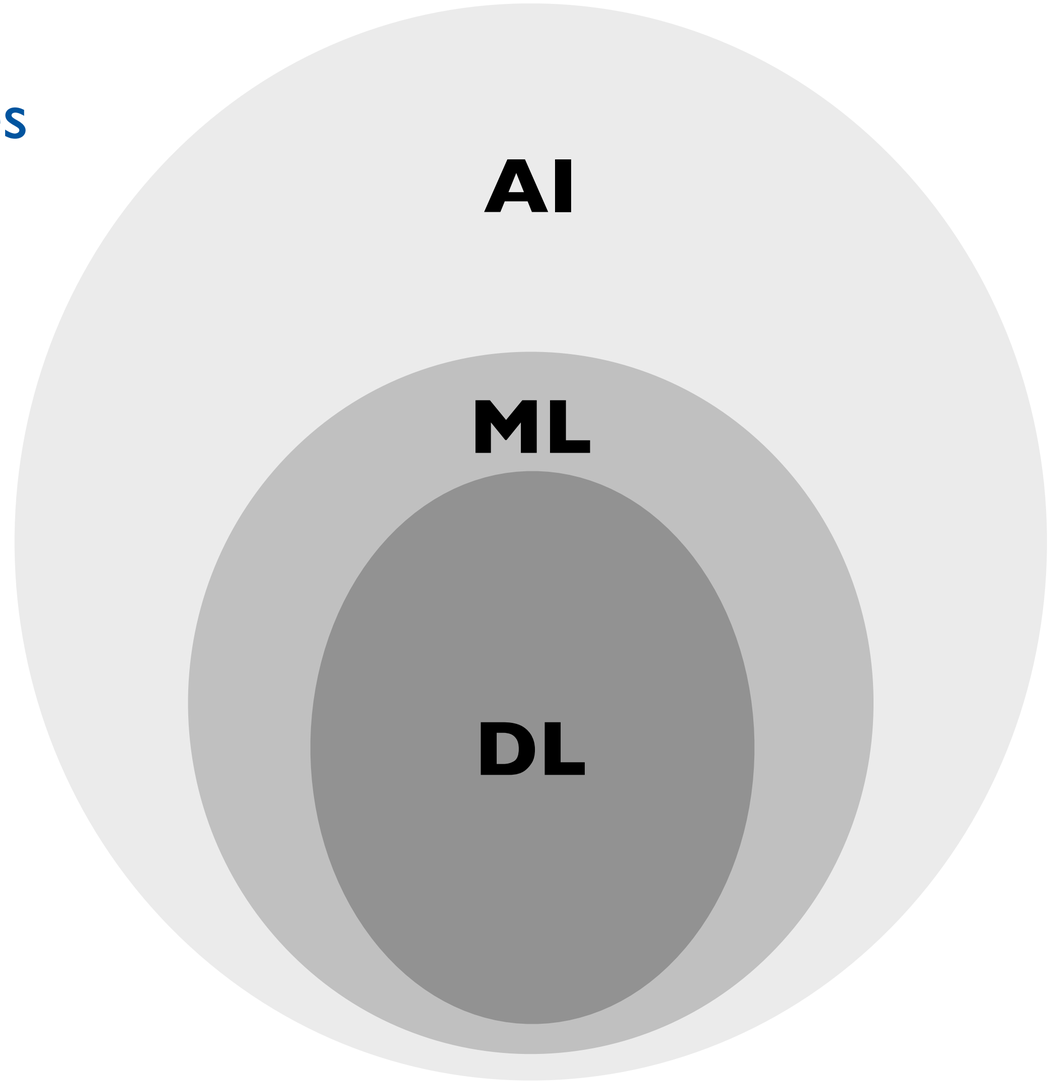
Artificial Intelligence

- Numerous AI applications in our every-day lives
- Machine learning is enabling fascinating capabilities
- Most recent examples:

ChatGPT

Stable Diffusion





Machine Learning

- Data + Task + "a mathematical formulation of what means 'better' " *

  (such as the number of type-1 and type-2 errors in a classification task)

- Optimization of the model parameters on the data

  = "Training" (aka the machine 'learns')

Task

Data

* often called a "performance measure"

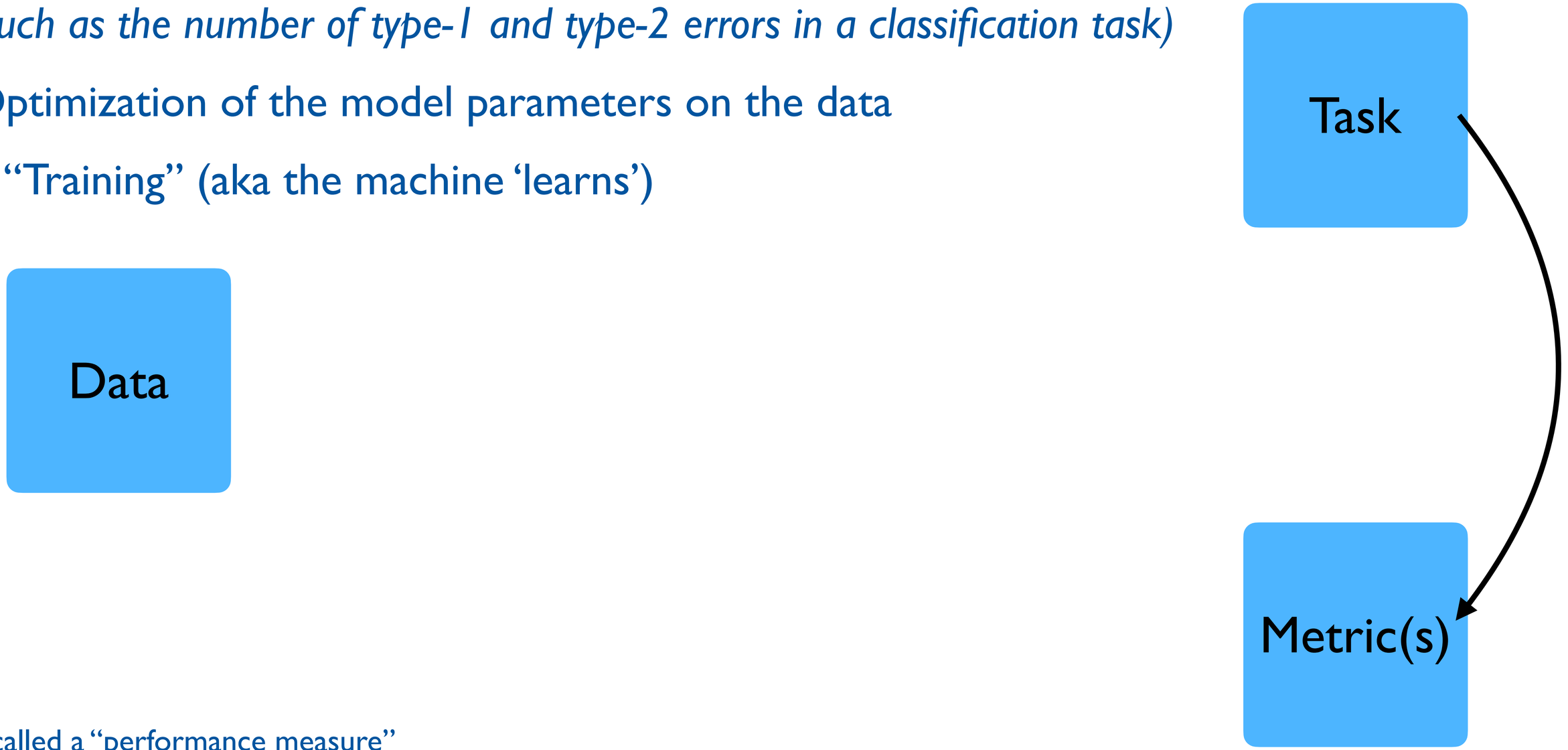
Machine Learning

- Data + Task + "a mathematical formulation of what means 'better' " *

  (such as the number of type-1 and type-2 errors in a classification task)

- Optimization of the model parameters on the data

  = "Training" (aka the machine 'learns')



* often called a "performance measure"

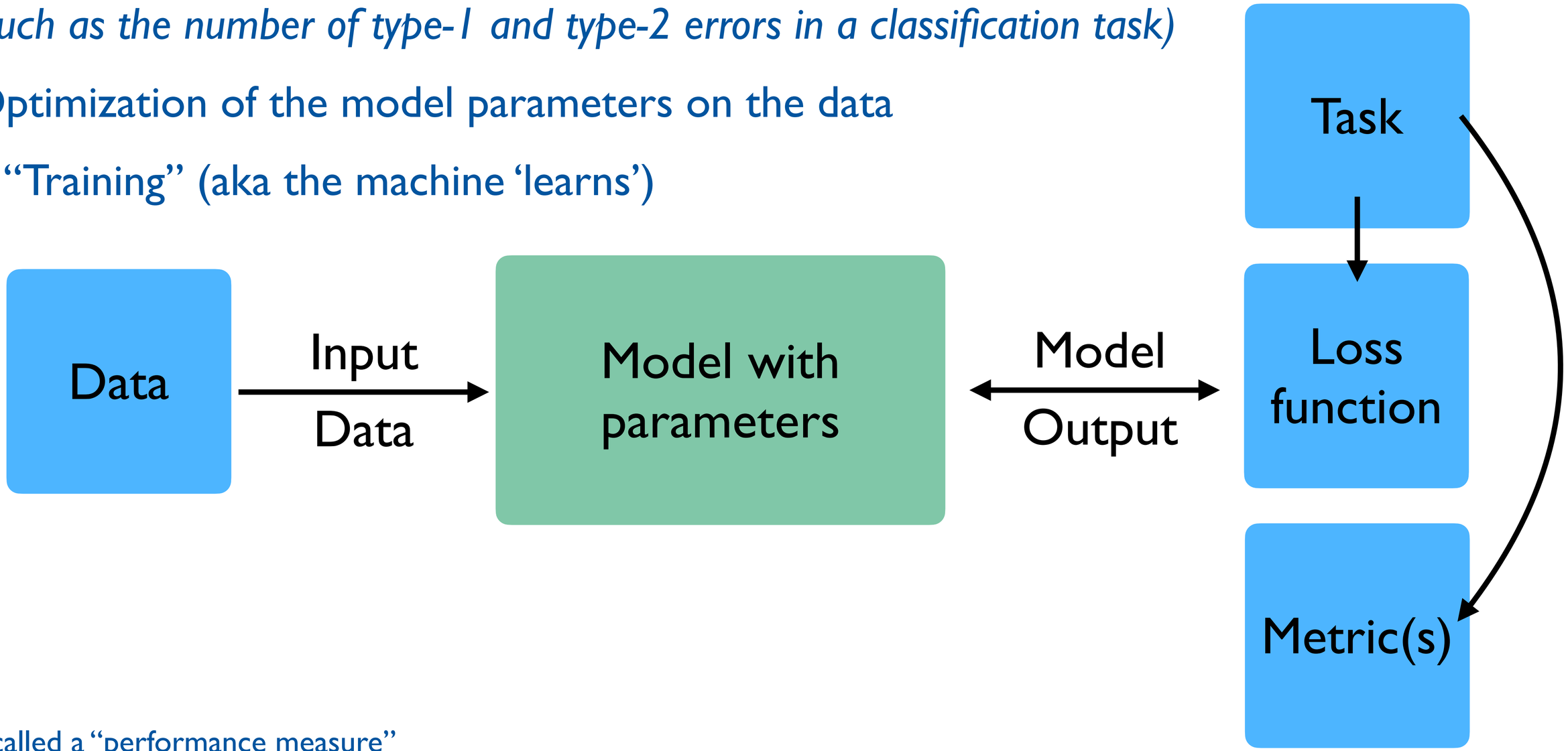
Machine Learning

- Data + Task + "a mathematical formulation of what means 'better' " *

  (such as the number of type-1 and type-2 errors in a classification task)

- Optimization of the model parameters on the data

  = "Training" (aka the machine 'learns')



* often called a "performance measure"

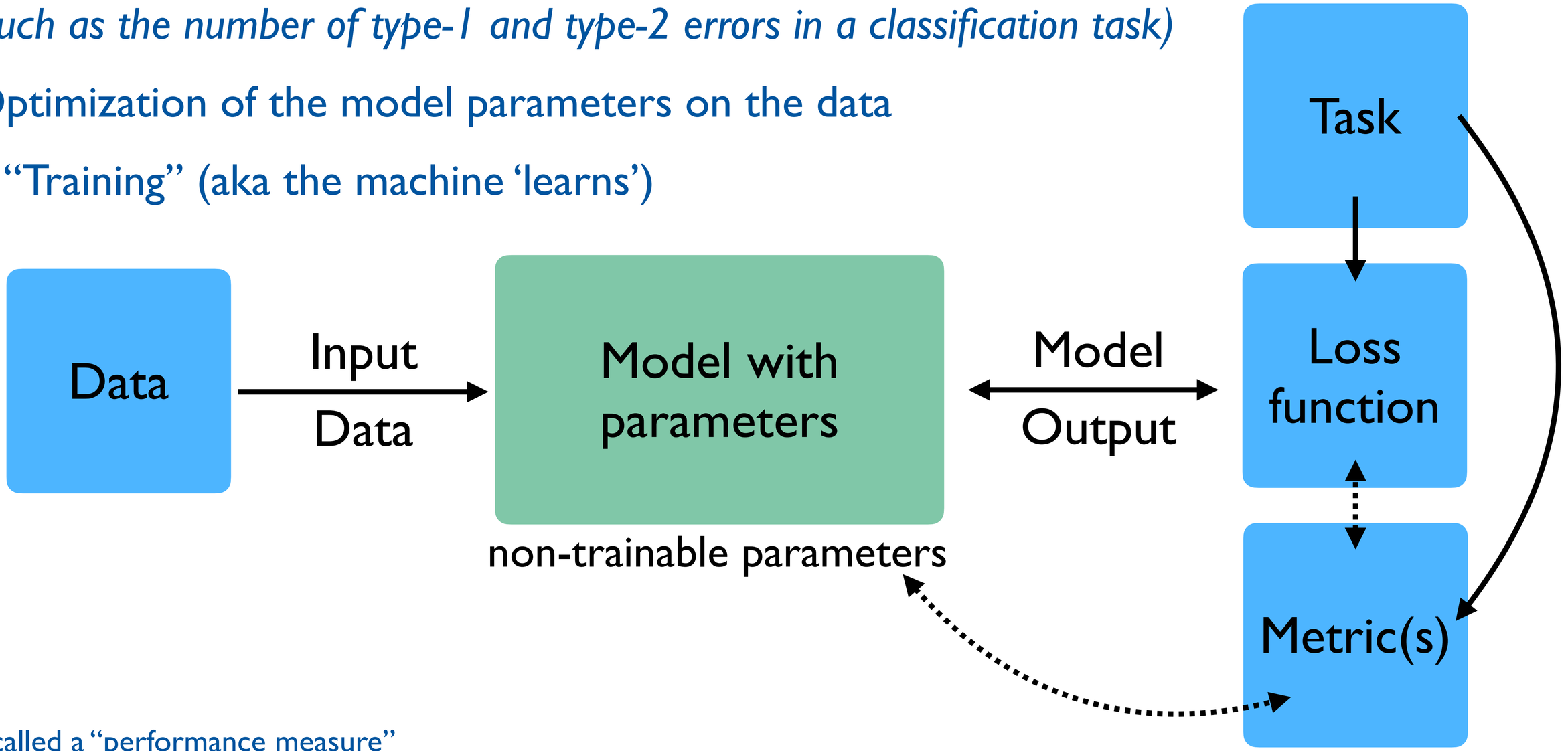
Machine Learning

- Data + Task + "a mathematical formulation of what means 'better' " *

  (such as the number of type-1 and type-2 errors in a classification task)

- Optimization of the model parameters on the data

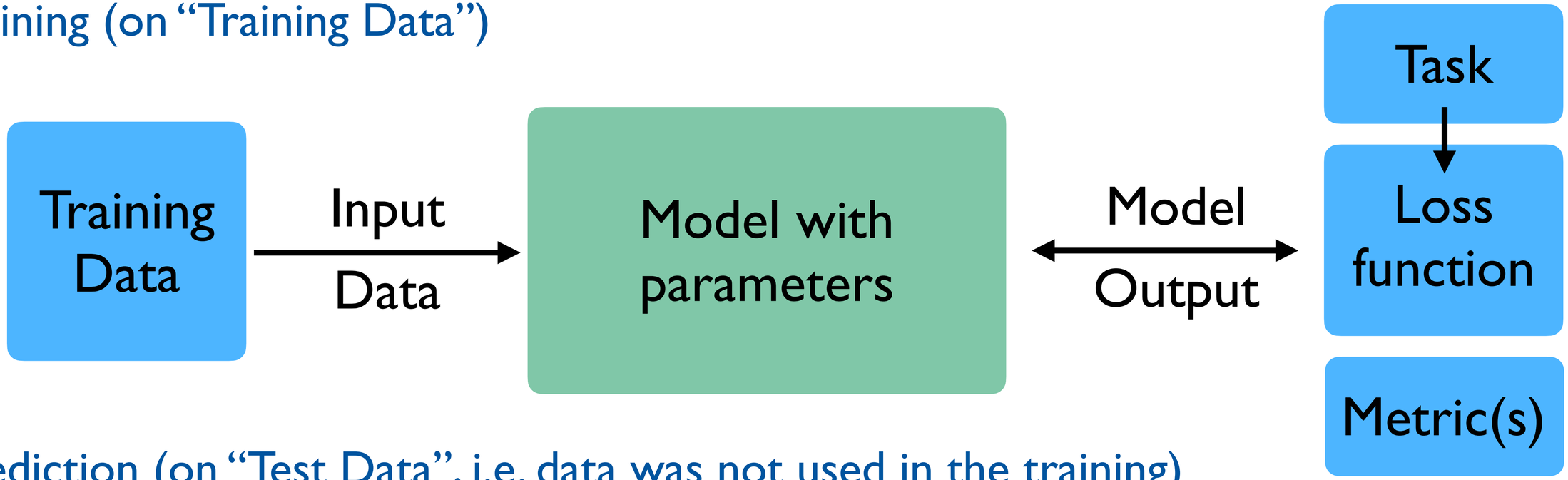
  = "Training" (aka the machine 'learns')



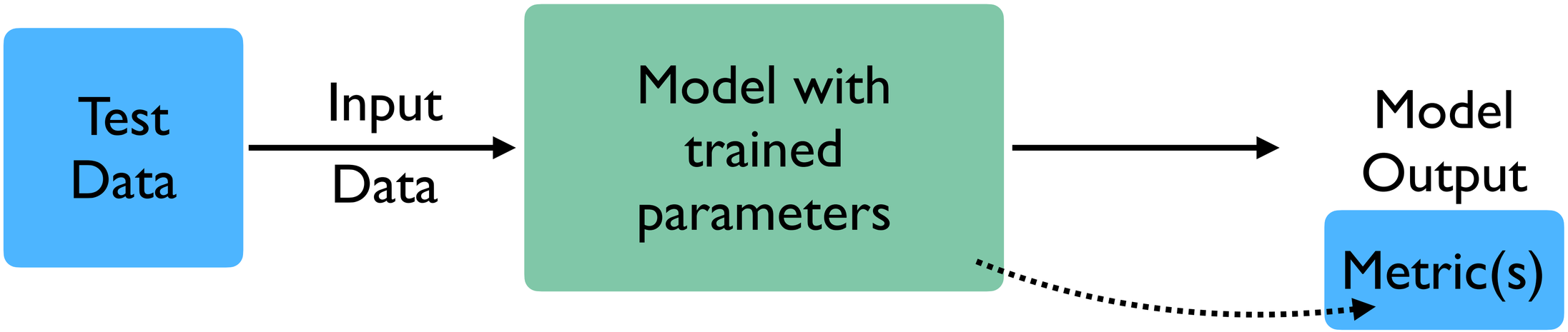
* often called a "performance measure"

Training (Parameter Optimization)

- Training (on "Training Data")

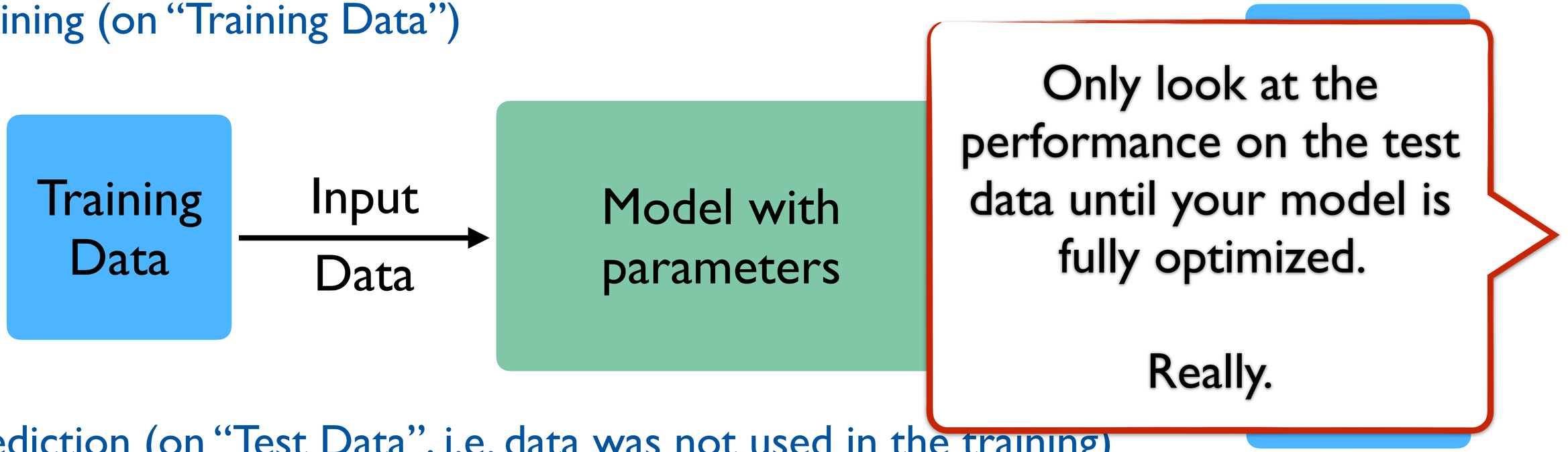


- Prediction (on "Test Data", i.e. data was not used in the training)

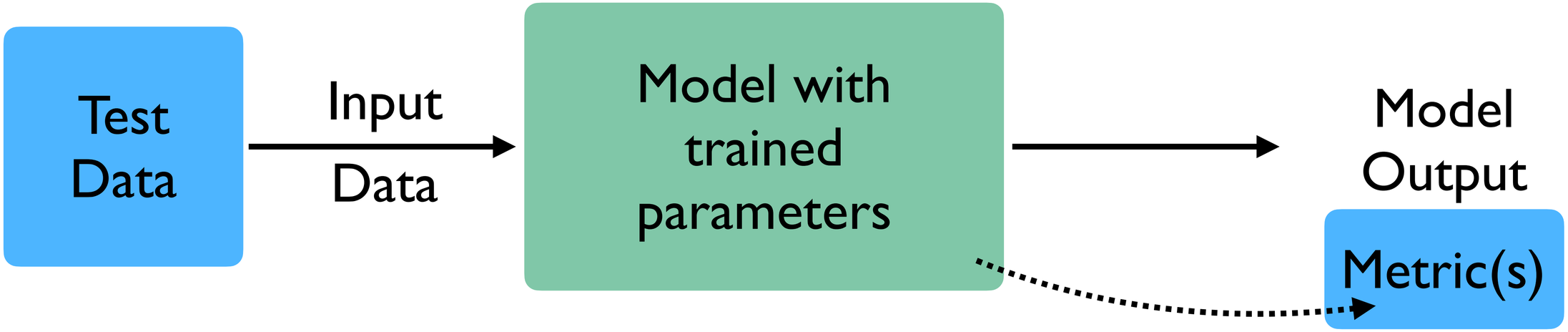


Training (Parameter Optimization)

- Training (on "Training Data")



- Prediction (on "Test Data", i.e. data was not used in the training)



Different Tasks

- ML type closely related to the available data for the task

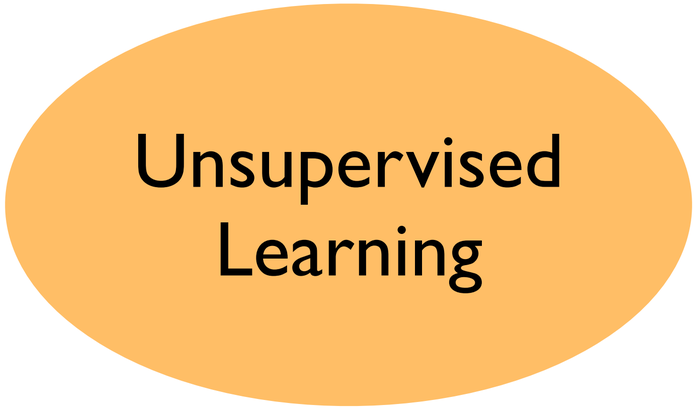
- Labelled data
- Classification

  (image classification, …)
- Regression

  (market forecasting, …)

Different Tasks

- ML type closely related to the available data for the task



- Labelled data
- Classification

  (image classification, …)
- Regression

  (market forecasting, …)

- Unlabelled data
- Clustering

  (recommender systems, …)
- Dimensionality Reduction

  (data compression, …)

Different Tasks

- ML type closely related to the available data for the task

Supervised Learning

- Labelled data
- Classification
  (image classification, …)
- Regression
  (market forecasting, …)

Unsupervised Learning

- Unlabelled data
- Clustering
  (recommender systems, …)
- Dimensionality Reduction
  (data compression, …)

Reinforcement Learning

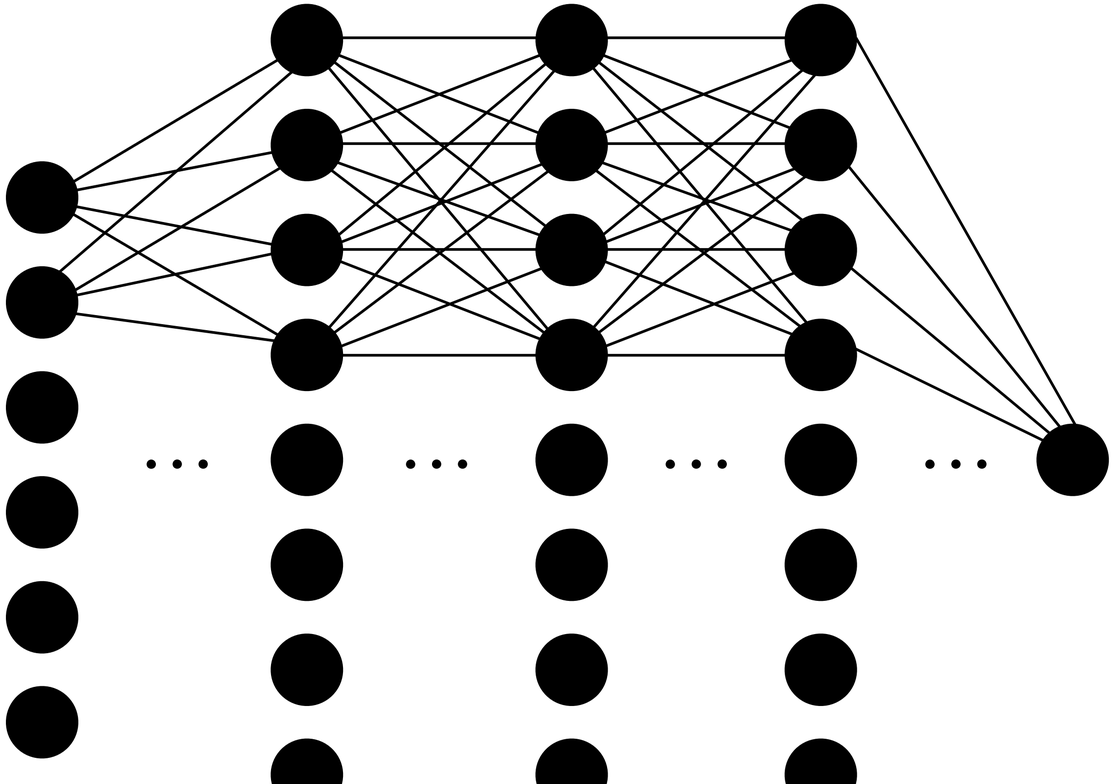
- An "agent" receives rewards for actions in an environment
- Robot Navigation, Games, …

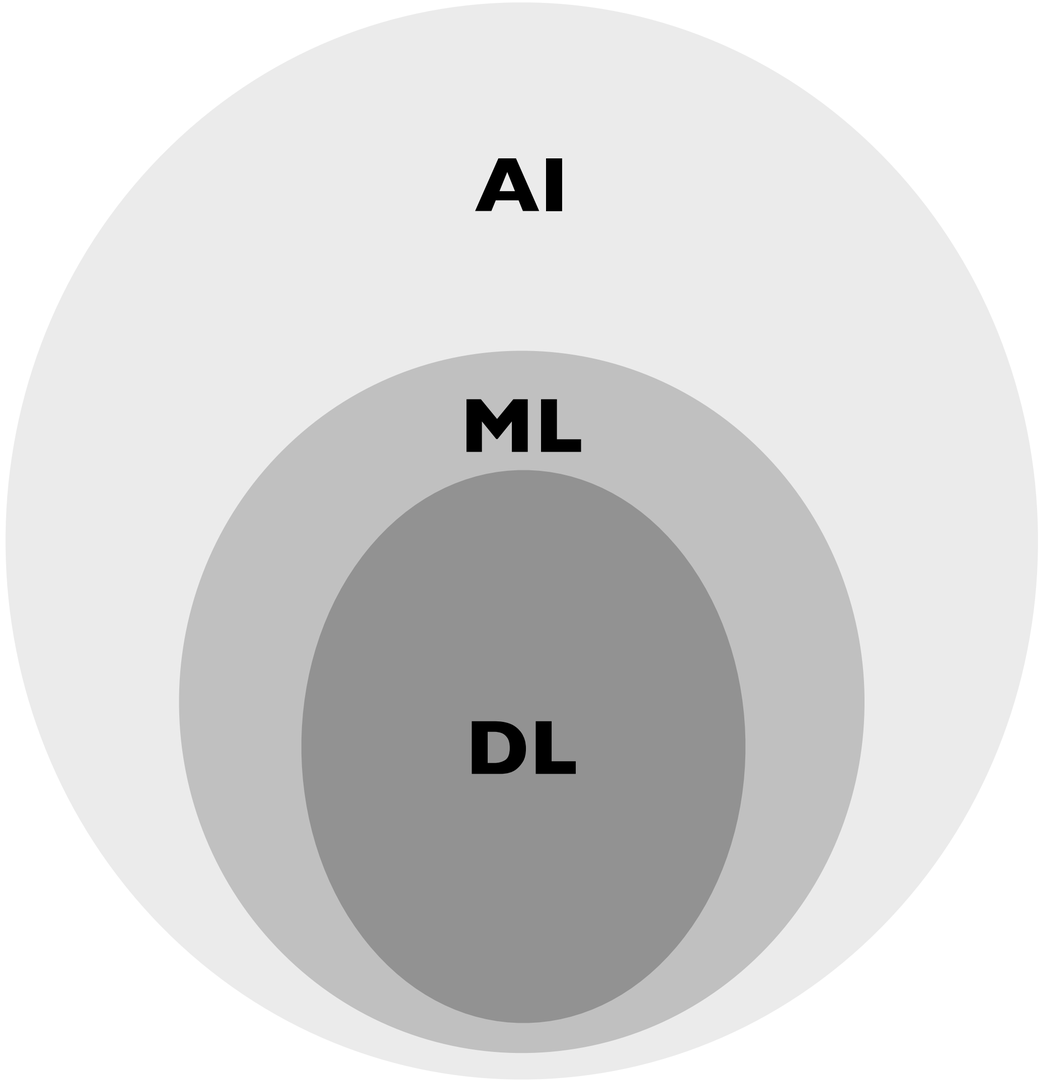
Deep Learning

- A large number of recent advances in ML

  is due to “deep learning”

  = training of deep neural networks



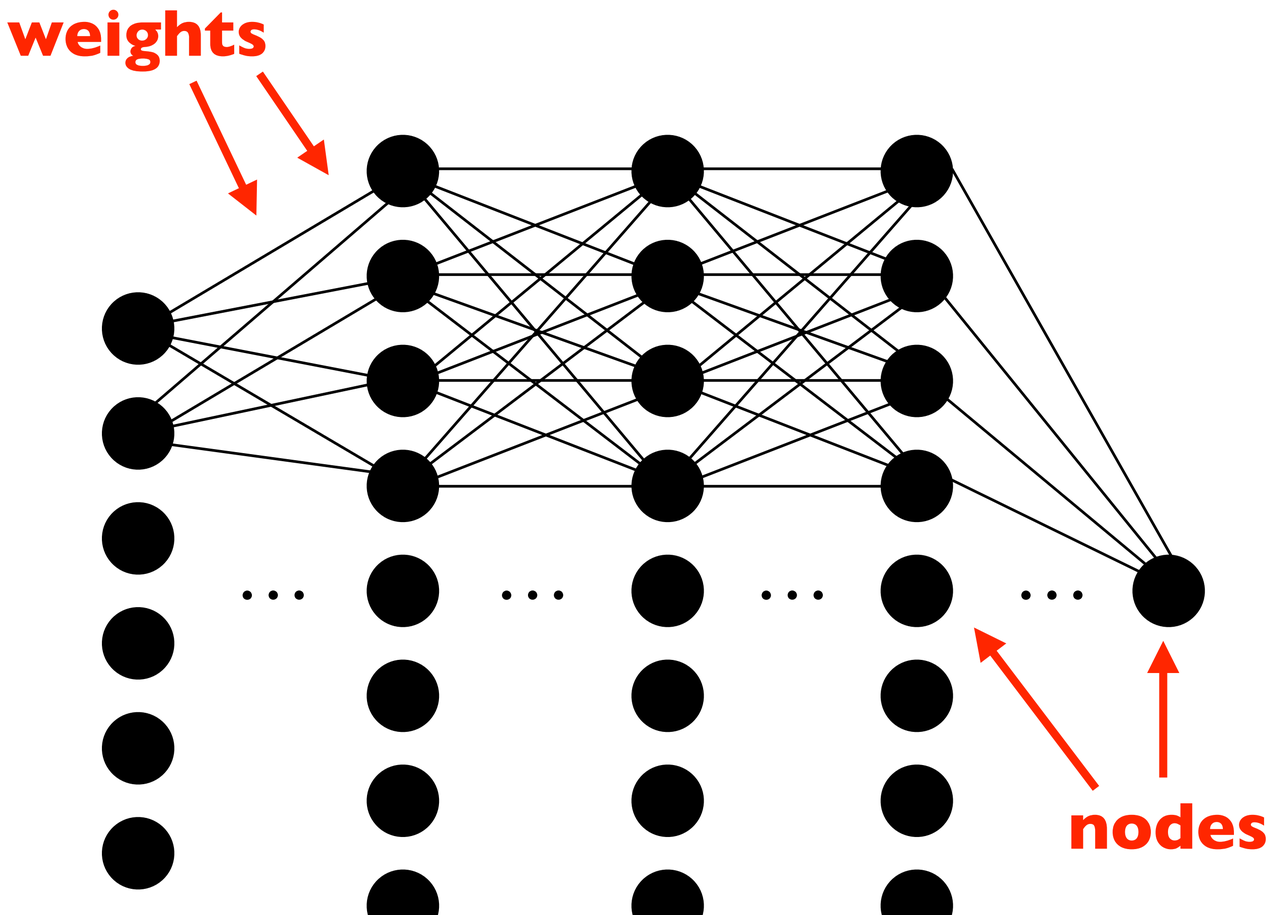


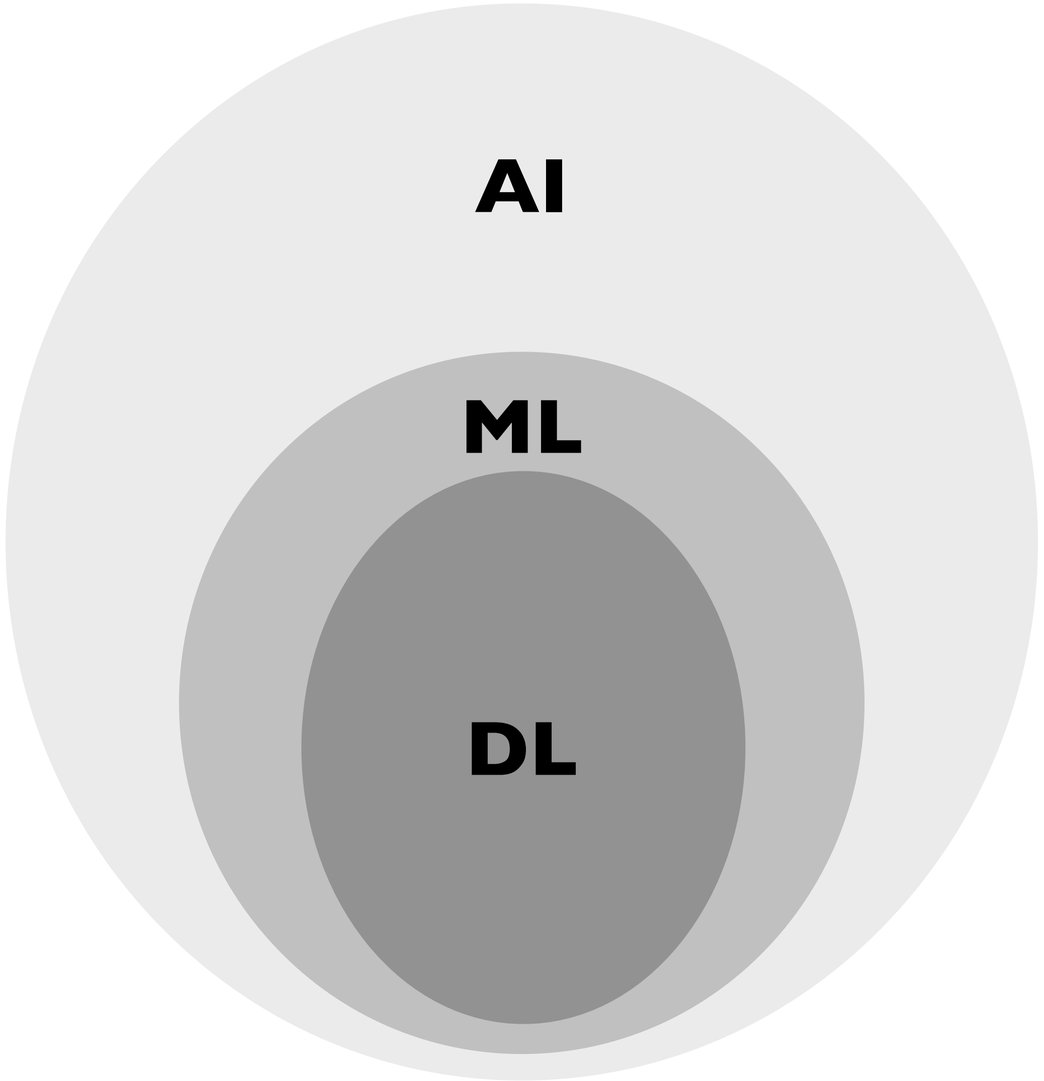
Deep Learning

- A large number of recent advances in ML

  is due to “deep learning”

  = training of deep neural networks



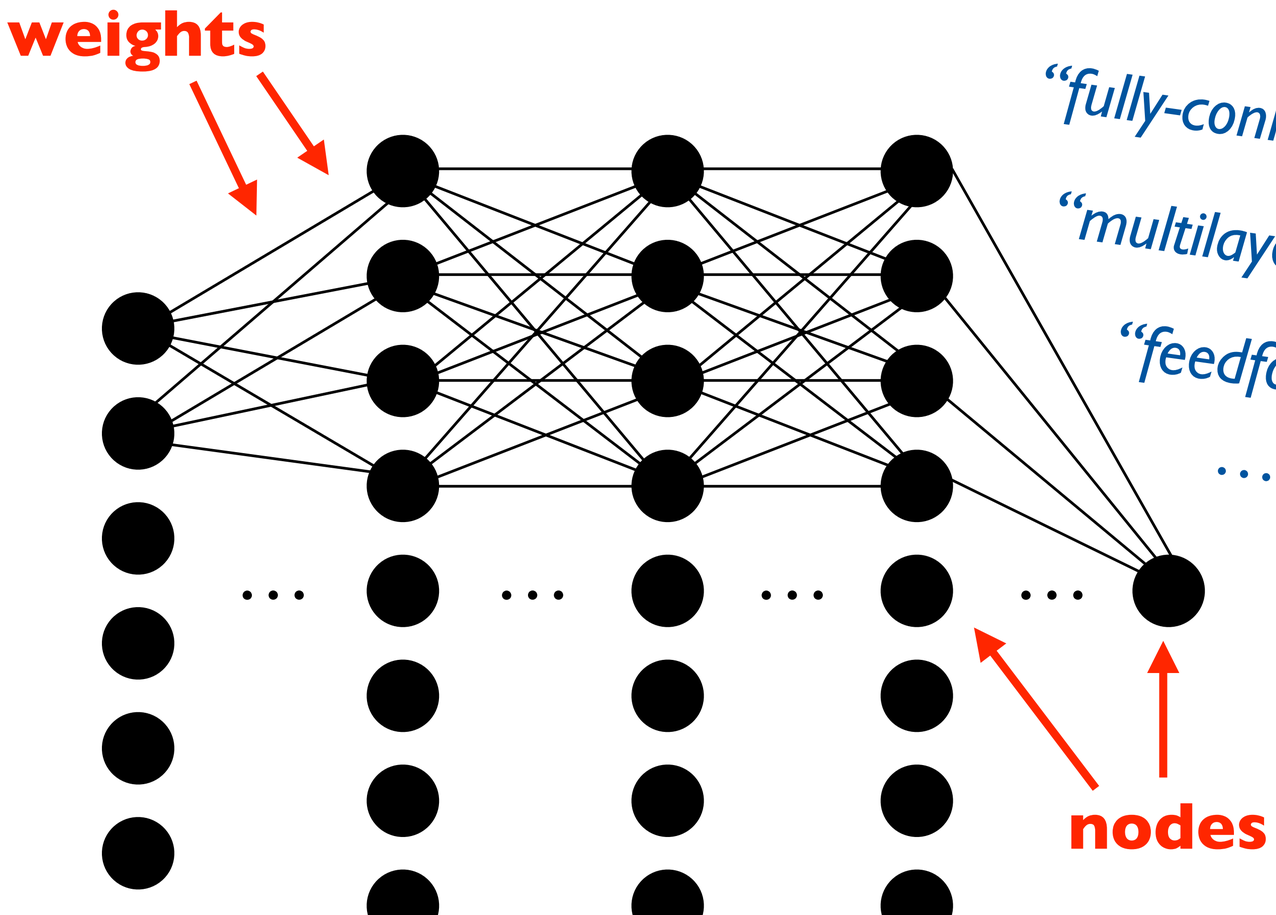


Deep Learning

- A large number of recent advances in ML

  is due to "deep learning"

  = training of deep neural networks

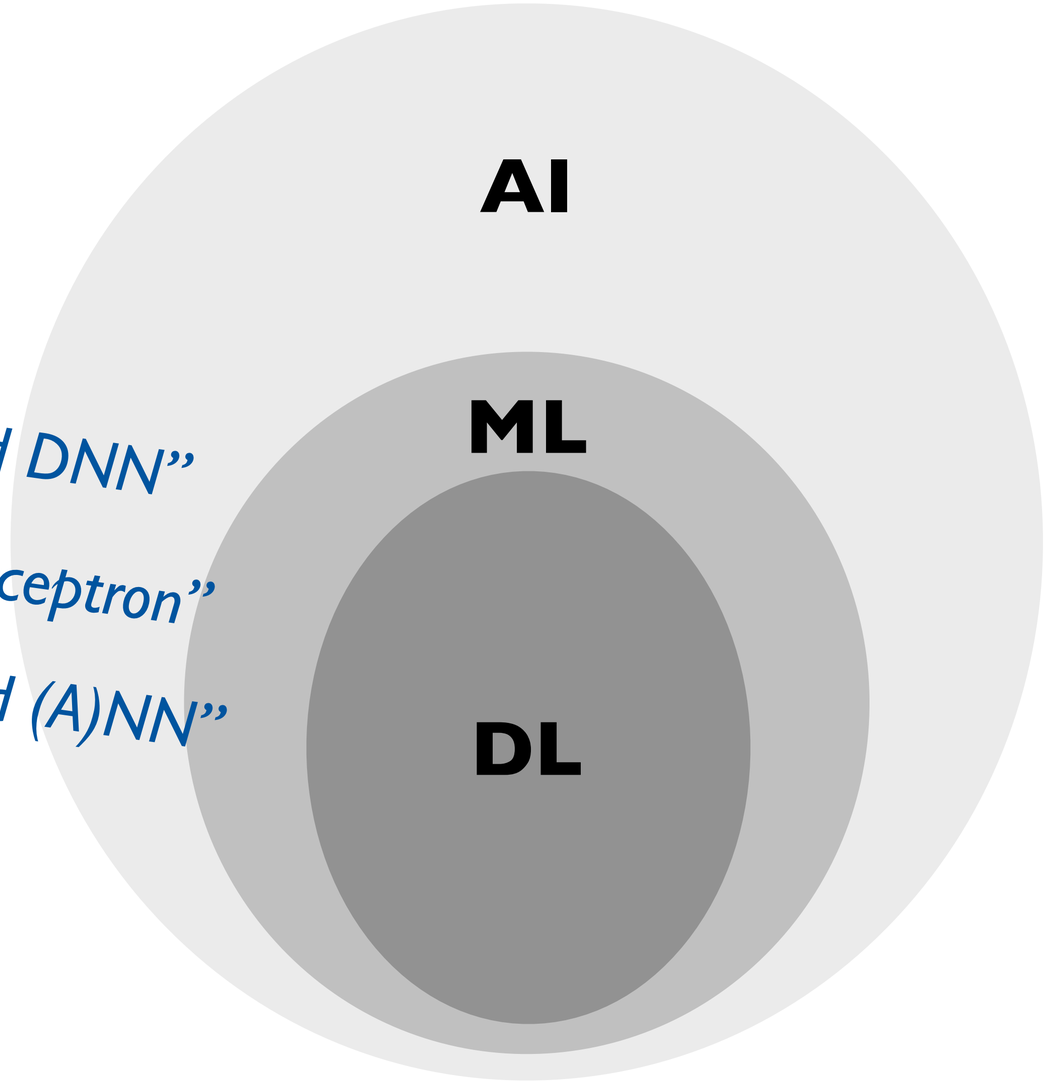


"fully-connected DNN"

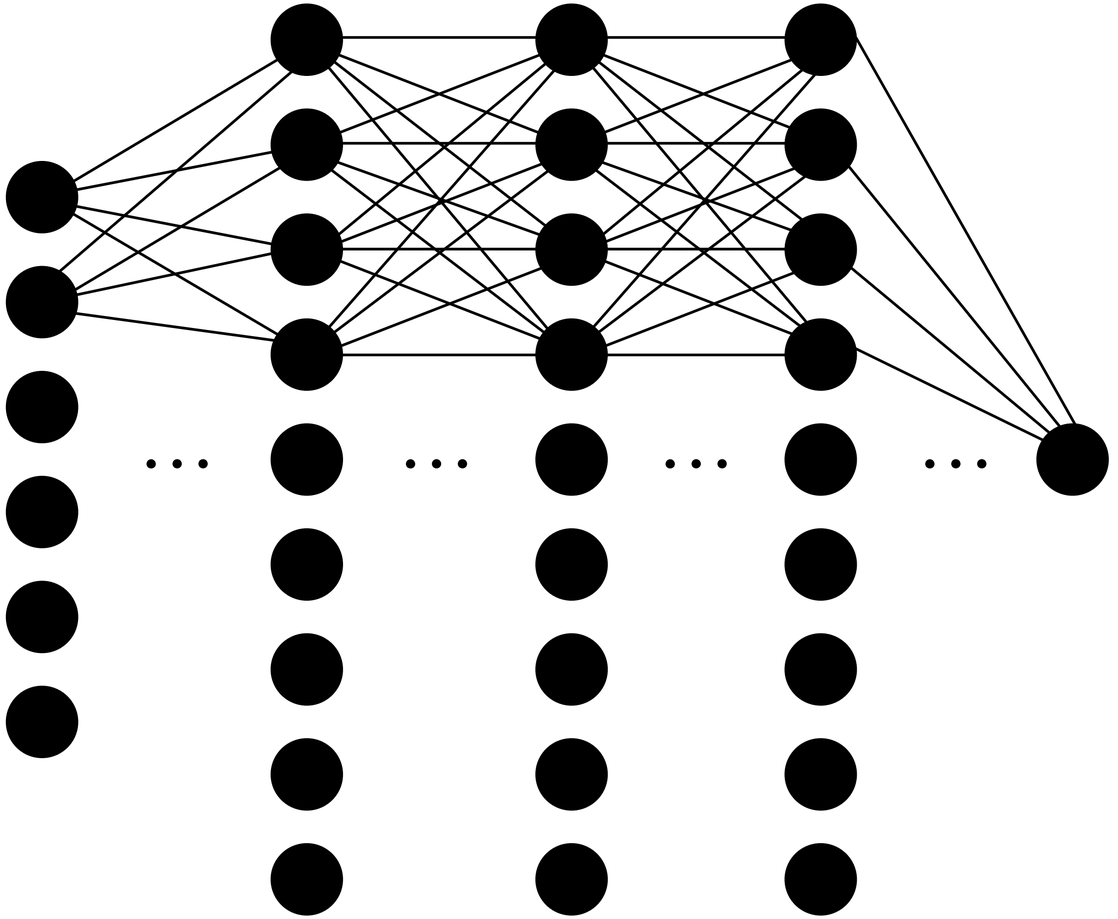
"multilayer perceptron"

"feedforward (A)NN"

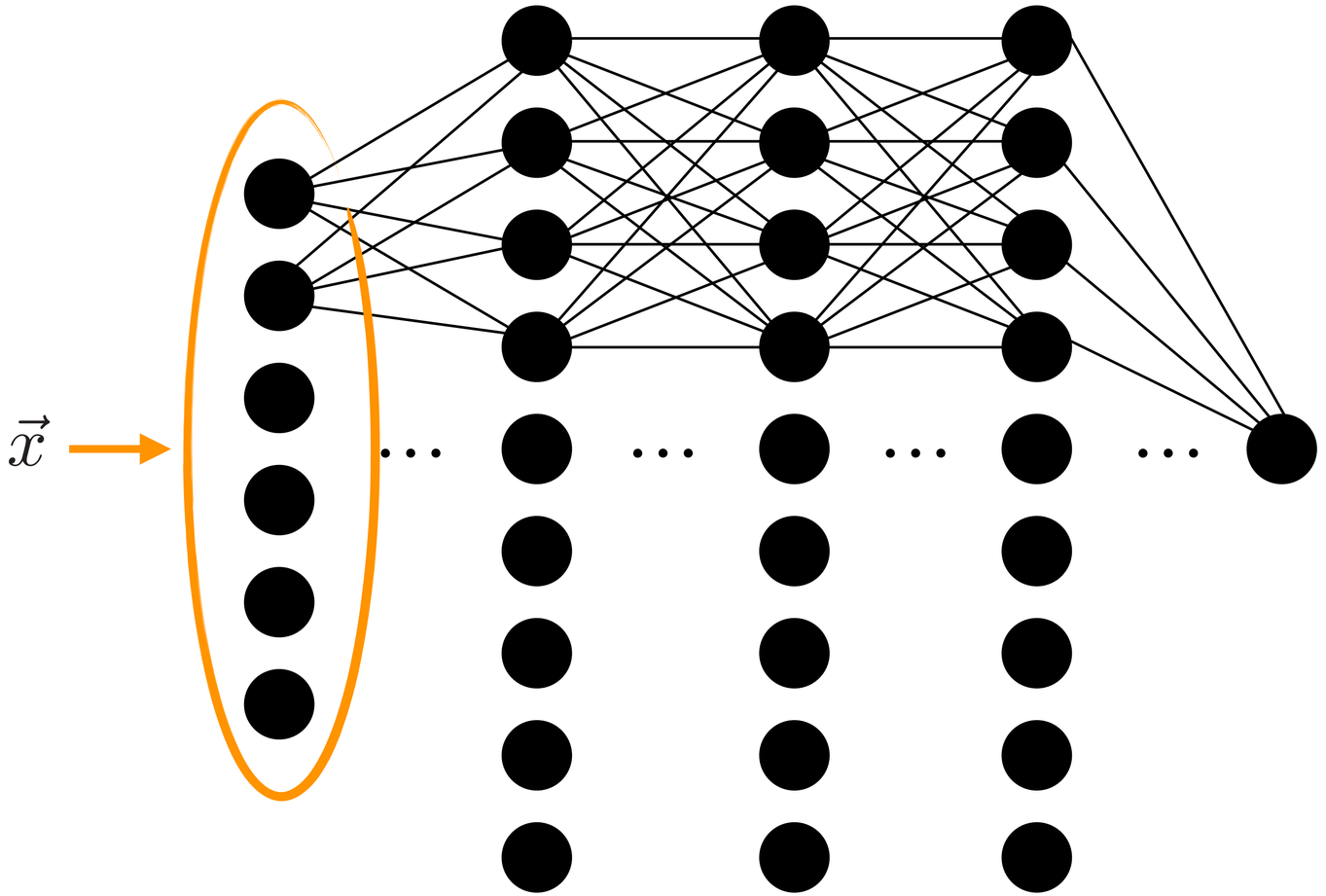
…



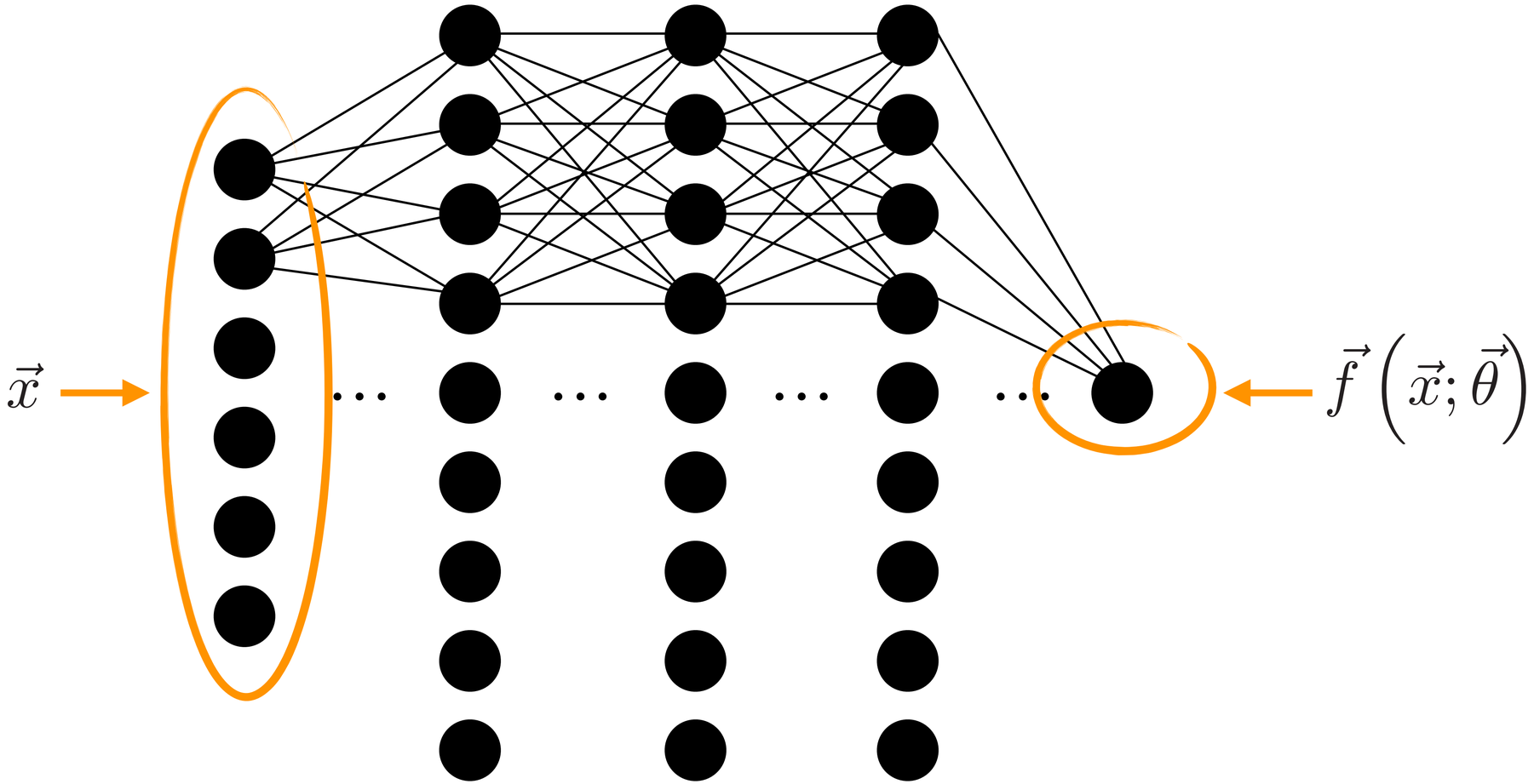
Deep Neural Networks



Deep Neural Networks

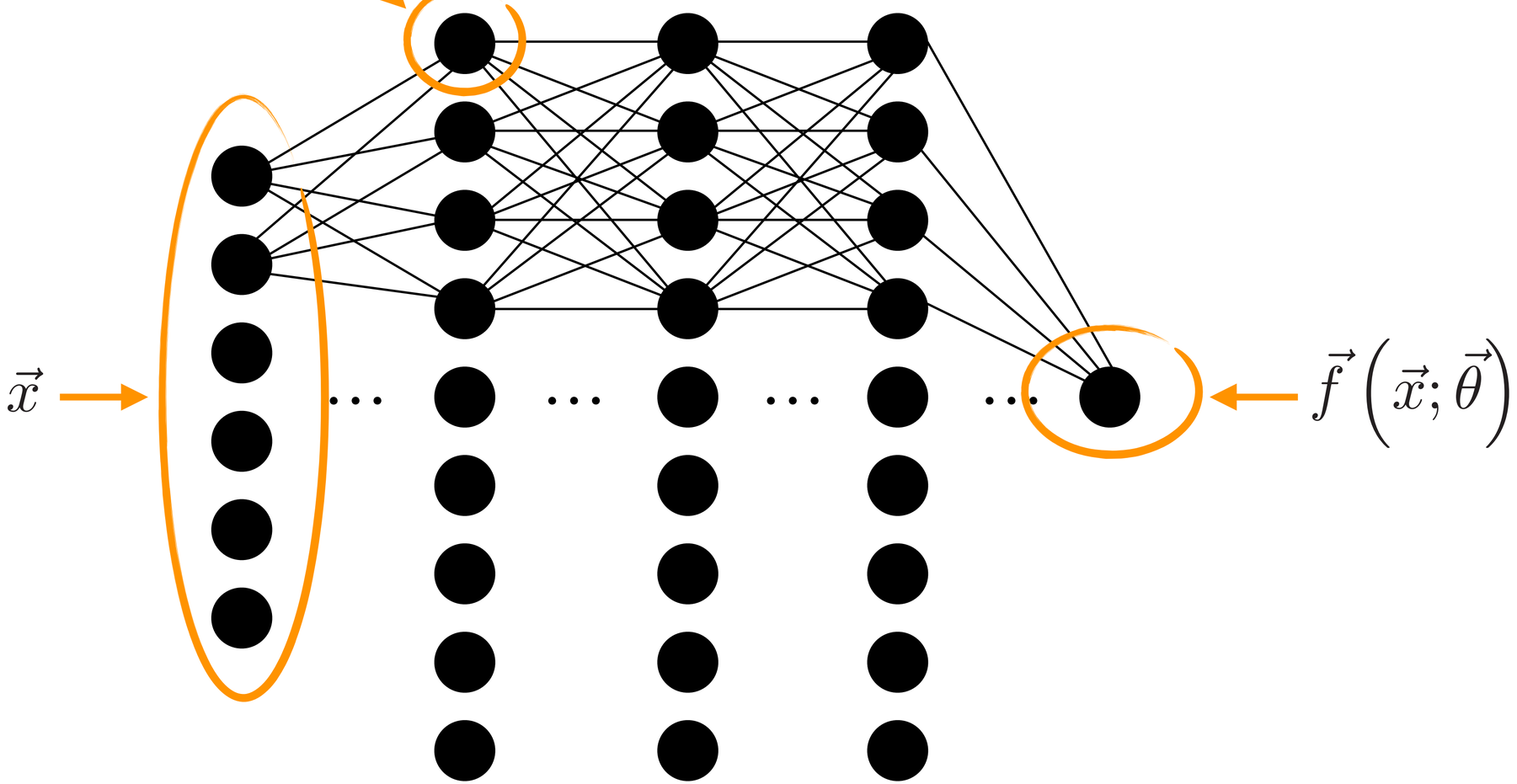


Deep Neural Networks



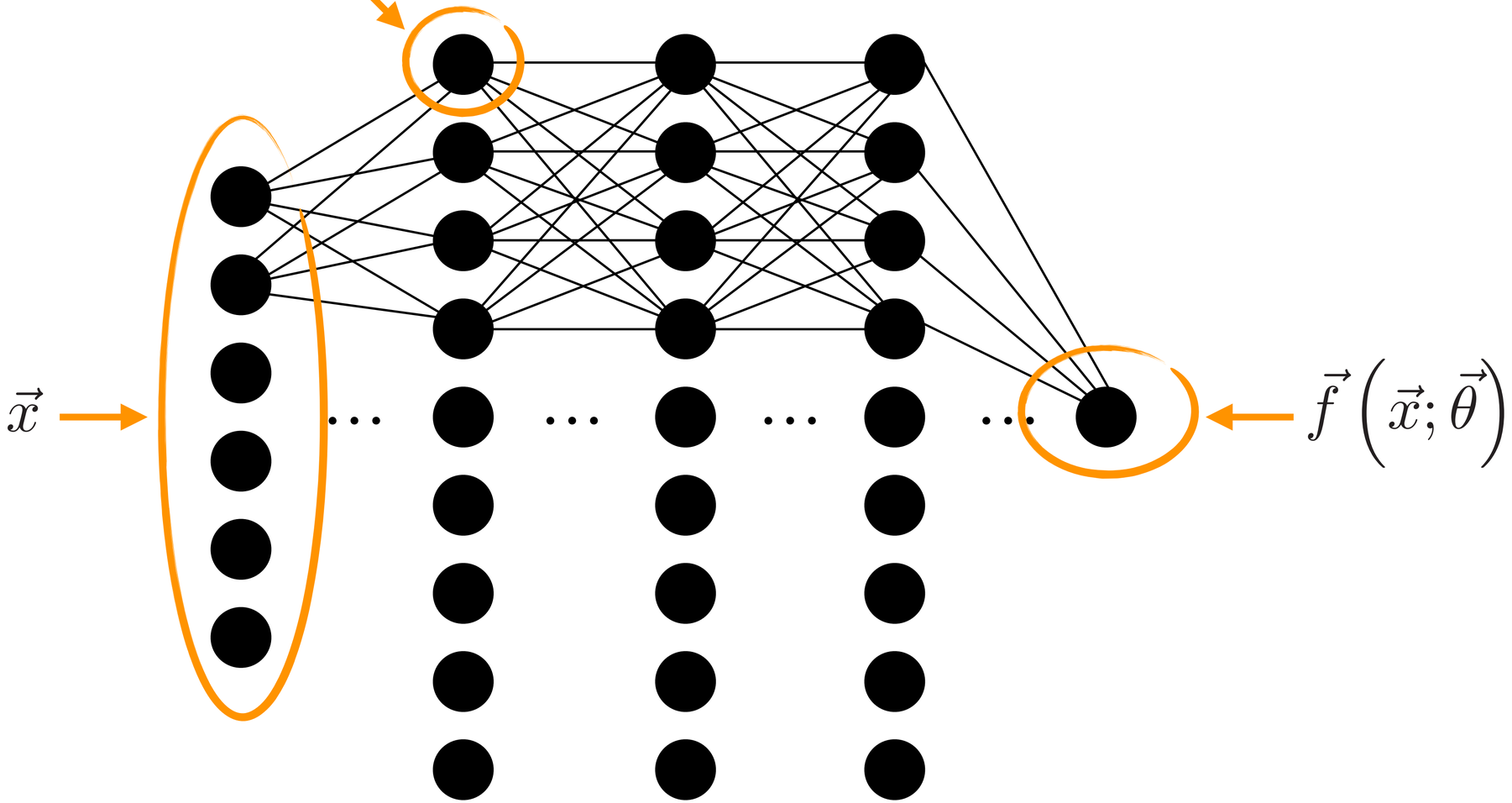
Deep Neural Networks

$$f_i^{(k)} = \sum_j w_{ij}^{(k-1)} f_j^{(k-1)}$$



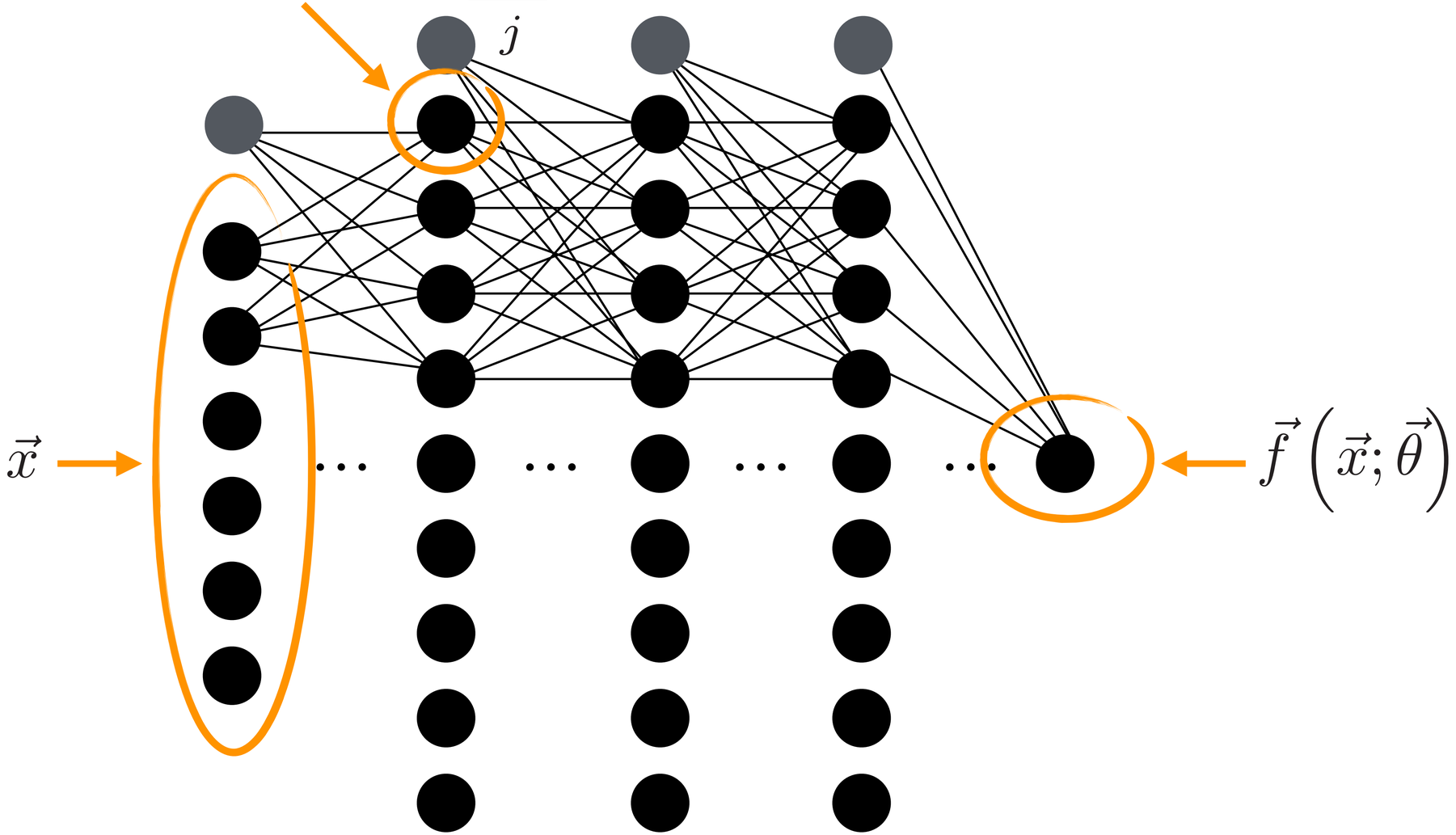
Deep Neural Networks

$$f_i^{(k)} = \sum_j w_{ij}^{(k-1)} f_j^{(k-1)} + b_i^{(k-1)}$$



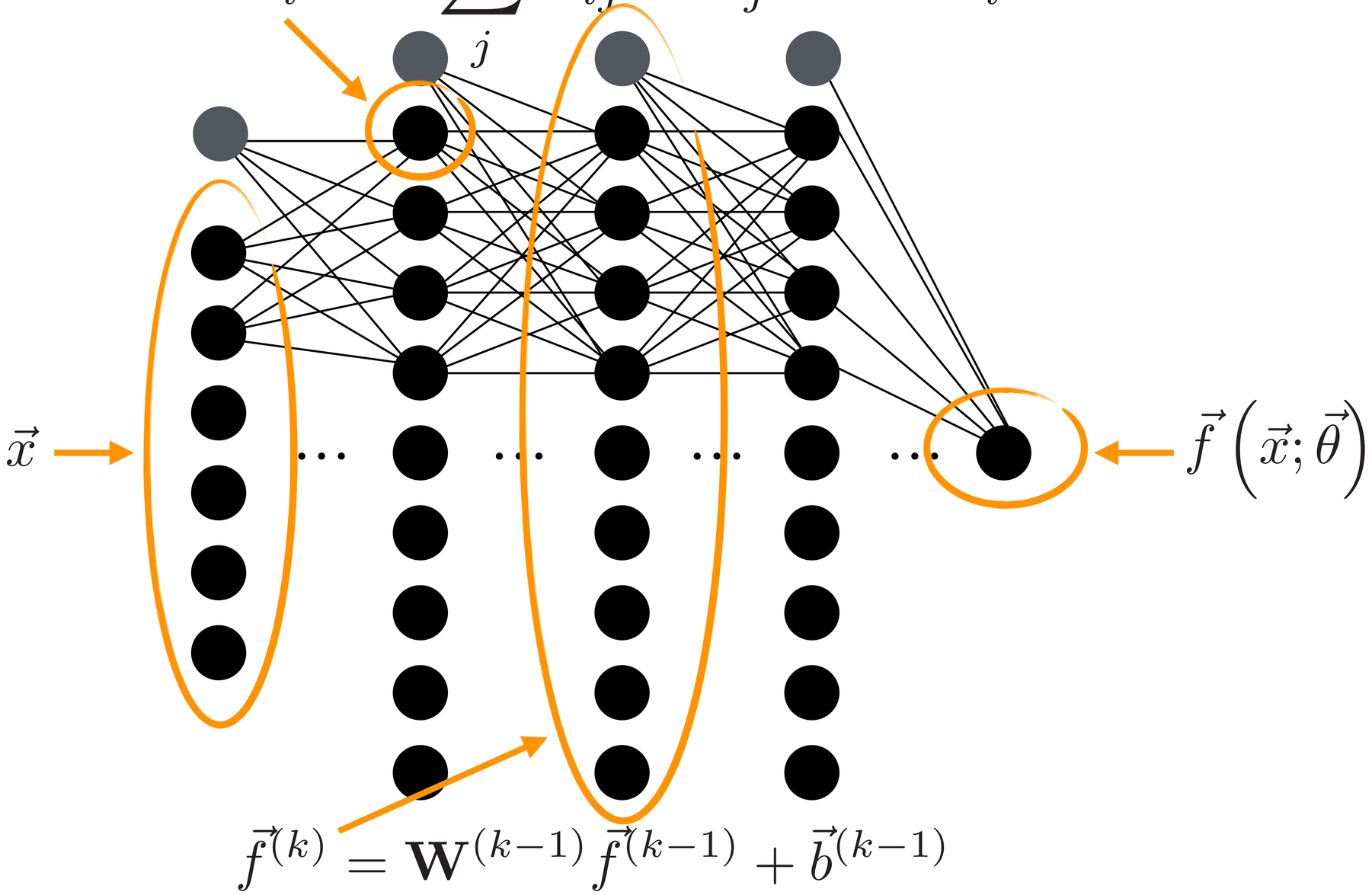
Deep Neural Networks

$$f_i^{(k)} = \sum_j w_{ij}^{(k-1)} f_j^{(k-1)} + b_i^{(k-1)}$$



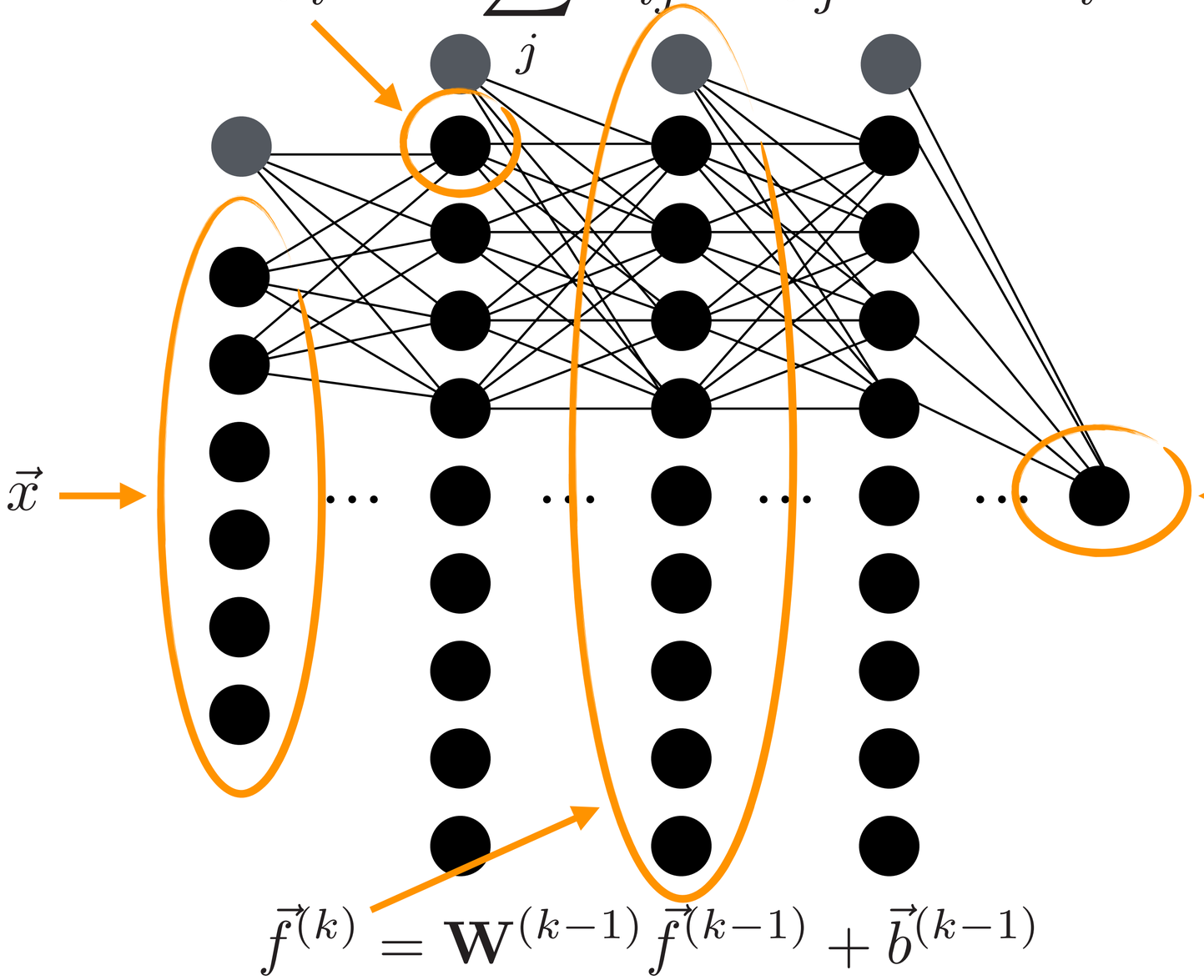
Deep Neural Networks

$$f_i^{(k)} = \sum_j w_{ij}^{(k-1)} f_j^{(k-1)} + b_i^{(k-1)}$$



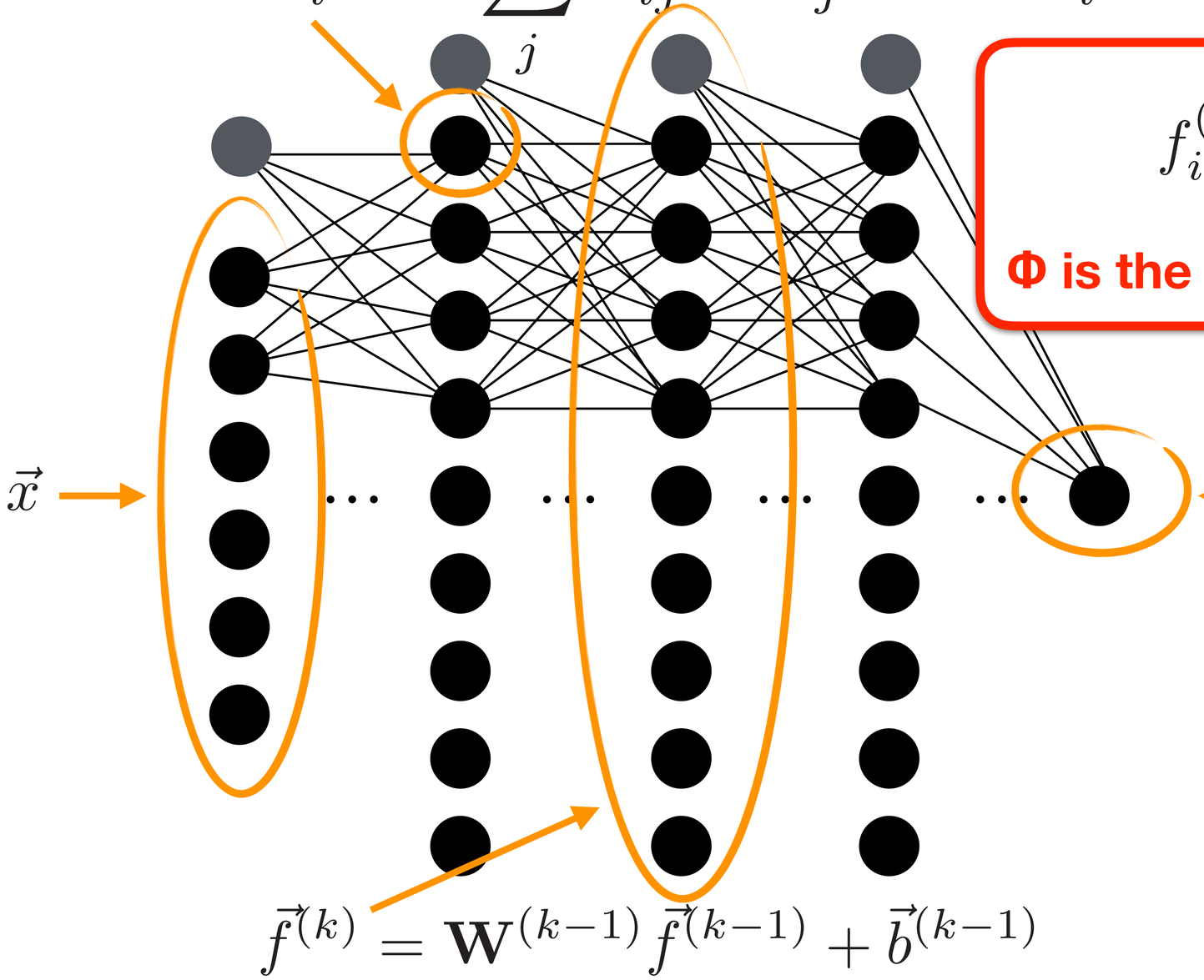
Deep Neural Networks

$$f_i^{(k)} = \sum_j w_{ij}^{(k-1)} f_j^{(k-1)} + b_i^{(k-1)}$$



Deep Neural Networks

$$f_i^{(k)} = \sum_j w_{ij}^{(k-1)} f_j^{(k-1)} + b_i^{(k-1)}$$



$$f_i^{(k)} \rightarrow \phi(f_i^{(k)})$$

Φ is the "activation function"

$$\vec{f}\left(\vec{x};\vec{\theta}\right)$$

$$\vec{f}\left(\vec{x};\{\mathbf{W}\},\left\{\vec{b}\right\}\right)$$

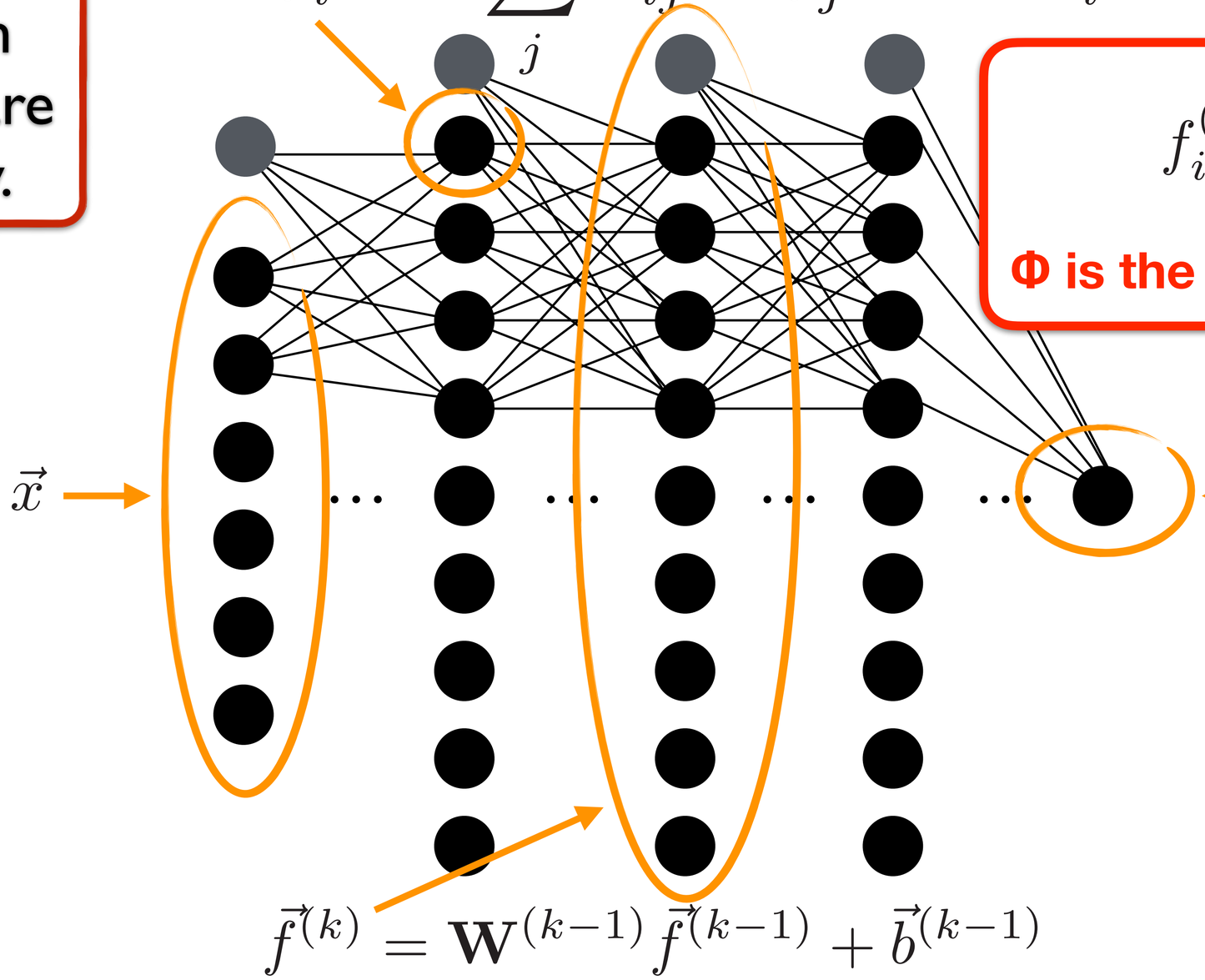
Deep Neural Networks

Non-linear activation functions are really key.

$$f_i^{(k)} = \sum_j w_{ij}^{(k-1)} f_j^{(k-1)} + b_i^{(k-1)}$$

$$f_i^{(k)} \rightarrow \phi(f_i^{(k)})$$

Φ is the "activation function"

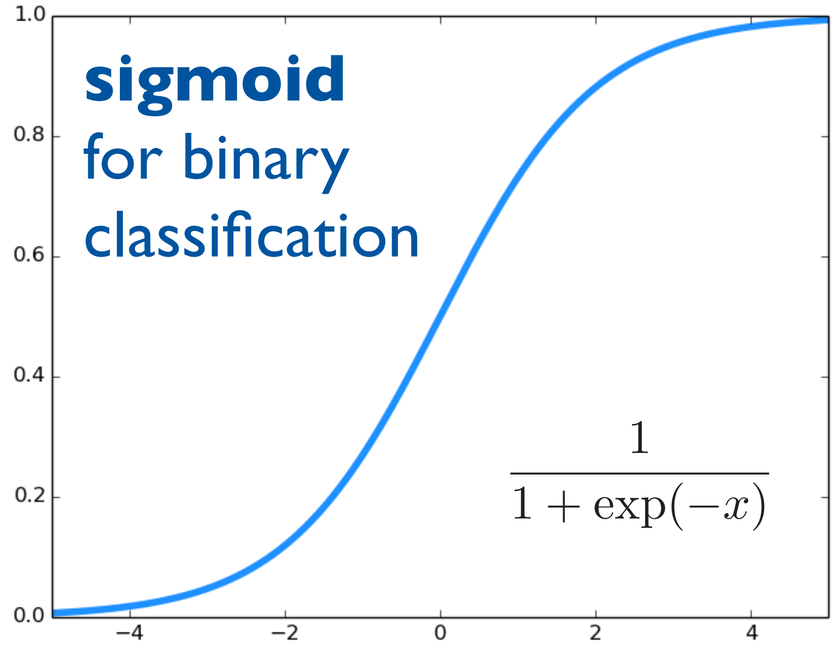


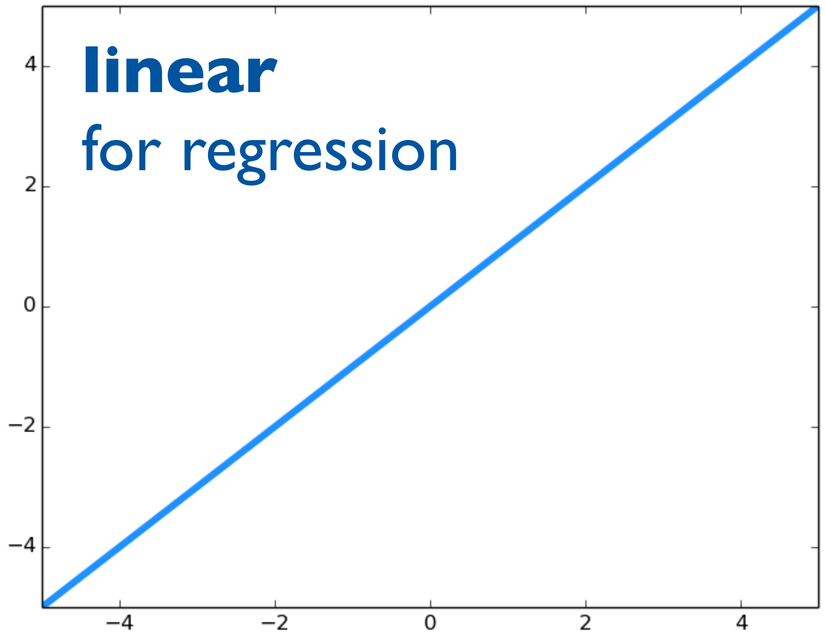
$$\vec{f}\left(\vec{x};\vec{\theta}\right)$$

$$\vec{f}\left(\vec{x};\{\mathbf{W}\},\left\{\vec{b}\right\}\right)$$

The Output Function(s)

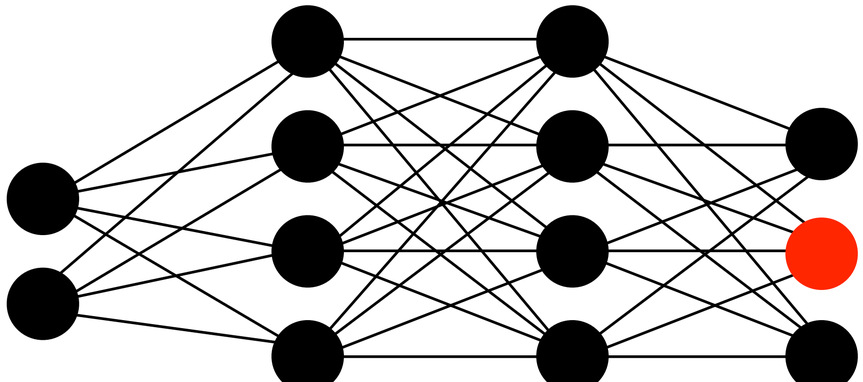
- Output function(s) = activation(s) in the output layer
- Closely connected to loss (and ultimately the task)





softmax of output node *i*

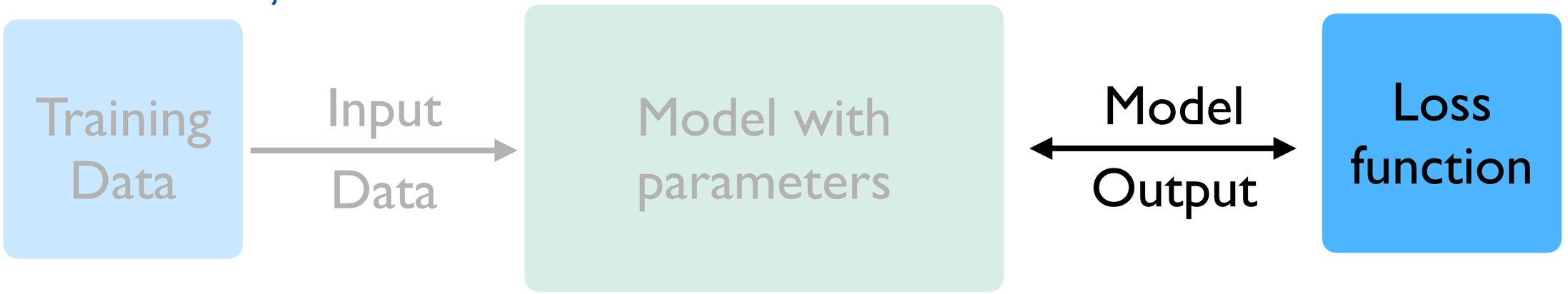
for classification of *N* classes



$$\frac{\exp(x_i)}{\sum_{j=1}^{N}\exp(x_j)}$$

The Loss Function*

* aka "cost function" aka "objective function"



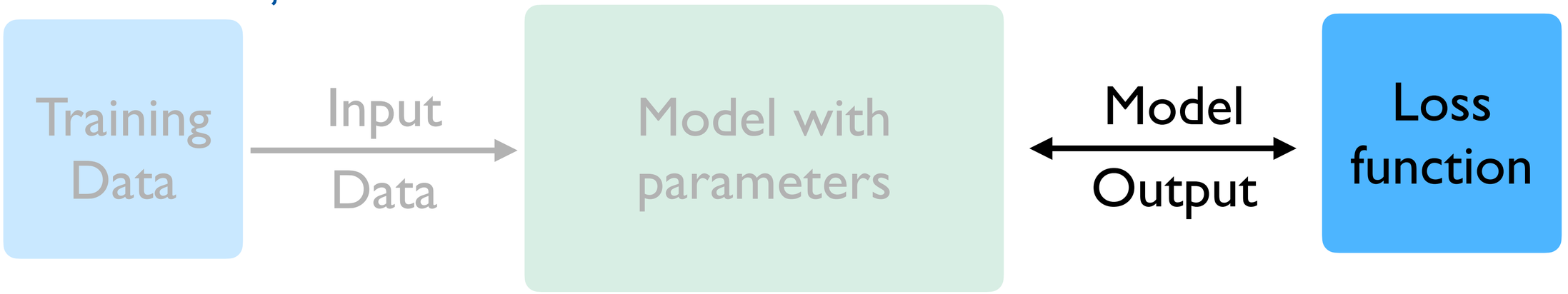
- How good is my current model?
- How to change the parameters to improve it?
  - Loss function must be easily differentiable!
- Regression:

  Mean Squared Error (MSE)

$$\frac{1}{n}\sum_{i=1}^{n}\left[\vec{y}_i - \vec{f}\left(\vec{x}_i; \{\mathbf{W}\}, \left\{\vec{b}\right\}\right)\right]^2$$

The Loss Function*

* aka "cost function" aka "objective function"



- How good is my current model?
- How to change the parameters to improve it?
  - Loss function must be easily differentiable!
- Classification:

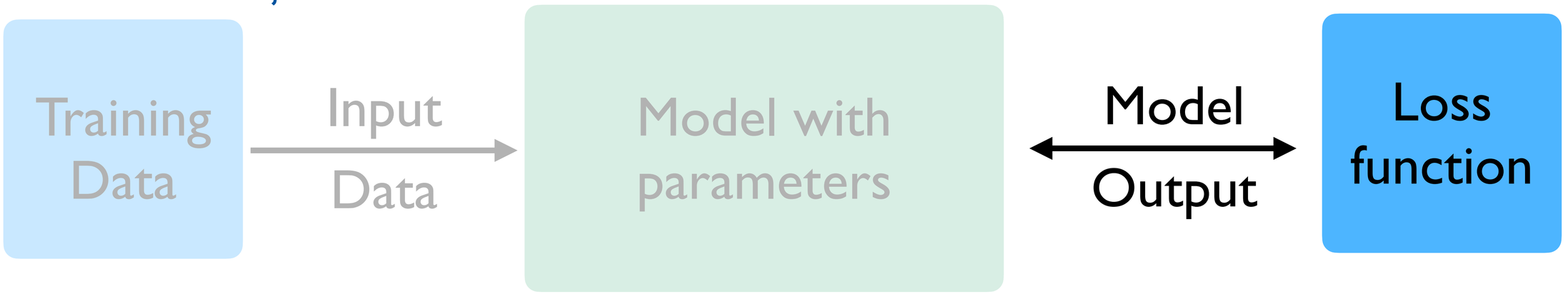
  Cross-entropy of prediction f (= interpreted as probability q) and the true probability p (= the classification labels y)

$$-\frac{1}{n}\sum_{i=1}^{n}\left[\sum_{\text{classes}} y_{i,\text{class}} \cdot \log f_{\text{class}}\left(\vec{x}_i; \{\mathbf{W}\}, \left\{\vec{b}\right\}\right)\right]$$

$$-\frac{1}{n}\sum_{i=1}^{n}\left[\sum_{\text{classes}} p_{i,\text{class}} \log q_{i,\text{class}}\right]$$

The Loss Function*

* aka "cost function" aka "objective function"



- How good is my current model?
- How to change the parameters to improve it?
  - Loss function must be easily differentiable!
- Classification:

  Cross-entropy of prediction f (= interpreted as probability q) and the true probability p (= the classification labels y)

for binary classification

$$-\frac{1}{n}\sum_{i=1}^{n}\begin{cases}\log q_i\,, & \text{signal}\\ \log\left(1-q_i\right)\,, & \text{background}\end{cases}$$

$$-\frac{1}{n}\sum_{i=1}^{n}\left[\sum_{\text{classes}} p_{i,\text{class}}\log q_{i,\text{class}}\right]$$

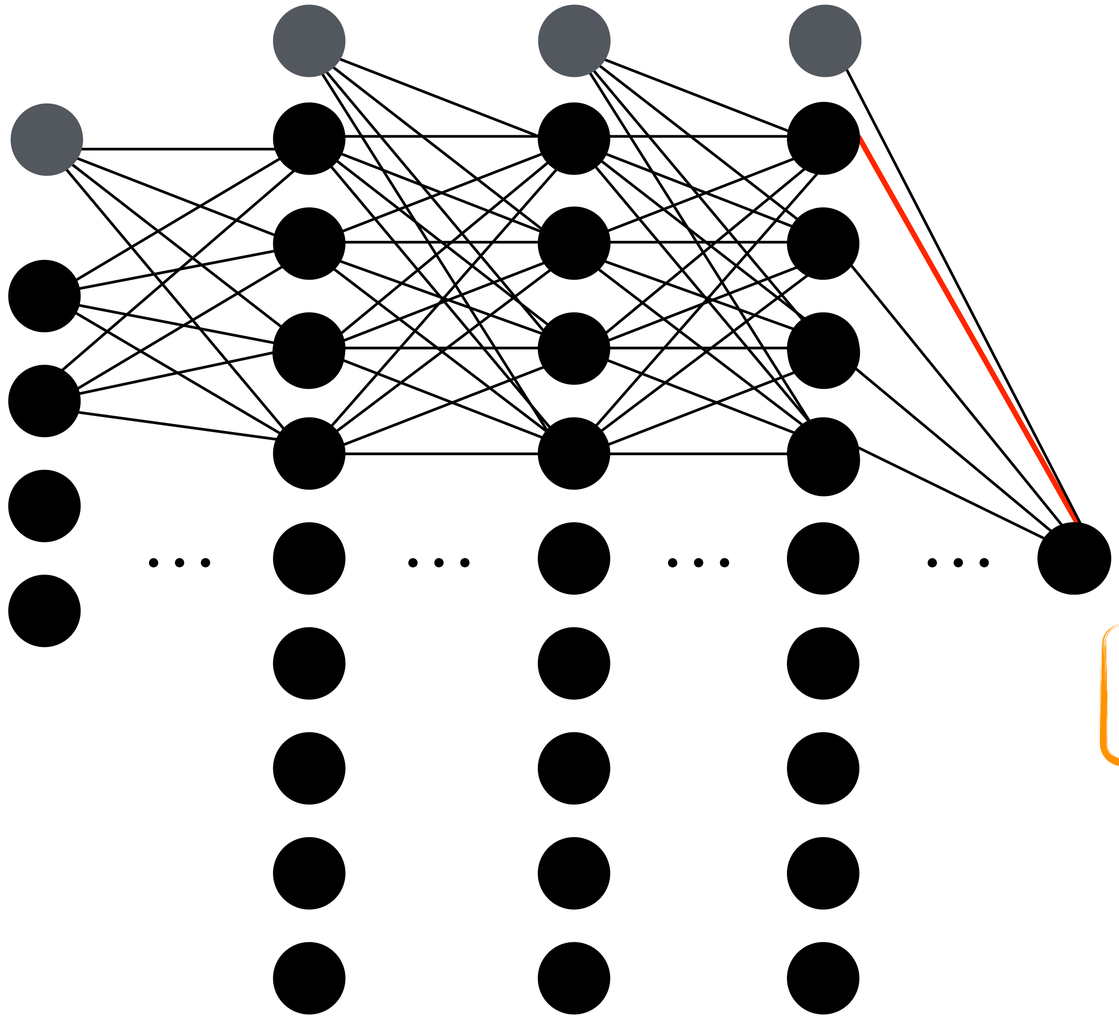
Optimization via Backpropagation

- We need the gradient w.r.t. each optimizable parameter
- Weights to last layer (output node):

$$\left.\frac{\partial \text{Loss}}{\partial \theta}\right|_{\text{last layer}} = \left.\frac{\partial \text{Loss}}{\partial \phi}\frac{\partial \phi}{\partial f}\frac{\partial f}{\partial \theta}\right|_{\text{last layer}}$$

$$f_i^{(k)} = \sum_j w_{ij}^{(k-1)} f_j^{(k-1)} + b_i^{(k-1)}$$

$$f_i^{(k)} \to \phi(f_i^{(k)})$$



$$\vec{f}\left(\vec{x};\vec{\theta}\right)$$

Optimization via Backpropagation

- We need the gradient w.r.t. each optimizable parameter
- Weights to last layer (output node):

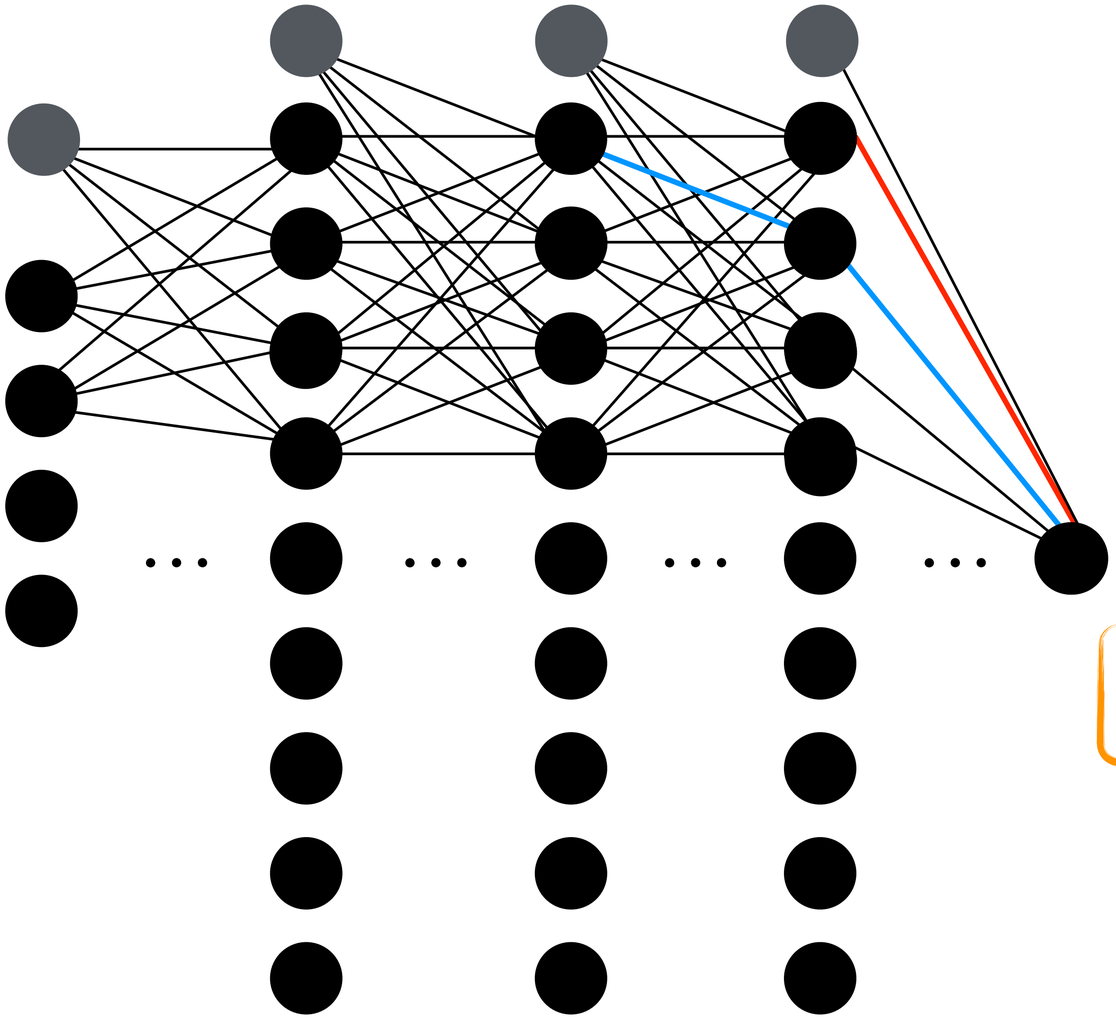
$$\left.\frac{\partial \text{Loss}}{\partial \theta}\right|_{\text{last layer}} = \left.\frac{\partial \text{Loss}}{\partial \phi}\frac{\partial \phi}{\partial f}\frac{\partial f}{\partial \theta}\right|_{\text{last layer}}$$

- Weights to 2nd last layer

$$\left.\frac{\partial \text{Loss}}{\partial \theta}\right|_{\text{2nd last layer}} = \left.\frac{\partial \text{Loss}}{\partial \phi}\cdot\frac{\partial \phi}{\partial f}\right|_{\text{last layer}}\cdot\frac{\partial f_{\text{last layer}}}{\partial \phi_{\text{2nd last layer}}} \cdot \left.\frac{\partial \phi}{\partial f}\cdot\frac{\partial f}{\partial \theta}\right|_{\text{2nd last layer}}$$

$$f_i^{(k)} = \sum_j w_{ij}^{(k-1)} f_j^{(k-1)} + b_i^{(k-1)}$$

$$f_i^{(k)} \to \phi(f_i^{(k)})$$



$$\vec{f}\left(\vec{x};\vec{\theta}\right)$$

Optimization via Backpropagation

- We need the gradient w.r.t. each optimizable parameter
- Weights to last layer (output node):

$$\left.\frac{\partial \text{Loss}}{\partial \theta}\right|_{\text{last layer}} = \left.\frac{\partial \text{Loss}}{\partial \phi}\frac{\partial \phi}{\partial f}\frac{\partial f}{\partial \theta}\right|_{\text{last layer}}$$

- Weights to 2nd last layer

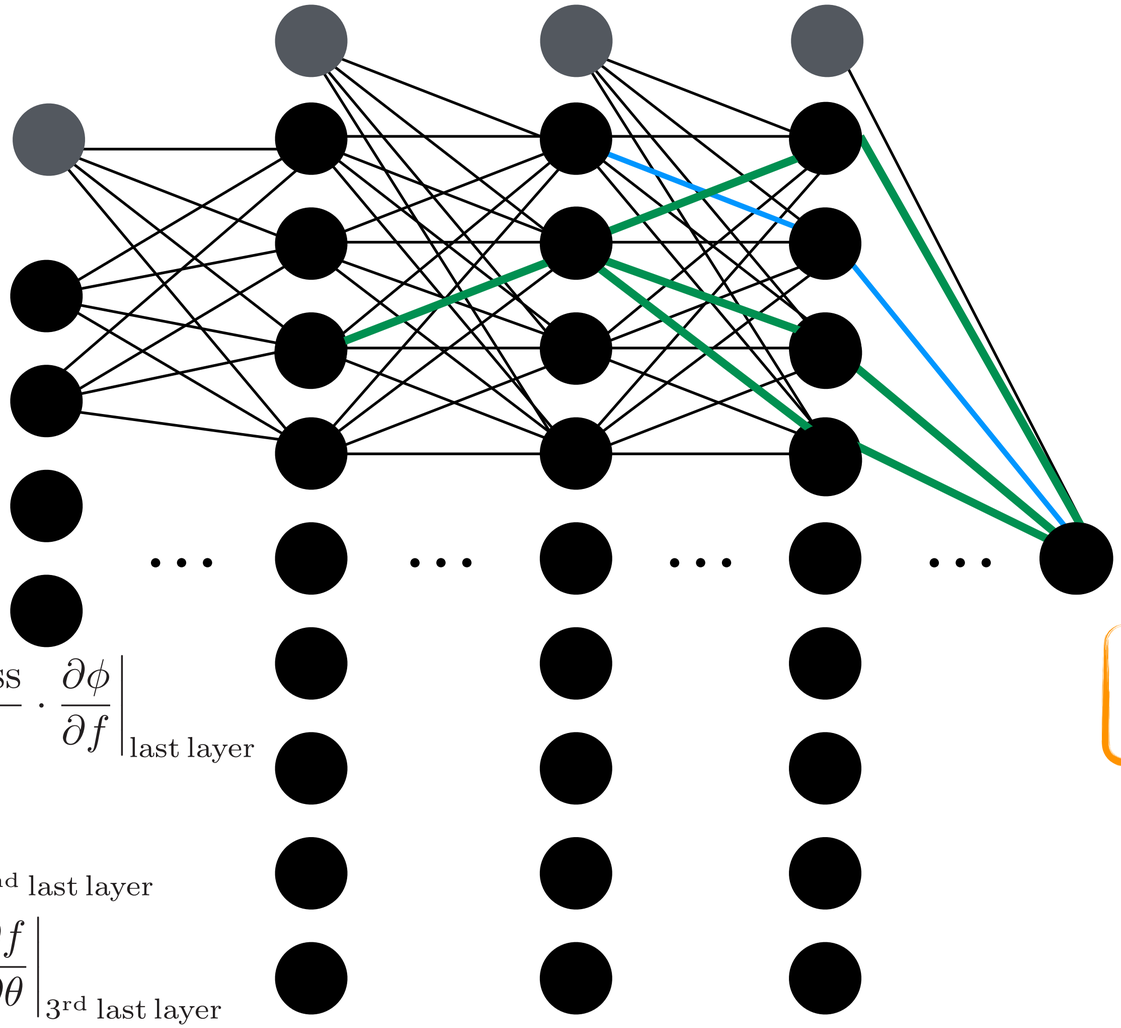
$$\left.\frac{\partial \text{Loss}}{\partial \theta}\right|_{\text{2nd last layer}} = \left.\frac{\partial \text{Loss}}{\partial \phi}\cdot\frac{\partial \phi}{\partial f}\right|_{\text{last layer}}\cdot\frac{\partial f_{\text{last layer}}}{\partial \phi_{\text{2nd last layer}}} \cdot\left.\frac{\partial \phi}{\partial f}\cdot\frac{\partial f}{\partial \theta}\right|_{\text{2nd last layer}}$$

- Weights to 3rd last layer

$$\left.\frac{\partial \text{Loss}}{\partial \theta}\right|_{\text{3rd last layer}} = \sum_{\text{nodes in 2nd last layer}} \left.\frac{\partial \text{Loss}}{\partial \phi}\cdot\frac{\partial \phi}{\partial f}\right|_{\text{last layer}} \cdot\frac{\partial f_{\text{last layer}}}{\partial \phi_{\text{2nd last layer}}}\cdot\left.\frac{\partial \phi}{\partial f}\right|_{\text{2nd last layer}} \cdot\frac{\partial f_{\text{2nd last layer}}}{\partial \phi_{\text{3rd last layer}}}\cdot\left.\frac{\partial \phi}{\partial f}\cdot\frac{\partial f}{\partial \theta}\right|_{\text{3rd last layer}}$$

$$f_i^{(k)} = \sum_j w_{ij}^{(k-1)} f_j^{(k-1)} + b_i^{(k-1)}$$

$$f_i^{(k)} \to \phi(f_i^{(k)})$$



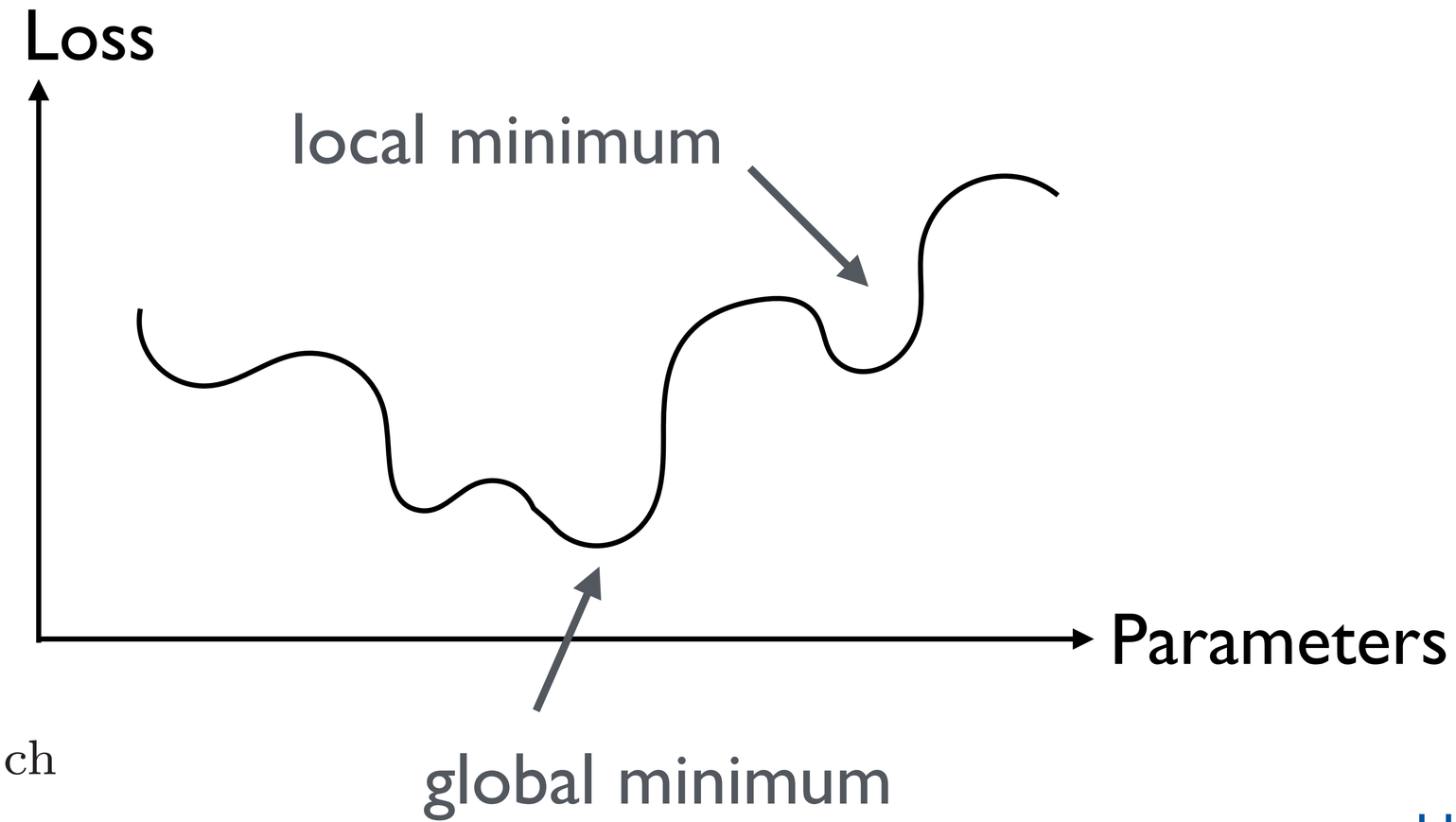
$$\vec{f}\left(\vec{x};\vec{\theta}\right)$$

Stochastic Gradient Descent

- Ideal: Converge to the parameters of the global minimum of the loss function
- Practical: Converge to a "good-enough" local minimum
- Bad: Converge to "some" local minimum

- SGD: gradient estimated on a mini batch of training data
  - Variance in gradient estimates
- Learning rate α:

$$\theta \rightarrow \theta - \alpha \left\langle \frac{\partial \mathrm{Loss}}{\partial \theta} \right\rangle \Bigg|_{\text{mini batch}}$$

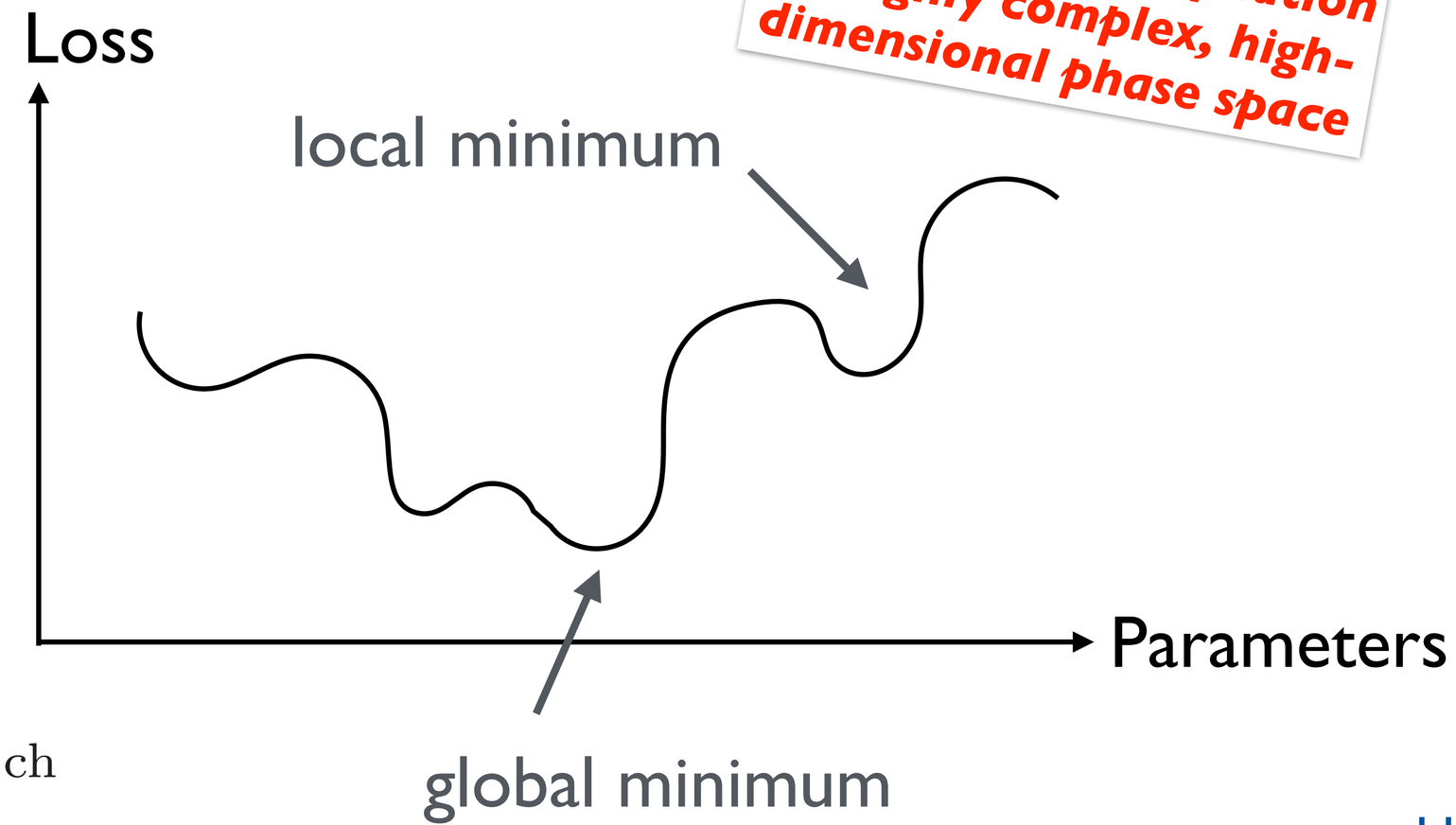


Stochastic Gradient Descent

- Ideal: Converge to the parameters of the global minimum of the loss function
- Practical: Converge to a "good-enough" local minimum
- Bad: Converge to "some" local minimum

- SGD: gradient estimated on a mini batch of training data
  - Variance in gradient estimates
- Learning rate α:

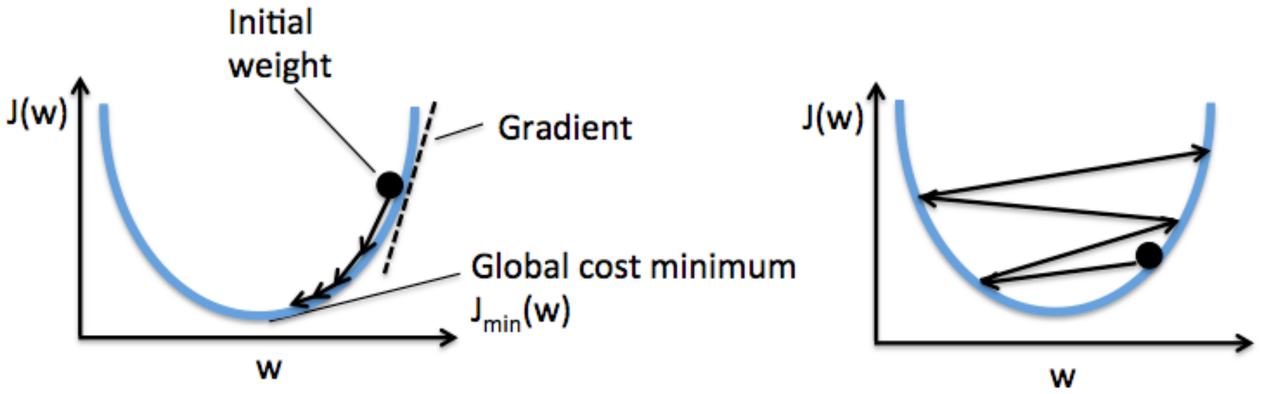
$$\theta \to \theta - \alpha \left\langle \frac{\partial \mathrm{Loss}}{\partial \theta} \right\rangle \Big|_{\text{mini batch}}$$



crude over-simplification of highly complex, high-dimensional phase space

Training Strategies

- Choosing a good learning rate is key for training convergence

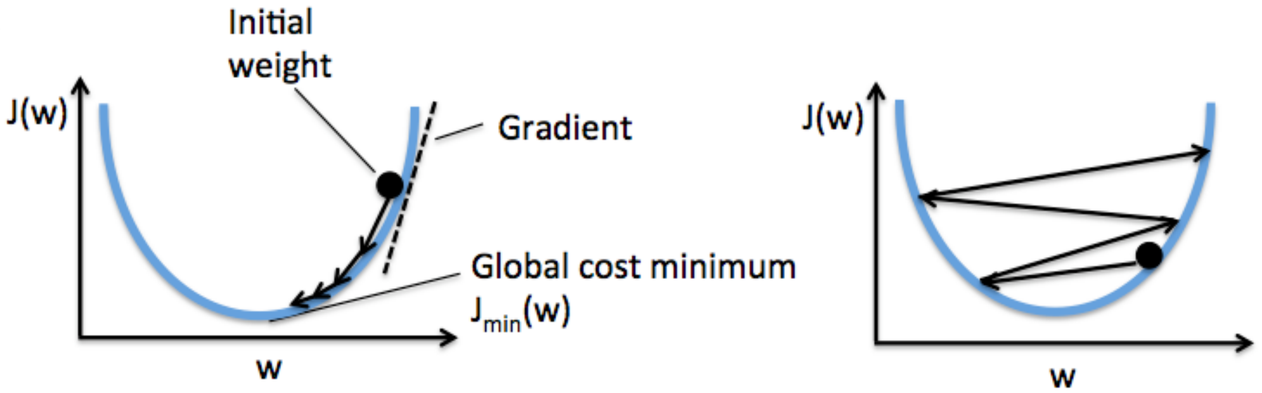


[https://raw.githubusercontent.com/rasbt/python-machine-learning-book/master/code/ch02/images/02_12.png, MIT License]

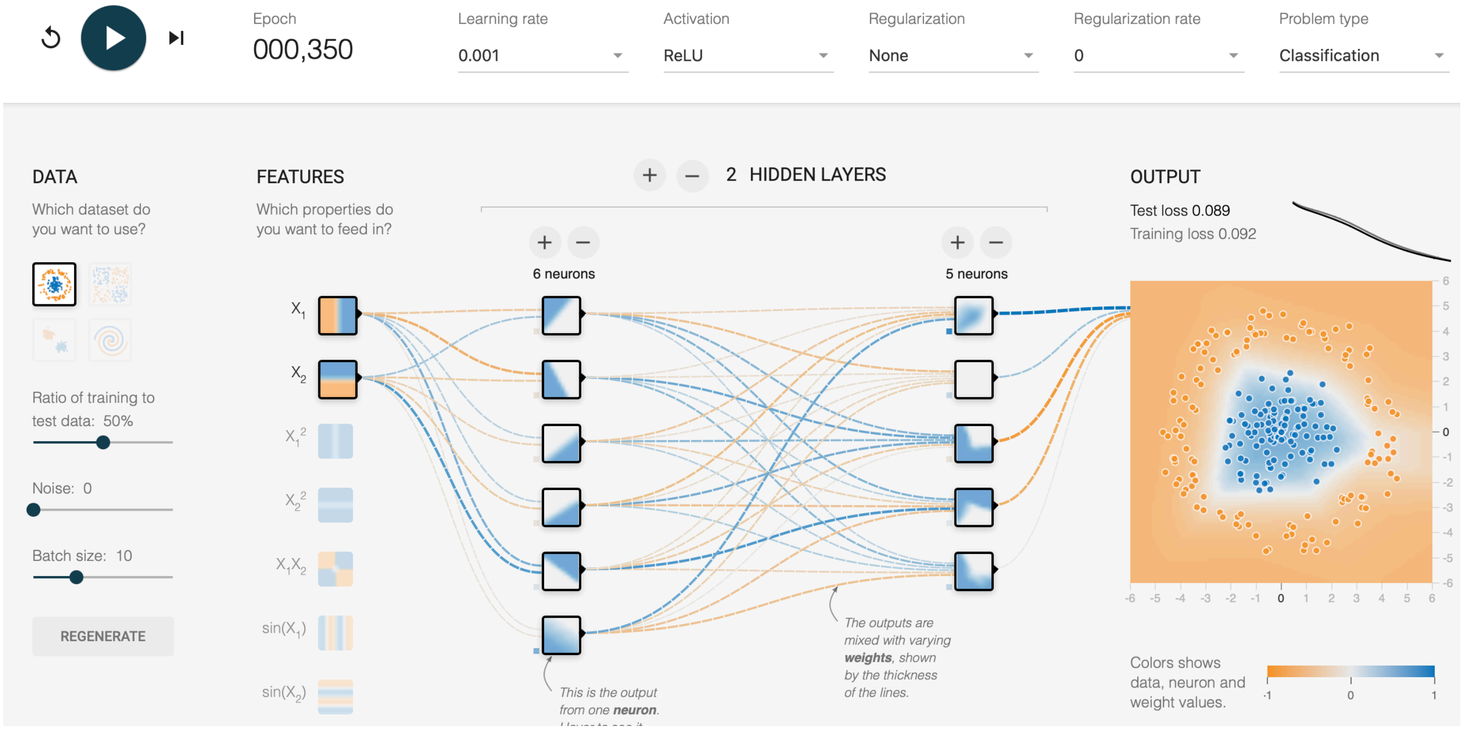
[https://playground.tensorflow.org, github: https://github.com/tensorflow/playground, Apache License 2.0]

Training Strategies

- Choosing a good learning rate is key for training convergence



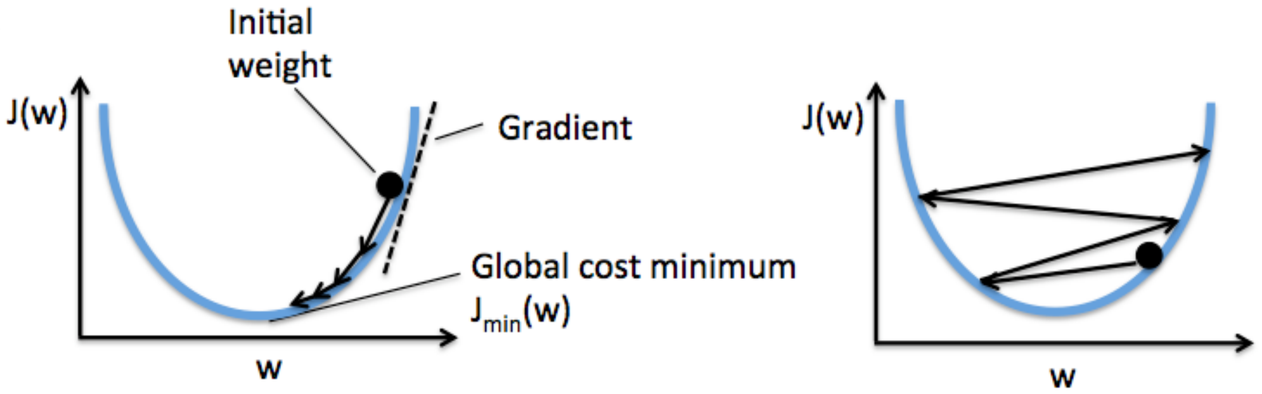
[https://raw.githubusercontent.com/rasbt/python-machine-learning-book/master/code/ch02/images/02_12.png, MIT License]



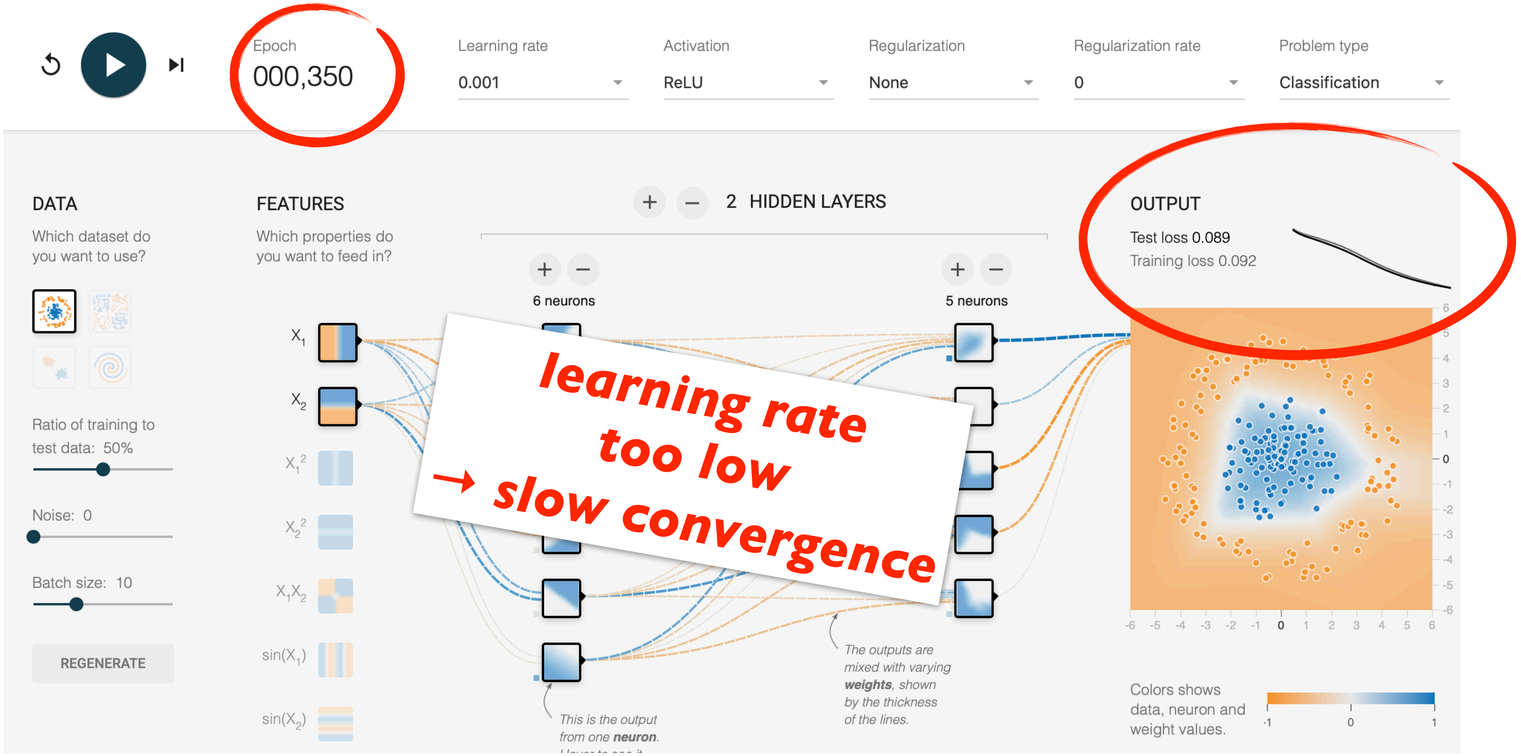
[https://playground.tensorflow.org, github: https://github.com/tensorflow/playground, Apache License 2.0]

Training Strategies

- Choosing a good learning rate is key for training convergence



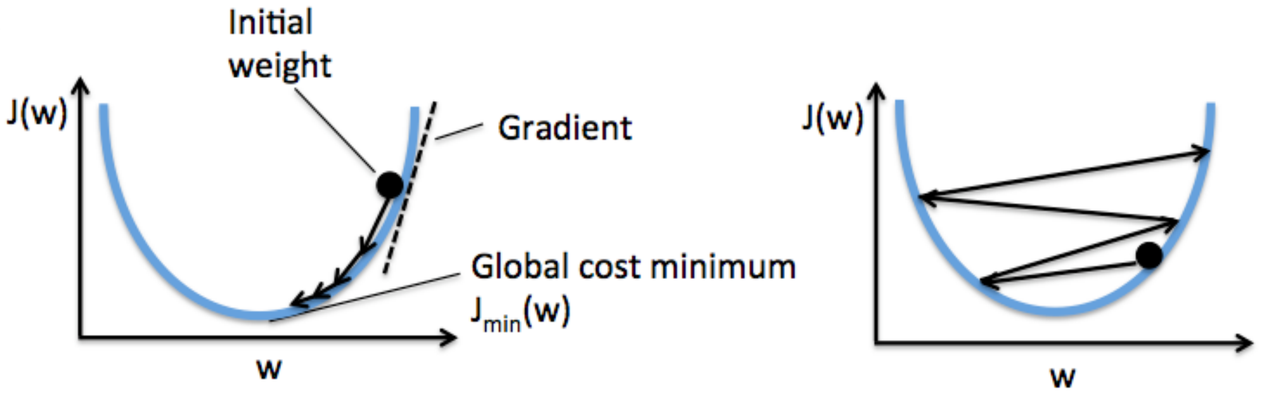
[https://raw.githubusercontent.com/rasbt/python-machine-learning-book/master/code/ch02/images/02_12.png, MIT License]



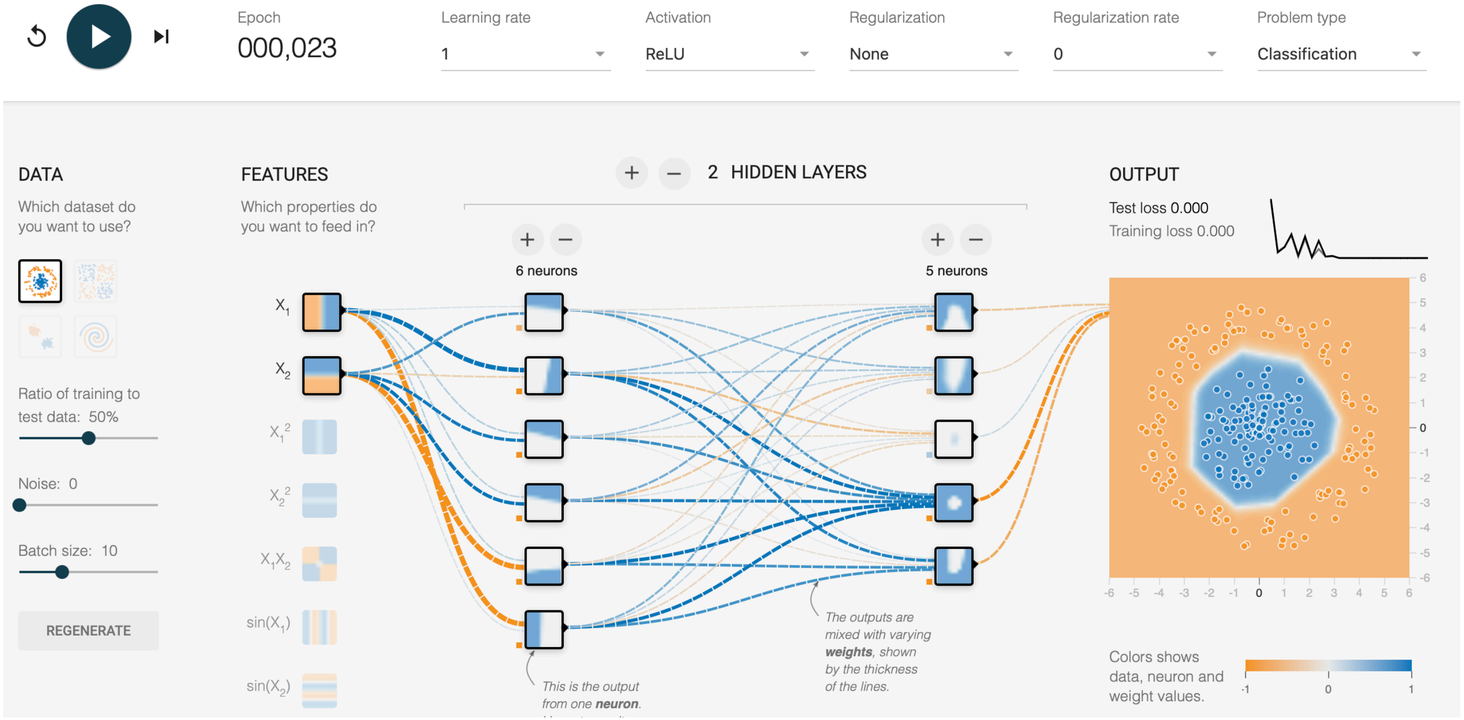
[https://playground.tensorflow.org, github: https://github.com/tensorflow/playground, Apache License 2.0]

Training Strategies

- Choosing a good learning rate is key for training convergence



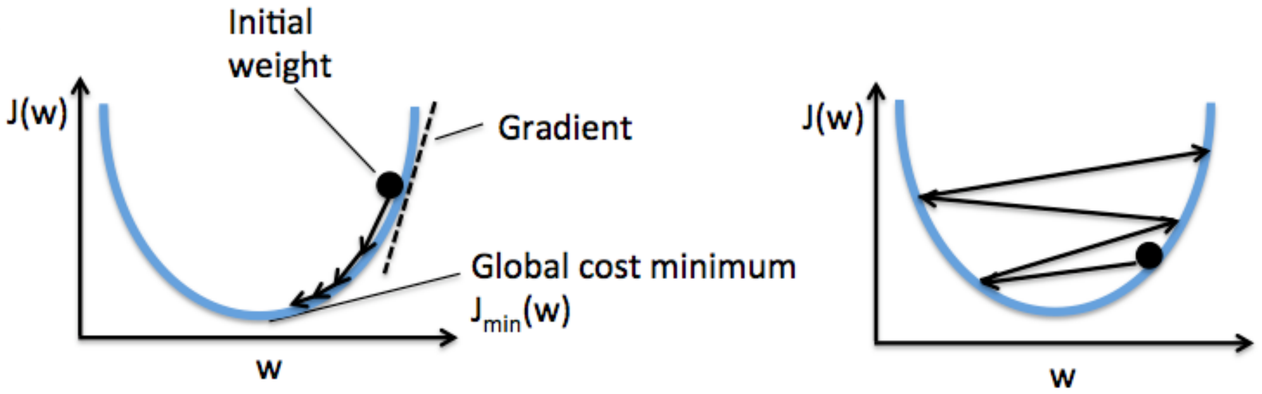
[https://raw.githubusercontent.com/rasbt/python-machine-learning-book/master/code/ch02/images/02_12.png, MIT License]



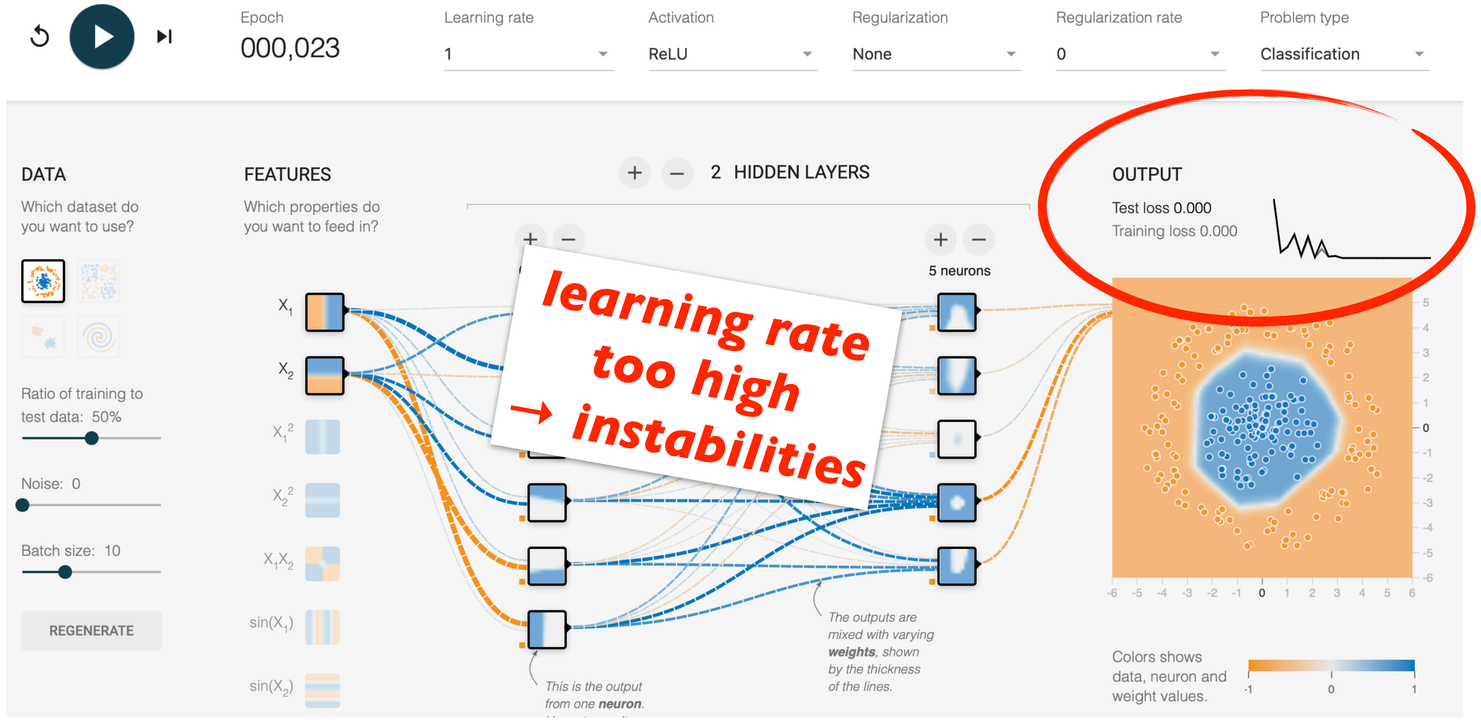
[https://playground.tensorflow.org, github: https://github.com/tensorflow/playground, Apache License 2.0]

Training Strategies

- Choosing a good learning rate is key for training convergence



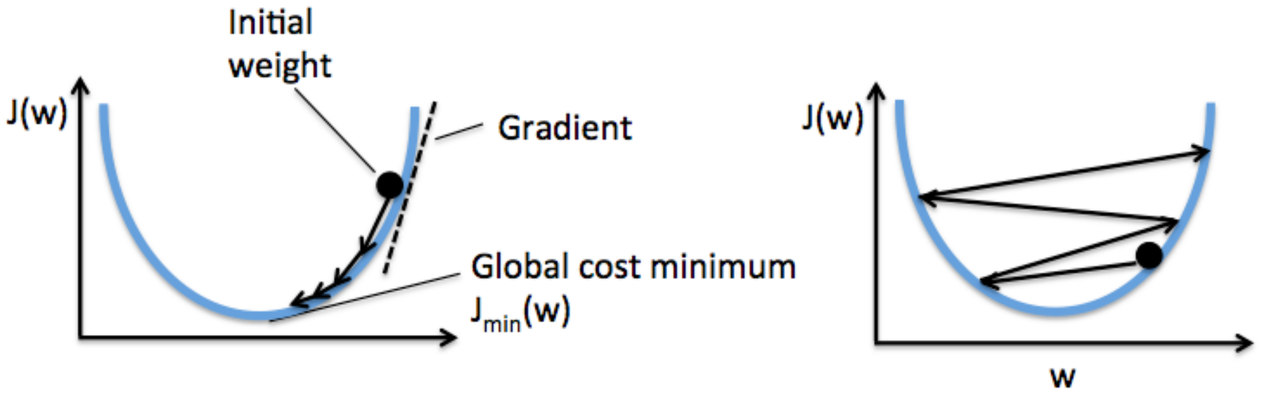
[https://raw.githubusercontent.com/rasbt/python-machine-learning-book/master/code/ch02/images/02_12.png, MIT License]



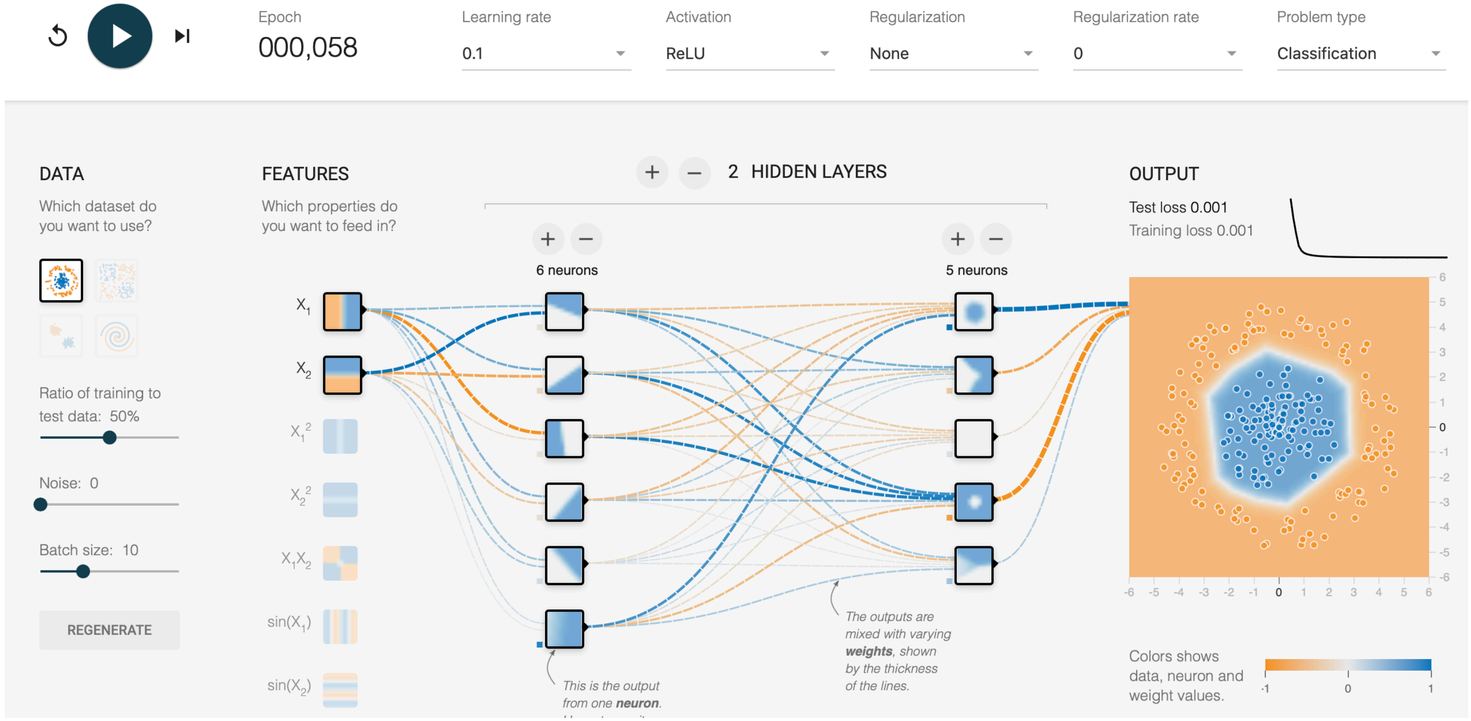
[https://playground.tensorflow.org, github: https://github.com/tensorflow/playground, Apache License 2.0]

Training Strategies

- Choosing a good learning rate is key for training convergence



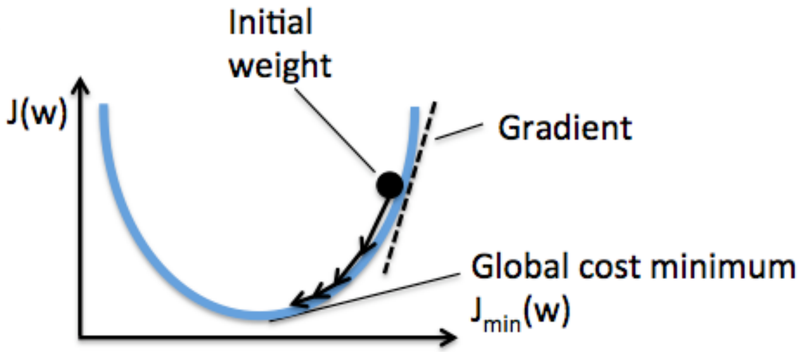
[https://raw.githubusercontent.com/rasbt/python-machine-learning-book/master/code/ch02/images/02_12.png, MIT License]

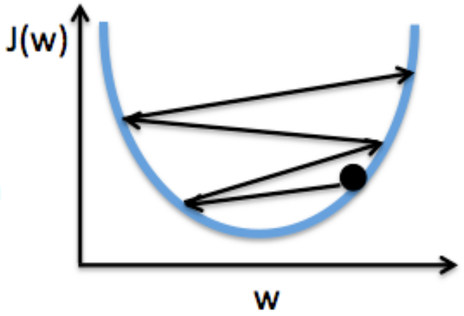


[https://playground.tensorflow.org, github: https://github.com/tensorflow/playground, Apache License 2.0]

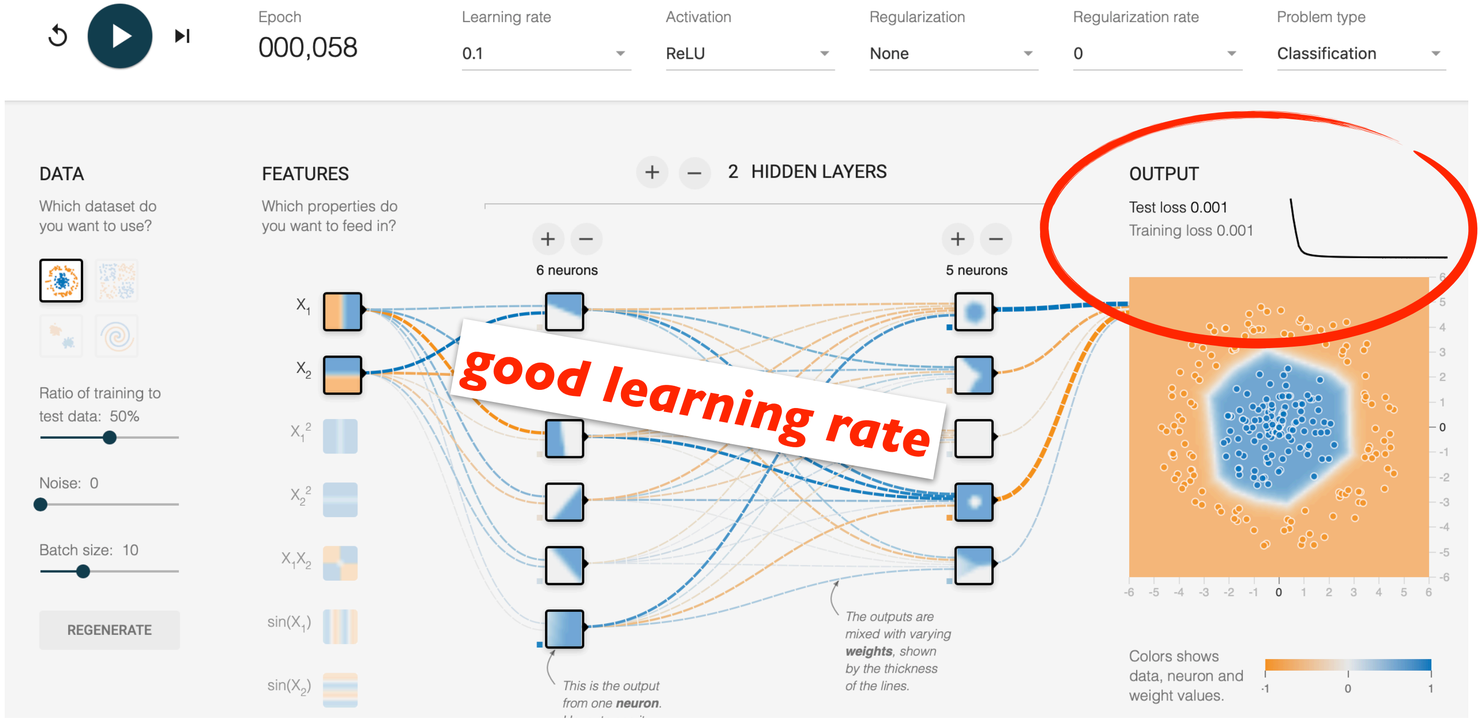
Training Strategies

- Choosing a good learning rate is key for training convergence





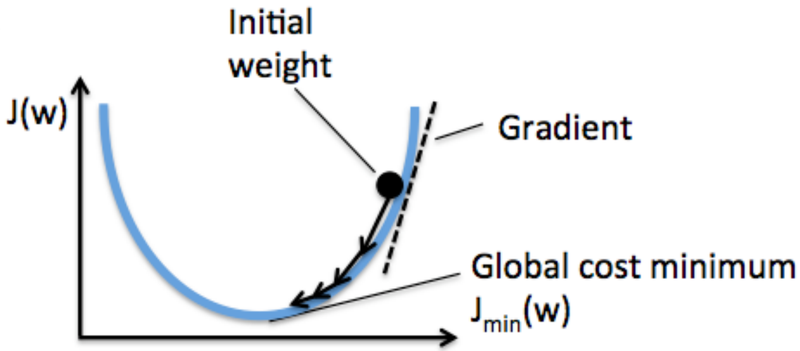
[https://raw.githubusercontent.com/rasbt/python-machine-learning-book/master/code/ch02/images/02_12.png, MIT License]

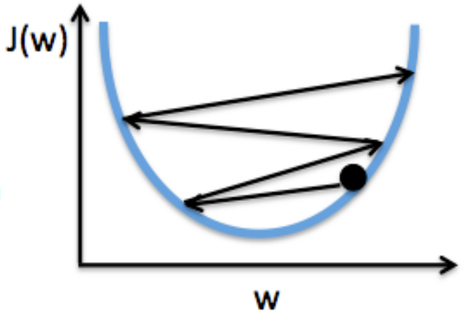


[https://playground.tensorflow.org, github: https://github.com/tensorflow/playground, Apache License 2.0]

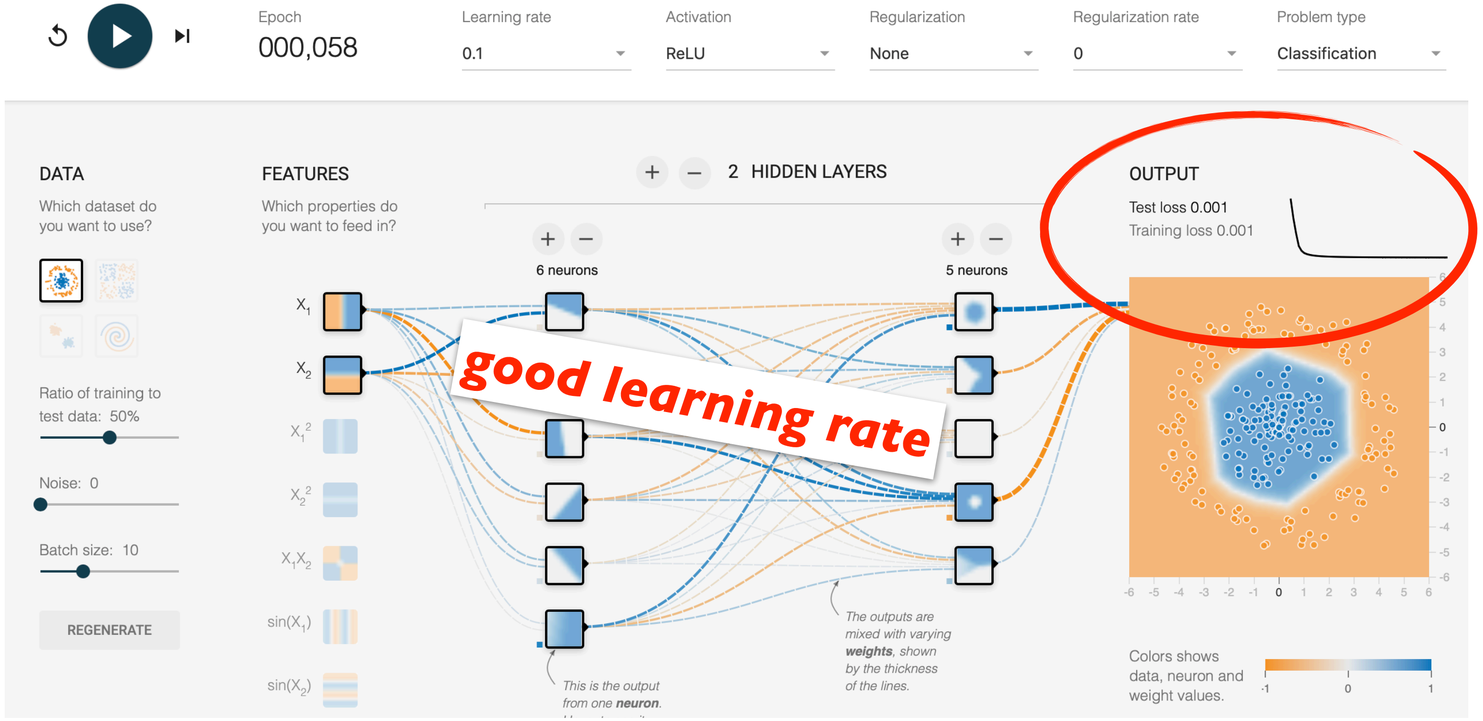
Training Strategies

- Choosing a good learning rate is key for training convergence





[https://raw.githubusercontent.com/rasbt/python-machine-learning-book/master/code/ch02/images/02_12.png, MIT License]



[https://playground.tensorflow.org, github: https://github.com/tensorflow/playground, Apache License 2.0]

Adaptive Moment Estimation (Adam)

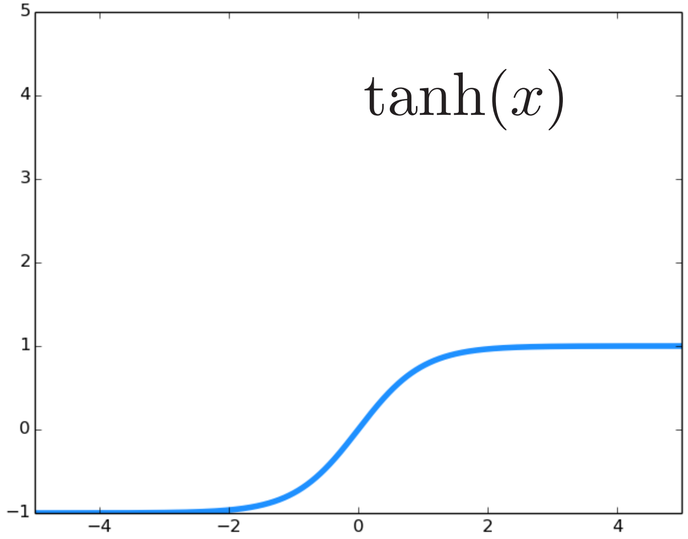
- Adaptive learning rates:
  - Reduce α during training
  - For each parameter separately
- Momentum:

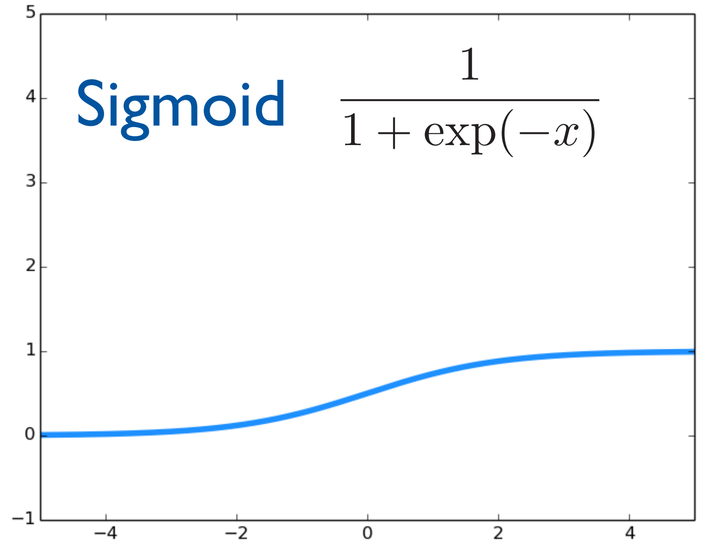
$$\theta \rightarrow \theta - (1-\beta)\left(\alpha\left\langle\frac{\partial \text{Loss}}{\partial\theta}\right\rangle\right) - \beta\,(\text{previous change of }\theta)$$

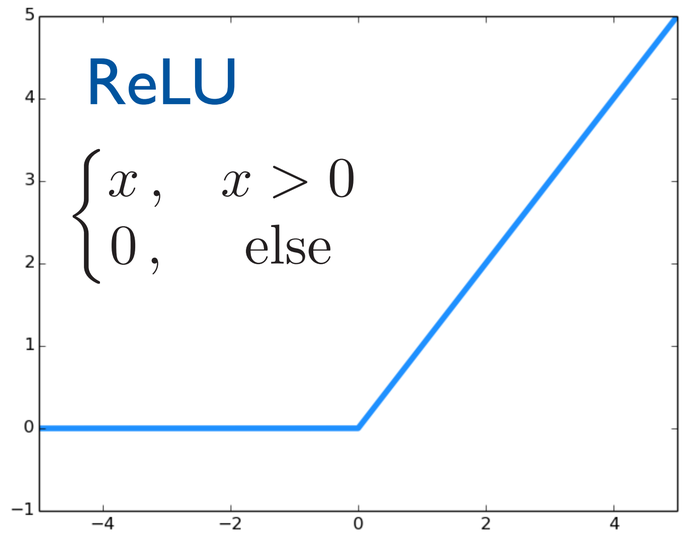
Activation Functions

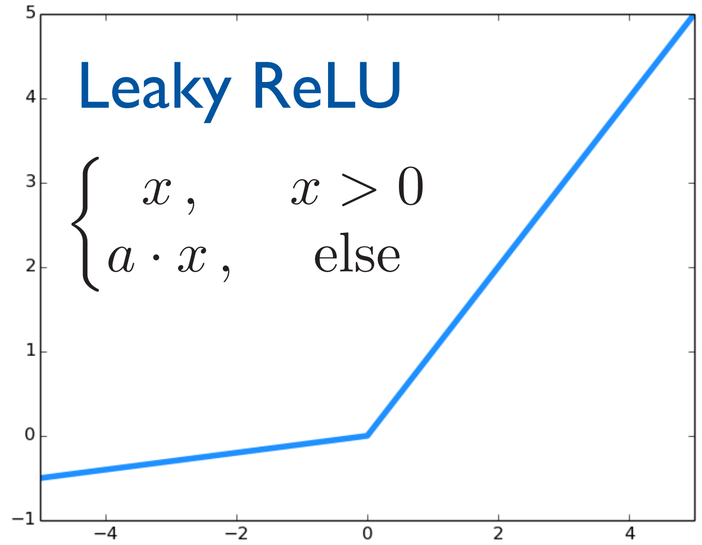
Requirements:

- Non-linear
- Fast differentiation
- Not bound to a fixed interval





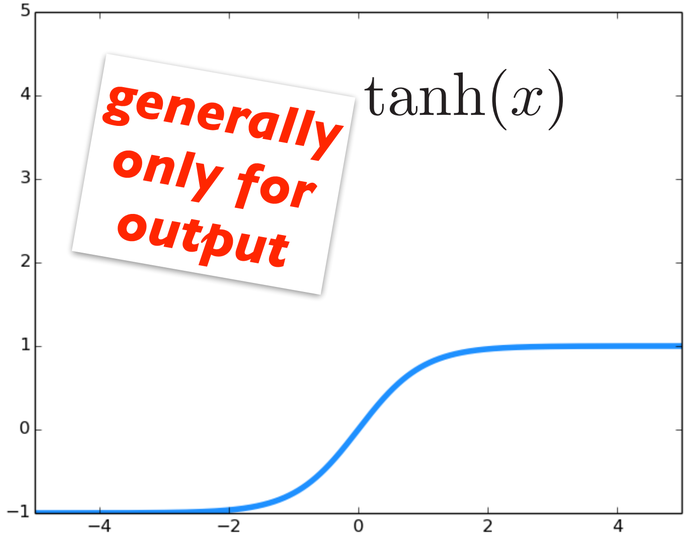


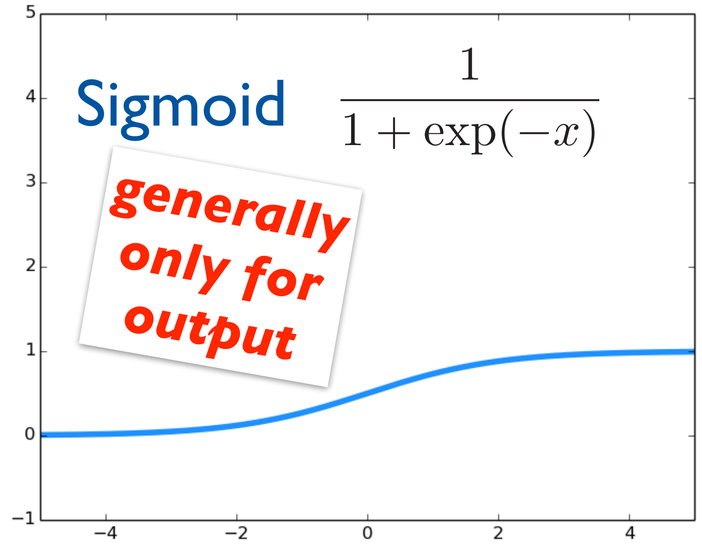


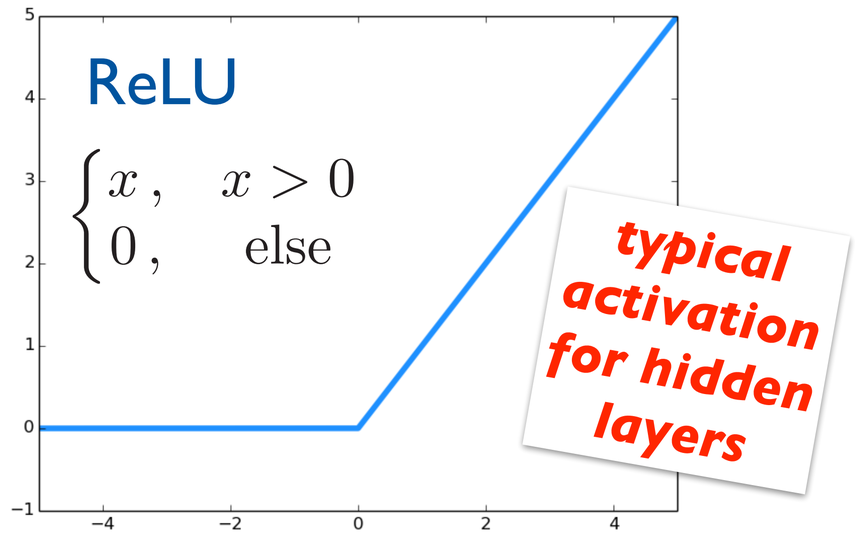
Activation Functions

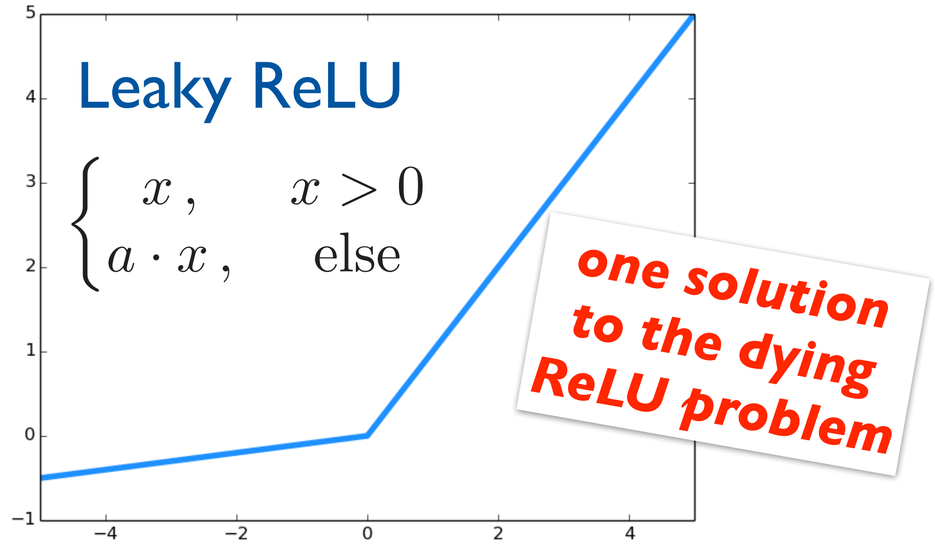
Requirements:

- Non-linear
- Fast differentiation
- Not bound to a fixed interval





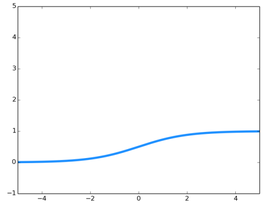




Parameter Initialization

- Initial weight must not be symmetric
- Random initialization

- For sigmoid:

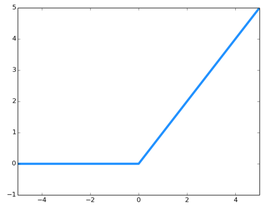


"Xavier Weight Initialization"

for n input values to the node

$$\text{Uniform in } \left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right]$$

- For ReLU:

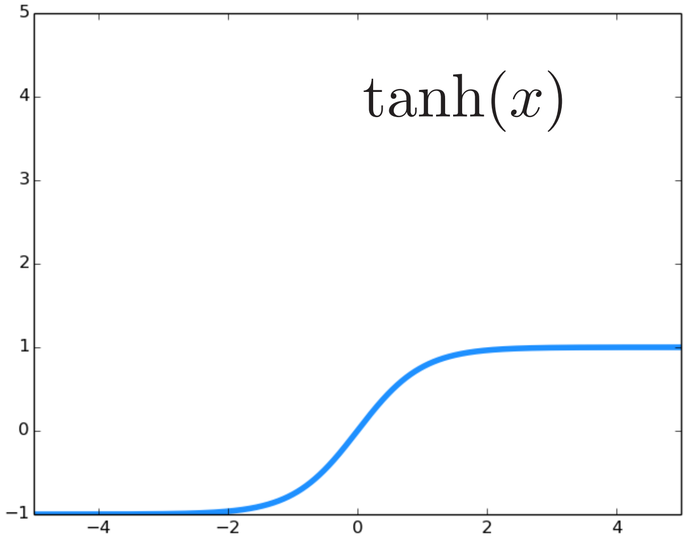


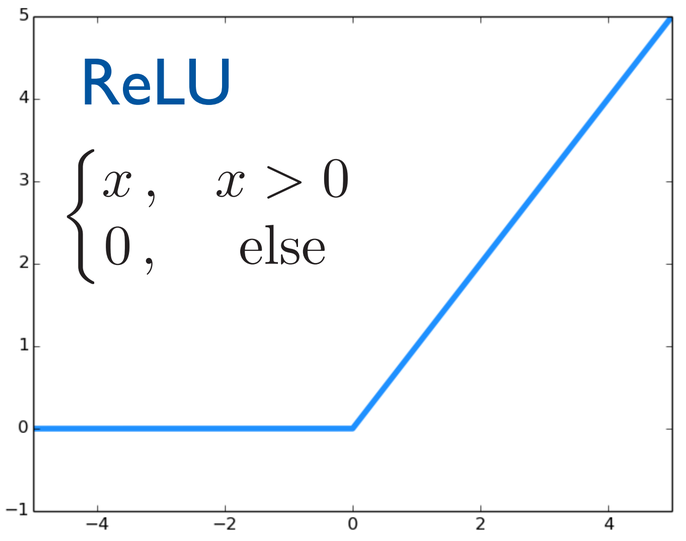
"He Initialization"

$$\text{Gaussian}\left(\mu = 0, \sigma = \sqrt{2/n}\right)$$

Exploding and Vanishing Gradients

$$\frac{\partial \text{Loss}}{\partial \theta} = \frac{\partial \text{Loss}}{\partial \phi} \frac{\partial \phi}{\partial f} \frac{\partial f}{\partial \theta}$$

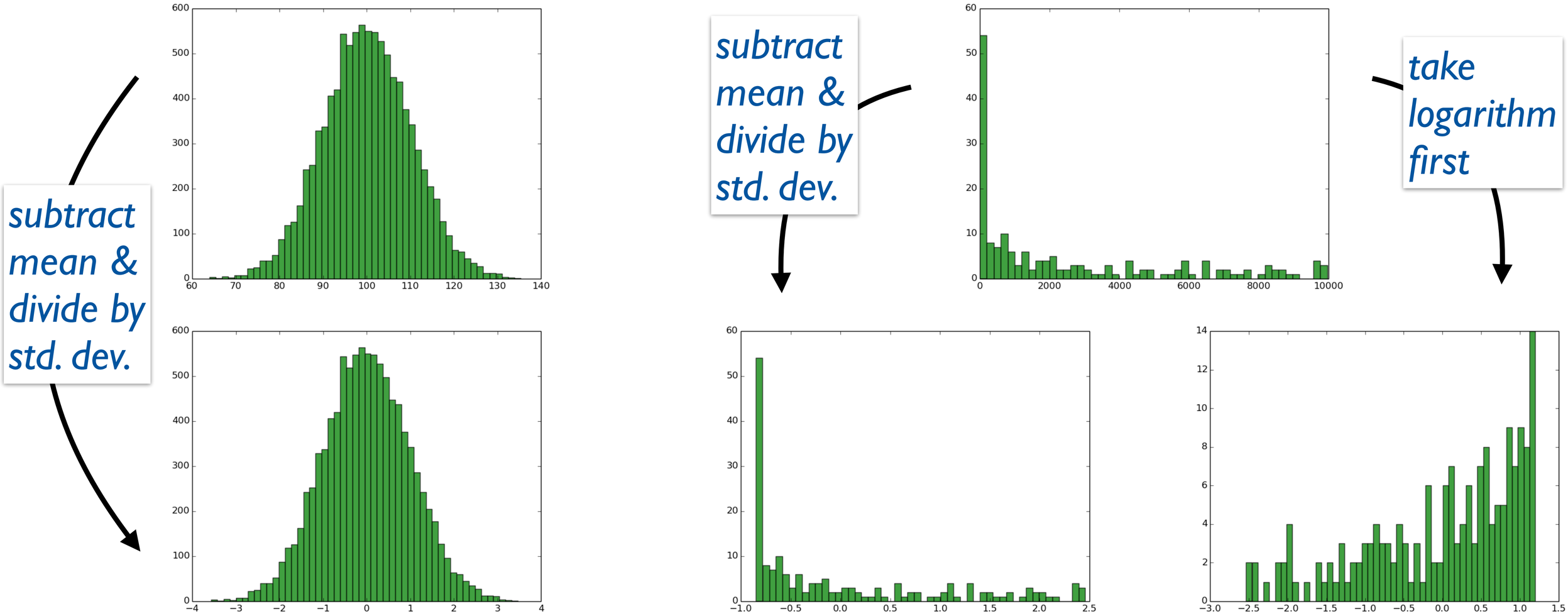




- Vanishing Gradients:
  - $\frac{\partial \text{Loss}}{\partial \theta}$ too small to result in a useful update
    - Example: tanh(x) has derivative ≤ 1
      - Many layers → $\frac{\partial \text{Loss}}{\partial \theta^{(k)}} \propto \prod_{i=k+1}^{N} \left.\frac{\partial \phi}{\partial f}\right|_{\text{layer } i} \to 0$
- Exploding Gradients:
  - $\frac{\partial \text{Loss}}{\partial \theta^{(k)}} \to \text{NaN}$
    - Gradient Clipping, revisit Weight Initialization, …

Preprocessing

- Pre-processing of the input data is essential for numerical stability
- Want them to be centered around zero and with O(1) variance



Splitting the Data

- Often, the dataset is split into three parts:
  - Test data set: keep it until your model is fully optimized !

    (you can look at it once…)
  - Training data set: for the optimization of the NN parameters

    (you use it extensively)
  - Validation data set: check performance during optimization

    → *more in Nicole's lecture!*
- Example split: Train:Val:Test → 60% : 20% : 20%

Hyperparameters

Network structure

- Number of hidden layers
- Number of nodes in the hidden layers
- Activation functions

Weight Initialization

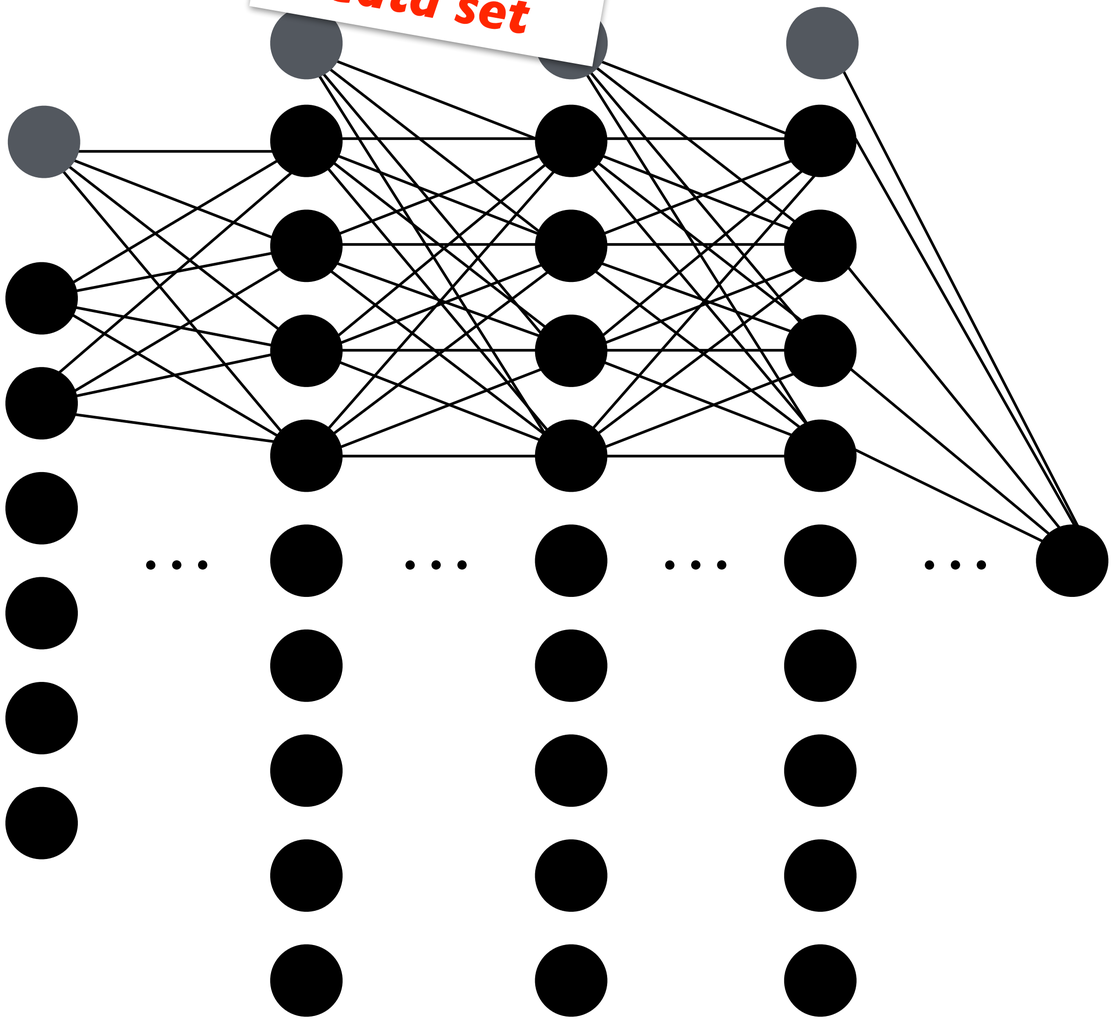
Optimizer and its parameters

- (Initial) learning rate, …

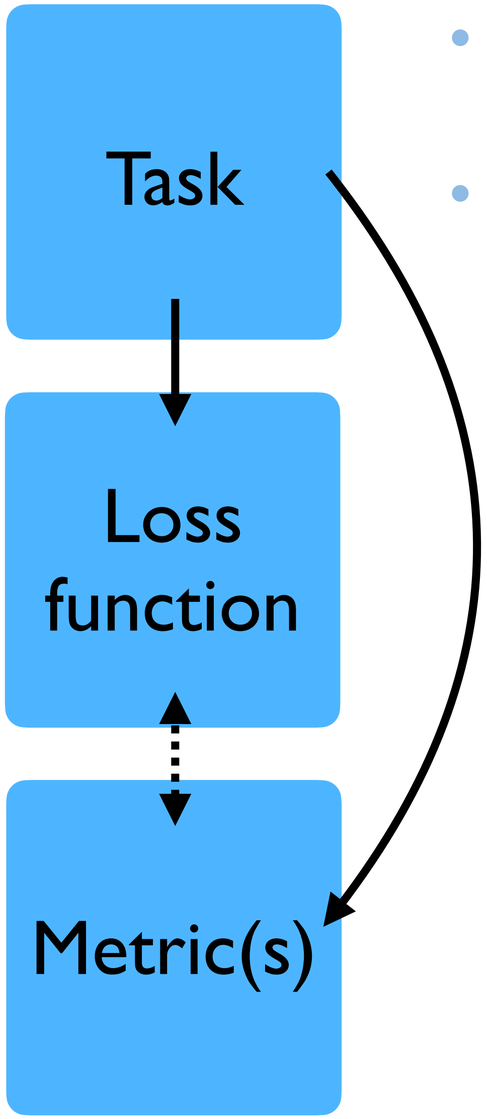
Batch size

Regularization → Nicole's lecture

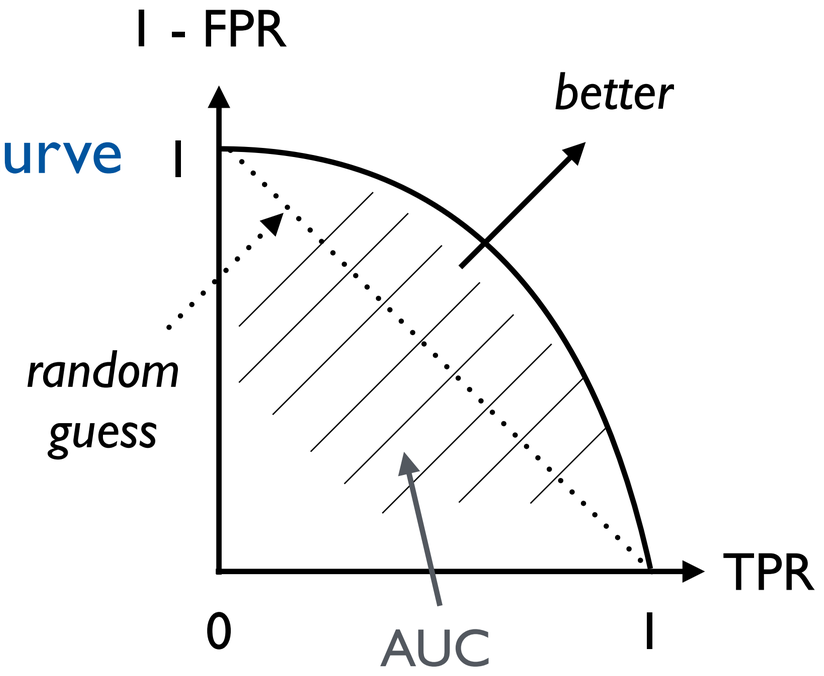
…



Metrics

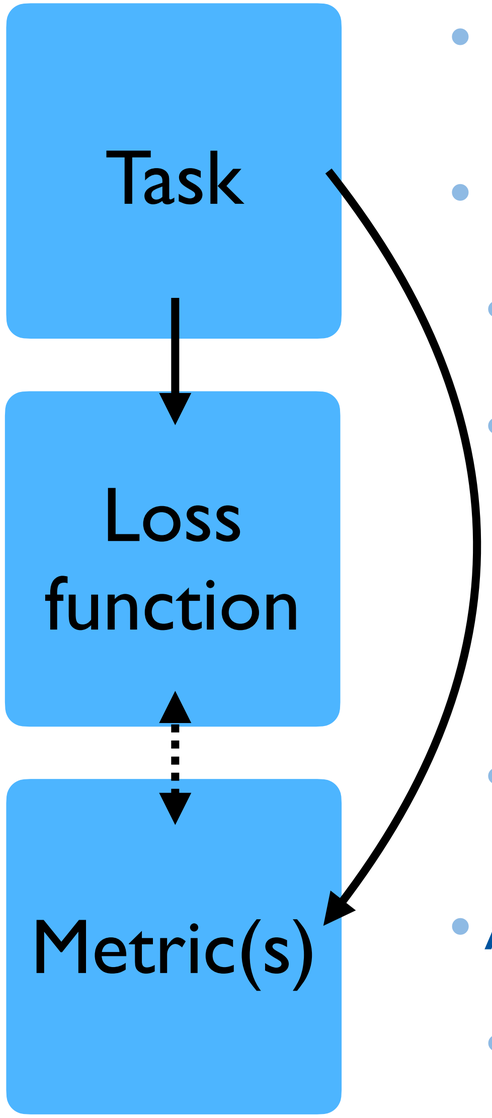


- Metric should represent a figure of merit for achieving the task
- Binary classification:
  - Receiver operating characteristic (ROC) curve
  - True Positive Rate* (TPR) vs.
    1 - False Positive Rate** (FPR)
    ("signal vs. '1 - background efficiency' ")
  - Area Under the Curve (AUC)

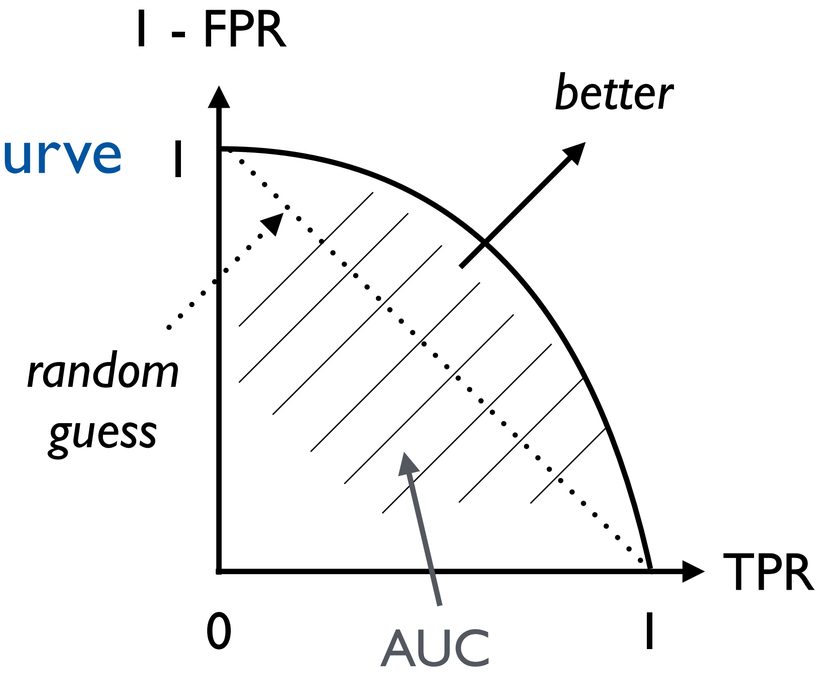


* also "sensitivity"
** also "1 - 'specificity' "

Metrics



- Metric should represent a figure of merit for achieving the task
- Binary classification:
  - Receiver operating characteristic (ROC) curve
  - True Positive Rate* (TPR) vs.
    1 - False Positive Rate** (FPR)
    ("signal vs. '1 - background efficiency' ")
  - Area Under the Curve (AUC)



- A (made-up) example for market forecasting (= regression)
  - Loss: Mean Squared Error
  - Metric: Error in the predicted profit, averaged over all products

* also "sensitivity"
** also "1 - 'specificity' "

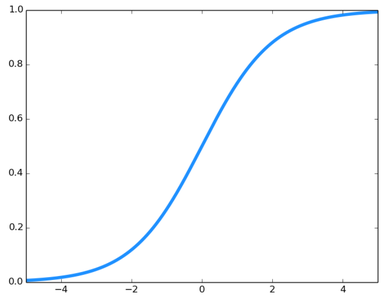
Instead of a Summary: A Classification Example

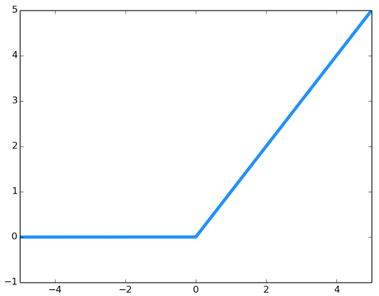
[JINST 14 (2019) P11015, arXiv:1907:11181]

- LHC events with several jets, a charged lepton and a neutrino
  - Classification of jet permutations:
    - Pick the correct jet order out of many possible orderings

- DNN implementation*
- Binary classification (ordering is either "right" or "wrong")

- Loss: binary cross-entropy $$-\frac{1}{n}\sum_{i=1}^{n}\begin{cases}\log q_i\,, & \text{signal}\\ \log\left(1-q_i\right)\,, & \text{background}\end{cases}$$
- Output: sigmoid
- Activation in hidden layers: ReLU





* More efficient proposals have been made since then.

Classification Example: Pre-Processing

- 26 input features:
  - 4-momenta of the jets
  - Boolean: Is the jet flagged as a b-jet?
  - 3-momentum of the charged lepton
  - Magnitude and azimuth (Φ) of the missing transverse momentum
    - Φ is 2π-continuous → replace by sin(Φ) and cos(Φ)

- Input scaling:

$$x_i \rightarrow \frac{x_i - \mu_i}{\sigma_i}$$

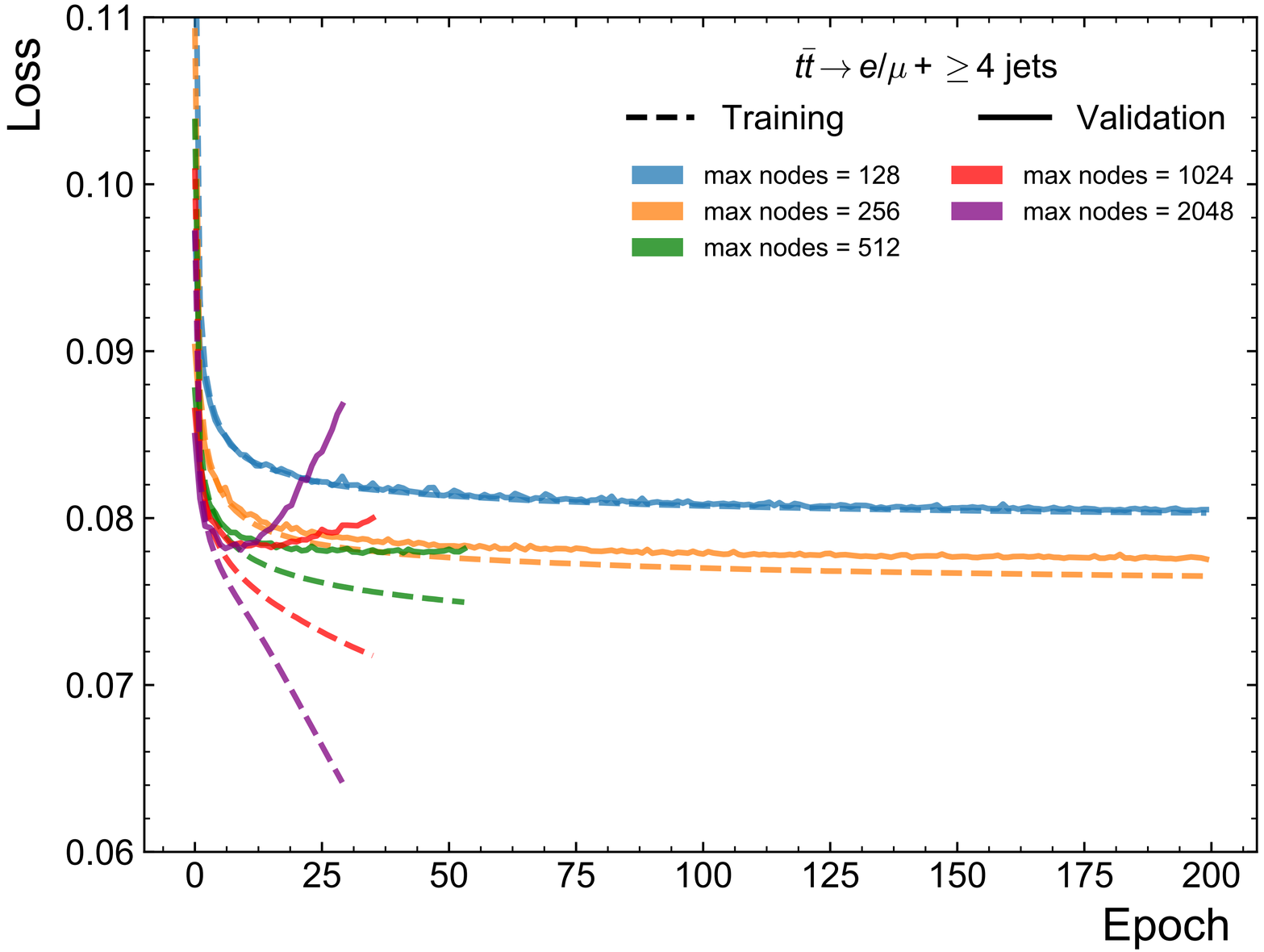
Classification Example: Training Progress

- Loss vs. Epoch
- 5 hidden layers:

  128-64-32-16-8

  256-128-64-32-16

  …



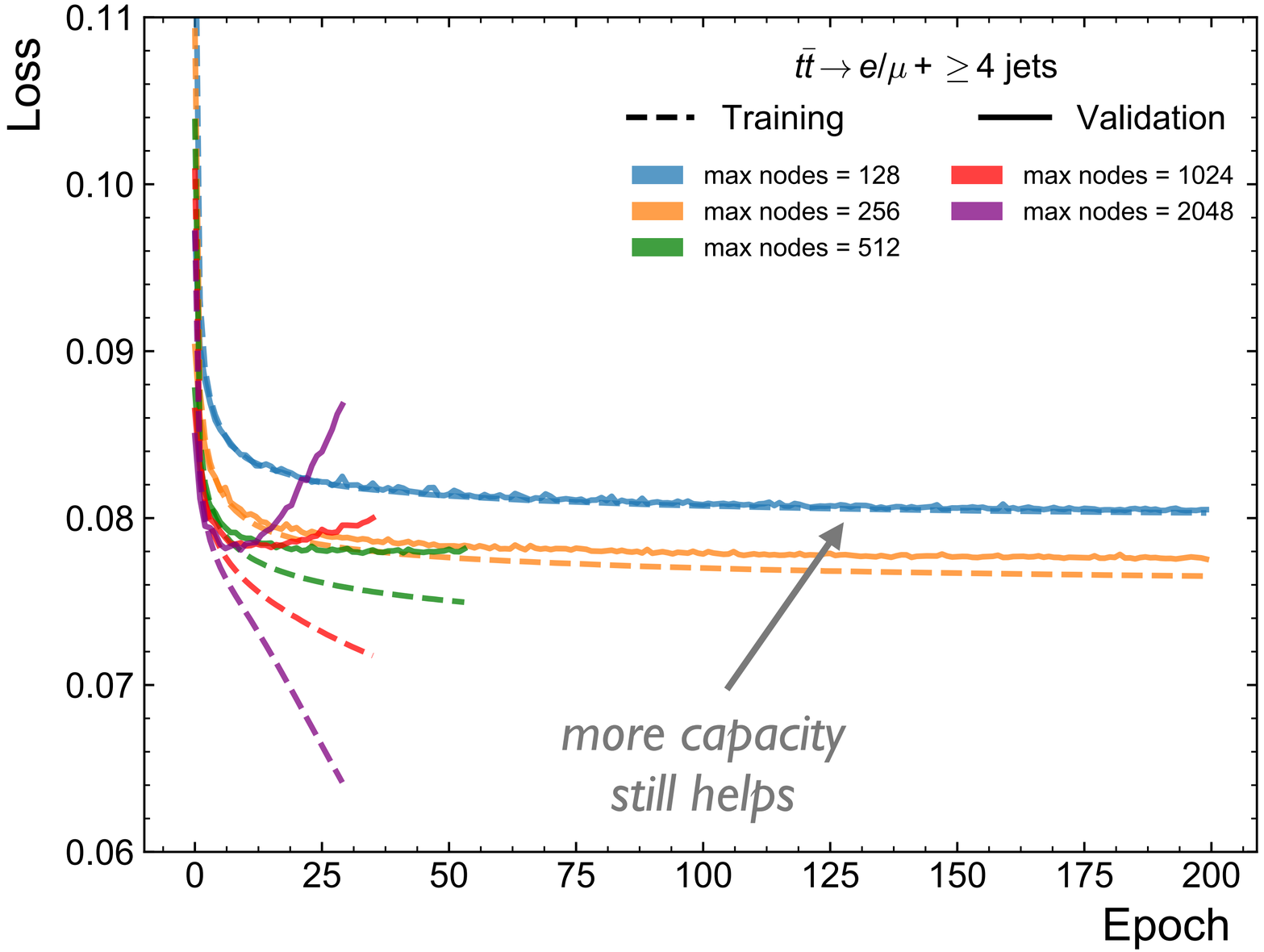
Classification Example: Training Progress

- Loss vs. Epoch
- 5 hidden layers:

  128-64-32-16-8

  256-128-64-32-16

  …



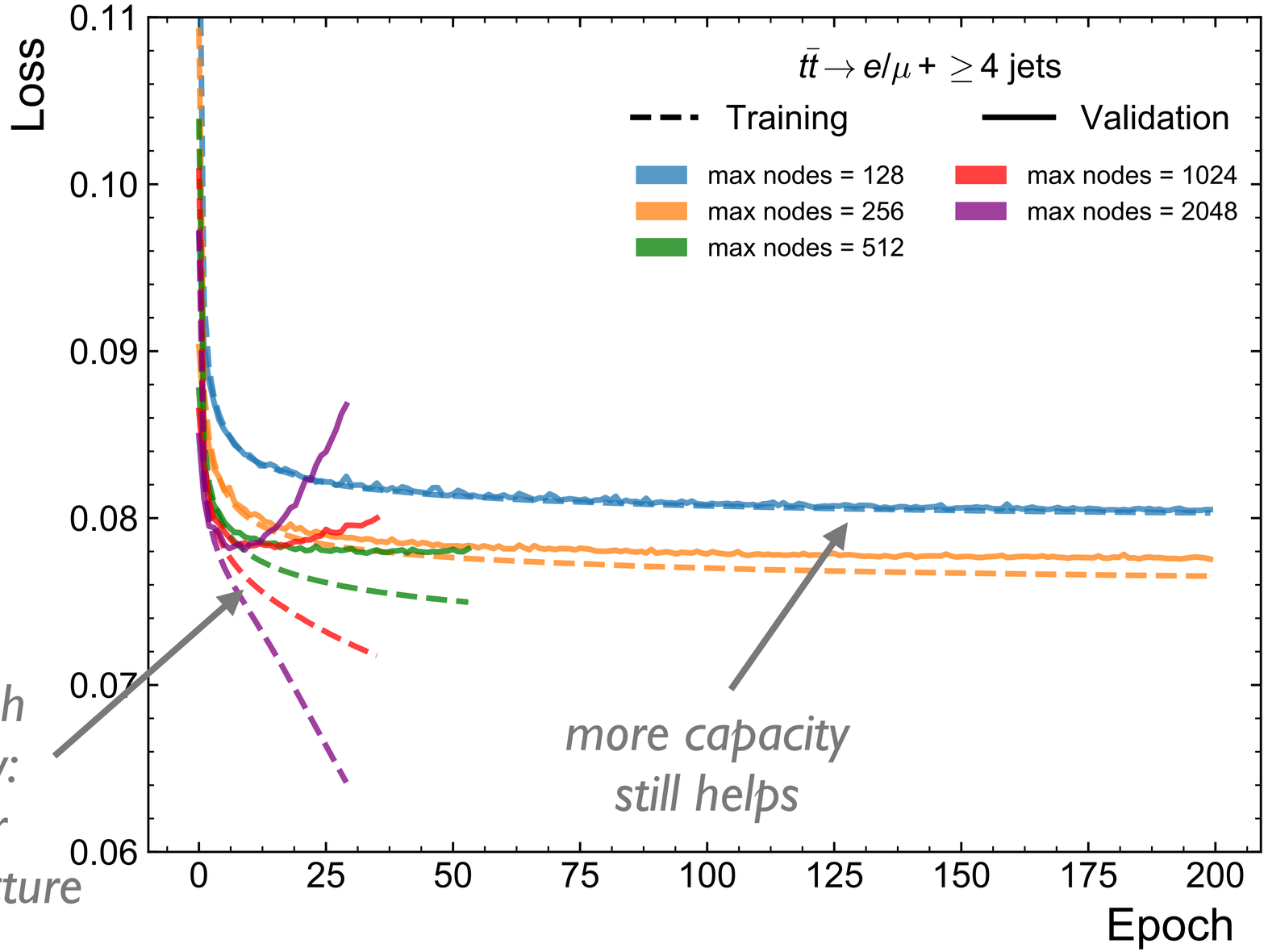
Classification Example: Training Progress

- Loss vs. Epoch
- 5 hidden layers:

  128-64-32-16-8

  256-128-64-32-16

  …

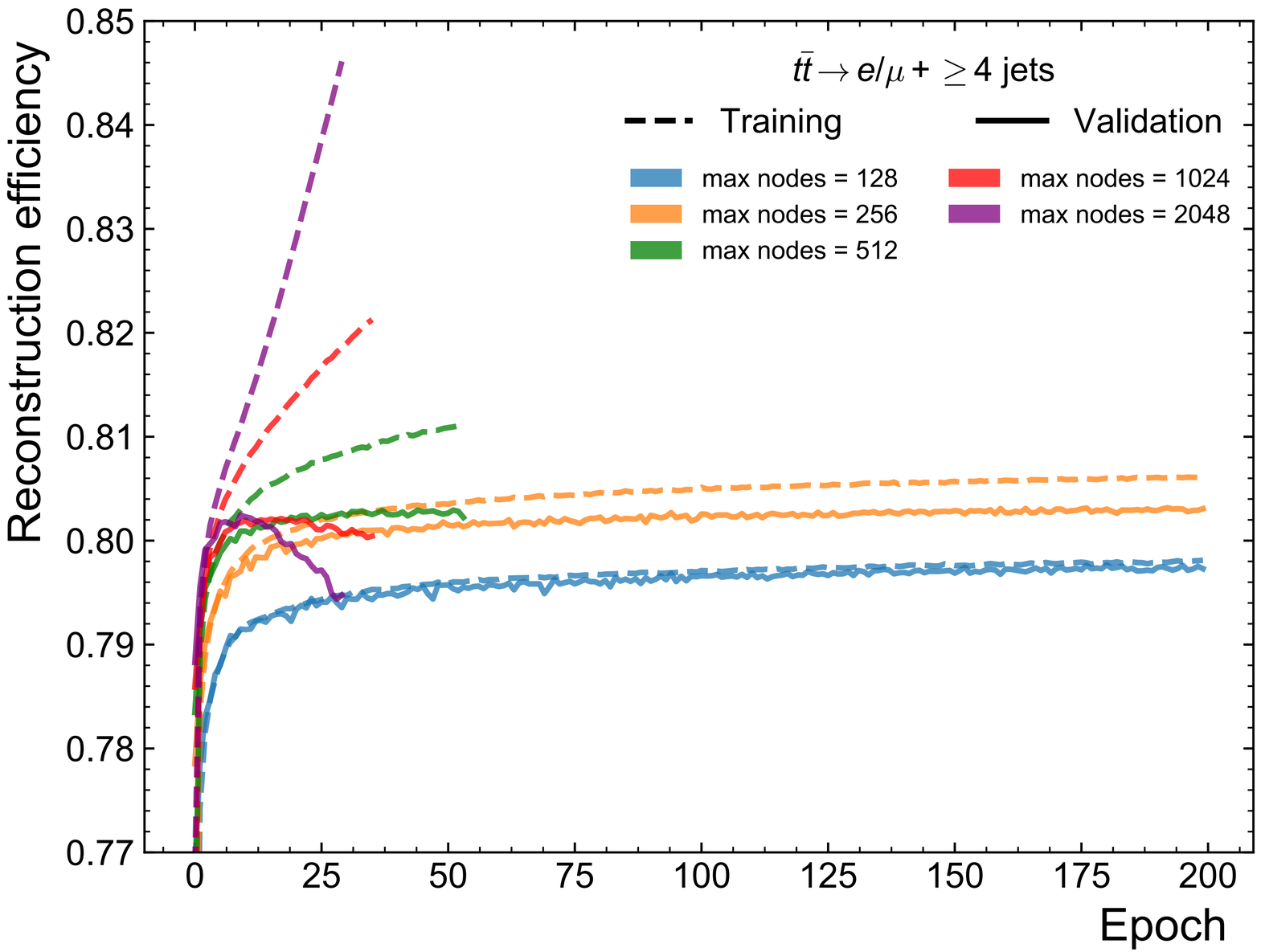


Classification Example: Metrics

- Metric vs. Epoch
- Task:

  Find the right jet ordering
- Metric:

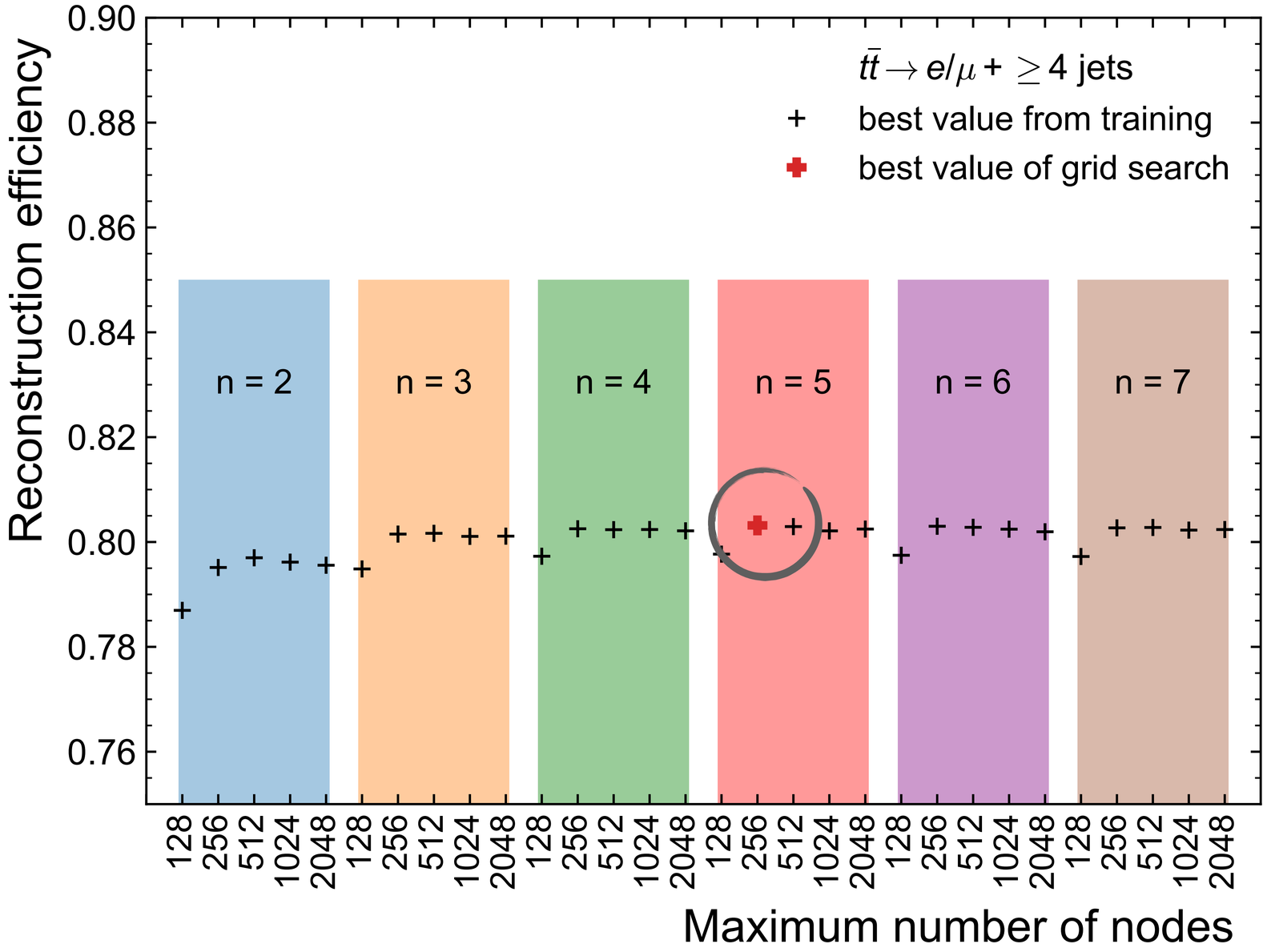
  Fraction of events with correctly predicted jet ordering



Classification Example: Hyperparameters

- 2D grid search:
  number of hidden layers n
  and maximum number
  of nodes in hidden layers

- Choice of best epoch
  based on metric on
  validation data set



Next Up

- Now: Time for discussion

- Then:

Coffee	
Faculty Club im TUM IAS-Gebäude in Garching, TUM / LMU München	14:55 - 15:15
Mastering Model Building	*Nicole Hartmann*
Faculty Club im TUM IAS-Gebäude in Garching, TUM / LMU München	15:15 - 16:30