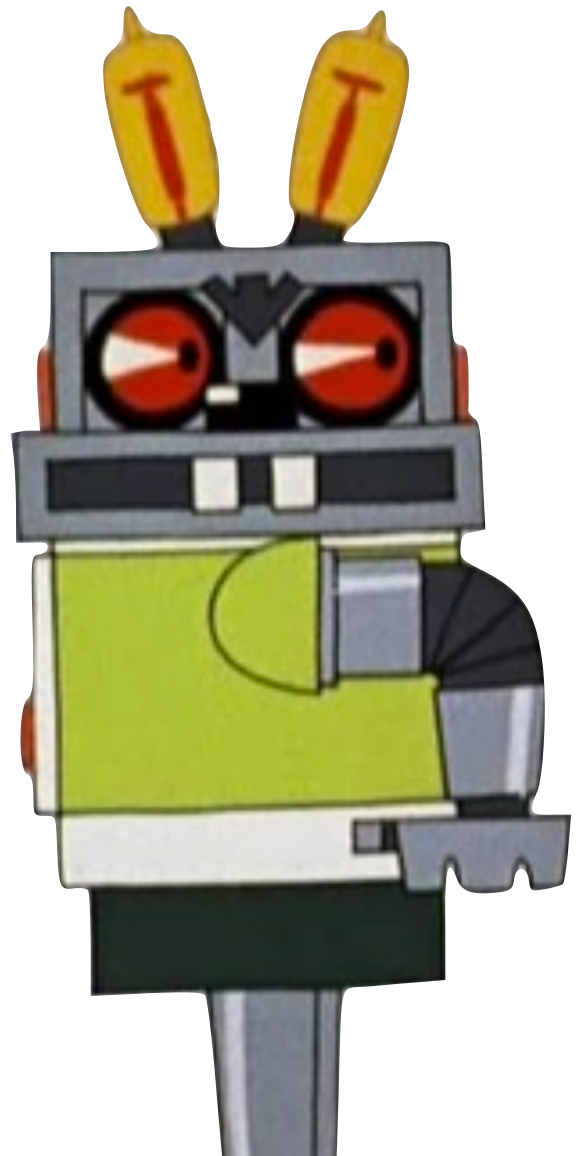
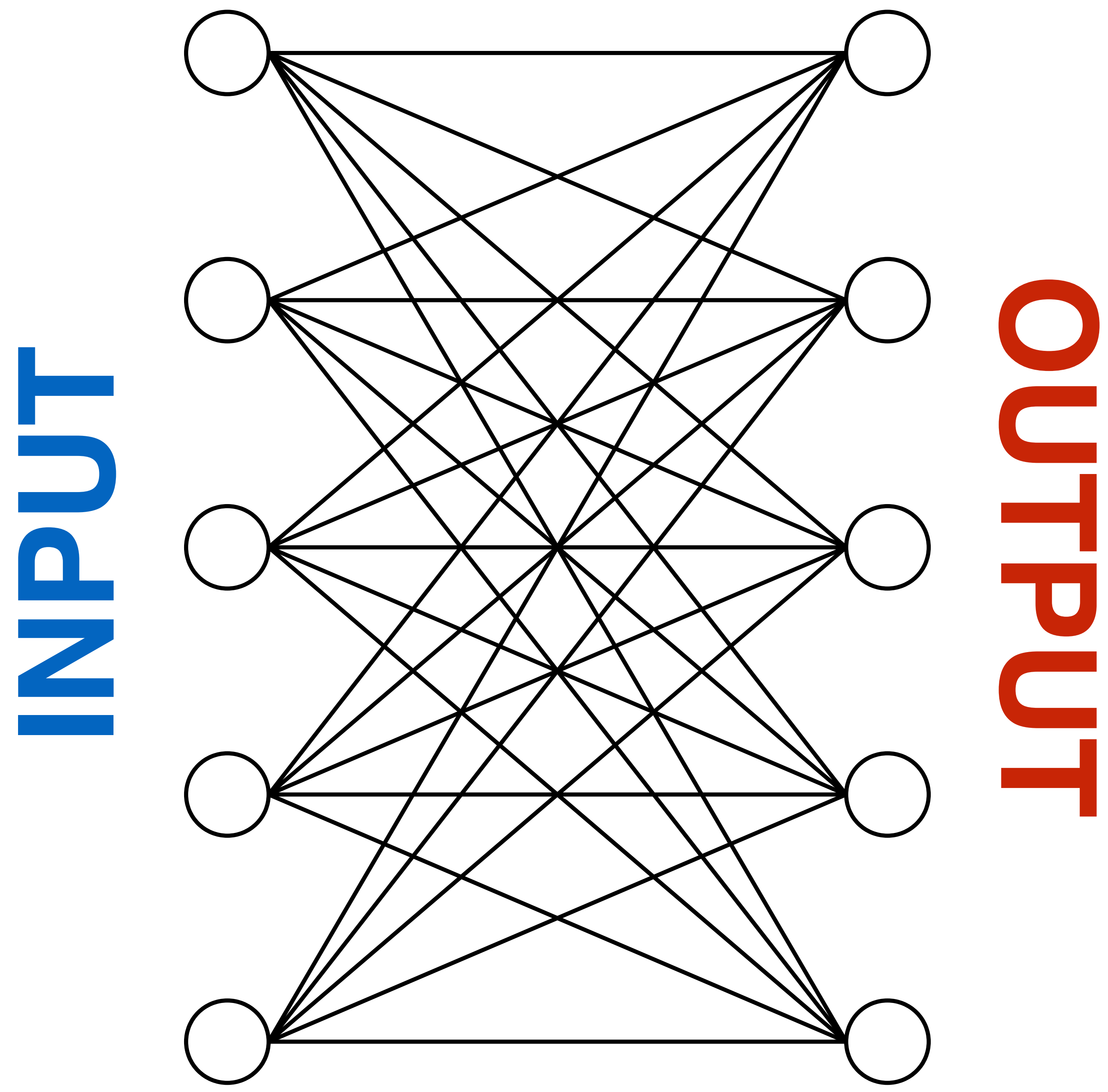
MACHINE LEARNING & DATA MINING



A view on deep networks

- NN without a hidden layer is just a regular linear model: $\hat{\mathbf{y}} = g(\mathbf{W}\mathbf{x})$

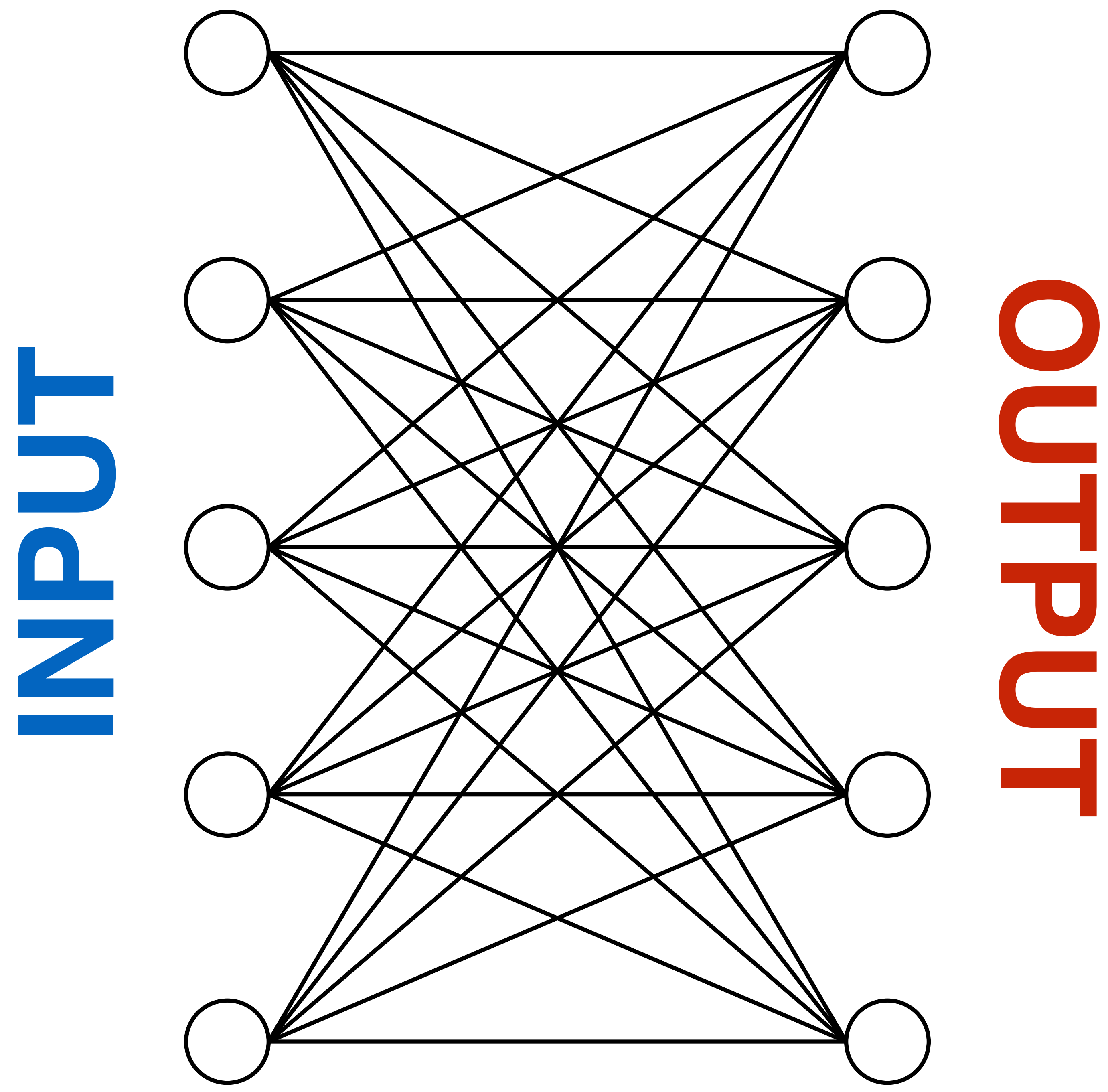


A view on deep networks

- NN without a hidden layer is just a regular linear model: $\hat{\mathbf{y}} = g(\mathbf{W}\mathbf{x})$

activation function
(e.g. softmax)

model parameters

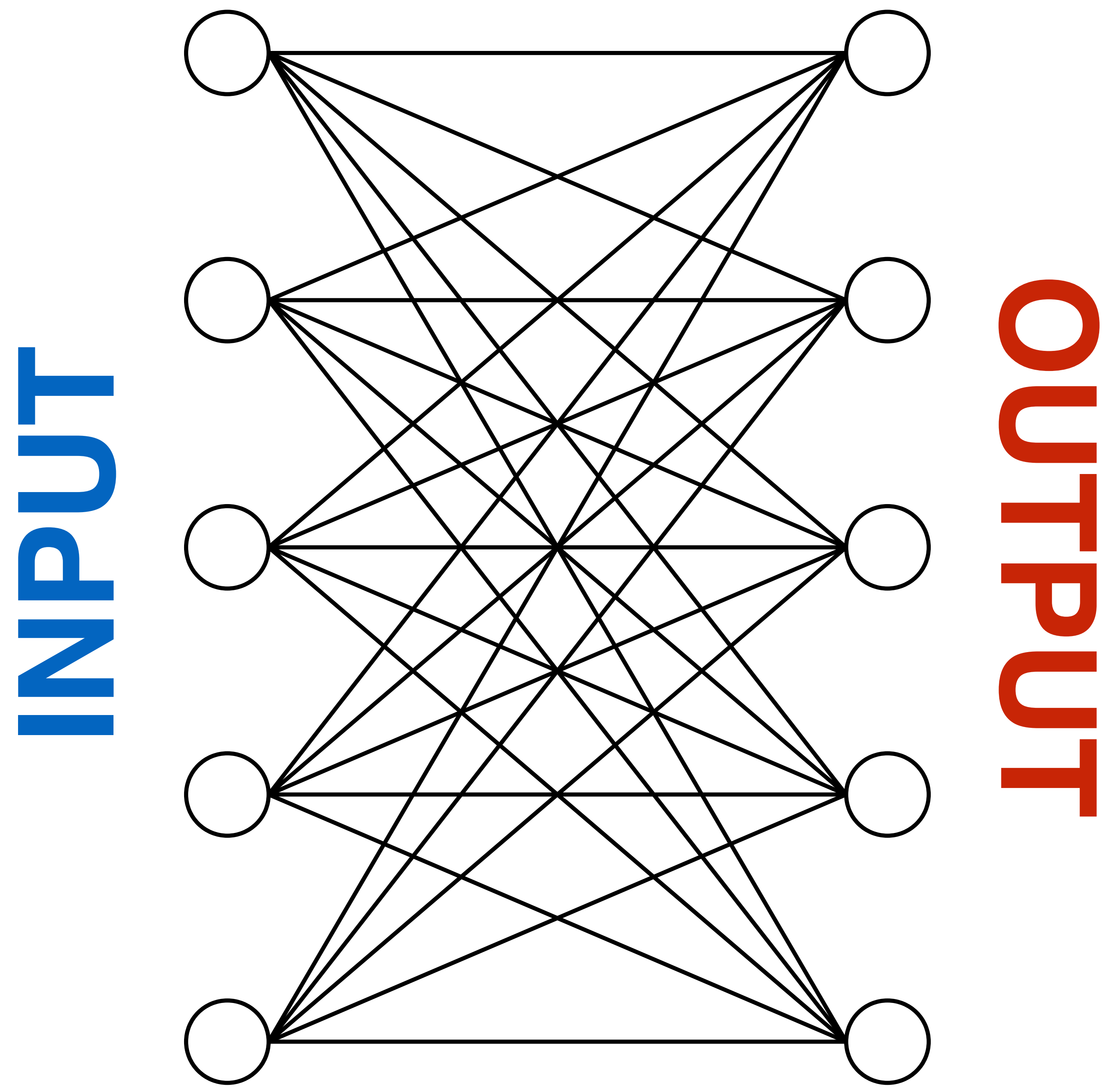


A view on deep networks

- NN without a hidden layer is just a regular linear model: $\hat{\mathbf{y}} = g(\mathbf{W}\mathbf{x})$

activation function
(e.g. softmax)

model parameters



- Can be reliably fit either in closed form or with convex optimization

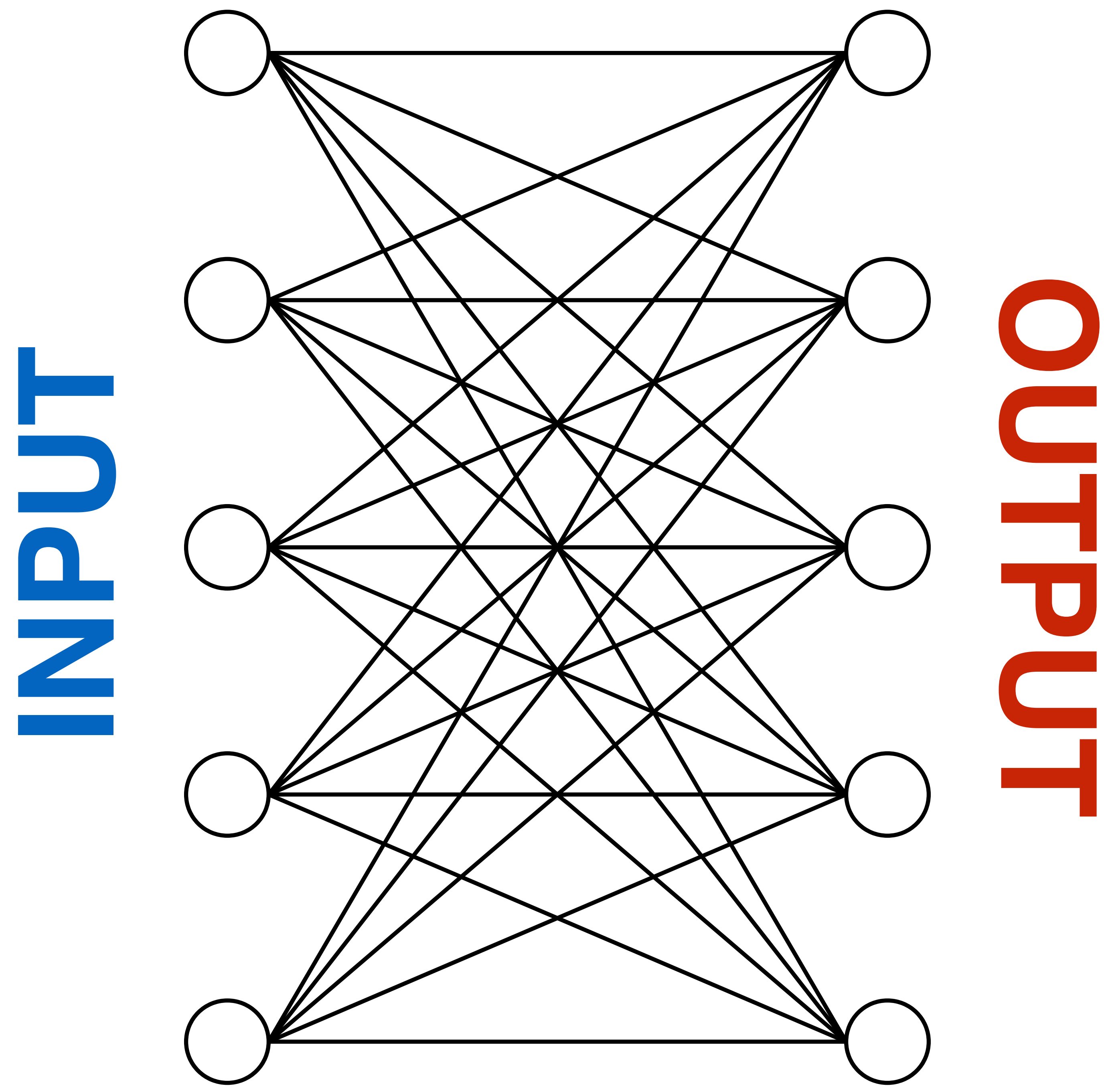
A view on deep networks

- NN without a hidden layer is just a regular linear model:

$$\hat{\mathbf{y}} = g(\mathbf{W}\mathbf{x})$$

activation function
(e.g. softmax)

model parameters



- Can be reliably fit either in closed form or with convex optimization
- Limited, e.g. cannot understand interactions between features. Possible ways to overcome:

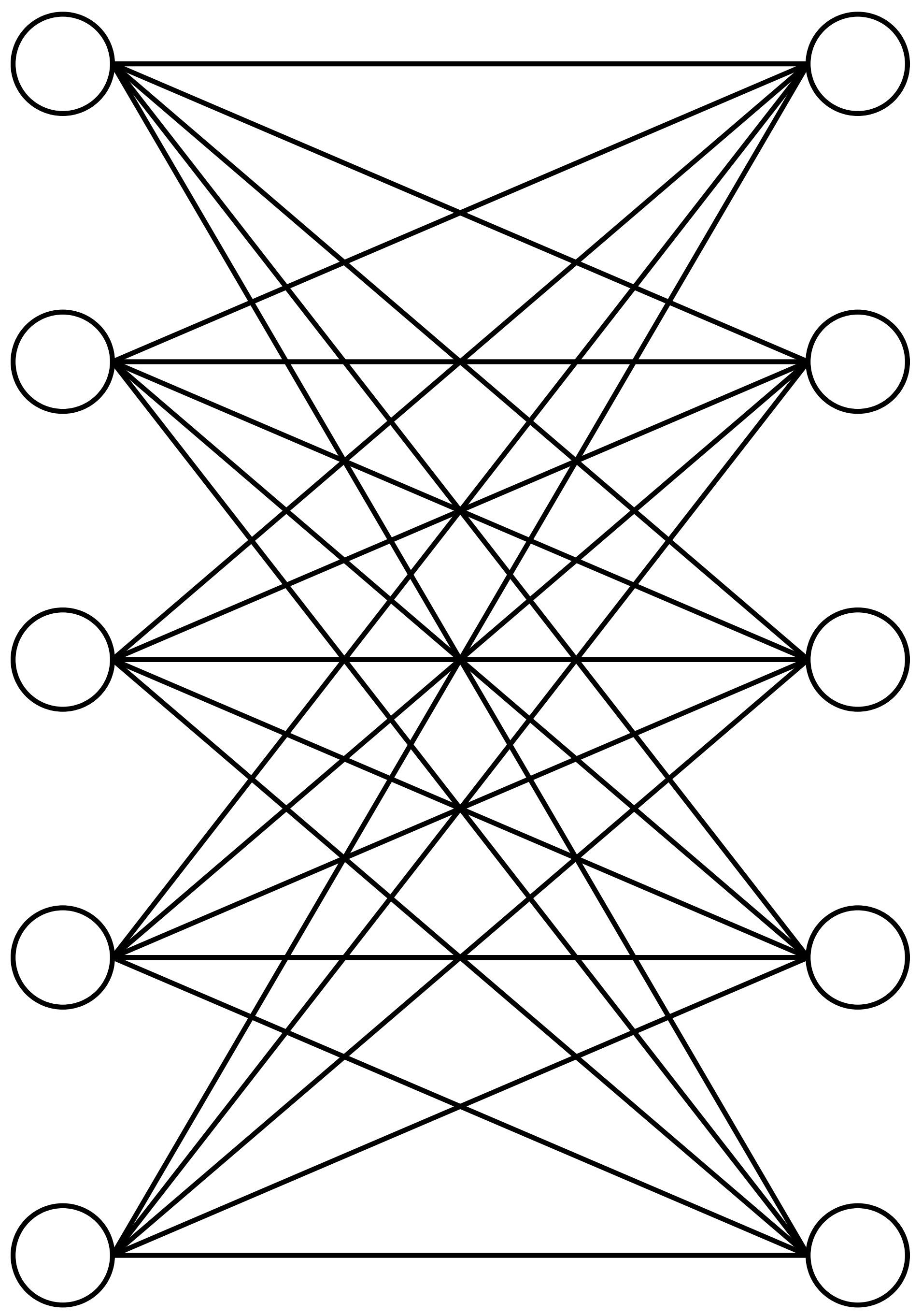
A view on deep networks

- NN without a hidden layer is just a regular linear model: $\hat{\mathbf{y}} = g(\mathbf{W}\mathbf{x})$

activation function
(e.g. softmax)

model parameters

INPUT



OUTPUT

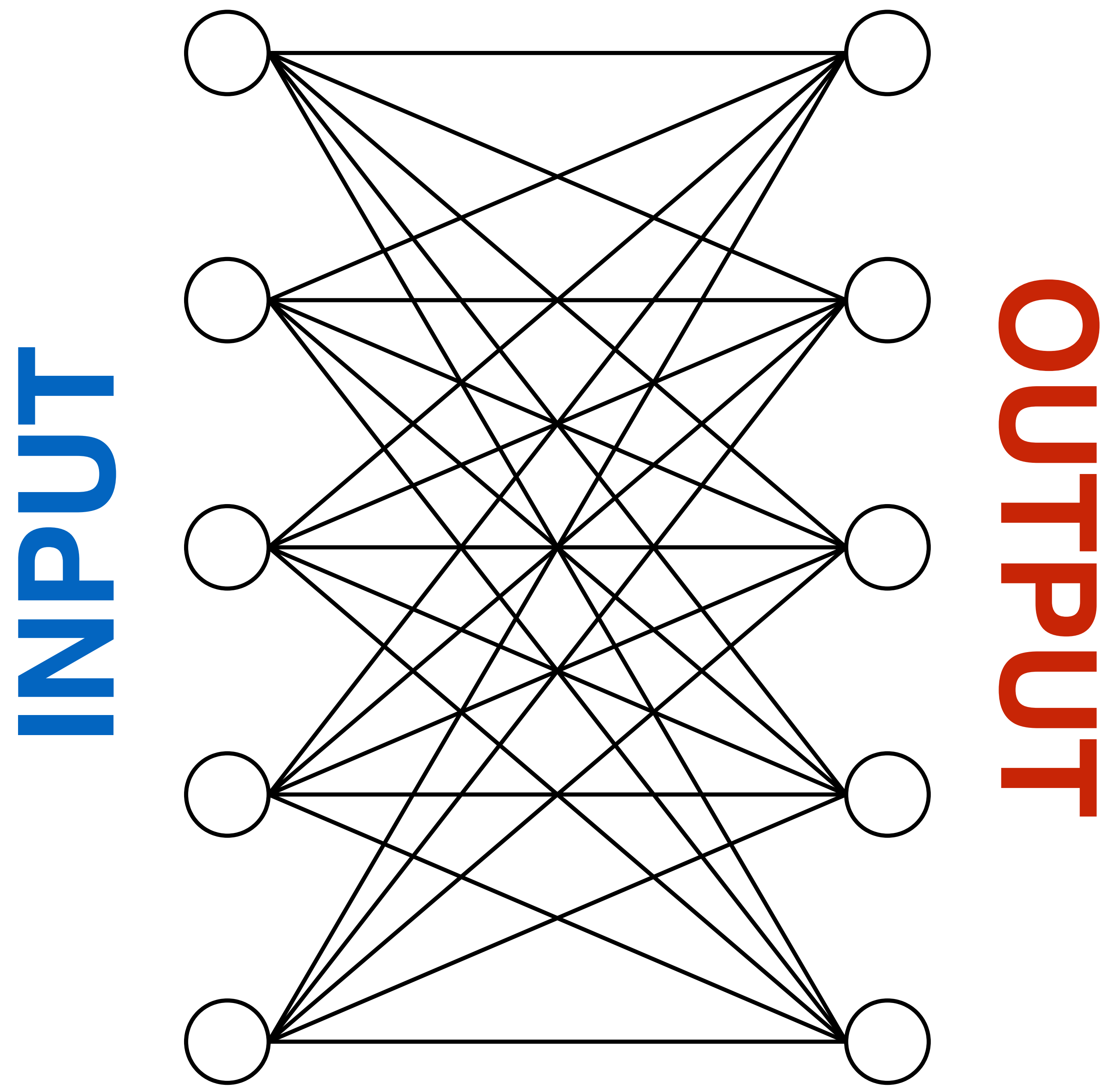
- Can be reliably fit either in closed form or with convex optimization
- Limited, e.g. cannot understand interactions between features. Possible ways to overcome:
  - engineer the best features for a given problem

A view on deep networks

- NN without a hidden layer is just a regular linear model: $\hat{\mathbf{y}} = g(\mathbf{W}\mathbf{x})$

activation function
(e.g. softmax)

model parameters



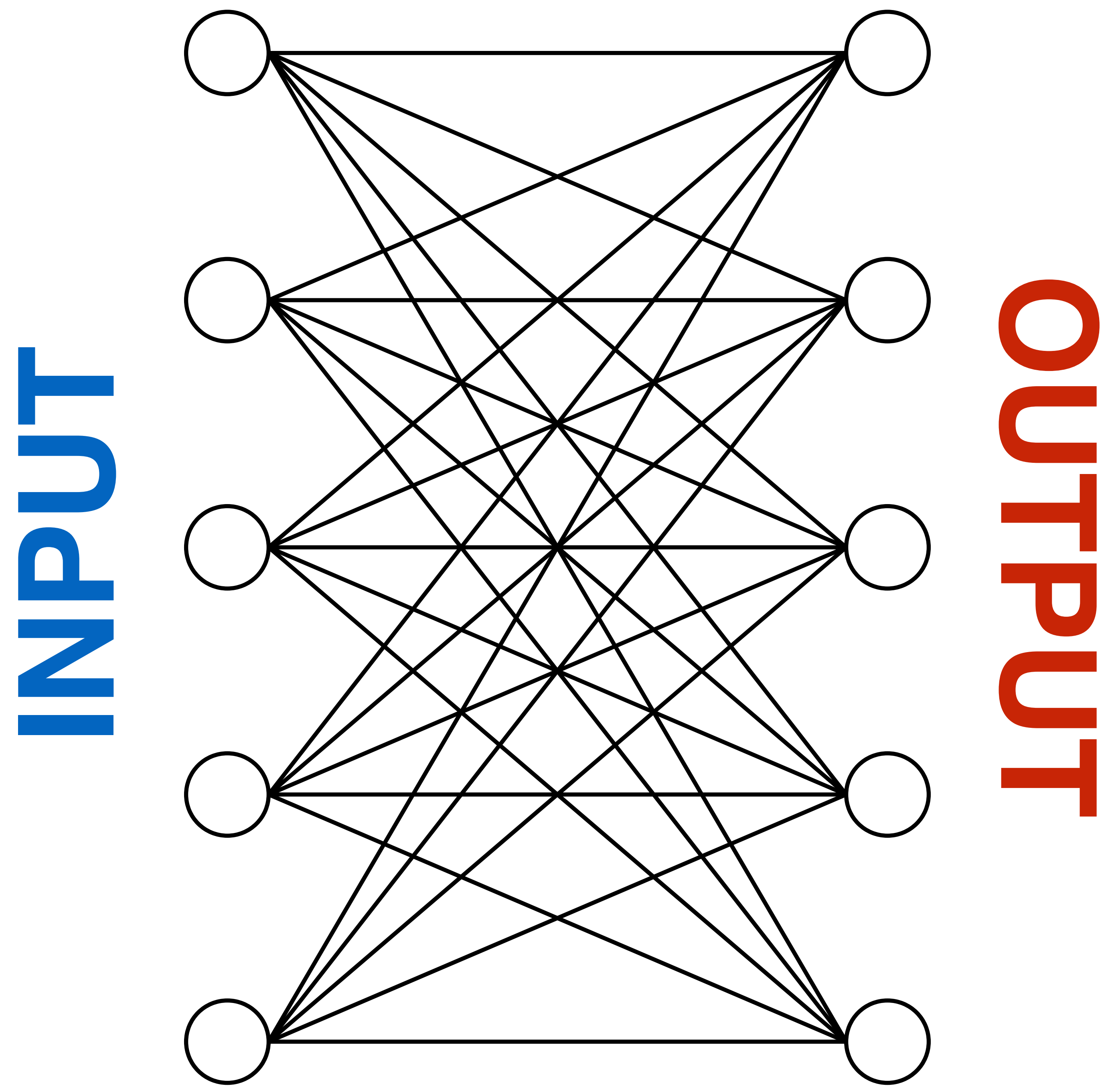
- Can be reliably fit either in closed form or with convex optimization
- Limited, e.g. cannot understand interactions between features. Possible ways to overcome:
  - engineer the best features for a given problem
  - use generic feature mapping (e.g. RBF kernel)

A view on deep networks

- NN without a hidden layer is just a regular linear model: $\hat{\mathbf{y}} = g(\mathbf{W}\mathbf{x})$

activation function
(e.g. softmax)

model parameters

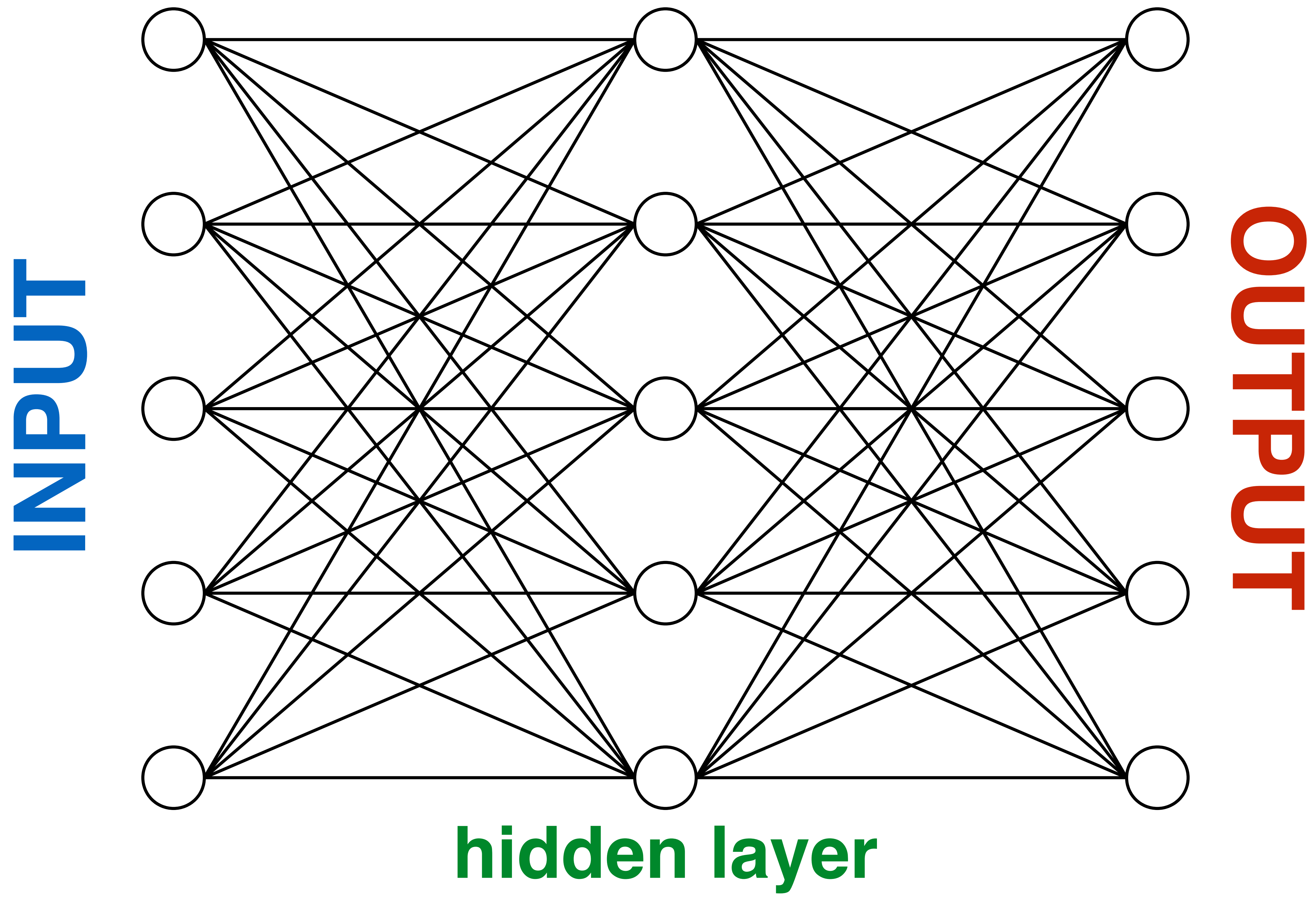


- Can be reliably fit either in closed form or with convex optimization
- Limited, e.g. cannot understand interactions between features. Possible ways to overcome:
  - engineer the best features for a given problem
  - use generic feature mapping (e.g. RBF kernel)
  - learn the features (≈ linear stacking)

A view on deep networks

- Single hidden layer – linear stacking of linear models:*

$$\hat{\mathbf{y}} = g_{(2)}\left(\mathbf{W}_{(2)}\, g_{(1)}(\mathbf{W}_{(1)}\,\mathbf{x})\right)$$



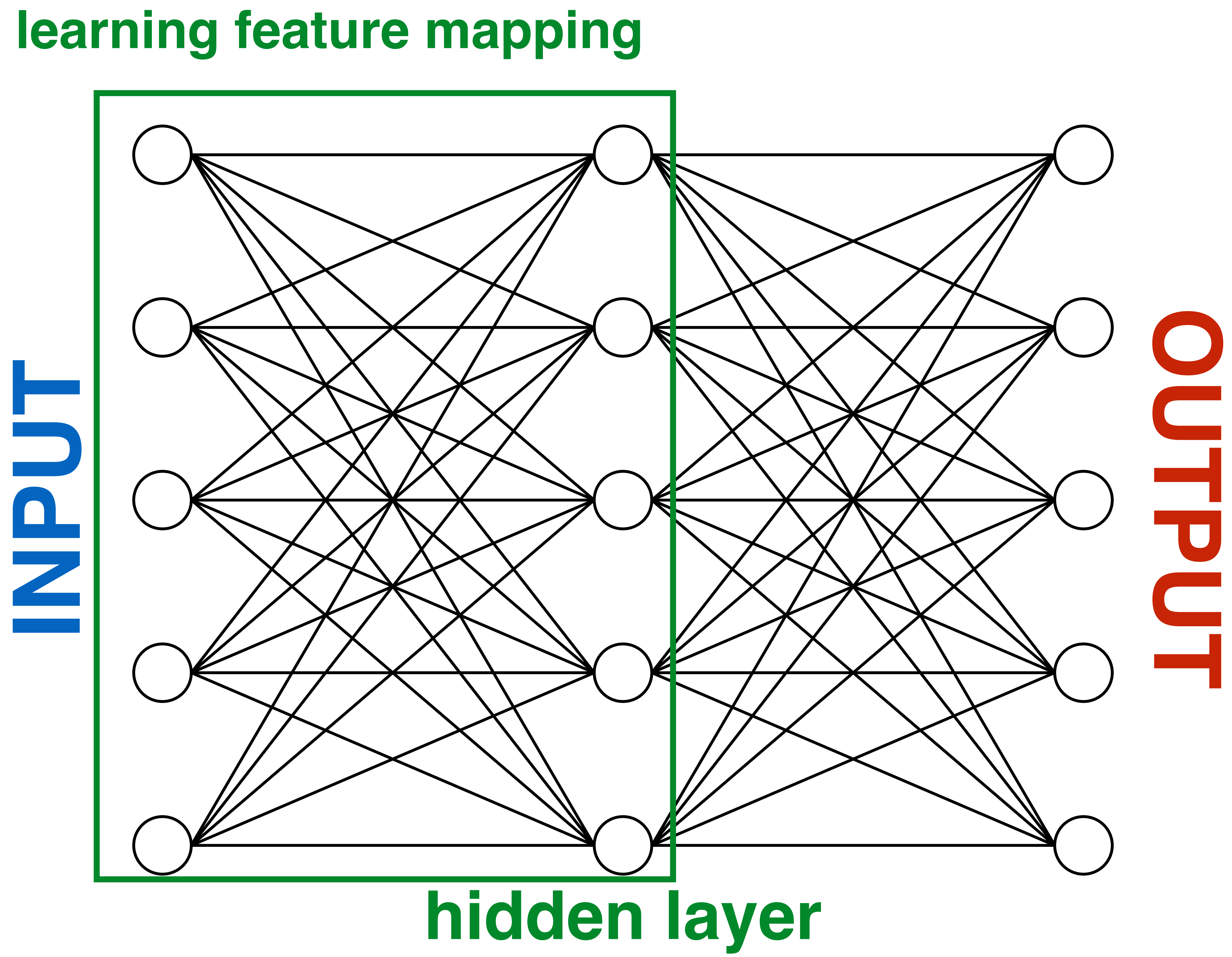
*does not make sense if inner models are purely linear, need a non-linearity (e.g. logistic regression)

A view on deep networks

- Single hidden layer – linear stacking of linear models:*

$$\hat{\mathbf{y}} = g_{(2)}\left(\mathbf{W}_{(2)}\, g_{(1)}(\mathbf{W}_{(1)}\, \mathbf{x})\right)$$

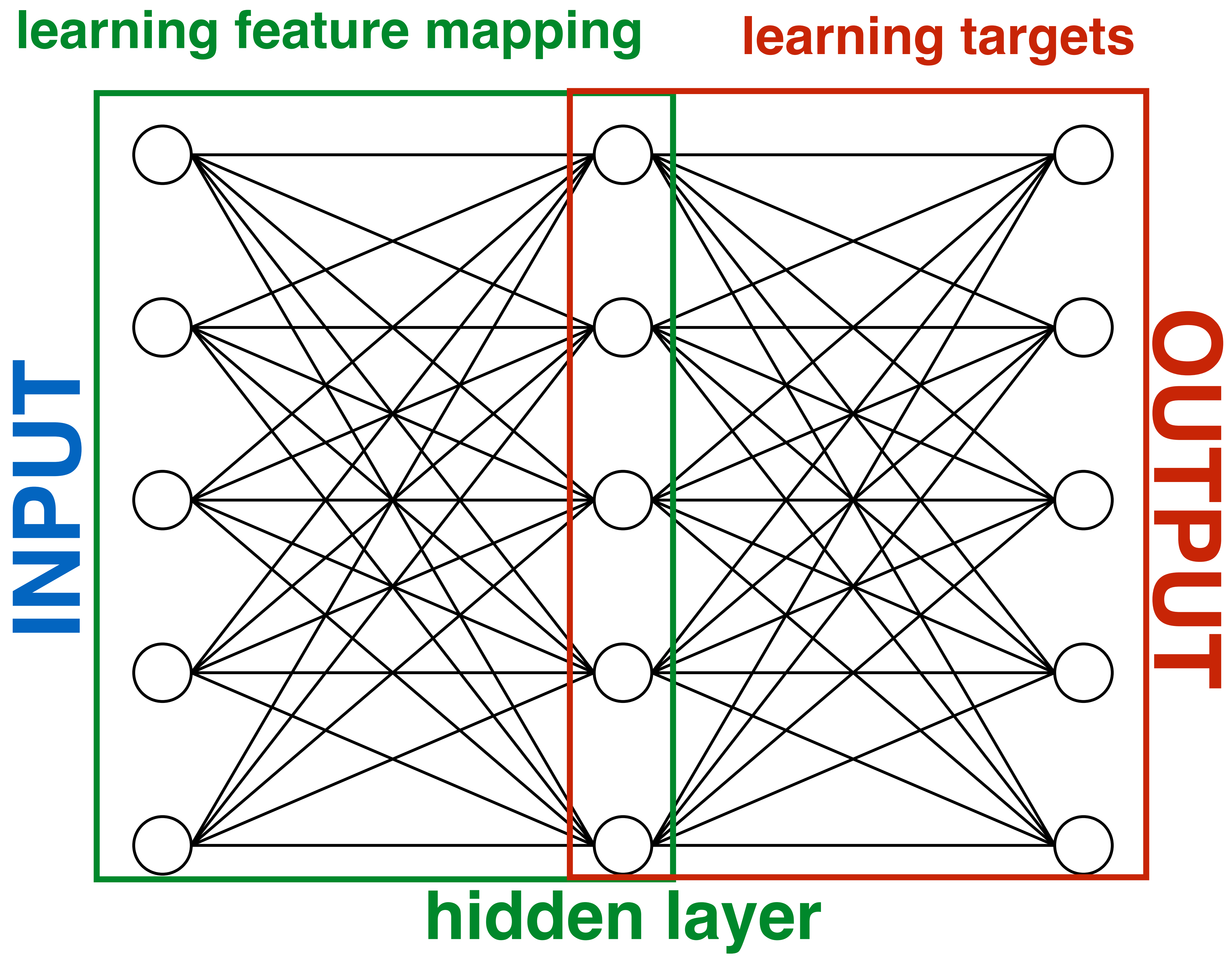
params. for learning the feature mapping



*does not make sense if inner models are purely linear, need a non-linearity (e.g. logistic regression)

A view on deep networks

- Single hidden layer – linear stacking of linear models:*



$$\hat{\mathbf{y}} = g_{(2)}\left(\mathbf{W}_{(2)}\, g_{(1)}\left(\mathbf{W}_{(1)}\,\mathbf{x}\right)\right)$$

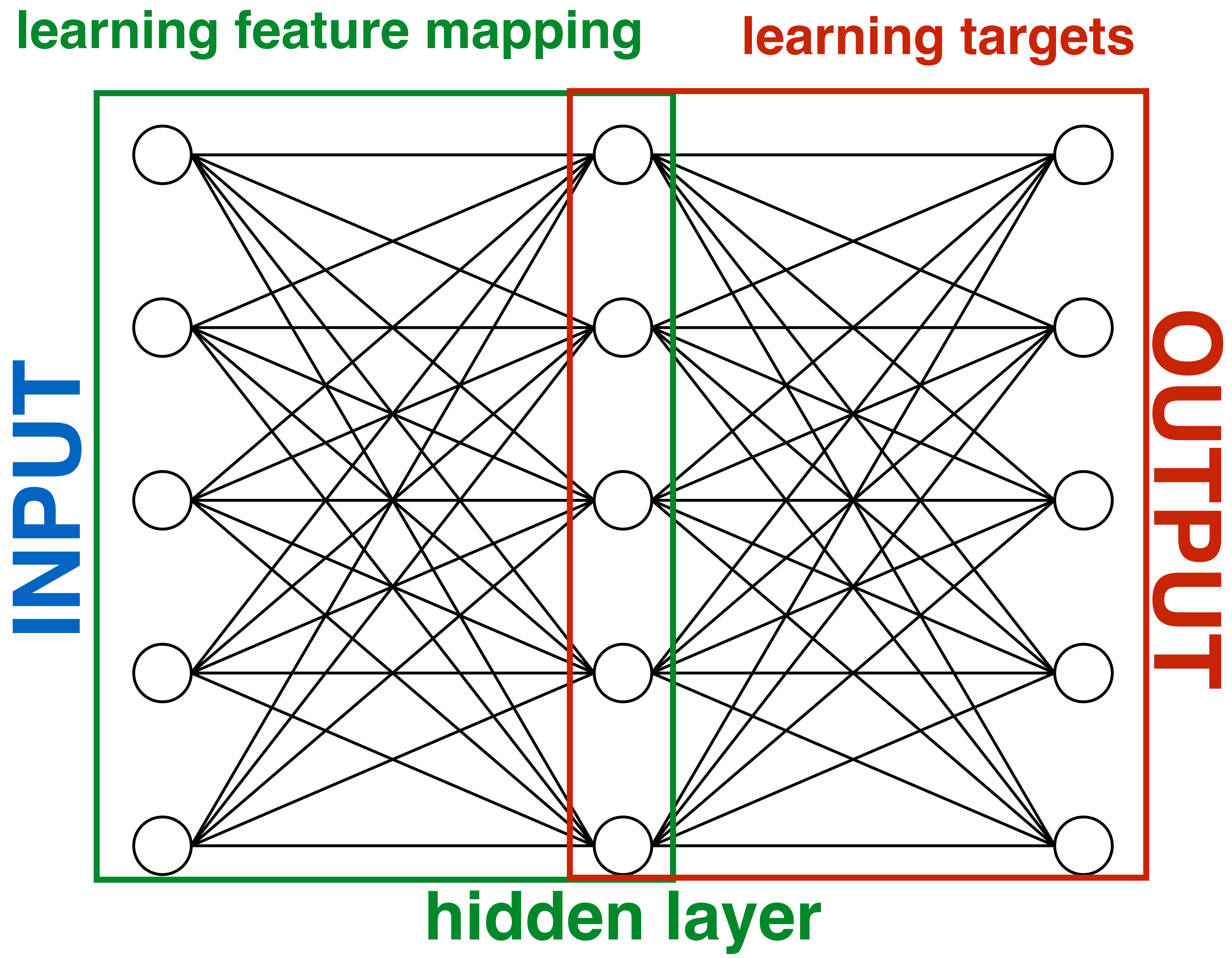
params. for learning targets

params. for learning the feature mapping

*does not make sense if inner models are purely linear, need a non-linearity (e.g. logistic regression)

A view on deep networks

- Single hidden layer – linear stacking of linear models:*



$$\hat{\mathbf{y}} = g_{(2)}\left(\mathbf{W}_{(2)}\, g_{(1)}(\mathbf{W}_{(1)}\, \mathbf{x})\right)$$

params. for learning targets

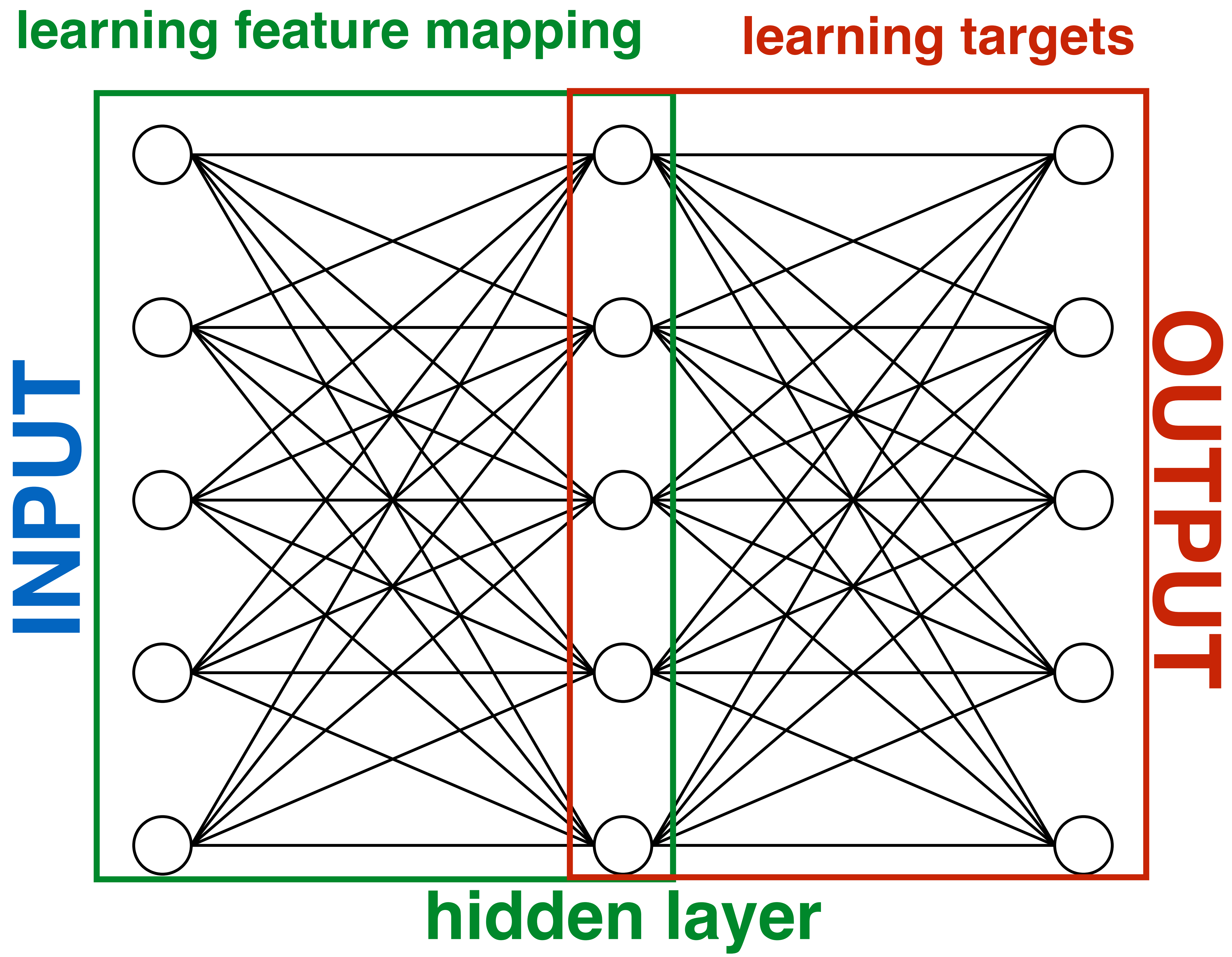
params. for learning the feature mapping

- Optimization problem no longer convex

*does not make sense if inner models are purely linear, need a non-linearity (e.g. logistic regression)

A view on deep networks

- Single hidden layer – linear stacking of linear models:*



$$\hat{\mathbf{y}} = g_{(2)}\left(\mathbf{W}_{(2)}\, g_{(1)}(\mathbf{W}_{(1)}\, \mathbf{x})\right)$$

params. for learning targets

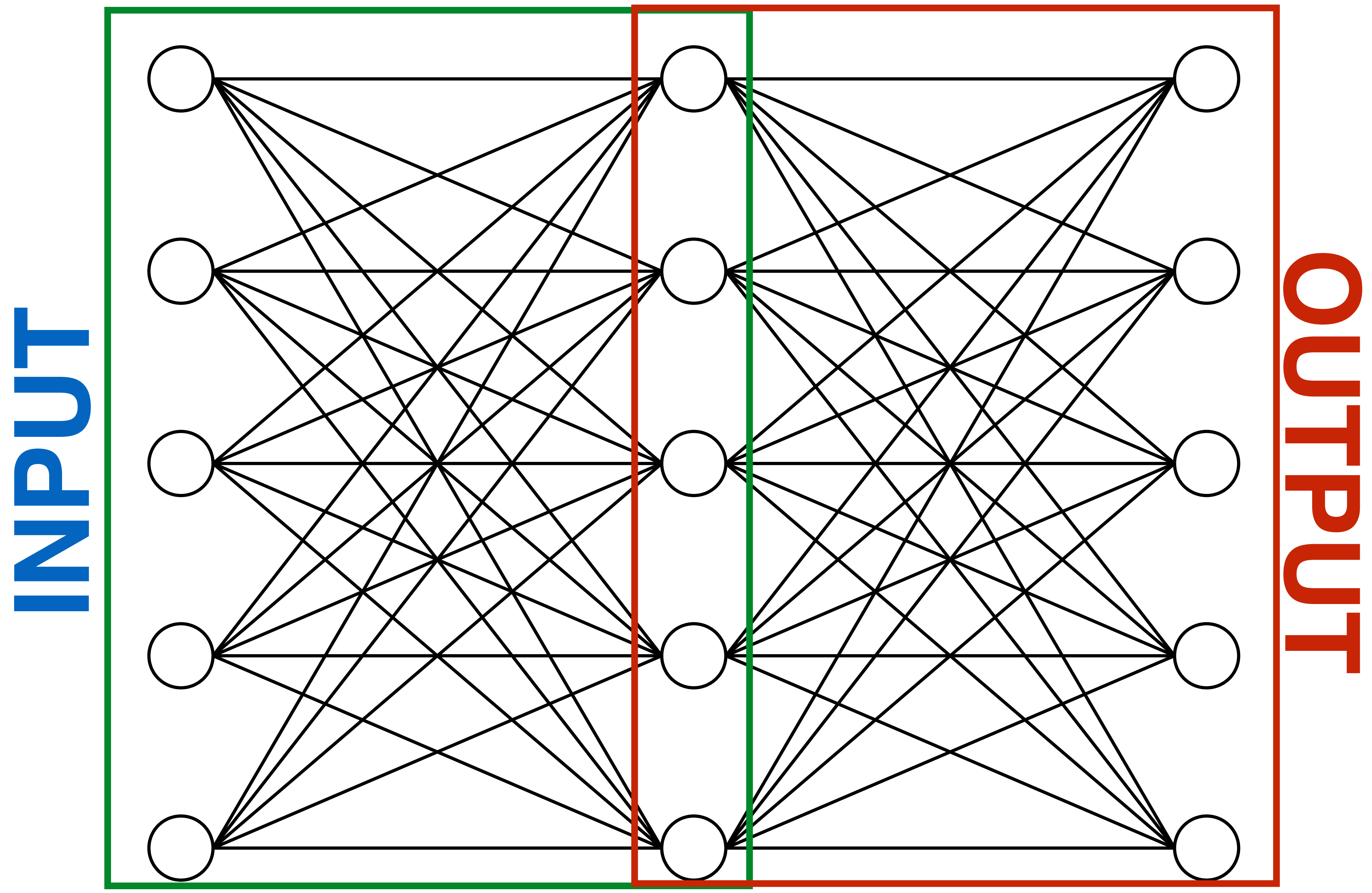
params. for learning the feature mapping

- Optimization problem no longer convex
  => global optimum is not guaranteed with gradient descent

*does not make sense if inner models are purely linear, need a non-linearity (e.g. logistic regression)

A view on deep networks

- Single hidden layer – linear stacking of linear models:*



$$\hat{\mathbf{y}} = g_{(2)}\left(\mathbf{W}_{(2)}\, g_{(1)}(\mathbf{W}_{(1)}\, \mathbf{x})\right)$$

params. for learning targets

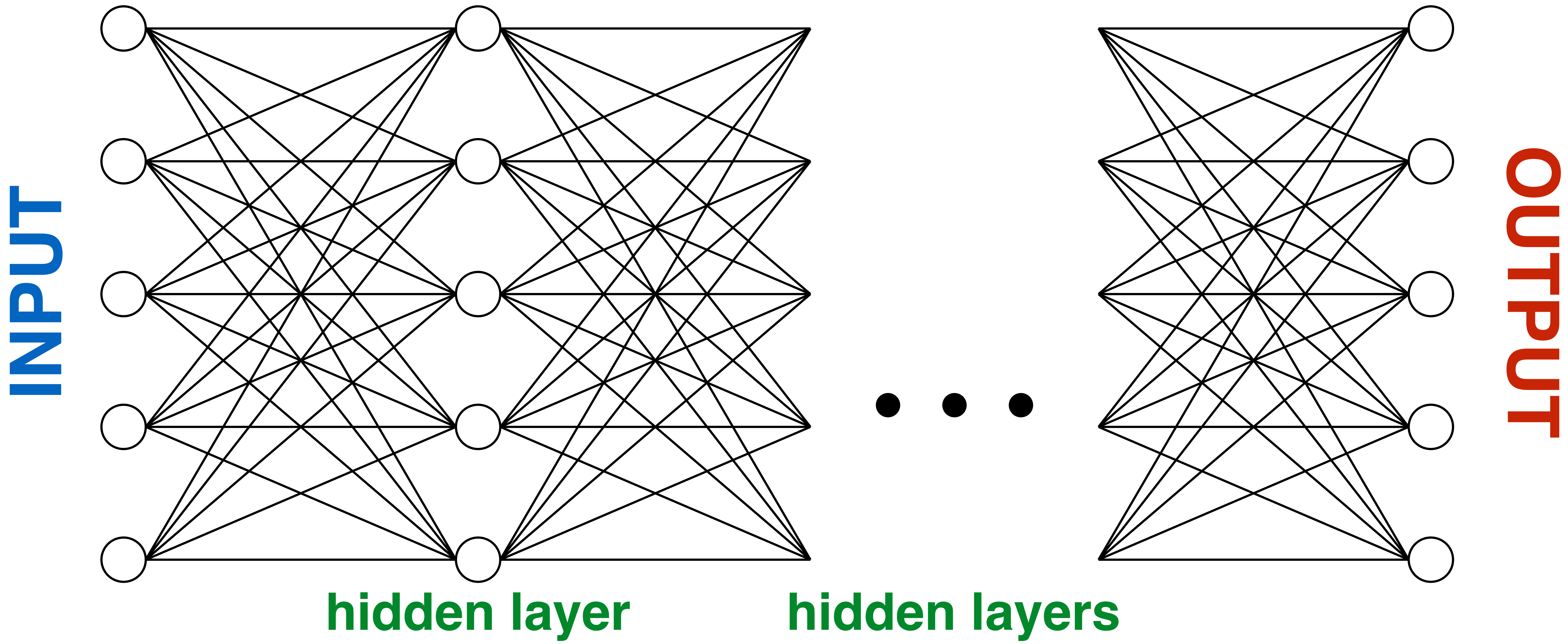
params. for learning the feature mapping

- Optimization problem no longer convex
  => global optimum is not guaranteed with gradient descent
  – But we don't need the global optimum if we can just find a **good enough** feature mapping!

*does not make sense if inner models are purely linear, need a non-linearity (e.g. logistic regression)

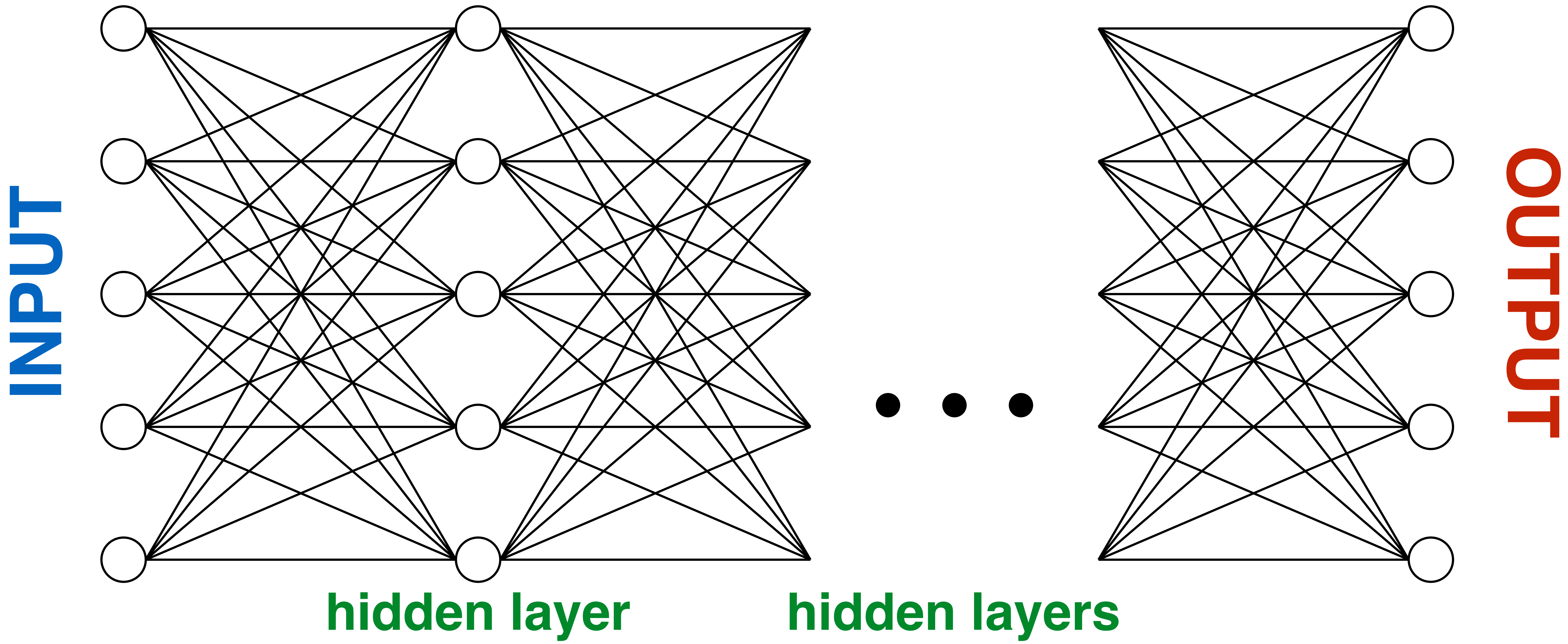
A view on deep networks

- Multiple hidden layers – linear stacking of linear stacking of … of linear models



A view on deep networks

- Multiple hidden layers – linear stacking of linear stacking of … of linear models
  - Learning feature mapping to learn feature mapping to learn … to learn targets

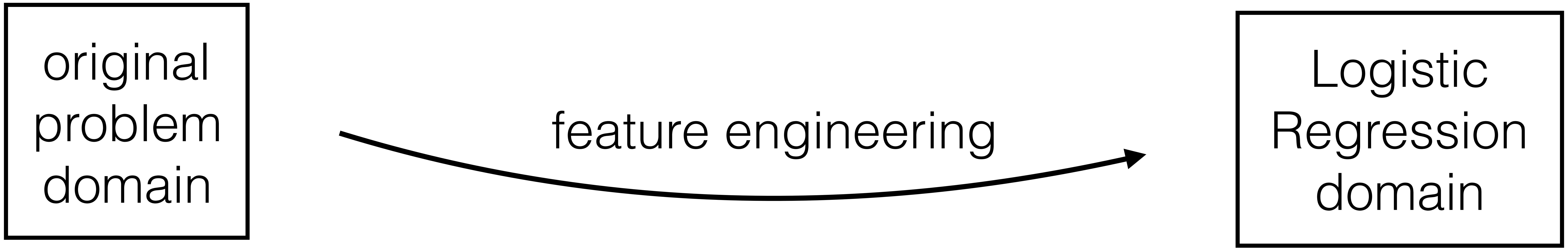


Solving an ML problem

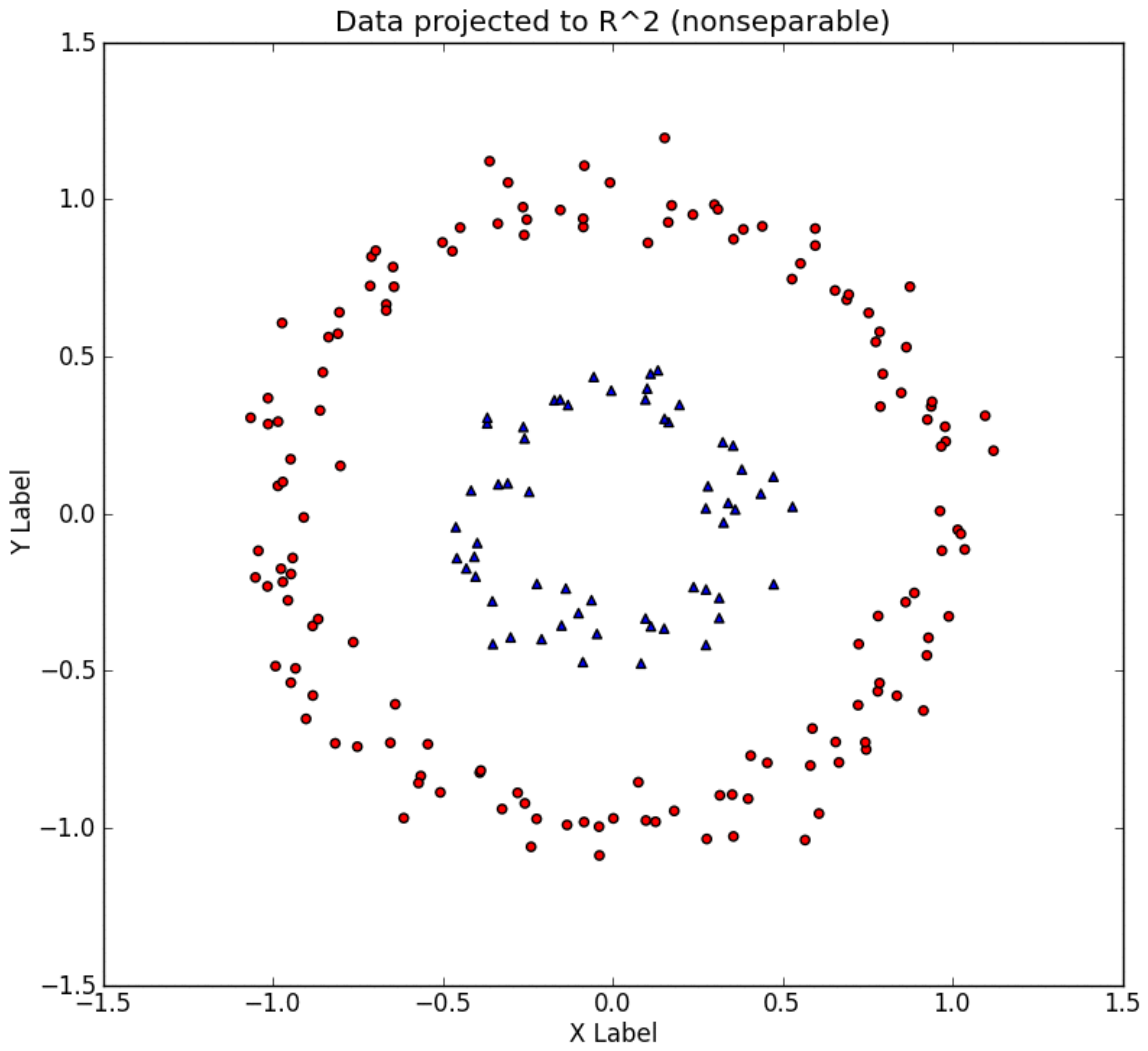
Traditional ML:

- Use:
  - prior knowledge about the problem
  - assumptions about the data

- to reformulate the problem
  - make it solvable by a given algorithm (e.g. Logistic Regression)



Example



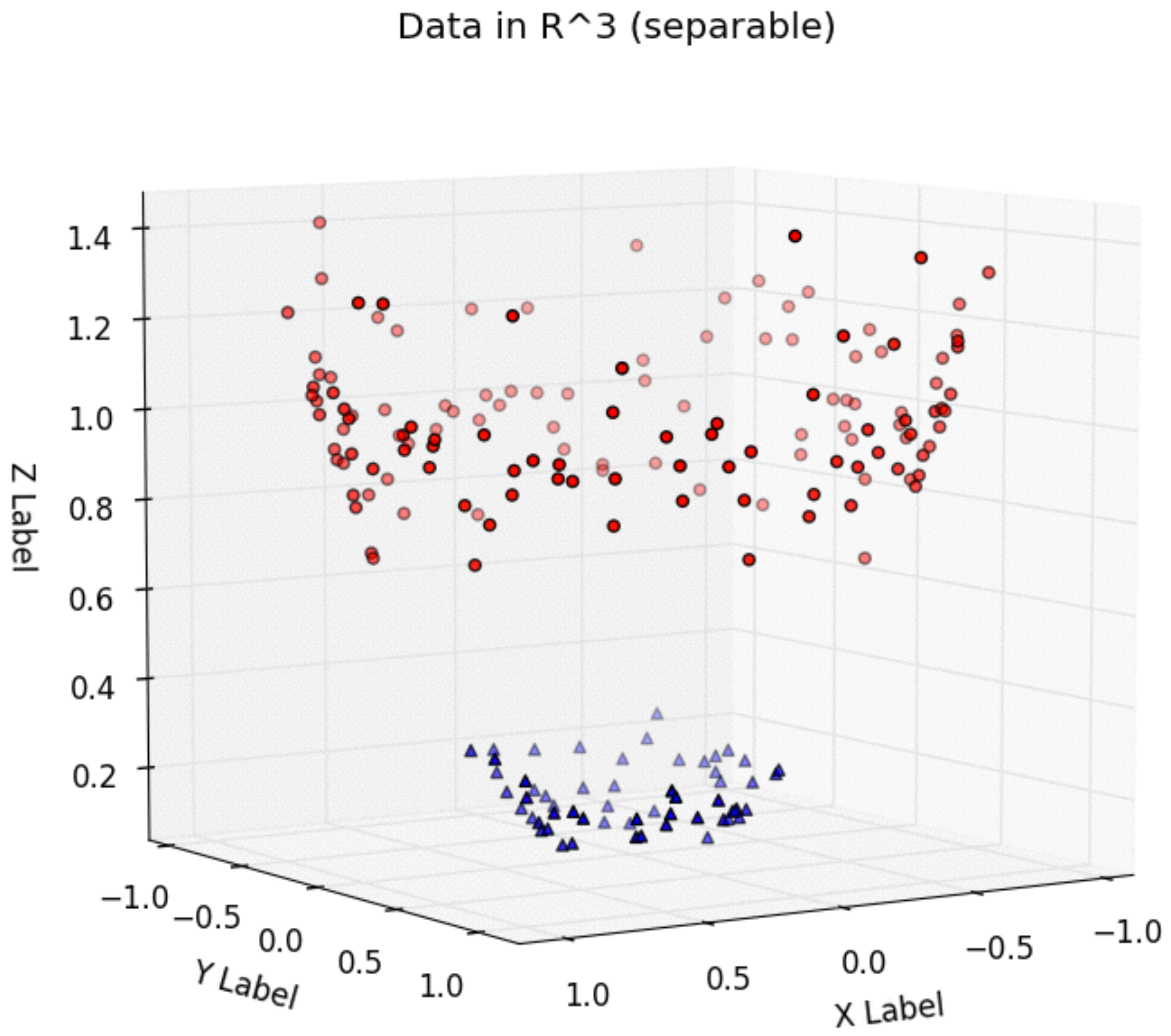
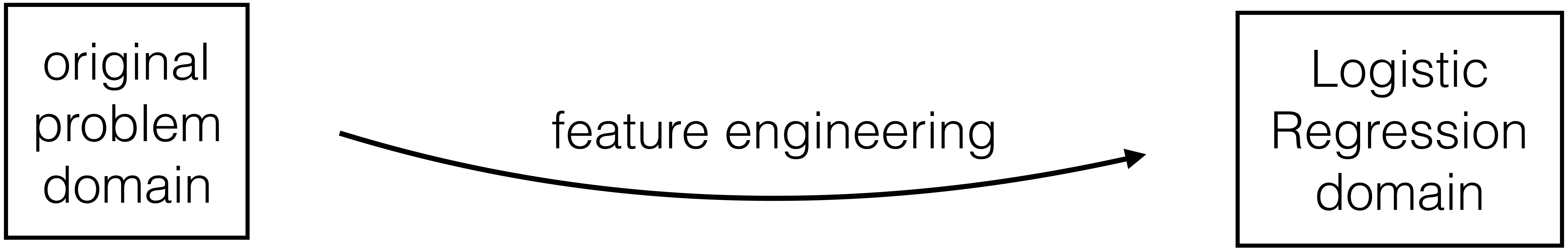


image source: https://towardsdatascience.com/understanding-the-kernel-trick-e0bc6112ef78

Solving an ML problem → DL

Traditional ML:

- Use:
  - prior knowledge about the problem
  - assumptions about the data
- to reformulate the problem
  - make it solvable by a given algorithm (e.g. Logistic Regression)

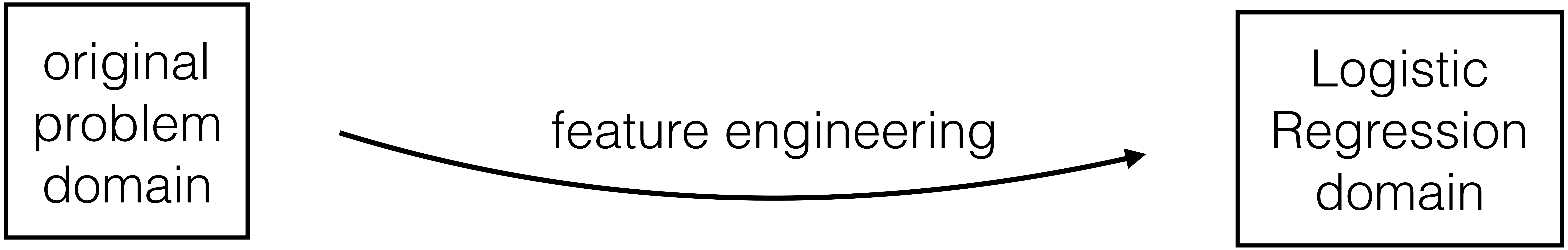


Solving an ML problem → DL

~~Traditional ML:~~ **Deep Learning:**

- Use:
  - prior knowledge about the problem
  - assumptions about the data

- to reformulate the problem
  - make it solvable by a given algorithm (e.g. Logistic Regression)



Solving an ML problem → DL

~~Traditional ML:~~ **Deep Learning:**

- Use:
  - ~~prior knowledge about the problem~~
  - ~~assumptions about the data~~

  – neural network

- to reformulate the problem
  - make it solvable by a given algorithm (e.g. Logistic Regression)

original problem domain

feature engineering →

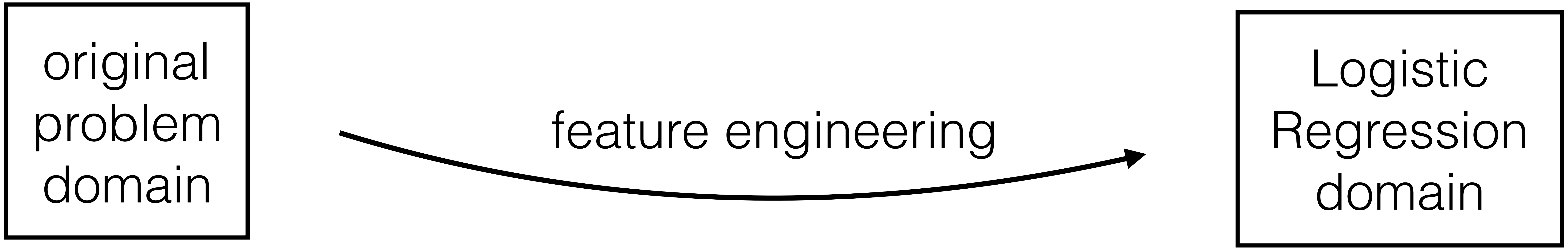
Logistic Regression domain

Solving an ML problem → DL

~~Traditional ML:~~ **Deep Learning:**

- Use:
  - ~~prior knowledge about the problem~~
  - ~~assumptions about the data~~

  – neural network
- to automatically reformulate the problem
  - make it solvable by a given algorithm (e.g. Logistic Regression)



Solving an ML problem → DL

~~Traditional ML:~~ **Deep Learning:**

- Use: – neural network
  - ~~prior knowledge about the problem~~
  - ~~assumptions about the data~~
- automatically to reformulate the problem
  - make it solvable ~~by a given algorithm~~ by the last layer (e.g. Logistic Regression)

original problem domain → feature engineering → Logistic Regression domain

Solving an ML problem → DL

~~Traditional ML:~~ **Deep Learning:**

- Use:
  - ~~prior knowledge about the problem~~
  - ~~assumptions about the data~~

  – neural network

- automatically to reformulate the problem
  - make it solvable ~~by a given algorithm~~ by the last layer (e.g. Logistic Regression)

original problem domain → neural network ~~feature engineering~~ → Logistic Regression domain

Solving an ML problem → DL

~~Traditional ML:~~ **Deep Learning:**

- Use: – neural network
  - ~~prior knowledge about the problem~~
  - ~~assumptions about the data~~
- automatically to reformulate the problem
  - make it solvable ~~by a given algorithm~~ by the last layer (e.g. Logistic Regression)

original problem domain $\xrightarrow{\text{neural network}}$ Logistic Regression domain

~~feature engineering~~

What's the role of the data scientist then?

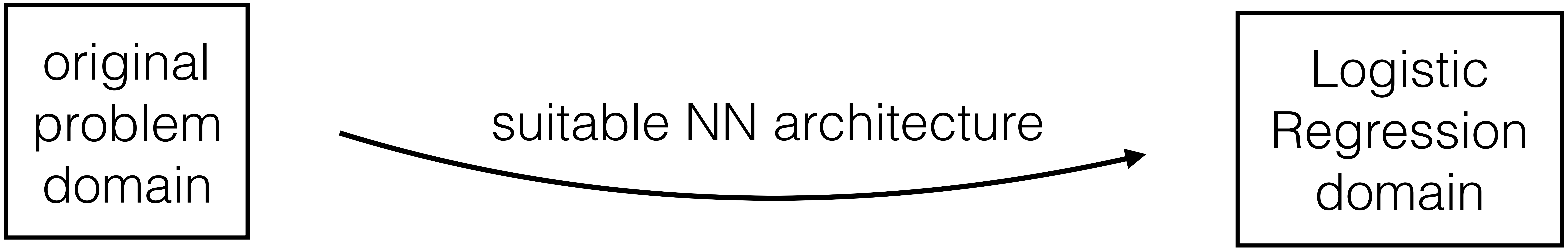
Deep Learning

- DL is not a universal tool to automatically find right feature mappings.
- It's rather a toolkit allowing to express the assumptions about the data in a general way
  - When done right, this helps the algorithm to find the right mappings

Solving an ML problem with DL

Deep Learning:

- Use:
  - prior knowledge about the problem
  - assumptions about the data
- to build a NN architecture
  - that can find the solution



Solving an ML problem with DL

Deep Learning:

- Use:
  - prior knowledge about the problem
  - assumptions about the data
- to build a NN architecture
  - that can find the solution

narrow down the class of feature mappings

original problem domain $\xrightarrow{\text{suitable NN architecture}}$ Logistic Regression domain

Example: detecting a kitten

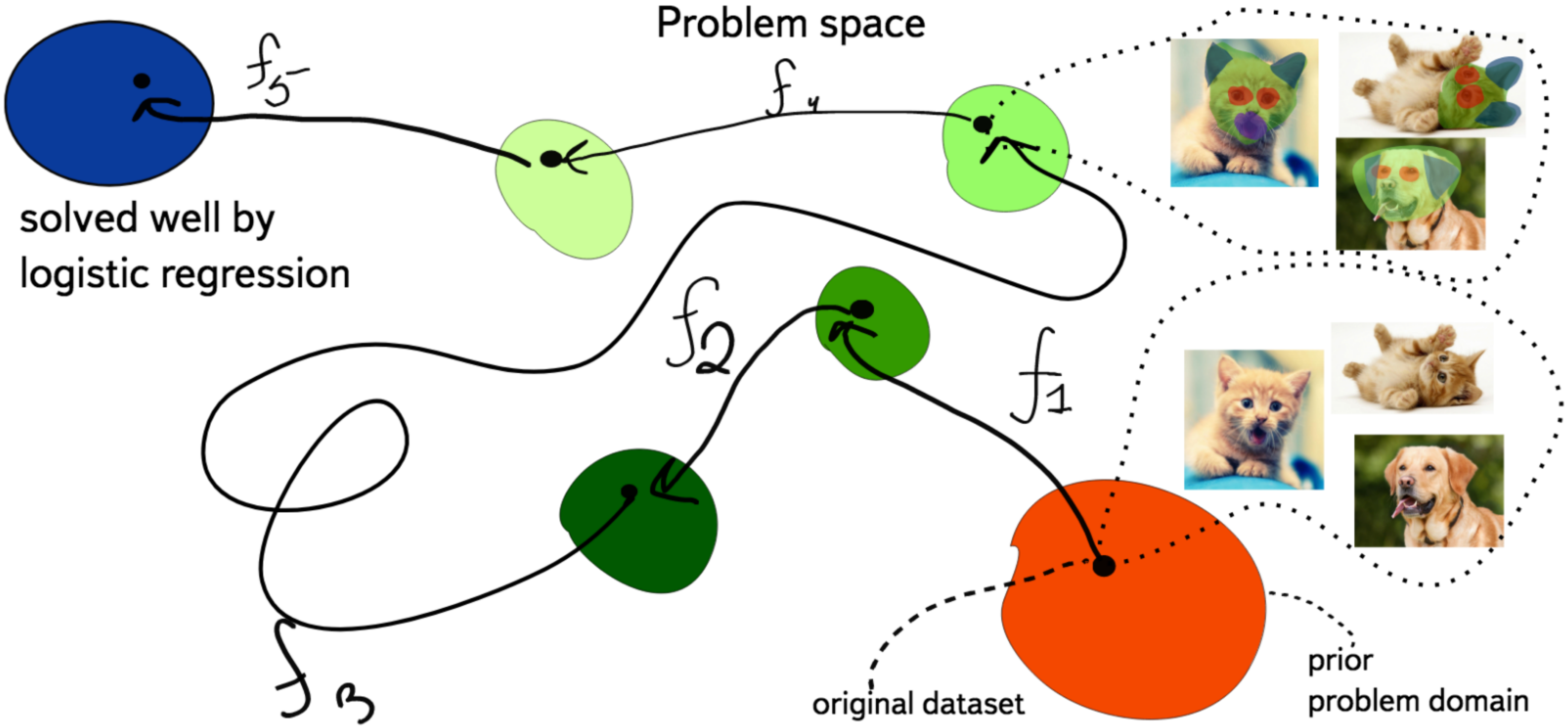
Original domain:

- Arrays of bytes of size 3xHxW (typically millions of bytes)

Building a traditional ML model:

- edge detection
- image segmentation
- eyes, ears, nose models
- fit nose, eyes, ears
- average color of segments
- kitten’s face model
- logistic regression

Example: detecting a kitten



Example: detecting a kitten

DL approach (naive)

- typical images are >1M pixels (features)
- how many mappings (hidden layer neurons) do we need?
  - each mapping triggered by the presence of a particular pixel combination
  - even for binary mappings it's 10^{12} possibilities
  - at least 1M of them are informative
    - since horizontal and vertical shifts of informative mappings give informative mappings as well
- Doing the math:

1M inputs x 1M hidden ~ 1M^2 weights ~ 4TB of weights

Example: detecting a kitten

DL approach (naive)

- typical images are >1M pixels (features)
- how many mappings (hidden layer neurons) do we need?
  - each mapping triggered by the presence of a particular pixel combination
  - even for binary mappings it's 10^{12} possibilities
  - at least 1M of them are informative
    - since horizontal and vertical shifts of informative mappings give informative mappings as well
- Doing the math:

1M inputs x 1M hidden ~ 1M^2 weights ~ 4TB of weights

Good luck training such network!

DL approach

- prior knowledge: translational symmetry
- architecture utilizing this symmetry: convolutional NN
- main idea:
  - reuse the same weights on different patches of the image to extract similar features from different parts

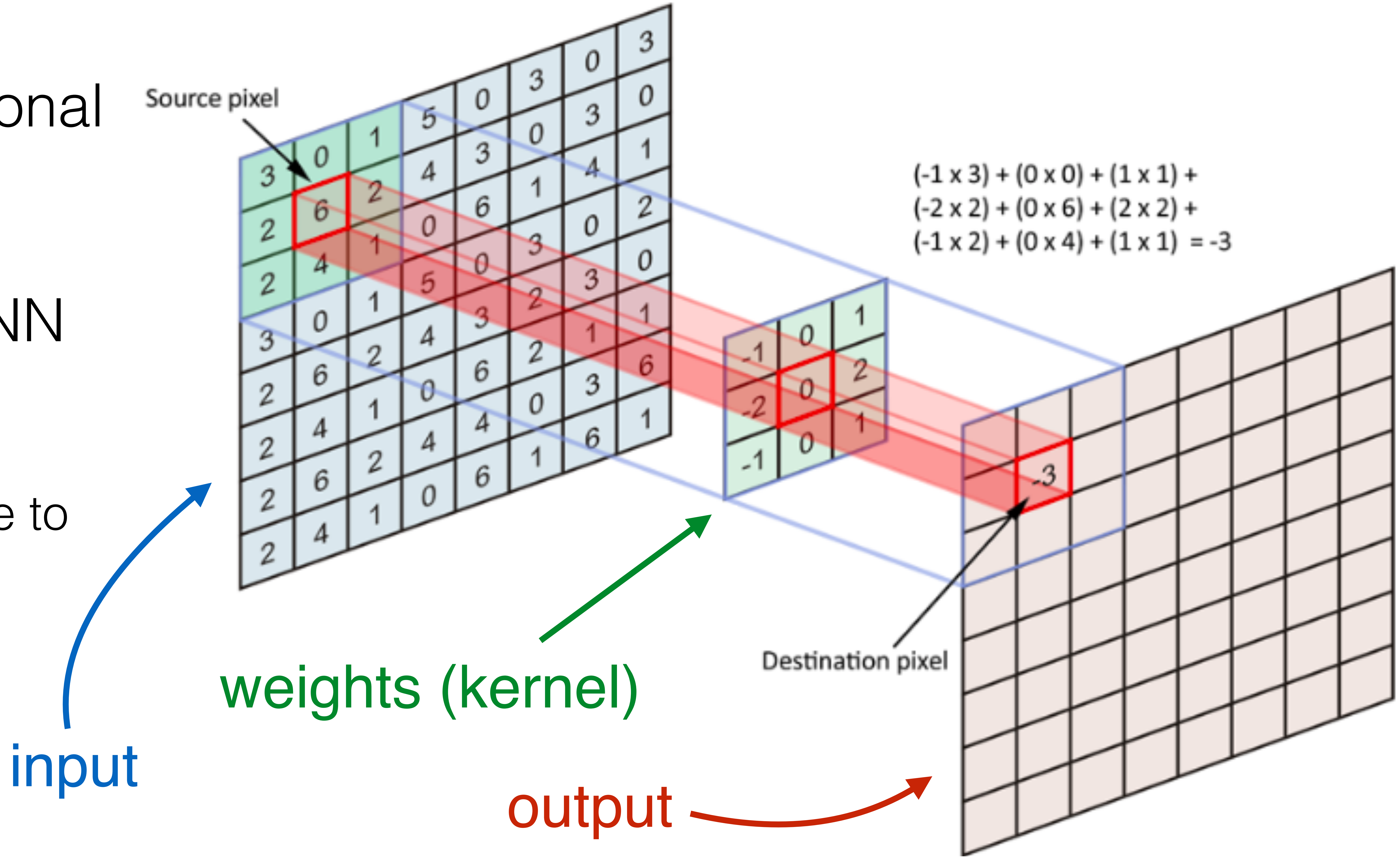


image source: https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/

DL approach

- prior knowledge: complicated images can be built from primitives
- architecture utilizing this: deep convolutional NN (stacked convolutions)

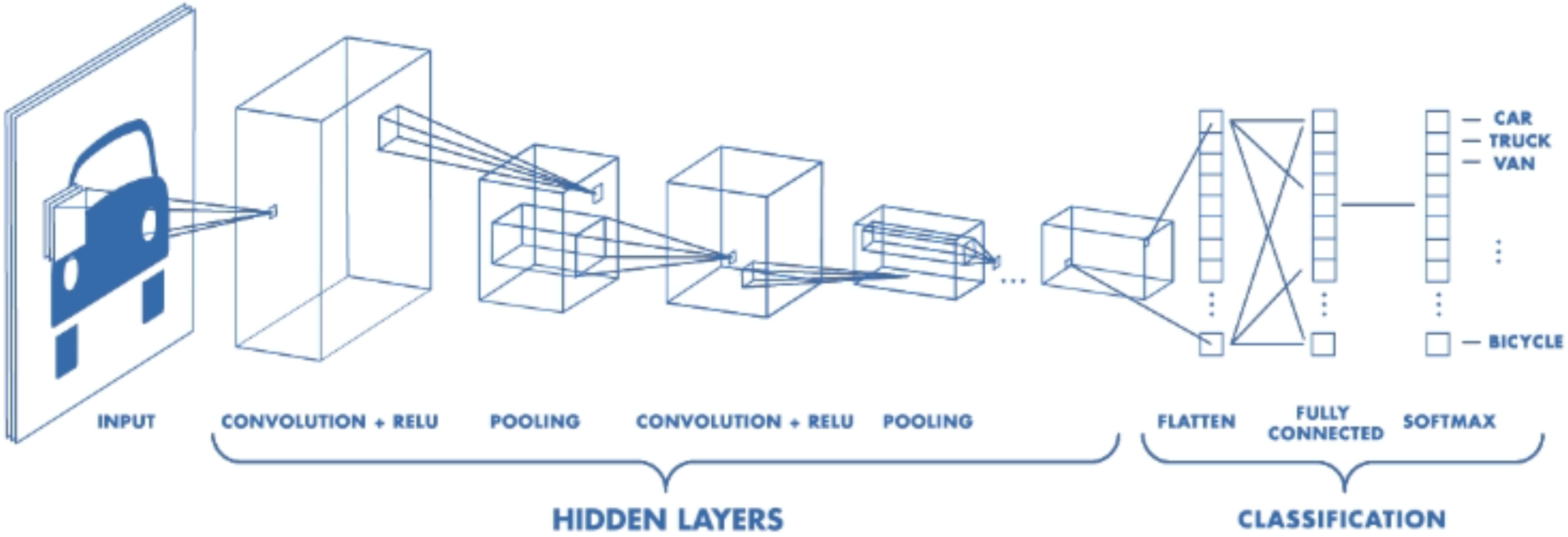
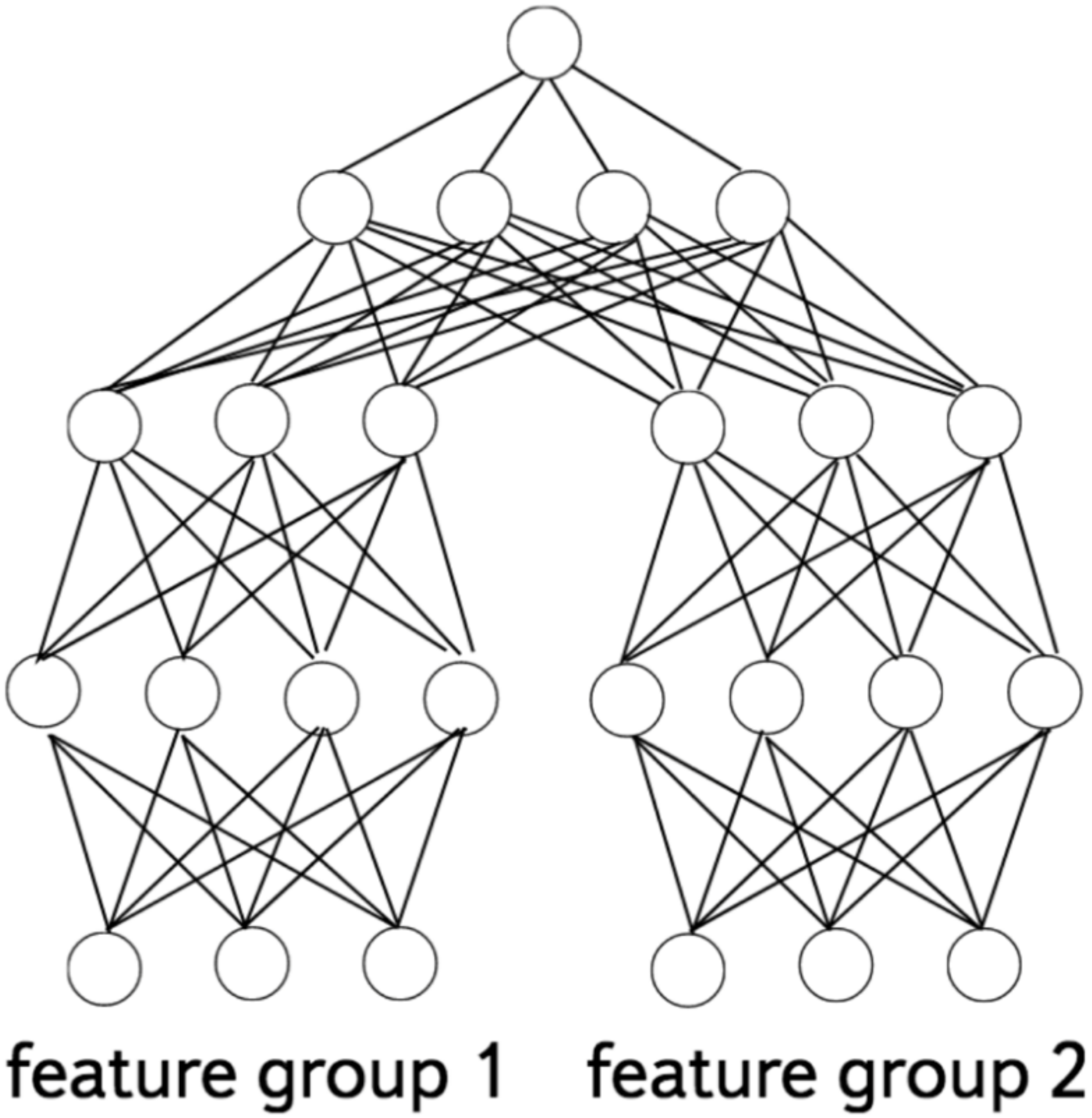


image source: https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/

Example: separate feature subspaces

- Prior knowledge:
  - groups of features should not interact directly (e.g. image and sound)
- Solution: two sub-networks merged at higher level



Example: autoencoders

- Prior knowledge:
  - the data lives on a low-dimensional manifold of the high-dimensional space (e.g. images, sounds, etc.)
- Idea: it should be possible to represent the data in a lower-dimensional space
- Solution: autoencoders

$$L \equiv L\left(\mathbf{x}, f_\theta\left(g_\phi(\mathbf{x})\right)\right)$$

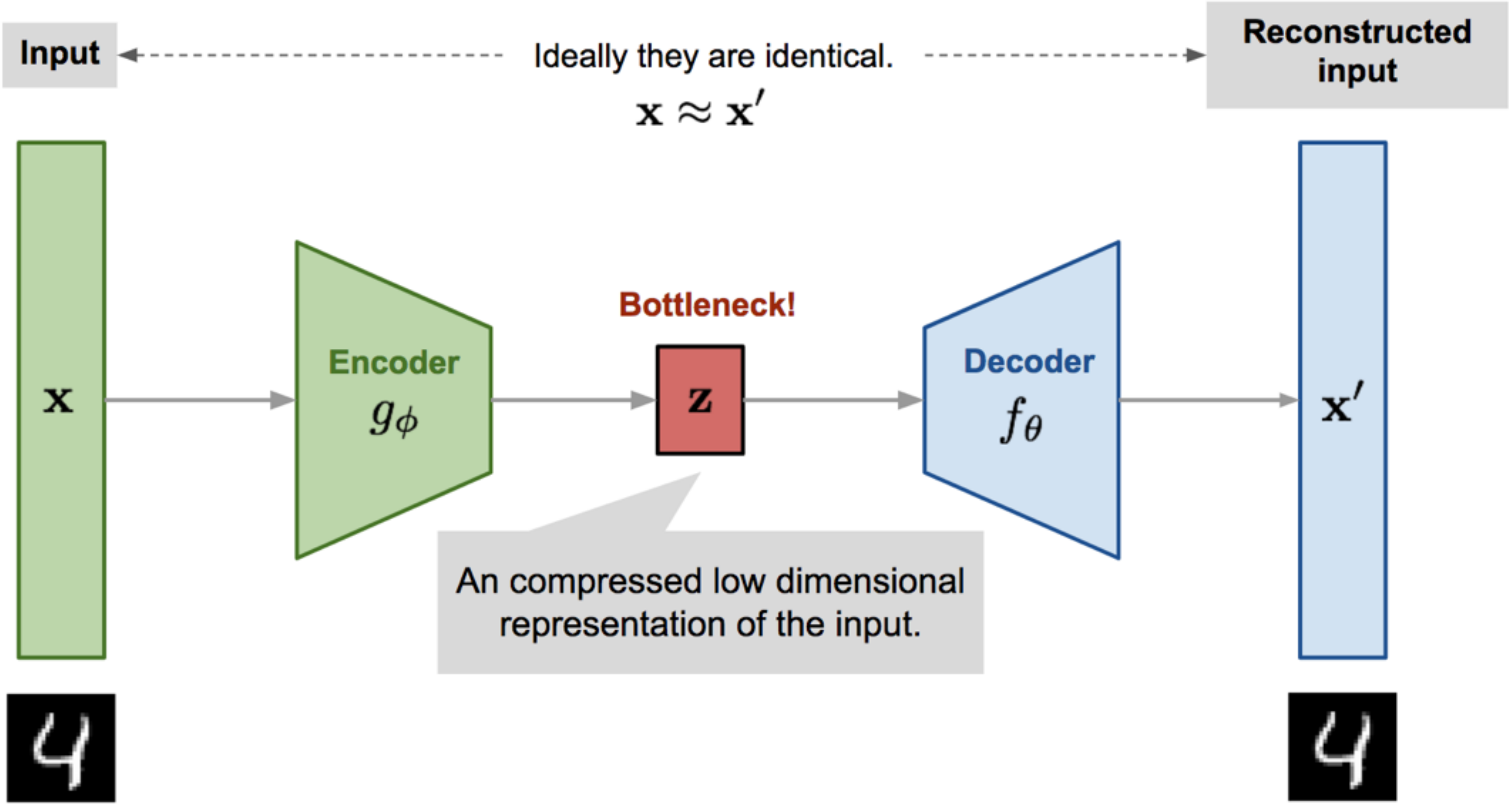


image source: https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html

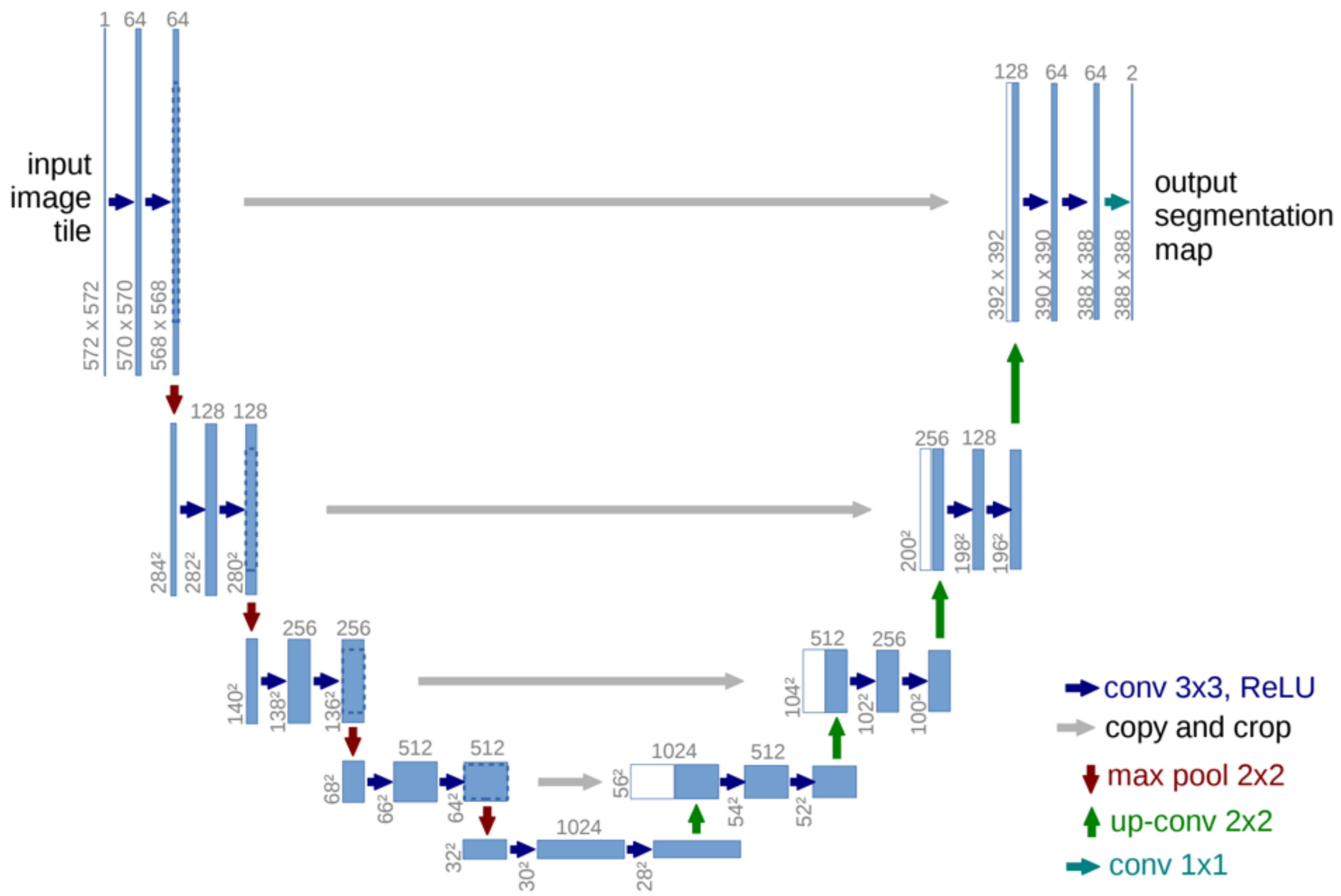
Example: autoencoders

- Autoencoders can be useful for supervised learning tasks
- E.g.: limited labelled data, abundance of unlabelled data
- Solution:
  - Train AE on the unlabelled data
  - Train supervised model on the AE's low-dimensional representation of the labeled data
- Optionally:
  - Do the supervised and unsupervised training simultaneously, i.e.:

$$L = \mathop{\mathbb{E}}_{X,Y\sim\text{supervised}} \left[l_1 \left(h_\psi(g_\phi(x)), y \right) \right] + \lambda \cdot \mathop{\mathbb{E}}_{X\sim\text{unsupervised}} \left[l_2 \left(f_\theta(g_\phi(x)), x \right) \right]$$

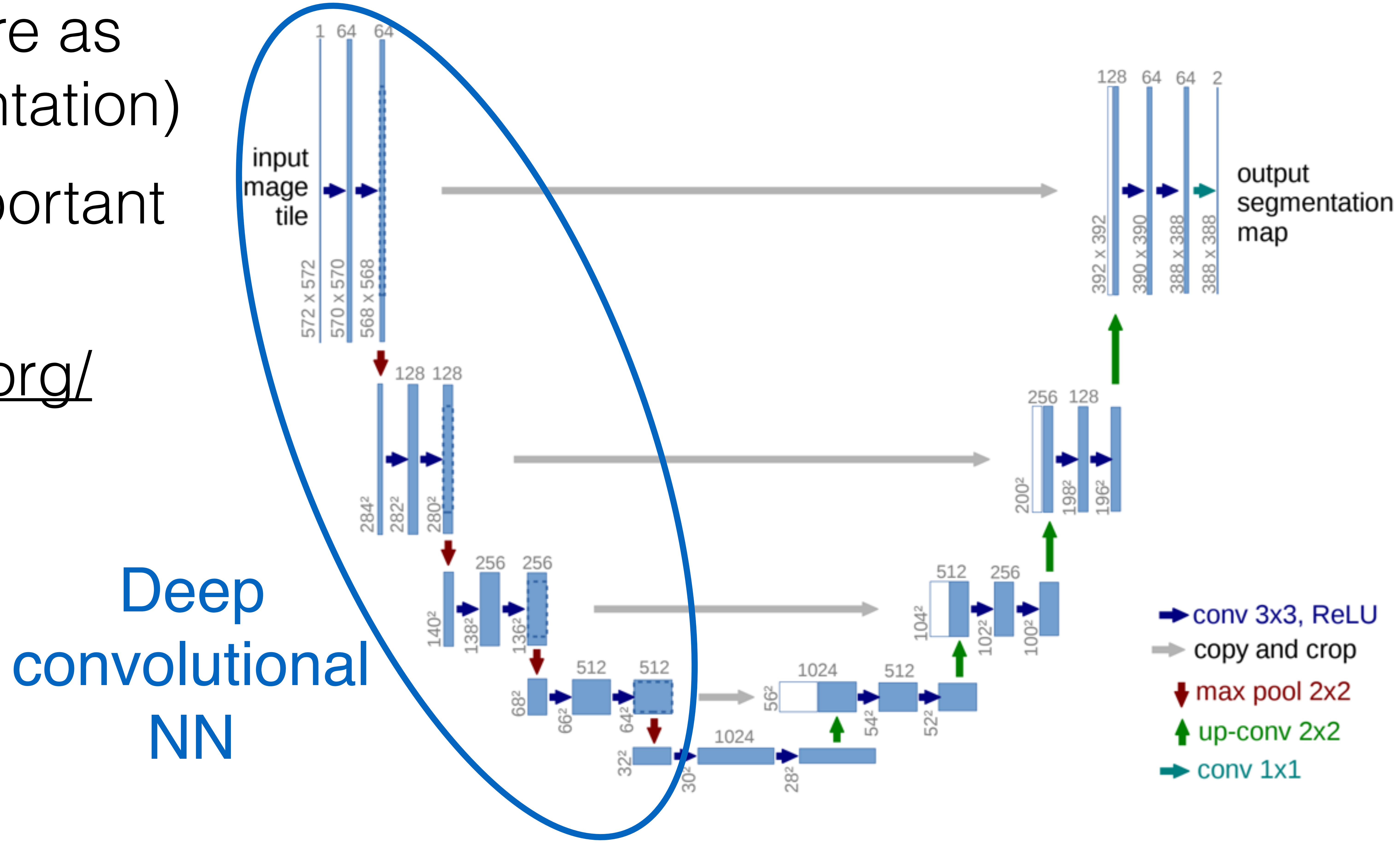
Example: U-net

- Prior knowledge:
  - targets have same structure as inputs (e.g. image segmentation)
- Idea: low-level context is important for higher-level features
- Solution: U-net (https://arxiv.org/abs/1505.04597)



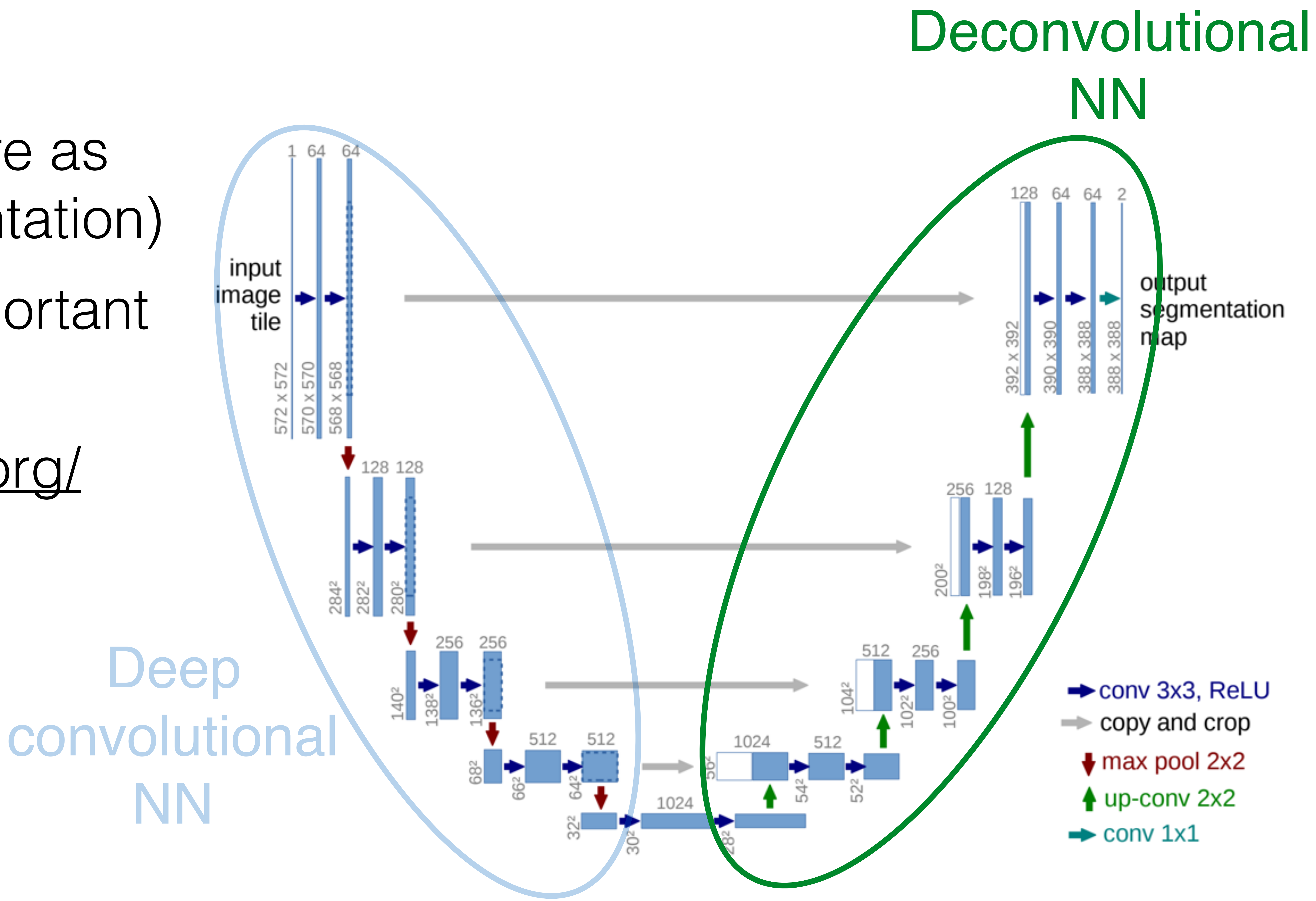
Example: U-net

- Prior knowledge:
  - targets have same structure as inputs (e.g. image segmentation)
- Idea: low-level context is important for higher-level features
- Solution: U-net (https://arxiv.org/abs/1505.04597)



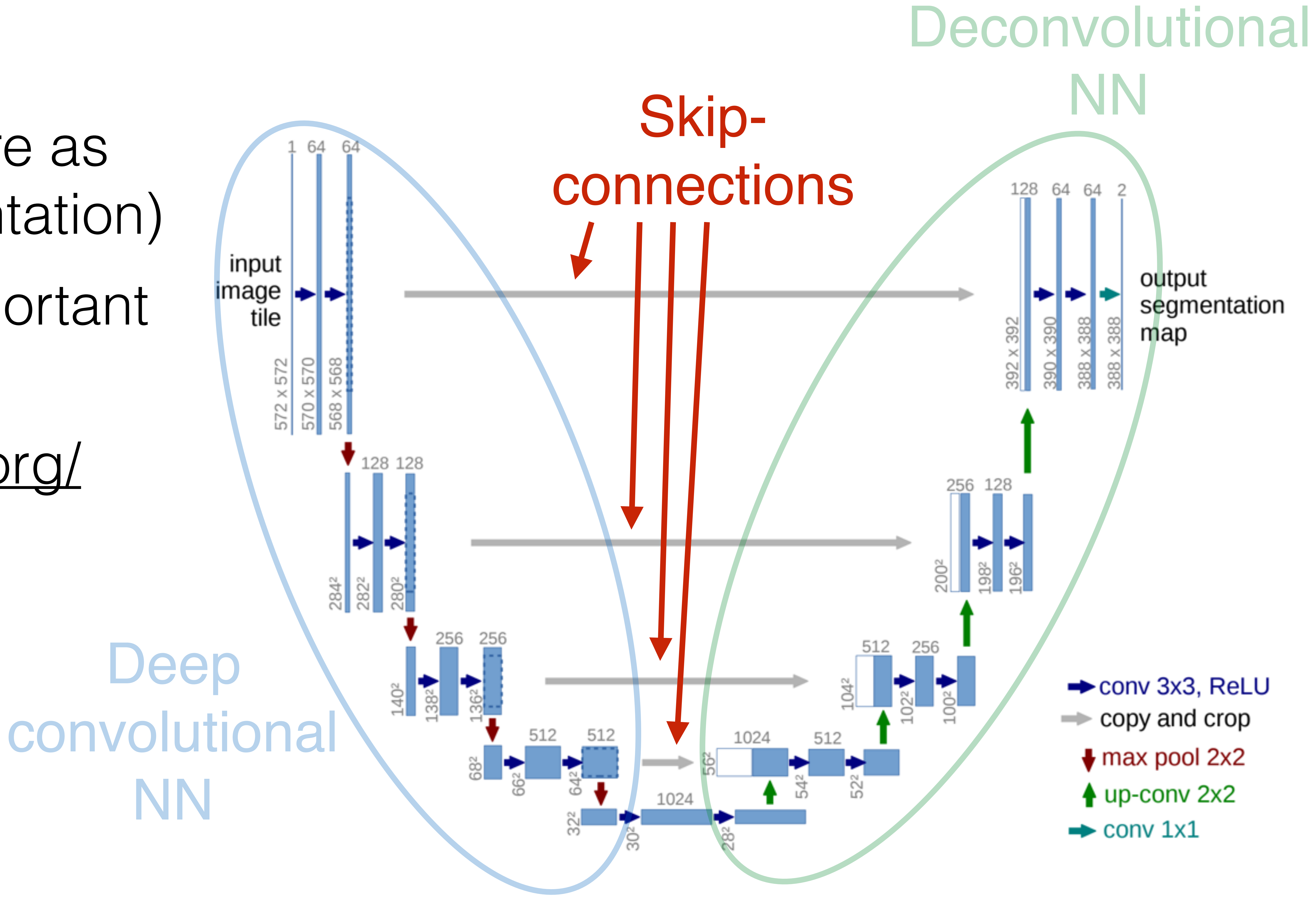
Example: U-net

- Prior knowledge:
  - targets have same structure as inputs (e.g. image segmentation)
- Idea: low-level context is important for higher-level features
- Solution: U-net (https://arxiv.org/abs/1505.04597)



Example: U-net

- Prior knowledge:
  - targets have same structure as inputs (e.g. image segmentation)
- Idea: low-level context is important for higher-level features
- Solution: U-net (https://arxiv.org/abs/1505.04597)



Summary

- DL is not about stacking as many layers as possible until the problem is solved
- It’s rather about being creative to incorporate the prior knowledge into the network architecture and loss function
  - Everything possible!*

*if it has a gradient

A few tips and tricks

Good initialization

Q: can we initialize all the weights with zeros?

Good initialization

Random initialization

- If the weights are too small, the signal shrinks as it passes through the network
- If the weights are too large – the signal may explode.
- Common heuristics:
  - Bias initialized with zero
  - Weight initialized randomly with zero mean and with variance equal to:
    - $1/n_{in}$ — Xavier/Glorot, suitable for tanh activation
    - alternatively, $2/(n_{in} + n_{out})$
    - $2/n_{in}$ — He, suitable for ReLU activation

Good initialization

- Simple way to check initialization:
  - Feed a standard normal into your NN (before training)
  - Check activations by hand
  - Check gradients by hand

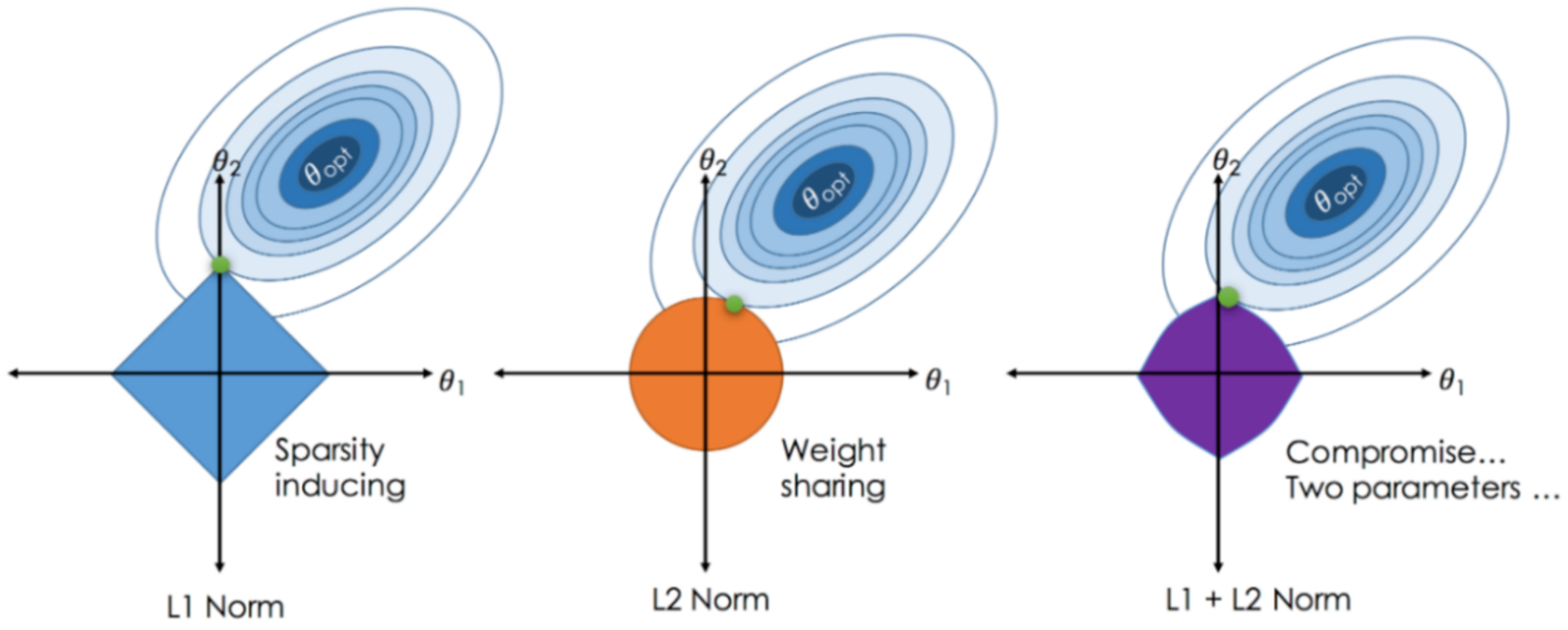
Data augmentation

- In many cases labels are symmetric wrt some input transformations
  - e.g. a mirrored or rotated kitten is still a kitten
  - note: not true for some objects, e.g. STOP sign, or any other object with text
- One can augment the training set by adding samples with such random transformations

- Another option: add random noise to the input data
  - tradeoff between information in the sample and the robustness of the output

L1/L2 regularization

- Argumentation for linear models still holds:

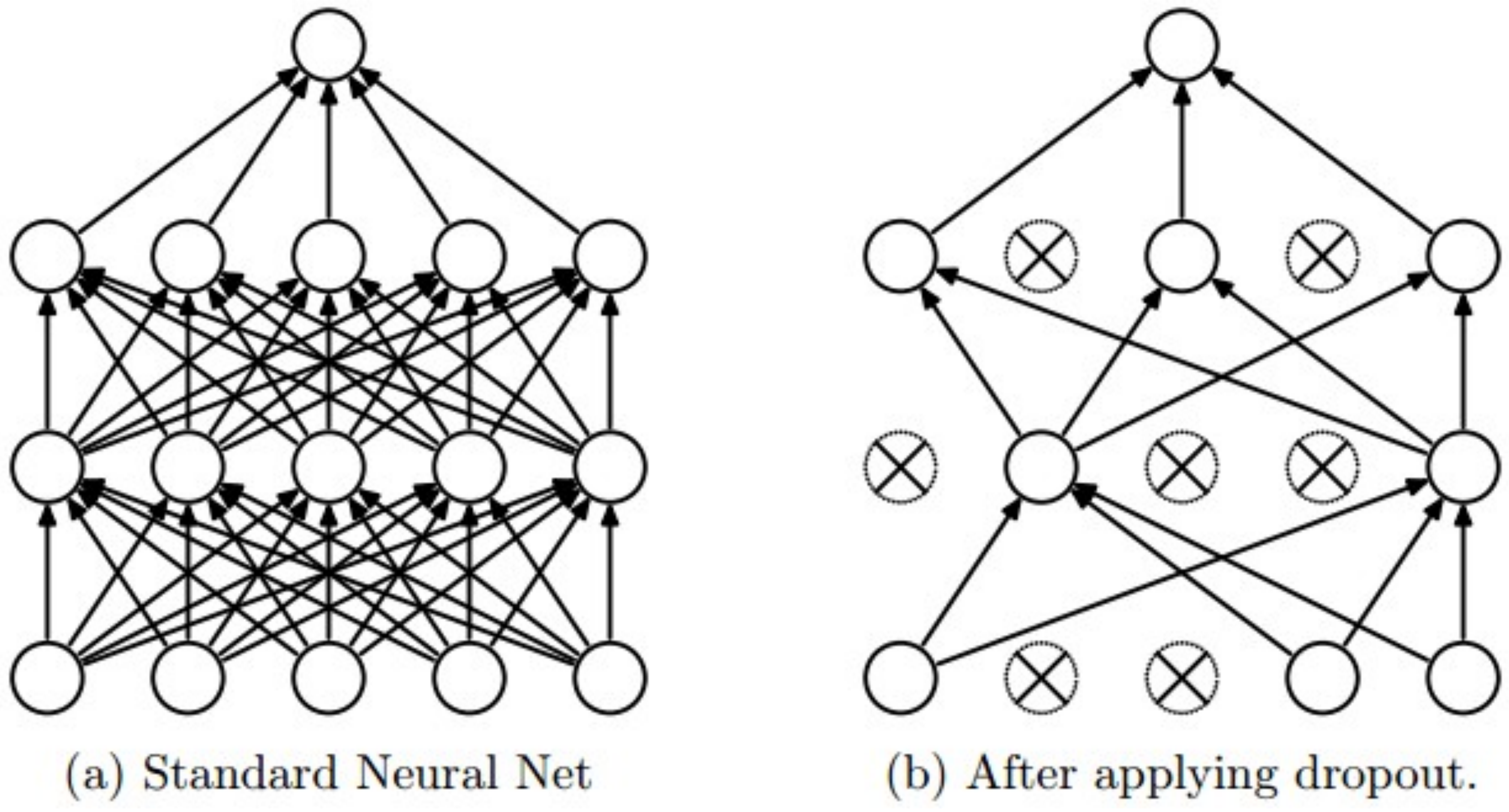


Dropout layers

$$\text{Dropout}(\mathbf{x}) = \text{diag}[\mathbf{a}] \cdot \mathbf{x}$$

$$\mathbf{a} \sim \text{Bernoulli}^n(\,\cdot\,, p)$$

where n is the size of vectors $\mathbf{x}$ and $\mathbf{a}$, and $\mathbf{a}$ is sampled independently for each object



- This means setting previous layer activations to 0 with probability (1 – p)
- Forces the network to learn same concepts through different routes
  - one can draw a parallel with bagging

Batch normalization

- For inference stage, mean and standard deviation are typically estimated from training data with a moving average

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Gradient-based optimization techniques

- Momentum SGD ~ use running-average gradient for parameter updates
- Adagrad ~ scale learning rate by 1 / sqrt(accumulated sum of gradients squared), independently for each parameter
- RMSprop ~ Adagrad with moving average squared gradient instead of accumulated sum
- Adadelta ~ RMSprop replacing learning rate with moving RMS(parameter update)
- Adam ~ RMSprop + Momentum
- AdaMax ~ Adam with l instead of the l2 norm

A comprehensive overview: http://ruder.io/optimizing-gradient-descent/index.html

Literature

- How BatchNorm helps optimization
  https://arxiv.org/abs/1805.11604
- Why momentum really works
  https://distill.pub/2017/momentum/