

# Supervised Learning: Neural networks and Deep learning

Jean Feng & Ali Shojaie

Aug 19-21, 2024  
Summer Institute in Statistics for Big Data  
University of Washington

# Outline

1. What is deep learning?
2. What can DL do and not do
3. Introduction to:
  1. Dense neural networks
  2. Convolutional neural networks
4. How to get started using neural networks

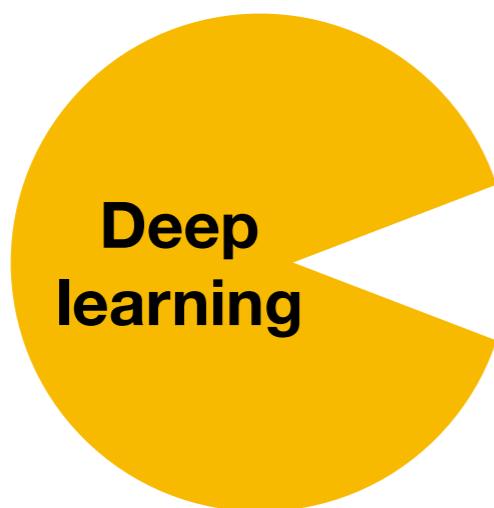
# Outline

1. What is deep learning?
2. What can DL do and not do
3. Introduction to:
  1. Dense neural networks
  2. Convolutional neural networks
4. How to get started using neural networks

# What is deep learning?

*“The grandly named statistical technique, put simply, gives computers a way to learn by processing vast amounts of data... Deep learning algorithms train on a batch of related data — like pictures of human faces — and are then fed more and more data, which steadily improve the software’s pattern-matching accuracy.”*

- The New York Times, “Is there a smarter path to artificial intelligence? Some experts hope so.” June 20, 2018.

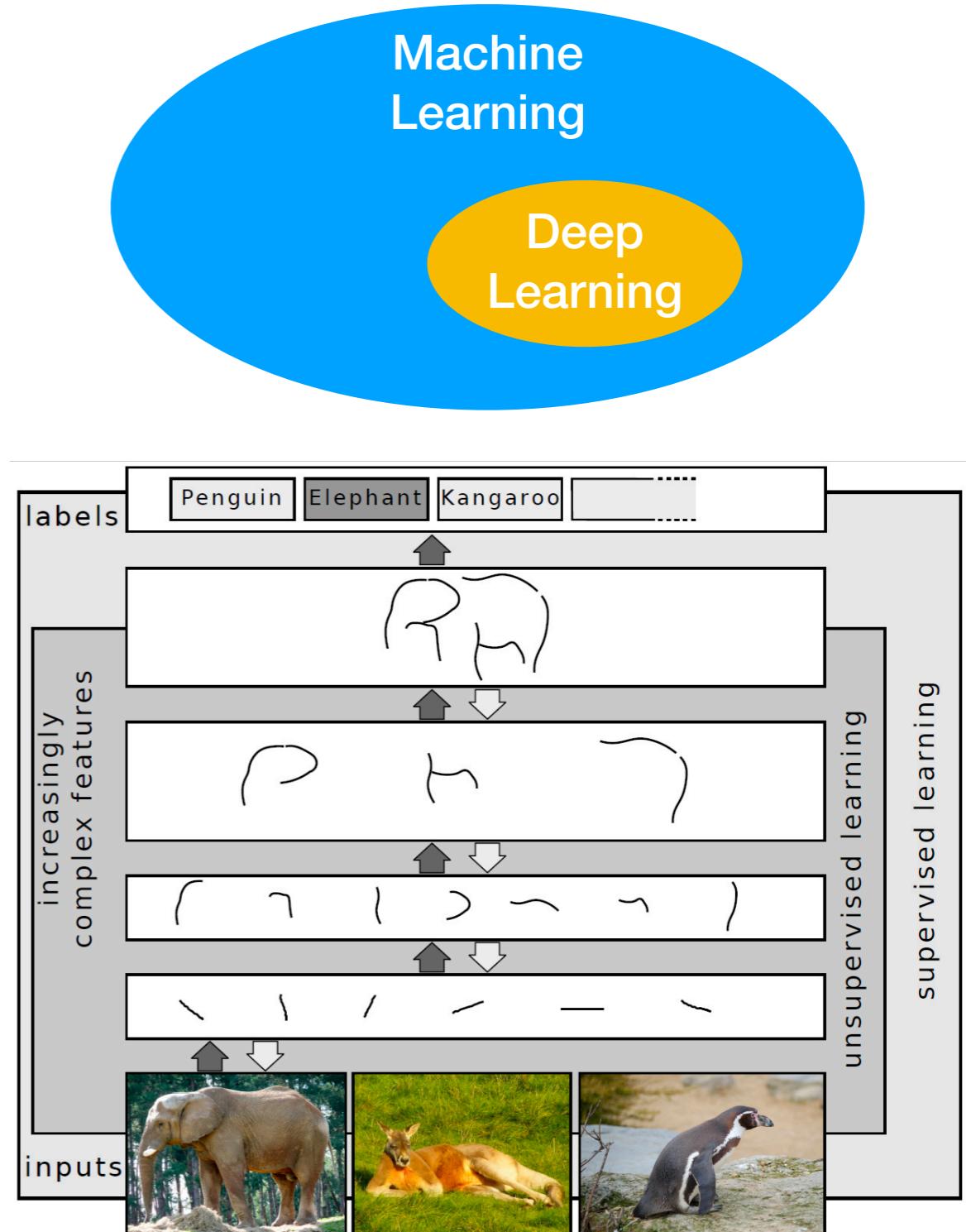


- **Data • • • Data • • • Data • • • Data • •**

# What is deep learning?

*“Deep learning is part of a broader family of machine learning methods based on artificial neural networks with representation learning... Deep learning is a class of machine learning algorithms that uses multiple layers to progressively extract higher level features from the raw input.”*

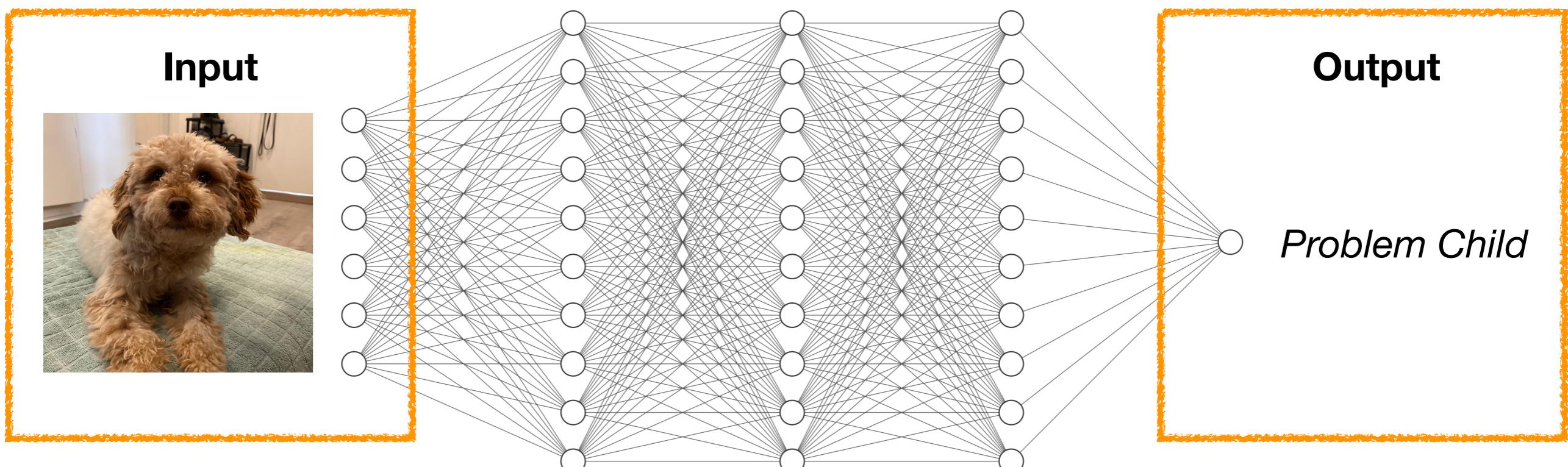
— Wikipedia



# What is deep learning?

Deep learning (DL) is a non-parametric machine learning method that builds large computational graphs called neural networks to model complex nonlinearities, interactions, and hierarchies in the data.

— A statistical perspective of DL

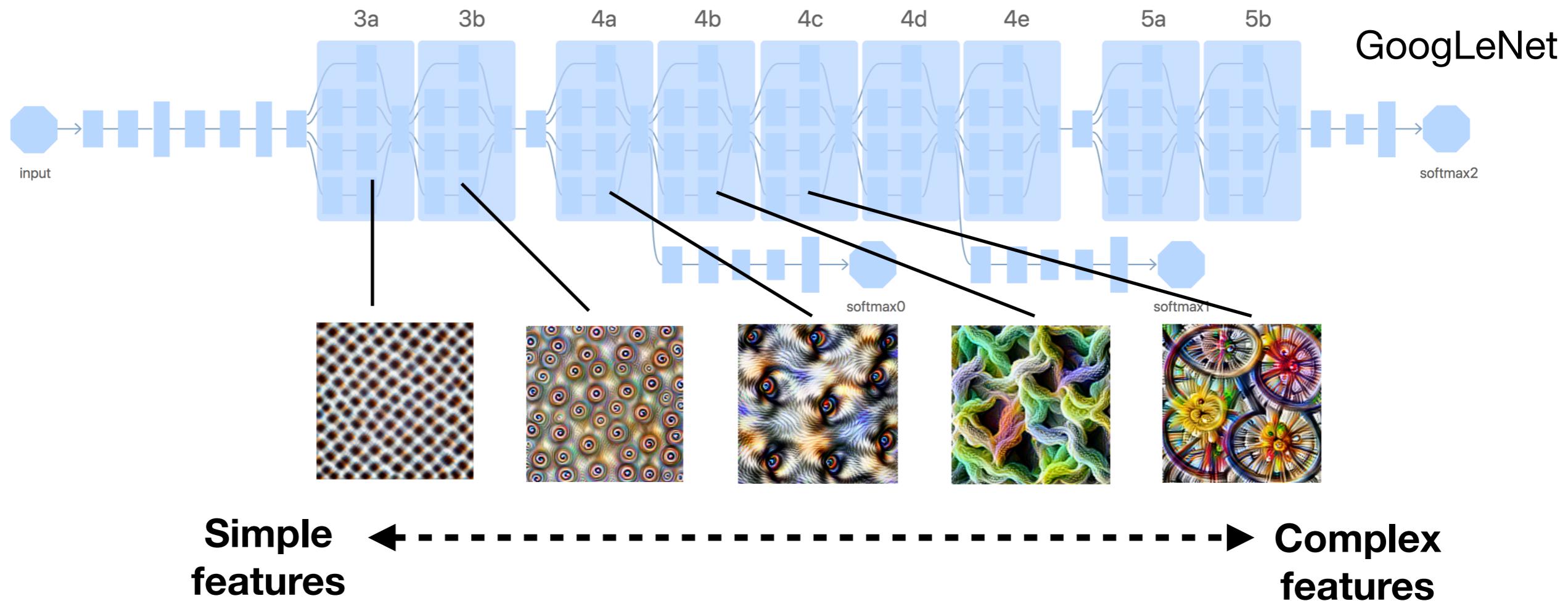


# Strengths of deep learning

- **Universal Approximation Theorem:** Given any continuous function, there exists a sufficiently large neural network that can approximate it within any positive margin of error (Cybenko 1989, Hornik 1991, Leshno 1993).
- **Memory efficient:** Neural networks are able to approximate complex functions using a relatively small number of parameters.
- **Computationally efficient:** Information can be propagated efficiently through neural networks, so training them is (reasonably) fast.

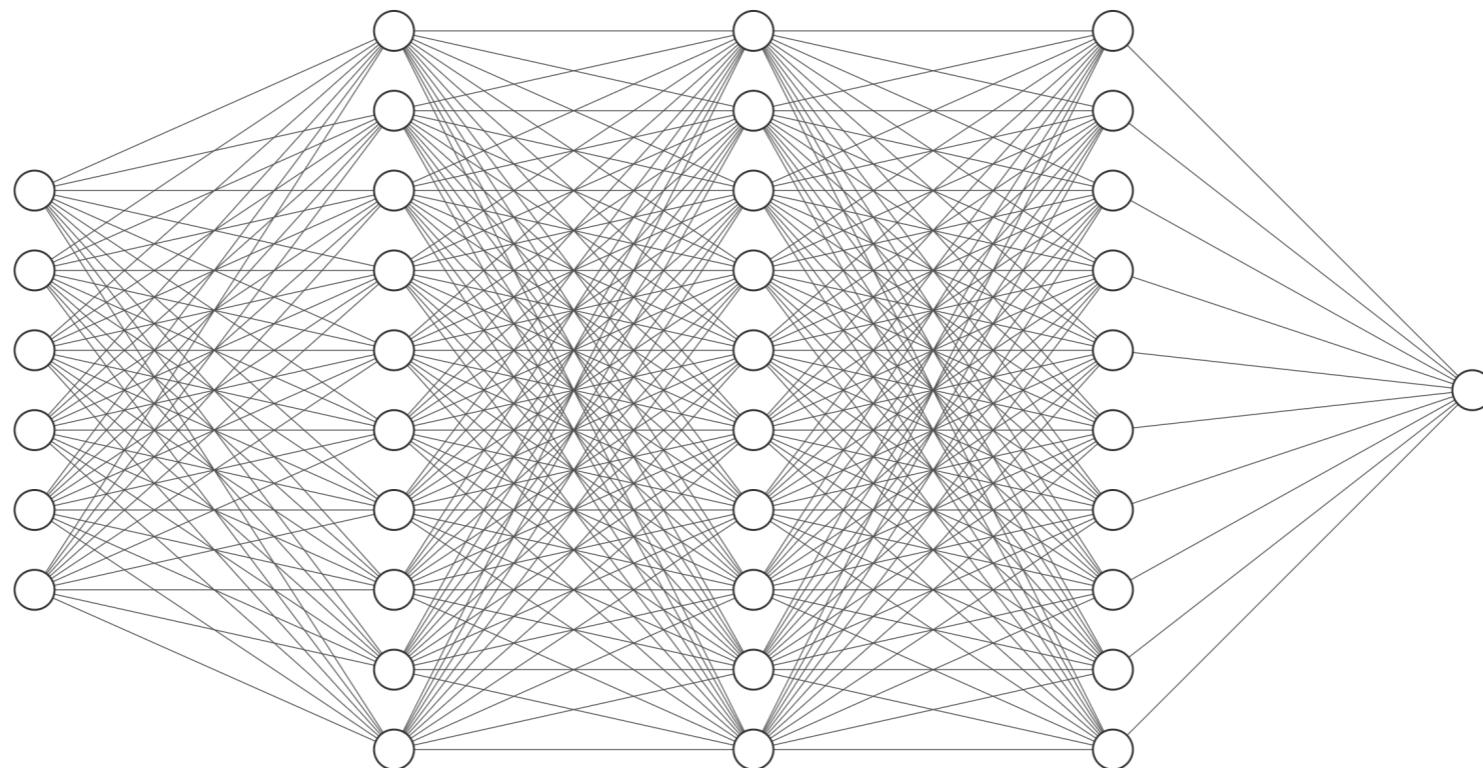
# Strengths of deep learning

- **Feature/Representational learning:** Neural networks iteratively apply non-linear transformations to the data to gradually build up increasingly abstract representations.



# Strengths of deep learning

- **Highly modular:** You can add nodes, layers, modules with different properties to the neural network. Software tools freely available online make this very easy.



# Outline

1. What is deep learning?
2. **What can DL do and not do**
3. Introduction to:
  1. Dense neural networks
  2. Convolutional neural networks
4. How to get started using neural networks

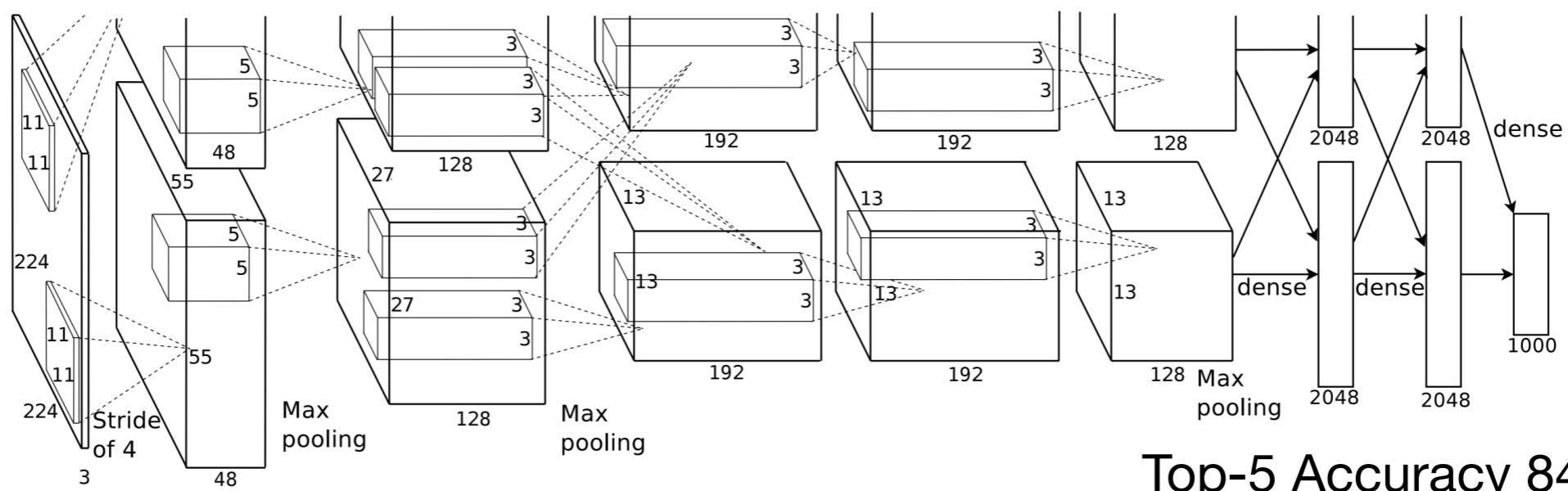
# Supervised learning

- **Task:** ImageNet Large Scale Visual Recognition Challenge (ILSVRC)
  - 1000 classes with roughly 1000 images per class
- **A more traditional approach:** Apply well-studied image feature extraction methods and train a classifier on top of this



# Supervised learning

- **Task:** ImageNet Large Scale Visual Recognition Challenge (ILSVRC)
  - 1000 classes with roughly 1000 images per class
- **DL Method:** AlexNet (Krizhevsky 2012): 8-layer convolutional neural network with 60 million parameters

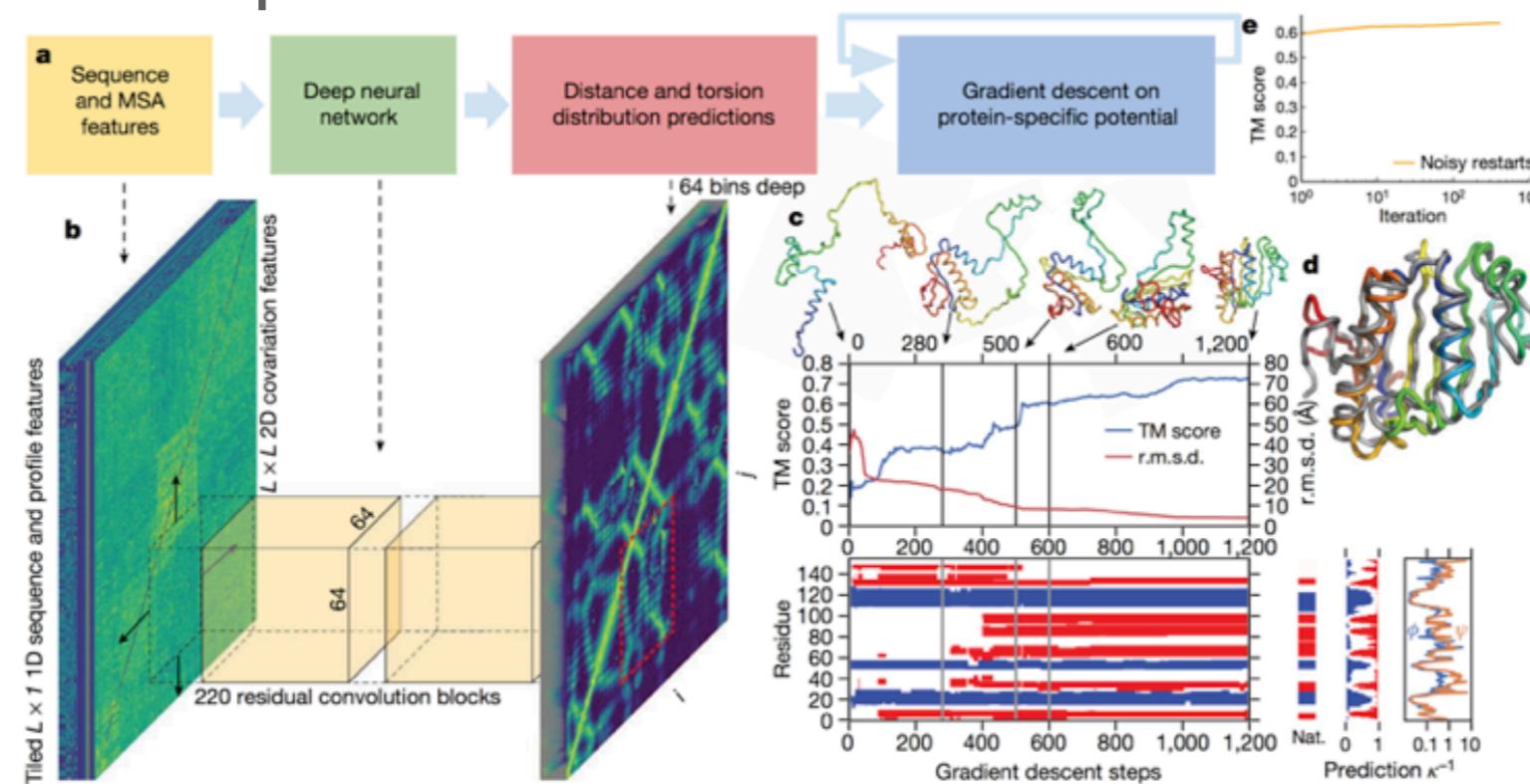


# Supervised Learning

- **Task:** Predict protein structure given only sequence information.
- **A more traditional approach:** Define a potential function and use stochastic sampling to minimize the potential. Given proteins with known structure, find the most similar protein and modify in accordance with the sequence differences.

# Supervised Learning

- **Task:** Predict protein structure given only sequence information.
- **DL Method:** AlphaFold: a CNN trained on PDB structures to predict distances between pairs of amino acids. Given the distance matrix, find the 3D protein structure that minimizes the potential.



# Unsupervised learning

- **Task:** Generate images of people
- **A more traditional approach:** Compose parametric models that learns the probability distribution of images

# Unsupervised learning

- **Task:** Generate images of people
- **DL Method:** Generative Adversarial Networks  
(Goodfellow 2014): Simultaneously trains two models from the perspective of a two-player game

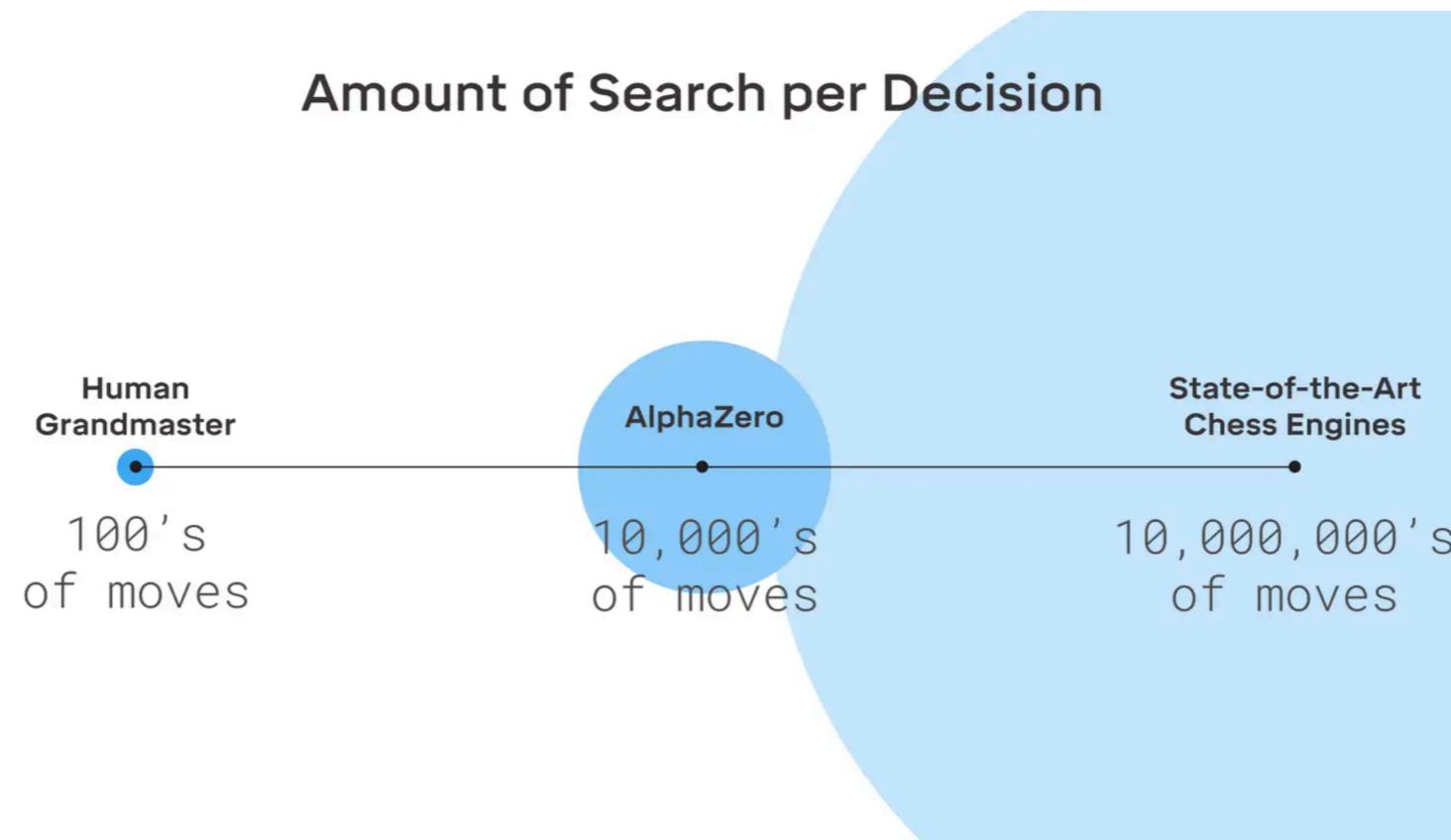


# Reinforcement learning

- **Task:** Teach a machine to play chess
- **A more traditional approach:** List out a subset of the possible move sequences and pick the best one. Rely on rules and heuristics handcrafted by experts.

# Reinforcement learning

- **Task:** Teach a machine to play chess
- **DL method:** AlphaZero (Silver 2018): A Monte Carlo tree search guided by a neural network. Learns the most promising moves by playing millions of games



# ChatGPT

this code is not working like i expect — how do i fix it?

```
resultWorkerErr := make(chan error)
defer close(resultWorkerErr)
go func() {
    defer cancel()
    resultWorkerErr <- b.resultWorker(ctx)
}()

err := b.worker(ctx)
cancel()
if err == nil {
    return <-resultWorkerErr
}
return multierror.Append(err, <-resultWorkerErr)
```

User

ChatGPT

It's difficult to say what's wrong with the code without more context. Can you provide more information about what the code is supposed to do and what isn't working as expected? Also, is this the entire code or just a part of it?

User

ChatGPT

just part of it — the error never surfaces. I think there is something wrong with the channel  
It's difficult to say without more information about what the code is supposed to do and what's happening when it's executed. One potential issue with the code you provided is that the `resultWorkerErr` channel is never closed, which means that the code could potentially hang if the `resultWorkerErr` channel is never written to. This could happen if `b.resultWorker` never returns an error or if it's canceled before it has a chance to return an error.

Which of the following techniques are used to train ChatGPT?

- A: Supervised learning
- B: Unsupervised learning
- C: Reinforcement learning

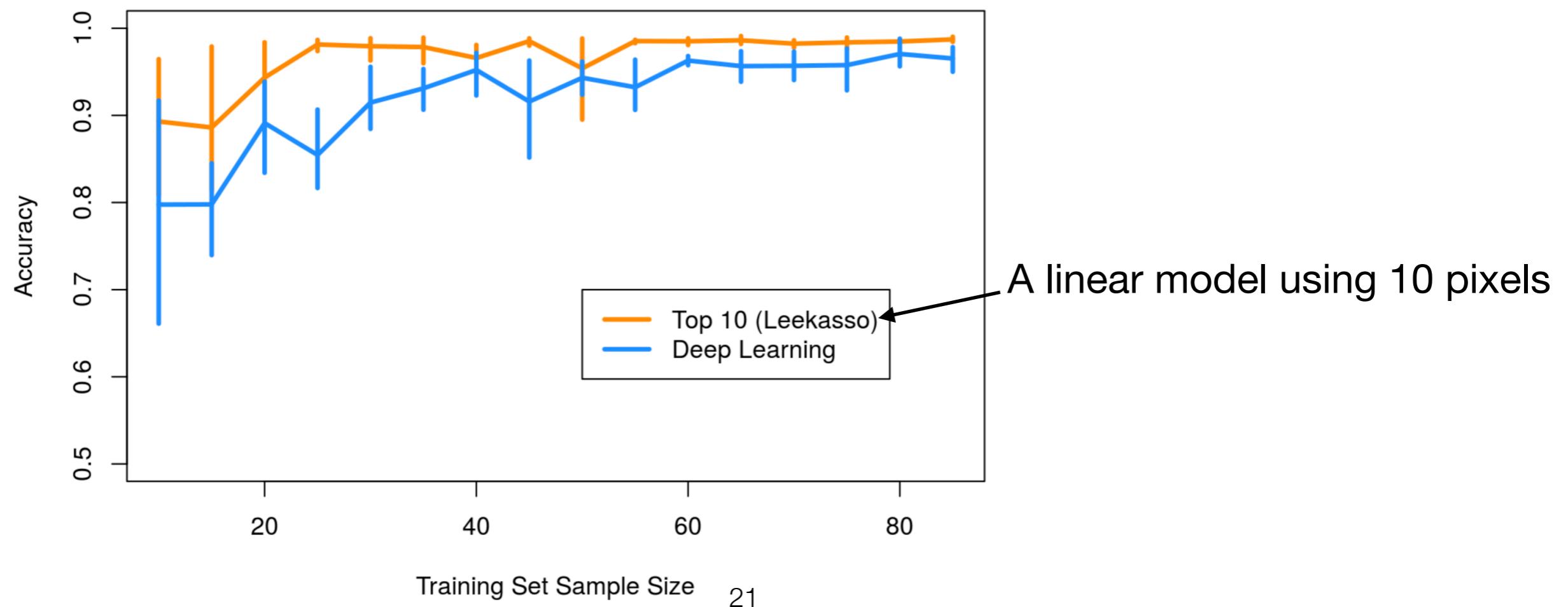
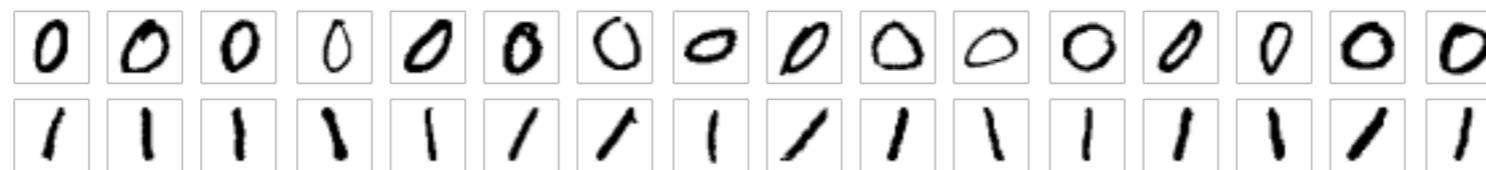


# Where deep learning struggles

“Don’t use deep learning - your data isn’t that big”

— Jeff Leek, [simplystats](#)

Example: Predict 0 vs 1 in the image



# Deep learning tends to struggle when...

- High-dimensional settings where the number of observations  $n$  is "small" compared to the number of covariates  $p$
- Signal-to-noise ratio is low
- You need an interpretable model



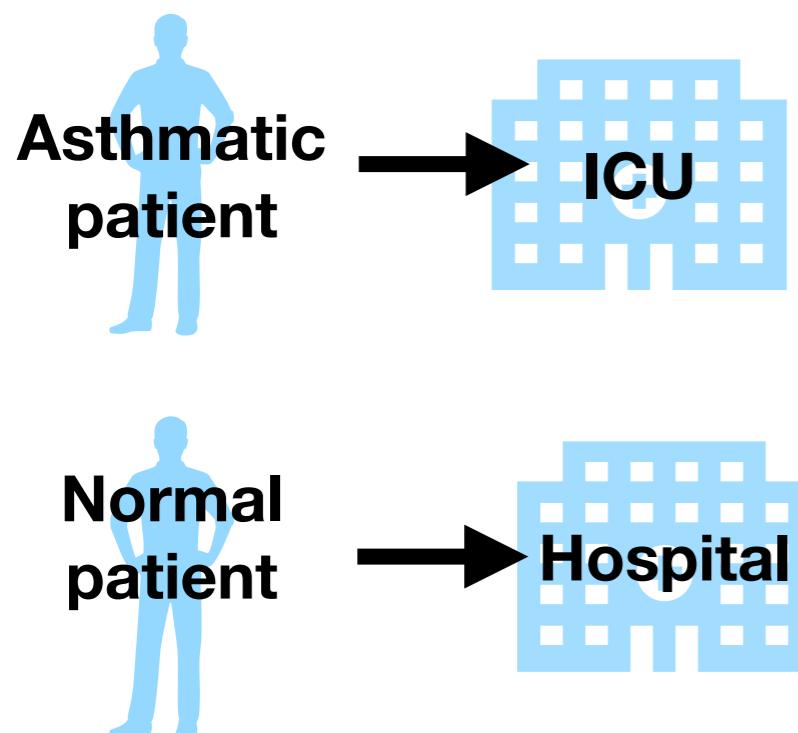
*Bias/Variance tradeoff!*

In these cases, try using a simpler methods like linear/logistic/cox regression or off-the-shelf machine learning algorithms like random forests.

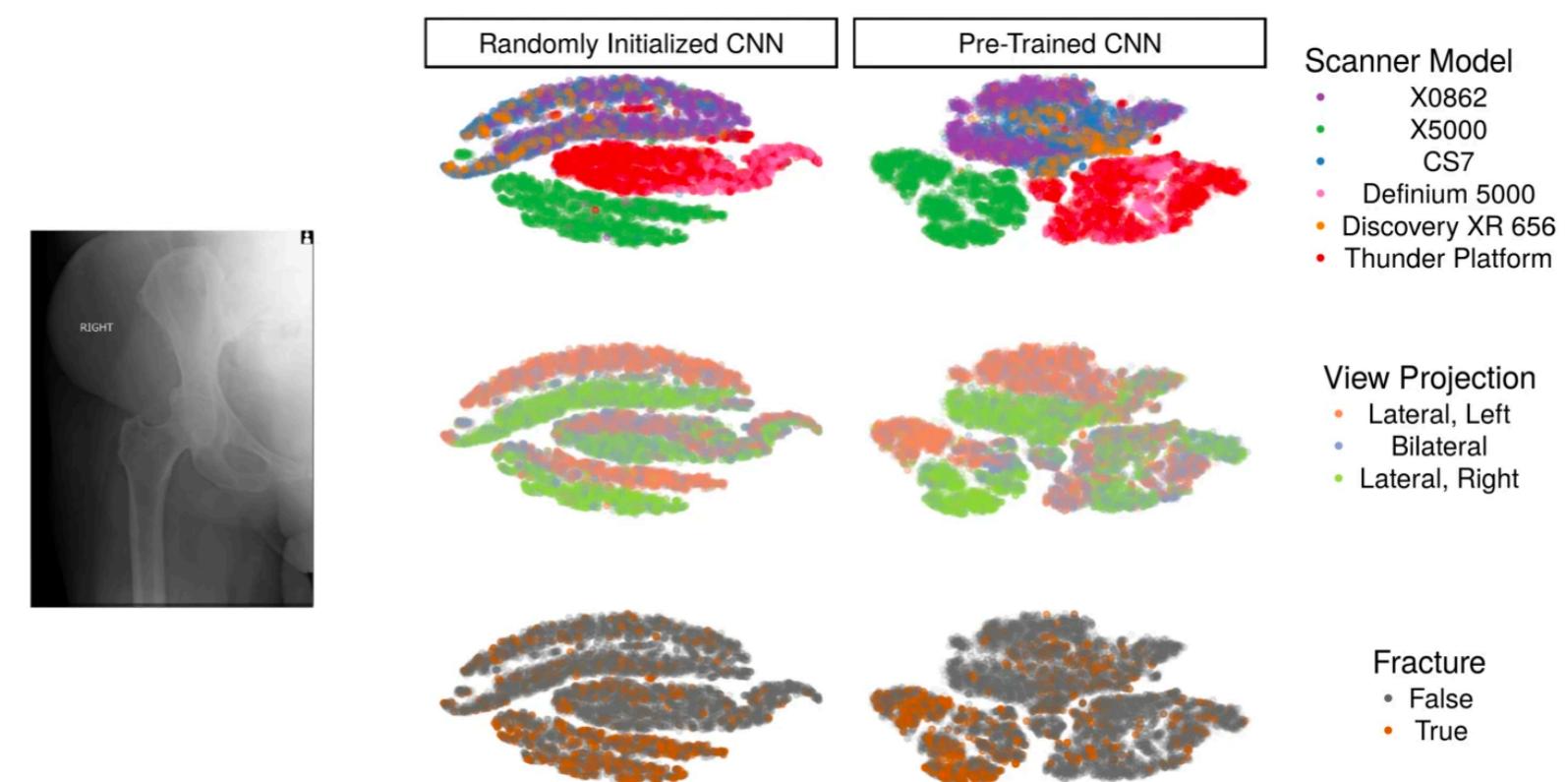
# Deep learning tends to struggle when...

- Deep learning will *not* correct biases in the data! It is *very* sensitive to any confounders in your data

Caruana 2015: DL predicted that asthmatics are less likely to die from pneumonia



Badgeley 2019: DL made predictions by detecting the scanner model from the radiograph



# Outline

1. What is deep learning?
2. What can DL do and not do
3. Introduction to:
  1. **Dense neural networks**
  2. Convolutional neural networks
4. How to get started using neural networks

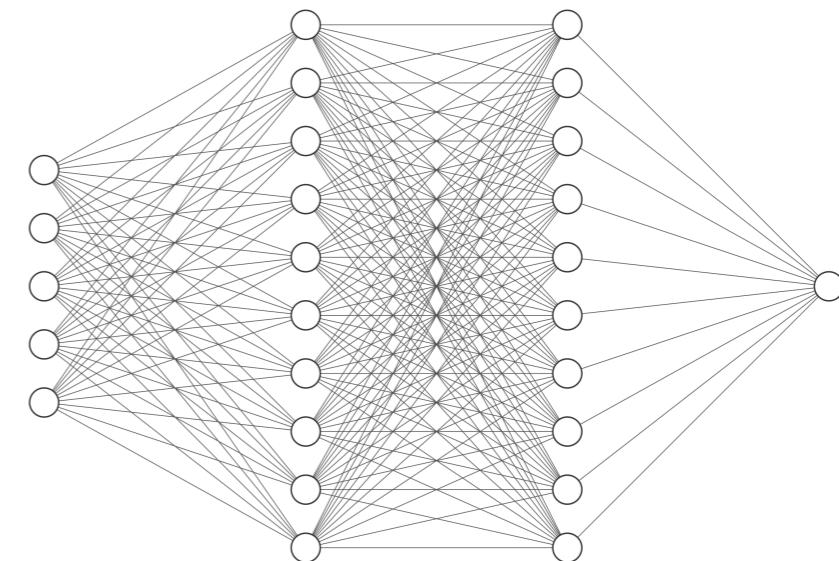
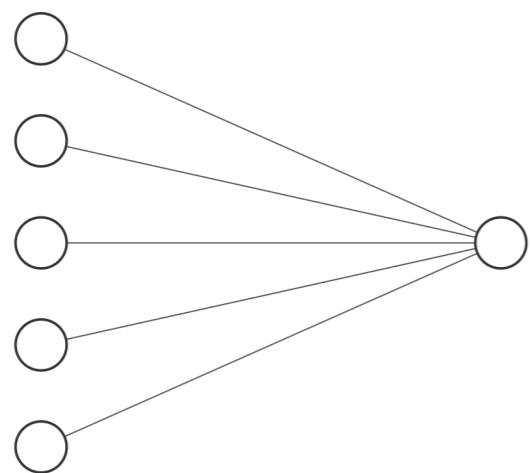
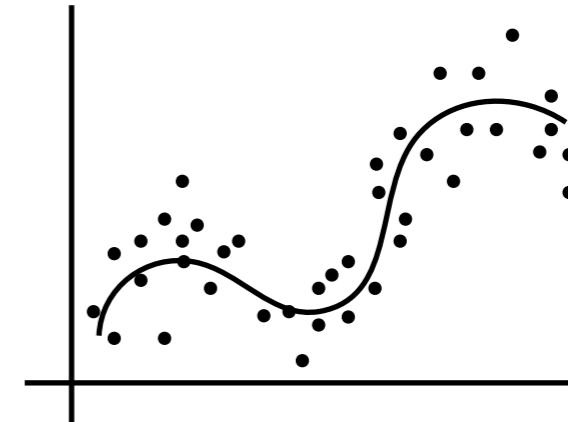
# Neural networks

- Neural networks span a wide range of models:

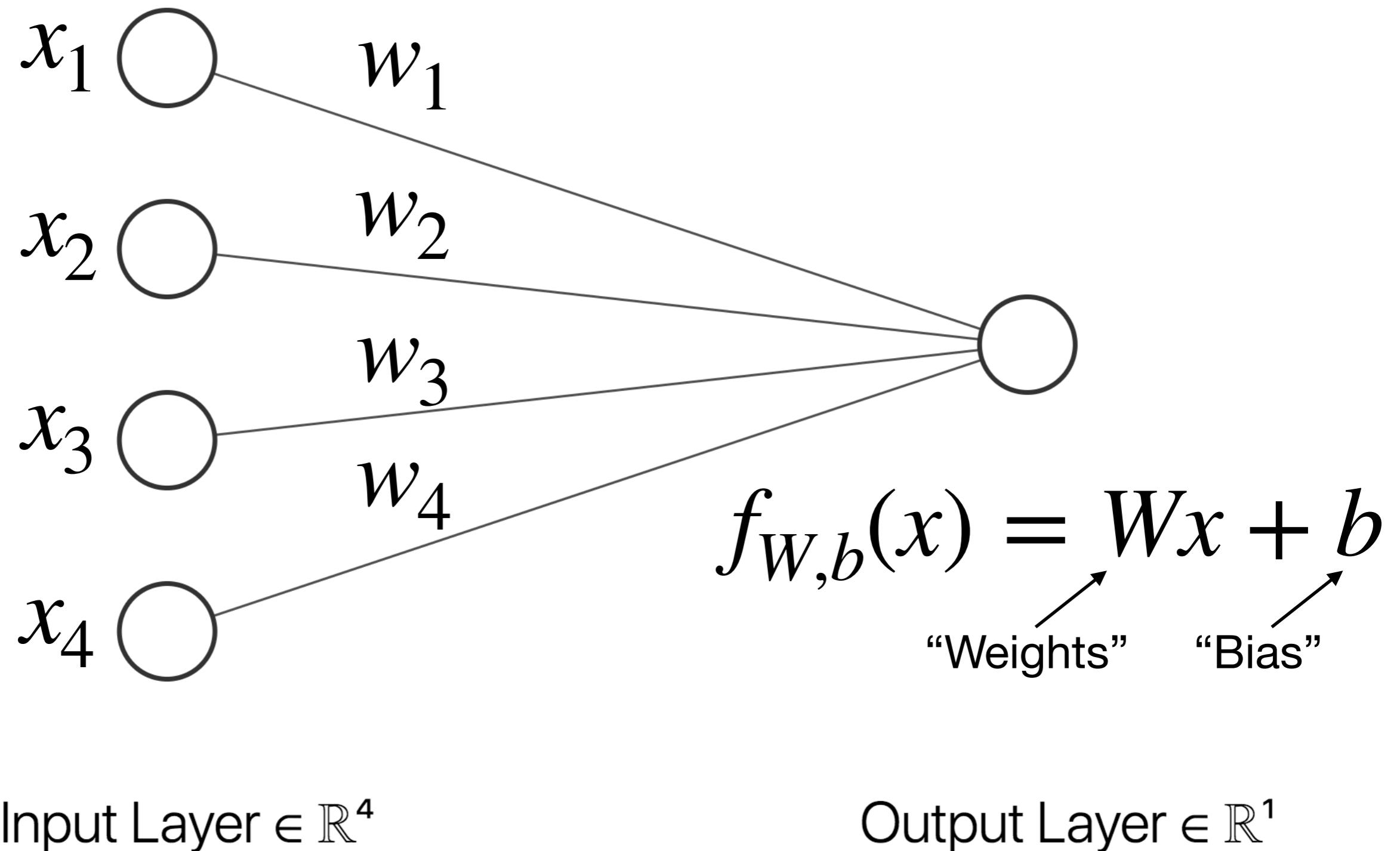
Linear model



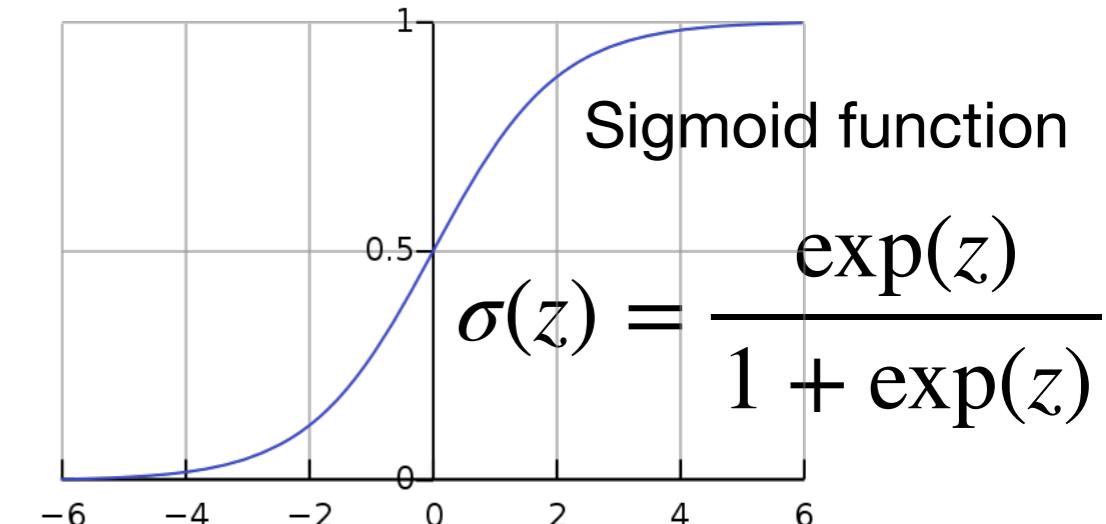
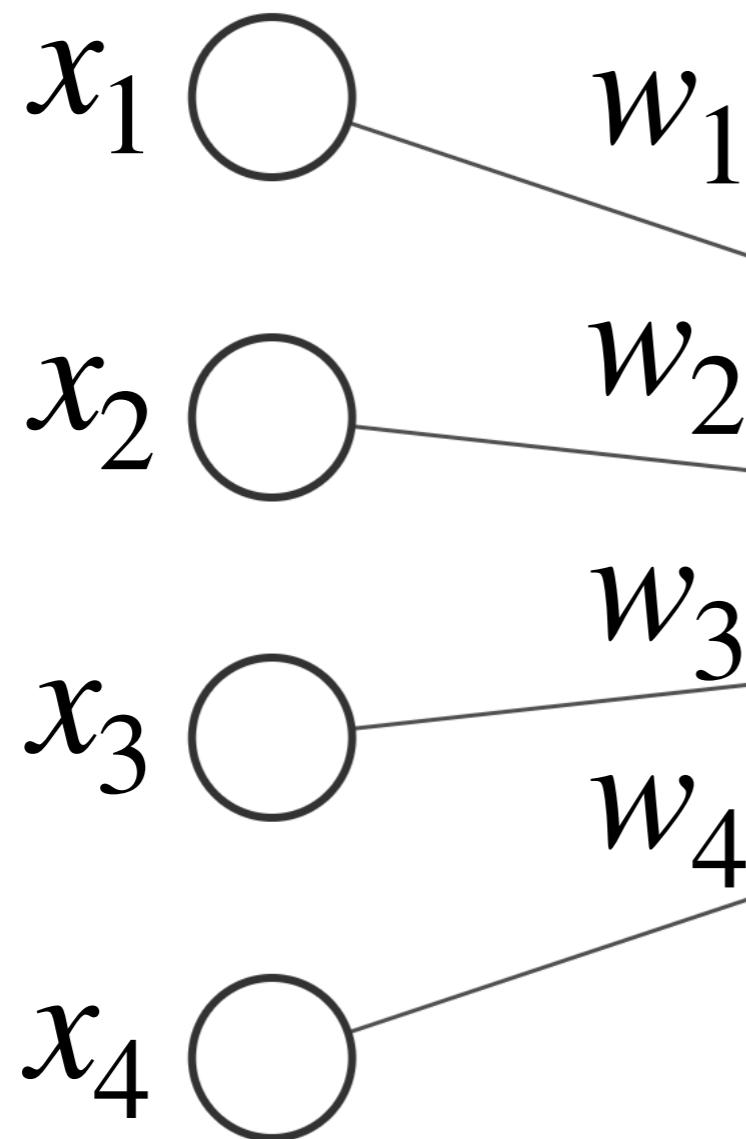
Nonlinear model with complex interactions



# Linear regression as a neural network



# Logistic regression as a neural network



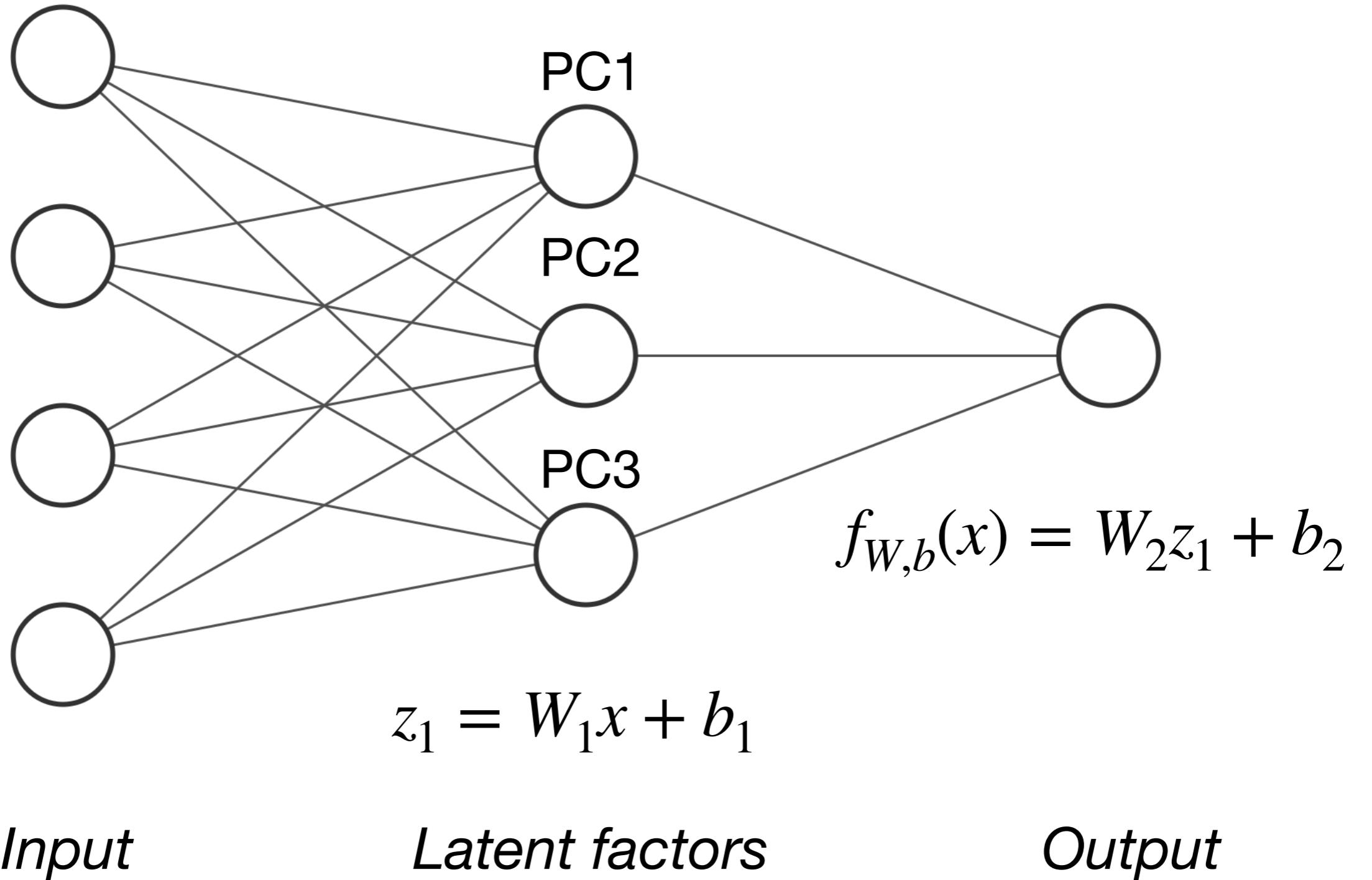
$$f_{W,b}(x) = \sigma(Wx + b)$$

“Activation function”

Input Layer  $\in \mathbb{R}^4$

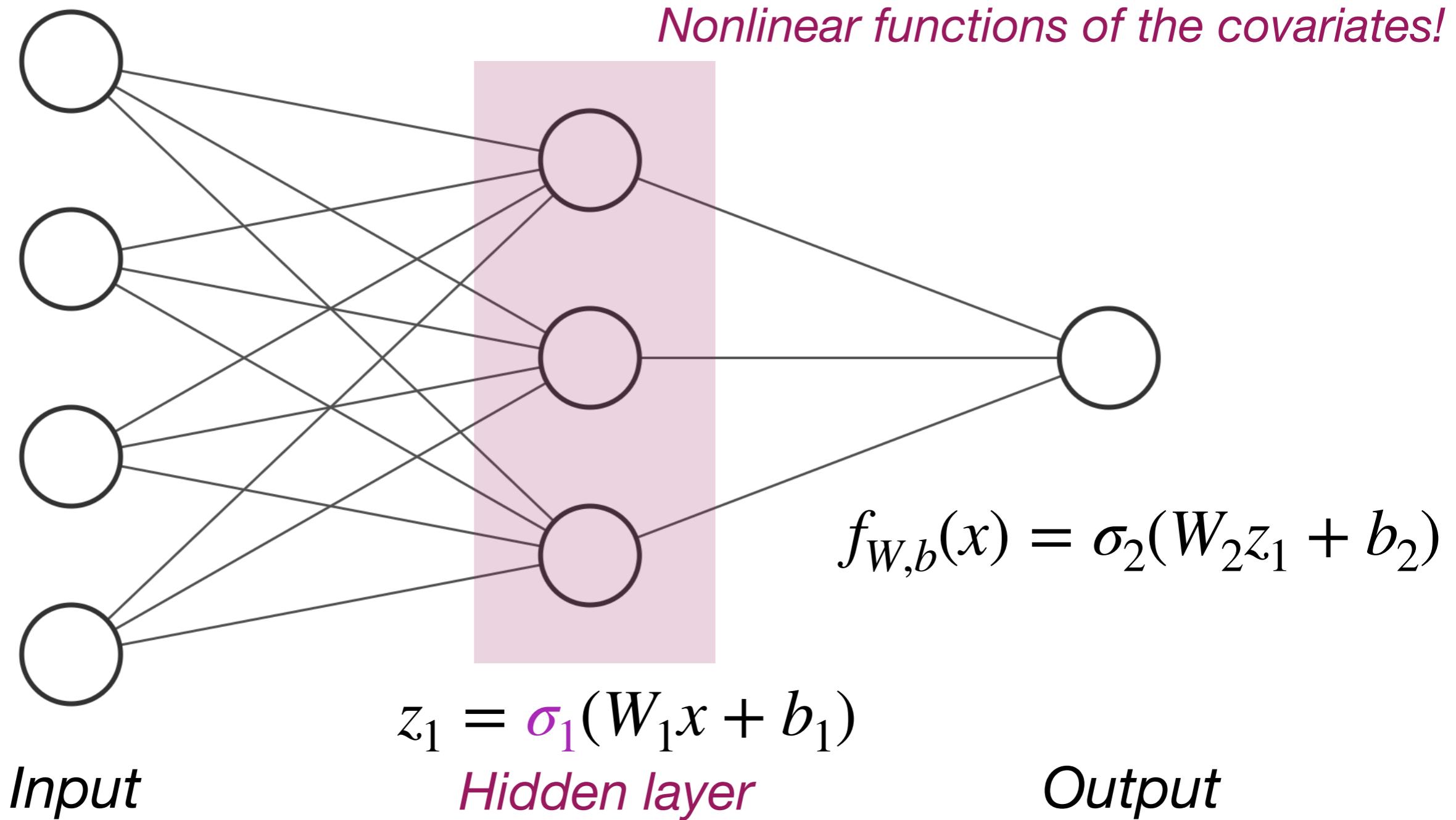
Output Layer  $\in \mathbb{R}^1$

# Principal component regression as a neural network



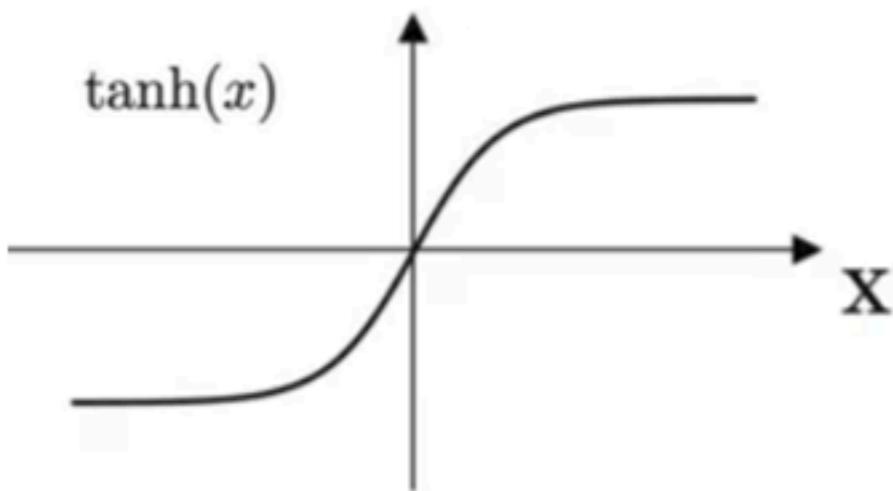
# Dense neural networks

- Also known as "Multilayer Perceptrons"

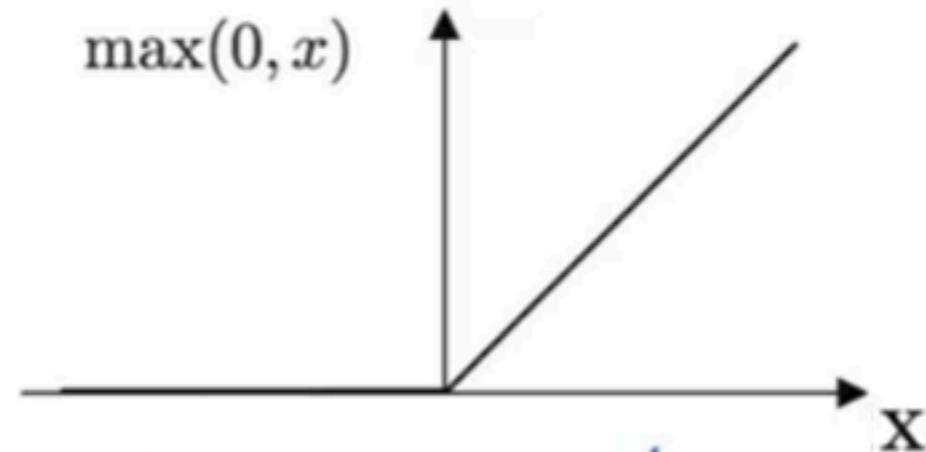


# Activation functions $\sigma$

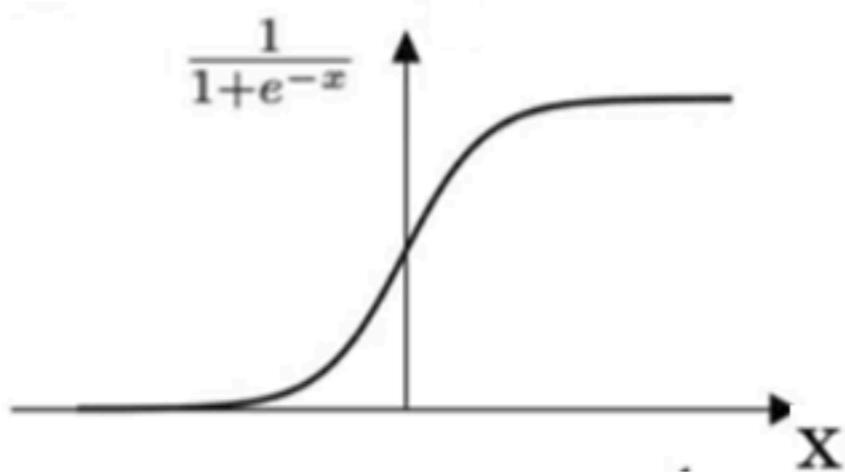
**Hyper Tangent Function**



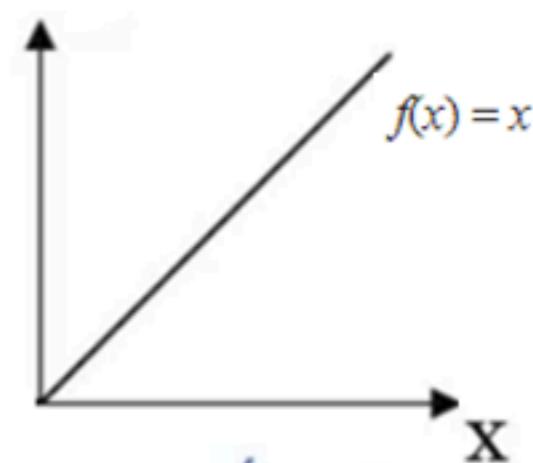
**ReLU Function**



**Sigmoid Function**

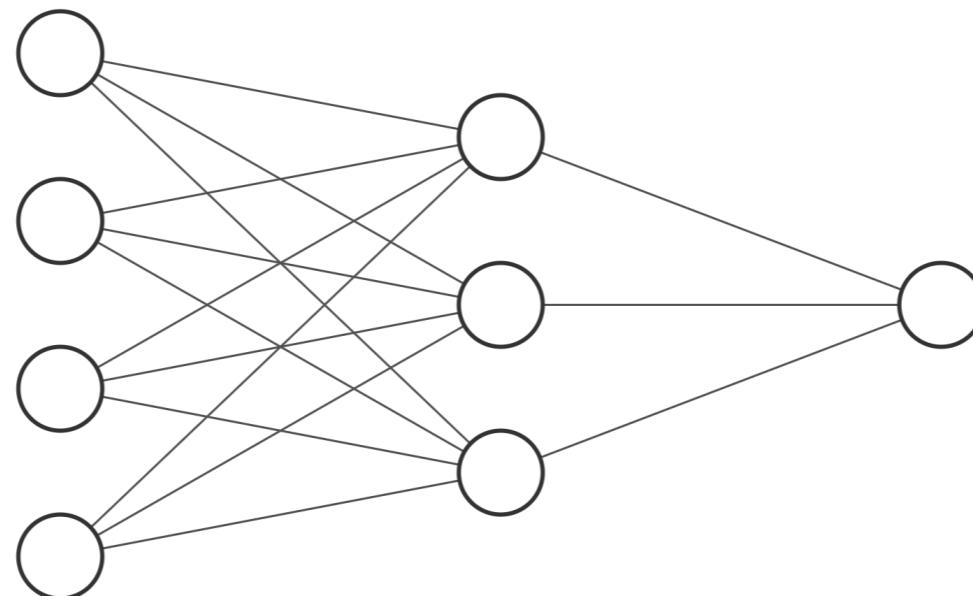


**Identity Function**



# Quick quiz

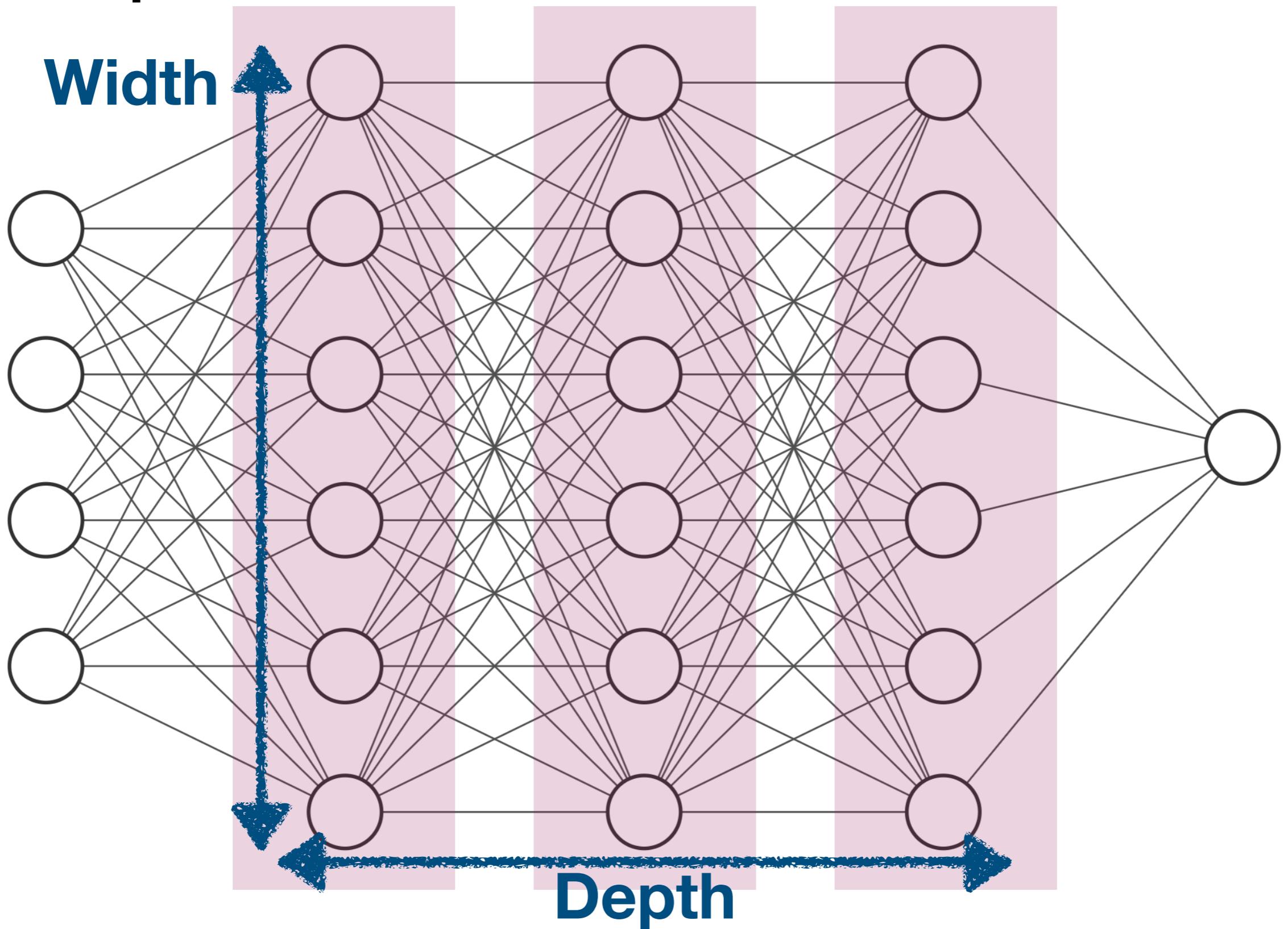
- How many parameters do we need to estimate in the following neural network?



- A: 8 parameters
- B: 15 parameters
- C: 19 parameters

*12 weights and 3 bias parameters in the first layer  
3 weights and 1 bias parameter in the second layer*

# Deep neural networks



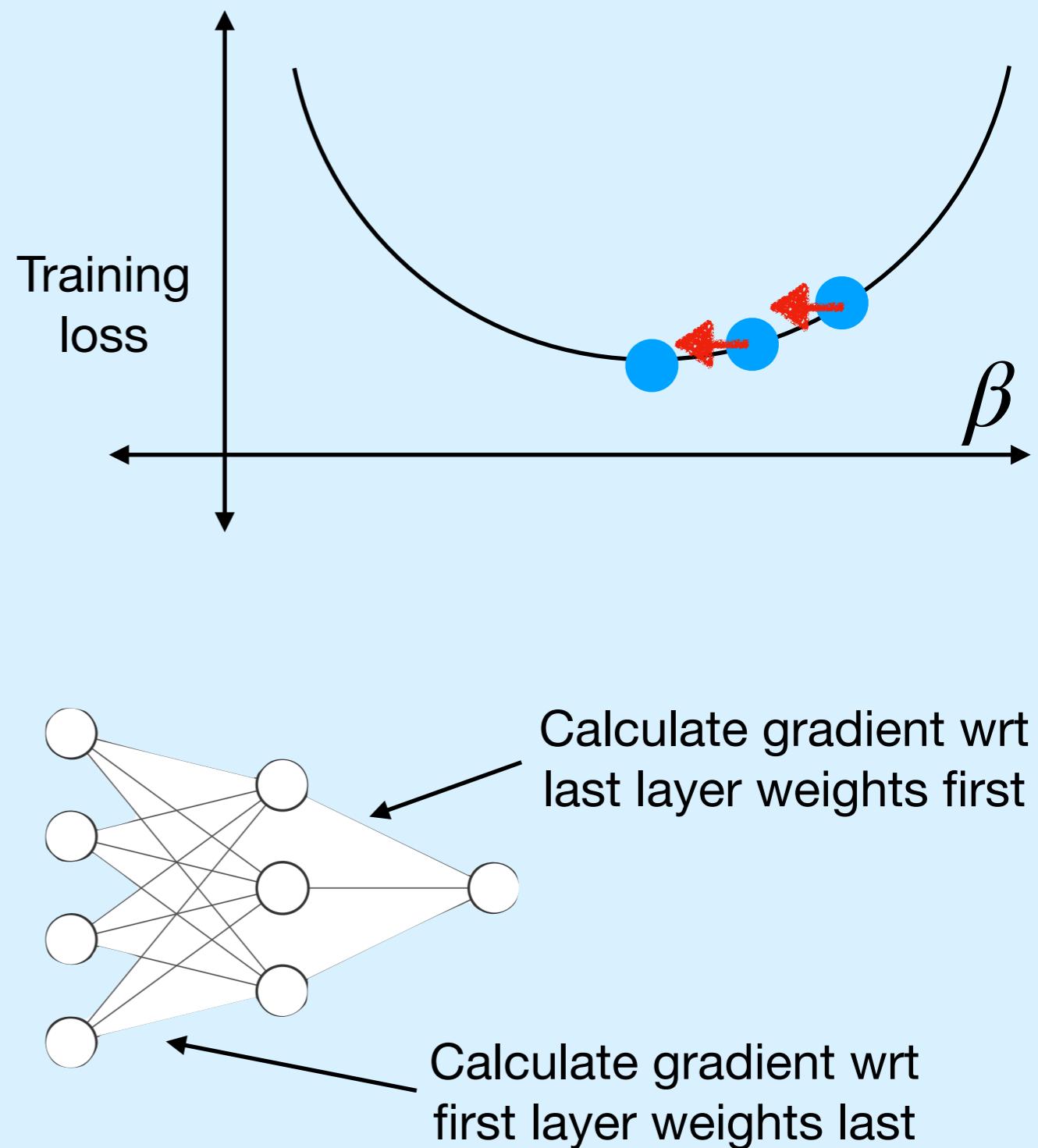
# Training a neural network

1. Specify a fixed network architecture:
  - Network depth (number of hidden layers)
  - Width of each hidden layer
  - Activation function at each node
2. Find the network parameters (weights and biases) that minimize the objective function, e.g. the empirical mean squared error.
3. Evaluate the model on a held-out data using sample-splitting or cross-validation. Go back to step 1 to evaluate other network architectures. Iterate until you have a good set of hyperparameters.

# Training a neural network: Step 2

2. Find the network parameters (weights and biases) that minimize the objective function, e.g. the empirical mean squared error.

- Use gradient-based minimization, e.g. batch gradient descent, stochastic gradient descent, Adam, etc.
- To calculate the gradients efficiently, apply **back-propagation**.



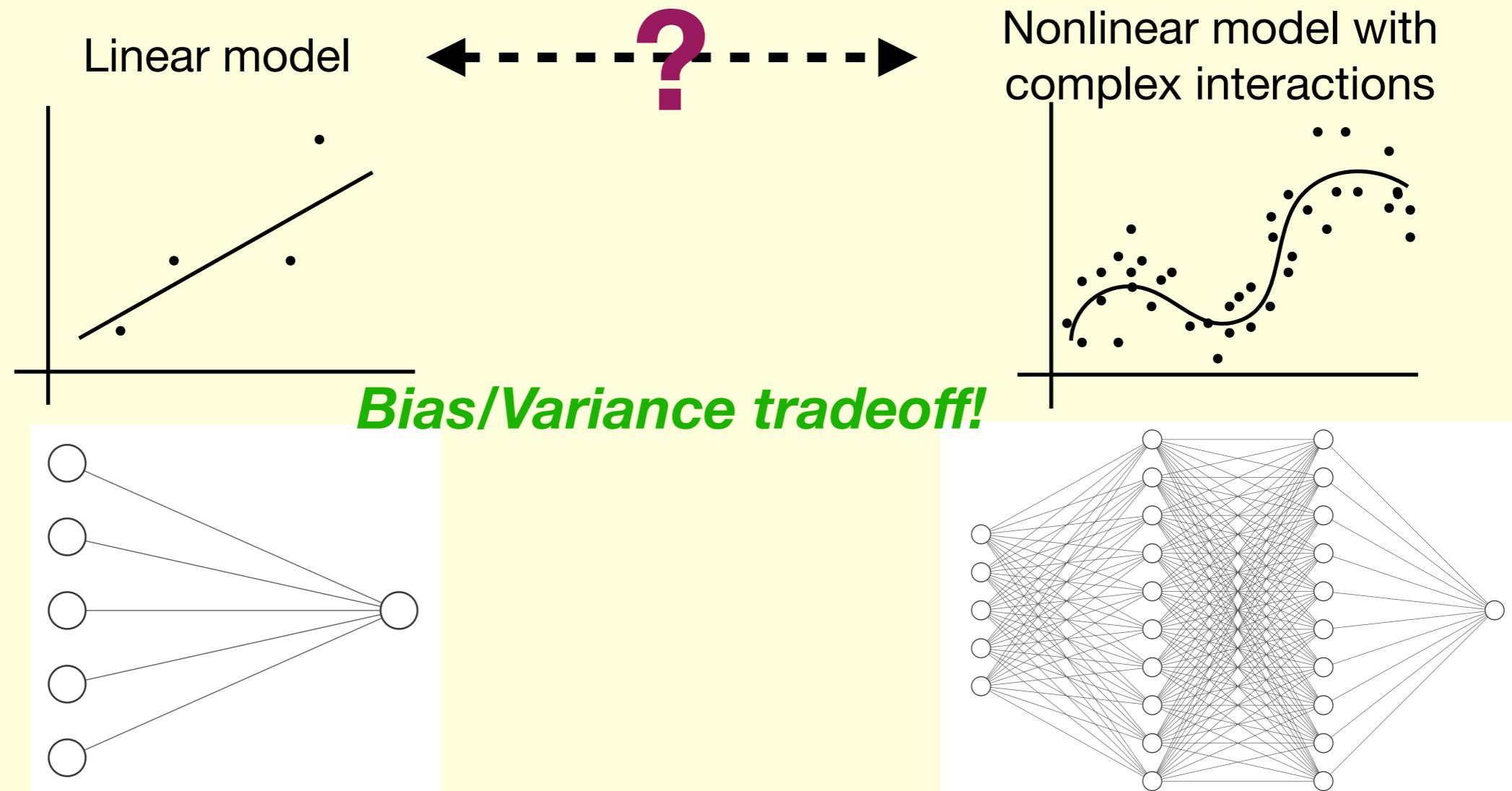
# Training a neural network

1. Specify a fixed network architecture:
  - Network depth (number of hidden layers)
  - Width of each hidden layer
  - Activation function at each node
2. Find the network parameters (weights and biases) that minimize the objective function, e.g. the empirical mean squared error.
3. Evaluate the model on a held-out data using sample-splitting or cross-validation. Go back to step 1 to evaluate other network architectures. Iterate until you have a good set of hyperparameters.

***This is the most difficult and time-consuming step!***

# Training a neural network: Step 3

3. Evaluate the model on a held-out data using sample-splitting or cross-validation. Go back to step 1 to evaluate other network architectures. Iterate until you have a good set of hyperparameters.

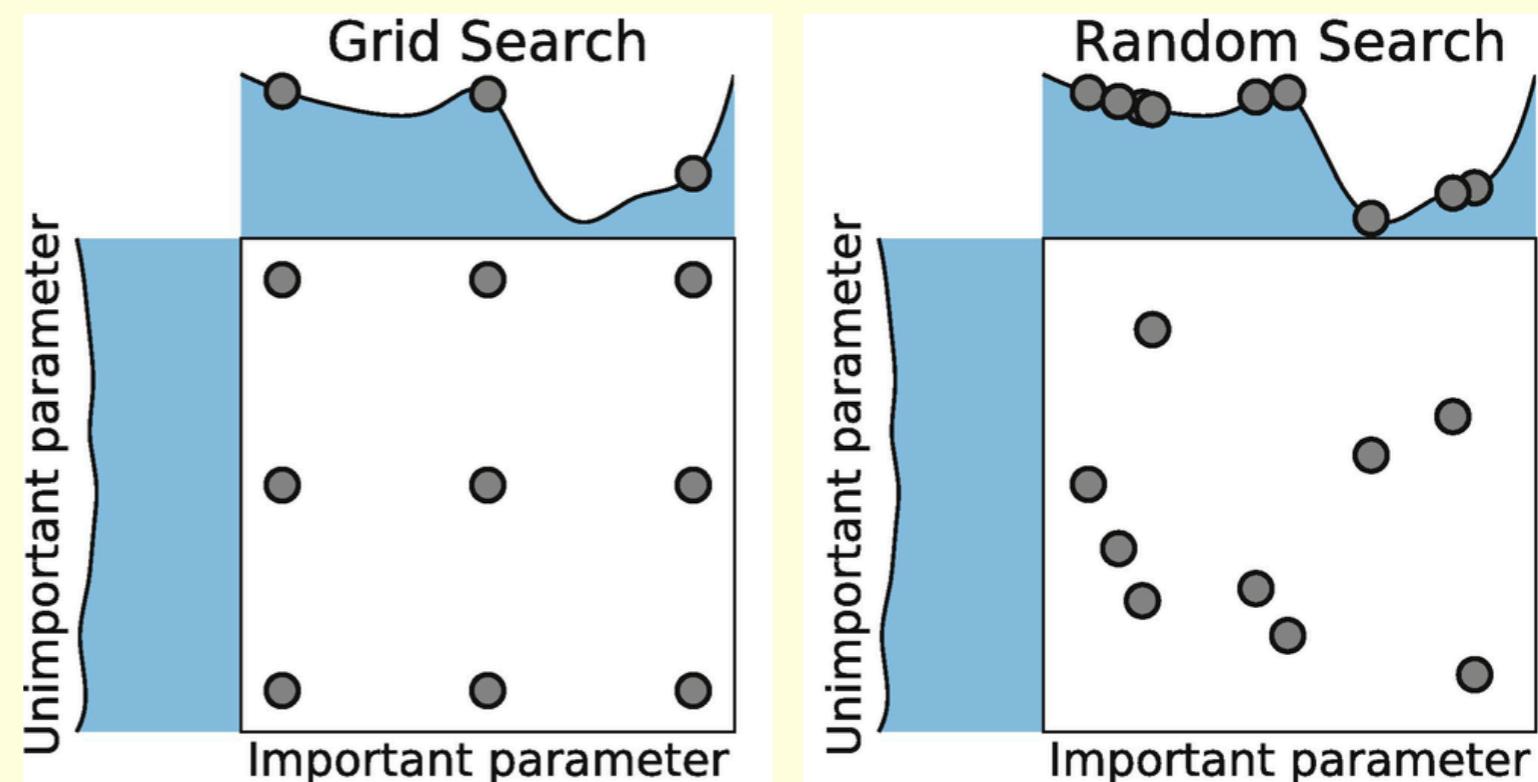


# Training a neural network: Step 3

3. Evaluate the model on a held-out data using sample-splitting or cross-validation. Go back to step 1 to evaluate other network architectures. Iterate until you have a good set of hyperparameters.

Hyperparameters to think about:

- Network depth (number of hidden layers)
- Width of each hidden layer
- Activation function at each node



# Regularization

- Since neural networks have a tendency to overfit to the training data, researchers have developed many new ways to regularize networks:
  - **Penalization:** Add penalties on the network weights to the training objective, e.g.  $\|W\|_2^2$  and  $\|W\|_1$
  - **Dropout** (Srivastava 2014): Randomly drop out nodes from the network at each training iteration
  - **Stochastic gradient descent:** Each training iteration only uses a random subset of observations. This both speeds up training and improves generalization.
  - ...

# Summary: Dense neural network

- Neural networks are expressive estimators that generalize many well-known statistical methods
- The size of the neural network determines the bias/variance tradeoff and should be carefully tuned
- Neural networks are highly modular and can be combined with other regression tools, e.g. regularization

# Outline

1. What is deep learning?
2. What can DL do and not do
3. Introduction to:
  1. Dense neural networks
  - 2. Convolutional neural networks**
4. How to get started using neural networks

# Let's talk about image data

- Neural networks can be drastically improved by taking advantage of prior knowledge.
- **Image data has important spatial structure.** When performing visual tasks like object detection, the outcome...

1. **Is invariant to translational shifts in the image:** the absolute location of the object in the image does not matter.

2. Is a complex function of **local image regions**

## Translational Invariance

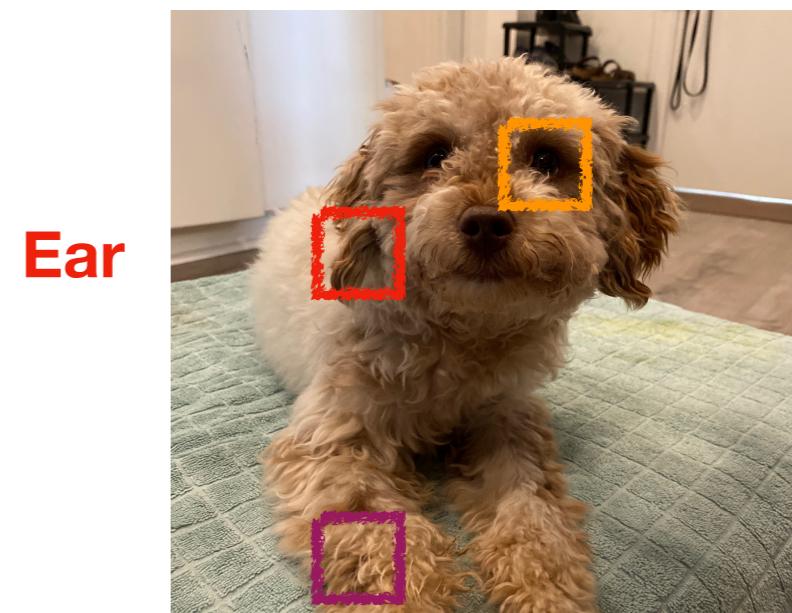


Dog



Still a dog

## Locality



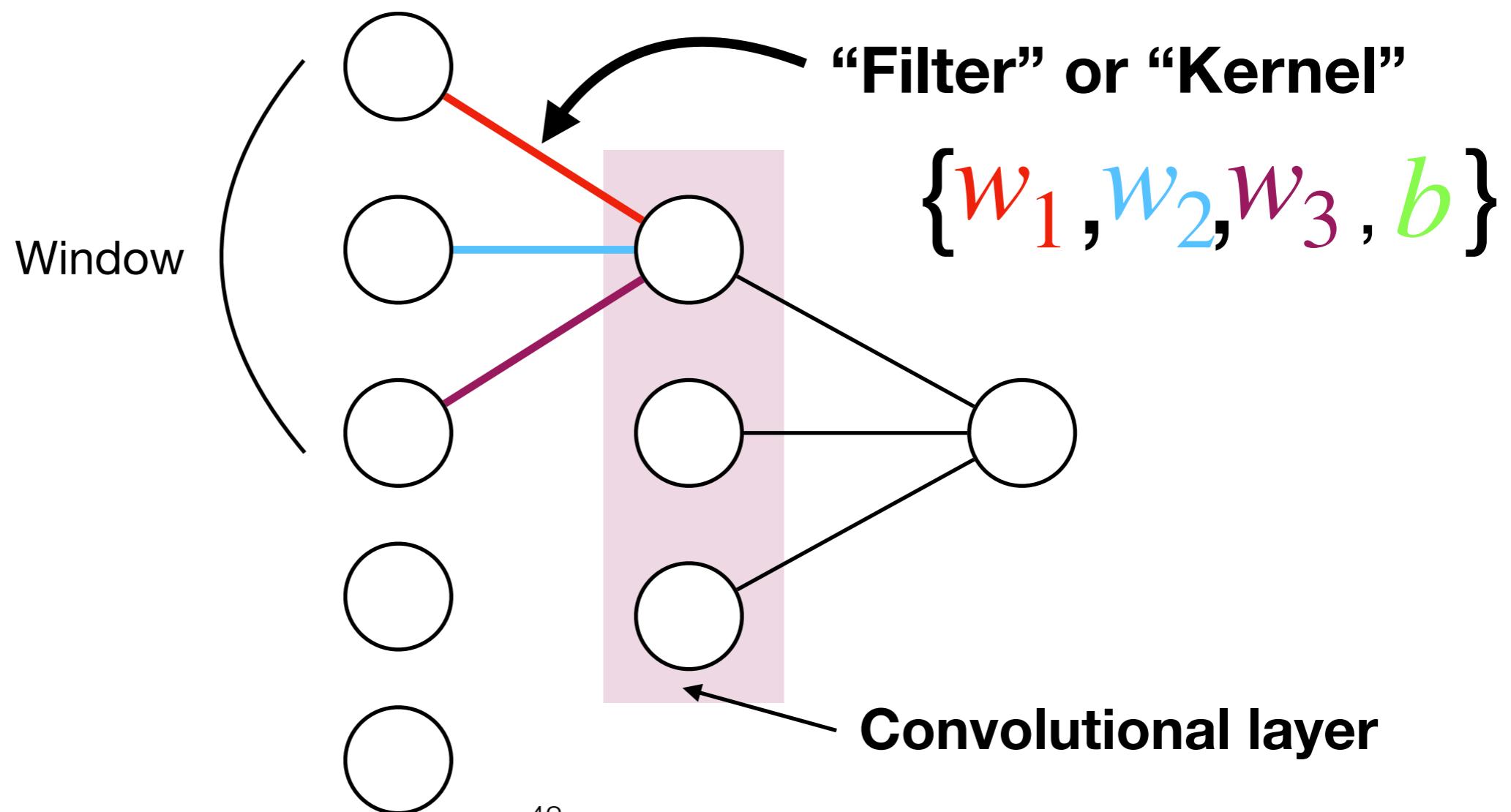
Ear

Eye

Fuzzy

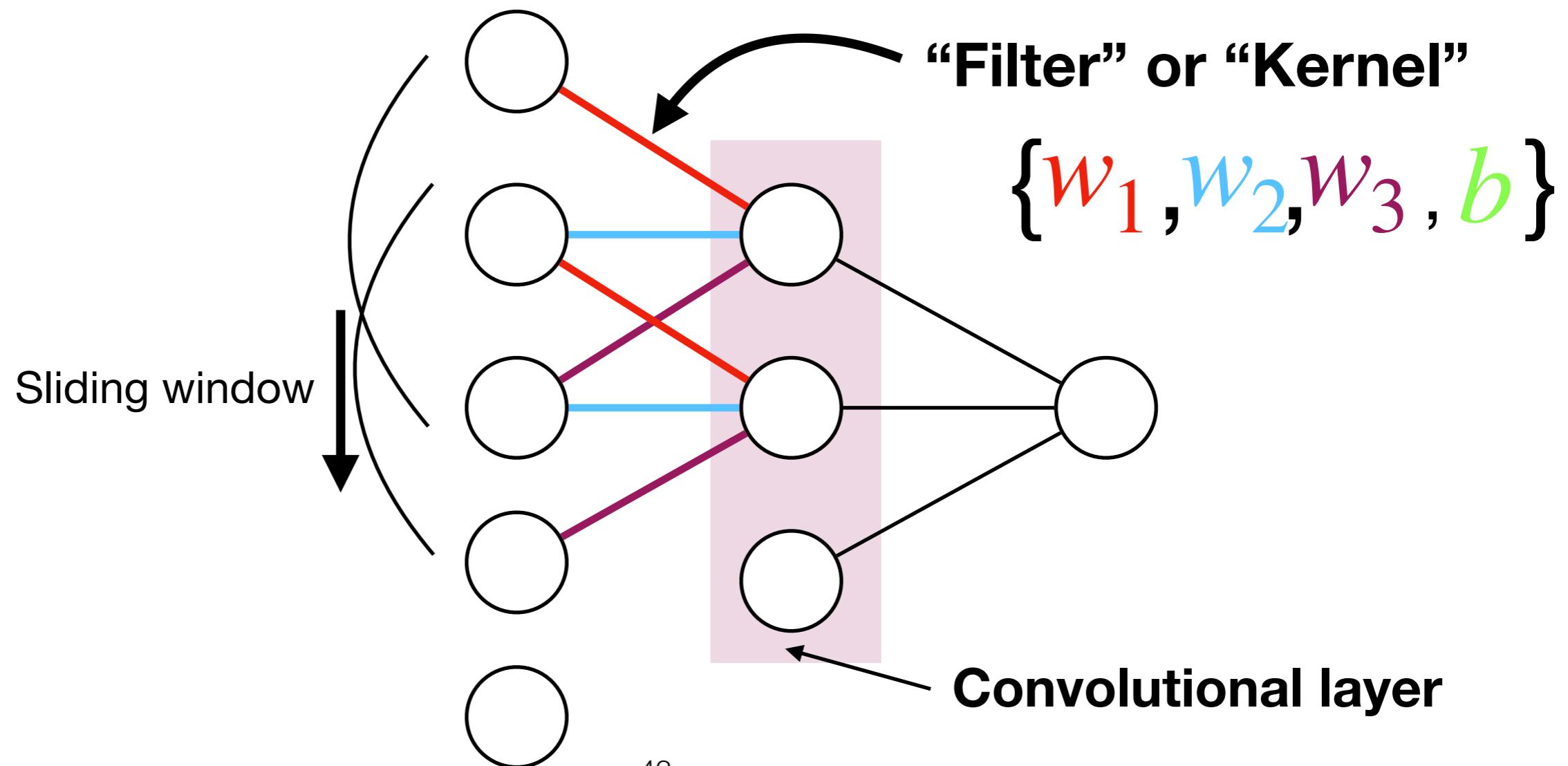
# Convolutional neural networks (CNNs)

- Convolutional layers constrain the network weights based on the ideas of *translational invariance* and *locality*.
- 1D example: We use the same set of weights



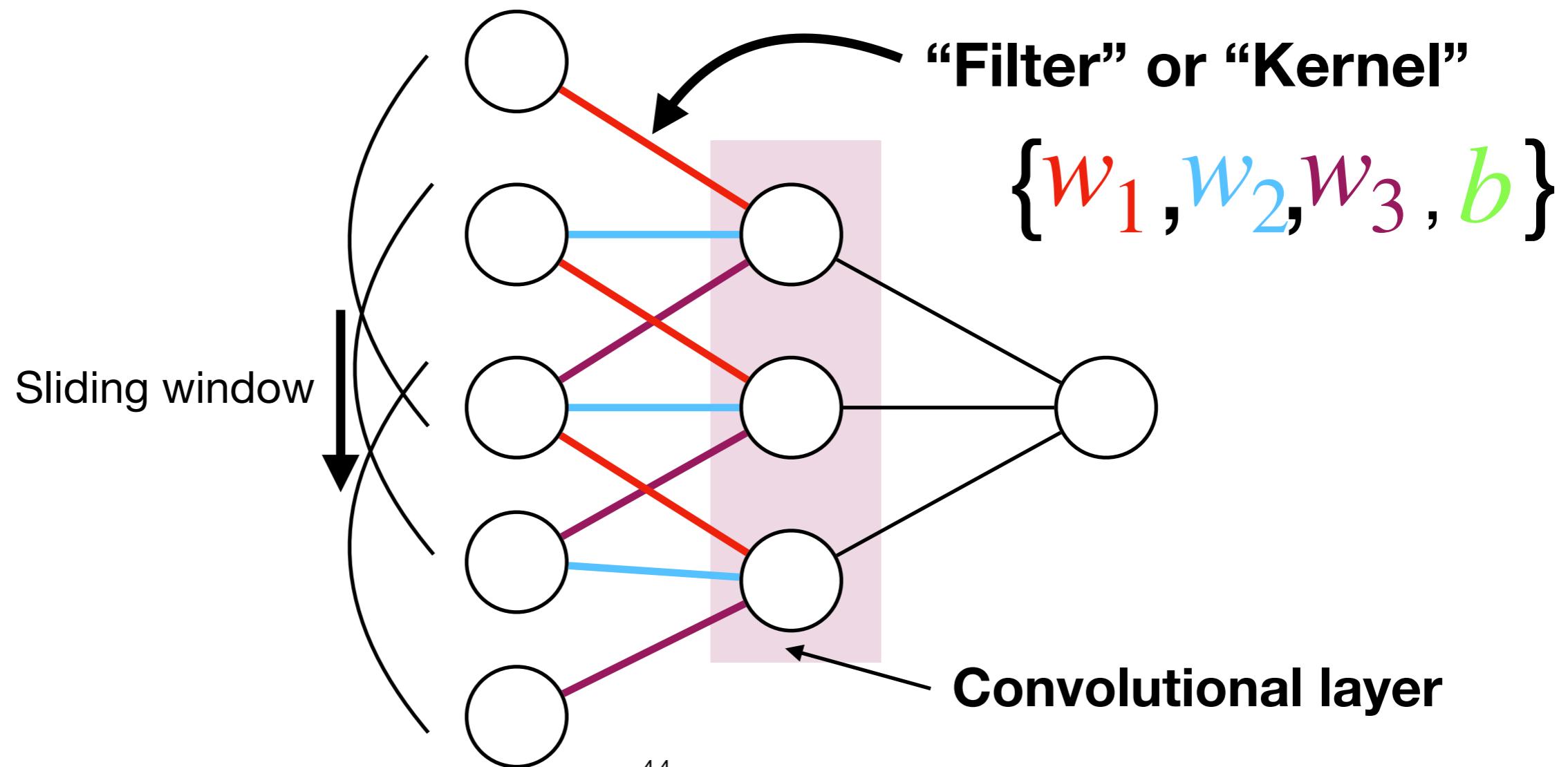
# Convolutional neural networks (CNNs)

- Convolutional layers constrain the network weights based on the ideas of *translational invariance* and *locality*.
- 1D example: We use the same set of weights



# Convolutional neural networks (CNNs)

- Convolutional layers constrain the network weights based on the ideas of *translational invariance* and *locality*.
- 1D example: We use the same set of weights



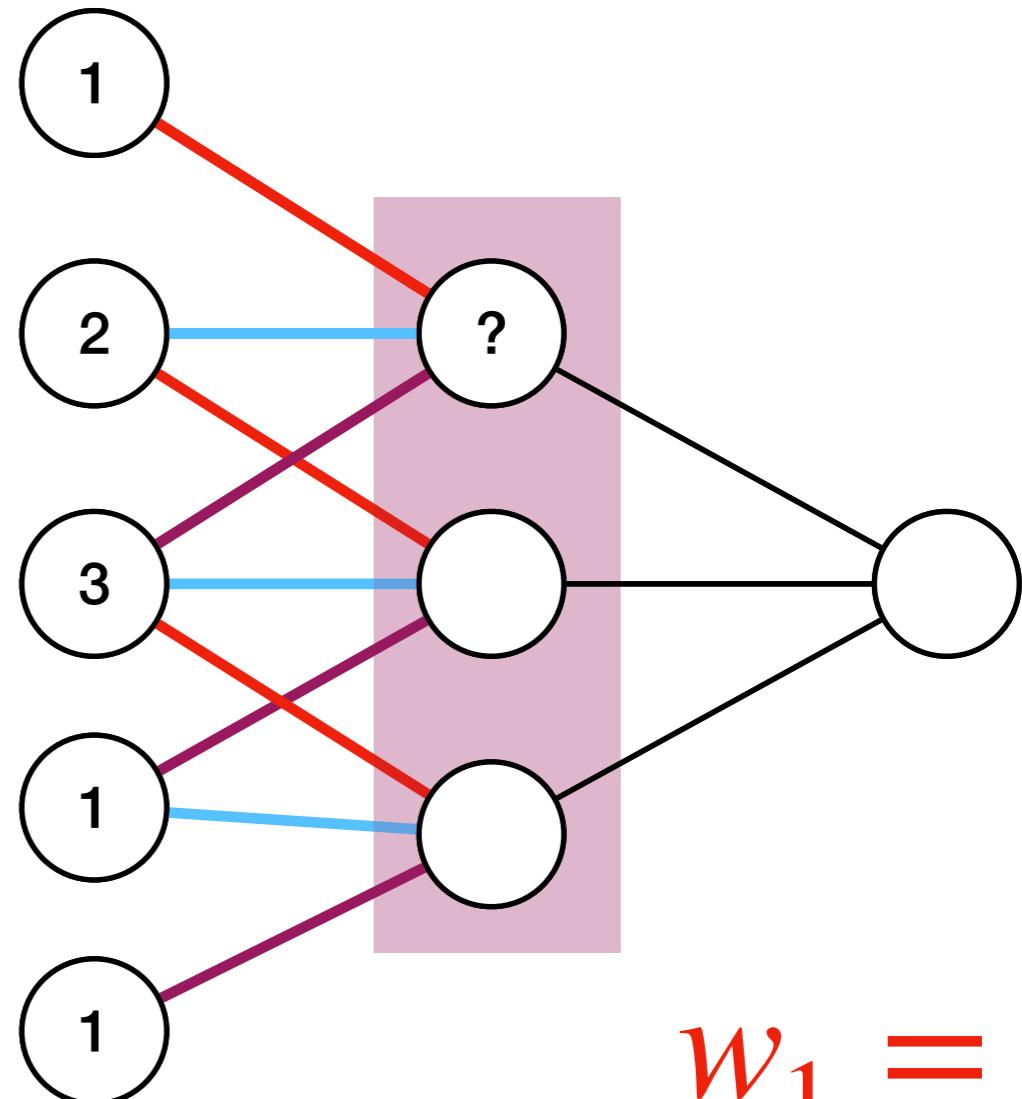
# Quick Quiz

Q: What is the value in the 1st hidden node?

A:  $1 * 1 + 2 * 2 + 3 * 1 + 0 = 8$

B:  $2 * 1 + 3 * 2 + 1 * 1 + 0 = 9$

C:  $1 * 1 + 2 * 1 + 3 * 1 + 0 = 6$



$w_1 = 1$

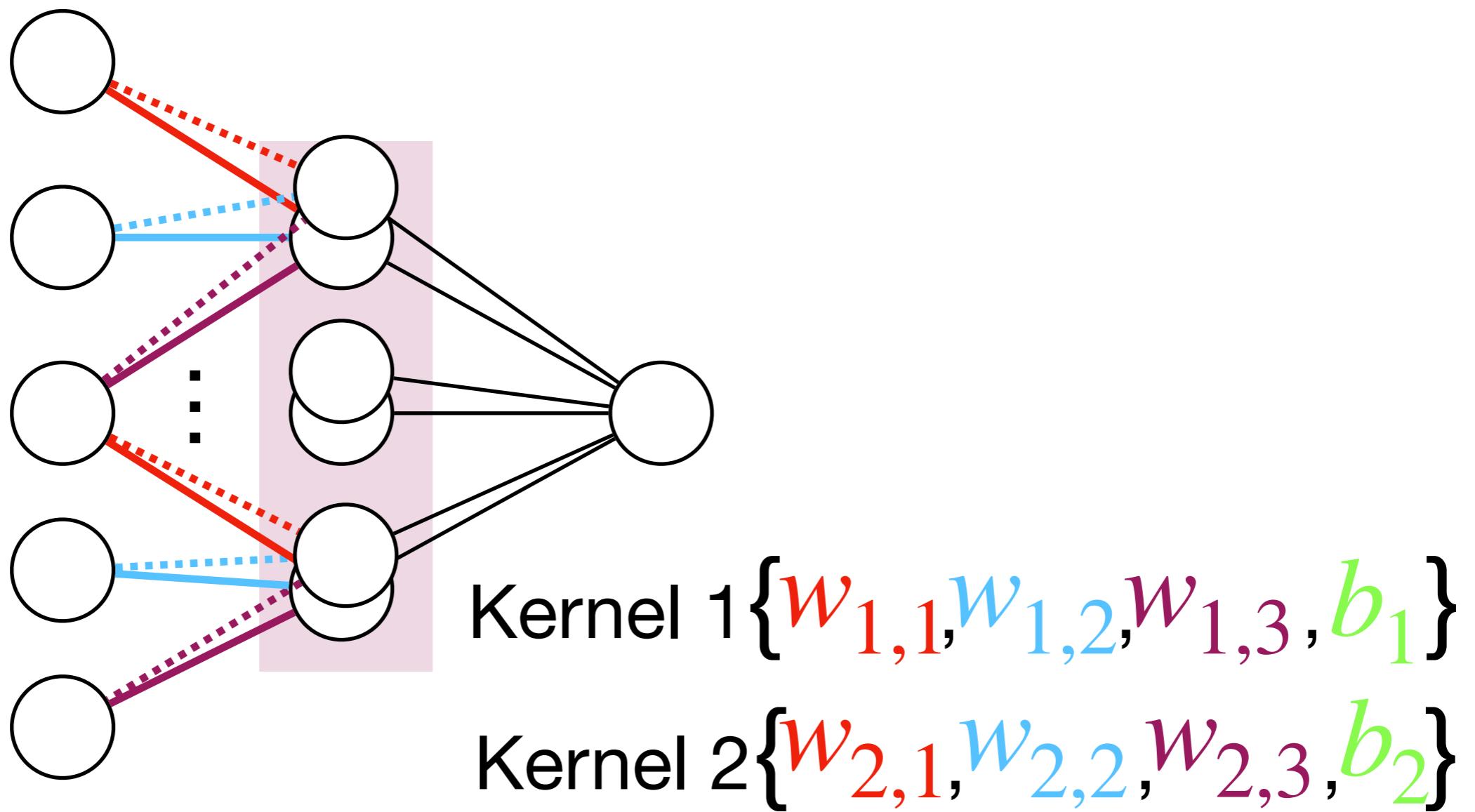
$w_2 = 2$

$w_3 = 1$

$b = 0$

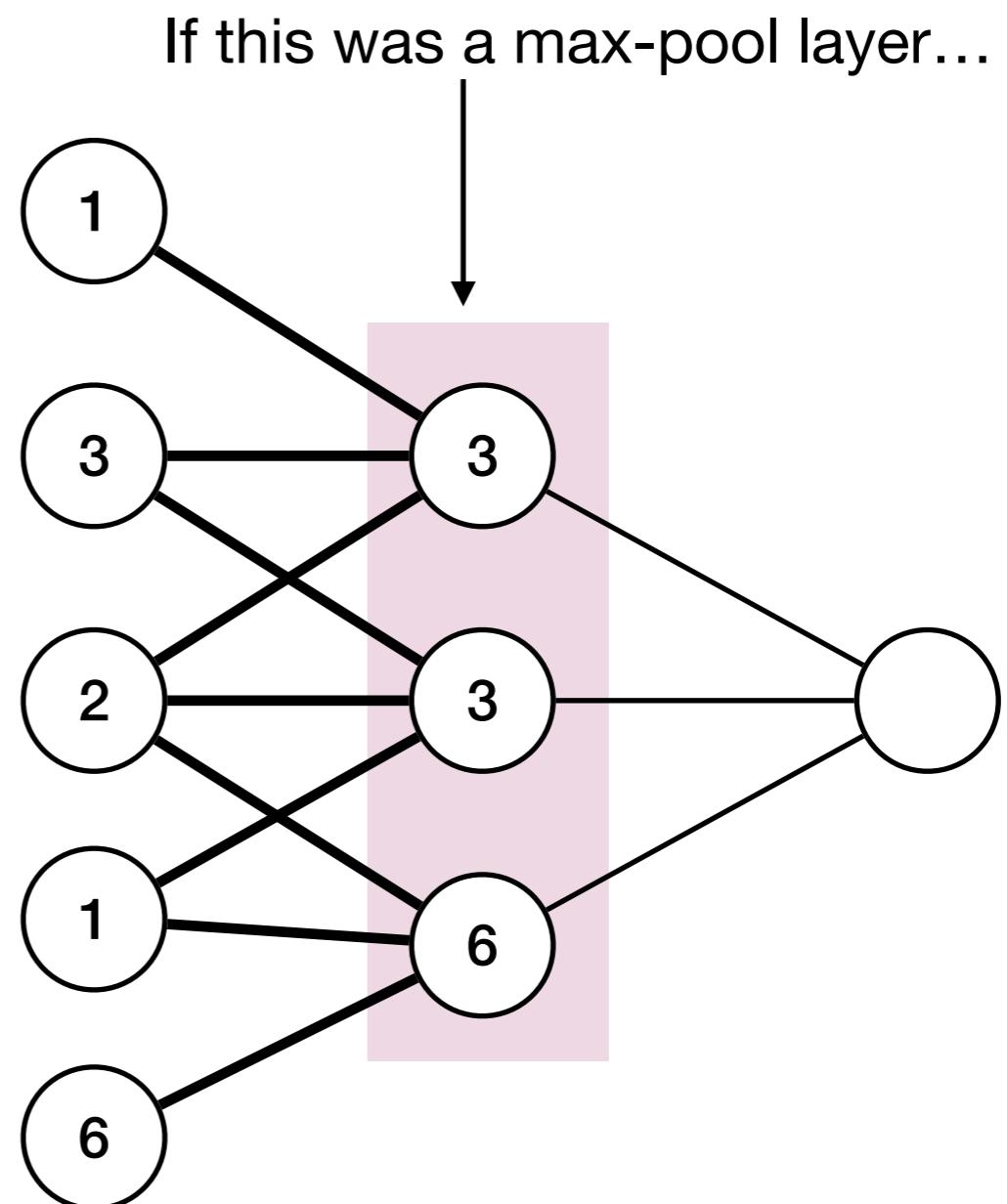
# Convolutional neural networks

- One kernel is usually not enough to summarize an image patch.  
A convolutional layer often use  $N$  kernels and output  $N$  channels.



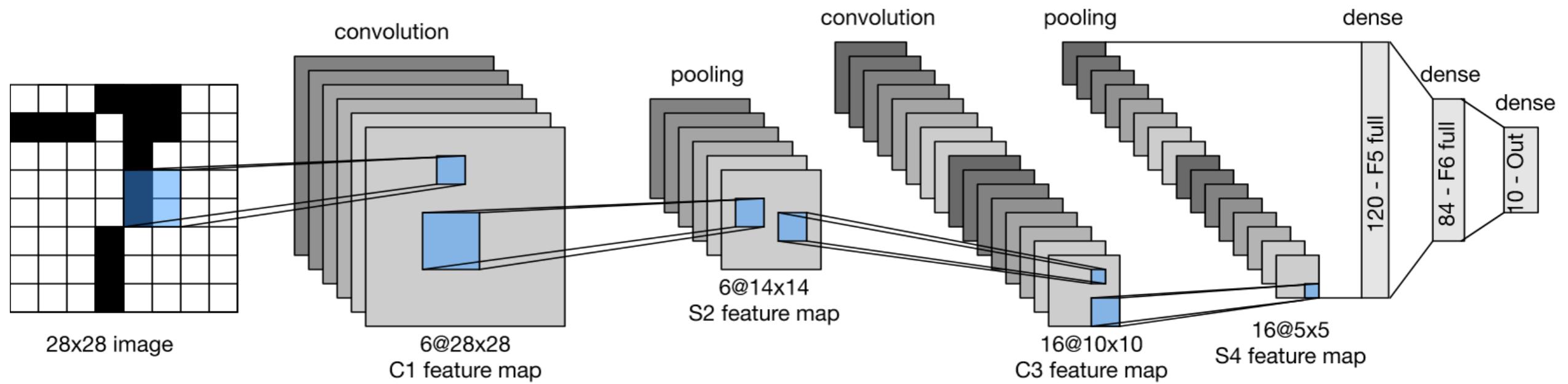
# Pooling layers

- Pooling operators similar to convolutional layers but contain no parameters. They usually just take the maximum or average over a window.
- Purpose is to decrease the number of nodes, on top of translational invariance and locality.



# Convolutional neural networks

Putting this all together...



LeNet-5 (Lecun 1998)

# Summary: CNNs

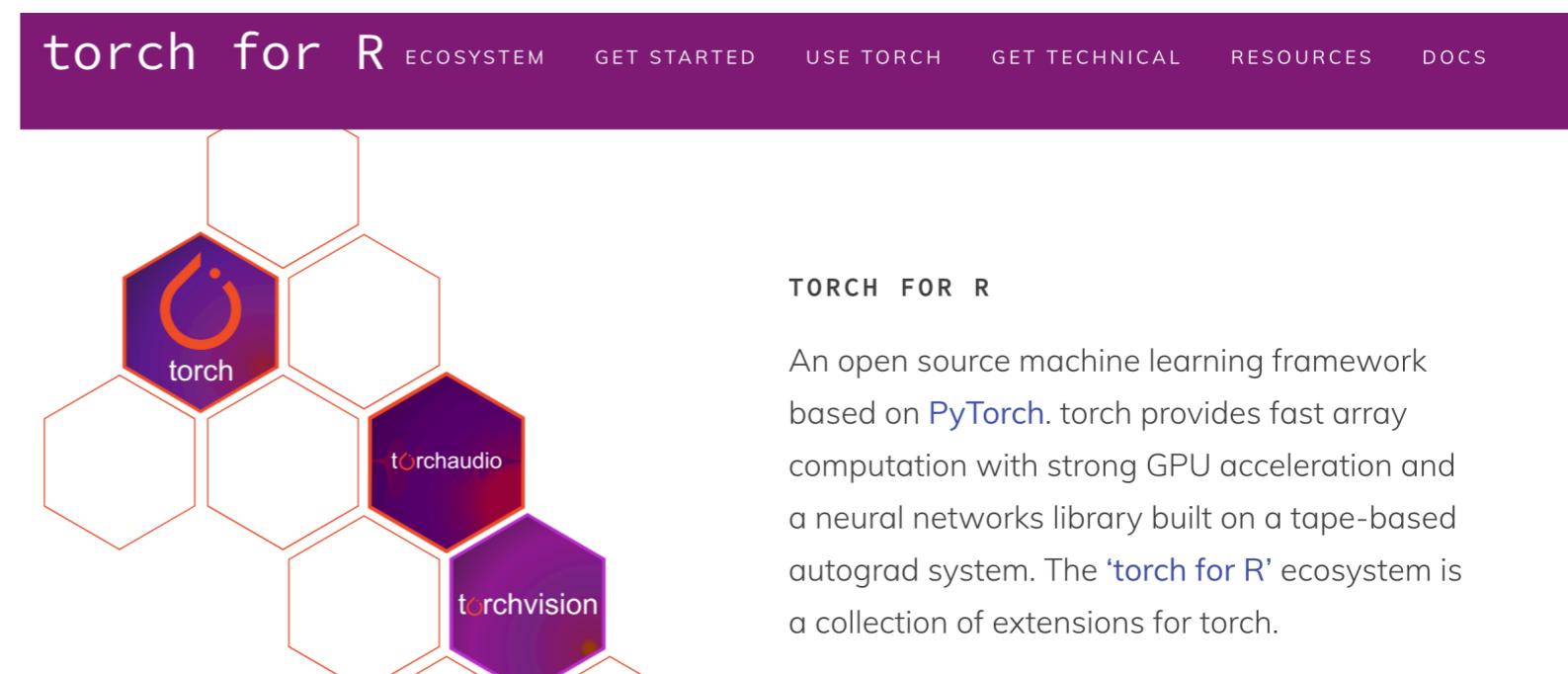
- CNNs are neural networks that are specially designed to handle data where the translational invariance and locality properties hold. This prior knowledge is encoded in the form of convolutional and pooling layers.
- They are useful for analyzing image data, molecular sequences, biological networks, and more.

# Outline

1. What is deep learning?
2. What can DL do and not do
3. Introduction to:
  1. Dense neural networks
  2. Convolutional neural networks
4. **How to get started using neural networks**

# Software for deep learning

- There are a lot of good software tools and tutorials online:
  - Python: PyTorch, TensorFlow, Keras
  - R: TensorFlow for R, Torch for R (<https://torch.mlverse.org/>)



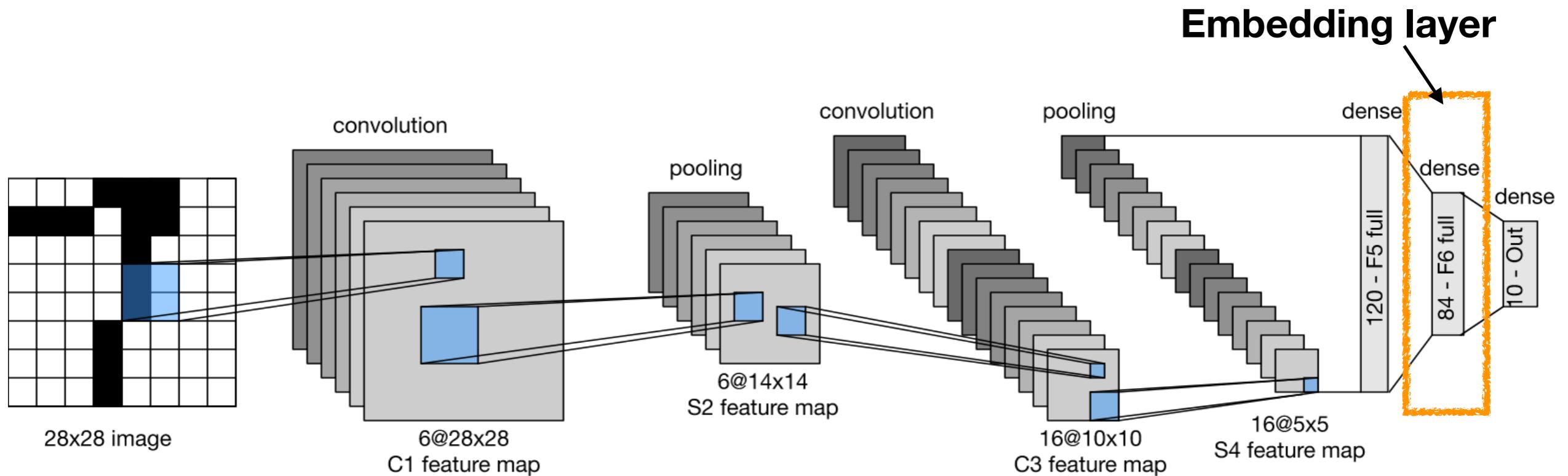
# Before you get too excited...

***Before you use a neural network, always try something simpler!***

- Neural networks take extra time and care. The amount of effort is not necessarily worth it. Generally, we find that...
  - For datasets with a small number of samples, simple methods like logistic regression often do as well as (and usually better than) a neural network.
  - For high-dimensional datasets, neural networks often struggle unless you properly employ regularization/feature selection.
- However, if there is a pre-trained neural network for a similar task, you can try to fine-tune this pre-trained model for the task of interest.

# Easiest way to get started

- Download pretrained networks and use them for feature extraction.



# Useful References

- Textbook: “Dive into Deep Learning” <https://d2l.ai/>
- Online courses: Coursera’s Deep learning courses
- A sequence of posts on torch on R:
  - <https://blogs.rstudio.com/ai/posts/2020-10-01-torch-network-from-scratch/>
  - <https://blogs.rstudio.com/ai/posts/2020-10-07-torch-modules/>
  - <https://blogs.rstudio.com/ai/posts/2020-10-09-torch-optim/>
  - <https://blogs.rstudio.com/ai/posts/2020-10-19-torch-image-classification/>

# Introduction to torch in R

- **Tensor:** a multi-dimensional array, e.g.
  - 1-D tensor for tabular data
  - 3-D tensor for images
- **Module:** A neural network layer, e.g.
  - `nn_conv1d()`: A convolutional layer over 1 dimension
- **Optimizers:** Optimization algorithms
- **Dataloaders:** Objects that load data

# Torch tensors

```
> library(torch)
> # a 1d vector of length 2
> t <- torch_tensor(c(1, 2))
> t
torch_tensor
1
2
[ CPUFloatType{2} ]
```

```
> # a 3x3 tensor (matrix)
> t <- torch_tensor(rbind(c(1,2,0), c(3,0,0), c(4,5,6)))
> t
torch_tensor
 1  2  0
 3  0  0
 4  5  6
[ CPUFloatType{3,3} ]
```

```
> t <- torch_zeros(3, 5, 5)
> t
torch_tensor
(1,...) =
 0  0  0  0  0
 0  0  0  0  0
 0  0  0  0  0
 0  0  0  0  0
 0  0  0  0  0

(2,...) =
 0  0  0  0  0
 0  0  0  0  0
 0  0  0  0  0
 0  0  0  0  0
 0  0  0  0  0

(3,...) =
 0  0  0  0  0
 0  0  0  0  0
 0  0  0  0  0
 0  0  0  0  0
 0  0  0  0  0

[ CPUFloatType{3,5,5} ]
```

# Torch tensors

## Indexing

```
> t <- torch_tensor(rbind(c(1,2,3), c(4,5,6)))
> t
torch_tensor
 1  2  3
 4  5  6
[ CPUFloatType{2,3} ]
>
> # a single value
> t[1, 1]
torch_tensor
1
[ CPUFloatType{} ]
>
> # first row, all columns
> t[1, ]
torch_tensor
 1
 2
 3
[ CPUFloatType{3} ]
```

## Converting torch tensor back to R

```
> t <- torch_tensor(matrix(1:9, ncol = 3, byrow = TRUE))
> as_array(t)
 [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

# Modules

## Creating a linear module

```
> l <- nn_linear(3, 1)
> l
An `nn_module` containing 4 parameters.

— Parameters ——————
• weight: Float [1:1, 1:3]
• bias: Float [1:1]
```

```
> l$parameters
$weight
torch_tensor
 0.1939  0.0619  0.5187
[ CPUFloatType{1,3} ]

$bias
torch_tensor
 0.1350
[ CPUFloatType{1} ]
```

## Calling a module given input data

```
> data <- torch_randn(10, 3)
> out <- l(data)
> out
torch_tensor
 0.1826
 0.3797
 0.1410
-0.3324
 0.4497
 1.2186
-0.2790
 0.0586
 0.3796
-0.4895
[ CPUFloatType{10,1} ]
```

# Modules

## Defining a model as a sequence of modules

```
> model <- nn_sequential(  
+     nn_linear(3, 16),  
+     nn_relu(),  
+     nn_linear(16, 1)  
)  
>  
> model  
An `nn_module` containing 81 parameters.
```

- 
- Modules —————
- 
- 0: <nn\_linear> #64 parameters
  - 1: <nn\_relu> #0 parameters
  - 2: <nn\_linear> #17 parameters

## Indexing into the model

```
> model[[1]]  
An `nn_module` containing 64 parameters.  
  
— Parameters —————  


---



- weight: Float [1:16, 1:3]
- bias: Float [1:16]

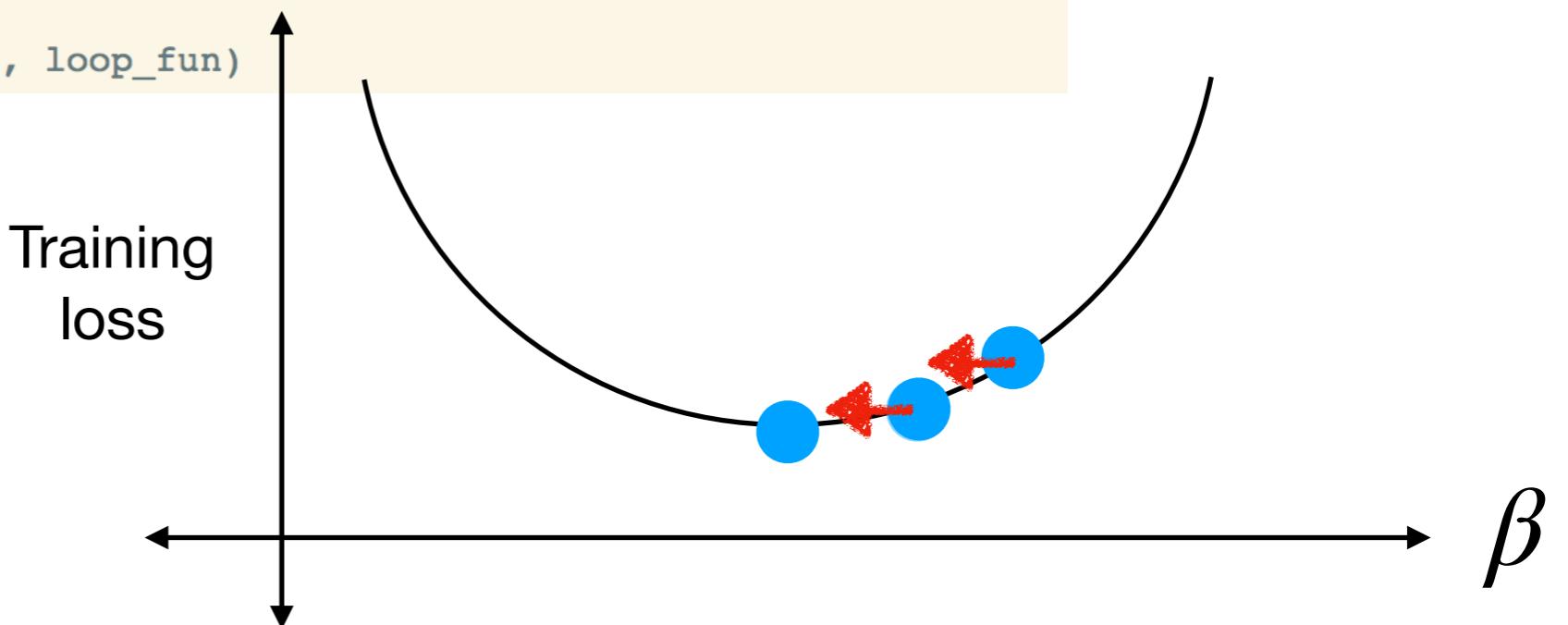
```

## Calling a model given input data

```
> out <- model(data)  
> out  
torch_tensor  
-0.3374  
-0.2348  
-0.5673  
-0.2602  
-0.1857  
-0.1011  
-0.3001  
-0.1868  
-0.2397  
-0.3328  
[ CPUFloatType{10,1} ]
```

# Optimizers

```
> optimizer <- optim_adam(model$parameters, lr = 0.01)
> optimizer
<optim_adam>
  Inherits from: <torch_Optimizer>
  Public:
    add_param_group: function (param_group)
    clone: function (deep = FALSE)
    defaults: list
    initialize: function (params, lr = 0.001, betas = c(0.9, 0.999), eps = 1e-08,
    param_groups: list
    state: list
    step: function (closure = NULL)
    zero_grad: function ()
  Private:
    step_helper: function (closure, loop_fun)
```



# Dataloader

```
> train_small
<dataset>
  Public:
    .getitem: function (idx)
    .length: function ()
    clone: function (deep = FALSE)
    dataset: tiny_imagenet, image_folder, folder, dataset, R6
    indices: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 ...
    initialize: function (dataset, indices)
```

```
> train_dl <- dataloader(train_small, batch_size=ntrain, shuffle = F)
> train_dl
<dataloader>
  Public:
    .auto_collation: active binding
    .dataset_kind: map
    .index_sampler: active binding
    .iter: function ()
    .length: function ()
    batch_sampler: utils_sampler_batch, utils_sampler, R6
    batch_size: 1500
    clone: function (deep = FALSE)
    collate_fn: function (batch)
    dataset: dataset, R6
    drop_last: FALSE
    generator: NULL
    initialize: function (dataset, batch_size = 1, shuffle = FALSE, sampler = NULL,
    multiprocessing_context: NULL
    num_workers: 0
    pin_memory: FALSE
    sampler: utils_sampler_sequential, utils_sampler, R6
    timeout: -1
    worker_init_fn: NULL
```