From CS188, FA18, UC Berkeley: http://ai.berkeley.edu.

Neural Networks

Introduction to Data Science
Spring 1404

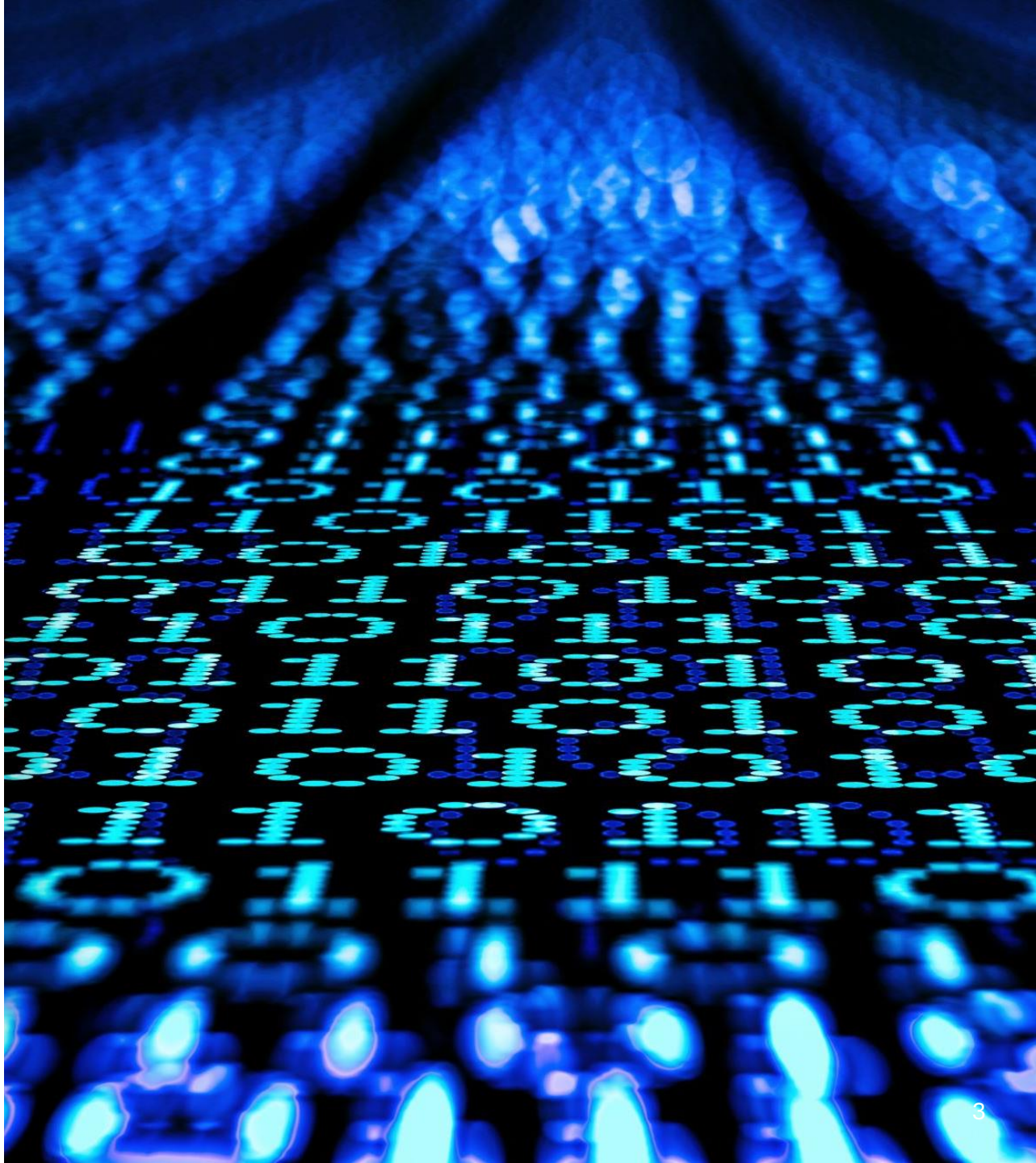
Yadollah Yaghoobzadeh

Deep Learning

Multi-layer artificial neural networks

Rise of DL

- Data
- Compute
- Algorithms



Deep learning

- What is its position wrt to ML and AI?

Key differences with other ML techniques

- It learns the features itself
- It is scalable and learn from any large data

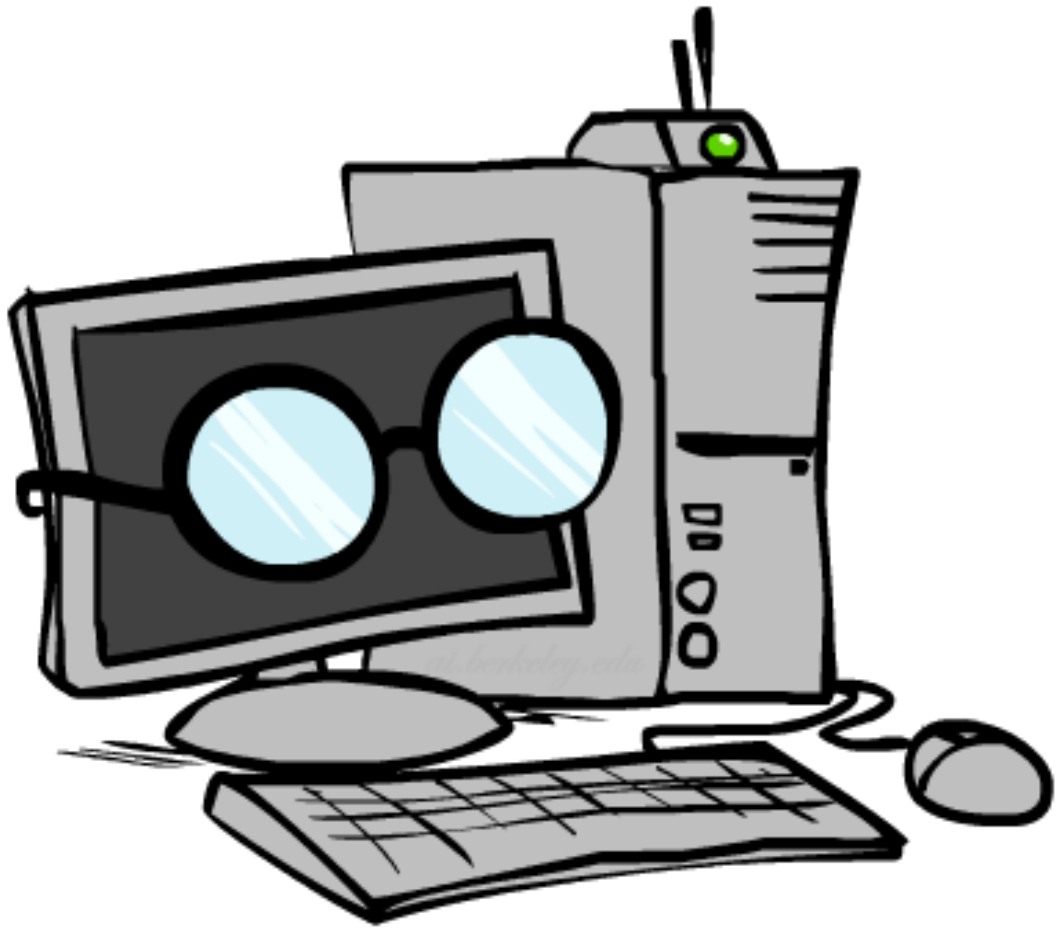
Applications

- Natural language processing
  - Machine translation
  - Search
  - Sentiment analysis

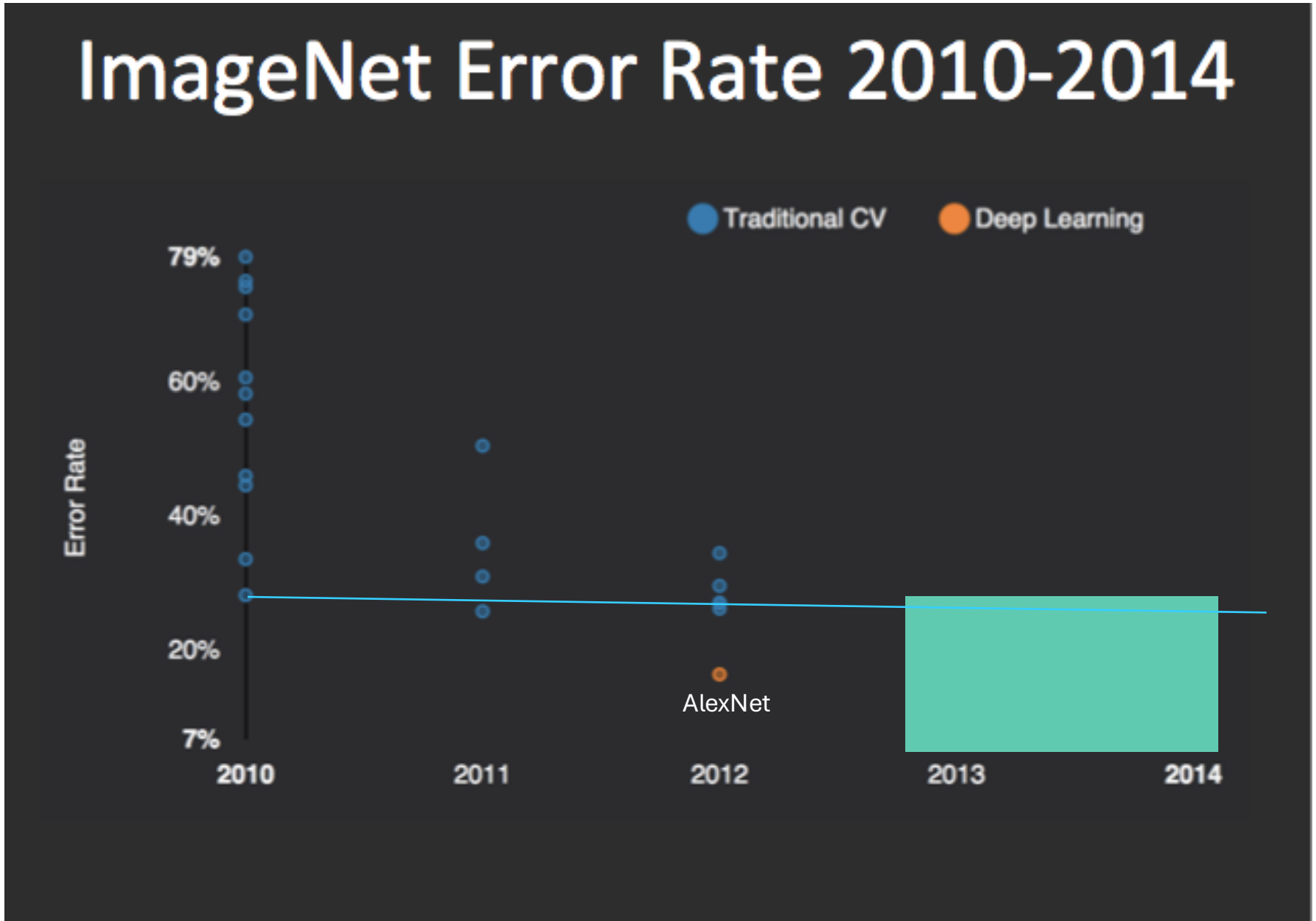
  ➔ Large language models
- Computer vision
  - Image captioning
  - Image generation
  - Image classification
  - Image segmentation
- Recommendation systems

How well does it work?

Computer Vision

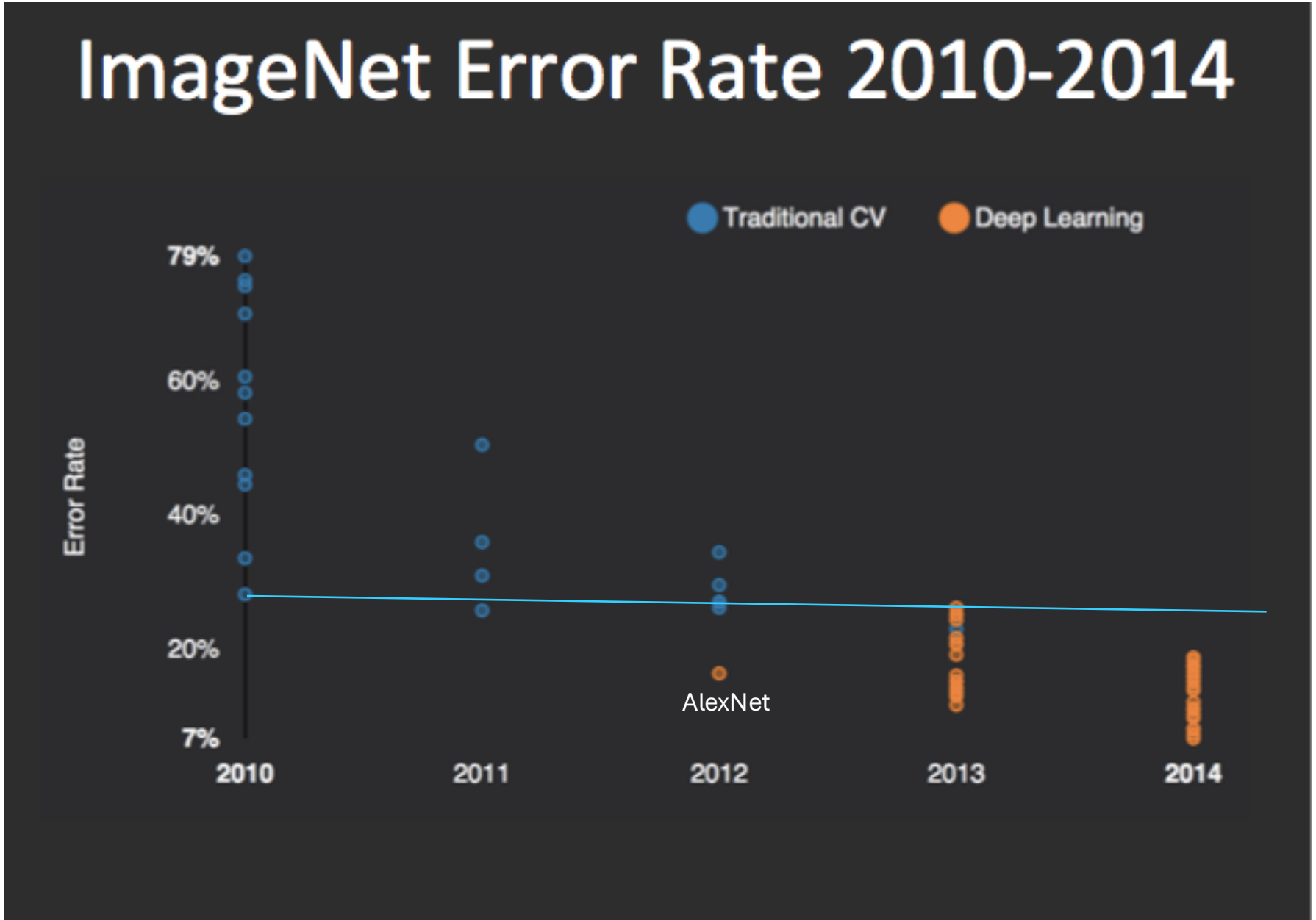


Performance



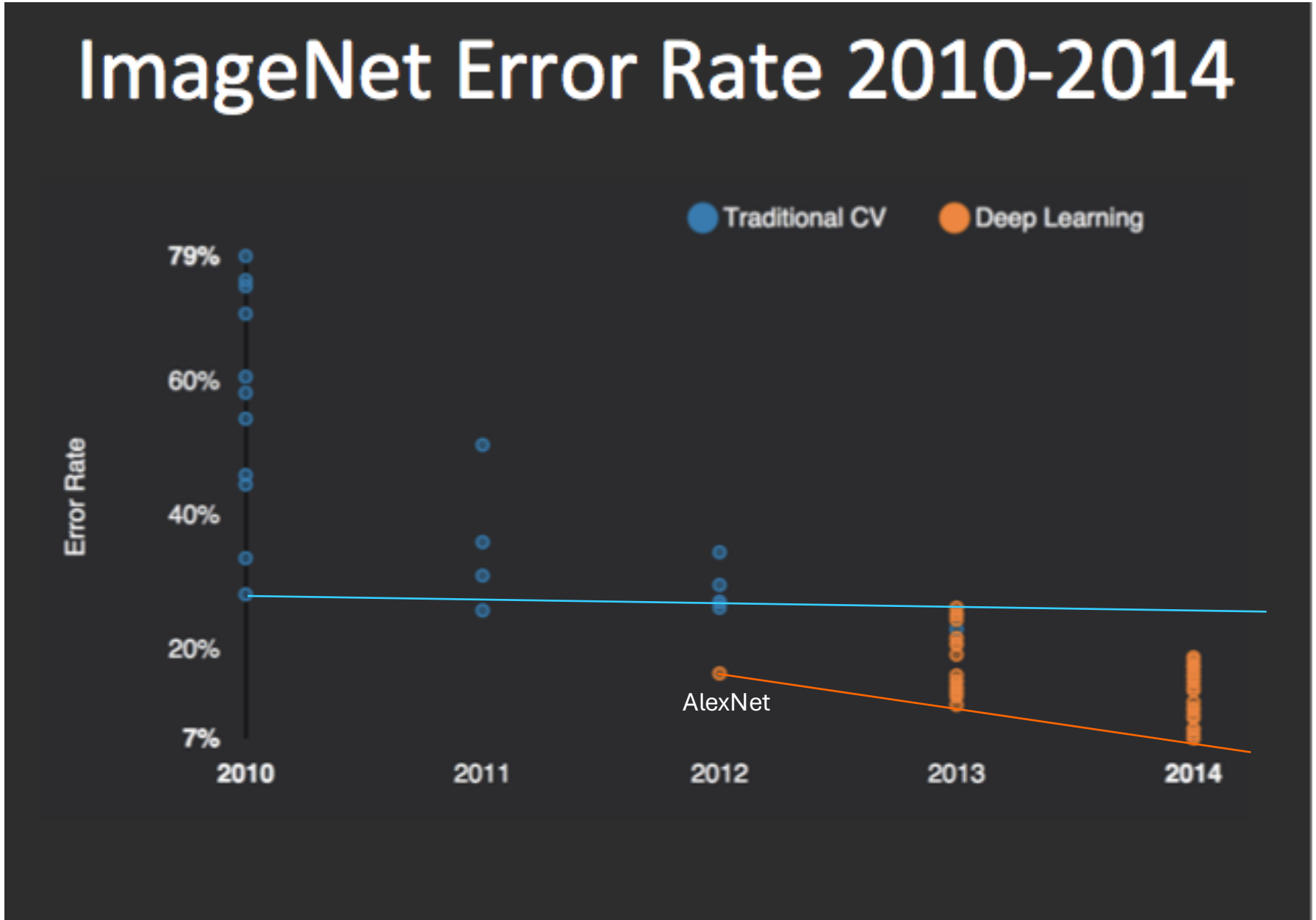
graph credit Matt Zeiler, Clarifai

Performance



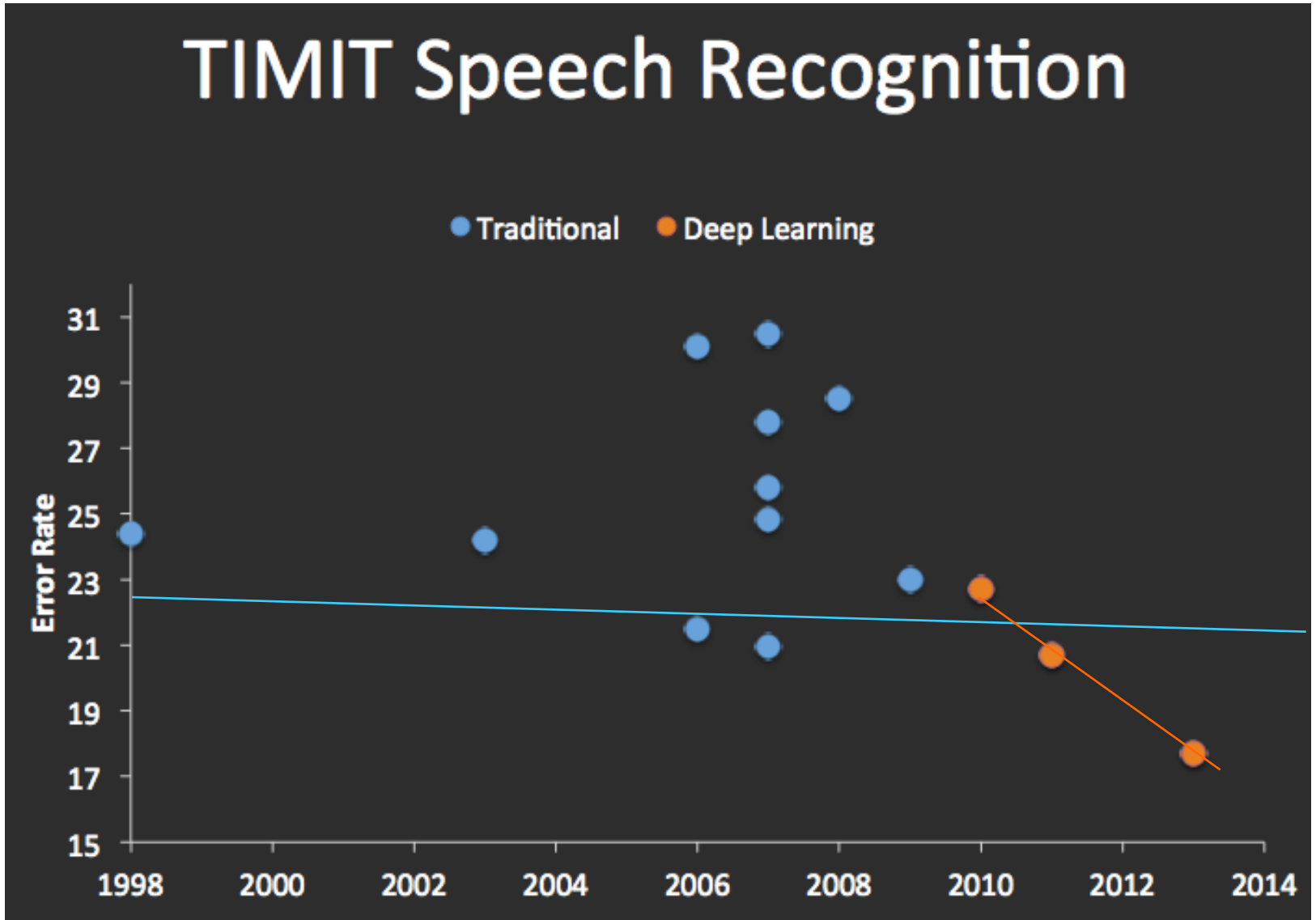
graph credit Matt Zeiler, Clarifai

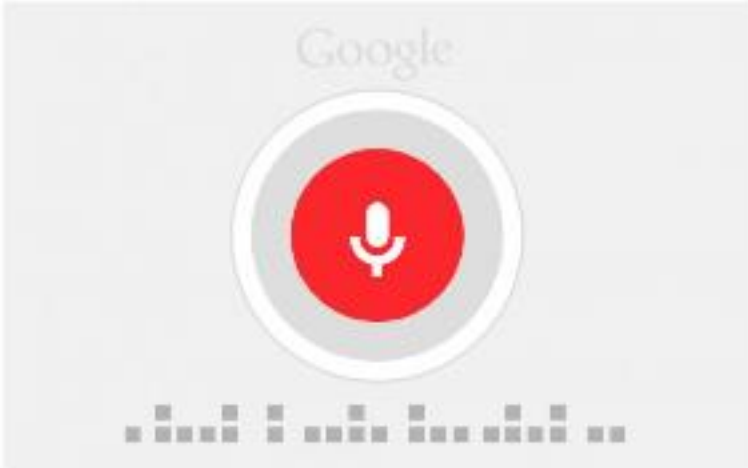
Performance



graph credit Matt Zeiler, Clarifai

Speech Recognition





graph credit Matt Zeiler, Clarifai

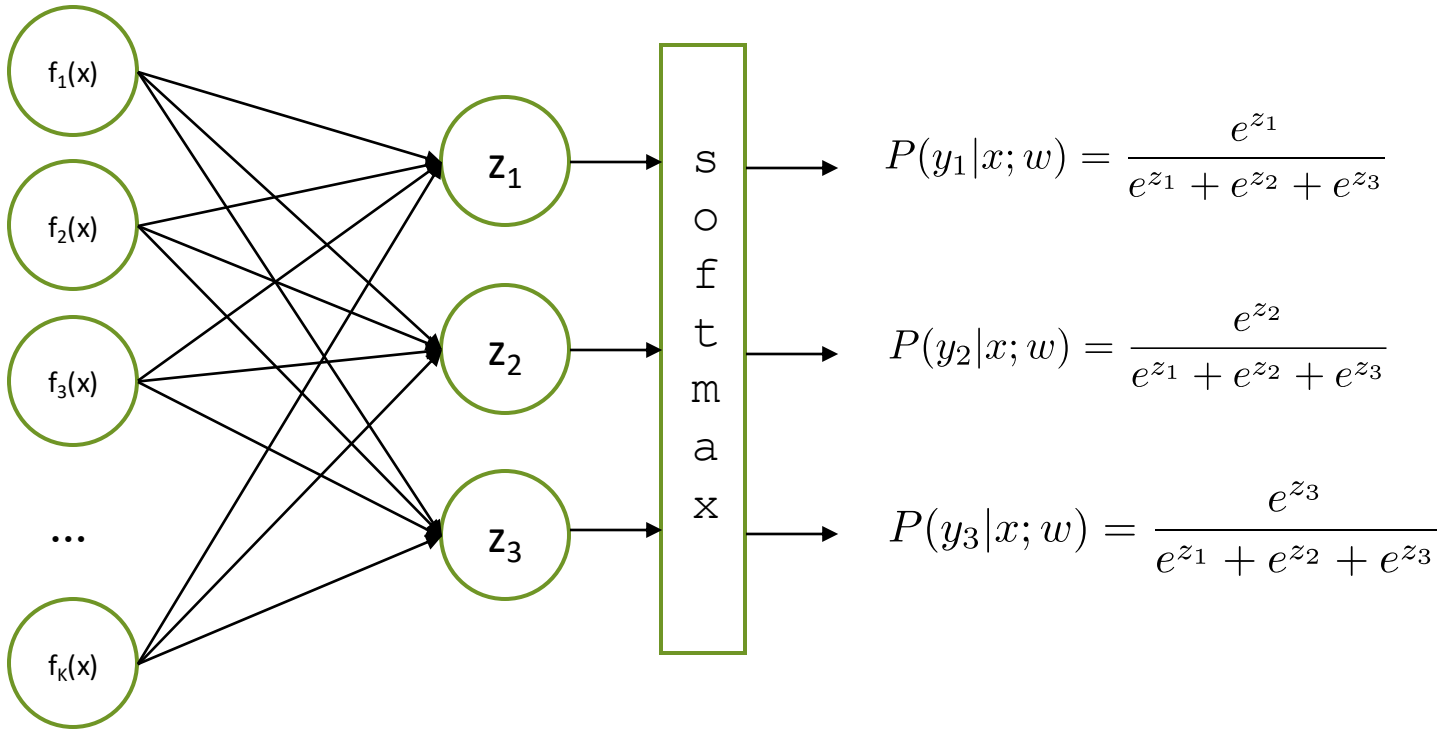
Natural language processing

- BERT (2017)
- ChatGPT (2022)

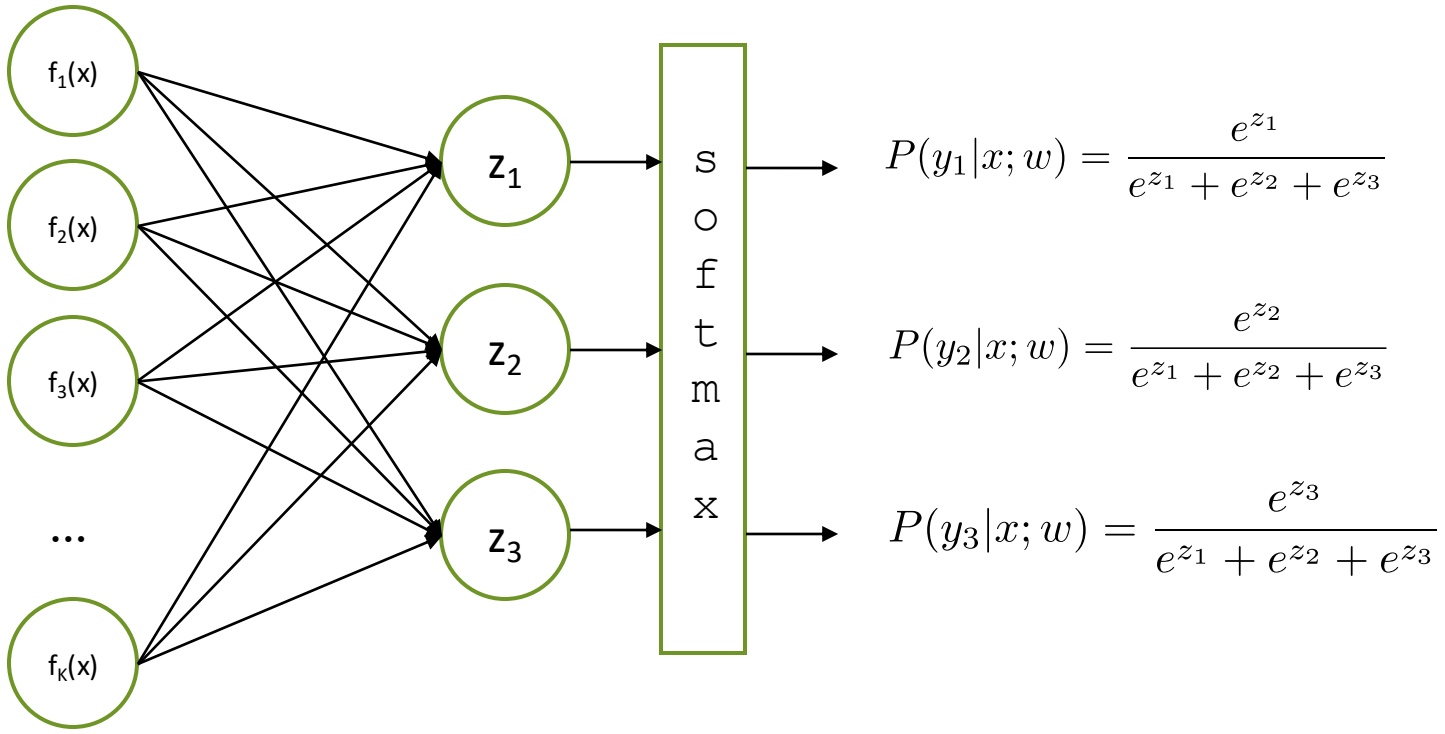
But what is ANN?

Multi-class Logistic Regression

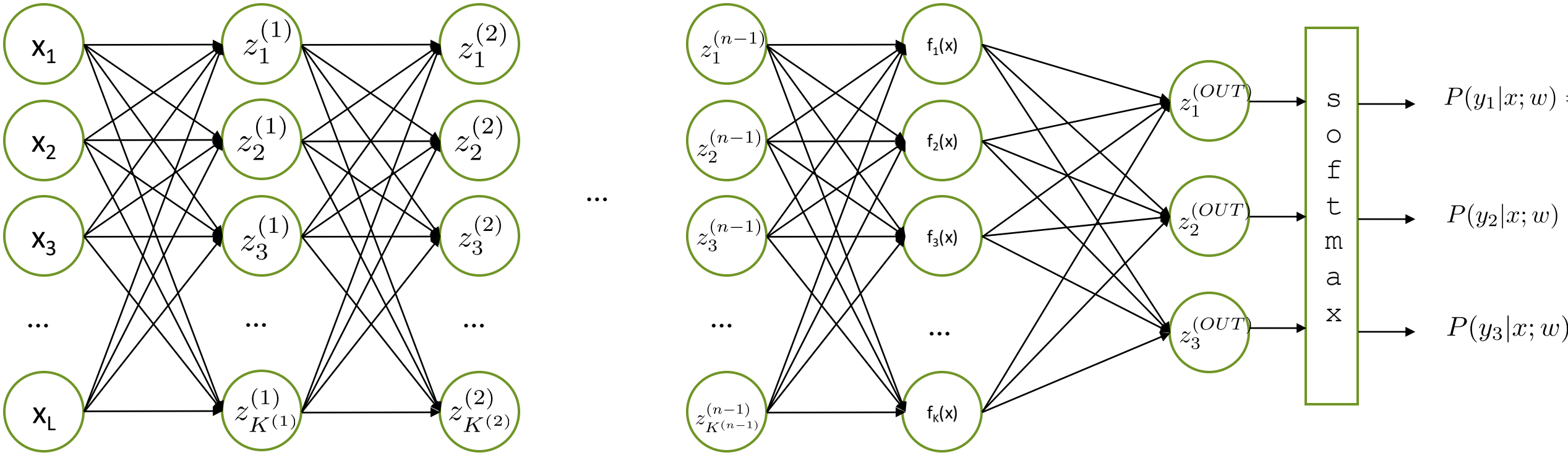
= special case of neural network



Deep Neural Network = Also learn the features!



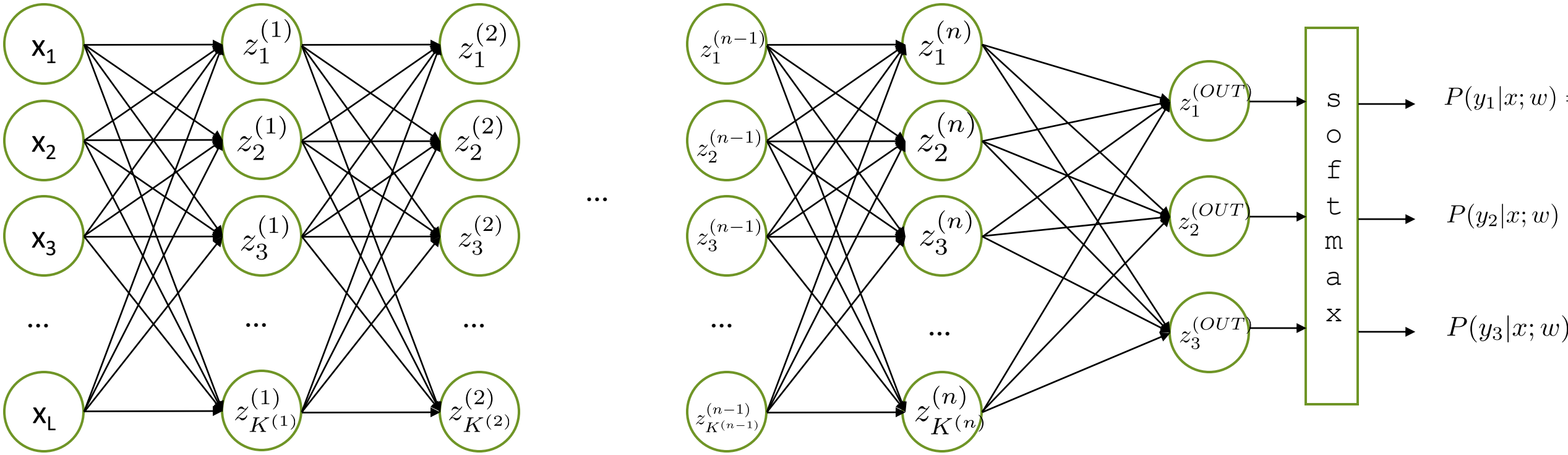
Deep Neural Network = Also learn the features!



$$z_i^{(k)} = g(\sum_j W_{i,j}^{(k-1,k)} z_j^{(k-1)})$$

g = nonlinear activation function

Deep Neural Network = Also learn the features!

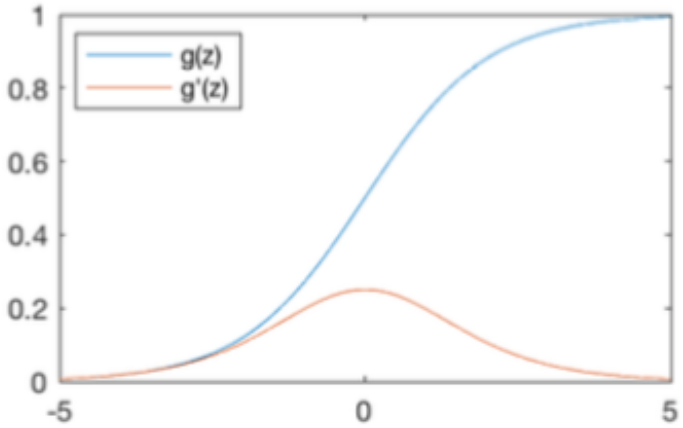


$$z_i^{(k)} = g(\sum_j W_{i,j}^{(k-1,k)} z_j^{(k-1)})$$

g = nonlinear activation function

Common Activation Functions

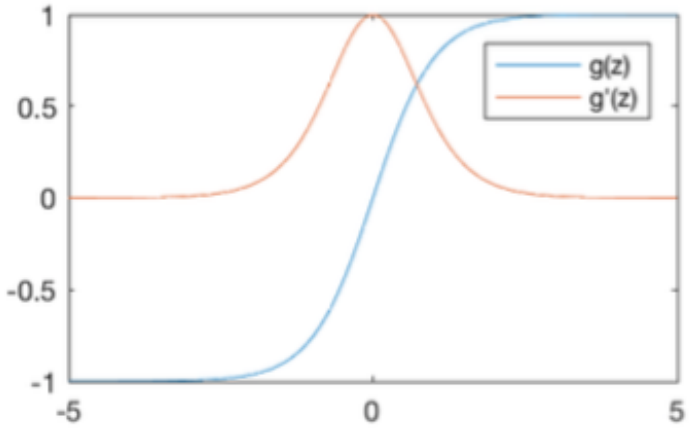
Sigmoid Function



$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

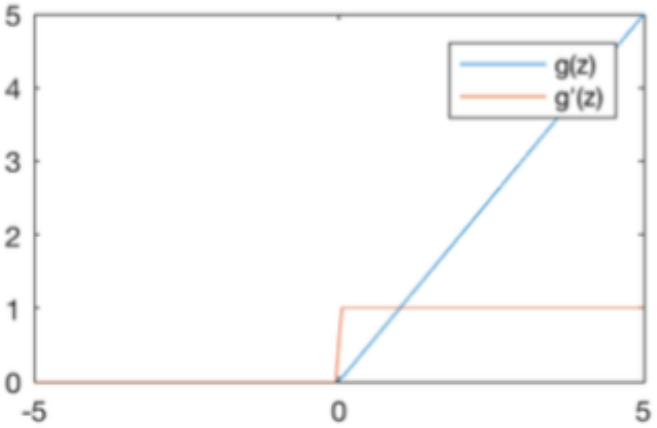
Hyperbolic Tangent



$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

Rectified Linear Unit (ReLU)



$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

[source: MIT 6.S191 introtodeeplearning.com]

Deep Neural Network: Also Learn the Features!

Training the deep neural network is just like logistic regression:

$$\max_w \quad ll(w) = \max_w \quad \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

just w tends to be a much, much larger vector ☺

→just run gradient ascent

+ stop when log likelihood of hold-out data starts to decrease

Optimization Procedure: Gradient Ascent

- init w
- for iter = 1, 2, …

$$w \leftarrow w + \alpha * \nabla g(w)$$

- α: learning rate --- tweaking parameter that needs to be chosen carefully
- How? Try multiple choices
  - Crude rule of thumb: update changes w about 0.1 – 1 %

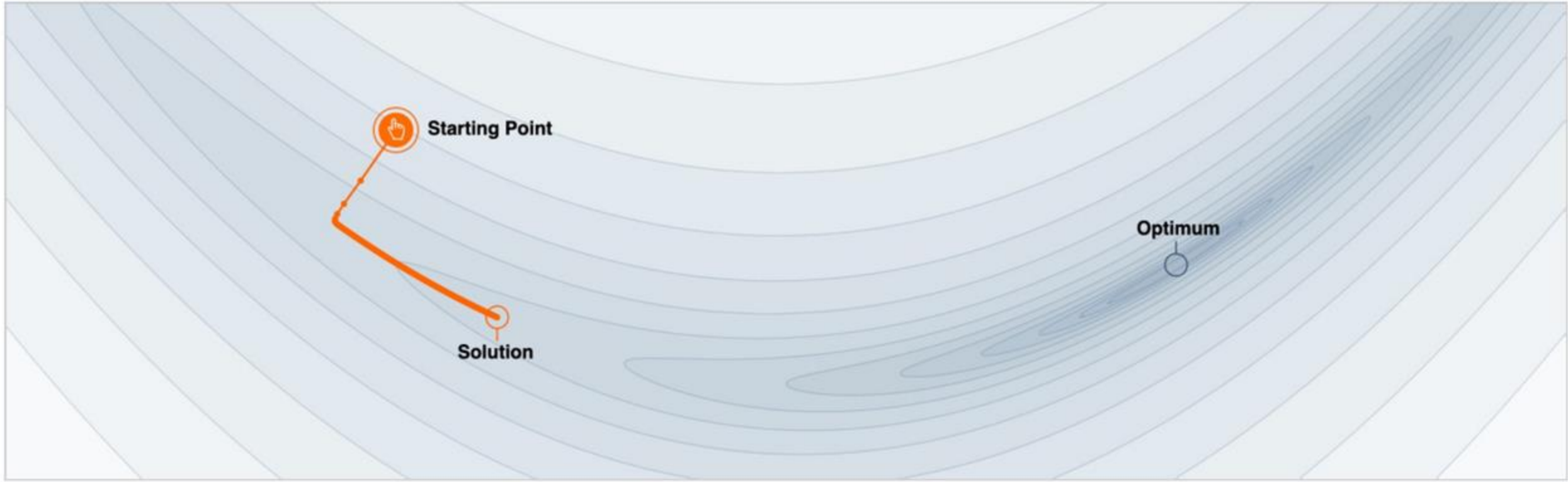
Minimizing loss functions

- In practice we minimize the loss instead of maximizing the objective function
- E.g.,
  - Cross-entropy loss for classification
  - Mean squared error for regression

Learning Rate

Choice of learning rate α is a hyperparameter

Example: α=0.001 (too small)

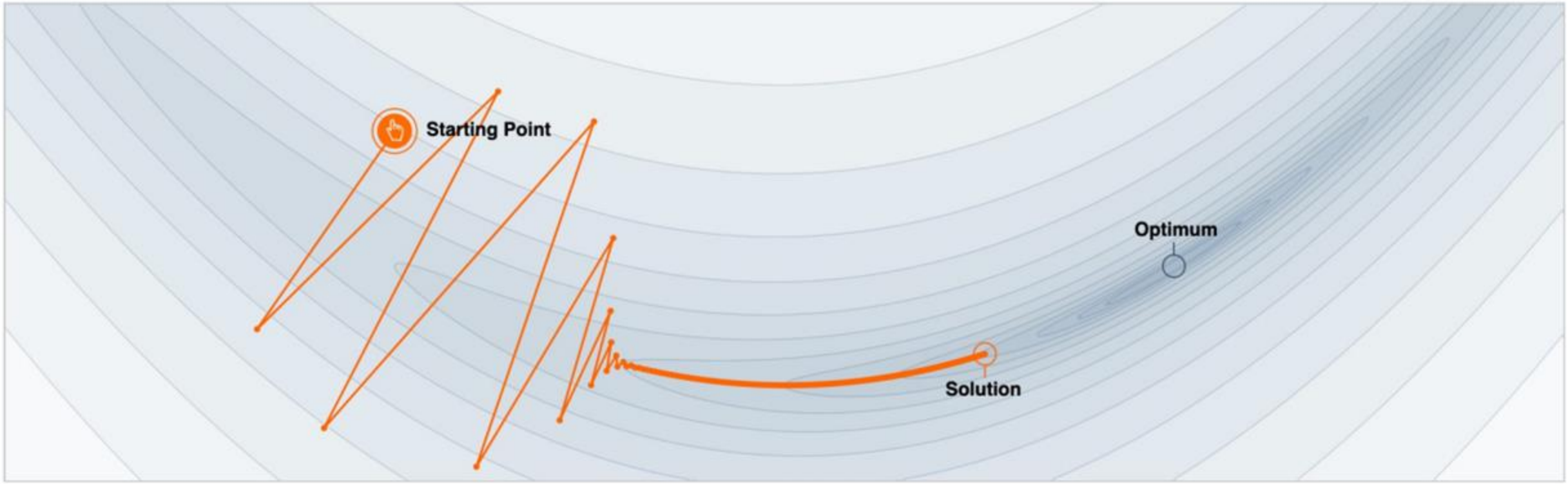


Source: https://distill.pub/2017/momentum/

Learning Rate

Choice of step size α is a hyperparameter

Example: α=0.004 (too large)



Fun Neural Net Demo Site

- Demo-site:
  - http://playground.tensorflow.org/

Backpropagation

- **Back-propagation:** gradients are computed in the direction from output to input layers and combined using chain rule

- CHAIN RULE:

  If

$$f(x) = g(h(x))$$

$$f'(x) = g'(h(x))h'(x)$$

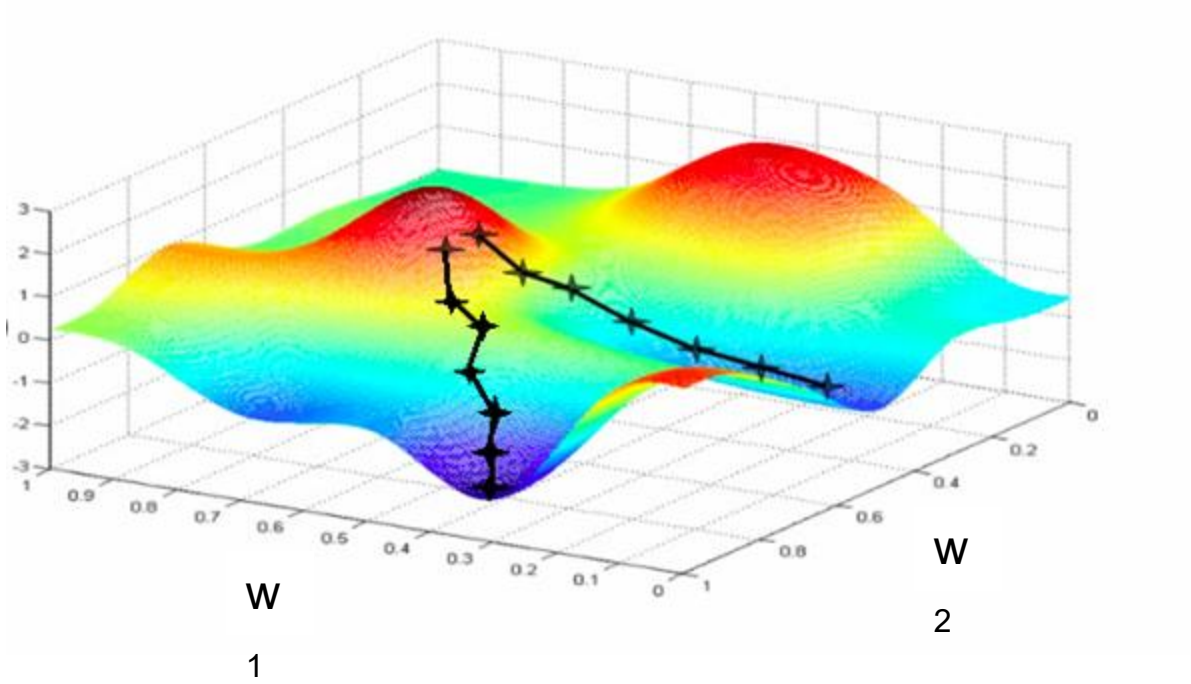
  Then

  → **Derivatives can be computed by following well-defined procedures**

Deep Neural Network: Training

- Update weights by **gradient descent:**

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial E}{\partial \mathbf{w}}$$



Automatic Differentiation

- Automatic differentiation software
  - e.g. Theano, TensorFlow, PyTorch, Chainer
  - Only need to program the function g(x,y,w)
  - Can automatically compute all derivatives w.r.t. all entries in w
  - This is typically done by caching info during forward computation pass of f, and then doing a backward pass = “backpropagation”
  - Autodiff / Backpropagation can often be done at computational cost comparable to the forward pass
- Need to know this exists
- How this is done? -- outside of scope of this class

Summary of Key Ideas

- Optimize probability of label given input $\max_w \; ll(w) = \max_w \; \sum_i \log P(y^{(i)}|x^{(i)}; w)$
- Continuous optimization
  - Gradient ascent:
    - Compute steepest uphill direction = gradient (= just vector of partial derivatives)
    - Take step in the gradient direction
    - Repeat (until held-out data accuracy starts to drop = "early stopping")
- Deep neural nets
  - Last layer = still logistic regression
  - Now also many more layers before this last layer
    - = computing the features
    - → the features are learned rather than hand-designed
  - Universal function approximation theorem
    - `If` neural net is large enough
    - `Then` neural net can represent any continuous mapping from input to output with arbitrary accuracy
    - But remember: need to avoid overfitting / memorizing the training data → early stopping!
  - Automatic differentiation gives the derivatives efficiently (how? = outside of scope of this class)

The Training Loop

Repeat for multiple epochs

- Get a batch of data
- Forward pass: Calculate predictions
- Calculate Loss
- Backward pass: Calculate gradients (Backprop)
- Update weights/biases (Optimizer)

- We usually have a number of hyperparameters to tune
  - Optimizer hyperparameters like learning rate
  - Batch size
  - Epochs

Connection Architecture

- Fully connected
- Convolution
- Recurrent
- Self-attention

Fully connected layers