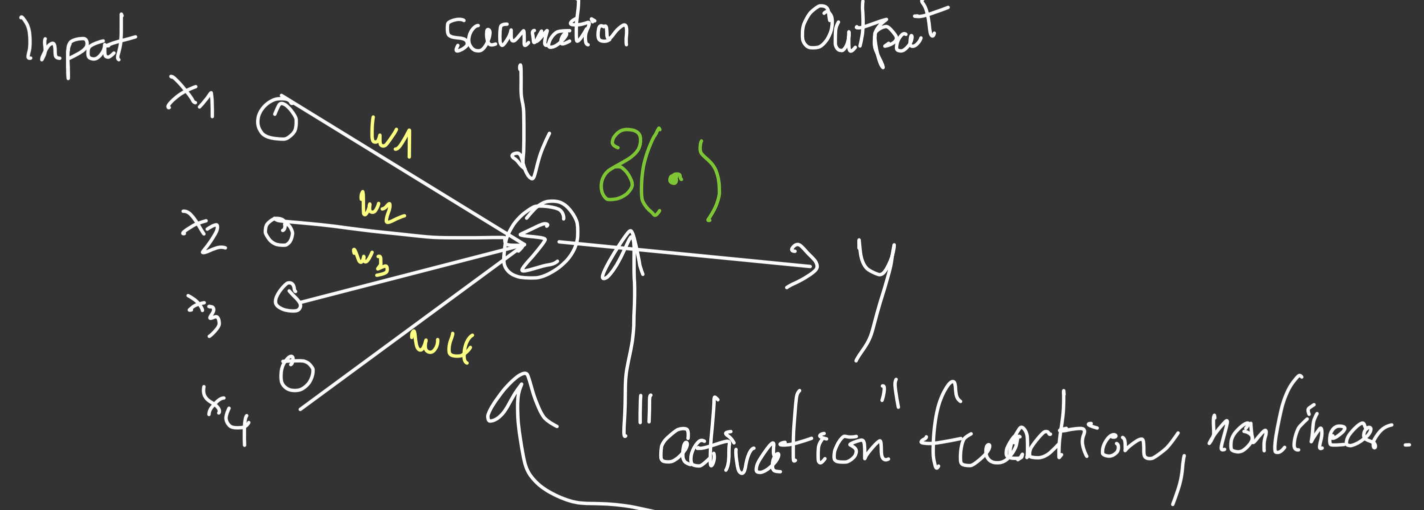
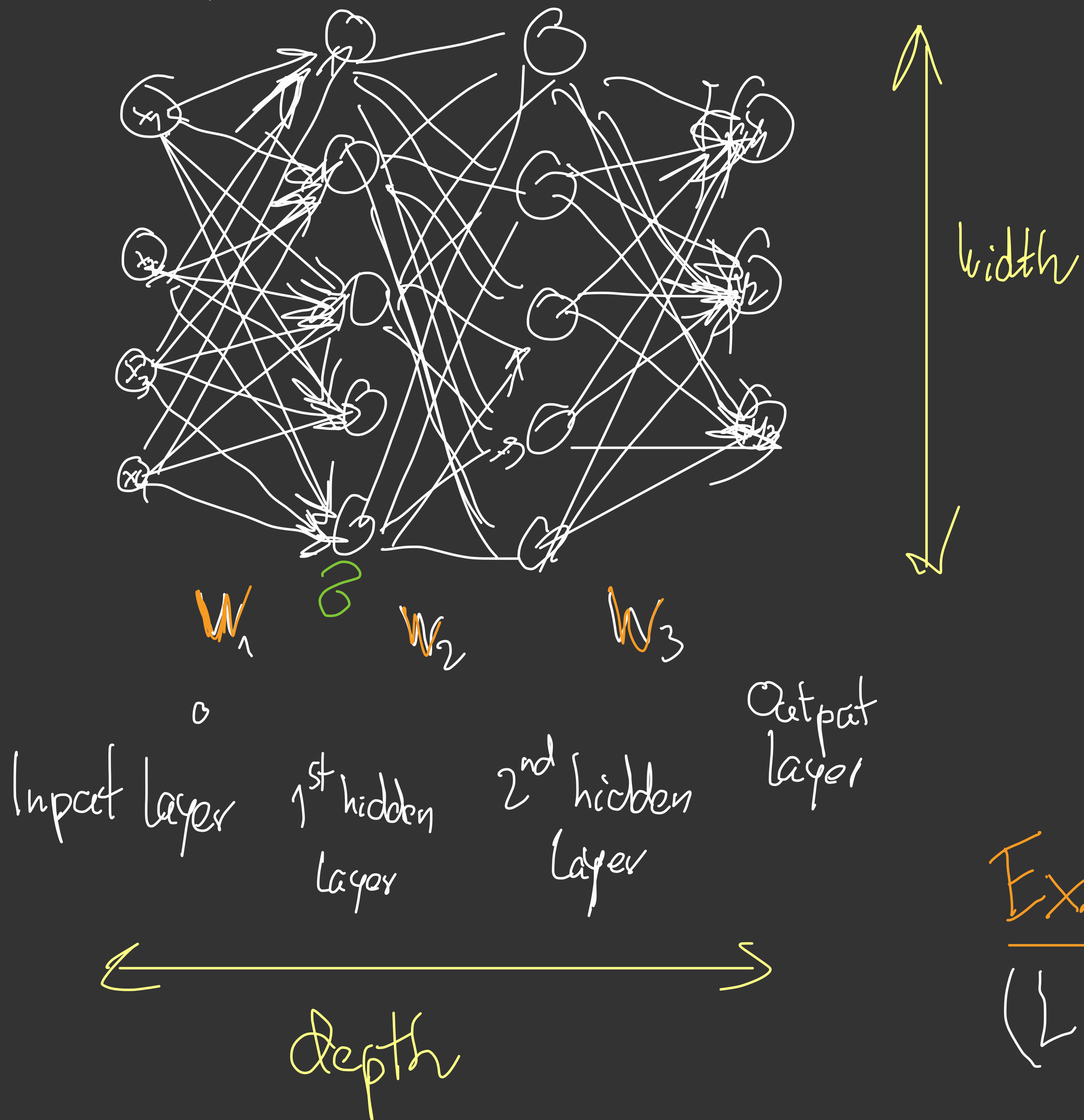
Artificial Neural Networks

(Selective) Little History: [McCulloch, Pitts '43]: Mathematical model for neurons in brain:



- 1950's: Perceptron, 1st numerical scheme to use collection of) to do classification [Rosenblatt].
- 1980's: "Deeper" networks, successful training via backpropagation (training algor.)
- Since ~2007: ▷ Efficient training ($\hat{=}$ determination of weights w_i from training set) becomes possible for deeper networks ⟶ regularization
  - ▷ Advances in computing technology: Use GPUs
  - ▷ Outperformance of traditional learning methods such as support vector machines

A template for neural networks:



A typical hypothesis space $\mathcal{F}$ associated to an ANN:

$$\mathcal{F}_L = \Big\{ h : X \to Y : h(x) = \sigma(W_L \sigma(W_{L-1} \sigma(\dots \sigma(W_1(x))))),$$

σ non-linear componentwise,

$W_\ell : \mathbb{R}^{N_{\ell-1}} \to \mathbb{R}^{N_\ell}$ (affine) linear $\forall \ell = 1, .., L \Big\}$.

N_0: dimension of input layer

N_1: ——"—— 1st hidden layer

$\vdots$

N_L: dimension of output layer.

Example: (Multiclass) Logistic Regression: 1-layer NN, ($L = 1$), $\sigma(\cdot) = \text{softmax}(\cdot)$ since

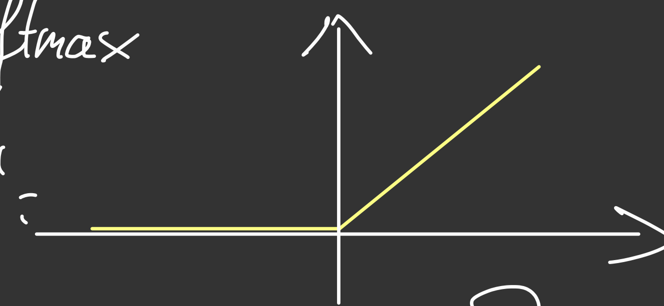
$$h(x) = \text{softmax}(Wx).$$

- If. ↙ and $L > 1$: Multilayer Perceptron.

Thm: [Hornik '91, Cybenko '89]. Every measurable function can be approximated by a NN with $L = 2$ (i.e., if wide enough).

Practical Q:

- How many layers L, how wide each layer?
- Choice of loss function?
- Which activation function? "Sigmoid"/Softmax; "ReLU":



- How to "train" the network (i.e., determine weights W_ℓ)?
  - ↳ Some variant of stochastic gradient descent, implemented via backpropagation (i.e., a very smart implementation of the "chain rule")

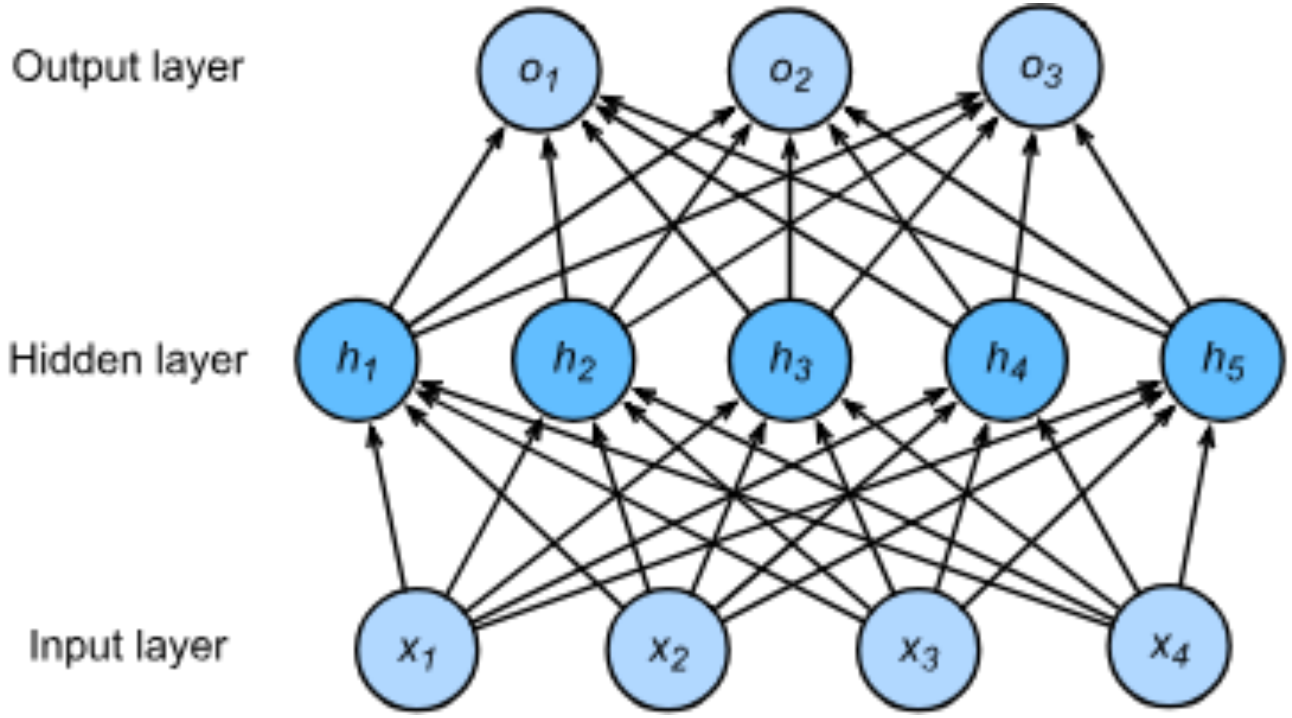
Modifications / variants:

- Constrain weights to have certain properties: Convolutional Neural Networks (CNN) / good for image problems (enforce spatial invariance).
- Fewer connections → less overfitting
- (Max/Average) Pooling: Reduces spatial sensitivity
- Dropout: Drops (at random) connections between layers in training phase
- Batch normalization: Normalizes input of a layer
  ↳ faster learning, better generalization

Epoch: One pass through entire training data

Learning Rate: Stepsize of (Minibatch) Stochastic Gradient Descent (or other optimizer)

Single Hidden Layer



Hyperparameter - size m of hidden layer

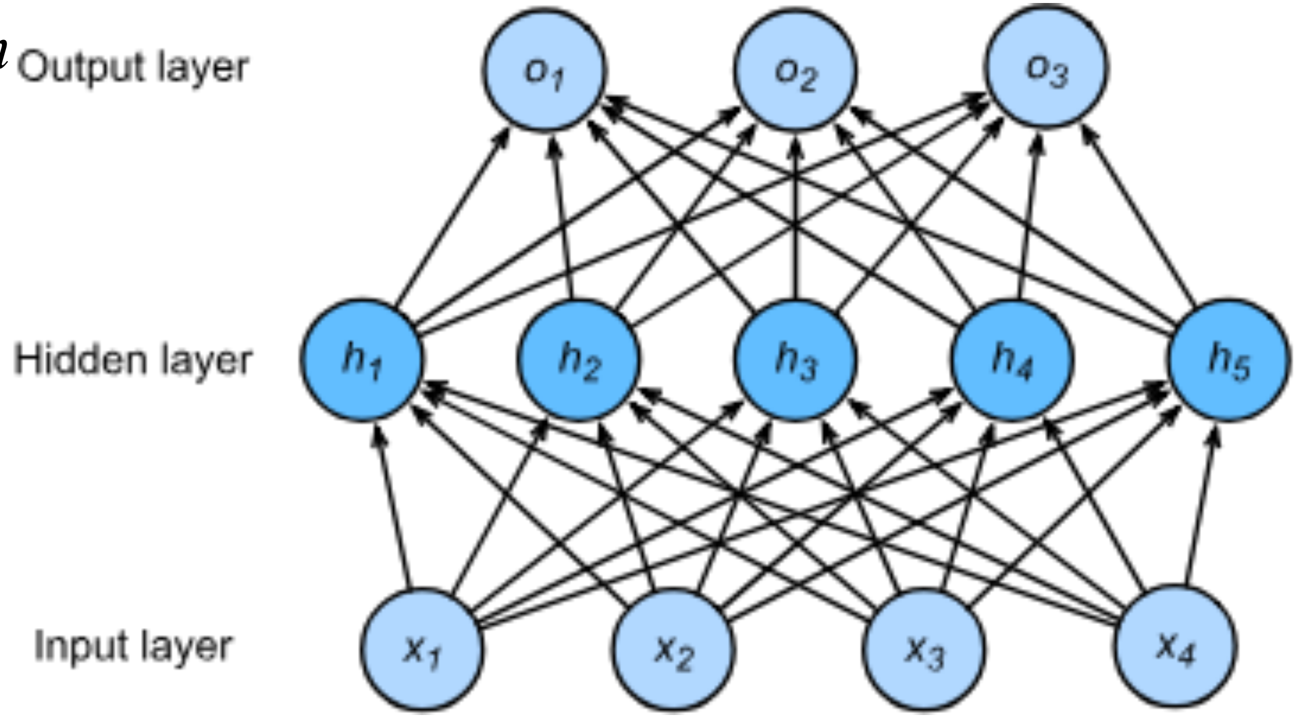
Single Hidden Layer

- Input $\mathbf{x} \in \mathbb{R}^n$
- Hidden $\mathbf{W}_1 \in \mathbb{R}^{m \times n}, \mathbf{b}_1 \in \mathbb{R}^m$
- Output $\mathbf{W}_2 \in \mathbb{R}^m, \mathbf{b}_2 \in \mathbb{R}$

$$\mathbf{h} = \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{o} = \mathbf{W}_2^T \mathbf{h} + \mathbf{b}_2$$

σ is an element-wise activation function



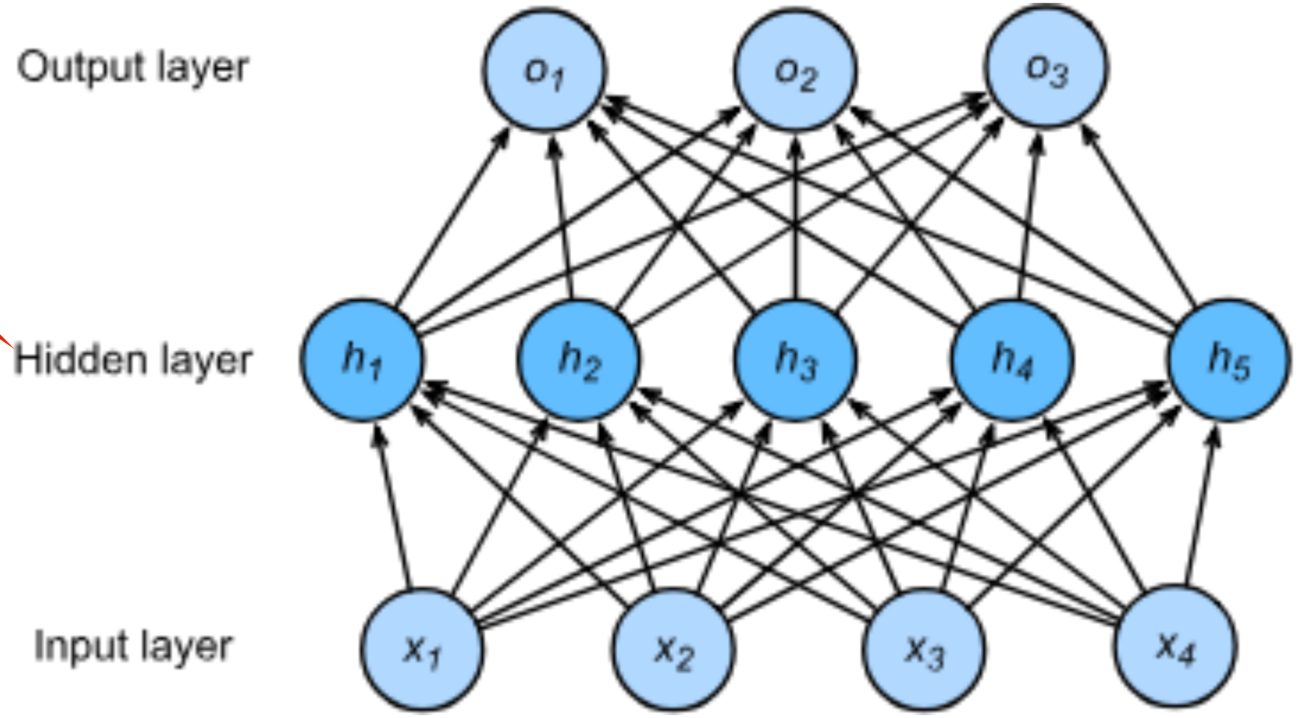
Single Hidden Layer

Why do we need an a nonlinear activation?

$$\mathbf{h} = \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{o} = \mathbf{W}_2^T \mathbf{h} + \mathbf{b}_2$$

σ is an element-wise activation function



Single Hidden Layer

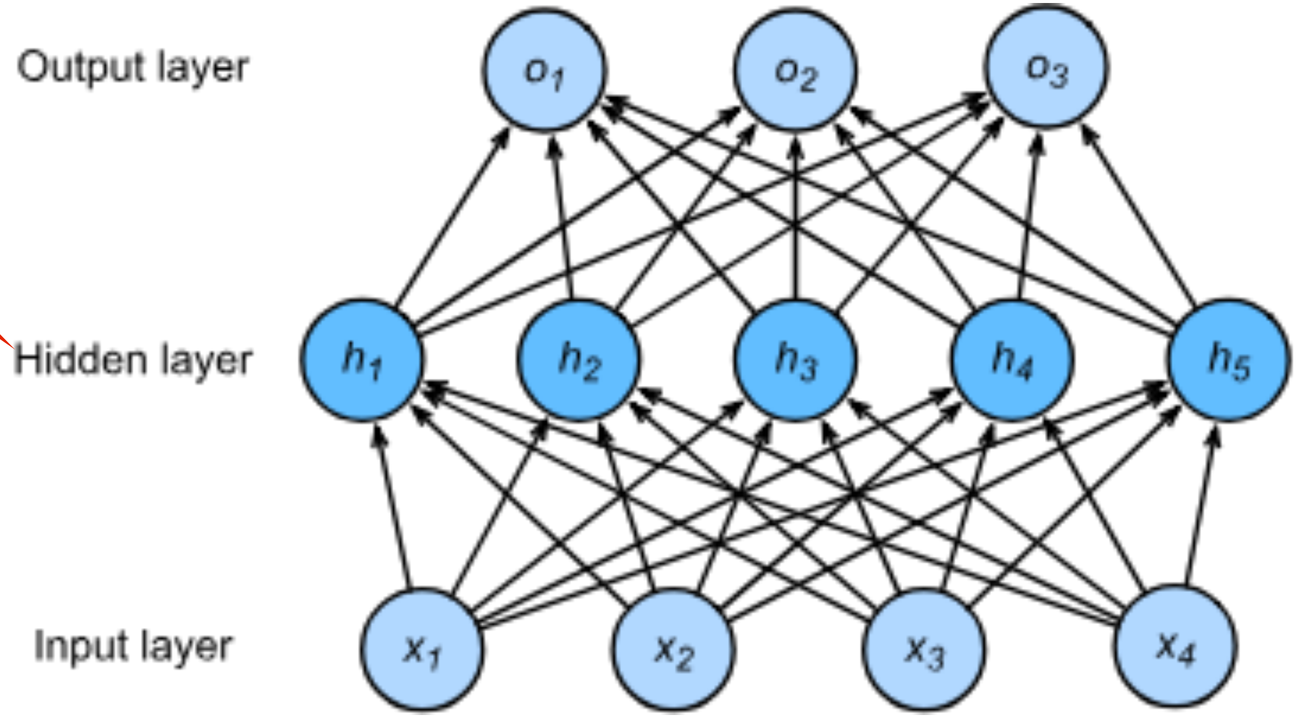
Why do we need an a nonlinear activation?

$$\mathbf{h} = \mathbf{W}_1 \mathbf{x} + \mathbf{b}_1$$

$$\mathbf{o} = \mathbf{W}_2^T \mathbf{h} + \mathbf{b}_2$$

hence $\mathbf{o} = \mathbf{W}_2^\top \mathbf{W}_1 \mathbf{x} + \mathbf{b}'$

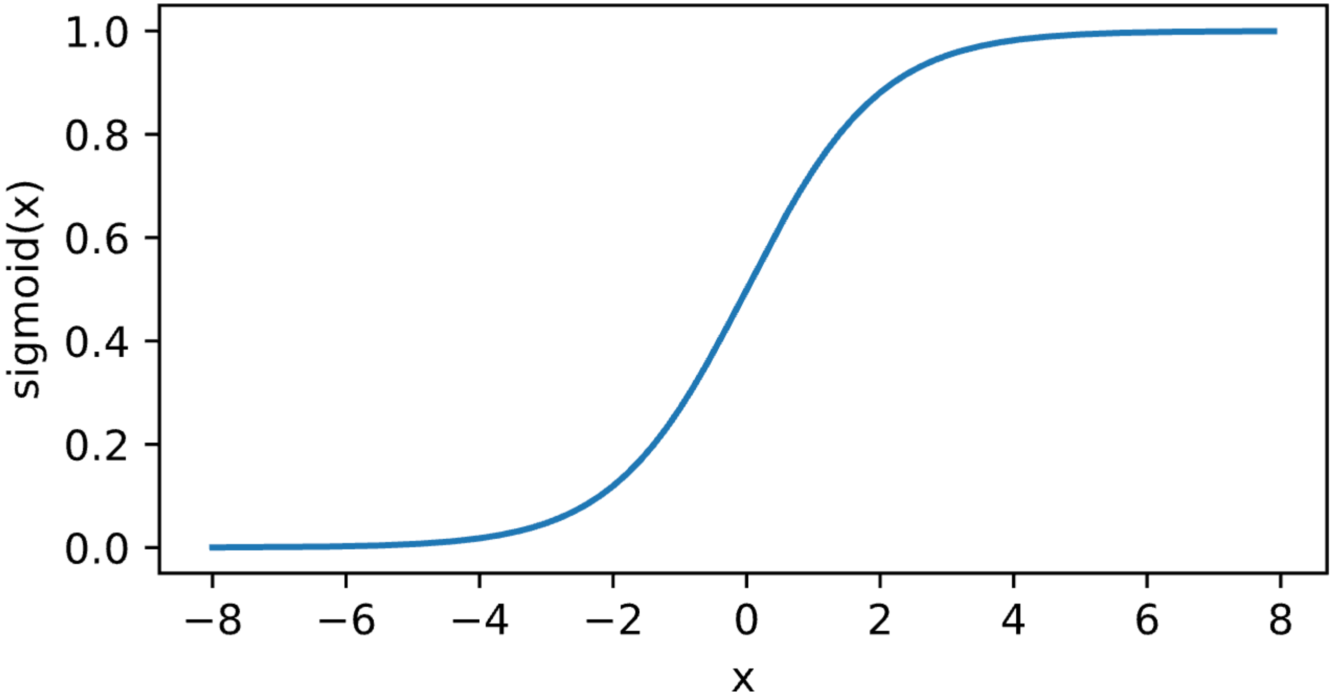
Linear ...



Sigmoid Activation

Map input into (0, 1), a soft version of $\sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$

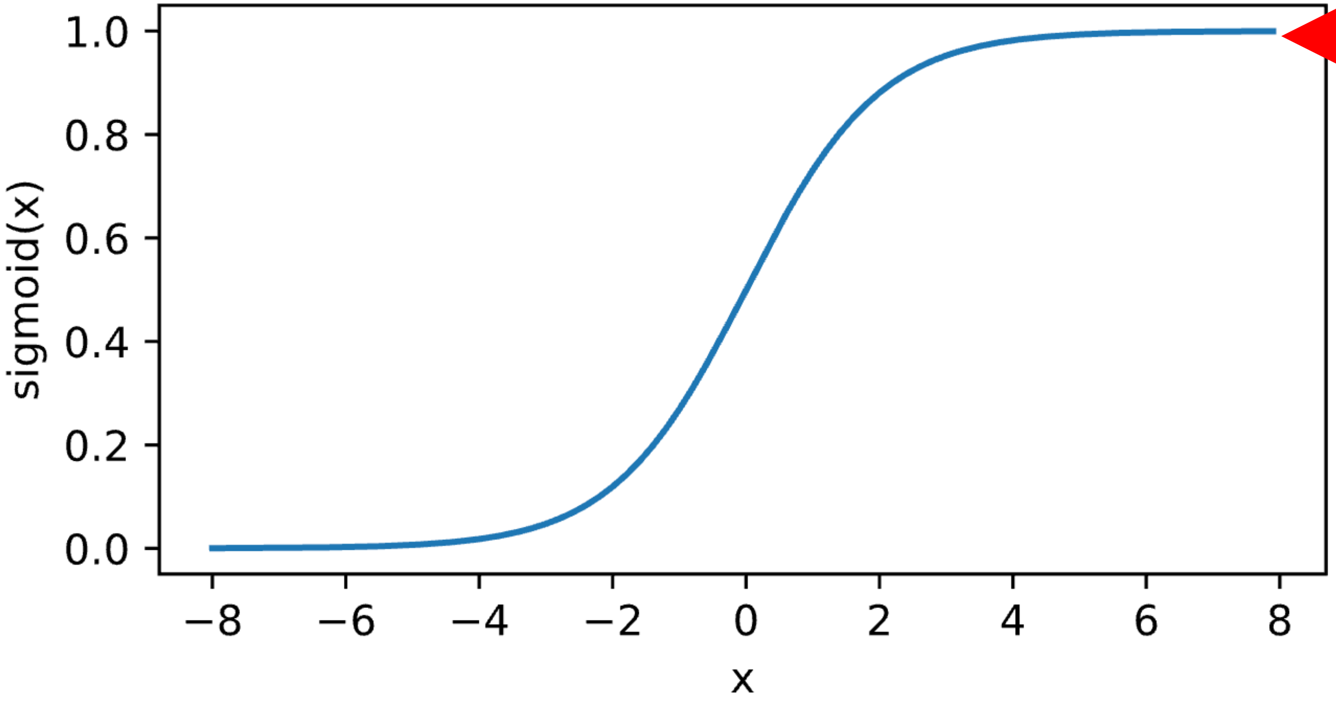
$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}$$



Sigmoid Activation

Map input into (0, 1), a soft version of $\sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$

$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}$$

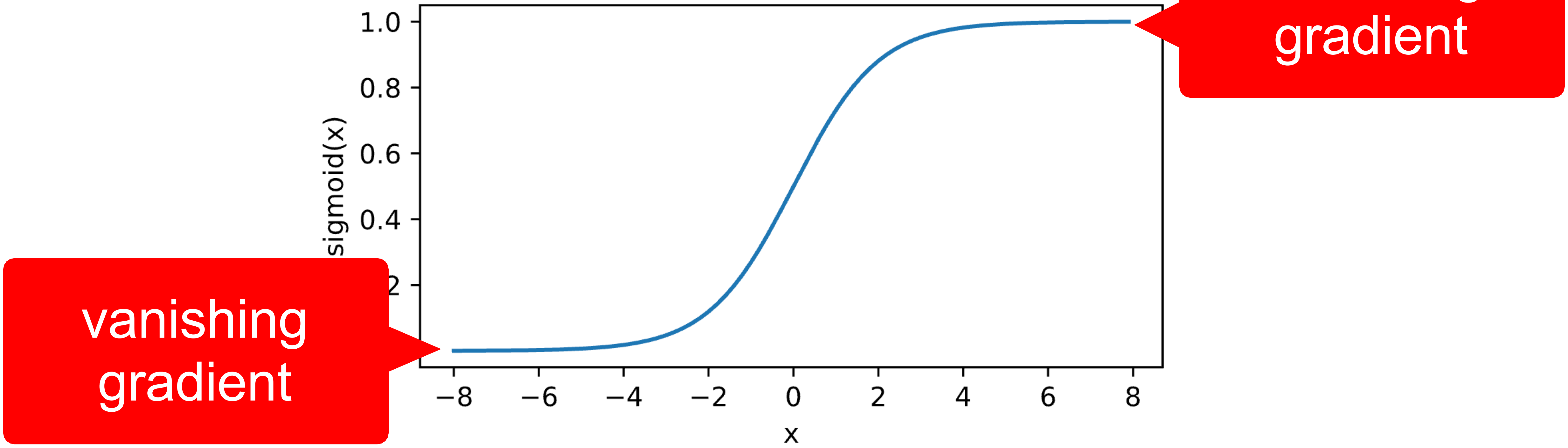


vanishing gradient

Sigmoid Activation

Map input into (0, 1), a soft version of $\sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$

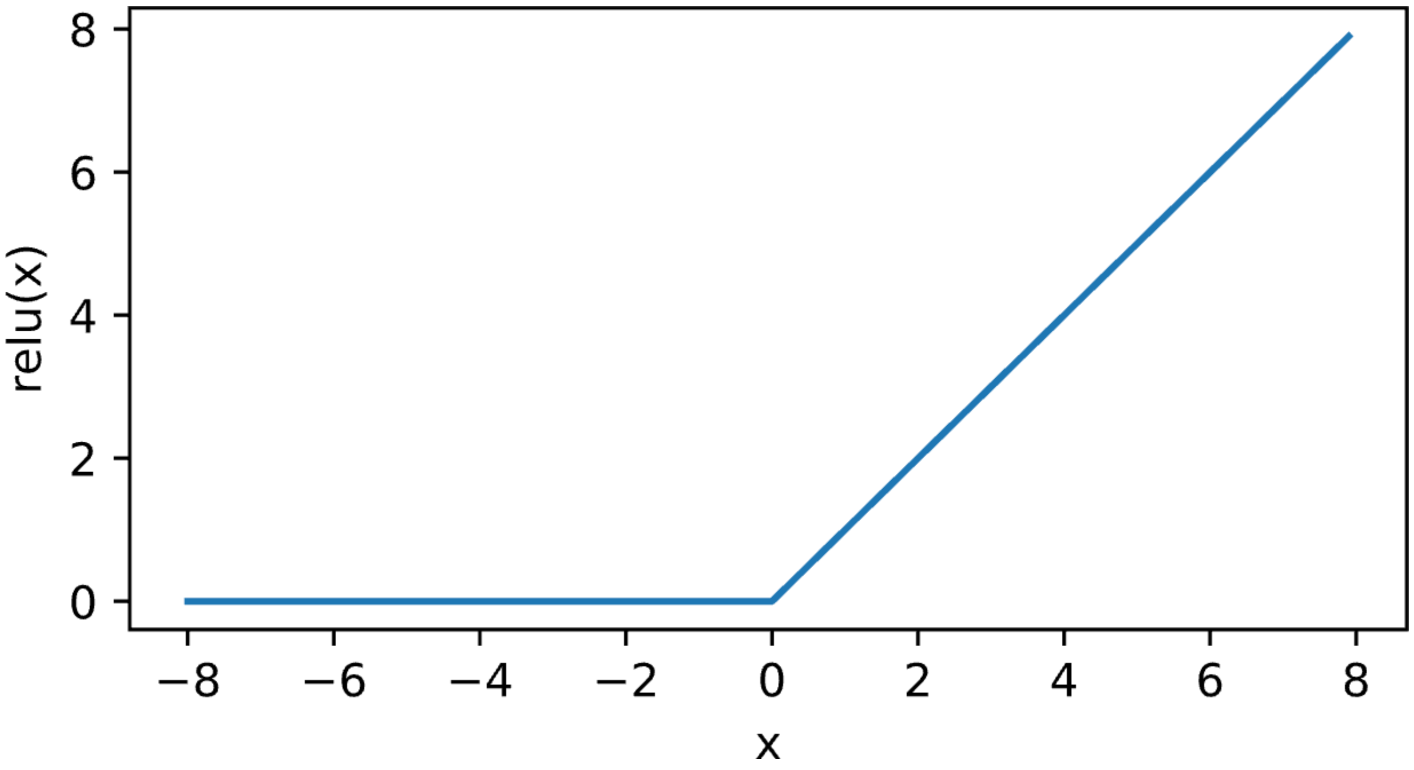
$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}$$



ReLU Activation

ReLU: rectified linear unit

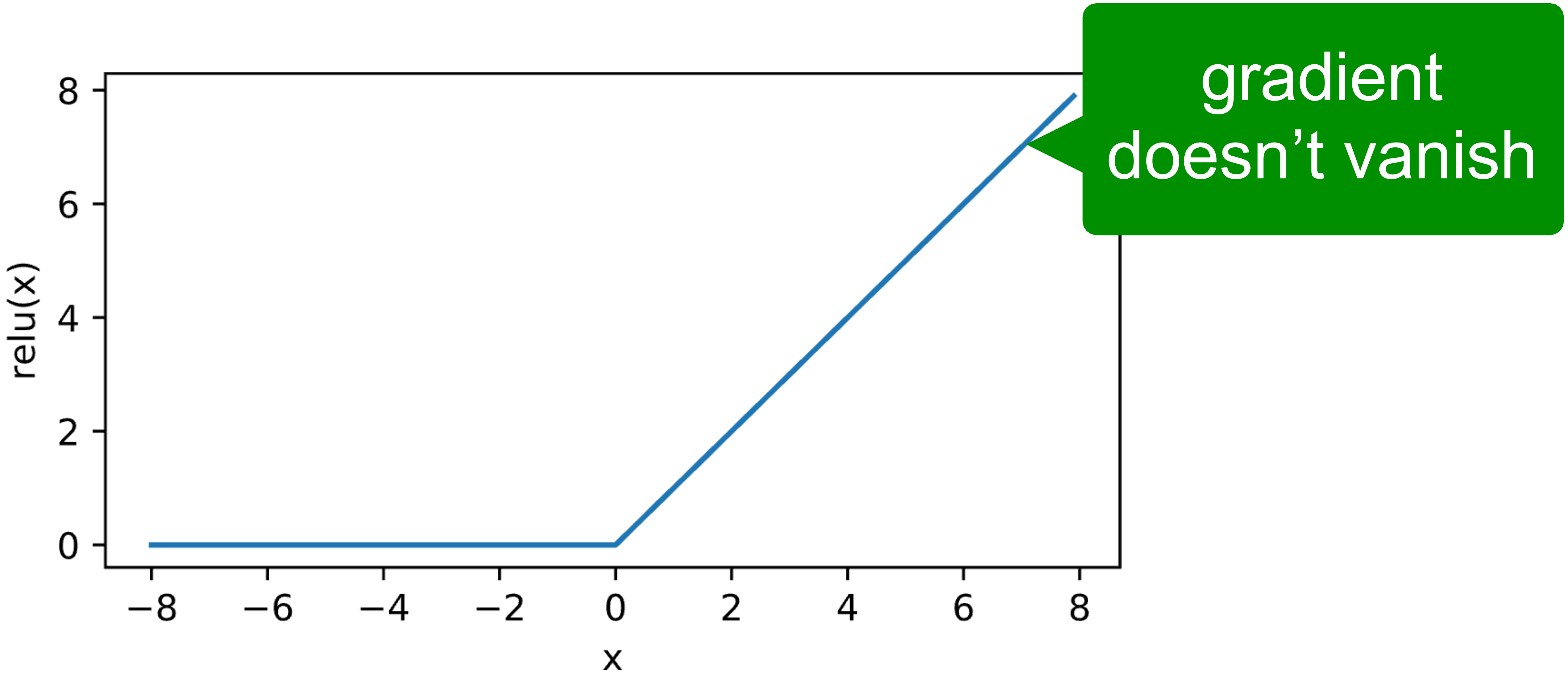
$$\mathrm{ReLU}(x) = \max(x, 0)$$



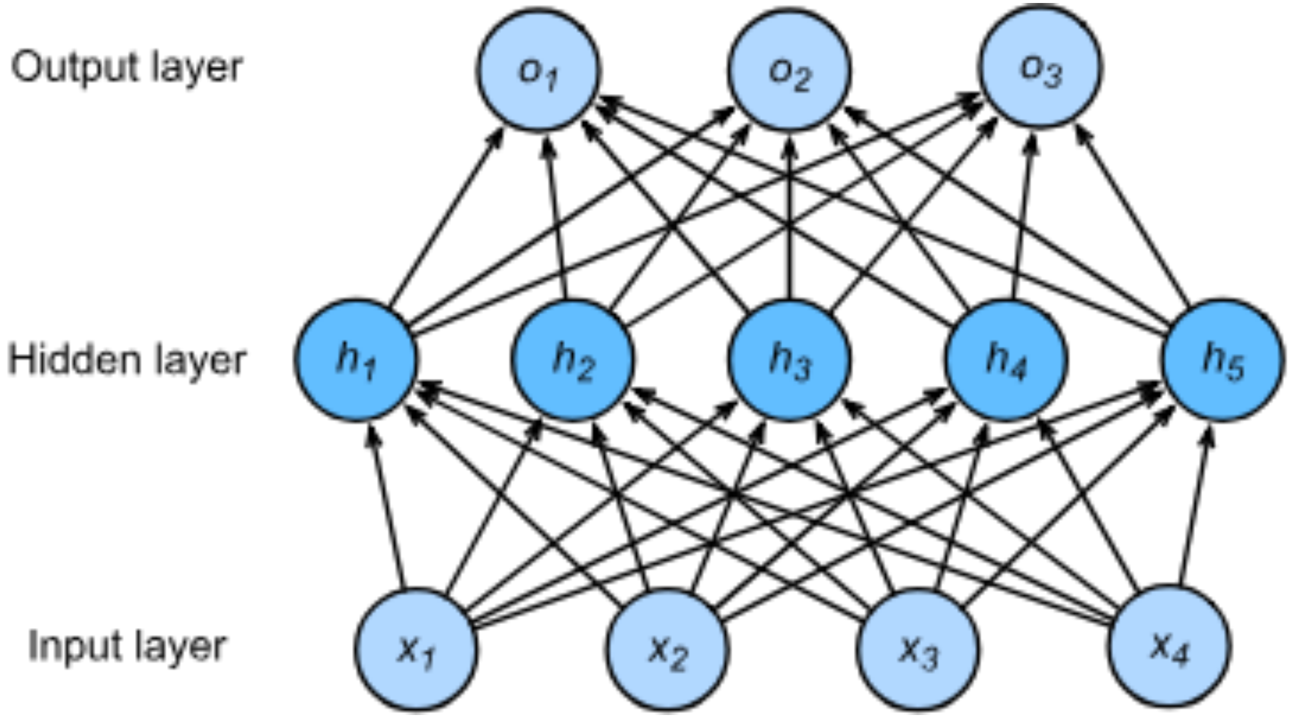
ReLU Activation

ReLU: rectified linear unit

$$\mathrm{ReLU}(x) = \max(x, 0)$$

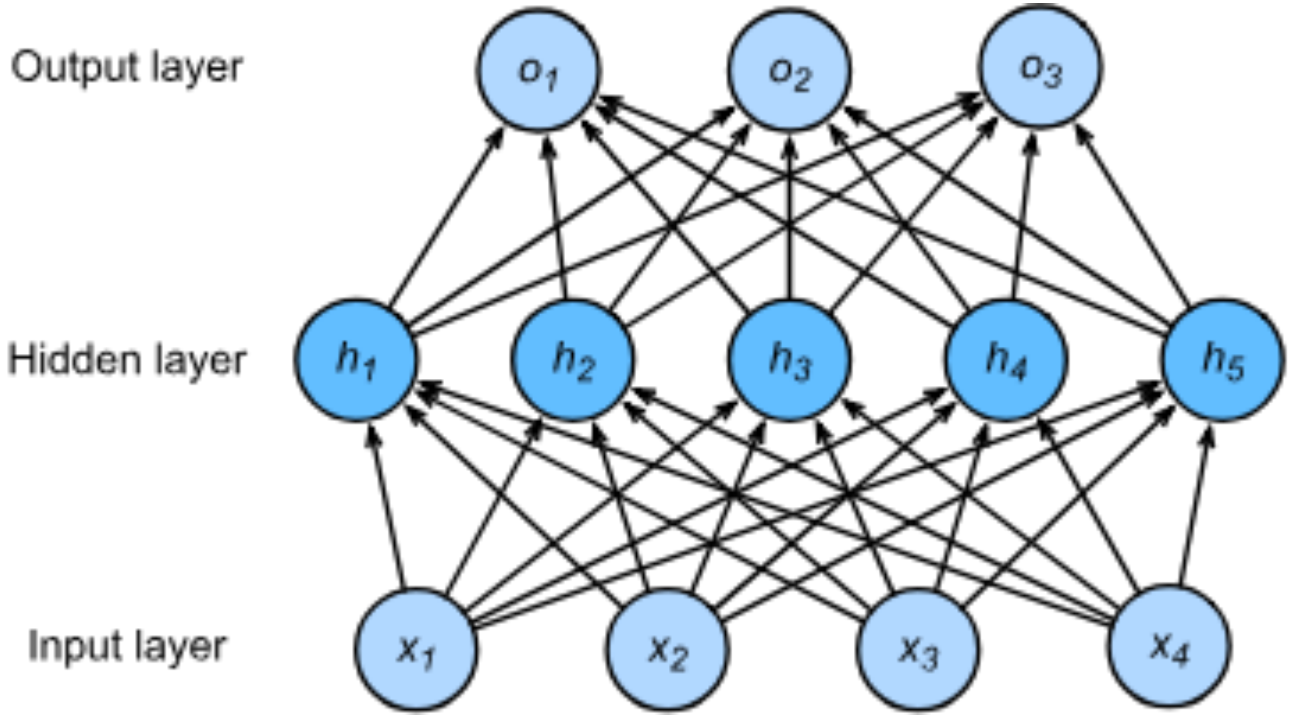


Multiclass Classification



Multiclass Classification

$$y_1, y_2, \ldots, y_k = \text{softmax}(o_1, o_2, \ldots, o_k)$$



Multiple Hidden Layers

$$\mathbf{h}_1 = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$

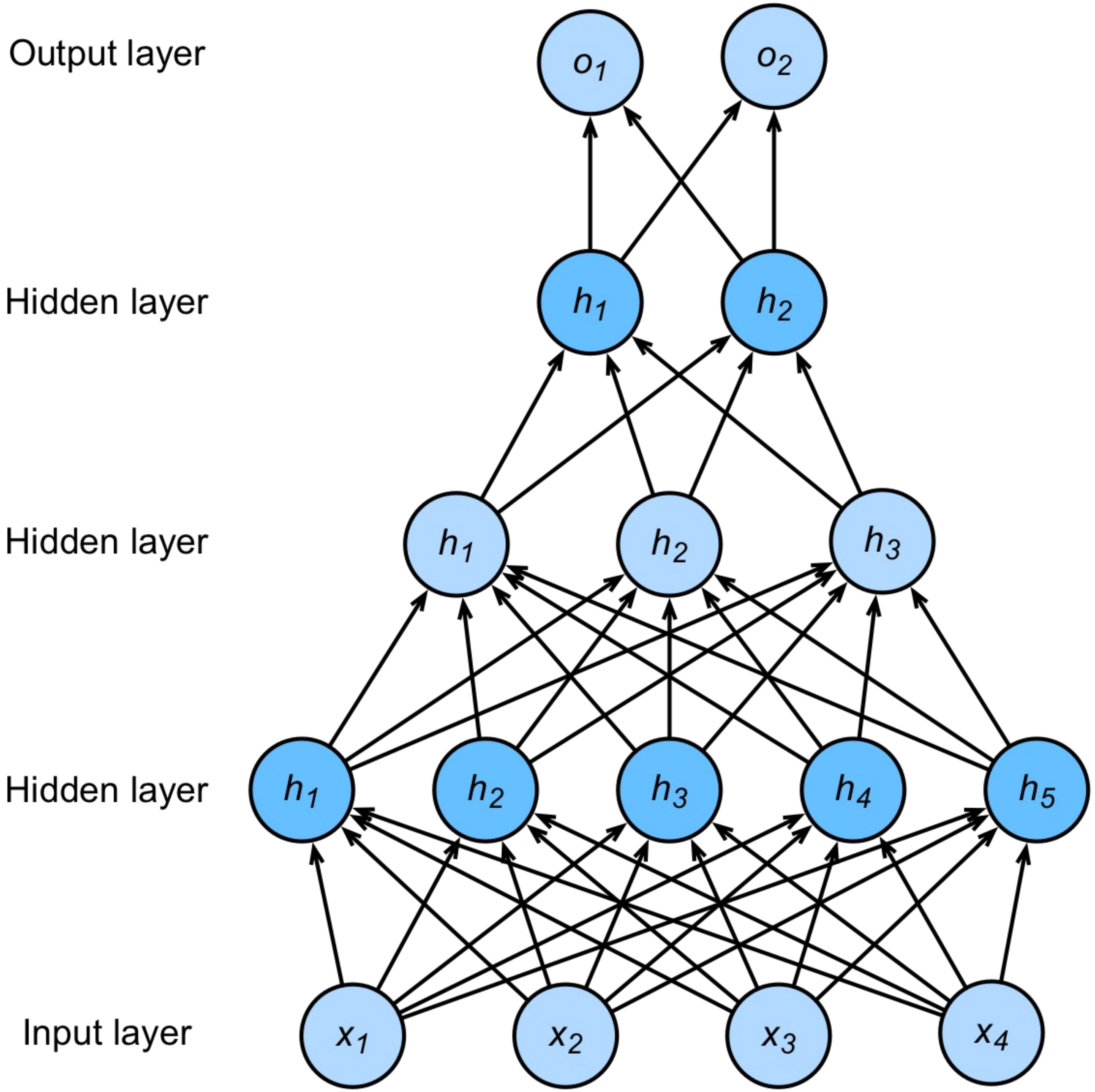
$$\mathbf{h}_2 = \sigma(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2)$$

$$\mathbf{h}_3 = \sigma(\mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3)$$

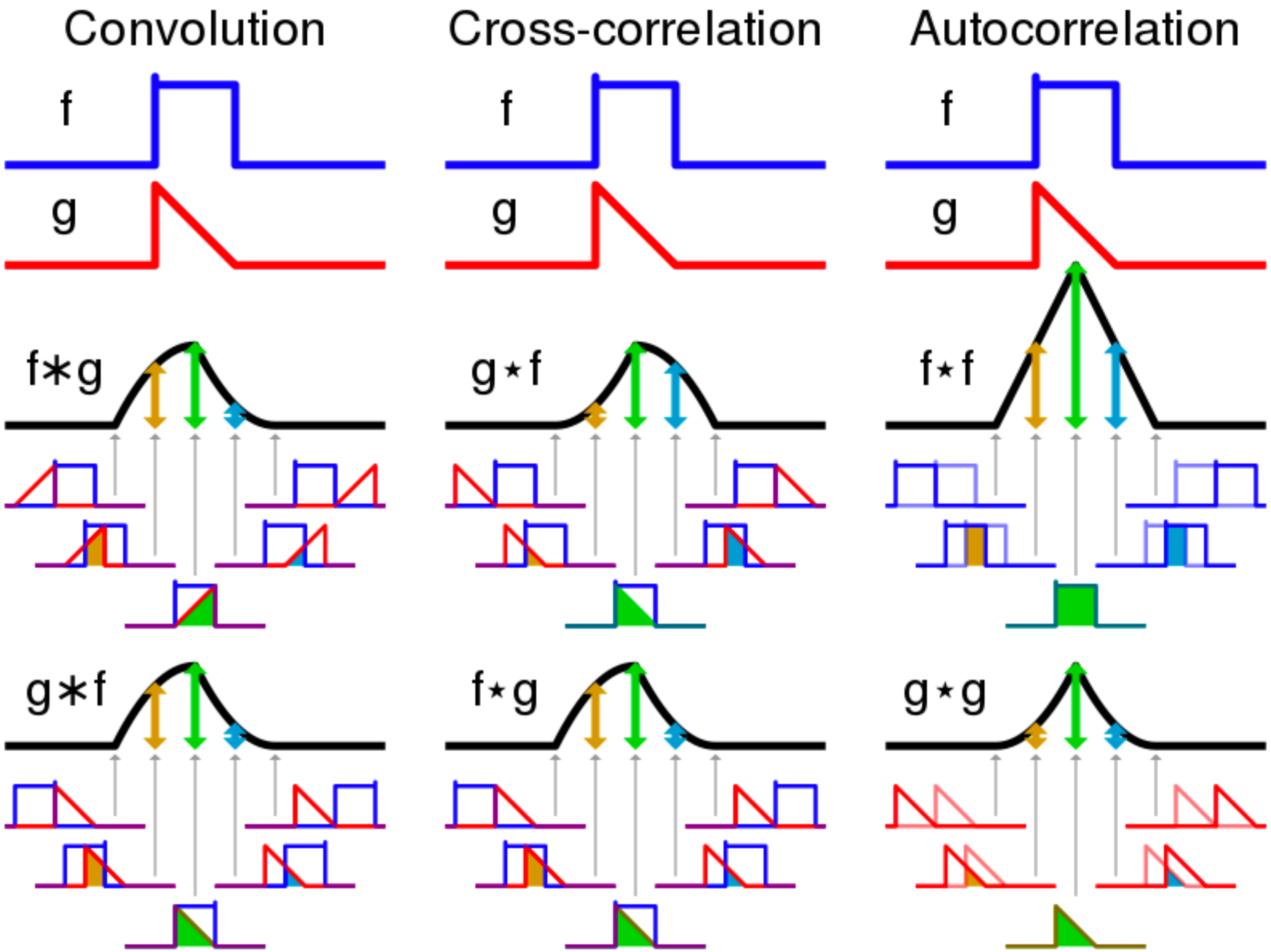
$$\mathbf{o} = \mathbf{W}_4\mathbf{h}_3 + \mathbf{b}_4$$

Hyperparameters

- # of hidden layers
- Hidden size for each layer



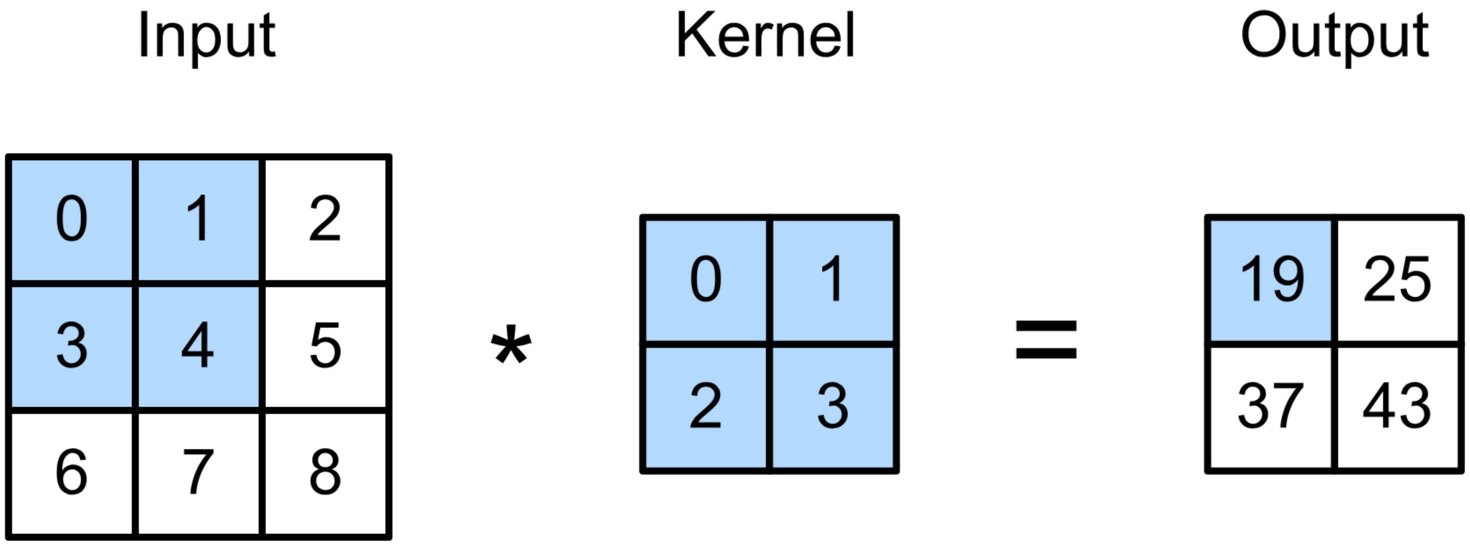
Convolution



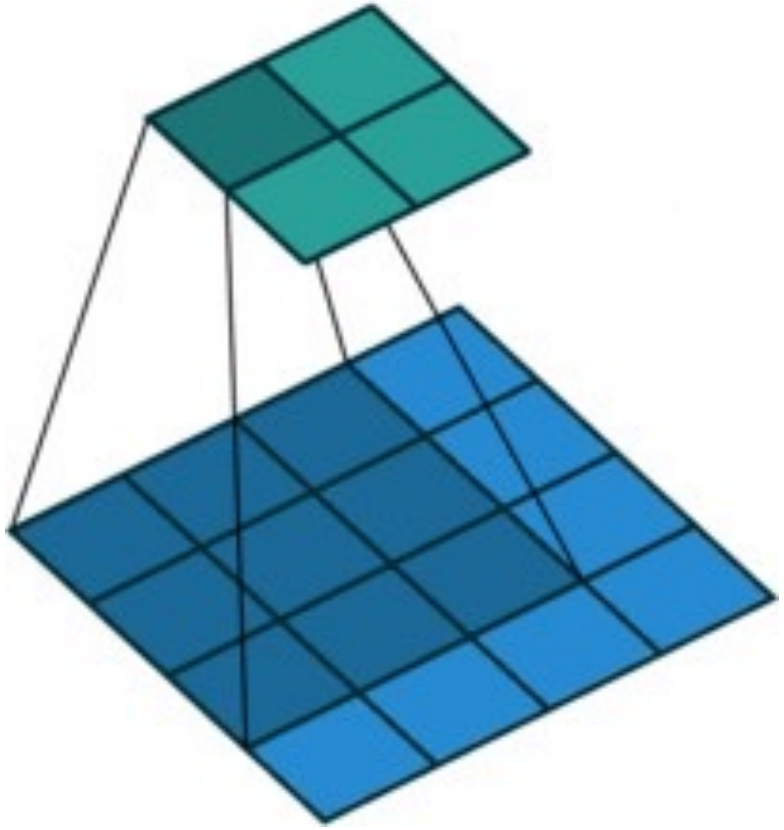
Two Principles

- Translation Invariance
- Locality

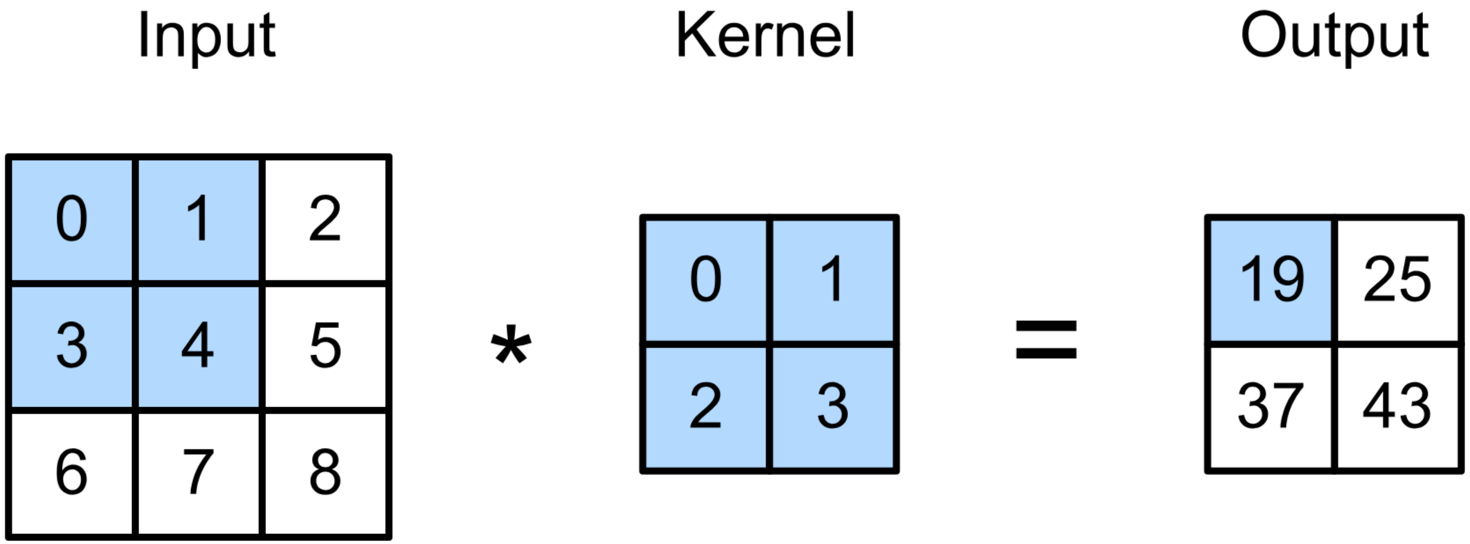
This yields Convolutions



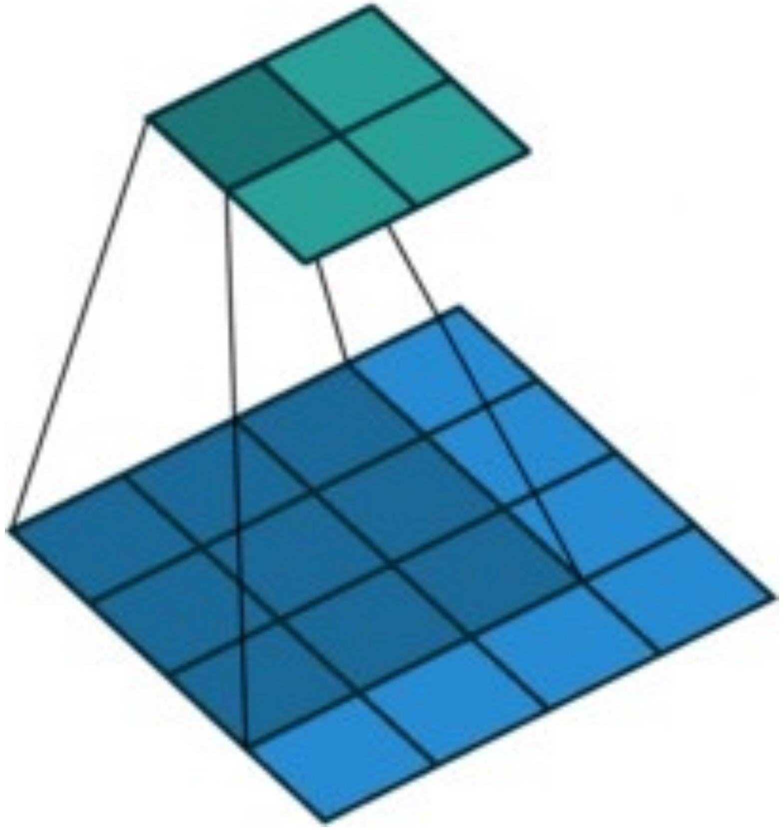
$$0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19,$$
$$1 \times 0 + 2 \times 1 + 4 \times 2 + 5 \times 3 = 25,$$
$$3 \times 0 + 4 \times 1 + 6 \times 2 + 7 \times 3 = 37,$$
$$4 \times 0 + 5 \times 1 + 7 \times 2 + 8 \times 3 = 43.$$



(vdumoulin@ Github)

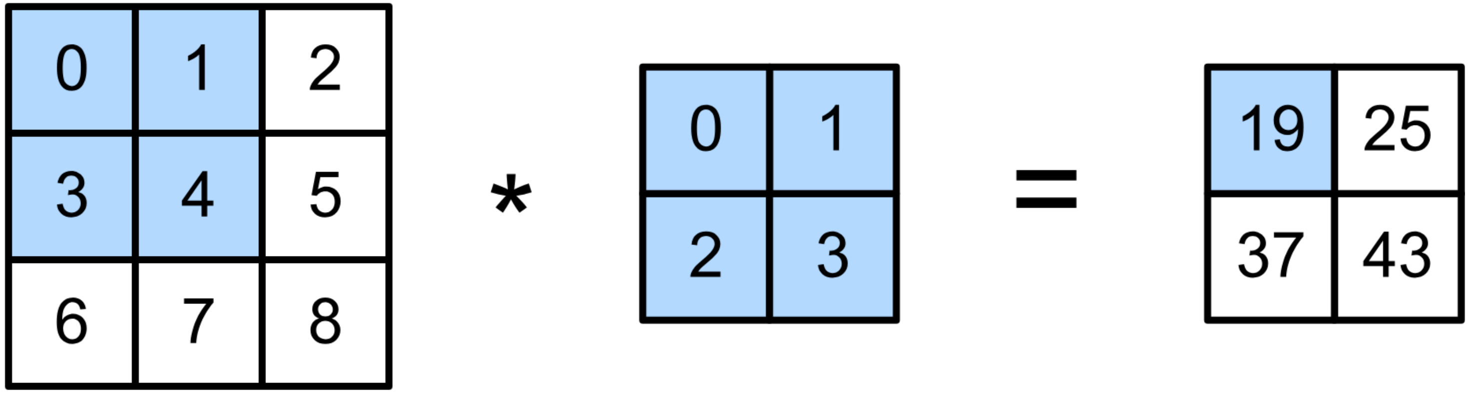


$$0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19,$$
$$1 \times 0 + 2 \times 1 + 4 \times 2 + 5 \times 3 = 25,$$
$$3 \times 0 + 4 \times 1 + 6 \times 2 + 7 \times 3 = 37,$$
$$4 \times 0 + 5 \times 1 + 7 \times 2 + 8 \times 3 = 43.$$



(vdumoulin@ Github)

2-D Convolution Layer



- $\mathbf{X}$: $n_h \times n_w$ input matrix
- $\mathbf{W}$: $k_h \times k_w$ kernel matrix
- b: scalar bias
- $\mathbf{Y}$: $(n_h - k_h + 1) \times (n_w - k_w + 1)$ output matrix

$$\mathbf{Y} = \mathbf{X} \star \mathbf{W} + b$$

- **W** and *b* are learnable parameters

Dropout Math (Training only)

- We want perturbation that keeps the mean unchanged

$$x_i' = \begin{cases} 0 & \text{with probablity } p \\ \frac{x_i}{1-p} & \text{otherwise} \end{cases}$$

$$\mathbf{E}[\mathbf{x}'] = \mathbf{x}$$

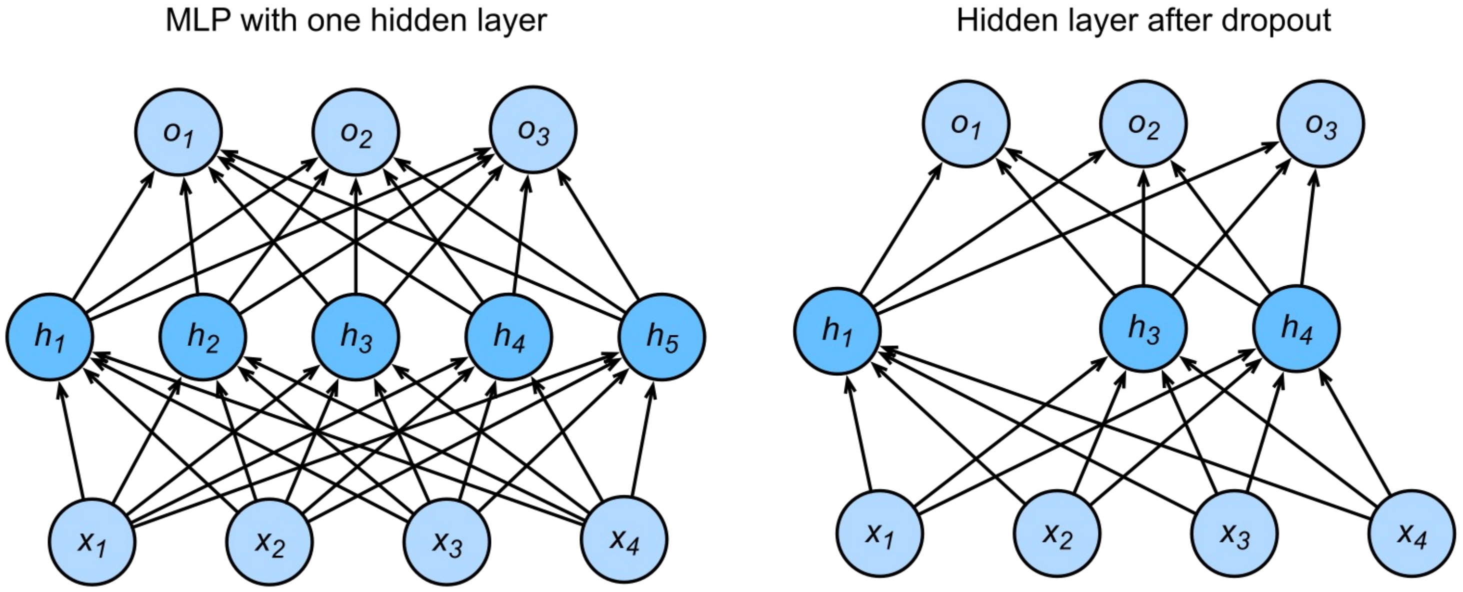
- Apply dropout to output of hidden fully-connected layers

$$\mathbf{h} = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$

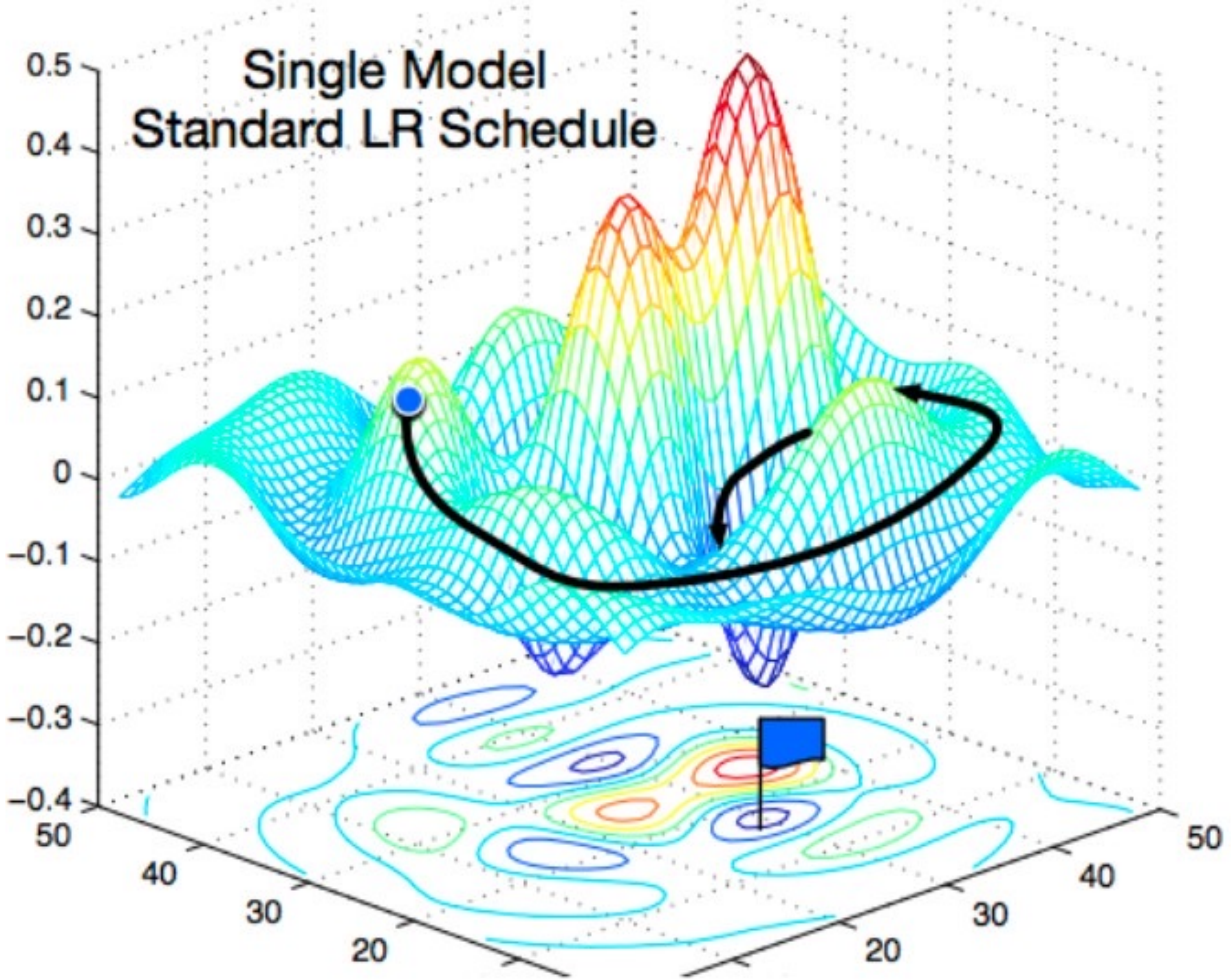
$$\mathbf{h}' = \text{dropout}(\mathbf{h})$$

$$\mathbf{o} = \mathbf{W}_2\mathbf{h}' + \mathbf{b}_2$$

$$\mathbf{y} = \text{softmax}(o)$$



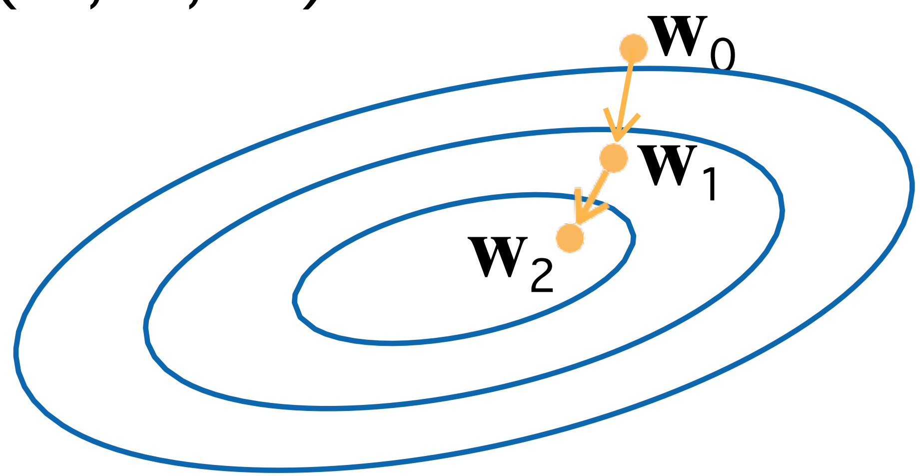
Basic Optimization



Gradient Descent

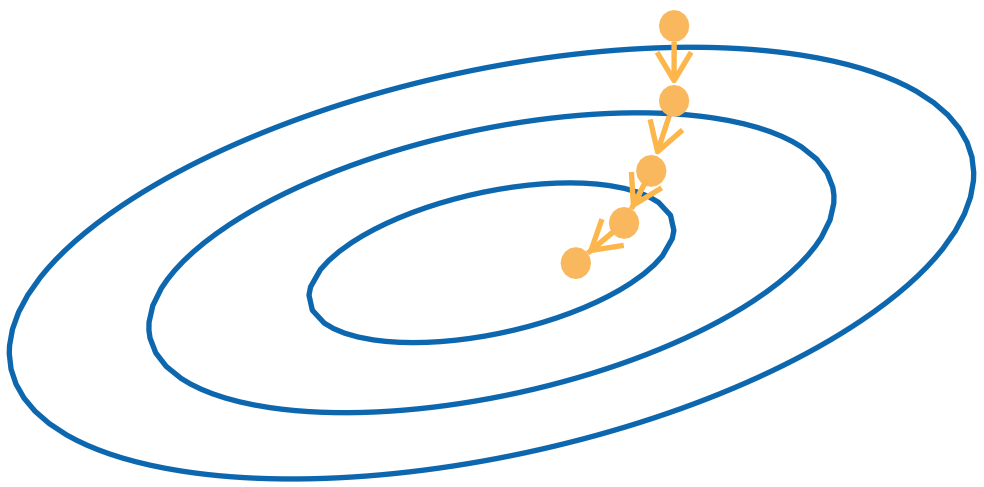
Objective function $L(X, Y, \mathbf{w}) = \sum_{i=1}^{m} l(y_i, f(\mathbf{x}_i), \mathbf{w})$

- Choose a starting point $\mathbf{w}_0$
- Gradient for descent direction $\partial_{\mathbf{w}} L(X, Y, \mathbf{w})$
- Update weights using learning rate
  $\mathbf{w}_t = \mathbf{w}_{t-1} - \eta \partial_{\mathbf{w}} L(X, Y, \mathbf{w}_{t-1})$

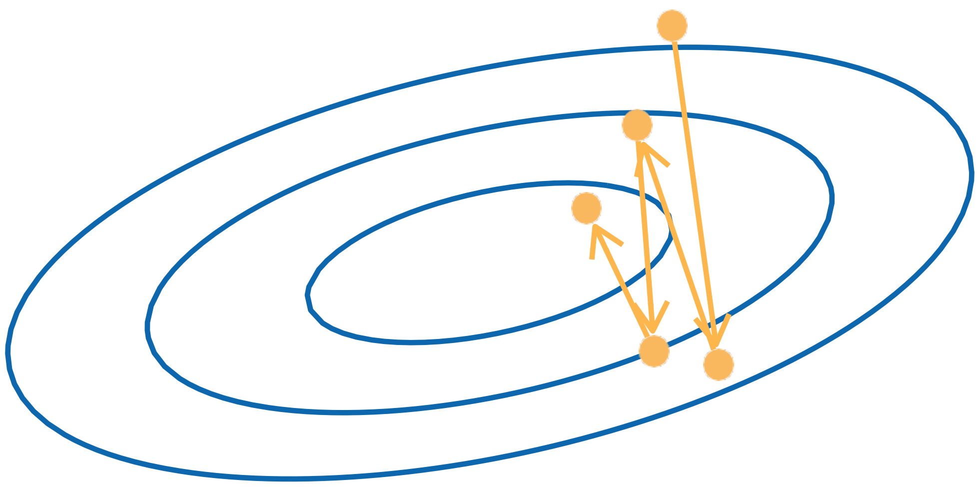


Choosing a Learning Rate

Too small



Too big



Minibatch Stochastic Gradient Descent (SGD)

Batch $\mathbf{w} \leftarrow \mathbf{w} - \frac{\eta}{m} \partial_{\mathbf{w}} L(X, Y, \mathbf{w})$

- Gradient over all data is too expensive (minutes to hours for DNNs)
- Not very informative if all training data is similar

Stochastic Gradient Descent $\mathbf{w} \leftarrow \mathbf{w} - \eta \partial_{\mathbf{w}} l(\mathbf{x}_i, y_i, \mathbf{w})$

- Pick one observation at a time and update
- Noisy and inefficient (GPUs love lots of data)

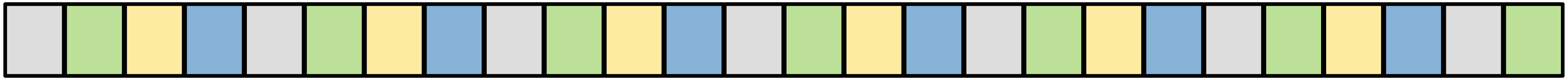
Minibatch SGD $\mathbf{w} \leftarrow \mathbf{w} - \frac{\eta}{b} \partial_{\mathbf{w}} \sum_{i \in B_j} l(\mathbf{x}_i, y_i, \mathbf{w})$

- Efficient (computationally and statistically)

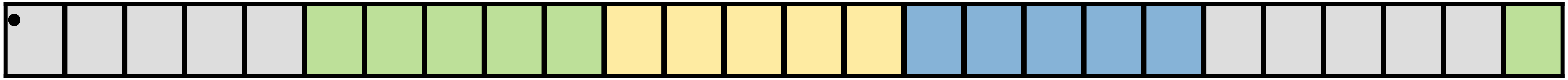
Minibatch Stochastic Gradient Descent (SGD)

Batch $\mathbf{w} \leftarrow \mathbf{w} - \frac{\eta}{m}\partial_{\mathbf{w}} L(X, Y, \mathbf{w})$

Stochastic Gradient Descent $\mathbf{w} \leftarrow \mathbf{w} - \eta \partial_{\mathbf{w}} l(\mathbf{x}_i, y_i, \mathbf{w})$



Minibatch SGD $\mathbf{w} \leftarrow \mathbf{w} - \frac{\eta}{b}\partial_{\mathbf{w}} \sum_{i \in B_j} l(\mathbf{x}_i, y_i, \mathbf{w})$



Choosing a Batch Size

Too small

- Workload is too small,
- difficult to fully utilize computation resources

Too big

- Memory issue
- Waste computation, e.g. when all x_i are identical