University of Amsterdam
Amsterdam School of Economics

Machine Learning for Econometrics, 2022

Lecture 2: Deep Learning

Yi He

November 8, 2022

Plan for Today

1. Feedforward Networks
2. Understanding Neural Networks
3. Convolutional Neural Networks
4. Training Neural Networks

Feedforward Networks

Why Nonlinear Models: An Example

- Two independent economic variables $X = (X_1, X_2) \in \{0, 1\}^2$
- $\mathbb{P}[X_j = 1] = \mathbb{P}[X_j = 0] = 1/2$, $j = 1, 2$
- XOR regression function $\mu(x) = \mathbb{E}[Y|X = x] = \mathbb{1}[x_1 \neq x_2]$

Consider the linear prediction rule

$$f(x; \theta) = \beta_1 x_1 + \beta_2 x_2 + \beta_0, \quad \theta = (\beta_1, \beta_2, \beta_0)$$

We can decompose the population squared risk

$$\begin{aligned}
&\mathbb{E}[(f(X;\theta) - Y)^2] \\
=&\mathbb{E}[(f(X;\theta) - \mu(X) + \mu(X) - Y)^2] \\
=&\mathbb{E}[(f(X;\theta) - \mu(X))^2] + 2\mathbb{E}[(f(X;\theta) - \mu(X))\underbrace{\mathbb{E}[\mu(X) - Y|X]}_{=\mu(X)-\mu(X)=0}] \\
&+ \underbrace{\mathbb{E}[(Y - \mu(X))^2]}_{\text{not depending on } \theta}.
\end{aligned}$$

Minimizing the first term

$$
\begin{aligned}
&\mathbb{E}[(f(X;\theta)-\mu(X))^2] \\
=&\frac{1}{4}\Big\{(\beta_0-0)^2+(\beta_0+\beta_1-1)^2 \\
&+(\beta_0+\beta_2-1)^2+(\beta_0+\beta_1+\beta_2-0)^2\Big\}
\end{aligned}
$$

yields the solution $(\beta_0, \beta_1, \beta_2) = (1/2, 0, 0)$, corresponding to the ideal linear prediction rule $f^*(x) = 1/2 \neq \mu(x)$.

We need a nonlinear model to express the XOR function.

Shallow Feedforward Networks: Regression

Consider functions $f : \mathbb{R}^d \to \mathbb{R}$ of the following form:

$$f(x) = \sum_{m=1}^{M} \underbrace{w_{1,m}^{(2)}}_{\text{weight}} z_m + \underbrace{b_1^{(2)}}_{\text{bias}},$$

$$z_m = \sigma\left(a_m^{(1)}\right), \; a_m^{(1)} = \sum_{j=1}^{d} \underbrace{w_{m,j}^{(1)}}_{\text{weight}} x_j + \underbrace{b_m^{(1)}}_{\text{bias}}.$$

where σ is some nonlinear activation function such as

$$\text{ReLU}(a) = \max\{0, a\}.$$

ReLU is called Rectified Linear Unit activation function.

In matrix form,

$$\begin{bmatrix} a_1^{(1)} \\ \vdots \\ a_M^{(1)} \end{bmatrix} = W^{(1)} \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix} + b^{(1)}, \; W^{(1)} = \left(w_{m,j}^{(1)}\right), \; b^{(1)} = \left(b_m^{(1)}\right)$$

$$\begin{bmatrix} z_1^{(1)} \\ \vdots \\ z_M^{(1)} \end{bmatrix} = \sigma\left(\begin{bmatrix} a_1^{(1)} \\ \vdots \\ a_M^{(1)} \end{bmatrix}\right) \overset{\text{elementwise}}{=} \begin{bmatrix} \sigma(a_1^{(1)}) \\ \vdots \\ \sigma(a_M^{(1)}) \end{bmatrix}$$

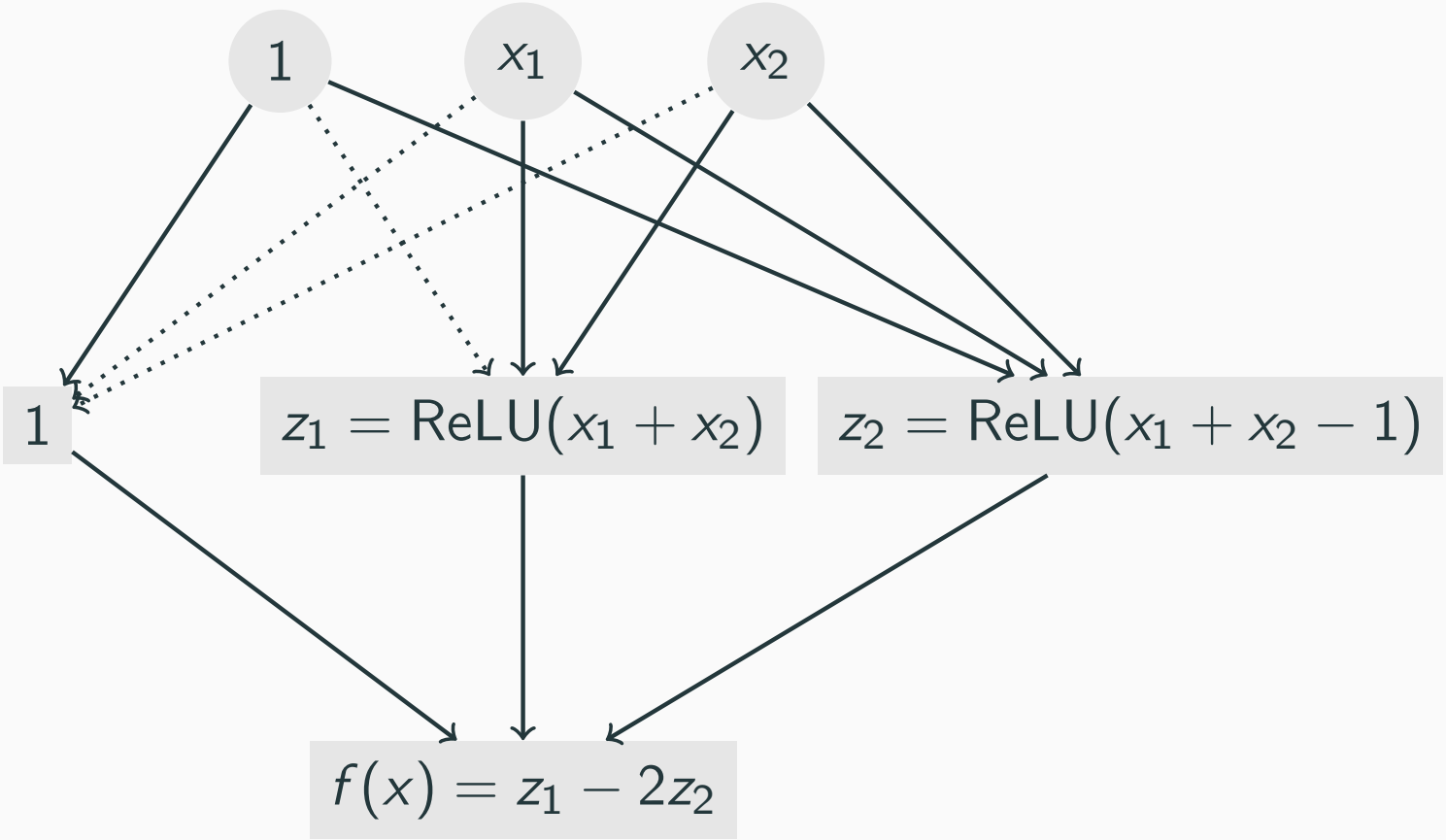
$$f(x) = W^{(2)} \begin{bmatrix} z_1^{(1)} \\ \vdots \\ z_M^{(1)} \end{bmatrix} + b^{(2)}, \; W^{(2)} = \left(w_{k,m}^{(2)}\right), \; b^{(2)} = \left(b_k^{(2)}\right), \; k = 1$$

To summarize: for $x = (x_1, \ldots, x_d)^T$,

$$f(x) = W^{(2)}\sigma(W^{(1)}x + b^{(1)}) + b^{(2)}$$

where σ is applied elementwisely.

Expressing the XOR Function



Shallow Feedforward Networks: Multiple-Output

For K-class classification, we need to estimate a multivariate regression function after one-hot encoding and softmax transformation (Lecture 1):

$$f(x) = \boldsymbol{\sigma}\left(W^{(2)}\sigma(W^{(1)}x + b^{(1)}) + b^{(2)}\right)$$

where $\boldsymbol{\sigma}$ outside the brackets denotes the softmax function.

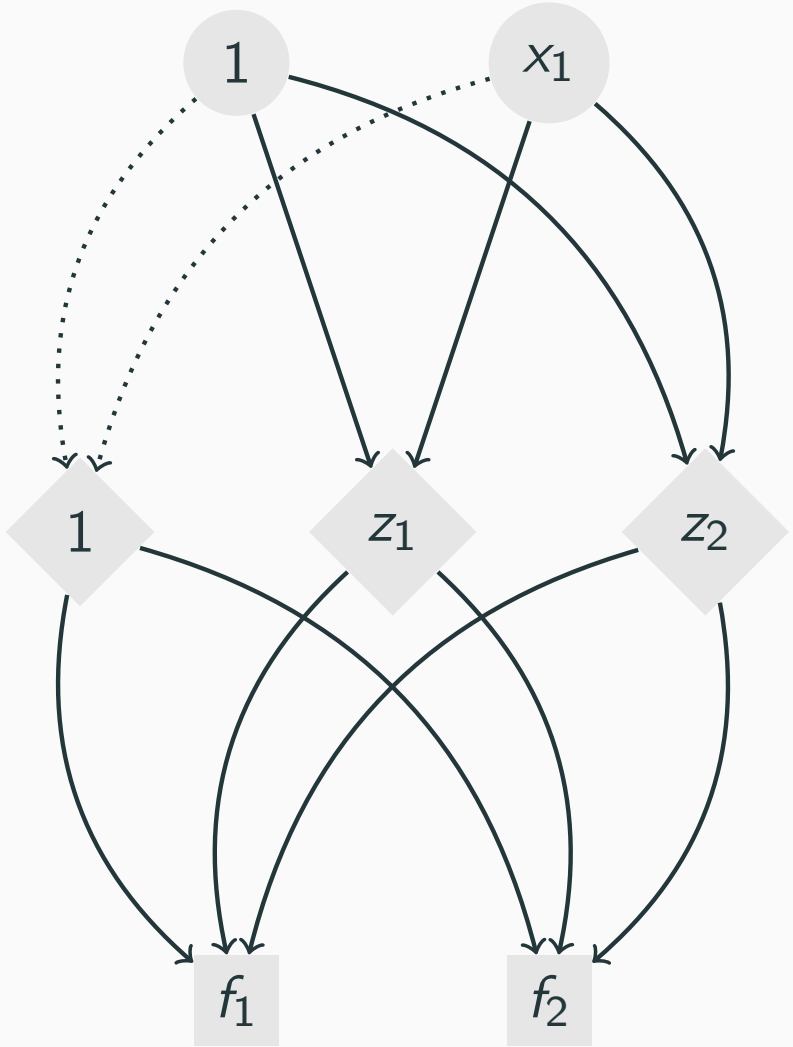
- Add more rows into $W^{(2)} \in \mathbb{R}^{K\times M}$ and $b^{(2)} \in \mathbb{R}^K$
- If softmax transformation is not needed (for multiple-output regression): use $\boldsymbol{\sigma}$ as the identity function instead
- In general, we may also take $\boldsymbol{\sigma}$ as another activation

An Example Neural Net With $K = 2$ Outputs

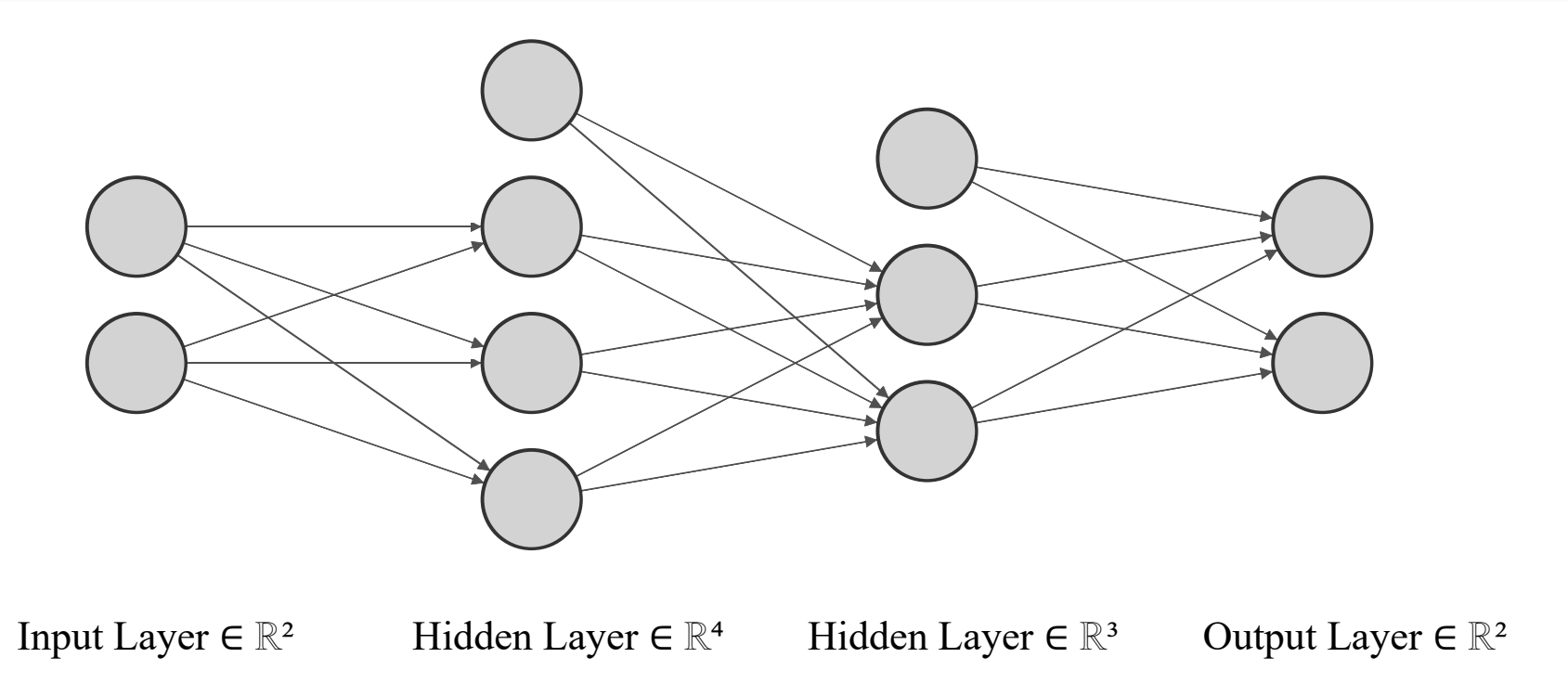
Input Layer

Hidden Layer

Output Layer



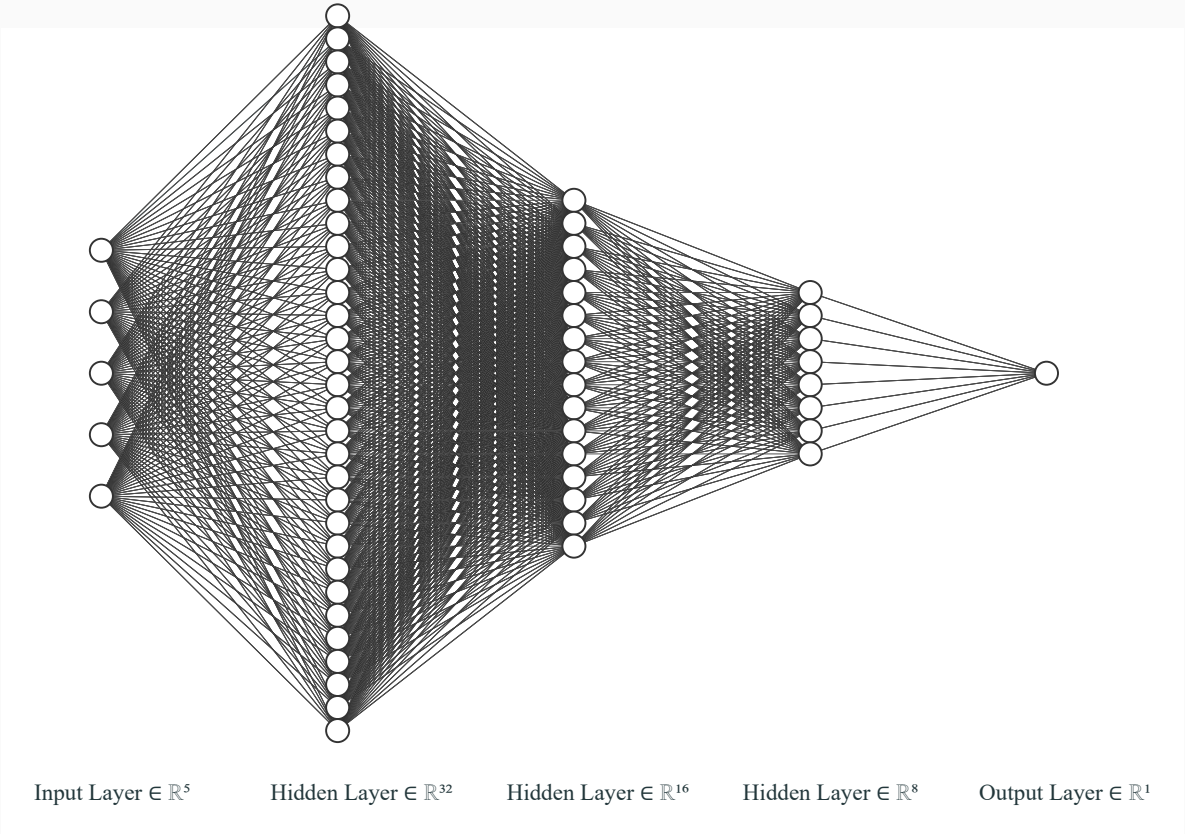
Adding Hidden Layers



- Input $z^{(0)} = (x_1, \ldots, x_d)^T$, $M_0 = d$
- h-th Hidden layer for $h = 1, \ldots, D-1$:

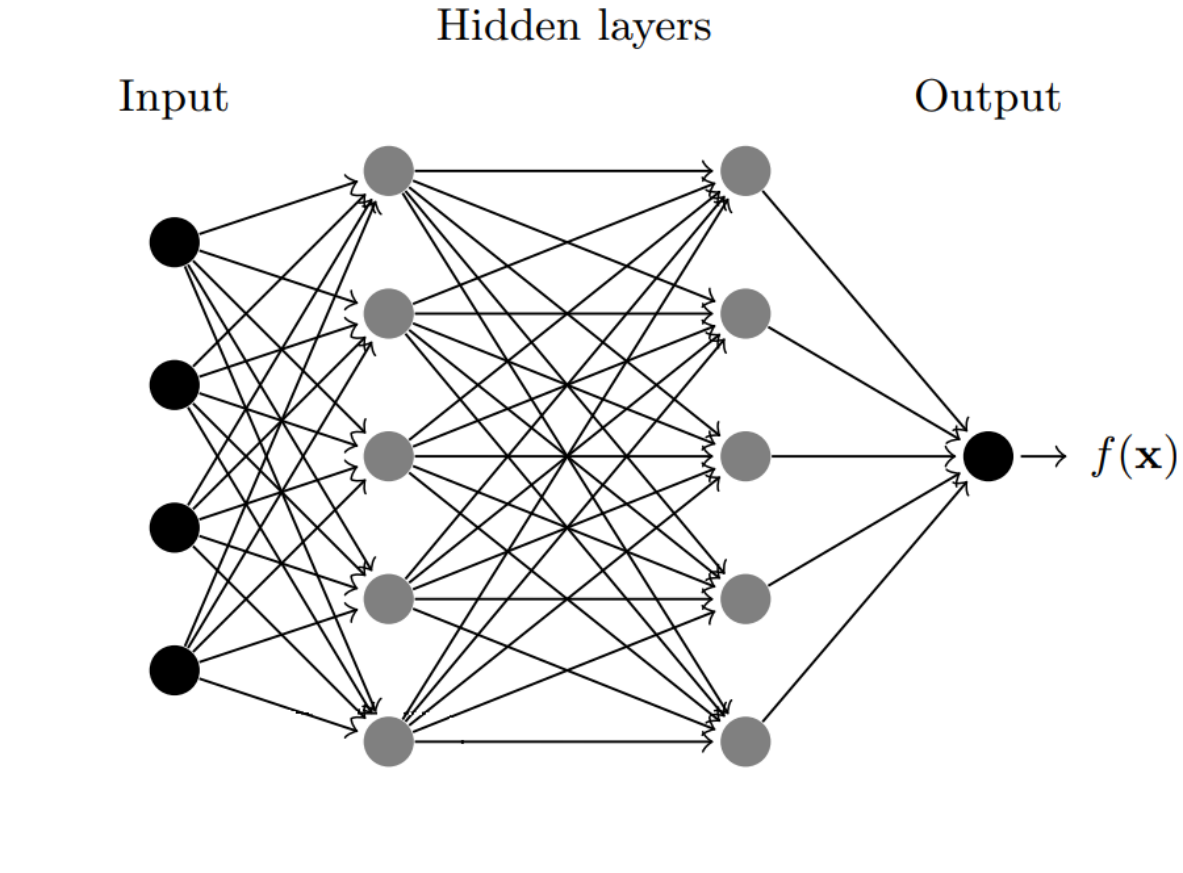
  $z^{(h)} = \sigma(a^{(h)}), \quad a^{(h)} = W^{(h)} z^{(h-1)} + b^{(h)}, \quad W^{(h)} \in \mathbb{R}^{M_h \times M_{h-1}}$
- Output layer: $f(x) = \boldsymbol{\sigma}(a^{(D)})$, $M_D = K$.

Geometric Pyramid Rule: Masters (1993)



Three hidden layers $M_1 = 32$, $M_2 = 16$, $M_3 = 8$. Works well in empirical asset pricing; see Gu et al. (2020, RFS).

Fully-Connected Networks



- $M_1 = \ldots = M_{D-1} = M$
- Easy to increase depth compared with the geometric pyradmid rule

Understanding Neural Networks

Sigmoid VS ReLU Activation

The most traditional choice of activation function for shallow neural networks is the sigmoid or the hyperbolic tangent function:

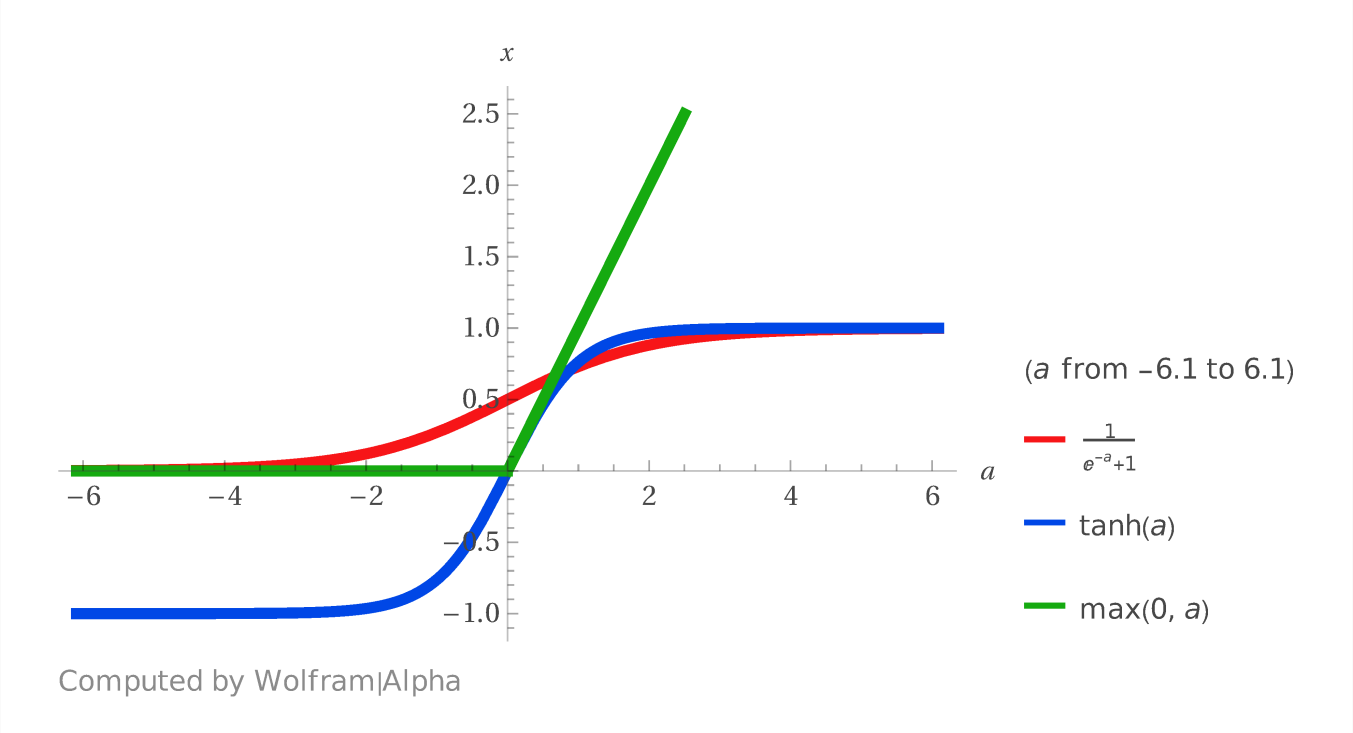
$$\text{Sigmoid}(a) = \frac{1}{1+\exp(-a)}$$

$$\tanh(a) = \frac{\exp(a)-\exp(-a)}{\exp(a)+\exp(-a)}.$$

In deep learning with (relatively) large D, however, it is often better to use the ReLU activation

$$\text{ReLU}(a) = \max\{0, a\}.$$

Plotting the Activation Function



The hyperbolic tangent function is a mirror transformation of the sigmoid via

$$\tanh(a) = 2 \times \text{Sigmoid}(2a) - 1,$$

so that it is centered around zero, that is, $\tanh(0) = 0$

Why ReLU Function

The ReLU function enjoys the so-called projection property:

$$\text{ReLU}(\ \text{ReLU}(a)\ ) = \text{ReLU}(a).$$

Activating many times does not change the signal, which can pass through several layers without change:

$$\underbrace{\text{ReLU}(\text{ReLU}(\ldots\text{ReLU}}_{k \text{ times}}(a))) = \text{ReLU}(a).$$

In contrast, applying the sigmoid function too many times through layers loses the signal:

$$\underbrace{\text{Sigmoid}(\text{Sigmoid}(\ldots\text{Sigmoid}}_{k \text{ times}}(a))) \to 0.659\ldots, \text{ as } k \to \infty.$$

The limit is a constant, making the input irrelevant.

Universal Approximation

For any sufficiently smooth function μ on a compact set with finitely many discontinuities,...

there exists a ReLU feedforward network f that can approximate it arbitrarily well if the width M and depth D are sufficiently large.

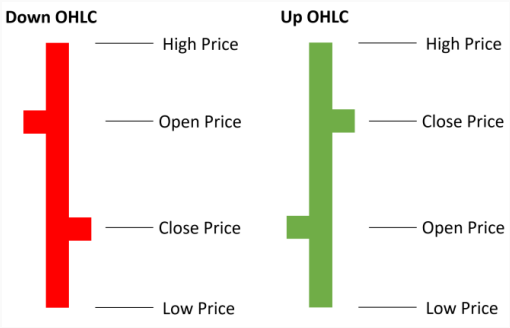
- The universal approximation property, however, does not tell precisely how many hidden units are required.
- Generally speaking, there is a trade-off between M and D: allowing large D may reduce the number of hidden units needed for approximations dramatically

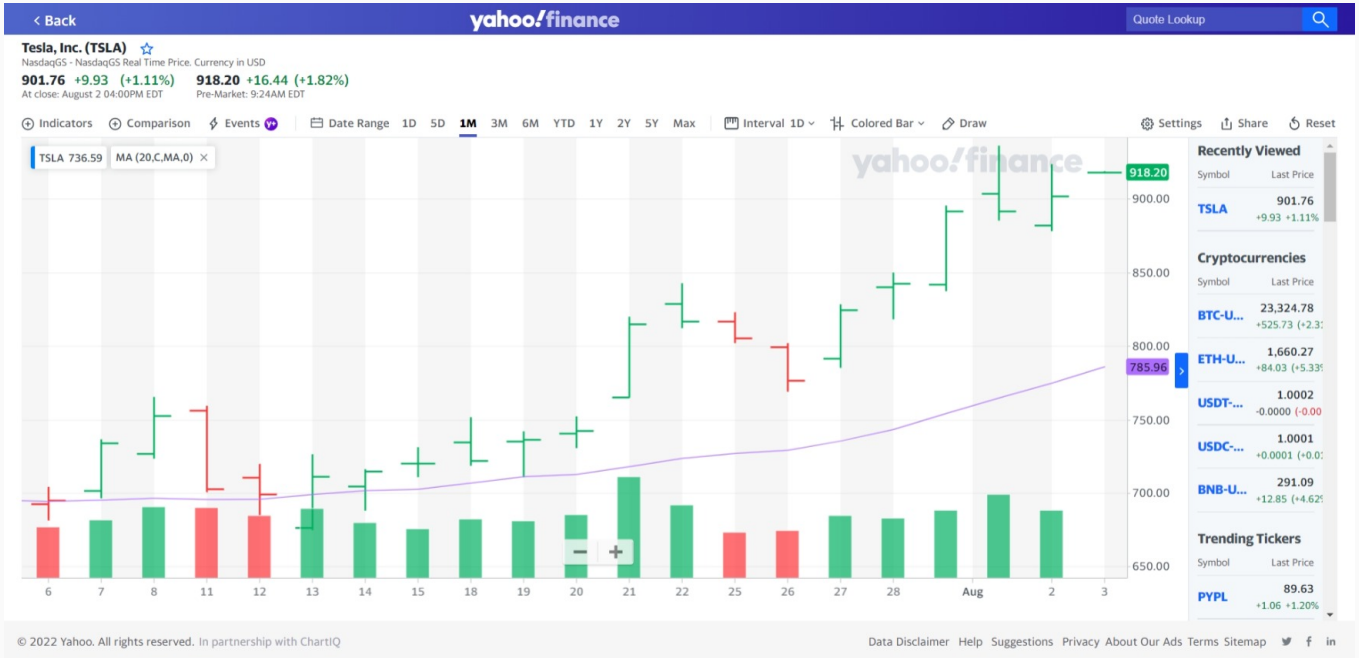
How Deep Should It Be?

- In finance applications, the depth D is usually between 3 and 6 for monthly equity data
- In theory, the depth D should grow slowly to infinity with the sample size n at the order of $log(n)$, for instance. Even with $1+$ million observations in AlexNet, for example, the depth D should be only a multiple of $\log(10^6) \approx 14$.

Convolutional Neural Networks

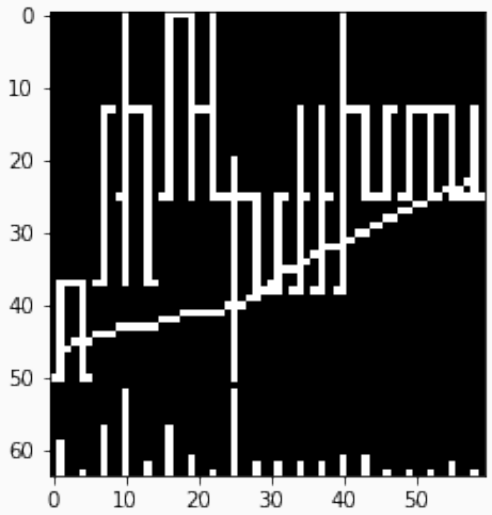
Imaging Price Trends: OHLC Chart





OHLC Chart with Volume Bar and 20-day Moving Average Line

Jiang, Kelly and Xiu (2022+): OHLC Image



$\Rightarrow$

```
[[  0,   0,   0, ...,   0,   0,   0],
 [  0,   0,   0, ...,   0,   0,   0],
 [  0,   0,   0, ...,   0,   0,   0],
 ...,
 [  0, 255,   0, ...,   0,   0,   0],
 [  0, 255,   0, ...,   0,   0,   0],
 [  0, 255,   0, ...,   0, 255,   0]]
```

An Example 20-day OHLC Image (64$\times$ 60): 255=white, 0=black

We can represent grayscale image as matrix

$$\boldsymbol{V} = \{V_{\mathrm{i,j}} : \mathrm{i} = 1, \ldots, \mathrm{I},\ \mathrm{j} = 1, \ldots, \mathrm{J}\}$$

where $V_{\mathrm{i,j}}$ indicates the grayscale of a pixel at the location (i, j).

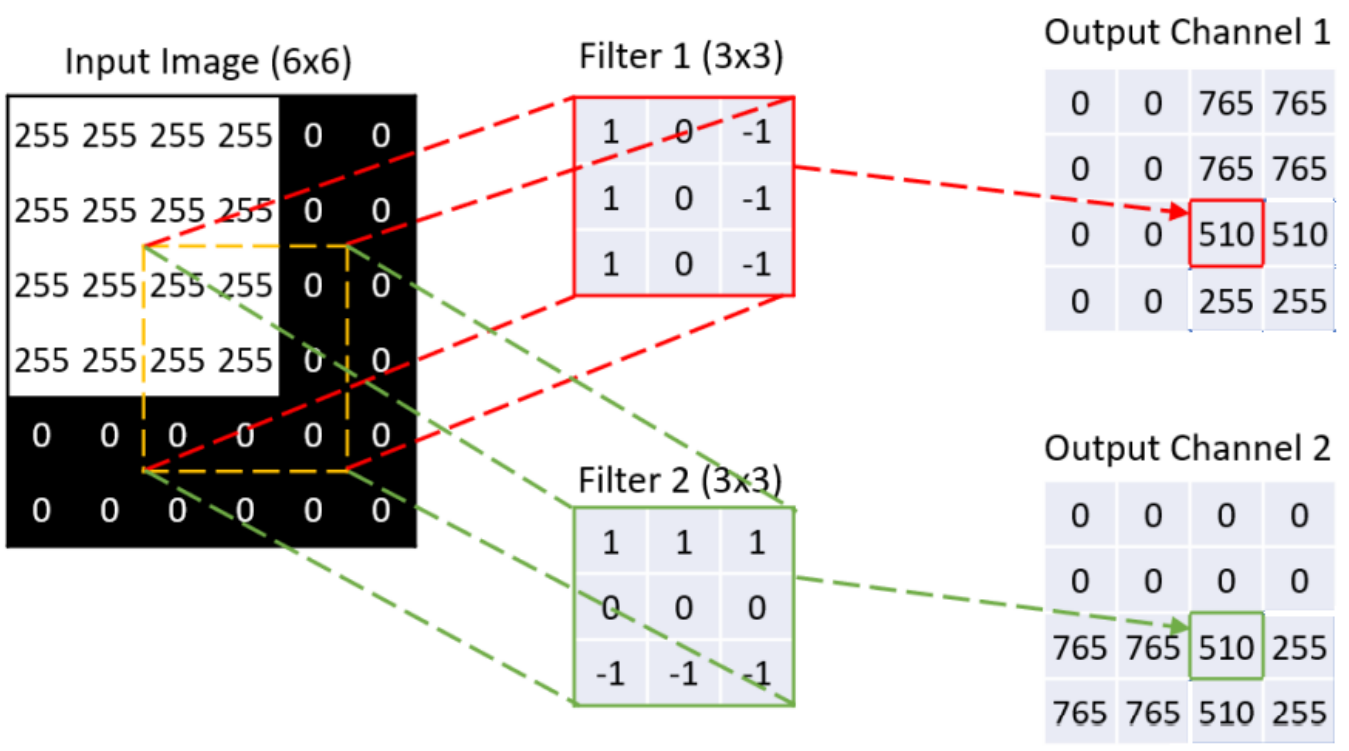
Flattening

- Vectorizing $\boldsymbol{V}$ yields a $\mathrm{I} \times \mathrm{J}$ dimensional vector

$$\text{vec}(\boldsymbol{V}) = (V_{1,1}, \ldots, V_{\mathrm{I},1}, V_{1,2}, \ldots, V_{\mathrm{I},2}, \ldots, V_{1,\mathrm{J}}, \ldots, V_{\mathrm{I},\mathrm{J}})^T,$$

- This process converting multiple grids into a vector is called *flattening* in machine learning.
- Inputting the flattened feature vector to a neural network ignores the spatial structure of the original matrices
- Use a **convolutional neural network** (CNN) to incorporate spatial information

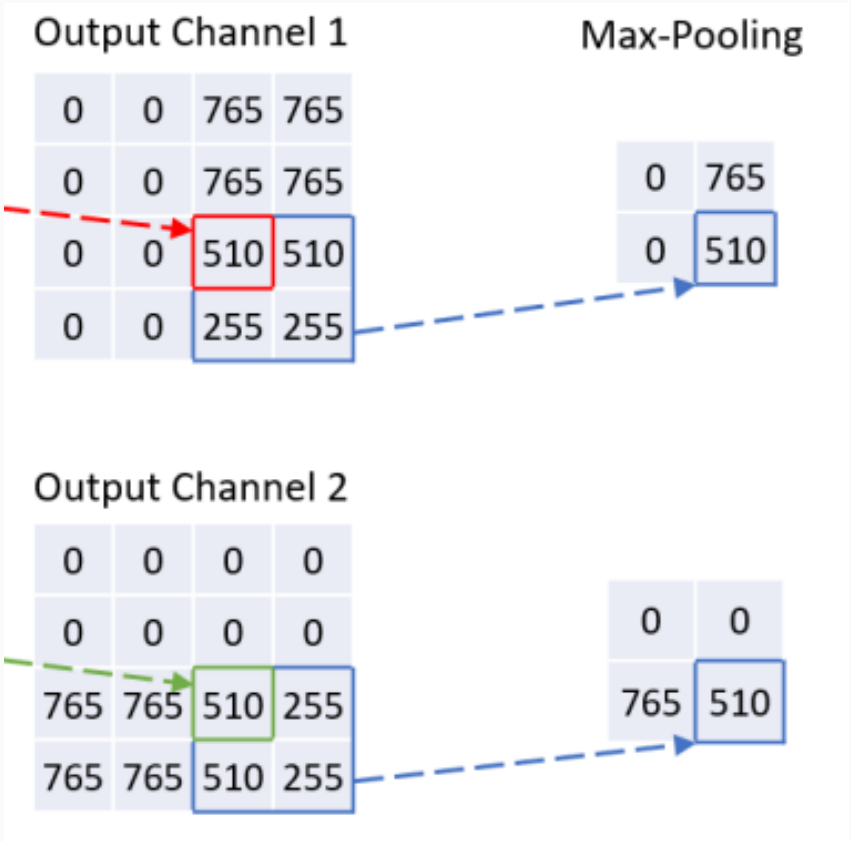
Convolution Layer



Outputs for $\mathrm{K} = 2$ filters

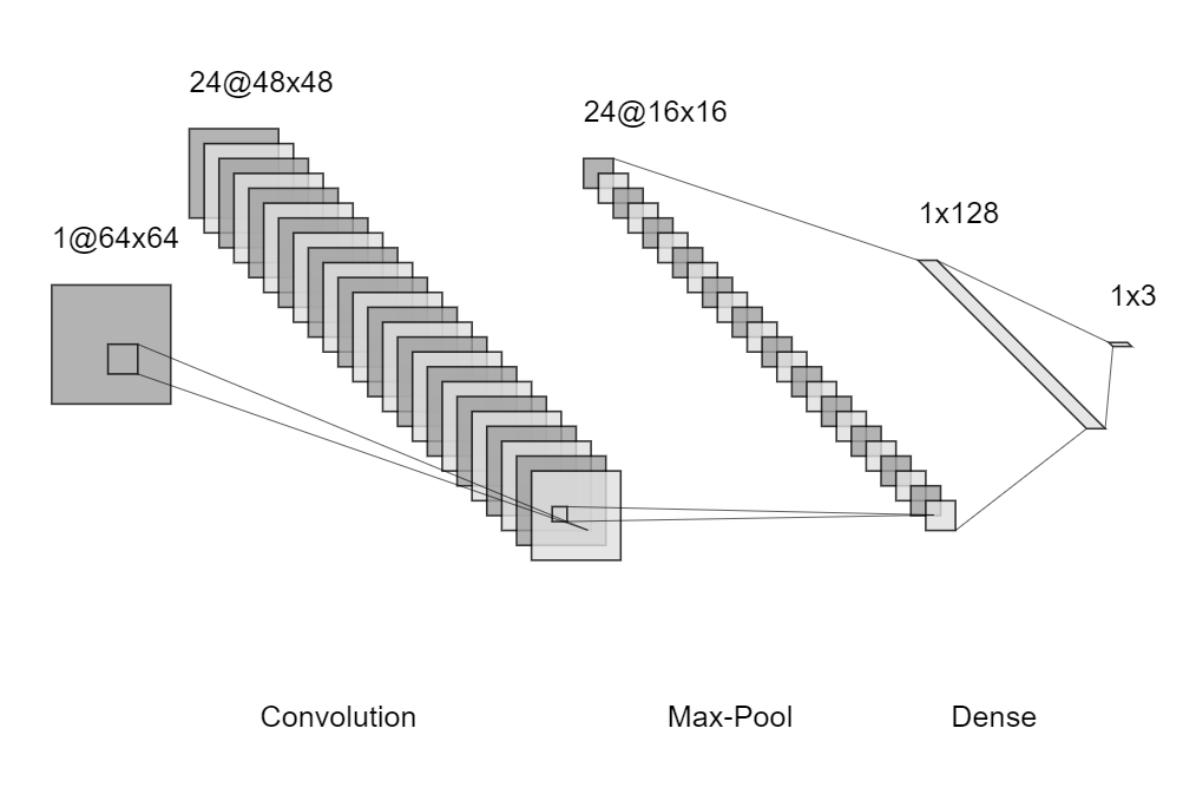
- The kernels are shared by all the subimages
- The kernels are weight parameters to be trained
- Feature maps $\boldsymbol{Z}_{\mathrm{k}} = \sigma(\boldsymbol{A}_{\mathrm{k}} + w_{\mathrm{k},0})$, $\mathrm{k} = 1, \ldots, \mathrm{K}$

Max Pooling



- Flattening the feature maps $\{\boldsymbol{Z}_{\mathrm{k}} : \mathrm{k} = 1, \ldots, \mathrm{K}\}$ directly yields too noisy (and too many) inputs
- Pooling $=$ Replacing **non-overlapping** divisions of the feature maps with their summary statistics (such as maximum)

Dense Layer



- Flatten the feature maps after pooling
- . . . and then input to an ordinary neural network
- Usually these new layers will be **fully connected** and called the dense layers

Training Neural Networks

Weight Decay

We can parameterize the neural nets by

$$f(x;\theta) = f(x;w,b)$$

where

- the vector w collects all the weights
- the vector b collects the biases.

By measuring the model complexity by $C(\theta) = \frac{1}{2}\|w\|^2$, the penalized method (Lecture 1) minimizes

$$L_S(f) + \frac{\lambda}{2}\|w\|^2, \quad \lambda \geq 0$$

where L_S is the empirical risk function.

Use squared loss for regression and cross-entropy for classification tasks (with one-hot encoded target)

Gradient Descent

Starting with an initial value $(w(0), b(0))$, the batch gradient descent algorithm updates the parameters for each step t:

$$\begin{bmatrix} w(t+1) \\ b(t+1) \end{bmatrix} = \begin{bmatrix} w(t) \\ b(t) \end{bmatrix} - \eta \cdot g(w(t), b(t)), \quad \eta > 0,$$

where g is the gradient function given by

$$\begin{aligned} & g(w, b) \\ = & \begin{bmatrix} \nabla_w (L_S(w, b) + \frac{\lambda}{2} \|w\|^2) \\ \nabla_b (L_S(w, b) + \frac{\lambda}{2} \|w\|^2) \end{bmatrix} \\ = & \frac{1}{n} \sum_{i=1}^{n} \begin{bmatrix} \nabla_w \ell(f(X_i; w, b), Y_i) \\ \nabla_b \ell(f(X_i; w, b), Y_i) \end{bmatrix} + \lambda \begin{bmatrix} w \\ \mathbf{0} \end{bmatrix}. \end{aligned}$$

To be continued next week.