Introduction to Statistical/Machine Learning

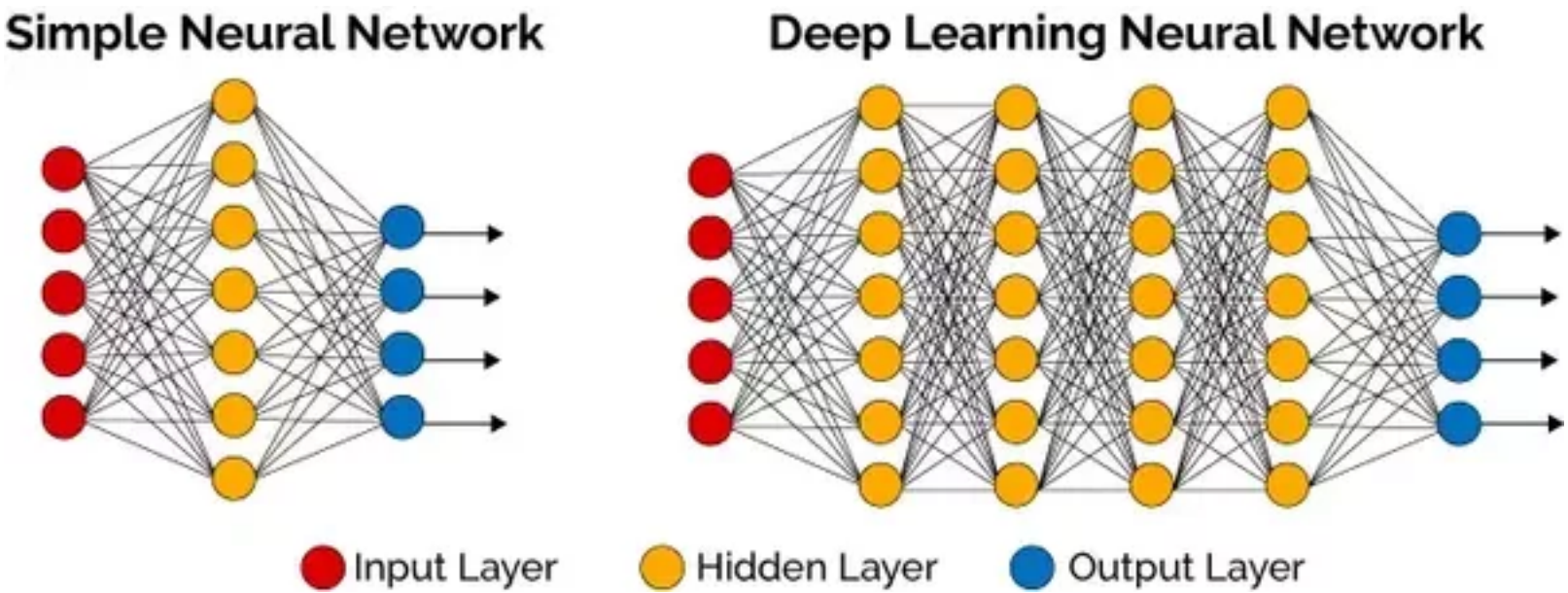
Neural Networks

Anthony Strittmatter

Literature

- James, Witten, Hastie, and Tibshirani (2023): "An Introduction to Statistical Learning", Springer, **Chapter 10**, download.
- Hastie, Tibshirani, and Friedman (2009): "Elements of Statistical Learning", 2nd ed., Springer, download,
  Chapter 11.1-11.8.
- Goodfellow, Bengio, and Courville (2016): "Deep Learning", MIT Press, download.
- Gurney (1997): "An Introduction to Neural Networks", UCL Press, download.

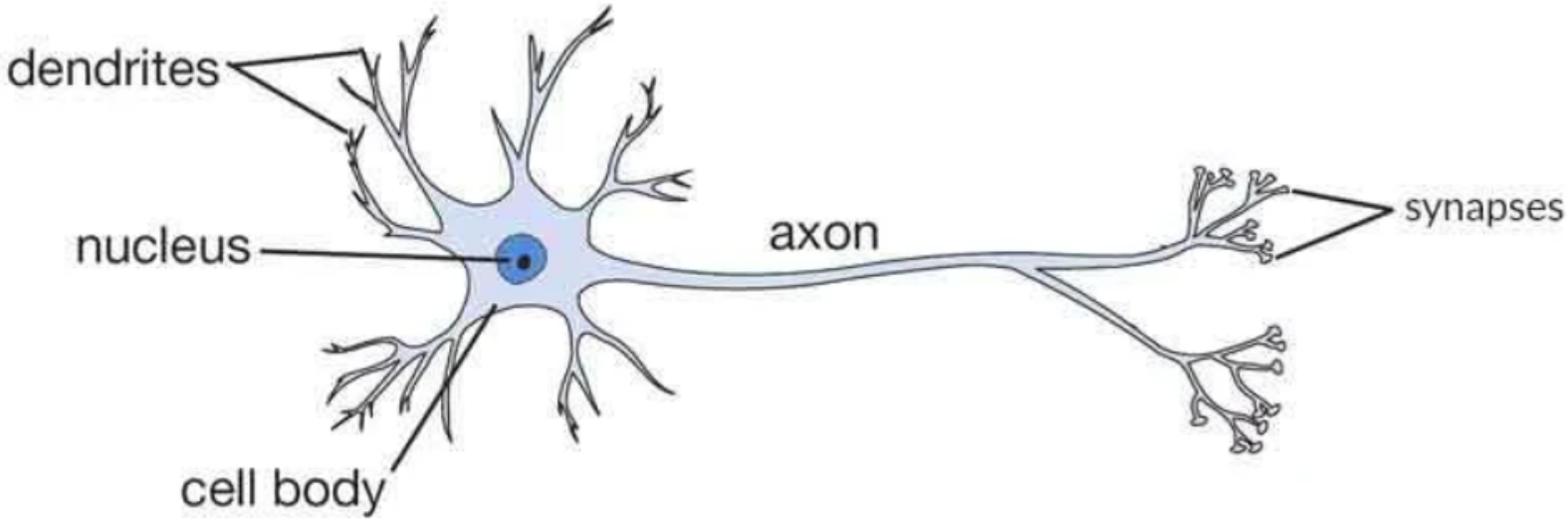
Artificial Neural Networks (ANN)



Basic Idea of Neural Networks

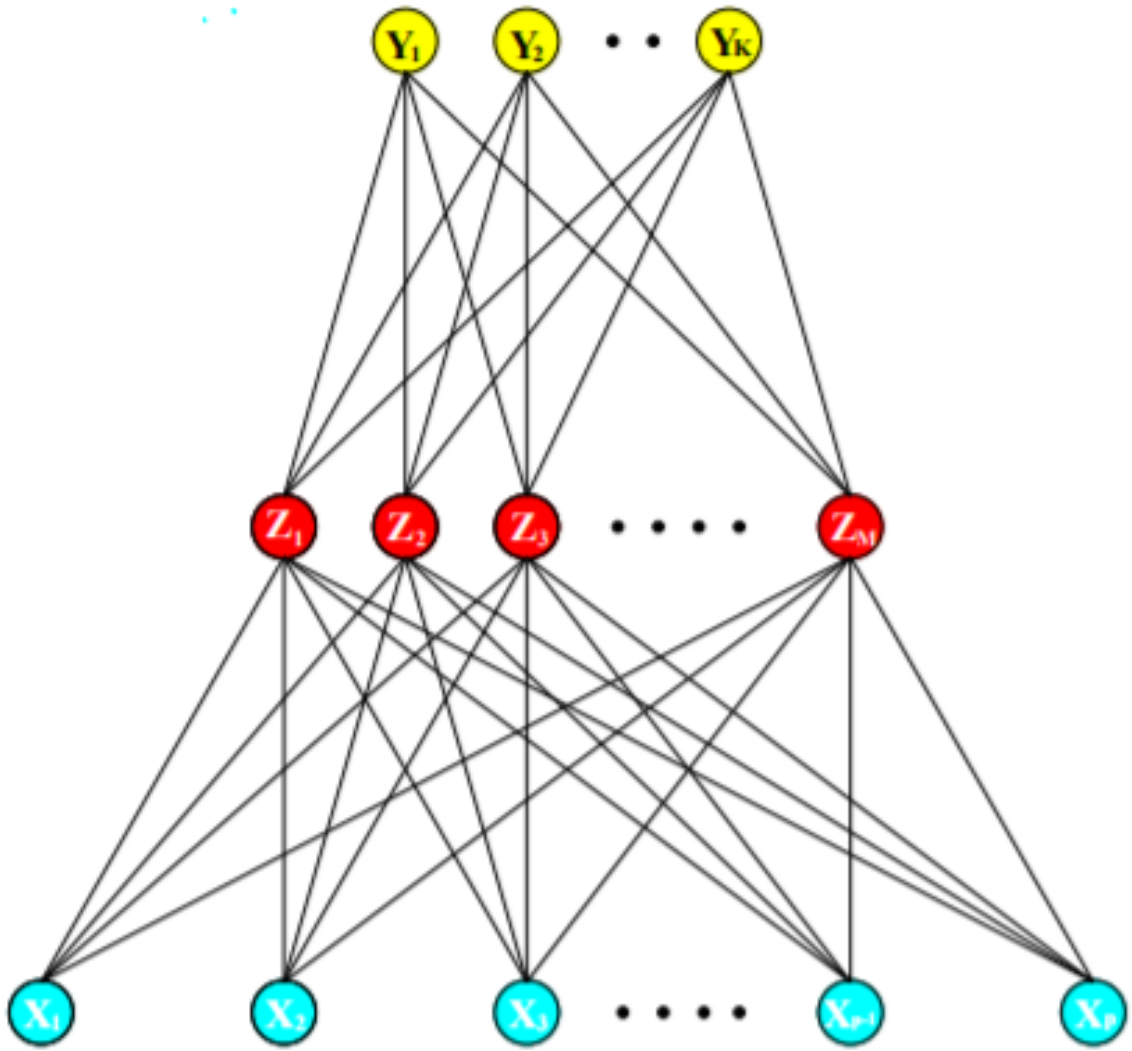
- Neural networks are complex (non-linear) adaptive systems
- Adaptive means it has the ability to change its internal structure by adjusting weights of inputs
- Neural networks were first developed to model human brains
- They are composed of a large number of highly interconnected processing elements known as the neurons
- The connections between neurons represent synapses
- Early neurons fired when the signal passed a certain threshold, which was mimicked by step-functions
- Nowadays, neurons use smoother threshold functions like the sigmoid

Biological Neuron



- Human brains consist of billions of neural cells that process information
- Each neural cell considered a simple processing system
- Dendrites of a neuron receive input signals from another neuron
- Based on those inputs, fire an output signal via an axon

Single Hidden Layer Network



Source: Hastie, Tibshirani, and Friedman (2009)

Two Step Linear Regression or Classification Model

- $X = (X_1, ..., X_P)$ are the **input** layers
- $Z = (Z_1, ..., Z_M)$ are the **neurons or hidden units** of the hidden layer (here only 1 hidden layer)
- $Y = (Y_1, ..., Y_K)$ are the **output** layers
- Predicted output $f_k(X) = (f_1(X), ..., f_K(X))$

$$Z_m = \alpha_{0m} + X\alpha_m \text{ for all } m = 1, ..., M,$$
$$f_k(X) = \beta_{0k} + Z\beta_k \text{ for all } k = 1, ..., K$$

- $\alpha_m = (\alpha_{1m}, ..., \alpha_{Pm})'$ and $\beta_k = (\beta_{1k}, ..., \beta_{Mk})'$ are coefficient vectors (called weights)
- α_{0m} and β_{0k} are intercepts (called bias terms)

Two Step Linear Regression or Classification Model

Merging the two equations:

$$f_k(X) = \beta_{0k} + (\alpha_{01} + X\alpha_1)\beta_{1k} + ... + (\alpha_{0M} + X\alpha_M)\beta_{Mk}$$

$$f_k(X) = \underbrace{\beta_{0k} + \sum_{m=1}^{M} \alpha_{0m}\beta_{1k}}_{=\gamma_0} + \sum_{m=1}^{M} X \underbrace{\alpha_m\beta_{mk}}_{=\gamma_{mk}}$$

$$f_k(X) = \gamma_0 + \sum_{m=1}^{M} X\gamma_{mk}$$

$\Rightarrow$ Breaks down to a simple linear model

Single Hidden Layer Network

Neural networks are non-linear modelling tools:

- $\sigma(\cdot)$ activation function
- $g_k(\cdot)$ is the outcome transformation function

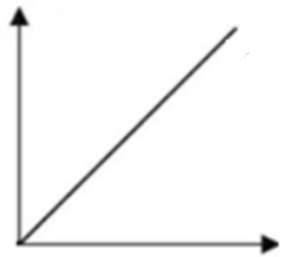
$$Z_m = \sigma(\alpha_{0m} + X\alpha_m) \text{ for all } m = 1, ..., M,$$
$$f_k(X) = g_k(\underbrace{\beta_{0k} + Z\beta_k}_{=T_k}) \text{ for all } k = 1, ..., K$$

Merging the two equations:

$$f_k(X) = g_k(\beta_{0k} + \sigma(\alpha_{01} + X\alpha_1)\beta_{1k} + ... + \sigma(\alpha_{0M} + X\alpha_M)\beta_{Mk})$$

Selection of Activation Functions

- **Identity functions** map input to the same output value, $\sigma(v) = v$.

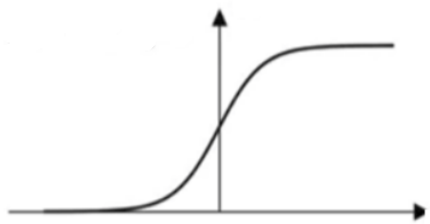


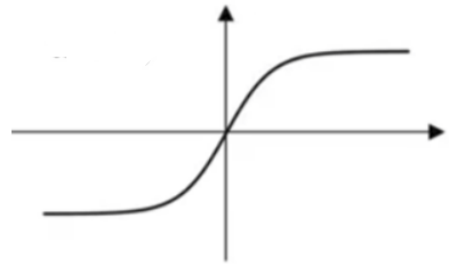
- **Binary step functions** set the output True (or activated) if the value of s is above a certain threshold value c and the output is false (or not activated) if the value of v is below the threshold, $\sigma(v) = 1\{v > s\}$.

Selection of Activation Functions

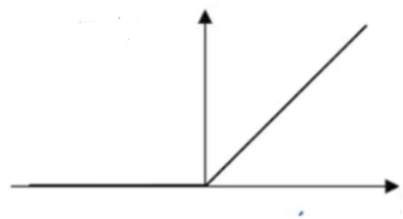
- **S-shaped functions** use often the sigmoid or the hyperbolic tangent transformation:

$$\sigma(v) = \frac{1}{1 + e^{-v}} \quad \text{or} \quad \sigma(v) = \tanh(v) = \frac{\sin(v)}{\cos(v)}.$$





- **Rectified linear unit (ReLU) functions** map negative inputs to 0 and positive inputs to the same output, $\sigma(v) = max(0, v)$.



Popular Activation Functions

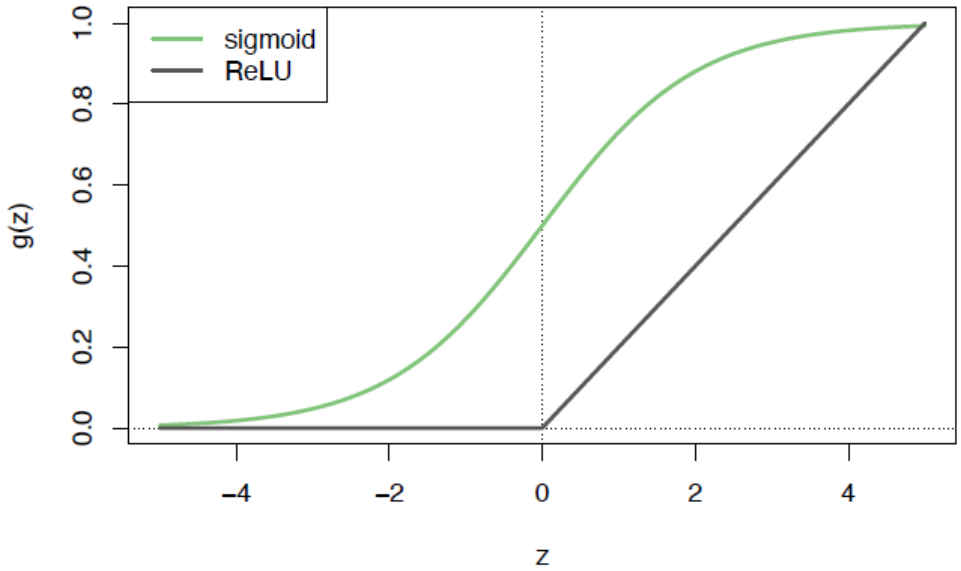


FIGURE 10.2. *Activation functions. The piecewise-linear* `ReLU` *function is popular for its efficiency and computability. We have scaled it down by a factor of five for ease of comparison.*

Outcome Transformation Function

- The output transformation allows to account for the distribution of the output (e.g., continuous or binary).
- This is relevant to distinguish between regression and classification problems.
- For regression problems we typically choose the identity function, $g_k(T_k) = T_k$.
- For classification problems the *softmax* function

$$g_k(T_k) = \frac{e^{T_k}}{\sum_{l=1}^{K} e^{T_l}}$$

is often used. This is the same transformation that is used for the multinomial Logit. It produces positive estimates that sum to one across the K outcomes (e.g. categorical outcome variables).

Optimization of the Network

- We need to estimate values for the coefficients (or weights) $\theta = \{\alpha_{0m}, \alpha_m, \beta_{0k}, \beta_k : m = 1, ..., M; k = 1, ..., K\}$.
- For regressions, we minimize the sum of squared errors

$$R(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_{ik}))^2.$$

- For classification, often the cross-entropy is minimized

$$R(\theta) = -\sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} log(f_k(x_{ik})).$$

- $R(\theta)$ is minimized with a specific gradient descent algorithm, which is called *back-propagation algorithm*.

Back-Propagation Algorithm: Analogy

- Imagine you're trying to learn how to play a video game. Initially, you might not be very good at it, but as you play more and more, you start to figure out which strategies work best to improve your score.

- A neural network is like a virtual brain composed of interconnected nodes. Each node takes some input data, processes it, and passes it on to the next layer of nodes until it reaches the output layer, which gives us the final result.
- The back-propagation algorithm is like the learning process for this neural network.

Back-Propagation Algorithm: Explained

Here's how it works:

- **Forward Pass:** First, we feed some input data into the neural network, and it computes the output. This is called the forward pass. Just like playing the video game, the initial output might not be very accurate.
- **Calculate Error:** Next, we compare the output of the neural network with the actual desired output to see how far off we are. This difference is our error.
- **Backward Pass (Back-propagation):** Now comes the clever part. We go back through the neural network, layer by layer, to see how each node contributed to the error. It's like retracing our steps in the game to figure out where we made mistakes.

Back-Propagation Algorithm: Explained

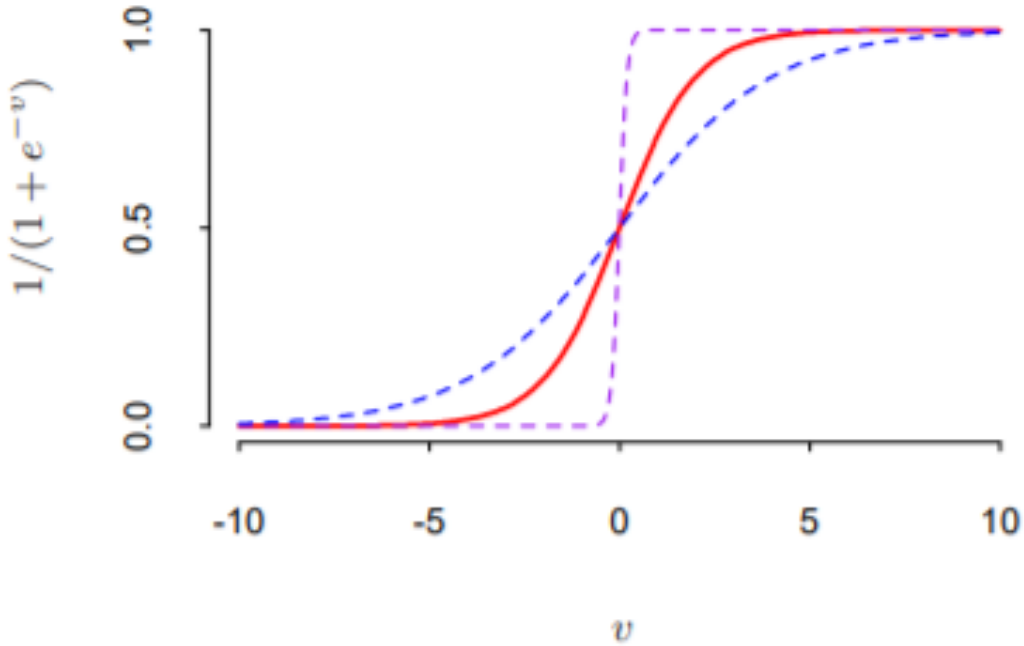
- **Adjust Weights:** As we go backward, we adjust the weights of each connection between nodes to reduce the error. We give more weight to connections that contributed more to the error and less weight to those that contributed less. This is like tweaking our strategies in the game based on where we went wrong.
- **Repeat:** We repeat this process many times, each time getting a bit better at reducing the error. It's like practicing the game over and over again, gradually improving our skills.

By continuously adjusting the weights based on the errors, the neural network learns to make better predictions or decisions over time. And that's essentially how the back-propagation algorithm works - it's a clever way for neural networks to learn from their mistakes and improve their performance.

Starting Values and Local Minima

- Usually starting values for the coefficients (weights) are chosen to be random values near zero.
- The sigmoid is then an almost linear function. Accordingly, we start (almost) linearly and move to a non-linear function.
- The algorithm does not converge when the starting values are exactly zero.
- Often the error function $R(\theta)$ is is non-convex and has many local minima.
- The final solution depends on the choice of the starting values.
- We have to try several starting values and select the results with the smallest error function $R(\theta)$.
- Alternatively, we can average the results across the different starting values (*bagging*), which reduces the out-of-sample variance.

Sigmoid Function



Source: Hastie, Tibshirani, and Friedman (2009)

- Red curve is the sigmoid function $\sigma(v) = \frac{1}{1+e^{-v}}$.
- The dashed lines show weighted sigmoid functions $\sigma(sv)$, where the blue line shows the function for $s = 0.5$ and the violet line for $s = 10$.

Rescaling

- The scale of the inputs affects the scaling of the coefficients (weights).
- This matters particular when we choose the starting values.
- Accordingly, all input layers $X = (X_1, ..., X_P)$ should be standardized (mean of zero and variance of one).
- With standardized input layers, the starting values are typically chosen from a uniform distribution with the range $[-0.7, +0.7]$.

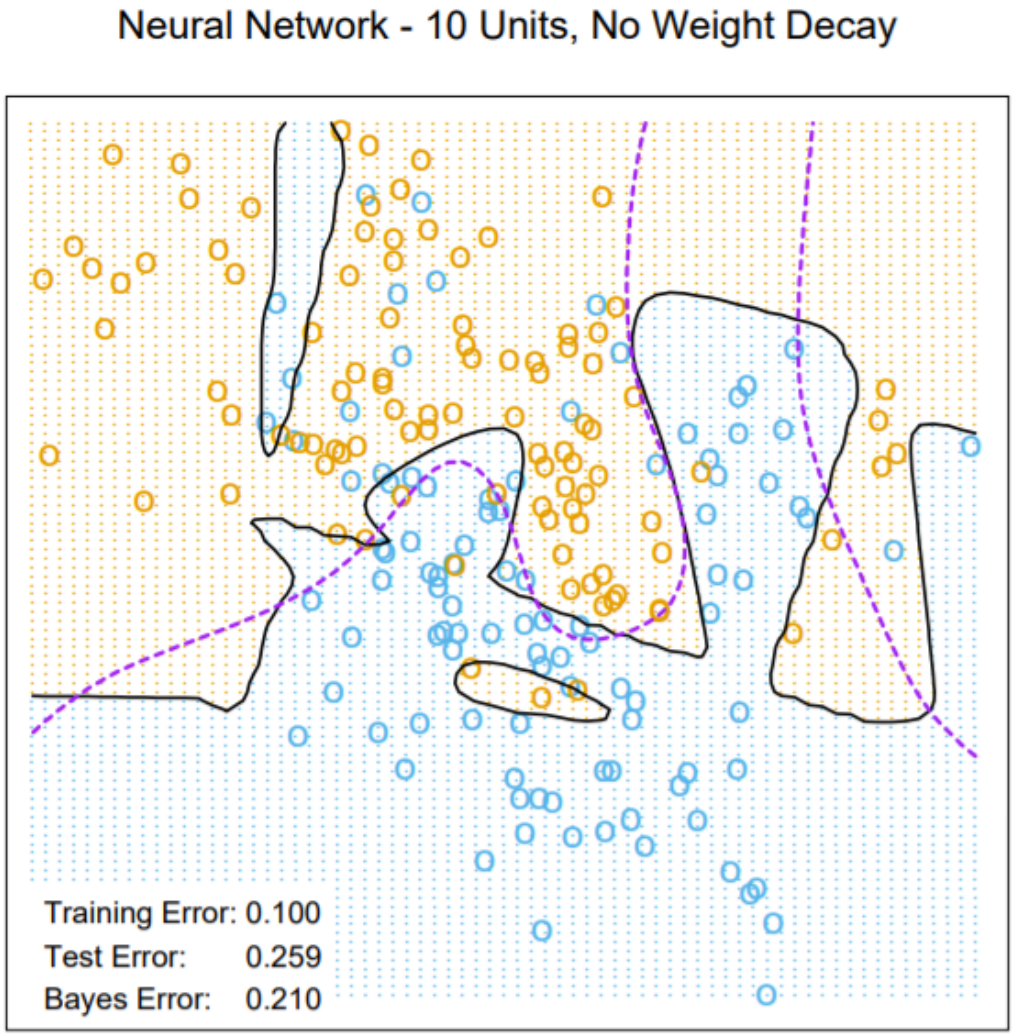
Overfitting

- Neural nets notoriously overfit the data.
- We can use *early stopping rules* to avoid overfitting. A validation data set can be used to determine the optimal time to stop.
- A more explicit method to avoid overfitting is *regularization* or *weight decay*. We add to the error function a penalty term, $R(\theta) + \lambda J(\theta)$.
- A common approach is to use the analogous of the Ridge regression, with the penalty

$$J(\theta) = \sum_{m=1}^{M} \alpha_m^2 + \sum_{k=1}^{K} \beta_k^2$$

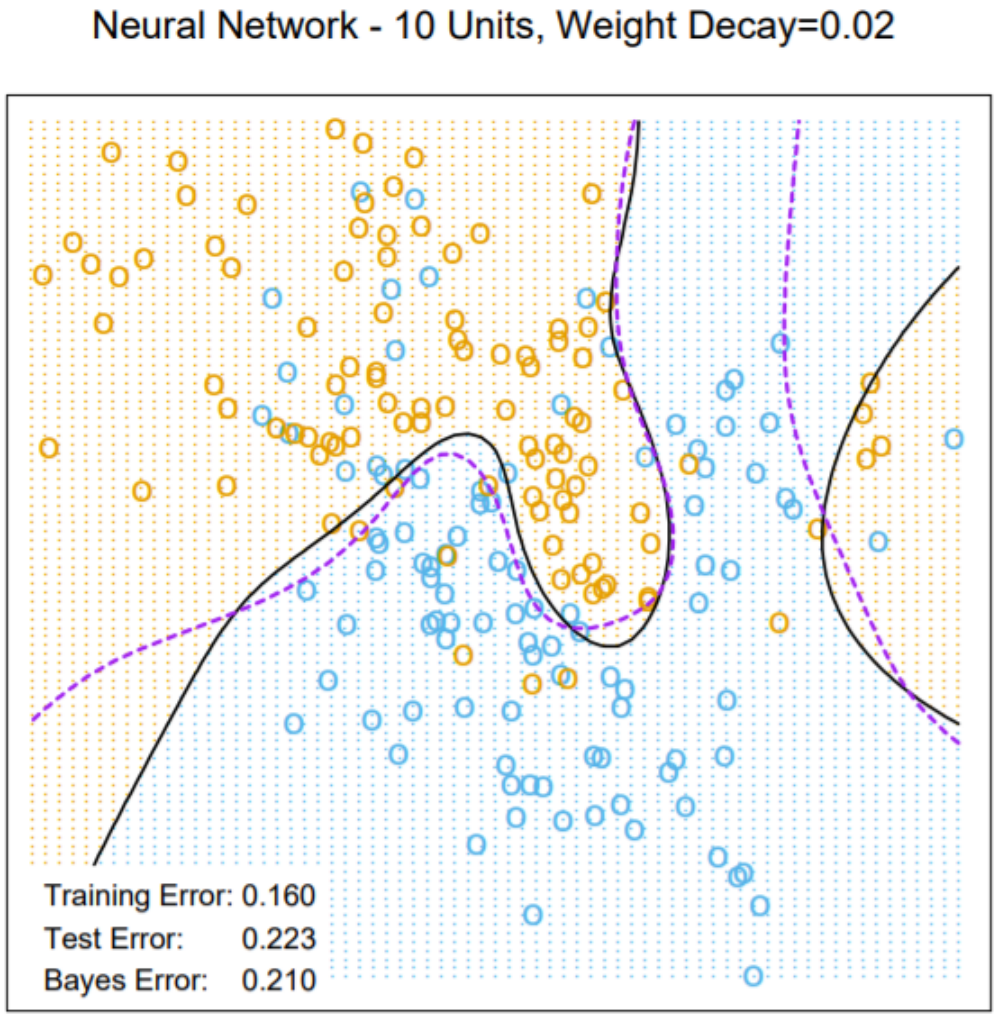
- The tuning parameter $\lambda \geq 0$ is specified with a cross-validation procedure.

Example of an Overfitted Network



Source: Hastie, Tibshirani, and Friedman (2009)

Example with Weight Decay



Source: Hastie, Tibshirani, and Friedman (2009)

Single Layer Neural Network

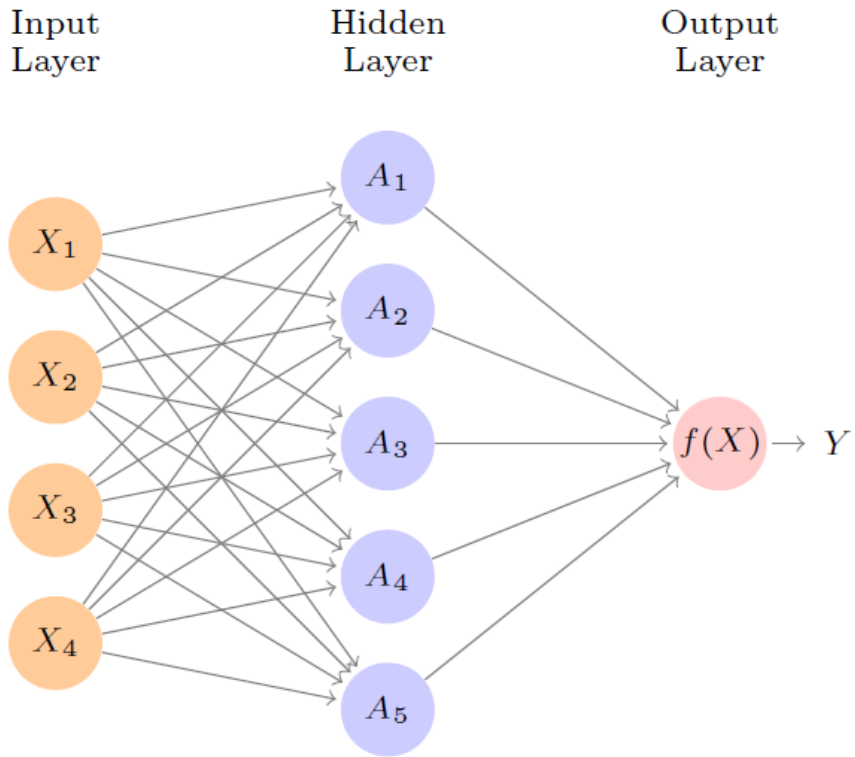


FIGURE 10.1. *Neural network with a single hidden layer. The hidden layer computes activations $A_k = h_k(X)$ that are nonlinear transformations of linear combinations of the inputs $X_1, X_2, \ldots, X_p$. Hence these A_k are not directly observed. The functions $h_k(\cdot)$ are not fixed in advance, but are learned during the training of the network. The output layer is a linear model that uses these activations A_k as inputs, resulting in a function $f(X)$.*

Multi Layer Neural Network

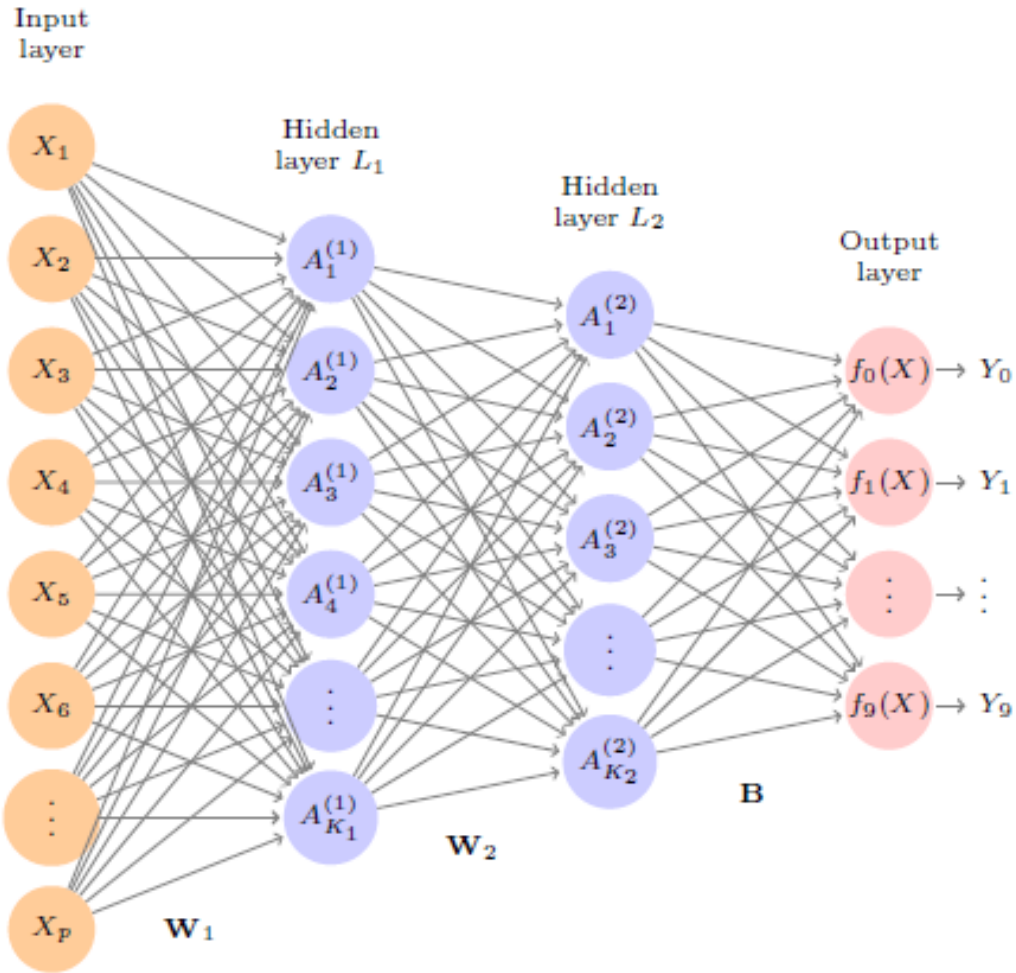


FIGURE 10.4. *Neural network diagram with two hidden layers and multiple outputs, suitable for the* `MNIST` *handwritten-digit problem. The input layer has* $p = 784$ *units, the two hidden layers* $K_1 = 256$ *and* $K_2 = 128$ *units respectively, and the output layer* 10 *units. Along with intercepts (referred to as* biases *in the deep-learning community) this network has 235,146 parameters (referred to as weights).*

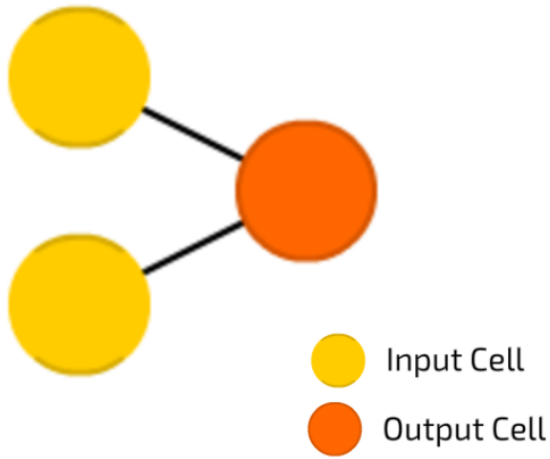
Number of Hidden Units or Neurons

- It is better to have too many than too few hidden units.
- With too few hidden units the model does not have enough flexibility.
- With too many hidden units, we can shrink the extra weights towards zero by choosing the appropriate regularization.
- Typically, the number of hidden units is between 2 and 100; and increases with the number of input layers and the sample size.
- Some people say that the number of hidden units should be between the size of the input layer and the output layer (but this is not always the case).

Number of Hidden Layers

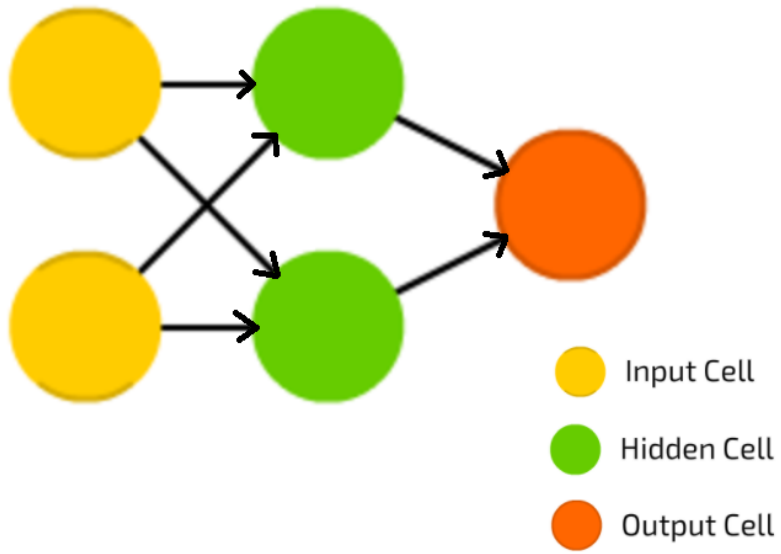
- The choice of the number of hidden layers is guided by background knowledge, experience, and experimentation.
- Using multiple hidden layers (as opposed to the single hidden layer we considered until now) allows to build hierarchical models or to model different resolution levels.
- Networks with 3 or more hidden layers are called *deep neural networks*.
- In theory a single hidden layer with a large number of units has the ability to approximate most functions.
- However, the learning task of discovering a good solution is made much easier with multiple layers each of modest size.

Perceptron



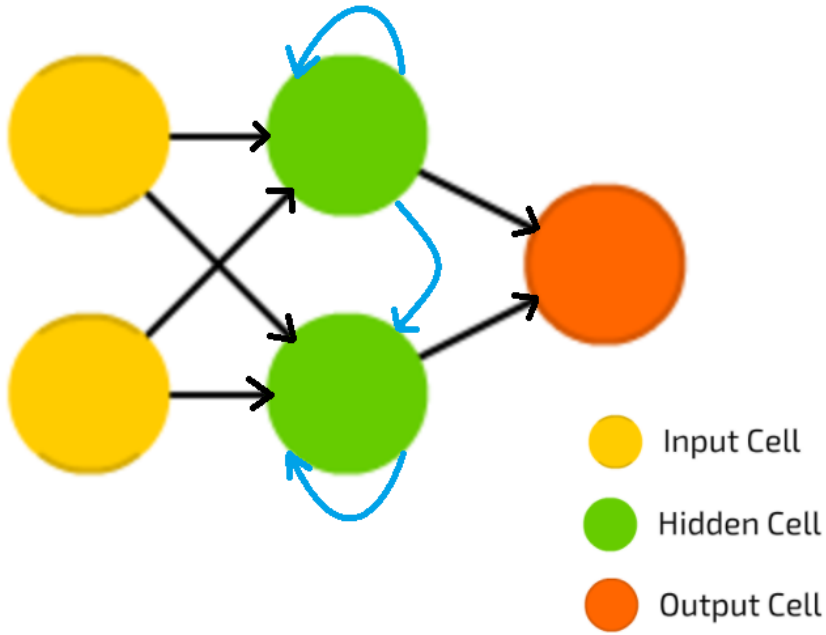
- Has no hidden layers and no hidden units.
- Sums up some inputs, applies the outcome transformation function, and passes them to the output layer.

Feed Forward Neural Network



- Connections between the nodes do not form a cycle.
- Activation flows from input layer to output, without back loops.

Recurrent Neural Network (RNN)

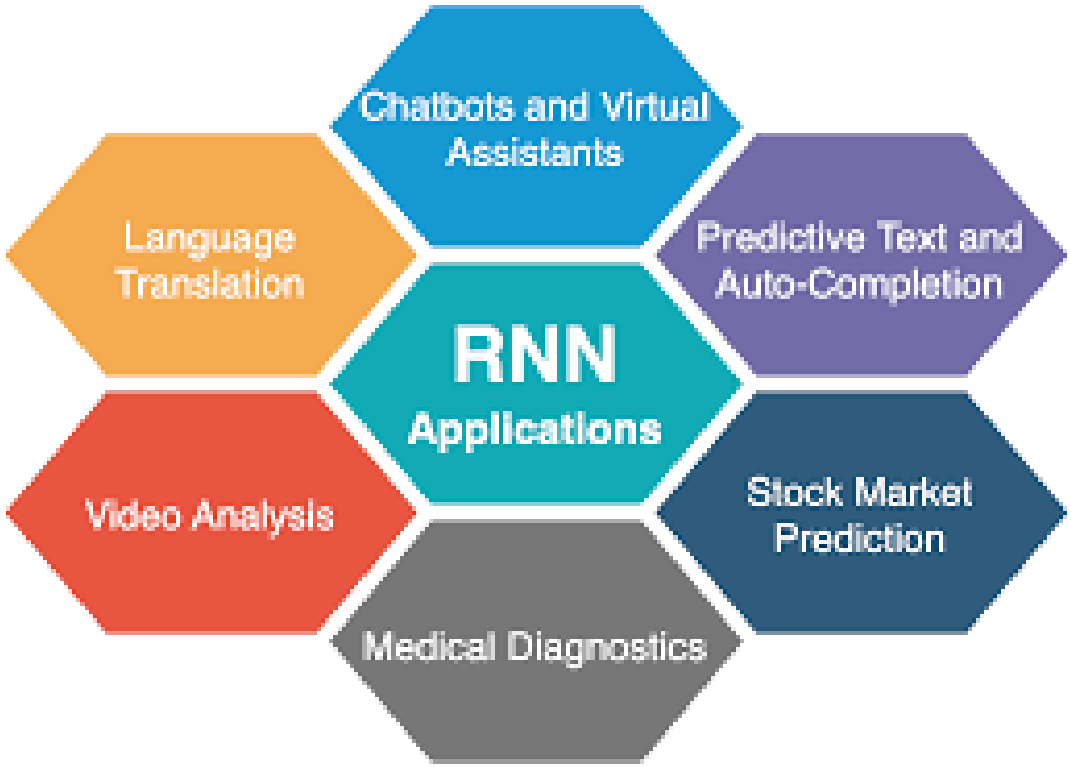


- RNN have connections to between hidden units of the same or previous layers (often with a temporal lag).
- This allows model different aggregation levels jointly or to "remember" the previous input.
- Accordingly, RNNs are suited to model data along a temporal sequence (or sequentially arriving data), e.g. words in a text, time series, voice records.

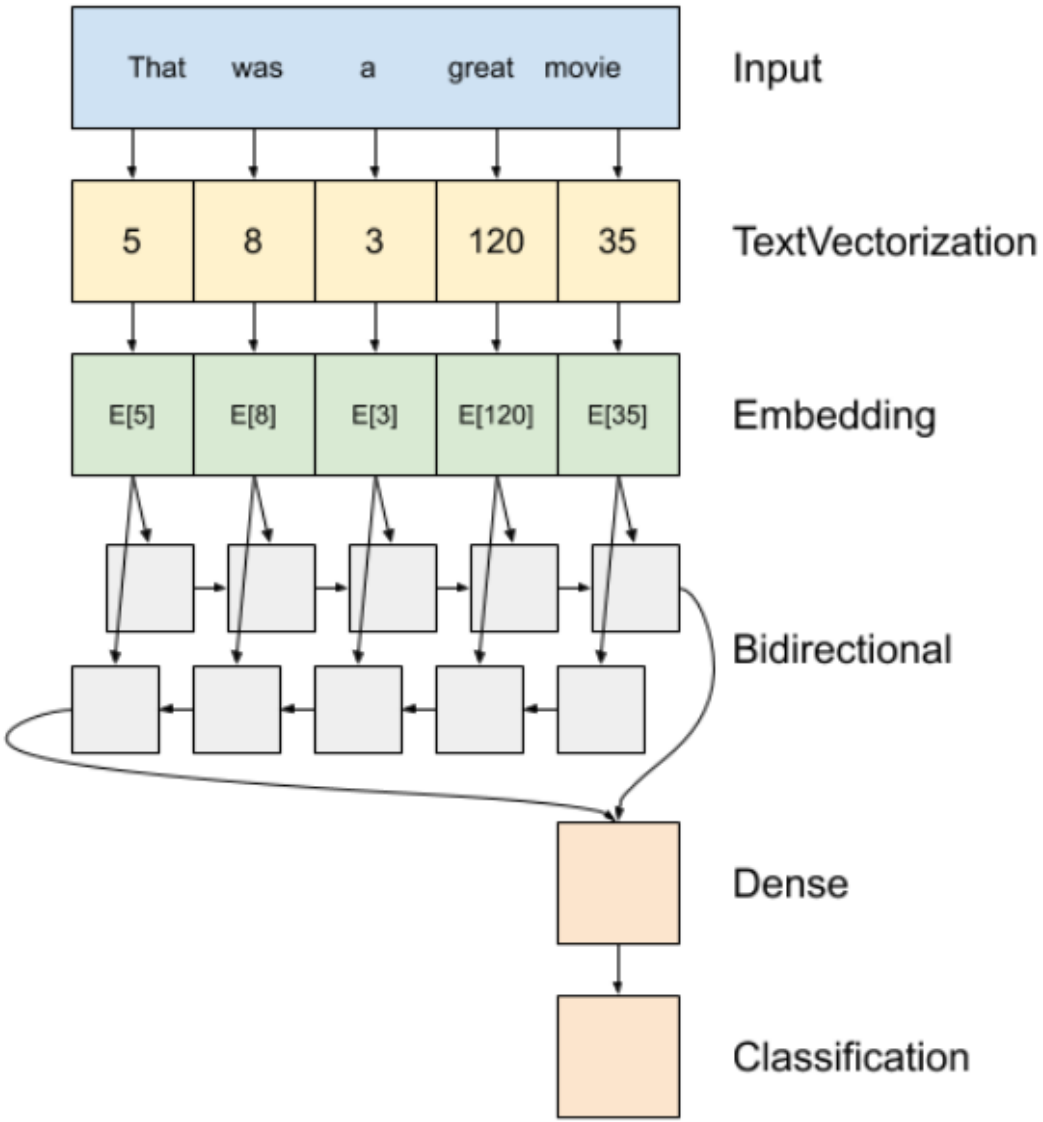
Recurrent Neural Network

- *Direct feedback* is the connection between the output and input of the same neuron.
- *Indirect feedback* is the connection between the output of a neuron and the input of a different neuron of the same layer.
- *Lateral feedback* is the connection between the output of a neuron and the input of a neuron of a previous layer.

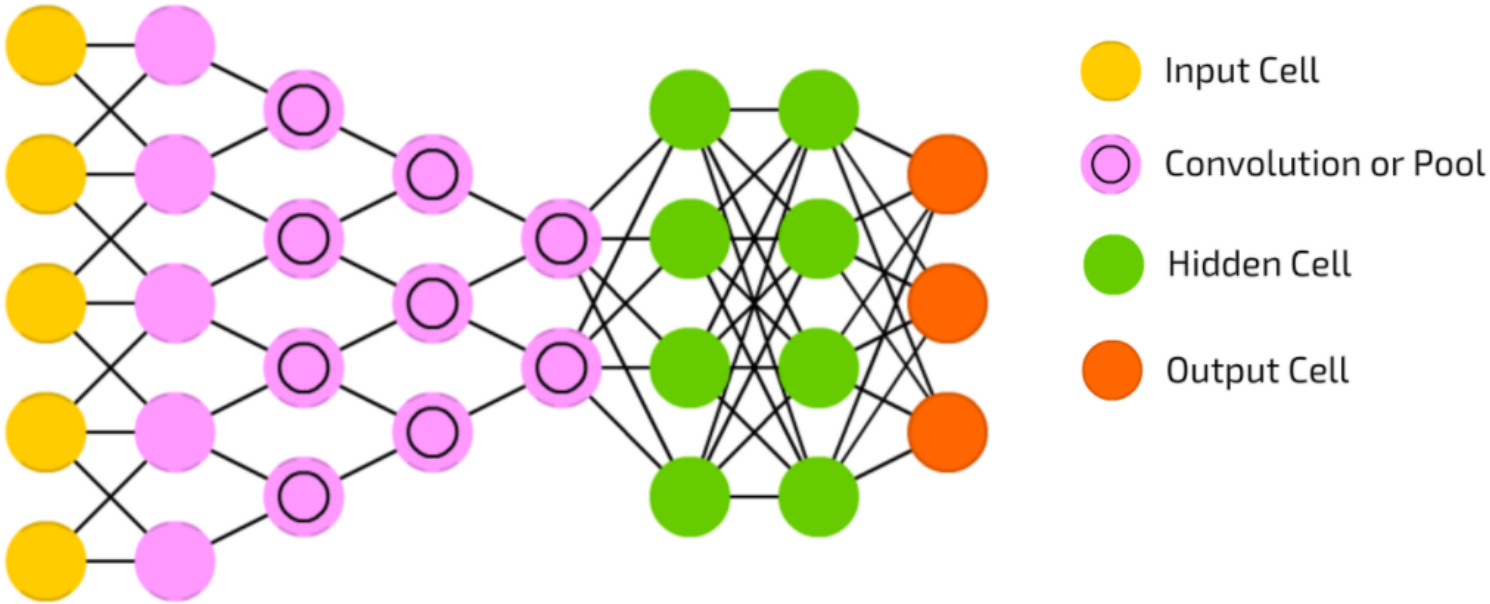
RNN Applications



Example: Text Recognition

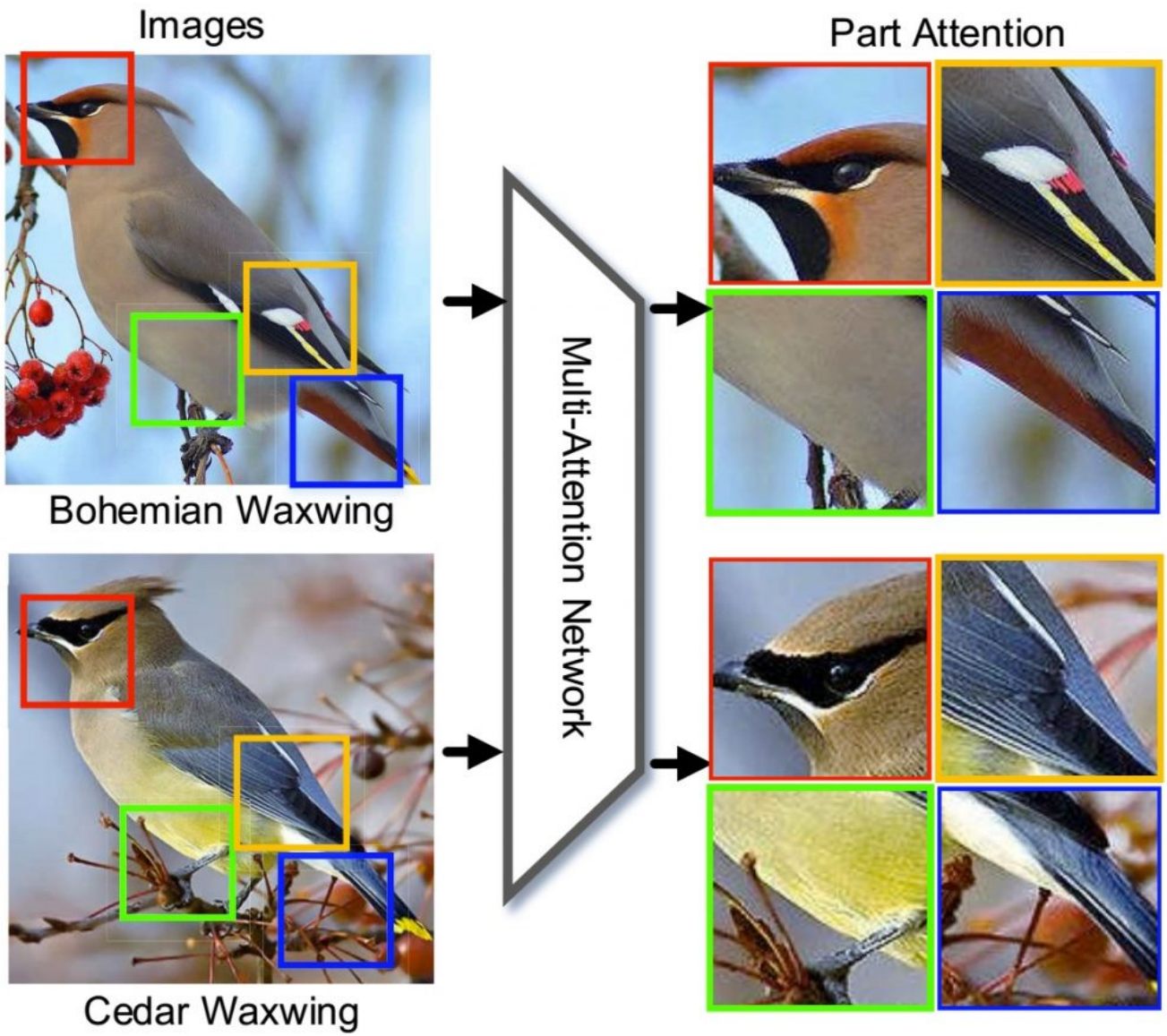


Convolutional Neural Network (CNN)



- CNNs reduce the dimension of the input layer without loosing their characteristics.
- They use a hierarchical structure instead of fully connected cells.
- Then they build a neural net using the convolutions as input.

CNN Example: Bird



CNN Example: Tiger

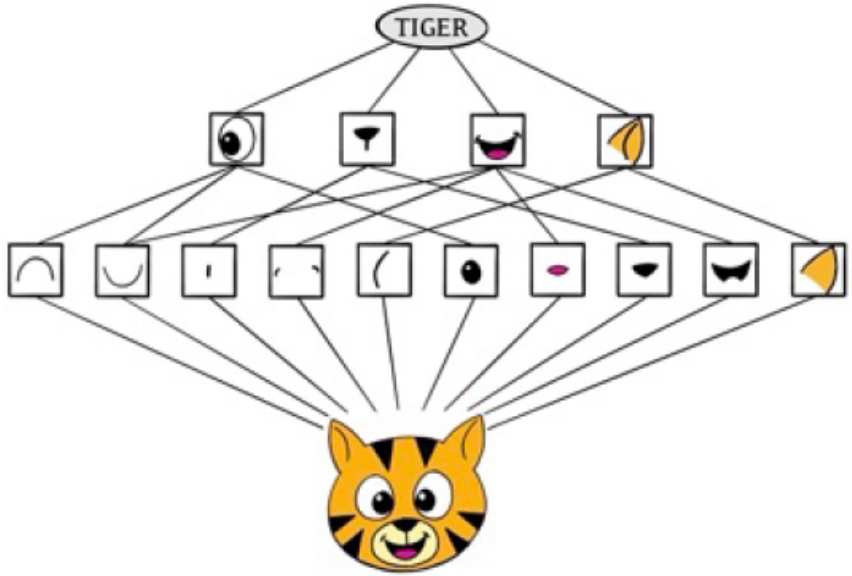


FIGURE 10.6. *Schematic showing how a convolutional neural network classifies an image of a tiger. The network takes in the image and identifies local features. It then combines the local features in order to create compound features, which in this example include eyes and ears. These compound features are used to output the label "tiger".*

- The network first identifies low-level features in the input image, such as small edges, patches of color, and the like.
- These low-level features are then combined to form higher-level features, such as parts of ears, eyes, and so on.

How does a CNN build up this hierarchy?

- CNN combine two specialized types of hidden layers, called convolution layers and pooling layers.
- Convolution layers search for instances of small patterns in the image, whereas pooling layers downsample these to select a prominent subset.
- In order to achieve state-of-the-art results, contemporary neuralnetwork architectures make use of many convolution and pooling layers. We describe convolution and pooling layers next.

Convolution Layers

A convolution layer is made up of a large number of convolution filters, each of which is a template that determines whether a particular local feature is present in an image.

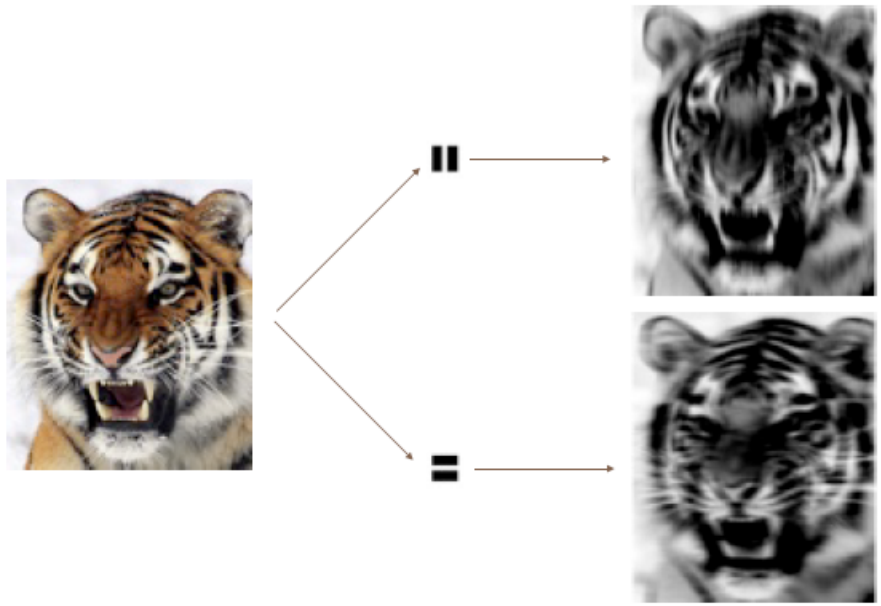
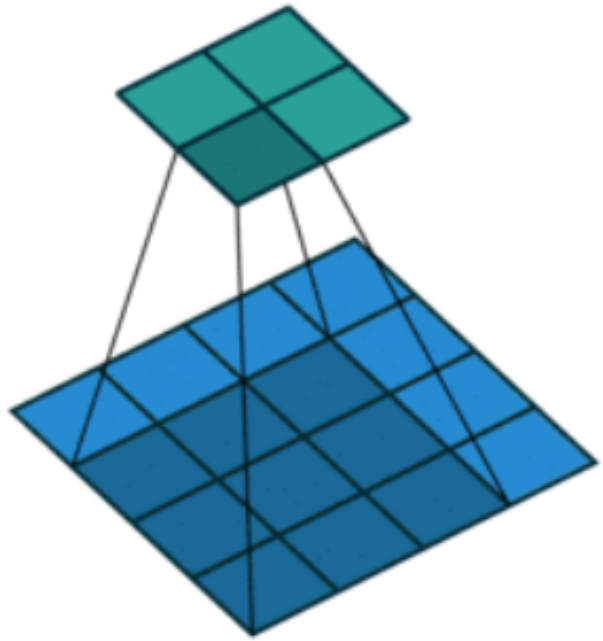


FIGURE 10.7. *Convolution filters find local features in an image, such as edges and small shapes. We begin with the image of the tiger shown on the left, and apply the two small convolution filters in the middle. The convolved images highlight areas in the original image where details similar to the filters are found. Specifically, the top convolved image highlights the tiger's vertical stripes, whereas the bottom convolved image highlights the tiger's horizontal stripes. We can think of the original image as the input layer in a convolutional neural network, and the convolved images as the units in the first hidden layer.*

Convolutions



- Input layer has dimension 4×4 (e.g. pixels)
- We use a 3×3 filter
- Output after convolution has dimension 2×2

Pooling Layers

A pooling layer provides a way to condense a large image into a smaller summary image.

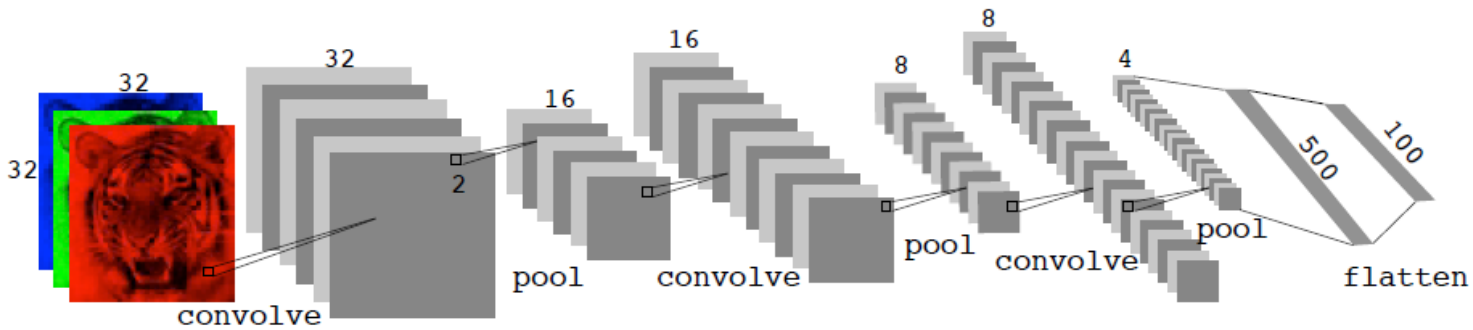
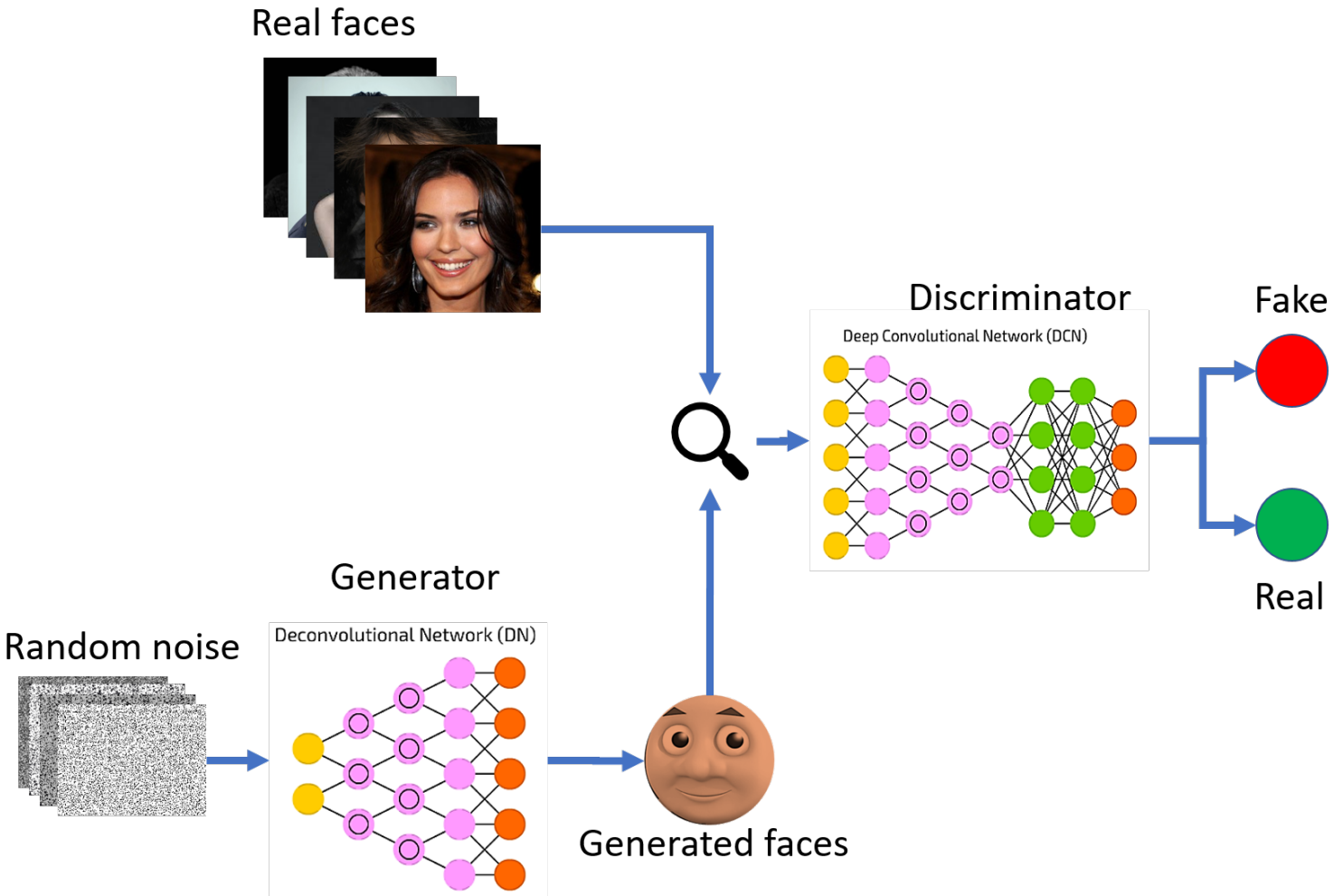
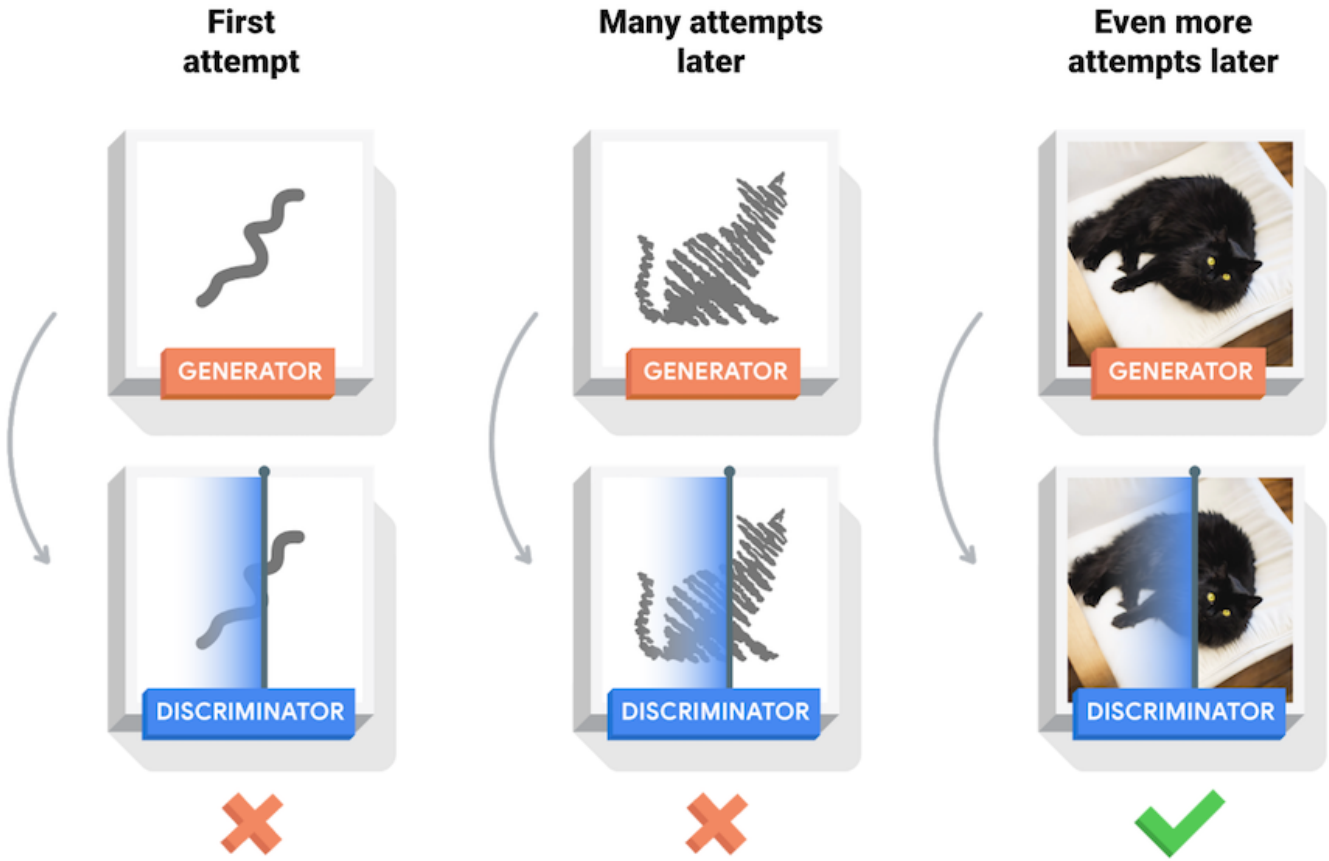


FIGURE 10.8. *Architecture of a deep CNN for the* `CIFAR100` *classification task. Convolution layers are interspersed with* 2×2 *max-pool layers, which reduce the size by a factor of 2 in both dimensions.*

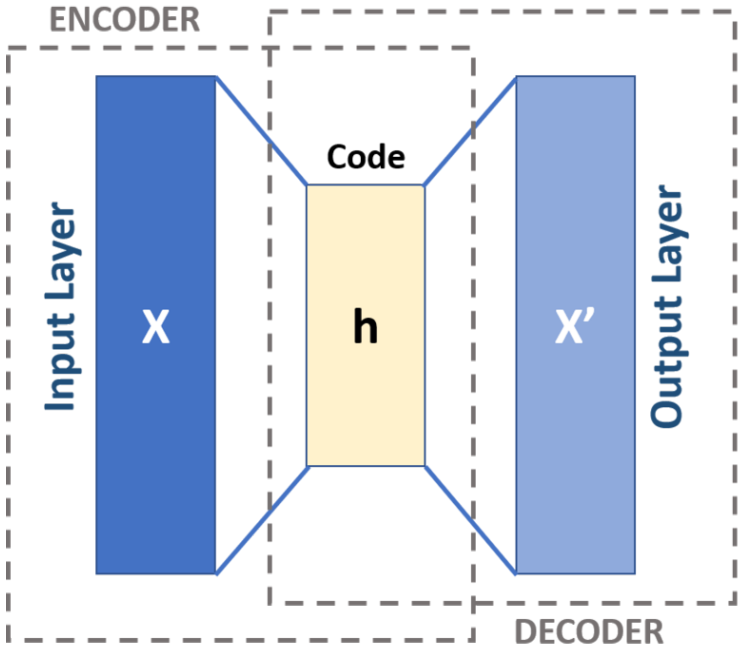
Generative Adversarial Neural Networks (GANs)



Generative Adversarial Neural Networks (GANs)



Autoencoder



- Encoder reduces the dimensionality of the input layer.
- Decoder tries to restore the input layer from the coded data.
- Model accuracy is measured by comparing the input layer X with the output layer X'.

$\Rightarrow$ Unsupervised machine Learning

Tools to Train Deep Neural Nets

Frameworks:

- Torch
- Tensor Flow (can be deployed in R using the *keras* package)
- H2O.ai (R package *h2o*)

Cloud infrastructure:

- Vertex AI (Google Cloud AI Platform)
- Amazon SageMaker
- RapidMiner

When to Use Deep Learning?

- The performance of deep learning can be impressive for: image classification tasks, speech and language translation, forecasting, and text recognition
- *Should we discard all our older tools, and use deep learning on every problem with data?*
- Occam's razor principle: when faced with several methods that give roughly equivalent performance, pick the simplest.

Model	# Parameters	Mean Abs. Error	Test Set R^2
Linear Regression	20	254.7	0.56
Lasso	12	252.3	0.51
Neural Network	1345	257.4	0.54

TABLE 10.2. *Prediction results on the* `Hitters` *test data for linear models fit by ordinary least squares and lasso, compared to a neural network fit by stochastic gradient descent with dropout regularization.*

- Deep learning tends to be an attractive choice when the sample size is extremely large, the data structure is complex, and when interpretability of the model is not a high priority.