Deep Learning in Wireless Communications

Prof. Dola Saha

Assistant Professor, Department of Electrical & Computer Engineering
co-Director, Mobile Emerging Systems and Applications (MESA) Lab
College of Engineering and Applied Sciences
University at Albany, SUNY

Air Force Research Laboratory, Summer 2021

College of Engineering and Applied Sciences
University at Albany
State University of New York

MESA

Introduction

Motivation

- Several Magazines, Journals, Conferences
- IEEE ComSoc Technical Committee
  - Emerging Technologies Initiative Machine Learning for Communications

Materials

- Deep Learning
  Ian Goodfellow, Yoshua Bengio and Aaron Courville
  https://www.deeplearningbook.org/
- Dive into Deep Learning
  Aston Zhang, Zachary Lipton, Mu Li and Alexander Smola
  https://d2l.ai
- Machine Learning: A Probabilistic Perspective
  Kevin P. Murphy
  https://probml.github.io/pml-book/
- Several published papers

Wireless Networking Applications (Some use cases)

- Channel Modeling
- Channel Estimation
- Beamforming and beam prediction
- Antenna tilting
- RF fingerprinting
- Spectrum availability prediction
- Modulation detection
- Waveform generation
- Channel Coding

- Resource Allocation
- Path planning for autonomous systems
- Handover
- Wireless user behavior
- Wireless content prediction
- UAV trajectory prediction

Why Deep Learning can yield better results in Wireless Communication?

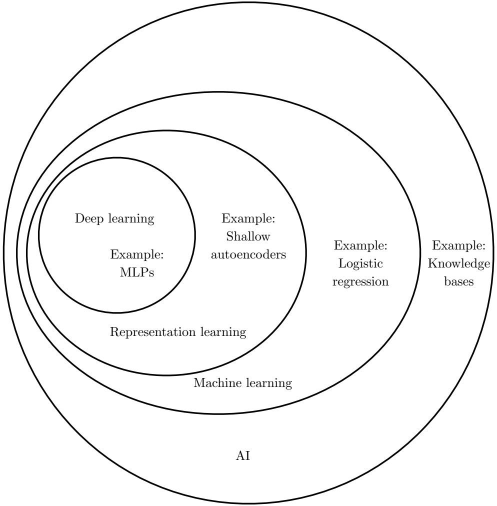
- Signal processing in Tx-Rx chains have been developed (and are optimal) for Gaussian channels
- Often, it is difficult to find a closed form representation of a problem
- Computational complexity of optimal solutions might be high
- New areas of research

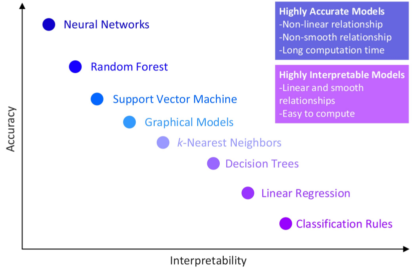
Understanding Wireless Data

- Signals are complex valued (I, Q), whereas image data is three dimensional (RGB)
- Spectrogram can be considered images, but we lose information
- Range varies from [0-254] in image, whereas between [-1,+1] in wireless signals

Challenge: Dataset

Evolution of Deep Learning

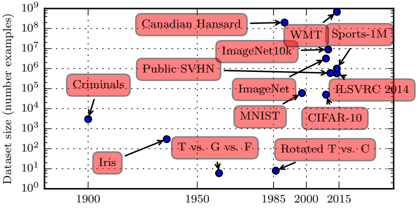


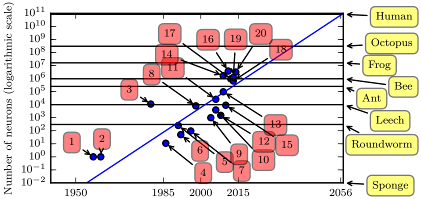


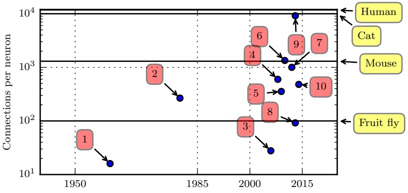
M. E. Morocho-Cayamcela, H. Lee and W. Lim, "Machine Learning for 5G/B5G Mobile and Wireless Communications: Potential, Limitations, and Future Directions," in IEEE Access, vol. 7, pp. 137184-137206, 2019, doi: 10.1109/ACCESS.2019.2942390.

Key Reasons for Success of Deep Learning

- Increasing Dataset Sizes
  - 5000 labeled samples/category
- Increasing Model Sizes
  - Hidden layers doubled every 2.4years
  - Availability of faster CPUs
  - Advent of GPUs
  - Faster network connectivity
  - Better software infrastructure

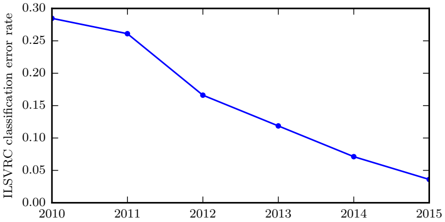






Improved Accuracy

Decade	Dataset	Memory	Floating point calculations per second
1970	100 (Iris)	1 KB	100 KF (Intel 8080)
1980	1 K (House prices in Boston)	100 KB	1 MF (Intel 80186)
1990	10 K (optical character recognition)	10 MB	10 MF (Intel 80486)
2000	10 M (web pages)	100 MB	1 GF (Intel Core)
2010	10 G (advertising)	1 GB	1 TF (Nvidia C2050)
2020	1 T (social network)	100 GB	1 PF (Nvidia DGX-2)



Learning Model

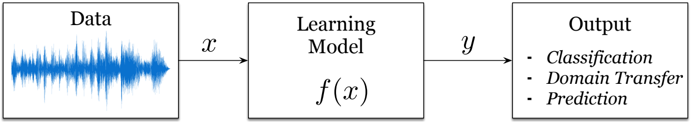


Figure: Approximate $f(x)$ from the data

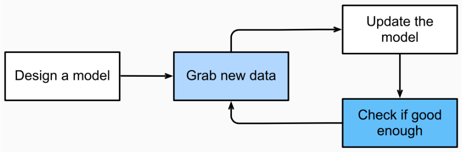


Figure: A typical training process

- *Learning* is the process by which we discover the right setting of the knobs yielding the desired behavior from our model. In other words, we *train* our model with data.
- Wireless Applications: modulation detection, RF fingerprinting, channel estimation, channel modeling, generate waveforms

Key Components of Learning

- The data that we can learn from: constitutes attributes or *features* from which the model should learn.
  - Training Set: set of examples used to fit the parameters of the model
  - Validation/Testing Set: set of examples used to test the performance of the model after training

Key Components of Learning

- The data that we can learn from: constitutes attributes or *features* from which the model should learn.
  - Training Set: set of examples used to fit the parameters of the model
  - Validation/Testing Set: set of examples used to test the performance of the model after training
- A model of how to transform the data

Key Components of Learning

- The data that we can learn from: constitutes attributes or *features* from which the model should learn.
  - Training Set: set of examples used to fit the parameters of the model
  - Validation/Testing Set: set of examples used to test the performance of the model after training
- A model of how to transform the data
- An objective function that quantifies how well (or badly) the model is doing.
  - a mathematical formulation to measure performance in each iteration (epoch) of training
  - conventionally, minimization, leading to the term, *loss function*

Key Components of Learning

- The data that we can learn from: constitutes attributes or *features* from which the model should learn.
  - Training Set: set of examples used to fit the parameters of the model
  - Validation/Testing Set: set of examples used to test the performance of the model after training
- A model of how to transform the data
- An objective function that quantifies how well (or badly) the model is doing.
  - a mathematical formulation to measure performance in each iteration (epoch) of training
  - conventionally, minimization, leading to the term, *loss function*
- An algorithm to adjust the model's parameters to optimize the objective function.

Types of Machine Learning Problems

- Supervised Learning: addresses the task of predicting labels given input features (labels)
  - Example Modulation Detection: Labels provided during training (modulation order)
  - Types: Regression, Classification, Tagging, Search, Recommender Systems, Sequence Learning

Types of Machine Learning Problems

- Supervised Learning: addresses the task of predicting labels given input features (labels)
  - Example Modulation Detection: Labels provided during training (modulation order)
  - Types: Regression, Classification, Tagging, Search, Recommender Systems, Sequence Learning
- Unsupervised Learning: no features or corresponding labels are provided
  - Example Modulation Detection: Can we find number of clusters in the constellation? - *clustering*
  - Generate new waveforms: *Generative adversarial networks*

Types of Machine Learning Problems

- Supervised Learning: addresses the task of predicting labels given input features (labels)
    - Example Modulation Detection: Labels provided during training (modulation order)
    - Types: Regression, Classification, Tagging, Search, Recommender Systems, Sequence Learning
- Unsupervised Learning: no features or corresponding labels are provided
    - Example Modulation Detection: Can we find number of clusters in the constellation? - *clustering*
    - Generate new waveforms: *Generative adversarial networks*
- Interacting with an Environment: offine learning, agents adapt to distribution shift in data

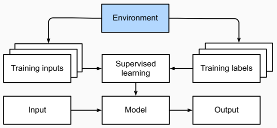


Figure: Interacting with an Environment

Types of Machine Learning Problems

- Supervised Learning: addresses the task of predicting labels given input features (labels)
  - Example Modulation Detection: Labels provided during training (modulation order)
  - Types: Regression, Classification, Tagging, Search, Recommender Systems, Sequence Learning
- Unsupervised Learning: no features or corresponding labels are provided
  - Example Modulation Detection: Can we find number of clusters in the constellation? - *clustering*
  - Generate new waveforms: *Generative adversarial networks*
- Interacting with an Environment: offine learning, agents adapt to distribution shift in data
- Reinforcement Learning: develop an agent that interacts with an environment, takes actions, a policy to reward the action

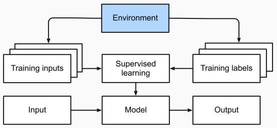


Figure: Interacting with an Environment

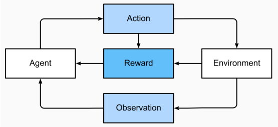


Figure: Reinforcement Learning

Machine Learning Basics

Machine Learning Basics

- Linear Algebra
- Probability
- Calculus

Scalars and Vectors

- A scalar is a single number
  - Integers, real numbers, rational numbers, etc.
  - We denote it with italic font: a, n, x
- A vector is a 1-D array of numbers
  - Can be real, binary, integer, etc.
  - Example notation for type and size: $\boldsymbol{x} \in \mathbb{R}^n$
- A matrix is a 2-D array of numbers
  - Can be real, binary, integer, etc.
  - Example notation for type and size: $\boldsymbol{A} \in \mathbb{R}^{m \times n}$
- A tensor is an array of numbers, that may have
  - zero dimensions, and be a scalar
  - one dimension, and be a vector
  - two dimensions, and be a matrix
  - or more dimensions.

Vector:

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Matrix:

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

Matrix Operations

- Matrix Transpose: $\mathbf{B} = \mathbf{A}^\top$

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \quad \mathbf{A}^\top = \begin{bmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & \cdots & a_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{mn} \end{bmatrix}$$

Matrix Operations

- Matrix Transpose: $\mathbf{B} = \mathbf{A}^\top$

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \quad \mathbf{A}^\top = \begin{bmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & \cdots & a_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{mn} \end{bmatrix}$$

- Hadamard Product: Elementwise multiplication $C = \mathbf{A} \odot \mathbf{B}$

$$\mathbf{A} \odot \mathbf{B} = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} & \cdots & a_{1n}b_{1n} \\ a_{21}b_{21} & a_{22}b_{22} & \cdots & a_{2n}b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}b_{m1} & a_{m2}b_{m2} & \cdots & a_{mn}b_{mn} \end{bmatrix}$$

Matrix Operations

- Matrix Transpose: $\mathbf{B} = \mathbf{A}^\top$

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \quad \mathbf{A}^\top = \begin{bmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & \cdots & a_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{mn} \end{bmatrix}$$

- Hadamard Product: Elementwise multiplication $C = \mathbf{A} \odot \mathbf{B}$

$$\mathbf{A} \odot \mathbf{B} = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} & \cdots & a_{1n}b_{1n} \\ a_{21}b_{21} & a_{22}b_{22} & \cdots & a_{2n}b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}b_{m1} & a_{m2}b_{m2} & \cdots & a_{mn}b_{mn} \end{bmatrix}$$

- Reduction: Sum of the elements

$$S = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}$$

Matrix Operations

- Matrix Multiplication: $\mathbf{C} = \mathbf{AB}$

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nk} \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1m} \\ b_{21} & b_{22} & \cdots & b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{k1} & b_{k2} & \cdots & b_{km} \end{bmatrix}, \quad \mathbf{C} = \mathbf{AB} = \begin{bmatrix} \mathbf{a}_1^\top \\ \mathbf{a}_2^\top \\ \vdots \\ \mathbf{a}_n^\top \end{bmatrix} \begin{bmatrix} \mathbf{b}_1 & \mathbf{b}_2 & \cdots & \mathbf{b}_m \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1^\top \mathbf{b}_1 & \mathbf{a}_1^\top \mathbf{b}_2 & \cdots & \mathbf{a}_1^\top \mathbf{b}_m \\ \mathbf{a}_2^\top \mathbf{b}_1 & \mathbf{a}_2^\top \mathbf{b}_2 & \cdots & \mathbf{a}_2^\top \mathbf{b}_m \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{a}_n^\top \mathbf{b}_1 & \mathbf{a}_n^\top \mathbf{b}_2 & \cdots & \mathbf{a}_n^\top \mathbf{b}_m \end{bmatrix}$$

Matrix Operations

- Matrix Multiplication: $\mathbf{C} = \mathbf{AB}$

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nk} \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1m} \\ b_{21} & b_{22} & \cdots & b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{k1} & b_{k2} & \cdots & b_{km} \end{bmatrix}, \quad \mathbf{C} = \mathbf{AB} = \begin{bmatrix} \mathbf{a}_1^\top \\ \mathbf{a}_2^\top \\ \vdots \\ \mathbf{a}_n^\top \end{bmatrix} \begin{bmatrix} \mathbf{b}_1 & \mathbf{b}_2 & \cdots & \mathbf{b}_m \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1^\top \mathbf{b}_1 & \mathbf{a}_1^\top \mathbf{b}_2 & \cdots & \mathbf{a}_1^\top \mathbf{b}_m \\ \mathbf{a}_2^\top \mathbf{b}_1 & \mathbf{a}_2^\top \mathbf{b}_2 & \cdots & \mathbf{a}_2^\top \mathbf{b}_m \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{a}_n^\top \mathbf{b}_1 & \mathbf{a}_n^\top \mathbf{b}_2 & \cdots & \mathbf{a}_n^\top \mathbf{b}_m \end{bmatrix}$$

- Matrix Inversion: $\mathbf{A}^{-1}\mathbf{A} = \mathbf{I}_n$, where $\mathbf{I}$ is the identity matrix
  Example: Let $\mathbf{Ax} = \mathbf{b}$ be a system of linear equation.
  Then, $\mathbf{A}^{-1}\mathbf{Ax} = \mathbf{A}^{-1}\mathbf{b}$, which implies $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$

Matrix Operations

- Norm: Function to measure size of a vector
- Does not represent dimensionality but rather the magnitude of the components.

Matrix Operations

- Norm: Function to measure size of a vector
- Does not represent dimensionality but rather the magnitude of the components.
- A vector norm is a function f that maps a vector to a scalar, satisfying following properties:
  1. if all elements of a vector are scaled by a constant factor α, its norm also scales by α
     $f(\alpha \mathbf{x}) = |\alpha| f(\mathbf{x})$
  2. Triangle inequality
     $f(\mathbf{x} + \mathbf{y}) \leq f(\mathbf{x}) + f(\mathbf{y})$
  3. Norm must be non-negative
     $f(\mathbf{x}) \geq 0$
  4. smallest norm is achieved by a vector consisting of all zeros
     $\forall i, [\mathbf{x}]_i = 0 \Leftrightarrow f(\mathbf{x}) = 0$

Matrix Operations

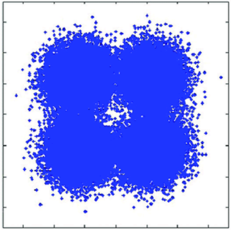
Norms

- Generalized Form: (L_p norm) $\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p}$, where $p \in \mathbb{R}, p \geq 1$
- L_2 norm is the Eucledian distance
  Suppose elements in the n-dimensional vector $\mathbf{x}$ are $x_1, \ldots, x_n$
  Then, L_2 norm of $\mathbf{x}$ is $\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$,
- L_1 norm is $\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i|$
  Compared to L_2 norm, it is less influenced by outliers
- Max norm (L_∞): absolute value of the element with the largest magnitude in the vector
  $\|\boldsymbol{x}\|_\infty = \max_i |x_i|$
- Frobenius norm of matrices is similar to L_2 Norm of vectors
  $\|\mathbf{X}\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij}^2}$, where $\mathbf{X} \in \mathbb{R}^{m \times n}$

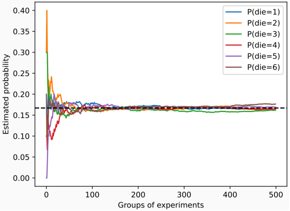
Objective functions are described as norms: minimize distance between predictions and ground-truth observations

Probability: Connection to Machine Learning

- Machine Learning is probabilistic (not deterministic)



(a) 95% QPSK



(b) Law of large numbers

- Basic Probability Theory: *law of large numbers*
- Sources of uncertainty
  - Inherent stochasticity in the system being modeled
  - Incomplete observability
  - Incomplete modeling

Basic Probability

- Sample space or outcome space: $\mathcal{S} = \{1, 2, 3, 4, 5, 6\}$, where output of random experiment is an event $\mathcal{A}$.
- Probability of an event $\mathcal{A}$ in the given sample space $\mathcal{S}$, denoted as $P(\mathcal{A})$ satisfies the following properties:
  - For any event $\mathcal{A}$, its probability is never negative, i.e., $P(\mathcal{A}) \geq 0$
  - Probability of the entire sample space is 1, or $P(\mathcal{S}) = 1$
  - For any countable sequence of events, $\mathcal{A}_1, \mathcal{A}_2, \ldots$ that are *mutually exclusive*, ($\mathcal{A}_i \cap \mathcal{A}_j = \emptyset$ for all $i \neq j$) the probability that any happens is equal to the sum of their individual probabilities, or $P(\bigcup_{i=1}^{\infty} \mathcal{A}_i) = \sum_{i=1}^{\infty} P(\mathcal{A}_i)$

Random Variables: A random variable is a variable that can take on different values randomly. Random variables may be discrete (integer or states) or continuous (real numbers).

Probability Distributions

- A *probability distribution* is a description of how likely a random variable or set of random variables is to take on each of its possible states.
- The probability that $\mathrm{x} = x$ is denoted as $P(x)$
- *Probability mass function (PMF)*: A probability distribution over discrete variables
- *Probability density function (PDF)*: A probability distribution over continuous variables

Multiple Random Variables

- Joint Probability: $P(A = a, B = b)$, where $P(A = a, B = b) \leq P(A = a)$
- Conditional Probability: $P(B = b \mid A = a)$ is the ratio $0 \leq \frac{P(A=a,B=b)}{P(A=a)} \leq 1$
  - probability of $B = b$, provided that $A = a$ has occurred
  - used in causal modeling

Bayes' Theorem

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

- The probability of two events A and B happening is $P(A \cap B) = P(A)P(B \mid A)$
- Similarly, $P(B \cap A) = P(B)P(A \mid B)$
- Equating them, $P(B)P(A \mid B) = P(A)P(B \mid A)$
- Hence, $P(A \mid B) = \frac{P(B|A)P(A)}{P(B)}$

Expectation, Variance and Covariance

To summarize key characteristics of probability distributions, we need some measures.

- The *expectation*, or expected value, of some function $f(x)$ with respect to a probability distribution $P(x)$ is the average, or mean value, that f takes on when x is drawn from P

$$\mathbb{E}_{x \sim P}[f(x)] = \sum_x f(x)P(x)$$

- The *variance* gives a measure of how much the values of a function of a random variable x vary as we sample different values of x from its probability distribution

$$\text{Var}[X] = \mathbb{E}\left[(X - \mathbb{E}[X])^2\right] = \mathbb{E}[X^2] - \mathbb{E}[X]^2$$

- *Covariance* describes how much two values are linearly related to each other

$$\text{Cov}(f(x), g(y)) = \mathbb{E}[(f(x) - \mathbb{E}[f(x)])(g(y) - \mathbb{E}[g(y)])]$$

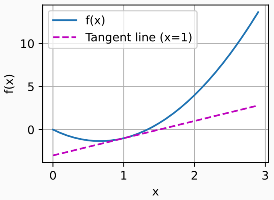
Differential Calculus

Optimization in neural networks uses *Differential Calculus*

- If a function $f : \mathbb{R} \rightarrow \mathbb{R}$ has scalar input and output
- *Derivative* of f is

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- if $f'(a)$ exists, f is said to be differentiable at a
- The derivative $f'(x)$ is instantaneous rate of change of $f(x)$ with respect to x.
- Common notations: $f'(x) = y' = \frac{dy}{dx} = \frac{df}{dx} = \frac{d}{dx}f(x) = Df(x) = D_x f(x)$
- Example: $f(x) = x^2 - 3x, f'(x) = 2x - 3$ is the tangent



Rules of Differentiation

- Constant Multiple Rule

$$\frac{d}{dx}[Cf(x)] = C\frac{d}{dx}f(x)$$

- Sum Rule

$$\frac{d}{dx}[f(x) + g(x)] = \frac{d}{dx}f(x) + \frac{d}{dx}g(x)$$

- Product Rule

$$\frac{d}{dx}[f(x)g(x)] = f(x)\frac{d}{dx}[g(x)] + g(x)\frac{d}{dx}[f(x)]$$

- Quotient Rule

$$\frac{d}{dx}\left[\frac{f(x)}{g(x)}\right] = \frac{g(x)\frac{d}{dx}[f(x)] - f(x)\frac{d}{dx}[g(x)]}{[g(x)]^2}$$

Partial Derivatives and Gradients

- In deep learning, functions depend on many variables
- In a multivariate function, $y = f(x_1, x_2, \ldots, x_n)$

$$\frac{\partial y}{\partial x_i} = \lim_{h \to 0} \frac{f(x_1, \ldots, x_{i-1}, x_i + h, x_{i+1}, \ldots, x_n) - f(x_1, \ldots, x_i, \ldots, x_n)}{h}$$

- Notations: $\frac{\partial y}{\partial x_i} = \frac{\partial f}{\partial x_i} = f_{x_i} = f_i = D_i f = D_{x_i} f$
- The *gradient vector* of a multivariate function is concatenated partial derivatives of the function with respect to all its variables, the gradient is
- If $\mathbf{x} = [x_1, x_2, \ldots, x_n]^\top$ is a vector,

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \nabla f(\mathbf{x}) = \left[\frac{\partial f(\mathbf{x})}{\partial x_1}, \frac{\partial f(\mathbf{x})}{\partial x_2}, \ldots, \frac{\partial f(\mathbf{x})}{\partial x_n}\right]^\top$$

Chain Rule

- Multivariate functions in deep learning are composite
- Chain rule enables us to differentiate composite functions
- If $y = f(u)$ and $u = g(x)$, then

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$$

- For arbitrary number of variables, (y has variables $u_1, u_2, \ldots, u_m$ and x has variable $x_1, x_2, \ldots, n_m$)

$$\frac{dy}{dx_i} = \frac{dy}{du_1}\frac{du_1}{dx_i} + \frac{dy}{du_2}\frac{du_2}{dx_i} + \cdots + \frac{dy}{du_m}\frac{du_m}{dx_i}$$

Linear Neural Networks

Linear Regression (Statistics)

- *Regression*: A set of methods for modeling the relationship between one or more independent variables and a dependent variable
- Assumptions:
  - Relationship between the independent variables x and the dependent variable y is linear
  - Noise is Gaussian
- Example: House price depends on *features* (age and area)

$$\text{price} = w_{\text{area}} \cdot \text{area} + w_{\text{age}} \cdot \text{age} + b, \quad \text{where } w_{area} \,\&\, w_{age} \text{ are } \textit{weights} \text{ and } b \text{ is } \textit{bias}$$

- For d features, the prediction, $\hat{y} = w_1 x_1 + \ldots + w_d x_d + b$
- In linear algebra notations,

$$\hat{y} = \mathbf{w}^\top \mathbf{x} + b, \text{where features of single data examples } \mathbf{x} \in \mathbb{R}^d \text{ and weights } \mathbf{w} \in \mathbb{R}^d$$

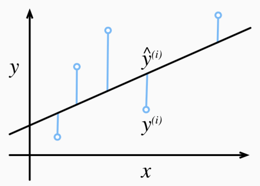
- For a collection of features $\mathbf{X}$, the predictions $\hat{\mathbf{y}} \in \mathbb{R}^n$ can be expressed via the matrix-vector product

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w} + b$$

Loss Function

- Measure of fitness: quantifies the distance between the real and predicted value of the target
- Usually a non-negative number, smaller is better, 0 for perfect prediction
- Most popular loss function in regression problems is the squared error

$$l^{(i)}(\mathbf{w}, b) = \frac{1}{2}\left(\hat{y}^{(i)} - y^{(i)}\right)^2$$



- Mean Squared Error (MSE): Average of losses on entire dataset quantifies quality of a model

$$L(\mathbf{w}, b) = \frac{1}{n}\sum_{i=1}^{n} l^{(i)}(\mathbf{w}, b) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{2}\left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)}\right)^2$$

- Objectives during training: $\mathbf{w}^*, b^* = \operatorname{argmin}_{\mathbf{w},b}\ L(\mathbf{w}, b)$

Analytic Solution

- Linear regression is a simple optimization problem
- Steps to solve:
  – Subsume the bias b into the parameter $\mathbf{w}$ by appending a column to the design matrix consisting of all ones
  – Prediction problem is to minimize $\|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2$
  – Taking the derivative of the loss w.r.t. $\mathbf{w}$ and setting it to 0, yields closed form solution

$$\mathbf{w}^* = (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{y}$$

- *Linear regression is extremely simple and limited learning algorithm*

Gradient Descent as Optimization Algorithm

- Optimization refers to the task of either minimizing or maximizing some function $f(x)$ by altering x
- In deep learning, most popular optimization is done by *Gradient Descent* [Cauchy, 1847]
  – Reduce the error by updating the parameters in the direction that iteratively lowers the loss function
  – Requires taking the derivative of the loss function, which is an average of the losses computed on *entire dataset*
  – Extremely slow in practice
- *Minibatch Stochastic Gradient Descent*: sample a random *minibatch* of examples every time there is a need to compute the update

Minibatch Stochastic Gradient Descent

- Randomly sample a minibatch $\mathcal{B}$ consisting of a fixed number of training examples
- Compute derivative (gradient) of average loss on minibatch with regard to model parameters
- Multiply the gradient by a predetermined positive value η
- Subtract the resulting term from the current parameter values

$$(\mathbf{w}, b) \leftarrow (\mathbf{w}, b) - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \partial_{(\mathbf{w}, b)} l^{(i)}(\mathbf{w}, b), \text{ where } |\mathcal{B}| \text{ is number of examples in each minibatch \& } \eta \text{ is learning rate}$$

Minibatch Stochastic Gradient Descent

- Randomly sample a minibatch $\mathcal{B}$ consisting of a fixed number of training examples
- Compute derivative (gradient) of average loss on minibatch with regard to model parameters
- Multiply the gradient by a predetermined positive value η
- Subtract the resulting term from the current parameter values

$$(\mathbf{w}, b) \leftarrow (\mathbf{w}, b) - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \partial_{(\mathbf{w}, b)} l^{(i)}(\mathbf{w}, b), \text{ where } |\mathcal{B}| \text{ is number of examples in each minibatch \& } \eta \text{ is learning rate}$$

- Steps of the algorithm:
  (i) initialize the values of the model parameters, typically randomly
  (ii) iteratively sample random minibatches from the data, updating the parameters in the direction of the negative gradient

$$\mathbf{w} \leftarrow \mathbf{w} - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \partial_{\mathbf{w}} l^{(i)}(\mathbf{w}, b) = \mathbf{w} - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \mathbf{x}^{(i)} \left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)}\right),$$

$$b \leftarrow b - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \partial_b l^{(i)}(\mathbf{w}, b) = b - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)}\right).$$

Minibatch Stochastic Gradient Descent

- Randomly sample a minibatch $\mathcal{B}$ consisting of a fixed number of training examples
- Compute derivative (gradient) of average loss on minibatch with regard to model parameters
- Multiply the gradient by a predetermined positive value η
- Subtract the resulting term from the current parameter values

$$(\mathbf{w}, b) \leftarrow (\mathbf{w}, b) - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \partial_{(\mathbf{w}, b)} l^{(i)}(\mathbf{w}, b), \text{ where } |\mathcal{B}| \text{ is number of examples in each minibatch \& } \eta \text{ is learning rate}$$

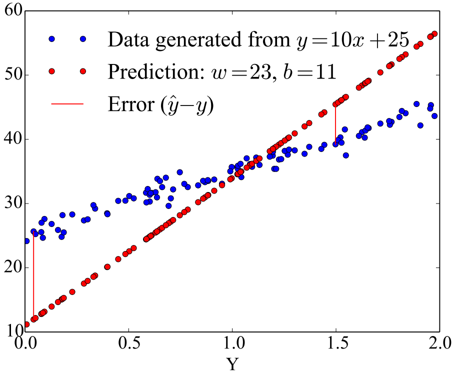
- Steps of the algorithm:
  (i) initialize the values of the model parameters, typically randomly
  (ii) iteratively sample random minibatches from the data, updating the parameters in the direction of the negative gradient

$$\mathbf{w} \leftarrow \mathbf{w} - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \partial_{\mathbf{w}} l^{(i)}(\mathbf{w}, b) = \mathbf{w} - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \mathbf{x}^{(i)} \left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)}\right),$$

$$b \leftarrow b - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \partial_b l^{(i)}(\mathbf{w}, b) = b - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)}\right).$$

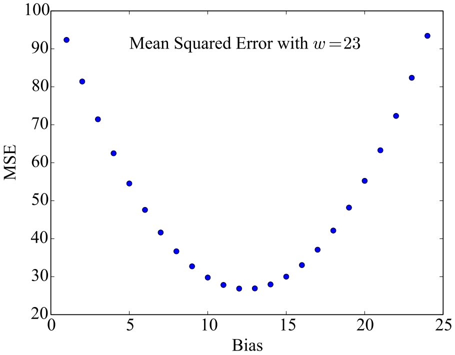
- The values of $|\mathcal{B}|$ and η are manually pre-specified and not learned through model training

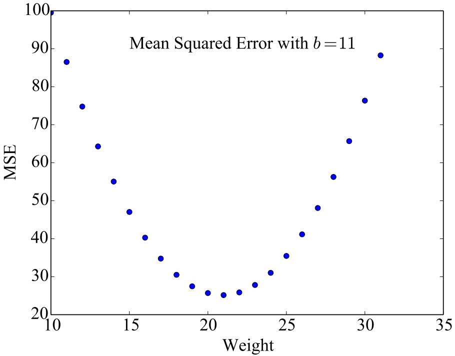
Linear Regression Example



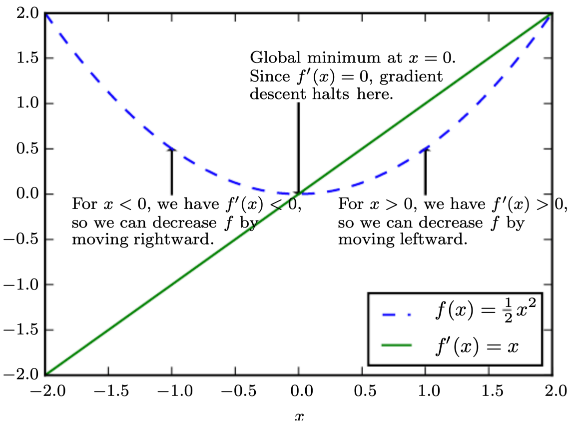

```
#Generate Data (blue dots)
X = 2 * np.random.rand(100,1)
Y = 25 + 10 * X+np.random.randn(100,1)


#Random Prediction (red dots)
weight = 23.0
bias = 11.0
Y_hat = bias + weight * X


#Error (red lines)
error = Y_hat - Y
sq_err = 0.5 * pow((Y - Y_hat),2)
mse = np.mean(sq_err)
```

The Gradient

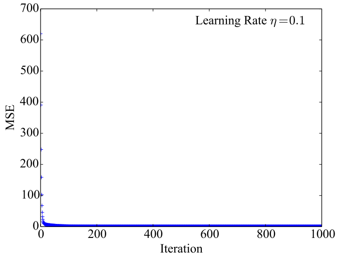


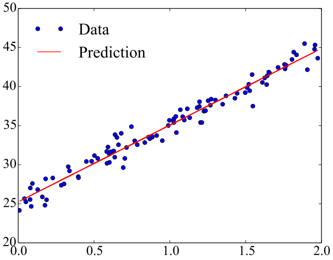


Gradient Descent



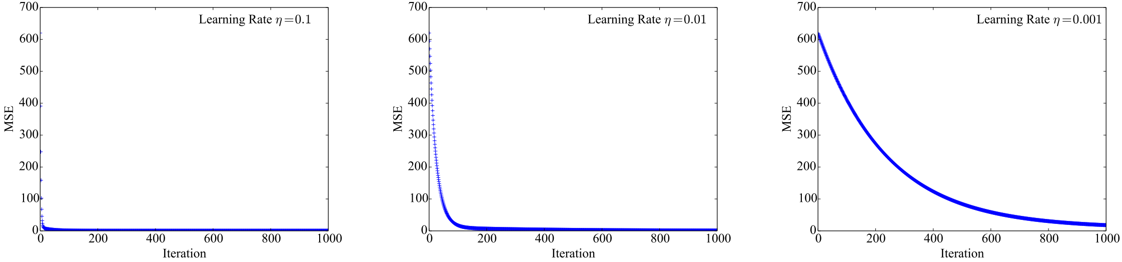
Calculate Gradient and Update






```
for i in range(1000):
    #Calculate the gradient
    err = Y-Y_hat
    grad_bias = -(err)
    grad_weight = -(err)*X

    #Mean of the gradients
    g_w = np.mean(grad_weight)
    g_b = np.mean(grad_bias)

    #Update the weight and bias
    weight = weight - rate * g_w
    bias = bias - rate * g_b
```

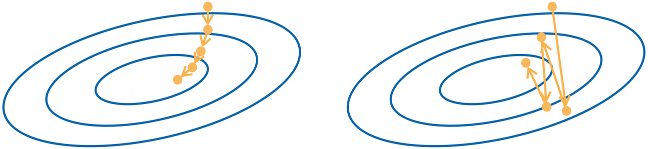
The Learning Rate



- *Learning Rate* and *Batch Size* are tunable but not updated in the training loop
- They are called *Hyperparameters*
- Hyperparameter tuning is the process by which hyperparameters are chosen
- Hyperparameters are adjusted based on the results of the training loop

Choice of Hyperparameters

- Learning Rate:
  - Too small will be too slow, too large might oscillate

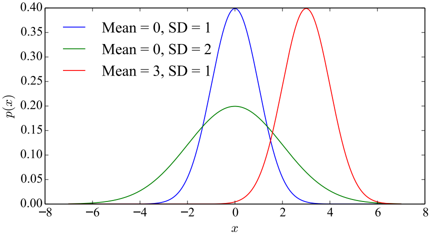


- Batch Size:
  - Too small: Workload is too small, hard to fully utilize computation resources
  - Too large: Memory issues, Wasteful computation when x_i are identical

Motivation behind Squared Loss

- Probability Density Function (PDF) of a normal distribution with mean μ and variance σ^2

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right)$$



- Changing the mean corresponds to a shift along the X-axis
- Increasing the variance spreads distribution out, lowering its peak

Motivation behind Squared Loss

- Noise in observations has a normal distribution $y = \mathbf{w}^\top \mathbf{x} + b + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma^2)$

Motivation behind Squared Loss

- Noise in observations has a normal distribution $y = \mathbf{w}^\top \mathbf{x} + b + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma^2)$
- Likelihood of seeing a particular y for a given $\mathbf{x}$

$$P(y \mid \mathbf{x}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(y - \mathbf{w}^\top \mathbf{x} - b)^2\right)$$

- According to the principle of *maximum likelihood*, the best values of parameters **w** and **b** are those that maximize the likelihood of the entire dataset $P(\mathbf{y} \mid \mathbf{X}) = \prod_{i=1}^{n} p(y^{(i)}|\mathbf{x}^{(i)})$

Motivation behind Squared Loss

- Noise in observations has a normal distribution $y = \mathbf{w}^\top \mathbf{x} + b + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma^2)$
- Likelihood of seeing a particular y for a given $\mathbf{x}$

$$P(y \mid \mathbf{x}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(y - \mathbf{w}^\top \mathbf{x} - b)^2\right)$$

- According to the principle of *maximum likelihood*, the best values of parameters **w** and **b** are those that maximize the likelihood of the entire dataset $P(\mathbf{y} \mid \mathbf{X}) = \prod_{i=1}^{n} p(y^{(i)} | \mathbf{x}^{(i)})$
- *Maximum Likelihood Estimators*: Estimators chosen according to the principle of maximum likelihood
- Maximizing product of many exponential functions could be a hard problem
- Instead we can choose maximizing the log of the likelihood that does not change the objective
- Convert to minimization by taking *negative log-likelihood*

$$-\log P(\mathbf{y} \mid \mathbf{X}) = \sum_{i=1}^{n} \frac{1}{2} \log(2\pi\sigma^2) + \frac{1}{2\sigma^2} \left(y^{(i)} - \mathbf{w}^\top \mathbf{x}^{(i)} - b\right)^2$$

Motivation behind Squared Loss

- Noise in observations has a normal distribution $y = \mathbf{w}^\top \mathbf{x} + b + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma^2)$
- Likelihood of seeing a particular y for a given $\mathbf{x}$

$$P(y \mid \mathbf{x}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(y - \mathbf{w}^\top \mathbf{x} - b)^2\right)$$

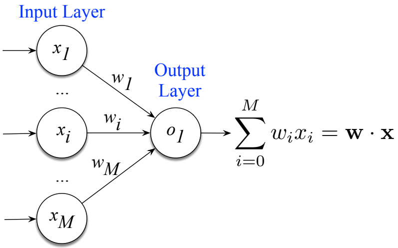
- According to the principle of *maximum likelihood*, the best values of parameters **w** and **b** are those that maximize the likelihood of the entire dataset $P(\mathbf{y} \mid \mathbf{X}) = \prod_{i=1}^{n} p(y^{(i)}|\mathbf{x}^{(i)})$
- *Maximum Likelihood Estimators*: Estimators chosen according to the principle of maximum likelihood
- Maximizing product of many exponential functions could be a hard problem
- Instead we can choose maximizing the log of the likelihood that does not change the objective
- Convert to minimization by taking *negative log-likelihood*

$$-\log P(\mathbf{y} \mid \mathbf{X}) = \sum_{i=1}^{n} \frac{1}{2} \log(2\pi\sigma^2) + \frac{1}{2\sigma^2}\left(y^{(i)} - \mathbf{w}^\top \mathbf{x}^{(i)} - b\right)^2$$

- Assume σ is a constant, it is identical to squared loss

Single Layer Neural Network (NN): Linear Regression

- Input Layer: Number of inputs = M = Feature Dimensionality, given
- Output Layer: Number of outputs = 1
- One single computed neuron
- Number of layers of the NN = 1 [input layer is not counted]
- Linear Regression can be cast in this NN
- *Fully-connected layer* or *Dense layer*: when all inputs are connected to all outputs



Connection to Neurons

- Information x_i arriving from other neurons (or environmental sensors such as the retina) is received in the dendrites.
- That information is weighted by synaptic weights w_i determining the effect of the inputs (e.g., activation or inhibition via the product $x_i w_i$).
- The weighted inputs arriving from multiple sources are aggregated in the nucleus as a weighted sum $y = \sum_i x_i w_i + b$.
- This information is then sent for further processing in the axon y, after some nonlinear processing via $\sigma(y)$.
- From there it either reaches its destination (e.g., a muscle) or is fed into another neuron via its dendrites.

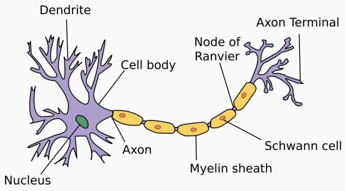
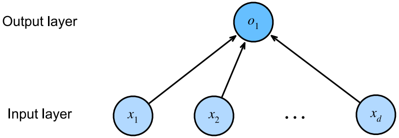


Figure: Dendrites (input terminals), the nucleus (CPU), the axon (output wire), and the axon terminals (output terminals), enables connections to other neurons via synapses

Regression vs Classification

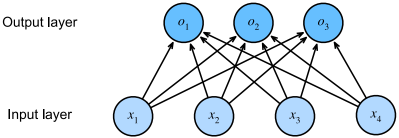
Regression: estimates a continuous value

- How much?
- Example Application: Channel estimation
- Single continuous output
- Natural scale in $\mathbb{R}$
- Loss in terms of difference: $y - f(x)$



Classification: predicts a discrete category

- Which one?
- Example Application: Modulation recognition, RF fingerprinting
- Multiple classes, typically multiple outputs
- Score should reflect confidence



Multi-class Classification

- *One-hot Encoding*: A one-hot encoding is a vector with as many components as the number of categories
- The component corresponding to particular instance's category is set to 1 and all other components are set to 0
- Example: QPSK has 4 constellation points

Constellation	**Hot-one Encoded Labels**
1+j	(1, 0, 0, 0)
-1+j	(0, 1, 0, 0)
-1-j	(0, 0, 1, 0)
1-j	(0, 0, 0, 1)

Network Architecture

- Estimate the conditional probabilities associated with all the possible classes
- A model with multiple outputs, one per class
- With linear models, we need as many affine functions as we have outputs
- Each output will correspond to its own affine function
- Example: 4 features and 3 output categories
- 12 scalars to represent the weights, 3 scalars to represent the biases

$$
\begin{aligned}
o_1 &= x_1 w_{11} + x_2 w_{12} + x_3 w_{13} + x_4 w_{14} + b_1,\\
o_2 &= x_1 w_{21} + x_2 w_{22} + x_3 w_{23} + x_4 w_{24} + b_2,\\
o_3 &= x_1 w_{31} + x_2 w_{32} + x_3 w_{33} + x_4 w_{34} + b_3.
\end{aligned}
$$

$$\mathbf{o} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

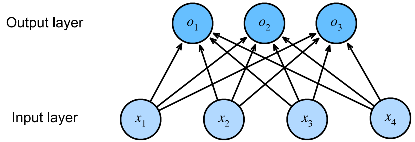


Figure: Softmax regression is a fully connected single-layer neural network

Softmax Operation

- Interpret the outputs of the model as probabilities
- Optimize parameters to produce probabilities that maximizes the likelihood of the observed data
- Need output $\hat{y}_j$, to be interpreted as the probability that a given item belongs to class j
- Choose the class with the largest output value as the prediction $\operatorname{argmax}_j y_j$
- For example, $\hat{y}_1 = 0.1$, $\hat{y}_2 = 0.8$ and $\hat{y}_3 = 0.1$ indicates output is Category 2.

Softmax Operation

- Goals:
  - Logits should not be negative
  - Sum of all logits equals 1
  - Model should be differentiable
- Softmax function:

$$\hat{\mathbf{y}} = \mathrm{softmax}(\mathbf{o}) \quad \text{where} \quad \hat{y}_j = \frac{\exp(o_j)}{\sum_k \exp(o_k)}.$$

- Exponential function, $\exp(o_j)$ ensures non negativity
- Dividing by their sum $\sum_k \exp(o_k)$ ensures they sum up to 1.
- Prediction:

$$\operatorname*{argmax}_j \hat{y}_j = \operatorname*{argmax}_j o_j.$$

Loss Function

- The softmax function outputs a vector $\hat{\mathbf{y}}$, which can be interpreted as estimated conditional probabilities of each class given any input $\mathbf{x}$
$$\hat{y} = P(y = \text{Category } 1 \mid \mathbf{x})$$
- For the entire dataset, $P(\mathbf{Y} \mid \mathbf{X}) = \prod_{i=1}^{n} P(\mathbf{y}^{(i)} \mid \mathbf{x}^{(i)})$
- According to maximum likelihood estimation, we maximize $P(\mathbf{Y} \mid \mathbf{X})$
- This is equivalent to minimizing the negative log-likelihood
$$-\log P(\mathbf{Y} \mid \mathbf{X}) = \sum_{i=1}^{n} -\log P(\mathbf{y}^{(i)} \mid \mathbf{x}^{(i)}) = \sum_{i=1}^{n} l(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)})$$
- For any pair of label $\mathbf{y}$ and model prediction $\hat{\mathbf{y}}$ over q classes, the loss function l is
$$l(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{j=1}^{q} y_j \log \hat{y}_j \qquad \text{– \textit{Cross Entropy Loss}}$$
- $\mathbf{y}$ one-hot vector of length q, so sum over all its coordinates j vanishes for all except one term
- $\hat{y}_j$ are probabilities, so their logarithms are never greater than 0.

Softmax and Derivatives

Loss:

$$l(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{j=1}^{q} y_j \log \frac{\exp(o_j)}{\sum_{k=1}^{q} \exp(o_k)}$$

$$= \sum_{j=1}^{q} y_j \log \sum_{k=1}^{q} \exp(o_k) - \sum_{j=1}^{q} y_j o_j$$

$$= \log \sum_{k=1}^{q} \exp(o_k) - \sum_{j=1}^{q} y_j o_j.$$

Derivative:

$$\partial_{o_j} l(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\exp(o_j)}{\sum_{k=1}^{q} \exp(o_k)} - y_j = \mathrm{softmax}(\mathbf{o})_j - y_j.$$

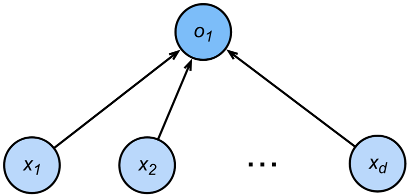
Gradient is the difference between the observation and estimate

Multi Layer Perceptron

Perceptron

- An algorithm for supervised learning of binary classifiers
- Binary classification outputs 0 or 1
  - vs. scalar real value for regression
  - vs. probabilities for logistic regression
- Given input **x**, weight **w** and bias b, perceptron outputs:

$$o = \sigma\left(\langle \mathbf{w}, \mathbf{x} \rangle + b\right) \qquad \sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$



Binary Classification

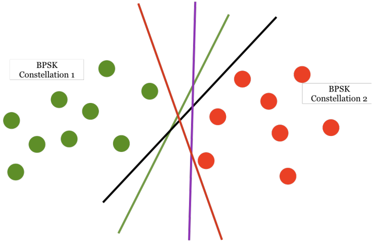


Figure: Example Application: Decoding BPSK

initialize $w = 0$ and $b = 0$
repeat
  if $y_i \left[\langle w, x_i \rangle + b\right] \leq 0$ **then**
    $w \leftarrow w + y_i x_i$ and $b \leftarrow b + y_i$
  end if
until all classified correctly

- Equals to SGD (batch size is 1) with the following loss

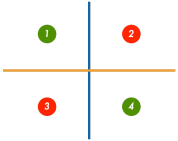
$$\ell(y, \mathbf{x}, \mathbf{w}) = \max(0, -y\langle \mathbf{w}, \mathbf{x} \rangle)$$

- Convergence Theorem: If a data set is linearly separable, the Perceptron will find a separating hyperplane in a finite number of updates.

Assumption of Linearity

Linearity is a strong assumption

- XOR Problem (Minsky Papert, 1969): A perceptron cannot learn an XOR function

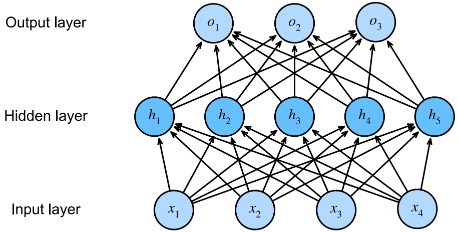


	1	2	3	4
	+	-	+	-
	+	+	-	-
product	+	-	-	+

- Wireless domain:
  – Frequency response of a multipath channel
  – Non-linear region of hardware

Multilayer Perceptrons or Feed Forward Networks

- Multiple layers of perceptrons
- The goal of a feedforward network is to approximate some function f^*
- Defines a mapping $y = f(x;\theta)$ and learns the value of the parameters θ that result in the best function approximation.
- These models are called feedforward
  – Information flows through the function being evaluated from x
  – Intermediate computations used to define f
  – Outputs y
  – No feedback connections

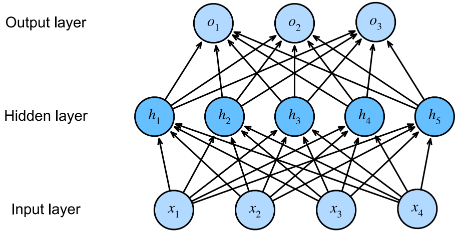


Feed Forward Network

- Called *networks* as they represent composition of many functions
- Model can be represented by Directed Acyclic Graph
- Functions connected in a chain: $f(\boldsymbol{x}) = f^{(3)}\left(f^{(2)}\left(f^{(1)}(\boldsymbol{x})\right)\right)$
- $f(i)$ is called the i^{th} layer of the network
- Length of chain is the *depth* of the network
  – It is a *Hyperparameter*
- Final layer is the *output* layer
- Layers in between input and output are *hidden* layers
  – training data does not show any output for each of these layers

Single Hidden Layer

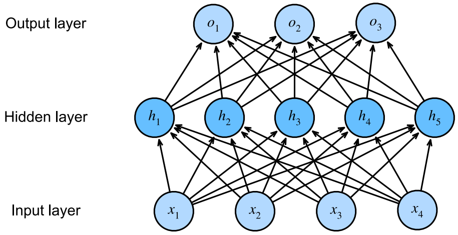
- Minibatch of n examples, each example has d inputs (features)
- Input: $\mathbf{X} \in \mathbb{R}^{n \times d}$
- One hidden layer with h hidden units.
- Hidden variable: $\mathbf{H} \in \mathbb{R}^{n \times h}$
- Hidden layer
  - weights: $\mathbf{W}^{(1)} \in \mathbb{R}^{d \times h}$
  - biases: $\mathbf{b}^{(1)} \in \mathbb{R}^{1 \times h}$
- Output layer with q units.
- Output layer
  - weights: $\mathbf{W}^{(2)} \in \mathbb{R}^{h \times q}$
  - biases: $\mathbf{b}^{(2)} \in \mathbb{R}^{1 \times q}$



Single Hidden Layer

- Outputs $\mathbf{O} \in \mathbb{R}^{n \times q}$

$$\mathbf{H} = \mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)},$$
$$\mathbf{O} = \mathbf{H}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}.$$



- Adding hidden layer requires tracking and updating additional sets of parameters
- Rewriting :

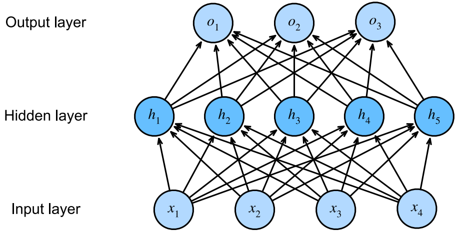
$$\mathbf{O} = (\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)})\mathbf{W}^{(2)} + \mathbf{b}^{(2)} = \mathbf{X}\mathbf{W}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(2)} = \mathbf{X}\mathbf{W} + \mathbf{b}$$

- Can be represented by a single layer neural network

Single Hidden Layer

- Outputs $\mathbf{O} \in \mathbb{R}^{n \times q}$

$$\mathbf{H} = \mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)},$$
$$\mathbf{O} = \mathbf{H}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}.$$



- Adding hidden layer requires tracking and updating additional sets of parameters
- Rewriting :

$$\mathbf{O} = (\mathbf{X}\mathbf{W}^{(1)}+\mathbf{b}^{(1)})\mathbf{W}^{(2)}+\mathbf{b}^{(2)} = \mathbf{X}\mathbf{W}^{(1)}\mathbf{W}^{(2)}+\mathbf{b}^{(1)}\mathbf{W}^{(2)}+\mathbf{b}^{(2)} = \mathbf{X}\mathbf{W}+\mathbf{b}$$

- Can be represented by a single layer neural network
- Still a Linear Model

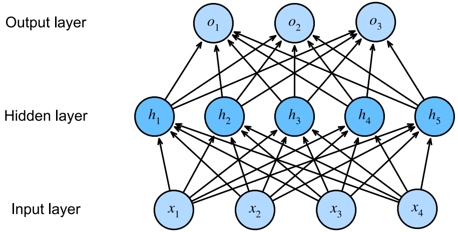
Non Linear Activation Function

- A nonlinear activation function σ should be applied to each hidden unit following the affine transformation
- With activation functions, it is no longer possible to collapse our MLP into a linear model

$$\mathbf{H} = \sigma(\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)}),$$
$$\mathbf{O} = \mathbf{H}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}.$$

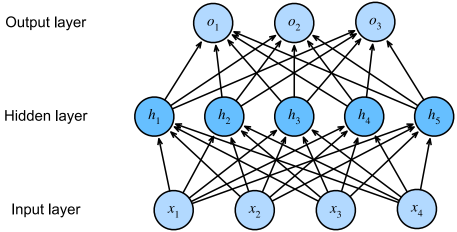
- In general, with more hidden layers

$$\mathbf{H}^{(1)} = \sigma_1(\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)})$$
$$\mathbf{H}^{(2)} = \sigma_2(\mathbf{H}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(2)})$$



Activation Functions

- Decide whether a neuron should be activated or not by calculating the weighted sum and further adding bias with it.
- Need to be differentiable operators to transform input signals to outputs
- Adds non-linearity to the model



ReLU Function

- Rectified Linear Unit (ReLU): simple nonlinear transformation
- Given an element x, the function is defined as the maximum of that element and 0
- $\text{ReLU}(x) = \max(x, 0)$
- ReLU is piecewise linear
- Deivative: $f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases}$, undefined at 0.
- At $x = 0$, we default to the left-hand-side derivative

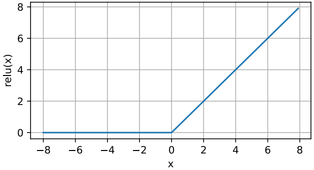


Figure: ReLU

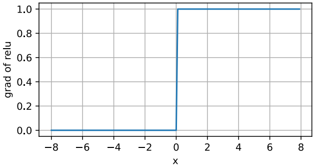
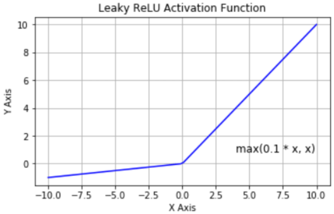


Figure: Derivative of ReLU

Generalization of ReLU

- Generalizations are based on the principle that models are easier to optimize if their behavior is closer to linear
- Add non-linear slope: α_i, when $x_i < 0$
- $h_i = g(\boldsymbol{x}, \boldsymbol{\alpha})_i = \max(0, x_i) + \alpha_i \min(0, x_i)$
- *Leaky ReLU*: Fixes α_i to a small value like 0.01 (Maas et al., 2013)
- *parameterized ReLU (pReLU)*: Treats α_i as a learnable parameter (He et al., 2015)
- *Maxout units*: Divides $\boldsymbol{x}$ into groups of k values (Goodfellow et al., 2013) $g(\boldsymbol{x})_i = \max_{j \in \mathbb{G}^{(i)}} x_j$



Sigmoid Function

- Transforms its inputs from domain $\mathbb{R}$ to outputs that lie on the interval (0, 1).

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

- Smooth, differentiable approximation to a thresholding unit
- Derivative of the Sigmoid function:

$$\frac{d}{dx}\text{sigmoid}(x) = \frac{\exp(-x)}{(1+\exp(-x))^2} = \text{sigmoid}(x)\left(1-\text{sigmoid}(x)\right).$$

- When the input is 0, the derivative of the sigmoid function reaches a maximum of 0.25.

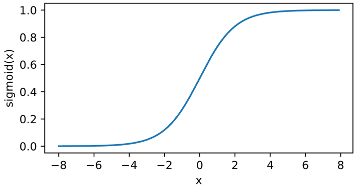


Figure: Sigmoid

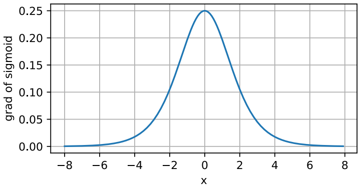


Figure: Gradient of Sigmoid

Tanh (Hyperbolic Tangent) Function

- Transforms its inputs from domain $\mathbb{R}$ to outputs that lie on the interval (-1, 1).

$$\tanh(x) = \frac{1-\exp(-2x)}{1+\exp(-2x)}$$

- As the input nears 0, it approaches a linear transformation
- Derivative of the Tanh function:

$$\frac{d}{dx}\tanh(x) = 1 - \tanh^2(x)$$

- As the input nears 0, the derivative of the tanh function approaches a maximum of 1

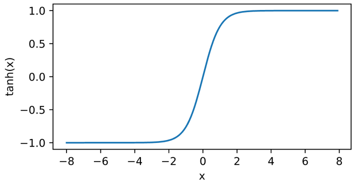


Figure: Tanh

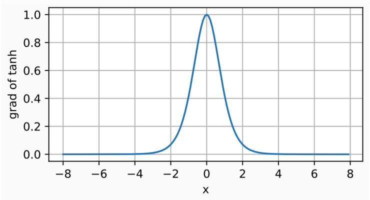
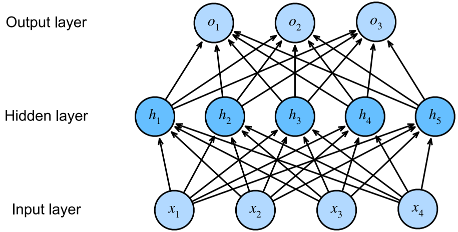


Figure: Gradient of Tanh

Multiclass Classification

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

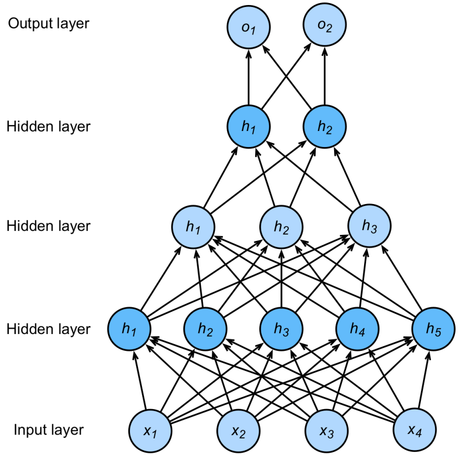
$$\mathbf{h} = \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$
$$\mathbf{o} = \mathbf{w}_2^T \mathbf{h} + \mathbf{b}_2$$
$$\mathbf{y} = \text{softmax}(o)$$



Multiclass Classification

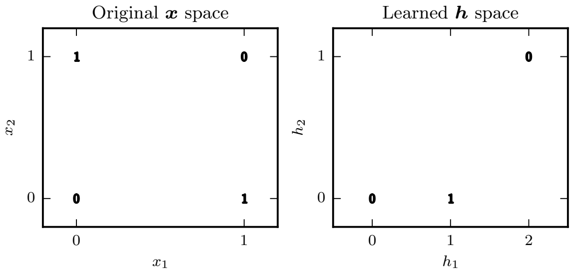
$$y_1, y_2, \ldots, y_k = \text{softmax}(o_1, o_2, \ldots, o_k)$$

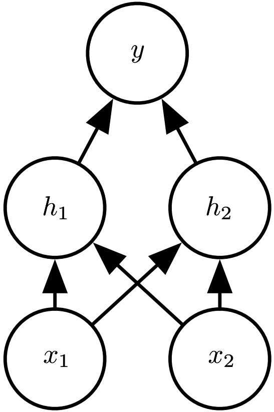
$$\mathbf{h}_1 = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$
$$\mathbf{h}_2 = \sigma(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2)$$
$$\mathbf{h}_3 = \sigma(\mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3)$$
$$\mathbf{o} = \mathbf{W}_4\mathbf{h}_3 + \mathbf{b}_4$$



Solving the XOR Problem

- One hidden layer with two hidden units
- MSE loss function
  $J(\boldsymbol{\theta}) = \frac{1}{4} \sum_{\boldsymbol{x} \in \mathbb{X}} (f^*(\boldsymbol{x}) - f(\boldsymbol{x}; \boldsymbol{\theta}))^2$
- ReLU Activation function





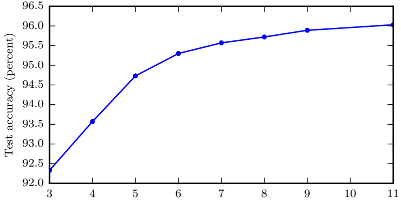
Architecture Basics

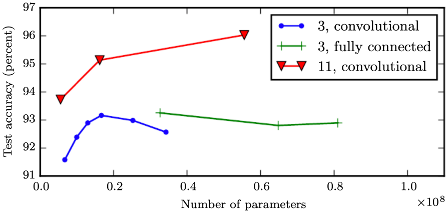
The Universal Approximation Theorem - [Hornik et al., 1989; Cybenko, 1989]

One hidden layer is enough to represent (not learn) an approximation of any function to an arbitrary degree of accuracy

So why deeper Neural Network?

- Shallow net may need (exponentially) more width
- Shallow net may overfit more





Generalization Error

- *Generalization*: ability to perform well on previously unobserved inputs
- *Training Error*: prediction error based on data available at training (like optimization)
- *Generalization Error / Testing Error*: error on data not seen during testing
- Machine Learning vs Optimization: In ML, we aim to minimize the generalization loss
- Linear Regression Training Error:

$$\frac{1}{m^{(\text{train})}} \left\| \boldsymbol{X}^{(\text{train})} \boldsymbol{w} - \boldsymbol{y}^{(\text{train})} \right\|_2^2$$

- Linear Regression Testing Error:

$$\frac{1}{m^{(\text{test})}} \left\| \boldsymbol{X}^{(\text{test})} \boldsymbol{w} - \boldsymbol{y}^{(\text{test})} \right\|_2^2$$

- Machine Learning vs Optimization: In ML, we aim to minimize the generalization loss

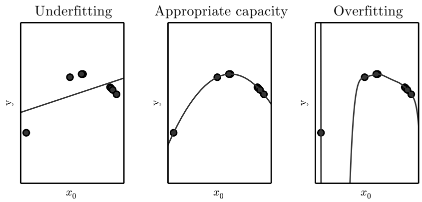
Data Distribution

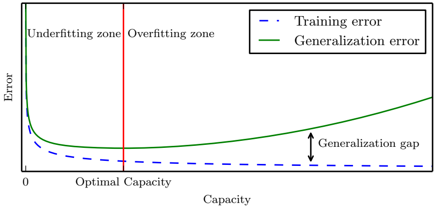
How can we affect performance on the test set when we get to observe only the training set?

- Assumptions:
  - training and test sets are not collected arbitrarily
  - Independent and identically distributed (i.i.d. or IID)
    - + examples in each dataset are independent from each other
    - + train and test set are identically distributed (drawn from same probability distribution)
- Under these assumptions: expected training set error = expected test set error
- Training process: training dataset is used to choose the parameters that minimize the chosen loss function
- Test error $\geq$ Train error
- Goal of ML algorithm:
  - Make the training error small
  - Make the gap between training and test error small

Underfitting vs Overfitting

- *Underfitting*: the model is not able to obtain a sufficiently low error value on the training set
- *Overfitting*: the gap between the training error and test error is too large
- *Capacity*: ability to fit a wide variety of functions





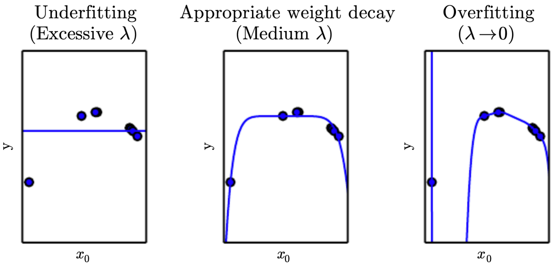
- *No Free Lunch Theorem:* [Wolpert, 1996] averaged over *all possible* data generating distributions, every classification algorithm has the same error rate when classifying previously unobserved points

Regularization

- *Weight Decay*: Reduce model complexity by limiting value range
- Original loss of linear regression:
  - minimize the prediction loss on the training labels
  - $L(\mathbf{w}, b) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{2} \left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)} \right)^2 .$
- Regularized loss:
  - minimizing the sum of the prediction loss and the penalty term
  - $L(\mathbf{w}, b) + \frac{\lambda}{2} \|\mathbf{w}\|^2$
- Weight Updates:
  - $\mathbf{w} \leftarrow (1 - \eta\lambda) \mathbf{w} - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \mathbf{x}^{(i)} \left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)} \right) .$
- Optimization algorithm *decays the weight* at each step of training

Effect of Regularization

- Train a high-degree polynomial regression model

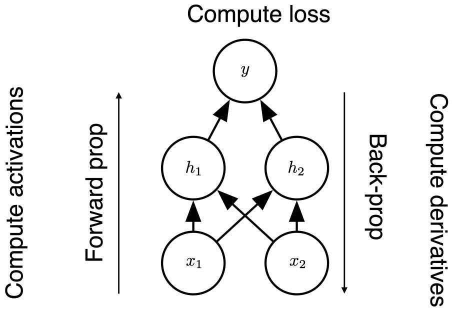


Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.

Training Neural Networks

Minibatch Stochastic Gradient Descent

- Forward Propagation: calculation and storage of intermediate variables (including outputs) in order from the input layer to the output layer
- Backward Propagation: method of calculating the gradient of neural network parameters for the weights to be updated

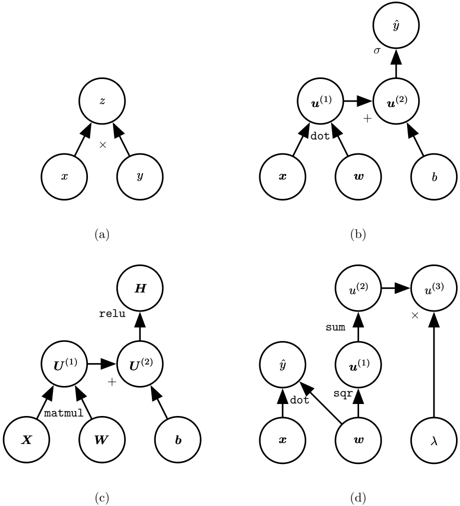


Forward Propagation

- Simple Assumptions:
  - Input: $\mathbf{x} \in \mathbb{R}^d$
  - Intermediate variable: $\mathbf{z} \in \mathbb{R}^h$
  - Weight parameter of the hidden layer: $\mathbf{W}^{(1)} \in \mathbb{R}^{h \times d}$
  - Before activation: $\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x}$, when hidden terms do not have a bias
  - After passing through activation function (ϕ):
    hidden variable $\mathbf{h} = \phi(\mathbf{z})$
  - Weight parameter of output layer: $\mathbf{W}^{(2)} \in \mathbb{R}^{q \times h}$
  - Output layer variable: $\mathbf{o} = \mathbf{W}^{(2)}\mathbf{h}$
  - Loss function: $L = l(\mathbf{o}, y)$

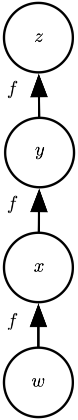
Computational Graph

- A formal way to represent math in graph theory
- Each node represents a variable (a scalar, vector, matrix, tensor) or an operation
- Operation: a simple function of one or more variables
- Directed edge: input and output relations of operators and variables
- Example
  - (a) Multiplication
  - (b) Logistic regression
  - (c) ReLU layer
  - (d) Linear regression and weight decay



Backpropagation

- A method of calculating the gradient of neural network parameters
- Computes the chain rule of calculus
- Stores any intermediate variables (partial derivatives)
- Done by table filling
- Brings down the complexity from $O\left(n^2\right)$ to $O\left(n\right)$



$$\begin{aligned}&\frac{\partial z}{\partial w}\\ &=\frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial w}\\ &=f'(y)f'(x)f'(w)\\ &=f'(f(f(w)))f'(f(w))f'(w)\end{aligned}$$

Back-prop avoids computing this twice

Backpropagation

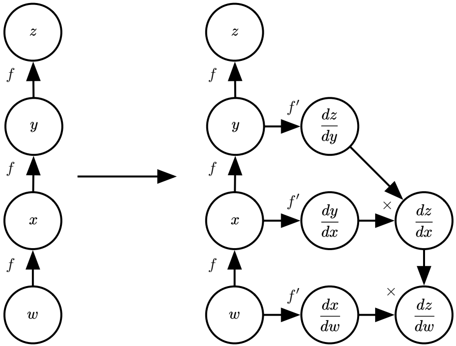


Figure: Symbol-to-symbol approach to computing derivatives

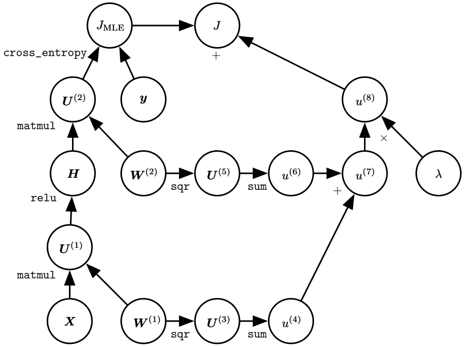
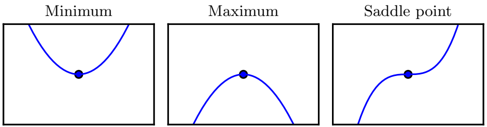
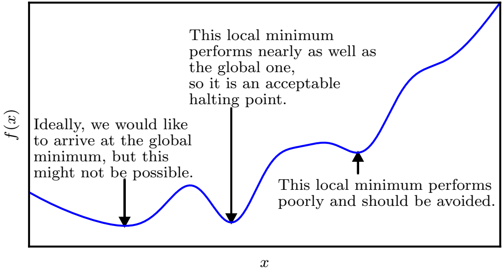


Figure: Cross entropy loss for forward propagation

Loss Function and Gradient Descent

Loss function should be convex.

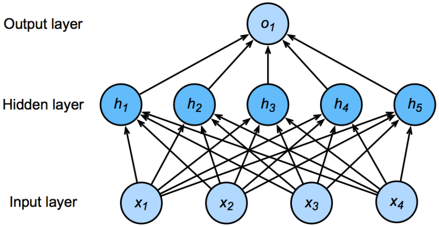




Convolutional Neural Network

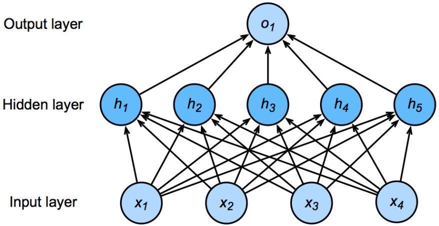
Scalability of Feed Forward Network

- Example Application: Distinguish cat and dog
  - Phone camera (12MP)
  - RGB image has 36M elements
  - Single hidden layer of 100 hidden neurons
  - The model size is 3.6 Billion parameters
  - Infeasible to learn these many parameters



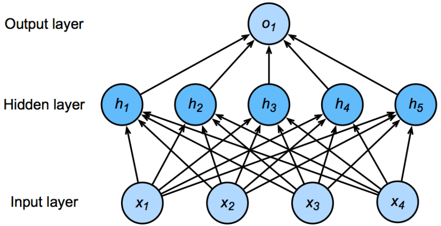
Scalability of Feed Forward Network

- Example Application: Distinguish cat and dog
  - Phone camera (12MP)
  - RGB image has 36M elements
  - Single hidden layer of 100 hidden neurons
  - The model size is 3.6 Billion parameters
  - Infeasible to learn these many parameters
- Reduce size of image (1MP)
  - Hidden layer size is underestimated
  - Requires enormous dataset



Scalability of Feed Forward Network

- Example Application: Distinguish cat and dog
  - Phone camera (12MP)
  - RGB image has 36M elements
  - Single hidden layer of 100 hidden neurons
  - The model size is 3.6 Billion parameters
  - Infeasible to learn these many parameters
- Reduce size of image (1MP)
  - Hidden layer size is underestimated
  - Requires enormous dataset
- Convolutional Neural Networks(CNNs) are designed to build efficient models



Two Principles

Translation Invariance: network should respond similarly to the same patch, regardless of where it appears in the image

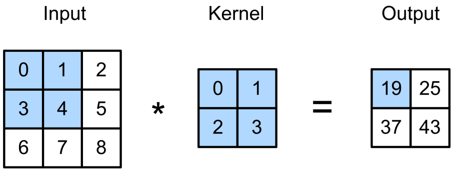
Locality: network should focus on local regions, without regard for the contents of the image in distant regions

Convolutional Neural Network

- Neural networks that use *convolution* in place of general matrix multiplication in at least one of their layers
- Convolution:
  - $s(t) = \int x(a)w(t-a)da$
  - $s(t) = (x * w)(t)$ – denoted by asterisk
  - In ML terminology, x is input and w is kernel or filter
  - Measure of the overlap between f and g when one function is "flipped" and shifted by t
  - Discrete representation:
    $s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$
  - On finite input:
    $s(t) = (x * w)(t) = \sum_a x(a)w(t-a)$
  - On two dimensional input I and kernel K,
    $S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m,j-n)$
    $S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m,j-n)K(m,n)$ – commutative, easy to implement

Convolution vs Cross Correlation

- Cross Correlation:
  $S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)$
- Same as convolution, without flipping kernel
- No difference in practice due to symmetry



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

Figure: Cross Correlation.

Convolution Layer

- Input Matrix: $\mathbf{I} : n_h \times n_w$
- Kernel Matrix: $\mathbf{K} : k_h \times k_w$
- Scalar Bias: b
- Output Matrix: $\mathbf{S} : (n_h - k_h + 1) \times (n_w - k_w + 1)$
- $\mathbf{S} = \mathbf{I} \star \mathbf{K} + b$
- $\mathbf{K}$ and b are learnable parameters
- Common choice: k_h or k_w are odd (1, 3, 5, 7).

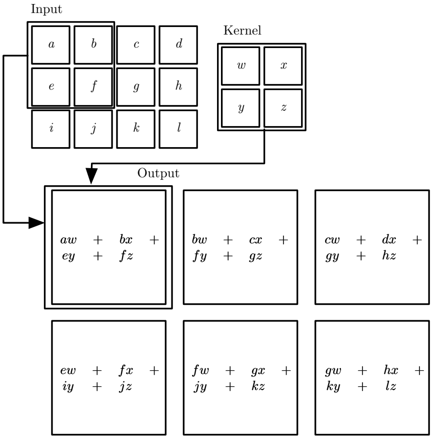


Figure: 2-D convolution without kernel-flipping.

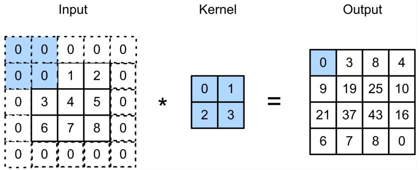
Padding

- Given a 32 x 32 input image
- Apply convolutional layer with 5 x 5 kernel
- 28 x 28 output with 1 layer
- 4 x 4 output with 7 layers
- Shape decreases faster with larger kernels
- Shape reduces from $n_h \times n_w$ to $(n_h - k_h + 1) \times (n_w - k_w + 1)$

Padding

- Given a 32 x 32 input image
- Apply convolutional layer with 5 x 5 kernel
- 28 x 28 output with 1 layer
- 4 x 4 output with 7 layers
- Shape decreases faster with larger kernels
- Shape reduces from $n_h \times n_w$ to $(n_h - k_h + 1) \times (n_w - k_w + 1)$

- *Padding* adds rows/columns around input



- Padding p_h rows and p_w columns, output shape will be $(n_h - k_h + p_h + 1) \times (n_w - k_w + p_w + 1)$
- Common choice: $p_h = k_h - 1$, $p_w = k_w - 1$
  - Odd k_h: pad $p_h/2$ on both sides
  - Even k_h: pad $\lceil p_h/2 \rceil$ on top, $\lfloor p_h/2 \rfloor$ on bottom

Stride

- Padding reduces shape linearly with number of layers:
  - a 224×224 input with a 5×5 kernel, needs 44 layers to reduce the shape to 4×4
  - Still large amount of computation
- *Stride:* number of rows and columns traversed per slide

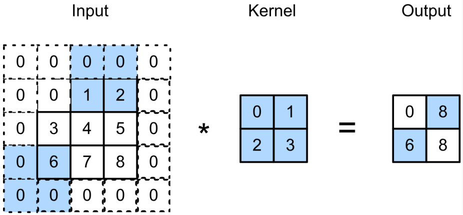


Figure: Stride 3 and 2 for height and width

- With strides s_h and s_w, output shape:
  $\lfloor (n_h - k_h + p_h + s_h)/s_h \rfloor \times \lfloor (n_w - k_w + p_w + s_w)/s_w \rfloor$
- With $p_h = k_h - 1, p_w = k_w - 1$
  $\lfloor (n_h + s_h - 1)/s_h \rfloor \times \lfloor (n_w + s_w - 1)/s_w \rfloor$
- If input height/width are divisible by strides
  $(n_h/s_h) \times (n_w/s_w)$

Convolution with stride

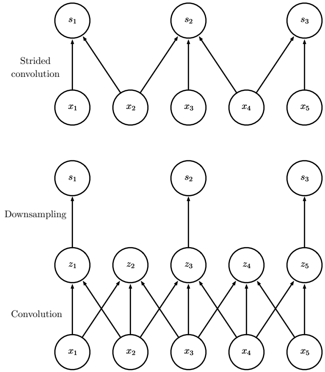


Figure: Equivalent Models

Multiple Input Channels

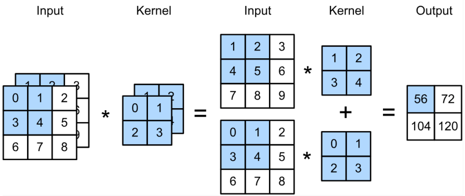
- *Channel*: another dimension in tensor
  - Wireless Signals: Real/Imaginary
  - Image: RGB
- Have a kernel for each channel, and then sum results over channels

  Input: $\mathbf{X} : c_i \times n_h \times n_w$

  Kernel: $\mathbf{W} : c_i \times k_h \times k_w$

  Output: $\mathbf{Y} : m_h \times m_w$

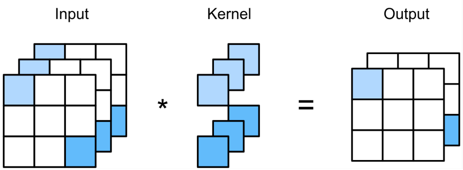
  $\mathbf{Y} = \sum_{i=0}^{c_i} \mathbf{X}_{i,:,:} \star \mathbf{W}_{i,:,:}$



$$(1\times1+2\times2+4\times3+5\times4)+(0\times0+1\times1+3\times2+4\times3) = 56$$

Multiple Output Channels

- Each output channel may recognize a different pattern
- A kernel for every output channel
  Input: $\mathbf{X} : c_i \times n_h \times n_w$
  Kernel: $\mathbf{W} : c_o \times c_i \times k_h \times k_w$
  Output: $\mathbf{Y} : c_o \times m_h \times m_w$
  $\mathbf{Y}_{i,:,:} = \mathbf{X} \star \mathbf{W}_{i,:,:,:}$ for for $i = 1, \ldots, c_o$



Pooling

Convolution is sensitive to position

```
X    [[1. 1. 0. 0. 0.        Y    [[ 0.  1.  0.  0.
      [1. 1. 0. 0. 0.              [ 0.  1.  0.  0.
      [1. 1. 0. 0. 0.              [ 0.  1.  0.  0.
      [1. 1. 0. 0. 0.              [ 0.  1.  0.  0.
```

Figure: Example: Vertical Edge Detection

- *Pooling* over spatial regions produces invariance to translation
- Summarizes the responses over a whole neighborhood
- Calculate either maximum or average value of the elements in the pooling window

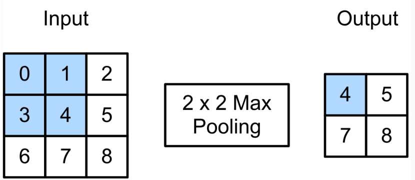
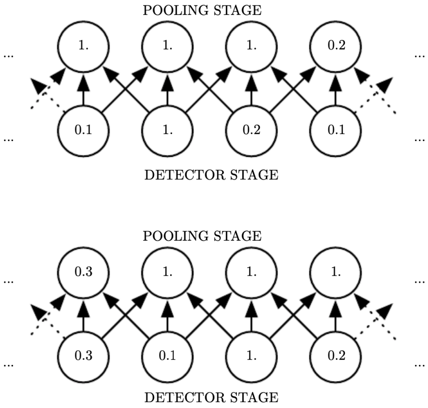
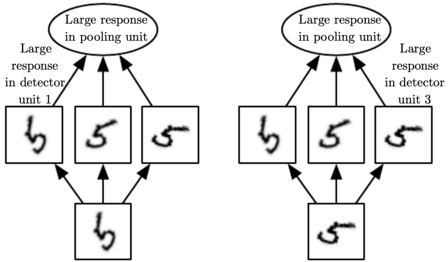


Figure: Example: Max Pool

Pooling introduces invariance





Pooling with downsampling

- Have similar padding and stride as convolutional layers
- Stride introduces downsampling
- No learnable parameters
- Apply pooling for each input channel to obtain the corresponding output channel
  No. output channels = No. input channels

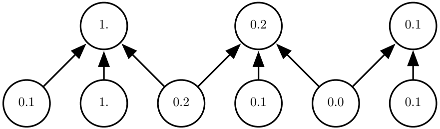
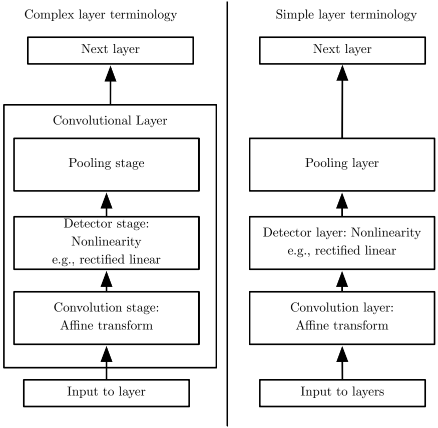


Figure: Max-pooling with a pool window of 3 and a stride of 2

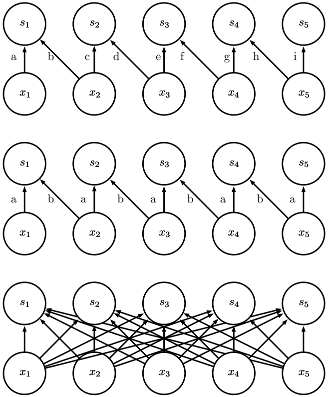
Convolutional Layer

- Consists of three stages:
  - *Convolution*: several convolutions in parallel to produce a set of linear activations
  - *Detector*: each linear activation is run through a nonlinear activation function
  - *Pooling*: replaces the output with a summary statistic of the nearby outputs

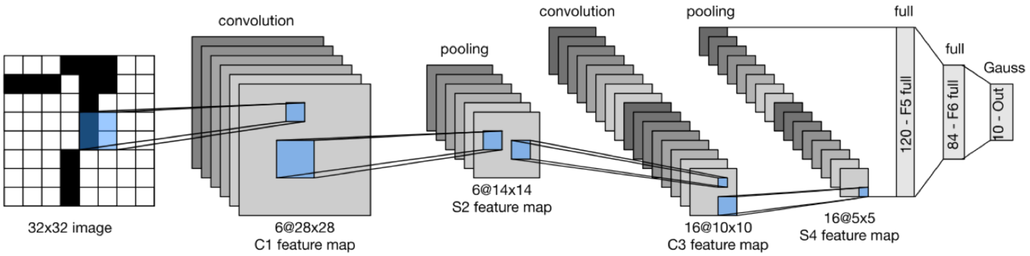


Convolution vs Fully Connected Network

- Local Connections, no sharing of parameters
- Convolution
- Fully Connected



LeNet [Yann LeCun et al. in 1989]



Modern Convolutional Neural Networks

- LeNet (the first convolutional neural network)
- AlexNet [2012]
  - More of everything
  - ReLu, Dropout, Invariances
- VGG [2014]
  - Repeated blocks: even more of everything (narrower and deeper)
- NiN [2013]
  - 1x1 convolutions + global pooling instead of dense
- GoogLeNet [2015]
  - Inception block: explores parallel paths with different kernels
- Residual network (ResNet) [2016]
  - Residual block: computes residual mapping $f(\mathbf{x}) - \mathbf{x}$ in forward path

Sequence Modeling

Data

- So far . . .
- Collect observation pairs $(x_i, y_i) \sim p(x, y)$ for training
- Estimate for $y|x \sim p(y|x)$ unseen $x' \sim p(x)$
- Examples:
  - Images & objects
  - Regression problem
  - House & house prices
- *The order of the data did not matter*

Dependence on time

- Natural Language Processing
- Stock price prediction
- Movie prediction
- Wireless Applications:
  - Change in wireless channel
  - Channel Coding
  - Secure waveform generation
  - User mobility/beam prediction

Data usually is not independently and identically distributed (IID)

Autoregressive Model

- Observations from previous time steps are input to a regression model
- Linear Regression: $Y = wX + b$
- Number of inputs $x_{t-1}, \ldots, x_1$ vary with t
- First Strategy:
  - Long sequence is not required
  - $x_{t-1}, \ldots, x_{t-\tau}$ are used
- Second Strategy:
  - Keep summary h_t and update it every step
  - Latent Autoregressive model, where $\hat{x}_t = P(x_t \mid h_t)$ and $h_t = g(h_{t-1}, x_{t-1})$

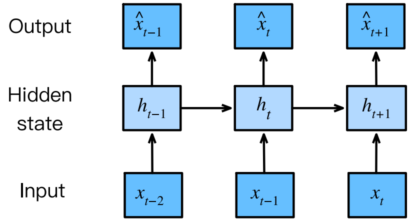


Figure: Latent Autoregressive Model

Sequence Model

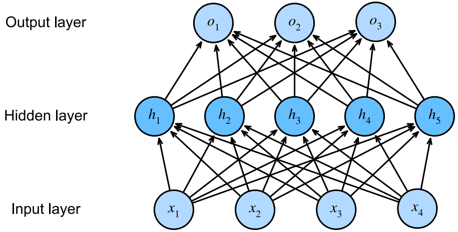
- Input x_t, where t is discrete step in time $t \in \mathbb{Z}^+$
- Dependent random variables
  $(x_1, \dots x_T) \sim p(x)$
- Conditional Probability Expansion
  $p(x) = p(x_1) \cdot p(x_2|x_1) \cdot p(x_3|x_1, x_2) \cdot \dots p(x_T|x_1, \dots x_{T-1})$
- Could also find reverse direction
  $p(x) = p(x_T) \cdot p(x_{T-1}|x_T) \cdot p(x_{T-2}|x_{T-1}, x_T) \cdot \dots p(x_1|x_2, \dots x_T)$
- Causality (physics) prevents the reverse direction (future events cannot influence the past)
- Train with sequence of data (not randomized)

Recurrent Neural Networks

- Neural Networks with hidden states
- Hidden state (h_{t-1}) capable of storing the sequence information $P(x_t \mid x_{t-1}, \ldots, x_1) \approx P(x_t \mid h_{t-1})$
- Hidden state computed as: $h_t = f(x_t, h_{t-1})$.
- Function f approximates all hidden information
- *Hidden State* is different from *Hidden Layer*

Neural Network without Hidden State

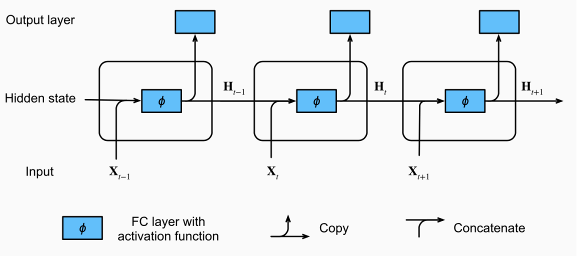
- With batch size n and width d input $\mathbf{X} \in \mathbb{R}^{n \times d}$
- Hidden layer output: $\mathbf{H} \in \mathbb{R}^{n \times h}$
  $\mathbf{H} = \phi(\mathbf{X}\mathbf{W}_{xh} + \mathbf{b}_h)$
- Activation function ϕ
- Weight and bias of h hidden units:
  $\mathbf{W}_{xh} \in \mathbb{R}^{d \times h}$, $\mathbf{b}_h \in \mathbb{R}^{1 \times h}$
- Output: $\mathbf{O} \in \mathbb{R}^{n \times q}$
  $\mathbf{O} = \mathbf{H}\mathbf{W}_{hq} + \mathbf{b}_q$,
- Weight and bias of q output units:
  $\mathbf{W}_{hq} \in \mathbb{R}^{h \times q}$, $\mathbf{b}_q \in \mathbb{R}^{1 \times q}$



- This is similar to the autoregression problem

Recurrent Neural Network - with hidden state

- Input $\mathbf{X}_t \in \mathbb{R}^{n \times d}$
- For a minibatch of size n, each row of $\mathbf{X}_t$ corresponds to one example at time step t from the sequence
- Hidden variable of current time step depends on input of the current time step and hidden variable of the previous time step
  $$\mathbf{H}_t = \phi(\mathbf{X}_t \mathbf{W}_{xh} + \underbrace{\mathbf{H}_{t-1} \mathbf{W}_{hh}}_{\text{hidden state dependency}} + \mathbf{b}_h)$$
- Output:
  $$\mathbf{O}_t = \mathbf{H}_t \mathbf{W}_{hq} + \mathbf{b}_q.$$



RNN for Natural Language Processing

- In 2003, Bengio et al. first proposed to use a neural network for language modeling
- Tokenize text into characters

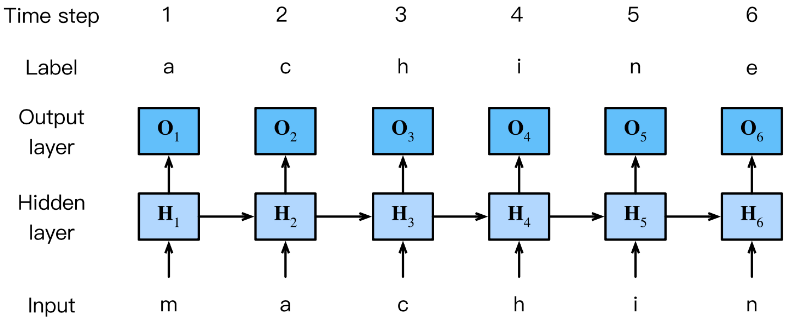


Figure: Character level language model

Loss Function for Softmax - Revisited

- The softmax function outputs a vector $\hat{\mathbf{y}}$, which can be interpreted as estimated conditional probabilities of each class given any input $\mathbf{x}$

$$\hat{y} = P(y = \text{Category } 1 \mid \mathbf{x})$$

- For the entire dataset, $P(\mathbf{Y} \mid \mathbf{X}) = \prod_{i=1}^{n} P(\mathbf{y}^{(i)} \mid \mathbf{x}^{(i)})$
- According to maximum likelihood estimation, we maximize $P(\mathbf{Y} \mid \mathbf{X})$
- This is equivalent to minimizing the negative log-likelihood

$$-\log P(\mathbf{Y} \mid \mathbf{X}) = \sum_{i=1}^{n} -\log P(\mathbf{y}^{(i)} \mid \mathbf{x}^{(i)}) = \sum_{i=1}^{n} l(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)})$$

- For any pair of label $\mathbf{y}$ and model prediction $\hat{\mathbf{y}}$ over q classes, the loss function l is

$$l(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{j=1}^{q} y_j \log \hat{y}_j \qquad \text{– } \textit{Cross Entropy Loss}$$

- $\mathbf{y}$ one-hot vector of length q, so sum over all its coordinates j vanishes for all except one term
- $\hat{y}_j$ are probabilities, so their logarithms are never greater than 0.

Loss Function for RNN

- Which token to choose in the next time step?
- Quality of the model can be measured by computing the likelihood of the sequence
- Issue: shorter sequences are much more likely to occur than the longer ones
- Solution: Cross-entropy loss *averaged* over all the n tokens of a sequence
  $\frac{1}{n}\sum_{t=1}^{n} -\log P(x_t \mid x_{t-1}, \ldots, x_1)$
- *Perplexity* in NLP: Harmonic mean of the number of real choices
  $\exp\left(-\frac{1}{n}\sum_{t=1}^{n} \log P(x_t \mid x_{t-1}, \ldots, x_1)\right)$
  - Best case: perplexity is 1.
  - Worst case: perplexity is 0.
  - Baseline: predicts a uniform distribution over all the available tokens

Gradients in RNN

- Hidden and Output Layer: $h_t = f(x_t, h_{t-1}, w_h), \quad o_t = g(h_t, w_o)$
- Chain of values that depend on recurrent computation: $\{\dots, (x_{t-1}, h_{t-1}, o_{t-1}), (x_t, h_t, o_t), \dots\}$
- Forward propagation: Compute (x_t, h_t, o_t) at each time step
- Difference between output o_t and label y_t is:
  $L(x_1, \dots, x_T, y_1, \dots, y_T, w_h, w_o) = \frac{1}{T}\sum_{t=1}^{T} l(y_t, o_t)$
- Backward propagation (by Chain Rule):

$$\begin{aligned}\frac{\partial L}{\partial w_h} &= \frac{1}{T}\sum_{t=1}^{T}\frac{\partial l(y_t, o_t)}{\partial w_h} \\ &= \frac{1}{T}\sum_{t=1}^{T}\frac{\partial l(y_t, o_t)}{\partial o_t}\frac{\partial g(h_t, w_o)}{\partial h_t}\underbrace{\frac{\partial h_t}{\partial w_h}}_{\text{Recurrent computation needed}}\end{aligned}$$

- h_t depends on h_{t-1} and w_h

Backpropagation Through Time

- Note: $h_t = f(x_t, h_{t-1}, w_h)$
- Third Term of $\frac{\partial L}{\partial w_h}$:

$$\frac{\partial h_t}{\partial w_h} = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial w_h}$$

- Can be written as three sequences:

$$a_t = b_t + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^{t} c_j \right) b_i$$

$$a_t = \frac{\partial h_t}{\partial w_h},$$

$$b_t = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h},$$

$$c_t = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial h_{t-1}}$$

Backpropagation Through Time

- BPTT:

$$\frac{\partial h_t}{\partial w_h} = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^{t} \frac{\partial f(x_j, h_{j-1}, w_h)}{\partial h_{j-1}} \right) \frac{\partial f(x_i, h_{i-1}, w_h)}{\partial w_h}.$$

- Chain rule can be used to compute $\partial h_t / \partial w_h$ recursively
- Chain gets long with t
- Solution: truncate time steps [2002]

Backpropagation Through Time

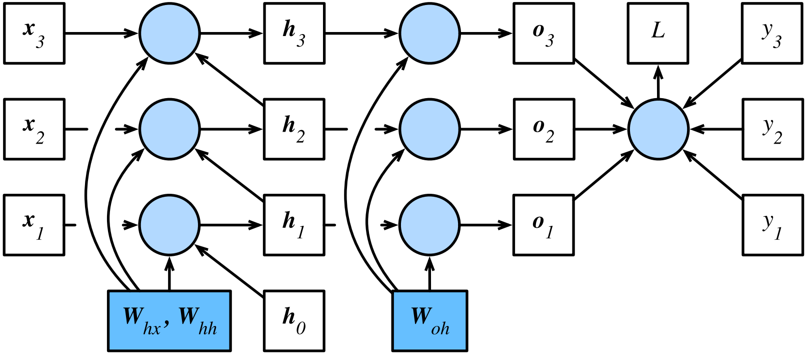
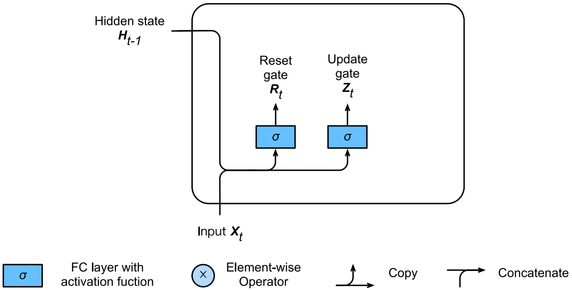


Figure: Computational graph showing dependencies for an RNN model with three time steps

Gated Recurrent Unit (GRU)

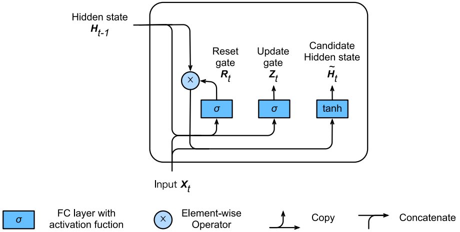
- Not all observations are equally relevant
- Engineered Gates: Reset and Update
  - Reset: mechanism to forget
  - Update: mechanism to pay attention

$$R_t = \sigma(X_t W_{xr} + H_{t-1} W_{hr} + b_r),$$
$$Z_t = \sigma(X_t W_{xz} + H_{t-1} W_{hz} + b_z)$$

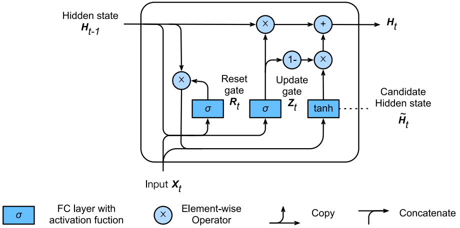


Gated Recurrent Unit (GRU)

- Candidate Hidden State
  $\tilde{\mathbf{H}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xh} + (\mathbf{R}_t \odot \mathbf{H}_{t-1})\,\mathbf{W}_{hh} + \mathbf{b}_h)$

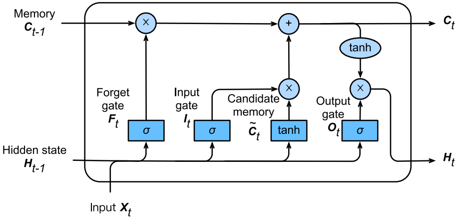


- Hidden State (incorporates update)
  $\mathbf{H}_t = \mathbf{Z}_t \odot \mathbf{H}_{t-1} + (1 - \mathbf{Z}_t) \odot \tilde{\mathbf{H}}_t$



Long Short Term Memory (LSTM)

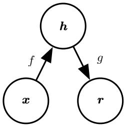
$$I_t = \sigma(X_t W_{xi} + H_{t-1} W_{hi} + b_i)$$
$$F_t = \sigma(X_t W_{xf} + H_{t-1} W_{hf} + b_f)$$
$$O_t = \sigma(X_t W_{xo} + H_{t-1} W_{ho} + b_o)$$
$$\tilde{C}_t = \tanh(X_t W_{xc} + H_{t-1} W_{hc} + b_c)$$
$$C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t$$
$$H_t = O_t \odot \tanh(C_t)$$

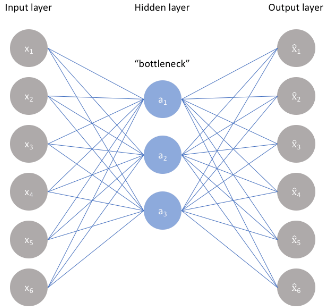


Other Networks

Autoencoder

- Autoencoder:
  - an encoder function f: converts the input data into a different representation
  - a decoder function g: converts the new representation back into the original format
- Bottleneck: Compressed knowledge representation
- Unsupervised learning
- Reconstruction Error: $L(\boldsymbol{x}, \boldsymbol{r}) = L(\boldsymbol{x}, g(f(\boldsymbol{x})))$
- Dimensionality Reduction
- Without non-linear activation functions, performs Principal Component Analysis (PCA)
- Denoising Autoencoder: Input $\hat{\boldsymbol{x}}$ is corrupted $\boldsymbol{x}$, and reconstruction loss is $L(\boldsymbol{x}, g(f(\hat{\boldsymbol{x}})))$



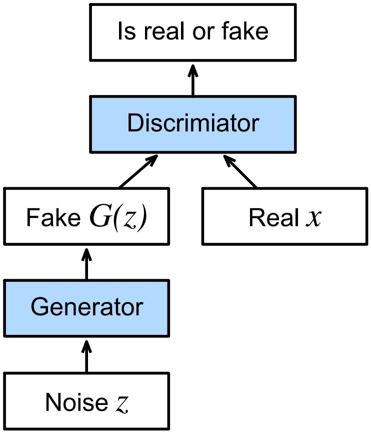


Generative Adversarial Network

- Discriminative Models: Given input $\boldsymbol{X}$, predict label $\boldsymbol{Y}$
  - Estimates $P(\boldsymbol{Y}|\boldsymbol{X})$
- Discriminative models have limitations:
  - Can't model $P(\boldsymbol{X})$
  - Can't sample from $P(\boldsymbol{X})$, i.e. can't generate new data
- Generative Model:
  - Can model $P(\boldsymbol{X})$
  - Can generate $P(\boldsymbol{X})$

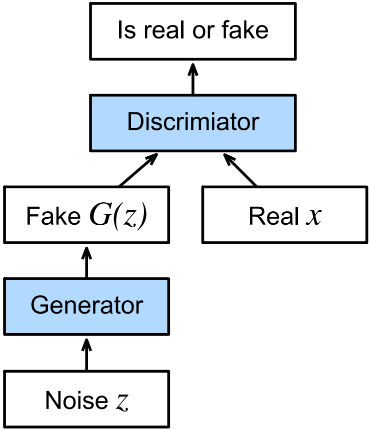
Generative Adversarial Network [Goodfellow et al., 2014]

- Leverages power of discriminative models to get good generative models
- A generator is good if we cannot tell fake data apart from real data
- *Statistical Tests*: identifies whether the two datasets come from the same distribution
- *GAN*: Based on the feedback from discriminator, creates a generator until it generates something that resembles the real data
- Generator Network: Needs to generate data (signals, images)
- Discriminator Network: Distinguishes generated and real data



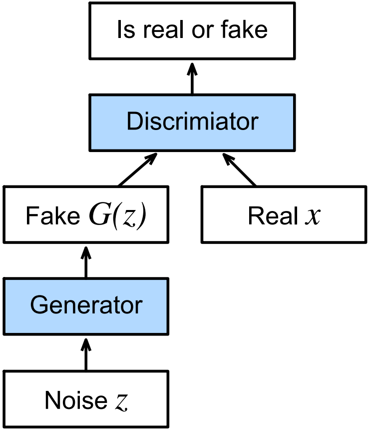
Generative Adversarial Network [Goodfellow et al., 2014]

- Training Process
  - Networks compete with each other
  - Generator attempts to fool the Discriminator
  - Discriminator adapts to the new fake/generated data
  - This information is used to improve the generator
- Discriminator:
  - A binary classifier, outputs a scalar prediction $o \in \mathbb{R}$
  - Applies *sigmoid* function to obtain predicted probability $D(\mathbf{x}) = 1/(1+e^{-o})$
  - The label y for true data is 1 and for fake data is 0
  - Train the discriminator to minimize the cross-entropy loss: $\min_D\{-y \log D(\mathbf{x}) - (1-y)\log(1-D(\mathbf{x}))\}$



Generative Adversarial Network [Goodfellow et al., 2014]

- For a given discriminator D, update the parameters of the generator G to maximize the cross-entropy loss when $y = 0$
  $\max_G\{-(1-y)\log(1-D(G(\mathbf{z})))\} = \max_G\{-\log(1-D(G(\mathbf{z})))\}$
- Conventionally, we minimize
  $\min_G\{-y\log(D(G(\mathbf{z})))\} = \min_G\{-\log(D(G(\mathbf{z})))\}$
- D and G are playing a "minimax" game with the comprehensive objective function
  $min_D max_G\{-E_{x\sim\text{Data}} log D(\mathbf{x}) - E_{z\sim\text{Noise}} log(1 - D(G(\mathbf{z})))\}$



Complex NN

- Complex Convolution: $\mathbf{W} * \mathbf{h} = (\mathbf{A} * \mathbf{x} - \mathbf{B} * \mathbf{y}) + i(\mathbf{B} * \mathbf{x} + \mathbf{A} * \mathbf{y})$
- ReLU:
  - ModReLU

$$\mathrm{modReLU}(z) = \mathrm{ReLU}(|z| + b)e^{i\theta_z} = \begin{cases} (|z| + b)\frac{z}{|z|} & \text{if } |z| + b \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

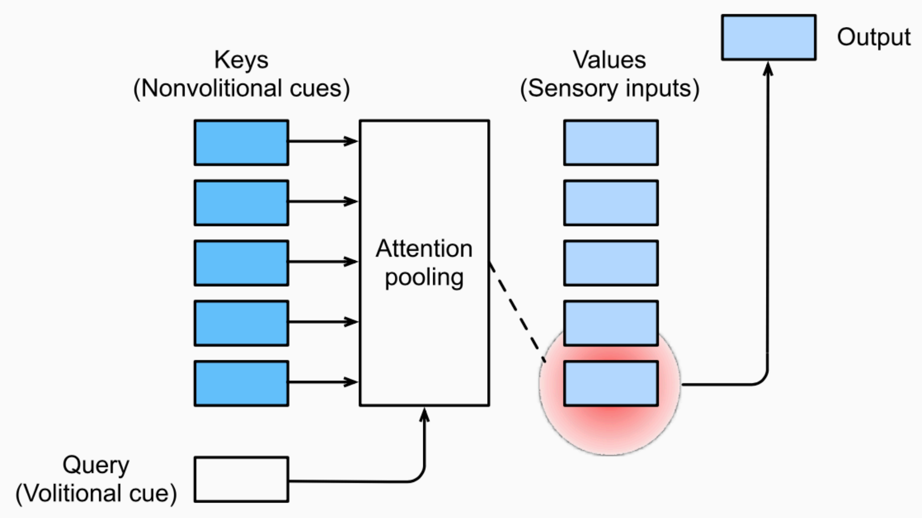
  - CReLU

$$\mathbb{C}\mathbb{R}\mathbb{L}\mathbb{U}(z) = \mathrm{ReLU}(\Re(z)) + i\,\mathrm{ReLU}(\Im(z))$$

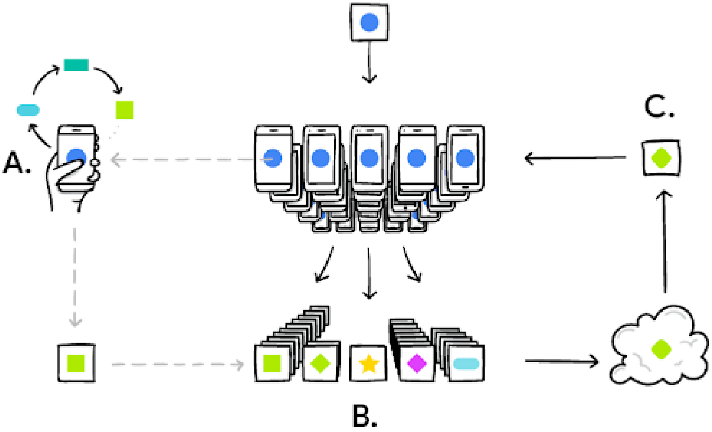
  - ZReLU

$$z\,\mathrm{ReLU}(z) = \begin{cases} z & \text{if } \theta_z \in [0, \pi/2] \\ 0 & \text{otherwise} \end{cases}$$

Attention Based NN / Transformer



Federated Learning



Wireless Applications

Wireless Applications

- Signal Detection
- Channel Encoding and Decoding
- Channel Estimation, Prediction, and Compression
- End-to-End Communications and Semantic Communications
- Distributed and Federated Learning and Communications
- Resource Allocation
- RF Fingerprinting
- Federated Learning
- Waveform Generation

https://www.comsoc.org/publications/best-readings/machine-learning-communications