# Fundamentals of Neural Networks

Xiaodong Cui

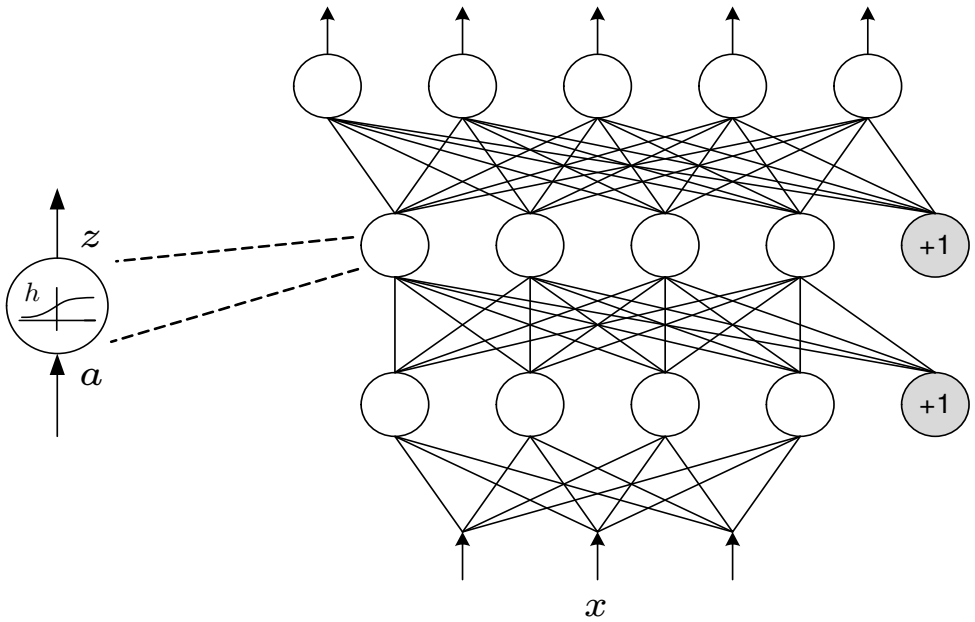
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598

Fall, 2018

# Outline

- Feedforward neural networks
- Forward propagation
- Neural networks as universal approximators
- Back propagation
- Jacobian
- Vanishing gradient problem
- Generalization

# Feedforward Neural Networks




$$a_i^{(l)} = \sum_j w_{ij}^{(l)} z_j^{(l-1)} + b_i^{(l)}$$

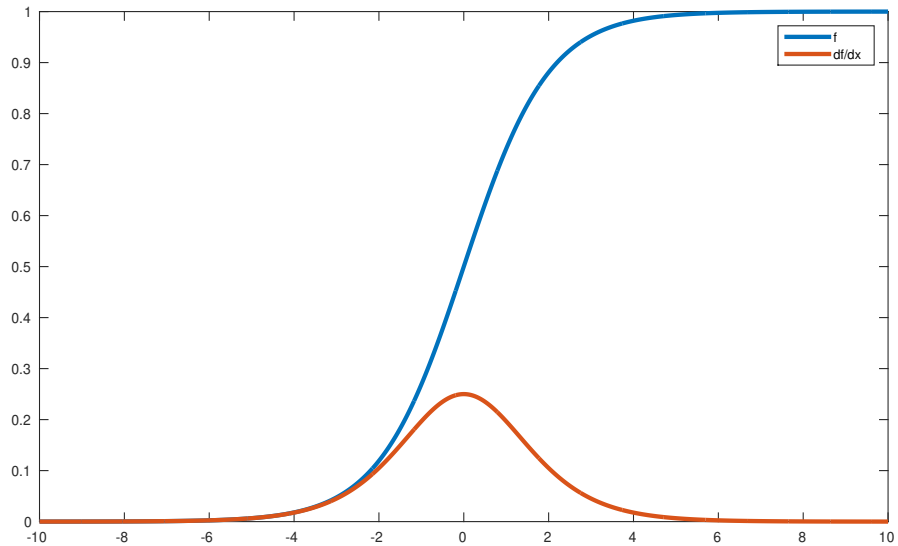
$$z_i^{(l)} = h_i^{(l)}\left(a_i^{(l)}\right)$$

# Activation Functions – Sigmoid

also known as logistic function

$$h(a) = \frac{1}{1 + e^{-a}}$$

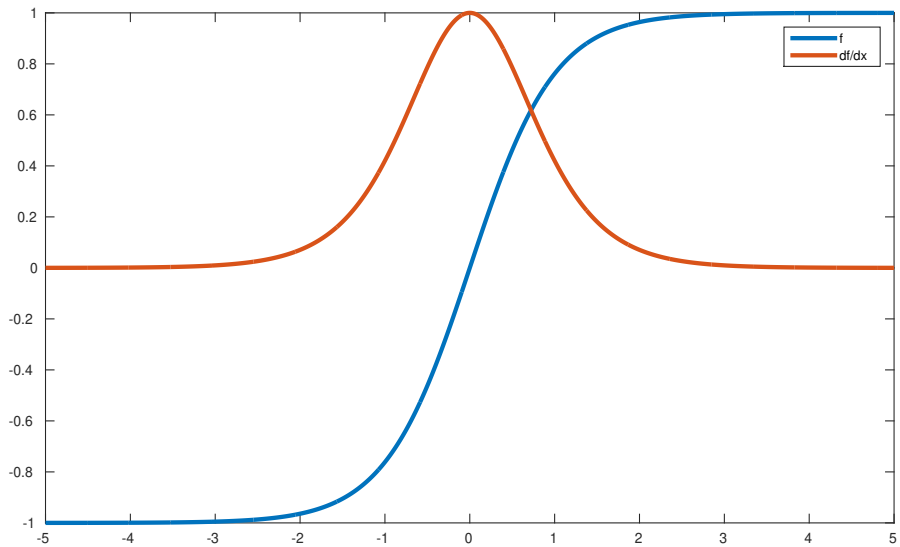
$$h'(a) = \frac{e^{-a}}{(1 + e^{-a})^2} = h(a)\big[1 - h(a)\big]$$



# Activation Functions – Hyperbolic Tangent (tanh)

$$h(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$$

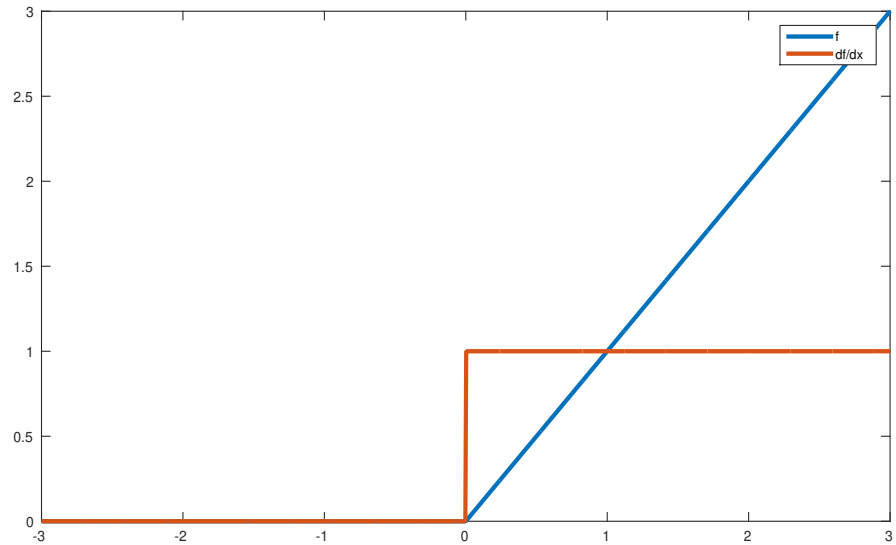
$$h'(a) = \frac{4}{(e^a + e^{-a})^2} = 1 - \left[h(a)\right]^2$$



# Activation Functions – Rectified Linear Unit (ReLU)

$$h(a) = \max(0, a) = \begin{cases} a & \text{if } a > 0, \\ 0 & \text{if } a \leq 0. \end{cases}$$

$$h'(a) = \begin{cases} 1 & \text{if } a > 0, \\ 0 & \text{if } a \leq 0 \end{cases}$$



# Activation Functions – Softmax

also known as normalized exponential function, a generalization of the logistic function to multiple classes.

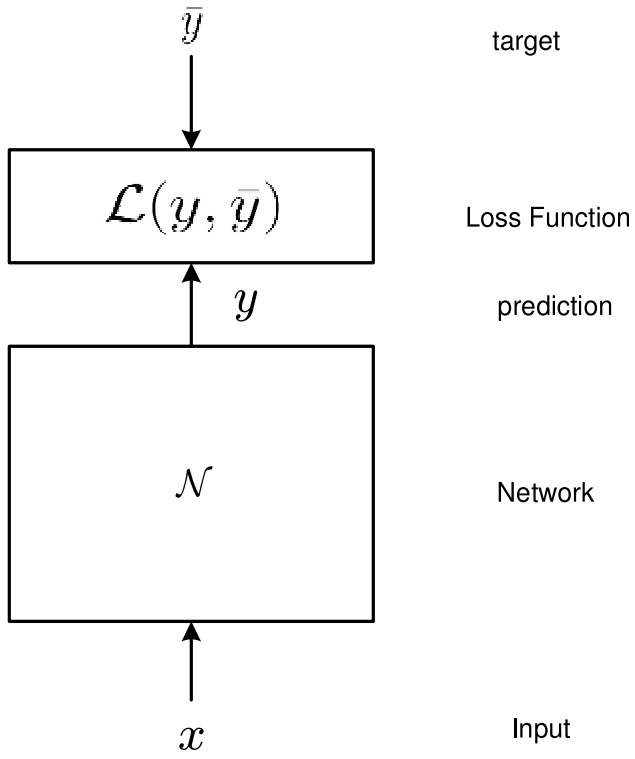
$$h(a_k) = \frac{e^{a_k}}{\sum_j e^{a_j}}$$

$$\frac{\partial h(a_k)}{\partial a_j} = \frac{\partial h(a_k)}{\partial \log h(a_k)} \frac{\partial \log h(a_k)}{\partial a_j} = h(a_k)\Big(\delta_{kj} - h(a_j)\Big)$$

Why "softmax"?

$$\max\{x_1, \cdots, x_n\} \leq \log(e^{x_1} + \cdots + e^{x_n}) \leq \max\{x_1, \cdots, x_n\} + \log n$$

# Loss Functions




- Regression
- Classification

## Regression: Least Square Error

Let $y_{n,k}$ and $\bar{y}_{n,k}$ be the output and target respectively. For regression, the typical loss function is least square error (LSE):

$$\begin{aligned}\mathcal{L}(y, \bar{y}) &= \frac{1}{2}\sum_{n}(y_n - \bar{y}_n)^2 \\ &= \frac{1}{2}\sum_{n}\sum_{k}(y_{nk} - \bar{y}_{nk})^2\end{aligned}$$

where $n$ is the sample index and $k$ is the dimension index. The derivative of LSE with respect to each dimension of each sample:

$$\frac{\partial \mathcal{L}(y, \bar{y})}{\partial y_{nk}} = (y_{nk} - \bar{y}_{nk})$$

# Classification: Cross-Entropy (CE)

Suppose there are two (discrete) probability distributions $p$ and $q$, the "distance" between the two distributions can be measured by the Kullback-Leibler divergence:

$$D_{\mathsf{KL}}(p||q) = \sum_k p_k \log \frac{p_k}{q_k}$$

For classification, the targets $\bar{y}_n$ are given

$$y_n = \{\bar{y}_{n1}, \cdots, \bar{y}_{nk}, \cdots, \bar{y}_{nK}\}$$

The predictions after the softmax layer are posterior probabilities after normalization

$$y_n = \{y_{n1}, \cdots, y_{nk}, \cdots, y_{nK}\}$$

Now measure the "distance" of these two distributions from all samples

$$\begin{aligned}\sum_n D_{\mathsf{KL}}(\bar{y}_n||y_n) &= \sum_n \sum_k \bar{y}_{nk} \log \frac{\bar{y}_{nk}}{y_{nk}} = \sum_n \sum_k \bar{y}_{nk} \log \bar{y}_{nk} - \sum_n \sum_k \bar{y}_{nk} \log y_{nk} \\ &= -\sum_n \sum_k \bar{y}_{nk} \log y_{nk} + \mathcal{C}\end{aligned}$$

# Classification: Cross-Entropy (CE)

Cross-entropy as the loss function:

$$\mathcal{L}(\bar{y}, y) = -\sum_n \sum_k \bar{y}_{nk} \log y_{nk}$$

Typically, targets are given in the form of 1-of-K coding

$$\bar{y}_n = \{0, \cdots, 1, \cdots, 0\}$$

where

$$\bar{y}_n = \begin{cases} 1 & \text{if } k = \tilde{k}_n, \\ 0 & \text{otherwise} \end{cases}$$

It follows that the cross-entropy has an even simpler form:

$$\mathcal{L}(\bar{y}, y) = -\sum_n \log y_{n\tilde{k}_n}$$

# Classification: Cross-Entropy (CE)

derivative of the composite function of cross-entropy and softmax:

$$\mathcal{L}(\bar{y}, y) = \sum_n \mathcal{E}_n = \sum_n \Big( - \sum_i \bar{y}_{ni} \log y_{ni} \Big)$$

$$y_{ni} = \frac{e^{a_{ni}}}{\sum_j e^{a_{nj}}}$$

$$\begin{aligned}
\frac{\partial \mathcal{E}_n}{\partial a_{nk}} &= \frac{\partial}{\partial a_{nk}} \Big( - \sum_i \bar{y}_{ni} \log y_{ni} \Big) = - \sum_i \bar{y}_{ni} \frac{\partial \log y_{ni}}{\partial a_{nk}} \\
&= - \sum_i \bar{y}_{ni} \frac{\partial \log y_{ni}}{\partial y_{ni}} \frac{\partial y_{ni}}{\partial a_{nk}} = - \sum_i \frac{\bar{y}_{ni}}{y_{ni}} \frac{\partial y_{ni}}{\partial \log y_{ni}} \frac{\partial \log y_{ni}}{\partial a_{nk}} \\
&= - \sum_i \frac{\bar{y}_{ni}}{y_{ni}} \cdot y_{ni} \cdot (\delta_{ik} - y_{nk}) = - \sum_i \bar{y}_{ni} (\delta_{ik} - y_{nk}) \\
&= - \sum_i \bar{y}_{ni} \delta_{ik} + \sum_i \bar{y}_{ni} y_{nk} = - \bar{y}_{nk} + y_{nk} \Big( \sum_i \bar{y}_{n,i} \Big) \\
&= y_{nk} - \bar{y}_{nk}
\end{aligned}$$

What's the derivative for an arbitrary differentiable function? – leave it for exercise

# A DNN Example In Torch

**Algorithm 1** Definition of a DNN with 3 hidden layers

```
function create_model()

– MODEL
local n_inputs = 460
local n_outputs = 3000
local n_hidden = 1024
local model = nn.Sequential()
model:add(nn.Linear(n_inputs, n_hidden))
model:add(nn.Sigmoid())
model:add(nn.Linear(n_hidden,n_hidden))
model:add(nn.Sigmoid())
model:add(nn.Linear(n_hidden,n_hidden))
model:add(nn.Sigmoid())
model:add(nn.Linear(n_hidden,n_outputs))
model:add(nn.LogSoftMax())

– LOSS FUNCTION
local criterion = nn.ClassNLLCriterion()

return model:cuda(), criterion:cuda()
end
```

# Neural Networks As Universal Approximators

**Universal Approximation Theorem**[1][2][3] Let $\phi(\cdot)$ be a nonconstant, bounded, and monotonically-increasing continuous function. Let $I_n$ represent an $n$-dimensional unit cube $[0,1]^n$ and $C(I_n)$ be the space of continuous functions on $I_n$. Then given any function $f \in C(I_n)$ and $\epsilon > 0$, there exist an integer $N$ and real constants $a_i, b_i \in \mathbb{R}$ and $w_i \in \mathbb{R}^n$, where $i = 1, 2, \cdots, N$, such that one may define

$$F(x) = \sum_{i=1}^{N} a_i \phi(w_i^{\mathsf{T}} x + b_i)$$

as an approximate realization of the function $f$ where $f$ is independent of $\phi$ such that

$$|F(x) - f(x)| < \epsilon$$

for all $x \in I_n$.

[1] Cybenko, G. (1989). "Approximation by Superpositions of a Sigmoidal Function," Math. Control Signals Systems, 2, 303-314.
[2] Hornik, K., Stinchcombe, M., and White, H. (1989). "Multilayer Feedforward Networks are Universal Approximators," Neural Networks, 2(5), 359-366.
[3] Hornik, K. (1991). "Approximation Capabilities of Multilayer Feedforward Networks," Neural Networks, 4(2), 251-257.

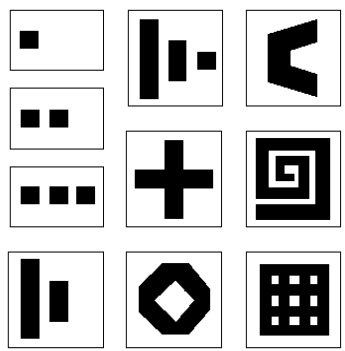
# Neural Networks As Universal Classifiers

Extends $f(x)$ to decision functions of the form [1]

$$f(x) = j, \quad \text{iff} \quad x \in P_j, \quad j = 1, 2, \cdots, K$$

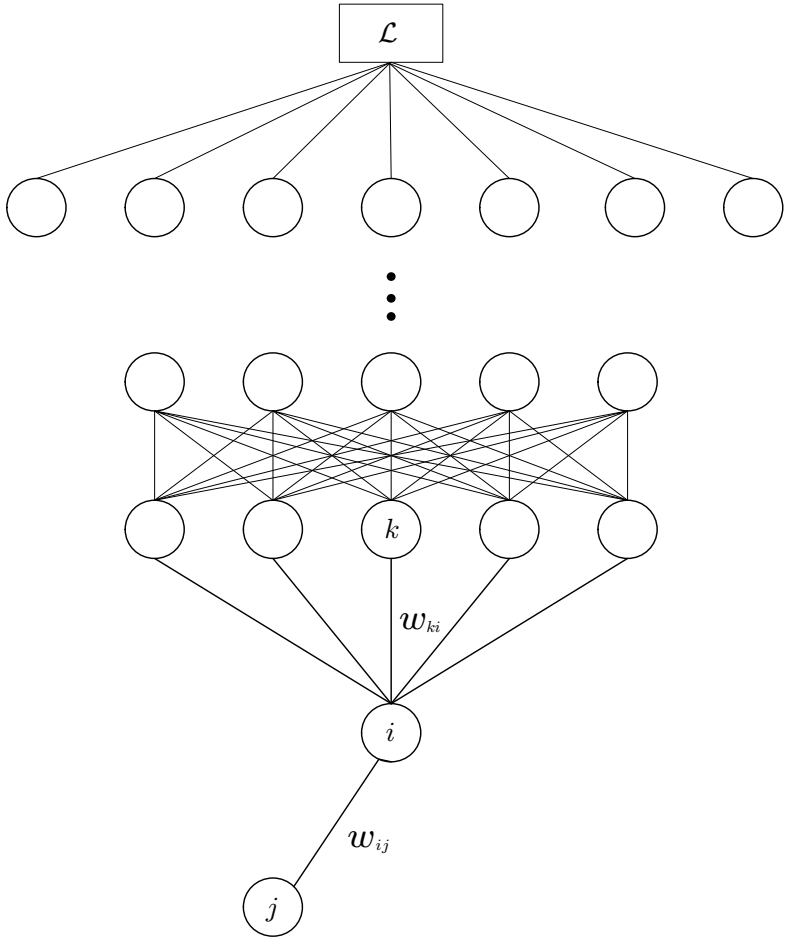
where $P_j$ partition $A_n$ into $K$ disjoint measurable subsets and $A_n$ is a compact subset of $\mathbb{R}^n$.

Arbitrary decision regions can be arbitrarily well approximated by continuous feedforward neural networks with only a single internal, hidden layer and any continuous sigmoidal nonlinearity!



[1] Cybenko, G. (1989). "Approximation by Superpositions of a Sigmoidal Function," Math. Control Signals Systems, 2, 303-314.
[2] Huang, W. Y. and Lippmann, R. P. (1988). "Neural Nets and Traditional Classifiers," in Neural Information Processing Systems (Denver 1987), D. Z. Anderson, Editor. American Institute of Physics, New York, 387-396.

# Back Propagation




How to compute the gradient of the loss function with respect to a weight $w_{ij}$?

$$\frac{\partial \mathcal{L}}{\partial w_{ij}}$$

D. E. Rumelhart, G. E. Hinton and R. J. Williams (1986). "Learning representations by back-propagating errors," Nature, vol. 323, 533-536.

# Back Propagation

$$\begin{cases} a_i^{(l)} &= \sum_j w_{ij}^{(l)} z_j^{(l-1)} \\ z_i^{(l)} &= h_i^{(l)}\left(a_i^{(l)}\right) \end{cases}$$

$$\frac{\partial \mathcal{L}}{\partial w_{ij}^{(l)}} = \frac{\partial \mathcal{L}}{\partial a_i^{(l)}} \frac{\partial a_i^{(l)}}{\partial w_{ij}^{(l)}}$$

It follows that

$$\frac{\partial a_i^{(l)}}{\partial w_{ij}^{(l)}} = z_j^{(l)} \quad \text{and} \quad \delta_i^{(l)} \triangleq \frac{\partial \mathcal{L}}{\partial a_i^{(l)}}$$

$\delta_i$ is often referred to as "errors". By the chain rule,

$$\delta_i^{(l)} = \frac{\partial \mathcal{L}}{\partial a_i^{(l)}} = \sum_k \frac{\partial \mathcal{L}}{\partial a_k^{(l+1)}} \frac{\partial a_k^{(l+1)}}{\partial a_i^{(l)}} = \sum_k \delta_k^{(l+1)} \frac{\partial a_k^{(l+1)}}{\partial a_i^{(l)}}$$

# Back Propagation

$$a_k^{(l+1)} = \sum_i w_{ki}^{(l+1)} h(a_i^{(l)})$$

therefore

$$\frac{\partial a_k^{(l+1)}}{\partial a_i^{(l)}} = w_{ki}^{(l+1)} h'(a_i^{(l)})$$

It follows that

$$\delta_i^{(l)} = h'(a_i^{(l)}) \sum_k w_{ki}^{(l+1)} \delta_k^{(l+1)}$$

which tells us that the errors can be evaluated recursively.

# Back Propagation: An Example

- A neural network with multiple layers
- Softmax output layer
- Cross-entropy loss function

- Forward propagation:
  - push the input $x_n$ through the network to get the activations to the hidden layers

$$a_i^{(l)} = \sum_j w_{ij}^{(l)} z_j^{(l-1)}, \qquad z_i^{(l)} = h_i^{(l)}\left(a_i^{(l)}\right)$$

- Back propagation:
  - start from the output layer

$$\delta_k^{(L)} = y_{nk} - \bar{y}_{nk}$$

  - back-propagate errors from the output layer all the way down to the input layer

$$\delta_i^{(l)} = h'(a_i^{(l)}) \sum_k w_{ki}^{(l+1)} \delta_k^{(l+1)}$$

  - evaluate gradients

$$\frac{\partial \mathcal{L}}{\partial w_{ij}^{(l)}} = \delta_i^{(l)} z_j^{(l)}$$

# Gradient Checking

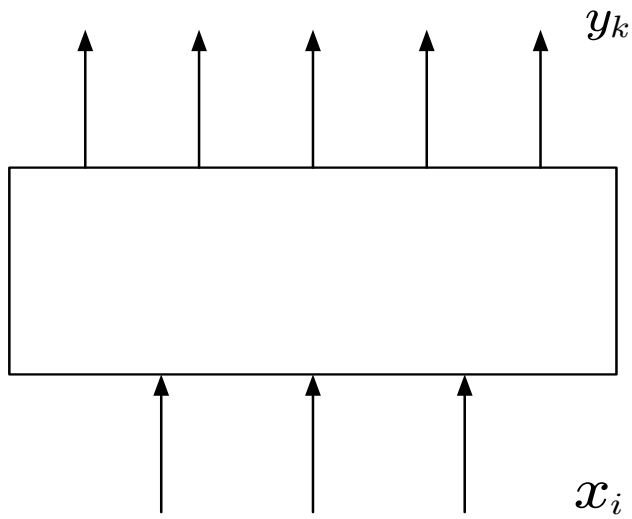
In practice, when you implement the gradient of a network or a module, one way to check the correctness of your implementation is via the following gradient checking by a (symmetric) finite difference:

$$\frac{\partial \mathcal{L}}{\partial w_{ij}} = \frac{\mathcal{L}(w_{ij} + \epsilon) - \mathcal{L}(w_{ij} - \epsilon)}{2\epsilon}$$

check the "closeness" of the two gradients (your own implementation and the one from the above difference)

$$\frac{||\boldsymbol{g}_1 - \boldsymbol{g}_2||}{||\boldsymbol{g}_1 + \boldsymbol{g}_2||}$$

# Jacobian Matrix




- measure the sensitivity of outputs with respect to inputs of a (sub-)network
- can be used as a modular operation for error backprop in a larger network

# Jacobian Matrix

The Jacobian matrix can be computed using a similar back-propagation procedure.

$$\frac{\partial y_k}{\partial x_i} = \sum_l \frac{\partial y_k}{\partial a_l} \frac{\partial a_l}{\partial x_i}$$

where $a_l$ are the activation functions having immediate connections with the inputs $x_i$.

$$\frac{\partial y_k}{\partial x_i} = \sum_l w_{li} \frac{\partial y_k}{\partial a_l}$$

Analogous to the error propagation, define

$$\delta_{kl} \triangleq \frac{\partial y_k}{\partial a_l}$$

and it can be computed recursively

$$\delta_{kl} = \frac{\partial y_k}{\partial a_l} = \sum_j \frac{\partial y_k}{\partial a_j} \frac{\partial a_j}{\partial a_l} = \sum_j \frac{\partial y_k}{\partial a_j} [w_{jl} h'(a_l)] = \sum_j \delta_{kj} [w_{jl} h'(a_l)]$$

where $j$ sweeps all the connections $w_{jl}$ with $l$.

What's the Jacobian matrix of softmax outputs to the inputs? – leave it as an exercise

# Vanishing Gradients

| function | nonlinearity | gradient |
|---|---|---|
| sigmoid | $x = \frac{1}{1+e^{-a}}$ | $\frac{\partial \mathcal{L}}{\partial a} = x(1-x)\frac{\partial \mathcal{L}}{\partial x}$ |
| tanh | $x = \frac{e^{a}-e^{-a}}{e^{a}+e^{-a}}$ | $\frac{\partial \mathcal{L}}{\partial a} = (1-x^2)\frac{\partial \mathcal{L}}{\partial x}$ |
| softsign | $x = \frac{a}{1+\lvert a\rvert}$ | $\frac{\partial \mathcal{L}}{\partial a} = (1-\lvert x\rvert)^2\frac{\partial \mathcal{L}}{\partial x}$ |

What's the problem???

$$|x| \leq 1 \quad \rightarrow \quad \frac{\partial \mathcal{L}}{\partial a} \leq \frac{\partial \mathcal{L}}{\partial x}$$

Nonlinearity causes vanishing gradients.

# Generalization – Statistical Learning Theory

- Statistical learning theory refers to the process of inferring general rules by machines from the observed samples.
- It attempts to answer the following questions in terms of learning
  - Which learning tasks can be performed by machines in general?
  - What kind of assumptions do we have to make such that machine learning can be successful?
  - What are the key properties a learning algorithm needs to satisfy in order to be successful?
  - Which performance guarantees can we give on the results of certain learning algorithms?

[1] U. v. Luxburg and B. Scholkopf, "Statistical learning theory: models, concepts, and results," arXiv:0810.4752v1, 2008.

# Formulation of Supervised Learning

Suppose $\mathcal{X}$ is the input space and $\mathcal{Y}$ is the output (label) space, the learning is to estimate a mapping function $f$ between the two spaces

$$f : \mathcal{X} \to \mathcal{Y}$$

More specifically, we choose $f$ from a function space $\mathcal{F}$. In order to estimate $f$, we have access to a set of training samples

$$(x_1, y_1), \cdots, (x_i, y_i), \cdots, (x_n, y_n) \ \in \mathcal{X} \times \mathcal{Y}$$

which are **independently** drawn from the underlying joint distribution $p(x, y)$ on $\mathcal{X} \times \mathcal{Y}$.
A loss function $\ell$ is defined on $\mathcal{X} \times \mathcal{Y}$ to measure the goodness of $f$

$$\ell(x, y, f(x))$$

On top of that, a risk function $R(f)$ is defined to measure the average loss over the underlying data distribution $p(x, y)$:

$$R(f) = E_p[\ell(x, y, f(x))] = \int p(x, y)\ell(x, y, f(x))dx$$

and we pick the minimizer of the risk

$$f_{\mathcal{F}} = \underset{f \in \mathcal{F}}{\operatorname{argmin}} R(f).$$

# Formulation of Supervised Learning

If the function space $\mathcal{F}$ includes all the functions, then the Bayesian risk is the minimal risk we can ever achieve. Assuming we know the underlying distribution $p(x, y)$, we can compute its conditional distribution $p(y|x)$ from which we can then compute the Bayes classifier

$$f_{\text{Bayes}}(x) = \operatorname*{argmax}_{y \in \mathcal{Y}} p(y|x).$$

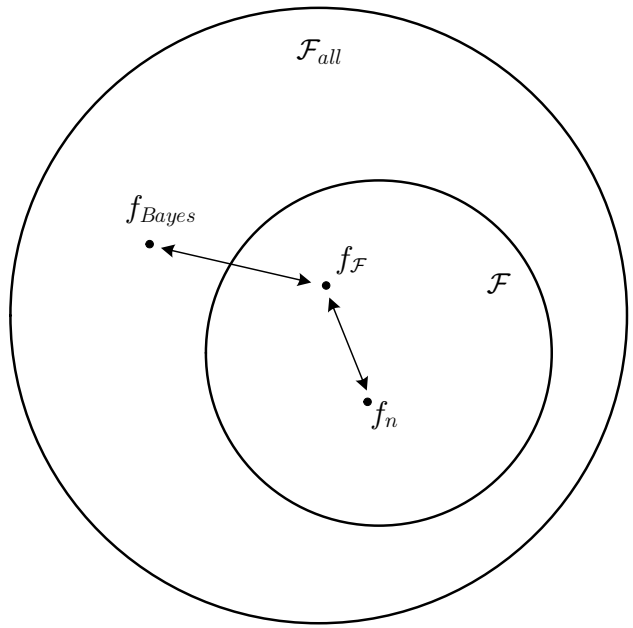
In the formulation of the supervised learning, we make no assumptions on the underlying distribution $p(x, y)$. It can be any distribution on $\mathcal{X} \times \mathcal{Y}$. However, we assume $p(x, y)$ is **fixed** but **unknown** to us at the time of learning.
In practice, we deal with empirical risk

$$R_{\text{emp}}(f) = \frac{1}{n} \sum_{i=1}^{n} \ell(x_i, y_i, f(x_i))$$

and pick its minimizer

$$f_n = \operatorname*{argmin}_{f \in \mathcal{F}} R_{\text{emp}}(f).$$

# The Bias-Variance Trade-off




$$R(f_n) - R(f_{\text{Bayes}}) = \Big[R(f_n) - R(f_{\mathcal{F}})\Big] + \Big[R(f_{\mathcal{F}}) - R(f_{\text{Bayes}})\Big]$$

# Generalization and Consistency

Let $(x_i, y_i)$ be an infinite sequence of training samples which have been drawn independently from some underlying distribution $p(x, y)$. Let $\ell$ be a loss function. For each $n$, let $f_n$ be a classifier constructed by some learning algorithm on the basis of the first $n$ training samples.

1. the learning algorithm is called consistent with respect to $\mathcal{F}$ and $p$ if the risk $R(f_n)$ converges in probability to the risk $R(f_{\mathcal{F}})$ of the best classifier in $\mathcal{F}$, that is for all $\epsilon > 0$

$$P(R(f_n) - R(f_{\mathcal{F}}) > \epsilon) \to 0 \quad \text{as} \quad n \to \infty$$

2. the learning algorithm is called Bayes-consistent with respect to $\mathcal{F}$ and $p$ if the risk $R(f_n)$ converges in probability to the risk $R(f_{\text{Bayes}})$ of the Bayes classifier, that is for all $\epsilon > 0$

$$P(R(f_n) - R(f_{\text{Bayes}}) > \epsilon) \to 0 \quad \text{as} \quad n \to \infty$$

3. the learning algorithm is called universally consistent with respect to $\mathcal{F}$ if it is consistent with respect to $\mathcal{F}$ for all distributions $p$.
4. the learning algorithm is called universally Bayes-consistent if it is Bayes-consistent for all distributions $p$.

# Empirical Risk Minimization

$$f_n = \operatorname*{argmin}_{f \in \mathcal{F}} R_{\text{emp}}(f).$$

where

$$R_{\text{emp}}(f) = \frac{1}{n} \sum_{i=1}^{n} \ell(x_i, y_i, f(x_i))$$

As $n$ changes, we are actually dealing with a sequence of classifiers $\{f_n\}$. We hope that $R(f_n)$ is consistent with respect to $R(f_{\mathcal{F}})$:

$$R(f_n) \to R(f_{\mathcal{F}}), \quad \text{when } n \to \infty$$

where $R(f_{\mathcal{F}})$ is the best risk we can achieve given $\mathcal{F}$.

# An Overfitting Example

Suppose the data space is $\mathcal{X} = [0, 1]$, the underlying distribution on $\mathcal{X}$ is uniform, the label $y$ for input $x$ is defined as follows [1]:

$$y = \begin{cases} -1 & \text{if } x < 0.5 \\ 1 & \text{if } x \geq 0.5 \end{cases}$$

Obviously, we have $R(f_{\text{Bayes}}) = 0$.
Suppose we observe a set of samples $(x_i, y_i)$, $i = 1, \cdots, n$, and construct the classifier below

$$f_n(x) = \begin{cases} y_i & \text{if } \; x = x_i, \quad i = 1, \cdots, n \\ 1 & \text{otherwise} \end{cases}$$

The constructed classifier $f_n$ perfectly classifies all training samples, which minimizes the empirical risk and drives it to 0. Suppose we draw test samples from the underlying distribution and assume they are not identical to the training samples then the $f_n$ constructed above simply predict every sample with label 1 which is wrong on half of the test samples.

$$\frac{1}{2} = R(f_n) \nrightarrow R(f_{\text{Bayes}}) = 0 \quad \text{as } n \to \infty$$

The classifier $f_n$ is not consistent. Obviously the classifier does not learn anything from the training samples other than memorize them.

[1] U. v. Luxburg and B. Scholkopf, "Statistical learning theory: models, concepts, and results," arXiv:0810.4752v1, 2008.

# Uniform Convergence of Empirical Risk

For learning consistency with respect to $\mathcal{F}$:

$$|R(f_n) - R(f_{\mathcal{F}})| \leq |R(f_n) - R_{emp}(f_n)| + |R_{emp}(f_{\mathcal{F}}) - R(f_{\mathcal{F}})|$$

The Chernoff-Hoeffding inequality:

$$P\left(\left|\frac{1}{n}\sum_{i=1}^{n}\xi_i - E(\xi)\right| \geq \epsilon\right) \leq 2\exp(-2n\epsilon^2)$$

It follows that

$$P(|R_{\text{emp}}(f) - R(f)| \geq \epsilon) \leq 2\exp(-2n\epsilon^2)$$

which shows that for any **fixed** function and sufficiently large number of samples, it is highly probable that the training error provides a good estimate of the test error.

Theorem (**Vapnik & Chervonenkis**)

*Uniform convergence*

$$P\Big(\sup_{f\in\mathcal{F}} |R(f) - R_{emp}(f)| > \epsilon\Big) \to 0 \quad \text{as } n \to \infty$$

*for all $\epsilon > 0$ is a necessary and sufficient condition for consistency of empirical risk minimization with respect to $\mathcal{F}$.*

# Capacity of Function Spaces

What kind of property should a function space $\mathcal{F}$ have to ensure such uniform convergence?

- Larger $\mathcal{F}$ gives rise to larger $P\Big(\sup_{f\in\mathcal{F}}|R(f)-R_{\text{emp}}(f)|>\epsilon\Big)$, which makes it harder to ensure a uniform convergence.
- This leads to the concept of the capacity of the function space $\mathcal{F}$.

**Uniform Convergence Bounds**:

$$P\Big(\sup_{f\in\mathcal{F}}|R(f)-R_{\text{emp}}(f)|)>\epsilon\Big)\leq 2\mathfrak{N}(\mathcal{F},2n)\exp(-n\epsilon^2)$$

The quantity $\mathfrak{N}(\mathcal{F},n)$ is referred to as the *shattering coefficient of the function class $\mathcal{F}$ with respect to sample size $n$*, which is also known as the *growth function*.

- It measures the number of ways that the function space can separate the patterns into two classes.
- It measures the "size" of a function space by counting the effective number of functions given a sample size $n$.

Theorem (**Vapnik & Chervonenkis**)

$$\frac{1}{n}\log\mathfrak{N}(\mathcal{F},n)\to 0$$

*is a necessary and sufficient condition for consistency of empirical risk minimization on $\mathcal{F}$.*

# Generalization Bounds

Given $\delta > 0$, with a probability of $1 - \delta$, any function $f \in \mathcal{F}$ satisfies

$$R(f) \leq R_{\text{emp}}(f) + \sqrt{\frac{1}{n}\Big(\log\left(2\mathfrak{N}(\mathcal{F}, n)\right) - \log\delta\Big)}$$

Or

$$R(f) \leq R_{\text{emp}}(f) + \sqrt{\frac{C + \log\frac{1}{\delta}}{n}}$$

happens with probability at least $1 - \delta$, where $C$ is a constant representing the complexity of the function space $\mathcal{F}$.
Generalization bounds are typically written in the following form

$$R(f) \leq R_{\text{emp}}(f) + \text{capacity}(\mathcal{F}) + \text{confidence}(\delta)$$

for any $f \in \mathcal{F}$.

Other common capacity measures

- VC dimension
- Rademacher complexity