MULTILAYER PERCEPTRON

Training the Neural Network using Supervised Learning

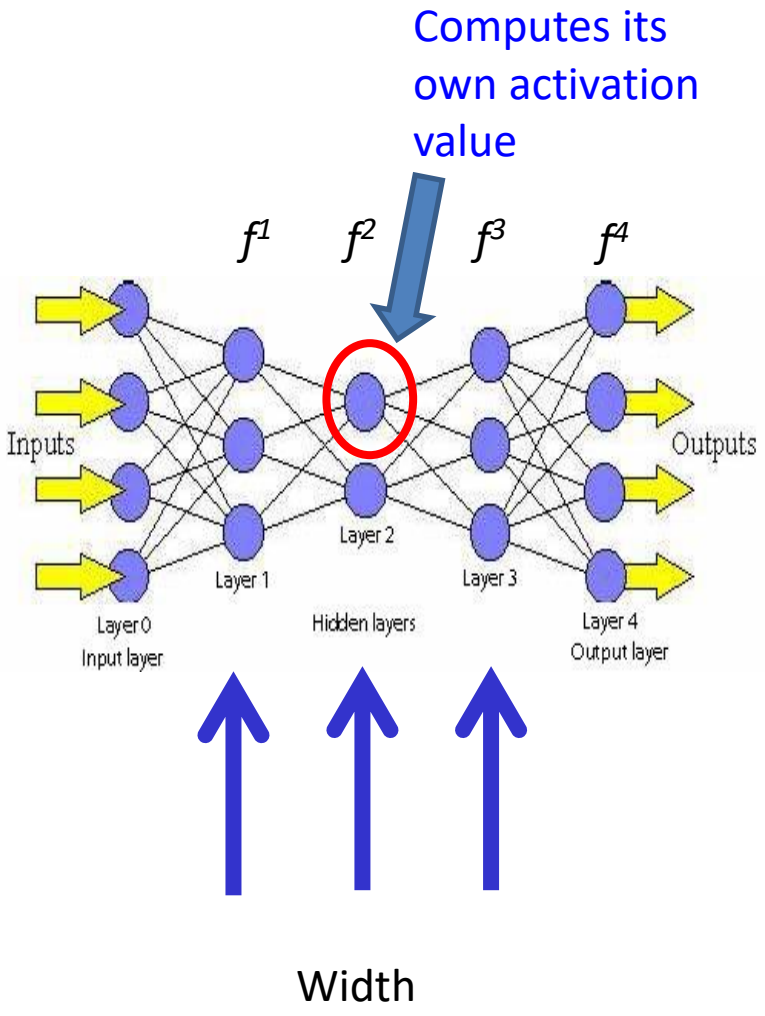
Forward and Backward Propagation

Regularization

CS8004: Lecture 2-3

Feed Forward NNs

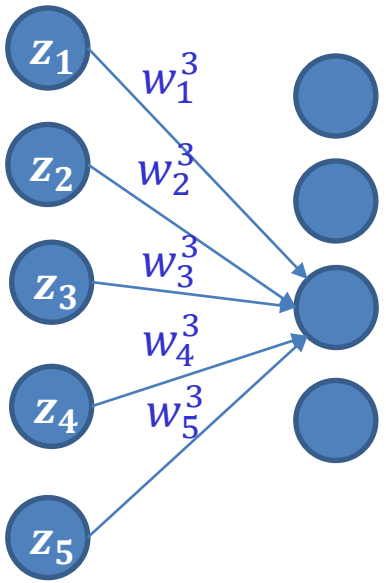
- **Feed forward neural networks** are the basic building blocks of NNs and deep learning.
- The model is represented with a directed acyclic graph describing how the functions are composed together.
- The function
  $f(x) = f^4(f^3(f^2(f^1(x))))$
  tends to approximate f^*.
- For each input x, the NN should approximate the corresponding y value.
- NN is trained to minimize the error of approximation.



Feed Forward NNs

- Each neuron of each layer receives a weighted sum of inputs from neurons of previous layers.
  - Suppose you have 5 neurons (units) in 1st hidden layer and 4 units in the 2nd layer.
  - 3rd unit of 2nd layer will receive weighted inputs from units of first layer as follows.

$$w_1^3 z_1 + w_2^3 z_2 + \cdots + w_5^3 z_5$$



Feed Forward NNs

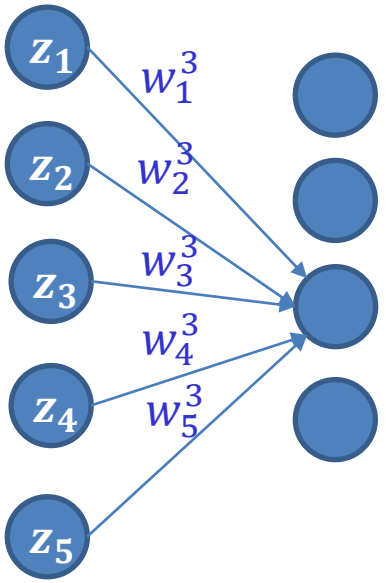
- Then the 3rd unit uses its own activation function to send the amount of input required to the next layer units.
  - $a = f(w_1^3 z_1 + w_2^3 z_2 + \cdots + w_5^3 z_5)$
  - This a becomes the input to the next layer from the 3rd unit of the second layer.
- To makes thing clear, we shall use following notations.
- Layer l : Input vector is a^{l-1} and output vector is a^l.

- So if there are
  - h_{l-1} number of hidden units in layer $l-1$ and
  - h_l number of hidden units in layer l
- What will be the dimension of a^{l-1} and a^l ? And what will be values of their components ?

Feed Forward NNs …

- Take an example: 1st hidden layer sending its input to the 2nd layer.
- Input to the 2nd layer: a^1
- Output from 2nd layer a^2
- 5 units in 1st layer will send 5 outputs.

$$\begin{bmatrix} a_1^1 \\ a_2^1 \\ a_3^1 \\ a_4^1 \\ a_5^1 \end{bmatrix} = \begin{bmatrix} f^1(w_1^1 a_1^0 + w_2^1 a_2^0 + w_3^1 a_3^0 + b_1^1) \\ f^1(w_1^2 a_1^0 + w_2^2 a_2^0 + w_3^2 a_3^0 + b_2^1) \\ f^1(w_1^3 a_1^0 + w_2^3 a_2^0 + w_3^3 a_3^0 + b_3^1) \\ f^1(w_1^4 a_1^0 + w_2^4 a_2^0 + w_3^4 a_3^0 + b_5^1) \\ f^1(w_1^5 a_1^0 + w_2^5 a_2^0 + w_3^5 a_3^0 + b_5^1) \end{bmatrix}$$



General Form

- Binary Classification: To classify from a set of objects to class 0 or 1.
- Problem Statement: Find out the probability of an object x belonging to one of the classes.
- A set of training examples $[x^1 \quad x^2 \quad \dots \quad x^m]$ is given together with the corresponding class labels $[y^1 \quad y^2 \quad \dots \quad y^m]$, where each y value is either 0 or 1.
- We find out the probability $\hat{y}^i$ for each y^i.
- Train the system to find out optimum values of parameters so as to reduce the training error (in other words minimize the loss or cost function).
- Once this is done, the NN is ready to do classification.

General Form…

- Let us now formulate the Feed forward neural network in the general form.
- Given the design matrix
- $[x^1 \quad x^2 \quad \dots \quad x^m]$
- $X=\begin{bmatrix} {x_1}^1 & {x_1}^2 & \dots & {x_1}^m \\ {x_2}^1 & {x_2}^2 & \dots & {x_2}^m \\ \vdots & \vdots & \dots & \vdots \\ {x_n}^1 & {x_n}^2 & \dots & {x_n}^m \end{bmatrix}$
- To compute $W^T X + b$ for the first layer.
- Let the first weight matrix be denoted by $W^{[1]}$.
- It is to randomly chosen and its dimension will be
  - Number of features in each sample × Width of the first hidden layer.
- The first bias vector $b^{[1]}$ will be a single column vector of dimension = Width of the first hidden layer

General Form…

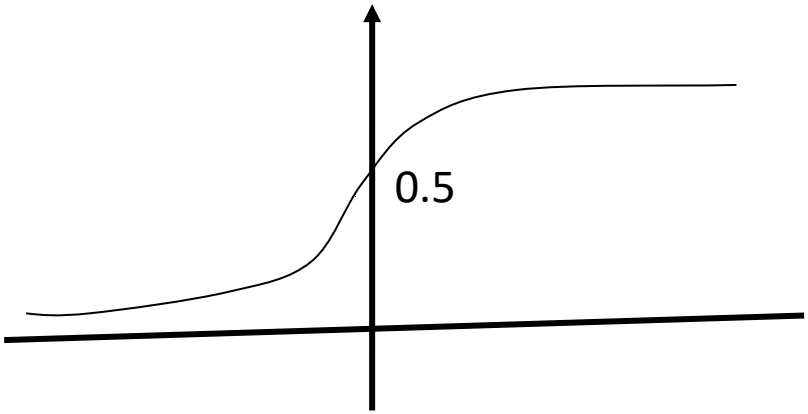
- Suppose there are l hidden layers and each layer uses an identity activation function, that is $I(x) = x$.
- We can write the general expression as follows -

- $z^{[1]} = W^{[1]^T} X + b^{[1]} = g^{(1)}(X) = I(X)$;
- $z^{[2]} = W^{[2]^T} z^{[1]} + b^{[2]} = g^{(2)}\left(z^{[1]}\right) = I(z^{[1]})$;
- … … …
- $z^{[l]} = W^{[l]^T} z^{[l-1]} + b^{[l]} = g^{(l)}\left(z^{[l-1]}\right) = I(z^{[l-1]})$;

- Since the problem is about binary classification, we use the sigmoid function to find out the probability of a given sample in a class.
- So $g^{(l+1)}(z^{[l]}) = \sigma(z^{[l]}) = \frac{1}{1+\exp(-z^{[l]})}$.

Logistic Regression

- Given $\boldsymbol{x} \in \boldsymbol{R^n}$ with a label $y \in \boldsymbol{R}$
- Find $\hat{y} = P(y = 1 \,|\boldsymbol{x})$ (Conditional probability)
- $\hat{y}$ is modelled as $\hat{y} = \sigma(w^T x + b) = \sigma(z)$.

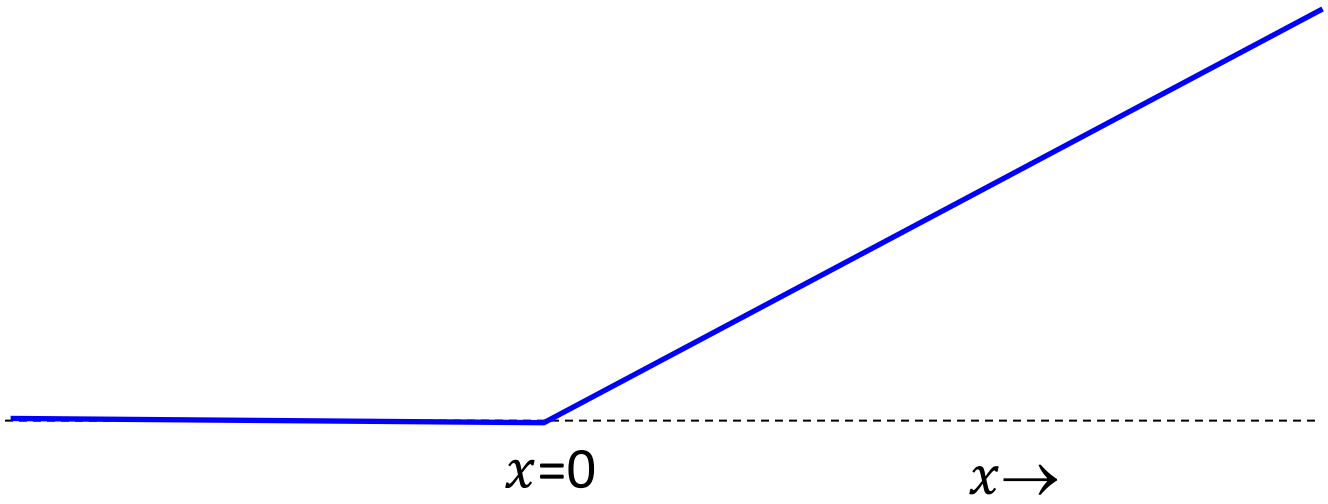


$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Some Other Functions

- Rectified Linear Unit

$$g(x) = \max\{ 0 , x\}$$



Some Other Functions…

- Hyperbolic tangent

$$g(x) = \tan h\,(x)$$
$$= 2\,\sigma(2x) - 1$$

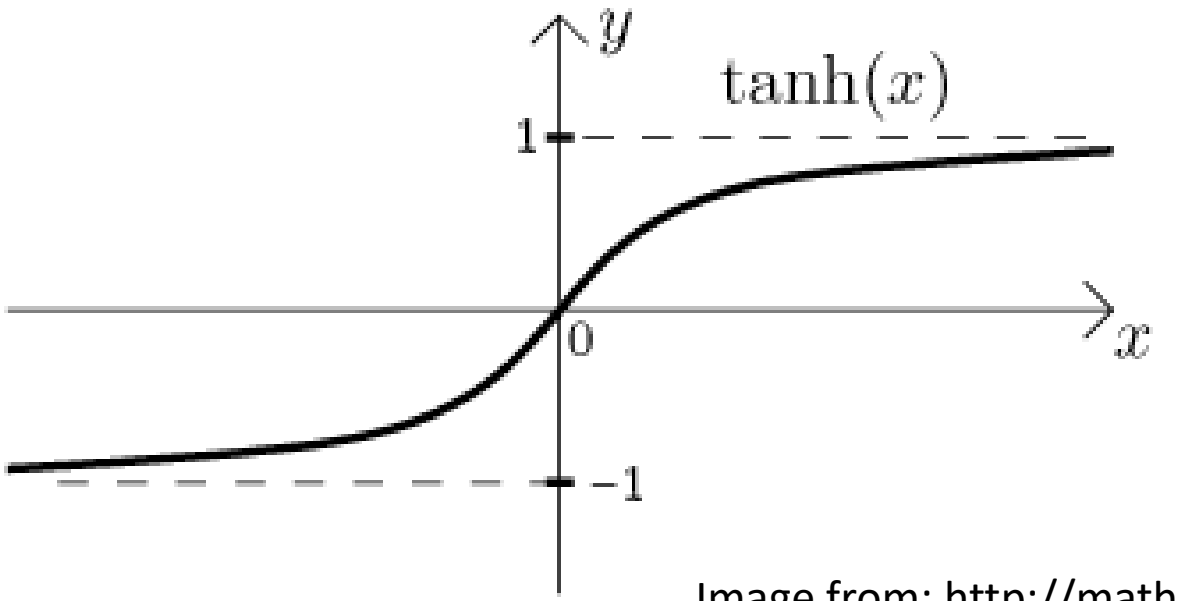


Image from: http://math.feld.cvut.cz/mt/

Gradient Based Solution

- NNs are usually trained using iterative gradient based optimization process.
- This drives the solution to result in a cost function with a very low value.
- To apply gradient based learning, choosing an appropriate cost function is important.

Gradient Based Solution…

- The parametric models mostly define a distribution $p(y|(x;\ \boldsymbol{\theta}))$
- Principle of maximum likelihood* is used to define the cost function.
- Some time we determine only a statistic of y dependent on input $\boldsymbol{x}$, not the entire probability distribution.

* Given at the end of the slide as supplementary reading material

Gradient Based Solution…

Logistic Regression : $\hat{y} = \sigma(w^T x + b),\ \sigma(z) = \frac{1}{1+e^{-z}}$

- Cost Function (coming from maximum likelihood function)

- Suppose the loss of approximation or simply the error of approximation is defined as

$$L(\hat{y}, y) = -(\, y \log \hat{y} + (1 - y) \log(1 - \hat{y}))$$

- If $y = 1$, then $L(\hat{y}, y) = -\log \hat{y}$. This will be approach a minimum when when it will be as close as possible to zero.
  - Since $\hat{y}$ can take maximum value 1, so by making $\hat{y}$ as large as possible, we can reduce the loss function.
- If $y = 0$, then we need to minimize $-\log(1 - \hat{y})$, bringing it as close as possible to zero.
  - If $\hat{y}$ is close to zero, the loss function will be minimized.

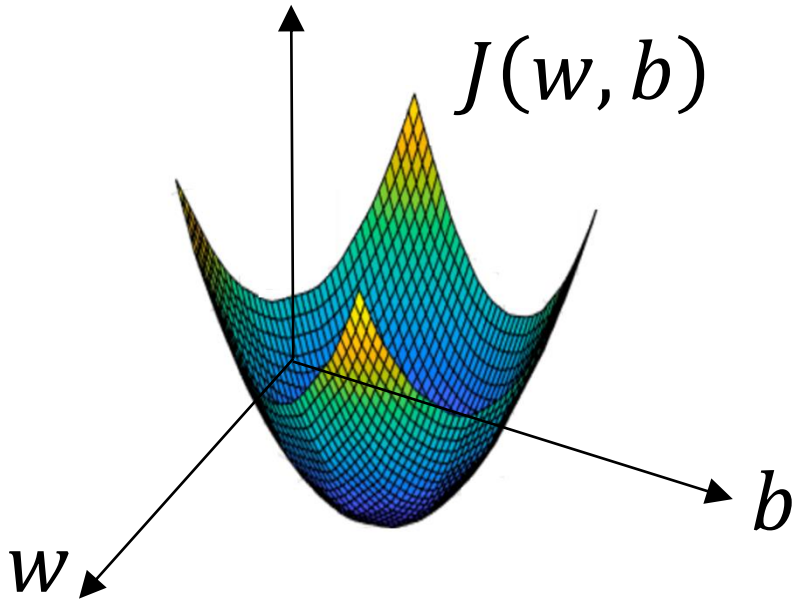
Logistic Regression Cost Function

- We define the cost function as follows -

- $J(w, b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^i, y^i)$
- $= -\frac{1}{m}\sum_{i=1}^{m}(y^i \log \hat{y}^i + \left(1 - y^i\right) \log(1 - \hat{y}^i))$

Gradient Descent

- Find w, b such that $J(w, b)$ is minimized.
- Assuming $J(w, b)$ is a convex function, that means it has just one minimum.
- So we proceed in the direction of steepest descent to arrive at the global minimum.



Method of Steepest Descent

- Suppose $\nabla_u f(x)$ denotes the directional derivative of f in the direction u at a point x.
- We know it can be expressed as
  - $\nabla_u f(x) = u^T \nabla f(x)$, here ∇f represents the gradient of f.
- To minimize ∇f we find out the direction u in which f decreases fast, that is the direction whose directional derivative is minimum.
  - Find u such that $u^T \nabla f(x)$ is minimum at x.

Method of Steepest Descent

- Now

$$\min_{u,\|u\|=1} u^T \nabla f(x) = \min_{u,\|u\|=1} \|u\| \, \|\nabla f(x)\| \cos\theta$$

$$= \min_{u,\|u\|=1} \|\nabla f(x)\| \cos\theta$$

- This will be minimum when $\cos\theta$ will be minimum , that is when $\theta = \pi$, that is when the direction of u will be opposite to the gradient (∇f).
- The gradient points in the direction of maximum , and the steepest descent is just in the opposite direction.
- This method of moving toward the negative of ∇f is known as the method of steepest descent.

Method of Steepest Descent…

- Steepest descent proposes a new point to proceed to the global minimum.
  - $x' = x \ - \ \alpha \nabla f(x)$
- Here α is called the learning rate, a positive scalar that determines the step size to move toward the global minimum.
- How to choose α ?
  - Choose a small positive number randomly.
  - Or compute $f(x \ - \ \alpha \nabla f(x))$ for several values of α and choose the one that gives minimum value – called line search.

Gradient Descent

- We proceed in the direction of steepest descent to arrive at the global minimum as follows.
- Since $J(w, b)$ has two parameters , we keep updating w, b using the following equations until we get convergence.

$$w = w - \alpha \frac{\partial J}{\partial w}; \qquad \mathrm{b} = b - \alpha \frac{\partial J}{\partial b}$$

- That is, until the difference between two consecutive values of the cost function is negligible.

Method of Steepest Descent

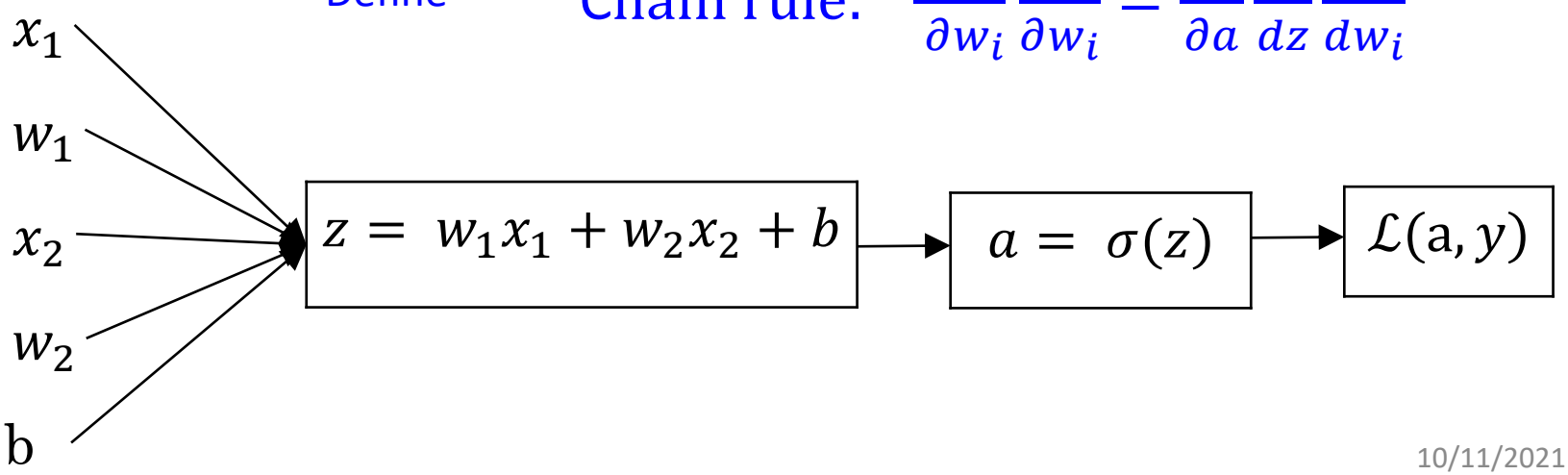
- Effect of learning rate on convergence
  - $(W, b) = (W, b) - \alpha \nabla J = (W, b) - \alpha \begin{bmatrix} \frac{\partial J}{\partial w} \\ \frac{\partial J}{\partial b} \end{bmatrix}$
  - (α is learning rate, a positive scalar that determines the step size to move toward the minimum.

Gradient Descent in Logistic Regression

- Recall that in the j-th hidden layer the activation function is given by
  - $z^{[j]} = W^{[j]^T} X + b^{[j]} = g^{(j)}(X)$;
- Suppose there is only one layer: Then this will become
  $z = W^T X + b$.
- Let $a = \sigma(z) = \hat{y}$; $L(\hat{y}, y) = L(a, y) = -(y \log a + (1-y)\log(1-a))$
- Use computation graph to implement the process of gradient descent for logistic regression.

Define

Chain rule: $\frac{\partial L}{\partial w_i}\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial a}\frac{da}{dz}\frac{dz}{dw_i}$



How to get dw1/dw2/db ?

$$Da = \frac{\partial L}{da}$$

$$L(a, y) = -(\, y \log\, a + (1 - y) \log(1 - a))$$

$$\therefore Da = -\frac{y}{a} + \frac{1-y}{1-a}$$

$$a = \sigma(z)$$

$$\mathrm{D}z = \frac{\partial L}{\partial z} = \frac{\partial L}{\partial a}\frac{da}{dz} = Da \cdot \frac{d\sigma}{dz} = Da[\sigma(z) - \sigma(z)^2]$$

$$z = W^T X + b = w_1 x_1 + w_2 x_2 + b, \textit{ say. Then}$$

$$\mathrm{D}w_i = \frac{\partial L}{\partial w_i} = Da \cdot Dz \frac{dz}{dw_i} = Da \cdot Dz \cdot x_i \quad i = 1, 2.$$

$$\mathrm{D}b = \frac{\partial L}{\partial b} = Da \cdot Dz \frac{dz}{db} = \mathrm{D}a \cdot Dz$$

Gradient Descent in Logistic Regression…

- Suppose there are m training examples, each with two features.
- Then the cost function
- $J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(a^i, y^i)\ ; a^i = \hat{y}^i = \sigma(z^i) = \sigma(w^T x^i + b)$
- For each example (x^i, y^i), we can calculate $Dw^i{}_1\ , Dw^i{}_2, Db^i$.
- Then $\frac{\partial}{\partial w^1} J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \frac{\partial}{\partial w^1} L(a^i, y^i) = \frac{1}{m}\sum_{i=1}^{m} dw^i{}_1$

Implementing Gradient Descent in Logistic Regression…

Let us write step by step now.

$$J = 0; Dw_1 = 0; Dw_2 = 0; Db = 0$$

for $i = 1\ to\ m$

$$z^i = w^T x^i + b$$

$$a^i = \sigma(z^i)$$

$$J\ += -[y^i \log a^i + (1 - y^i) \log(1 - a^i)]$$

$$Dz^i = a^i - y^i$$

$$Dw_1 += x_1{}^i Dz^i$$

$$Dw_2 += x_2{}^i Dz^i$$

$$Db += Dz^i$$

$$J/ = m$$

$$Dw_1/ = m\ ; Dw_2/ = m\ ; Db/ = m$$

Implementing Gradient Descent in Logistic Regression…

- Choose a learning rate α and compute

$$w_1 = w_1 - \alpha \frac{\partial J}{\partial w_1}$$

$$w_2 = w_2 - \alpha \frac{\partial J}{\partial w_2}$$

$$b = b - \alpha \frac{\partial J}{\partial b}$$

- This is when the samples have only two features. What if the samples have n features ?

Forward and Backward Propagation

- Layer L Deep Neural Network (DNN):
- Layer l: Forward propagation
- Input a^{l-1} and output a^{l}
  - $a^{l} = g^{l}(z^{l}) = g(w^{l}z^{l-1} + b^{l})$
- We keep z^{l}, w^{l}, b^{l} in the cache for backward propagation to compute cost function.
- Layer l: Backward propagation
- Input Da^{l} and output Da^{l-1}, Dw^{l}, Db^{l}

Implementing Neural Networks in Python

- Python is a high-level, interpreted scripting language developed in the late 1980s by Guido van Rossum at the Centrum Wiskunde & Informatica (CWI), Amsterdum, Netherlands.
- Initially published at the alt.sources newsgroup in 1991, the first version 1.0 was released in 1994.

Guido van Rossum: Photo source; Wikimedia

Source: https://realpython.com/python-introduction/

Python and Deep Learning

- Python is Portable
  - Because Python code is interpreted and not compiled into native machine instructions, code written for one platform will work on any other platform that has the Python interpreter installed.
- Python is Simple
  - Developers have deliberately kept it that way.
- Python is is widely used in data science.
- Well suited for coding deep learning algorithms using NumPy and Tensorflow.

Source: https://realpython.com/python-introduction/

TensorFlow

- TensorFlow™ is an open source software library for high performance numerical computation.
- Its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices.
- Originally developed by researchers and engineers from the Google Brain team within Google's AI organization, it comes with a strong support for machine learning and deep learning,
- Its flexible numerical computation core is used across many other scientific domains.

Source: https://www.tensorflow.org/

Keras: Python Deep Learning Library

- Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow.
- Developed for enabling fast experimentation.
  - Official website of Keras. (https://keras.io/)

- Developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System)
- François Chollet, a Google engineer is the primary author and maintainer of Keras.
- In 2017, Google's TensorFlow team decided to support Keras in TensorFlow's core library.
  - Wikipedia

Keras: Python Deep Learning Library

- "Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Runs seamlessly on CPU and GPU."
  - Official website of Keras. (https://keras.io/)

Limitations of Neural Networks

Random initialization + **densely connected networks** lead to:

- High cost
  - Each neuron in the neural network can be considered as a logistic regression.
  - Training the entire neural network means training all the interconnected logistic regressions.
- Difficult to train as the number of hidden layers increases
- Stuck in local optima
  - The objective function of the neural network is usually not convex.
  - Random initialization does not guarantee starting from the proximity of global optima.
- One solution: Regularization.

Some Important Terms

- Generalization
  - The ability to perform well on previously unobserved inputs

- Training error – error observed on the training set.
- Test error or Generalization error
  - expected value of the error on a new input or a test set.
- Underfitting
  - When the model is not able to obtain a sufficiently low error value on the training set.
- Overfitting
  - When the gap between the training error and test error is too large.

- Capacity and Hypothesis space

Regularization

- Any modification that we make to reduce the generalization error (not the training error).
- Any method that prevents overfitting or helps in optimization.
- Specifically: Additional terms in the training objective to prevent overfitting or to help in optimization.
- A deep learning algorithms strives to effectively provide solution to a wide range of tasks using very general purpose frameworks of regularization.

Overfitting: Example

$$t = \sin(2\pi x) + \epsilon$$

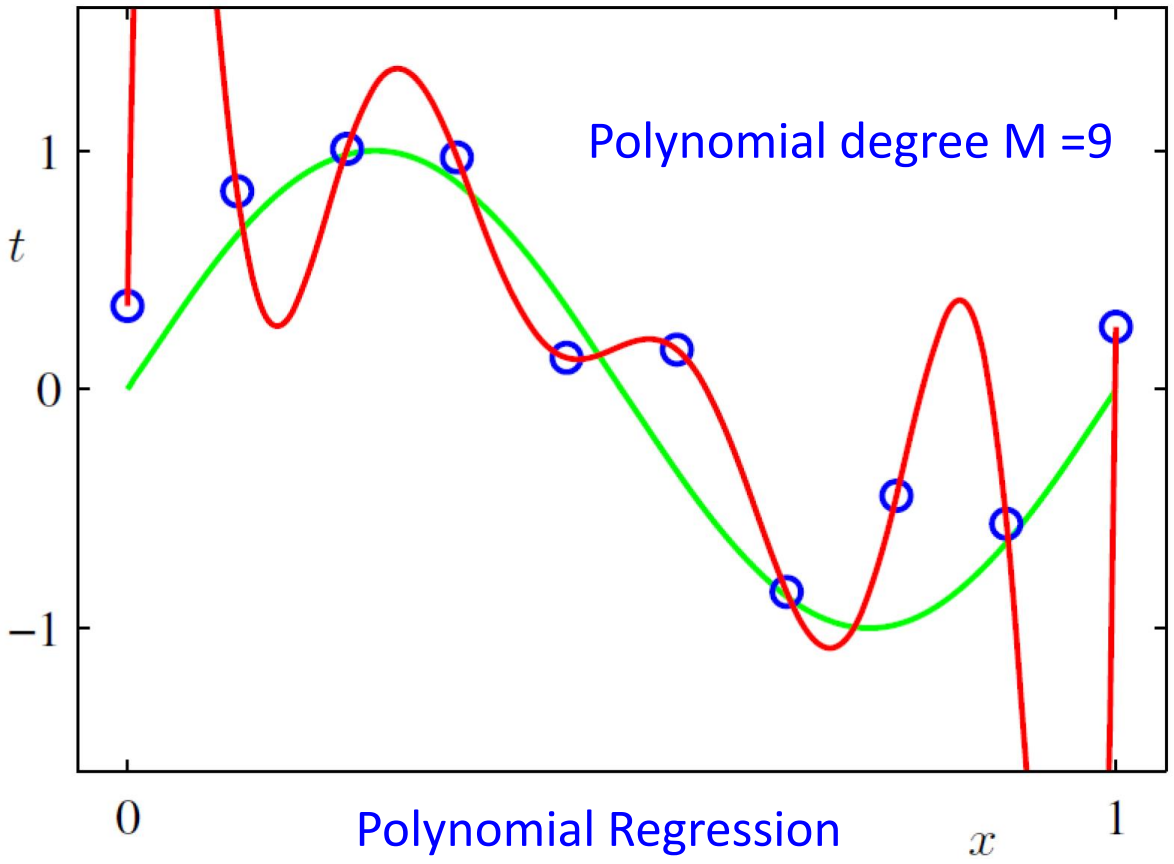


Figure from the book: Bishop, Machine Learning and Pattern Recognition

Overfitting: Example

- Regression Error

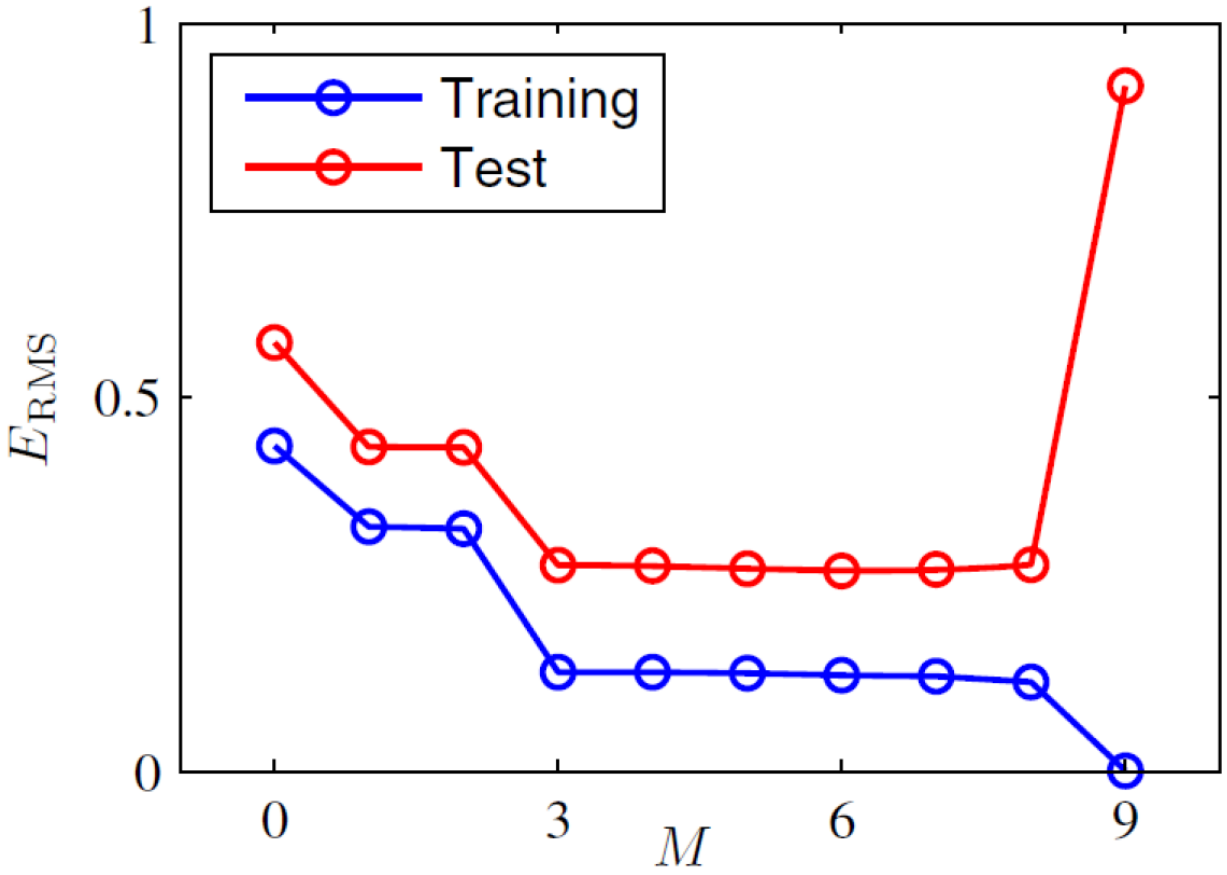
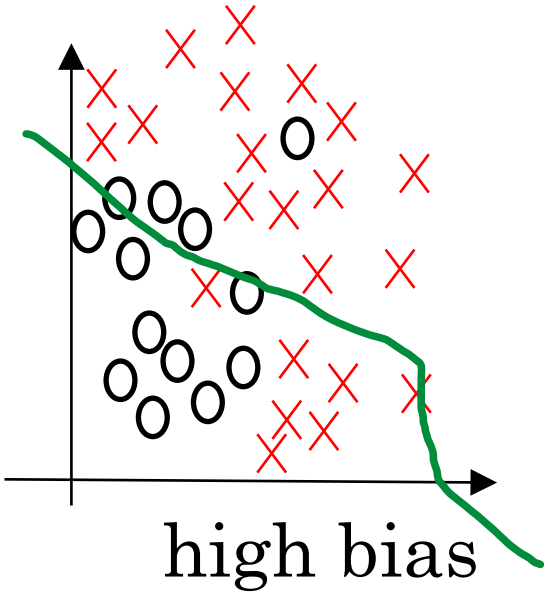


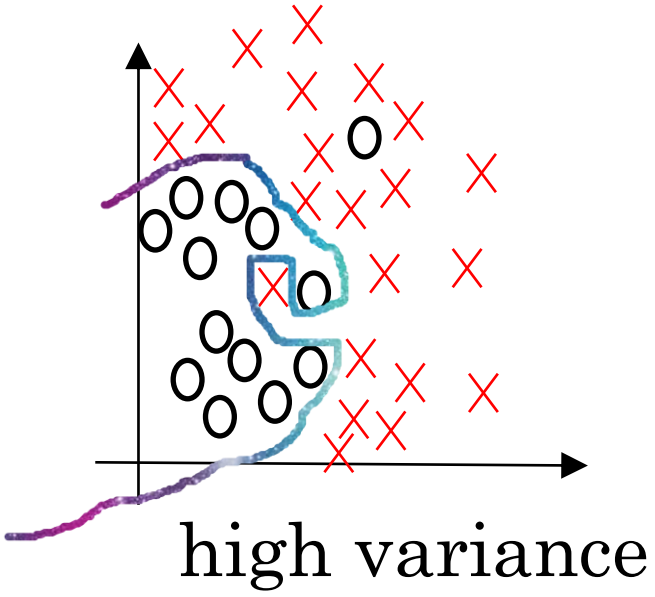
Figure from the book: Bishop, Machine Learning and Pattern Recognition

How to Prevent Overfitting ?

- Large dataset helps overcome overfitting
- Dropping hypotheses such as model capacity (high degree polynomial space) also helps reduce overfitting.
- Classical regularization methods : methods to constrain hypotheses.
- Other methods : data augmentation, early stopping etc.

Bias and Variance





How much away is the approximated value from the expected value

Variation of the values obtained by the model from the expected value.

Bias and Variance

- As capacity increases (*x*-axis), bias (dotted) tends to decrease and variance (dashed) tends to increase, yielding another U-shaped curve for generalization error.

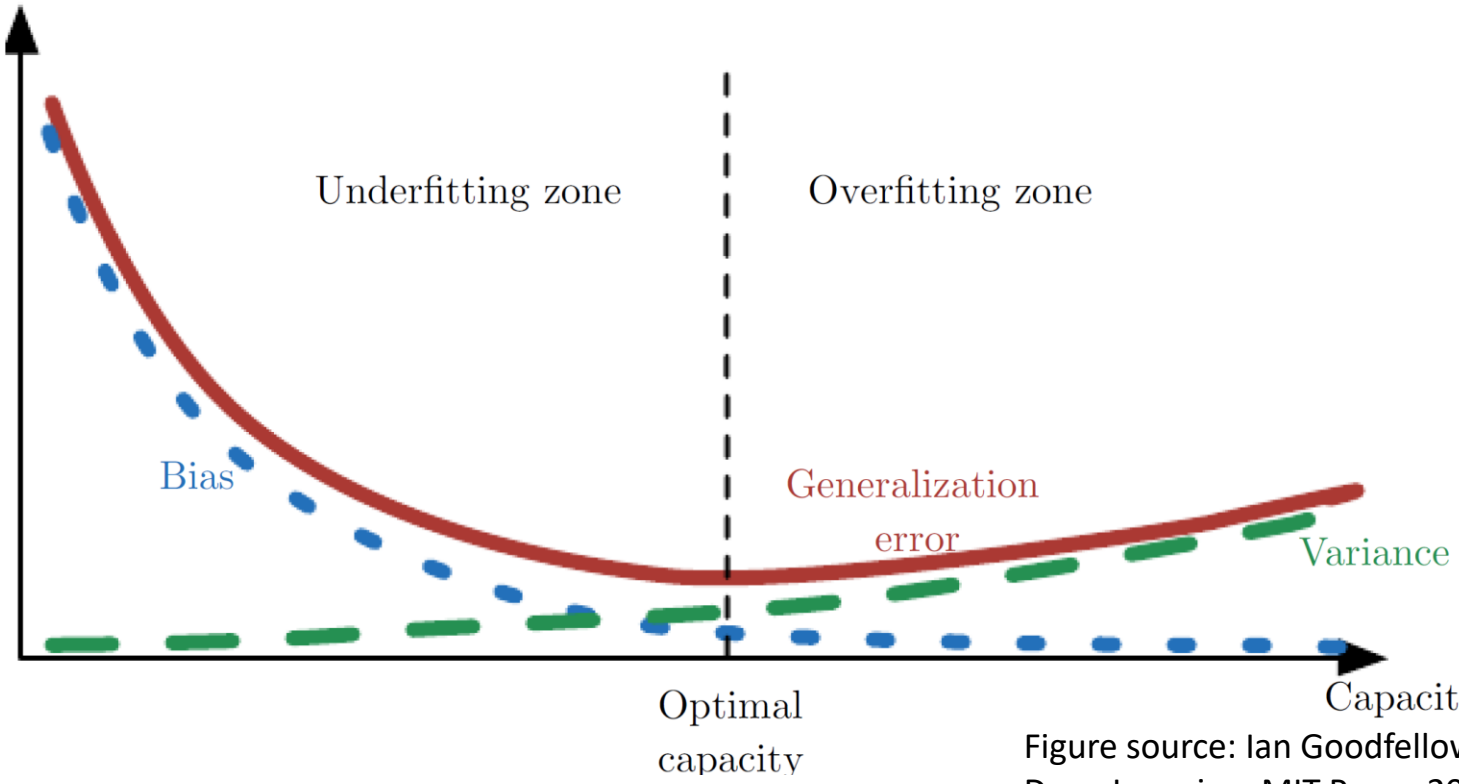


Figure source: Ian Goodfellow et al., Deep Learning, MIT Press, 2016.

Bias/Variance Tradeoff

- High bias :
  - Use a bigger network or Review the NN architecture
  - Tune more
- High Variance:
  - More data
  - Use regularization
  - Review the NN architecture
- Traditional ML applications
  - High bias, low variance
  - Low bias, high variance
  - Set a tradeoff
- Deep learning
  - Data is available and computation power is available so, use both the larger network, large amount of data and tune the NN to obtain low variance and low bias.

Regularization

- Example: modify the training criterion to include weight decay.
  - Minimize $J(w) = MSE_{train} + \lambda w^T w$.
  - Here λ is chosen to control the strength of our preferences for small weights.
- More generally, a model is regularized to learn a function $f(x, \theta)$ by adding a penalty (θ: Parameters)
- The penalty is called regularizer and is added to the cost function.
  - Example $w^T w$ is the regularizer in the above case.

Parameters and Hyperparameters

- Example
- Parameters: $W^{[1]}$, $b^{[1]}$, $W^{[2]}$, $b^{[2]}$, $W^{[3]}$, $b^{[3]}$ …
- Hyperparameters:
  - Learning rate
  - # hidden layers
  - # units in a hidden layer
  - # iterations for training (epochs)
  - Activation function in each layer
  - Capacity (for example degree of polynomial in regression)

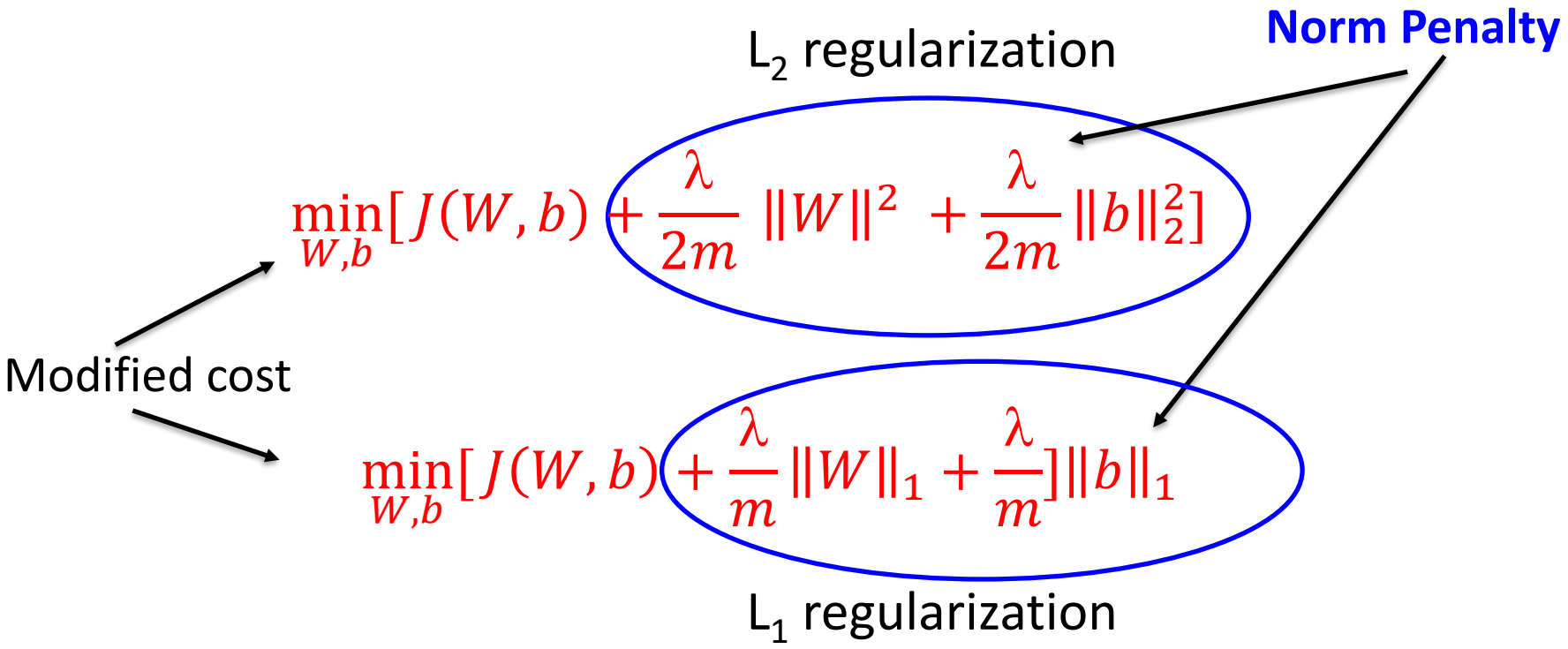
Regularization Methods

- There are many regularization strategies.
- Some put extra constraints on a machine learning model
  - Example: restrictions on the parameter values.
- Some add extra terms in the objective function that can be thought of as corresponding to a soft constraint on the parameter values.
- In the context of deep learning, most regularization strategies are based on regularizing estimators.
- An effective regularizer :
  - Reducing variance significantly while maintaining low bias.

Regularization Methods

- L_2 regularization
- L_1 regularization
- Data augmentation
- Multitask training
- Early stopping –tradeoff between training error and development error
- Dropout regularization

L_2 and L_1 Regularization



- Here λ is a hyperparameter that weights the relative contribution of the norm penalty term (relative to the standard objective function).

Data Augmentation

Crop

Rotate \ Flip

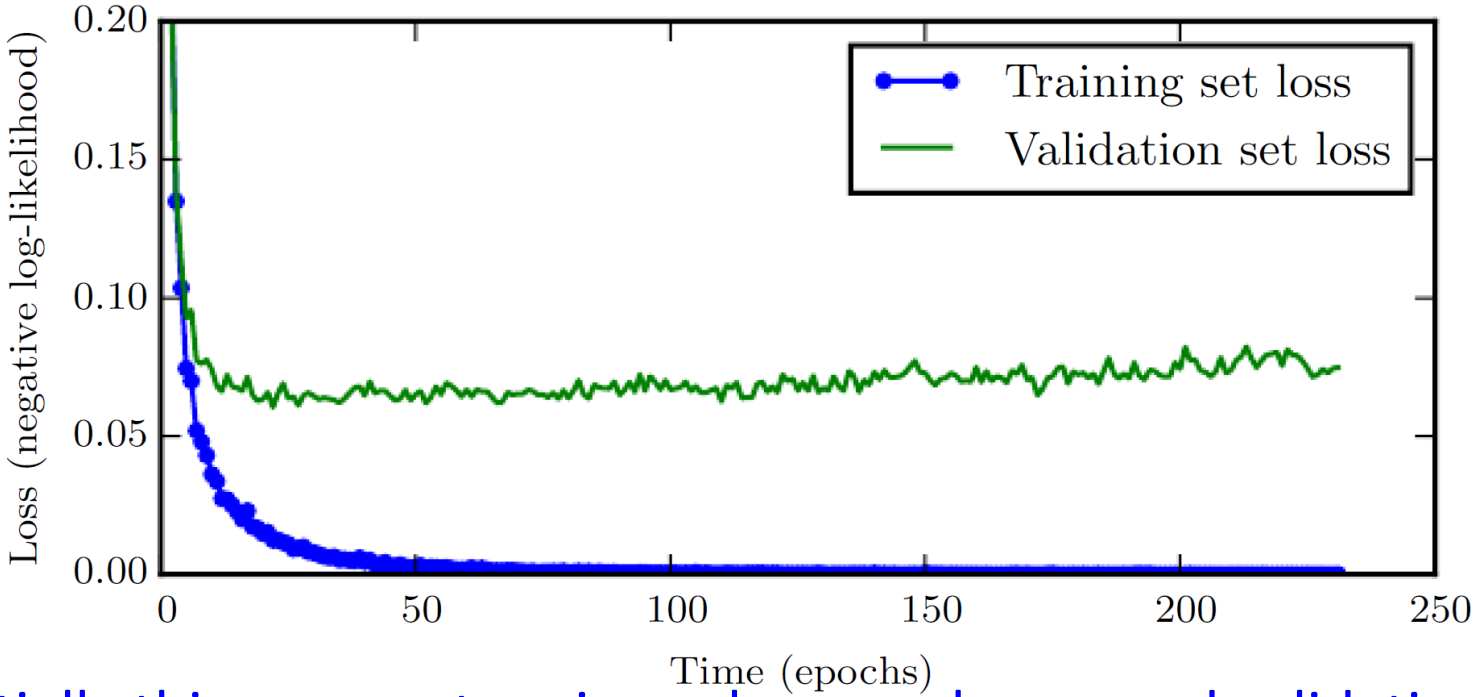
Add noise

Data Augmentation

- But one should be careful while adding noise or using rotation \flipping .
  - Example “ 6 “ and “9”.

Early Stopping

- When training large models with sufficient representational capacity to overfit the task, it is often observed that training error decreases steadily over time, but validation set error begins to rise again.



Essentially this means stopping when you have good validation error

Figure source: Ian Goodfellow et al, Deep Learning, MIT Press, 2016.

Parameter Tying and Sharing

- A common type of dependency that we often want to express is that certain parameters should be close to one another. Consider the following scenario: we have two models performing the same classification task (with the same set of classes) but with somewhat different input distributions.
- Formally, we have model A with parameters $w(A)$ and model B with parameters $w(B)$. The two models map the input to two different, but related outputs:
  - $\hat{y}(A) = f(w(A), x)$ and $\hat{y}(B) = g(w(B), x)$.

Parameter Sharing

- To use constraints: to force sets of parameters to be equal.

- A significant advantage: Only a subset of the parameters (the unique set) need to be stored in memory.

- Convolutional neural network—Extensive use of parameter sharing

Dropout Regularization

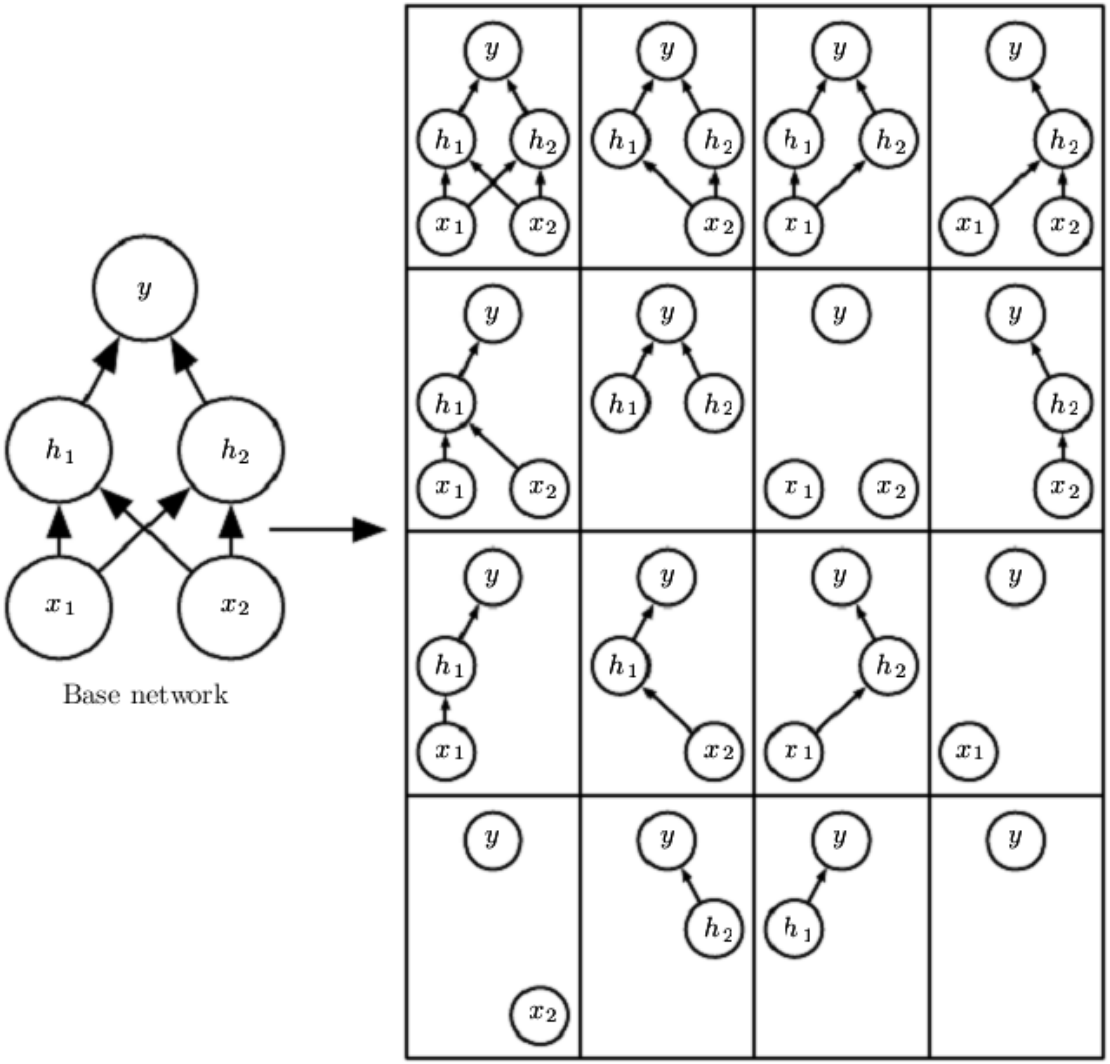


Figure source: Ian Goodfellow et al., Deep Learning, MIT Press, 2016.

Dropout Regularization

- Dropout provides a computationally inexpensive but powerful method of regularizing a broad family of models.
- At training (each iteration): Each unit is retained with a probability p.

- At test: The network is used as a whole.
  And the weights are scaled-down by a factor of p.

- Dropout trains a exponentially large set of networks.

Dropout Regularization

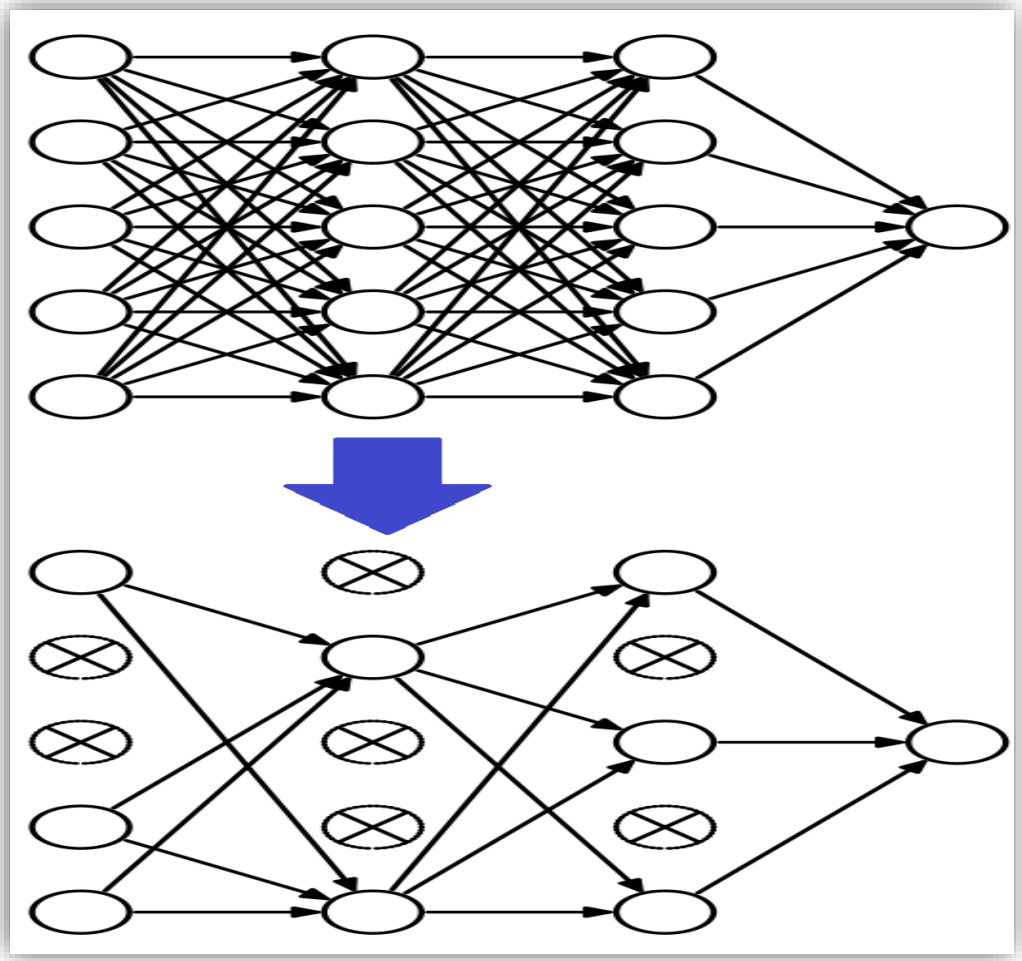
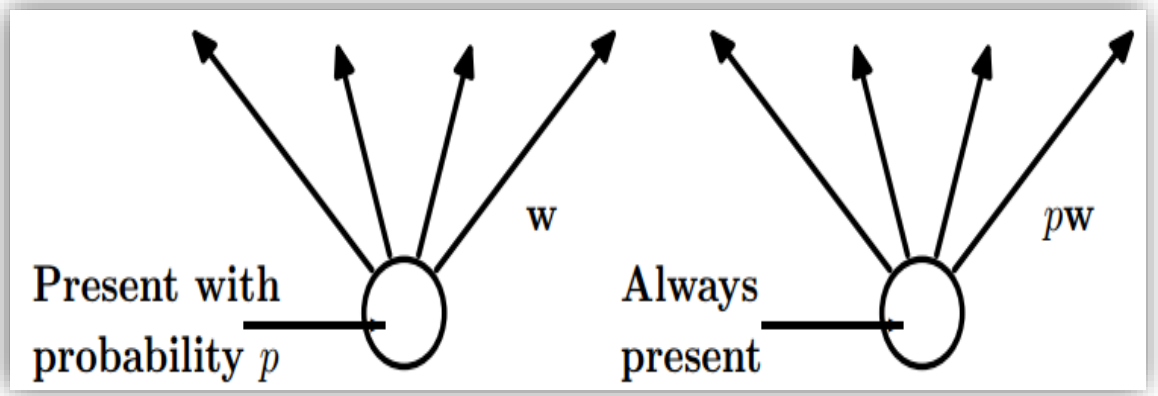


Figure source: Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov, J. Machine Learning Research, 2014

Dropout Regularization



Training

Testing

Figure source: Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov, J. Machine Learning Research, 2014

Dropout Regularization

- Equivalently (Inverted Dropout) –
- At training (each iteration): Each weight is scaled up with the factor $1/p$.
- At test: no scaling up is applied.
- For input layer, p =0.8 is chosen and for hidden layers p =0.5 is usually chosen.

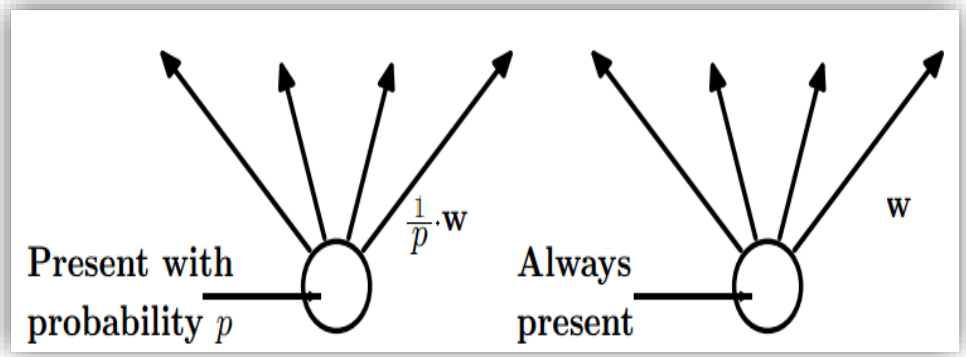


Figure source: Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov, J. Machine Learning Research, 2014

Dropout Regularization

- The feed-forward operation of a standard neural network is:

$$z_i^{(l+1)} = \boldsymbol{w}_i^{(l+1)} \boldsymbol{a}^{(l)} + b_i^{(l+1)}$$

$$a_i^{(l+1)} = f\left(z_i^{(l+1)}\right)$$

- With dropout, the feed-forward operation becomes:

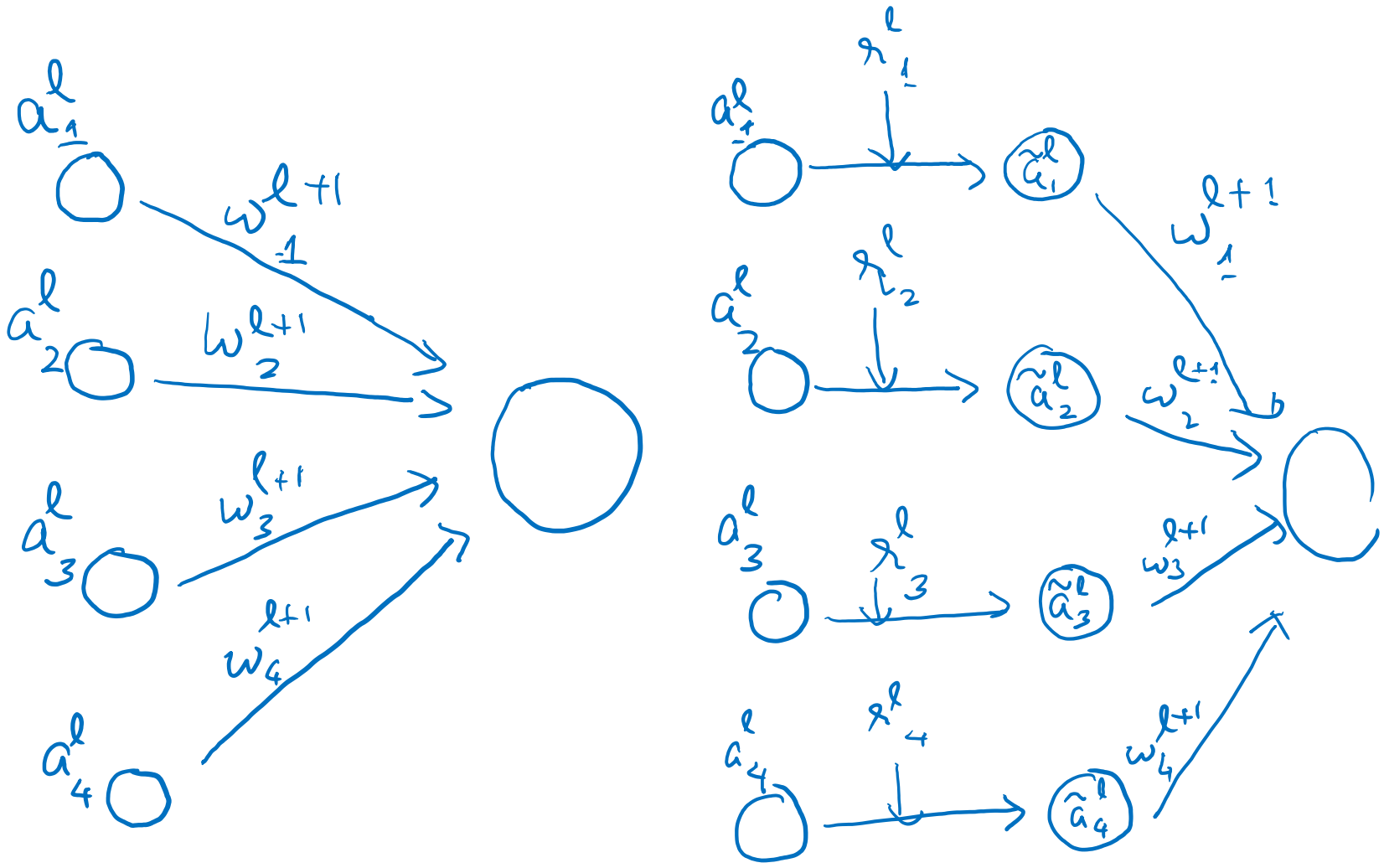
$$r_i^{(l)} \sim \text{Bernoulli}(p)$$

$$\tilde{\boldsymbol{a}}^{(l)} = \boldsymbol{r}^{(l)} * \boldsymbol{y}^{(l)}$$

$$z_i^{(l+1)} = \boldsymbol{w}_i^{(l+1)} \tilde{\boldsymbol{a}}^{(l)} + b_i^{(l+1)}$$

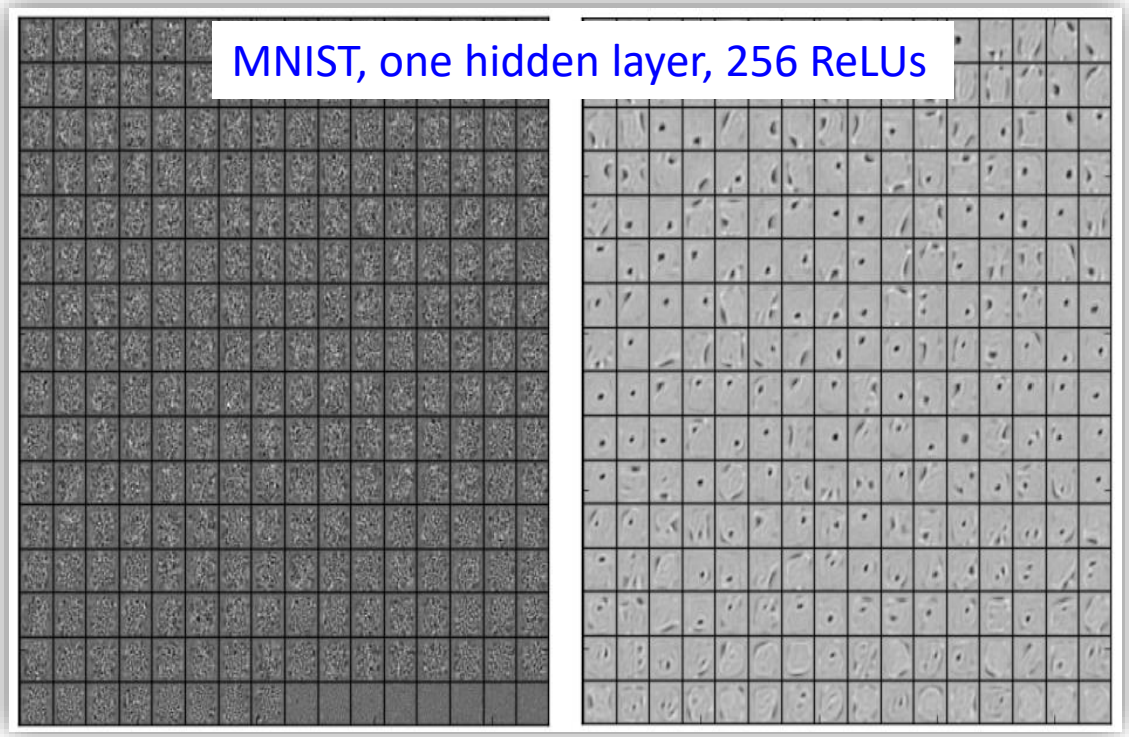
$$a_i^{(l+1)} = f\left(z_i^{(l+1)}\right)$$

Dropout Regularization



Dropout Implementation Results

- Without dropout, units tend to compensate for errors in other units. Results in overfitting, because that does not generalize to unseen data.
- Dropout prevents this by making the presence of other hidden units unreliable.

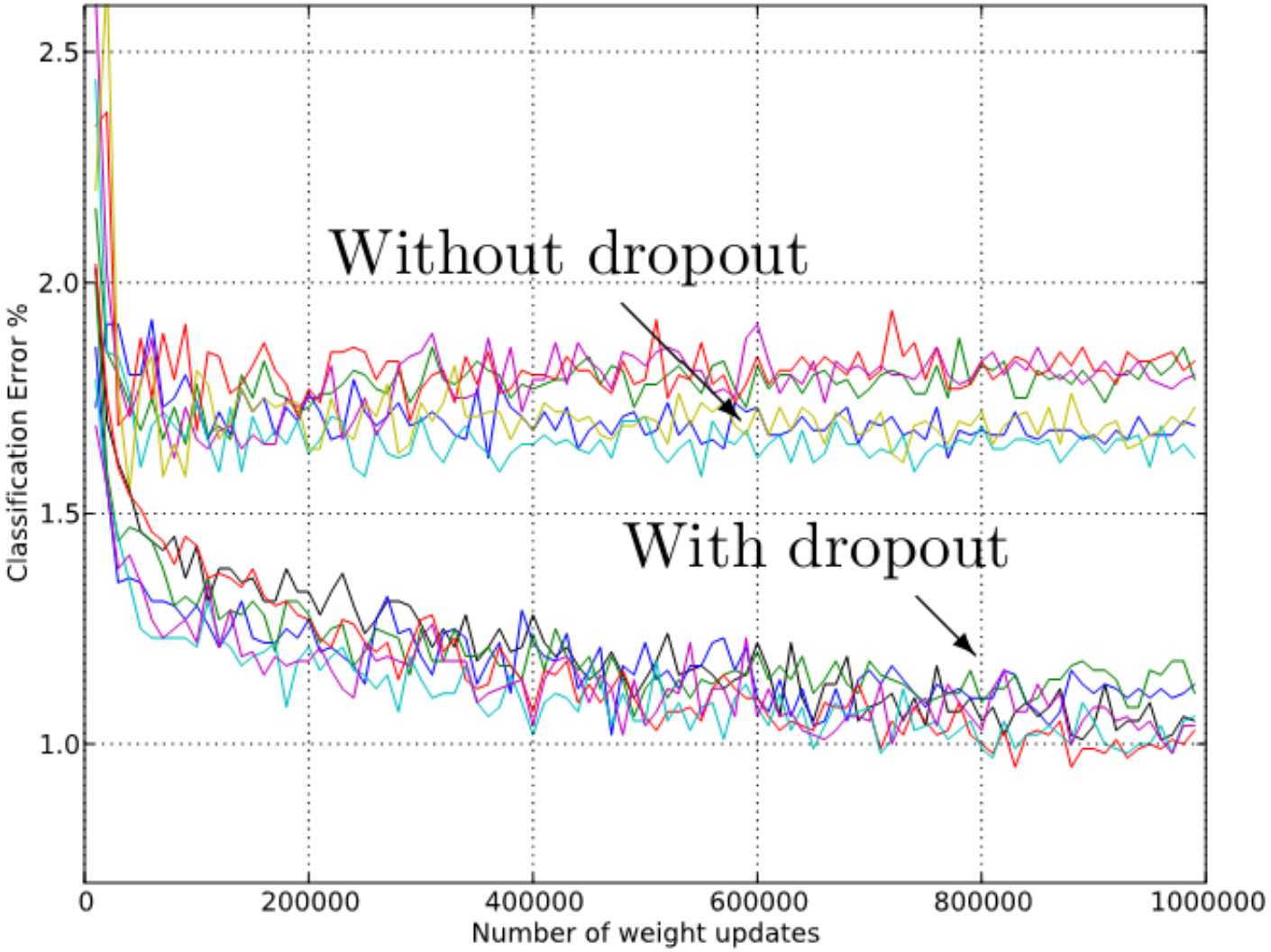


No dropout
Units have co-adapted. Each unit does not detect a meaningful feature.

Dropout ($p = 0.5$)
Units detect edges, strokes and spots in different parts of the image.

Source: Deep Learning Seminar Notes, School of Electrical Engineering, Tel Aviv University

Dropout Implementation Results



- Test error for different architectures with and without dropout.
- The networks have 2 to 4 hidden layers each with 1024 to 2048 units.

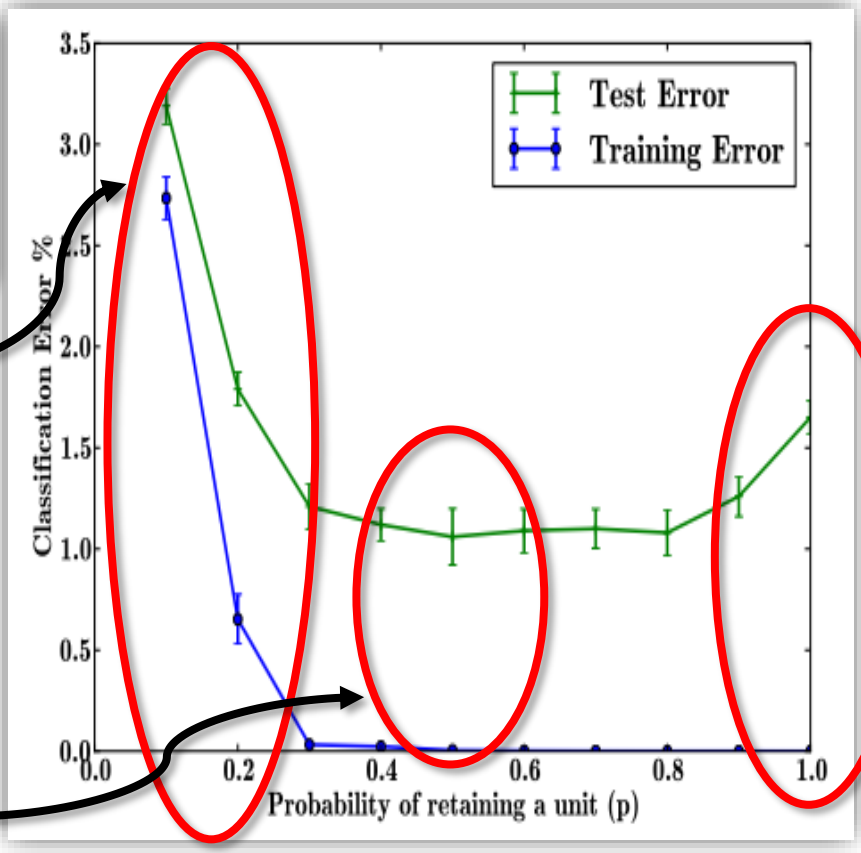
From: Srivastava et al. 2014, J. Machine Learning Research 15 (2014)

Dropout Rate and Performance

- An architecture of 784-2048-2048-2048-10 is used on the MNIST dataset.
- The dropout rate p is changed from small numbers (most units are dropped out) to 1.0 (no dropout).

High rate of dropout ($p < 0.3$)
- Training error is high
➔ Underfitting
- Very few units are turned on during training.

Best dropout rate ($p = 0.5$)
- Training error is low
- Test error is low
➔ Mission accomplished!



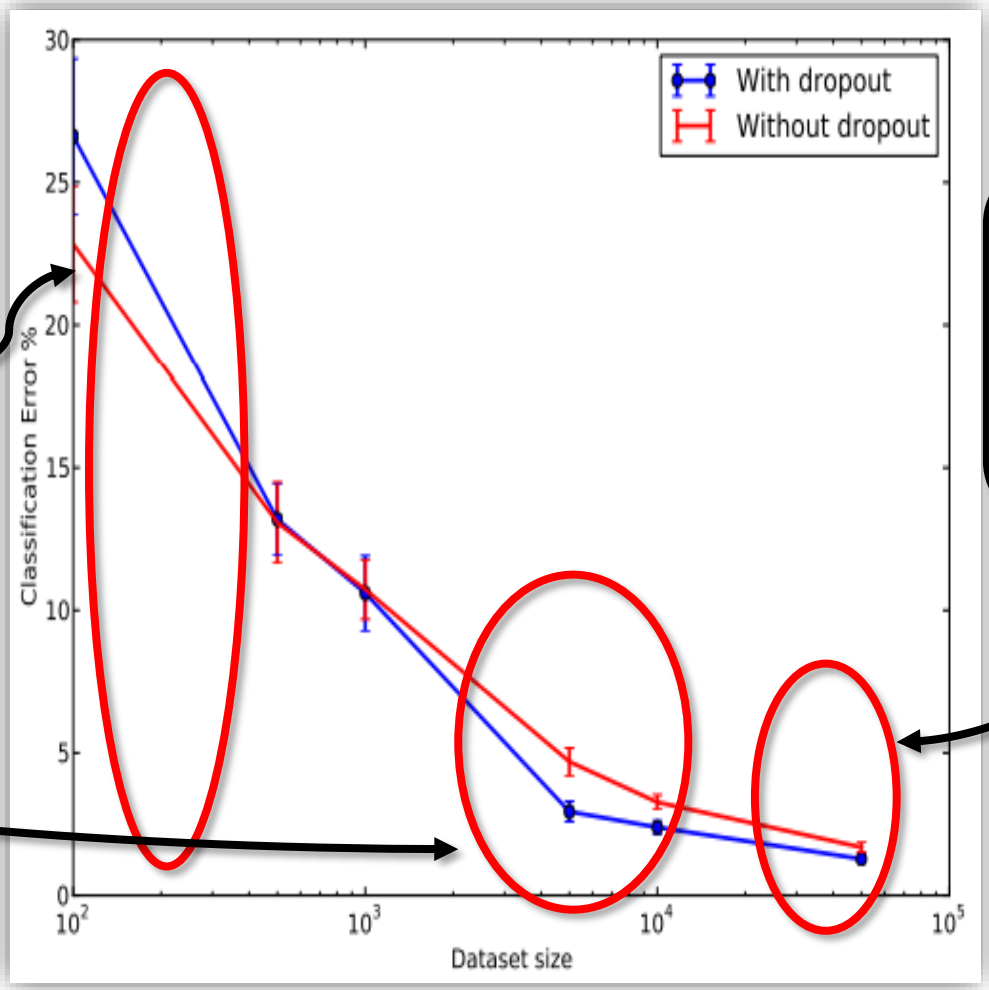
No dropout ($p = 1.0$)
- Training error is low
- Test error is high
➔ Overfitting

Source: Deep Learning Seminar Notes, School of Electrical Engineering, Tel Aviv University

Dataset Size and Performance

Extremely small data set
Dropout does not improve error rate, and even makes it worse.

Average to large data set
Dropout improves error rate.



Huge data set
Dropout barely improves the error rate. The data set is big enough, so that overfitting is not an issue.

Source: Deep Learning Seminar Notes, School of Electrical Engineering, Tel Aviv University

Dropout Benefits

- Dropout is scale-free: dropout does not penalize the use of large weights when needed.

- Dropout is invariant to parameter scaling: dropout is unaffected if weights in a certain layer are scaled up by a constant c, and the weights in another layer are scaled down by a constant c.

References

- Ian Goodfellow, Deep Learning, MIT Press, 2016.
- Andrew Ng Lectures on Introduction to Neural Network and Deep Learning (Coursera.org)
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun, Deep Residual Networks for Image Recognition, 2015 ; arxiv.org