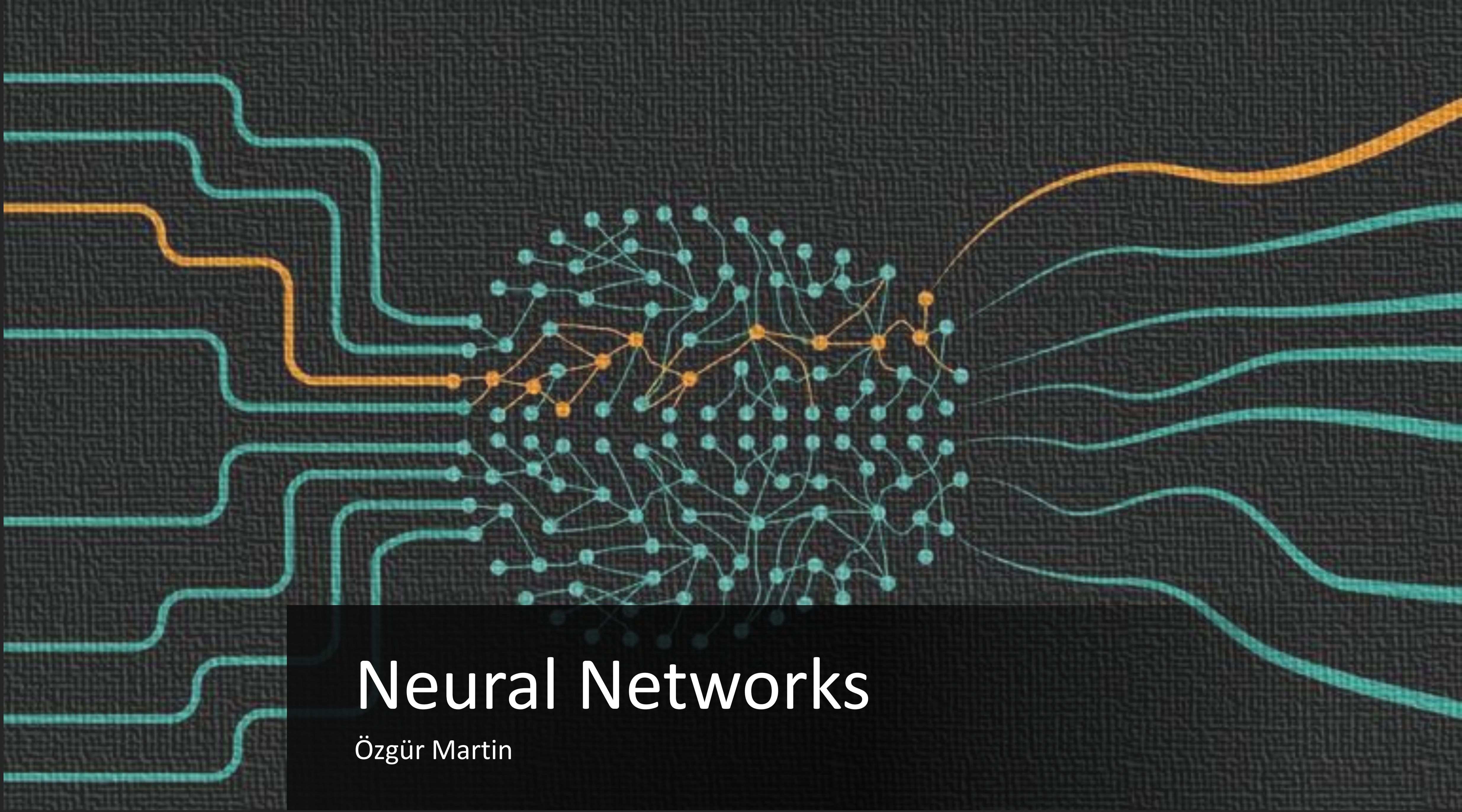
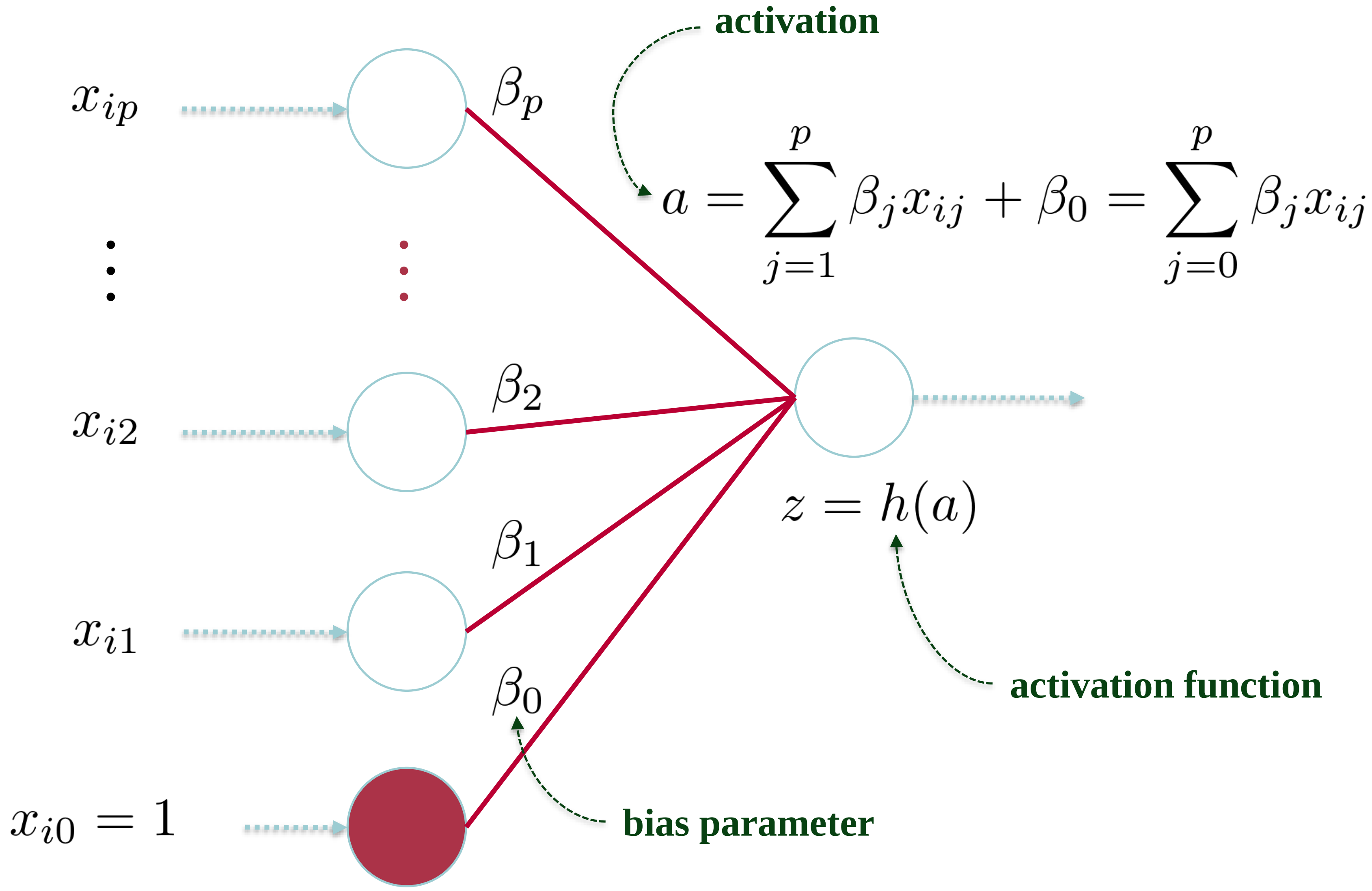


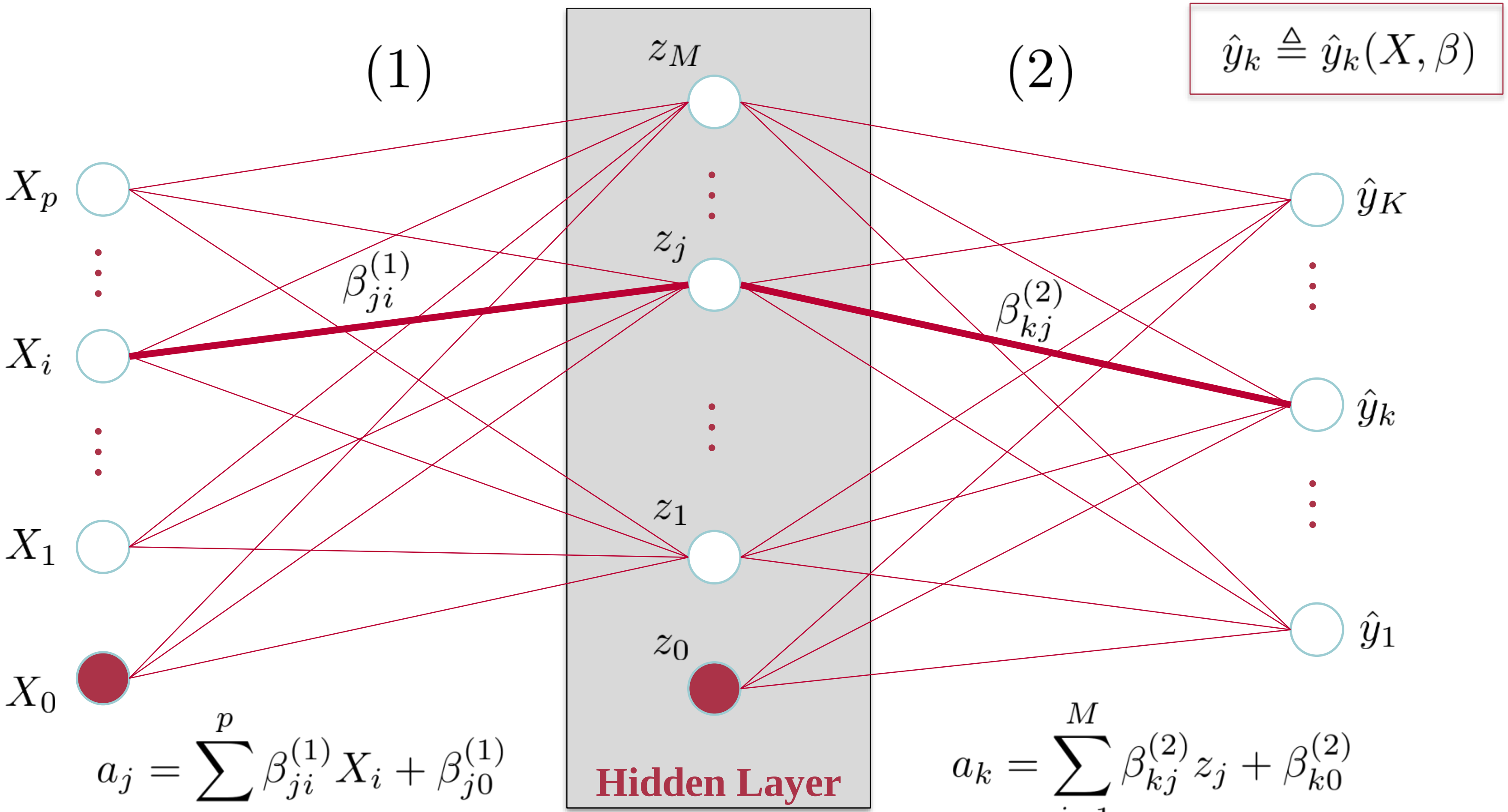
Neural Networks

Özgür Martin

Processing Units



Three-Layer (Feed-Forward) Neural Network



$$a_j = \sum_{i=1}^{p} \beta_{ji}^{(1)} X_i + \beta_{j0}^{(1)}$$

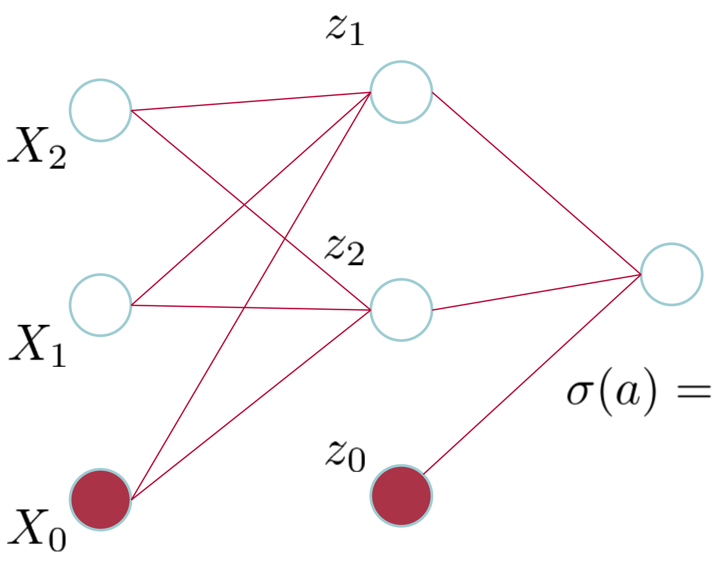
$$z_j = h(a_j), \; j = 1, \dots, M$$

$$a_k = \sum_{j=1}^{M} \beta_{kj}^{(2)} z_j + \beta_{k0}^{(2)}$$

$$\hat{y}_k = \sigma(a_k), \; k = 1, \dots, K$$

output unit activation function

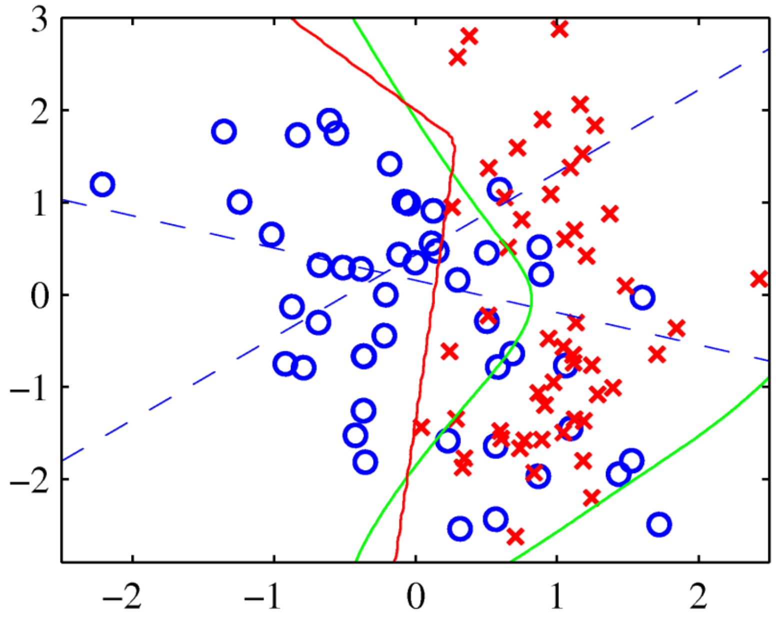
Binary Classification Example



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

$$h(a_j) = \tanh(a_j)$$

*



******Pattern Recognition and Machine Learning*, C. M. Bishop, Springer, 2006, pg. 232.

Activation Functions

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

Sigmoid Function

$$h(a_j) = \tanh(a_j)$$

"tanh" Function

$$h(a_j) = \max\{0, a_j\}$$

Rectified Linear Unit (ReLU)

$$\sigma(a_k) = a_k$$

Identity Function

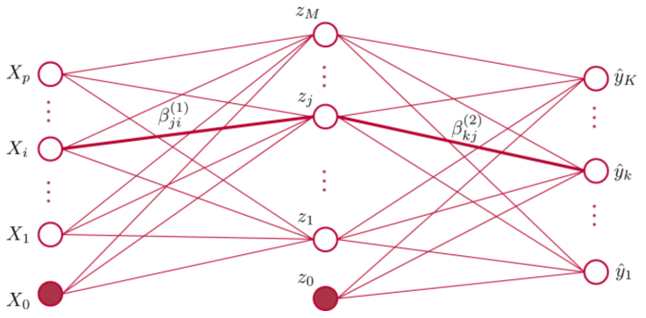
$$\sigma(a_k) = \frac{1}{1 + e^{-a_k}}$$

Sigmoid Function

$$\sigma(a_k) = \frac{e^{a_k}}{\sum_{j=1}^{K} e^{a_j}}$$

Softmax Function

Network Function



$$a_j = \sum_{i=1}^{p} \beta_{ji}^{(1)} X_i + \beta_{j0}^{(1)} \qquad a_k = \sum_{j=1}^{M} \beta_{kj}^{(2)} z_j + \beta_{k0}^{(2)}$$

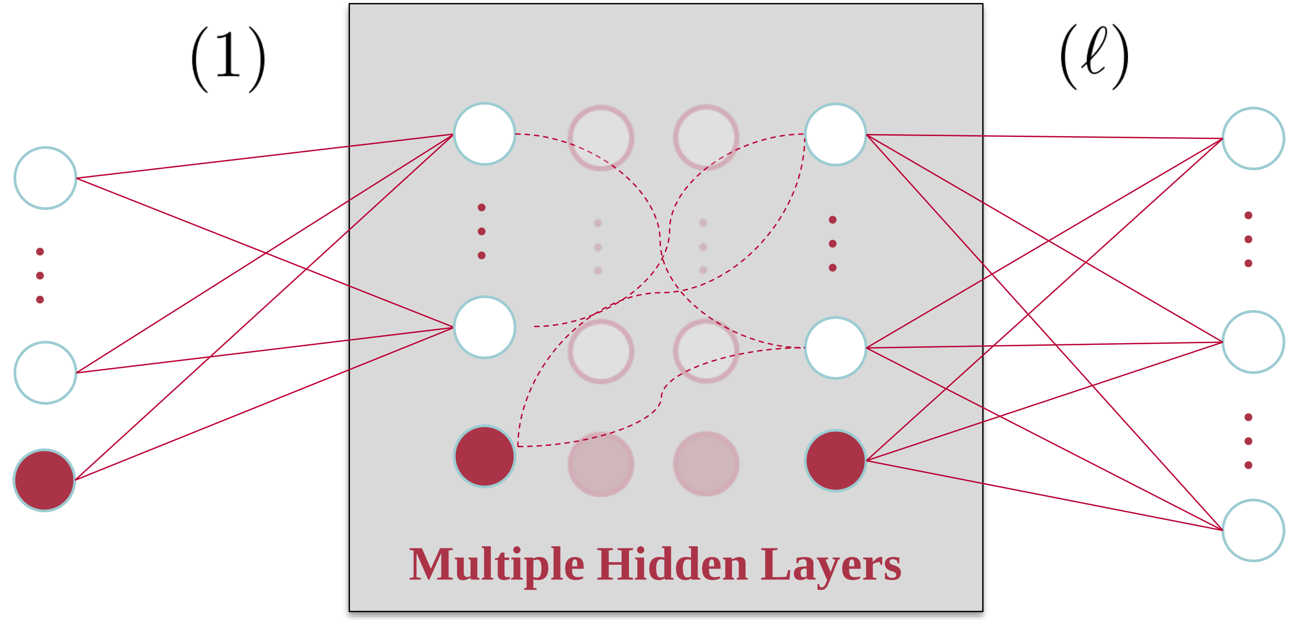
$$z_j = h(a_j), \;\; j = 1, \ldots, M \qquad \hat{y}_k = \sigma(a_k), \;\; k = 1, \ldots, K$$

$$\hat{y}_k(X, \beta) = \sigma \left(\sum_{j=1}^{M} \beta_{kj}^{(2)} h \left(\sum_{i=1}^{p} \beta_{ji}^{(1)} X_i + \beta_{j0}^{(1)} \right) + \beta_{k0}^{(2)} \right)$$

$$\boxed{X_0 = 1, \;\; z_0 = 1}$$

$$\hat{y}_k(X, \beta) = \sigma \left(\sum_{j=0}^{M} \beta_{kj}^{(2)} h \left(\sum_{i=0}^{p} \beta_{ji}^{(1)} X_i \right) \right)$$

Network Function



$$\hat{y}_k(X,\beta) = \sigma\left(\sum_j \beta_{kj}^{(\ell)} h\left(\sum_s \beta_{js}^{(\ell-1)} h\left(\dots h\left(\sum_i \beta_{ji}^{(1)} X_i\right)\dots\right)\right)\right)$$

Network Training

Error Functions

Regression

$$(x_i, y_i), i = 1, 2, \ldots, n$$

$$x_i \in \mathbb{R}^p \quad y_i \in \mathbb{R}^K$$

$$\boxed{E(\beta) = \sum_{i=1}^{n} E_i(\beta)}$$

Sum of Squares

$K = 1$

$$E(\beta) = \frac{1}{2}\sum_{i=1}^{n}(\hat{y}(x_i, \beta) - y_i)^2$$

$K > 1$

$$E(\beta) = \frac{1}{2}\sum_{i=1}^{n}\|\hat{y}(x_i, \beta) - y_i\|^2 = \frac{1}{2}\sum_{i=1}^{n}\sum_{k=1}^{K}(\hat{y}_k(x_i, \beta) - y_{ik})^2$$

$$\hat{y}_k = \sigma(a_k) = a_k$$

$$\boxed{\frac{\partial E_i}{\partial a_k} = \hat{y}_k(x_i, \beta) - y_{ik}}$$

Network Training

Error Functions

K-Separate Binary Classification

$$(x_i, y_i), i = 1, 2, \ldots, n$$

$$x_i \in \mathbb{R}^p \quad y_i \in \{0, 1\}^K$$

$$\boxed{E(\beta) = \sum_{i=1}^{n} E_i(\beta)}$$

Cross Entropy

$\boxed{K = 1}$

$$E(\beta) = -\sum_{i=1}^{n} \big(y_i \ln(\hat{y}(x_i, \beta)) + (1 - y_i) \ln(1 - \hat{y}(x_i, \beta))\big)$$

$\boxed{K > 1}$

$$E(\beta) = -\sum_{i=1}^{n} \sum_{k=1}^{K} \big(y_{ik} \ln(\hat{y}_k(x_i, \beta)) + (1 - y_{ik}) \ln(1 - \hat{y}_k(x_i, \beta))\big)$$

$$\hat{y}_k = \sigma(a_k) = \frac{1}{1 + e^{-a_k}}$$

$$\boxed{\frac{\partial E_i}{\partial a_k} = \hat{y}_k(x_i, \beta) - y_{ik}}$$

Network Training

Error Functions

Multiclass Classification

$$(x_i, y_i), i = 1, 2, \ldots, n \quad x_i \in \mathbb{R}^p$$

y_i is a unit vector in $\mathbb{R}^K$

$$\boxed{E(\beta) = \sum_{i=1}^{n} E_i(\beta)}$$

Cross Entropy

$$E(\beta) = -\sum_{i=1}^{n} \sum_{k=1}^{K} y_{ik} \ln(\hat{y}_k(x_i, \beta))$$

$$\hat{y}_k = \sigma(a_k) = \frac{e^{a_k}}{\sum_{j=1}^{K} e^{a_j}}$$

$$\boxed{\frac{\partial E_i}{\partial a_k} = \hat{y}_k(x_i, \beta) - y_{ik}}$$

Network Training

Optimization

$$\min_{\beta} \frac{1}{n} \sum_{i=1}^{n} E_i(\beta)$$

$$\boxed{\frac{1}{n} \sum_{i=1}^{n} \nabla E_i(\beta) = 0}$$

(Batch) Gradient Descent

$$\beta_{k+1} = \beta_k - \frac{\alpha_k}{n} \sum_{i=1}^{n} \nabla E_i(\beta_k)$$

k : iteration
$\propto_k$: learning rate

Stochastic Gradient Descent (SGD)

$$\beta_{k+1} = \beta_k - \alpha_k \nabla E_j(\beta_k), \quad j \in \{1, \ldots, n\}$$

Mini-batch SGD

$$\beta_{k+1} = \beta_k - \frac{\alpha_k}{|\mathcal{J}|} \sum_{j \in \mathcal{J}} \nabla E_j(\beta_k), \quad \mathcal{J} \subseteq \{1, \ldots, n\}$$

Network Training

Backpropagation

$$\nabla E(\beta) = \sum_{i=1}^{n} \nabla E_i(\beta) = 0 \qquad \frac{\partial E}{\partial \beta_{ts}^{(l)}} = \sum_{i=1}^{n} \frac{\partial E_i}{\partial \beta_{ts}^{(l)}} \quad ?$$

Recall

$$E(\beta) = -\sum_{i=1}^{n}\sum_{k=1}^{K} y_{ik} \ln(\hat{y}_k(x_i, \beta)) \qquad E(\beta) = \frac{1}{2}\sum_{i=1}^{n}\sum_{k=1}^{K} (\hat{y}_k(x_i, \beta) - y_{ik})^2 \qquad \frac{\partial E_i}{\partial a_k} = \hat{y}_k(x_i, \beta) - y_{ik}$$

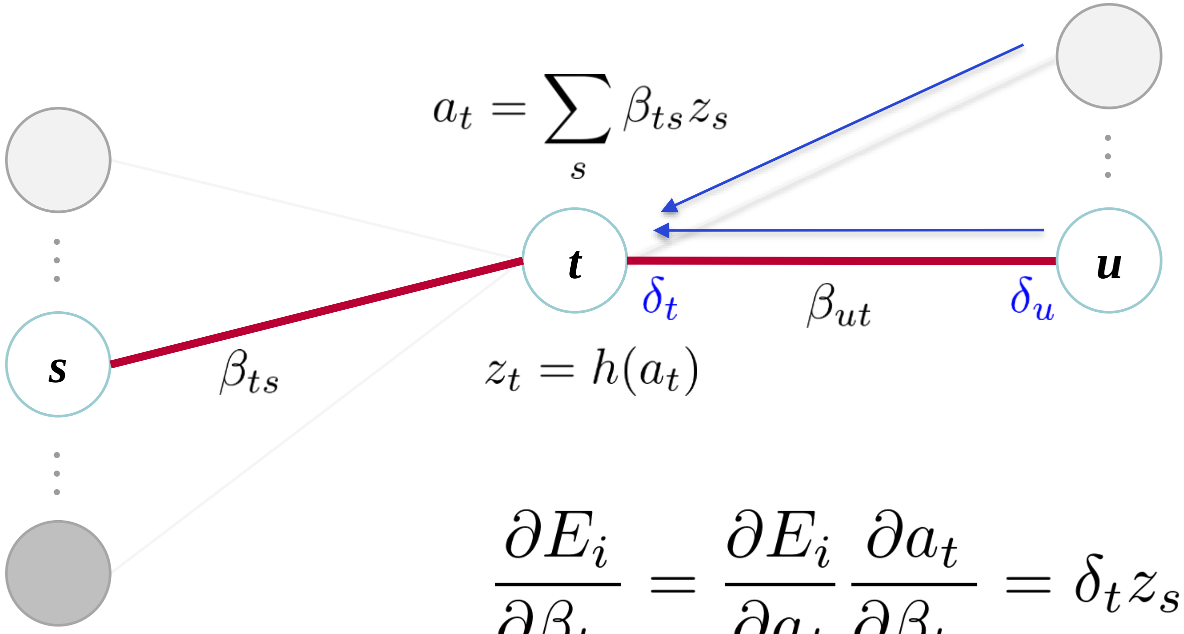
$$\hat{y}_k(X, \beta) = \sigma\left(\sum_j \beta_{kj}^{(\ell)} h\left(\sum_s \beta_{js}^{(\ell-1)} h\left(\ldots h\left(\sum_i \beta_{ji}^{(1)} X_i\right)\ldots\right)\right)\right)$$

Apply chain rule over and over again?

Yes <u>but</u> try it on the network!

Network Training

Backpropagation



$$\hat{y}_{ik} \triangleq \hat{y}_k(x_i, \beta)$$

$$\frac{\partial E_i}{\partial a_k} = \hat{y}_{ik} - y_{ik}$$

$$\frac{\partial E_i}{\partial \beta_{ts}} = \frac{\partial E_i}{\partial a_t} \frac{\partial a_t}{\partial \beta_{ts}} = \delta_t z_s$$

$$\delta_t \triangleq \frac{\partial E_i}{\partial a_t}$$

$$\delta_t = \frac{\partial E_i}{\partial a_t} = \sum_u \frac{\partial E_i}{\partial a_u} \frac{\partial a_u}{\partial a_t} = h'(a_t) \sum_u \delta_u \beta_{ut}$$

$$\delta_k = \frac{\partial E_i}{\partial a_k} = \hat{y}_{ik} - y_{ik}, \quad k = 1, \dots, K$$

Network Training

Backpropagation

Deep Learning: An Introduction for Applied Mathematicians

Catherine F. Higham* Desmond J. Higham†

January 19, 2018

Abstract

Multilayered artificial neural networks are becoming a pervasive tool in a host of application fields. At the heart of this deep learning revolution are familiar concepts from applied and computational mathematics; notably, in calculus, approximation theory, optimization and linear algebra. This article provides a very brief introduction to the basic ideas that underlie deep learning from an applied mathematics perspective. Our target audience includes postgraduate and final year undergraduate students in mathematics who are keen to learn about the area. The article may also be useful for instructors in mathematics who wish to enliven their classes with references to the application of deep learning techniques. We focus on three fundamental questions: what is a deep neural network? how is a network trained? what is the stochastic gradient method? We illustrate the ideas with a short MATLAB code that sets up and trains a network. We also show the use of state-of-the art software on a large scale image classification problem. We finish with references to the current literature.

Network Training

Backpropagation

Apply an input vector x_i and propagate it through the network using

$$a_t = \sum_s \beta_{ts} z_s \qquad z_t = h(a_t) \qquad \hat{y}_k = \sigma(a_k)$$

Evaluate the following for the output units:

$$\delta_k = \frac{\partial E_i}{\partial a_k} = \hat{y}_{ik} - y_{ik}, \quad k = 1, \ldots, K$$

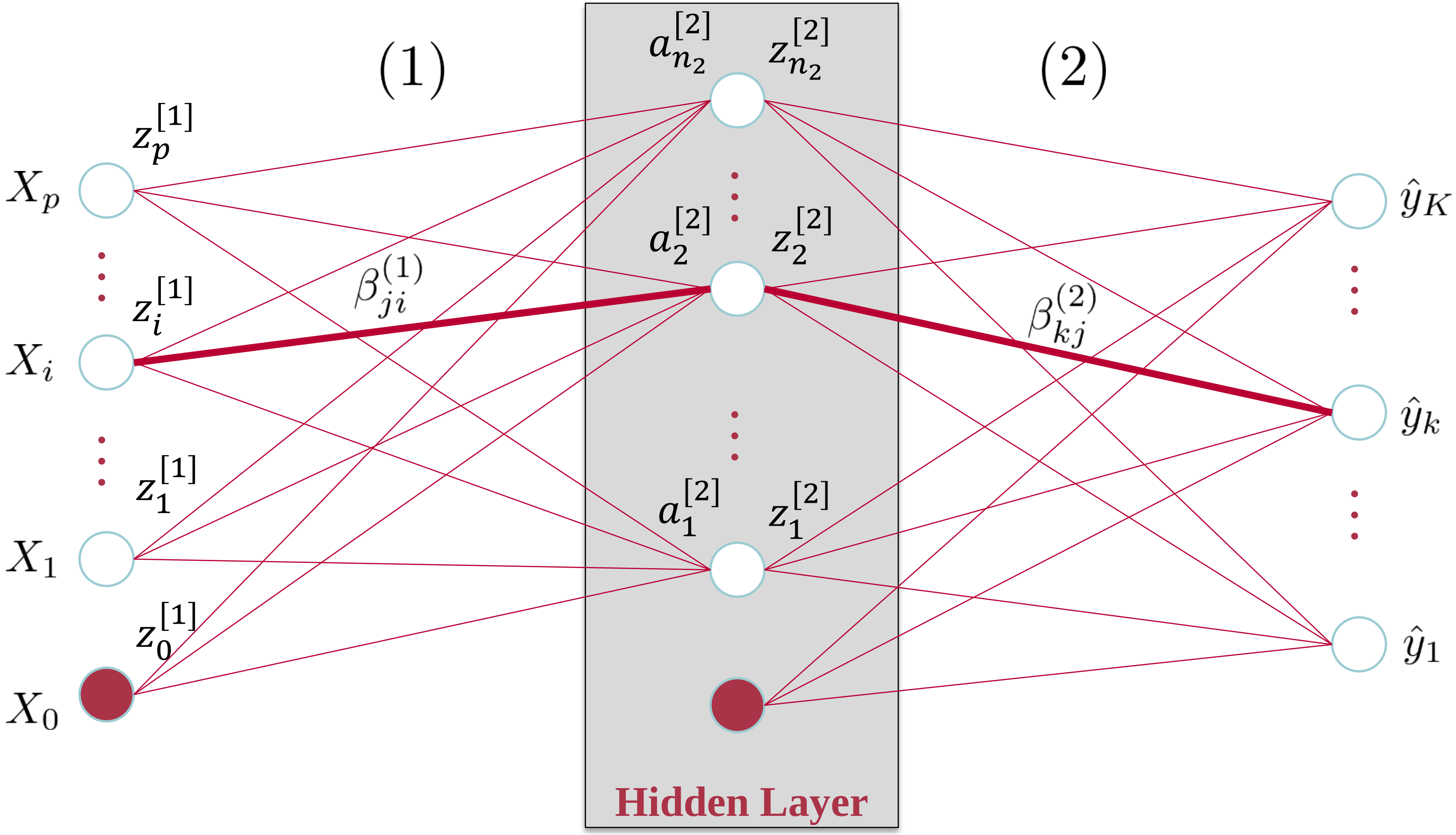
Backpropagate the following for the hidden units:

$$\delta_t = \frac{\partial E_i}{\partial a_t} = \sum_u \frac{\partial E_i}{\partial a_u} \frac{\partial a_u}{\partial a_t} = h'(a_t) \sum_u \delta_u \beta_{ut}$$

Evaluate the components of the gradient:

$$\frac{\partial E_i}{\partial \beta_{ts}} = \frac{\partial E_i}{\partial a_t} \frac{\partial a_t}{\partial \beta_{ts}} = \delta_t z_s$$

Network Training



$$l = 1,2,\dots,L$$

$$a^{[l]} = (a^{[l]}_0, a^{[l]}_1, \dots, a^{[l]}_{n_l})$$

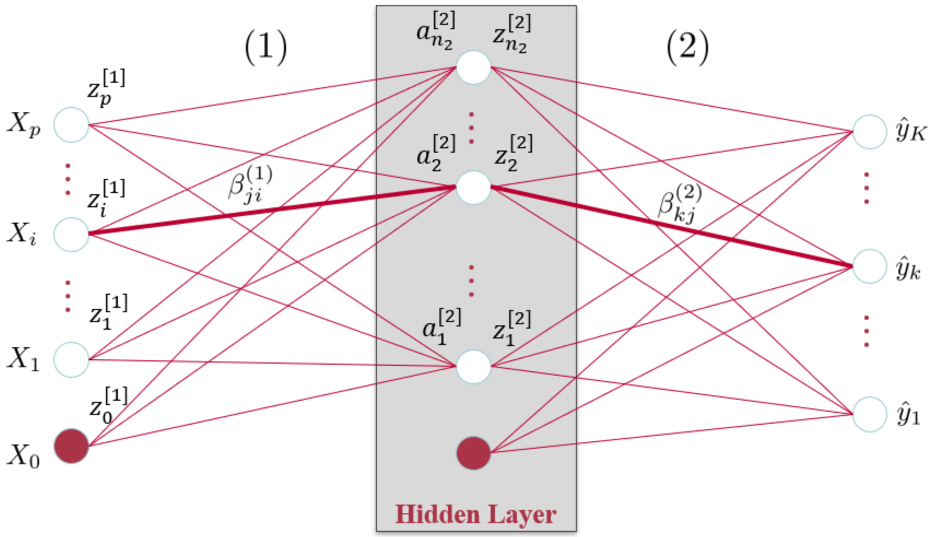
$$a^{[l]}_i = \sum_{j=0}^{n_{l-1}} \beta^{(l-1)}_{ji} z^{[l-1]}_j$$

$$a^{[l]} = W^{[l-1]} z^{[l-1]}$$

$$z^{[l]} = (z^{[l]}_0, z^{[l]}_1, \dots, z^{[l]}_{n_k})$$

$$z^{[l]} = h(a^{[l]})$$

Network Training



$$a^{[l]} = (a_0^{[l]}, a_1^{[l]}, \ldots, a_{n_l}^{[l]})$$

$$a_i^{[l]} = \sum_{j=0}^{n_{l-1}} \beta_{ji}^{(l-1)} z_j^{[l-1]}$$

$$a^{[l]} = W^{[l-1]} z^{[l]}$$

$$z^{[l]} = (z_0^{[l]}, z_1^{[l]}, \ldots, z_{n_k}^{[l]})$$

$$z^{[l]} = h(a^{[l]})$$

For counter 1 upto Niter

Choose an integer k uniformly at random from $\{1, 2, \ldots, N\}$
x_k is current training data point
$z^{[1]} = x_k$

For $l = 2$ up to L

$$a^{[l]} = W^{[l-1]} z^{[l-1]}$$
$$z^{[l]} = h(a^{[l]})$$
$$D^{[l]} = diag(h'(a^{[l]}))$$

end

$$\delta^{[L]} = D^{[L]}(z^{[L]} - y_k)$$

For $l = L - 1$ down to 2

$$\delta^{[l]} = D^{[l]}(W^{[l]})^T \delta^{[l+1]}$$

end

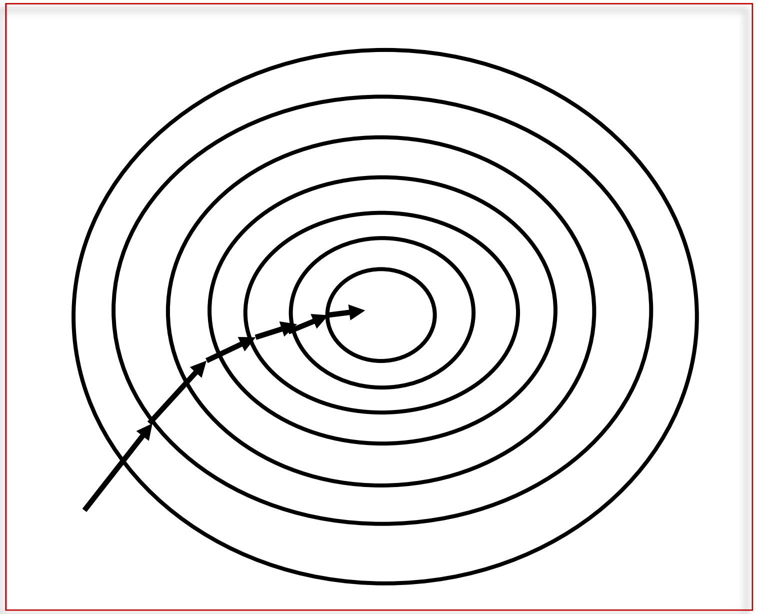
For $l = L - 1$ down to 1

$$W^{[l]} = W^{[l]} - \eta \delta^{[l]} {a^{[l-1]}}^T$$

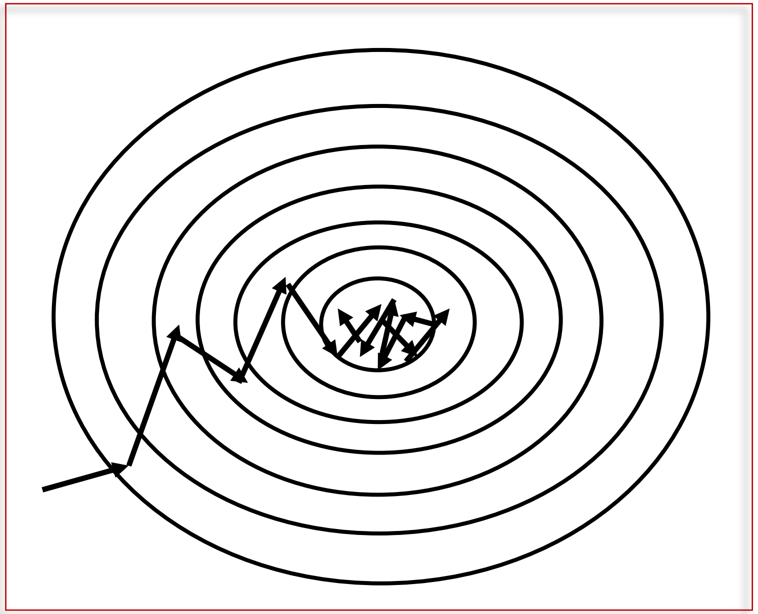
end

end

Network Training
GD vs. SGD



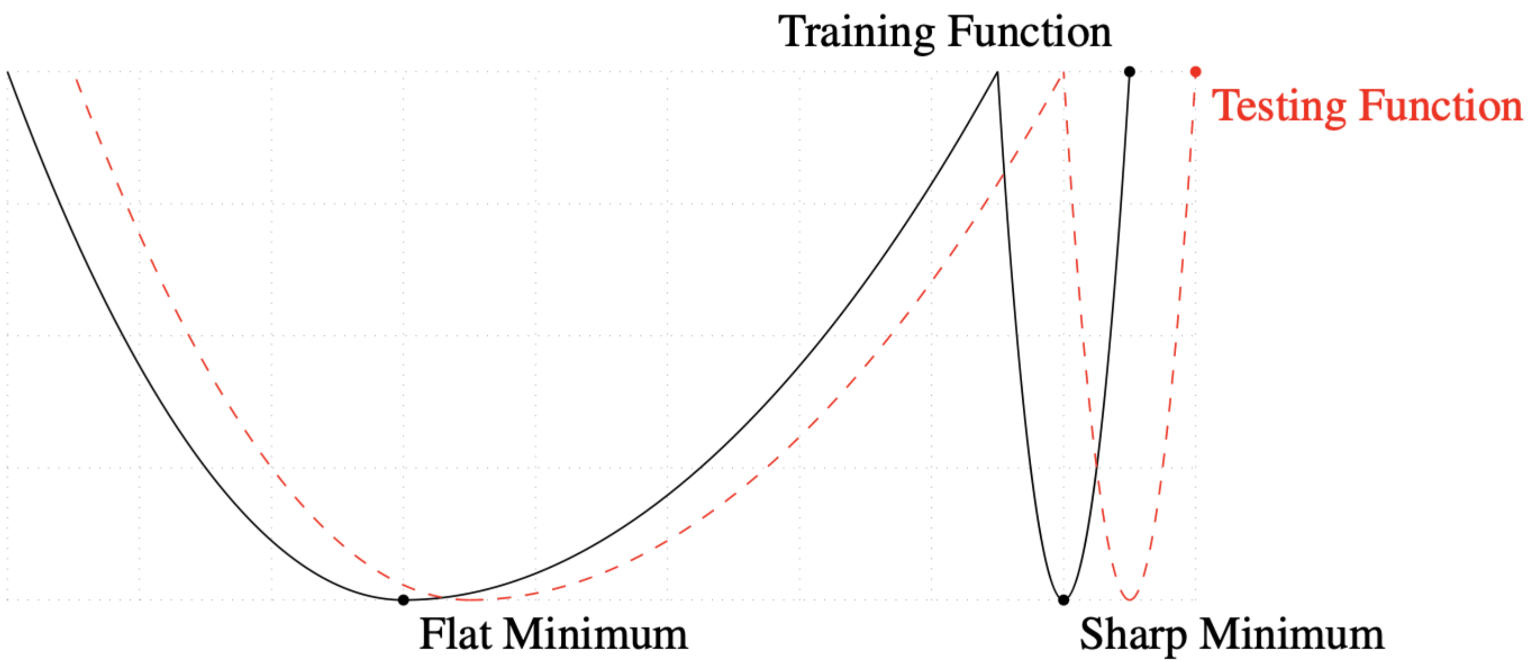
Gradient Descent



Stochastic Gradient Descent

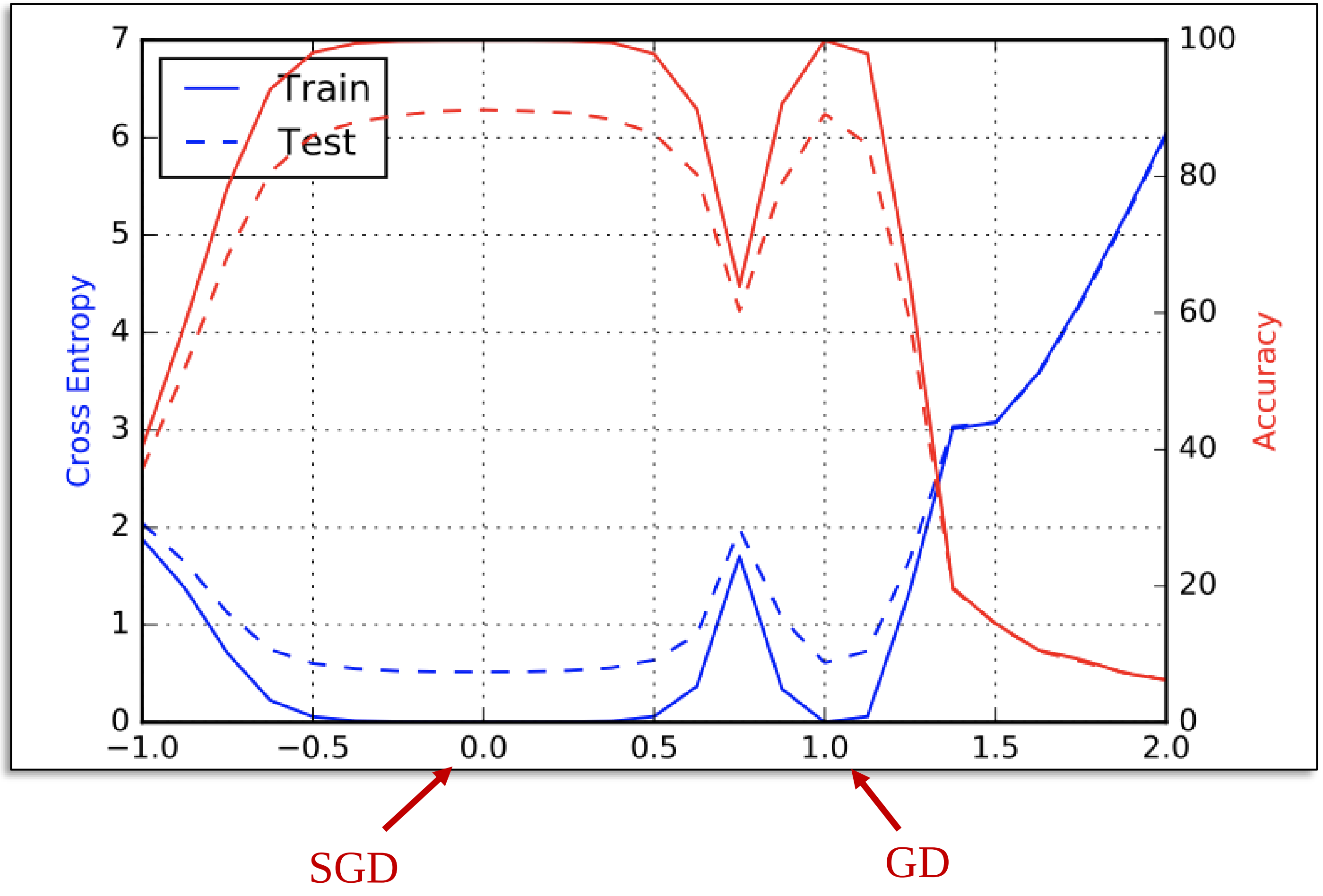
$$\beta_{k+1} = \beta_k - \frac{\alpha_k}{|\mathcal{J}|} \sum_{j \in \mathcal{J}} \nabla E_j(\beta_k), \qquad \mathcal{J} \subseteq \{1, \ldots, n\}$$

Network Training
Flat vs. Sharp Minima



*Keskar et al. 2017

Network Training
Flat vs. Sharp Minima



Deep Convolutional Neural Net on CIFAR-10

*Keskar et al. 2017

$$\min_{\beta} \underbrace{\frac{1}{n}\sum_{i=1}^{n} E_i(\beta)}_{F(\beta)} \underset{\beta\leftrightarrow w}{\equiv} \min_{w} F(w)$$

$$g(w_k, \xi_k) = \frac{1}{|\mathcal{J}|}\sum_{j\in\mathcal{J}_{\xi_k}} \nabla E_j(w_k), \qquad \mathcal{J}_{\xi_k} \subseteq \{1,\ldots,n\}$$

$$\boxed{w_{k+1} = w_k - \alpha_k g(w_k, \xi_k)}$$

Analysis

Assumption 1. Let $F : \mathbb{R}^p \to \mathbb{R}$ be continuously differentiable and the gradient $\nabla F : \mathbb{R}^p \to \mathbb{R}^p$ be Lipschitz continuous with Lipschitz constant L. That is,

$$\|\nabla F(w) - \nabla F(v)\|_2 \leq L\|w - v\|_2 \quad \text{for all } w, v \in \mathbb{R}^p.$$

Proposition 1. Under Assumption 1, $F : \mathbb{R}^p \to \mathbb{R}$ satisfies

$$F(w) \leq F(v) + \nabla F(v)^{\mathsf{T}}(w - v) + \frac{L}{2}\|w - v\|_2^2 \quad \text{for any } w, v \in \mathbb{R}^p.$$

Proof. Exercise...

Definition. $\mathbb{E}_{\xi_k}[\ \cdot\ ] = \mathbb{E}[\ \cdot \mid w_k]$

Lemma 1. Under Assumption 1, the iterates of SGD satisfies

$$\mathbb{E}_{\xi_k}[F(w_{k+1})] - F(w_k) \leq -\alpha_k \nabla F(w_k)^\intercal \mathbb{E}_{\xi_k}[g(w_k, \xi_k)] + \frac{1}{2}\alpha_k^2 L \mathbb{E}_{\xi_k}[\|g(w_k, \xi_k)\|_2^2]$$

for all $k = 1, \ldots, T$.

Proof. By Assumption 1 and Proposition 1, we have

$$\begin{aligned} F(w_{k+1}) - F(w_k) &\leq \nabla F(w_k)^\intercal (w_{k+1} - w_k) + \frac{L}{2}\|w_{k+1} - w_k\|_2^2 \\ &\leq -\alpha_k \nabla F(w_k)^\intercal g(w_k, \xi_k) + \frac{1}{2}\alpha_k^2 L \|g(w_k, \xi_k)\|_2^2. \end{aligned}$$

Taking expectation with respect to ξ_k and noting that w_k does not depend on ξ_k, we obtain

$$\begin{aligned} \mathbb{E}_{\xi_k}[F(w_{k+1})] - F(w_k) &\leq -\alpha_k \mathbb{E}_{\xi_k}[F(w_k)^\intercal g(w_k, \xi_k)] + \frac{1}{2}\alpha_k^2 L \|g(w_k, \xi_k)\|_2^2 \\ &= -\alpha_k \nabla F(w_k)^\intercal \mathbb{E}_{\xi_k}[g(w_k, \xi_k)] + \frac{1}{2}\alpha_k^2 L \mathbb{E}_{\xi_k}[\|g(w_k, \xi_k)\|_2^2] \end{aligned}$$

$\blacksquare$

Assumption 2. There exists $M \geq 0$ and $M_G \geq 1$ such that

$$\mathbb{E}_{\xi_k}\left[\|g(w_k,\xi_k)\|_2^2\right] \leq M + M_G\|\nabla F(w_k)\|_2^2$$

Lemma 2. Under Assumption 1 and Assumption 2, the iterates of SGD satisfies

$$\mathbb{E}_{\xi_k}\left[F(w_{k+1})\right] - F(w_k) \leq -(1 - \frac{1}{2}\alpha_k L M_G)\alpha_k\|\nabla F(w_k)\|_2^2 + \frac{1}{2}\alpha_k^2 LM$$

for all $k = 1, \ldots, T$.

Proof. Using Lemma 1 and the assumptions leads to the desired result:

$$\begin{aligned}
\mathbb{E}_{\xi_k}\left[F(w_{k+1})\right] - F(w_k) &\leq -\alpha_k \nabla F(w_k)^\intercal \mathbb{E}_{\xi_k}\left[g(w_k,\xi_k)\right] + \frac{1}{2}\alpha_k^2 L \mathbb{E}_{\xi_k}\left[\|g(w_k,\xi_k)\|_2^2\right] \\
&\leq -\alpha_k\|\nabla F(w_k)\|_2^2 + \frac{1}{2}\alpha_k^2 L(M + M_G\|\nabla F(w_k)\|_2^2) \\
&= -(1 - \frac{1}{2}\alpha_k L M_G)\alpha_k\|\nabla F(w_k)\|_2^2 + \frac{1}{2}\alpha_k^2 LM \qquad \blacksquare
\end{aligned}$$

Definition (Total Expectation). $\mathbb{E}\left[F(w_k)\right] = \mathbb{E}_{\xi_1}\left[\mathbb{E}_{\xi_2}\left[\ldots\mathbb{E}_{\xi_{k-1}}\left[F(w_k)\right]\right]\right]$

Theorem 1 (nonconvex, fixed stepsize). Under Assumption 1 and Assumption 2, suppose SGD is run with fixed stepsize, $\alpha_k = \alpha$, $k = 1, \ldots, T$ satisfying $0 < \alpha \leq \frac{1}{LM_G}$. Then, we have

$$\mathbb{E}\left[\frac{1}{T}\sum_{k=1}^{T}\|\nabla F(w_k)\|_2^2\right] \leq \alpha LM + \frac{2(F(w_1) - F_*)}{T\alpha},$$

where F_* is the minimum value of F.

Analysis

Theorem 1 (nonconvex, fixed stepsize). Under Assumption 1 and Assumption 2, suppose SGD is run with fixed stepsize, $\alpha_k = \alpha$, $k = 1, \dots, T$ satisfying $0 < \alpha \le \frac{1}{LM_G}$. Then, we have

$$\mathbb{E}\left[\frac{1}{T}\sum_{k=1}^{T}\|\nabla F(w_k)\|_2^2\right] \le \alpha LM + \frac{2(F(w_1) - F_*)}{T\alpha},$$

where F_* is the minimum value of F.

Proof. Using Lemma 2 and taking the total expectation yields

$$\begin{aligned}\mathbb{E}\left[F(w_{k+1})\right] - \mathbb{E}\left[F(w_k)\right] &\le -(1 - \frac{1}{2}\alpha LM_G)\alpha\mathbb{E}\left[\|\nabla F(w_k)\|_2^2\right] + \frac{1}{2}\alpha^2 LM \\ &\le -\frac{1}{2}\alpha\mathbb{E}\left[\|\nabla F(w_k)\|_2^2\right] + \frac{1}{2}\alpha^2 LM.\end{aligned}$$

Then, summing over $k = 1, \dots, T$ leads to

$$F_* - F(w_1) \le \mathbb{E}\left[F(w_{T+1})\right] - F(w_1) \le -\frac{1}{2}\alpha\sum_{k=1}^{T}\mathbb{E}\left[\|\nabla F(w_k)\|_2^2\right] + \frac{1}{2}T\alpha^2 M.$$

This implies

$$\sum_{k=1}^{T}\mathbb{E}\left[\|\nabla F(w_k)\|_2^2\right] \le T\alpha LM + \frac{2(F(w_1) - F_*)}{\alpha}. \qquad \blacksquare$$

Theorem 2 (nonconvex, diminishing stepsize). Under Assumption 1 and Assumption 2, suppose SGD is run with stepsizes satisfying

$$\sum_{k=1}^{\infty} \alpha_k = \infty \quad \text{and} \quad \sum_{k=1}^{\infty} \alpha_k^2 < \infty.$$

Then, we have

$$\lim_{T\to\infty} \mathbb{E}\left[\sum_{k=1}^{T} \alpha_k \|\nabla F(w_k)\|_2^2\right] < \infty.$$

Therefore,

$$\mathbb{E}\left[\frac{1}{A_T}\sum_{k=1}^{T} \alpha_k \|\nabla F(w_k)\|_2^2\right] \to 0 \quad \text{as} \quad T \to \infty,$$

where $A_T = \sum_{k=1}^{T} \alpha_k$.

Proof. Since $\alpha_k \to 0$ as $k \to \infty$, we can assume that there exists $k \in \mathbb{N}$ satisfying $\alpha_k L M_G \leq 1$. Using now Lemma 2 and taking total expectation gives

$$\begin{aligned}\mathbb{E}\left[F(w_{k+1})\right] - \mathbb{E}\left[F(w_k)\right] &\leq -(1 - \frac{1}{2}\alpha_k L M_G)\alpha_k \mathbb{E}\left[\|\nabla F(w_k)\|_2^2\right] + \frac{1}{2}\alpha_k^2 L M \\ &\leq -\frac{1}{2}\alpha_k \mathbb{E}\left[\|\nabla F(w_k)\|_2^2\right] + \frac{1}{2}\alpha_k^2 L M\end{aligned}$$

Then, summing over $k = 1, \ldots, T$ leads to

$$F_* - F(w_1) \leq \mathbb{E}\left[F(w_{T+1})\right] - F(w_1) \leq -\frac{1}{2}\sum_{k=1}^{T} \alpha_k \mathbb{E}\left[\|\nabla F(w_k)\|_2^2\right] + \frac{1}{2} L M \sum_{k=1}^{T} \alpha_k^2.$$

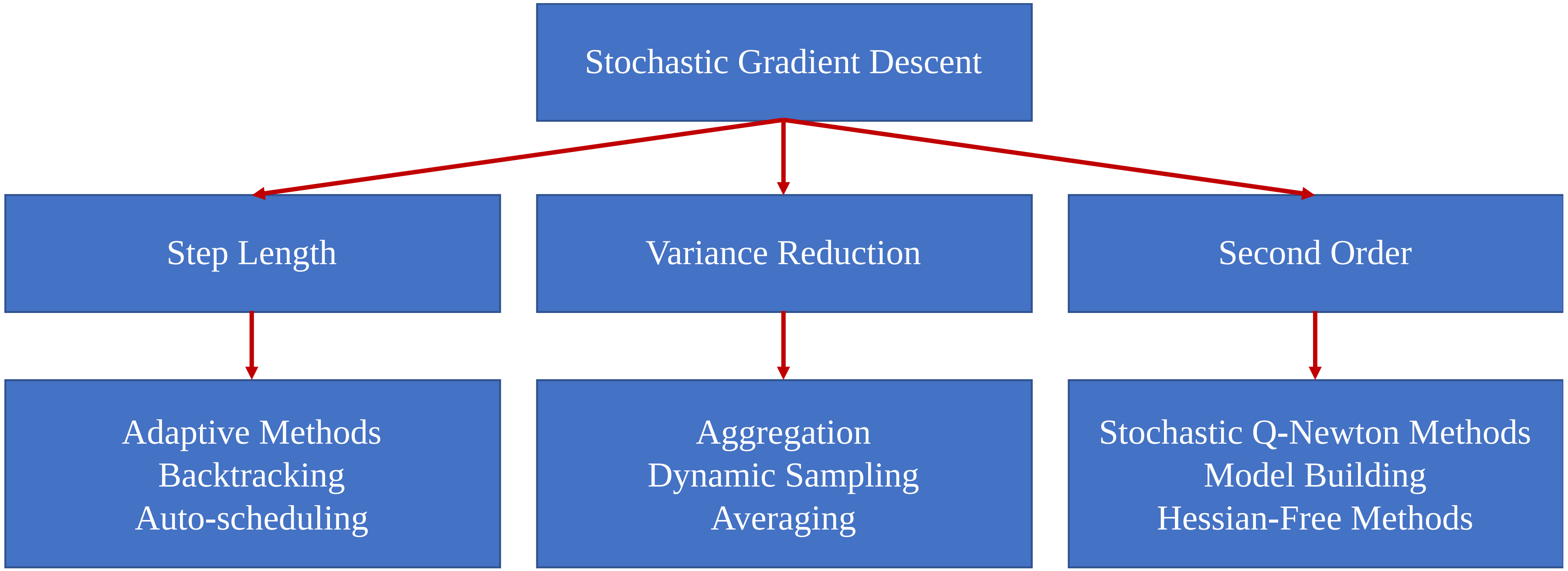
This implies

$$\sum_{k=1}^{T} \alpha_k \mathbb{E}\left[\|\nabla F(w_k)\|_2^2\right] \leq \underbrace{2(F(w_1) - F_*) + L M \sum_{k=1}^{T} T\alpha_k^2}_{<\infty \text{ as } T\to\infty}.$$

The second result also follows since $A_T \to \infty$ as $T \to \infty$. ■

SGD Variants

$$w_{k+1} = w_k - \alpha_k g(w_k, \xi_k)$$



- Generalization
- Work complexity
- Probabilistic results
- …

Optional Readings

Published as a conference paper at ICLR 2017

ON LARGE-BATCH TRAINING FOR DEEP LEARNING: GENERALIZATION GAP AND SHARP MINIMA

Nitish Shirish Keskar*
Northwestern University
Evanston, IL 60208
keskar.nitish@u.northwestern.edu

Dheevatsa Mudigere
Intel Corporation
Bangalore, India
dheevatsa.mudigere@intel.com

Jorge Nocedal
Northwestern University
Evanston, IL 60208
j-nocedal@northwestern.edu

Mikhail Smelyanskiy
Intel Corporation
Santa Clara, CA 95054
mikhail.smelyanskiy@intel.com

Ping Tak Peter Tang
Intel Corporation
Santa Clara, CA 95054
peter.tang@intel.com

Optimization Methods for Large-Scale Machine Learning

Léon Bottou* Frank E. Curtis† Jorge Nocedal‡

June 23, 2018

Network Training

Overfitting

- Cross validation

- **Dropout method:** randomly deleting some nodes and their connections from the network (link)

- Regularization

$$\min_{\beta} E(\beta) + \lambda \sum_{s,t} \beta_{st}^2 \qquad \text{or} \qquad \min_{\beta} E(\beta) + \lambda \sum_{s,t} |\beta_{st}|$$

Vanishing/Exploding Gradients

- Gradient gets very small (large) moving from last layers to earlier layers
- Training takes a long time or simply fails

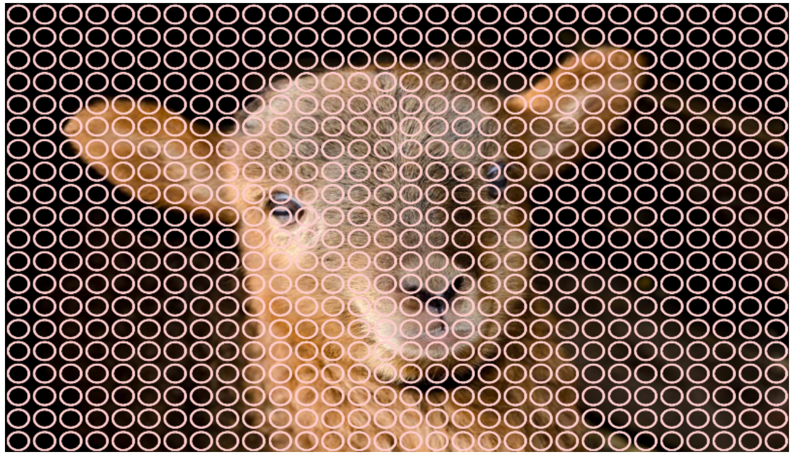
$$\boxed{\sigma(a) = \frac{1}{1+e^{-a}} \implies \max_{a\in\mathbb{R}} \sigma'(a) = \frac{1}{4}}$$

$$\begin{aligned}
\frac{\partial E_i}{\partial \beta_{ts}} &= \delta_t z_s \\
&= \left(\sigma'(a_t) \sum_u \delta_u \beta_{ut}\right) z_s \\
&= \left(\underbrace{\sigma'(a_t)}_{<1} \sum_u \left(\underbrace{\sigma'(a_u)}_{<1} \sum_v \delta_v \beta_{vu}\right) \beta_{ut}\right) \underbrace{z_s}_{\leq 1}
\end{aligned}$$

- Smaller initial weights lead to vanishing gradients
- Larger initial weights lead to explosion of gradients

Convolutional Neural Network (CNN)

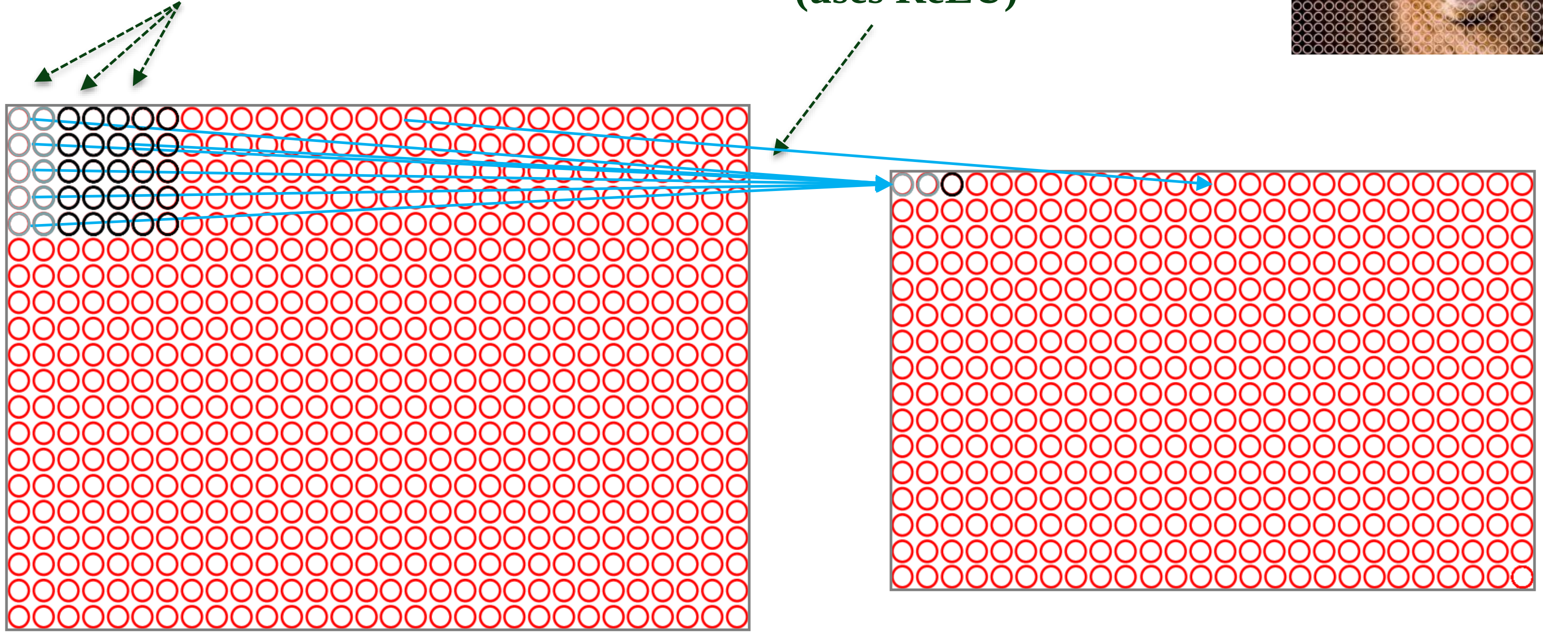
- Carefully constructed multi-layer neural network
- Exploits the spatial structure of data (object recognition)
- First layers learn simple patterns, subsequent layers recombine them
- Hidden layers detect the same local structure
- ReLU is used for activation
- The number of weights and biases decreases significantly



CNN - Construction

local receptive fields (5×5)

feature map (uses ReLU)



stride length (one)

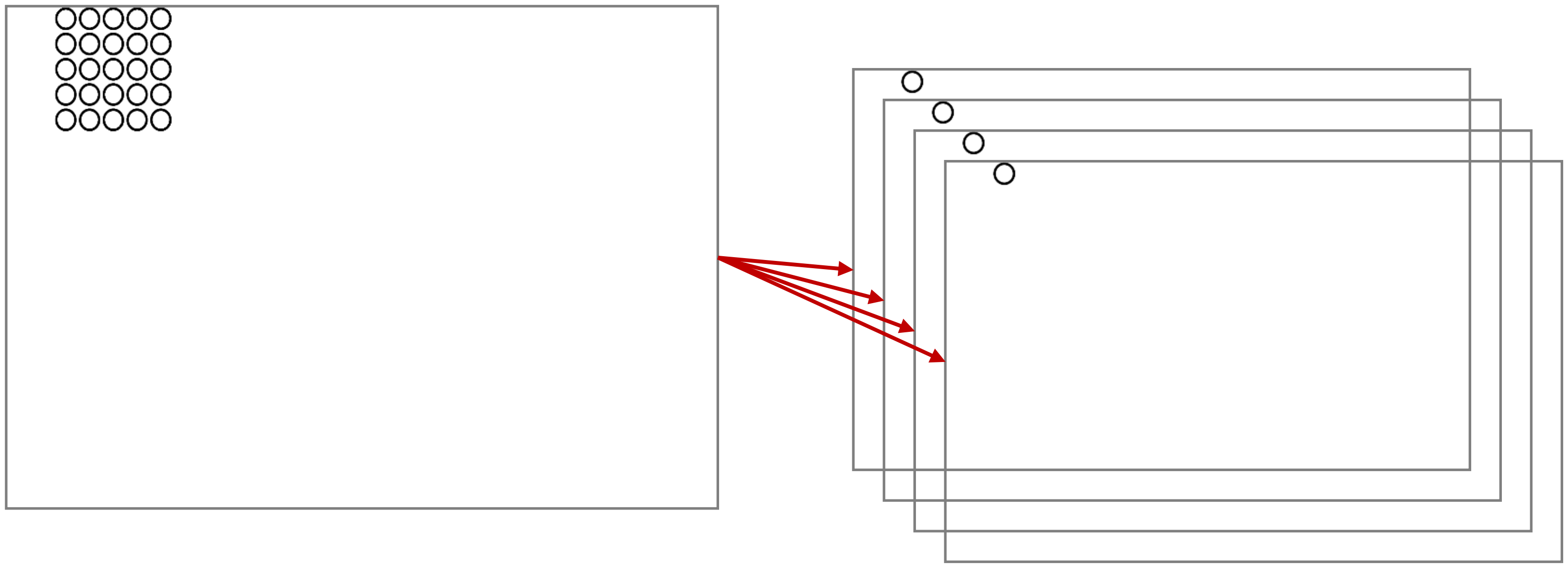
input: 20 × 30 neurons

16 × 26 neurons

weights and bias are shared for each hidden neuron
(shared weights and bias define a **filter** or a **kernel**)

CNN – Hidden Layer

multiple feature maps

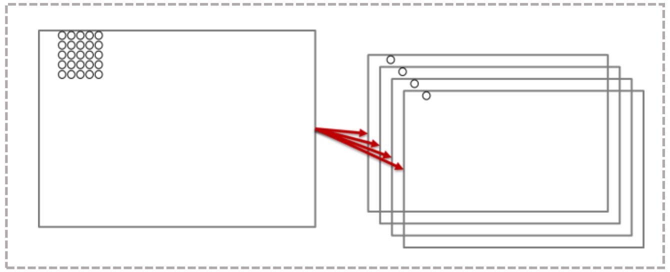


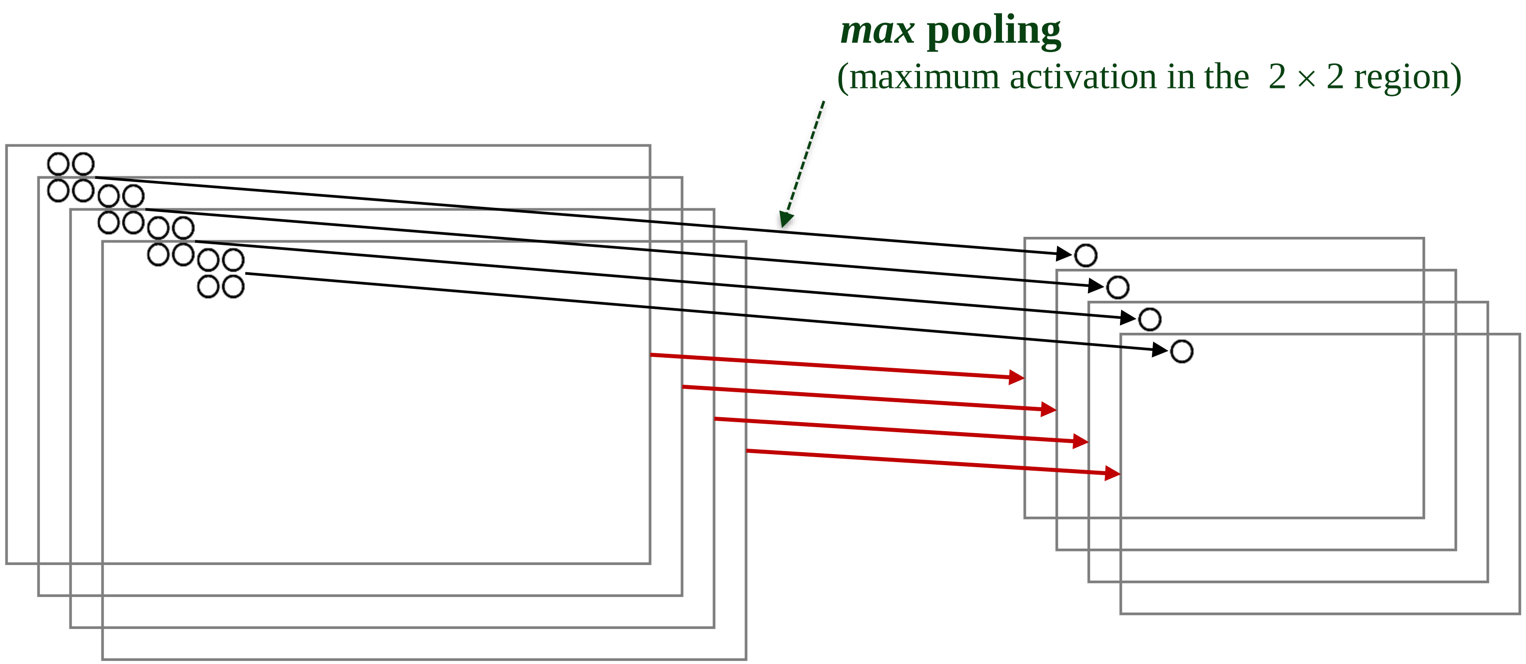
input: 20×30 neurons

first hidden layer: $4 \times 16 \times 26$ neurons
(**convolutional layer**)

number of weights and biases = $(25 + 1) \times 4 = 104$

CNN – Pooling Layer

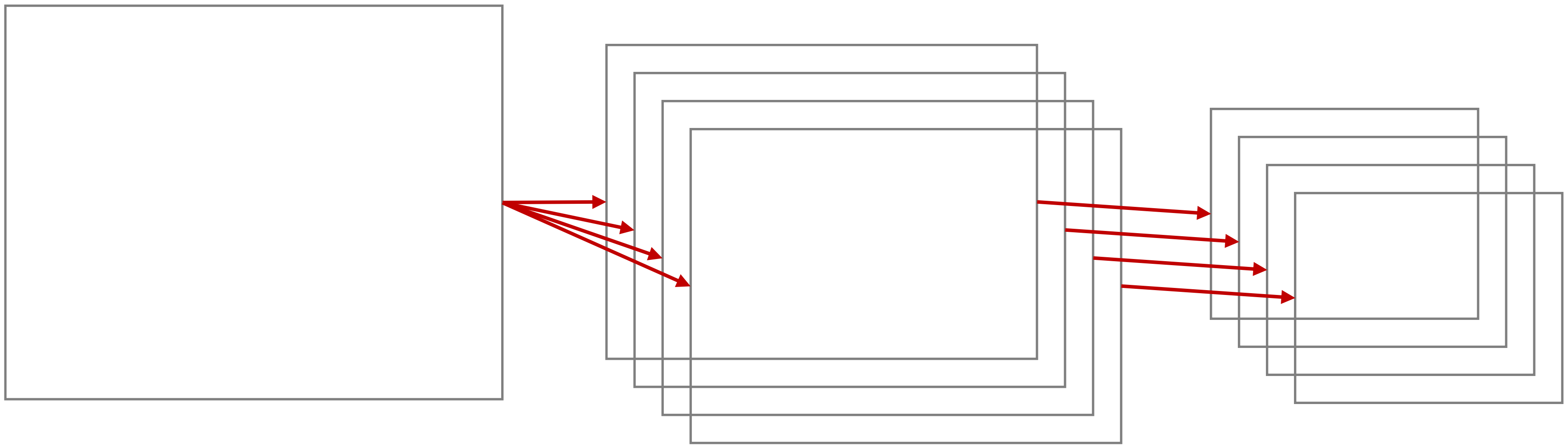




first hidden layer: $4 \times 16 \times 26$ neurons

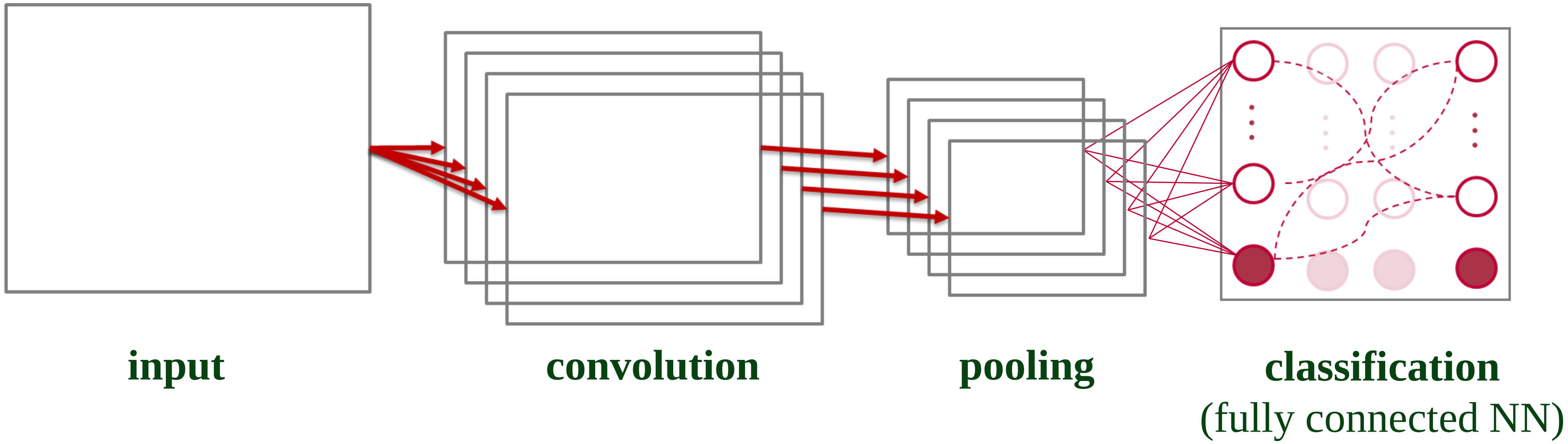
second hidden layer: $4 \times 8 \times 13$ neurons
(**pooling layer**)

CNN – Merging Layers



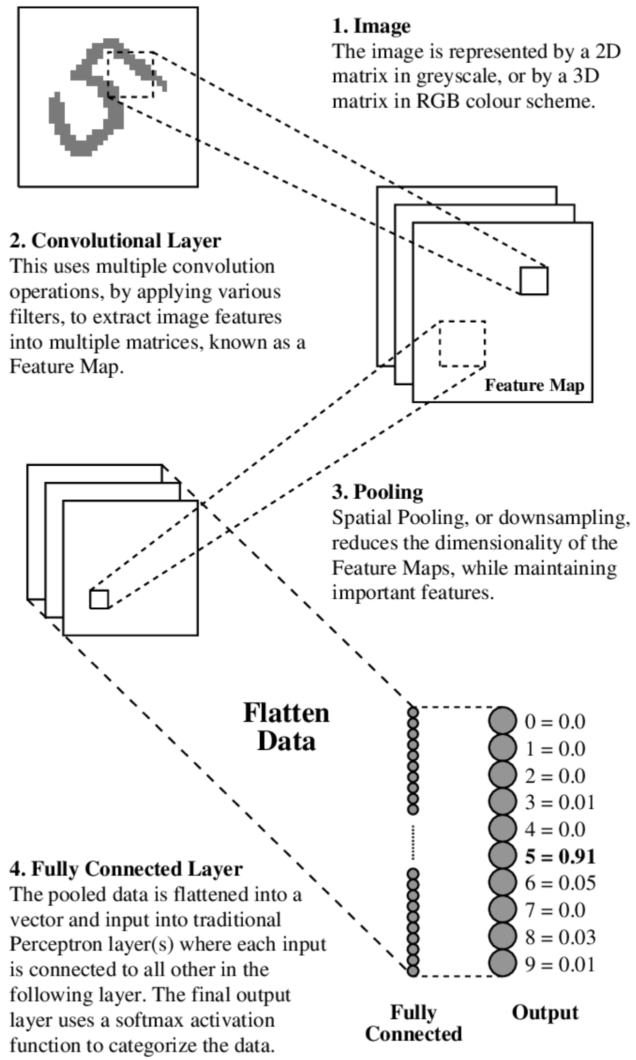
20×30 $\quad$ $4 \times 16 \times 26$ $\quad$ $4 \times 8 \times 13$

CNN – Complete Network



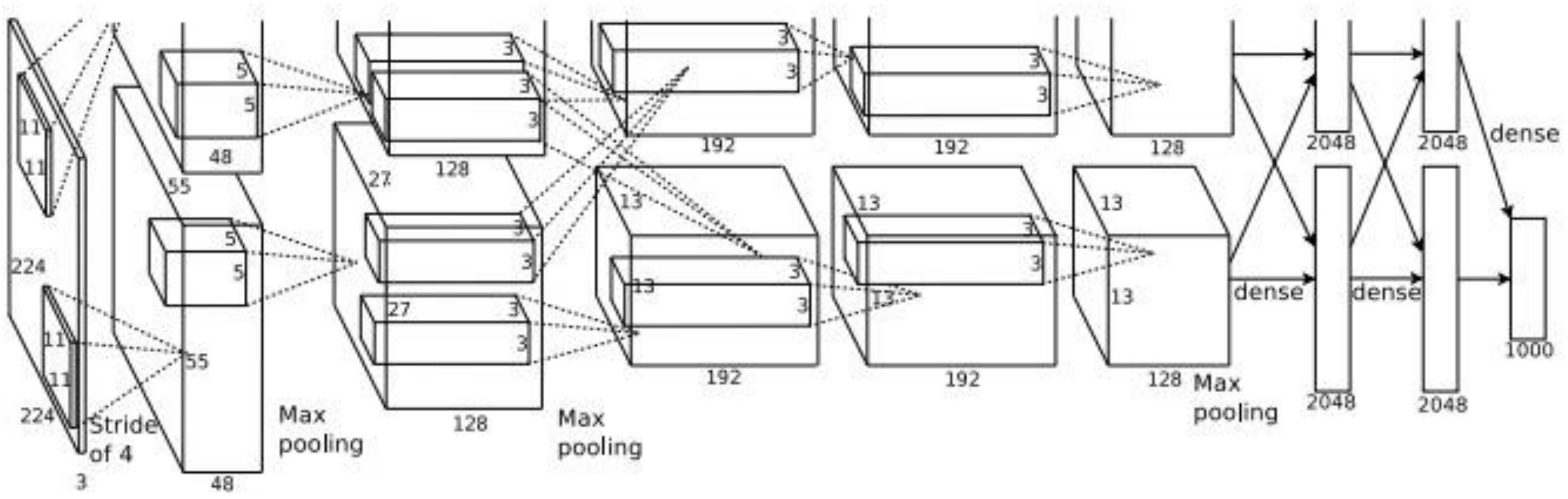
- Common to use one layer for classification in the last stage
- There could be multiple convolution layers
- Too many matrix or vector multiplications are required
- Vectorization along with GPUs are used (for all deep learning networks)

CNN – Object Recognition



(source)

CNN – ImageNet Classification

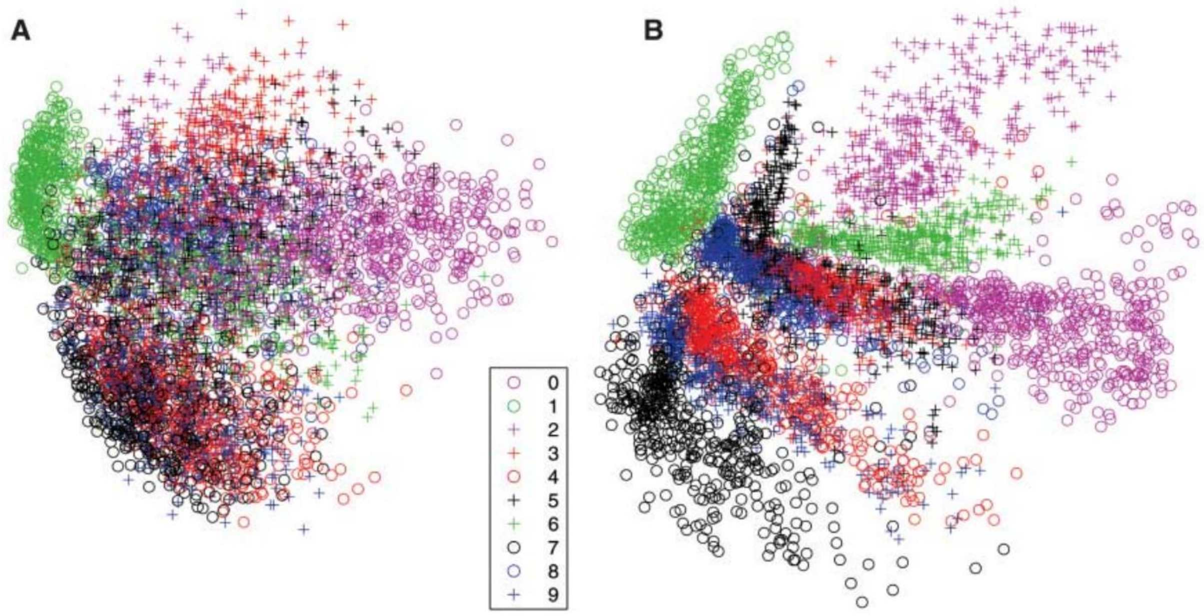


(source)

Autoencoders

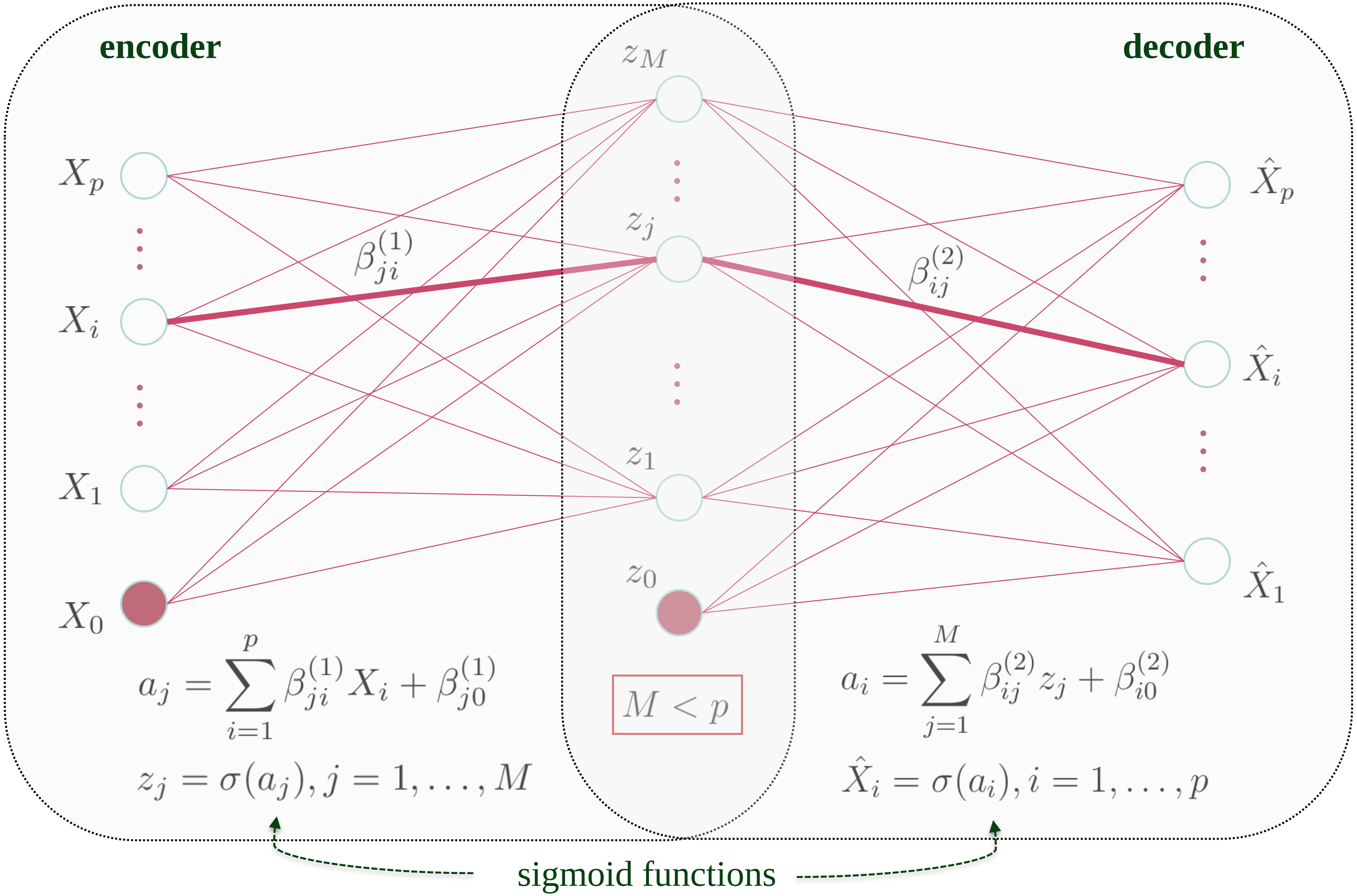
- Unsupervised learning
- Motto: Extract features, reconstruct input
- Dimension reduction technique

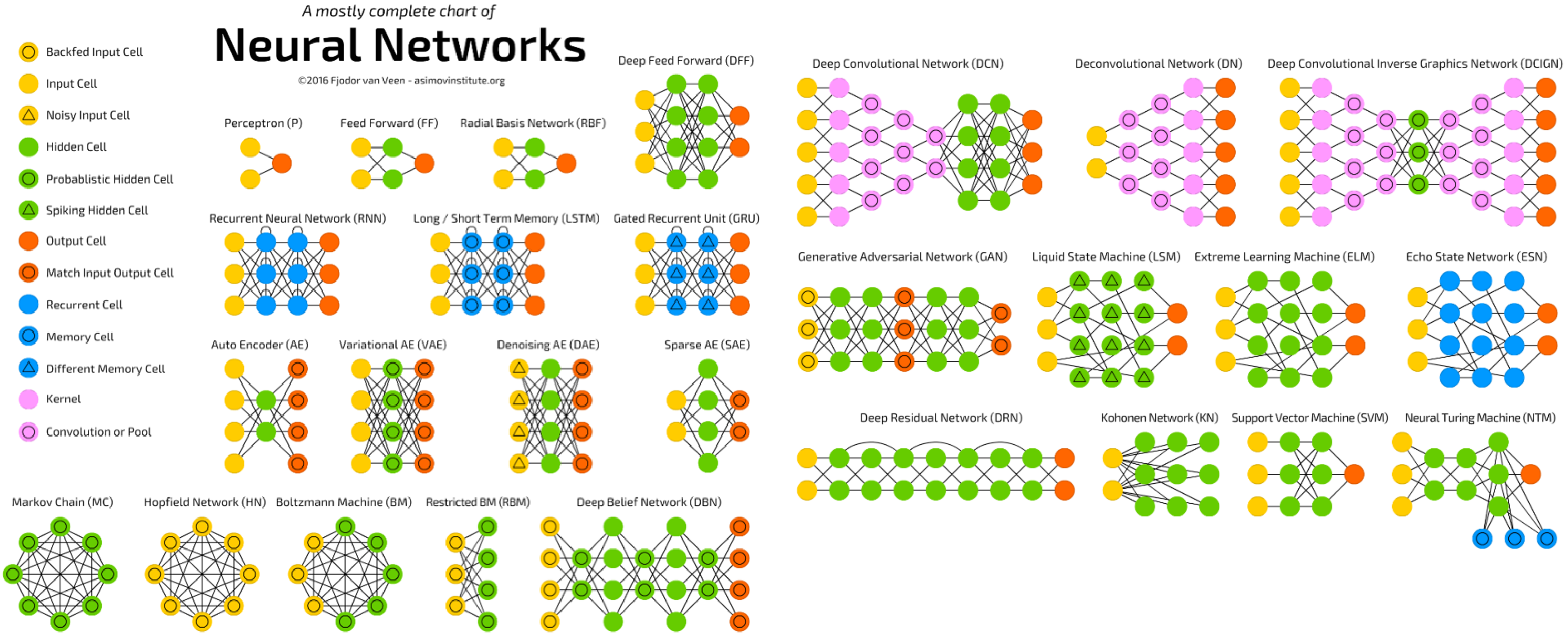
Fig. 3. (**A**) The two-dimensional codes for 500 digits of each class produced by taking the first two principal components of all 60,000 training images. (**B**) The two-dimensional codes found by a 784-1000-500-250-2 autoencoder. For an alternative visualization, see (*8*).



( )

Autoencoders





A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)

(source - 2016)