Computer Vision(I) - Neural Networks Basics

Seven

Research Scientist
www.julyedu.com

Seven@julyedu.com

2018.9.29

Preliminary Background

- Basic calculus, e.g. (partial) derivatives.
- Basic knowledge of machine leanring, e.g. linear classifier, loss functions, overfitting, underfitting.
- Basic optimization algorithm, e.g. SGD.
- Basic level of a programming language, e.g. Python.

In this lecture, you will learn

Preliminary Background

- Basic calculus, e.g. (partial) derivatives.
- Basic knowledge of machine leanring, e.g. linear classifier, loss functions, overfitting, underfitting.
- Basic optimization algorithm, e.g. SGD.
- Basic level of a programming language, e.g. Python.

In this lecture, you will learn

- the framework of a linear classifeir;
- loss function and its gradients;
- neural networks architecture;
- how a neural network outputs a prediction given an input feature vector;
- the big picture and intuition of backpropagation.

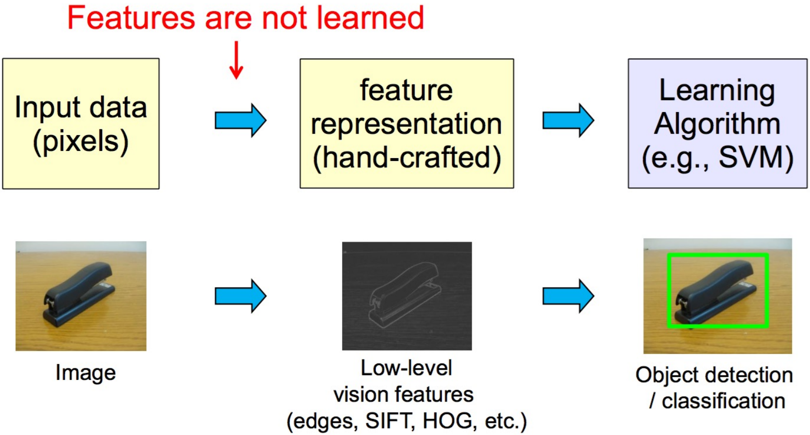
Overview

1. Motivation of Neural Networks and Deep Learning
   - Deep Learning's Duty
   - A Simple Example: Train a Linear Classifier
2. Introduction to Neural Networks(Shallow Learning)
   - Feedforward Propagation
   - The Big Picture of Backpropgation
   - Backpropagation Algorithm
   - Tuning Parameters for Neural Networks

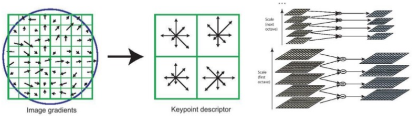
1. Motivation of Neural Networks and Deep Learning
   - Deep Learning's Duty
   - A Simple Example: Train a Linear Classifier

2. Introduction to Neural Networks(Shallow Learning)

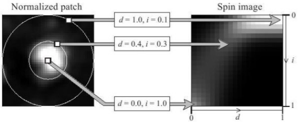
A General Recognition Pipeline



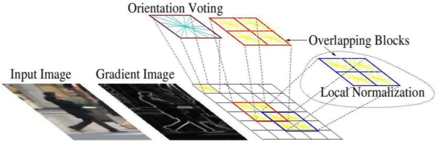
Handcrafted Computer Vision Features



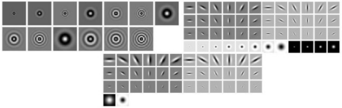
SIFT



Spin image



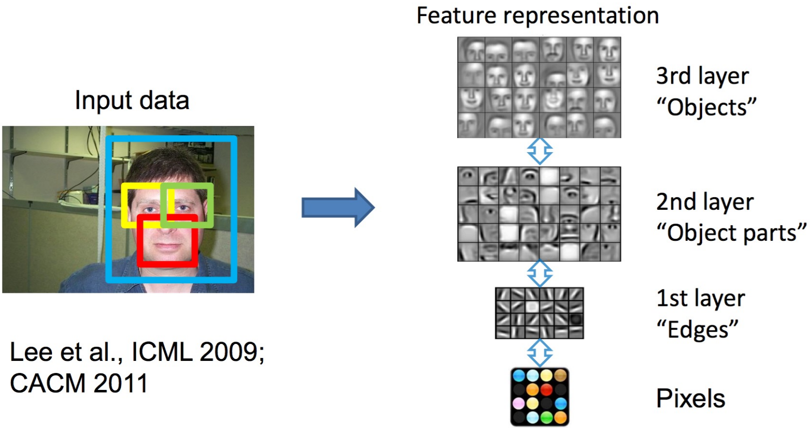
HoG



Textons

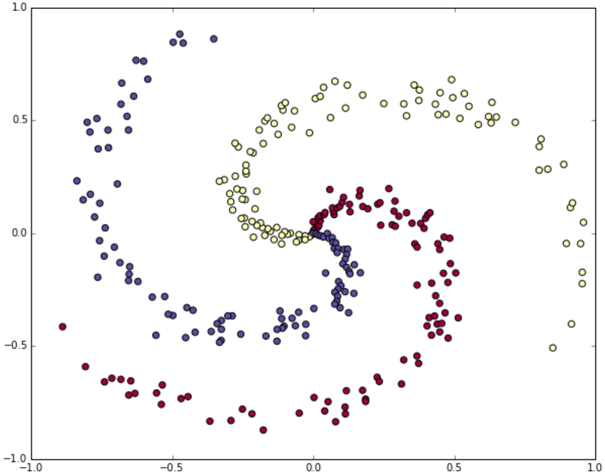
Deep Learning's Duty

Given a large scale of data, automatically learn hierarchical features useful for representation.



1. Motivation of Neural Networks and Deep Learning
   - Deep Learning's Duty
   - A Simple Example: Train a Linear Classifier
2. Introduction to Neural Networks(Shallow Learning)

A Simple Example



Train A Linear Classifier

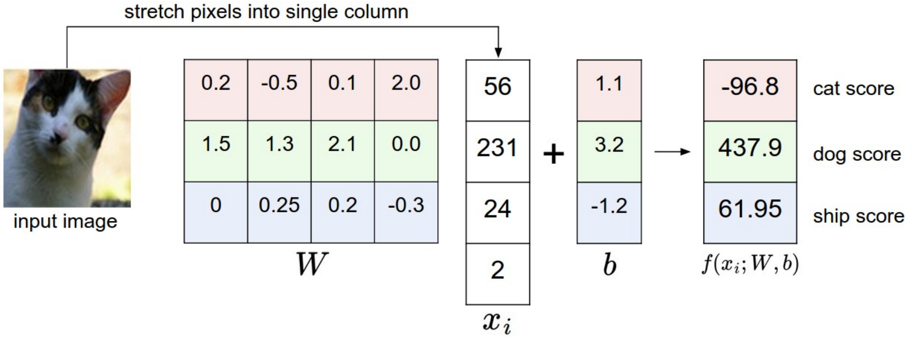
Suppose we have a weights matrix $W \in \mathbb{R}^{K \times D}$, a bias vector $b \in \mathbb{R}^{K \times 1}$ where K is number of classes and D is feature dimensionality, then given an input $x_i \in \mathbb{R}^{D \times 1}$, the score funtion $f : \mathbb{R}^D \rightarrow \mathbb{R}^K$ is:

$$f(x_i, W, b) = Wx_i + b \tag{1}$$

Train A Linear Classifier

Suppose we have a weights matrix $W \in \mathbb{R}^{K \times D}$, a bias vector $b \in \mathbb{R}^{K \times 1}$ where K is number of classes and D is feature dimensionality, then given an input $x_i \in \mathbb{R}^{D \times 1}$, the score funtion $f : \mathbb{R}^D \to \mathbb{R}^K$ is:

$$f(x_i, W, b) = Wx_i + b \tag{1}$$



Softmax

Often, we would like to obtain a probability distribution over all class lables. Thus, we map the result of score function f into an interval between 0 and 1 via **Softmax** funciton.

$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}} \tag{2}$$

Compute the Loss

Cross-Entropy Loss

Let (x_i, y_i) be a pair of training example, where $x_i \in \mathbb{R}^D$ is training data and $y_i \in \mathbb{R}^K$ is a one-hot vector with all zeros except for its true class index is one. Assume the prediction $\hat{y}_i \in \mathbb{R}^K$ is computed after Softmax, then the cross-entropy loss is:

$$L_i = -y_i \cdot log(\hat{y}_i) \tag{3}$$

Compute the Loss

Cross-Entropy Loss

Let (x_i, y_i) be a pair of training example, where $x_i \in \mathbb{R}^D$ is training data and $y_i \in \mathbb{R}^K$ is a one-hot vector with all zeros except for its true class index is one. Assume the prediction $\hat{y}_i \in \mathbb{R}^K$ is computed after Softmax, then the cross-entropy loss is:

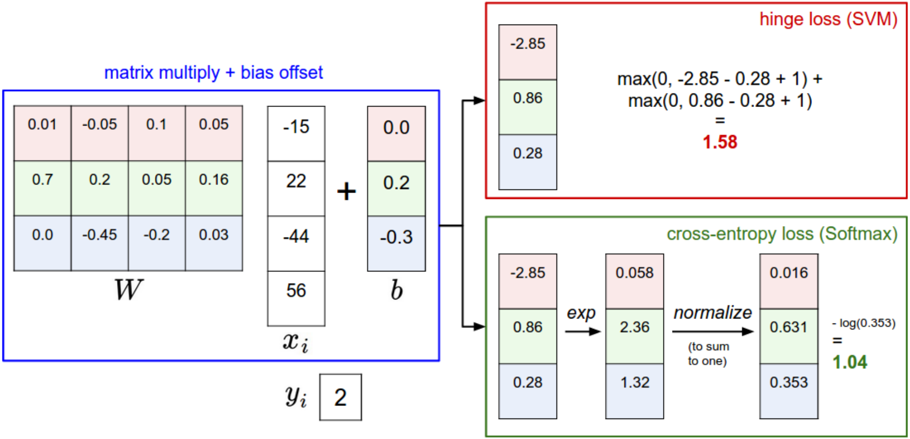
$$L_i = -y_i \cdot log(\hat{y}_i) \tag{3}$$

Hinge Loss

Note that the exponential computing in cross-entropy loss is expensive and sometimes we do not need a probability. Thus, the hinge loss function is:

$$L_i = \sum_{j \neq y_i} \max(0, f(x_i, W)_j - f(x_i, W)_{y_i} + \Delta) \tag{4}$$

Cross-Entropy vs. Hinge



Regularization

L2 Norm

Regularization is a common technique to prevent model learning from overfitting. The most common regularization penalty is **L2** norm which discourages large weights through an elementwise quadratic penalty over all parameters:

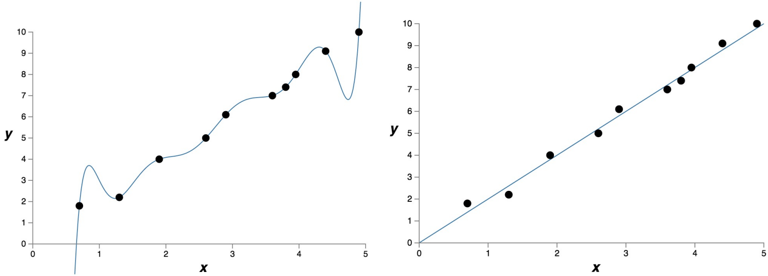
$$R(W) = \sum_k \sum_d W_{k,d}^2 \tag{5}$$

Regularization

L2 Norm

Regularization is a common technique to prevent model learning from overfitting. The most common regularization penalty is **L2** norm which discourages large weights through an elementwise quadratic penalty over all parameters:

$$R(W) = \sum_k \sum_d W_{k,d}^2 \tag{5}$$



Overall Loss

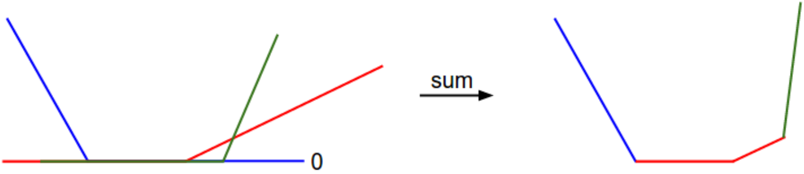
The overall loss consists of two items, namely data loss and regularization loss.

$$L = \frac{1}{N} \sum_i L_i + \lambda R(W) \tag{6}$$

Overall Loss

The overall loss consists of two items, namely data loss and regularization loss.

$$L = \frac{1}{N}\sum_i L_i + \lambda R(W) \tag{6}$$



Compute the Gradient

For a single data point, the hinge loss is :

$$L_i = \sum_{j \neq y_i} [\max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta)] \tag{7}$$

We can differentiate the loss function with respect to the weights. Taking the gradient with respect to w_{y_i} we obtain:

$$\frac{\partial L_i}{\partial w_{y_i}} =$$

Compute the Gradient

For a single data point, the hinge loss is :

$$L_i = \sum_{j \neq y_i} [\max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta)] \tag{7}$$

We can differentiate the loss function with respect to the weights. Taking the gradient with respect to w_{y_i} we obtain:

$$\frac{\partial L_i}{\partial w_{y_i}} = -[\sum_{j \neq y_i} \mathbb{1}(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0)] x_i \tag{8}$$

For other rows where $j \neq y_i$ the gradient is:

$$\frac{\partial L_i}{\partial w_j} =$$

Compute the Gradient

For a single data point, the hinge loss is :

$$L_i = \sum_{j \neq y_i} [\max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta)] \tag{7}$$

We can differentiate the loss function with respect to the weights. Taking the gradient with respect to w_{y_i} we obtain:

$$\frac{\partial L_i}{\partial w_{y_i}} = -[\sum_{j \neq y_i} \mathbb{1}(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0)] x_i \tag{8}$$

For other rows where $j \neq y_i$ the gradient is:

$$\frac{\partial L_i}{\partial w_j} = \mathbb{1}(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0) x_i \tag{9}$$

Optimization

Example (Vanilla Gradient Decent)

```
while True:
  weights_grad = gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad
```

Optimization

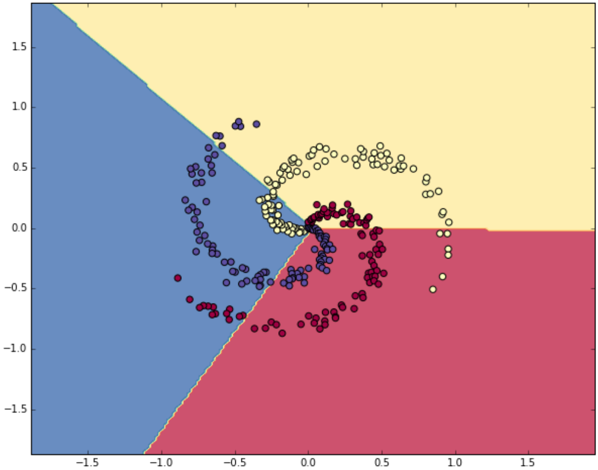
Example (Vanilla Gradient Decent)

```
while True:
  weights_grad = gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad
```

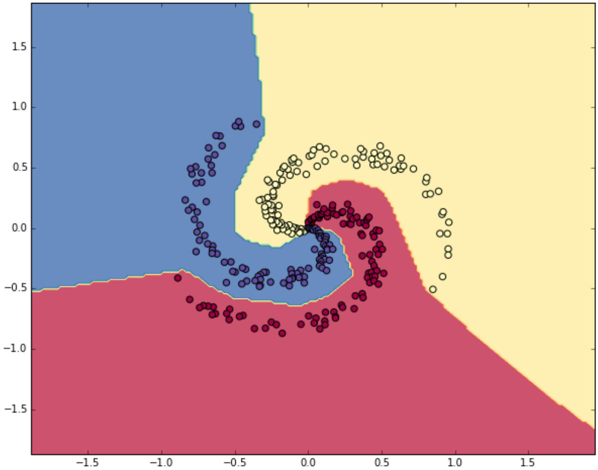
Example (Mini-batch Gradient Decent)

```
while True:
  data_batch = sample_training_data(data, 256)
  weights_grad = gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad
```

Linear Classifier: 49%



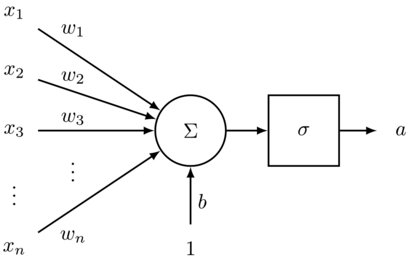
Neural Network: 98%

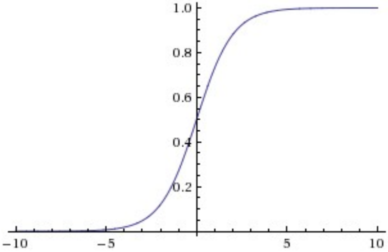


1. Motivation of Neural Networks and Deep Learning
2. Introduction to Neural Networks(Shallow Learning)
   - Feedforward Propagation
   - The Big Picture of Backpropgation
   - Backpropagation Algorithm
   - Tuning Parameters for Neural Networks

A Single Neuron

The input vector x is first scaled, summed, added to a bias unit, and then passed to the squashing sigmoid function.

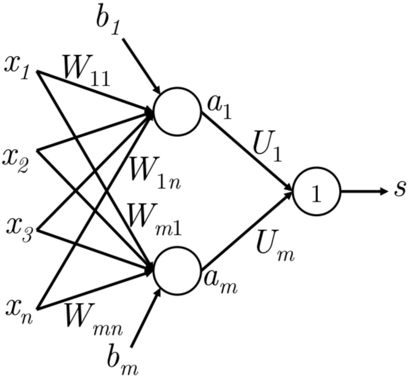




$$a = \frac{1}{1 + \exp(w_1x_1 + w_2x_2 + ... + w_nx_n + b)}$$

A Single Hidden Layer Neural Network

A neural network is nothing but a stack of single neurons. One can think of activations as indicators of the presence of some weighted combination of features. We can then use a combination of these activations to perform classification tasks.



$$z = Wx + b$$
$$a = \sigma(z)$$
$$s = U^T a$$

Objective Funciton

Let's consider hinge loss as our objective function. If we call the score computed for "true" labeled data as s and the score computed for "false" labeled data as s_c. Then the optimization objective is:

$$\text{minimize } J = \max\left(\Delta + s_c - s, 0\right)$$

where $s_c =$

Objective Funciton

Let's consider hinge loss as our objective function. If we call the score computed for "true" labeled data as s and the score computed for "false" labeled data as s_c. Then the optimization objective is:

$$\text{minimize } J = \max\left(\Delta + s_c - s, 0\right)$$

where $s_c = U^T\sigma(Wx_c + b)$ and $s =$

Objective Funciton

Let's consider hinge loss as our objective function. If we call the score computed for "true" labeled data as s and the score computed for "false" labeled data as s_c. Then the optimization objective is:

$$\text{minimize } J = \max(\varDelta + s_c - s, 0)$$

where $s_c = U^T \sigma(Wx_c + b)$ and $s = U^T \sigma(Wx + b)$.

Objective Funciton

Let's consider hinge loss as our objective function. If we call the score computed for "true" labeled data as s and the score computed for "false" labeled data as s_c. Then the optimization objective is:

$$\text{minimize } J = \max\left(\Delta + s_c - s, 0\right)$$

where $s_c = U^T \sigma(W x_c + b)$ and $s = U^T \sigma(W x + b)$.

Trick

We can scale this margin such that it is $\Delta = 1$ and let other parameters in the optimization problem adapt to this without any change in performance.

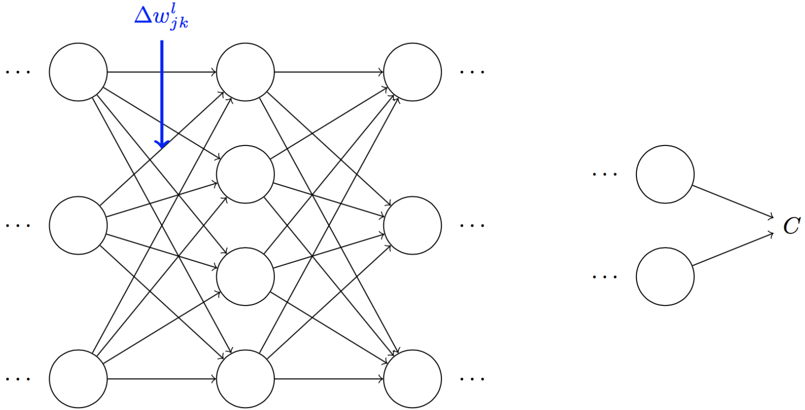
$$\text{minimize } J = \max\left(1 + s_c - s, 0\right) \tag{10}$$

1 Motivation of Neural Networks and Deep Learning

2 Introduction to Neural Networks(Shallow Learning)
- Feedforward Propagation
- The Big Picture of Backpropgation
- Backpropagation Algorithm
- Tuning Parameters for Neural Networks

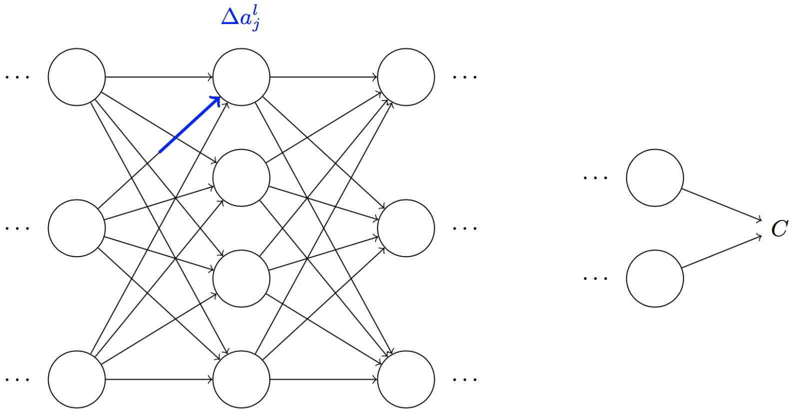
Backpropgation Intuition 1

Let's imagine that we have made a small change Δw_{jk}^{l} to some weight in the network, w_{jk}^{l}:



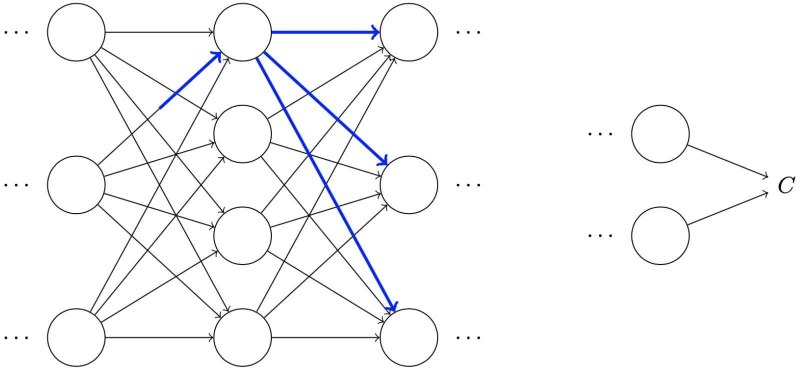
Backpropgation Intuition 2

That change in weight will cause a change in the output activation from the corresponding neuron, Δa_j^l:



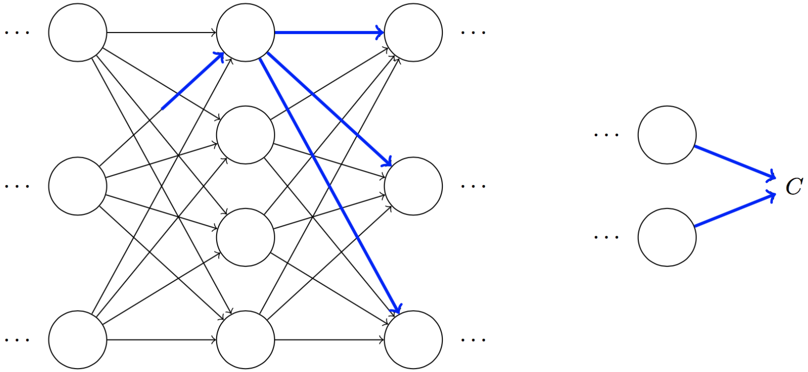
Backpropgation Intuition 3

That, in turn, will cause a change in *all* the activations in the next layer:



Backpropgation Intuition 4

Those changes will in turn cause changes in the next layer, and then next, and so on. Finally it will reach in the final layer and cause changes in the cost function:



Relationship between ΔC and Δw^l_{jk}

The change ΔC in the cost is related to the change Δw^l_{jk} in the weight by the equation:

$$\Delta C \approx \frac{\partial C}{\partial w^l_{jk}} \Delta w^l_{jk} \tag{11}$$

Relationship between ΔC and Δw^l_{jk}

The change ΔC in the cost is related to the change Δw^l_{jk} in the weight by the equation:

$$\Delta C \approx \frac{\partial C}{\partial w^l_{jk}} \Delta w^l_{jk} \tag{11}$$

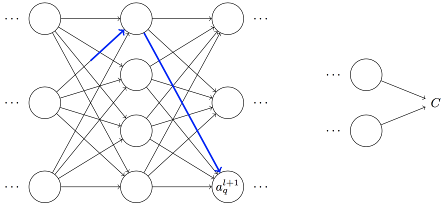
The change Δw^l_{jk} causes a samll change Δa^l_j in the activation layer:

$$\Delta a^l_j \approx \frac{\partial a^l_j}{\partial w^l_{jk}} \Delta w^l_{jk} \tag{12}$$

Relationship between Δa_q^{l+1} and Δa_j^l

The change in activation Δa_j^l will cause changes in all the activations in the next layer:

$$\Delta a_q^{l+1} \approx \frac{\partial a_q^{l+1}}{\partial a_j^l} \Delta a_j^l \tag{13}$$



Substituting in the expression from last equation, we get:

$$\Delta a_q^{l+1} \approx \frac{\partial a_q^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial w_{jk}^l} \Delta w_{jk}^l \tag{14}$$

A Path from w_{jk}^l to C

If the path goes through activations $a_j^l, a_q^{l+1}, ..., a_n^{L-1}, a_m^L$ then the resulting expression is:

$$\Delta C \approx \frac{\partial C}{\partial a_m^L} \frac{\partial a_m^L}{\partial a_n^{L-1}} \frac{\partial a_n^{L-1}}{\partial a_p^{L-2}} \cdots \frac{\partial a_q^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial w_{jk}^l} \Delta w_{jk}^l \tag{15}$$

We should sum over all the possible paths between the weight and the cost:

$$\Delta C \approx$$

A Path from w_{jk}^l to C

If the path goes through activations $a_j^l, a_q^{l+1}, ..., a_n^{L-1}, a_m^L$ then the resulting expression is:

$$\Delta C \approx \frac{\partial C}{\partial a_m^L} \frac{\partial a_m^L}{\partial a_n^{L-1}} \frac{\partial a_n^{L-1}}{\partial a_p^{L-2}} \cdots \frac{\partial a_q^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial w_{jk}^l} \Delta w_{jk}^l \tag{15}$$

We should sum over all the possible paths between the weight and the cost:

$$\Delta C \approx \sum_{mnp...q} \frac{\partial C}{\partial a_m^L} \frac{\partial a_m^L}{\partial a_n^{L-1}} \frac{\partial a_n^{L-1}}{\partial a_p^{L-2}} \cdots \frac{\partial a_q^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial w_{jk}^l} \Delta w_{jk}^l \tag{16}$$

$$\frac{\partial C}{\partial w_{jk}^l} \approx$$

A Path from w^l_{jk} to C

If the path goes through activations $a^l_j, a^{l+1}_q, ..., a^{L-1}_n, a^L_m$ then the resulting expression is:

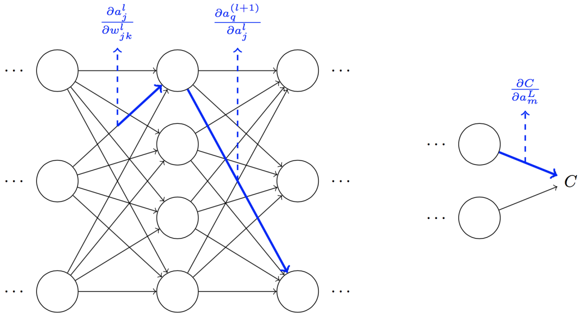
$$\Delta C \approx \frac{\partial C}{\partial a^L_m} \frac{\partial a^L_m}{\partial a^{L-1}_n} \frac{\partial a^{L-1}_n}{\partial a^{L-2}_p} \cdots \frac{\partial a^{l+1}_q}{\partial a^l_j} \frac{\partial a^l_j}{\partial w^l_{jk}} \Delta w^l_{jk} \tag{15}$$

We should sum over all the possible paths between the weight and the cost:

$$\Delta C \approx \sum_{mnp...q} \frac{\partial C}{\partial a^L_m} \frac{\partial a^L_m}{\partial a^{L-1}_n} \frac{\partial a^{L-1}_n}{\partial a^{L-2}_p} \cdots \frac{\partial a^{l+1}_q}{\partial a^l_j} \frac{\partial a^l_j}{\partial w^l_{jk}} \Delta w^l_{jk} \tag{16}$$

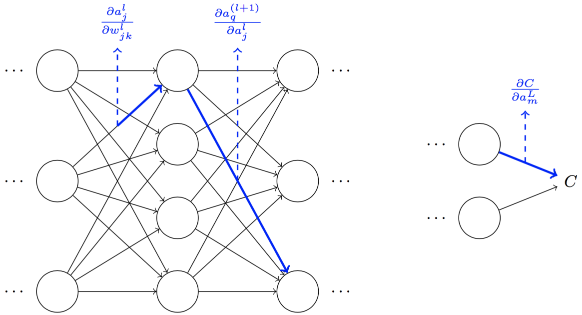
$$\frac{\partial C}{\partial w^l_{jk}} \approx \sum_{mnp...q} \frac{\partial C}{\partial a^L_m} \frac{\partial a^L_m}{\partial a^{L-1}_n} \frac{\partial a^{L-1}_n}{\partial a^{L-2}_p} \cdots \frac{\partial a^{l+1}_q}{\partial a^l_j} \frac{\partial a^l_j}{\partial w^l_{jk}} \tag{17}$$

Summary of The Big Picture of Backpropgation



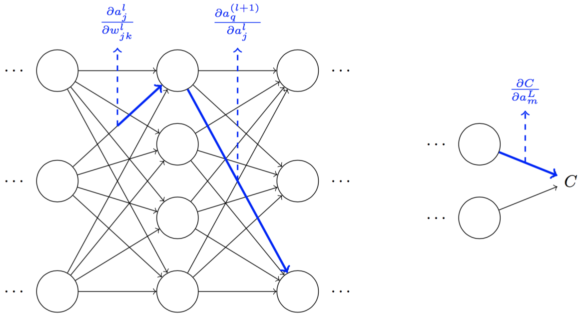
(1)

Summary of The Big Picture of Backpropgation



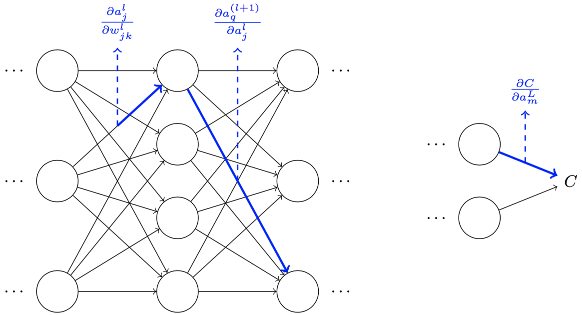
(1) Every edge between two neurons in the network is associated with a rate factor which is just the partial derivative of one neuron's activation with respect to the other; (2)

Summary of The Big Picture of Backpropgation



(1) Every edge between two neurons in the network is associated with a rate factor which is just the partial derivative of one neuron's activation with respect to the other; (2) The rate factor for a path is just the product of the rate factors along the path; (3)

Summary of The Big Picture of Backpropgation

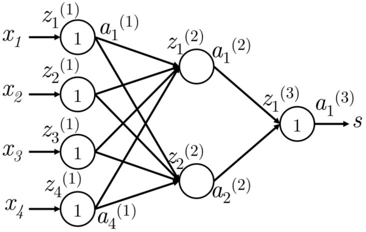


(1) Every edge between two neurons in the network is associated with a rate factor which is just the partial derivative of one neuron's activation with respect to the other; (2) The rate factor for a path is just the product of the rate factors along the path; (3) And the total rate of change $\frac{\partial C}{\partial w_{jk}^l}$ is just the sum of the rate factors of all possible paths.

1 Motivation of Neural Networks and Deep Learning

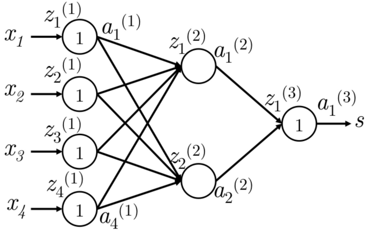
2 Introduction to Neural Networks(Shallow Learning)
- Feedforward Propagation
- The Big Picture of Backpropgation
- Backpropagation Algorithm
- Tuning Parameters for Neural Networks

Backpropagation Notation



- x_i is an input to the neural network.
- s is the output of the neural network.
- The j-th neuron of layer k receives the scalar input $z_j^{(k)}$ and produces the scalar activation output $a_j^{(k)}$.
- For the input layer, $x_j = z_j^{(1)} = a_j^{(1)}$.
- $W^{(k)}$ is the transfer/weights matrix that maps the output from the k-th layer to the input to the $(k+1)$-th.
- **$\delta_j^{(k)}$ is the backpropagated error calculated at $z_j^{(k)}$: $\delta_j^{(k)} = \frac{\partial J}{\partial z_j^{(k)}}$.**

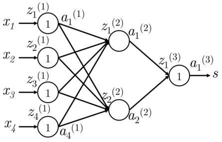
Compute $\frac{\partial J}{\partial W_{14}^{(1)}}$



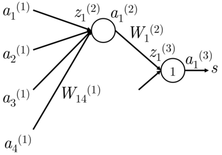
Suppose the cost $J = (1 + s_c - s)$ is positive and we want to perform the update of parameter $W_{14}^{(1)}$, we must realize that $W_{14}^{(1)}$ only contributes to $z_1^{(2)}$ and thus $a_1^{(2)}$. **Backpropagated gradients are only affected by values they contribute to.**

$$\frac{\partial J}{\partial s} = -1$$

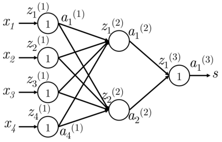
Compute $\frac{\partial J}{\partial W_{14}^{(1)}}$



$$\frac{\partial s}{\partial W_{14}^{(1)}} =$$

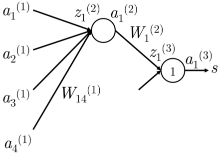


Compute $\frac{\partial J}{\partial W_{14}^{(1)}}$

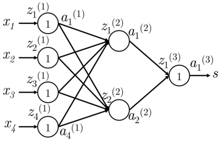


$$\frac{\partial s}{\partial W_{14}^{(1)}} = \frac{\partial W^{(2)} a^{(2)}}{\partial W_{14}^{(1)}} = \frac{\partial W_1^{(2)} a_1^{(2)}}{\partial W_{14}^{(1)}} = W_1^{(2)} \frac{\partial a_1^{(2)}}{\partial W_{14}^{(1)}}$$

$$\Rightarrow W_1^{(2)} \frac{\partial a_1^{(2)}}{\partial W_{14}^{(1)}} =$$



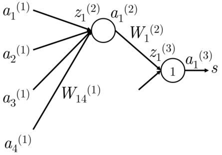
Compute $\frac{\partial J}{\partial W_{14}^{(1)}}$

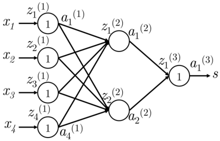


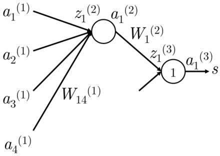
$$\frac{\partial s}{\partial W_{14}^{(1)}} = \frac{\partial W^{(2)} a^{(2)}}{\partial W_{14}^{(1)}} = \frac{\partial W_1^{(2)} a_1^{(2)}}{\partial W_{14}^{(1)}} = W_1^{(2)} \frac{\partial a_1^{(2)}}{\partial W_{14}^{(1)}}$$

$$\Rightarrow W_1^{(2)} \frac{\partial a_1^{(2)}}{\partial W_{14}^{(1)}} = W_1^{(2)} \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial W_{14}^{(1)}}$$

$$=$$



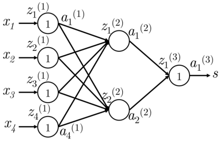




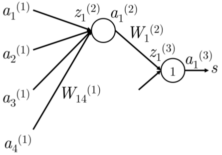
$$\frac{\partial s}{\partial W_{14}^{(1)}} = \frac{\partial W^{(2)} a^{(2)}}{\partial W_{14}^{(1)}} = \frac{\partial W_1^{(2)} a_1^{(2)}}{\partial W_{14}^{(1)}} = W_1^{(2)} \frac{\partial a_1^{(2)}}{\partial W_{14}^{(1)}}$$

$$\begin{aligned}
\Rightarrow W_1^{(2)} \frac{\partial a_1^{(2)}}{\partial W_{14}^{(1)}} &= W_1^{(2)} \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial W_{14}^{(1)}} \\
&= W_1^{(2)} \sigma'(z_1^{(2)}) \frac{\partial}{\partial W_{14}^{(1)}} (b_1^{(1)} + \sum_k a_k^{(1)} W_{1k}^{(1)}) \\
&=
\end{aligned}$$

Compute $\frac{\partial J}{\partial W_{14}^{(1)}}$



$$\frac{\partial s}{\partial W_{14}^{(1)}} = \frac{\partial W^{(2)} a^{(2)}}{\partial W_{14}^{(1)}} = \frac{\partial W_1^{(2)} a_1^{(2)}}{\partial W_{14}^{(1)}} = W_1^{(2)} \frac{\partial a_1^{(2)}}{\partial W_{14}^{(1)}}$$



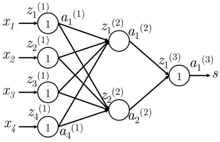
$$\Rightarrow W_1^{(2)} \frac{\partial a_1^{(2)}}{\partial W_{14}^{(1)}} = W_1^{(2)} \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial W_{14}^{(1)}}$$

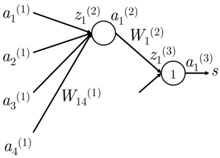
$$= W_1^{(2)} \sigma'(z_1^{(2)}) \frac{\partial}{\partial W_{14}^{(1)}} (b_1^{(1)} + \sum_k a_k^{(1)} W_{1k}^{(1)})$$

$$= W_1^{(2)} \sigma'(z_1^{(2)}) a_4^{(1)}$$

$$=$$

Compute $\frac{\partial J}{\partial W_{14}^{(1)}}$

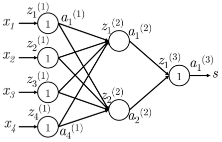


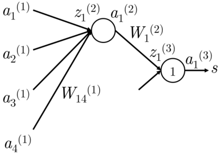


$$\frac{\partial s}{\partial W_{14}^{(1)}} = \frac{\partial W^{(2)} a^{(2)}}{\partial W_{14}^{(1)}} = \frac{\partial W_1^{(2)} a_1^{(2)}}{\partial W_{14}^{(1)}} = W_1^{(2)} \frac{\partial a_1^{(2)}}{\partial W_{14}^{(1)}}$$

$$\begin{aligned}
\Rightarrow W_1^{(2)} \frac{\partial a_1^{(2)}}{\partial W_{14}^{(1)}} &= W_1^{(2)} \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial W_{14}^{(1)}} \\
&= W_1^{(2)} \sigma'(z_1^{(2)}) \frac{\partial}{\partial W_{14}^{(1)}} (b_1^{(1)} + \sum_k a_k^{(1)} W_{1k}^{(1)}) \\
&= W_1^{(2)} \sigma'(z_1^{(2)}) a_4^{(1)} \\
&= \frac{\partial J}{\partial z_1^{(2)}} a_4^{(1)} \\
&=
\end{aligned}$$

Compute $\frac{\partial J}{\partial W_{14}^{(1)}}$

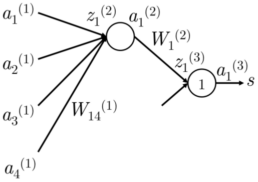




$$\frac{\partial s}{\partial W_{14}^{(1)}} = \frac{\partial W^{(2)} a^{(2)}}{\partial W_{14}^{(1)}} = \frac{\partial W_1^{(2)} a_1^{(2)}}{\partial W_{14}^{(1)}} = W_1^{(2)} \frac{\partial a_1^{(2)}}{\partial W_{14}^{(1)}}$$

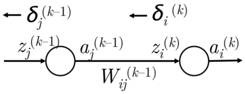
$$\begin{aligned}
\Rightarrow W_1^{(2)} \frac{\partial a_1^{(2)}}{\partial W_{14}^{(1)}} &= W_1^{(2)} \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial W_{14}^{(1)}} \\
&= W_1^{(2)} \sigma'(z_1^{(2)}) \frac{\partial}{\partial W_{14}^{(1)}} (b_1^{(1)} + \sum_k a_k^{(1)} W_{1k}^{(1)}) \\
&= W_1^{(2)} \sigma'(z_1^{(2)}) a_4^{(1)} \\
&= \frac{\partial J}{\partial z_1^{(2)}} a_4^{(1)} \\
&= \delta_1^{(2)} a_4^{(1)}
\end{aligned}$$

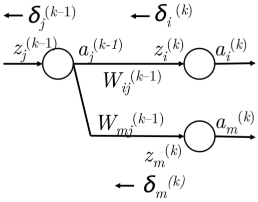
Error Distribution Interpretation of Backpropagation



1. We start with an error signal of 1 propagating backwards from $a_1^{(3)}$.
2. We then multiply this error by the local gradient of the neuron which maps $z_1^{(3)}$ to $a_1^{(3)}$. This happens to be 1 in this case and thus, the error is still 1. This is now known as $\delta_1^{(3)}$.
3. Then the error reaches to $a_1^{(2)}$ is the error at $z_1^{(3)}$ mutiplies $W_1^{(2)}$, which is $\delta_1^{(3)} W_1^{(2)} = W_1^{(2)}$.
4. As we did in step 2, we need to move the error across the neuron which maps $z_1^{(2)}$ to $a_1^{(2)}$. We do this by multiplying the error signal at $a_1^{(2)}$ by the local gradient of the neuron which happens to be $\sigma'(z_1^{(2)})$.
5. The error signal at $z_1^{(2)}$ is $W_1^{(2)} \sigma'(z_1^{(2)})$, which is known to be $\delta_1^{(2)}$.
6. Finally we need to distribute the "fair share" of the error to $W_{14}^{(1)}$ by simply multiplying it by the input it was responsible for forwarding, which happens to be $a_4^{(1)}$.
7. Thus, the gradient of loss with respect to $W_{14}^{(1)}$ is calculated to be $W_1^{(2)} \sigma'(z_1^{(2)}) a_4^{(1)}$.

Backpropagate $\delta^{(k)}$ to $\delta^{(k-1)}$





1. We have error $\delta_i^{(k)}$ propagating backwards from $z_j^{(k)}$, i.e. neuron i at layer k.
2. We propagate this error backwards to $a_j^{(k-1)}$ by multiplying $\delta_i^{(k)}$ by the path weight $W_{ij}^{(k-1)}$.
3. Thus, the error received at $a_j^{(k-1)}$ is $\delta_i^{(k)} W_{ij}^{(k-1)}$.
4. However, $a_j^{(k-1)}$ may have been forwarded to multiple nodes in the next layer(i.e. node m in layer k).
5. Thus, the total error received at $a_j^{(k-1)}$ is $\delta_i^{(k)} W_{ij}^{(k-1)} + \delta_m^{(k)} W_{mj}^{(k-1)}$.
6. In fact, we can generalize this to be $\sum_i \delta_i^{(k)} W_{ij}^{(k-1)}$.
7. Now, we can have the correct error at $a_j^{(k-1)}$, we move it across neuron j at layer $k-1$ by multiplying with the local gradient $\sigma'(z_j^{(k-1)})$.
8. Finally, the error that reaches at $z_j^{(k-1)}$, called $\delta_j^{(k-1)}$ is $\sigma'(z_j^{(k-1)}) \sum_i \delta_i^{(k)} W_{ij}^{(k-1)}$.

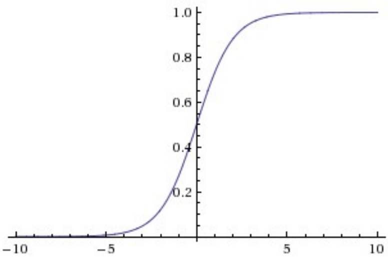
Homework 1-1

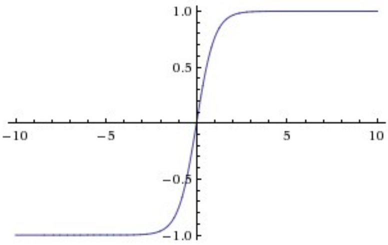
Derive the gradient with regard to the inputs of a softmax function when cross entropy loss is used for evaluation, i.e., find the gradients with respect to the softmax input vector θ, when the prediction is made by $\hat{y} = softmax(\theta)$. Remember the cross entropy function is:

$$CE(y, \hat{y}) = -\sum_i y_i \log(\hat{y}_i)$$

Activation Functions

- **Sigmoid.** $\sigma(x) = \frac{1}{1+\exp(-x)}$
- **Tanh.** $tanh(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$



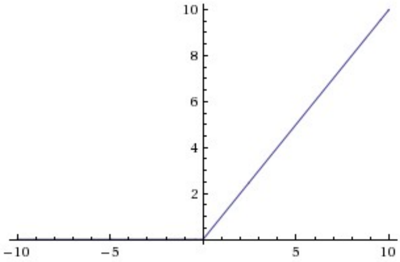


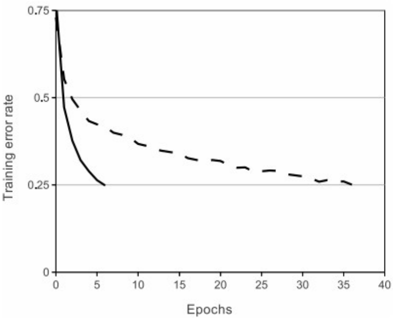
Sigmoids and Tanhs may saturate and kill gradients!

Preferred Activation Function - ReLU

Rectified Linear Units. $f(x) = \max(0, x)$

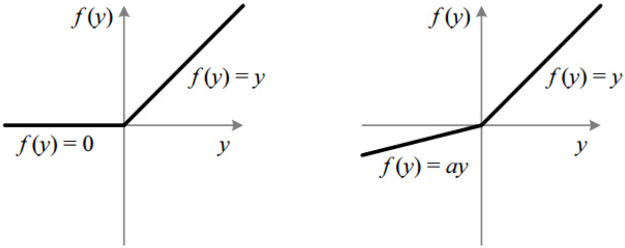
- (+) Accelerate the convergence significantly($\times 6$).
- (+) More efficient implementation compared with exponencials in Sigmoid/Tanh.
- ($-$) ReLU units can be "dead" during training.





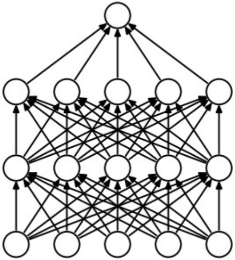
Leaky ReLU

Leaky ReLU fixes the "dying ReLU" problem. $f(x) = \max(ax, x)$ e.g. $a = 0.3$

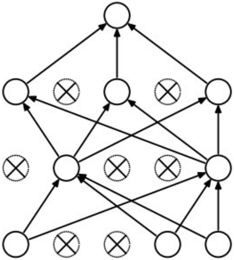


Regularization for Neural Networks - Dropout

Training: Sampling a sub-network within the full Neural Network.
Testing: Ensembles of all sub-networks(exponentially-sized) without dropout.



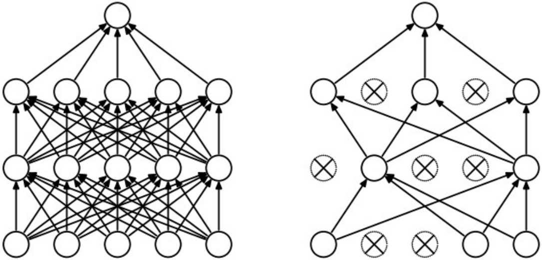
(a) Standard Neural Net



(b) After applying dropout.

Regularization for Neural Networks - Dropout

Training: Sampling a sub-network within the full Neural Network.
Testing: Ensembles of all sub-networks(exponentially-sized) without dropout.



(a) Standard Neural Net (b) After applying dropout.

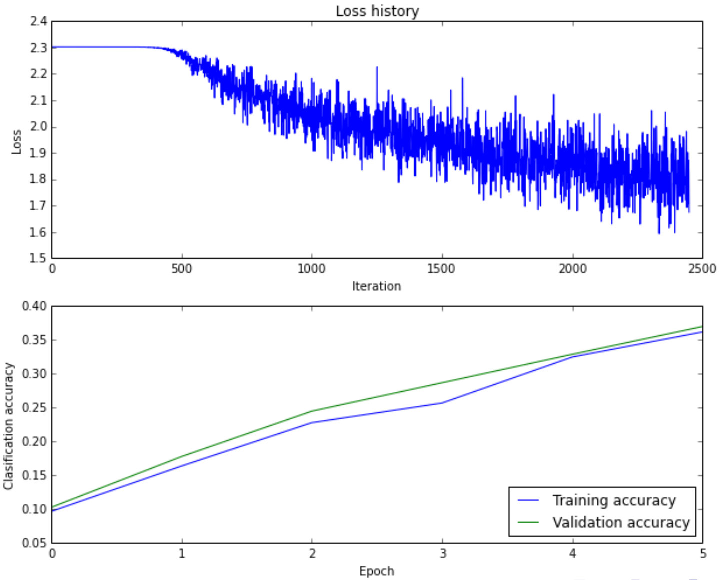
Example (Numpy Code)

```
H1 = np.maximum(0, np.dot(W1, X) + b1) # forward pass
U1 = np.random.rand(H1.shape) < p # dropout mask
H1 *= U1 # drop
```

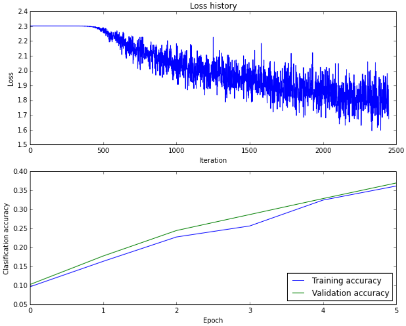
1 Motivation of Neural Networks and Deep Learning

2 Introduction to Neural Networks(Shallow Learning)
- Feedforward Propagation
- The Big Picture of Backpropgation
- Backpropagation Algorithm
- Tuning Parameters for Neural Networks

Tuning Parameters for Neural Networks

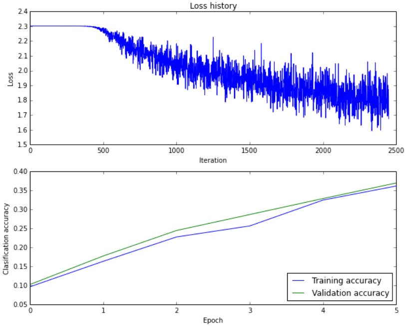


Tuning Parameters for Neural Networks



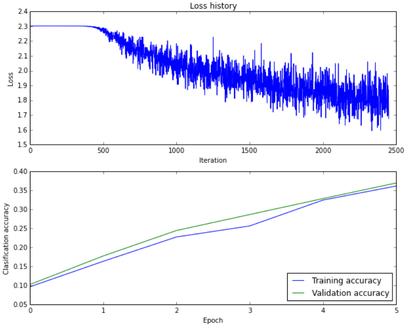
1. The loss is decreasing more or less linearly,

Tuning Parameters for Neural Networks



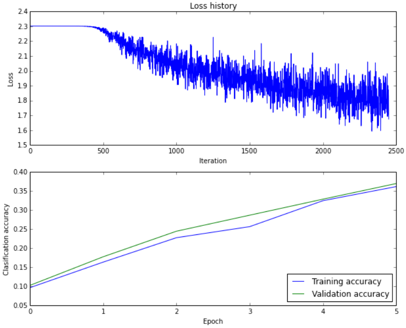
1. The loss is decreasing more or less linearly, indicating learning rate may be too low.

Tuning Parameters for Neural Networks



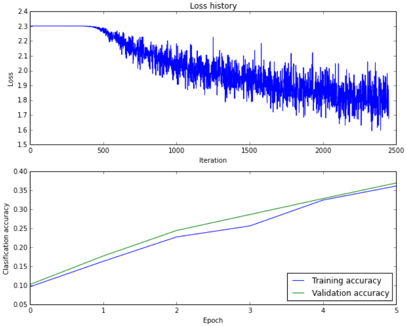
1. The loss is decreasing more or less linearly, indicating learning rate may be too low.
2. The loss fluctuates a lot,

Tuning Parameters for Neural Networks



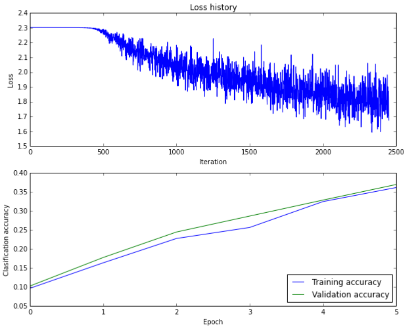
1. The loss is decreasing more or less linearly, indicating learning rate may be too low.
2. The loss fluctuates a lot, suggesting batch size may be too small.

Tuning Parameters for Neural Networks



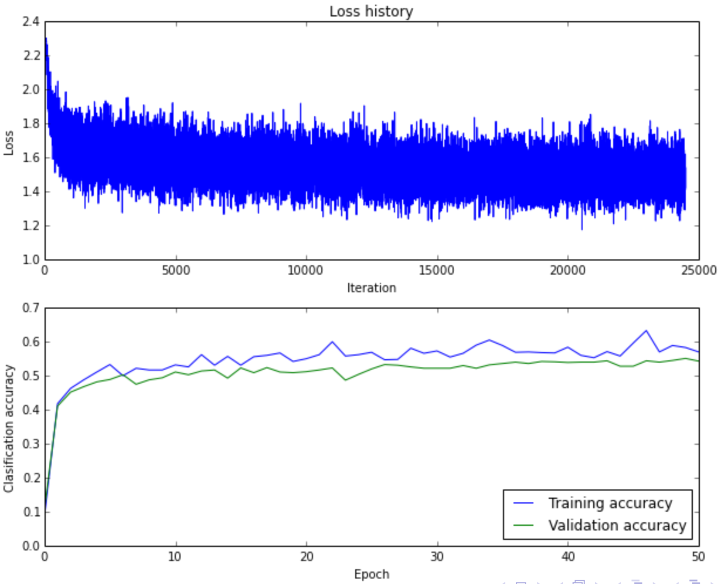
1. The loss is decreasing more or less linearly, indicating learning rate may be too low.
2. The loss fluctuates a lot, suggesting batch size may be too small.
3. There is no gap between the training and validation accuracy(Overfitting?),

Tuning Parameters for Neural Networks



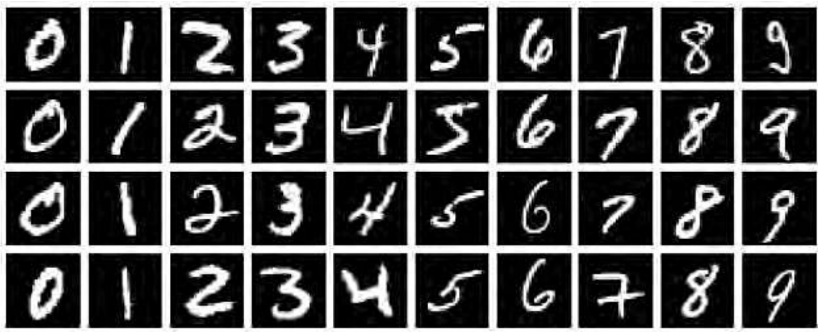
1. The loss is decreasing more or less linearly, indicating learning rate may be too low.
2. The loss fluctuates a lot, suggesting batch size may be too small.
3. There is no gap between the training and validation accuracy(~~Overfitting?~~low compacity), and we should increase training size(what if size is limited?).

Tuning Parameters for Neural Networks



Homework 1-2: MNIST Classification with MLP

In this homework, you are going to implent a MLP/DNN with Keras/TensorFlow. Since this is your first time coding task, you will be given some start code. It is important to note that different parameters could result quite different performance. Please feel free to tune these hyper-parameters, e.g. number of hidden layers, number of neurons in each hidden layer, learning rate, batch size, optimizer, etc.



Next Topic

ConvNets for Image Classification &
Object Detection