Biological Applications of Deep Learning
Lecture 3

Alexander Schönhuth

UNIVERSITÄT BIELEFELD
Faculty of Technology

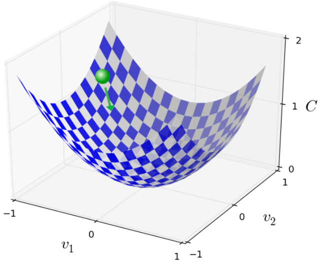
Bielefeld University
October 26, 2022

Contents Today

- Gradient Descent: Reminder
- The Backpropagation Algorithm
- Regularization in detail:
  - L1 / L2 Regularization
  - Dropout / Early Stopping

Reminder: Gradient Descent for Neural Networks

Gradient Descent



- Let $C(v_1, ..., v_n)$ be a differentiable function in n variables, here $n = 2$. We look for the minimum of C.
- *Idea*: At point v_1, v_2 (green ball), move into direction of steepest decline (green arrow). Do this iteratively.
- The steepest decline is given by the gradient

$$\nabla_{v_1,\ldots,v_n} C = \left(\frac{\partial C}{\partial v_1}, ..., \frac{\partial C}{\partial v_n}\right)$$

Gradient Descent for Neural Networks

Practical Scheme

Input

- A NN of depth L where parameters $\mathbf{w}$ represent both
  - weights $\mathbf{W}^{(j)} \in \mathbb{R}^{d(l) \times d(l-1)}, j = 1, ..., L$
  - biases $\mathbf{b}^j, j = 1, ..., L$
- Let $\mathbf{w}_0$ be appropriately chosen initial parameters
- Let $\mathbf{X}^{(\text{train})} \in \mathbb{R}^{m \times n}, \mathbf{y}^{(\text{train})} \in \mathbb{R}^m$ be m training data points $x \in \mathbb{R}^n$
- Let

$$C = \frac{1}{m} \sum_x C_x = \frac{1}{m} \sum_x C(f_{\mathbf{w}}(x), y(x))$$

  be a *cost function*.
- One can view $C = C(\mathbf{w})$ as a function in the parameters $\mathbf{w}$.

Gradient Descent for Neural Networks

Practical Scheme

- Let η be an appropriately chosen *learning rate*.

Iteration i

1. Compute $\nabla_{\mathbf{w}} C(\mathbf{w}_{i-1})$
   - Need training data to update C, based on having updated $\mathbf{w}$
2. Update: $\mathbf{w}^{(\mathbf{i})} \leftarrow \mathbf{w}^{(\mathbf{i-1})} + \eta \nabla_{\mathbf{w}} C$
   - $w_k^{(i)} \leftarrow w_k^{(i-1)} - \eta \frac{\partial C}{\partial w_k}$
   - $b_l^{(i)} \leftarrow b_l^{(i-1)} - \eta \frac{\partial C}{\partial b_l}$
3. Stop, if appropriate

This minimizes the cost C, hence adjusts the NN to the training data.

Deep Learning: Challenges

- The function f representing a neural network with L layers (with depth L) are written

$$y = f(\mathbf{x}^0) = f^{(L)}(f^{(L-1)}(\ldots(f^{(1)}(\mathbf{x}^{(0)}))\ldots))$$

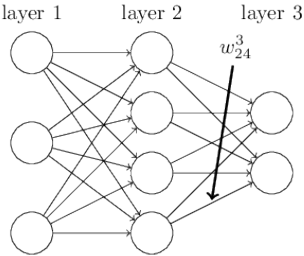
where $\mathbf{x}^l = f^{(l)}(\mathbf{x}^{l-1}) = \mathbf{a}^{\mathbf{l}}(\mathbf{W}^{(\mathbf{l})}\mathbf{x}^{l-1} + \mathbf{b}^{\mathbf{l}})$

- ☞ Functions $f_{\mathbf{w}}$ representing NN's cannot be described in closed form
- Hence the loss $C(\mathbf{w}) := C(f_{\mathbf{w}}) := C(f_{\mathbf{w}}, f^*)$ cannot be described in closed form either

How to compute gradients and perform gradient descent?

Computing Gradients: The Backpropagation Algorithm

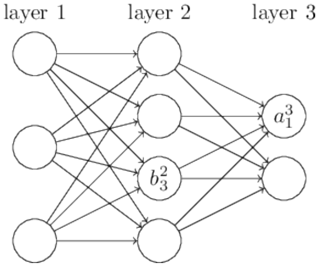
Notation



w^l_{jk} is the weight from the k^{th} neuron in the $(l-1)^{\text{th}}$ layer to the j^{th} neuron in the l^{th} layer

- weight w^l_{jk} links node k in layer $l-1$ with node j in layer l
- $w^l_{jk} = \mathbf{W}^{(l)}_{jk}$ in the earlier notation
- *Reminder*: width of layer l: $d(l)$, so $\mathbf{W}^{(l)} \in \mathbb{R}^{d(l)\times d(l-1)}$

Notation



- b_j^l is the bias of neuron j in layer l
- a_j^l is the activation *value* of neuron j in layer l
- $b_j^l = \mathbf{b}_j^{(l)}, a_j^l = \mathbf{x}_j^{(l)}, \mathbf{a}^l = \mathbf{x}^{(l)}$ in earlier notation

Notation

Using a sigmoid function σ as activation function, we obtain

$$a_j^l = \sigma(\sum_k w_{jk}^l a_k^{l-1} + b_j^l) \tag{1}$$

which can further be written

$$\mathbf{a}^l = \sigma(\mathbf{W}^{(l)} \mathbf{a}^{l-1} + \mathbf{b}^l) \tag{2}$$

Remark: here and in the following, σ can be replaced by an *arbitrary activation function that is differentiable*.

We further define

$$z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l \quad \text{that is} \quad a_j^l = \sigma(z_j^l) \tag{3}$$

such that

$$\mathbf{z}^l := (z_1^l, ..., z_{d(l)}^l)^T = \mathbf{W}^{(l)} \mathbf{a}^{l-1} + \mathbf{b}^l \quad \text{that is} \quad \mathbf{a}^l = \sigma(\mathbf{z}^l) \tag{4}$$

Notation

We further write

- $y(x)$ for the label of a training data point x
- *Note*: $y(x)$ can be identified with $f^*(x)$ where f^* is the true function
- $\mathbf{a}^L(x)$, the output of the last layer, represents the network function, so $\mathbf{a}^L(x) = f(x)$ in earlier notation.

Backpropagation

Goal

- We would like to compute gradient $\nabla_{\mathbf{W},\mathbf{b}} C$
- Therefore, we need to compute all partial derivatives

$$\frac{\partial C}{\partial w_{jk}^l} \quad \text{and} \quad \frac{\partial C}{\partial b_j^l} \tag{5}$$

- For further convenience, we define

$$\delta_j^l := \frac{\partial C}{\partial z_j^l} \tag{6}$$

BACKPROPAGATION

- For further convenience, we define

$$\delta_j^l := \frac{\partial C}{\partial z_j^l}$$

- For example, by the chain rule of differentiation (†):

$$\begin{array}{ccccc} \frac{\partial C}{\partial b_j^l} & \overset{(\dagger)}{=} & \delta_j^l \frac{\partial z_j^l}{\partial b_j^l} = \delta_j^l \frac{\partial(\sum_k w_{jk}^l a_k^{l-1} + b_j^l)}{\partial b_j^l} & = & \delta_j^l \\ \frac{\partial C}{\partial w_{jk^*}^l} & \overset{(\dagger)}{=} & \delta_j^l \frac{\partial z_j^l}{\partial w_{jk^*}^l} = \delta_j^l \frac{\partial(\sum_k w_{jk}^l a_k^{l-1} + b_j^l)}{\partial w_{jk^*}^l} & = & \delta_J^l a_{k^*}^{l-1} \end{array} \tag{7}$$

- *Idea*: Focus on computing δ_j^l, derive $\frac{\partial C}{\partial b_j^l}$ and $\frac{\partial C}{\partial w_{jk}^l}$ by (7)

Notation

- Let m be the total number of training examples. Then we define C

$$C(f, f^*) = C(a^L) := \frac{1}{2m} \sum_x ||y(x) - a^L(x)||^2 \tag{8}$$

as *quadratic cost function* (only for easier presentation!)
- *Note*: y resp. $f^*(x)$ are fixed, so C varies in a^L (= f) only.
- *Important*: $C = \frac{1}{m} \sum_x C_x$ where $C_x = \frac{1}{2}||y(x) - a^L(x)||^2$ is the cost on one individual training example
- *Idea*: Compute $\frac{\delta C_x}{\delta w}, \frac{\delta C_x}{\delta b}$ for all training data x and recover $\frac{\delta C}{\delta w}, \frac{\delta C}{\delta b}$ by averaging over x

DEFINITION

THE HADAMARD PRODUCT

Definition

Let $\mathbf{s}, \mathbf{t} \in \mathbb{R}^n$ be two vectors of equal length. Then the *Hadamard product* $\mathbf{s} \odot \mathbf{t}$ is defined by

$$(\mathbf{s} \odot \mathbf{t})_j = \mathbf{s}_j \cdot \mathbf{t}_j \quad \text{for } j = 1, ..., n \tag{9}$$

Backpropagation

Start: Output Layer – Computing δ^L

We have $a_j^L = \sigma(z_j^L)$, so

$$\delta_j^L = \frac{\partial C}{\partial z_j^L} = \sum_k \frac{\partial C}{\partial a_k^L} \frac{\partial a_k^L}{\partial z_j^L} \overset{\frac{\partial a_k^L}{\partial z_j^L}=0, j\neq k}{=} \frac{\partial C}{\partial a_j^L} \cdot \sigma'(z_j^L) \tag{10}$$

In other words,

$$\delta^L = \nabla_{\mathbf{a}^L} C \odot \sigma'(\mathbf{z}^L) \tag{11}$$

Backpropagation

Start: Output Layer – Computing δ^L

Further

$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$

and

$$\frac{\partial C}{\partial a_j^L} = \frac{\partial(\frac{1}{2}\sum_{j'}(y_{j'} - a_{j'}^L)^2)}{\partial a_j^L} = (a_j^L - y_j),$$

so overall

$$\delta_j^L = (a_j^L - y_j)\sigma(z_j^L)(1 - \sigma(z_j^L)) \tag{12}$$

Backpropagation

Start: Output Layer – Computing δ^L

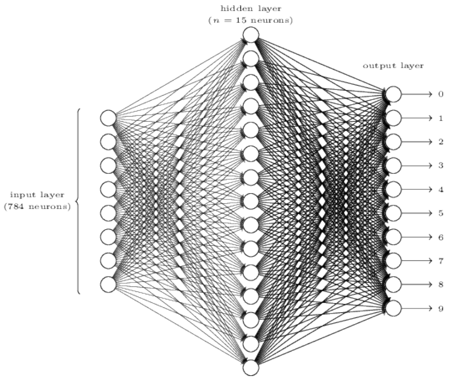
$$\delta_j^L = (a_j^L - y_j)\sigma'(z_j^L) \quad \text{that is} \quad \delta^L = (\mathbf{a}^L - \mathbf{y}) \odot \sigma'(\mathbf{z}^L) \qquad (13)$$

Interpretation

- $a_j^L - y_j$ determines how far off a_j^L from y_j is
- The further off, the steeper the gradient, the greater the adjustment
- $\sigma'(z_j^L)$ is close to zero if $\sigma(z_j^L)$ is either close to zero or close to one
- This can make sense, but can cause problems, because updates get very small (note remarks on alternative activation functions)

EXAMPLE

MNIST NETWORK



- *Truth*: One y_j is one, all others are zero
- If a_j^L is not one, updates are large: we need to make changes
- If a_j^L is close to one, and all others are close to zero, updates are small: no further adjustments necessary

Propagation – Computing δ^l from δ^{l+1}

We compute

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \delta_k^{l+1} \tag{14}$$

We further observe

$$z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1} \sigma(z_j^l) + b_k^{l+1} \tag{15}$$

which, by differentiation, leads to

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} \sigma'(z_j^l) \tag{16}$$

Backpropagation

Propagation – Computing δ^l from δ^{l+1}

Substituting (16) into (14), we obtain

$$\delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} \sigma'(z_j^l) \tag{17}$$

which can be overall expressed as

$$\delta^l = ((\mathbf{W}^{(l+1)})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{18}$$

- (18) "moves the error one layer backward" ☞ *backpropagation*
- Applying $\mathbf{W}^{(l+1)}$ to δ^{l+1} moves the error from the input of neurons in layer $l+1$ to the outputs of neurons in layer l
- $\sigma'(z^l)$ moves the error from the output of neurons in layer l to the inputs of neurons in layer l

Backpropagation

Computing $\frac{\partial C}{\partial b^l_j}$ and $\frac{\partial C}{\partial w^l_{jk}}$

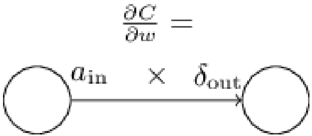
We further see that

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \tag{19}$$

and

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{20}$$

(20) explains that changes in weights are small if the input is small, or the error in the output is small:



Backpropagation

The Equations

Summary: the equations of backpropagation

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \tag{BP1}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{BP2}$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \tag{BP3}$$

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{BP4}$$

Backpropagation

The Algorithm

1. **Input x:** Set the corresponding activation a^1 for the input layer.

2. **Feedforward:** For each $l = 2, 3, \dots, L$ compute $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma(z^l)$.

3. **Output error δ^L:** Compute the vector $\delta^L = \nabla_a C \odot \sigma'(z^L)$.

4. **Backpropagate the error:** For each $l = L-1, L-2, \dots, 2$ compute $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$.

5. **Output:** The gradient of the cost function is given by $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$ and $\frac{\partial C}{\partial b^l_j} = \delta^l_j$.

BACKPROPAGATION

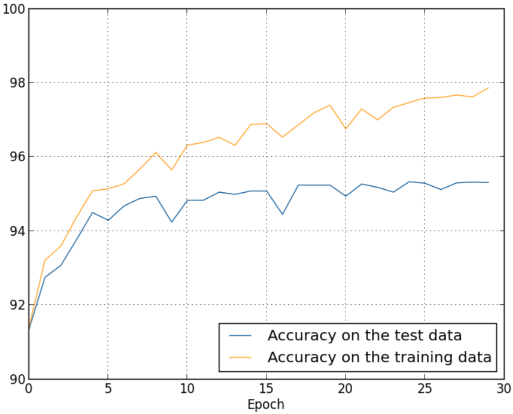
STOCHASTIC GRADIENT DESCENT

1. **Input a set of training examples**
2. **For each training example x:** Set the corresponding input activation $a^{x,1}$, and perform the following steps:
   - **Feedforward:** For each $l = 2, 3, \ldots, L$ compute $z^{x,l} = w^l a^{x,l-1} + b^l$ and $a^{x,l} = \sigma(z^{x,l})$.
   - **Output error $\delta^{x,L}$:** Compute the vector $\delta^{x,L} = \nabla_a C_x \odot \sigma'(z^{x,L})$.
   - **Backpropagate the error:** For each $l = L-1, L-2, \ldots, 2$ compute $\delta^{x,l} = ((w^{l+1})^T \delta^{x,l+1}) \odot \sigma'(z^{x,l})$.
3. **Gradient descent:** For each $l = L, L-1, \ldots, 2$ update the weights according to the rule $w^l \rightarrow w^l - \frac{\eta}{m} \sum_x \delta^{x,l} (a^{x,l-1})^T$, and the biases according to the rule $b^l \rightarrow b^l - \frac{\eta}{m} \sum_x \delta^{x,l}$.

Employing Regularization

Regularization Revisited

Motivation



No regularization leads to overfitting

L2-regularized Cross Entropy

We add a L2 regularization term to the cost (here: cross-entropy). Thereby λ is the *regularization parameter*.

$$C = -\frac{1}{m}\sum_x \sum_j [y_j \log a_j^L + (1-y_j)\log(1-a_j^L)] + \frac{\lambda}{2m}\sum_w w^2 \quad (21)$$

Writing $C_0 = -\frac{1}{m}\sum_x \sum_j [y_j \log a_j^L + (1-y_j)\log(1-a_j^L)]$ then makes

$$C = C_0 + \frac{\lambda}{m}\sum_w w^2 \quad (22)$$

Remark: This can be done with any cost function C_0.

L2-regularized Cross Entropy

This further yields the *partial derivatives*

$$\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{m} w \tag{23}$$

$$\frac{\partial C}{\partial b} = \frac{\partial C_0}{\partial b} \tag{24}$$

with *update rules* (rescaling weights with $(1 - \frac{\eta\lambda}{m})$ is called *weight decay*)

$$b \leftarrow b - \eta \frac{\partial C_0}{\partial b} \tag{25}$$

$$w \leftarrow w - \eta \frac{\partial C_0}{\partial w} - \eta \frac{\lambda}{m} w = (1 - \frac{\eta\lambda}{m}) w - \eta \frac{\partial C_0}{\partial w} \tag{26}$$

Update rules for *stochastic gradient descent*, for overall m training data, batch size $\hat{m}$:

$$b \leftarrow b - \frac{\eta}{\hat{m}} \sum_x \frac{\partial C_x}{\partial b} \tag{27}$$

$$w \leftarrow (1 - \frac{\eta\lambda}{m}) w - \frac{\eta}{\hat{m}} \sum_x \frac{\partial C_x}{\partial w} \tag{28}$$

L2 Regularization

Explanations

- For sake of better illustration, consider
  - C_0 to be a quadratic cost function, like mean squared loss
  - In general, one can consider the quadratic (second order term) approximation of C_0
  - only one training example, that is $m = 1$ in the following
- Let
$$\mathbf{w}^* := \arg\min_{\mathbf{w}} C_0(\mathbf{w}) \tag{29}$$
  be the true minimum (which we don't know).
- Let k be the length of $\mathbf{w}$ (so k the number of weights to be trained)

L2 REGULARIZATION

EXPLANATIONS

- Let the *Hessian matrix* $\mathbf{H} \in \mathbb{R}^{k \times k}$ be defined by

$$\mathbf{H}_{ww'} = \frac{\partial C_0}{\partial w \partial w'} \tag{30}$$

- The gradient of C_0 vanishes at $\mathbf{w}^*$, because $\mathbf{w}^*$ is the minimum.
- By Taylor's approximation, because C_0 is quadratic, we know that

$$C_0(\mathbf{w}) = C_0(\mathbf{w}^*) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T \mathbf{H}(\mathbf{w} - \mathbf{w}^*) \tag{31}$$

- That means that the minimum of C_0 appears where

$$\nabla_{\mathbf{w}} C_0(\mathbf{w}) = \mathbf{H}(\mathbf{w} - \mathbf{w}^*) = \mathbf{0} \tag{32}$$

L2 Regularization

Explanations

- Let $\tilde{\mathbf{w}}$ be the minimum of $C = C_0 + \frac{1}{2}||\mathbf{w}||^2$
- Recalling $\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \lambda w$ (see (23) with $m = 1$), we know that

$$\mathbf{H}(\tilde{\mathbf{w}} - \mathbf{w}^*) + \lambda \tilde{\mathbf{w}} = 0 \tag{33}$$

- This further leads to ($\mathbf{I}$ is the identity)

$$\tilde{\mathbf{w}} = (\mathbf{H} + \lambda \mathbf{I})^{-1} \mathbf{H} \mathbf{w}^* \tag{34}$$

- For $\lambda \to 0$, we get $\tilde{\mathbf{w}} \to \mathbf{w}^*$

L2 Regularization

Explanations

- Let $\mathbf{D}$ be diagonal where entries $\mathbf{D}_{ii}$ are the eigenvalues of $\mathbf{H}$
- Let $\mathbf{Q}$ collect the eigenvectors of $\mathbf{H}$
- Since $\mathbf{H}$ is real and symmetric, $\mathbf{Q}$ is orthogonal, and $\mathbf{H}$ can be written

$$\mathbf{H} = \mathbf{Q}\mathbf{D}\mathbf{Q}^T \tag{35}$$

- Substituting (35) in (34), we obtain

$$\tilde{\mathbf{w}} = (\mathbf{Q}\mathbf{D}\mathbf{Q}^T + \lambda\mathbf{I})^{-1}\mathbf{Q}\mathbf{D}\mathbf{Q}^T\mathbf{w}^* \tag{36}$$

- further yielding

$$\tilde{\mathbf{w}} = \mathbf{Q}(\mathbf{D} + \lambda\mathbf{I})^{-1}\mathbf{D}\mathbf{Q}^T\mathbf{w}^* \tag{37}$$

L2 REGULARIZATION

EXPLANATIONS

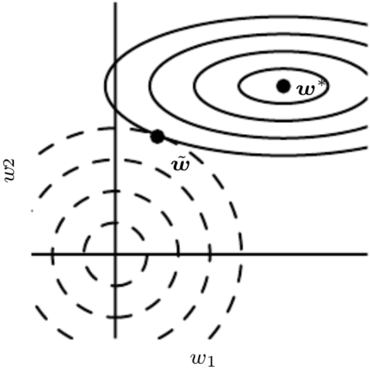
- *Interpretation*:
  - $\tilde{\mathbf{w}}$ is a rescaled version of $\mathbf{w}^*$
  - The component of $\mathbf{w}^*$ that aligns with the i-th eigenvector of $\mathbf{H}$ is rescaled by a factor of

$$\frac{\mathbf{D}_{ii}}{\mathbf{D}_{ii} + \lambda} \tag{38}$$

- Eigenvectors of $\mathbf{H}$ referring to large eigenvalues indicate directions where the gradient rapidly changes (increases when going away from $\mathbf{w}^*$, where it is zero)
- Eigenvectors of $\mathbf{H}$ referring to small eigenvalues indicate directions where the gradient hardly changes
- The latter directions can be neglected
- In other words, components of weights referring to such directions can be decayed away by regularization

Regularization Revisited

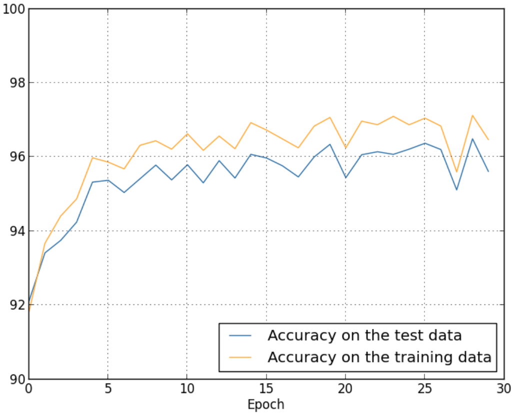
Motivation



L2 regularization shrinks weights along eigenvectors of the Hessian

Regularization Revisited

Motivation



Regularization prevents overfitting

Regularization Revisited

L1 Regularization

For L1 regularization, we modify the cost function

$$C = C_0 + \frac{\lambda}{m} \sum_w |w| \tag{39}$$

by adding the sum of the absolute values of the weights.

Gradient:

$$\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{m} \operatorname{sgn}(w) \tag{40}$$

Update:

$$w \leftarrow w' = w - \frac{\eta \lambda}{m} \operatorname{sgn}(w) - \eta \frac{\partial C_0}{\partial w} \tag{41}$$

L1 Regularization

Explanations

- L1 regularization does not have a similarly neat algebraic explanation like L2 regularization
- An approximate explanation is that components referring to small eigenvalues of the Hessian are set to zero, rather than smoothly shrunken
- Overall, a *sparse* set of weights is achieved

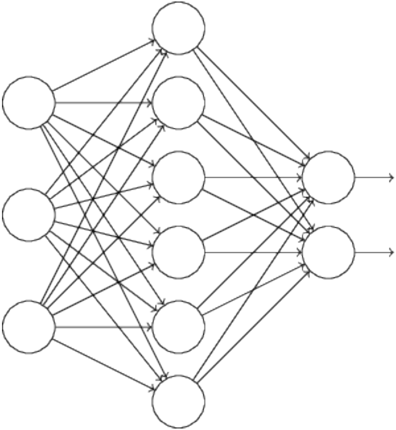
Regularization Revisited

L1 versus L2 Regularization

- In L1 regularization, weights shrink by a *constant* amount.
- In L2 regularization, weights shrink by an amount *proportionally* to w.
- L1 regularization tends to bring forward a small number of *high-importance connections*.
- L2 regularization tends to keep all weights small.

REGULARIZATION REVISITED

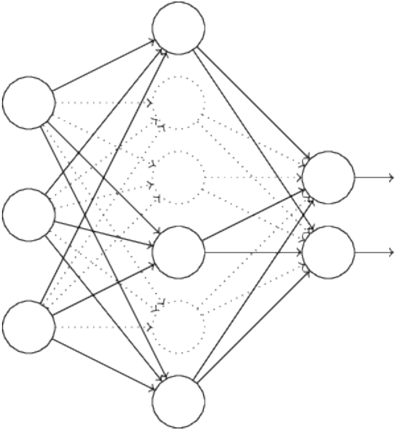
DROPOUT



Full network, before dropout

Regularization Revisited

Dropout



Network after having dropped half of the hidden nodes

Procedure

1. Choose a mini batch of training data of size $\hat{m}$
2. Randomly delete half of the hidden nodes, while keeping all input and output nodes
3. Train the resulting network using the mini batch; update all weights and biases
4. If validation accuracy not yet satisfying, return to 1.
5. After each epoch, decrease each weight by a factor of $\frac{1}{2}$

Dropout

Explanations

- Dropout can be perceived as averaging over several smaller networks, where averaging over several models is generally helpful to prevent overfitting
- Dropout can be perceived as projecting points in parameter space onto the linear subspace defined by only half of the elementary basis vectors.
- Combining optima in subspaces yields a selection of parameters that are not optimal, but nearby an optimum
  ☞ experience shows that this prevents overfitting
- Dropout prevents “co-adaptation of neurons”

L1/2 Regularization, Dropout, Early Stopping

Take-Home Message

Try to find a reasonable point near the very optimum

- *L1/2 regularization*: shrink or eliminate weights that don't change much
- *Dropout*: Randomly project points to linear subspaces, and optimize there, and then average out
- *Early stopping*: Stop before reaching the optimum

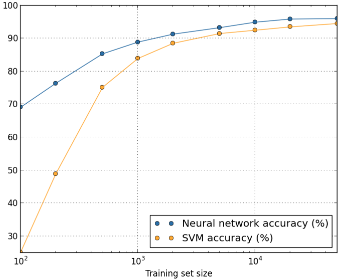
Regularization Revisited

Artificial Expansion of Training Data

More training data improves test accuracy

Regularization Revisited

Artificial Expansion of Training Data

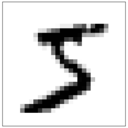


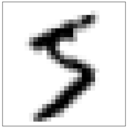
NN versus SVM on same training data

- Sometimes better training data delivers substantial improvements
- Always good to aim for methodical improvements, but:
- Don't miss "easy wins" by generating more and/or better training data

Regularization Revisited

Generating Artificial Training Data





Rotating 5 by 15 degrees to the left yields new training datum

Other Techniques

- Translating, skewing
- "Elastic distortions"
- For more details, see [Simard, Steinkraus & Platt, 2003]
  `https://ieeexplore.ieee.org/document/1227801`

LECTURE3: SUMMARY

- Backpropagation: See `http://www.deeplearningbook.org/` 6.5 and `http://neuralnetworksanddeeplearning.com/`, Chapter 2, until and including "The Backpropagation Algorithm"
- Regularization: See `http://www.deeplearningbook.org/` Chapter 7, (for example 7.1, 7.8, 7.12) and `http://neuralnetworksanddeeplearning.com/`, Chapter 3
- For *further reading*, also consider:
- Read "In what sense is backpropagation a fast algorithm?" in Nielsen's book, chapter 2 (`http://neuralnetworksanddeeplearning.com/chap2.html`),
- Read "Backpropagation: the big picture" in Nielsen's book, chapter 2
- and try to make sense of what you have read.

Outlook

- Convolutional Neural Networks
- `http://www.deeplearningbook.org/`, Chapter 9
- `http://neuralnetworksanddeeplearning.com/`, "Deep Learning"

Thanks for your attention