Introduction to Neural Networks and Deep Learning

Regularization

Andres Mendez-Vazquez

December 2, 2019

Outline

Outline

Introduction

What did we see until now?

The design of learning machines from two main points:

- Statistical Point of View
- Linear Algebra and Optimization Point of View

Introduction

What did we see until now?

The design of learning machines from two main points:

- Statistical Point of View
- Linear Algebra and Optimization Point of View

Going back to the probability models

We might think in the machine to be learned as a function $g(x|\mathcal{D})$....

- Something as curve fitting...

Remark: Where the $x_g \sim p(x|\Theta)$!!!

Introduction

What did we see until now?

The design of learning machines from two main points:

- Statistical Point of View
- Linear Algebra and Optimization Point of View

Going back to the probability models

We might think in the machine to be learned as a function $g(x|\mathcal{D})$....

- Something as curve fitting...

Under a data set

$$\mathcal{D} = \{(x_i, y_i) | i = 1, 2, ..., N\} \tag{1}$$

Remark: Where the $x_i \sim p(x|\Theta)$!!!

Introduction

What did we see until now?

The design of learning machines from two main points:

- Statistical Point of View
- Linear Algebra and Optimization Point of View

Going back to the probability models

We might think in the machine to be learned as a function $g(\boldsymbol{x}|\mathcal{D})$....

- Something as curve fitting...

Under a data set

$$\mathcal{D} = \{(x_i, y_i) \,|\, i = 1, 2, ..., N\} \quad (1)$$

Remark: Where the $x_i \sim p(\boldsymbol{x}|\Theta)$!!!

Introduction

What did we see until now?

The design of learning machines from two main points:

- Statistical Point of View
- Linear Algebra and Optimization Point of View

Going back to the probability models

We might think in the machine to be learned as a function $g(\boldsymbol{x}|\mathcal{D})$....

- Something as curve fitting...

Under a data set

$$\mathcal{D} = \{(x_i, y_i) \,|\, i = 1, 2, ..., N\} \quad (1)$$

Remark: Where the $x_i \sim p(\boldsymbol{x}|\Theta)$!!!

Introduction

What did we see until now?

The design of learning machines from two main points:

- Statistical Point of View
- Linear Algebra and Optimization Point of View

Going back to the probability models

We might think in the machine to be learned as a function $g(\boldsymbol{x}|\mathcal{D})$....

- Something as curve fitting...

Under a data set

$$\mathcal{D} = \{(\boldsymbol{x}_i, y_i) \,|i = 1, 2, ..., N\} \tag{1}$$

Remark: Where the $x_i \sim p(\boldsymbol{x}|\Theta)$!!!

Introduction

What did we see until now?

The design of learning machines from two main points:

- Statistical Point of View
- Linear Algebra and Optimization Point of View

Going back to the probability models

We might think in the machine to be learned as a function $g(\boldsymbol{x}|\mathcal{D})$....

- Something as curve fitting...

Under a data set

$$\mathcal{D} = \{(\boldsymbol{x}_i, y_i) \, | i = 1, 2, ..., N\} \tag{1}$$

Remark: Where the $\boldsymbol{x}_i \sim p(\boldsymbol{x}|\Theta)$!!!

Thus, we have that

Two main functions

- A function $g(\boldsymbol{x}|\mathcal{D})$ obtained using some algorithm!!!
- $E[y|\boldsymbol{x}]$ the optimal regression...

Thus, we have that

Two main functions

- A function $g(\boldsymbol{x}|\mathcal{D})$ obtained using some algorithm!!!
- $E[y|\boldsymbol{x}]$ the optimal regression...

Important

The key factor here is the dependence of the approximation on $\mathcal{D}$.

- This is the reason of studying fusion of information at decision level...

Thus, we have that

Two main functions

- A function $g(\boldsymbol{x}|\mathcal{D})$ obtained using some algorithm!!!
- $E[y|\boldsymbol{x}]$ the optimal regression...

Important

The key factor here is the dependence of the approximation on $\mathcal{D}$.

Why?

The approximation may be very good for a specific training data set but very bad for another.

- This is the reason of studying fusion of information at decision level...

Thus, we have that

Two main functions

- A function $g(\boldsymbol{x}|\mathcal{D})$ obtained using some algorithm!!!
- $E[y|\boldsymbol{x}]$ the optimal regression...

Important

The key factor here is the dependence of the approximation on $\mathcal{D}$.

Why?

The approximation may be very good for a specific training data set but very bad for another.

- This is the reason of studying fusion of information at decision level...

Thus, we have that

Two main functions

- A function $g(\boldsymbol{x}|\mathcal{D})$ obtained using some algorithm!!!
- $E[y|\boldsymbol{x}]$ the optimal regression...

Important

The key factor here is the dependence of the approximation on $\mathcal{D}$.

Why?

The approximation may be very good for a specific training data set but very bad for another.

- This is the reason of studying fusion of information at decision level...

Outline

How do we measure the difference? [1]

We have that

$$Var(X) = E((X-\mu)^2)$$

How do we measure the difference? [1]

We have that

$$Var(X) = E((X - \mu)^2)$$

We can do that for our data

$$Var_{\mathcal{D}}\left(g\left(\boldsymbol{x}|\mathcal{D}\right)\right) = E_D\left(\left(g\left(\boldsymbol{x}|\mathcal{D}\right) - E\left[y|\boldsymbol{x}\right]\right)^2\right)$$

How do we measure the difference? [1]

We have that

$$Var(X) = E((X - \mu)^2)$$

We can do that for our data

$$Var_{\mathcal{D}}\left(g\left(\boldsymbol{x}|\mathcal{D}\right)\right) = E_D\left(\left(g\left(\boldsymbol{x}|\mathcal{D}\right) - E\left[y|\boldsymbol{x}\right]\right)^2\right)$$

Now, if we add and subtract

$$E_D\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right] \quad (2)$$

Remark: The expected output of the machine $g\left(\boldsymbol{x}|\mathcal{D}\right)$

How do we measure the difference? [1]

We have that

$$Var(X) = E((X-\mu)^2)$$

We can do that for our data

$$Var_{\mathcal{D}}\left(g\left(\boldsymbol{x}|\mathcal{D}\right)\right) = E_D\left(\left(g\left(\boldsymbol{x}|\mathcal{D}\right) - E\left[y|\boldsymbol{x}\right]\right)^2\right)$$

Now, if we add and subtract

$$E_D\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right] \tag{2}$$

Remark: The expected output of the machine $g\left(\boldsymbol{x}|\mathcal{D}\right)$

Thus, we have that

Or Original variance

$$
\begin{aligned}
Var_{\mathcal{D}}\left(g\left(\boldsymbol{x}|\mathcal{D}\right)\right) &= E_{D}\left(\left(g\left(\boldsymbol{x}|\mathcal{D}\right) - E\left[y|\boldsymbol{x}\right]\right)^{2}\right)\\
&= E_{D}\left(\left(g\left(\boldsymbol{x}|\mathcal{D}\right) - E_{D}\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right] + E_{D}\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right] - E\left[y|\boldsymbol{x}\right]\right)^{2}\right)\\
&= E_{D}\left(\left(g\left(\boldsymbol{x}|\mathcal{D}\right) - E_{D}\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right]\right)^{2} + \ldots\right.\\
&\quad \ldots 2\left(\left(g\left(\boldsymbol{x}|\mathcal{D}\right) - E_{D}\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right]\right)\right)\left(E_{D}\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right] - E\left[y|\boldsymbol{x}\right]\right) + \ldots\\
&\quad \left.\ldots\left(E_{D}\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right] - E\left[y|\boldsymbol{x}\right]\right)^{2}\right)
\end{aligned}
$$

Thus, we have that

Or Original variance

$$
\begin{aligned}
Var_{\mathcal{D}}\left(g\left(\boldsymbol{x}|\mathcal{D}\right)\right) &= E_D\left(\left(g\left(\boldsymbol{x}|\mathcal{D}\right) - E\left[y|\boldsymbol{x}\right]\right)^2\right)\\
&= E_D\left(\left(g\left(\boldsymbol{x}|\mathcal{D}\right) - E_D\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right] + E_D\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right] - E\left[y|\boldsymbol{x}\right]\right)^2\right)\\
&= E_D\left(\left(g\left(\boldsymbol{x}|\mathcal{D}\right) - E_D\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right]\right)^2 + \ldots\right.\\
&\ldots 2\left(\left(g\left(\boldsymbol{x}|\mathcal{D}\right) - E_D\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right]\right)\right)\left(E_D\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right] - E\left[y|\boldsymbol{x}\right]\right) + \ldots\\
&\ldots \left.\left(E_D\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right] - E\left[y|\boldsymbol{x}\right]\right)^2\right)
\end{aligned}
$$

Finally

$$
E_D\left(\left(\left(g\left(\boldsymbol{x}|\mathcal{D}\right) - E_D\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right]\right)\right)\left(E_D\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right] - E\left[y|\boldsymbol{x}\right]\right)\right) = ? \quad (3)
$$

Thus, we have that

Or Original variance

$$
\begin{aligned}
Var_{\mathcal{D}}\left(g\left(\boldsymbol{x}|\mathcal{D}\right)\right) &= E_D\left(\left(g\left(\boldsymbol{x}|\mathcal{D}\right) - E\left[y|\boldsymbol{x}\right]\right)^2\right) \\
&= E_D\left(\left(g\left(\boldsymbol{x}|\mathcal{D}\right) - E_D\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right] + E_D\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right] - E\left[y|\boldsymbol{x}\right]\right)^2\right) \\
&= E_D\left(\left(g\left(\boldsymbol{x}|\mathcal{D}\right) - E_D\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right]\right)^2 + \ldots\right. \\
&\ldots 2\left(\left(g\left(\boldsymbol{x}|\mathcal{D}\right) - E_D\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right]\right)\right)\left(E_D\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right] - E\left[y|\boldsymbol{x}\right]\right) + \ldots \\
&\ldots \left.\left(E_D\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right] - E\left[y|\boldsymbol{x}\right]\right)^2\right)
\end{aligned}
$$

Finally

$$
E_D\left(\left(\left(g\left(\boldsymbol{x}|\mathcal{D}\right) - E_D\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right]\right)\right)\left(E_D\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right] - E\left[y|\boldsymbol{x}\right]\right)\right) = ? \quad (3)
$$

Thus, we have that

Or Original variance

$$
\begin{aligned}
Var_{\mathcal{D}}\left(g\left(\boldsymbol{x}|\mathcal{D}\right)\right) &= E_D\left(\left(g\left(\boldsymbol{x}|\mathcal{D}\right) - E\left[y|\boldsymbol{x}\right]\right)^2\right) \\
&= E_D\left(\left(g\left(\boldsymbol{x}|\mathcal{D}\right) - E_D\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right] + E_D\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right] - E\left[y|\boldsymbol{x}\right]\right)^2\right) \\
&= E_D\left(\left(g\left(\boldsymbol{x}|\mathcal{D}\right) - E_D\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right]\right)^2 + ...\right. \\
&\quad ...2\left(\left(g\left(\boldsymbol{x}|\mathcal{D}\right) - E_D\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right]\right)\right)\left(E_D\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right] - E\left[y|\boldsymbol{x}\right]\right) + ... \\
&\quad \left. ...\left(E_D\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right] - E\left[y|\boldsymbol{x}\right]\right)^2\right)
\end{aligned}
$$

Finally

$$
E_D\left(\left(\left(g\left(\boldsymbol{x}|\mathcal{D}\right) - E_D\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right]\right)\right)\left(E_D\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right] - E\left[y|\boldsymbol{x}\right]\right)\right) = ? \tag{3}
$$

Outline

We have the Bias-Variance

Our Final Equation

$$E_D\left(\left(g\left(\boldsymbol{x}|\mathcal{D}\right)-E\left[y|\boldsymbol{x}\right]\right)^2\right)=\underbrace{E_D\left(\left(g\left(\boldsymbol{x}|\mathcal{D}\right)-E_D\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right]\right)^2\right)}_{VARIANCE}+\underbrace{\left(E_D\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right]-E\left[y|\boldsymbol{x}\right]\right)^2}_{BIAS}$$

We have the Bias-Variance

Our Final Equation

$$E_D\left((g(\boldsymbol{x}|\mathcal{D}) - E[y|\boldsymbol{x}])^2\right) = \underbrace{E_D\left((g(\boldsymbol{x}|\mathcal{D}) - E_D[g(\boldsymbol{x}|\mathcal{D})])^2\right)}_{VARIANCE} + \underbrace{(E_D[g(\boldsymbol{x}|\mathcal{D})] - E[y|\boldsymbol{x}])^2}_{BIAS}$$

Where the variance

It represents the measure of the error between our machine $g(\boldsymbol{x}|\mathcal{D})$ and the expected output of the machine under $\boldsymbol{x}_i \sim p(\boldsymbol{x}|\Theta)$.

We have the Bias-Variance

Our Final Equation

$$E_D\left(\left(g\left(\boldsymbol{x}|\mathcal{D}\right)-E\left[y|\boldsymbol{x}\right]\right)^2\right)=\underbrace{E_D\left(\left(g\left(\boldsymbol{x}|\mathcal{D}\right)-E_D\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right]\right)^2\right)}_{VARIANCE}+\underbrace{\left(E_D\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right]-E\left[y|\boldsymbol{x}\right]\right)^2}_{BIAS}$$

Where the variance

It represents the measure of the error between our machine $g\left(\boldsymbol{x}|\mathcal{D}\right)$ and the expected output of the machine under $\boldsymbol{x}_i \sim p\left(\boldsymbol{x}|\Theta\right)$.

Where the bias

It represents the quadratic error between the expected output of the machine under $\boldsymbol{x}_i \sim p\left(\boldsymbol{x}|\Theta\right)$ and the expected output of the optimal regression.

We have the Bias-Variance

Our Final Equation

$$E_D\left(\left(g\left(\boldsymbol{x}|\mathcal{D}\right)-E\left[y|\boldsymbol{x}\right]\right)^2\right)=\underbrace{E_D\left(\left(g\left(\boldsymbol{x}|\mathcal{D}\right)-E_D\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right]\right)^2\right)}_{VARIANCE}+\underbrace{\left(E_D\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right]-E\left[y|\boldsymbol{x}\right]\right)^2}_{BIAS}$$

Where the variance

It represents the measure of the error between our machine $g\left(\boldsymbol{x}|\mathcal{D}\right)$ and the expected output of the machine under $\boldsymbol{x}_i \sim p\left(\boldsymbol{x}|\Theta\right)$.

Where the bias

It represents the quadratic error between the expected output of the machine under $\boldsymbol{x}_i \sim p\left(\boldsymbol{x}|\Theta\right)$ and the expected output of the optimal regression.

Remarks

We have then

Even if the estimator is unbiased, it can still result in a large mean square error due to a large variance term.

Remarks

We have then

Even if the estimator is unbiased, it can still result in a large mean square error due to a large variance term.

The situation is more dire in a finite set of data $\mathcal{D}$

We have then a trade-off:

1. Increasing the bias decreases the variance and vice versa.
2. This is known as the **bias–variance dilemma**.

Remarks

We have then

Even if the estimator is unbiased, it can still result in a large mean square error due to a large variance term.

The situation is more dire in a finite set of data $\mathcal{D}$

We have then a trade-off:

1. Increasing the bias decreases the variance and vice versa.
2. This is known as the **bias–variance dilemma**.

Remarks

We have then

Even if the estimator is unbiased, it can still result in a large mean square error due to a large variance term.

The situation is more dire in a finite set of data $\mathcal{D}$

We have then a trade-off:

1. Increasing the bias decreases the variance and vice versa.
2. This is known as the **bias–variance dilemma**.

Similar to...

Curve Fitting

If, for example, the adopted model is complex (many parameters involved) with respect to the number N, the model will fit the idiosyncrasies of the specific data set.

Similar to...

Curve Fitting

If, for example, the adopted model is complex (many parameters involved) with respect to the number N, the model will fit the idiosyncrasies of the specific data set.

Thus

Thus, it will result in low bias but will yield high variance, as we change from one data set to another data set.

- However, N is always finite!!!

Similar to...

Curve Fitting

If, for example, the adopted model is complex (many parameters involved) with respect to the number N, the model will fit the idiosyncrasies of the specific data set.

Thus

Thus, it will result in low bias but will yield high variance, as we change from one data set to another data set.

Furthermore

If N grows we can have a more complex model to be fitted which reduces bias and ensures low variance.

- However, N is always finite!!!

Similar to...

Curve Fitting

If, for example, the adopted model is complex (many parameters involved) with respect to the number N, the model will fit the idiosyncrasies of the specific data set.

Thus

Thus, it will result in low bias but will yield high variance, as we change from one data set to another data set.

Furthermore

If N grows we can have a more complex model to be fitted which reduces bias and ensures low variance.

- However, N is always finite!!!

Thus

You always need to compromise

However, you always have some a priori knowledge about the data

Allowing you to impose restrictions

Lowering the bias and the variance

Nevertheless

We have the following example to grasp better the bothersome **bias–variance dilemma**.

Thus

You always need to compromise

However, you always have some a priori knowledge about the data

Allowing you to impose restrictions

Lowering the bias and the variance

Nevertheless

We have the following example to grasp better the bothersome **bias–variance dilemma**.

Thus

You always need to compromise

However, you always have some a priori knowledge about the data

Allowing you to impose restrictions

Lowering the bias and the variance

Nevertheless

We have the following example to grasp better the bothersome **bias–variance dilemma**.

For this

Assume

The data is generated by the following function

$$
\begin{aligned}
y =& f(x) + \epsilon, \\
\epsilon \sim& \mathcal{N}\left(0, \sigma_\epsilon^2\right)
\end{aligned}
$$

We know that

The optimum regressor is $E[y|x] = f(x)$

Furthermore

Assume that the randomness in the different training sets, $\mathcal{D}$, is due to the y_i's (Affected by noise), while the respective points, x_i, are fixed.

For this

Assume

The data is generated by the following function

$$y = f(x) + \epsilon,$$
$$\epsilon \sim \mathcal{N}\left(0, \sigma_\epsilon^2\right)$$

We know that

The optimum regressor is $E[y|x] = f(x)$

Furthermore

Assume that the randomness in the different training sets, $\mathcal{D}$, is due to the y_i's (Affected by noise), while the respective points, x_i, are fixed.

For this

Assume

The data is generated by the following function

$$y = f(x) + \epsilon,$$
$$\epsilon \sim \mathcal{N}\left(0, \sigma_\epsilon^2\right)$$

We know that

The optimum regressor is $E[y|x] = f(x)$

Furthermore

Assume that the randomness in the different training sets, $\mathcal{D}$, is due to the y_i's (Affected by noise), while the respective points, x_i, are fixed.

Outline

Sampling the Space [2]

Imagine that $\mathcal{D} \subset [x_1, x_2]$ in which x lies

For example, you can choose $x_i = x_1 + \frac{x_2 - x_1}{N-1}(i-1)$ with $i = 1, 2, ..., N$

Case 1

Choose the estimate of $f(x)$, $g(x|\mathcal{D})$, to be independent of $\mathcal{D}$

For example, $g(x) = w_1 x + w_0$

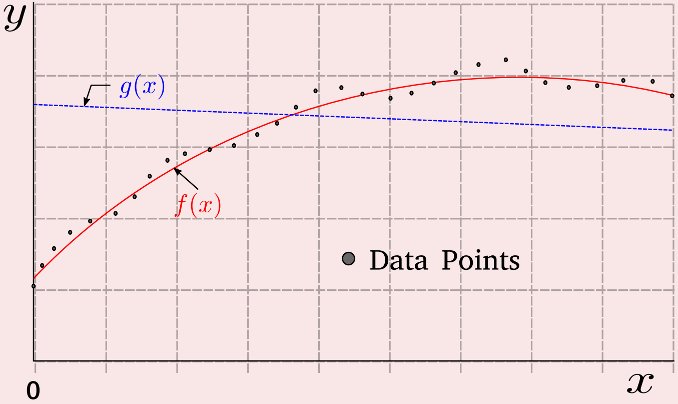
For example, the points are spread around $(x, f(x))$

Case 1

Choose the estimate of $f(x)$, $g(x|\mathcal{D})$, to be independent of $\mathcal{D}$

For example, $g(x) = w_1 x + w_0$

For example, the points are spread around $(x, f(x))$



Case 1

Since $g(x)$ is fixed

$$E_{\mathcal{D}}\left[g\left(x|\mathcal{D}\right)\right] = g\left(x|\mathcal{D}\right) \equiv g\left(x\right) \tag{4}$$

With

$$Var_{\mathcal{D}}\left[g\left(x|\mathcal{D}\right)\right] = 0 \tag{5}$$

On the other hand

Because $g(x)$ was chosen arbitrarily the expected bias must be large.

$$\underbrace{\left(E_{\mathcal{D}}\left[g\left(x|\mathcal{D}\right)\right] - E\left[y|x\right]\right)^2}_{BIAS} \tag{6}$$

Case 1

Since $g(x)$ is fixed

$$E_{\mathcal{D}}\left[g\left(x|\mathcal{D}\right)\right] = g\left(x|\mathcal{D}\right) \equiv g\left(x\right) \tag{4}$$

With

$$Var_{\mathcal{D}}\left[g\left(x|\mathcal{D}\right)\right] = 0 \tag{5}$$

On the other hand

Because $g(x)$ was chosen arbitrarily the expected bias must be large.

$$\underbrace{\left(E_{\mathcal{D}}\left[g\left(x|\mathcal{D}\right)\right] - E\left[y|x\right]\right)^2}_{BIAS} \tag{6}$$

Case 1

Since $g(x)$ is fixed

$$E_{\mathcal{D}}\left[g\left(x|\mathcal{D}\right)\right] = g\left(x|\mathcal{D}\right) \equiv g\left(x\right) \tag{4}$$

With

$$Var_{\mathcal{D}}\left[g\left(x|\mathcal{D}\right)\right] = 0 \tag{5}$$

On the other hand

Because $g(x)$ was chosen arbitrarily the expected bias must be large.

$$\underbrace{\left(E_{D}\left[g\left(\boldsymbol{x}|\mathcal{D}\right)\right] - E\left[y|\boldsymbol{x}\right]\right)^{2}}_{BIAS} \tag{6}$$

Case 2

In the other hand

Now, $g_1(x)$ corresponds to a polynomial of high degree so it can pass through each training point in $\mathcal{D}$.

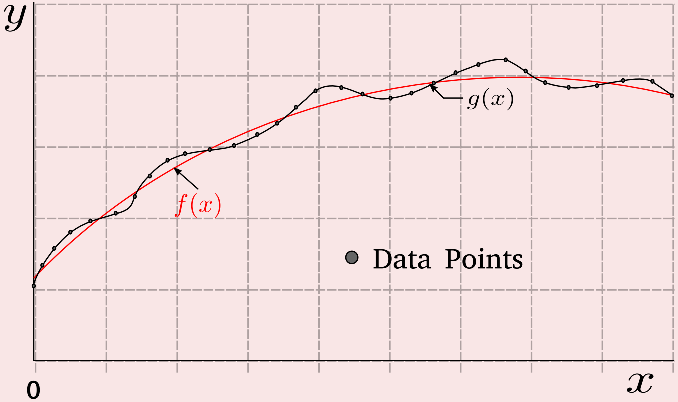
Example of $g_1(x)$

Case 2

In the other hand

Now, $g_1(x)$ corresponds to a polynomial of high degree so it can pass through each training point in $\mathcal{D}$.

Example of $g_1(x)$



Case 2

Due to the zero mean of the noise source

$$E_D\left[g_1\left(\boldsymbol{x}|\mathcal{D}\right)\right] = f\left(x\right) = E\left[y|x\right] \text{ for any } x = x_i \tag{7}$$

Remark: At the training points the bias is zero.

However, the variance increases

$$\begin{aligned} E_D\left[\left(g_1\left(\boldsymbol{x}|\mathcal{D}\right) - E_D\left[g_1\left(\boldsymbol{x}|\mathcal{D}\right)\right]\right)^2\right] &= E_D\left[\left(f\left(x\right) + \epsilon - f\left(x\right)\right)^2\right] \\ &= \sigma_\epsilon^2, \text{ for } x = x_i, i = 1, 2, \dots, N \end{aligned}$$

In other words

The bias becomes zero (or approximately zero) but the variance is now equal to the variance of the noise source.

Case 2

Due to the zero mean of the noise source

$$E_D\left[g_1\left(\boldsymbol{x}|\mathcal{D}\right)\right] = f\left(x\right) = E\left[y|x\right] \text{ for any } x = x_i \tag{7}$$

Remark: At the training points the bias is zero.

However the variance increases

$$\begin{aligned} E_D\left[\left(g_1\left(\boldsymbol{x}|\mathcal{D}\right) - E_D\left[g_1\left(\boldsymbol{x}|\mathcal{D}\right)\right]\right)^2\right] &= E_D\left[\left(f\left(x\right) + \epsilon - f\left(x\right)\right)^2\right] \\ &= \sigma_\epsilon^2, \text{ for } x = x_i, i = 1, 2, ..., N \end{aligned}$$

In other words

The bias becomes zero (or approximately zero) but the variance is now equal to the variance of the noise source.

Case 2

Due to the zero mean of the noise source

$$E_D\left[g_1\left(\boldsymbol{x}|\mathcal{D}\right)\right] = f\left(x\right) = E\left[y|x\right] \text{ for any } x = x_i \tag{7}$$

Remark: At the training points the bias is zero.

However the variance increases

$$\begin{aligned} E_D\left[\left(g_1\left(\boldsymbol{x}|\mathcal{D}\right) - E_D\left[g_1\left(\boldsymbol{x}|\mathcal{D}\right)\right]\right)^2\right] &= E_D\left[\left(f\left(x\right) + \epsilon - f\left(x\right)\right)^2\right] \\ &= \sigma_\epsilon^2, \text{ for } x = x_i, i = 1, 2, ..., N \end{aligned}$$

In other words

The bias becomes zero (or approximately zero) but the variance is now equal to the variance of the noise source.

Observations

First

Everything that has been said so far applies to both the regression and the classification tasks.

However

Mean squared error is not the best way to measure the power of a classifier.

Think about

A classifier that sends everything far away of the hyperplane!!! Away from the values $+-1$!!!

Observations

First

Everything that has been said so far applies to both the regression and the classification tasks.

However

Mean squared error is not the best way to measure the power of a classifier.

Think about

A classifier that sends everything far away of the hyperplane!!! Away from the values $+-1$!!!

Observations

First

Everything that has been said so far applies to both the regression and the classification tasks.

However

Mean squared error is not the best way to measure the power of a classifier.

Think about

A classifier that sends everything far away of the hyperplane!!! Away from the values $+-1$!!!

Outline

The house example (From Andrew Ng Course)

Imagine the following data set

Now assume that we use a regressor

For the fitting

$$\frac{1}{2}\sum_{i=1}^{N}\left(h_{\boldsymbol{\theta}}\left(x_i\right)-y_i\right)^2$$

We can then run one of our machine to see what minimize better the previous equation

Question: Did you notice that I did not impose any structure to $h_{w}\left(x\right)$?

Now assume that we use a regressor

For the fitting

$$\frac{1}{2}\sum_{i=1}^{N}\left(h_{\boldsymbol{\theta}}\left(x_i\right)-y_i\right)^2$$

We can then run one of our machine to see what minimize better the previous equation

Question: Did you notice that I did not impose any structure to $h_{\boldsymbol{w}}\left(x\right)$?

Then, First fitting

What about using $h_1(x) = \theta_0 + \theta_1 x + \theta_2 x^2$?

Second fitting

What about using $h_2(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 + \theta_5 x^5$?

Therefore, we have a problem

We get weird over fitting effects!!!

What do we do? What about minimizing the influence of $\theta_3, \theta_4, \theta_5$?

How do we do that?

$$\min_{\theta} \frac{1}{2} \sum_{i=1}^{N} \left(h_{\theta}\left(x_i\right) - y_i \right)^2$$

What about integrating those values to the cost function? Ideas

Therefore, we have a problem

We get weird over fitting effects!!!

What do we do? What about minimizing the influence of $\theta_3, \theta_4, \theta_5$?

How do we do that?

$$\min_{\boldsymbol{\theta}} \frac{1}{2} \sum_{i=1}^{N} \left(h_{\boldsymbol{\theta}}\left(x_i\right) - y_i\right)^2$$

What about integrating those values to the cost function? Ideas

Outline

We have

Regularization intuition is as follow

Small values for parameters $\theta_0, \theta_1, \theta_2, ..., \theta_n$

It implies

1. "Simpler" function
2. Less prone to overfitting

We have

Regularization intuition is as follow

Small values for parameters $\theta_0, \theta_1, \theta_2, ..., \theta_n$

It implies

1. "Simpler" function
2. Less prone to overfitting

We can do the previous idea for the other parameters

We can do the same for the other parameters

$$\min_{\boldsymbol{\theta}} \frac{1}{2}\sum_{i=1}^{N}\left(h_{\boldsymbol{\theta}}\left(x_i\right)-y_i\right)^2+\sum_{i=1}^{d}\lambda_i\theta_i^2 \tag{8}$$

However handling such many parameters can be so difficult

Combinatorial problem in reality!!!

We can do the previous idea for the other parameters

We can do the same for the other parameters

$$\min_{\boldsymbol{\theta}} \frac{1}{2} \sum_{i=1}^{N} \left(h_{\boldsymbol{\theta}}\left(x_i\right) - y_i\right)^2 + \sum_{i=1}^{d} \lambda_i \theta_i^2 \tag{8}$$

However handling such many parameters can be so difficult

Combinatorial problem in reality!!!

Better, we can

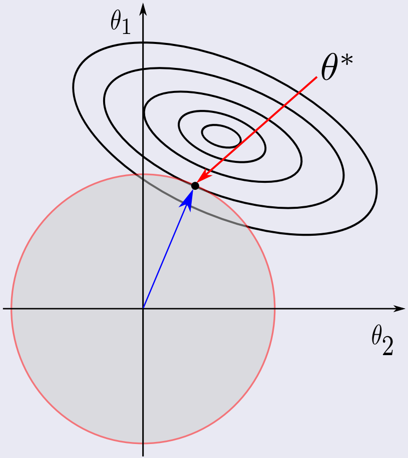
We better use the following

$$\min_{\boldsymbol{\theta}} \frac{1}{2} \sum_{i=1}^{N} \left(h_{\boldsymbol{\theta}}\left(x_i\right) - y_i\right)^2 + \lambda \sum_{i=1}^{d} \theta_i^2 \tag{9}$$

Graphically

Geometrically Equivalent to send our function to something quadratic

$$\frac{1}{2}\sum_{i=1}^{N}\left(h_{\boldsymbol{\theta}}\left(x_i\right)-y_i\right)^2+\lambda\sum_{i=1}^{d}\theta_i^2$$



Outline

Ridge Regression

Equation

$$\widehat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{w}} \left\{ \sum_{i=1}^{N} \left(y_i - \theta_0 - \sum_{j-1}^{d} x_{ij}\theta_j \right)^2 + \lambda \sum_{j=1}^{d} \theta_j^2 \right\}$$

Here

- $\lambda \geq 0$ is a complexity parameter that controls the amount of shrinkage

Ridge Regression

Equation

$$\widehat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{w}} \left\{ \sum_{i=1}^{N} \left(y_i - \theta_0 - \sum_{j-1}^{d} x_{ij}\theta_j \right)^2 + \lambda \sum_{j=1}^{d} \theta_j^2 \right\}$$

Here

- $\lambda \geq 0$ is a complexity parameter that controls the amount of shrinkage

Therefore

The Larger $\lambda \geq 0$

- The coefficients are shrunk toward zero (and each other).

This is also used in Neural Networks

- where it is known as weight decay

Therefore

The Larger $\lambda \geq 0$

- The coefficients are shrunk toward zero (and each other).

This is also used in Neural Networks

- where it is known as weight decay

This is also can be written

Optimization Solution

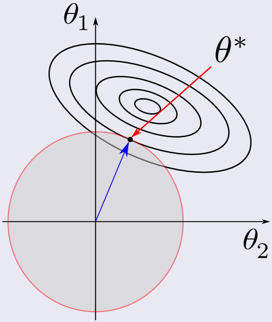
$$\arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \left(y_i - \theta_0 - \sum_{j-1}^{d} x_{ij}\theta_j \right)^2$$

subject to $\sum_{j=1}^{d} \theta_j^2 < t$

Graphically

Geometrically Equivalent to

$$\begin{gathered}\arg\min \ \sum_{i=1}^{N}\left(y_i - \boldsymbol{x}_i^T\boldsymbol{\theta}\right)^2 \\ \text{subject to } \sum_{i=1}^{d+1}\theta_i^2 < t\end{gathered}$$



Outline

Least Absolute Shrinkage and Selection Operator (LASSO)

It was introduced by Robert Tibshirani in 1996 based on Leo Breiman's nonnegative garrote

$$\widehat{\boldsymbol{\theta}}^{garrote} = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \left(y_i - \theta_0 - \sum_{j=1}^{d} x_{ij}\theta_j \right)^2 + N\lambda \sum_{j=1}^{d} \theta_j$$

s.t. $\theta_j > 0 \ \forall j$

This is quite derivable

However, Tibshirani realized that you could get a more flexible model by using the absolute value at the constraint!!!

Robert Tibshirani proposed the use of the L_1 norm

$$\|\boldsymbol{\theta}\|_1 = \sum_{i=1}^{d} |\theta_i|$$

Least Absolute Shrinkage and Selection Operator (LASSO)

It was introduced by Robert Tibshirani in 1996 based on Leo Breiman's nonnegative garrote

$$\widehat{\boldsymbol{\theta}}^{garrote} = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \left(y_i - \theta_0 - \sum_{j=1}^{d} x_{ij}\theta_j \right)^2 + N\lambda \sum_{j=1}^{d} \theta_j$$

s.t. $\theta_j > 0 \ \forall j$

This is quite derivable

However, Tibshirani realized that you could get a more flexible model by using the absolute value at the constraint!!!

Robert Tibshirani proposed the use of the L_1 norm

$$\|\boldsymbol{\theta}\|_1 = \sum_{i=1}^{d} |\theta_i|$$

Least Absolute Shrinkage and Selection Operator (LASSO)

It was introduced by Robert Tibshirani in 1996 based on Leo Breiman's nonnegative garrote

$$\widehat{\boldsymbol{\theta}}^{garrote} = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \left(y_i - \theta_0 - \sum_{j=1}^{d} x_{ij}\theta_j \right)^2 + N\lambda \sum_{j=1}^{d} \theta_j$$

$$\text{s.t. } \theta_j > 0 \ \forall j$$

This is quite derivable

However, Tibshirani realized that you could get a more flexible model by using the absolute value at the constraint!!!

Robert Tibshirani proposed the use of the L_1 norm

$$\left\|\boldsymbol{\theta}\right\|_1 = \sum_{i=1}^{d} |\theta_i|$$

The Final Optimization Problem

LASSO

$$\widehat{\boldsymbol{\theta}}^{LASSO} = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \left(y_i - \theta_0 - \sum_{j=1}^{d} x_{ij}\theta_j \right)^2$$

$$\text{s.t. } \sum_{i=1}^{d} |\theta_i| \leq t$$

This is not derivable

More advanced methods are necessary to solve this problem!!!

The Final Optimization Problem

LASSO

$$\widehat{\boldsymbol{\theta}}^{LASSO} = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \left(y_i - \theta_0 - \sum_{j=1}^{d} x_{ij}\theta_j \right)^2$$

$$\text{s.t.} \quad \sum_{i=1}^{d} |\theta_i| \leq t$$

This is not derivable

More advanced methods are necessary to solve this problem!!!

The Lagrangian Version

The Lagrangian

$$\widehat{\boldsymbol{\theta}}^{LASSO} = \arg\min_{\boldsymbol{\theta}} \left\{ \sum_{i=1}^{N} \left(y_i - \boldsymbol{x}^T \boldsymbol{\theta} \right)^2 + \lambda \sum_{i=1}^{d} |\theta_i| \right\}$$

However

You have other regularizations as $\|\theta\|_2 = \sqrt{\sum_{i=1}^{d} |\theta_i|^2}$

The Lagrangian Version

The Lagrangian

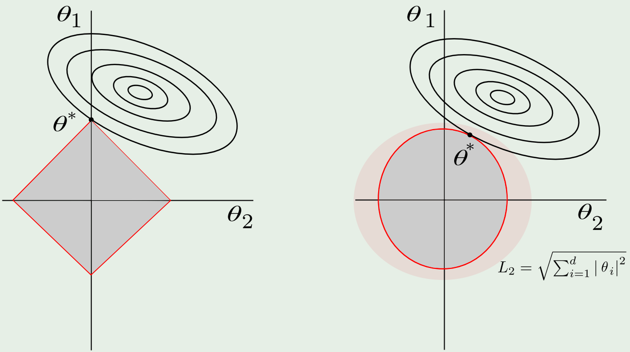
$$\widehat{\boldsymbol{\theta}}^{LASSO} = \arg\min_{\boldsymbol{\theta}} \left\{ \sum_{i=1}^{N} \left(y_i - \boldsymbol{x}^T \boldsymbol{\theta} \right)^2 + \lambda \sum_{i=1}^{d} |\theta_i| \right\}$$

However

You have other regularizations as $\|\boldsymbol{\theta}\|_2 = \sqrt{\sum_{i=1}^{d} |\theta_i|^2}$

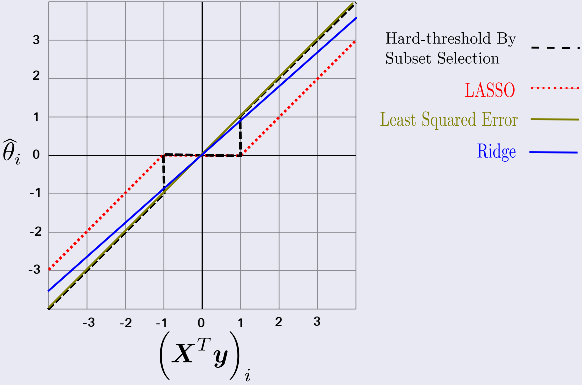
Graphically

Yes the circle defined as $\|\boldsymbol{\theta}\|_2 = \sqrt{\sum_{i=1}^{d} |\theta_i|^2}$



For Example

In the Case of $\boldsymbol{X}$ is a Orthogonal Matrix



The seminal paper by Robert Tibshirani

An initial study of this regularization can be seen in

"Regression Shrinkage and Selection via the LASSO" by Robert Tibshirani - 1996

Outline

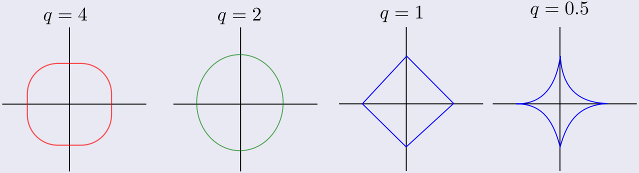
Furthermore

We can generalize ridge regression and the lasso, and view them as Bayes estimates

$$\widehat{\boldsymbol{\theta}}^{LASSO} = \arg\min_{\boldsymbol{w}} \left\{ \sum_{i=1}^{N} \left(y_i - L\left(\boldsymbol{x}_i, \boldsymbol{\theta}\right)\right)^2 + \lambda \sum_{i=1}^{d} |\theta_i|^q \right\} \text{ with } q \geq 0$$

For Example

We have when $d = 2$

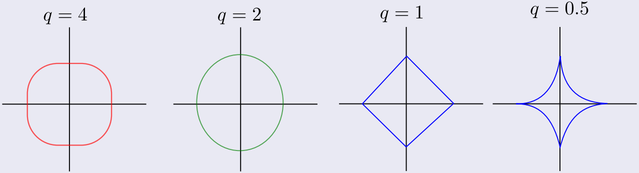


Here, when $q > 1$

- You are having a derivable Lagrangian, but you lose the LASSO properties

For Example

We have when $d = 2$



Here, when $q > 1$

- You are having a derivable Lagrangian, but you lose the LASSO properties

Therefore

Zou and Hastie (2005) introduced the elastic-net penalty [3]

$$\lambda \sum_{i=1}^{d} \left\{ \alpha \theta_i^2 + (1-\alpha) \left| \theta_i \right| \right\}$$

This is Basically

- A Compromise Between the Ridge and LASSO.

Therefore

Zou and Hastie (2005) introduced the elastic-net penalty [3]

$$\lambda \sum_{i=1}^{d} \left\{ \alpha \theta_i^2 + (1-\alpha) \left| \theta_i \right| \right\}$$

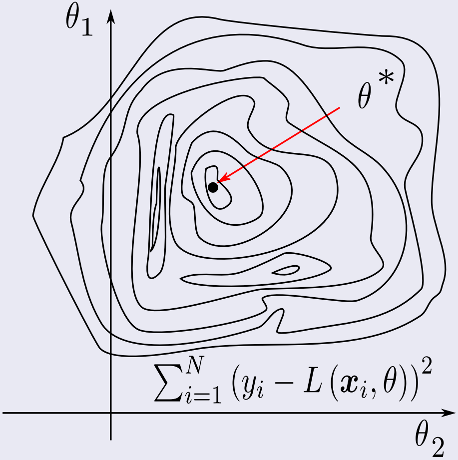
This is Basically

- A Compromise Between the Ridge and LASSO.

Outline

What can be done?

Remember that our optimization Landscape is highly variable



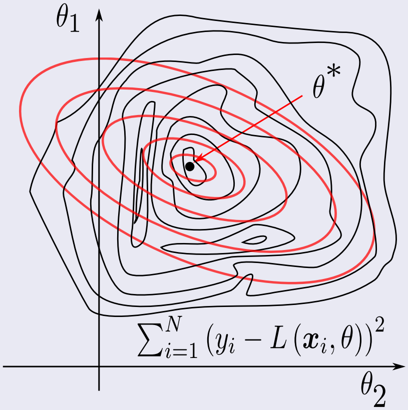
Overfitting?

Basically (Intuition)

$$(y_i - L(\boldsymbol{x}_i, \theta))^2 = 0 \text{ for } i \in Training$$
$$(y_j - L(\boldsymbol{x}_i, \theta))^2 \gg 0 \text{ for } i \in Validation$$

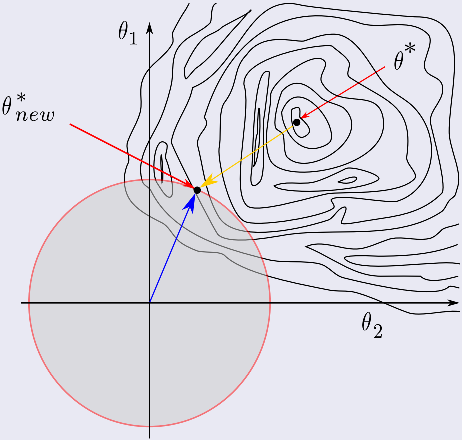
We do not want too much simplification

Look at this



Basically this simplification is due to the constrained optimization landscape

Basically our constraint is too Euclidean for Optimization Landscape



Well-Posed Problem

Definition by Hadamard (Circa 1902)

- Models of physical phenomenas should have the following properties
  1. A solution exists,
  2. The solution is unique,
  3. The solution's behavior changes continuously with the initial conditions.

Any other problem that fails in any of this conditions

- It is considered an Ill-Posed Problem.

Well-Posed Problem

Definition by Hadamard (Circa 1902)

- Models of physical phenomenas should have the following properties
  1. A solution exists,
  2. The solution is unique,
  3. The solution's behavior changes continuously with the initial conditions.

Any other problem that fails in any of this conditions

- It is considered an Ill-Posed Problem.

It seems to be that

The Deep Learners are highly ill-posed problems

- Ridge and LASSO have two possible effects

Too much simplification

- The Deep Learners losses power of representation.
  - Weights are eliminated

The constraints forces the $\theta's$

- They are forced to live in a too smooth optimization landscape

It seems to be that

The Deep Learners are highly ill-posed problems

- Ridge and LASSO have two possible effects

Too much simplification

- The Deep Learners losses power of representation.
  - Weights are eliminated

The constraints forces the $\theta's$

- They are forced to live in a too smooth optimization landscape

It seems to be that

The Deep Learners are highly ill-posed problems

- Ridge and LASSO have two possible effects

Too much simplification

- The Deep Learners losses power of representation.
  - Weights are eliminated

The constraints forces the $\theta's$

- They are forced to live in a too smooth optimization landscape

Outline

DeVris and Taylor [5]

For many years

- Dataset augmentation has been a standard regularization technique used to reduce overfitting while training supervised learning models

For Example, LeCun et al. [4] when training the LeNet5

- They applied a series of transformations to the input images in order to improve the robustness of the model.

Unfortunately

- Dataset augmentation is not as straightforward to apply in all domains as it is for images.

DeVris and Taylor [5]

For many years

- Dataset augmentation has been a standard regularization technique used to reduce overfitting while training supervised learning models

For Example, LeCun et al. [4] when training the LeNet5

- They applied a series of transformations to the input images in order to improve the robustness of the model.

Unfortunately

- Dataset augmentation is not as straightforward to apply in all domains as it is for images.

DeVris and Taylor [5]

For many years

- Dataset augmentation has been a standard regularization technique used to reduce overfitting while training supervised learning models

For Example, LeCun et al. [4] when training the LeNet5

- They applied a series of transformations to the input images in order to improve the robustness of the model.

Unfortunately

- Dataset augmentation is not as straightforward to apply in all domains as it is for images.

For Example

In voice detection, adding

1. Gaussian noise to the input,
2. Shifting the pitch of the audio signal,
3. Time stretching,
4. Varying the loudness of the audio signal,
5. Applying random frequency filters,
6. Interpolating between samples in input space.

For Example

In voice detection, adding

1. Gaussian noise to the input,
2. Shifting the pitch of the audio signal,
3. Time stretching,
4. Varying the loudness of the audio signal,
5. Applying random frequency filters,
6. Interpolating between samples in input space.

Actually, only the following techniques worked out

- Pitch shifting and random frequency filtering

For Example

In voice detection, adding

1. Gaussian noise to the input,
2. Shifting the pitch of the audio signal,
3. Time stretching,
4. Varying the loudness of the audio signal,
5. Applying random frequency filters,
6. Interpolating between samples in input space.

Actually, only the following techniques worked out

- Pitch shifting and random frequency filtering

For Example

In voice detection, adding

1. Gaussian noise to the input,
2. Shifting the pitch of the audio signal,
3. Time stretching,
4. Varying the loudness of the audio signal,
5. Applying random frequency filters,
6. Interpolating between samples in input space.

Actually, only the following techniques worked out

- Pitch shifting and random frequency filtering

For Example

In voice detection, adding

1. Gaussian noise to the input,
2. Shifting the pitch of the audio signal,
3. Time stretching,
4. Varying the loudness of the audio signal,
5. Applying random frequency filters,
6. Interpolating between samples in input space.

Actually, only the following techniques worked out

- Pitch shifting and random frequency filtering

For Example

In voice detection, adding

1. Gaussian noise to the input,
2. Shifting the pitch of the audio signal,
3. Time stretching,
4. Varying the loudness of the audio signal,
5. Applying random frequency filters,
6. Interpolating between samples in input space.

Actually, only the following techniques worked out

- Pitch shifting and random frequency filtering

For Example

In voice detection, adding

1. Gaussian noise to the input,
2. Shifting the pitch of the audio signal,
3. Time stretching,
4. Varying the loudness of the audio signal,
5. Applying random frequency filters,
6. Interpolating between samples in input space.

Actually, only the following techniques worked out

- Pitch shifting and random frequency filtering

DeVris and Taylor [5]

They did something different

- First learning a data representation
- Then applying transformations to samples mapped to that representation.

They hypothesized

- Due to manifold unfolding in feature space, simple transformations applied to **encoded** rather than raw inputs
  - They will result in more plausible synthetic data.

DeVris and Taylor [5]

They did something different

- First learning a data representation
- Then applying transformations to samples mapped to that representation.

They hypothesized

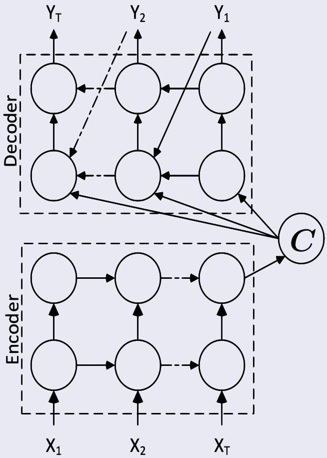
- Due to manifold unfolding in feature space, simple transformations applied to **encoded** rather than raw inputs
  - They will result in more plausible synthetic data.

Outline

Decoder/Encoder Part

We have a Decoder and Encoder Architecture

$$\boldsymbol{C} = \text{Context Vector}$$



Here

We have a K-coding symbol set

- The Encoder and Decoder are based in a novel hidden unit.

We have the following configuration per row element j

$$r_j = \sigma\left(\left[\mathbf{W}_r \mathrm{x}\right]_j + \left[\mathbf{U}_r \mathrm{h}_{t-1}\right]_j\right) \leftarrow \text{ Reset Gate}$$

- σ a sigmoid function

The Update gate

$$z_j = \sigma\left(\left[\mathbf{W}_z \mathrm{x}\right]_j + \left[\mathbf{U}_z \mathrm{h}_{t-1}\right]_j\right)$$

Here

We have a K-coding symbol set

- The Encoder and Decoder are based in a novel hidden unit.

We have the following configuration per row element j

$$r_j = \sigma\left([\boldsymbol{W}_r \mathrm{x}]_j + [\boldsymbol{U}_r \mathrm{h}_{t-1}]_j\right) \leftarrow \text{ Reset Gate}$$

- σ a sigmoid function

The Update gate

$$z_j = \sigma\left([\boldsymbol{W}_z \mathrm{x}]_j + [\boldsymbol{U}_z \mathrm{h}_{t-1}]_j\right)$$

Here

We have a K-coding symbol set

- The Encoder and Decoder are based in a novel hidden unit.

We have the following configuration per row element j

$$r_j = \sigma\left([\boldsymbol{W}_r \mathrm{x}]_j + [\boldsymbol{U}_r \mathrm{h}_{t-1}]_j\right) \leftarrow \text{ Reset Gate}$$

- σ a sigmoid function

The Update gate

$$z_j = \sigma\left([\boldsymbol{W}_z \mathrm{x}]_j + [\boldsymbol{U}_z \mathrm{h}_{t-1}]_j\right)$$

Where

The Activation Gate update

$$h_j^t = z_j h_j^{t-1} + (1 - z_j)\,\widetilde{h}_j^t$$

- Where $\widetilde{h}_j^t = \phi\left([\boldsymbol{W}\mathrm{x}]_j + [\boldsymbol{U}(\boldsymbol{r} \odot \boldsymbol{h}_{t-1})]_j\right)$

In this formulation

- When the reset gate is close to 0, the hidden state is forced to ignore the previous hidden state!!!

Where

The Activation Gate update

$$h_j^t = z_j h_j^{t-1} + (1 - z_j)\,\widetilde{h}_j^t$$

- Where $\widetilde{h}_j^t = \phi\left([\boldsymbol{W}\mathrm{x}]_j + [\boldsymbol{U}(\boldsymbol{r} \odot \boldsymbol{h}_{t-1})]_j\right)$

In this formulation

- When the reset gate is close to 0, the hidden state is forced to ignore the previous hidden state!!!

Finally, at output

We have a probability of producing a symbol of a set of

$$p\left(y_t | y_{t-1}, ..., y_1, \boldsymbol{c}\right) = \frac{\exp\left(W_o \boldsymbol{h}_t + U_o y_{t-1} + \boldsymbol{c}_{t-1}\right)}{\sum_{j=1}^{K} \exp\left(W_j \boldsymbol{h}_t + U_o y_{t-1} + \boldsymbol{c}_{t-1}\right)}$$

Then

- The encoder learns to predict the next symbol x_t based in the previous $x_{t-1}, x_{t-2}, ..., x_1$ by using the maximization

$$\max_{\theta} \frac{1}{N} \sum_{n=1}^{N} p\left(y_n | x_n\right)$$

Finally, at output

We have a probability of producing a symbol of a set of

$$p\left(y_t|y_{t-1},\ldots,y_1,\boldsymbol{c}\right)=\frac{\exp\left(W_o\boldsymbol{h}_t+U_oy_{t-1}+\boldsymbol{c}_{t-1}\right)}{\sum_{j=1}^{K}\exp\left(W_j\boldsymbol{h}_t+U_oy_{t-1}+\boldsymbol{c}_{t-1}\right)}$$

Then

- The encoder learns to predict the next symbol x_t based in the previous $x_{t-1},x_{t-2},\ldots,x_1$ by using the maximization

$$\max_{\theta}\frac{1}{N}\sum_{n=1}^{N}p\left(\boldsymbol{y}_n|\boldsymbol{x}_n\right)$$

Here, the Noise

Generate noise by drawing from

- A Gaussian distribution with **zero mean** and **per-element standard deviation** calculated across all context vectors in the dataset

$$c_i' = c_i + \gamma X,\ X \sim N\left(0, \sigma_i^2\right)$$

We can generate this using a more direct approach

- For each sample in the dataset, we find its K nearest neighbors in feature space, then

$$c' = (c_k - c_j)\lambda + c_j$$

- $\lambda = 0.5$

Here, the Noise

Generate noise by drawing from

- A Gaussian distribution with **zero mean** and **per-element standard deviation** calculated across all context vectors in the dataset

$$c_i' = c_i + \gamma X, \; X \sim N\left(0, \sigma_i^2\right)$$

We can generate this using a more direct approach

- For each sample in the dataset, we find its K nearest neighbors in feature space, then

$$\boldsymbol{c}' = \left(\boldsymbol{c}_k - \boldsymbol{c}_j\right)\lambda + \boldsymbol{c}_j$$

- $\lambda = 0.5$

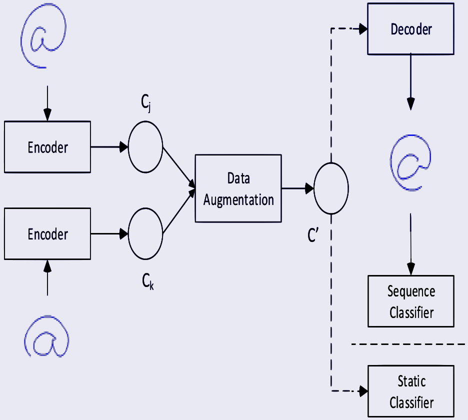
Then

Once this new augmented context vectors with noise are ready

- As input for a learning task,
- They can be decoded to generate new sequences

Finally, we have

The following architecture



Outline

Outline

Regularization in Deep Forward

In Layers of a Deep Forward

- We want to find and estimation x_t^r to an input at $x_0 \in \mathbb{R}^d$ in layer t satisfying

$$\sigma\left(A_t^T x_t\right) = y_{t+1}$$

Regularization in Deep Forward

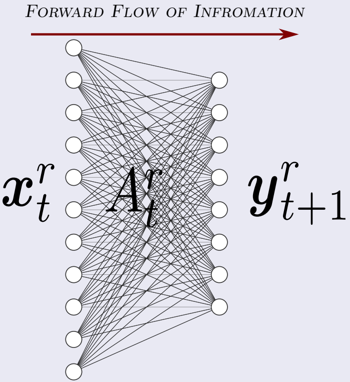
In Layers of a Deep Forward

- We want to find and estimation $\boldsymbol{x}_t^r$ to an input at $\boldsymbol{x}_0 \in \mathbb{R}^d$ in layer t satisfying

$$\sigma\left(A_t^r \boldsymbol{x}_t\right) = \boldsymbol{y}_{t+1}$$

We can see this

A flow of information



In all such situations

The vector $\boldsymbol{x}_t$ is generated by $\boldsymbol{y}_{t+1}$ using back-propagation

$$A_t^r = A_t^{r-1} - \eta \frac{\partial L\left(A_T^{r-1}, ..., A_0^{r-1}, x_0\right)}{\partial A_t^{r-1}}$$

It is usually a meaningless bad approximation

- to x^* optimal at layer t for all possible inputs x_0's.

In all such situations

The vector $\boldsymbol{x}_t$ is generated by $\boldsymbol{y}_{t+1}$ using back-propagation

$$A_t^r = A_t^{r-1} - \eta \frac{\partial L\left(A_T^{r-1}, ..., A_0^{r-1}, x_0\right)}{\partial A_t^{r-1}}$$

It is usually a meaningless bad approximation

- to $\boldsymbol{x}^*$ optimal at layer t for all possible inputs $\boldsymbol{x}_0's$.

Then

We can see the Deep Forward Network as

$$y_T = \sigma\left(A_T \sigma\left(A_{T-1} \sigma\left(A_{T-2}\left(\ldots \sigma\left(A_0 x_0\right)\right)\right)\right)\right)$$

Here

- The σ is applied to the generated vectors point wise...

Then

We can see the Deep Forward Network as

$$y_T = \sigma\left(A_T \sigma\left(A_{T-1} \sigma\left(A_{T-2}\left(\ldots\sigma\left(A_0 x_0\right)\right)\right)\right)\right)$$

Here

- The σ is applied to the generated vectors point wise...

The Jacobian of the Gradient Descent

Here, we assume a Least Squared Error cost function

$$\frac{\partial L\left(A_T^{r-1},...,A_0^{r-1},x_0^i\right)}{\partial A_t^{r-1}} = -\left(z^i - y_T\right)\times\sigma'\left(A_{T-1}^r \boldsymbol{x}_{T-1}\right)\times\frac{\partial A_{T-1}^r \boldsymbol{x}_{T-1}}{\partial \boldsymbol{x}_{T-1}}\times...\times\sigma'\left(A_t^r \boldsymbol{x}_t\right)\times\frac{\partial A_t^r \boldsymbol{x}}{\partial \boldsymbol{x}_t}$$

Where

$$\sigma'\left(A_k^r x_k\right) = \begin{pmatrix} \sigma'\left(a_{1k}^r x_k\right) & 0 & \cdots & 0 \\ 0 & \sigma'\left(a_{2k}^r x_k\right) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma'\left(a_{Mk}^r x_k\right) \end{pmatrix}$$

The Jacobian of the Gradient Descent

Here, we assume a Least Squared Error cost function

$$\frac{\partial L\left(A_T^{r-1},...,A_0^{r-1},x_0^i\right)}{\partial A_t^{r-1}} = -\left(z^i - y_T\right)\times\sigma'\left(A_{T-1}^r \boldsymbol{x}_{T-1}\right)\times\frac{\partial A_{T-1}^r \boldsymbol{x}_{T-1}}{\partial \boldsymbol{x}_{T-1}}\times...\times\sigma'\left(A_t^r \boldsymbol{x}_t\right)\times\frac{\partial A_t^r \boldsymbol{x}}{\partial \boldsymbol{x}_t}$$

Where

$$\sigma'\left(A_k^r \boldsymbol{x}_k\right) = \begin{pmatrix} \sigma'\left(\boldsymbol{a}_{1k}^r \boldsymbol{x}_k\right) & 0 & \cdots & 0 \\ 0 & \sigma'\left(\boldsymbol{a}_{2k}^r \boldsymbol{x}_k\right) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma'\left(\boldsymbol{a}_{Mk}^r \boldsymbol{x}_k\right) \end{pmatrix}$$

What will happen in the following situation?

Imagine that $A_k' s$ are diagonal matrix

$$A_k^r = \begin{pmatrix} a_{1k} & 0 & \cdots & 0 \\ 0 & a_{2k} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{Mk} \end{pmatrix}$$

Therefore, we have

$$\sigma'\left(A_k^r x_k\right) = \begin{pmatrix} \sigma'\left(a_{1k}^r x_{1k}\right) & 0 & \cdots & 0 \\ 0 & \sigma'\left(a_{2k}^r x_{2k}\right) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma'\left(a_{Mk}^r x_{2k}\right) \end{pmatrix}$$

What will happen in the following situation?

Imagine that $A_k^{\prime}s$ are diagonal matrix

$$A_k^r = \begin{pmatrix} a_{1k} & 0 & \cdots & 0 \\ 0 & a_{2k} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{Mk} \end{pmatrix}$$

Therefore, we have

$$\sigma' \left(A_k^r \boldsymbol{x}_k \right) = \begin{pmatrix} \sigma' \left(a_{1k}^r x_{1k} \right) & 0 & \cdots & 0 \\ 0 & \sigma' \left(a_{2k}^r x_{2k} \right) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma' \left(a_{Mk}^r x_{2k} \right) \end{pmatrix}$$

Then, we have that

First

$$\sigma'\left(A^r_{T-1}\boldsymbol{x}_{T-1}\right)\times\frac{\partial A^r_{T-1}\boldsymbol{x}_{T-1}}{\partial\boldsymbol{x}_{T-1}}\times\ldots\times\sigma'\left(A^r_t\boldsymbol{x}_t\right)\times\frac{\partial A^r_t\boldsymbol{x}_t}{\partial\boldsymbol{x}_t}=*$$

Then, we have that

$$*=\begin{pmatrix}\prod_{k=T-1}^{t}\sigma'\left(a^r_{1k}x_{1k}\right)a_{1k} & 0 & \cdots & 0\\ 0 & \prod_{k=T-1}^{t}\sigma'\left(a^r_{2k}x_{2k}\right)a_{2k} & \cdots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \cdots & \prod_{k=T-1}^{t}\sigma'\left(a^r_{Mk}x_{2k}\right)a_{2k}\end{pmatrix}$$

Then, we have that

First

$$\sigma'\left(A^r_{T-1}\boldsymbol{x}_{T-1}\right)\times\frac{\partial A^r_{T-1}\boldsymbol{x}_{T-1}}{\partial \boldsymbol{x}_{T-1}}\times ... \times\sigma'\left(A^r_t\boldsymbol{x}_t\right)\times\frac{\partial A^r_t\boldsymbol{x}_t}{\partial \boldsymbol{x}_t}=*$$

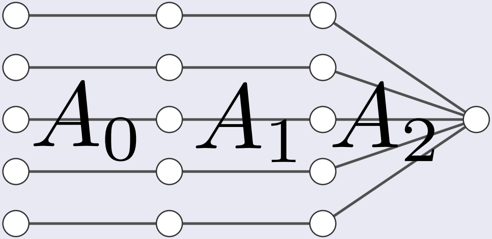
Then, we have that

$$*=\begin{pmatrix} \prod_{k=T-1}^{t}\sigma'\left(a^r_{1k}x_{1k}\right)a_{1k} & 0 & \cdots & 0 \\ 0 & \prod_{k=T-1}^{t}\sigma'\left(a^r_{2k}x_{2k}\right)a_{2k} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \prod_{k=T-1}^{t}\sigma'\left(a^r_{Mk}x_{2k}\right)a_2 \end{pmatrix}$$

Actually

Choosing Matrices in such way

- It is like a heavy simplification of the Deep Forward Network



Something happens with the LASSO and Ridge

At the top of the Optimization Cost Function

- We do not know how such shallow regularization can affect the Neural Network

However, we could do the following at each layer

Properties

Something happens with the LASSO and Ridge

At the top of the Optimization Cost Function

- We do not know how such shallow regularization can affect the Neural Network

However, we could do the following at each layer

Properties

Something happens with the LASSO and Ridge

At the top of the Optimization Cost Function

- We do not know how such shallow regularization can affect the Neural Network

However, we could do the following at each layer

Properties

Therefore

So heavy regularization

- It can not be a so good idea...

We need a new way of doing stuff

- For example, we could do the following

Therefore

So heavy regularization

- It can not be a so good idea...

We need a new way of doing stuff

- For example, we could do the following

Outline

Dropout

It was introduced by Hinton and Google [6]

- To avoid the problem of over-fitting

You can see it as a regularization

- From [7] "Dropout training as adaptive regularization" by Wager et al.

Dropout

It was introduced by Hinton and Google [6]

- To avoid the problem of over-fitting

You can see it as a regularization

- From [7] "Dropout training as adaptive regularization" by Wager et al.

Srivastava et al.

He comments that with unlimited computations

- "the best way to "regularize" a fixed-sized model is to average the predictions of all possible settings of the parameters"

Something like Boosting [1]

- By Using simpler and smaller models

Srivastava et al.

He comments that with unlimited computations

- "the best way to "regularize" a fixed-sized model is to average the predictions of all possible settings of the parameters"

Something like Boosting [1]

- By Using simpler and smaller models

Problem

We have Deep Architectures with thousands of parameters and hyperparameters

- Therefore, we have a problem!!! We need to solve this in some way!!!

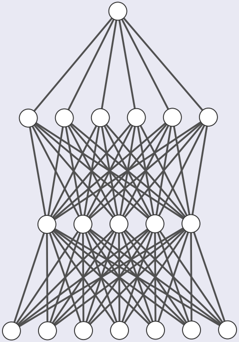
What if we fix our architecture

Problem

We have Deep Architectures with thousands of parameters and hyperparameters

- Therefore, we have a problem!!! We need to solve this in some way!!!

What if we fix our architecture



How it works?

You have forward layers

$$
\begin{aligned}
z_i^{l+1} &= W_i^{l+1}\boldsymbol{x}^l + b_i^{l+1} \\
x_i^{l+1} &= \sigma\left(z_i^{l+1}\right)
\end{aligned}
$$

With dropout, the feed-forward operation becomes

$$
\begin{aligned}
r_j^l &\sim Bernoulli\left(p\right) \\
\widetilde{\boldsymbol{x}}^l &= \boldsymbol{r}^l \odot \boldsymbol{x}^l \\
z_i^{l+1} &= W_i^{l+1}\widetilde{\boldsymbol{x}}^l + b_i^{l+1} \\
x_i^{l+1} &= \sigma\left(z_i^{l+1}\right)
\end{aligned}
$$

How it works?

You have forward layers

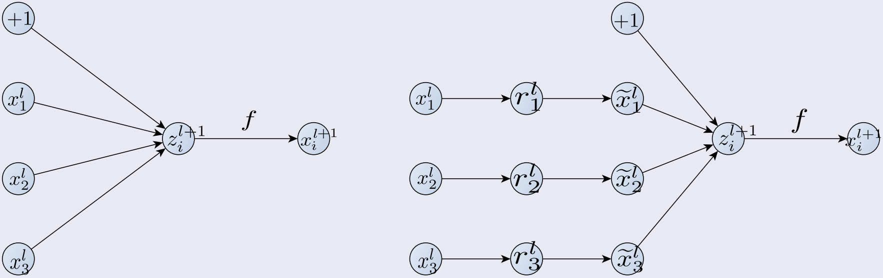
$$
\begin{aligned}
z_i^{l+1} &= W_i^{l+1}\boldsymbol{x}^l + b_i^{l+1}\\
x_i^{l+1} &= \sigma\left(z_i^{l+1}\right)
\end{aligned}
$$

With dropout, the feed-forward operation becomes

$$
\begin{aligned}
r_j^l &\sim Bernoulli\left(p\right)\\
\widetilde{\boldsymbol{x}}^l &= \boldsymbol{r}^l \odot \boldsymbol{x}^l\\
z_i^{l+1} &= W_i^{l+1}\widetilde{\boldsymbol{x}}^l + b_i^{l+1}\\
x_i^{l+1} &= \sigma\left(z_i^{l+1}\right)
\end{aligned}
$$

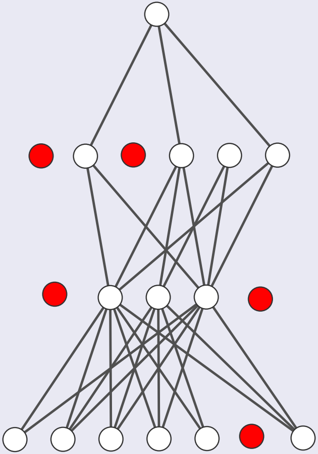
The Network

It looks like a series of gates



Then, we erase randomly connections through the network

We generate sparser version of the original one



Then assuming a Multilayer Perceptron

We have the following Architecture without bias to simplify with a single output

$$\min \frac{1}{N}\sum_{i=1}^{N}(z_i - t_i)^2$$

$$z_i = \sigma_1\left(W_{oh}\boldsymbol{y}_i\right)$$

$$\boldsymbol{y}_i = \sigma_2\left(W_{hi}\boldsymbol{x}_i\right)$$

Then, we get the following network after the sampling

$$L\left(W_{oh}, W_{hI}\right) = (t - z)^2$$

$$z = \sigma_1\left(W_{oh}\left(\boldsymbol{r}^2 \odot \boldsymbol{y}\right)\right)$$

$$\boldsymbol{y} = \sigma_2\left(W_{hI}\left(\boldsymbol{r}^1 \odot \boldsymbol{x}\right)\right)$$

Then assuming a Multilayer Perceptron

We have the following Architecture without bias to simplify with a single output

$$\min \frac{1}{N} \sum_{i=1}^{N} (z_i - t_i)^2$$
$$z_i = \sigma_1 \left(W_{oh} \boldsymbol{y}_i \right)$$
$$\boldsymbol{y}_i = \sigma_2 \left(W_{hi} \boldsymbol{x}_i \right)$$

Then, we get the following network after the sampling

$$\begin{aligned} L\left(W_{oh}, W_{hI}\right) = (t - z)^2 \\ z = \sigma_1 \left(W_{oh} \left(\boldsymbol{r}^2 \odot \boldsymbol{y} \right) \right) \\ \boldsymbol{y} = \sigma_2 \left(W_{hI} \left(\boldsymbol{r}^1 \odot \boldsymbol{x} \right) \right) \end{aligned}$$

Then, we have that

The Backpropagation at hidden weights

$$\frac{\partial L}{\partial W_{oh}} = -2\left(t - z\right) \times \frac{\partial \sigma_1'\left(net_{oh}\right)}{\partial net_{oh}} \times \left(\boldsymbol{r}^2 \odot \boldsymbol{y}\right)$$

Basically

$$\left(W_{oh}^{t+1}\right)_j = \begin{cases} \left(W_{oh}^{t}\right)_j + \eta 2\left(t - z\right) \times \frac{\partial \sigma_1'\left(net_{oh}\right)}{\partial net_{oh}} \left(\boldsymbol{y}\right)_j & \text{if } r_j = 1 \\ \left(W_{oh}^{t}\right)_j & \text{if } r_j = 0 \end{cases}$$

Then, we have that

The Backpropagation at hidden weights

$$\frac{\partial L}{\partial W_{oh}} = -2\left(t-z\right)\times\frac{\partial\sigma_1'\left(net_{oh}\right)}{\partial net_{oh}}\times\left(\boldsymbol{r}^2\odot\boldsymbol{y}\right)$$

Basically

$$\left(W_{oh}^{t+1}\right)_j = \begin{cases}\left(W_{oh}^{t}\right)_j + \eta 2\left(t-z\right)\times\frac{\partial\sigma_1'\left(net_{oh}\right)}{\partial net_{oh}}\left(\boldsymbol{y}\right)_j & \text{if } r_j = 1\\ \left(W_{oh}^{t}\right)_j & \text{if } r_j = 0\end{cases}$$

However, At Testing

There are a exponential number of possible sparse networks

- A neural net with n units, can be seen as a collection of 2^n possible thinned neural networks.

Something Notable

- These networks all share weights so that the total number of parameters is still $O(n^2)$, or less.

Problem, we cannot average such amount of sub-networks

- We average over the different passes to obtain a p for each node in the network
  - Meaning the probability of being active in the network.

$$p_{ijk} = \frac{\#\text{of subnets wehre node } ijk \text{ was active}}{\#\text{Of total subnets}}$$

However, At Testing

There are a exponential number of possible sparse networks

- A neural net with n units, can be seen as a collection of 2^n possible thinned neural networks.

Something Notable

- These networks all share weights so that the total number of parameters is still $O(n^2)$, or less.

Problem, we cannot average such amount of sub-networks

- We average over the different passes to obtain a p for each node in the network
  - Meaning the probability of being active in the network.

$$p_{ijk} = \frac{\#\text{of subnets wehre node } ijk \text{ was active}}{\#\text{Of total subnets}}$$

However, At Testing

There are a exponential number of possible sparse networks

- A neural net with n units, can be seen as a collection of 2^n possible thinned neural networks.

Something Notable

- These networks all share weights so that the total number of parameters is still $O(n^2)$, or less.

Problem, we cannot average such amount of sub-networks

- We average over the different passes to obtain a p for each node in the network
  - Meaning the probability of being active in the network.

$$p_{ijk} = \frac{\text{\#of subnets wehre node } ijk \text{ was active}}{\text{\#Of total subnets}}$$

Mathematically

We have the following ideas

- Each node has associated matrices

$$W_{out} = \begin{pmatrix} w_1 & w_2 & \cdots & w_K \end{pmatrix}$$
$$W_{in} = \begin{pmatrix} w_1 & w_2 & \cdots & w_H \end{pmatrix}$$

Then use the probability p

$$pW_{out} = p \begin{pmatrix} w_1 & w_2 & \cdots & w_K \end{pmatrix}$$
$$pW_{in} = p \begin{pmatrix} w_1 & w_2 & \cdots & w_H \end{pmatrix}$$

Mathematically

We have the following ideas

- Each node has associated matrices

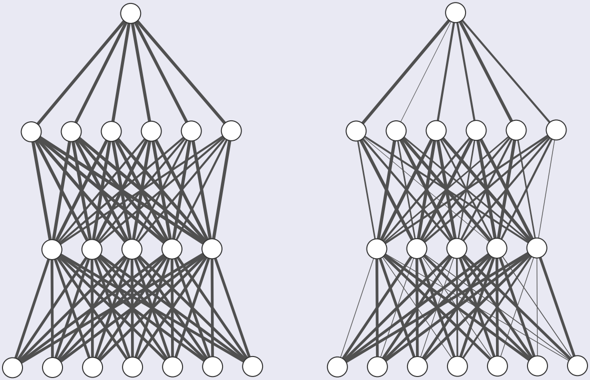
$$W_{out} = \begin{pmatrix} w_1 & w_2 & \cdots & w_K \end{pmatrix}$$
$$W_{in} = \begin{pmatrix} w_1 & w_2 & \cdots & w_H \end{pmatrix}$$

Then use the probability p

$$pW_{out} = p\begin{pmatrix} w_1 & w_2 & \cdots & w_K \end{pmatrix}$$
$$pW_{in} = p\begin{pmatrix} w_1 & w_2 & \cdots & w_H \end{pmatrix}$$

Then

We have the following structure where thiner lines represent smaller weights



The Original Structure

At Testing

Outline

Why dropout?

Srivastava et al. [6]

- A motivation for dropout comes from the theory of evolution!!!
  - Yes a original network and after a mutated one!!!

The most accepted interpretation of dropout

- It is implicitly bagging at test time a large number of neural networks which share parameters.

Why dropout?

Srivastava et al. [6]

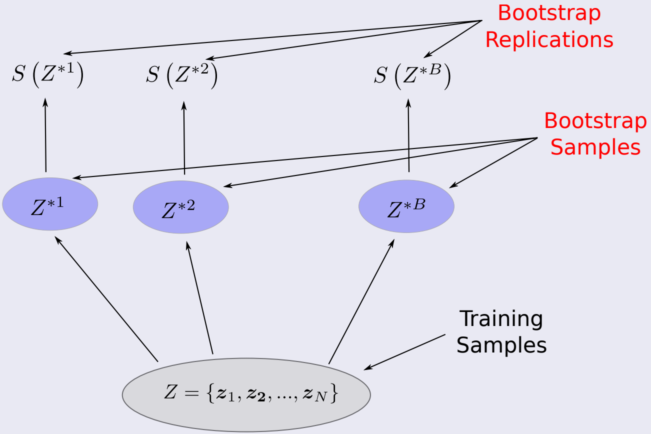
- A motivation for dropout comes from the theory of evolution!!!
  - Yes a original network and after a mutated one!!!

The most accepted interpretation of dropout

- It is implicitly bagging at test time a large number of neural networks which share parameters.

Bagging/Bootstrap Aggregation

Schematic of the Bootstrap Aggregation process [1]



Thus

Use each of them to train a copy $y_b(\boldsymbol{x})$ of a predictive regression model to predict a single continuous variable

$$y_{com}(\boldsymbol{x}) = \frac{1}{B}\sum_{b=1}^{B} y_b(\boldsymbol{x})$$

Results

We have that

Method	CIFAR-10 Error	CIFAR-100 Error
CNN+max pooling (hand tuned)	15.60%	43.48%
CNN+stochastic pooling (Zeiler and Fergus, 2013)	15.13%	42.51%
CNN+max pooling (Snoek et al., 2012)	14.98%	-
CNN+max pooling + dropout fully connected layers	14.32%	41.26%
CNN+max pooling + dropout in all layers	12.61%	37.20%
CNN+maxout (Goodfellow et al., 2013)	11.68%	38.57%

Outline

Why not use for the Data for Sparsity?

We have

$$p\left(\boldsymbol{y}^{l+1}|\boldsymbol{x}^{l},W\right)=\mathcal{N}\left(\sigma\left(W\boldsymbol{x}^{l}\right),\sigma^{2}I\right)$$

$$p\left(\sigma^{2}\right)\propto \text{"constant"}$$

$$p\left(W^{l}|\tau\right)=\prod_{i=1}^{d}\mathcal{N}\left(w_{j}^{l}|0,\tau_{j}^{l}\right)=\mathcal{N}\left(W^{l}|0,\left(\Upsilon\left(\boldsymbol{\tau}\right)\right)^{-1}\right)$$

$$p\left(\tau|\gamma\right)=\left(\frac{\gamma}{2}\right)^{d}\prod_{i=1}^{d}\exp\left\{-\frac{\gamma}{2}\tau_{i}\right\}$$

- With $\Upsilon\left(\tau\right)=diag\left(\tau_{1}^{-1},\ldots,\tau_{d}^{-1}\right)$ is the diagonal matrix with the inverse variances of all the w_i's.

Why not use for the Data for Sparsity?

We have

$$p\left(\boldsymbol{y}^{l+1}|\boldsymbol{x}^{l},W\right)=\mathcal{N}\left(\sigma\left(W\boldsymbol{x}^{l}\right),\sigma^{2}I\right)$$

$$p\left(\sigma^{2}\right)\propto \text{"}constant\text{"}$$

$$p\left(W^{l}|\tau\right)=\prod_{i=1}^{d}\mathcal{N}\left(w_{j}^{l}|0,\tau_{j}^{l}\right)=\mathcal{N}\left(W^{l}|0,\left(\Upsilon\left(\tau\right)\right)^{-1}\right)$$

$$p\left(\tau|\gamma\right)=\left(\frac{\gamma}{2}\right)^{d}\prod_{i=1}^{d}\exp\left\{-\frac{\gamma}{2}\tau_{i}\right\}$$

- With $\Upsilon\left(\tau\right)=diag\left(\tau_{1}^{-1},\ldots,\tau_{d}^{-1}\right)$ is the diagonal matrix with the inverse variances of all the w_i's.

Why not use for the Data for Sparsity?

We have

$$p\left(\boldsymbol{y}^{l+1}|\boldsymbol{x}^{l},W\right)=\mathcal{N}\left(\sigma\left(W\boldsymbol{x}^{l}\right),\sigma^{2}I\right)$$

$$p\left(\sigma^{2}\right)\propto\text{"}constant\text{"}$$

$$p\left(W^{l}|\tau\right)=\prod_{i=1}^{d}\mathcal{N}\left(w_{j}^{l}|0,\tau_{j}^{l}\right)=\mathcal{N}\left(W^{l}|0,\left(\Upsilon\left(\boldsymbol{\tau}\right)\right)^{-1}\right)$$

$$p\left(\tau|\gamma\right)=\left(\frac{\gamma}{2}\right)^{d}\prod_{i=1}^{d}\exp\left\{-\frac{\gamma}{2}\tau_{i}\right\}$$

- With $\Upsilon\left(\tau\right)=diag\left(\tau_{1}^{-1},\ldots,\tau_{d}^{-1}\right)$ is the diagonal matrix with the inverse variances of all the w_i's.

Why not use for the Data for Sparsity?

We have

$$p\left(\boldsymbol{y}^{l+1}|\boldsymbol{x}^{l},W\right)=\mathcal{N}\left(\sigma\left(W\boldsymbol{x}^{l}\right),\sigma^{2}I\right)$$

$$p\left(\sigma^{2}\right)\propto \text{"}constant\text{"}$$

$$p\left(W^{l}|\tau\right)=\prod_{i=1}^{d}\mathcal{N}\left(w_{j}^{l}|0,\tau_{j}^{l}\right)=\mathcal{N}\left(W^{l}|0,\left(\Upsilon\left(\boldsymbol{\tau}\right)\right)^{-1}\right)$$

$$p\left(\boldsymbol{\tau}|\gamma\right)=\left(\frac{\gamma}{2}\right)^{d}\prod_{i=1}^{d}\exp\left\{-\frac{\gamma}{2}\tau_{i}\right\}$$

- With $\Upsilon\left(\boldsymbol{\tau}\right)=diag\left(\tau_{1}^{-1},...,\tau_{d}^{-1}\right)$ is the diagonal matrix with the inverse variances of all the w_i's.

How do we build such distribution

Given that each w_i has a zero-mean Gaussian prior

$$p\left(w_i|\tau_i\right) = \mathcal{N}\left(w_i|0, \tau_i\right) \tag{10}$$

Where τ_i has the following exponential hyper-prior

$$p\left(\tau_i|\gamma\right) = \frac{\gamma}{2}\exp\left\{-\frac{\gamma}{2}\tau_i\right\} \text{ for } \tau_i \geq 0 \tag{11}$$

Then, we have

$$w_i \sim p\left(w_i|\gamma\right) = \int_0^{\infty} p\left(w_i|\tau_i\right) p\left(\tau_i|\gamma\right) d\tau_i = \frac{\sqrt{\gamma}}{2}\exp\left\{-\sqrt{\gamma}\left|w_i\right|\right\} \tag{12}$$

How do we build such distribution

Given that each w_i has a zero-mean Gaussian prior

$$p\left(w_i|\tau_i\right) = \mathcal{N}\left(w_i|0,\tau_i\right) \tag{10}$$

Where τ_i has the following exponential hyper-prior

$$p\left(\tau_i|\gamma\right) = \frac{\gamma}{2}\exp\left\{-\frac{\gamma}{2}\tau_i\right\} \text{ for } \tau_i \geq 0 \tag{11}$$

Then, we have

$$w_i \sim p\left(w_i|\gamma\right) = \int_0^{\infty} p\left(w_i|\tau_i\right) p\left(\tau_i|\gamma\right) d\tau_i = \frac{\sqrt{\gamma}}{2}\exp\left\{-\sqrt{\gamma}\left|w_i\right|\right\} \tag{12}$$

How do we build such distribution

Given that each w_i has a zero-mean Gaussian prior

$$p\left(w_i|\tau_i\right) = \mathcal{N}\left(w_i|0, \tau_i\right) \tag{10}$$

Where τ_i has the following exponential hyper-prior

$$p\left(\tau_i|\gamma\right) = \frac{\gamma}{2}\exp\left\{-\frac{\gamma}{2}\tau_i\right\} \text{ for } \tau_i \geq 0 \tag{11}$$

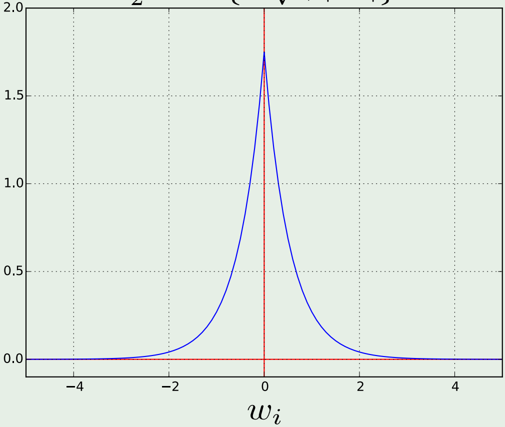
Then, we have

$$w_i \sim p\left(w_i|\gamma\right) = \int_0^\infty p\left(w_i|\tau_i\right) p\left(\tau_i|\gamma\right) d\tau_i = \frac{\sqrt{\gamma}}{2}\exp\left\{-\sqrt{\gamma}\left|w_i\right|\right\} \tag{12}$$

Example

The double exponential

$$\frac{\sqrt{\gamma}}{2}\exp\left\{-\sqrt{\gamma}\,|\,w_i|\right\}$$



Then using the Monte Carlo Method

We have

$$E\left[W^t | f\left(W_b^{tl}\boldsymbol{x}_b\right), \sigma^2 I\right] = \frac{p\left(\sigma^2\right)}{B}\sum_{b=1}^{B}\mathcal{N}\left(f\left(W_b^{tl}\boldsymbol{x}_b\right), \sigma^2 I\right) p\left(W_b^{tl}|\tau_i\right) p\left(\tau_i|\gamma\right)$$

Then, we use the mini batch per epoch to decide if we drop a weight

- Basically, the previous

Then using the Monte Carlo Method

We have

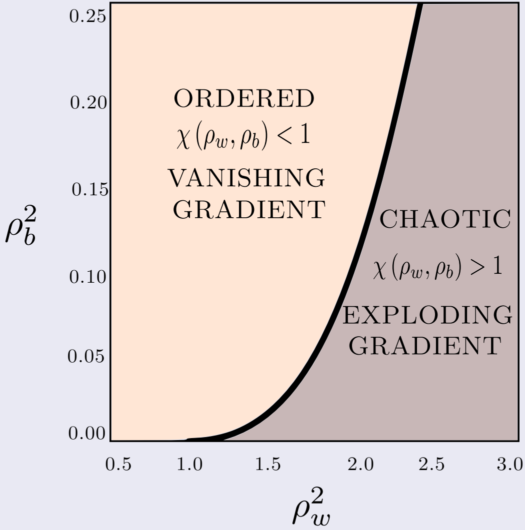
$$E\left[W^{t}|f\left(W_{b}^{tl}\boldsymbol{x}_{b}\right),\sigma^{2}I\right]=\frac{p\left(\sigma^{2}\right)}{B}\sum_{b=1}^{B}\mathcal{N}\left(f\left(W_{b}^{tl}\boldsymbol{x}_{b}\right),\sigma^{2}I\right)p\left(W_{b}^{tl}|\tau_{i}\right)p\left(\tau_{i}|\gamma\right)$$

Then, we use the mini batch per epoch to decide if we drop a weight

- Basically, the previous

We are using the following idea

Basically, we are using the fact that



Thus, we have that

The layer output can be bounded by

$$\mathcal{N}\left(f\left(W_b^{tl}\boldsymbol{x}_b\right), \sigma^2 I\right)$$

The other part of the equation is the sparsity part

$$p\left(W_b^{tl}|\tau_i\right)p\left(\tau_i|\gamma\right)$$

Thus, we have that

The layer output can be bounded by

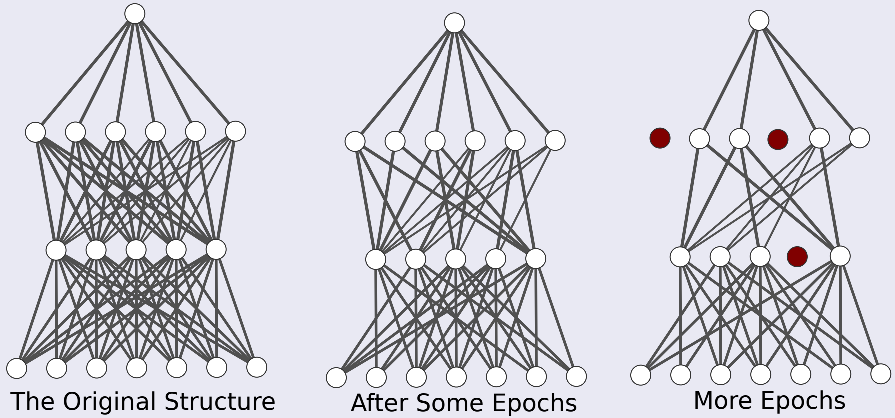
$$\mathcal{N}\left(f\left(W_b^{tl}\boldsymbol{x}_b\right), \sigma^2 I\right)$$

The other part of the equation is the sparsity part

$$p\left(W_b^{tl}|\tau_i\right) p\left(\tau_i|\gamma\right)$$

As the process progress

Once the weights fall below certain level we shutdown the weight



Outline

Bouthillier et al.[8]

The main goal when using dropout

- It is to regularize the neural network we are training

Those random modifications of the network's stucture

- They are believed to avoid co-adaptation of neurons by making it impossible for two subsequent neurons to rely solely on each other [6]

Bouthillier et al.[8]

The main goal when using dropout

- It is to regularize the neural network we are training

Those random modifications of the network's stucture

- They are believed to avoid co-adaptation of neurons by making it impossible for two subsequent neurons to rely solely on each other [6]

Therefore

We have a function that projects from a dimensional space to another

$$h(\boldsymbol{x}) = W\boldsymbol{x} + \boldsymbol{b}$$

Then, given the noisy version of an activation function where $M \sim \mathcal{B}(p_h)$

$$\widetilde{f}(h) = M \odot rect(h) \ \text{(Training)}$$

- Where $f(h) = rect(h)$ (Testing)

Actually Srivastava et al. [6]

- He mentions to use

$$p_{ijk} = \frac{\#\text{of subnets wehre node } ijk \text{ was active}}{\#\text{Of total subnets}}$$

Therefore

We have a function that projects from a dimensional space to another

$$h(\boldsymbol{x}) = W\boldsymbol{x} + \boldsymbol{b}$$

Then, given the noisy version of an activation function where $M \sim \mathcal{B}(p_h)$

$$\widetilde{f}(h) = M \odot rect(h) \text{ (Training)}$$

- Where $f(h) = rect(h)$ (Testing)

Actually Srivastava et al. [6]

- He mentions to use

$$p_{ijk} = \frac{\#\text{of subnets wehre node } ijk \text{ was active}}{\#\text{Of total subnets}}$$

Therefore

We have a function that projects from a dimensional space to another

$$h(\boldsymbol{x}) = W\boldsymbol{x} + \boldsymbol{b}$$

Then, given the noisy version of an activation function where $M \sim \mathcal{B}(p_h)$

$$\widetilde{f}(h) = M \odot rect(h) \text{ (Training)}$$

- Where $f(h) = rect(h)$ (Testing)

Actually Srivastava et al. [6]

- He mentions to use

$$p_{ijk} = \frac{\#\text{of subnets wehre node } ijk \text{ was active}}{\#\text{Of total subnets}}$$

Outline

Data Augmentation

In many previous works [5, 4]

- It has been shown that augmenting data by using domain specific transformations helps in learning better models

Therefore, the main idea

- It is to map input data to output labels

One way to learn such a mapping function

- It is to augment the data using noise:
  - Hypothesis!!! Noise based regularization techniques seems to be increasing training data coverage as augmentation

Data Augmentation

In many previous works [5, 4]

- It has been shown that augmenting data by using domain specific transformations helps in learning better models

Therefore, the main idea

- It is to map input data to output labels

One way to learn such a mapping function

- It is to augment the data using noise:
  - Hypothesis!!! Noise based regularization techniques seems to be increasing training data coverage as augmentation

Data Augmentation

In many previous works [5, 4]

- It has been shown that augmenting data by using domain specific transformations helps in learning better models

Therefore, the main idea

- It is to map input data to output labels

One way to learn such a mapping function

- It is to augment the data using noise:
  - Hypothesis!!! Noise based regularization techniques seems to be increasing training data coverage as augmentation

Outline

Augmenting by Noise [8]

We assume that for a given $\widetilde{f}(h)$, there is an optimal $\boldsymbol{x}^*$

$$(f \circ h)(\boldsymbol{x}^*) = rect\left(h\left(\boldsymbol{x}^*\right)\right) :\approx M \odot rect\left(h\right) = \left(\widetilde{f} \circ h\right)(\boldsymbol{x}^*)$$

This $\boldsymbol{x}^*$ can be found by minimizing by stochastic gradient descent

$$L\left(\boldsymbol{x}, \boldsymbol{x}^*\right) = \left[(f \circ h)(\boldsymbol{x}^*) - \left(\widetilde{f} \circ h\right)(\boldsymbol{x}^*)\right]^2$$

Augmenting by Noise [8]

We assume that for a given $\widetilde{f}(h)$, there is an optimal $\boldsymbol{x}^*$

$$(f \circ h)(\boldsymbol{x}^*) = rect\left(h\left(\boldsymbol{x}^*\right)\right) :\approx M \odot rect\left(h\right) = \left(\widetilde{f} \circ h\right)(\boldsymbol{x}^*)$$

This $\boldsymbol{x}^*$ can be found by minimizing by stochastic gradient descent

$$L\left(\boldsymbol{x}, \boldsymbol{x}^*\right) = \left[(f \circ h)(\boldsymbol{x}^*) - \left(\widetilde{f} \circ h\right)(\boldsymbol{x}^*)\right]^2$$

Extending to n layers

For this, we define

$$\widetilde{g}^{(i)}\left(\boldsymbol{x}\right) = \left[\widetilde{f}^{(i)} \circ h^{(i)} \circ \cdots \circ \widetilde{f}^{(1)} \circ h^{(1)}\right]\left(\boldsymbol{x}\right)$$

$$g^{(i)}\left(\boldsymbol{x}^{*}\right) = \left[f^{(i)} \circ h^{(i)} \circ \cdots \circ f^{(1)} \circ h^{(1)}\right]\left(\boldsymbol{x}^{*}\right)$$

Then, it is possible to compute the back propagation projection corresponding to all hidden layer activations at once

$$L\left(\boldsymbol{x}, \boldsymbol{x}^{(1)^{*}}, \ldots, \boldsymbol{x}^{(n)^{*}}\right) = \sum_{i=1}^{n} \lambda_i \left[g^{(i)}\left(\boldsymbol{x}^{(i)^{*}}\right) - \widetilde{g}^{(i)}\left(\boldsymbol{x}\right)\right]^2$$

Extending to n layers

For this, we define

$$\widetilde{g}^{(i)}\left(\boldsymbol{x}\right) = \left[\widetilde{f}^{(i)} \circ h^{(i)} \circ \cdots \circ \widetilde{f}^{(1)} \circ h^{(1)}\right]\left(\boldsymbol{x}\right)$$
$$g^{(i)}\left(\boldsymbol{x}^{*}\right) = \left[f^{(i)} \circ h^{(i)} \circ \cdots \circ f^{(1)} \circ h^{(1)}\right]\left(\boldsymbol{x}^{*}\right)$$

Then, it is possible to compute the back propagation projection corresponding to all hidden layer activations at once

$$L\left(\boldsymbol{x}, \boldsymbol{x}^{(1)^{*}}, \ldots, \boldsymbol{x}^{(n)^{*}}\right) = \sum_{i=1}^{n} \lambda_i \left[g^{(i)}\left(\boldsymbol{x}^{(i)^{*}}\right) - \widetilde{g}^{(i)}\left(\boldsymbol{x}\right)\right]^2$$

However

Small Problem

- It is possible to show by contradiction that one is unlikely to find a single $\boldsymbol{x}^* = \boldsymbol{x}^{(1)^*} = \cdots = \boldsymbol{x}^{(n)^*}$
  - Such that you can significantly reduce L

Proof of the unlikeness of $\boldsymbol{x}^{*} = \boldsymbol{x}^{(1)^{*}} = \cdots = \boldsymbol{x}^{(n)^{*}}$

By the associative property of function composition

$$g^{(i)}\left(\boldsymbol{x}^{*}\right) = \left(f^{(i)} \circ h^{(i)}\right)\left(g^{(i-1)}\left(\boldsymbol{x}^{*}\right)\right)$$

Suppose there exist $\boldsymbol{x}^{*} = \boldsymbol{x}^{(1)^{*}} = \cdots = \boldsymbol{x}^{(n)^{*}}$ an such that

$$\left(f^{(i)} \circ h^{(i)}\right)\left(g^{(i-1)}\left(\boldsymbol{x}^{*}\right)\right) = \left(\widetilde{f}^{(i)} \circ h^{(i)}\right)\left(\widetilde{g}^{(i-1)}\left(\boldsymbol{x}\right)\right)$$

$$\left(f^{(i-1)} \circ h^{(i-1)}\right)\left(g^{(i-2)}\left(\boldsymbol{x}^{*}\right)\right) = \left(\widetilde{f}^{(i-1)} \circ h^{(i-1)}\right)\left(\widetilde{g}^{(i-2)}\left(\boldsymbol{x}\right)\right)$$

Proof of the unlikeness of $\boldsymbol{x}^* = \boldsymbol{x}^{(1)^*} = \cdots = \boldsymbol{x}^{(n)^*}$

By the associative property of function composition

$$g^{(i)}\left(\boldsymbol{x}^*\right) = \left(f^{(i)} \circ h^{(i)}\right)\left(g^{(i-1)}\left(\boldsymbol{x}^*\right)\right)$$

Suppose there exist $\boldsymbol{x}^* = \boldsymbol{x}^{(1)^*} = \cdots = \boldsymbol{x}^{(n)^*}$ an such that

$$\left(f^{(i)} \circ h^{(i)}\right)\left(g^{(i-1)}\left(\boldsymbol{x}^*\right)\right) = \left(\widetilde{f}^{(i)} \circ h^{(i)}\right)\left(\widetilde{g}^{(i-1)}\left(\boldsymbol{x}\right)\right)$$

$$\left(f^{(i-1)} \circ h^{(i-1)}\right)\left(g^{(i-2)}\left(\boldsymbol{x}^*\right)\right) = \left(\widetilde{f}^{(i-1)} \circ h^{(i-1)}\right)\left(\widetilde{g}^{(i-2)}\left(\boldsymbol{x}\right)\right)$$

Then

Based on the previous equations

$$g^{(i-1)}\left(\boldsymbol{x}^*\right) = \widetilde{g}^{(i-1)}\left(\boldsymbol{x}\right)$$

Then, we get

$$\left(f^{(i)} \circ h^{(i)}\right)\left(g^{(i-1)}\left(\boldsymbol{x}^*\right)\right) = \left(\widetilde{f}^{(i)} \circ h^{(i)}\right)\left(\widetilde{g}^{(i-1)}\left(\boldsymbol{x}\right)\right)$$

Finally

$$rect\left(h^{(i)}\left(g^{(i-1)}\left(\boldsymbol{x}^*\right)\right)\right) = M^{(i)} \odot rect\left(h^{(i)}\left(g^{(i-1)}\left(\boldsymbol{x}^*\right)\right)\right)$$

Then

Based on the previous equations

$$g^{(i-1)}\left(\boldsymbol{x}^{*}\right)=\widetilde{g}^{(i-1)}\left(\boldsymbol{x}\right)$$

Then, we get

$$\left(f^{(i)}\circ h^{(i)}\right)\left(g^{(i-1)}\left(\boldsymbol{x}^{*}\right)\right)=\left(\widetilde{f}^{(i)}\circ h^{(i)}\right)\left(\widetilde{g}^{(i-1)}\left(\boldsymbol{x}\right)\right)$$

Finally

$$rect\left(h^{(i)}\left(g^{(i-1)}\left(\boldsymbol{x}^{*}\right)\right)\right)=M^{(i)}\odot rect\left(h^{(i)}\left(g^{(i-1)}\left(\boldsymbol{x}^{*}\right)\right)\right)$$

Then

Based on the previous equations

$$g^{(i-1)}\left(\boldsymbol{x}^{*}\right)=\widetilde{g}^{(i-1)}\left(\boldsymbol{x}\right)$$

Then, we get

$$\left(f^{(i)}\circ h^{(i)}\right)\left(g^{(i-1)}\left(\boldsymbol{x}^{*}\right)\right)=\left(\widetilde{f}^{(i)}\circ h^{(i)}\right)\left(\widetilde{g}^{(i-1)}\left(\boldsymbol{x}\right)\right)$$

Finally

$$rect\left(h^{(i)}\left(g^{(i-1)}\left(\boldsymbol{x}^{*}\right)\right)\right)=M^{(i)}\odot rect\left(h^{(i)}\left(g^{(i-1)}\left(\boldsymbol{x}^{*}\right)\right)\right)$$

Therefore

This is only true if $M^{(i)} = 1$

- When $rect_j\left(h^{(i)}\left(g^{(i-1)}\left(\boldsymbol{x}^*\right)\right)\right) > 0$

This only happens with a probability $p_{(i)}^{d_{(i)}s_{(i)}}$

- Where:
  - $p_{(i)}$ is the Bernoulli success probability.
  - $d_{(i)}$ is the number of of hidden units.
  - $s_{(i)}$ is the mean sparsity level at i (Mean percentage of active hidden units).

Therefore

This is only true if $M^{(i)} = 1$

- When $rect_j\left(h^{(i)}\left(g^{(i-1)}\left(\boldsymbol{x}^*\right)\right)\right) > 0$

This only happens with a probability $p_{(i)}^{d_{(i)} s_{(i)}}$

- Where:
  - $p_{(i)}$ is the Bernoulli success probability.
  - $d_{(i)}$ is the number of of hidden units.
  - $s_{(i)}$ is the mean sparsity level at i (Mean percentage of active hidden units).

Which is quite low!!!

This probability is very low for standard hyper-parameters values

- With $p_{(i)} = 0.5$, $d_{(i)} = 1000$ and $s_{(i)} = 0.15$

$$p_{(i)}^{d_{(i)} s_{(i)}} = 10^{-47}$$

However

Fortunately

- It is easy to find a different x^* for each hidden layer

by providing multiple inputs

$$\left(x, x^{(1)^*}, x^{(2)^*}, \ldots, x^{(n)^*}\right)$$

However

- This raises the question whether we can train the network deterministically on the $x^{(i)^*}$ instead of using dropout

However

Fortunately

- It is easy to find a different $\boldsymbol{x}^*$ for each hidden layer

by providing multiple inputs

$$\left(\boldsymbol{x}, \boldsymbol{x}^{(1)^*}, \boldsymbol{x}^{(2)^*}, \ldots, \boldsymbol{x}^{(n)^*}\right)$$

However

- This raises the question whether we can train the network deterministically on the $x^{(i)^*}$ instead of using dropout

However

Fortunately

- It is easy to find a different $\boldsymbol{x}^*$ for each hidden layer

by providing multiple inputs

$$\left(\boldsymbol{x}, \boldsymbol{x}^{(1)^*}, \boldsymbol{x}^{(2)^*}, ..., \boldsymbol{x}^{(n)^*}\right)$$

However

- This raises the question whether we can train the network deterministically on the $\boldsymbol{x}^{(i)^*}$ instead of using dropout

Outline

Co-adaptation/Overfitting

Definition

- Co-adaptation is the accumulation of interacting genes in the gene pool of a population by selection.
  - Selection pressures on one of the genes will affect its interacting proteins, after which compensatory changes occur.

In Neural Networks

- In neural network, co-adaptation means that some neurons are highly dependent on others:
  - Getting into over-fitting!!!

Co-adaptation/Overfitting

Definition

- Co-adaptation is the accumulation of interacting genes in the gene pool of a population by selection.
  - Selection pressures on one of the genes will affect its interacting proteins, after which compensatory changes occur.

In Neural Networks

- In neural network, co-adaptation means that some neurons are highly dependent on others:
  - Getting into over-fitting!!!

Question

We have that

- Question: Can we train the network deterministically on $\boldsymbol{x}^{(i)^*}$?

- The gradients of the linear projections will differ greatly different from dropout!!!

Question

We have that

- Question: Can we train the network deterministically on $\boldsymbol{x}^{(i)^*}$?

This is not trivial given that

- Dropout is not effectively applied to every layer at the same time when using
$$\left(\boldsymbol{x}, \boldsymbol{x}^{(1)^*}, \boldsymbol{x}^{(2)^*}, \ldots, \boldsymbol{x}^{(n)^*}\right)$$
- The gradients of the linear projections will differ greatly, different from dropout!!!

Question

We have that

- Question: Can we train the network deterministically on $\boldsymbol{x}^{(i)^*}$?

This is not trivial given that

- Dropout is not effectively applied to every layer at the same time when using
$$\left(\boldsymbol{x}, \boldsymbol{x}^{(1)^*}, \boldsymbol{x}^{(2)^*}, ..., \boldsymbol{x}^{(n)^*}\right)$$

- The gradients of the linear projections will differ greatly, different from dropout!!!

Therefore

We can then

- Modifying the probability distribution is the most straightforward way to improve the set of transformations.

For example

- A simple way to vary the transformation magnitude randomly is to replace p_{hij} by a random variable!!!

Therefore

We can then

- Modifying the probability distribution is the most straightforward way to improve the set of transformations.

For example

- A simple way to vary the transformation magnitude randomly is to replace p_{hij} by a random variable!!!

Therefore

Define

$$M_{hij} \sim \mathcal{B}(\rho_h) \text{ (Bernoulli)}$$
$$\rho_h \sim U(0, p_h) \text{ (Uniform)}$$

- where h defines the layer, i the sample, and j the layer's neuron.

Here, the authors use the same ρ for all the layers of the neurons, then

$$\widetilde{f}(h) = \frac{1}{1-\rho} M \odot rect(h)$$

Therefore

Define

$$M_{hij} \sim \mathcal{B}\left(\rho_h\right) \text{ (Bernoulli)}$$
$$\rho_h \sim U\left(0, p_h\right) \text{ (Uniform)}$$

- where h defines the layer, i the sample, and j the layer's neuron.

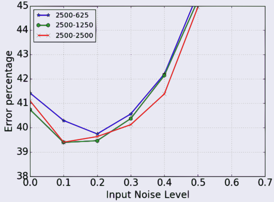
Here, the authors use the same ρ for all the layers of the neurons, then

$$\widetilde{f}\left(h\right) = \frac{1}{1-\rho} M \odot rect\left(h\right)$$

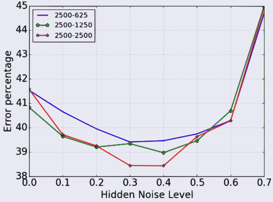
Results

Something Notable

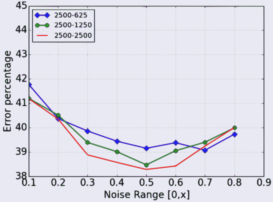
Using dropout with varying input noise and fixed hidden noise of 0.5.



Using dropout with varying input noise and fixed hidden noise of 0.2.



Using Random-dropout with varying noise range [0, x] used at hidden and input layers



Outline

Here, the people at Google [9] around 2015

They commented in the "Internal Covariate Shift Phenomena"

- Due to the change in the distribution of each layer's input

They claim

- The min-batch forces to have those changes which impact on the learning capabilities of the network.

In Neural Networks, they define this

- Internal Covariate Shift as the change in the distribution of network activations due to the change in network parameters during training.

Here, the people at Google [9] around 2015

They commented in the "Internal Covariate Shift Phenomena"

- Due to the change in the distribution of each layer's input

They claim

- The min-batch forces to have those changes which impact on the learning capabilities of the network.

In Neural Networks, they define this

- Internal Covariate Shift as the change in the distribution of network activations due to the change in network parameters during training.

Here, the people at Google [9] around 2015

They commented in the "Internal Covariate Shift Phenomena"

- Due to the change in the distribution of each layer's input

They claim

- The min-batch forces to have those changes which impact on the learning capabilities of the network.

In Neural Networks, they define this

- Internal Covariate Shift as the change in the distribution of network activations due to the change in network parameters during training.

Transformation

Batch Normalizing Transform

Input: Values of $\boldsymbol{x}$ over a mini-batch: $\mathcal{B} = \{\boldsymbol{x}_{1\ldots m}\}$, Parameters to be learned: γ, β

Output: $\{y_i = BN_{\gamma,\beta}(\boldsymbol{x}_i)\}$

1. $\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{x}_i$
2. $\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m} (\boldsymbol{x}_i - \mu_{\mathcal{B}})^2$
3. $\hat{\boldsymbol{x}} = \frac{\boldsymbol{x}_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$
4. $y_i = \gamma^{(k)}\hat{\boldsymbol{x}}_i + \beta = BN_{\gamma,\beta}(\boldsymbol{x}_i)$

Transformation

Batch Normalizing Transform

Input: Values of $\boldsymbol{x}$ over a mini-batch: $\mathcal{B} = \{\boldsymbol{x}_{1\ldots m}\}$, Parameters to be learned: γ, β

Output: $\{y_i = BN_{\gamma,\beta}(\boldsymbol{x}_i)\}$

1. $\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{x}_i$
2. $\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m} (\boldsymbol{x}_i - \mu_{\mathcal{B}})^2$
3. $\hat{\boldsymbol{x}} = \frac{\boldsymbol{x}_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$
4. $y_i = \gamma^{(k)} \hat{\boldsymbol{x}}_i + \beta = BN_{\gamma,\beta}(\boldsymbol{x}_i)$

Transformation

Batch Normalizing Transform

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$, Parameters to be learned: γ, β

Output: $\{y_i = BN_{\gamma,\beta}(x_i)\}$

1. $\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i$
2. $\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2$
3. $\hat{x} = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$
4. $y_i = \gamma^{(k)}\hat{x}_i + \beta = BN_{\gamma,\beta}(x_i)$

Transformation

Batch Normalizing Transform

Input: Values of x over a mini-batch: $\mathcal{B} = \{\boldsymbol{x}_{1\ldots m}\}$, Parameters to be learned: γ, β

Output: $\{y_i = BN_{\gamma,\beta}(\boldsymbol{x}_i)\}$

1. $\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{x}_i$
2. $\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m} (\boldsymbol{x}_i - \mu_{\mathcal{B}})^2$
3. $\widehat{\boldsymbol{x}} = \frac{\boldsymbol{x}_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$
4. $y_i = \gamma^{(k)}\widehat{\boldsymbol{x}}_i + \beta = BN_{\gamma,\beta}(\boldsymbol{x}_i)$

Transformation

Batch Normalizing Transform

Input: Values of $\boldsymbol{x}$ over a mini-batch: $\mathcal{B} = \{\boldsymbol{x}_{1\dots m}\}$, Parameters to be learned: γ, β

Output: $\{y_i = BN_{\gamma,\beta}(\boldsymbol{x}_i)\}$

1. $\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{x}_i$
2. $\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m} (\boldsymbol{x}_i - \mu_{\mathcal{B}})^2$
3. $\widehat{\boldsymbol{x}} = \frac{\boldsymbol{x}_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$
4. $\boldsymbol{y}_i = \gamma^{(k)} \widehat{\boldsymbol{x}}_i + \beta = BN_{\gamma,\beta}(\boldsymbol{x}_i)$

Outline

Remember

Using Min-Batch inputs, we have

$$\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{x}_i$$

And Variance

$$\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m} \left(\boldsymbol{x}_i - \mu_{\mathcal{B}}\right)^2$$

Remember

Using Min-Batch inputs, we have

$$\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{x}_i$$

And Variance

$$\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m} \left(\boldsymbol{x}_i - \mu_{\mathcal{B}}\right)^2$$

Therefore, Ba et al. [10]

We get the mean over the output of the layer l with H number of hidden units

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} y_i^l$$

- Basically, do the forward process then add over the output $y_i^l = w_i^{lT} h^l$ where $h_i^{l+1} = f\left(y_i^l + b_i^l\right)$

Then the standard deviation layer l

$$\sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(y_i^l - \mu^l\right)^2}$$

Therefore, Ba et al. [10]

We get the mean over the output of the layer l with H number of hidden units

$$\mu^l = \frac{1}{H} \sum_{i=1}^{H} y_i^l$$

- Basically, do the forward process then add over the output $y_i^l = w_i^{lT} h^l$ where $h_i^{l+1} = f\left(y_i^l + b_i^l\right)$

Then the standard deviation layer l

$$\sigma^l = \sqrt{\frac{1}{H} \sum_{i=1}^{H} \left(y_i^l - \mu^l\right)^2}$$

Remarks

We have that

- All the hidden units in a layer share the same normalization terms μ and σ
  - but different training cases have different normalization terms.

Layer normalization does not impose any constraint

- On the size of a mini-batch and it can be used in the pure on-line regime with batch size 1.

Remarks

We have that

- All the hidden units in a layer share the same normalization terms μ and σ
  - but different training cases have different normalization terms.

Layer normalization does not impose any constraint

- On the size of a mini-batch and it can be used in the pure on-line regime with batch size 1.

Outline

The Flow of Information through time

First, the new $\boldsymbol{h}^t$ with a gain vector $\boldsymbol{g}$

$$\boldsymbol{h}^t = f\left[\frac{\boldsymbol{g}}{\sigma^t} \odot \left(\boldsymbol{y}^t - \mu^t\right) + b\right]$$

The Temporal Layer Mean Normalization

$$\mu^t = \frac{1}{H}\sum_{i=1}^{H} y_i^t$$

The Temporal Layer STD Normalization

$$\sigma^t = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(y_i^t - \mu^t\right)^2}$$

The Flow of Information through time

First, the new $\boldsymbol{h}^t$ with a gain vector $\boldsymbol{g}$

$$\boldsymbol{h}^t = f\left[\frac{\boldsymbol{g}}{\sigma^t} \odot \left(\boldsymbol{y}^t - \mu^t\right) + b\right]$$

The Temporal Layer Mean Normalization

$$\mu^t = \frac{1}{H}\sum_{i=1}^{H} y_i^t$$

The Temporal Layer STD Normalization

$$\sigma^t = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(y_i^t - \mu^t\right)^2}$$

The Flow of Information through time

First, the new $\boldsymbol{h}^t$ with a gain vector $\boldsymbol{g}$

$$\boldsymbol{h}^t = f\left[\frac{\boldsymbol{g}}{\sigma^t} \odot \left(\boldsymbol{y}^t - \mu^t\right) + b\right]$$

The Temporal Layer Mean Normalization

$$\mu^t = \frac{1}{H}\sum_{i=1}^{H} y_i^t$$

The Temporal Layer STD Normalization

$$\sigma^t = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(y_i^t - \mu^t\right)^2}$$

Outline

Weight re-scaling and re-centering

Observe that under batch normalization and weight normalization

- Any re-scaling to the incoming weights w_i of a single neuron has no effect on the normalized summed inputs to a neuron.

Meaning

- If the weight vector is scaled by δ_i the two scalars μ and σ will also be scaled by δ

Properties

- The batch and weight normalization are invariant to the re-scaling of the weights.

Weight re-scaling and re-centering

Observe that under batch normalization and weight normalization

- Any re-scaling to the incoming weights w_i of a single neuron has no effect on the normalized summed inputs to a neuron.

Meaning

- If the weight vector is scaled by δ_i the two scalars μ and σ will also be scaled by δ

Properties

- The batch and weight normalization are invariant to the re-scaling of the weights.

Weight re-scaling and re-centering

Observe that under batch normalization and weight normalization

- Any re-scaling to the incoming weights w_i of a single neuron has no effect on the normalized summed inputs to a neuron.

Meaning

- If the weight vector is scaled by δ_i the two scalars μ and σ will also be scaled by δ

Properties

- The batch and weight normalization are invariant to the re-scaling of the weights.

In the other hand

Layer normalization

- It is not invariant to the individual scaling of the single weight vectors.

- Also it is invariant to a shift to all of the incoming weights in the weight matrix.

In the other hand

Layer normalization

- It is not invariant to the individual scaling of the single weight vectors.

However

- Layer normalization is invariant to scaling of the entire weight matrix.
- Also it is invariant to a shift to all of the incoming weights in the weight matrix.

In the other hand

Layer normalization

- It is not invariant to the individual scaling of the single weight vectors.

However

- Layer normalization is invariant to scaling of the entire weight matrix.
- Also it is invariant to a shift to all of the incoming weights in the weight matrix.

How?

Imagine the following

- Let there be two sets of model parameters θ, θ' with weigh matrices

$$W' = \delta W + 1\gamma^T$$

We have

Given that $y_i^l = w_i^{lT} \boldsymbol{x}^l$

$$y_i'^l = \left(\delta W + 1\gamma^T\right)_i \boldsymbol{x}^l$$

Then, we have

$$\mu'^l = \frac{\delta}{H}\sum_{i=1}^{H} W_i \boldsymbol{x}^l + \frac{1}{H}\sum_{i=1}^{H}\left(1\gamma^T\right)_i \boldsymbol{x}^l = \delta\mu + \left(1\gamma^T\right)_i \boldsymbol{x}^l$$

We have

Given that $y_i^l = w_i^{lT} \boldsymbol{x}^l$

$$y_i'^l = \left(\delta W + 1\gamma^T\right)_i \boldsymbol{x}^l$$

Then, we have

$$\mu'^l = \frac{\delta}{H}\sum_{i=1}^{H} W_i \boldsymbol{x}^l + \frac{1}{H}\sum_{i=1}^{H}\left(1\gamma^T\right)_i \boldsymbol{x}^l = \delta\mu + \left(1\gamma^T\right)_i \boldsymbol{x}^l$$

Now

Standard Deviation

$$\sigma' = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(y_i'^{l} - \mu'\right)^2} = \delta\sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(y_i^{l} - \mu\right)^2}$$

Finally, Under Layer Normalization, we have the same output

$$\begin{aligned} h' &= f\left[\frac{g}{\sigma'}\left(W'x - \mu'\right) + b\right] \\ &= f\left[\frac{g}{\sigma'}\left(\left[\delta W + 1\gamma^T\right]x - \mu'\right) + b\right] \\ &= f\left[\frac{g}{\sigma}\left(Wx - \mu\right) + b\right] = h \end{aligned}$$

Now

Standard Deviation

$$\sigma' = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(y_i'^{l} - \mu'\right)^2} = \delta\sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(y_i^{l} - \mu\right)^2}$$

Finally, Under Layer Normalization, we have the same output

$$\begin{aligned}
\boldsymbol{h}' &= f\left[\frac{\boldsymbol{g}}{\sigma'}\left(W'\boldsymbol{x} - \mu'\right) + \boldsymbol{b}\right] \\
&= f\left[\frac{\boldsymbol{g}}{\sigma'}\left(\left[\delta W + 1\gamma^T\right]\boldsymbol{x} - \mu'\right) + \boldsymbol{b}\right] \\
&= f\left[\frac{\boldsymbol{g}}{\sigma}\left(W\boldsymbol{x} - \mu\right) + \boldsymbol{b}\right] = \boldsymbol{h}
\end{aligned}$$

Remarks

Something Notable

- if normalization is only applied to the input before the weights, the model will not be invariant to re-scaling and re-centering of the weights.

Data re-scaling and re-centering

We can show

- All the normalization methods are invariant to re-scaling the dataset

Layer normalization is invariant to re-scaling of individual training cases

$$h'_i = f\left[\frac{g_i}{\sigma'}\left(w_i^T x' - \mu'\right) + b_i\right] = f\left[\frac{g_i}{\delta\sigma}\left(\delta w_i^T x - \delta\mu\right) + b_i\right] = h_i$$

Data re-scaling and re-centering

We can show

- All the normalization methods are invariant to re-scaling the dataset

Layer normalization is invariant to re-scaling of individual training cases

$$h_i' = f\left[\frac{g_i}{\sigma'}\left(w_i^T \boldsymbol{x}' - \mu'\right) + b_i\right] = f\left[\frac{g_i}{\delta\sigma}\left(\delta w_i^T \boldsymbol{x} - \delta\mu\right) + b_i\right] = h_i$$

Additionally

Layer Normalization has a relation with the Fisher Information Matrix

$$F(\theta) = E_{\boldsymbol{x}\sim P(\boldsymbol{x}), y\sim P(y|\boldsymbol{x})}\left[\frac{\partial \log P(y|\boldsymbol{x})}{\partial\theta}\left(\frac{\partial \log P(y|\boldsymbol{x})}{\partial\theta}\right)^T\right]$$

Basically, we can write the generalized linear model as

$$\log P(y|\boldsymbol{x}, w, b) = \frac{(a+b)\,y - \eta(a+b)}{\Phi} + c(y, \Phi)$$

$$E[y|\boldsymbol{x}] = f(a+b) = f\left(w^T\boldsymbol{x} + b\right)$$

$$Var[y|\boldsymbol{x}] = \Phi f'(a+b)$$

Additionally

Layer Normalization has a relation with the Fisher Information Matrix

$$F(\theta) = E_{\boldsymbol{x}\sim P(\boldsymbol{x}), y\sim P(y|\boldsymbol{x})}\left[\frac{\partial \log P(y|\boldsymbol{x})}{\partial \theta}\left(\frac{\partial \log P(y|\boldsymbol{x})}{\partial \theta}\right)^T\right]$$

Basically, we can write the generalized linear model as

$$\log P(y|\boldsymbol{x}, w, b) = \frac{(a+b)\,y - \eta(a+b)}{\Phi} + c(y, \Phi)$$

$$E[y|\boldsymbol{x}] = f(a+b) = f\left(w^T\boldsymbol{x} + b\right)$$

$$Var[y|\boldsymbol{x}] = \Phi f'(a+b)$$

The curvature of a Riemannian manifold

It is entirely captured by its Riemannian metric

$$ds^2 \approx \frac{1}{2}\delta^T F(\theta)\,\delta$$

- where, δ is a small change to the parameters.

Then, under Layer Normalization, we have

$$F(\theta) = \frac{1}{\Phi^2} E_{x\sim P(x)} \begin{bmatrix} Cov(y_1, y_2|x)\,\frac{(a_1-\mu)^2}{\sigma^2} & \cdots & Cov(y_1, y_H|x)\,\frac{(a_1-\mu)(a_H-\mu)}{\sigma^2} \\ \vdots & \ddots & \vdots \\ Cov(y_H, y_1|x)\,\frac{(a_1-\mu)(a_H-\mu)}{\sigma^2} & \cdots & Cov(y_H, y_H|x)\,\frac{(a_H-\mu)^2}{\sigma^2} \end{bmatrix}$$

The curvature of a Riemannian manifold

It is entirely captured by its Riemannian metric

$$ds^2 \approx \frac{1}{2}\delta^T F(\theta)\,\delta$$

- where, δ is a small change to the parameters.

Then, under Layer Normalization, we have

$$F(\theta) = \frac{1}{\Phi^2} E_{x\sim P(\boldsymbol{x})} \begin{bmatrix} Cov\left(y_1, y_2|\boldsymbol{x}\right)\frac{(a_1-\mu)^2}{\sigma^2} & \cdots & Cov\left(y_1, y_H|\boldsymbol{x}\right)\frac{(a_1-\mu)(a_H-\mu)}{\sigma^2} \\ \vdots & \ddots & \vdots \\ Cov\left(y_H, y_1|\boldsymbol{x}\right)\frac{(a_1-\mu)(a_H-\mu)}{\sigma^2} & \cdots & Cov\left(y_H, y_H|\boldsymbol{x}\right)\frac{(a_H-\mu)^2}{\sigma^2} \end{bmatrix}$$

Where

We have that $a_i = w_i^T \boldsymbol{x}$

- We project the gradient updates to the gain parameter δ_{gi} of the i^{th} neuron to its weight vector as

$$\frac{\delta_{gi}\delta_{gj}}{2\Phi^2} E_{x\sim P(\boldsymbol{x})}\left[Cov\left(y_i, y_j|\boldsymbol{x}\right)\frac{\left(a_1-\mu\right)\left(a_H-\mu\right)}{\sigma^2}\right]$$

Basically

- We have that the normalization layer is more robust to the scaling of the input and parameters

Where

We have that $a_i = w_i^T \boldsymbol{x}$

- We project the gradient updates to the gain parameter δ_{gi} of the i^{th} neuron to its weight vector as

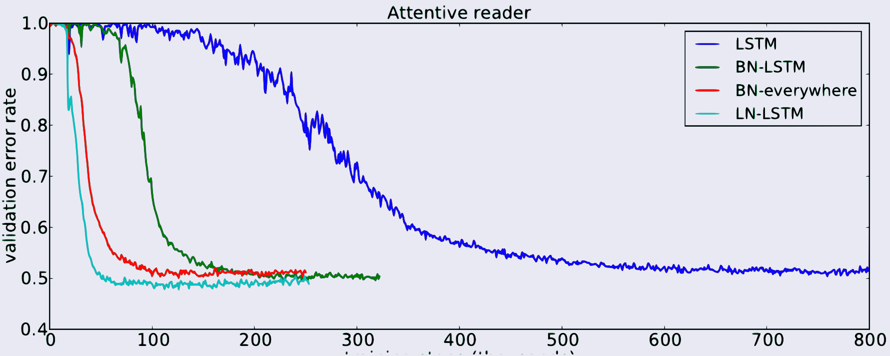
$$\frac{\delta_{gi}\delta_{gj}}{2\Phi^2} E_{x\sim P(\boldsymbol{x})}\left[Cov\left(y_i, y_j|\boldsymbol{x}\right)\frac{\left(a_1-\mu\right)\left(a_H-\mu\right)}{\sigma^2}\right]$$

Basically

- We have that the normalization layer is more robust to the scaling of the input and parameters

Results

In a LSTM



Outline

We have the following paper

Please Take a Look

- Kukačka, J., Golkov, V., & Cremers, D. (2017). Regularization for deep learning: A taxonomy. arXiv preprint arXiv:1710.10686.

T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. Springer Series in Statistics, Springer New York, 2009.

S. Theodoridis, *Machine Learning: A Bayesian and Optimization Perspective*.
Academic Press, 1st ed., 2015.

H. Zou, T. Hastie, and R. Tibshirani, "Sparse principal component analysis," *Journal of computational and graphical statistics*, vol. 15, no. 2, pp. 265–286, 2006.

Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

T. DeVries and G. W. Taylor, "Dataset augmentation in feature space," *arXiv preprint arXiv:1702.05538*, 2017.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

S. Wager, S. Wang, and P. S. Liang, "Dropout training as adaptive regularization," in *Advances in neural information processing systems*, pp. 351–359, 2013.

X. Bouthillier, K. Konda, P. Vincent, and R. Memisevic, "Dropout as data augmentation," *arXiv preprint arXiv:1506.08700*, 2015.

S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.