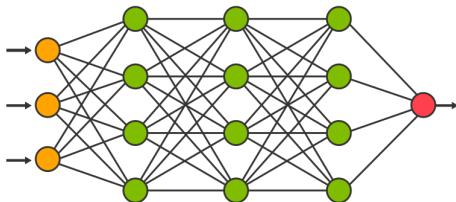


Introduction to Neural Networks

ML Instruction Team, Fall 2022

CE Department
Sharif University of Technology



Problem: OverFitting in a Neural Network

- Why does overfitting happen in a neural network?
 - ▷ There are **Too many free parameters**.

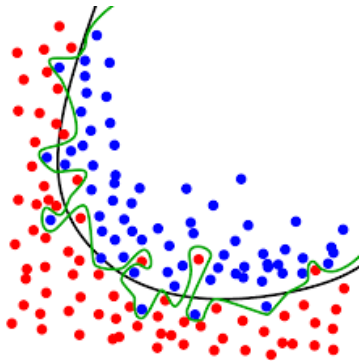


Figure: OverFitting in a neural network, [Source](#)

Solution 1: L1/L2 Regularization

- It is like a linear regression regularizer.
- Sum the regularizer term for every **layer weight**!

$$L = \frac{1}{N} \sum_{i=1}^N L(\phi(x_i), y_i) + \lambda \sum_{i,j,k} R(W_{j,k}^{(i)})$$

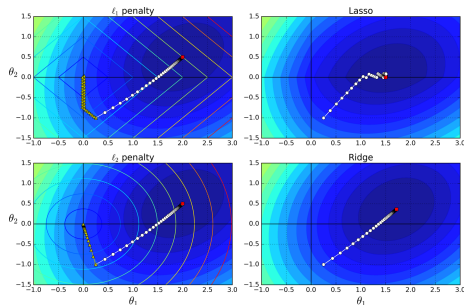


Figure: Convergence diagram for different losses,

Source

L1/L2 Regularization

L1/L2 regularizer functions (review)

$$L1 : R(w) = |w|$$

$$L2 : R(w) = w^2$$

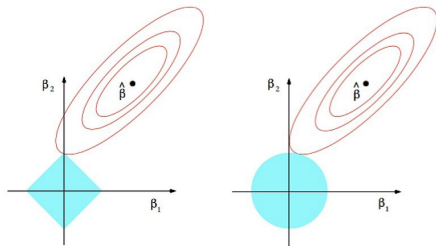


Figure: L1/L2 regularizers' solution diagram, [Source](#)

You can also combine the two different regularizers (Elastic Net).

$$R(w) = \beta w^2 + |w|$$

Solution 2: Early Stopping

- Stop the training procedure when the validation error is **minimum**.

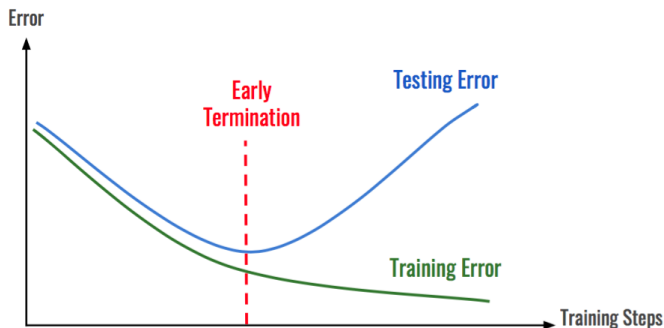


Figure: Early Stopping diagram, [Source](#)

Dropout: Training Time

- In each forward pass, **randomly** set some neurons to zero.
- The probability of dropping out for each neuron, which is called **dropout rate**, is a hyperparameter.
 - ▷ 0.5 is a common dropout rate.
- The probability of not dropping out is also called the **keep probability**.

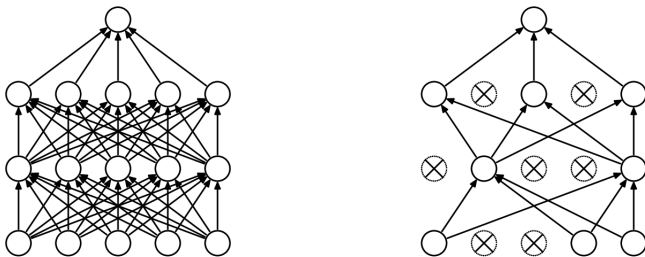


Figure: Behavior of dropout at training time, Source

Dropout: Why can this possibly be a good idea?

- Dropout-trained neurons are unable to **co-adapt** with their surrounding neurons.
- They also can't depend too heavily on a small number of input neurons.
- They become less responsive to even little input changes.
- The result is a stronger network that **generalizes** better.



Figure: Discrimination of neurons at training time. [1]

Dropout: Why can this possibly be a good idea?

- Dropout trains a **large ensemble of models** that share parameters.
- Every possible dropout state for neurons of a network, which is called a **mask**, is one model.
- A fully connected network with 4096 neurons has $2^{4096} \sim 10^{1233}$ possible masks! There are only 10^{82} atoms in the universe!

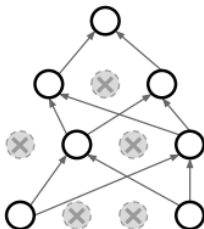


Figure: Behavior of dropout at training time. [1]

Dropout: Test Time

- Dropout makes our output random at training time.

$$y = f_W(x, \underbrace{z}_{\text{random mask}})$$

- We want to **average out** the randomness at test time,

$$y = f(x) = E_z[f(x, z)] = \int p(z) f(x, z) dz$$

- But this integral seems complicated.
- Let's approximate the integral for a superficial layer where dropout rate is 0.5.

Dropout: Test Time

$$\begin{aligned}
 E_{train}[a] &= \frac{1}{4}(w_1x + w_2y) + \frac{1}{4}(w_1x + 0y) \\
 &\quad + \frac{1}{4}(0x + w_2y) + \frac{1}{4}(0x + 0y) \\
 &= \frac{1}{2}(w_1x + w_2y)
 \end{aligned}$$

$$E_{test}[a] = w_1x + w_2y$$

$$\Rightarrow E_{test}[a] = \underbrace{0.5}_{\text{keep probability}} E_{train}[a]$$

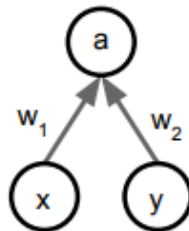


Figure: Simple neural network. [1]

Solution: Batch Norm Layer

- It is used to **normalize** the data.

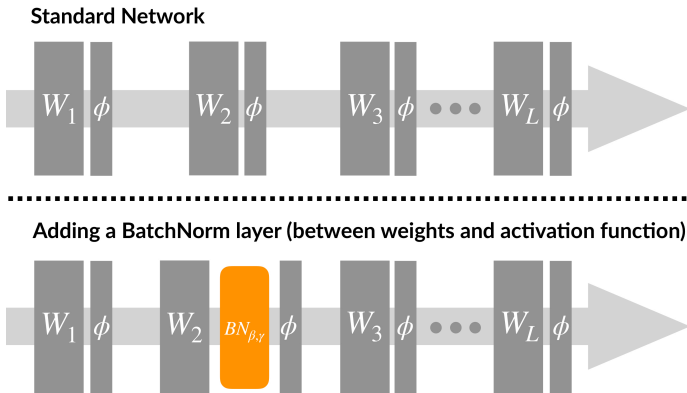


Figure: The suggested place to put a BatchNorm layer, **Source**

Batch Norm: Training Time

- First, it zero-centers and normalizes the batch.

$$\mu_B := \frac{1}{N_B} \sum x_B^{(i)}$$
$$\sigma_B^2 := \frac{1}{N_B} \sum (x_B^{(i)} - \mu_B)^2$$
$$\hat{x}_B^{(i)} = \frac{x_B^{(i)} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

- Then, scales and shifts the batch with two learnable parameters γ, β .

$$y_B^{(i)} = \gamma \hat{x}_B^{(i)} + \beta$$

Batch Norm: Test Time

- To zero-center and normalize the input, we need the average and variance of the whole data.
- Those parameters can be acquired during the training.
- Therefore, we need two more trainable parameters.

$$\mu_D := \frac{1}{N} \sum x^{(i)}$$
$$\sigma_D^2 := \frac{1}{N} \sum (x^{(i)} - \mu)^2$$

Batch Norm: Performance

- Normalizing the data improves the convergence speed by a considerable amount.

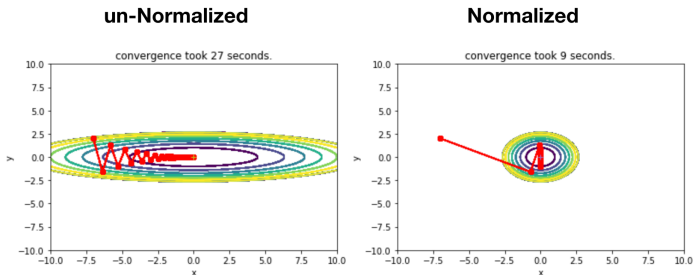


Figure: BatchNorm performance. Convergence speed is increased by 200%, [Source](#)

Batch Norm: Pros

- Vanishing/Exploding gradient problem is reduced by a considerable amount.
- You can use even saturating activation functions.
- The network is much less sensitive to the initial weight.
- We're able to use larger learning rates, which speeds up the training.
- It also acts as a regularizer.
 - ▷ There is no need for other regularizer techniques.

Batch Norm: Cons

- It increases model parameters and prediction latency.
 - ▷ Solution: after the training procedure, we can mix the BatchNorm layer with its previous layer to hold the prediction latency.

$$\begin{aligned}
 x'^{(i)} &= Wx^{(i)} + b \\
 y^{(i)} &= \frac{x'^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta
 \end{aligned}
 \Rightarrow
 \begin{aligned}
 y^{(i)} &= W'x^{(i)} + b' \\
 W' &:= \frac{1}{\sqrt{\sigma^2 + \epsilon}} W \\
 b' &:= \beta + \frac{b - \mu}{\sqrt{\sigma^2 + \epsilon}}
 \end{aligned}$$

- Where μ and σ are the mean and variance for the test time.

Thank You!

Any Question?

References



F.-F. L. . J. J. . S. Yeung, “Training neural networks,” 2018.

http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture07.pdf.