CS 480/680

2/26/26

# Lecture 13

Overfitting & DD
Normalizations
Dropout
Weight decay
...

Q2 – WP 20% 3/5
80% 3/10

LIV-2 updated

# Lecture Notes IV – Neural Networks, Part 2

Marina Meilă
mmp@uwaterloo.ca

With Thanks to Pascal Poupart & Gautam Kamath
Cheriton School of Computer Science
University of Waterloo

February 8, 2026

Training a single unit

Training a 2-layer network

Training a $L$-layer network

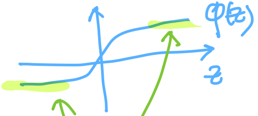
Backpropagation in practice

Practical remedies to I, T, S, L, O

**Reading** HTF Ch.: 11.3 Neural networks, Murphy Ch.: (16.5 neural nets), Bach Ch.: –, Deep Learning Book (Goodfellow, Bengio, Courville) 6.1-4, ResNet 7.6, ConvNet 9., Autoencoders 14.1, Dive Into Deep Learning 4.1-4.3.

# Backpropagation – some issues

- I Computation – how many ops / iteration? ✓
- T Convergence – how many iterations ($T$) ? ✓ # epochs = T
- S Saturation – $\phi'(x) \approx 0$ for large $|z|$ ←
- L Local minima ✓
- 0 Overfitting? ←




$\varphi'(z) \approx 0$

VANISHING gradients

1 epoch = 1 pass over $\mathcal{D}$

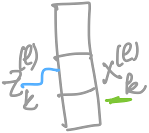
= 1 iteration GD

$= \dfrac{n}{n'}$ iterations of SGD ($n'$)

want: $\dfrac{\partial \mathcal{L}}{\partial W} = 0 = \sum_i \dfrac{\partial \mathcal{L}}{\partial W}(y^i, f(x^i; W))$

$W \in \mathbb{R}^p$

layer $\ell$



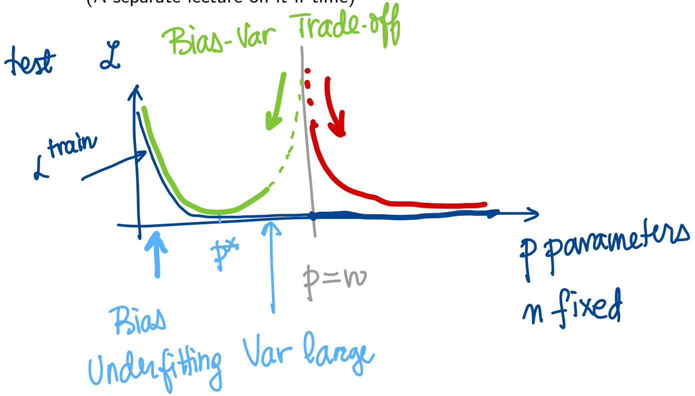

$|z_k^{(\ell)}(x^i)|$ large

# Overfitting

$p = \dim W = \#$ parameters

- $p$ can be very large deep networks
- Overfitting possible for everyday/in-house/not too large networks
- (Remedies later on in this lecture)

- Modern (very large) nn regime
  - Very large nn, with $p \gg n$ **are not** subject to the classical bias-variance tradeoff
  - **This was a surprising discovery!** The phenomenon is called **double descent**
  - (A separate lecture on it if time)



# Normalizations – Controlling saturation S,O

Std in very large nn

- These refer to data/inputs to neurons, not to weights
- **Classic standardization**

1) Recenter data around mean $x^i \leftarrow x^i - \mu$

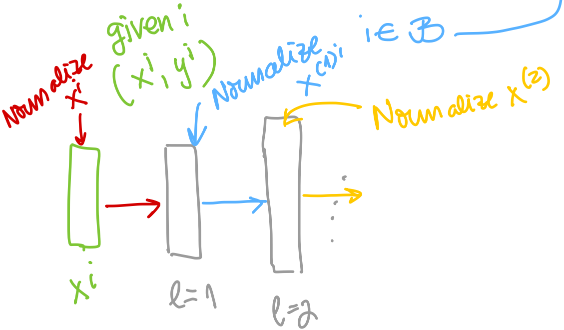
$$\boxed{\mu = \frac{1}{n}\sum_{i=1}^{n} x^i} \quad (33)$$

2) Rescale each data coordinate $j$: $x^i_j \leftarrow x^i_j / \sigma_j$

$$\boxed{\sigma_j^2 = \frac{1}{n}\sum_{i=1}^{n} (x^i_j)^2} \quad (34)$$

- Batch normalization – standardization within a batch
- Layer normalization – standardization within a layer

$\Rightarrow \{x^i_j\, y_j\}_{i=1:n}$ 0 mean, 1 = variance



# Batch normalization

1)
- Normalize the inputs in each $\mathcal{B}$
- Normalize the inputs to the sigmoids in each batch and layer $\{z^{i(l)}, i \in \mathcal{B}\}$
- Normalize the inputs to the sigmoids in each batch and neuron $\{z_j^{i(l)}, i \in \mathcal{B}\}, j = 1 : m_l, l = 1 : L - 1$
- Normalize the inputs to each layer, i.e. $x^{i(l)}$ for $x^i \in \mathcal{B}, l = 0 : L - 1$
- Then add learnable scale and shift (same for all batches), e.g. $x^i \leftarrow \sigma_0 \frac{x^i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} + \mu_o$

1) $\{x^i\}_{i \in \mathcal{B}} \xrightarrow{\text{Normalize}}$

2) $\{z_i^{(l)}\}_{i \in \mathcal{B}} \xrightarrow{\text{Normalize}}$ $l = 1 : L$ recursively

Store all $\mu$'s, $\sigma$'s

At Test time

For Prediction : $x \leftarrow \frac{x - \mu}{\sigma}$

# Layer normalization

- Standardize $\{z_j^{(l)}, j = 1 : m_l\}$ for each data point and each layer

# Avoiding overfitting **O**

Bias $\quad w_i$'s not too large !!

- Regularization, e.g. weight decay
- Dropout
- Data augmentation
- Early stopping
- Model averaging (also called **bagging**) or Bayesian Neural Networks

# Weight decay – O

$\mathcal{L}$ = TRAINING LOSS

- **Idea** Prevent overfitting by penalizing weights that are too large (Regularization)
- New "loss"

min

$$\min \rightarrow \mathcal{L}_\lambda(\mathbb{W}) = \underbrace{\mathcal{L}(\mathbb{W})}_{\text{no regularization}} + \underbrace{\frac{\lambda}{2}\|\mathbb{W}\|^2}_{\text{regularization term} > 0} \qquad (35)$$

and $\|\mathbb{W}\|^2 = \sum_{j=1}^{p} w_j^2$

$\lambda > 0$ reg. parameter

- Effect on gradient $g_\lambda = \frac{\partial \mathcal{L}_\lambda}{\partial \mathbb{W}}$

$$\rightarrow \frac{\partial \mathcal{L}_\lambda}{\partial w_j} = \frac{\partial}{\partial w_j}\left(\mathcal{L}(\mathbb{W}) + \frac{\lambda}{2}\|\mathbb{W}\|^2\right) = \overbrace{\boxed{\frac{\partial \mathcal{L}(\mathbb{W})}{\partial w_j}}}^{g_j} + \underbrace{\lambda w_j} \qquad (36)$$

$$w_j^{t+1} \leftarrow w_j^t - \eta\left(\frac{\partial \mathcal{L}(\mathbb{W})}{\partial w_j} + \lambda w_j^t\right) = \underset{=}{w_j^t} - \eta\boxed{\frac{\partial \mathcal{L}(\mathbb{W})}{\partial w_j}} - \overset{\eta'}{\eta\lambda}\underset{=}{w_j^t} \qquad (37)$$

$$= w_j^t\underbrace{(1-\eta\lambda)}_{<1} - \eta\boxed{\frac{\partial \mathcal{L}(\mathbb{W})}{\partial w_j}} \qquad (38)$$

decay

$g_j$

same as GD before

$$\|\mathbb{W}\|^2 = \sum_{j=1}^{p} w_j^2 \Rightarrow \frac{\partial}{\partial w_j}\sum_{j'=1}^{n} w_{j'}^2 = \underline{2w_j}$$

# Dropout

## Dropout

- Idea: randomly "drop" some units from the network when training
- Training: at each iteration of gradient descent
  - Each input unit is dropped with probability $p_1$ (e.g., 0.2)
  - Each hidden unit is dropped with probability $p_2$ (e.g., 0.5)
- Prediction (testing):
  - Multiply each input unit by $1 - p_1$
  - Multiply each hidden unit by $1 - p_2$



https://www.cs.toronto.edu/ rsalakhu/papers/srivastava14a.pdf "Dropout: A simple way to prevent neural networks from overfitting" by Srivastava, Hinton, Krizhevsky, Sutskever, Salkhutdinov, JMLR 2014.

# Training with dropout

## Train

- For each iteration
  - For each training example $(x^i, y^i)$
    1. Sample $r^{(l)} \in \{0,1\}^{m_l}$ from Bernoulli$(p_{\text{drop}}^{(l)})$, for $l = 1 : L$
    2. Backward propagation: skip/ignore dropped out units (do not compute gradient contribution from them)
    3. Update: only $w_i^{(l)}$ with $r_i^{(l)} = 1$

## Test/Predict

- Forward propagation $x^{(l)} = \phi(W^{(l)} z^{(l)} (1 - p_{\text{drop}}^{(l)})$

use all units

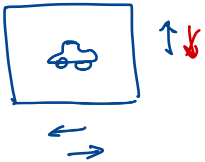
# Intuition

- Dropout can be viewed as an approximate form of ensemble learning

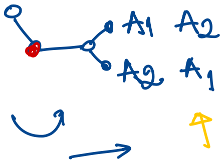
- In each training iteration, a different subnetwork is trained
- At test time, these subnetworks are "merged" by averaging their weights

# Data augmentation – O → Better solution incorporate symmetries in model

If there are symmetries in X, then create synthetic examples



X




artificial X'

$\mathcal{D} \longrightarrow \mathcal{D}^{S}$

and all synthetic examples