Lecture 13

Overfitting & DD
Weight decay
Dropout
Normalizations
...

OH → 9:30, 10:30 MC 2505

HW5 out - Pb2 needs corrections

Q2 3/10 or 3/12

Sol 4 - available tomorrow

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 ← Overfitting ← Saturation

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

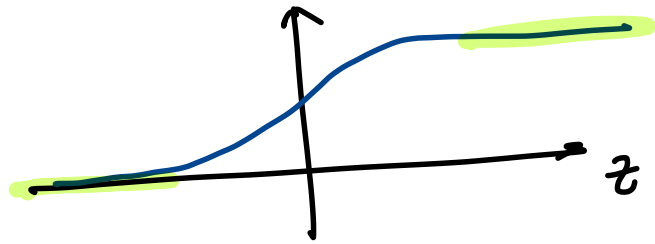
I Computation – how many ops / iteration?

T Convergence – how many iterations (T) ?

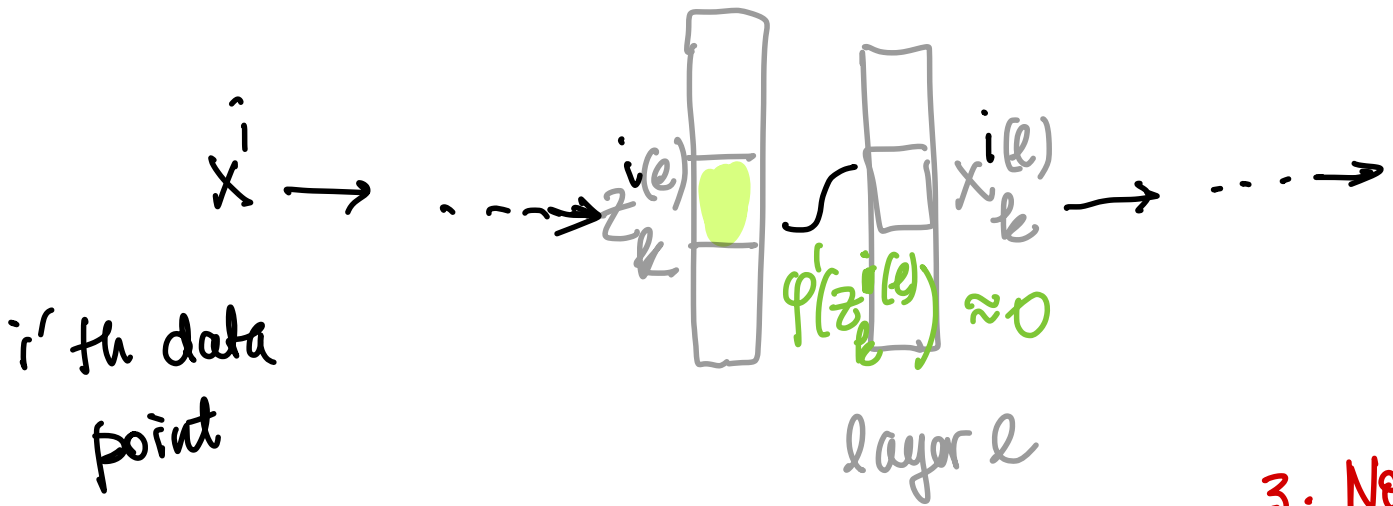
→ S Saturation – $\phi'(x) \approx 0$ for large $|z|$ = vanishing gradients

L Local minima

→ 0 Overfitting?



$|z|$ large $\Rightarrow \varphi'(z) \approx 0$



Fixes:

1. ReLU

2. initialization

$W_j \approx 0$

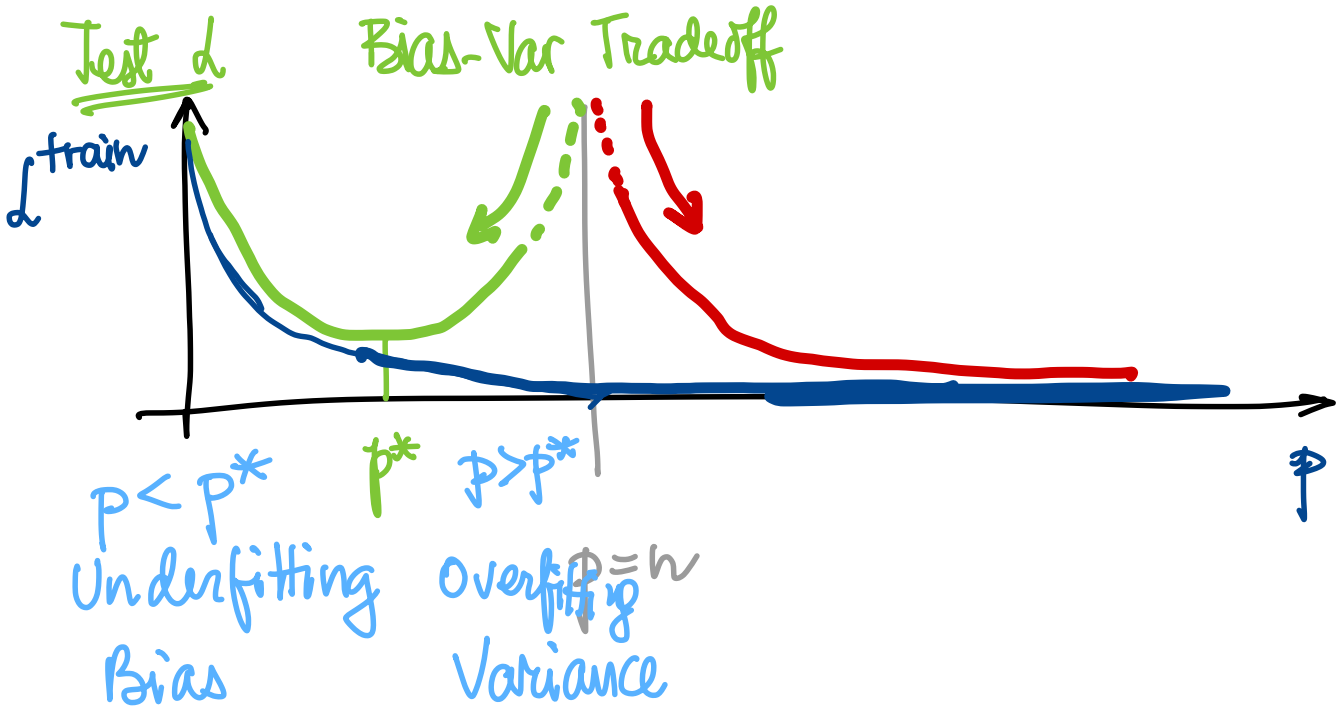
3. Normalization $j = 1:p$

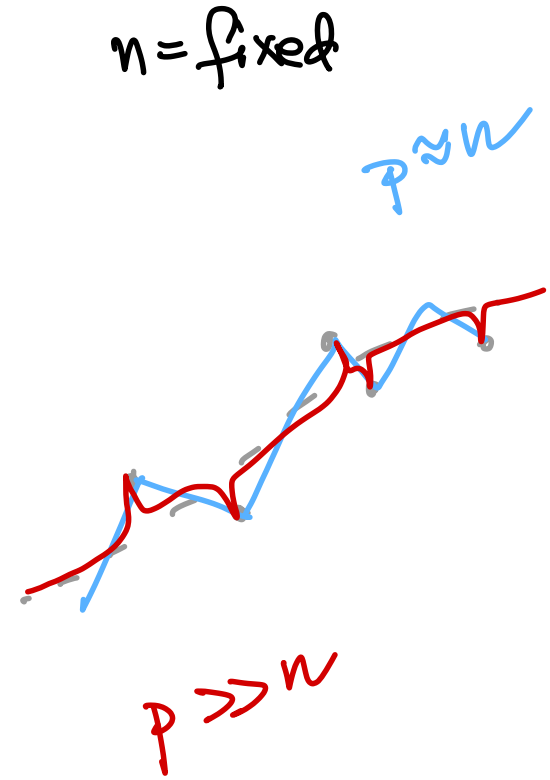
Overfitting

p = #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

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

1) Recenter data around mean $x^i \leftarrow x^i - \mu$ $\quad \mu = \frac{1}{n}\sum_{i=1}^{n} x^i$ (33)

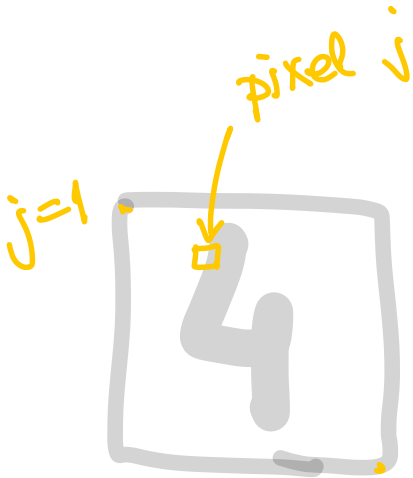
2) Rescale each data coordinate j $x_j^i \leftarrow x_j^i / \sigma_j$ $\quad \sigma_j^2 = \frac{1}{n}\sum_{i=1}^{n}(x_j^i)^2$ (34)

$j = 1:d$

$\sigma_j = \sqrt{\sigma_j^2}$

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

$x_j \sim$ mean $= 0$, std $= 1$ $\quad j = 1:d$



$\{x_j^i\}$ ---> Normalize $\quad i = 1:n$

Batch normalization

- 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. $x^i \leftarrow \sigma_0 \frac{x^i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \mu_o$

$x^{i(0)} \equiv x^i$

$x^{i(l)}$ input to layer $l+1$

Layer normalization

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

Avoiding overfitting O

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

Weight decay – 0

train for min

Bias

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

$$\mathcal{L}_\lambda(\mathbb{W}) = \mathcal{L}(\mathbb{W}) + \frac{\lambda}{2}\|\mathbb{W}\|^2 \tag{35}$$

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

Regularization term

$\lambda > 0$ reg. param.

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

$$\frac{\partial \mathcal{L}_\lambda}{\partial w_j} = \frac{\partial}{\partial w_j}\left(\mathcal{L}(\mathbb{W}) + \frac{\lambda}{2}\|\mathbb{W}\|^2\right) = \frac{\partial \mathcal{L}(\mathbb{W})}{\partial w_j} + \lambda w_j \tag{36}$$

g_j

$$w_j^{t+1} \leftarrow w_j^t - \eta\left(\frac{\partial \mathcal{L}(\mathbb{W})}{\partial w_j} + \lambda w_j^t\right) = w_j^t - \eta\frac{\partial \mathcal{L}(\mathbb{W})}{\partial w_j} - \eta\lambda w_j^t \tag{37}$$

w_j

old update

$$= w_j^t(1 - \eta\lambda) - \eta\frac{\partial \mathcal{L}(\mathbb{W})}{\partial w_j} \tag{38}$$

$g_j = \frac{\partial \mathcal{L}}{\partial w_j}$

old

< 1

$$\frac{\partial}{\partial w_j}\left(\frac{1}{2}\sum_{j'=1}^{p} w_{j'}^2\right) = w_j$$

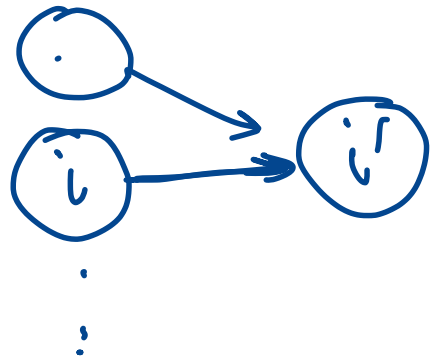
Dropout — specific nn

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 $1 - p_2$

= inputs

no drop out



CS480/680 Winter 2023 - Lecture 11 - Pascal Poupart PAGE 20 UNIVERSITY OF WATERLOO

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($p^{(l)}_{\text{drop}}$), for $l = 1 : L$
    2. Backward propagation: skip/ignore dropped out units (do not compute gradient contribution from them)
    3. Update: only $w^{(l)}_i$ with $r^{(l)}_i = 1$

Test/Predict

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

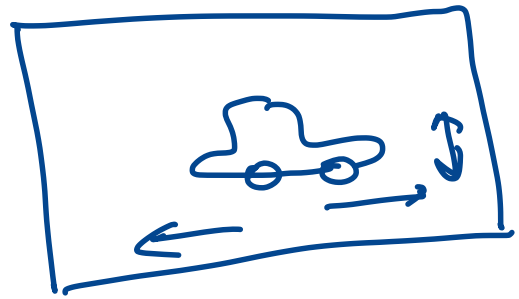
Intuition

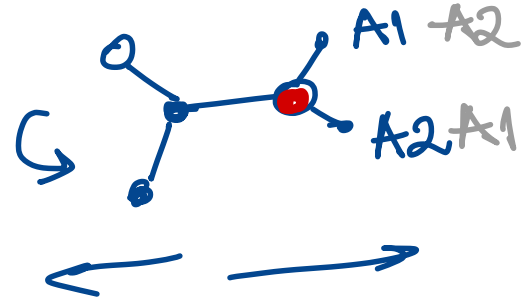
- Dropout can be viewed as an approximate form of ensemble learning

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

Data augmentation – O

x has symmetries
invariances





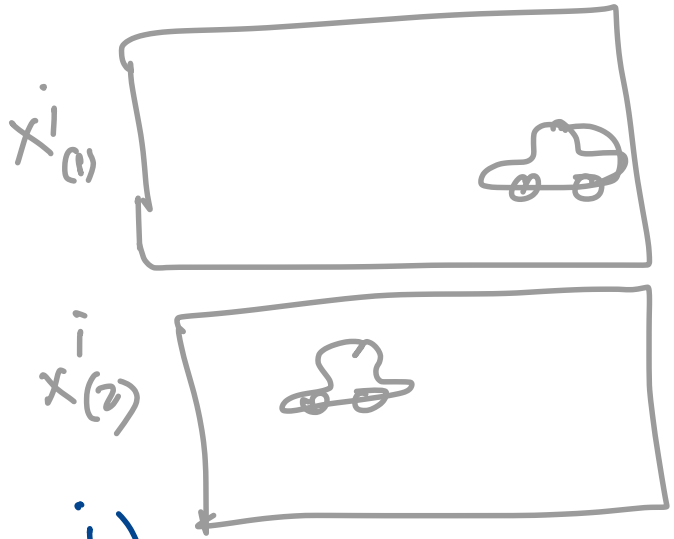
x^i

create synthetic

$x^i_{(1,2,\dots)}$ from x^i

to "teach" nn

about invariance



$$(x^i, y^i) \rightarrow (x^i_{(1,2,\dots)}, y^i)$$

Drawbacks: $n \rightarrow Sn \gg n$

Better : Invariant / Equivariant nn.

Early stopping (Regularization)