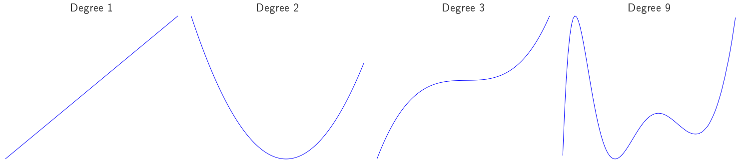
EC-332 Machine Learning

Regularization in Neural Networks

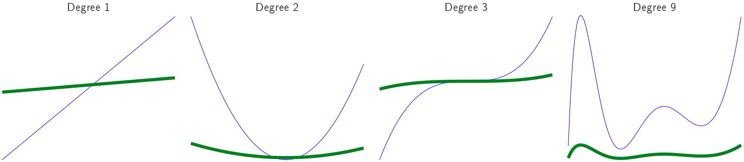
Nazar Khan
Department of Computer Science
University of the Punjab

Before we start
A primer on ML

1. Capabilities of polynomials (lines, quadratics, cubics, ..., degree M).



2. Capability can be reduced by restricting coefficients.



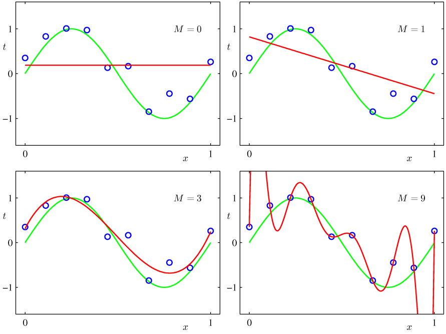
green curve = blue curve divided by 10

Before we start
A primer on ML

3. Everything is noisy.

$$\text{Observation} = \text{Reality} + \text{Noise}$$

4. Therefore, zero *training* error is bad. Over-fitting vs generalisation.



5. Over-fitting can be reduced via regularization.

Weight Penalties

- Similar to polynomials, networks with large weights are more powerful.
- Therefore, more prone to overfitting.
- So penalise magnitudes of weights to restrict capability.

$$\tilde{L}(\mathbf{w}) = L(\mathbf{w}) + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

- *Hyperparameter*[1] λ controls the level of overfitting.
- Alternative: separately penalise each layer

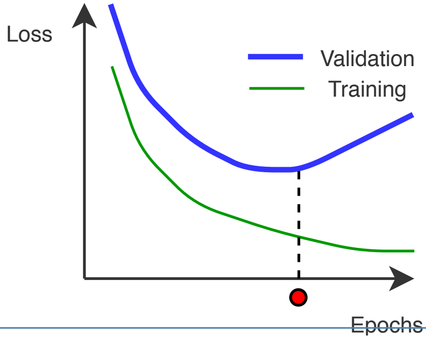
$$\tilde{L}(\mathbf{w}) = L(\mathbf{w}) + \sum_{l=1}^{L} \frac{\lambda_l}{2}\|\mathbf{w}^{(l)}\|^2$$

Not used often due to increased number of hyperparameters.

[1] Something that is not a parameter but influences what the parameters will be.

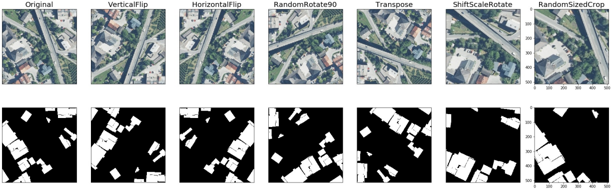
Early Stopping

- Split some part of the training set into a validation set that will not be used for training.
- During training, record loss on training as well as validation set.
- When validation loss starts increasing while training loss is still going down, the model has started overfitting.
- So stop training at that point.



Data Augmentation

- Augment training set with transformed versions of training samples.
- Domain specific data augmentations
  - Images: Color, Geometry
  - Text: Synonyms, Tense, Order
  - Speech: Speed, Sound effects



`https://github.com/albumentations-team/albumentations`

Data Augmentation

https://github.com/aleju/imgaug

Label Smoothing

- Training adjusts the model to make outputs as close as possible to the targets/labels.
- So if labels are smoothed a little, overfitting will be reduced.
- For example, if label 0 is mapped to 0.1 and 1 is mapped to 0.9, training will converge early.
- Training procedure will not try as hard as before to output as close as possible to 0 or 1.
- Leads to well-calibrated neural networks.

Dropout

- One of the most used regularization techniques in neural nets.
- *During training*, a randomly selected subset of activations are set to zero within each layer.
- This makes the neural network less powerful.
- Dropout layer implementation is very simple.
  - For each neuron (including inputs),
    1. Generate a uniform random number between 0 and 1.
    2. If the number is greater than α, set the neuron's output to 0.
    3. Otherwise, don't touch the neuron's output.
- Probably of dropping out is $1 - \alpha$.
- *Remember* which neurons were dropped so that gradients are also zeroed out during backpropagation.

Detour – Bagging

- Bagging is a popular ML meta-algorithm.
- Multiple ML models are trained separately to solve the same problem on *separate subsets* of the training data.
- Final answer is the average of all models.

$$F(x) = \frac{1}{M}\sum_{m=1}^{M} f_m(x)$$

- Bagging results are usually better than the best individual model.
- Dropout can be viewed as bagging.

Dropout as Bagging

- An architecture with n neurons can have 2^n sub-architectures depending on which neurons are switched off.
- Whenever a random subset of neurons is switched off, we are essentially training only one of the 2^n sub-architectures.
- At test time, use expected output of neuron, $E[y] = \alpha h(a)$, i.e., bagging.

y	0	$h(a)$
$P(y)$	$1-\alpha$	α

- Alternatives:
  1. Push α into the next layer's weights after training and do testing as before.

$$
\begin{aligned}
z_k &= \sum w_{kj} y_j + b_k \\
&= \sum w_{kj} \alpha h(a_j) + b_k = \sum \underbrace{(\alpha w_{kj})}_{\tilde{w}_{kj}} h(a_j) + b_k
\end{aligned}
$$

  2. During training, multiply every output by $\frac{1}{\alpha}$ and do testing as before.

Dropout vs. DropConnect

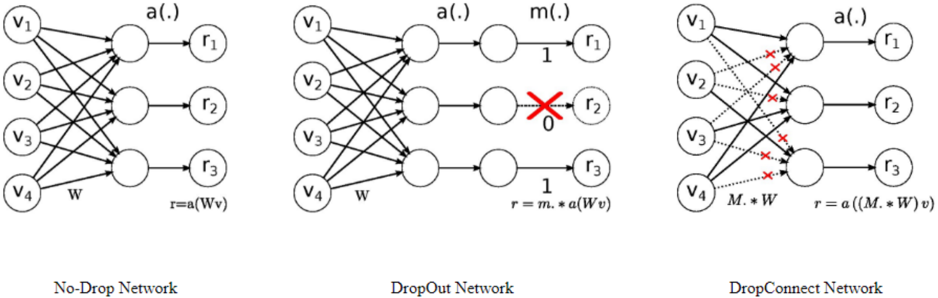
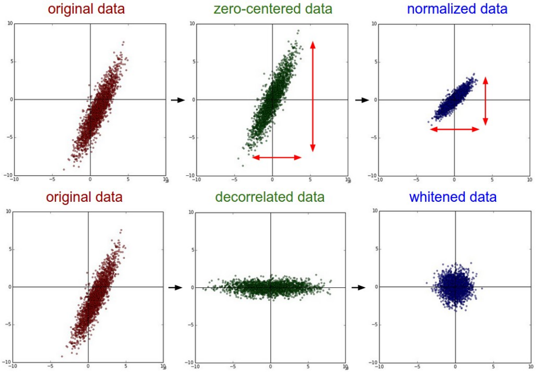


Figure: Dropout vs. DropConnect[3]. Image taken from `https://cs.nyu.edu/~wanli/dropc/`

[3]Wan et al., 'Regularization of Neural Network using DropConnect'.

Normalisation

- The importance of normalising inputs is well-understood in ML.
- Improves numerical stability and reduces training time.
- Makes all features equally important before learning takes place.



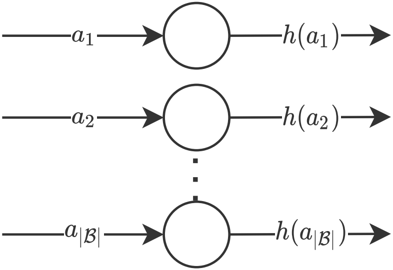
Normalisation of 2D data. Taken from
`http://cs231n.github.io/neural-networks-2/`

Batch Normalisation

- In neural networks, a neuron's input depends on previous neurons' outputs.
- Those outputs can vary wildly during training as the weights are adjusted.
- Normalising the input sample is not enough.
- Later neuron's input needs to be normalised as well.
- Inputs to every neuron in every layer must be normalised *in a differentiable manner*.
- Normalisation is useless for learning if gradient ignores it.

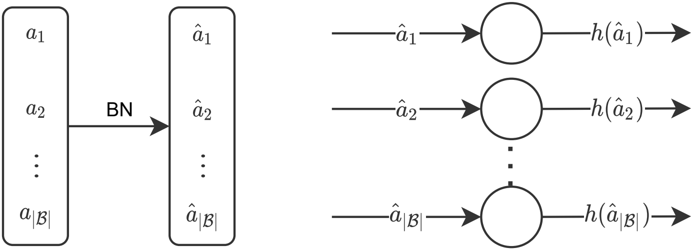
Batch Normalisation

- For the i-th input sample, a neuron passes its pre-activation a_i into its activation function $h(a_i)$.
- For a minibatch $\mathcal{B}$, the neuron will perform this step for each input sample in $\mathcal{B}$ *separately*.



- BatchNorm takes place between this step.

Batch Normalisation

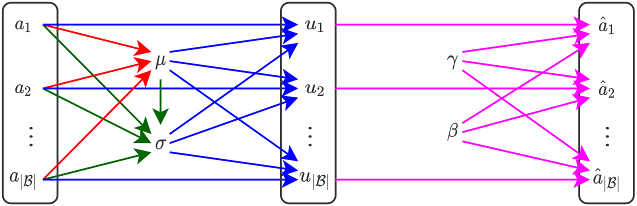


- Each a_i is converted to $\hat{a}_i$ by looking at the other a_j values in the minibatch.
- Instead of a_i, the new $\hat{a}_i$ is passed into the activation function.

Batch Normalisation

Consider a neuron's pre-activations $a_1, a_2, \ldots, a_{|\mathcal{B}|}$ over a minibatch $\mathcal{B}$.

1. Compute mean $\mu = \frac{\sum a_i}{|\mathcal{B}|}$.
2. Compute variance $\sigma^2 = \frac{\sum (a_i - \mu)^2}{|\mathcal{B}|}$.
3. Standardize the pre-activations as $u_i = \frac{a_i - \mu}{\sigma}$.
   This makes the set $u_1, u_2, \ldots, u_{|\mathcal{B}|}$ have zero-mean and unit-variance.
4. Recover expressive power by **learnable** transformation $\hat{a}_i = \gamma u_i + \beta$.



Batch Normalisation

The $\hat{a}_i$ values that are now passed into the activation function will have mean β and standard deviation γ, *irrespective of original moments μ and σ* for the minibatch.

The whole process is differentiable and therefore suitable for gradient descent.

Benefits of BatchNorm

- Avoids vanishing gradients for sigmoidal non-linearities.
- Allows much higher learning rates and therefore dramatically speeds up training.
- Reduces dependence on good weight initialisation.
- Regularizes the model and reduces the need for dropout.

BatchNorm at testing time

- Testing is not done on minibatches.
- But each neuron trained itself on batchnormed pre-activations.
- It expects batchnormed pre-activations at testing time as well.
- *Solution*: Once the network is trained, for *each neuron*, compute the average μ, σ^2 over the set $\mathcal{S}$ of all training minibatches.

$$\mu_{\text{test}} = \frac{1}{|\mathcal{S}|} \sum_{\mathcal{B} \in \mathcal{S}} \mu(\mathcal{B})$$

$$\sigma^2_{\text{test}} = \frac{|\mathcal{B}|}{|\mathcal{B}| - 1} \frac{1}{|\mathcal{S}|} \sum_{\mathcal{B} \in \mathcal{S}} \sigma^2(\mathcal{B})$$

- $\frac{|\mathcal{B}|}{|\mathcal{B}|-1}$ for computing unbiased estimator of variance.
- Use $\mu_{\text{test}}, \sigma_{\text{test}}$ to normalize every testing sample.

Layer Normalization

- BatchNorm violates the *i.i.d* assumption by making one training sample's output *dependent* on other *randomly chosen* training samples.
- LayerNorm is an alternative method that normalizes based on the activations of a layer for a single training sample as

$$u_i = \frac{a_i - \mu}{\sigma}$$

where $\mu = \frac{1}{M}\sum_{i=1}^{M} a_i$ and $\sigma = \sqrt{\frac{1}{M}\sum_{i=1}^{M}(a_i - \mu)^2}$ are computed from the activations $a_1, a_2, \ldots, a_M$ for a layers with M neurons.

- Improves training time and generalization performance of models dealing with sequential data (RNNs and Transformers).
- No constraint on mini-batch size. Can work with batch size of 1 (online training).
- Same operations at training and testing time.

Summary

- All data contains noise.
- Given enough power, a neural network will model noise as well.
- Restricting the network's power allows it to model the underlying behaviour of data instead of noise.
- This reduces over-fitting on training data and improves generalisation of the network on unseen data.