# Deep Neural Networks (2) ReLU layers; Generalisation and Regularisation

Hakan Bilen

Machine Learning Practical — MLP Lecture 4 8 October 2019

#### Recap: Training multi-layer networks



Are there alternatives to Sigmoid Hidden Units?

# Sigmoid function



#### tanh



$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$
  
sigmoid(x) = 
$$\frac{1 + tanh(x/2)}{2}$$

Derivative:  $\frac{d}{dx} \tanh(x) = 1 - \tanh^2(x)$ 

# tanh hidden units (TanhLayer)

- tanh has same shape as sigmoid but has output range  $\pm 1$
- Results about approximation capability using sigmoid layers also apply to tanh layers
- Possible reason to prefer tanh over sigmoid: allowing units to be positive or negative allows gradient for weights into a hidden unit to have a different sign
- Saturation still a problem



#### **Rectified Linear Unit – ReLU**



$$\operatorname{relu}(x) = \max(0, x)$$

#### Derivative:

$$\frac{d}{dx}\operatorname{relu}(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases}$$

# ReLU hidden units (ReluLayer)

- Similar approximation results to tanh and sigmoid hidden units
- Empirical results for speech and vision show consistent improvements using relu over sigmoid or tanh
- Unlike tanh or sigmoid there is no positive saturation saturation results in very small derivatives (and hence slower learning)
- Negative input to relu results in zero gradient (and hence no learning)
- Relu is computationally efficient: max(0, x)
- Relu units can "die" (i.e. respond with 0 to everything)
- Relu units can be very sensitive to the learning rate

#### **ReLU Variants: Leaky and Parametric ReLU**



#### **ReLU Variants: Leaky and Parametric ReLU**

- Leaky ReLU has a small predetermined slope for negative values,  $y = \alpha x$  (e.g.  $\alpha = 0.01$ )
- Parametric ReLU (PReLU) is a type of Leaky ReLU but  $\alpha$  is learned during training
- It fixes the "dying ReLU" problem, no saturation for x < 0
- Gradient is not sparse
- Not always superior to ReLU

# Generalisation

#### Generalization

- Generalization:
  - what is the expected error on a test set?
  - how to compare the accuracy of different networks trained on the same data?
- Causes of error
  - Network too flexible: Too many weights compared with number of training examples
  - Network not flexible enough: Not enough weights (hidden units) to represent the desired mapping

When comparing models, it can be helpful to compare systems with the same number of *trainable parameters* (i.e. the number of trainable weights in a neural network)

• Optimizing training set performance does not necessarily optimize test set performance....

# Training / Test / Validation Data

- Partitioning the data...
  - **Training** data data used for training the network
  - Validation data frequently used to measure the error of a network on "unseen" data (e.g. after each epoch)
  - Test data less frequently used "unseen" data, ideally only used once
- Frequent use of the same test data can indirectly "tune" the network to that data (e.g. by influencing choice of *hyperparameters* such as learning rate, number of hidden units, number of layers, ....)

#### Measuring generalisation

- Generalization Error The predicted error on unseen data. How can the generalization error be estimated?
  - Training error?

$$E_{ ext{train}} = -\sum_{ ext{training set}} \sum_{k=1}^{K} t_k^n \ln y_k^n$$

• Validation error?

$$E_{\mathsf{val}} = -\sum_{\mathsf{validation set } k=1}^{K} t_k^n \ln y_k^n$$

- Optimize network performance given a fixed training set
- *Hold out* a set of data (validation set) and predict generalization performance on this set
  - 1. Train network in usual way on training data
  - 2. Estimate performance of network on validation set
- If several networks trained on the same data, choose the one that performs best on the validation set (**not** the training set)
- *n-fold* Cross-validation: divide the data into *n* partitions; select each partition in turn to be the validation set, and train on the remaining (*n* 1) partitions. Estimate generalization error by averaging over all validation sets.

- Overtraining corresponds to a network function too closely fit to the training set (too much flexibility)
- Undertraining corresponds to a network function not well fit to the training set (too little flexibility)
- Solutions
  - If possible increasing both network complexity in line with the training set size
  - Use prior information to constrain the network function
  - Control the flexibility: **Structural Stabilization**
  - Control the effective flexibility: early stopping and regularization

Directly control the number of weights:

- Compare models with different numbers of hidden units
- Start with a large network and reduce the number of weights by pruning individual weights or hidden units
- Weight sharing use prior knowledge to constrain the weights on a set of connections to be equal.
  - $\rightarrow$  Convolutional Neural Networks

Lab 4 explores overfitting and how we can measure how well the models we train generalise their predictions to unseen data.

- Setting up a 1-dimension regression problem
- Using a radial basis functions (RBF) network as a model for this problem
- Exploring the behaviour of the RBF network as the number of model parameters (basis functions) increases

- Use validation set to decide when to stop training
- Training Set Error monotonically decreases as training progresses

- Use validation set to decide when to stop training
- Training Set Error monotonically decreases as training progresses
- Validation Set Error will reach a minimum then start to increase
- Best generalization predicted to be at point of minimum validation set error

# **Early Stopping**

- Use validation set to decide when to stop training
- Training Set Error monotonically decreases as training progresses
- Validation Set Error will reach a minimum then start to increase
- Best generalization predicted to be at point of minimum validation set error



- Use validation set to decide when to stop training
- Training Set Error monotonically decreases as training progresses
- Validation Set Error will reach a minimum then start to increase
- Best generalization predicted to be at point of minimum validation set error
- "Effective Flexibility" increases as training progresses
- Network has an increasing number of "effective degrees of freedom" as training progresses
- Network weights become more tuned to training data
- Very effective used in many practical applications such as speech recognition and optical character recognition

# **Early Stopping**

- Use validation set to decide when to stop training
- Training Set Error monotonically decreases as training progresses
- Validation Set Error will reach a minimum then start to increase
- Best generalization predicted to be at point of minimum validation set error



# **Early Stopping**



- Regularisation penalise the weights: L1 (sparsity), L2 (weight decay)
- Data augmentation generate additional (noisy) training data
- Model combination smooth together multiple networks
- Dropout randomly delete a fraction of hidden units each minibatch
- Parameter sharing e.g. convolutional networks

# Weight Decay (L2 Regularisation)

- Weight decay puts a "spring" on weights
- If training data puts a consistent force on a weight, it will outweigh weight decay
- If training does not consistently push weight in a direction, then weight decay will dominate and weight will decay to 0
- Without weight decay, weight would walk randomly without being well determined by the data
- Weight decay can allow the data to determine how to reduce the effective number of parameters

#### **Penalizing Complexity**

• Consider adding a *complexity term*  $E_w$  to the network error function, to encourage smoother mappings:

$$E^{n} = \underbrace{E_{\text{train}}^{n}}_{\text{data term}} + \underbrace{\beta E_{W}}_{\text{prior term}}$$

#### **Penalizing Complexity**

• Consider adding a *complexity term*  $E_w$  to the network error function, to encourage smoother mappings:

$$E^n = \underbrace{E_{\text{train}}^n}_{\text{train}} + \underbrace{\beta E_W}_{\text{train}}$$

data term prior term

• *E*<sub>train</sub> is the usual error function:

$$\boxed{E_{\text{train}}^n} = -\sum_{k=1}^K t_k^n \ln y_k^n$$

#### **Penalizing Complexity**

• Consider adding a *complexity term*  $E_w$  to the network error function, to encourage smoother mappings:

$$E^n = \underbrace{E_{\text{train}}^n}_{\text{train}} + \underbrace{\beta E_W}_{\text{train}}$$

data term prior term

• *E*<sub>train</sub> is the usual error function:

$$\underbrace{E_{\text{train}}^n}_{k=1} = -\sum_{k=1}^K t_k^n \ln y_k^n$$

•  $E_W$  should be a differentiable flexiblity/complexity measure, e.g.

$$\overline{E_W = E_{L2}} = \frac{1}{2} \sum_i w_i^2$$
$$\overline{\frac{\partial E_{L2}}{\partial w_i}} = w_i$$

$$\begin{aligned} \overline{\frac{\partial E^n}{\partial w_i}} &= \frac{\partial (E_{\text{train}}^n + E_{L2})}{\partial w_i} = \left(\frac{\partial E_{\text{train}}^n}{\partial w_i} + \beta \frac{\partial E_{L2}}{\partial w_i}\right) \\ &= \left(\frac{\partial E_{\text{train}}^n}{\partial w_i} + \beta w_i\right) \\ \Delta w_i &= -\eta \left(\frac{\partial E_{\text{train}}^n}{\partial w_i} + \beta w_i\right) \end{aligned}$$

- Weight decay corresponds to adding  $E_{L2} = 1/2 \sum_{i} w_i^2$  to the error function
- Addition of complexity terms is called *regularisation*
- When used with gradient descent, weight decay corresponds to L2 regularisation

# L1 Regularisation

• L1 Regularisation corresponds to adding a term based on summing the absolute values of the weights to the error:

$$E^{n} = \underbrace{E_{\text{train}}^{n}}_{\text{data term}} + \underbrace{\beta E_{L1}^{n}}_{\text{prior term}}$$
$$= E_{\text{train}}^{n} + \beta |w_{i}|$$

• Gradients

$$\frac{\partial E^{n}}{\partial w_{i}} = \frac{\partial E_{\text{train}}^{n}}{\partial w_{i}} + \beta \frac{\partial E_{L1}}{\partial w_{i}}$$
$$= \frac{\partial E_{\text{train}}^{n}}{\partial w_{i}} + \beta \operatorname{sgn}(w_{i})$$

Where sgn $(w_i)$  is the sign of  $w_i$ : sgn $(w_i) = 1$  if  $w_i > 0$  and sgn $(w_i) = -1$  if  $w_i < 0$ 

# L1 vs L2

- L1 and L2 regularisation both have the effect of penalising larger weights
  - In L2 they shrink to 0 at a rate proportional to the size of the weight  $(\beta w_i)$
  - In L1 they shrink to 0 at a constant rate  $(\beta \operatorname{sgn}(w_i))$
- Behaviour of L1 and L2 regularisation with large and small weights:
  - when  $|w_i|$  is large L2 shrinks faster than L1
  - when  $|w_i|$  is small L1 shrinks faster than L2
- So L1 tends to shrink some weights to 0, leaving a few large important connections L1 encourages *sparsity*
- $\partial E_{L1}/\partial w$  is undefined when w = 0; assume it is 0 (i.e. take sgn(0) = 0 in the update equation)

#### Data Augmentation – Adding "fake" training data

- Generalisation performance goes with the amount of training data (change MNISTDataProvider to give training sets of 1000 / 5000 / 10000 examples to see this)
- Given a finite training set we could create further training examples...
  - Create new examples by making small rotations of existing data
  - Add a small amount of random noise
- Using "realistic" distortions to create new data is better than adding random noise

- Combining the predictions of multiple models can reduce overfitting
- Model combination works best when the component models are *complementary* no single model works best on all data points
- Creating a set of diverse models
  - Different NN architectures (number of hidden units, number of layers, hidden unit type, input features, type of regularisation, ...)
  - Different models (NN, SVM, decision trees, ...)
- How to combine models?
  - Average their outputs
  - Linearly combine their outputs
  - Train another "combiner" neural network whose input is the outputs of the component networks
  - Architectures designed to create a set of specialised models which can be combined (e.g. mixtures of experts)

Lab 5 explores different methods for regularising networks to reduce overfitting and improve generalisation

In the context of a feed-forward network using ReLU hidden layers, the lab explores

- L1 and L2 regularisation
- Data augmentation

#### Summary

- ReLU and its variants
- Generalisation and overfitting
- Preventing overfitting
  - L2 regularisation weight decay
  - L1 regularisation sparsity
  - Creating additional training data
  - Model combination
- Reading:
  - Nielsen, chapter 3 (section on overfitting and regularization) of *Neural Networks and Deep Learning*

http://neuralnetworksanddeeplearning.com/chap3.html#overfitting\_ and\_regularization

• Goodfellow et al, chapter 7 *Deep Learning* (sections 7.1-7.5) http://www.deeplearningbook.org/contents/regularization.html