Computational graphs, Pretraining

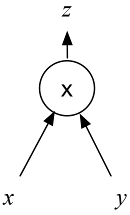
Steve Renals

Machine Learning Practical — MLP Lecture 6
25 October 2017 / 30 October 2017

Computational graphs

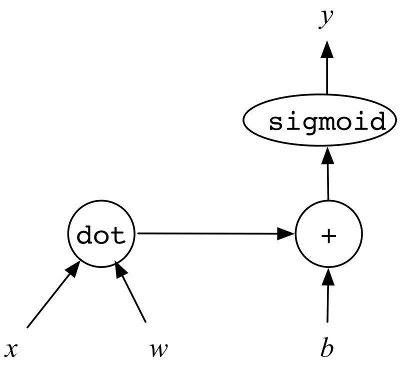
- Each node is an operation
- Data flows between nodes (scalars, vectors, matrices, tensors)
- More complex operations can be formed by composing simpler operations

Computational graph example 1



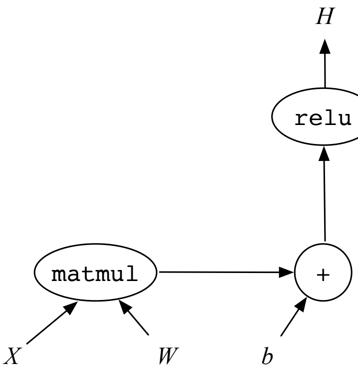
Graph for $\times$ to compute $z = xy$

Computational graph example 2

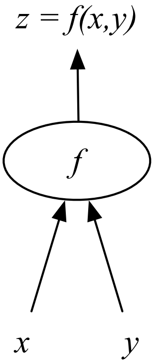


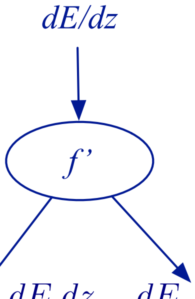
Graph for logistic regression:
$y = \text{sigmoid}(\boldsymbol{w}^\top \boldsymbol{x} + b)$

Computational graph example 3



Graph for ReLU layer:
$\boldsymbol{H} = \mathrm{relu}(\boldsymbol{W}\boldsymbol{X} + \boldsymbol{b})$





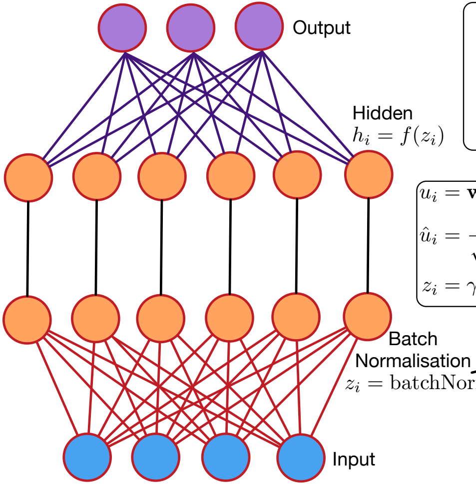
$$\frac{dE}{dx} = \frac{dE}{dz}\frac{dz}{dx} \qquad \frac{dE}{dy} = \frac{dE}{dz}\frac{dz}{dy}$$

Chain rule of differentiation as the backward pass through the computational graph

Computational graphs

- Each node is an operation
- Data flows between nodes (scalars, vectors, matrices, tensors)
- More complex operations can be formed by composing simpler operations
- Implement chain rule of differentiation as a backward pass through the graph
- Back-propagation: Multiply the local gradient of an operation with an incoming gradient (or sum of gradients)
- See http://colah.github.io/posts/2015-08-Backprop/

Batch Normalisation



$z_i = \mathrm{batchNorm}(\mathbf{w}_i \mathbf{x})$

$$\mu_i \leftarrow \frac{1}{M} \sum_{m=1}^{M} u_i^m$$

$$\sigma_i^2 \leftarrow \frac{1}{M} \sum_{m=1}^{M} (u_i^m - \mu_i)^2$$

Compute mean and variance of each hidden unit activation across the minibatch (size M)

$$u_i = \mathbf{w}_i \mathbf{x}$$

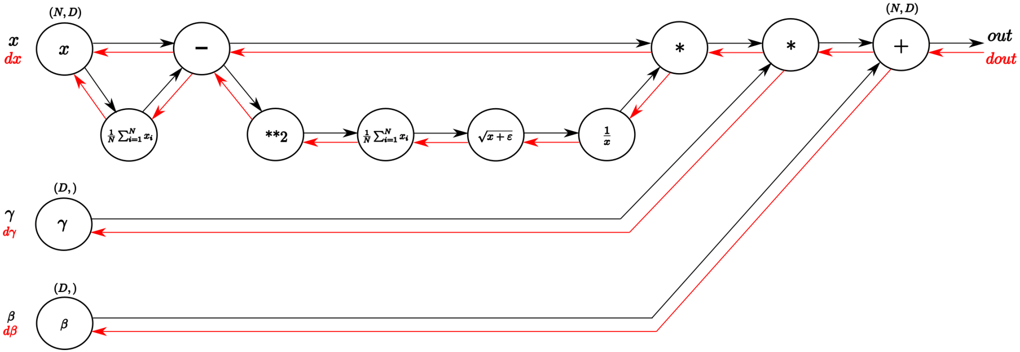
$$\hat{u}_i = \frac{u_i - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$$

$$z_i = \gamma_i \hat{u}_i + \beta_i = \mathrm{batchNorm}(u_i)$$

Ioffe & Szegedy, "Batch normalization", ICML-2015

http://www.jmlr.org/proceedings/papers/v37/ioffe15.html

Computational graph for batch normalisation



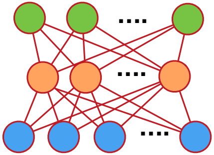
https://kratzert.github.io/2016/02/12/understanding-the-gradient-flow-through-the-batch-normalization-layer.html

Pretraining

- **Why is training deep networks hard?**
  - Vanishing (or exploding) gradients – gradients for layers closer to the input layer are computed multiplicatively using backprop
  - If sigmoid/tanh hidden units near the output saturate then back-propagated gradients will be very small
  - Good discussion in chapter 5 of *Neural Networks and Deep Learning*
- **Solve by stacked pretraining**
  - Train the first hidden layer
  - Add a new hidden layer, and train only the parameters relating to the new hidden layer. Repeat.
  - The use the pretrained weights to initialise the network – emphfine-tune the complete network using gradient descent
- **Approaches to pre-training**
  - Supervised: Layer-by-layer cross-entropy training
  - Unsupervised: Autoencoders
  - Unsupervised: Restricted Boltzmann machines (not covered in this course)

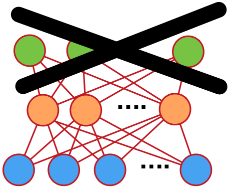
Greedy Layer-by-layer cross-entropy training

1. Train a network with one hidden layer
2. Remove the output layer and weights leading to the output layer
3. Add an additional hidden layer and train only the newly added weights
4. Goto 2 or finetune & stop if deep enough



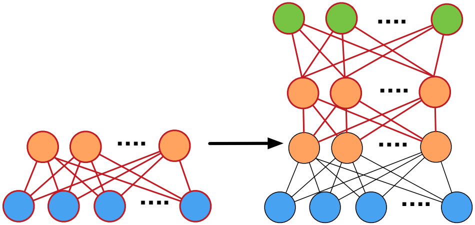
Greedy Layer-by-layer cross-entropy training

1. Train a network with one hidden layer
2. Remove the output layer and weights leading to the output layer
3. Add an additional hidden layer and train only the newly added weights
4. Goto 2 or finetune & stop if deep enough



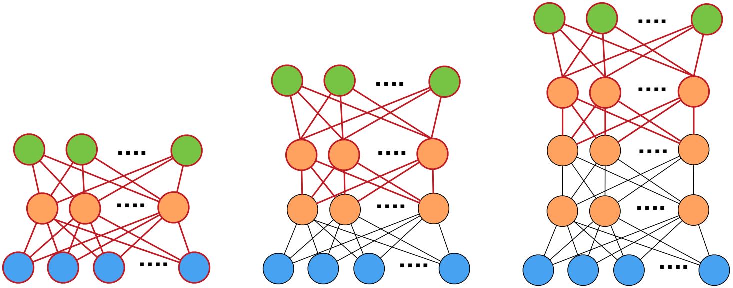
Greedy Layer-by-layer cross-entropy training

1. Train a network with one hidden layer
2. Remove the output layer and weights leading to the output layer
3. Add an additional hidden layer and train only the newly added weights
4. Goto 2 or finetune & stop if deep enough



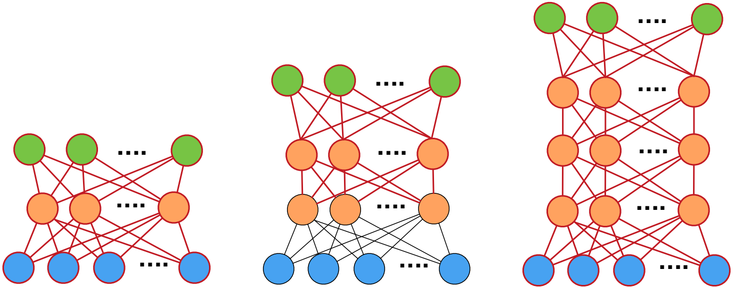
Greedy Layer-by-layer cross-entropy training

1. Train a network with one hidden layer
2. Remove the output layer and weights leading to the output layer
3. Add an additional hidden layer and train only the newly added weights
4. Goto 2 or finetune & stop if deep enough



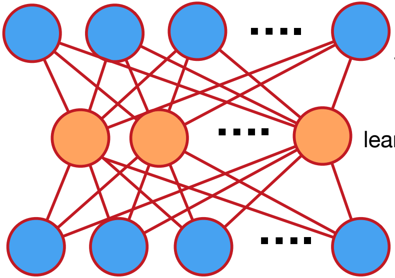
Greedy Layer-by-layer cross-entropy training

1. Train a network with one hidden layer
2. Remove the output layer and weights leading to the output layer
3. Add an additional hidden layer and train only the newly added weights
4. Goto 2 or finetune & stop if deep enough



Autoencoders

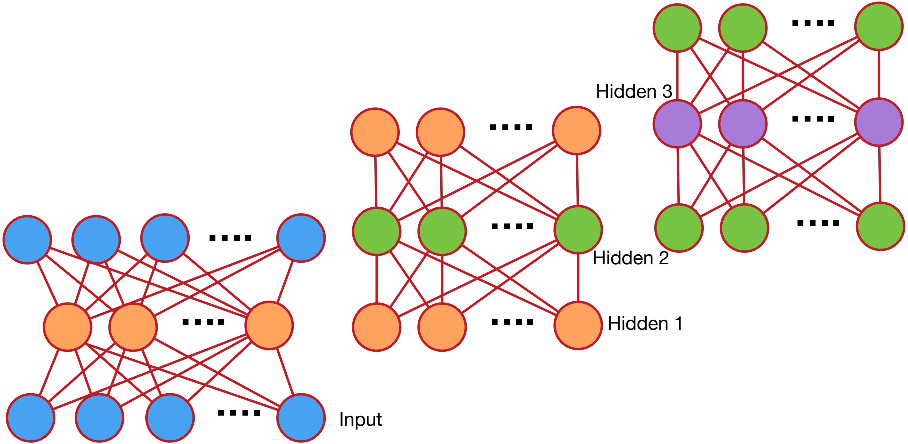
- An autoencoder is a neural network trained to map its input into a distributed representation from which the input can be reconstructed
- Example: single hidden layer network, with an output the same dimension as the input, trained to reproduce the input using squared error cost function



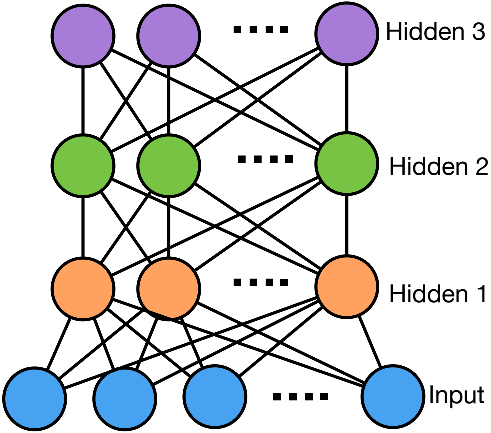
Stacked autoencoders

- Can the hidden layer just copy the input (if it has an equal or higher dimension)?
  - In practice experiments show that nonlinear autoencoders trained with stochastic gradient descent result in useful hidden representations
  - Early stopping acts as a regulariser
- **Stacked autoencoders** – train a sequence of autoencoders, layer-by-layer
  - First train a single hidden layer autoencoder
  - Then use the learned hidden layer as the input to a new autoencoder

Stacked Autoencoders

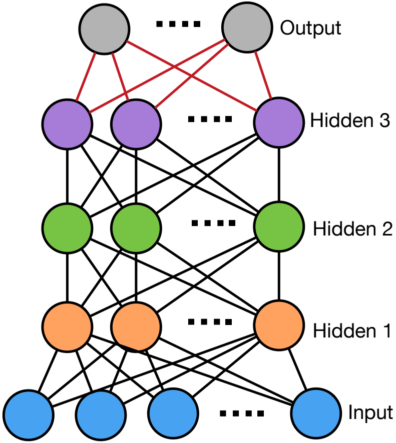


Pretraining using Stacked autoencoder



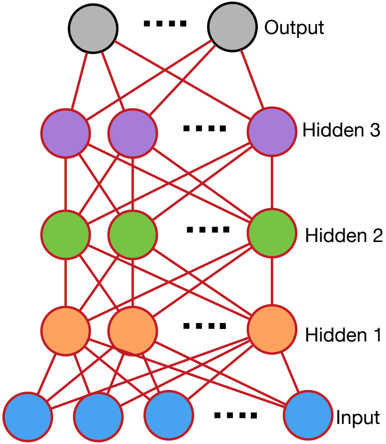
Initialise hidden layers

Pretraining using Stacked autoencoder



Train output layer

Pretraining using Stacked autoencoder

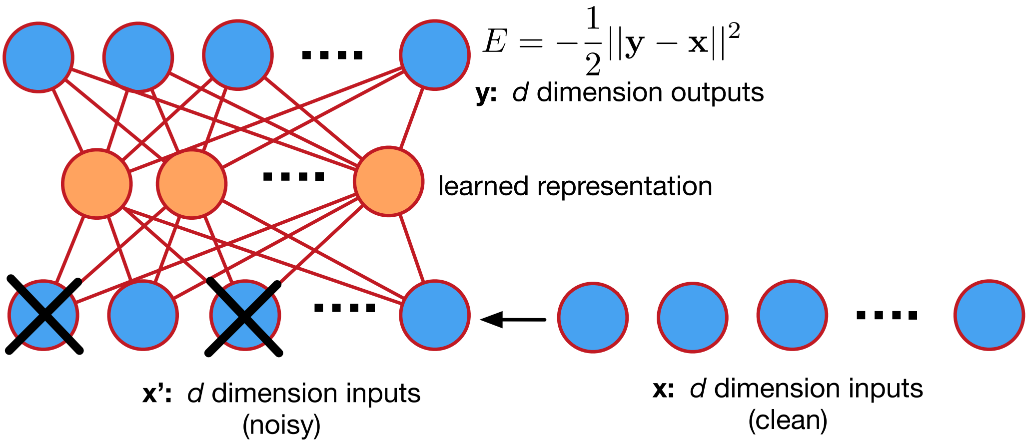


Fine tune whole network

Denoising Autoencoders

- Basic idea: Map from a corrupted version of the input to a clean version (at the output)
- Forces the learned representation to be stable and robust to noise and variations in the input
- To perform the denoising task well requires a representation which models the important structure in the input
- The aim is to learn a representation that is robust to noise, not to perform the denoising mapping as well as possible
- Noise in the input:
  - Random **Gaussian** noise added to each input vector
  - **Masking** – randomly setting some components of the input vector to 0
  - **"Salt & Pepper"** – randomly setting some components of the input vector to 0 and others to 1
- Stacked denoising autoencoders – noise is only applied to the input vectors, not to the learned representations

Denoising Autoencoder



Summary

- Layer-by-layer Pretraining and Autoencoders
  - For many tasks (e.g. MNIST) pre-training seems to be necessary / useful for training deep networks
  - For some tasks with very large sets of training data (e.g. speech recognition) pre-training may not be necessary
  - (Can also pre-train using stacked restricted Boltzmann machines)
- Reading: Michael Nielsen, chapter 5 of *Neural Networks and Deep Learning*
  http://neuralnetworksanddeeplearning.com/chap5.html
  Pascal Vincent et al, "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion", JMLR, 11:3371–3408, 2010.
  http://www.jmlr.org/papers/volume11/vincent10a/vincent10a.pdf