Neural Networks: Regularization and Encodings

Mark Craven and David Page
Computer Sciences 760
Spring 2019

Goals for the lecture

you should understand the following concepts

- regularization
- early stopping
- dropout
- data augmentation
- simple input encodings (one-hot, thermometer)
- output encodings
- most appropriate loss functions for each type of output encoding
- autoencoders
- word embeddings
- skip-gram models

Regularization

- *Regularization* refers to an approach for avoiding overfitting by biasing the learning process away from completely fitting the training data

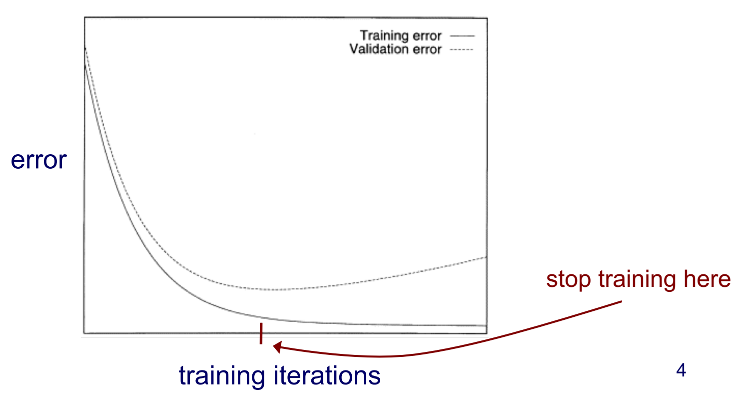
- E.g. decision-tree pruning is a regularization method

- some regularization methods for neural networks
  - early stopping
  - dropout
  - data augmentation
  - L2 penalty term (we'll discuss this later in the context of linear regression)

Early stopping

- conventional gradient descent: train until local minimum reached

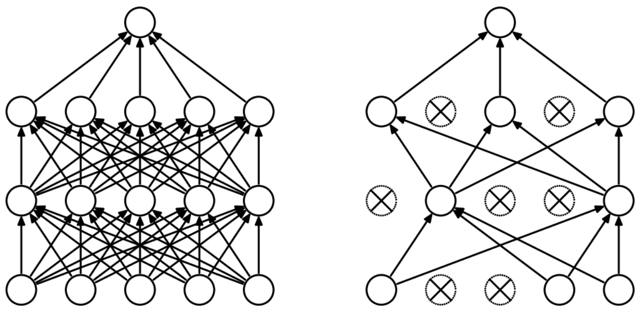
- empirically sometimes better: *early stopping*
  - use a validation set to monitor accuracy during training iterations
  - return the weights that result in minimum validation-set error



4

Dropout

On each training iteration
- randomly "drop out" a subset of the units and their weights
- ignore these units; do forward and backprop on remaining network

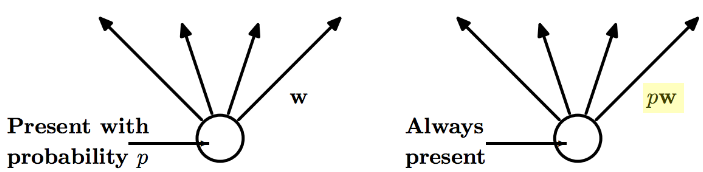


(a) Standard Neural Net (b) After applying dropout.

Figures from Srivastava et al., *Journal of Machine Learning Research* 2014

Dropout

At test time
- final model uses all units and weights in the network
- adjust each weight by multiplying it by the fraction of iterations it was used during training



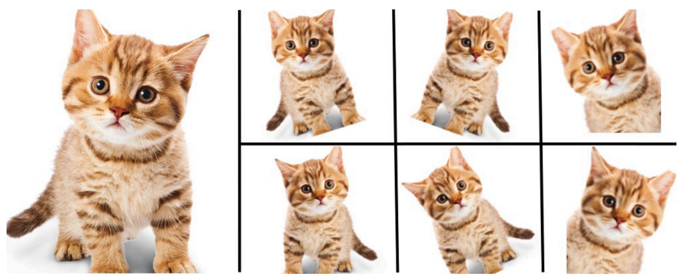
(a) At training time (b) At test time

Figures from Srivastava et al., *Journal of Machine Learning Research* 2014

Note: this approach is very similar to ensembles, which we'll discuss in a few weeks

Data Augmentation

- entails turning each training instance into many training instances
- especially applicable in image analysis tasks; can augment an image by rotation, re-scaling, shifting, flipping about axis, adding noise
- image still contains the same objects, exhibits the same event or action

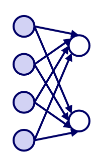


7

Input (feature) encoding for neural networks

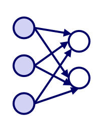
nominal features are usually represented using a *1-hot* encoding

$$A = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad C = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad G = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad T = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$



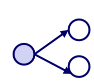
ordinal features can be represented using a *thermometer* encoding

$$\text{small} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad \text{medium} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \quad \text{large} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$



real-valued features can be represented using individual input units (we may want to scale/normalize them first though)

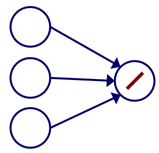
$$\text{precipitation} = \begin{bmatrix} 0.68 \end{bmatrix}$$



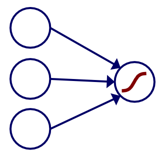
8

Output encoding for neural networks

regression tasks usually use output units with linear transfer functions

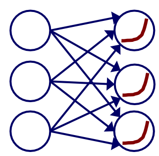


binary classification tasks usually use one sigmoid output unit



k-ary classification tasks usually use *k* sigmoid or *softmax* output units

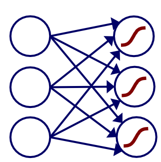
$$o_i = \frac{e^{net_i}}{\sum_{j \in outputs} e^{net_j}}$$



9

Output encoding for neural networks (continued)

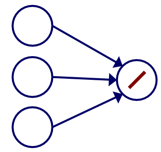
Multi-label classification tasks (i.e. multiple binary labels are predicted for each instance) usually use multiple sigmoid output units



Most appropriate objective function for each output encoding

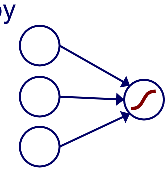
regression: squared error

$$E(\mathbf{w}) = \frac{1}{2}\sum_{d \in D}\left(y^{(d)} - o^{(d)}\right)^2$$



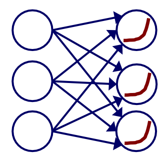
binary classification, multi-label classification: cross entropy

$$E(\mathbf{w}) = \sum_{d \in D} -y^{(d)}\ln\left(o^{(d)}\right) - \left(1 - y^{(d)}\right)\ln\left(1 - o^{(d)}\right)$$



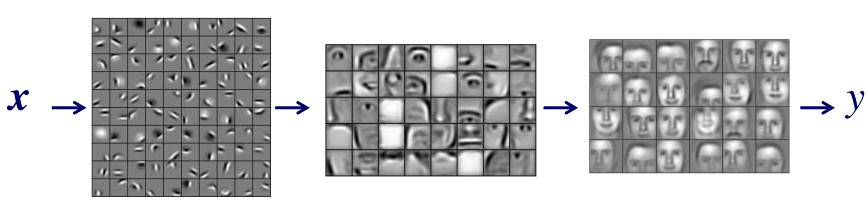
k-ary classification: multiclass cross entropy

$$E(\mathbf{w}) = -\sum_{d \in D}\sum_{i=1}^{\#\,classes} y_i^{(d)}\, ln\left(o_i^{(d)}\right)$$



Learning representations

- the feature representation provided is often the most significant factor in how well a learning system works
- an appealing aspect of multilayer neural networks is that they are able to change the feature representation
- can think of the nodes in the hidden layer as new features constructed from the original features in the input layer



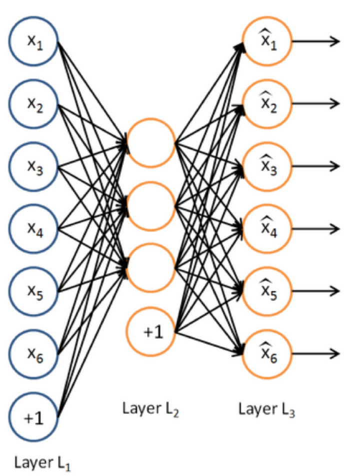
[Figures from Lee et al., *ICML* 2009]

Learning encodings

Can we use multi-layer networks to learn good encodings that can be used for multiple tasks?

Autoencoders

- one approach: use *autoencoders* to learn hidden-unit representations
- in an autoencoder, the network is trained to reconstruct the inputs
- the hidden units can then be used as a new encoding for inputs



14

Autoencoder variants

- how to encourage the autoencoder to generalize
  - *bottleneck*: use fewer hidden units than inputs
  - *sparsity*: use a penalty function that encourages most hidden unit activations to be near 0 [Goodfellow et al. 2009]
  - *denoising*: train to predict true input from corrupted input [Vincent et al. 2008]
  - *contractive*: force encoder to have small derivatives (of hidden unit output as input varies) [Rifai et al. 2011]

15

Word embeddings

- standard representation for words is a one-hot (*1-of-k*) encoding

$$\begin{array}{r} \text{aardvark} \\ \text{anteater} \\ \text{apple} \\ \\ \text{cat} \\ \\ \text{dog} \\ \\ \text{zymergy} \end{array} \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{bmatrix} \qquad \begin{array}{r} \text{aardvark} \\ \text{anteater} \\ \text{apple} \\ \\ \text{cat} \\ \\ \text{dog} \\ \\ \text{zymergy} \end{array} \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{bmatrix} \qquad \begin{array}{r} \text{aardvark} \\ \text{anteater} \\ \text{apple} \\ \\ \text{cat} \\ \\ \text{dog} \\ \\ \text{zymergy} \end{array} \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

- this encoding doesn't capture any information about the semantic similarity among words which may be useful for many NLP tasks

$$\boldsymbol{w}_{cat} \cdot \boldsymbol{w}_{dog} = 0$$

Word embeddings

- How can we find dense, lower-dimensional vectors that capture semantic similarity among words?

$$\begin{array}{cc} \text{cat} & \text{dog} \\ \begin{bmatrix} .26 \\ .01 \\ .37 \\ \vdots \\ .07 \end{bmatrix} & \begin{bmatrix} .24 \\ .04 \\ .38 \\ \vdots \\ .13 \end{bmatrix} \end{array}$$

- The key is to learn to model the context around words.
  Firth 1957: "*You shall know a word by the company it keeps*."

"…what if your dog has cold paws…"

The skip-gram model

- learn to predict context words in a window around given word
  - collect a large corpus of text
  - define a window size s
  - construct training instances that map from word → other word in window

"…what if your dog has cold paws…"

dog → *what*

dog → *if*

dog → *your*

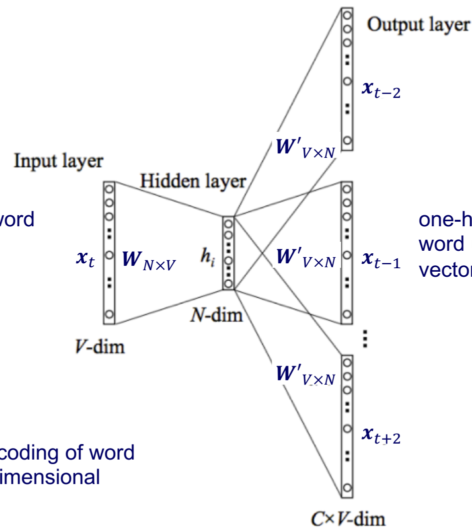
dog → *has*

dog → *cold*

dog → *paws*

The skip-gram model

one-hot encoding of word using V-dimensional vector

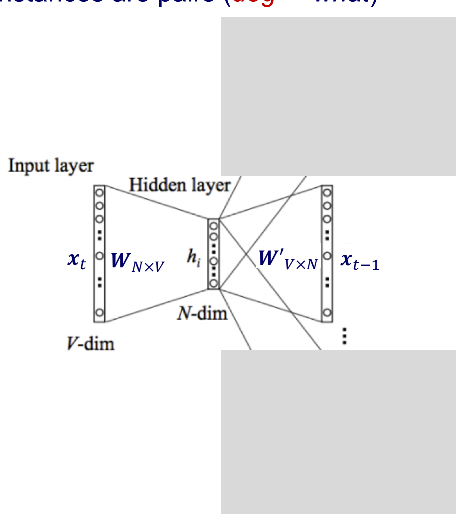


one-hot encoding of context word using V-dimensional vector

dense encoding of word using N-dimensional vector

The skip-gram model

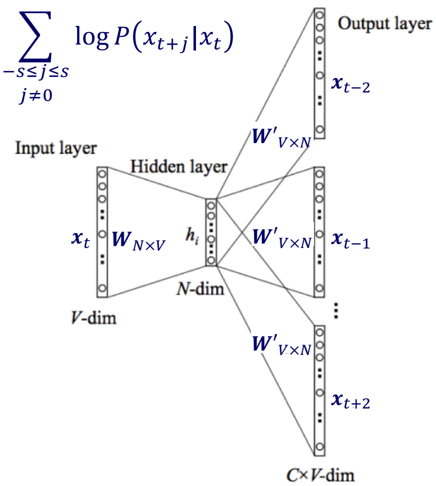
- same weights used for all context positions, so effectively
  - network has one set of ouptut units
  - training instances are pairs (*dog* → *what*)



The skip-gram model

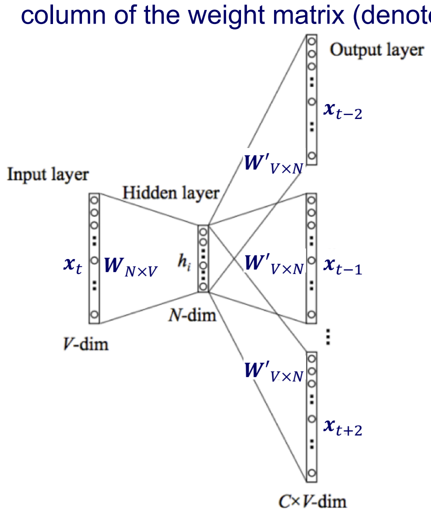
- objective function: learn weights that minimize negative log probability (maximize probability) of words in context for each given word

$$E(\boldsymbol{w}) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-s\le j\le s\\ j\ne 0}} \log P\left(x_{t+j}|x_t\right)$$



The skip-gram model

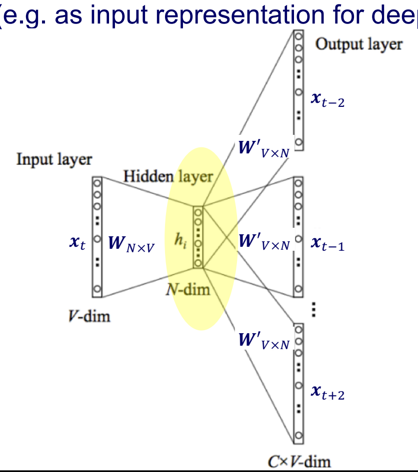
- output units use softmax function to compute $P(x_O|x_I)$
- hidden units are linear
- since input vector uses a one-hot encoding, it effectively selects a column of the weight matrix (denoted by w_{x_I})



$$P(x_O|x_I) = \frac{\exp\left(w'_{x_O} w_{x_I}\right)}{\sum_x \exp\left(w'_x w_{x_I}\right)}$$

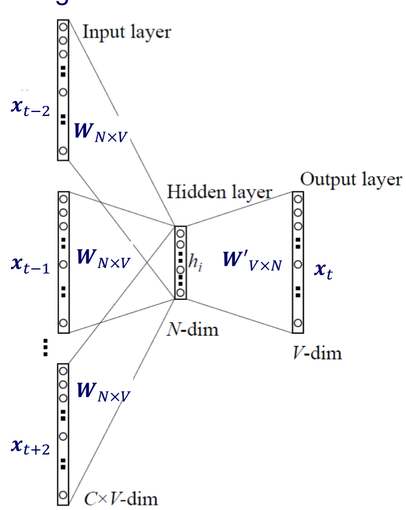
Using word embeddings

- after training, it's the hidden layer (equivalently, the first layer of weights) we care about
- the hidden units provide a dense, low-dimensional representation for each word
- we can use these low-dimensional vectors in any NLP application (e.g. as input representation for deep network)



An alternative: the Continuous Bag of Words (CBOW) model

- learn to predict a word given its context



Pretrained embeddings

- Word2vec
  - skip-gram model developed by Google [Mikolov et al. 2013]
  - *Original version:* V = 692k, N = 300
  - Current versions: V = 1.4 – 3 million, N = 300 – 1000
  - trained on corpora with billions of words

- GloVe [Pennington et al. 2014]
  - word embeddings

- Med2Vec [Choi et al. 2006]
  - embeddings of medical concepts