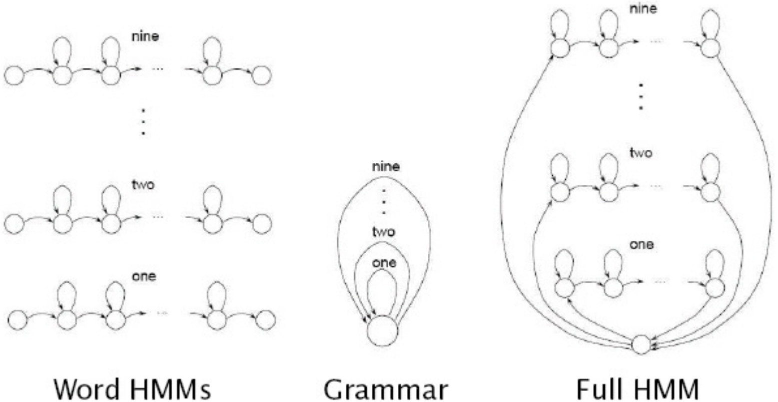
TTIC 31110
Speech Technologies

April 30, 2020

Questions from last time: Continuous speech recognition with HMMs

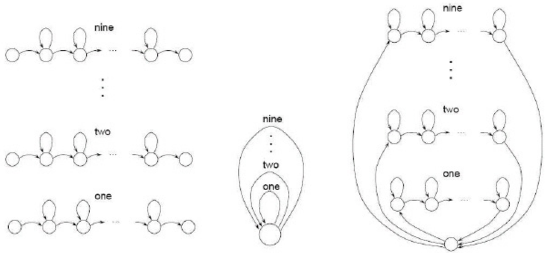
Most basic model: String together word HMMs to make sentence HMM, using a grammar (Note: Start and end states of these word HMMs are non-emitting)



Questions from last time: Continuous speech recognition with HMMs

What is this grammar?

- A representation of all allowed sequences of words in the language (and optionally their probabilities)
- Allowed sequences are those that can be produced by traversing edges from the start state to the end state (in this case, these are the same single state)
- In this case, all transitions (edges) are equally probable



Questions from last time: How does state tying affect EM training?

Recall: The Baum-Welch algorithm

- E step: Compute the *expected counts*

$$\sum_{t=1}^{T} \gamma_t(i) \quad = \quad \text{expected count of state } i$$

$$\sum_{t=1}^{T-1} \gamma_t(i) \quad = \quad \text{expected count of transitions from state } i$$

$$\sum_{t=1}^{T-1} \xi_t(i,j) \quad = \quad \text{expected count of transitions from } i \text{ to } j$$

Questions from last time: How does state tying affect EM training?

Recall: The Baum-Welch algorithm M step

$$\hat{a}_{ij} = \frac{\text{expected-count}(i \rightarrow j)}{\text{expected-count}(\text{transitions from } i)} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

$$\hat{b}_i(k) = \frac{\text{expected-count}(v_k \text{ in state } i)}{\text{expected-count}(i)} = \frac{\sum_{t=1, o_t=v_k}^{T} \gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)}$$

$$\hat{\pi}_i = \text{expected-count}(q_1 = i) = \gamma_1(i)$$

Questions from last time: How does state tying affect EM training?

E step (computation of expected counts)

- Counts are computed together for all tied states, e.g. if states i and j are tied:

$$\frac{1}{2}\left(\sum_{t=1}^{T}\gamma_t(i)+\sum_{t=1}^{T}\gamma_t(j)\right) = \text{ expected count of state } i \text{ or } j$$

M step

- Unaffected!

Questions from last time: How do we represent the state transition matrix for very large models?

- The state transition matrix is typically very sparse
- Usually, only transitions allowed are to self with probability p or next with probability $1-p$
- So, we need only store each state's self-transition probability

Questions from last time: Where does supervised training come in?

HMM training with Baum-Welch is unsupervised, so how do we used labeled training sets?

Training an ASR system is more than just training a single HMM

- Typically, we have one HMM per word or phone label
- Supervised training: Spoken utterances with corresponding label sequences
- Start/end times of each label may or may not be given
- If start/end times are given, then the problem becomes one of training each of the word/phone HMMs on the collection of segments corresponding to that label
- If start/end times are not given:
  - Estimate the start and end times (do *forced alignment* – more on this later)
  - Consider these to be additional latent variables in the EM algorithm (rarely done)

Recap: Continuous ASR with the Viterbi algorithm

Recognition = finding the most probable word string $\mathbf{w}^*$ given the acoustic observations $\mathbf{O}$: $\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmax}}\, p(\mathbf{w}|\mathbf{O})$

From Bayes' rule, $p(\mathbf{w}|\mathbf{O}) = \frac{p(\mathbf{O}|\mathbf{w})p(\mathbf{w})}{p(\mathbf{O})}$. Therefore,

$$\begin{aligned} \mathbf{w}^* &= \underset{\mathbf{w}}{\operatorname{argmax}}\, p(\mathbf{w}|\mathbf{O}) \\ &= \underset{\mathbf{w}}{\operatorname{argmax}}\, p(\mathbf{O}|\mathbf{w})p(\mathbf{w}) \end{aligned}$$

- The "whole-sentence" HMM gives $p(\mathbf{O}|\mathbf{w})$, the acoustic model
- $p(\mathbf{w})$ is the language model

Recap: Continuous ASR with the Viterbi algorithm

Summing over all possible state sequences $\mathbf{q}$ for the word string $\mathbf{w}$:

$$\begin{aligned}\mathbf{w}^* &= \operatorname*{argmax}_{\mathbf{w}} p(\mathbf{O}|\mathbf{w})p(\mathbf{w}) \\ &= \operatorname*{argmax}_{\mathbf{w}} \sum_{\mathbf{q}} p(\mathbf{O}|\mathbf{q},\mathbf{w})p(\mathbf{q}|\mathbf{w})p(\mathbf{w})\end{aligned}$$

- $p(\mathbf{O}|\mathbf{q},\mathbf{w})$ is given by the observation (emission) distribution
- $p(\mathbf{q}|\mathbf{w})$ is given by the state transition probabilities
- *Viterbi approximation*: Assume there is a single most probable state sequence $\mathbf{q}^*$ such that all other $\mathbf{q}$ contribute a negligible amount to the sum. Then $\mathbf{w}^* = \operatorname*{argmax}_{\mathbf{w}} p(\mathbf{O}|\mathbf{q}^*,\mathbf{w})p(\mathbf{q}^*|\mathbf{w})p(\mathbf{w})$
- So we can maximize jointly over $\mathbf{w}$ and $\mathbf{q}$: $\mathbf{w}^*, \mathbf{q}^* = \operatorname*{argmax}_{\mathbf{w},\mathbf{q}} p(\mathbf{O}|\mathbf{q},\mathbf{w})p(\mathbf{q}|\mathbf{w})p(\mathbf{w})$

Recap: Continuous ASR with the Viterbi algorithm

- To do continuous speech recognition, we can string together word HMMs according to a *grammar* or *language model*
- (Will get back to language modeling later)
- We will find the best state sequence using the Viterbi algorithm, and output the corresponding word string

Questions from last time: Isn't it unnatural to chop up speech into distinct segments?

- As we've discussed, speech doesn't have sharp divisions between one word and the next, or one phone and the next
- Transitions from one sound/word to the next are often gradual
- When we run Viterbi, we make a decision about when each state/HMM start and end
- This is not great! Some solutions:
  - Some older work modified Viterbi to sum over multiple possible paths with slightly different start/end times – much slower, and not too much gained in performance (so the "distinct segment assumption" wasn't hurting HMMs too badly)
  - Newer neural models allow for fuzzier (or no) decisions about start and end times, e.g. neural encoder-decoders and connectionist temporal classification (CTC)
  - We'll discuss the latter in the coming weeks

The need for subword units

Whole-word HMMs have some problems

- Cannot model unseen words, or even words seen too few times in training data
- Number of parameters is proportional to the vocabulary size
- (Note: The more parameters, the more data needed; rule of thumb: $\sim$10 data points per parameter)

$\Rightarrow$ Whole-word models mainly restricted to small-vocabulary tasks

Subword units

- In the same way that sentences can be composed of whole-word HMMs, words can be composed of sub-word HMMs
- Units are then shared among words

Type of units	Approximate # (in English)
words	>100,000
phones	50
diphones	2,000
triphones	10,000
syllables	5,000

- Number of parameters now proportional to the number of sub-word units, not number of words

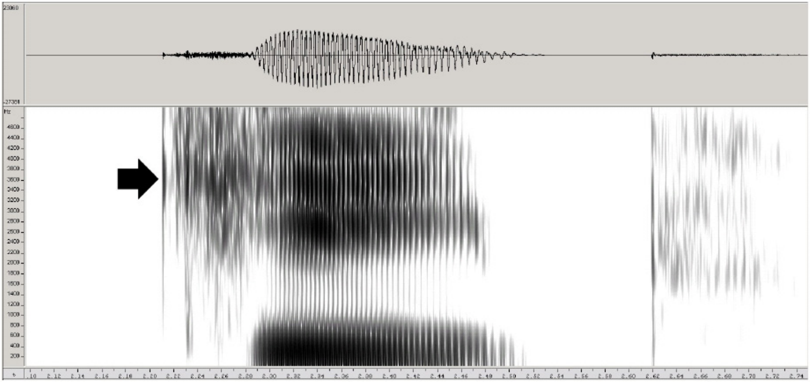
Baseforms

- Each word can be represented as a sequence of phonemes, the word's *baseform*
  `dogs` $\longrightarrow$ d ao g z
- Some words may need more than one baseform
  `the` $\longrightarrow$ dh ah, dh iy
  `either` $\longrightarrow$ iy dh er, ay dh er
- Each word's baseform(s) can be looked up in a dictionary
- All words in training and test data must be in the dictionary, or else we get them wrong
- Typically, baseform dictionaries are written by hand $\longrightarrow$ prone to errors, inconsistencies, disagreements among linguists, ...

The need for context-dependent units

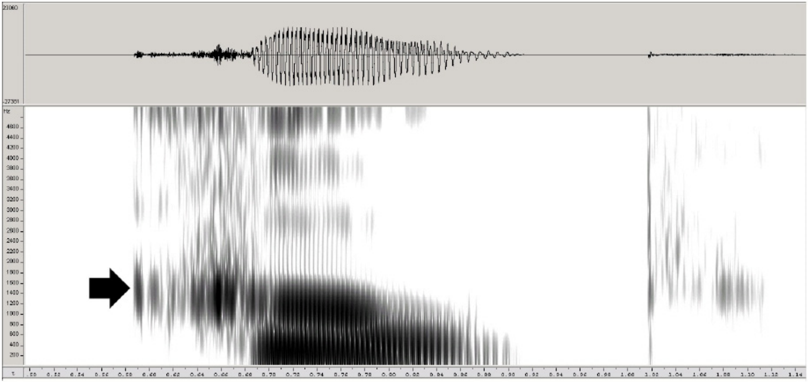
- Recall: phonemes have variants (*allophones*)
  - Aspirated and un-aspirated stops: *pin* vs. *spin*
  - Allophones of /t/: *too* vs. *stew* vs. *butter* vs. *but* (the last with "unreleased" /t/)
- Phonemes are also influenced by surrounding context due to *co-articulation*, e.g. due to articulatory inertia
  - *keep*, *geese* (front /k/, /g/) vs. *coop*, *goose* (back /k/, /g/)

The need for context-dependent units: Example



“keep”

The need for context-dependent units: Example (2)



"coop"

Triphones: Context-dependent phone models

- Model each phoneme in the context of its left and right neighbor: `k-iy+p` triphone is /iy/ in the context of a preceding /k/ and following /p/ (not a model of the *sequence* /k iy p/)
- Build separate HMM for each triphone
- ~ 50 phonemes $\Rightarrow$ 50^3 triphones!
- Not all triphones are possible (e.g., `k-l+p` is not)
- Still, $\sim 10,000$ are possible and some are very rare
- Solution: tying!

A comparison of model sizes

- Consider continuous-density HMMs where state observation distribution is Gaussian mixture model (GMM)
- Assume 10 diagonal Gaussians per state and 39-dimensional MFCC vectors
- Context-independent (CI) phone models
  - ~ 50 phonemes with ~ 3 states each $\Rightarrow \sim$ 150 GMMs
  - 10 Gaussians per GMM $\Rightarrow \sim 1500$ Gaussians
  - 39 dimensions per Gaussian, each requiring a mean and variance $\Rightarrow \sim 120,000$ parameters
  - Manageable with ~ 3 hours of speech training data
- Context-dependent (CD) triphone models
  - $\sim 10,000$ triphones with ~ 3 states each $\Rightarrow \sim 30,000$ GMMs $\Rightarrow \sim 300,000$ Gaussians $\Rightarrow \sim 24,000,000$ parameters $\Rightarrow \sim 600$ hours of speech?
  - Actually... since some of the triphones are extremely rare, this may still not be enough!

Tying via agglomerative (bottom-up) clustering

- Cluster triphones (or individual triphone states) into "sufficiently similar" groups
- Start with each triphone/state in cluster by itself, then iterate:
  - Find "closest" pair of clusters
  - Merge them into single cluster
- Until some convergence criterion
  - Distance between closest clusters is above some threshold
  - Number of data points in each cluster is above some threshold
- Different clusterings based on distance measure:
  - a and b are individual triphones, A and B are clusters
  - $\mathcal{D}(a, b)$ is some distance function between triphones
  - *Single-linkage*: $\mathcal{D}(A, B) = \min_{a \in A, b \in B} \mathcal{D}(a, b)$
  - *Complete-linkage*: $\mathcal{D}(A, B) = \max_{a \in A, b \in B} \mathcal{D}(a, b)$

Bottom-up vs. top-down clustering

- Agglomerative clustering is limited to the case where we have *some* examples of each item
- In the case of triphones, many may be unseen!
- Agglomerative clustering therefore tends not to be used for tying triphones
- But still used for other tasks, e.g. automatic discovery of speech units
- Alternative: top-down clustering, via decision trees

Decision tree triphone clustering

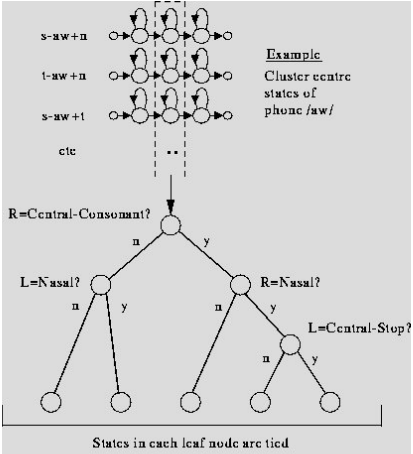


Fig. 9.3 Decision tree-based state tying

Decision tree construction

- Start with all frames for a given phone in a single node (the root)
- Find the best "question" for partitioning the data at a given node into two classes
- Repeat for each node
- Stop when there is insufficient data at each node, or when the best question isn't helpful enough

Details: What questions shall we ask?

- Typically involve identities of previous/following phones
- Or other attributes (e.g. *phonetic features*) of previous/following phones: voicing, nasality, place of articulation, manner of articulation, etc.
- In principle, can ask about any subset of the features that are relevant at the current node
  - Is the following phone in $\{\mathrm{aa}, \mathrm{k}, \mathrm{l}\}$ and the previous phone voiced, fricated, or nasal?
  - That's a lot of questions! $(\sum_j \binom{\#\text{features}}{j})$
- In practice, use smaller set of pre-determined questions
  - Create in advance a list of "reasonable" questions
  - In principle, questions (and therefore tree splits) can be n-ary; in practice, binary

When to stop?

- Cross-validation: Measure likelihood with different tree sizes on a held-out data set, choose the tree that maximizes likelihood on held-out data
- In practice, simple heuristics are often used:
  - Data at node has fewer than threshold T samples
  - Best question does not improve likelihood significantly (note: best question should *always* improve likelihood somewhat!)

Where does the data come from?

- In general, we don't have phonetic labels for each frame of training data
- One solution: Automatically align the data given some "simple" initial model, e.g. monophone-based recognizer
- Optionally, iterate: Once we have a better triphone model, re-align training data and re-build DT

Other issues in sub-word modeling

Pronunciation modeling

- Pronunciations often don't match the dictionary (dialects, non-native accents, conversational speech)
- New words (e.g. names) – how to get their baseforms?
- One idea: Apply letter-to-sound DTs
- More modern idea: Learn a neural "grapheme-to-phoneme" (G2P) model

Language-specific issues beyond English

- Some languages have more predictable mapping from orthography to sounds: Spanish, French, German, Korean, ... $\Rightarrow$ pronouncing dictionaries virtually unnecessary!
- Some languages have *less* regular mapping from orthography to sounds, e.g. Mandarin Chinese
- ... or even an ambiguous mapping, e.g. Arabic, Hebrew

Hybrid generative/discriminative models

- Generative models are models of the data generation process $p(\mathbf{O}, \mathbf{w}) = p(\mathbf{O}|\mathbf{w})p(\mathbf{w})$
- Recognition is "inversion" of generation, $\mathbf{w}^* = \text{argmax}_{\mathbf{w}}\, p(\mathbf{w}|\mathbf{O}) = \text{argmax}_{\mathbf{w}}\, p(\mathbf{O}|\mathbf{w})p(\mathbf{w})$
- Discriminative approaches attempt to solve the task more directly, e.g. by modeling $p(\mathbf{w}|\mathbf{O})$ directly or even minimizing the intended error rate directly
- Main motivation for hybrid models:
  - Discriminative models are good! Let's use them!
  - Discriminative models for *sequences* are hard! (More later)
  - Let's combine discriminative **frame classifiers** with generative **sequence models**

Hybrid generative/discriminative models

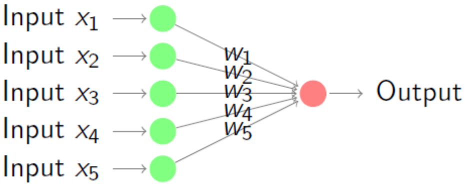
Typical approach:

1. Train a frame-based discriminative classifier of sub-word units (e.g. phones, phone states, triphone states) given some labeled training data, $c^* = f_c(\mathbf{o})$, where c is the class and $\mathbf{o}$ is a frame feature vector
2. The output is a posterior probability $p(c|\mathbf{o})$
3. Convert $p(c|\mathbf{o})$ to something like an observation model (a "likelihood"): $p(\mathbf{o}|c) \propto \frac{p(c|\mathbf{o})}{p(c)}$
4. Use the result in place of the observation model in an HMM

Hybrid generative/discriminative models

- Most popular type of frame classifier by far: neural network
- Long line of research on hybrid HMM/NN models by groups at ICSI Berkeley, IDIAP, and elsewhere (e.g., Bourlard and Morgan 1994)
- Motivations for using NNs:
  - For right choice of NNs and training criterion, the output gives an estimate of the class posterior probabilities $p(c|\mathbf{o})$
  - NNs are "universal function approximators": Whatever the optimal classifier function $f_c(\mathbf{o})$ is, there is some neural network that approximates it arbitrarily well (under mild assumptions)
- Main other (distant) competitor for frame classification: Support vector machines

Linear classifiers



Let's start with a binary linear classifier (change of notation $\mathbf{o} \longrightarrow \mathbf{x}$):

$$
\begin{aligned}
f(\mathbf{x}) &= 1, \quad \mathbf{w} \cdot \mathbf{x} > 0 \\
&= 0, \quad \text{otherwise}
\end{aligned}
$$

- Can add an extra dimension to the input $\mathbf{x}$ which is always 1, to induce a "bias"

Linear classifiers: Learning weights with the perceptron algorithm [Rosenblatt 1957]

1. Initialize the weights $\mathbf{w}$
2. At each iteration t, retrieve an example input $\mathbf{x}_j$ in a training set, with corresponding output y_j:
   - Compute the output $f(\mathbf{x}_j)$ using the current weights
   - Update the weights: for all nodes $0 \leq i \leq n$,

$$w_i(t+1) = w_i(t) + \alpha(y_j - f(\mathbf{x}_j))x_{j,i}$$

3. Repeat step 2 until the average error $\frac{1}{t}\sum_{j=1}^{t} |\hat{y}_j - f(\mathbf{x}_j)|$ is less than some threshold, or until some maximum iteration number t

This does gradient descent on the perceptron loss $\sum_j \max(0, -\mathbf{y}_j \mathbf{w} \cdot \mathbf{x}_j)$

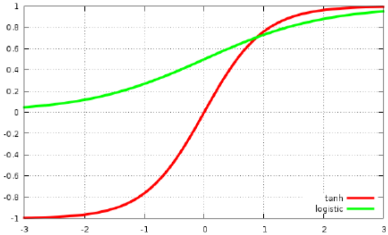
Nonlinear classifiers

What if the classification boundary between classes is not linear?
Add nonlinearity:

$$\begin{aligned} f(\mathbf{x}) &= 1, \quad \sigma(\mathbf{w}\cdot\mathbf{x}) > \tau \\ &= 0, \quad \text{otherwise} \end{aligned}$$

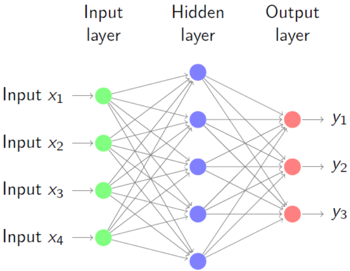
Examples:

- Heaviside (step, threshold): $\sigma(z) = 0, z < 0;\ 1, \text{otherwise}$
- Logistic sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$
- Hyperbolic tangent: $\sigma(z) = \tanh(z)$



Multilayer perceptrons (MLP)

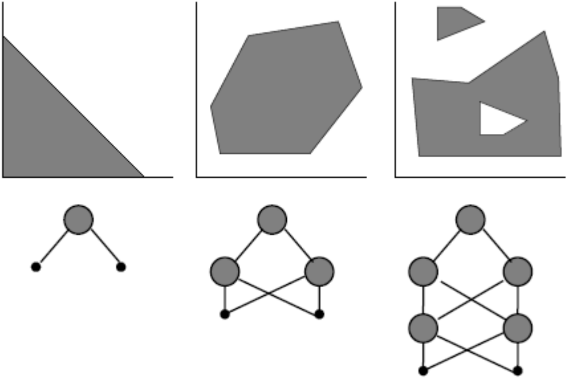
What if the nonlinear perceptron is still not enough to represent the decision boundary? Add more layers:



- Each node i in each layer l now outputs $y_i^l = \sigma(\mathbf{w}_i^l \cdot \mathbf{y}_i^{l-1} + b_i^l)$ (with bias b now explicit)
- Or writing each layer's output as a vector: $\mathbf{y}^l = \sigma_l(\mathbf{W}_l \mathbf{y}^{l-1} + \mathbf{b}^l)$, where σ is applied element-wise
- Final output: $\mathbf{y} = \mathbf{y}^L$ for an L-layer network

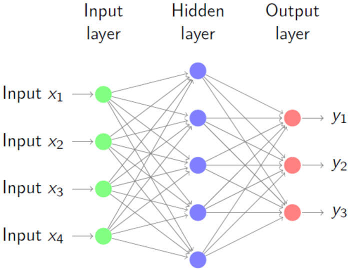
Example decision boundaries

Here the activation functions are all threshold functions:



Multi-class outputs

We've also expanded from a single output node to multiple nodes, to allow for more than a single class



Typical activation function: Softmax $y_i = \frac{\exp(z_i)}{\sum_{j=1}^{n} \exp(z_j)}$
where $z_i = \mathbf{w}_i \cdot \mathbf{x} + b_i$ (Note: layer indexing dropped; $\mathbf{x}$ refers to the input of the current layer)

Feedforward neural networks

More generally, a feedforward neural network (NN, or DNN) is any vector function $f(\mathbf{x})$ of a vector input $\mathbf{x}$ that can be written as a composition of the layers we've defined:

$$\begin{aligned} f(\mathbf{x}) &= \mathbf{y}^L \\ \mathbf{y}^l &= \sigma_l(\mathbf{W}_l \mathbf{y}^{l-1} + \mathbf{b}^l) \\ \mathbf{y}^0 &= \mathbf{x} \end{aligned}$$

Training neural networks

The parameters are learned to minimize some loss, or measure of badness of the outputs

- A NN is an MLP if it is trained with perceptron loss (though often MLP is used to refer to any feedforward NN)
- Other losses (per training example):
  - Squared loss: $\mathcal{L}_{SE} = \sum_i (y_i - \hat{y}_i)^2$
  - Cross-entropy loss (log loss): $\mathcal{L}_{CE} = \sum_i \hat{y}_i \log y_i$ (typical for multi-class classification)
- Total loss is the sum of the loss over all training examples

Training neural networks (2)

To minimize loss $\mathcal{L}$, we typically use gradient descent:

$$w(t+1) = w(t) - \eta \frac{\partial \mathcal{L}}{\partial w}$$

- w is any single weight
- η is a user-defined learning rate (set, e.g., by tuning on held-out data or via some rule of thumb)
- $\mathcal{L}$ is computed for 1 training example, a training subset (minibatch), or the full training set (a batch)
- If a single example or a minibatch is used, then this is stochastic gradient descent
- Gradient can be computed via the chain rule; this is called backpropagation
- Fortunately, for typical losses, the gradients are similar and simple for all weights
- ... and we often don't have to compute them as there are excellent toolkits to do it for us

When is it enough complexity?

- In theory*, one hidden layer is sufficient to approximate any output function arbitrarily well
- In practice**, multiple hidden layers can be *very* helpful

- * In theory, theory is the same as practice; in practice, it is almost never the case
- ** "In practice" $=$ it may be much easier to learn parameters with multiple hidden layers***
- *** There is some theory to this practice too

More activation functions

Examples:

- ReLU: $\sigma(z) = \max(0, z)$
- Softsign: $\sigma(z) = \frac{z}{1+|z|}$
- Softplus: $\sigma(z) = \log(1 + e^z)$