Neural Networks

Hugo Larochelle (@hugo_larochelle)
Google Brain
Topics: online videos

- for a more detailed description of neural networks...
- ... and much more!

http://info.usherbrooke.ca/hlarochelle/neural_networks
Topics: online videos

- for a more detailed description of neural networks...

- ... and much more!

http://info.usherbrooke.ca/hlarochelle/neural_networks

**RESTRICTED BOLTZMANN MACHINE**

**Topics:** RBM, visible layer, hidden layer, energy function

\[
\begin{align*}
\text{Energy function: } E(x, h) &= -h^T W x - c \cdot x - b \cdot h \\
&= - \sum_j \sum_k W_{j,k} h_j x_k - \sum_k c_k x_k - \sum_j b_j h_j
\end{align*}
\]

\[
\text{Distribution: } p(x, h) = \exp(-E(x, h))/Z
\]

(partition function (intractable))
NEURAL NETWORKS

• What we’ll cover
  ‣ how neural networks take input \( x \) and make predict \( f(x) \)
    - forward propagation
    - types of units
  ‣ how to train neural nets (classifiers) on data
    - loss function
    - backpropagation
    - gradient descent algorithms
    - tricks of the trade
  ‣ deep learning
    - unsupervised pre-training
    - dropout
    - batch normalization
Neural Networks
Making predictions with feedforward neural networks
ARTIFICIAL NEURON

**Topics:** connection weights, bias, activation function

- Neuron pre-activation (or input activation):

  \[ a(x) = b + \sum_i w_i x_i = b + w^T x \]

- Neuron (output) activation

  \[ h(x) = g(a(x)) = g(b + \sum_i w_i x_i) \]

  *w* are the connection weights
  
  *b* is the neuron bias

  *g(·)* is called the activation function
Topics: connection weights, bias, activation function

Math for my slides "Feedforward neural network".

\begin{align*}
a(x) &= b_i \sum_i w_i x_i = b_i + w \cdot x \\
h(x) &= g(a(x)) = g(b_i \sum_i w_i x_i) \\
\end{align*}

Range determined by $g(\cdot)$

Bias $b$ only changes the position of the riff

(from Pascal Vincent’s slides)
CAPACITY OF NEURAL NETWORK

**Topics:** single hidden layer neural network

(from Pascal Vincent’s slides)
CAPACITY OF NEURAL NETWORK

Topics: single hidden layer neural network

(from Pascal Vincent’s slides)
Topics: single hidden layer neural network

(Capacity of Neural Network)

(from Pascal Vincent's slides)
Topics: universal approximation

• Universal approximation theorem (Hornik, 1991):
  ▸ “a single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units”

• The result applies for sigmoid, tanh and many other hidden layer activation functions

• This is a good result, but it doesn’t mean there is a learning algorithm that can find the necessary parameter values!
Topics: multilayer neural network

• Could have $L$ hidden layers:
  
  ‣ layer pre-activation for $k > 0$ \( h^{(0)}(x) = x \)
  
  \[
  a^{(k)}(x) = b^{(k)} + W^{(k)}h^{(k-1)}(x)
  \]

  ‣ hidden layer activation ($k$ from 1 to $L$):
  
  \[
  h^{(k)}(x) = g(a^{(k)}(x))
  \]

  ‣ output layer activation ($k = L+1$):
  
  \[
  h^{(L+1)}(x) = o(a^{(L+1)}(x)) = f(x)
  \]
ACTIVATION FUNCTION

**Topics:** sigmoid activation function

- Squashes the neuron's pre-activation between 0 and 1
- Always positive
- Bounded
- Strictly increasing

$$g(a) = \text{sigm}(a) = \frac{1}{1+\exp(-a)}$$
ACTIVATION FUNCTION

**Topics:** hyperbolic tangent ("tanh") activation function

- Squashes the neuron's pre-activation between -1 and 1
- Can be positive or negative
- Bounded
- Strictly increasing

\[ g(a) = \tanh(a) = \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)} = \frac{\exp(2a) - 1}{\exp(2a) + 1} \]
**ACTIVATION FUNCTION**

**Topics:** rectified linear activation function

- Bounded below by 0 (always non-negative)
- Not upper bounded
- Strictly increasing
- Tends to give neurons with sparse activities

\[ g(a) = \text{reclin}(a) = \max(0, a) \]
**ACTIVATION FUNCTION**

**Topics:** softmax activation function

• For multi-class classification:
  ‣ we need multiple outputs (1 output per class)
  ‣ we would like to estimate the conditional probability $p(y = c|x)$

• We use the softmax activation function at the output:

  $$o(a) = \text{softmax}(a) = \left[ \frac{\exp(a_1)}{\sum_c \exp(a_c)} \ldots \frac{\exp(a_C)}{\sum_c \exp(a_c)} \right]^T$$

  ‣ strictly positive
  ‣ sums to one

• Predicted class is the one with highest estimated probability
**Topics:** flow graph

• Forward propagation can be represented as an acyclic flow graph

• It's a nice way of implementing forward propagation in a modular way
  
  ‣ each box could be an object with an fprop method, that computes the value of the box given its parents
  
  ‣ calling the fprop method of each box in the right order yield forward propagation
Neural Networks

Training feedforward neural networks
**Topics:** empirical risk minimization, regularization

- **Empirical (structural) risk minimization**
  - framework to design learning algorithms

  \[
  \arg \min_{\theta} \frac{1}{T} \sum_{t} l(f(x^{(t)}; \theta), y^{(t)}) + \lambda \Omega(\theta)
  \]

  - \(l(f(x^{(t)}; \theta), y^{(t)})\) is a loss function
  - \(\Omega(\theta)\) is a regularizer (penalizes certain values of \(\theta\))

- **Learning is cast as optimization**
  - ideally, we’d optimize classification error; but it’s not smooth
  - loss function is a surrogate for what we truly should optimize (e.g. upper bound)
Supervised learning example:

To apply this algorithm to neural network training, we need

- algorithm that performs updates after each example

\[ \theta \ ( \ \theta \equiv \{ W^{(1)}, b^{(1)}, \ldots, W^{(L+1)}, b^{(L+1)} \} ) \]

- for N epochs

\[ \begin{align*}
\Delta &= -\nabla_{\theta}l(f(x^{(t)}; \theta), y^{(t)}) - \lambda \nabla_{\theta} \Omega(\theta) \\
\theta &\leftarrow \theta + \alpha \Delta
\end{align*} \]

- training epoch = iteration over all examples

- To apply this algorithm to neural network training, we need

\[ l(f(x^{(t)}; \theta), y^{(t)}) \]

- the loss function

\[ \nabla_{\theta}l(f(x^{(t)}; \theta), y^{(t)}) \]

- a procedure to compute the parameter gradients

\[ \Omega(\theta) \] (and the gradient \( \nabla_{\theta} \Omega(\theta) \))

- the regularizer

- initialization method for \( \theta \)
Topics: loss function for classification

- Neural network estimates $f(x)_c = p(y = c|x)$
  - we could maximize the probabilities of $y^{(t)}$ given $x^{(t)}$ in the training set

- To frame as minimization, we minimize the negative log-likelihood

  $$l(f(x), y) = - \sum_c 1_{(y=c)} \log f(x)_c = - \log f(x)_y$$

- we take the log to simplify for numerical stability and math simplicity
- sometimes referred to as cross-entropy
BACKPROPAGATION

**Topics:** backpropagation algorithm

- Use the chain rule to efficiently compute gradients, *top to bottom*
  - compute output gradient (before activation)
    \[ \nabla_{a^{(L+1)}(x)} \log f(x)_y \Leftarrow -(e(y) - f(x)) \]
  - for \( k \) from \( L+1 \) to 1
    - compute gradients of hidden layer parameter
      \[ \nabla_{W^{(k)}} \log f(x)_y \Leftarrow \left( \nabla_{a^{(k)}(x)} \log f(x)_y \right) h^{(k-1)}(x)^T \]
      \[ \nabla_{b^{(k)}} \log f(x)_y \Leftarrow \nabla_{a^{(k)}(x)} \log f(x)_y \]
    - compute gradient of hidden layer below
      \[ \nabla_{h^{(k-1)}(x)} \log f(x)_y \Leftarrow W^{(k)^T} \left( \nabla_{a^{(k)}(x)} \log f(x)_y \right) \]
    - compute gradient of hidden layer below (before activation)
      \[ \nabla_{a^{(k-1)}(x)} \log f(x)_y \Leftarrow \left( \nabla_{h^{(k-1)}(x)} \log f(x)_y \right) \odot [\ldots, g'(a^{(k-1)}(x)_j), \ldots] \]
Topics: sigmoid activation function gradient

- Partial derivative:

\[ g'(a) = g(a)(1 - g(a)) \]

\[ g(a) = \text{sigm}(a) = \frac{1}{1 + \exp(-a)} \]
**ACTIVATION FUNCTION**

**Topics:** tanh activation function gradient

- Partial derivative:
  \[
g'(a) = 1 - g(a)^2
\]

\[
g(a) = \tanh(a) = \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)} = \frac{\exp(2a) - 1}{\exp(2a) + 1}
\]
Topics: rectified linear activation function gradient

- Partial derivative:

\[ g'(a) = 1_{a > 0} \]

\[ g(a) = \text{reclin}(a) = \max(0, a) \]
**Topics:** automatic differentiation

- Each object also has a bprop method
  - it computes the gradient of the loss with respect to each parent
  - fprop depends on the fprop of a box’s parents, while bprop depends the bprop of a box’s children
- By calling bprop in the reverse order, we get backpropagation
  - only need to reach the parameters
Topics: automatic differentiation

- Each object also has a bprop method
  - it computes the gradient of the loss with respect to each parent
  - fprop depends on the fprop of a box’s parents, while bprop depends the bprop of a box’s children
- By calling bprop in the reverse order, we get backpropagation
  - only need to reach the parameters
Topics: automatic differentiation

- Each object also has a bprop method
  - it computes the gradient of the loss with respect to each parent
  - fprop depends on the fprop of a box's parents, while bprop depends the bprop of a box's children
- By calling bprop in the reverse order, we get backpropagation
  - only need to reach the parameters
**Topics:** automatic differentiation

- Each object also has a bprop method
  - it computes the gradient of the loss with respect to each parent
  - fprop depends on the fprop of a box's parents, while bprop depends the bprop of a box's children
- By calling bprop in the reverse order, we get backpropagation
  - only need to reach the parameters
**Topics:** automatic differentiation

- Each object also has a bprop method
  - it computes the gradient of the loss with respect to each parent
  - fprop depends on the fprop of a box’s parents, while bprop depends the bprop of a box’s children
- By calling bprop in the reverse order, we get backpropagation
  - only need to reach the parameters
**FLOW GRAPH**

**Topics:** automatic differentiation

- Each object also has a bprop method
  - it computes the gradient of the loss with respect to each parent
  - fprop depends on the fprop of a box’s parents, while bprop depends the bprop of a box’s children
- By calling bprop in the reverse order, we get backpropagation
  - only need to reach the parameters
**Topics:** automatic differentiation

- Each object also has a bprop method
  - it computes the gradient of the loss with respect to each parent
  - fprop depends on the fprop of a box's parents, while bprop depends the bprop of a box's children
- By calling bprop in the reverse order, we get backpropagation
  - only need to reach the parameters
**Topics:** automatic differentiation

- Each object also has a bprop method
  - it computes the gradient of the loss with respect to each parent
  - fprop depends on the fprop of a box’s parents, while bprop depends the bprop of a box’s children

- By calling bprop in the reverse order, we get backpropagation
  - only need to reach the parameters
**Topics:** automatic differentiation

- Each object also has a bprop method
  - it computes the gradient of the loss with respect to each parent
  - fprop depends on the fprop of a box’s parents, while bprop depends the bprop of a box’s children
- By calling bprop in the reverse order, we get backpropagation
  - only need to reach the parameters
FLOW GRAPH

**Topics:** automatic differentiation

- Each object also has a bprop method
  - it computes the gradient of the loss with respect to each parent
  - fprop depends on the fprop of a box’s parents, while bprop depends the bprop of a box’s children
- By calling bprop in the reverse order, we get backpropagation
  - only need to reach the parameters
**Topics:** L2 regularization

\[
\Omega(\theta) = \sum_k \sum_i \sum_j \left( W_{i,j}^{(k)} \right)^2 = \sum_k \| W^{(k)} \|_F^2
\]

- **Gradient:** \( \nabla_{W^{(k)}} \Omega(\theta) = 2W^{(k)} \)

- Only applied on weights, not on biases (weight decay)
- Can be interpreted as having a Gaussian prior over the weights
**Topics:** initialization

- For biases
  - initialize all to 0

- For weights
  - Can’t initialize weights to 0 with tanh activation
    - we can show that all gradients would then be 0 (saddle point)
  - Can’t initialize all weights to the same value
    - we can show that all hidden units in a layer will always behave the same
    - need to break symmetry
  - Recipe: sample $\mathbf{W}_{i,j}^{(k)}$ from $U[-b, b]$ where $b = \frac{\sqrt{6}}{\sqrt{H_k + H_{k-1}}}$
    - the idea is to sample around 0 but break symmetry
    - other values of $b$ could work well (not an exact science) (see Glorot & Bengio, 2010)
MODEL SELECTION

**Topics:** grid search, random search

• To search for the best configuration of the hyper-parameters:
  ‣ you can perform a grid search
    ‧ specify a set of values you want to test for each hyper-parameter
    ‧ try all possible configurations of these values
  ‣ you can perform a random search (Bergstra and Bengio, 2012)
    ‧ specify a distribution over the values of each hyper-parameters (e.g. uniform in some range)
    ‧ sample independently each hyper-parameter to get configurations
  ‣ bayesian optimization or sequential model-based optimization …

• Use a **validation set** (not the test set) performance to select the best configuration

• You can go back and refine the grid/distributions if needed
Topics: early stopping

- To select the number of epochs, stop training when validation set error increases (with some look ahead)
OTHER TRICKS OF THE TRADE

**Topics:** normalization of data, decaying learning rate

• Normalizing your (real-valued) data
  ‣ for dimension $x_i$ subtract its training set mean
  ‣ divide by dimension $x_i$ by its training set standard deviation
  ‣ this can speed up training (in number of epochs)

• Decaying the learning rate
  ‣ as we get closer to the optimum, makes sense to take smaller update steps
    (i) start with large learning rate (e.g. 0.1)
    (ii) maintain until validation error stops improving
    (iii) divide learning rate by 2 and go back to (ii)
OTHER TRICKS OF THE TRADE

**Topics:** mini-batch, momentum

- Can update based on a mini-batch of example (instead of 1 example):
  - the gradient is the average regularized loss for that mini-batch
  - can give a more accurate estimate of the risk gradient
  - can leverage matrix/matrix operations, which are more efficient

- Can use an exponential average of previous gradients:

\[
\overline{\nabla}_{\theta}^{(t)} = \nabla_{\theta} l(f(x^{(t)}), y^{(t)}) + \beta \overline{\nabla}_{\theta}^{(t-1)}
\]

  - can get through plateaus more quickly, by “gaining momentum”
OTHER TRICKS OF THE TRADE

**Topics:** Adagrad, RMSProp, Adam

- Updates with adaptive learning rates ("one learning rate per parameter")
  - **Adagrad:** learning rates are scaled by the square root of the cumulative sum of squared gradients
    \[
    \gamma(t) = \gamma(t-1) + \left( \nabla_{\theta} l(f(x^{(t)}), y^{(t)}) \right)^2
    \]
    \[
    \nabla_{\theta}^{(t)} = \frac{\nabla_{\theta} l(f(x^{(t)}), y^{(t)})}{\sqrt{\gamma(t) + \epsilon}}
    \]
  - **RMSProp:** instead of cumulative sum, use exponential moving average
    \[
    \gamma(t) = \beta \gamma(t-1) + (1 - \beta) \left( \nabla_{\theta} l(f(x^{(t)}), y^{(t)}) \right)^2
    \]
    \[
    \nabla_{\theta}^{(t)} = \frac{\nabla_{\theta} l(f(x^{(t)}), y^{(t)})}{\sqrt{\gamma(t) + \epsilon}}
    \]
  - **Adam:** essentially combines RMSProp with momentum
GRADIENT CHECKING

Topics: finite difference approximation

- To debug your implementation of fprop/bprop, you can compare with a finite-difference approximation of the gradient

\[ \frac{\partial f(x)}{\partial x} \approx \frac{f(x + \epsilon) - f(x - \epsilon)}{2\epsilon} \]

- \( f(x) \) would be the loss
- \( x \) would be a parameter
- \( f(x + \epsilon) \) would be the loss if you add \( \epsilon \) to the parameter
- \( f(x - \epsilon) \) would be the loss if you subtract \( \epsilon \) to the parameter
DEBUGGING ON SMALL DATASET

**Topics:** debugging on small dataset

- Next, make sure your model is able to (over)fit on a very small dataset (~50 examples)

- If not, investigate the following situations:
  - Are some of the units saturated, even before the first update?
    - scale down the initialization of your parameters for these units
    - properly normalize the inputs
  - Is the training error bouncing up and down?
    - decrease the learning rate

- Note that this isn’t a replacement for gradient checking
  - could still overfit with some of the gradients being wrong
Neural Networks
Training deep feed-forward neural networks
Topics: inspiration from visual cortex
Topics: inspiration from visual cortex
Topics: inspiration from visual cortex

[picture from Simon Thorpe]
Topics: inspiration from visual cortex
Topics: inspiration from visual cortex

[picture from Simon Thorpe]
Topics: inspiration from visual cortex
Topics: inspiration from visual cortex

[picture from Simon Thorpe]
Topics: theoretical justification

• A deep architecture can represent certain functions (exponentially) more compactly

• Example: Boolean functions
  ‣ a Boolean circuit is a sort of feed-forward network where hidden units are logic gates (i.e. AND, OR or NOT functions of their arguments)
  ‣ any Boolean function can be represented by a “single hidden layer” Boolean circuit
    - however, it might require an exponential number of hidden units
  ‣ it can be shown that there are Boolean functions which
    - require an exponential number of hidden units in the single layer case
    - require a polynomial number of hidden units if we can adapt the number of layers
  ‣ See “Exploring Strategies for Training Deep Neural Networks” for a discussion
### Topics: success story: speech recognition

**Speech Recognition Leaps Forward**

By Janie Chang  
August 26, 2011 12:01 AM PT

During **Interspeech 2011**, the 12th annual Conference of the International Speech Communication Association being held in Florence, Italy, from Aug. 28 to 31, researchers from Microsoft Research will present work that dramatically improves the potential of real-time, speaker-independent, automatic speech recognition.

**Dong Yu**, researcher at **Microsoft Research Redmond**, and **Frank Seide**, senior researcher and research manager with **Microsoft Research Asia**, have been spearheading this work, and their teams have collaborated on what has developed into a research breakthrough in the use of artificial neural networks for large-vocabulary speech recognition.

**The Holy Grail of Speech Recognition**

Commercially available speech-recognition technology is behind applications such...
Topics: success story: computer vision

Figure 4: (Left) Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). (Right) Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.

In the left panel of Figure 4 we qualitatively assess what the network has learned by computing its top-5 predictions on eight test images. Notice that even off-center objects, such as the mite in the top-left, can be recognized by the net. Most of the top-5 labels appear reasonable. For example, only other types of cat are considered plausible labels for the leopard. In some cases (grille, cherry) there is genuine ambiguity about the intended focus of the photograph.

Another way to probe the network’s visual knowledge is to consider the feature activations induced by an image at the last, 4096-dimensional hidden layer. If two images produce feature activation vectors with a small Euclidean separation, we can say that the higher levels of the neural network consider them to be similar. Figure 4 shows five images from the test set and the six images from the training set that are most similar to each of them according to this measure. Notice that at the pixel level, the retrieved training images are generally not close in L2 to the query images in the first column. For example, the retrieved dogs and elephants appear in a variety of poses. We present the results for many more test images in the supplementary material.

Computing similarity by using Euclidean distance between two 4096-dimensional, real-valued vectors is inefficient, but it could be made efficient by training an auto-encoder to compress these vectors to short binary codes. This should produce a much better image retrieval method than applying auto-encoders to the raw pixels [14], which does not make use of image labels and hence has a tendency to retrieve images with similar patterns of edges, whether or not they are semantically similar.

Discussion
Our results show that a large, deep convolutional neural network is capable of achieving record-breaking results on a highly challenging dataset using purely supervised learning. It is notable that our network’s performance degrades if a single convolutional layer is removed. For example, removing any of the middle layers results in a loss of about 2% for the top-1 performance of the network. So the depth really is important for achieving our results.

To simplify our experiments, we did not use any unsupervised pre-training even though we expect that it will help, especially if we obtain enough computational power to significantly increase the size of the network without obtaining a corresponding increase in the amount of labeled data. Thus far, our results have improved as we have made our network larger and trained it longer but we still have many orders of magnitude to go in order to match the infero-temporal pathway of the human visual system. Ultimately we would like to use very large and deep convolutional nets on video sequences where the temporal structure provides very helpful information that is missing or far less obvious in static images.
**Topics:** why training is hard

- First hypothesis: optimization is harder (underfitting)
  - vanishing gradient problem
  - saturated units block gradient propagation

- This is a well known problem in recurrent neural networks
**Topics:** why training is hard

- Second hypothesis: overfitting
  - we are exploring a space of complex functions
  - deep nets usually have lots of parameters
- Might be in a high variance / low bias situation
Topics: why training is hard

• Second hypothesis: overfitting
  ‣ we are exploring a space of complex functions
  ‣ deep nets usually have lots of parameters

• Might be in a high variance / low bias situation

low variance/ high bias

good trade-off

high variance/ low bias
Topics: why training is hard

- Depending on the problem, one or the other situation will tend to dominate

- If first hypothesis (underfitting): better optimize
  - use better optimization methods
  - use GPUs

- If second hypothesis (overfitting): use better regularization
  - unsupervised pre-training
  - stochastic «dropout» training
Topics: why training is hard

- Depending on the problem, one or the other situation will tend to dominate

- If first hypothesis (underfitting): better optimize
  - use better optimization methods
  - use GPUs

- If second hypothesis (overfitting): use better regularization
  - unsupervised pre-training
  - stochastic «dropout» training
Topics: unsupervised pre-training

- Solution: initialize hidden layers using unsupervised learning
  - force network to represent latent structure of input distribution
  - encourage hidden layers to encode that structure

character image

random image
Topics: unsupervised pre-training

- Solution: initialize hidden layers using unsupervised learning
  - force network to represent latent structure of input distribution
  - encourage hidden layers to encode that structure

Why is one a character and the other is not?
Topics: unsupervised pre-training

- Solution: initialize hidden layers using unsupervised learning
  - this is a harder task than supervised learning (classification)
  - hence we expect less overfitting

Why is one a character and the other is not?

character image

random image
Topics: autoencoder, encoder, decoder, tied weights

- Feed-forward neural network trained to reproduce its input at the output layer

**Decoder**

\[
\hat{x} = o(\hat{a}(x)) = \text{sigm}(c + W^*h(x))
\]

- For binary inputs

**Encoder**

\[
h(x) = g(a(x)) = \text{sigm}(b + Wx)
\]

Autoencoder, encoder, decoder, tied weights
**Topics:** unsupervised pre-training

- We will use a greedy, layer-wise procedure
  - train one layer at a time, from first to last, with unsupervised criterion
  - fix the parameters of previous hidden layers
  - previous layers viewed as feature extraction
Topics: fine-tuning

• Once all layers are pre-trained
  ‣ add output layer
  ‣ train the whole network using supervised learning

• Supervised learning is performed as in a regular feed-forward network
  ‣ forward propagation, backpropagation and update

• We call this last phase fine-tuning
  ‣ all parameters are “tuned” for the supervised task at hand
  ‣ representation is adjusted to be more discriminative
Why Does Unsupervised Pre-training Help Deep Learning?
Erhan, Bengio, Courville, Manzagol, Vincent and Bengio, 2011
DEEP LEARNING

Topics: impact of initialization

Why Does Unsupervised Pre-training Help Deep Learning?
Erhan, Bengio, Courville, Manzagol, Vincent and Bengio, 2011
Topics: why training is hard

• Depending on the problem, one or the other situation will tend to dominate

• If first hypothesis (underfitting): better optimize
  ‣ use better optimization methods
  ‣ use GPUs

• If second hypothesis (overfitting): use better regularization
  ‣ unsupervised pre-training
  ‣ stochastic «dropout» training
**Topics:** dropout

- Idea: «cripple» neural network by removing hidden units stochastically
  - each hidden unit is set to 0 with probability 0.5
  - hidden units cannot co-adapt to other units
  - hidden units must be more generally useful

- Could use a different dropout probability, but 0.5 usually works well
**Topics:** dropout

- Idea: «cripple» neural network by removing hidden units stochastically
  - each hidden unit is set to 0 with probability 0.5
  - hidden units cannot co-adapt to other units
  - hidden units must be more generally useful

- Could use a different dropout probability, but 0.5 usually works well
Topics: dropout

- Idea: «cripple» neural network by removing hidden units stochastically
  - each hidden unit is set to 0 with probability 0.5
  - hidden units cannot co-adapt to other units
  - hidden units must be more generally useful

- Could use a different dropout probability, but 0.5 usually works well
Topics: dropout

• Use random binary masks $m^{(k)}$
  ‣ layer pre-activation for $k > 0$  \( h^{(0)}(x) = x \)
  \[
a^{(k)}(x) = b^{(k)} + W^{(k)}h^{(k-1)}(x)
\]
  ‣ hidden layer activation ($k$ from 1 to $L$):
  \[
h^{(k)}(x) = g(a^{(k)}(x))
\]
  ‣ output layer activation ($k = L + 1$):
  \[
h^{(L+1)}(x) = o(a^{(L+1)}(x)) = f(x)
\]
Topics: dropout

- Use random binary masks $m^{(k)}$
  - layer pre-activation for $k > 0$ \( h^{(0)}(x) = x \)
    \[
    a^{(k)}(x) = b^{(k)} + W^{(k)} h^{(k-1)}(x)
    \]
  - hidden layer activation ($k$ from 1 to $L$):
    \[
    h^{(k)}(x) = g(a^{(k)}(x)) \odot m^{(k)}
    \]
  - output layer activation ($k = L + 1$):
    \[
    h^{(L+1)}(x) = o(a^{(L+1)}(x)) = f(x)
    \]
Topics: dropout backpropagation

• This assumes a forward propagation has been made before

  ‣ compute output gradient (before activation)
    \[ \nabla_{a^{(L+1)}(x)} - \log f(x)_y \iff - (e(y) - f(x)) \nabla_{a^{(L+1)}(x)} - \log f(x)_y \iff (\nabla_{a^{(k)}(x)} - \log f(x)_y) h^{(k-1)}(x)^T \nabla_{w^{(k)}} - \log f(x)_y \iff (\nabla_{a^{(k)}(x)} - \log f(x)_y) h^{(k-1)}(x)^T \]

  ‣ compute gradients of hidden layer parameter
    \[ \nabla_{b^{(k)}} - \log f(x)_y \iff \nabla_{a^{(k)}(x)} - \log f(x)_y \]

  ‣ compute gradient of hidden layer below
    \[ \nabla_{h^{(k-1)}(x)} - \log f(x)_y \iff W^{(k)}(\nabla_{a^{(k)}(x)} - \log f(x)_y) \]

  ‣ compute gradient of hidden layer below (before activation)
    \[ \nabla_{a^{(k-1)}(x)} - \log f(x)_y \iff (\nabla_{h^{(k-1)}(x)} - \log f(x)_y) \odot [..., g'(a^{(k-1)}(x)_j), ...] \]
Topics: dropout backpropagation

• This assumes a forward propagation has been made before

  ‣ compute output gradient (before activation)
  \[ \nabla a^{(L+1)}(x) - \log f(x)_y \leftarrow - (e(y) - f(x)) \]

  ‣ for \( k \) from \( L+1 \) to 1

    - compute gradients of hidden layer parameter
      \[ \nabla w^{(k)} - \log f(x)_y \leftarrow (\nabla a^{(k)}(x) - \log f(x)_y) \cdot h^{(k-1)}(x)^T \]
      \[ \nabla b^{(k)} - \log f(x)_y \leftarrow \nabla a^{(k)}(x) - \log f(x)_y \]

    - compute gradient of hidden layer below
      \[ \nabla h^{(k-1)}(x) - \log f(x)_y \leftarrow W^{(k)^T} \cdot (\nabla a^{(k)}(x) - \log f(x)_y) \]

    - compute gradient of hidden layer below (before activation)
      \[ \nabla a^{(k-1)}(x) - \log f(x)_y \leftarrow (\nabla h^{(k-1)}(x) - \log f(x)_y) \circ [\ldots, g'(a^{(k-1)}(x)_j), \ldots] \circ m^{(k-1)} \]
**DROPOUT**

**Topics:** test time classification

- At test time, we replace the masks by their expectation
  - this is simply the constant vector 0.5 if dropout probability is 0.5
  - for single hidden layer, can show this is equivalent to taking the geometric average of all neural networks, with all possible binary masks

- Beats regular backpropagation on many datasets, but is slower (~2x)
Topics: why training is hard

• Depending on the problem, one or the other situation will tend to dominate

• If first hypothesis (underfitting): better optimize
  ‣ use better optimization methods
  ‣ use GPUs

• If second hypothesis (overfitting): use better regularization
  ‣ unsupervised pre-training
  ‣ stochastic «dropout» training
Topics: why training is hard

• Depending on the problem, one or the other situation will tend to dominate

• If first hypothesis (underfitting): better optimize
  ‣ use better optimization methods
  ‣ use GPUs

• If second hypothesis (overfitting): use better regularization
  ‣ unsupervised pre-training
  ‣ stochastic «dropout» training

Batch normalization
**Batch Normalization**

**Topics:** batch normalization

- Normalizing the inputs will speed up training
  (Lecun et al. 1998)
  - could normalization also be useful at the level of the hidden layers?

- **Batch normalization** is an attempt to do that
  (Ioffe and Szegedy, 2014)
  - each unit's pre-activation is normalized (mean subtraction, stddev division)
  - during training, mean and stddev is computed for **each minibatch**
  - backpropagation **takes into account** the normalization
  - at test time, the **global mean / stddev is used**
**Batch Normalization**

**Topics:** batch normalization

- **Batch normalization**

| Input: | Values of $x$ over a mini-batch: $\mathcal{B} = \{x_1...m\}$; |
|        | Parameters to be learned: $\gamma, \beta$ |
| Output: | $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$ |

- $\mu_B \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i$        // mini-batch mean
- $\sigma_B^2 \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_B)^2$  // mini-batch variance
- $\tilde{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$  // normalize
- $y_i \leftarrow \gamma \tilde{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i)$  // scale and shift
BATCH NORMALIZATION

Topics: batch normalization

• Batch normalization

Input: Values of \( x \) over a mini-batch: \( \mathcal{B} = \{x_1...m\} \);
Parameters to be learned: \( \gamma, \beta \)
Output: \( \{y_i = \text{BN}_{\gamma,\beta}(x_i)\} \)

\[
\mu_\mathcal{B} \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i \quad \text{// mini-batch mean}
\]

\[
\sigma^2_\mathcal{B} \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2 \quad \text{// mini-batch variance}
\]

\[
\hat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma^2_\mathcal{B} + \epsilon}} \quad \text{// normalize}
\]

\[
y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \quad \text{// scale and shift}
\]

Learned linear transformation to adapt to non-linear activation function
(\( \gamma \) and \( \beta \) are trained)
Topics: online videos

- for a more detailed description of neural networks...
- ... and much more!

http://info.usherbrooke.ca/hlarochelle/neural_networks
**Topics:** online videos

- for a more detailed description of neural networks...
- ... and much more!

http://info.usherbrooke.ca/hlarochelle/neural_networks

**Restricted Boltzmann Machine**

Topics: RBM, visible layer, hidden layer, energy function

\[
E(x, h) = -h^T W x - c^T x - b^T h \\
= -\sum_j \sum_k W_{j,k} h_j x_k - \sum_k c_k x_k - \sum_j b_j h_j
\]

Distribution: \( p(x, h) = \exp(-E(x, h))/Z \)

(partition function (intractable))
MERCI!