Deep Learning with Neural Networks

The Structure and Optimization of Deep Neural Networks

Allan Zelener

Machine Learning Reading Group

January 7th 2016

The Graduate Center, CUNY

Objectives

- Explain some of the trends of deep learning and neural networks in machine learning research.
- Give a theoretical and practical understanding of neural network structure and training.
- Provide baseline for reading neural network papers in machine learning and related fields.
- Brief look at some major work that could be used in further reading group discussions.

Why are neural networks back again?

- State-of-the-art performance on benchmark perception datasets.
  - TIMIT – (Mohamed, Dahl, Hinton 2009)
    - 23.0% phoneme error rate vs 24.4% ensemble method.
    - 17.7% with LSTM RNN (Graves, Mohamed, Hinton 2013)
  - Imagenet – (Krizhevsky, Sutskever, Hinton 2012)
    - 16% top-5 error vs 25% of competing methods.
    - In 2015 deep nets can achieve ~3.5% top-5 error.
- Larger datasets and faster computation.
  - Good enough that industry is now investing resources.
- A few innovations: ReLU, Dropout.

Why should neural networks work?

- No strong and useful theoretical guarantees yet.
- Universal approximation theorems
  - Taylor's theorem (differentiable functions)
  - Stone-Weierstrass theorem (continuous functions)
  - $F(x) = \sum_{i=1}^{N} v_i \phi\left(w_i^T x + b_i\right),\ |F(x) - f(x)| < \epsilon$
    - $F(x)$ is a piecewise constant approximation of $f(x)$.
    - The neural network unit $\phi\left(w_i^T x + b_i\right)$ should be 1 if $f(x) \approx v_i$ and 0 otherwise.
- Optimization of neural networks
  - "Many equally good local minimum" for simplified ideal models.
  - Saddle point problem in non-convex optimization. (Dauphin et al. 2014)
  - Loss surface of multilayer neural networks. (Choromanska et al. 2015)

Why go deeper?

- Deep Neural Networks
  - Universal approximation theorem is for single layer networks.
  - For complicated functions we may need very large N.
  - Empirically, deep networks learn better representations than shallow networks using fewer parameters.
  - For applications where data is highly structured, e.g. vision, facilitates composition of features, feature sharing, and distributed representation.
  - Caveat: Deep nets can be compressed. (Ba and Caruana 2014)

Why go deeper?

- Deep Learning for Representation Learning
  - Classic pipeline
    - Raw Measurements → Features → Prediction.
  - Replace human heuristic feature engineering with learned representations.
  - New pipeline
    - Raw Measurements → Prediction. (Representation is inside the →)
  - End-to-end optimization, but not necessarily a neural network.
  - Caveat: Replaces feature engineering with pipeline engineering.
- Deep Learning as composition of classical models
  - Feed-forward neutral network ≡ Recursive generalized linear model.

Why neural?

- Loosely biologically inspired architecture.
  - LeNet CNN inspired by cat and monkey visual cortex. (Hubel and Wisel 1968)
  - Real neurons respond to simple structures like edges.
- Probably not actually a good model for how brains work, although there may be some similarities.
- Pitfall: Mistaking neural networks for neuroscience.

- I will try to avoid neural inspired jargon where possible but it has become standard in the field.

The Architecture

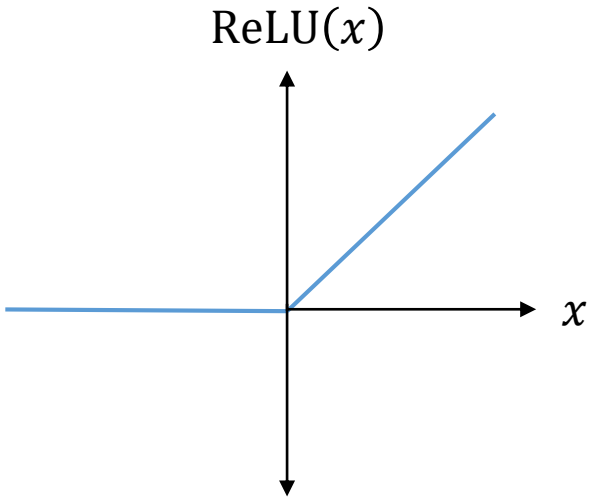
- In machine learning we want to find good approximations to interesting functions $f: X \rightarrow Y$ that describe mappings from observations to useful predictions.
- The approximation function should be:
  - Computationally tractable
  - Nonlinear (if f is nonlinear of course)
  - Parameterizable (so we can learn parameters given training data)

The Architecture - Tractable

- Step 1: Start with a linear function.
  - $y = w^T \boldsymbol{x} + b$ – Linear unit.
  - $\boldsymbol{y} = W\boldsymbol{x} + \boldsymbol{b}$ – Linear layer.
- Efficient to compute, optimized and stable. BLAS libraries and GPUs.
  - $Y = WX + \boldsymbol{b}$ – Linear layer with batch input.
- Easily differentiable, and thus optimizable.
  - $\frac{dy}{dw} = \boldsymbol{x}, \frac{dy}{db} = 1$
- Many parameters, $O(nm)$ for n input dim and m outputs.

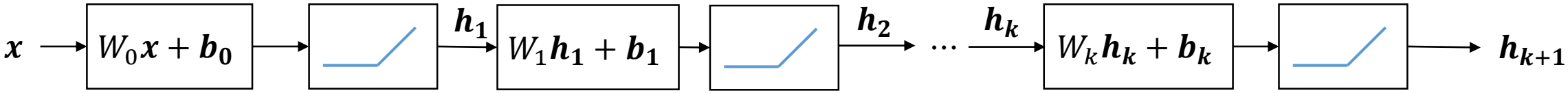
The Architecture - Nonlinear

- Step 2: Add a non-linearity.
  - $\boldsymbol{y} = \phi(W\boldsymbol{x} + \boldsymbol{b})$
  - $\phi(\cdot)$ is some nonlinear function, historically sigmoid.
    - Logistic function σ: $\mathbb{R} \to (0,1)$
    - Hyperbolic tangent $\tanh$: $\mathbb{R} \to (-1,1)$
  - ReLU (Rectified Linear Unit) is a popular choice now.
    - $\mathrm{ReLU}(x) = \max(0, x)$
    - Computationally efficient and surprisingly just works.
    - $\frac{d\mathrm{ReLU}}{dx} \approx \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$
    - Note: ReLU is not differentiable at $x = 0$, but we take 0 for the subgradient.



The Architecture – Parameterizable

- Step 3: Repeat until deep.



- Multilayer Perceptron
- Parameters for linear functions, but entire network is nonlinear.
- Each h_i is called an activation. Internal layers are called hidden.
- Final activation can be used as linear regression.
- Differentiable using backpropagation.

The Architecture

Google

Orders of Magnitude

State of the Art Object Recognition Models:

15-25 layers
10-200M parameters
1-5B multiply-adds / image

- (From Vincent Vanhoucke's slides.)

The Architecture – Classification

- Softmax regression (aka multinomial logistic regression)
- $\boldsymbol{a}(\boldsymbol{x}) = \text{softmax}(\text{evidence}) = \text{normalize}(\boldsymbol{e}^{\boldsymbol{x}}) = \frac{\boldsymbol{e}^{\boldsymbol{x}}}{\|\boldsymbol{e}^{\boldsymbol{x}}\|_1}$
- $a_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} = p(\bar{y} = i \mid x)$ where $\text{class}(x) = \bar{y}$
- Exponentiate $\boldsymbol{x}$ to exaggerate differences between features.
- Normalize so $\boldsymbol{a}$ is a probability distribution over K classes.
- Softmax is a differentiable approximation to the indicator function.
- $\mathbf{1}_{\text{class}(\boldsymbol{x})}(k) = \begin{cases} 1 & k = \arg\max(x_1, \ldots, x_K) = \text{class}(\boldsymbol{x}) \\ 0 & \text{otherwise.} \end{cases}$

The Architecture - Objective

- How far away is the network?
- Let $y = \mathrm{nn}(x, w)$ be the network's prediction on x.
- Let $\bar{y}$ be the "ground truth" target for x.
- Let $L(\bar{y}, y)$ be the loss for our prediction on x.
  - If $y = \bar{y}$ then this should be 0.
  - Squared Euclidean distance/L_2 loss: $\|\bar{y} - y\|_2^2$
  - Cross-entropy/negative log likelihood loss: $-\sum \bar{y} \log y$
- The objective function is $\sum L(y, \bar{y})$ over training pairs $(x, \bar{y})$.

Learning Algorithm for Neural Networks

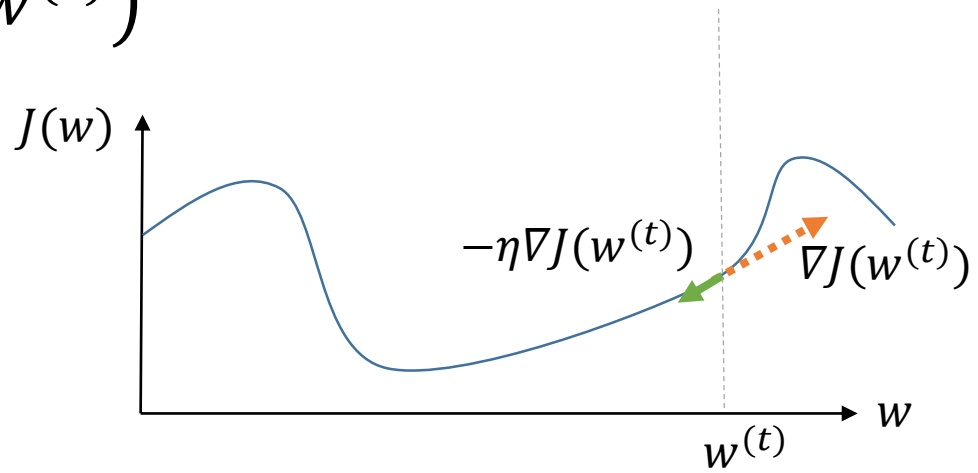
- Training is the process of minimizing the objective function with respect to the weight parameters.

$$w^* = \arg\min_w J(w) = \arg\min_w \sum_{(x,\bar{y})\in T} L\left(\bar{y}, \mathrm{nn}(x, w)\right)$$

- This optimization is done by iterative steps of gradient descent.

$$w^{(t+1)} = w^{(t)} - \eta \nabla J\left(w^{(t)}\right)$$

- $\nabla J(w)$ is the gradient direction.
- η is the learning rate/step size.
  - Needs to be "small enough" for convergence.

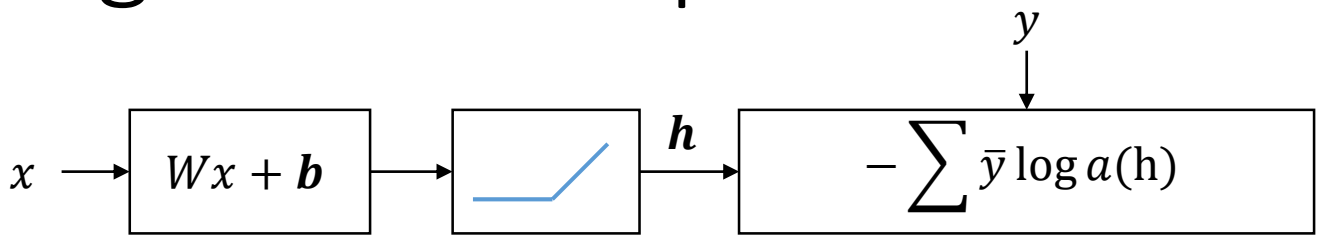


Learning Algorithm for Neural Networks

- $J(w)$ is highly non-linear and non-convex, but it is differentiable.
- The backpropagation algorithm applies the chain rule for derivation backwards through the network.

$$\frac{\partial f_1 \circ f_2 \circ \cdots \circ f_n}{\partial x} = \frac{\partial f_1}{\partial f_2} \cdot \frac{\partial f_2}{\partial f_3} \cdot \ldots \cdot \frac{\partial f_n}{\partial x}$$

Backpropagation Example



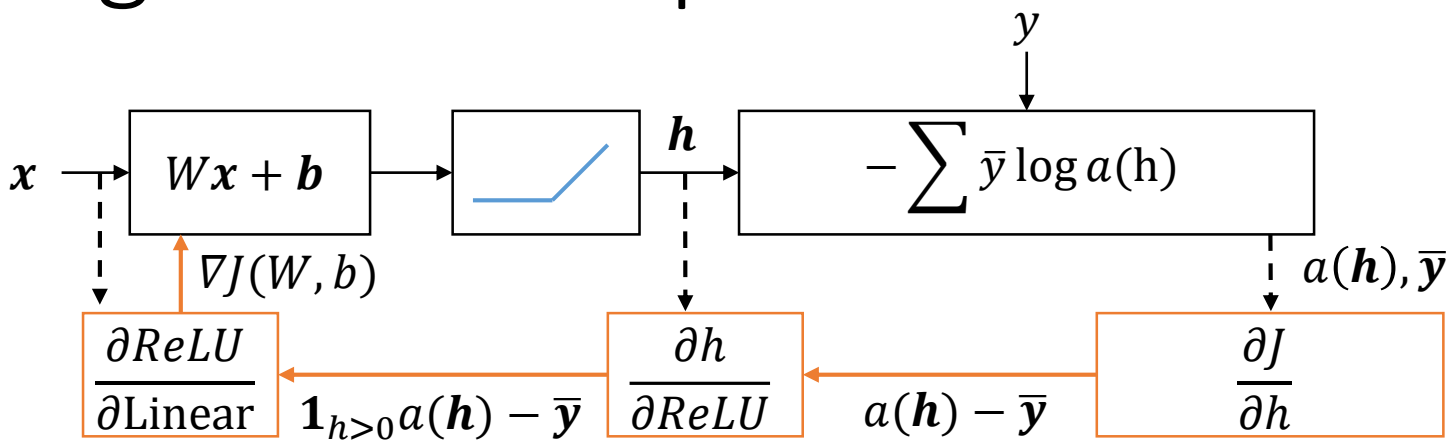
- Let $a(\cdot) = \mathrm{softmax}(\cdot)$
- $\frac{\partial J}{\partial a_i} = -\sum_k \frac{\overline{y_k}}{a_k} \frac{\partial a_k}{\partial h_i}$
- $\frac{\partial a_i}{\partial h_i} = a_i(1 - a_i)$, $\frac{\partial a_i}{\partial h_j} = -a_i a_j$ for $i \neq j$
- $\frac{\partial h_i}{\partial W_i} = \mathbf{1}_{h_i > 0} \cdot \boldsymbol{x}$, $\frac{\partial h_i}{\delta b_i} = \mathbf{1}_{h_i > 0}$

Homework:

Prove $\frac{\partial J}{\partial h_i} = a_i - \overline{y_i}$

and verify softmax derivatives.

Backpropagation Example



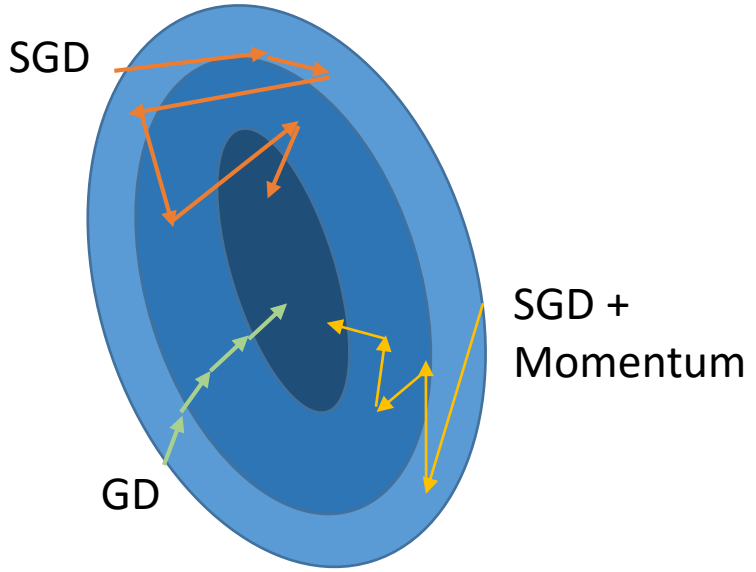
- $\frac{\partial J}{\partial W_i}=\frac{\partial J}{\partial h_i}\frac{\partial h_i}{\delta ReLU_i}\frac{\partial ReLU_i}{\partial \mathrm{Linear}_i}\frac{\partial \mathrm{Linear}_i}{\partial W_i}=(a_i-\bar{y_i})\mathbf{1}_{h_i>0}\cdot\boldsymbol{x}$
- $\frac{\partial J}{\partial b_i}=\frac{\partial J}{\partial h_i}\frac{\partial h_i}{\delta ReLU_i}\frac{\partial ReLU_i}{\partial \mathrm{Linear}_i}\frac{\partial \mathrm{Linear}_i}{\partial b_i}=(a_i-\bar{y_i})\mathbf{1}_{h_i>0}\cdot 1$
- Homework: Work out backprop with two linear layers and a batch of inputs X.

The Data is Too Damn Big

- The objective function for gradient descent requires summing over the entire training set: $J(w) = \sum_{(x,\bar{y})\in T} L\big(\bar{y}, \mathrm{nn}(x, w)\big)$.
- This is too costly for big datasets, we need to approximate.
- Stochastic Gradient Descent uses small batches of the training set. $J(w) \approx \sum_{(x,\bar{y})\in B\subset T} L\big(\bar{y}, \mathrm{nn}(x, w)\big)$.
  - After every batch is used, one epoch, training data is randomly permuted.
  - Poor estimates but repeated many times and smoothed.
  - Online (batch size = 1) might be great if we didn't lose low-level efficiency of batching several examples for matrix multiplications, e.g. $H = WX$.

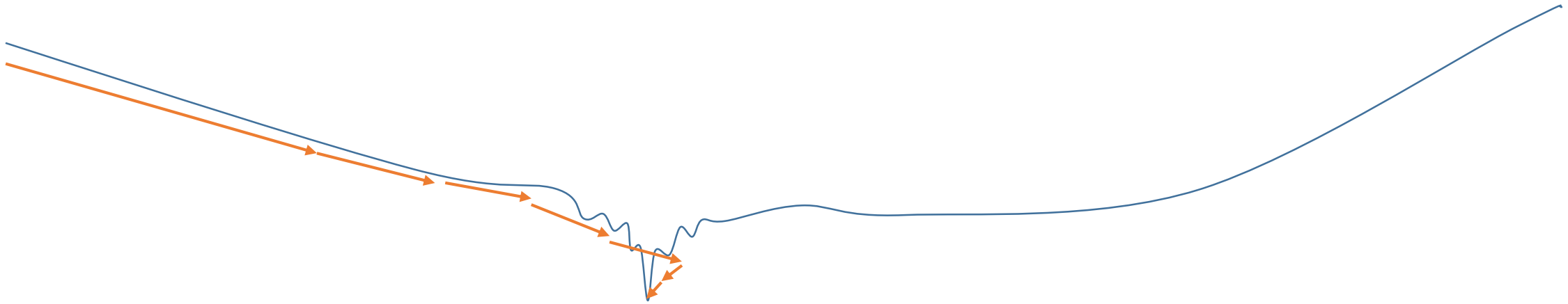
SGD Tricks

- Momentum
- $g^{(t+1)} = \mu g^{(t)} + \nabla J\left(w^{(t)}\right)$
- $w^{(t+1)} = w^{(t)} - \eta g^{(t+1)}$
- Typically large μ, e.g. $\mu = 0.9$.
- Assume we're going the right way over time.
- Reduce impact of noisy estimates.



SGD Tricks

- Learning Rate Decay
- $\eta = \eta_0 e^{-\beta t}$ - Exponentially decaying learning rate.
- Loss surface can have structures at varying scales.
- Anneal learning rate to explore various scales.



SGD Tricks

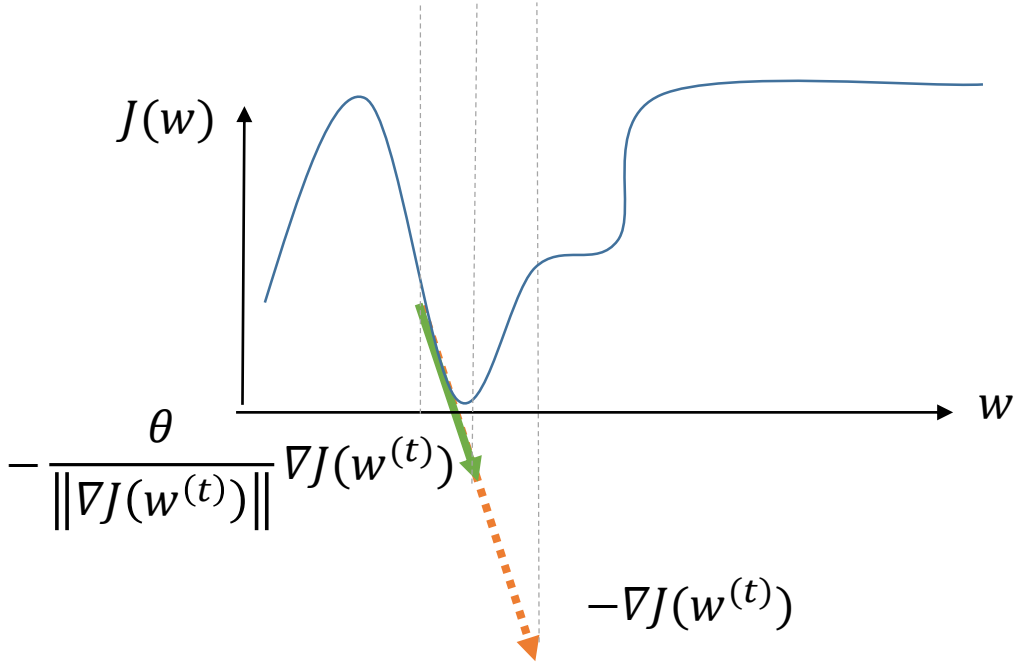
- AdaGrad (Duchi, Hazan, Singer 2011)
- Adaptive per-parameter learning rate decay.
- Keep history of squared L2 norm for each parameter w.
- $h^{(t+1)} = h^{(t)} + \nabla J^2\left(w^{(t)}\right)$
- $w^{(t+1)} = w^{(t)} - \frac{\eta}{\sqrt{h^{(t+1)}}} \nabla J\left(w^{(t)}\right)$
- Intuitively, update less if there have been many changes and update more if there have been few changes.

SGD Tricks

- Averaged SGD - Parameter averaging
- $\widehat{w}^{(t)} = \frac{1}{t-t_0}\sum_{i=t_0+1}^{t} w^{(i)} = \widehat{w}^{(t-1)} + \frac{1}{t-t_0}\left(w^{(t)} - \widehat{w}^{(t-1)}\right)$
- Smooth fluxuation in the model over time.
- SGD may be biased to most recent batches.
- Like an ensemble of models through time.
- Can be computed without storing all parameters over time.
- Note: Use average parameters only at test time, not training.
  - Why?

SGD Tricks

- Gradient Clipping
- If $\left\|\nabla J(w^{(t)})\right\| > \theta$ then $w^{(t+1)} = w^{(t)} - \frac{\eta\theta}{\left\|\nabla J(w^{(t)})\right\|}\nabla J(w^{(t)})$
- Prevent big jumps from exploding gradients.



SGD Tricks

- Other popular variants on SGD update:
  - Nesterov Accelerated Gradient
  - Adadelta
  - RMSprop
  - ADAM
- Visual comparison of some of these methods:
  - http://imgur.com/a/Hqolp
- Second-order alternatives to SGD
  - Conjugate Gradient, L-BFGS

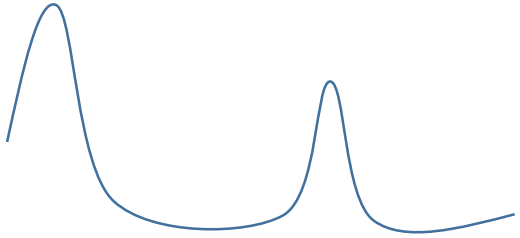
Initialization

- Weight initialization
- Weights need to be numerically stable.
  - Saturation: Units stuck outputting extremal values.
  - Exploding and shrinking gradients.
  - Uncentered input can make gradients always positive or negative.
  - Units with identical weights produce same results, break symmetries.
  - Keep norm of weights throughout network roughly equal.
- Initialize $w \sim \mathcal{N}(0, \sigma)$. Set σ based on $\dim(\boldsymbol{x})^{-\frac{1}{2}}$ *(for sigmoid).*
- Initialize b as small positive values so ReLU will be nonzero.

Initialization

- Weights at the Loss Layer
- $\mathcal{L} = -\sum \bar{\boldsymbol{y}} \log \boldsymbol{y}$
  - $\|\boldsymbol{y}\|$ is “peakiness” or “temperature” of predicted probability distribution.
  - Determines magnitude of gradients backpropagated.
  - Bigger peaks = big errors = bigger gradients.
  - Initialize with small weights, small peak distribution.
  - Anneals to more peaky distribution as classifier gets more confident.

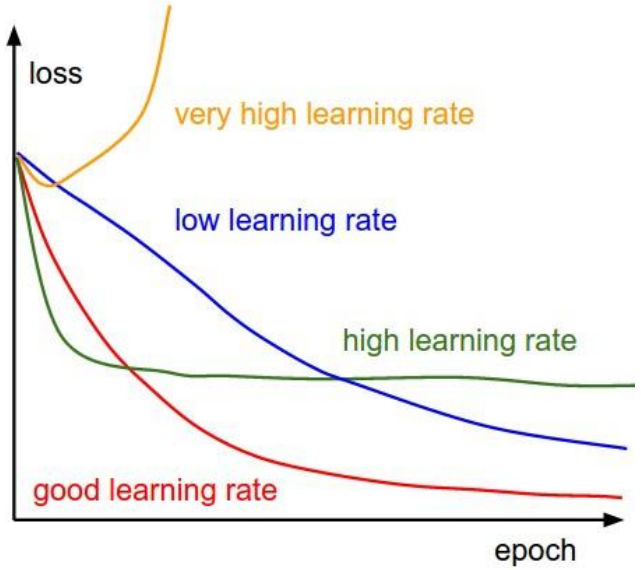
High temperature, low peakiness
Many small weights, small gradients



Low temperature, high peakiness
Few big weights, big gradients

The Most Important Training Tip

- Lower your learning rate by 10x and try again.
  - 0.1 is often a recommended default.
  - Going down to 0.0001 or lower may be needed sometimes.
  - Faster learning rate ≠ better training.



Regularization

- Underfitting
  - The model does not have enough parameters to represent the complexity of the target function.
- Overfitting
  - The model has too many degrees of freedom and tries to represent every little detail in the training data.
  - Poor performance on heldout dataset relative to training set.
- Solution: Take the biggest model we can train but nudge the parameters to generalize better.

Regularization

- L2 Norm Regularization
- Penalize large weights by augmenting loss function.
- Large weights describe extreme examples on the decision boundary.
- $\mathcal{L}(x, \bar{y}; w) = L\big(\bar{y}, \mathrm{nn}(x, w)\big) + \frac{\lambda}{2}\|w\|_2^2$
- Hyperparameter λ controls contribution of regularization.

Regularization

- Dropout
- Probability $p(= 0.5)$ to set output of a unit to 0 during training.
- At test time multiply each output by p.
- Force other units to learn redundant representations.
- Prevent units from relying on input from other specific units.
- Approximates an ensemble method using one network.
  - Each redundant path is like a weak classifier.
  - Multiplying by p like taking geometric mean.

Starting Out on a New Problem

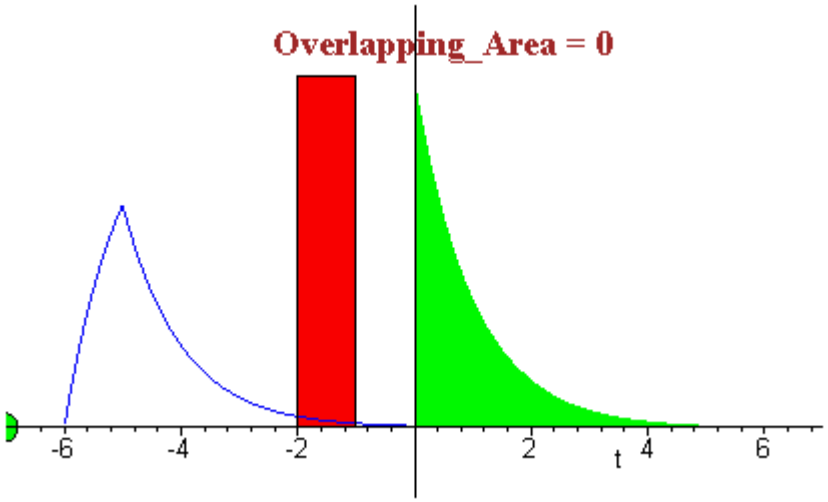
- First try logistic regression, random forests, or gradient boosting.
  - Good performance with fewer hyperparameters.
- Try feasibility of data/problem on simple models and a small dataset.
- For a neural net starting with 2-3 layers with 128 – 1024 units per layer is reasonable, then scale up as needed.
  - Very recent work on transferring parameters to wider and deeper nets:
  - Net2Net: Accelerating Learning via Knowledge Transfer (Chen, Goodfellow, Shlens 2015)

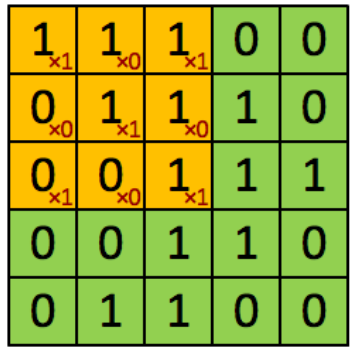
Common Variations

- Convolutional Neural Networks
  - Spatially tied (reused) weights for imagery.
- Recurrent Neural Networks
  - Temporally tied weights for recurrent connections and sequences.
  - May be stateful, next output depends on previous inputs.
- Embeddings
  - Find semantic preserving learned representation.
  - Simple math operations in feature space have semantic properties.
- Generative Models
  - Learn $p(x, y)$ or $p(x|y)$ instead of just $p(y|x)$.

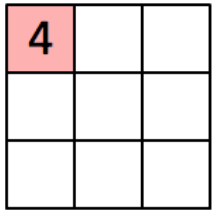
Convolution/Cross-Correlation

- Convolution: $(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$
- Can be thought of as a sliding dot product.
- Overlap of two functions as they slide across each other.
- Many applications and interpretations exist.





Image



Convolved Feature

Convolutional Neural Networks

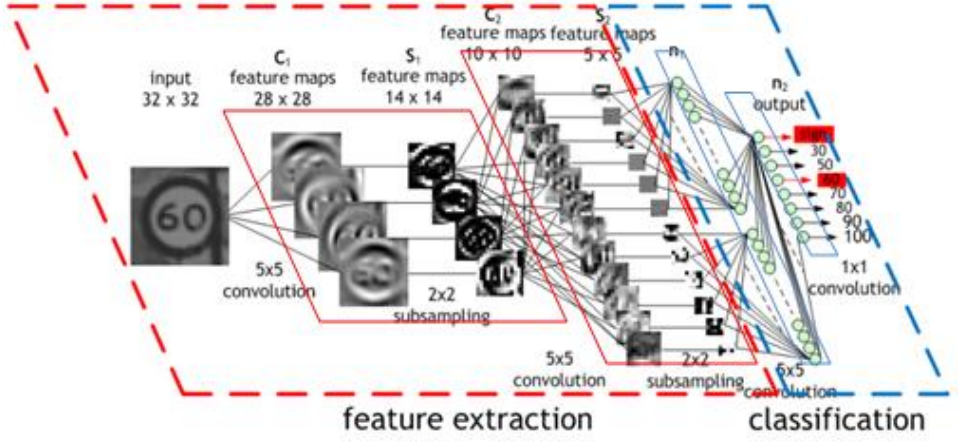
- High dimensional $\boldsymbol{x}$, but similar structure.
- Balloons can occur in many places in an image.
- Replace $W\boldsymbol{x}$ with $W' * \boldsymbol{x}$ where W' is much smaller than W.

A Note on Tensors (Multidimensional Arrays)

- For many types of input (images, video, etc.) it's helpful to think of your data as a tensor rather than a feature vector.
- Each layer of a network transforms one tensor into another.
- Convolution uses assumptions on original tensor structure and roughly preserves it, e.g. image dimensions.
- But mostly still just doing dot products with tensor components.
  - Other operations are possible.
  - See Recursive Neural Tensor Networks (Socher et al. 2013)

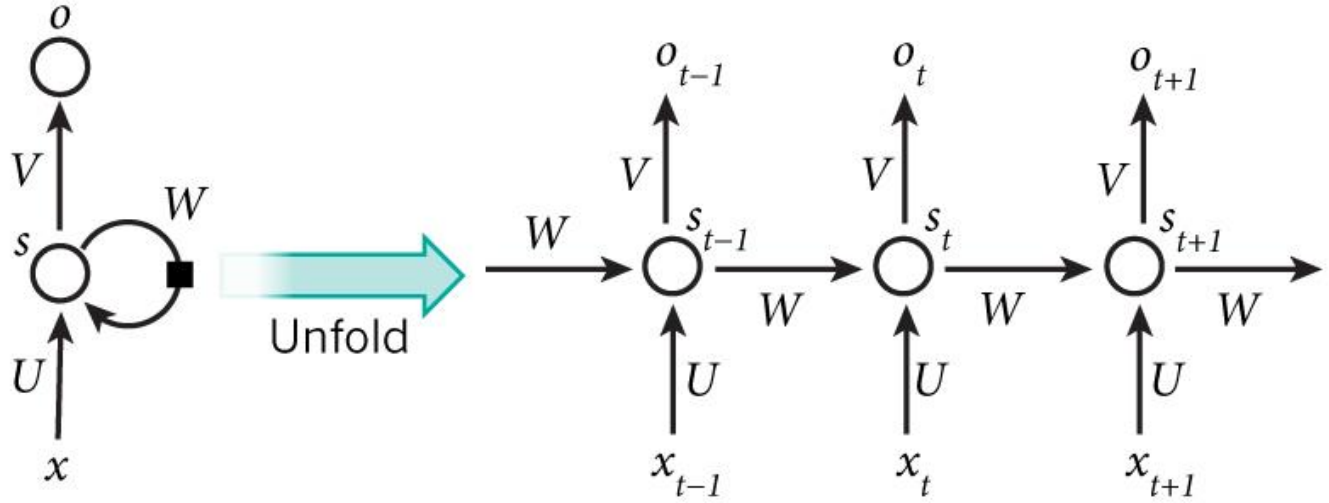
Convolutional Layer Example

- If $\boldsymbol{x}$ is a 320x240x3 RGB image and we want a single output per pixel, then W is 240,400 x 76,800. If we convolve with a 5x5 filter then W' is 75 x 1 and is applied 76,800 times.
- ~18 billion add/mul operations vs ~5 million. 3200 times more.
- Fewer parameters, fewer operations, models stationarity in space.
- Often used with pooling for further reduction and spatial invariance.



Recurrent Neural Networks

- Allow a layer to take input from itself.
- Input can also vary, e.g. a sequence $x_1, x_2, \ldots, x_n$.
- In practice, approximated by discrete number of time steps.
- Can simply backpropagate through unrolled time steps.
- Reuse weights at each time step.
- $s_t = [U \mid W] \begin{bmatrix} x_t \\ s_{t-1} \end{bmatrix}$

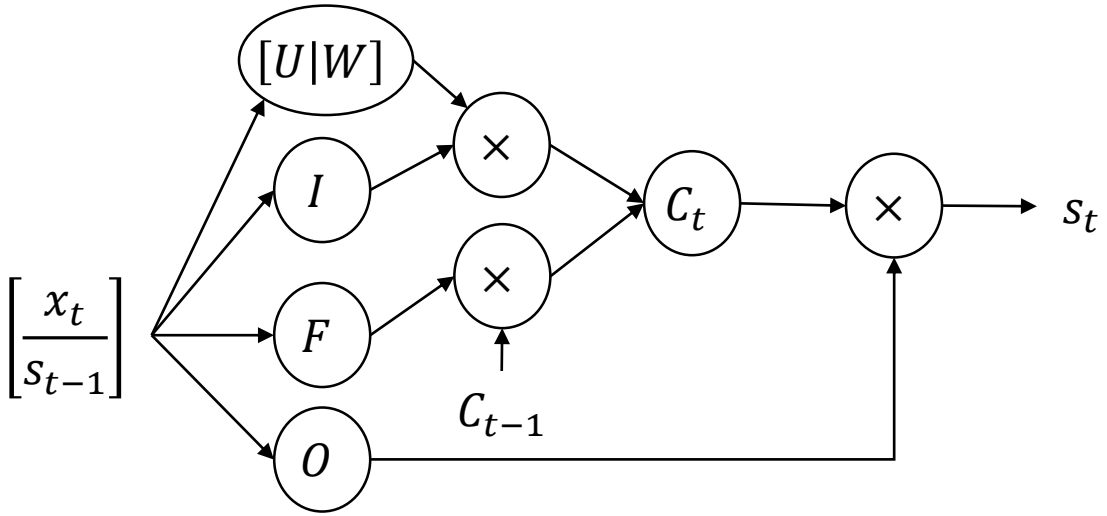


LSTMs (Long Short-Term Memory)

- LSTMs are a type of RNN that work well in practice.
- Conceptually adds "memory" cell C that can be controlled over time.
- Three control "gates": Input, Forget, Output.
- Each gate is a linear layer with sigmoid (0,1) nonlinearity.
  - Differentiable control so we can learn these parameters using backprop.
- Outputs s_t and C_t are a function of $x_t, s_{t-1}, \text{and } C_{t-1}$.
- Many variations exist, for a more detailed breakdown see Christopher Olah's blog post on LSTMs.

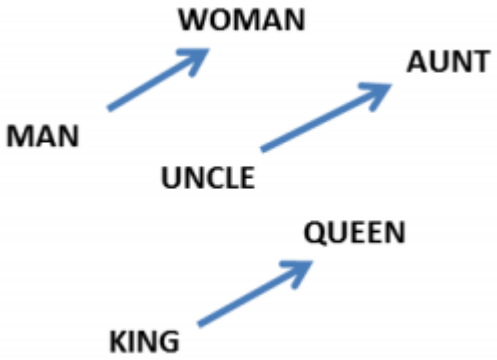
LSTMs

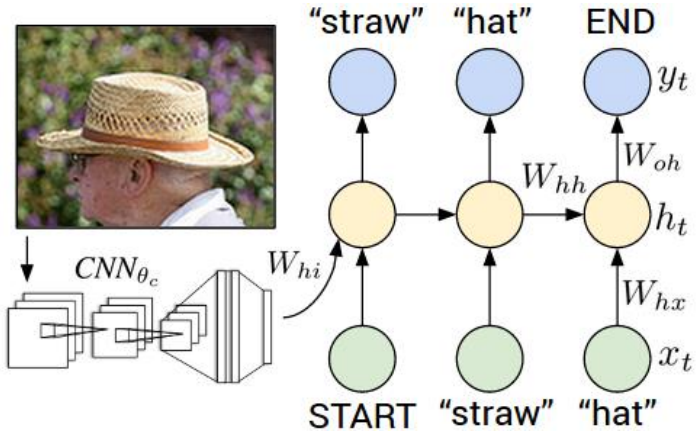
- Each gate uses the current input x_t and previous output s_{t-1}.
- Input gate I controls use of RNN result on x_t and s_{t-1}.
- Forget gate F controls use of previous cell state C_{t-1}.
- Next cell state C_t is determined by RNN result, C_{t-1}, I and F.
- Output gate controls the next output s_t as given by C_t.

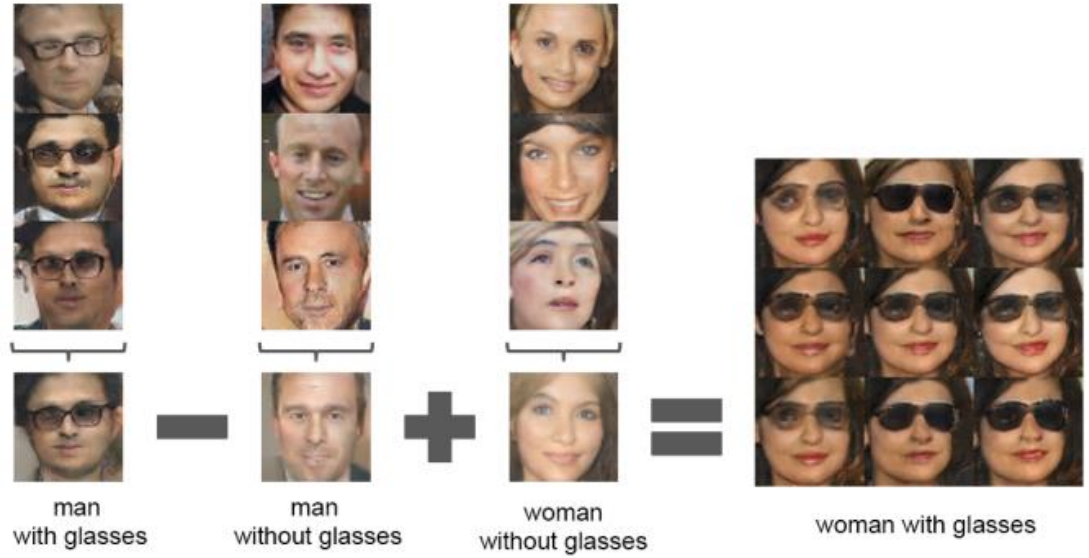


Embeddings and Generative Models

- In practice, unsupervised learning has had limited success compared to supervised learning despite the initial neural net revival being prompted by the generative model of (Mohamed, Dahl, Hinton 2009).
- However interesting research is ongoing and will hopefully be a topic of future ML Reading Group meetings!







Neural Network Libraries

- Torch (Lua)
- Caffe (Protobuf, Python, C++)
- Theano + Keras/Blocks/Lasagne (Python)
- Tensorflow (Python, C++)
- And more…

Further Reading

- Micheal Nielsen's Neural Networks and Deep Learning online textbook.
- Course notes for Convolutional Neural Networks for Visual Recognition by Fei-Fei Li and Andrej Karpathy.
  - Winter 2016 class just began, now with videos!
- http://deeplearning.net/
- Reddit.com/r/MachineLearning

References

- Many slides based on Vincent Vanhoucke's Large Scale Deep Learning slides presented at the Machine Learning Summer School 2015.
  - http://www.iip.ist.i.kyoto-u.ac.jp/mlss15
  - More on scaling up and parallelizing neural networks.
- Some derivations from Marc'Aurelio Ranzato's deep learning tutorial from CVPR 2014.
  - More on ConvNets and tips on training.
- Geoff Hinton's 2015 keynote at the Royal Society: Youtube
  - More on unsupervised nets and history of neural net research.