ECE 417 Fall 2018 Lecture 19: Mini-Batch Training and Data Augmentation

Mark Hasegawa-Johnson

University of Illinois

October 25, 2018

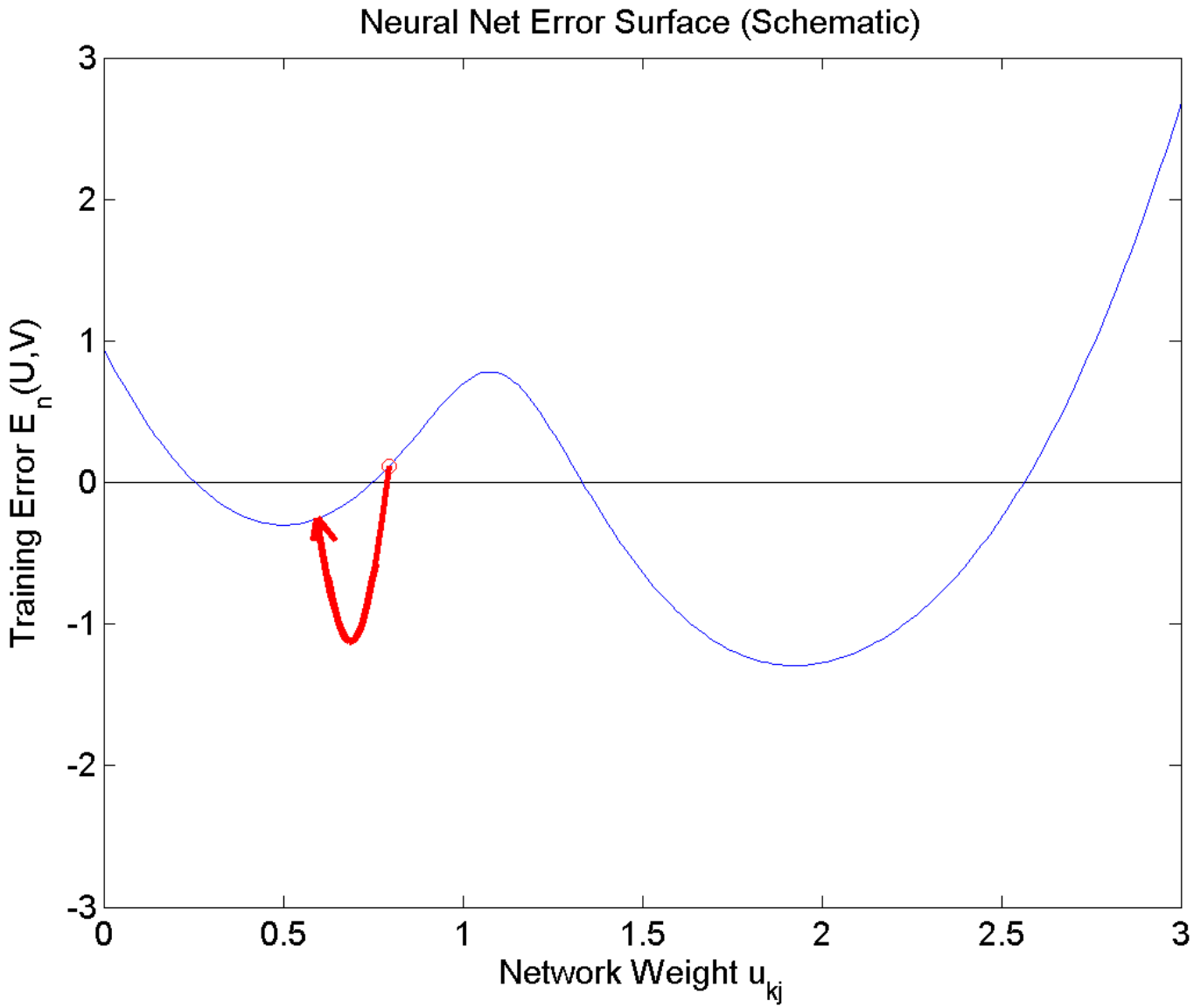
Outline

Simulated Annealing: How can we find the globally optimum U, V?

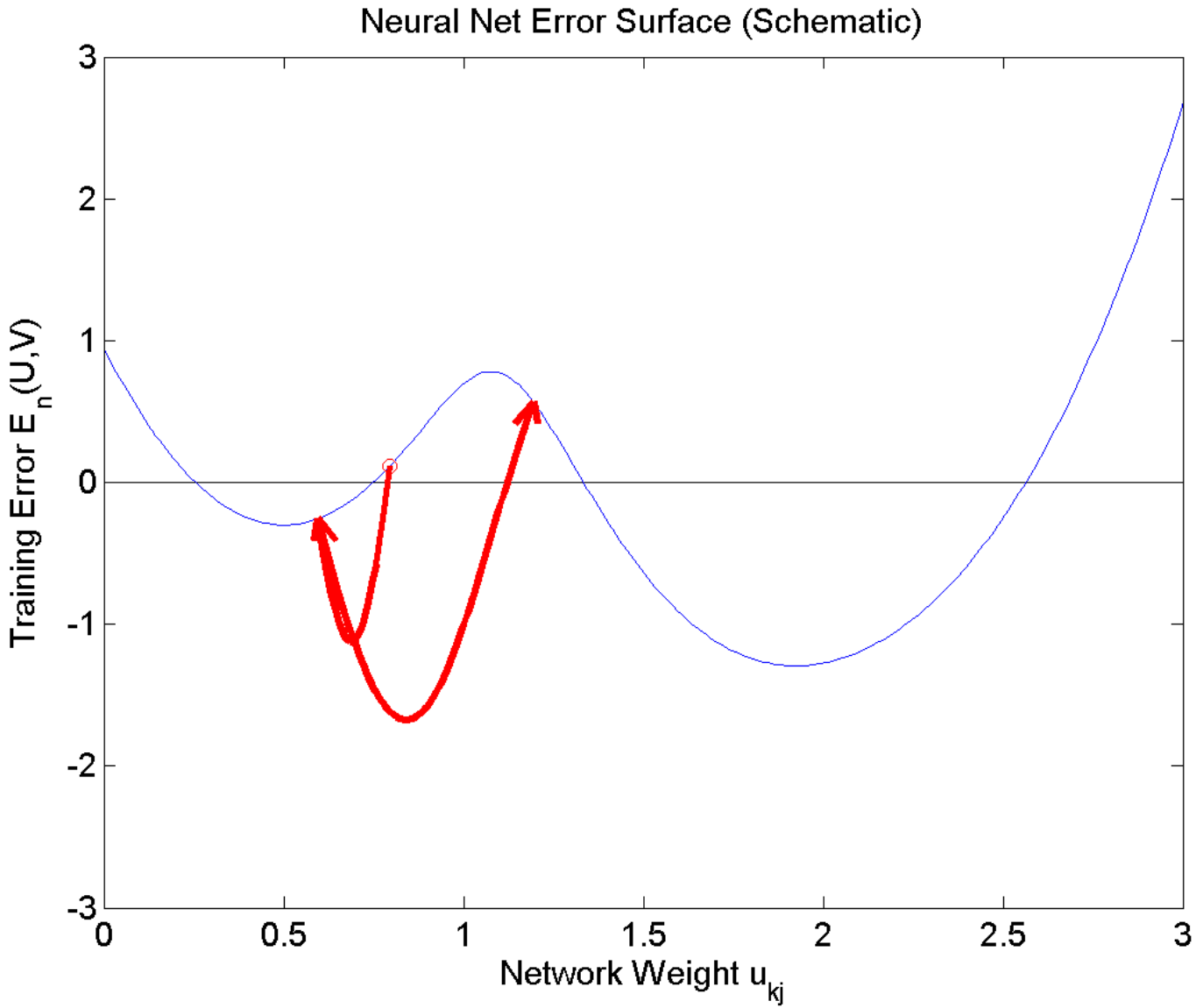
- Gradient descent finds a local optimum. The $\hat{U}, \hat{V}$ you end up with depends on the U, V you started with.
- How can you find the **global optimum** of a non-convex error function?
- The answer: Add randomness to the search, in such a way that...

$$P(\text{reach global optimum}) \overset{t \to \infty}{\longrightarrow} 1$$

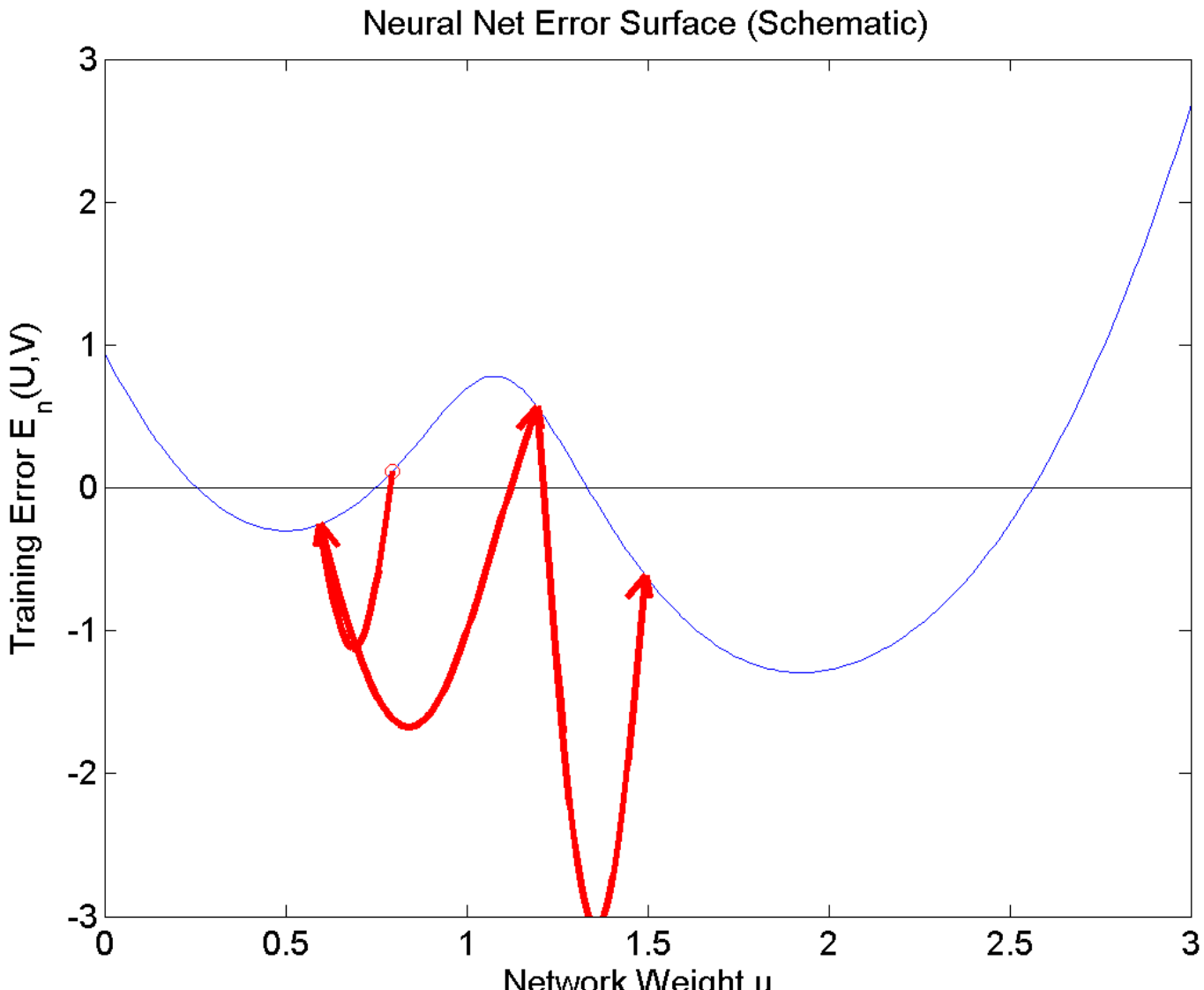
- Take a random step. If it goes downhill, do it.



- Take a random step. If it goes downhill, do it.
- If it goes uphill, SOMETIMES do it.



- Take a random step. If it goes downhill, do it.
- If it goes uphill, SOMETIMES do it.
- Uphill steps become less probable as $t \to \infty$



Simulated Annealing: Algorithm

FOR $t = 1$ TO ∞, DO

1. Set $\hat{U} = U + \text{RANDOM}$
2. If your random step caused the error to decrease ($E_n(\hat{U}) < E_n(U)$), then set $U = \hat{U}$
   (**prefer to go downhill**)
3. Else set $U = \hat{U}$ with probability P
   (**. . . but sometimes go uphill!**)
   1. $P = \exp(-(E_n(\hat{U}) - E_n(U))/\text{Temperature})$
      (**Small steps uphill are more probable than big steps uphill.**)
   2. $\text{Temperature} = T_{max}/\log(t+1)$
      (**Uphill steps become less probable as $t \to \infty$.**)
4. Whenever you reach a local optimum (U is better than both the preceding and following time steps), check to see if it's better than all preceding local optima; if so, remember it.

Convergence Properties of Simulated Annealing

(Hajek, 1985) proved that, if we start out in a "valley" that is separated from the global optimum by a "ridge" of height T_{max}, and if the temperature at time t is $T(t)$, then simulated annealing converges in probability to the global optimum if

$$\sum_{t=1}^{\infty} \exp\left(-T_{max}/T(t)\right) = +\infty$$

For example, this condition is satisfied if

$$T(t) = T_{max}/\log(t+1)$$

If Simulated Annealing is Guaranteed to Work, Why Doesn't Anybody Use It?

Answer: it takes much, much, much longer than gradient descent. Usually thousands of times longer.

Outline

The Three Types of Gradient Descent

Remember that gradient descent means:

$$u_{kj} \leftarrow u_{kj} - \eta \frac{\partial \epsilon}{\partial u_{kj}}$$

1. Batch Training: $\frac{\partial E}{\partial u_{jk}}$ is computed over the entire training database.
2. Stochastic Gradient Descent (SGD): $\frac{\partial E}{\partial u_{jk}}$ is computed for **just one** randomly chosen training token.
3. Mini-Batch Training: $\frac{\partial E}{\partial u_{jk}}$ is computed for a small set of randomly chosen training tokens (e.g., 8, 32, 128).

Gradient Descent Review

Suppose we have an error of the form

$$E = \frac{1}{n} \sum_{i=1}^{n} E_i$$

where E_i might be cross-entropy:

$$E_i = -\log z_{\ell^* i}, \quad \ell^* = \text{the value of } \ell \text{ s.t. } \zeta_{\ell^*} = 1$$

or squared error:

$$E_i = \frac{1}{2} \sum_{\ell} (z_{\ell i} - \zeta_{\ell i})^2$$

or anything else.

Gradient Descent Review

Then the error gradient is

$$\frac{\partial E}{\partial u_{kj}} = \frac{1}{n}\sum_{i=1}^{n}\frac{\partial E_i}{\partial u_{kj}}, \qquad \nabla_U E = \frac{1}{n}\sum_{i=1}^{n}\nabla_U E_i$$

where, for any error that can be decomposed using back-propagation:

$$\frac{\partial E_i}{\partial v_{\ell k}} = \frac{\partial E_i}{\partial b_{\ell i}}\frac{\partial b_{\ell i}}{\partial v_{\ell k}} = \epsilon_{\ell i} y_{ki}, \qquad \nabla_V E_i = \vec{\epsilon}_i \vec{y}_i^T$$

$$\frac{\partial E_i}{\partial u_{kj}} = \frac{\partial E_i}{\partial a_{ki}}\frac{\partial a_{ki}}{\partial u_{kj}} = \delta_{ki} x_{ji}, \qquad \nabla_U E_i = \vec{\delta}_i \vec{x}_i^T$$

Gradient Descent Review

For both cross-entropy and sum-squared error, we actually can get the same equations for back-propagation:

$$\epsilon_{\ell i} = \frac{\partial E_i}{\partial b_{\ell i}} = z_{\ell i} - \zeta_{\ell i} \qquad \vec{\epsilon}_i = \nabla_{\vec{b}_i} E_i = \vec{z}_i - \vec{\zeta}_i$$

$$\delta_{ki} = \frac{\partial E_i}{\partial a_{ki}} = \sum_{\ell} \epsilon_{\ell i} v_{\ell k} f'(a_{ki}) \qquad \vec{\delta}_i = \nabla_{\vec{a}_i} E_i = f'(\vec{a}_i) \odot V^T \vec{\epsilon}_i$$

where $\odot$ means scalar array multiplication.

The Three Types of Gradient Descent

Now we have the context we need, in order to define the three types of gradient descent.

1. Batch Training: $\mathcal{D} = \left\{(\vec{x}_1, \vec{\zeta}_1), \ldots, (\vec{x}_n, \vec{\zeta}_n)\right\}$ is the set of all training tokens, and

$$u_{kj} \leftarrow u_{kj} - \frac{\eta}{n} \sum_{i=1}^{n} \frac{\partial E_i}{\partial u_{kj}}$$

2. Stochastic Gradient Descent: $(\vec{x}_i, \vec{\zeta}_i)$ is a training token chosen at random (with or without replacement), and

$$u_{kj} \leftarrow u_{kj} - \eta \frac{\partial E_i}{\partial u_{kj}}$$

The Three Types of Gradient Descent

Now we have the context we need, in order to define the three types of gradient descent.

3. Mini-Batch Training: $\mathcal{D}^{(t)} = \left\{ (\vec{x}_1^{(t)}, \vec{\zeta}_1^{(t)}), \ldots, (\vec{x}_m^{(t)}, \vec{\zeta}_m^{(t)}) \right\}$ is a set of $m < n$ training tokens chosen randomly (with or without replacement), for the t^{th} iteration of training, and $E_i^{(t)}$ is the error computed for minibatch token $(\vec{x}_i^{(t)}, \vec{\zeta}_i^{(t)})$, and

$$u_{kj} \leftarrow u_{kj} - \frac{\eta}{m} \sum_{i=1}^{m} \frac{\partial E_i^{(t)}}{\partial u_{kj}}$$

When should you use batch training?

- Why should you use batch training? **Pro:** in some sense, minimizing error on the whole training corpus is what training is trying to achieve, so you might as well go ahead and explicitly minimize it.
- Why should you **not** use batch training?
  1. Over-training.
  2. Bad local optima.
  3. Computational complexity.

Why should you **not** use batch training?

1. **Over-training**: Minimizing **training corpus** error might not minimize **test corpus** error (e.g., because training corpus is too small).
2. **Bad local optima**:
   - gradient descent converges to a u_{jk} such that small changes to u_{jk} increase training corpus error.
   - But there might be some other value of u_{jk}, very far away, that has much better training corpus error. For example, simulated annealing would find this by sometimes taking steps at random.
3. **Computational complexity.** Your GPU might not be big enough to load the entire training corpus.

Why should you use SGD?

- Reasons to use SGD:
  1. **Over-training**: SGD doesn't really help, but you can easily control this using **early stopping** (meaning, stop training before you reach full convergence).
  2. **Bad local optima**: SGD adds randomness that is kind of like simulated annealing. In fact, nobody has ever proven that SGD works as well as simulated annealing. But SGD seems to help a lot in practice.
  3. **Computational complexity.** Complexity of SGD is much less than batch training.
- Reasons to **not** use SGD:
  1. Too much random variability.
  2. Computational complexity: GPU can hold 8 or 32 training tokens. Why waste cycles by loading just 1 training token?

Why should you use mini-batch?

1. **Over-training**: Control it with early stopping.
2. **Bad local optima**: If your batch size contains $m \ll n$ tokens, then (there is no proof, but in practice) you seem to get all the stochastic-search benefits of SGD, without the...
3. **Variability:** you can tweak the size of the minibatch. Larger m reduces variability, but makes it harder to "anneal;" smaller m increases variability, therefore increases annealing.
4. **Computational complexity.** Tweak m to be exactly the number of tokens that fit onto your GPU.

Outline

Neural Nets are Data-Hungry

Neural nets need lots and **lots** of training data:

- Training corpus error is bounded as c_1/q, for some constant c_1 that you don't know until after you've done the training, where q is the number of hidden nodes.
- Test corpus error is **always worse** than training corpus error, by an additive percentage of $c_2 q/n$, where c_2 is some other constant that you don't know until you've done the experiment.
- Therefore the total test error is $E_{\text{test}} < c_1 \frac{1}{q} + c_2 \frac{q}{n}$.
- This can be minimized by setting $q = \sqrt{n}$, in which case you always get

$$E_{\text{test}} < (c_1 + c_2) \frac{1}{\sqrt{n}}$$

- So, no matter how big your training corpus is, you can always get better performance by making it even bigger.

Training corpora are never as big as you wish they were.

Data Augmentation

- For every example in your training corpus, $\vec{x_i}$ with label $\vec{\zeta_i}$,. . .
- how many "fake examples" can you create that you're sure will have exactly the same label?

Examples of Data Augmentation

- Add noise to the image: random numbers between $(-\epsilon, \epsilon)$. Multiply the image by random numbers between $(0.95, 1.05)$. Blur the image by blurring factors of $2 - 20$ pixels.
  - Works as long as a human can still recognize the object—i.e., as long as the noise isn't bad enough to hide the object.
- Rotate, shift, scale the image.
  - Works as long as a human would give the rotated, shifted, scaled image the same label as the un-modified image.

Limits of Data Augmentation

It only helps the neural net to learn about the type of variability that you've added. For example, it doesn't help the network to learn that the same object can occur in different background scenes (unless you somehow modify the background scene).

Outline

Dealing with large training corpora

- Simulated annealing: guarantees convergence to the globally optimum network weights, but takes a very long time to train.
- SGD: computationally cheap alternative to simulated annealing (with no theoretical proof that it works), but sometimes has too much variability.
- Mini-batch training: optimize the mini-batch size to fit your GPU, and to trade off between too much versus too little variability. Often $m \approx 32 - 128$.
- Data augmentation. Helps to make a small corpus larger. Use your creativity: use every image modification you can think of, as long as it doesn't change the label of the image.